Unstructured P2P Networks-Multidimensional Historical Collection Data

[1]V. Suresh, [2]Amzed Ali Shaik
[1,2]Dept. of CSE, PYDAH College of Engg. & Tech., Gambheeram, Visakhapatnam, AP, India

Abstract
Efficient handling of multidimensional data is a challenging issue in P2P systems. P2P is a distributed application architecture that partitions tasks or workloads among peers. Peers are equally privileged, equipotent participants in the application. Each computer in the network is referred to a node. The owner of each computer on a P2P network would set aside a portion of its resources, such as processing power, disk storage or network bandwidth, directly available to other network participants, without the need for central coordination by servers or stable hosts. A P2P-based framework supporting the extraction of aggregates from historical multidimensional data is proposed, which provides efficient and robust query evaluation. When a data population is published, data are summarized in a synopsis, consisting of an index built on top of a set of subsynopses (storing compressed representations of distinct data portions). The index and the subsynopses are distributed across the network, and suitable replication mechanisms taking into account the query workload and network conditions are employed that provide the appropriate coverage for both the index and the subsynopses.

Keywords
Data Aggregation, Data Publication, P2P Networks, Multidimensional Data Management, Data Compression

I. Introduction

Peer-to-peer (P2P) computing has emerged as a powerful key paradigm for structuring large scale distributed systems in an ad-hoc manner, offering a large variety of features such as efficient storage and location of data items, wide area routing architecture, massive scalability, and fault-tolerance. P2P systems are fundamentally different from traditional client-server systems, in the sense that they do not employ any central authority nor assume any global knowledge. Participating nodes act simultaneously as clients and servers and exchange information and services directly with each other. By nature, P2P systems are dynamic where nodes can join and leave the network freely. Each node keeps contacts with other nodes in the system (called neighbours).

The success of P2P-based solutions is strictly related to the use of lossy data compression techniques (such as MPEG formats), which yield reasonable detail levels in representing large amounts of information and make data exchange feasible in practice by significantly reducing data transmission costs. However, the problem of suitably extending-data-compression-based solutions to application contexts other than file sharing has not been deeply investigated yet. In this scenario, information is represented as points in a multidimensional space whose dimensions correspond to different perspectives over data: users explore data and retrieve aggregates by issuing range queries, i.e., queries specifying an aggregate operator and the range of the data domain from which the aggregate information should be retrieved.

Although the multidimensional data model is substantially more complex than the representation paradigm adopted in the file sharing context (where data are organized according to < name, file > pairs), analytical applications dealing with historical multidimensional data and file-sharing applications share a fundamental aspect: they can rely on lossy data compression. In fact, analogously to tools for reproducing audio and/or video files, a lot of applications dealing with multidimensional data can effectively accomplish their tasks even in the case that only an approximate representation of data is available.

II. Definations

The management of compressed data on unstructured P2P networks is an intriguing issue, but poses several definitions, which we discuss in the following.

A. Compression

In computer science and information theory, data compression, source coding, or bit-rate reduction involves encoding information using fewer bits than the original representation. Compression can be either lossy or lossless. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by identifying marginally important information and removing it.

Compression is useful because it helps reduce the consumption of resources such as data space or transmission capacity. Because compressed data must be decompressed to be used, this extra processing imposes computational or other costs through decompression. For instance, a compression scheme for video may require expensive hardware for the video to be decompressed fast enough to be viewed as it is being decompressed, and the option to decompress the video in full before watching it may be inconvenient or require additional storage. The design of data compression schemes involve trade-offs among various factors, including the degree of compression, the amount of distortion introduced (e.g., when using lossy data compression), and the computational resources required to compress and uncompress the data.

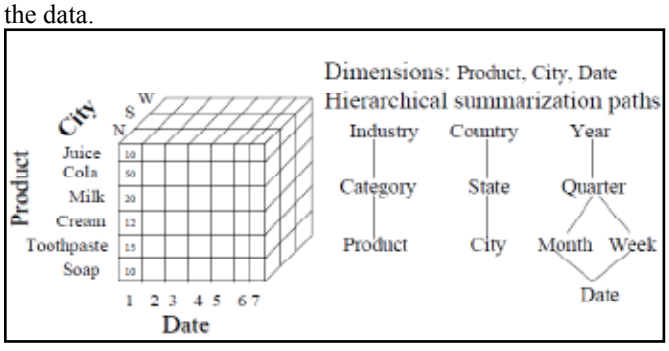

Fig. 1: Multidimensional Data

These drawbacks would be overcome if the compressed synopsis were subdivided into tiny subsynopses which are independently replicated and disseminated on the network when needed. Peers would, therefore, be asked to host replicas of small chunks of data. This way, the autonomy requirement would not result in a limit on the overall size of the synopsis (since the whole storage capacity of the network could be employed to store the whole synopsis), thus enabling the construction of synopses that provide

high-quality estimates. Moreover, the fragmentation could be exploited to make replication more fine-grained, thus even less resource-consuming.

