# Effects of Network Structure Improvement on Distributed RDF Querying

Liaquat Ali, Thomas Janson, Georg Lausen, and Christian Schindelhauer

University of Freiburg
{ali,janson,lausen,schindel}@informatik.uni-freiburg.de
http://www.informatik.uni-freiburg.de

**Abstract.** In this paper, we analyze the performance of distributed RDF systems in a peer-to-peer (P2P) environment. We compare the performance of P2P networks based on Distributed Hash Tables (DHTs) and search-tree based networks. Our simulations show a performance boost of factor 2 when using search-tree based networks. This is achieved by grouping related data in branches of the tree, which tend to be accessed combined in a query, e.g. data of a university domain is in one branch. We observe a strongly unbalanced data distribution when indexing the RDF triples by subject, predicate, and object, which raises the question of scalability for huge data sets, e.g. peer responsible for predicate 'type' is overloaded. However, we show how to exploit this unbalanced data distribution, and how we can speed up the evaluation of queries dramatically with only a few additional routing links, so-called *shortcuts*, to these frequently occurring triples components. These routing shortcuts can be established with only a constant increase of the peer's routing tables. To cope with hotspots of unfair load balancing, we propose a novel indexing scheme where triples are indexed 'six instead of three times' with only 23% data overhead in experiments and the possibility of more parallelism in query processing. For experiments, we use the LUBM data set and benchmark queries.

## 1 Introduction

A goal of the Semantic Web [1] initiative is to integrate data from web resources into machine-driven evaluation. The Resource Description Framework (RDF [2]) data model has been proposed by the W3C to encode these data. To cope with the anticipated load of the Semantic Web data, several projects have emerged that have studied distributed solutions for the storage and querying of RDF data. State-of-the-art distributed RDF data stores such as RDFPeers [3], Atlas [4,5], and BabelPeers [6] use Distributed Hash Tables (DHTs) to store and query RDF data in a distributed manner. To attain an efficient search for RDF triples with the same subject, predicate, or object, the triples are indexed three times for each triple component (subject, predicate, or object) in these distributed RDF databases. DHTs while provide fair load balancing properties with easy data management under churn[1] they also destroy the ordering of the index by using hashing, and along with it the grouping of semantically-related data, e.g. data of a university domain cannot be stored on a contiguous interval and is spread over the complete table. This can cause more routing when collecting data from the same domain to evaluate a query.

GridVine [7] and our proposed distributed RDF system 3rdf [8] address this by using the P2P networks P-Grid [9] and 3nuts [10] respectively, providing a distributed search tree for order-preserving indexing. Domain-related prefixes (namespaces) in subjects, predicates, and objects of RDF triples order triples of the same domain in the same branches of the search tree. In return, the data belonging to the same domain is stored on nearby peers (in the metric of the overlay routing structure) or even at the same peer.

In this paper, we evaluate the performance of distributed RDF systems when using either the Chord or 3nuts P2P-network and when using two different RDF data distributions in the network, the state-of-the-art indexing for subject, predicate, and object and a novel indexing introduced in this paper. In Section 2 we provide an overview about distributed RDF systems and distributed query evaluation techniques relating to this work. In Section 3 we present our simulation including overlay networks, RDF data and query model, data distribution including our new indexing scheme for a fairer data distribution, and the query processing including speed-ups by exploiting additional features of the 3nuts network. Based on this simulator, the simulation results regarding the performance of the RDF system with the performance metrics routing-steps and time for RDF query evaluation are presented in Section 4. Finally, we conclude in Section 5, and give a brief outlook on future work.

