View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Edinburgh Research Explorer

THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

# A Peer-to-Peer Architecture for e-Science

**Citation for published version:**
Viglas, S 2006, 'A Peer-to-Peer Architecture for e-Science'. in UK e-Science All Hands Meeting.

**Link:**
Link to publication record in Edinburgh Research Explorer

**Document Version:**
Author final version (often known as postprint)

**Published In:**
UK e-Science All Hands Meeting

**General rights**
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

**Take down policy**
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 20. Feb. 2015

# A Peer-to-Peer Architecture for *e*-Science

**Stratis D. Viglas**
School of Informatics, University of Edinburgh, UK
sviglas@inf.ed.ac.uk

**Abstract**

One of the key issues in supporting *e*-Science is managing data in distributed, flexible, scalable and autonomous ways. This paper proposes an architecture based on Peer-to-Peer systems that can be used to facilitate large-scale distributed data management. It identifies the important problems that need to be addressed and presents possible solutions, along with their expected advantages. These ideas, we believe, are helpful in extending the discussion of alternative approaches of supporting the multiple facets of *e*-Science.

## 1 Introduction

Research organisations produce data at ever-increasing rates. Central management is impossible for a variety of reasons, including, but not limited to, the sheer volume of data, their rate of change, and their geographical distribution. This means that flexible and strictly distributed architectures need to be in place. The purpose of this paper is to present such an architecture with an increased focus on decentralised, scalable and reliable data management. That is not to say that high performance issues are to be discarded; rather, by focusing on reliable data management we can "free" the applications built on top of the data management layer to concentrate on application-specific performance issues without having to address management aspects as well.

**Large-scale decentralisation.** Accumulating information in central structures means that central points of failure are created, and fault-tolerance is decreased. The situation can be alleviated by employing overlay networks (*e.g.*, [17, 18, 19]) but, even in those cases, targeted attacks can still take place, while secondary maintenance protocols need to be continuously executed to keep the overlay structure updated. We would like the system not to have any rigidly imposed structure, but be fully adaptable.

**Increased autonomy.** In a decentralised system, nodes join and leave at will. There are no established "contracts" as to how long a node should be in the system, or of replication of the data available at a single node. The node itself manages its behaviour, along with access to the data it serves. This poses questions as to what connections the node establishes and what protocols are to be executed upon node arrivals and departures.

**Security.** Naturally, there is need for security, especially if there is no central management authority. We focus on user-level data access, so as not to (*i*) compromise data integrity, or (*ii*) allow access to users who the nodes of the system do not want to grant access to. Both these aspects are in direct accordance to the autonomy notion previously described.

**Efficient data retrieval and manipulation.** Performance in a decentralised system is of paramount importance; in our setting, performance means response time. Current research has addressed the problem by either (*i*) building high-bandwidth connections and relying on the speed of those connections to account for rapid data exchange, or (*ii*) building overlay networks and measuring performance in terms of the number of routing hops necessary to route data retrieval requests. Both metrics provide localised solutions to a truly global problem in a decentralised data management system. For instance, rapid data retrieval does not account for what data manipulation takes place over said data; or, the number of routing hops to locate data does not take into account data volume, or network latency. Being agnostic to the rest of the computing environment is not helpful in a decentralised data management scenario, especially one as intricate and collaborative as *e*-Science.

We aim to address all these issues by developing customisable middleware between the nodes comprising the system. Participating nodes will need only implement a specific interface, therefore being independent of any ties to operating systems or programming environments. In light of the needs for autonomy and decentralisation, we propose a Peer-to-Peer (P2P) architecture. In the following sections, we shall present such an architecture, focus on a subset of the problems at hand, and present possible solutions.

## 2 System Overview

The general overview of the system is shown in Figure 1. The main assumptions are: (*i*) Each peer exports its data in XML. This imposes no restriction on the peer's native data format; the only requirement is that an XML view of it is exported. (*ii*) Each new peer is introduced to the system by following a "hand-shake" protocol with an existing peer. (*iii*) Each peer maintains connections to other peers, forming its *routing table* that contains both semantic and structural links. (*iv*) While present in the system, the peer's routing table evolves to account for peer arrivals or departures. (*v*) To process queries, a peer first identifies the peers relevant to the query. The query is then rewritten in ways that can be processed by the relevant peers. (*vi*) A departing peer executes a specialised exit protocol.

