Data Management in the APPA System

Reza Akbarinia, Vidal Martins

Data Management in the APPA System[1]

Reza Akbarinia[1,3],Vidal Martins[1,2]

[1]ATLAS group, INRIA and LINA, University of Nantes, France
[2]PPGIA/PUCPR – Pontifical Catholic University of Paraná, Brazil
[3]Shahid Bahonar University of Kerman, Iran
{FirstName.LastName@univ-nantes.fr }

Abstract. Combining Grid and P2P technologies can be exploited to provide high-level data sharing in large-scale distributed environments. However, this combination must deal with two hard problems: the scale of the network and the dynamic behavior of the nodes. In this paper, we present our solution in APPA (Atlas Peer-to-Peer Architecture), a data management system with high-level services for building large-scale distributed applications. We focus on data availability and data discovery which are two main requirements for implementing large-scale Grids. We have validated APPA's services through a combination of experimentation over Grid5000, which is a very large Grid experimental platform, and simulation using SimJava. The results show very good performance in terms of communication cost and response time.

1. Introduction

Grid technology has been successful at providing high-level resource sharing services for virtual organizations, typically formed by geographically distributed institutions and companies [12]. As Grid technology is evolving to support large-scale virtual organizations, *e.g.* with very large numbers of members, the requirements for data management get harder. Data management in Grids has been initially achieved using distributed file systems for scientific computing applications. Recently, in the context of the standard Open Grid Services Architecture (OGSA) [30], the need for high-level database access has been recognized. This led to the definition of OGSA-DAI [31], a service-based architecture for database access over the Grid. OGSA-DAI extends the distributed database architecture [32] to provide distribution transparency using Web services. However, as in distributed database systems, it relies on some centralized directory. This may make it inappropriate for virtual organizations which are highly dynamic, with large numbers of autonomous members which may join or leave the Grid very often. Examples of such dynamic virtual organizations include home users of a large image editing application, schools involved in a joint project, or small businesses organized as a federation. In these examples, the members may wish to collaborate simply using their individual machines without relying on a centralized Web site and database.

[1] Work partially funded by ARA “Massive Data” of the French ministry of research and the European Strep Grid4All project.

To support dynamic, scalable virtual organizations, the main requirements for Grid data management are to scale up to large numbers of nodes and support autonomic and dynamic behavior. To some extent, these requirements have been addressed by Peer-to-peer (P2P) systems which adopt a completely decentralized approach to data sharing. Popular examples of P2P systems such as Gnutella [15] and Freenet [11] have millions of users sharing petabytes of data over the Internet. However, most deployed P2P systems are quite simple (*e.g.* file sharing) and support limited functions (*e.g.* keyword search). Most of the research on P2P systems has focused on dealing with the dynamic behavior of nodes (also called peers) and improving the performance of query routing in the unstructured systems which rely on flooding. This work led to structured solutions based on distributed hash tables (DHT), *e.g.* CAN [34] and Chord [39], or hybrid solutions with super-peers that index subsets of peers [27].

The complementary nature of Grid and P2P computing suggests that the two are likely to converge over time [13]. Grids can take advantage of P2P techniques to support large-scale and dynamic virtual organizations. On the other hand, P2P systems can exploit Grid techniques to support high-level services, deal with semantically rich data (*e.g.* XML documents, relational tables, etc.), provide a more secure and trusted environment for users, etc. Following this convergence, P-Grid [1] and Organic Grid [8] propose self-organizing and scalable Grid services on top of a P2P network. The expected result of such convergence is a new class of technologies which address scalability, high data availability, and self organization, while providing a persistent and standardized infrastructure for advanced applications.

Such convergence is also having impact on Grid standardization. One problem with OGSA is that it does not support the dynamic behavior of nodes, which is typical of P2P. For instance, a node's IP address may change due to mobility or firewall network address translation. To support the specific features of P2P, OGSA-P2P [29] has been recently proposed to revisit OGSA: scale up, dynamic data discovery, data availability, group support, location awareness, security, and connectivity.

Providing an infrastructure for advanced data management in large-scale Grid or P2P systems is quite challenging because of the scale of the network and the autonomy and unreliable nature of nodes. Most techniques designed for distributed database systems which statically exploit schema and network information no longer apply. New techniques are needed which should be decentralized, dynamic and self-adaptive and satisfy the requirements of dynamic Grids.

In this paper, we present our solution in APPA (Atlas Peer-to-Peer Architecture), a data management system for large-scale P2P and Grid applications. APPA has a network-independent architecture that can be implemented over various overlay networks. This allows us to exploit continuing progress in such networks. APPA uses novel solutions to provide high level data management services in large-scale distributed environments. We focus on data availability and data discovery which are two main requirements for supporting OGSA-P2P. We have validated APPA's services through a combination of experimentation over Grid5000 [17], a very large Grid experimentation platform in France, and simulation using SimJava [18]. Furthermore, we have implemented APPA on top of JXTA [19] and other P2P networks such as CAN [34] and Chord [39]. The results show very good performance in terms of communication cost and response time.

This work is done in the context of the Grid4All European project [16] whose goal is to democratize Grid technology by enabling all kinds of users (*e.g.* domestic users, schools, small enterprises) to share their resources. To deal with dynamicity, autonomy and scaling issues, Grid4All uses P2P techniques.

The rest of the paper is organized as follows. Section 2 describes the APPA architecture. Section 3 introduces APPA's solution to persistent data management and support for updates. Section 4 describes high-level data replication and distributed semantic reconciliation. Section 5 describes query processing in APPA. In Section 6, we first describe the validation of APPA over JXTA, Chord and CAN, and then we present a performance evaluation of APPA's services through experimentation and simulation. Section 7 discusses related work. Section 8 concludes.

2. APPA Architecture

APPA (Atlas P2P Architecture) has a layered service-based architecture (see Figure 1). Besides the traditional advantages of using services (encapsulation, reuse, portability, etc.), this enables APPA to be network-independent so it can be implemented over different structured (*e.g.* DHT) and super-peer P2P networks. The main reason for this choice is to be able to exploit rapid and continuing progress in such networks. Another reason is that it is unlikely that a single network design will be able to address the specific requirements of many different Grid applications. Obviously, different implementations will yield different trade-offs between performance, fault-tolerance, scalability, quality of service, etc. For instance, fault-tolerance can be higher in DHTs because no node is a single point of failure. On the other hand, through index servers, super-peer systems enable more efficient query processing. Furthermore, different P2P networks could be combined in order to exploit their relative advantages, *e.g.* DHT for key-based search and super-peer for more complex searching.

There are three layers of services in APPA: P2P network, basic services and advanced services.

