PeerSum: a Summary Service for P2P Applications

Rabab Hayek, Guillaume Raschia, Patrick Valduriez, Noureddine Mouaddib

PeerSum: a Summary Service for P2P Applications

Rabab Hayek, Guillaume Raschia, Patrick Valduriez and Noureddine Mouaddib
Atlas team, INRIA and LINA, University of Nantes, France

***Abstract*— Sharing huge databases in distributed systems is inherently difficult. As the amount of stored data increases, data localization techniques become no longer sufficient. A practical approach is to rely on compact database summaries rather than raw database records, whose access is costly in large distributed systems. In this paper, we propose PeerSum, a new service for managing summaries over shared data in large P2P and Grid applications. Our summaries are synthetic, multidimensional views with two main virtues. First, they can be directly queried and used to approximately answer a query without exploring the original data. Second, as semantic indexes, they support locating relevant nodes based on data content. Our main contribution is to define a summary model for P2P systems, and the algorithms for summary management. Our performance evaluation shows that the cost of query routing is minimized, while incurring a low cost of summary maintenance.**

***Index Terms*— Distributed systems, Database summarization**

I. Introduction

Research on distributed systems is focusing on supporting advanced applications which must deal with semantically rich data (e.g. XML documents, relational tables, etc.), using a high-level SQL-like query language. As a potential example of applications, consider the cooperation of scientists who are willing to share their private data for the duration of a given experiment. Such cooperation may be efficiently supported by improving the data localization and data description techniques.

Initially developed for moderate-sized scientific applications, Grid technology is now evolving to provide database sharing services, in large virtual organizations. In [13], a service-based architecture for database access (OGSA-DAI) has been defined over the Grid. OGSA-DAI extends the distributed database architecture [27] to provide distribution transparency using Web services. However, it relies on some centralized schema and directory management, which is not an adequate solution for supporting highly dynamic organizations, with a large number of autonomous members.

Peer-to-Peer (P2P) techniques that focus on scaling up, dynamicity, autonomy and decentralized control can be very useful to Grid data management. The complementary nature of the strengths and weaknesses of the two technologies suggests that the interests of the two communities are likely to grow closer over time [15]. For instance, P-Grid [26] and Organic Grid [2] develop self-organizing and scalable services on top of P2P systems.

A central issue in the operation of P2P systems as distributed systems is object locating. Initially, P2P search systems rely on flooding mechanism and its variations. Though simple and robust, this approach suffers from high query execution cost and poor query recall. Many works on P2P systems have addressed the problem of search efficiency, and proposed various techniques that can be classified into four main categories: data indexing (e.g. [1], [24]), data caching (e.g. [5]), mediation (e.g. [16], [22]) and network clustering (e.g. [4]). According to this classification, there are hybrid techniques such as caching data indexes [7], or clustering a P2P network based on index similarity [10].

So far, data localization has been the main issue addressed in P2P systems, since their scalability is constrained by the employment of efficient search techniques. However, nowadays we are asking the following question: with the ever increasing amount of information stored each day into data sources, are these techniques still sufficient to support advanced P2P applications? To illustrate, in a scientific collaborative application, a doctor may require information about patients diagnosed with some disease, without being interested in individual patients records. Besides, a user in today's decision-support applications may prefer an approximate but fast answer, instead of waiting a long time for an exact one. Therefore, reasoning on compact data descriptions that can return approximate answers like "dead Malaria patients are typically *children* and *old*" to queries like "*age* of dead Malaria patients", is much more efficient than retrieving raw records, which may be very costly to access in highly distributed, massive databases.

In this paper, we propose PeerSum, a new service for managing summaries over shared data in P2P systems. Our summaries are synthetic, multidimensional views with two main virtues. First, they support locating relevant nodes based on their data descriptions. Second, they provide an intelligible representation of the underlying data, and allow an *approximate query processing* since a query can be processed entirely in their domain, i.e. outputs are summaries.

This paper makes the following contributions. First, we define a summary model which deals with the distributed and autonomous nature of P2P systems. Second, we propose efficient algorithms for summary management. We validated our algorithmic solutions through simulation, using the BRITE topology generator and SimJava. The performance results show that the cost of query routing is minimized, while incurring a low cost of summary maintenance.

The rest of this paper is organized as follows. Section 2 gives an overview of the employed summarization technique. Section 3 states the addressed problem and describes the architecture of PeerSum summary model. Section 4 describes PeerSum's summary management with its algorithms. Section 5 discusses query processing with PeerSum. Section 6 gives a performance evaluation with a cost model and a simulation model. Section 7 compares our solution with related work. Section 8 concludes.

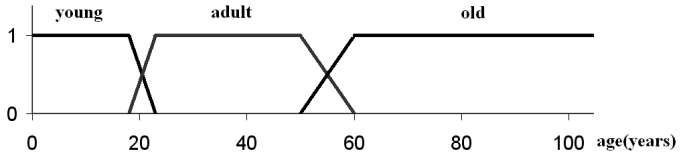

Fig. 1. Fuzzy Linguistic Partition on **age**

II. Summarization Process

In this work, the approach used for data summarization is based on SaintEtiQ [11]: an online linguistic approach for summarizing databases. The SAINTETIQ model aims at apprehending the information disseminated in large data sets in a synthetic manner. In the following, we define the input data and describe the summarization process. Then, we formally define the structure of the produced summaries.

A. Input Data

The summarization process takes as input the original data to be summarized, and the *"Background Knowledge"* BK which guides the process by providing information about the user perception of the domain. The data to be summarized come from relational databases. As such, the data are organized into records, with a schema $R(A_1, A_2, \ldots, A_n)$. Each attribute A_i is defined on an attribute domain D_i, which may be numeric or symbolic.

A unique feature of the summarization system is its use of a *Background Knowledge (BK)*, which relies on Zadeh's fuzzy set theory [18] and, more specifically on linguistic variables [19] and fuzzy partitions [20] to represent data in a concise form. Consider a medical database which is reduced to a single *Patient* relation (Table I)[1]. Figure 1 shows a linguistic variable defined on the attribute AGE where descriptor YOUNG is defined as being plainly satisfactory to describe values between 23 and 50 and less satisfactory as the age is out of this range. Thus, linguistic variables come with linguistic terms (i.e. descriptors) used to characterize domain values and, by extension, database tuples. For a continuous domain D_i, the linguistic variable is a fuzzy partition of the attribute domain. For a discrete domain D_i (e.g. DISEASE), the BK element is a fuzzy set of nominal values. In short, the BK supports the summarization process with means to match attribute domain values with the summary expression vocabulary.

