Advances in Data Management - P2P Data Management
A.Poulovassilis

1 Introduction

In **peer-to-peer (P2P)** systems a number of autonomous servers, or *peers*, share their computing resources, data and services.

Each peer can be both a provider and a consumer of services.

P2P systems have many advantages:

- scalability — by addition of extra peers
- self-organisation — no centralised control or information
- availability — by replication
- adaptability and fault-tolerance — robust to node and network failures; no single point of failure; self-repairing

But they also present several challenges:

- security —
  - trustworthiness of peers e.g. in correctly responding to messages and requests, in correctly routing messages to other peers
  - monitoring of denial-of-service attacks; detection and management of such attacks
  - digital rights management
- job scheduling —
  - generating execution plans given the available levels of knowledge regarding the capability, availability and reliability of peers
  - monitoring execution of distributed applications and handling failures
  - monitoring and achieving QoS requirements e.g. with respect to response time, cost, number of query results, relevance of query results
- query performance — finding the data relevant to a query and processing the query
- transactional semantics — peers may fail, and may join/leave the network at any time
- data exchange and integration — heterogeneity of peers' data

In keeping with the previous themes of this ADM course, I will focus on data-sharing P2P systems and the last three of these challenges.

2 Query performance

Data-sharing P2P systems distribute the cost of data storage, access and management across the peers of the network.

Queries can be submitted to a peer and they need to be answered with respect to the entire available data distributed across the peers.

There are three dimensions that effect query performance:

- network topology
- data placement
- message routing protocol

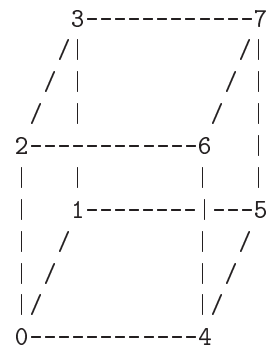
The **network topology** might be

- unstructured, peers connect to whomever they wish
- structured according to some protocol e.g. HyperCup

The HyperCup topology guarantees that:

- each peer receives a message only once
- if the number of peers is N, a total of $N-1$ hops are required to reach all the peers
- the most distant peers are reached after $\log_d N$ hops (where $d \geq 2$ is the dimensionality of the hypercube) — this is known as the network diameter

E.g. see below for a 2-dimensional hypercube topology — the numbers represent nodes in the network; the connections also have unique labels (not shown):



Data placement defines how data and metadata is distributed; it might be:

- according to ownership: each peer stores only its own data and metadata
- according to some distributed data placement algorithm
- according to the semantics of the data e.g. in Edutella there is schema-based clustering of peers at superpeers

Message routing protocol:

- this defines how messages are transmitted from peer to peer
- it can take advantage of the network topology, the data placement strategy, and any routing indexes available at the peers
- P2P networks that broadcast all queries to all peers do not scale: as the number of peers increases, the message transmission costs increase exponentially

For example, in Edutella:

- superpeers are connected by a HyperCuP topology
- each other peer is connected to only one superpeer
- the information stored at each peer is described by an RDF schema
- there is schema-based clustering of peers at superpeers
- superpeers maintain routing indexes which summarise the information stored in their peergroup
- these can be used to determine whether an RDF query arriving at the superpeer is forwarded to each peer in its peergroup
- there is summarised global schema information maintained at each superpeer to support routing of queries from one superpeer to another

3 Transactional semantics

In theory, any distributed concurrency control protocol could be adapted to a P2P environment e.g. the AMOR system adopts optimistic concurrency control:

- As local transactions execute at a peer, a local serialisation graph is maintained at the peer
- AMOR assumes that conflicts are only possible between those transactions that are accessing a particular 'region' of resources — e.g. superpeers and their local peergroup
- Thus, subgraphs of the global serialisation graph only need to be replicated amongst those transaction managers which service a particular region
- However, the regions are not static and these subgraphs are dynamically merged and replicated as transactions execute and regions evolve.

In the classical approach to distributed transactions, global transactions hold on to the resources necessary to achieve their ACID properties until such time as the whole transaction commits or aborts.