### 2.1 Protocols

The functionality of the system can be summarised in three main protocols. The implementation of these protocols dictates additional building blocks of the architecture.

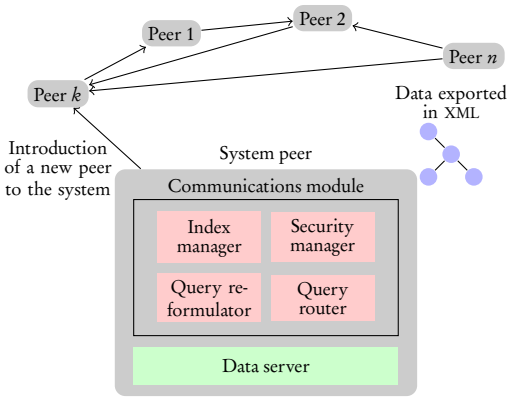



**Figure 1:** System overview and general architecture

**Entrance protocol.** This is the "bootstrapping" sequence executed by a peer joining the system. The requirement is that a joining peer knows of another peer that is already part of the system. Consider, *e.g.*, peer $P_n$ joining the system and being aware of peer $P_o$. There are two steps: (*i*) exporting of $P_n$'s data and indexing of its data at $P_o$, and (*ii*) exchange of information between the two peers concerning further nodes. The first step introduces $P_n$ to the system and makes it accessible by peers known to $P_o$, *i.e.*, $P_o$ is now capable of delegating queries to $P_n$. The second step accounts for the reverse direction, *i.e.,* for $P_n$ to be able to route queries to $P_o$; $P_o$ becomes part of $P_n$'s routing table and vice-versa. These steps can be recursively applied: $P_o$ can forward $P_n$'s credentials to peers it knows about and it can forward the credentials of those peers to $P_n$.

**Querying protocol.** Each peer should be in a position to retrieve data served by any other peer in the system. The querying protocol undertakes the following: (*i*) identifying the peers relevant to a query; (*ii*) translating the query to a destination peer's exported schema; (*iii*) forwarding the query to the destination peer and reporting the results back to the user; and (*iv*) while answering the query, update the originating peer's routing table with newly discovered information. These steps are not executed only at a single peer. Since multiple peers may be useful in answering a single query, the steps are recursively applied by all participating nodes. This forms the basis of *data discovery* and *query routing*, two salient features in decentralised indexing and query evaluation.

**Exit protocol.** On exiting, a peer disassociates itself from peers it knows about. Assuming peer $P_n$ leaves the network, during the exit protocol, it: (*i*) propagates information it has gathered during its stay in the system to peers it knows about, so as for any associations it has identified to "live on," and (*ii*) makes its data inaccessible to the peers it is connected to, so as to bring the system to a consistent state. As before, the first step can be recursively applied.

### 2.2 Modules

To realise the previous protocols, each peer has two modules: (*i*) a data server, responsible for exporting data to XML and locally evaluating queries, and (*ii*) a communication module, providing access to remote peers of the system (see also Figure 1). The responsibilities of the communication module are delegated to four entities, described next.

**Index manager.** The index manager is responsible for maintaining data connections between related peers. These connections are either (*i*) *semantic*, representing related concepts (*e.g.*, peers that maintain information about related scientific data may be aware of each other), or (*ii*) *structural*, used when reformulating and routing queries to relevant peers. Semantic connections are stored as mappings, while structural ones are stored as communication links.

**Security manager.** Autonomy is one of the important aspects of the architecture, *i.e.,* a peer should be responsible for both serving data, as well as controlling access to it. This means that security policies need to be in place. In the spirit of decentralisation, this information needs to be completely distributed throughout the system.