Id	Age	Sex	BMI	Disease
t_1	15	*female*	17	*Anorexia*
t_2	20	*male*	20	*Malaria*
t_3	18	*female*	16.5	*Anorexia*

TABLE I
RAW DATA

B. Process Architecture

A service oriented architecture has been designed for the summarization process in order to incrementally build data summaries. By "incremental", we mean that the database tuples are processed one by one, and a final hierarchy of summaries is obtained by repeating this summarization process for each tuple in the table at hand. The architecture is organized into two separate services: online mapping and summarization.

1) Mapping Service: The mapping service takes as input the original relational records and performs an abstraction of the data using the BK. Basically, the mapping operation replaces the original attribute values of every record in the table by a set of linguistic descriptors defined in the BK. For instance, with the linguistic variable defined on the attribute AGE (Figure 1), a value $t.AGE = 20$ years is mapped to $\{$*0.7/young, 0.3/adult*$\}$ where 0.7 is a membership grade that tells how well the label *young* describes the value 20. Extending this mapping to all the attributes of a relation could be seen as locating the overlapping cells in a grid-based multidimensional space which maps records of the original table. The fuzzy grid is provided by the BK and corresponds to the user's perception of the domain.

In our example, tuples of Table I are mapped into three distinct grid-cells denoted by c_1, c_2, and c_3 in Table II. A priori, the fuzzy label *underweight* provided by the BK on attribute BMI, perfectly matches (with degree 1) range $[15, 17.5]$, while the fuzzy label *normal* perfectly matches range $[19.5, 24]$ of raw values. The tuple count column gives the proportion of records that belongs to the cell and 0.3*/adult* says that *adult* fits the data only with a small degree (0.3). It is computed as the maximum of membership grades of tuple values to *adult* in c_3.

Id	Age	BMI	tuple count
c_1	*young*	*underweight*	2
c_2	0.7/*young*	*normal*	0.7
c_3	0.3/*adult*	*normal*	0.3

TABLE II
GRID-CELLS MAPPING

The fuzziness in the vocabulary definition of BK permits to express any single value with more than one fuzzy descriptor and thus avoid threshold effect thanks to the smooth transition between different categories. For instance, the tuple t_2 of Table I is mapped to the two grid cells c_2 and c_3 since the value of the attribute *age* is mapped to the two linguistic terms: *young* and *adult*. Hence, the BK leads to the point where tuples become indistinguishable and then are grouped into grid-cells such that there are finally many more records than cells. Every new (coarser) tuple stores a record count and attribute-dependent measures (min, max, mean, standard deviation, etc.). It is then called a *summary*.

2) Summarization Service: The summarization service is the last and the most sophisticated step of the SAINTETIQ system. It takes *grid-cells* as input and outputs a collection of summaries hierarchically arranged from the most generalized one (the root) to the most specialized ones (the leaves) [11]. Summaries are clusters of grid-cells, defining hyperrectangles in the multidimensional space. In the basic process, leaves are grid-cells themselves and the clustering task is performed on K cells rather than N tuples ($K << N$).

[1]BMI: the patient's body weight divided by the square of its height.

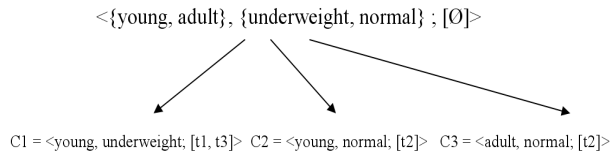

Fig. 2. Example of SaintEtiQ hierarchy

From the mapping step, cells are introduced continuously in the hierarchy with a top-down approach inspired of D.H. Fisher's Cobweb [17], a conceptual clustering algorithm. Then, they are incorporated into best fitting nodes descending the tree. Three more operators could be applied, depending on partition's score, that are *create*, *merge* and *split* nodes. They allow developing the tree and updating its current state. Figure 2 represents the summary hierarchy built from the cells c_1, c_2 and c_3.

3) Scalability Issues: Memory consumption and time complexity are the two main factors that need to be taken care of in order to guaranty the capacity of the summary system to handle massive datasets. First, the time complexity of the SAINTETIQ process is in $O(K)$, where *K* is the number of cells to incorporate into a hierarchy of summaries. Besides, an important feature is that in the summary algorithm, raw data have to be parsed only once, and this are processed with a low time cost. Second, the system requires low memory consumption for performing the summary construction algorithm as well as for storing the produced summaries.

On the other hand, the parallelization of the summary system is a key feature to ensure a smooth scalability. The implementation of the system is based on the Message-Oriented Programming paradigm. Each sub-system is autonomous and collaborates with the others through disconnected asynchronous method invocations. It is among the least demanding approaches in terms of availability and centralization. The autonomy of summary components allows for a distributed computing of the process.

C. Distributed Summary Representation

In this section, we introduce basic definitions related to the summarization process.

Definition 1 ***Summary*** *Let $E = \langle A_1, \ldots, A_n \rangle$ be a n-dimensional space equipped with a grid that defines basic n-dimensional areas called cells in E. Let R be a relation defined on the cartesian product of domains D_{A_i} of dimensions A_i in E. Summary z of relation R is the bounding box of the cluster of cells populated by records of R.*

The above definition is constructive since it proposes to build generalized summaries (hyper-rectangles) from cells that are specialized ones. In fact, it is equivalent to performing an *addition* on cells:

$$z = c_1 + c_2 + \ldots + c_p$$

where $c_i \in L_z$, the set of p cells (summaries) covered by z.

A summary z is then an *intentional description* associated with a set of tuples R_z as its *extent* and a set of cells L_z that are populated by records of R_z.

Thus, summaries are areas of E with hyper-rectangle shapes provided by BK. They are nodes of the summary tree built by the SAINTETIQ system.

Definition 2 ***Summary Tree*** *A summary tree is a collection S of summaries connected by $\preccurlyeq$, the following partial order:*

$$\forall z, z' \in \mathcal{Z}, \quad z \preccurlyeq z' \Longleftrightarrow R_z \subseteq R_{z'}$$

The above link between two summaries provides a generalization/specialization relationship. And assuming that summaries are hyper-rectangles in a multidimensional space, the partial ordering defines *nested summaries* from the larger one to the single cells. General trends in the data could be identified in the very first levels of the tree whereas precise information has to be looked at near the leaves.

For our purpose, we also consider a summary tree as an indexing structure over distributed data in a P2P system. Thus, we add a new dimension to the definition of a summary node z: a *peer-extent* P_z, which provides the set of peers having data described by z.

