Query Planning in the PORDaS P2P Database System

Kjetil Nørvåg[1], Eirik Eide and Odin Hole Standal
Dept. of Computer Science
Norwegian University of Science and Technology
Trondheim, Norway
[1] Kjetil.Norvag@idi.ntnu.no

Journal of Digital
Information Management

ABSTRACT: *Computational science is a rapidly growing multidisciplinary field that has a need for scalable, distributed, and efficient data management. In our research, we see the peer-to-peer (P2P) paradigm a possible solution to some of the problems in distributed data management. P2P has already proved to be suitable in contexts like file sharing, distributed computations, and distributed search. In our research we are aiming at using P2P to solve some problems in the domain of distributed databases. In this paper we 1) present PORDaS, a distributed DBMS based on P2P techniques, 2) describe query processing and query planning in PORDaS, and 3) present results from an experimental evaluation of different query planning variants.*

Categories and Subject Descriptors
E.1 [Data structure]; Distributed data structure: **H.2.4** [**Systems**]:Query processing: **H.2.1 [Logical design];** Data models

General Terms
P2P Systems, Query planning, Distributed search

Keywords: Peer-to-peer, distributed databases, query processing, data grid

Received 10 June 2007; Reviewed and accepted 7 Sep. 2007

1. Introduction

Computational science is a rapidly growing multidisciplinary field that has a need for scalable, distributed, and efficient data management. In our research, we see the peer-to-peer (P2P) paradigm a possible solution to some of the problems in distributed data management. P2P has already proved to be suitable and efficient for file sharing, distributed computations, and distributed search.

In our research we are aiming at using P2P to solve some problems in the domain of *distributed databases*, and in particular in the application area of database support for Grid applications. So far, Grid computing as gained some maturity with respect to the actual computation. However, the management of data in Grid networks is still a very immature area. In general, simple files are used. The need for using databases to a larger extent has been identified and there has also been some work on standardized data access services like OGSA-DAI (OGSA-DAI, 2007). In a smaller scale, on Enterprise Grids, transactional database features has been identified as needed but currently not supported (Jimenez-Peris, Patino-Martinez, & Kemme, 2006).

The goal behind our research is to provide database facilities where distribution (and the availability of the Grid backbone) is transparent to the user. It should also provide services for metadata discovery and seamless queries between heterogeneous sources. In this paper we will give an overview of PORDaS, which is the distributed database system layer of the DASCOSA Grid database framework (Nørvåg, 2006). We will also present some details from query processing and planning in the PORDaS prototype.

Many of the architectural decisions of PORDaS are affected by characteristics of the intended application areas: relatively complex and structured data, and the fact that much of the data will be local data that should be made available to the outside world for querying, but for various reasons (including the size of the data volumes) the raw/source data itself should not be distributed. For this reason our system-wide data model is based on the traditional object-relational data model in order to present users/application the same data model as what is used in their applications. The main contributions of this paper are: 1) a presentation of the PORDaS P2P DBMS, 2) query processing in a P2P DBMS, and 3) query planning in a P2P DBMS. Finally, we 4) provide an extensive experimental evaluation of query planning strategies.

The organization of the rest of this paper is as follows. In Section 2 we give an overview of related work. In Section 3 we give an overview of PORDaS, and in Section 4 we give a more detailed description of query processing. In Section 5 we present results from an experimental evaluation of different query planning variants. Finally, in Section 6, we conclude the paper.

2. Related work

Much of the previous work on distributed database systems is obviously relevant. For a survey of state of the art in this area we refer to (Kossmann, 2000). Recent work in this area includes query processor for Internet data sources, for example ObjectGlobe (Braumandl et al., 2001).

Our Grid DBMS PORDaS is based on distributed hash tables (DHT). A number of papers deal with focused issues like query processing in DHT networks (Bauer, Hurley, Pletka, & Waldvogel, 2004; Harren et al., 2002; Renesse, Birman, & Vogels, 2003), and replica management (Maniatis et al., 2003). (Ntarmos, Triantafillou, & Weikum, 2006) describes how to use Distributed Hash Sketches to estimate cardinality of multisets in the context of P2P system based on DHTs.