## 2 Related Work

With more and more Web resources annotated with RDF information, distributed solutions for storage and querying of RDF data is a need. Several projects have proposed peer-to-peer networks for the distributed evaluation of RDF data. The majority of these projects, such as RDFPeers [3], Atlas [4,5], and BabelPeers [6], use DHTs for the storage and querying of RDF data. The basic idea here is to store each triple at three locations using the hash values of subject, predicate, and object. Triples with a specific subject, predicate, or object are obtained during query evaluation by computing the hash value of that specific key again to resolve the peer providing these triples. To improve the query load distribution, authors in [5] additionally index the triples by combinations of triple components 'subject+predicate', 'subject+object', 'predicate+object', and 'subject+predicate+object', with 7 replications of each triple in total. In Section 3.3 we present a similar technique but with another objective, which offers a more balanced data distribution. In the work [3] they measure the amount of highly frequent triple components. The authors of [3,6] mention that triples

[1] Peers entering/leaving the network only invoke local changes and take over/shed data to neighbors.

are not distributed uniformly in the network because of non-uniform distributed index-keys. In this paper we will provide an analysis of the data distribution.

Traditional DHT-based P2P networks, such as Chord [11], are used as underlying overlay networks in the aforementioned distributed RDF data stores applying uniform hash functions to map data keys to peers in the network. This achieves good storage load balancing but sacrifices the preservation of the semantic proximity of RDF triples because it destroys existing relations among the inserted triples keys (attributes) based on their order. RDF triple keys which are semantically close at the application level are heavily fragmented in DHTs, and hence the efficiency of RDF range queries or queries posed on semantically-related attributes is significantly spoiled in these networks.

GridVine [7] and 3rdf [8] are other distributed RDF systems proposed for the storage and querying of RDF data. GridVine and 3rdf use the P-Grid [9] and 3nuts [10] P2P networks respectively, to provide an order-preserving search tree instead of a DHT-based search structure. The ordering in the tree can represent the semantical proximity of closely related RDF triples (e.g. triples predicates with the same prefix will be organized in the same subtree). In contrast to P-Grid, 3nuts provides the additional feature of so-called *interest locality*, where peers with a special interest in a particular search key or path can voluntary participate in managing these paths. When co-managing a path and establishing routing there, a peer increases its routing table but retains fast routing in that path with direct links to other peers in the branch of the path. This is the reason why we say the peer has a *shortcut* to that path. Additionally, the peer may also participate in voluntary managing data in a path (which we do not make use of in this work).

## 3 Simulation

### 3.1 Network Models

We simulate a distributed RDF system using either the DHT-based overlay Chord or the search-tree based overlay 3nuts and compare the performance between both. The results might be transferable to the complete class of both DHT-based and search-tree based peer-to-peer networks. The basic difference between both will be explained below.

**DHT-Based Overlay Networks.** The majority of the state-of-the-art distributed RDF systems still use DHTs [12] for data allocation in the distributed system. In a DHT, each peer and data item has an identifier, e.g. network address and file name, which are hashed to a hash key in key space $[0, 2^m)$ for a typical constant $m = 128$ for 128-bit keys. A peer then gets all data assigned which has a hash key between its hash key and the next larger hash key of another peer in the key space ring. It can be shown that the key space range assigned to any peer is not greater than factor $\mathcal{O}(\log n)$ as the expected key space range which is $2^m/n$ for $n$ peers in the network. This results in a fair load balancing of data IDs. The DHT implementations then provide a routing structure to store and look up data items with a lookup time, for instance, for Chord [11] with $\mathcal{O}(\log n)$. Fair load-balancing and logarithmic lookup time achieve scalability where the network performance does not decrease considerably with the size of the network.

**Tree-Based Overlay Networks.** GridVine and 3rdf are distributed RDF systems using search-tree based overlay networks, such as P-Grid [9] and 3nuts [10]. The fundamental difference when using a search tree instead of a hash table is omitting any hashing of data keys in order to preserves the order of data keys in key space. In addition to logarithmic point queries this achieves efficient range queries in key space, whereas hash tables do not provide an efficient implementation for range queries and the complete table has to be searched. Therefore, when applications such as distributed RDF systems can organize data in key space so that RDF triples needed during query evaluation have nearby keys in key space, (e.g. a similar prefix in the search key equivalent to a small key range), the advantage of range queries can be exploited and lookup time is significantly reduced. The price for efficient range queries in this class of overlays is more complicated and up to a constant factor larger routing structure being more difficult to uphold under churn and data sets with high dynamics.