B. Indexing

Once a compression technique yielding subsynopses with the desired properties has been defined, the problem of making the compressed data efficient to be located over the network must be tackled. Appropriate techniques are thus needed to distribute the compressed data and index them for efficient access. A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of slower writes and increased storage space. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records. The disk space required to store the index is typically less than that required by the table (since indices usually contain only the key-fields according to which the table is to be arranged, and exclude all the other details in the table), yielding the possibility to store indices in memory for a table whose data is too large to store in memory.

A better way to address this issue is to design an indexing mechanism that supports the efficient location of the subsynopses involved in the query evaluation. Hence, the challenge is to define an indexing technique supporting the location of our subsynopses in an unstructured network, such that the portions of the index and the responsibility of hosting it can be dynamically distributed among the peers, while preserving their autonomy.

Table 1:

ID	Name	Other Fields
12	Plug	...
13	Lamp	...
I4	Fuse	...

C. Indexing

Replication is the process of sharing information so as to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility. It could be data replication if the same data is stored on multiple storage devices, or computation replication if the same computing task is executed many times. A computational task is typically replicated in space, i.e. executed on separate devices, or it could be replicated in time, if it is executed repeatedly on a single device. The access to a replicated entity is typically uniform with access to a single, non-replicated entity. The replication itself should be transparent to an external user. Also, in a failure scenario, a failover of replicas is hidden as much as possible.

A replication scheme capable of maintaining appropriate levels of coverage w.r.t. the evolution of user interests and network conditions must be designed, to ensure accessibility and robustness. Existing replication strategies for unstructured P2P networks treat data sets as atomic objects, as they perform a number of replicas of a data set each time a query is posed on it. As explained before, in our scenario, this would limit both the size of the synopsis (thus affecting the accuracy of the compressed data) and the frequency of replica creations (thus limiting the responsiveness w.r.t. volatility); moreover, the index itself must be properly replicated too.

D. Data Publication

Let p be a peer which is willing to share a historical multidimensional data set D so that the other peers can pose aggregate range queries against it. In order to make its data suitable for being distributed across the network, p builds a synopsis of D by first appropriately partitioning D, and then, compressing each portion of data in the partition. Peer p also builds an index over these subsynopses, which, again, is properly fragmented in order to make it prone to be distributed. Finally, the subsynopses and the index portions are disseminated across the network, along with metadata about D. The assignment of data and index portions to peers takes into account the willingness of peers to share their resources.

E. Data Querying

Exploration queries can be issued by peers to discover the shared data sets in which they may be interested. These queries specify criteria that are matched against the metadata associated with each available data set. The result of the exploration process is a set of matching data sets and for each of them, a set of peers that should be contacted to start the evaluation of range queries, i.e., peers hosting portions of the distributed index that are thus capable to appropriately route range queries.

After issuing an explorative query, a peer p can decide to pose range queries against a matching data set. To do so, p contacts one of the peers that can route the query toward the peers hosting the subsynopses needed for evaluating the query. Finally, p gathers the partial results obtained by these peers and combines them to compute the final answer. The completeness of the answer can be checked through an appropriate mechanism which, if some partial result has not been received yet, allows p to complete the answer without issuing the range query from scratch.

```
Aggregation Query
SELECT Agg-Op(Col) FROM T
WHERE selection-condition
```

III. Compressing and Indexing Data

In this section, we describe the strategy adopted for compressing the data to be distributed across the network. Our approach aims at satisfying the following requirements:

1. The compressed data must be prone to be fairly distributed across the network, in order to allow parallel evaluation of queries and suitable robustness;
2. The compressed data must support the efficient extraction of (approximate) aggregate information.
3. The representation of the compressed data must enable efficient location of the data portions involved in the queries.

A. Partitioning the Data Domain

The aim of the partitioning step is to divide the data domain into nonoverlapping blocks. These blocks will be com-pressed separately, yielding distinct subsynopses. For each of them, a portion of the amount of storage space B chosen to represent the whole synopsis will be invested. The distribution of B among blocks will take into account the following requirements:

1. B must be fairly distributed among blocks: The assignment of different amounts of storage space to the blocks for representing their subsynopses should depend on the differences in homogeneity among the blocks. Intuitively enough, the more homogeneous the data inside a block, the smaller the amount of information needed to effectively accomplish its summarization.