In a P2P environment this may not be feasible:

- the resources available at peers may be limited
- peers may not wish to cooperate in the execution of global transactions
- peers may disconnect at any time from the network, including during the execution of a global transaction in which they are participating

- the triggering and execution of ECA rules will cause longer-running transactions which may further exacerbate these problems.

It is therefore necessary to relax the Atomicity and Isolation properties of transactions.

In particular, subtransactions executing at different peers may be allowed to commit or abort independently of their parent transaction committing or aborting, and parent transactions may be able to commit even if some of their subtransactions have failed.

Subtransactions that have committed ahead of their parent transaction committing can be reversed, if necessary, by executing *compensating* subtransactions (see earlier notes in this course).

These are generated as transactions execute and they reverse the effects of a transaction by compensating each of the transaction's updates in reverse order of their execution.

If transactions have read from committed (sub)transactions which are subsequently compensated, then a *cascade of compensations* will result.

If we assume as the default that a parent transaction (or subtransaction) and its immediate subtransactions are able to commit independently of each other, then an *abort dependency* needs to be specified for each parent (sub)transaction and child subtransaction.

Possible abort dependencies are as follows, with T_p being the parent and T_c the child:

- **ParentChild**: If T_p aborts then T_c is to abort.
- **ChildParent**: If T_c aborts then T_p is to abort.
- **Mutual**: If either T_p or T_c aborts then so must the other.
- **Independent**: There is no abort dependency between T_p and T_c.

For coordinating the execution of compensating transactions or subtransactions, an *abort graph* can be maintained that describes the abort dependencies between parent transactions and their subtransactions.

The abort graph will be distributed amongst the peers that participate in any subtransaction of a top-level transaction.

The graph can be constructed dynamically with each new subtransaction.

In particular, each time a transaction T_n at a peer P_i initiates a new subtransaction T_m to be executed at a peer P_j (where it may be that $i = j$) then depending on the abort dependency between T_n and T_m, the following actions are taken:

1. **ParentChild**: The identifier of T_m and the peer P_j that it will execute on are transmitted to P_i and recorded there, together with an arc $T_n \rightarrow T_m$ in the local abort graph at P_i.
2. **ChildParent**: The identifier of T_n and the superpeer P_i that it is executing on are transmitted to P_j and recorded there, together with an arc $T_m \rightarrow T_n$ in the local abort graph at P_j.
3. **Mutual**: A combination of the actions for **ParentChild** and **ChildParent** above is taken.
4. **Independent**: No local abort graph is updated.

In case of a subtransaction failure, all the necessary information is now available to initiate a compensating subtransaction, at any level of nesting of the subtransaction.

Figure 1 gives an example of a distributed abort graph.

A failure in subtransaction T_7 at SP_3 leads to compensation of T_7 at SP_3 but leaves the rest of the transaction unaffected, while a failure in subtransaction T_6 at SP_5 initiates a compensating transaction for T_6 at SP_5, a compensating transaction for T_1 at SP_2 (due to the **Mutual** abort dependency between T_6 and T_1), and a compensating transaction for T_5 at SP_2 (due to the **ParentChild** dependency between T_1 and T_5).