Three systems, PIER, AmbientDB and PeerDB also aim at providing DBMS support using P2P technology: PIER, AmbientDB, PeerDB, and Atlas.

PIER (Huebsch et al., 2003) hasmany similarities with PORDaS. It is a middleware query engine built on top of a storage manager and DHT. However, it is not designed to

[1]Contact author: Kjetil.Norvag@idi.ntnu.no

support replication and does not maintain system metadata, and essentially only indexes whatever the applications register in the system.*AmbientDB* (Boncz & Treijtel, 2003) is a system designed to provide full relational database functionality for stand-alone operation in autonomous devices that may be mobile and disconnected for long periods of time, while enabling them to cooperate in an ad-hoc way with (many) other AmbientDB devices. A DHT is used both as a means for connection peers in a resilient ways as well as supporting indexing of data.

PeerDB (Ng, Ooi, Tan, & Zhou, 2003) is a P2P system supporting queries against data stored on remote nodes. The system is based on an unstructured P2P system, focusing on data retrieval instead of distributed querying. Instead of relying on global schemas or mediators, information retrieval techniques are used to find matching relations. Both relation matching and queries are performed by agents.

Atlas (Akbarinia, Martins, Pacitti, & Valduriez, 2006) also stores data in the P2P system. Their research focus is on reconciliation, while other aspects are solved in relatively traditional ways.

Other approaches to querying data in DHTs include work by Sattler et al (Sattler, Rösch, Buchmann, & Böhm, 2004), Abdallah et al (Abdallah & Le, 2005), and Akbarinia et al (Akbarinia et al., 2006).

One problem in the context of a Grid DB database system, is the problem of different schemas etc. at different sites. One way of solving this is issue is schema agreement. (Ives, Khandelwal, Kapur, & Cakir, 2005) describe how to man age disagreement among multiple data representations and instances. This is achieved by the use of a novel data model and language for managing conflicting information, and using a DHT to replicate and exchange updates. Bernstein et al. (Bernstein et al., 2002) introduce the Local Relational Model, a data model specifically designed for P2P applications. The main goals of their data model are to allow for inconsistent databases and to support semantic interoperability in the absence of a global schema. We believe exact schema agreement is difficult in our context and instead base our approach on more approximate schema agreement techniques using ontologies.

A general problem in P2P systems is selfish behavior: most peers want to receive more than they contribute. In order to reduce the impact of this behavior, techniques using accounting (Ntarmos & Triantafillou, 2004) and other approacheas to enable trust have been developed.

3. PORDaS

PORDaS is a distributed DBMS using P2P techniques to achieve high performance, scalability, and availability. It is built as a database layer on which larger applications can be built, and provides location-transparent storage of data with Grid applications as main application area. Each node in PORDaS is autonomous. Data is created and stored locally but globally available for querying.

In both Grids and other large distributed systems heterogeneous hardware and operating systems will be the case. In order to ease deployment, we have based PORDaS on components written in Java. In the current version, the P2P communication is performed by the FreePastry DHT (FreePastry, 2007), and for local storage the Derby DBMS (Derby, 2007) is used. It should be mentioned that both the DHT and DBMS are defined as interfaces so that they can easily be replaced with other implementations if desired. They are also both embedded, so that no separate installation of DHT and DBMS is necessary.