P2P network. This layer provides network independence with services that are common to different P2P networks:

- **Peer id assignment:** assigns a unique id to a peer using a specific method, *e.g.* a combination of super-peer id and counter in a super-peer network.
- **Peer linking:** links a peer to some other peers, *e.g.* by locating a zone in CAN.
- **Key-based storage and retrieval (KSR):** stores and retrieves a (*key*, *data*) pair in the P2P network, *e.g.* through hashing over all peers in DHT networks or using super-peers in super-peer networks. An important aspect of KSR is that it allows managing data using object semantics (*i.e.* with KSR it is possible to get and set specific data attributes).
- **Key-based timestamping (KTS):** generates monotonically increasing timestamps which are used for ordering the events occurred in the P2P system. This service is useful to improve data availability.

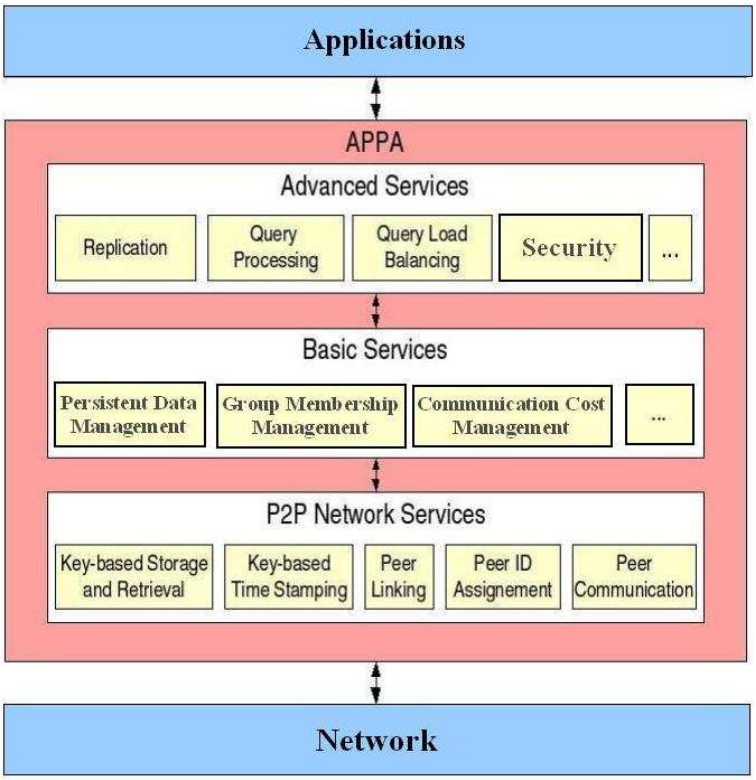

Fig. 1. APPA Architecture

- **Peer communication:** enables peers to exchange messages. It also allows a peer to call a remote service, *e.g.* a Web service using SOAP, which is provided by another peer over the P2P network.

Basic services. This layer provides elementary services for the advanced services using the P2P network layer:

- **Persistent data management (PDM):** provides high availability for the *(key, data)* pairs which are stored in the P2P network.
- **Communication cost management:** estimates the communication costs for accessing a set of data items that are stored in the P2P network. These costs are computed based on latency and transfer rates, and they are refreshed according to the dynamic connections and disconnections of nodes.
- **Group membership management:** allows peers to join an abstract *group*, become *members* of the group and send and receive membership notifications. This is similar to group communication [7][10].

Advanced services. This layer provides advanced services for semantically rich data sharing including schema management, replication [24][25] , query processing [2][3], security, etc. using the basic services.

APPA provides support for the requirements specified by OGSA-P2P as follows:

- **Scale up:** this is the most important requirement of a P2P system and is met by all services of APPA.
- **Dynamic data discovery:** it is needed for looking up the desired data in the Grid system. In APPA, this requirement is supported mainly by the Query Processing service.
- **Data availability:** P2P environments are very dynamic, and the nodes may leave the system at any time, thereby the data stored at them get unavailable. So, we

need some mechanisms for improving data availability. In APPA, this requirement is satisfied by the PDM and Replication services.

- **Group support:** groups are an essential mechanism to collect and aggregate a set of resources or users with common characteristics together. In APPA, the Group Membership Management service provides support for groups.
- **Location awareness:** it allows the applications to use location information to optimize their communications over the network. In APPA, the Communication Cost Management service provides support for location awareness.
- **Security:** P2P systems bring a set of unique notions of trust and security requirements which must be dealt with. In APPA, the Security service is responsible for satisfying these requirements. The work on this service is ongoing and we are using mechanisms such as those proposed in [6] and [38].
- **Connectivity:** to enable decentralized sharing of computing resources, collaborative workspaces, information and services, it is necessary for the nodes at the edge of the network to communicate with each other and with the services at the heart of the network. In APPA, this requirement is supported by the Peer Linking and Peer Communication services.

3. Persistent Data Management

One of the main characteristics of the systems we address is the dynamic behavior of nodes which can join and leave the system frequently, at anytime. When a node gets offline, the data it stores becomes unavailable. To improve data persistence, we can rely on data replication by storing (*k, data*) pairs at several nodes. If one node is unavailable, the data can still be retrieved from the other nodes that hold a replica. However, the mutual consistency of the replicas after updates can be compromised as a result of nodes leaving the network or concurrent updates. Therefore, some of the replicas may not be *current, i.e.* they do not reflect the latest data stored with *k* in the network. For some applications (*e.g.* agenda management, bulletin boards, cooperative auction management, reservation management, etc.) having the ability to get a current replica is very important.

In APPA, the PDM service provides data persistence through replication by using multiple hash functions. It also addresses efficiently the problem of retrieving current replicas based on timestamping. For doing its tasks, PDM takes advantage of KSR and KTS which are two services in the lower layer of APPA architecture.

In this section, we first discuss how PDM provides data persistence, then we introduce the concept of timestamping, and finally we present the update operations which are the main operations of the PDM service.

3.1 Data Persistence Using Multiple Hash Functions

In APPA, the KSR service maps a key *k* to a node *p* using a hash function *h*. We call *p* the *responsible for k wrt. h*, and denote it by *rsp(k, h)*. A node may be responsible for *k wrt.* a hash function h_1 but not responsible for *k wrt.* another hash function h_2.

There is a set of hash functions H which can be used for mapping the keys to nodes. The KSR service has an operation $put_h(k, data)$ that, given a hash function $h \in H$, a data item *data* and its associated key k, stores the pair $(k, data)$ at $rsp(k,h)$. This operation can be issued concurrently by several nodes. There is another operation $get_h(k)$ that retrieves the data associated with k stored at $rsp(k,h)$.

To improve data persistence, PDM stores each data and its associated key at several nodes using a set of hash functions $H_r \subset H$. the set H_r is called the set of *replication hash functions*. The number of replication hash functions, *i.e.* $/H_r/$, can be different for different networks. For instance, in a P2P network with low node's availability, data availability can be increased using a high value of $/H_r/$ (*e.g.* 20).