## 3.2 RDF Data and Query Model

Each node in the distributed RDF system can publish RDF resources in the network. In RDF, resources are expressed as subject-predicate-object expressions, called triples. The subject in a RDF triple denotes the resource, and the predicate expresses a relationship between the subject and the object. Let $U$ and $L$ represent URIs and Literals, respectively in RDF, a triple $(v_1, v_2, v_3) \in U \times U \times (U \cup L)$ over certain resources $U$ and $L$ is called a *RDF triple* [2]. RDF data can also be represented as graph where subject and object are nodes and the predicates are edges.

RDF resources are normally represented by URIs [2], and resources belonging to a particular application domain usually share common namespaces or prefixes (e.g. the 'Professor', 'name', 'email', and 'teach' keys in triples of Listing 1.1 share a common prefix 'ubd0v0'). Thus, the support of efficient range queries in tree-based overlays, which is equivalent to short lookup times between network keys with the same prefix, achieves short querying time when RDF data with the same prefix is associated in a query (see the example in Listing 1.1).

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix ubd0v0: <http://www.lehigh.edu/zhp2/2004/0401/univ-bench.owl#>
@prefix d0v0: <http://www.Department0.University0.edu#>
d0v0:P3    rdf:type    ubd0v0:Professor.
d0v0:P3    ubd0v0:name     Georg.
d0v0:P3    ubd0v0:email    georg@ub.com.
d0v0:P3    ubd0v0:teach    d0v0:course1.
```

**Listing 1.1.** RDF triples about a resource *d0v0:P3* encoded in RDF/N3 format

Each node in the distributed RDF system can also pose SPARQL basic graph pattern queries to retrieve the RDF data stored in the system. These basic graph pattern queries are composed of triple patterns. Let $U$, $L$, and $V$ represent URIs, Literals, and Variables, a triple $(v_1, v_2, v_3) \in (U \cup V) \times (U \cup V) \times (U \cup V \cup L)$ is called a *SPARQL triple pattern* [13]. We evaluate the system performance, such as response time and routing hops needed per query, by performing conjunctive triple pattern queries in simulated RDF systems with either the Chord or 3nuts overlay. Following [5], a *conjunctive query* $Q$ is a formula:

$$?x_1, \ldots, ?x_n :- (s_1, p_1, o_1) \wedge \cdots \wedge (s_n, p_n, o_n)$$

where $?x_1, \ldots, ?x_n$ are variables and each $(s_i, p_i, o_i)$ is a triple pattern. Each variable $x_k$ appears in at least one triple pattern. These patterns are matched against the input RDF triples to find all assignments of variables to URIs and literals such that all triples of the query can be found in the RDF data-store.

## 3.3 Data Distribution

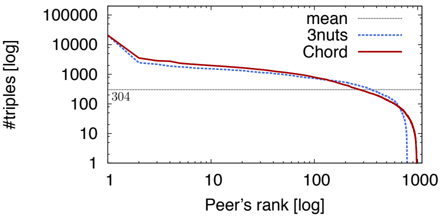
SPARQL, as well as the majority of RDF query languages, mainly support constraint search of the triples's subject, predicate, or object values. Thus, as in systems like (RDFPeers, Atlas, and BabelPeer), we index each triple three times using the subject, predicate and object identifier.

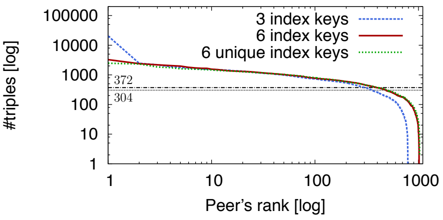
The degree of the underlying RDF graph is not limited and predicates occur several times to identify the same type of association between nodes. If we index triples by subject, predicate or object, we group the triples with the same identifier on the same peer, with the advantage that we can perform a constraint search for a specific subject, predicate or object in a local database. The main drawback is that we leverage the load balancing techniques of most overlays because data items with the same key are managed on the same peer. Since the degree of this graph is not limited in any way, the load-balancing might be unfair, so that typically there are lots of 'type'-predicates in most practical RDF graphs only managed by one peer.

To cope with unfair load balancing, we propose another indexing scheme where a triple is indexed for each possible combination of its 2 out of 3 components, giving the $\frac{3!}{(3-2)!} = 6$ combinations subject+predicate, subject+object, predicate+subject, predicate+object, object+subject, object+predicate. This means that, for instance, triples containing the predicate 'type' are subdivided according to all possible classes. Thus, in the example of Listing 1.1 there is one specific peer managing all 'types' with object 'professor' but not necessarily 'types' for other objects. So if we already have a constraint for this object in a query, all triples on this peer are sufficient. Queries where two triple components are unknown become more challenging, when not all data for a specific triple component are on the same peer and have to be collected from many peers. This is equivalent to range queries, which are not practical in DHT-based overlays. We therefore see a trade-off in DHT-based systems with two options, either a more balanced data distribution with 6 indexes but limited functionality (evaluation of triple patterns with only 1 constant is not possible) or all functions but more unbalanced data with the 3 index scheme. In contrast, in a search-tree based system these kinds of range queries are very efficient. To retrieve all triples of a specific predicate with the predicate+object index, we go to an arbitrary peer in the predicate's path in the search-tree and scan the subtree for all predicate-object combinations only using direct routing links.

