Technical report LSIR-REPORT-2006-011

UniStore: Querying a DHT-based Universal Storage

Marcel Karnstedt, Kai-Uwe Sattler,
Martin Richtarsky, Jessica Müller
TU Ilmenau
Germany

Manfred Hauswirth, Roman Schmidt,
Renault John
EPFL
Switzerland

Abstract

In recent time, the idea of collecting and combining large public data sets and services became more and more popular. The special characteristics of such systems and the requirements of the participants demand for strictly decentralized solutions. However, this comes along with several ambitious challenges a corresponding system has to overcome. In this demonstration paper, we present a light-weight distributed universal storage capable of dealing with those challenges, and providing a powerful and flexible way of building Internet-scale public data management systems. We introduce our approach based on a triple storage on top of a DHT overlay system, based on the ideas of a universal relation model and RDF, outline solved challenges and open issues, and present usage as well as demonstration aspects of the platform.

1 A Universal Storage based on DHTs

An increasing number of applications on the Web are based on the idea of collecting and combining large public data sets and services. In such *public data management* scenarios, the information, its structure, and its semantics are controlled by a large number of participants. Despite being distributed or decentralized in respect to data from a conceptual point of view, the supporting infrastructures of these systems still are based on inherently centralized concepts. The downsides at the physical layer of such centralized systems, such as bottlenecks, single-point-of-failures and enormous costs for providing the needed resources, are extended by problems on a more logical level, e.g., the problem of integrating data/services and the need of database processing functionality. Examples of such applications include (specialized) Web search engines, scientific database applications, naming or directory services and "social" applications such as file/picture sharing, encyclopedias, friend-of-a-friend networks or recommender systems.

In this paper, we argue for a decentralization of data management by creating a universal distributed storage for such public data/metadata, which exploits the gigantic storage and processing capacity of the worldwide available Internet nodes in the same way as the network layer exploits the worldwide communication devices for routing messages between nodes. Information sources are highly distributed, data is described according to heterogeneous schemas, no participant has a global view of all information, and data and service quality can only be guaranteed in a best effort way. In this context, the global challenge is to develop a light-weight, generic data management component playing the same role as the TCP/IP stack and a highly scalable infrastructure enforcing a fair distribution of storage and processing load in a highly dynamic world without any central control.

For such type of public information management, DHT-based overlay systems offer an interesting alternative to existing information system architectures. While problems like scalability, robustness and fair balance of load and work are covered by modern DHTs, new research problems have to be addressed, the most prominent being: Data may exist in a large number of different schema organizations and expressiveness of queries and possible guarantees (existence, completeness, etc.) are limited at the moment.

Concerning a distributed universal storage as we propose, the key issues can be classified along three questions:

(1) How to structure and organize data in massively distributed settings?
(2) How to query data and how to query efficiently?
(3) What is needed to get a robust and practical solution?

The first question raises two main problems: We need a generic and flexible schema for structuring data and we have to deal with heterogeneities on schema and on data level. The second question highlights challenges of query processing: The system has to support the combination of both, classical DB-like queries allowing to restrict and combine data (selection, projection, join, set operations) as well as IR-style queries (e.g., keyword search over all attributes, similarity). Moreover, querying schema data (attributes, correspondences) has to be supported as well. Physical query processing should exploit the features of the underlying infrastructure (e.g., hash-based placement, topology-aware routing and multicasting), come with worst-case guarantees, and involve cost-based and adaptive query opti-

mization considering the dynamicity of the whole network and the autonomy of the individual nodes. Finally, question (3) touches practicability of a large-scale distributed platform, where the main challenges are: scalability, robustness and availability, as well as privacy, trust, and fairness.

In this work, we present a platform we implemented in order to attack several of the mentioned challenges. It is based on a triple storage on top of the P-Grid [1] overlay, a universal relation model treating data and schema information uniformly and a light-weight query language inspired by RDF query languages. The software is not intended to run simulations, rather we introduce a platform intended for usage. In section 2 we will give a brief overview of the system's architecture, whereupon in section 3 we will highlight special features of our approach. Finally, in section 4 we will introduce usage of the system and outline what we are going to demonstrate.

2 Architecture

Structured P2P overlays are a good basis for a distributed universal storage as we propose, because they scale well, offer logarithmic search complexity in the number of nodes and are based on hashing for data placement, which allows for realizing efficient query processing strategies. Additionally, they offer guarantees and limits needed for defining an appropriate cost model.