Over time, some of the replicas stored with k at some nodes may get stale, *e.g.* due to the absence of some nodes at update time. To be able to return current replicas, before storing a data, PDM "stamps" it with a logical timestamp which is generated by KTS. Therefore, given a data item *data* and its associated key k, $\forall h \in H_r$, PDM replicates the pair $(k, \{data, timestamp\})$ at $rsp(k,h)$. Upon a request for the data associated with a key, PDM returns one of the replicas which are stamped with the latest timestamp.

3.2 Timestamping

To generate timestamps, APPA uses KTS which is a distributed service. The main operation of KTS is $gen\_ts(k)$ which, given a key k, generates a real number as a *timestamp for k*. The timestamps generated by KTS have the *monotonicity* property, *i.e.* two timestamps generated for the same key are monotonically increasing. In other words, for any two timestamps ts_1 and ts_2 generated for a key k respectively at times t_1 and t_2, if $t_1 < t_2$ then we have $ts_1 < ts_2$. This property permits us to order the timestamps generated for the same key according to the time at which they have been generated.

KTS generates the timestamps in a completely distributed fashion, using local logical counters. At anytime, it generates at most one timestamp for a key k. Thus, regarding the monotonicity property, there is a total order on the set of timestamps generated for the same key. However, there is no total order on the timestamps generated for different keys. In addition to *gen_ts*, KTS has another operation denoted by $last\_ts(k)$ which, given a key k, returns the last timestamp generated for k by KTS.

The idea of timestamping by KTS is like the idea of data storage in DHTs which is based on having a responsible for storing each data and determining the responsible dynamically using a hash function. In KTS, for each key, there is a responsible of timestamping which is determined dynamically using a hash function. Due to space limitations, we don not describe the details of KTS.

3.3 Update Operations

The main operations of the PDM service are *insert* and *retrieve* operations. The detail of these operations is as follows.

Insert(k, data): replicates a data and its associated key in the network as follows. First, it uses KTS to generate a timestamp for k, *e.g.* ts. Then, for each $h \in H_r$ it stores

the pair *(k, {data, ts})* at the node that is *rsp(k,h)*. When a node *p*, which is responsible for *k wrt.* one of the hash functions involved in H_r, receives the pair *(k, {data, ts})*, it compares *ts* with the timestamp, say ts_0, of its data (if any) associated with *k*. If $ts > ts_0$, *p* overwrites its data and timestamp with the new ones. Recall that, at anytime, *KTS.gen_ts (k)* generates at most one timestamp for *k*, and different timestamps for *k* have the monotonicity property. Thus, in the case of concurrent calls to *insert(k, data), i.e.* from different nodes, only the one that obtains the latest timestamp will succeed to store its data in the network.

Retrieve(k): retrieves the most recent replica associated with *k* in the network as follows. First, it uses KTS to determine the latest timestamp generated for *k*, *e.g.* ts_1. Then, for each hash function $h \in H_r$, it uses the KSR operation $get_h(k)$ to retrieve the pair *{data, timestamp}* stored along with *k* at *rsp(k,h)*. If *timestamp* is equal to ts_1, then the data is a current replica which is returned as output and the operation ends. Otherwise, the retrieval process continues while saving in $data_{mr}$ the most recent replica. If no replica with a timestamp equal to ts_1 is found (*i.e.* no current replica is found) then the operation returns the most recent replica available, *i.e.* $data_{mr}$.

4. Data Replication

Data replication is largely used to improve data availability and performance in distributed systems. In APPA, PDM is a low-level service that employs data replication to improve the availability of pairs (*key*, *data*) stored in the network. For solving update conflicts by taking into account application semantics, APPA provides a higher-level replication service. This service is an optimistic solution [35] that allows the asynchronous updating of replicas such that applications can progress even though some nodes are disconnected or have failed. As a result, users can collaborate asynchronously. However, concurrent updates may cause replica divergence and conflicts, which should be reconciled.

In this section, we present the DSR algorithm (Distributed Semantic Reconciliation) [24][25], a dynamic distributed version of the semantic reconciliation provided by IceCube [21][33]. Unlike IceCube, DSR is based on a distributed and parallel approach. With DSR, a subset of nodes, called reconcilers, are selected to concurrently reconcile conflicting updates. DSR works properly over clusters, P2P, and Grid systems (*e.g.* we have implemented a DSR prototype [24] and validated it on the Grid5000 platform). We now describe the main terms and assumptions we consider for DSR followed by the main DSR algorithm itself.

We assume that DSR is used in the context of a virtual community which requires a high level of collaboration and relies on a reasonable number of nodes (typically hundreds or even thousands of interacting users) [45].

In our solution, an *object* is the minimal unit of replication in a system, *i.e.* it can be a relational table, an XML document, etc. We call *object item* a component of the object, *e.g.* a tuple in a relational table or an element in an XML document. A *replica* is a copy of an object (*e.g.* copy of a relational table or XML document) while a *replica item* is a copy of an object item (*e.g.* a copy of a tuple or XML element). We assume *multi-master* replication, *i.e.* multiple replicas of an object *R*, noted R_1, R_2, …,

R_n, are stored in different nodes which can read or write R_1, R_2, …, R_n. Conflicting updates are expected, but it is assumed that the application tolerates some level of replica divergence until reconciliation.

In order to update replicas, nodes produce *tentative* actions (henceforth actions) that are executed only if they conform to the application semantics. An *action* is defined by the application programmer and represents an application-specific operation (*e.g.* a write operation on a file or document, or a database transaction). The application semantics is described by means of constraints between actions. A *constraint* is the formal representation of an application invariant (*e.g.* an update cannot follow a delete).

On the one hand, users and applications can create constraints between actions to make their intents explicit (they are called *user-defined constraints*). On the other hand, the reconciler node identifies conflicting actions, and asks the application if these actions may be executed together in any order (*commutative* actions) or if they are mutually dependent. New constraints are created to represent semantic dependencies between conflicting actions (they are called *system-defined constraints*). Details about the language used to express constraints can be found in [33].

A *cluster* is a set of actions related by constraints, and a *schedule* is an ordered list of actions that do not violate constraints.

With DSR, data replication proceeds basically as follows. First, nodes execute local actions to update replicas while respecting user-defined constraints. Then, these actions (with the associated constraints) are stored in the network using the PDM service. Finally, reconciler nodes retrieve actions and constraints from the network and produce a global schedule, by performing conflict resolution in 5 distributed steps based on the application semantics. This schedule is locally executed at every node, thereby assuring eventual consistency [33]. The replicated data is eventually consistent if, when all nodes stop the production of new actions, all nodes will eventually reach the same value in their local replicas.

In order to avoid communication overhead and due to dynamic connections and disconnections, we distinguish *replica nodes*, which are the nodes that hold replicas, from *reconciler nodes*, which is a subset of the replica nodes that participate in distributed reconciliation.