Definition 3 ***Peer-extent*** *Let z be a summary in a given hierarchy of summaries S, and P the set of all peers who participated to the construction of S. The* peer-extent *P_z of the summary z is the subset of peers owning, at least, one record of its extent R_z: $P_z = \{p \in P \mid R_z \cap R_p \neq \emptyset\}$, where R_p is the view over the database of node p, used to build summaries.*

Due to the above definition, we extend the notion of *data-oriented* summary in a given database, to a *source-oriented* summary in a given P2P network. In other words, our summary can be used as a database index (e.g. referring to relevant tuples), as well as a semantic index in a distributed system (e.g. referring to relevant nodes).

The summary hierarchy S will be characterized by its *Coverage* in the P2P system; that is, the fraction of data sources described by S. Relative to the hierarchy S, we call *Partner Peer* a peer whose data is described by at least a summary of S.

Definition 4 ***Partner peers*** *The set of* Partner peers *P_S of a summary hierarchy S is the union of peer-extents of all summaries in S: $P_S = \{\cup_{z \in S} P_z\}$.*

For simplicity, in the following we designate by "summary" a hierarchy of summaries maintained in a P2P system, unless otherwise specified.

III. PEERSUM SUMMARY MODEL

In this section, we study the integration of a new summary service, PEERSUM, into an existing P2P architecture. Here we work in the context of $APPA$ (Atlas Peer to Peer Architecture), a P2P data management system which provides high level services for advanced P2P applications [22]. We first state

the addressed problem, and then we present the architecture of the PeerSum summary model.

Given a P2P network, we consider the two following assumptions.

- Each peer p owns some tuples (R_p) in a global, horizontally partitioned relation R.
- Users that are willing to cooperate agree on a Background Knowledge BK, which represents their common perception of the domain.

Thus, here we do not address the problem of semantic heterogeneity among peers, since it is a separate P2P issue on its own. Besides, our work mainly targets collaborative database applications where the participants are supposed to work on "related" data. In such a context, the number of participants is also supposed to be limited, and thus the assumption of a common BK seems not to be a strength constraint. An example of such BK in a medical collaboration is the Systematized Nomenclature of Medicine Clinical Terms (SNOMED CT) [14], which provides a common language that enables a consistent way of capturing, sharing and aggregating health data across specialties and sites of care. On the other hand, our summaries are data structures that respect the original data schemas [23]. Hence, we can assume that the techniques that have been proposed to deal with information integration in P2P systems (e.g. [16], [22]) can be used here to overcome the heterogeneity of both data and summary representations, in the context of heterogeneous data.

Let $G = (V, E)$ be the graph corresponding to a P2P network of size N, where V is the set of nodes (i.e. $|V| = N$), and E is the set of links between nodes. Our ultimate goal is to maintain a global summary that completely describes the global relation R. However, as stated before, the relation R is horizontally partitioned and distributed among autonomous peers. Hence, the problem can be defined as follows. Given that each peer p_i locally maintains a Local Summary LS_i, we aim to construct the global summary GS_c such that:

$$GS_c = \cup_{i=1}^{N}(LS_i)$$

The local summaries are obtained by integrating the summarization process previously defined into each peer's DataBase Management System (DBMS). The operator $\cup$ designates the summary merging operation which will be discussed later. Note that GS_c is an approximation of the summary which might be obtained if the global relation R were totally available and summarized under a central coordination.

Once again, the autonomous and dynamic nature of P2P networks imposes additional constraints and makes the convergence to GS_c quite challenging. It is difficult to build and to keep this summary consistent relative to the current data instances it describes. So, the problem can be redefined as follows.

Given the set of materialized local summaries $\{LS_i, 1 \leq i \leq N\}$, we require to build/materialize the set of global summaries $\{GS_j, 1 \leq j \leq N_G\}$ such that:

- $GS_j = \cup_{i=1}^{l}(S_i)$, where S_i is a local or global summary. Here, the latter is defined as being the merging result of, at least, two summaries (i.e. $l \geq 2$).

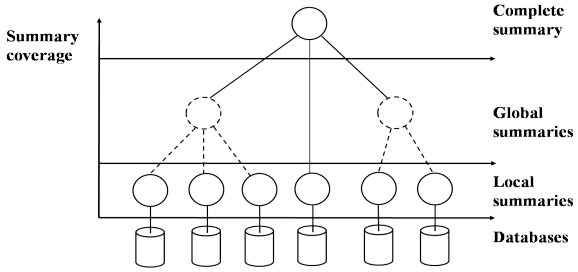

Fig. 3. Summary Model Architecture

- The set of materialized local/global summaries (each having its set of partner peers P_{GS_j}), and the set of links ($E_s \subset E$) between nodes belonging to different sets of partner peers (i.e. links connecting different summaries), approximate together the *virtual* summary GS_c.

$$GS_c \approx (\{LS_i, 1 \leq i \leq N_L\}, \{GS_j, 1 \leq j \leq N_G\}, E_s) \quad (1)$$

  N_L is the number of local summaries, i.e. the number of peers that have not participated to any global summary GS_j. While N_G is the number of global summaries built in the network (i.e. $|\cup_{j=1}^{N_G}(P_{GS_j})| + N_L = N$).
- A "good" trade-off should be achieved between the cost of updating the set of materialized summaries and the benefits obtained from exploiting these summaries in query processing.

In APPA, we adopt an incremental mechanism for summary construction. The "*coverage*" of a summary S in the network is defined as being the fraction of peers that own data described by S. This coverage quantifies the convergence of S to the complete summary GS_c, which is obviously characterized by a coverage = 1.

The architecture of our summary model is presented in Figure 3. The incremental aspect of the summary construction approach is described as follows. Peers that cooperate are exchanging and merging summaries, in order to build a Global Summary GS_j over their shared data. This summary is characterized by a continuous evolution in term of coverage, i.e. the cooperation between two sets of peers, each having constructed a global summary, results in a higher-coverage one.

IV. Summary management in PeerSum

APPA has a network-independent architecture so it can be implemented over different types of P2P networks. $APPA$ provides three layers of services: P2P network, basic services and advanced services. PeerSum is integrated at the advanced layer and defined based on the underlying services. Due to space limitations, we will only mention the services required for PeerSum definition.

According to our summary model, PeerSum must address the following requirements:

- Peers cooperate for exchanging and merging summaries into a global summary,
- Peers share a common storage in which the global summary is maintained.