The data amount in the network is not necessarily twice when doubling the number of indexes. Imagine a predicate's path in the tree managed by a single peer in the 3 indexes scheme. This peer then will manage the same triples in the 6 indexes scheme since all triples for indexes predicate+subject and predicate+object will be in the subtree of the path of the predicate's identifier, and a triple has to be transmitted only once in this case instead of twice for different peers for paths predicate+subject and predicate+object. If $m$ peers are in the path of a triple component, the expected overhead factor of triples is $(2 - 1/m)$.

We have analyzed the distribution of triples in our RDF system with $1,000$ peers and $100,000$ triples. Figure 1 shows the triples managed by the peers in decreasing order to the number of triples per peer, e.g. the peer with rank 1 in Figure 1a has $20,000$ triples. Certainly, in an optimal load balancing, all peers would manage the same amount of triples indicated by the constant function of the mean value. For 3 indexes the mean number of triples managed by a peer





(a) Chrod/3nuts comparison for 3 indexes (b) #triples/peer in 3nuts for 3/6 indexes

**Fig. 1.** Triple distribution on peers in distributed RDF system

was 304, but the median peer in 3nuts has 178 triples and in Chord only 132, indicating a slightly better load balancing in 3nuts. When we use 6 instead of 3 indexes (Fig. 1b) and compare the top ranked peers, we can in fact prevent hotspots where peers are overloaded by data and query requests slowing down the system. Surprisingly, the number of triples is only 23% more compared to 3 indexes and (and not 100% more). And, as can be seen in Figure 1b, the peers with only a few triples at the low end for 3 indexes get additional triples in the 6 indexes scheme. Accordingly, the median peer has 279 triples and the mean number of triples is 372 indicating a better distribution than for 3 indexes. We also checked a scheme with 6 indexes but unique ID extensions for all triples to prevent triple clustering. The effect is negligible and the clustering of triples by the same ID does not interfere with load balancing for 6 indexes.

## 3.4 Query Processing

Data of several peers has to be combined for evaluating conjunctive queries, and we use the query processing algorithm, Query Chain (QC), originally presented in [5], where the triple patterns contained in the query are iteratively resolved by a chain of nodes. The query evaluation starts with a single triple pattern of the query by doing a lookup for the peer responsible for the evaluation of this triple pattern. This peer adds to an intermediate result all triples of its local database qualified for the evaluated triple pattern. The intermediate result is then extended by doing a lookup for a second triple pattern and joining the results. This operation is executed until all triple patterns of the query have been processed.

When triples are indexed six times for good load balancing, the triple storage cost is 23% more for our test data than using three indexes. But we can exploit this extra storage to achieve a better distribution of query processing load and faster query response time by bundling bandwidth with parallelism. The Spread By Value (SBV) query processing, presented in [5], extends the ideas of QC by exploiting the values of matching triples found during processing triple patterns incrementally; it rewrites the next triple pattern and distributes the responsibility of evaluating it to more peers than QC. In our experiment we consider the case where triples are indexed three times, and thus we use QC query processing. Due to the small triple overhead of 23% caused by grouping of data with same data in the same subtree, we will only get parallelism for identifiers of triple components with a large set of triples managed by several peers. But if the number of triples correlates with the result data size, it is not a bad thing.