2. Each block must be assigned a "small" portion of B: The subsynopses over the blocks are the data that will be hosted by peers and exchanged across the P2P network.

We denote the maximum amount of storage space which can be invested for summarizing a single block as B_{max}. In our prototype, we set B_{max} =256 KB: this is the threshold value which proved effectiveness in several file sharing applications (such as Gnutella itself) for limiting the download segment size, i.e., the size of the atomic file portions.

The partitioning ends when every block is assigned an amount of storage space which does not exceed B_{max}. In fig. 2, a partitioning of a 2D data population is shown (dashed line boxes represent the MBRs of the blocks).

We now explain the criteria adopted to select and split blocks, and the strategy for distributing B among the blocks of a partition.

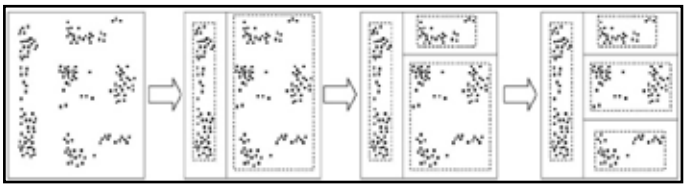

Fig. 2. Partitioning a 2D Data Population

B. Distributing B Among the Blocks of the Current Partition

First, a fixed portion B_{min} of B is assigned to every block (the meaning of Bmin will be clearer in the following), and then, the remainder of B is distributed on the basis of the homogeneity of the blocks. That is, if the current partition consists of k blocks b_1, . . . , b_k, then each b_i is assigned the following amount of storage space:

$$B(bi) = B(b_{min}) + \frac{SSE(b_i)}{\sum_{j=1}^{k} SSE(b_j)} \cdot (B - k \cdot b_{min}).$$

```
Function domainPartitioning
Input: Data population D, overall storage space bound B, upper bound B_max
       and lower bound B_min for the storage space assigned to a single block
Output: List of pairs ⟨ range, storage space ⟩
begin
  SSE_tot ← SSE(D)
  q ← new PriorityQueue() // at each step, q will contain the blocks of the
                          //current partition; the priority of blocks is their SSE
  b ← new Block(MBR(D),SSE_tot)
  q.enqueue(b) // the block corresponding to the whole D is inserted into q
  nBlocks ← 1
  space_max ← B
  while space_max > B_max
      b* ← q.dequeue() // the least homogeneous block is extracted
      {b', b''} ← binarySplit(b*) // b* is split into two blocks
      nBlocks ← nBlocks + 1
      q.enqueue(b'); q.enqueue(b'')
      SSE_tot ← SSE_tot − SSE(b*) + SSE(b') + SSE(b'')
      SSE_max ← SSE(q.first())
      space_max ← B_min + (B − nBlocks · B_min) · SSE_max/SSE_tot
  l ← new List()
  for each b ∈ q
      space_b ← B_min + (B − nBlocks · B_min) · SSE(b)/SSE_tot
      l.add(⟨ MBR(b), space_b ⟩)
  return l
end
```

Fig. 3: Data Domain Partitioning Algorithm

The value of B_{min} is the amount of space needed to store the most compact representation of a block according to the compression technique adopted. For instance, if a histogram is employed, B_{min} is the space consumption of representing both the range of the block and a set of aggregate values (such as the sum) summarizing the data inside a block. Basically, assigning at least B_{min} to each block means that every block of the partition is guaranteed to be represented in the overall data synopsis.

It uses a priority queue which, at each step, contains the blocks of the current partition, ordered by their SSE. Variable SSE_{tot} stores the sum of the SSEs of the blocks in the current partition, whereas variables SSE_{max} and spacemax represent the SSE of the least homogeneous block of the partition (i.e., the block at the head of the queue) and the storage space assigned to it, respectively. Iteratively, the least homogeneous block is extracted from the queue and split into two new blocks, which are, in turn, inserted into the queue. After splitting a block, SSE_{tot} is updated to take into account the overall SSE reduction due to the split, SSE_{max} is assigned the SSE of the new head of the queue, and spacemax is recomputed on the basis of the new values of SSE_{tot} and SSE_{max}.