We now present DSR in more details. We first introduce the reconciliation objects necessary to DSR. Then, we present the five steps of the DSR algorithm. Finally, we describe how DSR deals with dynamic connections and disconnections.

4.1 Reconciliation Objects

Data managed by DSR during reconciliation are held by *reconciliation objects* that are stored in the network giving the object identifier. To enable the storage and retrieval of reconciliation objects, each reconciliation object has a unique identifier. DSR uses five reconciliation objects:

- **Action log R (noted L_R)**: it holds all actions that try to update any replica (noted R_1, R_2, …, R_n) of the object R.
- **Action groups of R (noted G_R)**: actions that manage a common object item are put together into the same action group in order to enable the parallel checking of

semantic conflicts among actions (each action group can be checked independently of the others); every object *R* may have a set of action groups, which are stored in the *action groups of R* reconciliation object.

- **Clusters set (noted *CS*)**: all clusters produced during reconciliation are included in the *clusters set* reconciliation object; a cluster is not associated with an object.
- **Action summary (noted *AS*)**: it comprises constraints and action memberships (an action is a *member* of one or more clusters).
- **Schedule (noted *S*)**: it contains an ordered list of actions.

The node that holds a reconciliation object is called the *provider node* for that object (*e.g. schedule provider* is the node that currently holds *S*).

4.2 DSR Algorithm

DSR executes reconciliation in 5 distributed steps as shown in Figure 2.

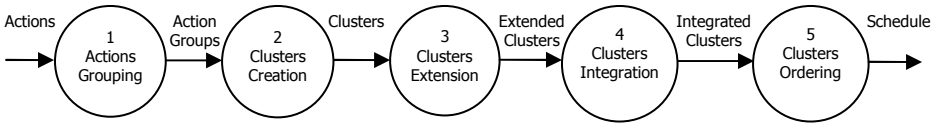

Fig. 2. DSR Steps

- **Step 1 – actions grouping**: for each object *R*, reconcilers put actions that try to update common object items of *R* into the same group, thereby producing G_R.
- **Step 2 – clusters creation**: reconcilers split action groups into clusters of semantically dependent conflicting actions (two actions a_1 and a_2 are *semantically independent* if the application judge safe to execute them together, in any order, even if a_1 and a_2 update a common object item; otherwise, a_1 and a_2 are *semantically dependent*). Clusters produced in this step are stored in the clusters set, and the associated action memberships are included in the action summary.
- **Step 3 – clusters extension**: user-defined constraints are not taken into account in clusters creation. Thus, in this step, reconcilers extend clusters by adding to them new conflicting actions, according to user-defined constraints. The associated action memberships are also included in the action summary.
- **Step 4 – clusters integration**: clusters extensions lead to the overlap of clusters' actions (an overlap occurs when different clusters have common actions, and this is identified by analyzing action memberships). In this step, reconcilers bring together overlapping clusters, thereby producing integrated clusters.
- **Step 5 – clusters ordering**: in this step, reconcilers produce the global schedule by ordering actions of integrated clusters; all replica nodes execute this schedule.

At every step, the DSR algorithm takes advantage of data parallelism, *i.e.* several nodes perform simultaneously independent activities on a distinct subset of actions (*e.g.* ordering of different clusters). No centralized criterion is applied to partition actions. In fact, whenever a set of reconciler nodes request data to a provider, the provider node naively supplies reconcilers with about the same amount of data (the provider node knows the maximal number of reconcilers because it receives this information from the node that launches reconciliation).

DSR avoids network overhead by minimizing the number of exchanged messages and the amount of transferred data. The number of messages is linear *wrt*. the number of reconcilers, and the number of reconcilers is not large. Concerning data transfer, most of messages carry only data identifiers (*e.g.* actions' identifiers) instead of the entire data items.

4.3 Managing Dynamic Disconnections and Reconnections

Whenever distributed reconciliation takes place, a set of nodes N_d may be disconnected. As a result, the global schedule is not applied by nodes of N_d. Moreover, actions produced by N_d nodes and not yet stored in the network via APPA PDM service are not reconciled. In order to assure eventual consistency despite disconnections, the APPA replication service proceeds as follows. Each node locally stores the identifier of the last schedule it has locally executed (noted S_{last}). In addition, the replication service stores in the network (using the APPA PDM service) a chronological sequence of schedules' identifiers produced by reconciliations, which is called *schedule history* and noted $H = (S_{id1}, S_{id2}, \ldots, S_{idn})$. As any reconciliation object, the schedule history has a unique identifier. The application knows this identifier and can provide it to the reconciler nodes. When a node n of N_d reconnects, it proceeds as follows: (1) n checks whether S_{last} is equal to S_{idn}, and, if not (*i.e.* n's replicas are out of date), n locally applies all schedules that follow S_{last} in the H history; (2) actions locally produced by n and not yet stored in the network using the APPA PDM service are put into the involved action logs for later reconciliation.

At the beginning of reconciliation, a set of connected replica nodes must be allocated to proceed as reconciler nodes. To minimize reconciliation time, such allocation should be dynamic, *i.e.* nodes should be allocated based on the reconciliation context (*e.g.* number of actions, number of replicas, network properties, etc.). We elaborated a cost model and the associated algorithms for allocating reconciler nodes based on communication costs [25][26]. These algorithms take into account cost changes due to dynamic disconnections and reconnections.

5. Query Processing

Query processing in APPA deals with schema-based queries and considers data replication. In this section, we first present schema mapping in APPA, and then we describe the main phases of query processing. We also introduce support for Top-k queries as a way to reduce network communication.

5.1 Schema Mapping

In order to support schema-based queries, APPA must deal with heterogeneous schema management. In systems composed of autonomous nodes, a node should be able to express queries over its own schema without relying on a centralized global schema as in data integration systems [40][43]. Several solutions have been proposed

to support decentralized schema mapping, *e.g.* [27][41]. For instance, Piazza [41] proposes a general, network-independent, solution that supports a graph of pair-wise mappings between heterogeneous node schemas. APPA uses a simpler solution that takes advantage of the collaborative nature of the applications. It assumes that nodes that wish to cooperate, *e.g.* for the duration of an experiment, agree on a *Common Schema Description* (CSD). Given a CSD, a node schema can be specified using views. This is similar to the local-as-view approach in data integration [23] except that, in APPA, queries at a node are expressed against the views, not the CSD.

When a node decides to share data, it needs to define a node schema, only once, to map its local schema to the CSD. To simplify the discussion, we use the relational model (APPA uses XML) and the Datalog-like notation of [40] for mapping rules. Thus, a node schema includes node mappings, one per local relation. Given 2 CSD relation definitions r_1 and r_2, an example of node mapping at node p is:

$p{:}r(A,B,D) \subseteq csd{:}r_1(A,B,C), csd{:}r_2(C,D,E)$

In APPA, mapped schemas are stored in the network using the PDM service.