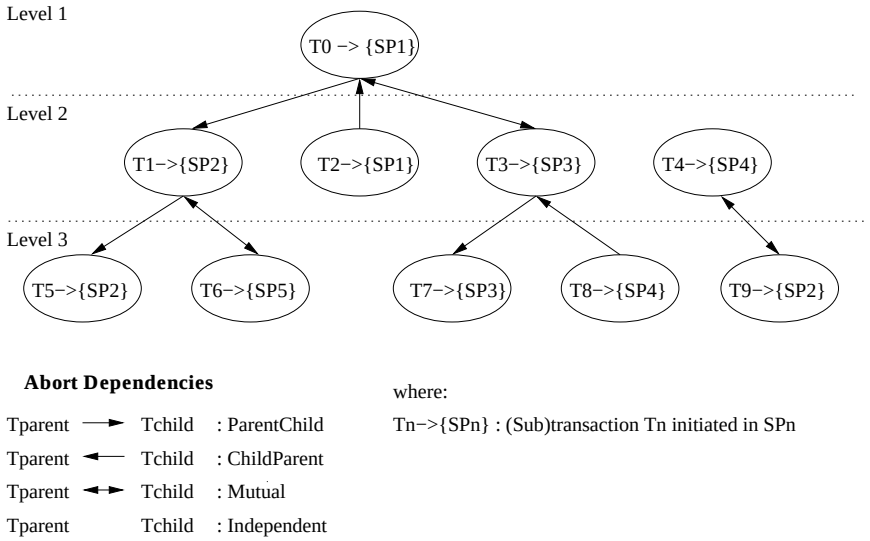


Figure 1: Abort graph example

4 Data Exchange and Integration

File-sharing P2P systems exchange files that are identified by a limited set of attributes e.g. name, type, location and possibly a description of the file.

The lack of information about the data *within* these files makes it impossible to support general-purpose mechanisms by which peers can exchange and integrate heterogeneous data.

Thus, current research is aiming towards *schema-based* P2P data exchange and integration e.g. Edutella (L3S, U. Hannover), Piazza (U. Washington), AutoMed (Birkbeck and Imperial).

In particular, we have seen in earlier lectures how AutoMed supports a *both as view* (BAV) approach to transforming and integrating heterogeneous data.

Previous data integration approaches have been either **global as view (GAV)** or **local as view (LAV)**.

In GAV, a global virtual schema is defined as a set of views over one or more data source schemas.

These view definitions are used to rewrite queries over the global schema into queries over the data sources.

One disadvantage of GAV is that it does not readily support the *evolution* of source schemas, since a change in a source schema construct impacts on all the GAV view definitions depending on that construct.

In LAV, the constructs of the source schemas are defined as views over a global virtual schema, and processing queries over the global schema involves "rewriting queries using views".

LAV confines changes to a source schema to impact only on the view definitions defined for that schema.

However, LAV has problems if one needs to evolve the global schema, since all the view definitions for the source schemas will need to be reviewed.

In BAV, schemas are mapped to each other using a sequence of schema transformations — a transformation

pathway.

From such pathways it is possible to extract the definition of a global schema construct as a view over source schemas (i.e. GAV) and it is also possible to extract definitions of source schema constructs as views over a global schema (i.e. LAV).

One advantage of BAV over GAV and LAV is that it readily supports the evolution of *both* global and data source schemas: such evolutions can be expressed as extensions to the existing pathways, and new view definitions can then be regenerated from the new pathways as needed for query processing.

This feature makes BAV well suited to the needs of P2P data integration, where autonomous peers may join or leave the network at any time, may change one of their data source schemas, or may add or drop a schema.

In particular, consider a network of servers each supporting an installation of the AutoMed system, with simple P2P communication supported between these servers.

Each peer is able to make public any of the schemas stored in its local AutoMed repository e.g. by using a directory service such as UDDI.

Each peer is able to create pathways between its own local schemas and schemas made public by itself or other peers. Such pathways are stored in the peer's own AutoMed repository.

Peers are also able to publish the information that they support a pathway to a public schema (without publishing the actual pathway).

Suppose a peer P wishes to send a query formulated with respect to one of its local schemas, LS, to other peers that have access to data semantically related to LS.

P can find out to which public schemas, S say, there exists in its own repository a pathway $LS \rightarrow S$.

P can also find out which other peers support pathways to S by consulting the directory service.

Suppose P' is such a peer. Then P can request from P' its set of pathways to S.

Suppose $L' \rightarrow S$ is one of this set of pathways.

P can then combine the reverse pathway $S \rightarrow L'$ with its own pathway $L \rightarrow S$ in order to generate a pathway from L to L' (consisting of $L \rightarrow S$ followed by $S \rightarrow L'$).

P can then use this pathway to automatically translate a query expressed on its own schema L to an equivalent query expressed on L' which can then be sent to P' for processing.

As well as its ready support for schema evolution in P2P data integration environments, another advantage of BAV compared to LAV and GAV is that statements about the relationships between schemas can be made at a finer level of granularity e.g. for relational schemas we can make statements at the level of individual attributes as opposed to for entire tables as in LAV and GAV.