The peer linking and peer communication services of the $APPA$'s P2P network layer allow peers to communicate and exchange messages (through service calls), while cooperating for a global summary construction. Besides, the update management service (UMS) [3] of the basic layer and the Key-based Storage and Retrieval (KSR) service of the P2P network layer, work together to provide a common storage in which a global summary is maintained. This common storage increases the probability that "*P2P data*" (e.g. metadata, indexes, summaries) produced and used by advanced services are available even if peers that have produced them are disconnected. The UMS and KSR services manage data based on keys. A key is a data identifier which determines which peer should store the data in the system, e.g. through hashing over all peers in DHT networks or using super-peers for storage and retrieval in super-peer networks. All data operations on the common storage are key-based, i.e. they require a key as parameter.

In the following, we will describe our algorithms for summary construction and maintenance. First, we work in a static context where all the participants remain connected. Then, we address the dynamicity of peers.

A. Summary construction

Starting up with a local summary level (see Figure 3), we present the algorithm for peer cooperation that allows constructing a global summary GS. We assume that each global summary is associated with a *Cooperation List* (CL) that provides information about its partner peers. An element of the cooperation list is composed of two fields. A partner peer identifier *PeerID*, and a *2*-bit freshness value v that provides information about the freshness of the descriptions as well as the availability of the corresponding database.

- value 0 (initial value): the descriptions are fresh relative to the original data,
- value 1: the descriptions need to be refreshed,
- value 2: the original data are not available. This value will be used while addressing peer volatility in Section IV-C.

Both the global summary and its cooperation list are considered as "summary data" and are maintained in the common storage, using the P2PDM and KSR services.

1) Cooperation request: The algorithm starts at an *initiator peer* P_{init} who sends a cooperation request message to its neighbors, to participate to a global summary construction. This message contains P_{init}'s identifier and a given value of TTL (Time-To-Live). One may think that a large value of TTL allows to obtain directly a high-coverage summary. However, due to the autonomous nature of P2P systems, P_{init} may keep waiting for a very long time without having constructed that global summary. Therefore, we choose to limit the value to TTL and adopt an incremental construction mechanism, as discussed in Section III.

2) Cooperation response: A peer *p* who receives the message, performs the following steps. First, if the request has already been received, it discards the message. Else, it saves the address of the sender as its parent. Then, its decrements TTL by one. If the value of TTL remains positive, it sends the message to its neighbors (except the *parent*).

After propagating the message, p must wait to receive the responses of its neighbors. However, since some of the neighbors may leave the system and never response, the waiting time must be limited. We compute p's waiting time using a cost function based on TTL, and network dependent parameters.

A cooperation response of a peer p has the following structure: $Coop_Resp = \langle$CS, PeerIDs, GSKeys$\rangle$. CS is the current summary obtained at p, $PeerIDs$ is the list of identifiers of peers that have responded to p, and $GSKeys$ is the list of keys of global summaries. If *p* is a partner peer, that is, *p* has already participated to an existing global summary, its $Coop\_Resp$ will include the key of the global summary it knows, as well as the peer identifiers contained in the corresponding *CL*, i.e. $Coop\_Resp = \langle\emptyset$, extractPeerIDs(CL), {GSKey}$\rangle$. In that case, *p* locates at the boundary of two knowledge scopes of two different summaries. Hence, it allows merging them into a higher-coverage one. Otherwise, its response will include its local summary and its identifier, i.e. $Coop_Resp = \langle$p.LS, {p.ID}, $\emptyset\rangle$.

3) Summary data storage: In the waiting phase, when a child's $Coop\_Resp$ arrives, a parent peer *p* merges it with its own response by making the union of $PeerIDs$ and $GSKeys$ lists, and merging the current summaries. Once the time expires, *p* sends the result to its parent. But, if *p* is the initiator peer P_{init}, it will store the new summary data, i.e. the new global summary GS and its cooperation list CL, using the KSR service: *GSKey := KSR_insert(CS, CL)*. CL contains each peer identifier obtained in the final $PeerIDs$ list, associated with a freshness value v equal to zero. At the end, P_{init} sends the new key (*GSKey*) to all participant peers, which become GS's *partner peers*.

B. Summary maintenance

An important issue for any indexing technique is to efficiently maintain the indexes against data changes. For a local summary, it has been demonstrated that the summarization process guarantees an incremental maintenance (using a *push* mode for exchanging data with the DBMS), while performing with a low complexity. In this section, we propose a strategy for maintaining the summary data: a global summary GS which has been obtained by merging the local summaries of the set of peers P_{GS}, and its associated cooperation list CL. The objective is to keep GS consistent with the current instances of the local summaries.

a) Push: Cooperation List Update: Each peer p in P_{GS} is responsible for refreshing its own element in the cooperation list CL. The partner p monitors the modification rate issued on its local summary LS. When LS is considered as enough modified, the peer p sets its freshness value v to 1, through a push message. The value 1 indicates that the local summary version being merged while constructing GS does not correspond any more to the current instance of the database.

An important feature is that the frequency of push messages depends on modifications issued on local summaries, rather than on the underlying databases. It has been demonstrated in [23] that, after a given process time, a summary becomes

very stable. As more tuples are processed, the need to adapt the hierarchy decreases and hopefully, once all existing attribute combinations have been processed, incorporating new tuple consists only in sorting it in a tree. A summary modification may be detected by observing the appearance/disappearance of descriptors in the summary intention.

b) Pull: Summary Update: The summary service monitors the fraction of old descriptions in GS, i.e. the number of ones in CL. We may consider that the peer that has been delegated to store CL (by the KSR and UMS services) is in charge of performing this task. Upon each push message sent to update a freshness value, the fraction of ones in CL is checked. If it exceeds a threshold value α, the summary update mechanism will be then triggered. Note that the threshold α is our parameter by which the freshness degree of GS will be controlled. In order to update GS, all the partner peers will be pulled to merge their current local summaries into a GS's version. The algorithm is described as follows.

A summary update message Sum_Update is propagated from a partner to another. This message contains a new summary $NewGS$ (initially empty), and a list of peer identifiers $PeerIDs$ which initially contains the identifiers of all GS's partners (provided by CL). When a partner p receives the Sum_Update message, it first merges $NewGS$ with its local summary and removes its identifier from $PeerIDs$. Then, it sends the message to another partner chosen from $PeerIDs$. If p is the last visited peer (i.e. $PeerIDs$ is empty), it updates the summary data: GS is replaced by the new version $NewGS$, and all the freshness values in CL are reset to zero. This strategy avoids conflicts and guarantees a high availability of the summary data, since only one update operation is performed by the last visited partner.