Figure 1 shows the architecture of the implemented system. Based on the P-Grid [1] DHT layer, triple storage functionality is provided by a second layer, which is used by P-Grid's *StorageService* to store triple data and to process structured queries.

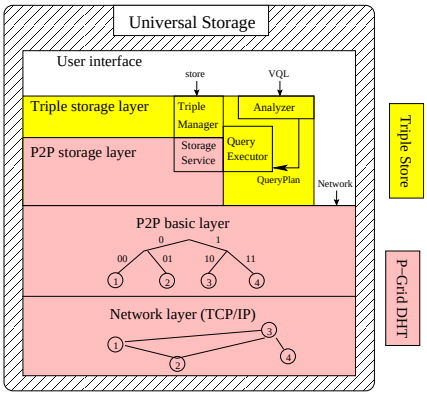

Figure 1: Architecture

In P-Grid, nodes are at the leaf level of a virtual binary trie inducing no hierarchy of nodes. The trie is constructed by pair-wise interactions between nodes without central coordination nor global knowledge. While nodes incrementally partition the key space during runtime of the overlay, they keep references to each other to enable prefix-based query routing. A prefix-preserving hash-function assigns data stored in P-Grid to key partitions respectively nodes. While an order-preserving hash function as used in P-Grid keeps semantic relations between data, it requires sophisticated load-balancing to deal with skewed data distributions. P-Grid includes a mature load-balancing technique able to deal with nearly arbitrary data skews [2]. P-Grid supports efficient substring search and range queries through its basic infrastructure, where other DHTs require additional structures (e.g., in Chord an additional trie-structure is constructed on top of its ring-based overlay network to support range queries). Additionally, P-Grid comes with an update functionality with lose consistency guarantees [4] and enables the merging of two, formerly independent, overlays in a parallel fashion. P-Grid is implemented in Java and available from http://www.p-grid.org/.

In order to face the challenges of data organization, we follow the idea of the universal relation model allowing schema-independent query formulation. However, because exploiting the features of a DHT for fast lookups requires to index all attributes, we store data vertically, similar to the idea of RDF. If we assume relational data, each tuple $(OID, v_1, \ldots, v_n)$ of a given relation schema $R(A_1, \ldots, A_n)$ is stored in the form of n triples

$$(OID, A_1, v_1), \ldots, (OID, A_n, v_n)$$

where OID is a unique key, e.g., a URI, and the attribute names A_i may contain a namespace prefix ns which allows the user to distinguish different relations and avoid conflicts. Furthermore, the vertical storage supersedes the explicit representation of null values making the universal relation approach feasible even for heterogeneous data. Obviously, this data storage model is exactly the same layout as RDF – therefore RDF data can be stored seamlessly. Note that, though we use an OID field, we do not assume unique and homogeneous identifiers for all objects – instead the OID is system generated allowing to group the triples for a logical tuple.

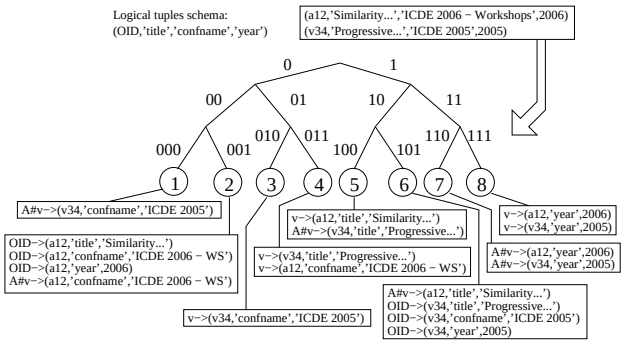

Figure 2: Example tuples in the triple store

By default, we index each triple on the OID, $A_i \# v_i$ (the concatenation of A_i and v_i), and v_i. This enables search based on the unique key, queries of the form $A_i \geq v_i$, and using v_i as the key for queries on an arbitrary attribute. Like this, efficient reproduction of origin data, as well as access to parts of special interest, is ensured in each situation. Figure 2 illustrates this for two example tuples,

each containing three attributes: 18 resulting triples are distributed in the network of 8 peers (corresponding hash keys are sketched: e.g., `OID->t` means triple `t` was inserted according to `hash(OID)`). Additionally, we allow to store triples representing a simple kind of schema mappings in order to overcome schema heterogeneities. This additional metadata can be queried explicitly by the user – or even automatically by the system to retrieve relevant data without needing the user to interact.