5.2 Query Processing Phases

Given a user query on a node schema, the objective is to find the minimum set of relevant nodes (query matching), route the query to these nodes (query routing), collect the answers and return a (ranked) list of answers to the user. Since the relevant nodes may be disconnected, the returned answers may be incomplete.

Query processing proceeds in four main phases: (1) query reformulation, (2) query matching, (3) query optimization and (4) query decomposition and execution.

Query reformulation. The user query (on the node schema) is rewritten in a query on CSD relations. This is similar to query modification using views. For instance, the following query at node p:

select A,D from r where B=b

would be rewritten on the CSD relations as:

select A,D from r_1, r_2 *where* $B=b$ *and* $r_1.C=r_2.C$

Query matching. Given a reformulated query Q, it finds all the nodes that have data relevant to the query. For simplicity, we assume conjunctive queries. Let P be the set of nodes in the system, the problem is to find $P' \subseteq P$ where each p in P' has relevant data, *i.e.* refers to relations of Q in its mapped schema. These nodes can be iteratively (for each Q's relation) retrieved using the PDM service. Let R be the set of relations involved in Q, and $ms(p,r)$ denote that the mapped schema of node p involves relation r, query matching produces:

$P' = \{ p \mid (p \in P) \wedge (\exists r \in R \wedge ms(p,r)) \}$

Query optimization. Because of data replication, each relevant data may be replicated at some nodes in P'. The optimization objective is to minimize the cost of query processing by selecting best candidate node(s) for each relevant data based on a cost function. Selecting more than one candidate node is necessary in a very dynamic environment since some candidate nodes may have left the network. Thus, selecting several candidate nodes increases the answer's completeness but at the expense of redundant work. This step produces a set $P'' \subseteq P'$ of best nodes.

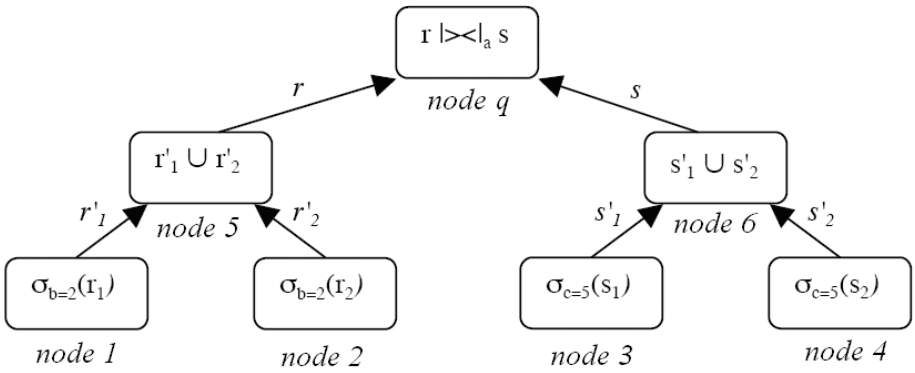

Fig. 3. Example of parallel execution using intermediate nodes. This strategy exhibits independent parallelism between nodes 1-4 (the select (σ) operations can all be done in parallel) and nodes 5-6 (the union operations can be done in parallel). It can also yield pipelined parallelism. For instance, if the left-hand operand of an intermediate node is smaller than the right-hand operand, then it would be entirely transferred first so the other operand could be pipelined thus yielding parallelism between nodes 2-5-*q* and nodes 4-6-*q*. Parallel execution strategies improve both the query response time and the global efficiency of the system.

Query decomposition and execution. This phase is similar to that in data integration systems and APPA reuses well-known, yet sophisticated techniques. Since some nodes in *P"* may have only subsets of *Q*'s relations, query decomposition produces a number of subqueries (not necessarily different), one for each node, together with a composition query to integrate, *e.g.* through join and union operations, the intermediate results [23]. Finally, the subqueries are sent to the nodes in *P"*, which reformulate it on their local schema (using the node mappings), execute it, and send the results back to the sending node, who integrates the results. Result composition can also exploit parallelism [44] using intermediate nodes. For instance, let us consider relations r_1 and r_2 defined over CSD *r* and relations s_1 and s_2 defined over CSD *s*, each stored at a different node, and the query *select * from r, s where r.a=s.a and r.b=2 and s.c=5* issued by a node *q*. A parallel execution strategy for *Q* is shown in Figure 3.

5.3 Top-k Queries

High-level queries over a large-scale distributed system may produce very large numbers of results that may overwhelm the users. To avoid such overwhelming, APPA uses Top-k queries whereby the user can specify a limited number (*k*) of the most relevant answers [2]. For example, consider a Grid system with medical doctors who want to share some (restricted) patient data for an epidemiological study. Then, one doctor may want to submit the following query over the system to obtain the 10 top answers ranked by a scoring function over age and weight:

```
SELECT   *
FROM     Patient P
WHERE    (P.disease = "hepathitis") AND
```

(P.age < 50) AND (P.weight > 70)
ORDER BY scoring-function(age, weight)
STOP AFTER 10

The scoring function specifies how closely each data item matches the conditions. For relational data, the most used scoring functions are *Min*, *Euclidean* and *Sum* functions [9]. For instance, in the query above, the scoring function could be *sum*((age/10)*2,weight/20) thus giving more importance to age.

Formally, let *Q* be a Top-k query and *P"* the set of nodes that have relevant data to *Q*. Let *D* be the set of all relevant data items (*i.e.* tuples) that are owned by the nodes involved in *P"*. Let *Sc(d, Q)* be a scoring function that denotes the score of relevance of a data item $d \in D$ to *Q*. The goal is to find the set $T \subseteq D$, such that: $|T| = k$ and $\forall d_1 \in T, \forall d_2 \in (D - T)$ then $Sc(d_1, Q) \geq Sc(d_2, Q)$.

Efficient execution of Top-k queries in a large-scale distributed system is difficult. To process a Top-k query, a naïve solution is that the query originator sends the query to all nodes and merges all the results, which it gets back. This solution hurts response time as the central node is a bottleneck and does not scale up. APPA takes advantage of parallelism and executes Top-k queries by a tree-based algorithm, in which several nodes participate in merging the results and bubbling up the top results to the query originator.

6. APPA Validation

To validate the design of APPA and perform experiments with collaborative applications, we have developed a prototype on top of JXTA, Chord and CAN. In this section, we describe APPA's implementation. Then, we report on the results of performance evaluation which was done through experimentation and simulation.

6.1 APPA over JXTA

JXTA (JuXTAposition) is an open network computing platform designed for P2P computing [19]. JXTA provides various services and abstractions for implementing P2P applications. Furthermore, it can integrate with Web service standards to provide higher-level peer-to-peer communication. Since Grid standards (OGSA and OGSA-P2P) rely on Web services, using JXTA is a good basis for building Grids. JXTA protocols aim to establish a network overlay on top of the Internet and non-IP networks, allowing nodes to directly interact and self-organize independently of their physical network. JXTA technology leverages open standards like XML, Java technology, and key operating system concepts. By using existing, proven technologies and concepts, the objective is to yield a P2P system that is familiar to developers.