C. Peer dynamicity

In P2P systems, another crucial issue is to maintain the data indexes against network changes. Besides the freshness of summary descriptions, the availability of the original data sources should be also taken into account, given the dynamic behavior of peers.

1) Peer arrival: When a new peer *p* joins the system, it contacts some existing peers to determine the set of its neighbors. If one of those neighbors is a partner peer, *p* becomes a new partner: a new element is added to the cooperation list with a freshness value v equal to one. Recall that the value 1 indicates the need of pulling the peer to get new data descriptions. Furthermore, if *p* is a neighbor of two partners of two different summaries, it allows merging them in a higher-coverage one.

2) Peer departure: When a partner peer *p* decides to leave the system, it first sets its freshness value v to two in the cooperation list, through a push message. This value reminds the participation of the disconnected peer *p* to the corresponding global summary, but also indicates the unavailability of the original data. There are two alternatives to deal with such a freshness value. First, we can keep the data descriptions and use it, when a query is approximately answered using the global summary. A second alternative consists in considering the data descriptions as expired, since the original data are not accessible. Thus, a partner departure will accelerate the summary update initiating. In the rest of this paper, we adopt the second alternative and consider only a *1*-bit freshness value v: a value 0 to indicate the freshness of data descriptions, and a value 1 to indicate either their expiration or their unavailability.

However, if peer *p* failed, it could not notify its partners by its departure. In that case, its data descriptions will remain in the global summary until a new summary update is executed. The update algorithm does not require the participation of a disconnected peer. The global summary GS is reconstructed, and descriptions of unavailable data will be then omitted.

V. Query processing

Now, we discuss how a query Q, posed at a peer p, is processed. Our approach consists in querying at first the global summary GS available to peer p. As highlighted in the introduction and all along this paper, summary querying allows to achieve two distinct tasks depending on the user/application requirements: *peer localization* to return the original results, and *approximate answering* to return approximate answers. Summary querying is divided into two phases: 1) query reformulation and 2) query evaluation.

A. Query Reformulation

First, a selection query Q must be rewritten into a flexible query Q^* in order to be handled by the summary querying process. For instance, consider the following query Q on the Patient relation in Table I:

SELECT AGE FROM PATIENT WHERE SEX = "FEMALE" AND BMI $<$ 19 AND DISEASE = "ANOREXIA"

This phase replaces the original value of each selection predicate by the corresponding descriptors defined in the Background Knowledge (*BK*). Therefore, the above query is transformed to Q^*:

SELECT AGE FROM PATIENT WHERE SEX = "FEMALE" BMI IN $\{underweight, normal\}$ AND DISEASE = "ANOREXIA"

Let QS (resp.QS^*) be the *Query Scope* of query Q (resp.Q^*) in the P2P system, that is, the set of peers that should be visited to answer the query. Obviously, the query extension phase may induce false positives in query results. To illustrate, a patient having a BMI value of 20 could be returned as an answer to the query Q^*, while the selection predicate on the attribute `BMI` of the original query Q is not satisfied. However, false negatives cannot occur, which is expressed by the following inclusion: $QS \subseteq QS^*$.

In the rest of this paper, we suppose that a user query is directly formulated using descriptors defined in the BK (i.e. $Q = Q^*$). As we discussed in the introduction of this work, a doctor that participates to a given medical collaboration, may ask query Q like "the age of *female* patients diagnosed with *anorexia* and having an *underweight* or *normal* BMI". Thus, we eliminate potential false positives that may result from query extension.

B. Query Evaluation

This phase deals with matching a set of summaries organized in a hierarchy S, against the query Q. The query is transformed into a logical proposition P used to qualify the link between each summary and the query. Proposition P is under a conjunctive form in which all descriptors appears as literals. In consequence, each set of descriptors yields on corresponding clause. For instance, the above query Q is transformed to P = *(female) AND (underweight OR normal) AND (anorexia)*. A valuation function has been defined to valuate the proposition P in the context of a summary z. Then, a selection algorithm performs a fast exploration of the hierarchy and returns the set Z_Q of most abstract summaries that satisfy the query. For more details see [28]. Once Z_Q determined, the evaluation process can achieve two distinct tasks: 1) Peer localization, and 2) Approximate answering.

1) Peer Localization: Since the extended definition of a summary z provides a peer-extent, i.e. the set of peers P_z having data described by its intent (see Definition 4), we can define the set P_Q of relevant peers for the query Q as follows: $P_Q = \{\cup_{z \in Z_Q} P_z\}$. The query Q is directly propagated to these relevant peers. However, the efficiency of this query routing depends on the completeness and the freshness of summaries, since stale answers may occur in query results. We define a *False Positive* as the case in which a peer p belongs to P_Q and there is actually no data in the p source that satisfies Q (i.e. $p \notin QS$). A *False Negative* is the reverse case in which a p does not belong to P_Q, whereas there exists at least one tuple in the p data source that satisfies Q (i.e. $p \in QS$).

2) Approximate Answering: A distinctive feature of our approach is that a query can be processed entirely in the summary domain. An approximate answer can be provided from summary descriptions, without having to access original, distributed database records. The selected summaries Z_Q are aggregated according to their interpretation of proposition P: summaries that have the same required characteristics on all predicates (i.e. *sex*, *BMI* and *disease*) form a class. The aggregation in a given class is a union of descriptors: for each attribute of the selection list (i.e. *age*), the querying process supplies a set of descriptors which characterize summaries that respond to the query through the same interpretation [28]. For example, according to Table I, the output set obtained for the two classes $\{female, underweight, anorexia\}$, and $\{female, normal, anorexia\}$ is *age* = $\{young\}$. In other words, *all* female patients diagnosed with *anorexia* and having an *underweight* or *normal* BMI are *young* girls.

VI. Performance evaluation

In this section, we devise a simple model of the summary management cost. Then, we evaluate and analyze our model through simulation.

A. Cost Model

A critical issue in summary management is to trade off the summary updating cost against the benefits obtained for queries.

1) Summary Update Cost: Here, our first undertaking is to optimize the update cost while taking into account *query accuracy*. In the next section, we discuss query accuracy which is measured in terms of the percentage of false positives and false negatives in query results. The cost of updating summaries is divided into: usage of peer resources, i.e. time cost and storage cost, and the traffic overhead generated in the network.