**Query reformulator.** The query reformulator is responsible for rewriting queries before forwarding them to remote peers. It "translates" the query at hand to use terms that are known to the destination peers. For instance, if a query is to be routed to institutions using different words for the same term, the query reformulator consults local mappings and forwards the query rewritten in ways that can be processed by the remote peers.

**Query router.** After relevant peers have been identified and the queries have been reformulated, the question is how should these queries be evaluated. Query performance characterises system performance, so it is crucial to be addressed in an efficient manner. In a dynamic and decentralised system, local optimisation decisions may prove quite limiting. We propose query evaluation through routing; the entity routed can either be a part of the query, or a partial result of the query, or even the entire query if this is deemed the best evaluation strategy.

## 3 Research Issues

We now turn to the core research agenda of our proposal, decomposed into four major categories: indexing, security, query reformulation and query evaluation.

### 3.1 Decentralised Indexing

Each peer autonomously manages and extends its own routing table to be used during query evaluation. The general form of this index structure is shown in Figure 2a. A local routing table is conceptually a three-column relational table. The first column is a destination peer, the second a local term and the third column a remote term (*i.e.,* a part of the schema exported by the destination peer). As shown in Figure 2a, a peer's routing table contains both semantic and structural information. Any entry of the routing table contains structural information; semantic information comes into play by maintaining *term mappings* wherever that is applicable. For instance, the first row in Figure 2a's routing table means that local term $X$ maps to remote term $A$. Note that the same peer (*e.g.*, Peer 1) may appear multiple times in the routing table. In addition, the same local term may be mapped to multiple remote

terms at different peers (*e.g.,* Peers 1 and $i$ in Figure 2a). Finally, the same remote term may be served by multiple peers (*e.g.,* for Peers 1 and $n$ in Figure 2a). This give us substantial flexibility in identifying relevant peers or even choosing between alternative peers serving related information.

The questions that arise have to do with forming and maintaining routing tables: (*i*) How much information is exchanged whenever a new peer joins the network? Alternatives include the new peer "downloading" the existing peer's entire routing table, or a part of it. (*ii*) How is the routing table updated as peers join and leave the network? One option is to "piggy-back" the routing table updates when accessing remote peers for the purposes of querying. Another option is to have a maintenance protocol being executed periodically. (*iii*) How can the routing table be further utilised during query evaluation? In particular, can the maintenance protocol provide performance guarantees about the connections stored in the routing table (*e.g.,* latency, probability of the connection being up to date *etc.*). Decentralised indexing allows for a great deal of research to be undertaken in the area.

### 3.2 Security

Data management means that, in addition to serving, peers manage access to the served data. The question that emerges is one of security: how can peers control which peers access their data? One solution is for each peer to request each other peer to register with it. However, this solution will not scale: it goes against the idea of complete decentralisation (as it means that each peer is aware of all peers in the system), while it also inhibits the maintenance protocols described earlier (as even the slightest local changes need to be globally transmitted). Additionally, we would like all forbidden requests to *fail fast*, *i.e.,* to fail as soon as possible – ideally at the originating peer. This means that each peer not only keeps track of who has access to its data, it also maintains information about what data it has access to.

The solution we propose is based on XML security views [8]. Each peer exports different views of its data to different peers, depending on the peer it is communicating with. An example is shown in Figure 2b where Peer $k$, exports different views to Peers $i$ and $j$. The system adheres to the fail fast principle: since neither Peer $i$ nor Peer $j$ are aware of the data they cannot access, they cannot request it.

### 3.3 Query Reformulation

Query reformulation can be thought of as the query compilation step in a decentralised system. During query reformulation the system executes a *resource discovery* protocol, which aids in: (*i*) identifying peers that may contain relevant terms; (*ii*) translating the query to terms that are understandable by other peers; and (*iii*) ensuring that each requesting peer has access to a particular term by consulting security policies.