JXTA provides a good support for the APPA's P2P Network services. The functionality provided by APPA's peer id assignment, peer linking, and peer communication service are already available in the JXTA core layer. Thus, APPA simply uses JXTA's corresponding functionality. In contrast, JXTA does not provide an equivalent service for key-based storage and retrieval (KSR). Thus, we

implemented KSR on top of Meteor which is an open-source JXTA service. Also, for implementing the KTS service, we use Meteor. APPA's advanced services, like replication and query processing, are provided as JXTA community services. The key advantage of APPA's implementation is that only its P2P network layer depends on the JXTA platform. Thus, APPA is portable and can be used over other platforms by replacing the services of the P2P network layer.

6.2 APPA over Chord and CAN

In addition to JXTA and to further validate APPA's network independence, we have implemented APPA's services over two of the most known DHTs, Chord and CAN. Most of the APPA's services can be easily implemented over Chord and CAN, in particular the KSR and KTS services.

Chord is a simple and efficient DHT. It can lookup a data, which is stored at some node in the network, in $O(\log n)$ routing hops where n is the number of nodes. A Chord node requires information about $\log (n)$ other nodes for efficient routing. Chord has an effective algorithm for maintaining this information in a dynamic environment. Its lookup mechanism is provably robust in the face of frequent node failures and re-joins, and it can answer queries even if the system is continuously changing.

CAN (Content Addressable Network) is based on a logical d-dimensional Cartesian coordinate space, which is partitioned into hyper-rectangles, called zones. Each node in the system is responsible for a zone, and a node is identified by the boundaries of its zone. A data is hashed to a point in the coordinate space, and it is stored at the node whose zone contains the point's coordinates. Each node maintains information about all its neighbors, *i.e.* $2*d$ neighbors. The lookup operation is implemented by forwarding the message along a path that approximates the straight line in the coordinate space from the sender to the node storing the data. In CAN, a stored data can be retrieved in $O(dn^{1/d})$ where n is the number of nodes.

The performance of APPA's services over Chord corresponds qualitatively with their performance over CAN. However, there are some quantitative differences in performance because of inherent differences in the protocols of Chord and CAN. For example, the KSR service is more efficient over Chord than CAN. In contrast, communicating messages between neighbors, which is supported by the Communication Management service, is more efficient over CAN because in CAN the nodes' neighborhood is organized according to communication latencies.

6.3 Performance Evaluation

We evaluated the performance of APPA's advanced services through experimentation and simulation. The experimentation over Grid5000 was useful to validate services and calibrate our simulator. The simulator allows us to scale up to high numbers of nodes. In this section, we first describe our experimental and simulation setup, and then we report the main performance evaluation results which we observed during our tests.

6.3.1 Experimental and Simulation Setup

We validated APPA's services (*e.g.* KSR, KTS, PDM and Replication) over the Grid5000 platform [17]. Grid5000 aims at building a highly reconfigurable, controllable and monitorable experimental Grid platform, gathering 9 sites geographically distributed in France featuring a total of 5000 nodes. Within each site, the nodes are located in the same geographic area and communicate through Gigabyte Ethernet links as clusters. Communications between clusters are made through the French academic network (RENATER). Grid5000's nodes are accessible through the OAR batch scheduler, from a central user interface shared by all the users of the Grid. A cross-clusters super-batch system, OARGrid, is currently being deployed and tested. The home directories of the users are mounted with NFS on each of the infrastructure's clusters. Data can thus be directly accessed inside a cluster. Data transfers between clusters have to be handled by the users. The storage capacity inside each cluster is a couple of hundreds of gigabytes.

To have a topology close to P2P overlay networks, we determine the nodes' neighbors and we allow that every node communicate only with its neighbors in the overlay network. Additionally, in order to study the scalability of these services with larger numbers of nodes, we implemented simulators using Java and SimJava [18] (a process based discrete event simulation package). Simulations were executed on an Intel Pentium IV with a 2.6 GHz processor, and 1 GB of main memory, running the Windows XP operating system.

Performing tests over GRID5000 has been easier than over a P2P network because Grid5000 is much more controllable. For example to test a new version of a service, we only need to reserve the required number of nodes, deploy the service over the nodes and execute the test program. But, in a P2P network it is more difficult to do so because of the dynamic nature of peers, *e.g.* some of peers may leave the system during the execution of the test program. Our tests showed that the APPA's service can work well over both Grid and P2P networks, although there are some quantitative differences in performance, *i.e.* the performance of the services over Grid5000 is better than over a typical P2P network because of the high speed communication network.

6.3.2 Main Results

In this section, we briefly report on the main performance evaluation results which we observed during our tests on the performance of APPA's services. More details can be found in [2][3][24][25][26].

We evaluated the scalability of the PDM and KTS services through simulation over a large number of nodes using SimJava. We compared the performance of PDM and BRK (from the BRICK project [22]) which we used as baseline algorithm. The experimental and simulation results show that using KTS, PDM achieves major performance gains, in terms of response time and communication cost, compared with BRK. The response time and communication cost of PDM grow logarithmically with the number of nodes of the system. Increasing the number of replicas of each data increases very slightly the response time and communication cost of PDM. In

addition, even with a high number of node failures, PDM still works well. We have done our tests in an environment where the lifetime of nodes is low. However, the simulation results show that increasing nodes' lifetime increases the performance of PDM.

We also evaluated the query processing service through experimentation and simulation. The results show very good performance, in terms of communication cost and response time. The response time and communication cost of the query processing service grow logarithmically with the number of nodes of the network. For top-k queries, we studied the effect of several parameters (*e.g.* number of nodes, number of requested answers, etc.) on the performance of the query processing service. The results show very good performance, in terms of communication cost and response time. For instance, increasing the number of requested answers, *i.e. k*, increases very slightly the response time of our algorithm.

In addition, we validated our semantic reconciliation solution through experimentation and simulation. Our algorithms take into account the communication costs for selecting the best reconciler nodes. For computing communication costs, we use local information and we deal with the dynamic behavior of nodes. We also limit the scope of event propagation (*e.g.* joins or leaves) in order to avoid network overload. We compared the performance of reconciliation using random selection of reconcilers and cost-based selection. The experimental results showed that the cost-based reconciliation outperforms the random approach by a factor of 26. In addition, the number of connected nodes does no affect the performance of cost-based reconciliation since the reconciler nodes are as close as possible to the reconciliation objects. Compared with the IceCube's centralized solution, our algorithm yields high data availability and excellent scalability, with acceptable performance and limited overhead.

7. Related Work

Data grid applications need to access, share, manage and integrate massive amounts of data distributed across heterogeneous and geographically spread Grid resources. The main work in this area has been on providing data access and integration services for the Grids with a relatively low dynamicity and moderated scale. The following research efforts are good representatives of such work.