Pseudo code for the data domain partitioning algorithm
Algorithm : Data Domain Partitioning

```
Input : Overall Data Population (D)
        Upper Bound (Bmax)
        Lower Bound (Bmin) for the storage space
        assigned to single block.
Output : List of pairs.
begin
Vector vector=new Vector()
spacemax = population size of the
data while(spacemax>0)
      String str= space max .split(Bmin , Bmax)
      B min = spacemax .indexOf(str)
      Spacemax = spacemax. split(Bmin) vector.add(str)
Return
 vector end
```

C. Compressing Data Blocks

At this step, a suitable compression algorithm is run on each of the k pairs <b_1....., b_k> resulting from the partitioning step, and sub synopses h_1.... . , h_k are obtained, where each h_i is a compressed representation of bi consuming storage space s_i. Several compression techniques can be employed to accomplish data summarization, as both the partitioning strategy and the techniques used for distributing and querying the sub-synopses (which will be described in the following sections) are orthogonal to this choice. Our prototype embeds, which has been shown to be very effective in constructing multidimensional histograms providing accurate estimates of range queries. Briefly, Clustering-based Histogram (CHIST) exploits a density-based clustering algorithm to construct a set of (possibly overlapping) blocks covering the nonempty portions of the data domain.

Density Based Clustering Algorithm

```
DBCA (SetOfObjects, NPred, MinCard, wCard)
// SetOfObjects is UNCLASSIFIED
ClusterId := nextId(NOISE);
for i FROM 1 TO SetOfObjects.size do
Object := SetOfObjects.get(i);
if Object.ClId = UNCLASSIFIED then
IF ExpandCluster(SetOfObjects,Object,ClusterId,
NPred,MinCard,wCard) then
ClusterId:=nextId(ClusterId)
end if
end if
end for
end;
```

To find a density-connected set, density based clustering algorithm starts with an arbitrary object p and retrieves all objects density-reachable from p with respect to NPred and MinWeight. If p is a core object, this procedure yields a density-connected set with

respect to NPred and MinWeight (. If p is not a core object, no objects are density-reachable from p and p is assigned to NOISE. This procedure is iteratively applied to each object p which has not yet been classified.

Algorithm for cluster expansion

```
ClusterExpansion (SetOfObjects, Object, ClId, NPred,
MinCard, wCard): Boolean;
if wCard({Object}) ≤ 0 then // point not in selection
SetOfPoints.changeClId(Object,UNCLASSIFIED);
return False;
end if
seeds:=Set of Objects.neighborhood(Object, NPred);
if wCard(seeds) < MinCard THEN // no core point
Set of Objects.changeClId(Object,NOISE);
return False;
end if
// still here? Object is a core object
Set of Objects.changeClIds(seeds, ClId);
seeds.delete(Object);
while seeds ≠ Empty do
current Object := seeds. first();
result := Set of Objects.neighborhood(currentObject, NPred);
if wCard(result) ≥ MinCard then
for i FROM 1 TO result.size do
P := result.get(i);
if wCard({P}) > 0 AND P.ClId IN {UNCLASSIFIED, NOISE}
then
IF P.ClId = UNCLASSIFIED then
seeds.append(P);
end if;
Set of Objects.changeClId(P, ClId);
end if; // wCard > 0 and UNCLASSIFIED or NOISE
end for;
end if; // wCard ≥ MinCard
seeds.delete(currentObject);
end while; // seeds □ □ Empty
return True;
end; //
```

D. Creating the Index

In order to limit the traffic needed for the maintenance of the index after its distribution, a compact index is desirable. Since data are historical, the storage space consumption of the R-Tree can be reduced by adopting packing strategies for its construction, which aim at obtaining 100 percent space utilization in each node. This was shown to perform better than other packing strategies in supporting region queries on rectangular data, which is the feature of interest in our case. Briefly, this technique works in three steps.

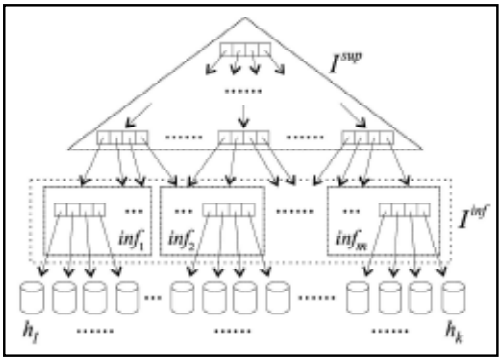

Fig. 3: Partitioning the R-Tree

First, the data domain is linearized according to a Hilbert space filling curve. Then, the MBRs of the subsynopses are sorted on the Hilbert values of their centers. Finally, the list of MBRs is packed in groups of cardinality f (where f is the R-Tree fan-out), and for each group, a node is created containing the MBRs in the group as well as the identifiers of the corresponding subsynopses. These nodes are the leafs of the R-Tree. As it will be clearer in the following, this enhances the query evaluation, as it reduces the number of peers to be accessed for computing query answers.