Time Cost: A unique feature of SAINTETIQ is that the changes in the database are reflected through an incremental maintenance of the summary hierarchy. The time complexity of the summarization process is in $O(K)$ where K is the number of cells to be incorporated in that hierarchy [23]. For a global summary update, we are concerned with the complexity of merging summaries. The MERGING method that has been proposed is based on the SAINTETIQ engine. This method consists in incorporating the leaves L_z of a given summary hierarchy S_1 into an another S_2, using the same algorithm described by the SAINTETIQ summarization service (referenced in Section II-B.3). It has been proved that the complexity C_{M12} of the MERGING(S_1, S_2) process is constant w.r.t the number of tuples [21]. More precisely, C_{M12} depends on the maximum number of leaves of S_1 to incorporate into S_2. However, the number of leaves in a summary hierarchy is not an issue because it can be adjusted by the user according to the desired precision. A detailed Background Knowledge (BK) will lead to a greater precision in summary description, with the natural consequence of a larger summary. Moreover, the hierarchy is constructed in a top-down approach and it is possible to set the summarization process so that the leaves have any desired precision.

Storage Cost: We denote by k the average size of a summary z. In the average-case assumption, there are $\sum_{i=0}^{d} B^i = (B^{d+1} - 1)/(B - 1)$ nodes in a B-arity tree with d, the average depth of the hierarchy. Thus the average space requirement is given by: $C_m = k.(B^{d+1} - 1)/(B - 1)$. Based on real tests, $k = 512$ bytes gives a rough estimation of the space required for each summary. An important issue is that the size of the hierarchy is quite related to its stabilization (i.e. B and d). As more cells are processed, the need to adapt the hierarchy decreases and incorporating a new cell may consist only in sorting a tree. Hence, the structure of the hierarchy remains stable and no additional space is required. On the other hand, when we merge two hierarchies S_1 and S_2 having sizes of C_{m1} and C_{m2} respectively, the size of the resultant hierarchy is always in the order of the $max\,(C_{m1}, C_{m2})$. However, the size of a summary hierarchy is limited to a maximum value which corresponds to a maximum number of leaves that cover all the possible combinations of the BK descriptors. Thus, storing the global summary at the summary peer is not a strength constraint.

According to the above discussion, the usage of peer resources is optimized by the summarization process itself, and the distribution of summary merging while updating a global summary. Thus, we restrict now our focus to the traffic overhead generated in the P2P network.

Network Traffic: Recall that there are two types of exchanged messages: *push* and *update*. Let local summaries

have an average lifetime of L seconds in a given global summary. Once L expired, the node sends a (push) message to update its freshness value v in the cooperation list CL. The update algorithm is then initiated whenever the following condition is satisfied: $\sum_{v \in CL} v/|CL| \geq \alpha$, where α is a threshold that represents the ratio of old descriptions tolerated in the global summary. During update, only one message is propagated among all partner peers until the new global summary version is stored at the summary peer SP. Let F_{rec} be the update frequency. The update cost is:

$$C_{up} = 1/L + F_{rec} \quad messages \quad per \quad node \quad per \quad second \quad (2)$$

In this expression, $1/L$ represents the number of push messages which depends either on the modification rate issued on local summaries or the connection/disconnection rate of peers in the system. Higher is the rate, lower is the lifetime L, and thus a large number of push messages are entailed in the system. F_{rec} represents the number of update messages which depends on the value of α. This threshold is our system parameter that provides a trade-off between the cost of summary updating and query accuracy. If α is large, the update cost is low since a low frequency of update is required, but query results may be less accurate due both to false positives stemming from the descriptions of non existent data, and to false negatives due to the loss of relevant data descriptions whereas they are available in the system. If α is small, the update cost is high but there are few query results that refer to data no longer in the system, and nearly all available results are returned by the query.

2) Query cost: We have seen that the use of summaries as data indexes may improve query processing. When a query Q is posed at a peer p, first it is matched against the global summary to determine the set of peers P_Q whose descriptions are considered as answers. Then, Q is directly propagated to those peers. As a consequence, the number of messages exchanged in the system is intended to be significantly reduced. Furthermore, the cooperation list associated with a global summary provides information about the relevance of each database description. Thus, it gives more flexibility in tuning the trade-off *recall* ρ / *precision* π of the query answers. Let V be the set of peers visited while processing a query. Then $\rho = |QS \cap V|/|QS|$ and $\pi = |QS \cap V|/|V|$, where QS is the set of all peers that really match the query (i.e. *Query Scope*).

The trade-off can be tuned by confronting the set P_Q with the cooperation list CL. The set of all partner peers P_H in CL can be divided into two subsets: $P_{old} = \{p \in P_H \mid p.v = 1\}$, the set of peers whose descriptions are considered old, and $P_{fresh} = \{p \in P_H \mid p.v = 0\}$ the set of peers whose descriptions are considered fresh according to their current data instances. Thus, if a query Q is propagated only to the set $V = P_Q \cap P_{fresh}$, then precision is maximum since all visited peers are certainly matching peers (no false positives), but recall depends on the fraction of false negatives in query results that could be returned by the set of excluded peers $P_Q \backslash P_{fresh}$. On the contrary, if the query Q is propagated to the extended set $V = P_Q \cup P_{old}$, recall value is maximum since all matching peers are visited (no false negatives), but precision depends on the fraction of false positives in query results that are returned by the set of peers P_{old}.

The above two situations are bounds of a range of strategies available to propagate the query. In our experiments, we assume $V = P_Q$, the initial peer set. Thus, for a query Q, the cost is computed as $C_Q = 2 \cdot |P_Q|$ number of messages. Both query and result messages are considered here.

Parameter	value
local summary lifetime L	skewed distribution, Mean=3h, Median=1h
number of peers n	16–5000
matching nodes/query *hits*	10%
freshness threshold α	0.1–0.8

TABLE III
SIMULATION PARAMETERS

B. Simulation

We evaluated the performance of PeerSum through simulation, based on the above cost model. First, we describe the simulation setup. Then we present simulation results to evaluate various performance dimensions and parameters: scale up, query accuracy, effect of the freshness threshold α.

1) Simulation setup: We used the SimJava package [9] and the BRITE universal topology generator to simulate a power law P2P network, with an average degree of 4. The simulation parameters are shown in Table III. In large-scale P2P file-sharing systems, a user connects mainly to download some data and may then leave the system without any constraint. As reported in [25], these systems are highly dynamic and node lifetimes are measured in hours. As such, data indexes should be mainly maintained against network changes. In collaborative database applications, however, the P2P system is supposed to be more stable. Here, the data indexes should be mainly maintained against data changes, since the shared data may be submitted to a significant modification rate.