This makes it possible to assert 'exact' knowledge about some attributes of a table, and 'sound' or 'complete' knowledge about other attributes of the table:

An **exact** rule for an attribute a equates it with a set of constructs C in another schema by means of a formula f in a rule of the form $a = f(C)$

A **sound** rule has the form $f(C) \subseteq a$

A **complete** rule has the form $a \subseteq f(C)$

Moreover, BAV is schema-oriented and the query language used to specify the data mappings is independent of the schema mappings — different query languages to IQL could be used within BAV pathways and AutoMed could be extended with different query processors.

4.1 Current work on AutoMed

We are extending the existing AutoMed system in a number of ways so as to fully support data integration in schema-based P2P environments.

Since BAV pathways subsume GAV and LAV in expressiveness, we are experimenting with flexible combinations of GAV and LAV query processing over BAV pathways in order to fully utilise the semantic information present within them (AutoMed's current global query processor supports only GAV query processing).

We also plan to extend AutoMed to support P2P *update processing* along BAV pathways, with the updates being expressed in a limited fragment of the full IQL language.

For this we will use 'push'-based techniques whereby peers automatically transmit updates to their local data sources to other peers who have subscribed to receive them, either periodically or whenever updates occur.

To support this kind of functionality, we are investigating the use of event-condition-action rules to express consumers' information needs and how providers should respond to them, building on our recent work on event-condition-action languages for XML and RDF (George Papamarkos' PhD research).

We are also developing a Data Analysis tool that will allow automatic or semi-automatic generation of transformation pathways from a local schema to a public schema, so as to aid AutoMed peers in establishing semantic connections to public schemas.

Schema matching techniques that compare schema structure, naming and data extents can be used to identify possible relationships between the constructs of an AutoMed peer's local schema and the constructs of a public schema (ongoing research at Imperial).

Graph matching techniques can be used to undertake any further necessary restructuring of the schemas and to complete the transformation pathways (Lucas Zamboulis' PhD research).

The resulting pathways will be usable for query and update processing, and also for tracing the *provenance* of data between peers (Hao Fan's PhD).

4.2 Application and evaluation

We are currently applying and evaluating this new functionality in two areas: biological data grids, as exemplified by the ISPIDER project (BBK,UCL,Manchester,EBI), and P2P data exchange between autonomous vehicles, as exemplified by the RoDEx project (Imperial).

ISPIDER: The heterogeneous and autonomous nature of biological data resources requires that data can be exchanged between research groups in a flexible manner.

Biological data sources typically have a very high degree of heterogeneity in terms of the type of data model used, the schema design within a given data model, as well as incompatible formats and naming of values.

AutoMed's particular strengths within the ISPIDER project are its support for extensible schema transformations across multiple data models, support for both virtual and materialised data integration, and use of transformation pathways for tracing data provenance.

The functionality provided by AutoMed will aid in the generation of semantic associations between biological data resources in ISPIDER and in the use of these associations for peer-to-peer exchange of queries and data.

RoDEx: Autonomous vehicles to date have been designed as closed systems, and the objective of the DTC programme at Imperial is to build more flexible systems of intercommunicating autonomous vehicles.

The RoDEX project aims to provide robust methods of data exchange in environments with network and host outages and failures.

AutoMed has particular strengths in this area, since its transformation pathways can be passed around the

network to allow different peers to plan their communication, and also to allow peers to determine what information needs to be cached when communication with certain other peers becomes unreliable.

References

Open Problems in Data-Sharing Peer-to-Peer Systems, N.Diswani et al., Proc. ICDE 2003, pages 1-15.

HyperCuP - Hypercubes, Ontologies and Efficient Search on P2P Networks, M. Schlosser et al., First Int. Workshop on Agents and P2P Computing, Springer LNCS 2530, 2002, pages 112-124.

Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks, W. Nejdl et al., Proc. WWW 2003, pages 536-543.

Transctional peer-to-peer information processing: The AMOR approach, K. Haller et al., Proc. 4th Int. Conf. on Mobile Data Management, Springer, 2003, pages 356-362.

Event-Condition-Action Rules on RDF Metadata in P2P Environments, G. Papamarkos, A. Poulovassilis and P.T. Wood, to appear in *Computer Networks*, October 2006.