**Exploiting 3nuts Locality Features.** The 3nuts *information locality* feature preserves key ordering; therefore for triple patterns with lookup keys sharing the same prefixes (e.g. the 'UndergraduateStudent', 'name', 'advisor', and 'teacherOf' etc lookup keys in Listing 1.2 share a common prefix 'ubd0v0'), the number of hops required to reach all these keys is reduced (at best, some keys are managed by the same peer).

The number of hops required to evaluate triple patterns of a query can be further reduced through a so-called feature *interest locality* provided in the distributed search tree of 3nuts. It allows peers responsible for triples components (subject, predicate, or object) to establish few additional routing links, *shortcuts*, to other frequently occurring components (subject, predicate, or object) of relevant triples. These routing shortcuts can be established with only a constant increase of the peer's routing tables. To support the creation of shortcuts, each peer in the network responsible for the triple components' values (i.e. subject, predicate, or object) creates a link to the peer, sharing at least one of the components' values of the same triple. In this way peers create Peers Link Graph defined as follows:

**Definition 1.** *A Peers Link Graph G is a tuple (N, E), where N is the set of peers in G and E is the set of directed weighted edges in G. Two peers from N* *are connected with an edge in E if and only if triples managed by these two peers share at least one triple components' value. The weight of an edge E represents the frequency of such related triples managed by corresponding peers.*

Peers maintain statistics of their weighted links with the help of their locally stored RDF triples. Each of these Peers Link Graphs is also connected with the rest of the peers in the network through a directed edge of weight 1.

An example partial Peers Link Graph created for the peers sharing the subjects of triples belonging to the class *Professor* is shown in Figure 2. The edge from *Peer1* to *Peer3* with weight 2 indicates that 'course1' is taught by two professors.

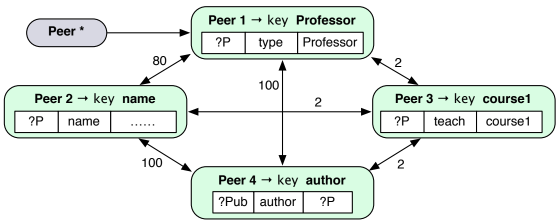


**Fig. 2.** Example of Peers Link Graph for triples of class Professor

We observe that the frequency of subject, predicate, and object occurrences in triples is not uniformly distributed, and we can assume that triples components (subject, predicate, or object) which frequently occur in these triples will also frequently occur in conjunctive triple pattern queries. We exploit this unbalanced triple distribution, and thus consider only the top weighted edges of Peers Link Graph for the creation of *shortcuts*. For example, in Figure 2, the edge from *Peer1* to predicate *name* with weight 80 has a high chance of being considered for the creation of a shortcut than the edge from *Peer1* to object 'course1' with weight 2.

## 4 Performance Analysis

In this section, we present experimental performance evaluation done with a simulator for our RDF system using either a Chord network or a 3nuts network and the 3 index scheme presented in Section 3.3 for triple storage in the overlay. The Java-implemented Simulator includes a simulation for the physical network, the overlay network, with the distributed RDF system on top. We repeated all experiments several times but the variation of result values was negligible and is thus not presented here. For the testing we use the Lehigh University Benchmark (LUBM [14]) data-set of two universities, each with 16 departments. There were 223,510 triples in total, which were indexed 3 times in the system. The network contained up to 16,285 peers.

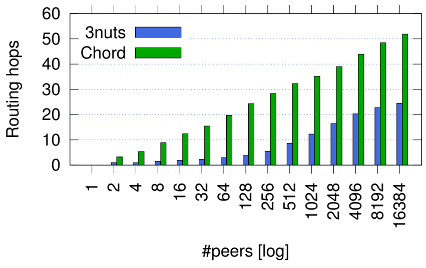
Application domains usually mark their own ontologies in RDF data with a unique namespace. Accordingly, each university department in the test set gets a distinctive namespace. In the example Listing 1.1, department 0 of university 0 uses the namespaces 'ubd0v0' and 'd0v0'. We have generated 5 query sets (based on LUBM queries) that sum up to 130 unique queries in total. Each set shares queries of a common structure with distinctive namespaces for each department. The first query set comprises for queries of the type in Listing 1.2 and we vary for instance the namespace 'ubd0v0' for 32 departments to generate more queries.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ubd0v0: <http://www.lehigh.edu/zhp2/2004/0401/univ-bench.owl#>
SELECT ?std_name ?teacher_name ?course_name
WHERE { ?X rdf:type ubd0v0:UndergraduateStudent.
?X ubd0v0:name ?std_name.
?X ubd0v0:advisor ?Y.
?Y ubd0v0:name ?teacher_name.
?Y ubd0v0:teacherOf ?Z.
?X ubd0v0:takesCourse ?Z.
?Z rdf:type ubd0v0:Course.
?Z ubd0v0:name ?course_name }
```