As stated before, our work targets collaborative applications sharing semantically rich data. In our tests, we have used synthetic data since it was difficult to obtain/use real, highly distributed P2P databases. Future works aim to employ our proposal in the context of astronomical applications, which seem to be attractive because of the huge amount of information stored into the databases. Thus, to provide meaningful results, we have evaluated the performance of our solutions in worst contexts where the data are highly updated. We have considered that local summary lifetimes follow a skewed distribution with a mean lifetime of 3 hours, and a median lifetime of 60 minutes. Note that summary lifetimes of hours means that the underlying data is submitted to a very high modification rate, since the summaries are supposed to be more stable than original data (as discussed in Section IV-B). Besides, such lifetime values allow to predict the performance of PeerSum in large-scale data sharing P2P systems where the rate of node departure/arrival dominates the global summary update initiating. In our tests, we work with moderated-size P2P networks, i.e. the number of peers varies between 16 and

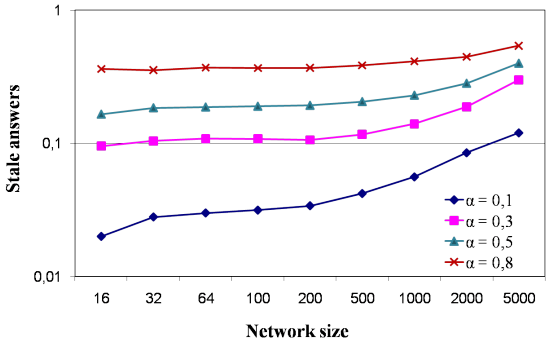

Fig. 4. Stale answers vs. number of peers

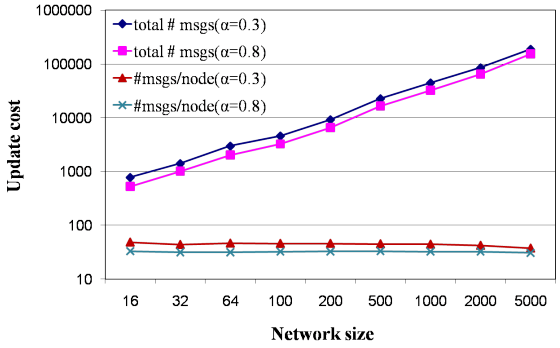

Fig. 6. number of messages vs. number of peers

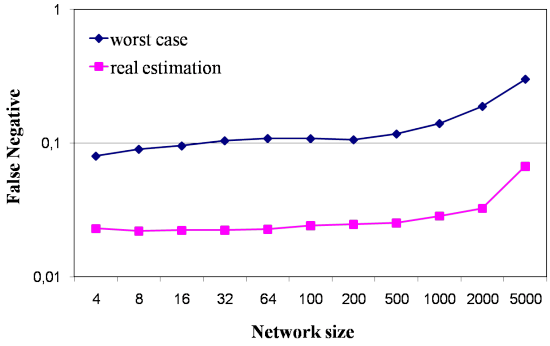

Fig. 5. False negative vs. number of peers

5000. In our query workload, the query rate is 0.00083 queries per node per second (one query per node per 20 minutes) as suggested in [6]. Each query is matched by 10% of the total number of peers. Finally, our system parameter α varies between 0.1 and 0.8.

2) Update cost: In this set of experiments we quantify the trade-off between query accuracy and the cost of updating a global summary. Figure 4 depicts the fraction of stale answers in query results for different values of the threshold α. Here we illustrate the worst case. For each partner peer p having a freshness value equal to 1, if it is selected in the set P_Q then it is considered as false positive. Otherwise, it is considered as false negative. However, this is not the real case. Though it has a value equal to 1, the peer p do not incur stale answers unless its database is changed relative to the posed query Q. Thus, Figure 4 shows the worst, but very reasonable values. For instance, the fraction of stale answers is limited to 11% for a network of 500 peers when the threshold α is set to 0.3 (30% of the peers are tolerated to have old/non existent descriptions).

As mentioned in Section VI-A.2, if we choose to propagate the query only to the set $V = P_Q \cap P_{fresh}$ we eliminate the possible false positives in query results. However, this may lead to additional false negatives. Figure 5 shows the fraction of false negatives in function of the total number of peers. Here we take into account the probability of the database modification relative to the query, for a peer having a freshness value equal to 1. We see that the fraction of false negatives is limited to 3% for a network size less than 2000. Moreover, the real estimation of stale answers shows a reduction by a factor of 4.5 with respect to the preceded values.

Figure 6 depicts the update cost in function of the total number of peers, and this for two threshold values. The total number of messages increases with the number of peers, but not surprisingly, the number of messages per node remains almost the same. In the expression of the update cost C_{up}, the number of push messages for a given peer is independent of network size. More interestingly, when the threshold value decreases (from 0.8 to 0.3) we notice a little cost increasing of 1.2 on average. For a network of 1000 peers, the update cost increases from 0.01056 to 0.01296 messages per node per minute (not shown in figure). However, a small value of the threshold α allows to reduce significantly the fraction of stale answers in query results, as seen in Figure 4. Therefore, tuning our system parameter α do not incur high traffic overhead in the system, while improving query accuracy.

3) Query cost: In this set of experiments, we compare our algorithm for query processing against non-index/flooding algorithms which are very used in real life, due to their simplicity and the lack of complex state information at each peer. Our algorithm is based on summary interrogation (SI) to determine the set of relevant peers P_Q. Ideally, all matching peers are targeted and directly visited. A flooding algorithm consists in routing the query till a stop condition is satisfied. Here, we limit the flooding by a value 3 of TTL (Time-To-Live). The Figure 7 depicts the number of exchanged messages to process a query Q, in function of the total number of peers. Our algorithm SI shows the best results that can be expected from any query processing algorithm, when no stale answers occur in query results (the ideal case). However, to give a real performance evaluation, we decide to study our algorithm in the worst case where the stale answers of Figure 4 occur in query results (for $\alpha = 0.3$). Even in that, SI shows a reduction of the number of messages that becomes more important with a large size of network. For instance, the query cost is reduced by a factor of 3 for a network of 2000 peers.