The three steps mentioned above are iteratively executed. For instance, in Figure 2a, if a query about term $X$ is received, the local peer knows that in addition to accessing its local data, it needs to forward an appropriate query to Peer 1. The query is formulated by translating term $X$ in the query to term $A$ for Peer 1 to be able to handle it. In addition, the same procedure can be undertaken once the reformulated query reaches Peer 1 and for term $A$. The outcome of this process is a path $(P_1, Q_1)/(P_2, Q_2)/\ldots/(P_n, Q_n)$ with the semantics that at each peer $P_i$ the corresponding reformulated query $Q_i$ should be evaluated.

The path may indeed contain duplicate peers, *i.e.,* the same peer may have to be visited multiple times in processing a query. The most efficient way of accessing such peers is an issue of query optimisation and evaluation and is the purpose of the query router module.

### 3.4 Query Evaluation

An integral part of query evaluation is query optimisation. Already a hard problem in centralised environments, the situation is aggravated in decentralised ones as the likelihood of optimisation-time assumptions holding during evaluation-time is even lower. We propose decomposing queries into a query algebra and reducing query evaluation to a routing problem. We shall use two basic routing/evaluation strategies, shown in Figures 2c and 2d. The first is *parallel evaluation*: data requests are forwarded to peers, which then upload their data to the requesting peer that locally evaluates the query. The alternative is *serial evaluation*: a route is established and all peers are visited serially until the complete result is produced.

The two approaches can be better explained through an example. Consider a relational query of the form $R_1.a_1 = R_2.a_2 = \ldots = R_n.a_n$ posed at peer $P_k$ where each $R_i$ resides on a different peer $P_i$ of the system. The parallel strategy would send requests for each $R_i$ to be transmitted to $P_k$ and for $P_k$ to locally evaluate the query. The serial strategy instructs that the query be sent to $P_1$, the peer responsible for $R_1$, which then rewrites the query[1] and forwards it to $P_2$, the next peer in the sequence. The process is repeated until $P_n$ is reached, which then sends the result to the originating peer $P_k$.

Note that the original query can be rewritten in many ways. In the previous example the original query can be decomposed into numerous blocks[2]; each block can be evaluated in parallel or serially. Each peer makes local routing, and, hence, optimisation decisions. Finally, note that a single peer may need to be visited multiple times. In such cases, care needs to be taken so that the number of times a single peer is visited is minimised.

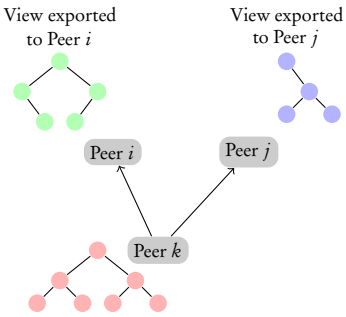
To summarise, we envision query evaluation through query routing to be a continuous cycle of (*i*) mapping the query to a query algebra and forming query blocks, (*ii*) performing local optimisation to identify whether parallel or serial evaluation is more beneficial for a particular block, (*iii*) forwarding a query block to the peers storing data relevant to the block, and (*iv*) rewriting the query to

[1] A simple, but most likely inefficient, rewrite would be to substitute values in $R_1.a_1$ with constants.

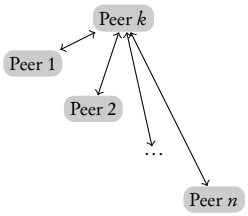
[2] $\sum_{i=1}^{n} i!\binom{n}{i} = \sum_{i=1}^{n} \frac{n!}{(n-i)!}$ to be exact.

| Peer | Local | Remote |
|---|---|---|
| 1 | $X$ | $A$ |
| 1 | | $B$ |
| 2 | $Y$ | $C$ |
| … | … | … |
| $i$ | $X$ | $F$ |
| $j$ | $Z$ | $C$ |
| … | … | … |
| $n$ | | $B$ |

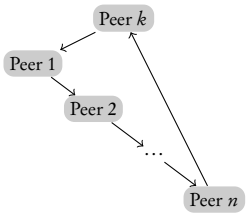
**(a)** Indexing




**(b)** Security views




**(c)** Parallel evaluation




**(d)** Serial evaluation

**Figure 2:** Various aspects of the proposed architecture

adjust for the partially computed result. The metrics to be employed in such an evaluation paradigm stem from three directions: (*i*) number of routing steps needed to evaluate the query; (*ii*) number of times a single peer is visited in evaluating the query; and (*iii*) raw size of data exchanged. Optimising for different dimensions, or combining all three metrics into a single one presents a very interesting multi-objective optimisation problem.