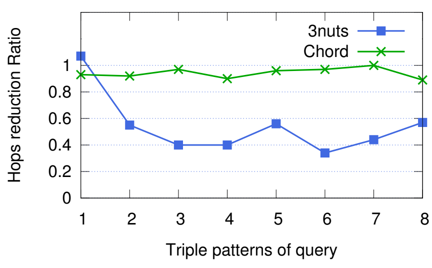
**Listing 1.2.** SPARQL query returning names of the students their advisors and courses taken, provided that courses taken are taught by their supervisors

## 4.1 Analysis of Routing

Distributed query processing in a RDF system consists of searching, transferring and evaluating RDF data. The distributed search structure of P2P-networks achieves scalability but slows the search down typically to $\mathcal{O}(\log n)$ routing steps (called hops) in a network with $n$ peers compared to a server with sufficient resources. When the bandwidth of modern networks is high but the ping or delay for data transmission is low, the response time of queries can by driven by these delays, because each routing step in chain produces an additional delay.



(a) Comparison of 3nuts and Chord



(b) Impact of information locality

**Fig. 3.** Measurement results for resolving the query in Listing 1.2

Figure 3a shows the mean numbers of hops of lookup-operation for resolving the query of Listing 1.2 depending on the network size $n$. The linear functions in log-lin scale specially for Chord indicate a logarithmic hop number. The triple components of this query are all in the domain 'ubd0v0'. Consequently, corresponding triples are stored in 3nuts in the same subtree with path 'ubd0v0' and peers managing this subtree have fast lookup to each other. This really keeps the hop numbers down for small networks up to 64 peers where single or only a few peers manage the data for that domain. For larger networks more routing is needed within the subtree and we see a linear increase starting a 128 peers like in Chord.