In order to support the formulation and processing of DB-like queries, we propose a structured query language VQL (Vertical Query Language), which is derived from SPARQL [8], and introduce an according logical algebra.

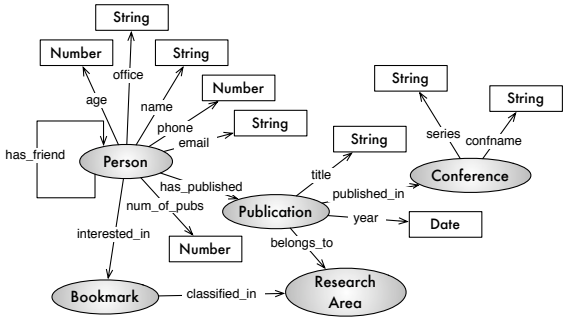

Figure 3: Example schema

In a VQL query, the targeted triples are formulated in braces, where variables are indicated by a question mark. Optional `FILTER` statements are used to provide filter predicates returned triples have to match. Further, the basic construct remembers the structure of SQL queries, including obligatory `SELECT` and `WHERE` blocks, optional statements like `ORDER BY` and `LIMIT`, as well as advanced ones like `SKYLINE OF`.

Imagine a simple example schema about authors and their publications as shown in figure 3. This schema could describe the data of one or more peers joining the system. The following example VQL query corresponds to this schema and exemplarily presents a part of the *full* supported query functionality:

```
SELECT ?name,?age,?cnt
WHERE {(?a,'name',?name) (?a,'age',?age)
       (?a,'num_of_pubs',?cnt)
       (?a,'has_published',?title) (?p,'title',?title)
       (?p,'published_in',?conf) (?c,'confname',?conf)
       (?c,'series',?sr) FILTER edist(?sr,'ICDE')<3
      }
ORDER BY SKYLINE OF ?age MIN, ?cnt MAX
```

The result of this query represents a skyline of authors that reaches from the youngest authors to those authors published the most publications, whereby we only consider authors published in ICDE series. Note that, for the name of the series we allow an edit distance of up to 2 to the term 'ICDE' in order to ignore typos and similar.

The algebra supports traditional "relational" operators ($\pi, \sigma, \bowtie, \ldots$) as well as special operators needed to query the distributed triple storage. Operators of both classes can be freely combined and are applicable to schema, instance and metadata level. Furthermore, in order to support large-scaled and heterogeneous data collections, we extend the set of operators by special operators like similarity operators (e.g., similarity join) and ranking operators (e.g., top-N, skyline). Similarity operations are an extremely important and essential part of a universal storage as we propose. By supporting similarity queries, among others, we provide tools for data integration tasks, which are finally due to the individual user. Despite the wide functionality, query formulation is based on a simple and small set of VQL clauses.

For each logical operator there are several physical implementations available and in development. Key lookups, range queries on key level and prefix search are supported by P-Grid. All of the implemented operators only rely on functionality provided by the overlay system. They differ in the kind of used indexes, applied routing strategy, parallelism, etc. For example, in [6] we introduced a q-gram index (q-gram: a substring of fixed length q) in order to be able to process string similarity efficiently.

The physical operators are used to build complex query plans. The processing of these plans can be described as an extension of the concept of *Mutant Query Plans* [7]. For each physical operator, and thus, for each query plan, we can determine worst-case guarantees (almost all are logarithmic) and predict exact costs [5]. We base these calculations on the characteristics of the used overlay system and the actual data distribution. By this, we derive a cost model for choosing concrete query plans, which is repeatedly applied at each peer involved in a query, resulting in an adaptive query processing approach.

For more details of the system, the underlying approach and aspects of physical query processing we refer to [5, 6].

3 Outstanding Features

The presented platform reveals some highlighting features that distinguish it from existing approaches, where the most outstanding are:

- it is based on a generic, self-descriptive and flexible schema for storing data and metadata analogously
- a light-weight query language leverages query expressiveness and allows simple formulation of structured queries with fuzzy and ranking predicates on both, schema and instance level
- there exist several implementations of physical operators, each beneficial in special situations – which is captured by an appropriate cost model
- operators can be applied to all levels of data (instance, schema and metadata, e.g., correspondences)
- we exploit powerful features of DHTs to create a robust, scalable and reliable massively distributed (up to 1000 peers and more) storage in arbitrary environments (even if they are unreliable and highly dynamic)
- the user interface is kept simple, but provides an intuitive way of full-featured usage

Besides the ability to overcome several burdens and challenges of a universal storage solution, our implementation reveals some other interesting features and benefits from a more technical point of view:

- it is extensible at ease, e.g., by new query processing strategies, replication approaches or indexing methods
- due to its logging capabilities results are traceable, analyzable and (in limits) repeatable
- its readily implemented for usage across multiple platforms, due to the utilization of Java and XML as the fundamental data model.