E. Title and Author Details

After being populated, I is partitioned in "small"-size portions which are prone to be distributed across the network. The reason for partitioning the index is the same as for limiting the amount of storage space invested for a single synopsis, that is, distributing small-size index portions across the network prevents peers from being overloaded in terms of upload and download traffic needed for supporting index replication.

Fig. 5, shows the aforementioned index partitioning scheme. Unlike I^{inf} , the I^{sup} s-block is not subpartitioned: in fact, its size is very small (in all practical cases, smaller than 256 KB).

This aims at reducing the number of leaf portions to be accessed in order to evaluate range queries, as the data involved in a range query belong to subsynopses whose MBRs are close to one another. As it will be clearer in the following, since the index will be distributed across the network by assigning $inf_1, \ldots, inf_m$ to different peers, this strategy will result in reducing the number of peers to be accessed for locating the subsynopses involved in range queries..

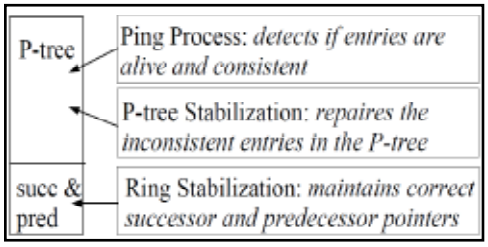

Fig. 4: P-Tree Structure

IV. Distributing Compressed Data

We now describe how the distribution of the synopsis and the index are performed.

A. System Primitives and Data Structures

We assume the existence of two system primitives named search and send. Primitive search(N)—which is used by the framework every time it is required to find sets of peers on the network—returns a set of N IP addresses of randomly-chosen peers. In order to choose a peer randomly, it suffices to locate a peer by starting a random walk of length greater than $\log_f N$ (where N is the number of peers in the network and f is the average fan-out) from the peer which invoked search. a random walk of this length makes the probability of reaching any peer converge to a stationary distribution, which is uniform if the network graph is well connected. Primitive send (P, o) transmits s-block o from the peer p which invoked the primitive to the peers whose IP addresses are in set P. This is achieved through decentralized dissemination: instead of sending | P | copies of o, p sends o to a subset of the peers in P which, in turn, keep a copy of o and forward it to different subsets of the remaining peers in P, and so on.

Our proposed distribution scheme makes use of a set of data structures named as location tables. Each location table will

be associated with a copy of an index portion and maintain correspondences between s-blocks and sets of peers. Specifically, the location table associated with I^{sup} will consist of a row for each leaf portion, plus a row for I^{sup} itself. Each row, in turn, will contain addresses of peers where copies of these index portions are hosted.

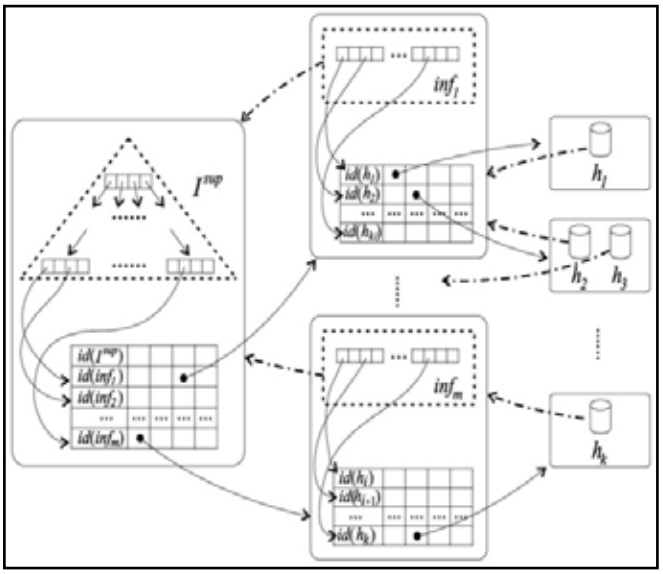

Fig. 5: s-Blocks and Location Tables

In a location table associated with a copy of a leaf portion infi, each row will contain the addresses of the peers hosting copies of a subsynopsis pointed by infi (see fig. 4, where rounded boxes represent peers). We denote the location tables associated with index portions as table (I^{sup}) and table(inf_i). At runtime, the local copies of these tables can be modified by the peers that host them (through the replication scheme described in Section V); hence, when needed to avoid confusion, we will denote the tables at a peer p as p: table(I^{sup}) and p: table(inf_i).