While Figure 3a showed the total numbers of hops during processing a query, Figure 3b shows the numbers of hops needed for routing of the next triple pattern in chain of triple patterns in the query (percentile to the expected hop numbers $\mathcal{O}(\log n)$). In the Chord overlay, we need almost the same numbers of hops to lookup all indexed triple components. On the contrary, in the 3nuts overlay, the hops for lookup decrease after the first step by 50% and the reason is again the information locality: the first step starts at an arbitrary peer which might not participate in the subtree of domain 'ubd0v0' in the 3nuts tree with $\mathcal{O}(\log n)$ hops, but when the query enters this subtree once, it stays inside for the rest of the query steps with fast routing.

Figure 4b shows further reductions in the numbers of hops needed for evaluating the query in Listing 1.2 with the creation of up to 5 shortcuts by each peer in 3nuts network. If a peer interested in evaluation of a triple pattern maintains a *shortcut* to the lookup key of this triple pattern, then it takes it 0 hops to reach this lookup key. The establishment of constant numbers of shortcuts for each peer to frequently occurring triple components in the network could result to a significant reduction in numbers of hops, e.g. the numbers of hops needed for our example query reduced from 26 to 9 with creation of 4 shortcuts.

## 4.2 Analysis of Query Response Time

Our simulation does not provide the simulation of a real-world physical network and it is impossible to find one suitable model covering all possible fields of application such as the Internet, Intra net, and so on. Therefore we will present in this section how to derive the query response time from the given experimental routing data with a given physical network model. Let $k = \delta \cdot b$ denote the product of delay $\delta$ and bandwidth $b$ in the physical network. The query response time $t$ is then for $m$ triples patterns $p_i$

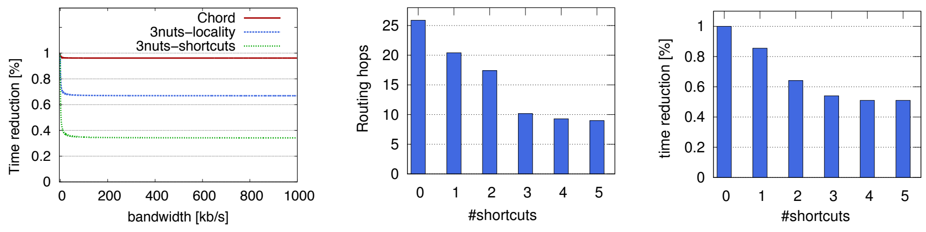
$$t = \sum_{i=1}^{m} \left( c_i + \mathrm{hops}_i \cdot \delta + \left( \frac{\mathrm{data}_i \cdot \delta}{k} + \delta \right) \right) \approx \delta \cdot \sum_{i=1}^{m} \left( \mathrm{hops}_i + \frac{\mathrm{data}_i}{b} + 1 \right) .$$

where $c_i$ is the computation time for evaluating pattern $p_i$, $\mathrm{hops}_i$ is the number of hops to lookup the peer providing the triples for pattern $p_i$, and $\mathrm{data}_i$ is the data size for triples of pattern $p_i$ to transfer to the next peer. For $c_i \ll \delta$ we neglect the computation time.

The response time is mainly driven by routing delays and data transfer time. When the bandwidth is very high, routing delays are a dominant factor and reducing routing hops with the 3nuts overlay has a big influence, e.g. can reduce the query response time in our example to 65%, see Figure 4a where we calculate the ratio of the query response time for the actual routing of the query and the expected time for an arbitrary routing in the network. The routing in query processing in the Chord overlay almost needs the same time like arbitrary routing steps between arbitrary peers with nearly no effect.

The improvement in query time shown in Figure 4a can be enhanced with the creation of constant numbers of shortcuts to frequently occurring triples components in underlying 3nuts network, e.g. the query time in our example can be further reduced to 30% with creation of 4 shortcuts as shown in the same Figure 4a. The creation of shortcuts reduces traffic and peers routing load as well because a peer maintaining shortcuts to triples components has a direct link to the peers responsible for these components.

Figure 4c shows the ratio of improvements in response time of the query in Listing 1.2 with the creation of up to 5 *shortcuts* by each peer in the 3nuts overlay. The response time of the example query is reduced to 50% with creation of 4 *shortcuts* in the underlying 3nuts overlay.