## 4 Related Work

There has been a host of work on P2P overlay networks and decentralised data structures in general (*e.g.,* [1, 3, 4, 6, 10, 16, 17, 18, 19]). These aim to solve the problem of efficiently identifying the peers of a system responsible for some particular data item by implementing a dictionary interface. In our case, the objective is to have highly unstructured networks that evolve as peers join and leave.

Another large area of work is concerned with distributed catalogs and covers peer data management systems (*e.g.,* [13, 20, 21]) and semantic overlay networks (*e.g.,* [5, 11]). In terms of query evaluation, there has been plenty of work on parallel databases (see *e.g.,* [7]) and distributed query processing (see *e.g.,* [14]). These approaches address environments of rigid structure and high predictability, while later studies have focused on the unpredictable behaviour of P2P systems (*e.g.,* [2, 9, 12]). All these focus on specific sub-problems without proposing a single, modular framework that is conducive to *e*-Science.

Finally, though *e*-Science oriented, existing approaches like the OGSA-DAI framework [15] address the problems at highly structured environments, without addressing intermittent peer behaviour or differing security policies. Rather, they focus on data integration and data delivery over Grid-like environments. It is certainly interesting to explore collaboration between the two approaches.

## 5 Conclusions

We have presented an architecture for building scalable data management systems over Web Services. We have focused on presenting the important problems in such a framework, along with solutions that appear to be viable at this stage. We have started implementing a prototype of the architecture with encouraging results. These results, we believe, are good initial steps in verifying the viability of our approach.

## References


[1] K. Aberer. P-Grid: A Self-organizing Access Structure for P2P Information Systems. In *CoopIS*, 2001.

[2] P. Boncz and C. Treijtel. AmbientDB: Relational Query Processing in a P2P Network. In *DBISP2P*, 2003.

[3] I. Clarke *et al.* Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science*, 2009, 2001.

[4] A. Crainiceanu *et al.* Querying Peer-to-Peer Networks Using P-Trees. In *WebDB*, 2004.

[5] A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, Computer Science Department, Stanford University, 2003.

[6] A. Datta *et al.* Range queries in trie-structured overlays. In *IEEE International Conference on Peer-to-Peer Computing*, 2005.

[7] D. J. DeWitt and J. Gray. Parallel database systems: The future of high performance database systems. *Commun. ACM*, 35(6), 1992.

[8] W. Fan *et al.* Secure XML Querying with Security Views. In *SIGMOD*, 2004.

[9] L. Galanis *et al.* Processing Queries in a Large P2P System. In *CAiSE*, 2003.

[10] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems. In *VLDB*, 2004.

[11] A. Halevy *et al.* Piazza: Data management infrastructure for semantic web applications. In *WWW*, 2003.

[12] R. Huebsch *et al.* Querying the Internet with PIER. In *VLDB*, 2003.

[13] G. Karvounarakis *et al.* RQL: A Declarative Query Language for RDF. In *WWW*, 2002.

[14] D. Kossmann. The State of the Art in Distributed Query Processing. *ACM Comp. Surveys*, 32(4):422–469, 2000.

[15] OGSA-DAI. `http://www.ogsadai.org.uk`.

[16] S. Ramabhadran *et al.* Brief announcement: Prefix hash tree. In *PODC*, 2004.

[17] S. Ratnasamy *et al.* A Scalable Content-Addressable Network. In *SIGCOMM*, 2001.

[18] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.

[19] I. Stoica *et al.* Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*, 2001.

[20] I. Tatarinov *et al.* The Piazza Peer Data Management Project. *SIGMOD Record*, 32(3), 2003.

[21] P. Valduriez and E. Pacitti. Data Management in Large-scale P2P Systems. In *VECPAR*, 2004.