The Spitfire project [42] in the European Data Grid Project provides a means to access relational databases on the Grid. It is a very thin layer on top of an RDBMS (by default MySQL) that provides a JDBC driver. It uses Web Service technology (Jakarta Tomcat) to provide SOAP-based RPC (through Apache Axis) to a few user-definable database operations.

The Open Grid Services Architecture Data Access and Integration (OGSA-DAI) [31] [5] is another project concerned with constructing middleware to assist with access and integration of data shared over the Grid, using Web services. It is engaged in identifying the requirements, designing solutions and delivering software that will meet this purpose. The project is working closely with the Global Grid Forum DAIS-WG [31] and the Globus team [14]. OGSA-DAI software currently supports the

exposure of data resources, such as relational or XML databases, over Grids. Various interfaces are provided and many popular database management systems are supported. The software also includes a collection of components for querying, transforming and delivering data in different ways, and a simple toolkit for developing client applications. One component is Distributed Query Processing (OGSA-DQP) that deals with processing queries over OGSA-DAI data services and over other services available on the Grid. OGSA-DQP adapts techniques from parallel databases to provide implicit parallelism for complex data-intensive queries.

The two above projects deal with data access and integration services by adapting distributed database technology [32] to the Grid using Web services. However, they do not address some issues which arise in large-scale and dynamic environments and which are important for data management in these environments, *e.g.* low data availability.

Grid-DBMS [4] deals with dynamically managing data sources in Grid environments. It automatically reconfigures its components, according to the Grid state, in order to maintain a desired performance level. It tries to offer a robust and uniform access to data sources shared over the Grid. However, for providing data availability, Grid-DBMS relies on replicating whole databases using the underlying DBMSs' replication services, which is ineffective in highly dynamic environments. Furthermore, the scalability of Grid-DBMS has not been demonstrated.

To summarize, these solutions for data access and integration in Grids do not address highly dynamic environments. Thus, they cannot meet the requirements of OGSA-P2P, *e.g.* data availability, which APPA supports.

Specific P2P data management systems have been developed for managing shared data in P2P networks. P-Grid (the Grid of Peers) [1] is a peer-to-peer lookup system based on a virtual distributed search tree, structured like a distributed hash table. In P-Grid, each node holds part of the overall tree depending on its path, *i.e.* the binary bit string representing the subset of the tree's information that the node is responsible for. A decentralized and self-organizing process builds P-Grid's routing infrastructure which is adapted to a given distribution of data keys stored by nodes. This process also addresses uniform load distribution of data storage and uniform replication of data to support uniform availability. On top of P-Grid's lookup system, other self-organizing services may be implemented (*e.g.* identity, adaptive media dissemination, trust management). Unlike APPA, which is independent of the overlay network, P-Grid relies on a specific virtual distributed search tree.

The JXTA-GRID project [20] addresses the use of JXTA technology for Grid computing. JXTA-GRID will take advantage of existing services of both JXTA and OGSA, *e.g.* using JXTA for data discovery and message communication, and OGSA for job allocation and work-load management. However, to our knowledge, no version of JXTA-GRID has been yet released.

Edutella [28] is a P2P system for data management in super-peer networks. In Edutella, a small percentage of nodes, *i.e.* super-peers, are responsible for indexing the shared data and routing the queries. The super-peers are assumed to be highly available with very good computing capacity. Super-peers are arranged in a hypercube topology, according to the HyperCuP protocol [36]. When a node connects to Edutella, it should register at one of the super-peers. Upon registration, the node provides to the super-peer its RDF-based metadata. The initial Edutella services are as

follows: 1) query service for processing the queries based on RDF metadata; 2) replication service that provides data availability and workload balancing; 3) mapping service which is responsible for doing the mapping between the metadata of different nodes to enable interoperability between them; and 4) annotation service which annotates materials stored anywhere within the Edutella network. The main difference with APPA is that Edutella can only be implemented on top of a super-peer network, but APPA can be built on both super-peer and structured networks.

PeerDB [37] is a P2P system designed with the objective of high level data management in unstructured P2P networks. It exploits mobile agents for flooding the query to the nodes such that their hop-distance from the query originator is less than a specified value, *i.e.* TTL (Time-To-Live). Then, the query answers are gathered by the mobile agents and returned back to the query originator. The architecture of PeerDB consists of three layers, namely the P2P layer that provides P2P capabilities (*e.g.* facilitates exchange of data and resource discovery), the agent layer that exploits agents as the workhorse, and the object management layer (which is also the application layer) that provides the data storage and processing capabilities.

These P2P systems are typically dependent on the network (*i.e.* unstructured, structured or super-peer) for which they have been designed and cannot be easily used in other P2P networks. Thus, they cannot easily address the requirements of dynamic Grids.

One of the distinguishing features of APPA is its network-independent architecture, so it can be implemented over different overlay networks. Furthermore, APPA can support all the requirements specified by OGSA-P2P.

8. Conclusion

In this paper, we have presented the main services of APPA (Atlas Peer-to-Peer Architecture), a data management system for large-scale P2P and Grid applications. APPA has a network-independent architecture that can be implemented over various overlay networks. The main advantage of such architecture is to be able to exploit rapid and continuing progress in such networks. It can also be used as a basis for implementing OGSA-P2P. APPA can support the requirements of OGSA-P2P such as scalability, dynamic data discovery, data availability, group support, location awareness, security, and connectivity.

We focused on two main requirements of OGSA-P2P: data availability which is addressed by the persistent data management and replication services, and data discovery which is addressed by the query processing service. APPA provides data persistence with high availability through replication by using multiple hash functions. It addresses efficiently the problem of retrieving current replicas based on timestamping. APPA also provides a higher-level replication service with multi-master replication. This service enables asynchronous collaboration among users. In order to resolve conflicting updates, we use a distributed semantic-based reconciliation algorithm which exploits parallelism. Query processing in APPA deals with schema-based queries and considers data replication. The main phases of query processing are query reformulation on a common schema description, query matching

to find relevant nodes, query optimization to select best nodes, and query decomposition and execution.

APPA is portable and can be used over other platforms by replacing the services of the P2P network layer. We have implemented APPA on top of JXTA and other P2P networks such as CAN and Chord. We have validated APPA's services through a combination of experimentation over the Grid5000 experimental platform. Additionally, in order to study the scalability of these services with larger numbers of nodes, we implemented simulators using SimJava. Experimental and simulation results showed that APPA's services have good performance and scale up.

References

[1] Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., and Schmidt, R. P-Grid: A Self-organizing Structured P2P System. *ACM SIGMOD Record, 32(3)*, 2003.

[2] Akbarinia, R., Pacitti, E., and Valduriez, P. Reducing Network Traffic in Unstructured P2P Systems Using Top-k Queries. *J. Distributed and Parallel Databases, 19(2-3)*, 2006.