(a) Impact of information locality and shortcuts

(b) Hops needed in the query of Listing 1.2

(c) Time reduction using shortcuts (Listing 1.2)

**Fig. 4.** Measurement results for the performance boost using 3nuts localities

## 5 Conclusions

In this paper we analyzed techniques for optimization of distributed RDF systems and showed the practical application by simulation with the benchmark (LUBM) data and queries. We have shown that using search-tree based instead of DHT-based peer-to-peer networks improves the query response time by up to a factor of two if RDF data of the same domain, for instance data of the same university is grouped in the same branch of the tree and the query only combines data of such a limited domain. We also tested the usage of shortcuts in the 3nuts overlay network, where the peers increase their routing structure by a constant factor. This technique achieves a further speed-up of query response times of factor for instance 2.5 for 3 shortcuts per peer. We also presented a novel indexing scheme with 6 indexes which only needs 23% more data in tests compared to the state-of-the-art 3 indexes scheme but improved the load balancing and could eliminate hotspots in the network where peers had to manage far more triples than others, e.g. the predicate 'type'.

## References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34–43 (2001), `http://www.scientificamerican.com/article.cfm?id=the-semantic-web`
2. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. World Wide Web Consortium. Tech. Rep. (2004), `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`
3. Cai, M., Frank, M.: RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In: Proceedings of the 13th International Conference on World Wide Web, New York, USA, pp. 650–657 (2004)
4. Kaoudi, Z., Koubarakis, M., Kyzirakos, K., Miliaraki, I., Magiridou, M., Papadakis-Pesaresi, A.: Atlas: Storing, updating and querying RDF(S) data on top of DHTs. J. Web Sem. 8(4), 271–277 (2010)
5. Liarou, E., Idreos, S., Koubarakis, M.: Evaluating Conjunctive Triple Pattern Queries over Large Structured Overlay Networks. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 399–413. Springer, Heidelberg (2006)
6. Battré, D., Heine, F., Höing, A., Kao, O.: On Triple Dissemination, Forward-Chaining, and Load Balancing in DHT Based RDF Stores. In: Moro, G., Bergamaschi, S., Joseph, S., Morin, J.-H., Ouksel, A.M. (eds.) DBISP2P 2005 and DBISP2P 2006. LNCS, vol. 4125, pp. 343–354. Springer, Heidelberg (2007)
7. Aberer, K., Cudré-Mauroux, P., Hauswirth, M., Van Pelt, T.: GridVine: Building Internet-Scale Semantic Overlay Networks. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 107–121. Springer, Heidelberg (2004)
8. Ali, L., Janson, T., Lausen, G.: 3rdf: Storing and Querying RDF Data on Top of the 3nuts Overlay Network. In: 10th International Workshop on Web Semantics (WebS 2011), Toulouse, France, pp. 257–261 (August 2011)
9. Aberer, K.: P-Grid: A Self-Organizing Access Structure for P2P Information Systems. In: Batini, C., Giunchiglia, F., Giorgini, P., Mecella, M. (eds.) CoopIS 2001. LNCS, vol. 2172, pp. 179–194. Springer, Heidelberg (2001)
10. Janson, T., Mahlmann, P., Schindelhauer, C.: A Self-Stabilizing Locality-Aware Peer-to-Peer Network Combining Random Networks, Search Trees, and DHTs. In: ICPADS, pp. 123–130 (2010)
11. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. SIGCOMM Comput. Commun. Rev. 31, 149–160 (2001)
12. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In: Proceedings of the 29th Symposium on Theory of Computing (STOC 1997), pp. 654–663. ACM, New York (1997)
13. SPARQL Query Language for RDF, `http://www.w3.org/TR/rdf-sparql-query/`
14. Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Web Semantics: Science, Services and Agents on the World Wide Web 3(2-3) (2005)