To the best of our knowledge, this is the first P2P platform attacking the challenges of a distributed universal storage in a concrete way, while being flexible, light-weight and ready for practical use.

4 Demonstration

Usage of our platform is quite simple. A user can download the software, start it by calling Java, and then he is provided with a simple but intuitive user interface. When joining (or even later), he decides if he wants to insert any data. Optional schema mapping data is inserted just the same, by defining and inserting according triples.

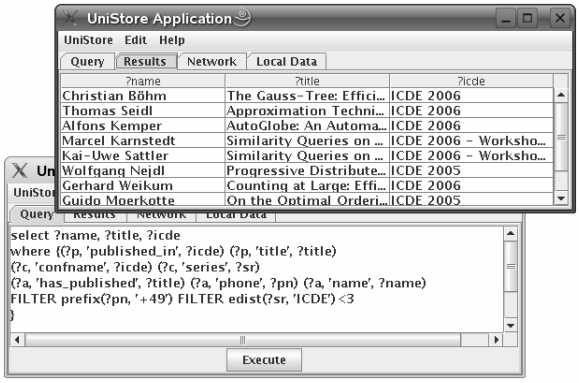

Figure 4: Example query and results

After successfully joining the system, the user can formulate VQL queries in a separate tabbed window, results will be displayed in the next tab. The basic interface is completed by the opportunities to inspect the local data and the locally built routing tables. Figure 4 shows screenshots of a query and according results.

At the conference we want to show the powerfulness of our approach in order to build a universal distributed storage and its applicability in praxis, which includes aspects like performance, scalability, robustness and usability. In order to achieve this, we plan to prepare some remote machines ready to share data. As a practical example, we decided to choose data about contacts and publications, similar to the schema introduced in section 2. A laptop at the conference will act as a demonstration peer, from where we can add and search for data. The platform is not limited to nor only intended to share contact data, rather this shall present one of many conceivable use cases. We plan to present the whole set of query formulation and processing capabilities, which includes (similarity) joins and filters, ranking operators like top-N and skylines, substring search, etc. on schema and instance level.

We will demonstrate the platform's ability to handle dynamicity and data heterogeneities by allowing interested people to include their own machines and/or data into a running (or even one built from scratch) P-Grid overlay. Like this, any new user will be able to experience the powerfulness and flexibility of our solution – while adhering to a simple but usable and intuitive user interface. With the will of the conference participants and their readiness to share personal data, this could even be extended to a conference data sharing system at ease. People could also insert data about restaurants, bars, sights or anything other that is conceivable – and apply queries intended for such distributed public data collections, e.g., skyline operators.

In order to show performance, scalability and robustness, we also plan to run a couple of peers on PlanetLab [3], if Internet access is available. PlanetLab is a world-wide consortium for evaluating large-scaled distributed applications, especially under the aspects of world-wide node distribution, changing load and network situations, as well as unreliable node behavior. We will show that even with up to 400 PlanetLab nodes query answer times are still only a couple of seconds. Additionally, we plan to demonstrate some benefits we earn from implementing different query processing strategies, routing techniques and indexing methods. In this context, we will execute identical queries sequentially while influencing the integrated optimizer and/or data placement, which will result in different performance results depending on the current data load, network state, etc.

References

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *CoopIS*, 2001.

[2] K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt. Indexing data-oriented overlay networks. In *VLDB*, 2005.

[3] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *SIGCOMM Comp. Comm. Rev.*, 33(3), 2003.

[4] A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *ICDCS*, 2003.

[5] M. Karnstedt, K.-U. Sattler, M. Hauswirth, and R. Schmidt. Cost-Aware Processing of Similarity Queries in Structured Overlays. In *IEEE P2P2006*, 2006.

[6] M. Karnstedt, K.-U. Sattler, M. Hauswirth, and R. Schmidt. Similarity Queries on Structured Data in Structured Overlays. In *NetDB'06*, 2006.

[7] V. Papadimos and D. Maier. Mutant Query Plans. *Information and Software Technology*, 44(4), 2002.

[8] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Cand. Recommendation 6 Apr 2006. http://www.w3.org/TR/rdf-sparql-query/.