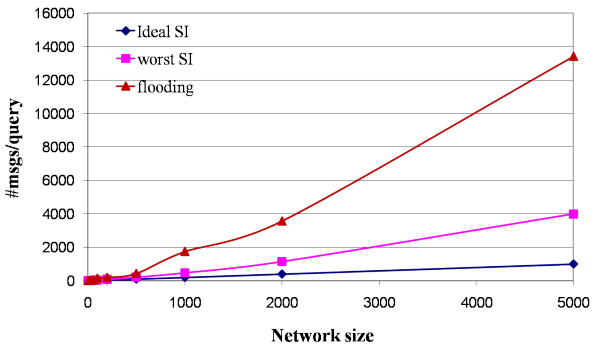

Fig. 7. Query cost vs. number of peers

VII. Comparison with Related Work

Current works on P2P systems aim to employ *content-based* routing strategies, since the content of data can be exploited to more precisely guide query propagation. These strategies require gathering information about the content of peer's data. However, the limits on network bandwidth and peer storage, as well as the increasing amount of shared data, call for effective summarization techniques. These techniques allow exchanging compact information on peer's content, rather than exchanging original data in the network. Existing P2P systems have used keyword-based approaches to summarize text documents. For instance, in [1] documents are summarized by keyword vectors, and each node knows an approximate number of documents matching a given keyword that can be retrieved through each outgoing link (i.e. Routing Indices RIs). Although the search is very bandwidth-efficient, RIs require flooding in order to be created and updated, so the method is not suitable for highly dynamic networks. Other works (e.g. [8]) investigate Vector Space Model (VSM) and build *Inverted Indexes* for every keyword to cluster content. In this model, documents and queries are both represented by a vector space corresponding to a list of orthogonal term vectors called *Term Vector*. The drawback of VSM is its high cost of vector representations in case of P2P churns. In [12], a *semantic-based* content search consists in combining VSM to Latent Semantic Index (LSI) model to find semantically relevant documents in a P2P network. This work is based on hierarchical summary structure over hybrid P2P architecture, which is closely related to what we are presenting in this paper. However, instead of representing documents by vector models, we describe structured data (i.e. relational database) by synthetic summaries that respect the original data schema. To the best of our knowledge, none of the summarization techniques used in P2P systems allows for an *approximate query answering*. All works have focused on facilitating content-based query routing, in order to improve search efficiency. We believe that the novelty of our approach relies on the fact that our data summaries allow for a semantic-based query routing, but also for approximately answering the query using their intentional descriptions.

VIII. Conclusion

In this paper, we proposed PeerSum, a new service for managing data summaries in P2P and Grid systems. PeerSum supports scaling up in terms of two dimensions: number of participants and amount of data. This paper made two main contributions. First, we defined a summary model for P2P systems, based on the SAINTETIQ process [11]. Second, we proposed efficient algorithms for summary management in PeerSum. Our analysis and simulation results showed that the use of summaries as data indexes reduces the cost of query routing by an important factor compared to flooding approaches, without incurring high costs in terms of update messages exchanged in the network.

References

[1] A.Crespo and H.G.Molina. Routing indices for peer-to-peer systems. In *Proc of the 28 tn Conference on Distributed Computing Systems*, 2002.

[2] A.J.Chakravarti, G.Baumgartner, and M.Lauria. 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.

[3] R. Akbarinia, E. Pacitti, and P. Valduriez. Data currency in replicated dhts. In *SIGMOD Conference*, pages 211–222, 2007.

[4] A.Oser, F.Naumann, W.Siberski, W.Nejdl, and U.Thaden. Semantic overlay clusters within super-peer networks. In *Proc of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing in Conjunction with the VLDB*, 2003.

[5] A.Rowstron and P.Druschel. Storage management and caching in PAST, a large–scale, persistent peer-to-peer storage utility. In *Proc.SOSP*, 2001.

[6] B.Yang and H.G.Molina. Comparing hybrid peer-to-peer systems. In *Proc VLDB*, 2001.

[7] C.Wang, L.Xiao, Y.Liu, and P.Zheng. Dicas: An efficient distributed caching mechanism for p2p systems. *IEEE Transactions on Parallel and Distributed Systems*, 2006.

[8] F.Cuenca-Acuna, C.Peery, R.Martin, and T.Nguyen. Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. In *HPDC-12*, 2003.

[9] F.Howell and R.McNab. Simjava: a discrete event simulation package for java with the applications in computer systems modeling. In *Int. Conf on Web-based Modelling and Simulation, San Diego CA, Society for Computer Simulation*, 1998.

[10] G.Koloniari, Y.Petrakis, and E.Pitoura. Content–based overlay networks of xml peers based on multi-level bloom filters. In *Proc VLDB*, 2003.

[11] G.Raschia and N.Mouaddib. A fuzzy set-based approach to database summarization. *Fuzzy sets and systems 129(2)*, pages 137–162, 2002.

[12] H.Shen, Y.Shu, and B.Yu. Efficient semantic-based content search in p2p network. *IEEE Transactions on Knowledge and Data Engineering*, 16(7), 2004.

[13] http://www.ogsadai.org.uk.

[14] http://www.snomed.org/snomedct.

[15] I.Foster and A.Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *IPTPS*, pages 118–128, 2003.

[16] I.Tartinov and *et al*. The Piazza peer data management project. In *SIGMOD*, 2003.

[17] K.Thompson and P.Langley. Concept formation in structured domains. In *Concept formation: Knowledge and experience in unsupervised learning*, pages 127–161. Morgan Kaufmann.

[18] L.A.Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[19] L.A.Zadeh. Concept of a linguistic variable and its application to approximate reasoning-I. *Information Systems*, 8:199–249, 1975.

[20] L.A.Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 100:9–34, 1999.

[21] M.Bechchi, G.Raschia, and N.Mouaddib. Merging distributed database summaries. In *ACM Sixteenth Conference on Information and Knowledge Management (CIKM)*, 2007.

[22] R.Akbarinia, V.Martins, E.Pacitti, and P.Valduriez. Design and implementation of appa. In *Global Data Management (Eds. R. Baldoni, G. Cortese and F. Davide)*. IOS press, 2006.

[23] R.Saint-Paul, G.Raschia, and N.Mouaddib. General purpose database summarization. In *Proc VLDB*, pages 733–744, 2005.

[24] S.Ratnasamy, P.Francis, M.Handley, R.M.Karp, and S.Shenker. A scalable content–addressable network. In *SIGCOMM*, 2001.

[25] S.Saroiu, P.Gummadi, and S.Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc of Multimedia Computing and Networking (MMCN)*, 2002.

[26] K. *et* al. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, 2003.

[27] T.Ozsu and P.Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1999.

[28] W.A.Voglozin, G.Raschia, L.Ughetto, and N.Mouaddib. Querying the SAINTETIQ summaries–a first attempt. In *Int.Conf.On Flexible Query Answering Systems (FQAS)*, 2004.