B. Disseminating Data and Index

The distribution process is started by a peer p that is willing to publish a data population, and works as follows: First, for each subsynopsis h_j (respectively, leaf portion inf_i), p invokes search(C_{min}) to find C_{min} peers which can host a copy of h_j (respectively, inf_i along with table(inf_i). Then, for each inf_i and subsynopsis h_j referenced by inf_i, location table table(inf_i) is filled with the IP addresses of the peers which will host h_j. Correspondingly, each h_j is augmented with a reverse pointer to one of the peers which will host inf_i. A similar process is performed to find C_{min} peers which will host I^{sup} along with a location table, and to fill the table as well as the reverse pointers of leaf portions. In particular, as explained before, the location table of each peer that will host a copy of Isup is filled with the addresses of the other peers which will host copies of I^{sup}. After all of the location tables have been filled, the copies of s-blocks along with their associated location tables are sent to the appropriate peers.

V. Distributed Querying

Peers participating to the network can issue two types of queries: explorative queries and range queries. In the following, we describe their semantics and evaluation process.

A. Explorative Queries

Explorative queries locate the data populations available over the network that match the interest of a user. Specifically, an explorative query Q_{exp} issued by a peer p interested in a data population D specifies either the identifier of D or the name, or a list of keywords associated with D. The answer of Q_{exp} is a set of peers (identified by their IP addresses) hosting the I^{sup} portion of compressed data populations matching the criteria specified in Q_{exp}. In our current prototype, an explorative query Q_{exp} issued by a peer p is processed by starting a random walker from p, which is propagated through the network until either the number of accessed peers satisfying Q_{exp} reaches the maximum number of results specified by p, or the overall number of accessed peers reaches a given threshold.

When the random walker reaches a peer p^0 hosting an s-block of a population D matching Q_{exp}, p^0 sends the following information to p:

- the metadata associated with D (such as the identifier, the exact name, and the range of the data domain);
- the number of intermediate peers accessed by the random walker between p and p0 (this number will be exploited by our replication strategies).

The result of an explorative query is a two-column table (called shortcut table), whose columns contain, respectively:

- the identifiers of the data populations matching the query, associated with metadata describing them;
- for each data population D reported in the first column, the list of IP addresses of the peers hosting D:I^{sup}; for each of these peers p^0, the IP address is stored along with the length of the path walked over the network to reach p^0.

Table 1: Partial Query Answers

$id(inf_i)$	ans	l	$A(inf_i)$
$id(inf_1)$	0	1	$\{\langle id(h_1), 3\rangle\}$
$id(inf_3)$	12	2	$\{\langle id(h_4), 8\rangle, \langle id(h_7), 6\rangle\}$
$id(inf_7)$	36	0	$\emptyset$

The "overall" answer of the query Q issued by p is computed by p itself by assembling the "partial" contributions received from different peers.

Query recovery. The messages sent to p from the peers involved in the evaluation of Q are exploited in a mechanism for managing critical events (such as peer departures and network failures), which avoids the reexecution of Q from scratch. As messages are received, peer p stores the partial answers of Q into a data structure of the form shown in Table 1. This structure consists in a table where each row is associated with a leaf portion inf_i and contains id(inf_i), two values ans, l (whose meaning will be clearer in the following) and a set A(inf_i) of pairs of the form hid(h_j); ansi, where, h_j is a subsynopsis referenced by inf_i.

Table 1, refers to a case where three leaf portions (inf_1, inf_3, and inf_7) and three subsynopses (h_1, h_4, and h_7) are involved in the query. Observe that A(inf_7) = ; (third row), meaning that all the MBRs of inf_7 overlapping the query range are fully contained into it. Analogously, the row associated with inf_1 contains ans = 0, meaning that all the MBRs of inf_1 overlapping the query range are not fully contained into it.

VI. Dynamic Replication

Our dynamic replication scheme aims at both providing the appropriate coverage of s-blocks and balancing the load at the peers. To this aim, besides guaranteeing a minimum coverage for each s-block (so that published data remain accessible over time), our replication scheme provides adaptivity to the dynamic query workload by creating new replicas of an s-block each time

it is queried and by removing less queried data through suitable aging policies.

A. Guaranteeing the Minimum Coverage

In our framework, location tables encode links among s-blocks spread over the network. Thus, they are kept updated w.r.t. events causing data unavailability by deleting the addresses of the peers that no longer host these data. Our approach is independent of the way the unavailability of data is identified; in practice, this can be done through periodic pinging (as in our prototype) or notification protocols.