[3] Akbarinia, R., Martins, V., Pacitti, E., and Valduriez, P. Top-k Query Processing in the APPA P2P System. *Int. Conf. on High Performance Computing for Computational Science (VecPar)*, 2006.

[4] Aloisio, G., Cafaro, M., Fiore, S., and Mirto, M. The Grid-DBMS: Towards Dynamic Data Management in Grid Environments. *IEEE Int. Symposium on Information Technology: Coding and Computing (ITCC)*, 2005.

[5] Antonioletti, M. et al. The Design and Implementation of Grid Database Services in OGSA-DAI. *Concurrency and Computation: Practice and Experience 17 (2-4)*, 2005.

[6] Balfe, S., Lakhani, A.D., Paterson, K.G. Trusted Computing: Providing Security for Peer-to-Peer Networks. *IEEE Int. Conf. on Peer-to-Peer Computing*, 2005.

[7] Castro, M., Jones, M.B., Kermarrec, A., Rowstron, A., Theimer, M., Wang, H., Wolman, A. An Evaluation of Scalable Application-level Multicast Built Using P2P Overlays. *IEEE Infocom*, 2003.

[8] Chakravarti, A. J., Baumgartner, G., Lauria, M. The Organic Grid: Self-organizing Computation on a Peer-to-peer Network. *IEEE Transactions on Systems, Man, and Cybernetics*, Part A 35(3): 373-384 (2005).

[9] Chaudhuri, S., and Gravano, L. Evaluating Top-k Selection queries. *VLDB Conf.*, 1999.

[10] Chockler, G., Keidar, I., Vitenberg, R. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33(427-469), 2001.

[11] Clarke, I., Miller, S., Hong, T.W., Sandberg, O., Wiley, B. Protecting Free Expression Online with Freenet. *J. IEEE Internet Computing, 6(1)*, 2002.

[12] Foster, I.T., Kesselman, C., and Tuecke, S.. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *J. of Supercomputer Applications*, 15(3), 2001.

[13] Foster, I.T., and Iamnitchi, A. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. *Int. Workshop on P2P Systems (IPTPS)*, 2003.

[14] Globus Alliance. http://www.globus.org/

[15] Gnutella. http://www.gnutelliums.com/.

[16] Grid4All project. www.grid4all.eu/.

[17] Grid5000 Project. http://www.grid5000.fr.

[18] Howell, F., and McNab, R. SimJava: a Discrete Event Simulation Package for Java with Applications in Computer Systems Modeling. *Int. Conf. on Web-based Modeling and Simulation*, 1998.

[19] JXTA. http://www.jxta.org/.
[20] JXTA-GRID. http://jxta-grid.jxta.org/.
[21] Kermarrec, A., Rowstron, A., Shapiro, M., Druschel P. The IceCube approach to the reconciliation of diverging replicas. *ACM Symp. on Principles of Distributed Computing*, 2001.
[22] Knezevic, P., Wombacher, A., and Risse, T. Enabling High Data Availability in a DHT. *Int. Workshop on Grid and P2P Computing Impacts on Large Scale Heterogeneous Distributed Database Systems (GLOBE)*, 2005.
[23] Levy, A., Rajaraman, A., Ordille, J. Querying heterogeneous information sources using source descriptions. *VLDB Conf.*, 1996.
[24] Martins, V., Akbarinia, R., Pacitti, E., Valduriez, P. Reconciliation in the APPA P2P System. *IEEE ICPADS*, 2006.
[25] Martins, V., Pacitti, E., and Valduriez, P. Dynamic and Distributed Reconciliation in P2P-DHT Networks. *European Conf. on Parallel Computing (Euro-Par)*, 2006.
[26] Martins, V., Pacitti, E., Jimenez-Peris, R., and Valduriez, P. Scalable and Available Reconciliation in P2P networks. *Journées Bases de Données Avancées (BDA)*, 2006.
[27] Nejdl, W., Siberski, W., Sintek, M. Design issues and challenges for RDF- and schema-based peer-to-peer systems. *ACM SIGMOD Record, 32(3)*, 2003.
[28] Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., and Risch, T. EDUTELLA: a P2P networking infrastructure based on RDF. *Int. World Wide Web conf. (WWW)*, 2002.
[29] OGSAP2P Research Group. http://www.ggf.org/4_GP/ogsap2p.htm.
[30] Open Grid Services Architecture. http://www.globus.org/ogsa/.
[31] Open Grid Services Architecture Data Access and Integration. http://www.ogsadai.org.uk/.
[32] Özsu, T., Valduriez, P. *Principles of Distributed Database Systems*. Prentice Hall, 1999.
[33] Preguiça, N., Shapiro, M., Matheson, C. Semantics-based reconciliation for collaborative and mobile environments. *Int. Conf. on Cooperative Information Systems (CoopIS)*, 2003.
[34] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S. A scalable content-addressable network. *ACM SIGCOMM Conf.*, 2001.
[35] Saito, Y., Shapiro, M. Optimistic Replication. *ACM Computing Surveys, 37(1)*, 2005.
[36] Schlosser, M., Sintek, M., Decker, S., and Nejdl, W. HyperCuP—Hypercubes, Ontologies and Efficient Search on P2P Networks. *Int. Workshop on Agents and Peer-to-Peer Computing*, 2002.
[37] Siong Ng, W., Ooi, B., Tan, k.L., and Zhou, A. PeerDB: A P2P-based System for Distributed Data Sharing. *Int. Conf. on Data Engineering (ICDE)*, 2003.
[38] Sit, E., Morris, R. Security Considerations for Peer-to-Peer Distributed Hash Tables. *Int. Workshop on P2P Systems (IPTPS)*, 2002.
[39] Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F., Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Conf.*, 2001.
[40] Tanaka, A., Valduriez, P. The Ecobase environmental information system: applications, architecture and open issues. *ACM SIGMOD Record, 3(5-6)*, 2000.
[41] Tatarinov, I., Ives, Z.G., Madhavan, J., Halevy, A., Suciu, D., Dalvi, N., Dong, X., Kadiyska, Y., Miklau, G., Mork, P. The Piazza peer data management project. *ACM SIGMOD Record 32(3)*, 2003.
[42] The Spitfire Project. http://edg-wp2.web.cern.ch/edg-wp2/spitfire/index.html.
[43] Tomasic, A., Raschid, L., Valduriez, P. Scaling access to heterogeneous data sources with DISCO. *IEEE Trans. on Knowledge and Data Engineering, 10(5)*, 1998.
[44] Valduriez, P. Parallel Database Systems: open problems and new issues. *J. Distributed and Parallel Databases, 1(2)*, 1993.
[45] Whittaker, S., Issacs, E., O'Day, V. Widening the Net: Workshop report on the theory and practice of physical and network communities. *ACM SIGCHI Bulletin*, 29(3), 1997.