Moreover, as said before, the survivability of the minimum number of copies of Isup across the network is controlled by a clique of peers, which is initially composed of the Cmin peers that are assigned Isup when the population is published. When a peer in the clique exits the system, a new one is chosen (through an election process among the peers in the clique) to host a new copy of Isup and become part of the clique. As it will be clearer in the following, other copies of Isup may be created at runtime to enhance data reachability in response to an increasing interest in those data; however, the peers that host these copies do not belong to the clique.

B. Query -Based Replication

We now describe two replication strategies, called Path-Based (PBS) and Reactive (RS), that aim at increasing the availability of most queried data, also pursuing load balancing when facing large and dynamic query workloads. Both strategies are inspired by the path replication strategy. This strategy replicates a data set each time a query is issued on it, and the number of replicas created is proportional to the length of the path the query has traversed to reach the data portion.

With both the proposed strategies, each peer p maintains a queue M(p) of pairs of the form hid(D); IP i, each represent-ing a pending range query on population D issued by the peer with address IP . Moreover, p chooses a threshold Mt(p) on the number of pending queries in the queue, which determines when p considers itself as overloaded.

1. Path-Based Strategy (PBS)

This strategy may be seen as a direct adaptation of the path replication strategy to the case of hierarchically organized data, like our three-level structure consisting of the inner portion of the index, its leaf portions, and the subsynopses. With PBS, peer pa takes a set PIsup = search(nrep) and sends them a new copy of D:Isup and its associated location table. In turn, each peer in PIsup first creates one replica of each leaf portion whose range intersects rQ, and then, augments table(D:Isup) with the addresses of the peers hosting the new copies of the leaf portions. Each peer that receives a copy of a leaf portion performs a similar process to replicate the subsynopses whose ranges intersect rQ.

2. Reactive Strategy (RS)

To summarize, PBS consists in a fine-grained version of the technique (which has been shown to be optimal in terms of reachability), as it performs the appropriate number of replications of only the s-blocks involved in the queries. RS aims at guaranteeing the same reachability as PBS while preventing useless replicas from being created. To accomplish this, it spreads IP addresses instead of copies of data and performs new data replications only when peers hosting the existing replicas are overloaded.

C. Storage Space Management

We now detail our data replacement strategy for managing the local storage space in a peer in response to a request to host new data. This strategy allows the framework to take into account the recent interest in data portions exhibited by users when choosing the s-blocks to be removed in order to free storage space.

By exploiting our data distribution scheme, this replacement strategy prevents the removal of s-blocks that are needed to maintain the minimum coverage, and is also capable of favouring the availability of recently queried s-blocks.

Table 2: Parameter Values

Name	*Meaning*	*Value*
B_{max}	Maximum size of an s-block	256KB
$\mathcal{N}$	Number of peers (steady state)	10^4
η	Number of data populations	50
f	Average fan-out (as in Gnutella [18])	8
λ_o	Average on-line time of peers	1 hour
λ_j	Average delay between new peer connections	360 ms
C_{min}	Minimum number of copies of any s-block referenced by location tables (coincides with the number of copies of I^{sup} belonging to the clique)	3
$M_t(p)$	Threshold on the number of queries in queue (after which p considers itself as overloaded)	random in [2..4]

VII. Conclusions and Future Work

We implement a framework for sharing and performing analytical queries on historical multidimensional data in unstructured peer-to-peer networks. In our approach, the resources are maintained across P2P network for the possibility of accessing and posing queries against the data published by others. Our solution is based on suitable data partitioning and indexing techniques, and on mechanisms for data distribution. The testing results showed the effectiveness of our approach in providing fast and accurate query answers, and ensuring the robustness that is mandatory in peer-to-peer setting.

Future work will be devoted to considering data updates. On the one hand, updates along the temporal dimension can be managed relatively easily. Queries over a time range that involves synopses referring to different time intervals could just be split into sub queries to be processed separately. On the other hand, removing the assumption of historical data makes the problem much more complex, as updates can affect the homogeneity of data, making the results of both the partitioning and the compression steps obsolete. The crucial objective is, therefore, that of avoiding the computation of the partitioning and the construction of the sub synopses from scratch by possibly detecting the regions of data whose features are not significantly affected by the update. This would limit the computational load for computing the up to date synopsis, as well as the network traffic for replacing old data. The absence of a centralized coordination in our setting poses further challenges, as it makes it necessary to devise a nontrivial mechanism for distinguishing among old and new data during query evaluation.

References

[1] B. Arai, G. Das, D. Gunopulos, V. Kalogeraki,"Approximating Aggregation Queries in Peer-to-Peer Networks", Proc. 22nd Int'l Conf. Data Eng., 2006.

[2] M. Bawa, H. Garcia-Molina, A. Gionis, R. Motwani, "Estimating Aggregates on a Peer-to-Peer Network", Technical report, Stanford InfoLab, 2003.

[3] N. Bruno, S. Chaudhuri, L. Gravano,"STHoles: A Multi-dimensional Workload-Aware Histogram", Proc. 2001 ACM SIGMOD, 2001.
[4] S. Chaudhuri, U. Dayal,"An Overview of Data Warehousing and OLAP Technology", Sigmod Record, Vol. 26, No. 1, pp. 65-74, Mar. 1997.
[5] E. Cohen, S. Shenker,"Replication Strategies in Unstructured Peer-to-Peer Networks", Proc. ACM SIGCOMM '02, 2002.
[6] A,Crainiceanu, P. Linga, J. Gehrke, J. Shanmugasundaram, "Querying Peer-to-Peer Networks Using P-Trees", Proc. Seventh Int'l Workshop Web and Databases, 2004.
[7] F. Furfaro, G.M. Mazzeo, C. Sirangelo,"Exploiting Cluster Analysis for Constructing Multi-Dimensional Histograms on Both Static and Evolving Data", Proc. 10th Int'l Conf. Extending Database Technology, 2006.
[8] P. Ganesan, M. Bawa, H. Garcia-Molina,"Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems", Proc. 30th Int'l Conf. Very Large Data Bases, 2004.
[9] D. Gunopulos, G. Kollios, V.J. Tsotras, C. Domeniconi, "Approximating Multi-Dimensional Aggregate Range Queries over Real Attributes", Proc. ACM SIGMOD, 2000.
[10] A. Gupta, D. Agrawal, A. El Abbadi,"Approximate Range Selection Queries in Peer-to-Peer Systems", Proc. First Conf. Innovative Data Systems Research, 2003.
[11] Y.E. Ioannidis, V. Poosala,"Balancing Histogram Optimality and Practicality for Query Result Size Estimation", Proc. ACM SIGMOD, 1995.
[12] H.V. Jagadish, B.C. Ooi, Q.H. Vu,"BATON: A Balanced Tree Structure for Peer-to-Peer Networks", Proc. 31st Int'l Conf. Very Large Data Bases, 2005.
[13] M. Jurgens, H.-J. Lenz,"The Ra-Tree: An improved R-Tree with Materialized Data for Supporting Range Queries on OLAP-Data", Proc. Ninth Int'l Workshop Database and Expert Systems Applications, 1998.
[14] R. Kooi,"The Optimization of Queries in Relational Databases", PhD thesis, Case Western Reserve Univ., 1980.
[15] N. Koudas, C. Faloutsos, I. Kamel,"Declustering Spatial Databases on a Multi-Computer Architecture", Proc. Fifth Int'l Conf. Extending Database Technology, 1996.
[16] M. Muralikrishna, D.J. DeWitt,"Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional", Proc. Int'l Conf. Management of Data, 1988.
[17] T. Pitoura, N. Ntarmos, P. Triantafillou,"Replication, Load Balancing and Efficient Range Query Processing in DHTs", Proc. 10th Int'l Conf. Extending Database Technology, 2006.

Mr.SURESH VAYYASI received his B.E. degree in ECE from University of Madras, Tamilnadu,India, in 1989, the M.Tech degree in Computer Science & Technology from Andhra University College of Engineering, Visakhapatnam, Andhra Pradesh,India, in 1992, doing Ph.D degree in computer science Engineering from Andhra University College of Engineering, Visakhapatnam, Andhra Pradesh, India,. He was an Assistant Professor in MVGR College of Engineering, vizianagaram, in 2000 and AITAM,Tekkali,in 2003. He was an Associate Professor in Chaitanya Engineering college,Kommadi, visakhapatnam,in 2005. Presently he is working in Pydah College of Engineering and Technology, Gambheeram, Visakhapatnam, Andhra Pradesh. His research interests include Embedded Systems and design methedologies.

Amzed Ali Shaik was born in Gajuwaka, Visakhapatnam District, Andhra Pradesh, India. He received his AMIE in Computer Engineering Branch in the year 2001 from The Institution of Engineers(India), Kolkata, India and also his M.B.A.(Human Resource and Marketing) in the year 2010 from Jawaharlal Nehru Technological University, Kakinada, Andhra Pradesh, India. Presently, he is pursuing M.Tech in C.S.E. from Pydah College of Engineering and Technology, Gambheeram, Visakhapatnam District, Andhra Pradesh, India. His research interest includes Database Management Systems, Data Mining and Programming Languages.