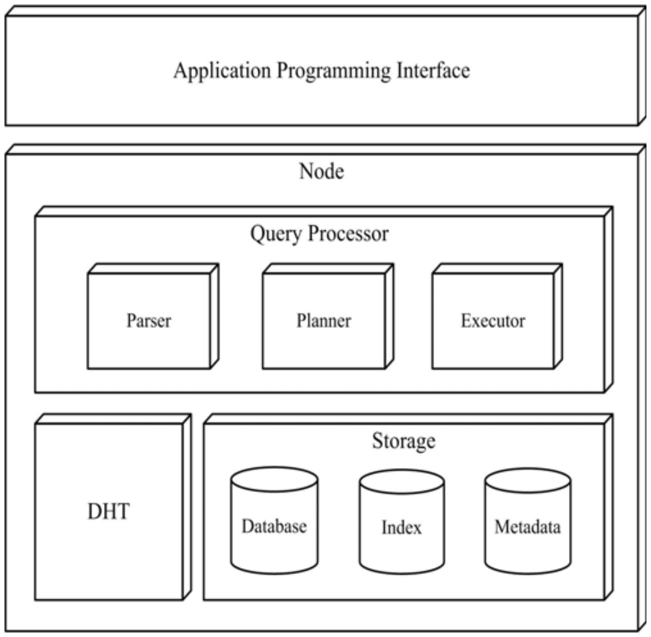

Figure 1. The PORDaS architecture

In a distributed DBMS, metadata management is an important issue. In PORDaS this is solved by separate table descriptions and data discovery tools, so that queries over related data sources using heterogeneous schema descriptions can be performed. This schema management is orthogonal to the work presented on query processing in this paper, and we will not go into further details on this issue in this paper.

In the rest of this section the overall architecture of PORDaS, while storage and query processing will be described in more detail in the subsequent sections.

3.1. Use and data access

PORDaS uses the object-relational data model, i.e., relations of tuples and possibility of tuple identifiers and attributes referencing other tuples.

When a table is created, a hash value based on the table schema is created. This is indexed in the DHT, so that it is possible to find other tables having the same schema signature. It is also possible to annotate a schema by keyword descriptions that can be taxonomy/ontology based. This information is also stored in the DHT. This means that it is possible to find related tables automatically.

Data that is inserted is stored in the local database. A query against local table names will be performed on the local database only. If it is desired that the global database should be queried, i.e., potentially all other local databases in the system, global tables have to be specified in the query. The identifier of global tables are found by using either schema signature or schema description as described above, or provided directly.

Given the identifier of a global table, the DHT can return the identifier of all peers storing elements of this table. It can also indicate value ranges that they store for the key value, which might speed up, e.g., selection queries. Although indexing individual tuples is an alternative using a DHT, this will in general be too fine-grained and having too high maintenance cost.

3.2. Overview of architecture

The architecture of PORDaS is illustrated in Figure 1. An application accesses the databases through the PORDaS API. The application layer can interact both with the local database and query data at other sites transparently, without users having to know where the data is located.

3.3. Storage

The storage component keeps track of all the data stored at a node. It holds the local database and a metadata repository that manages information about local tables. The storage component also stores parts of the distributed index, which all the nodes in the system participate in sharing. The index holds information about the contents and location of other tables in the system.

The local database is used for the persistent data stored in the system, i.e., tables and local metadata. PORDaS operates internally on an object relational data model. This is also the model exposed to users/applications of PORDaS. A query could be performed either against local data only, or against global data. In the case of global data, location is transparent.

3.4. Query processor

The query processor consists of a parser, a planner and an executor. It takes queries as input from the application layer, creates a plan and executes it. The resulting tuples are pipelined back to the application layer.

3.5. Communication

The nodes in a PORDaS network are loosely connected through a DHT. The purpose of the communication module is to enable resource location and to route requests for data. It also enables message sending and reception, and is responsible for unwrapping messages and forwarding them to the appropriate component.

Except when returning query results, every message in PORDaS is routed through the DHT layer. Messages are sent for keyword and query requests, to resolve subqueries and to maintain the distributed index. When returning results, direct connections between the nodes are used in order to improve performance.

The distributed index is realized using a DHT, and among its applications in PORDaS are 1) schema/table discovery, and 2) finding locations of arbitrary tables. The index makes PORDaS location transparent.

Soft state is used to maintain the distributed index. It means that every index record has an associated expiration time. When it expires, the record is deleted from the index. Thus, in order for a node to keep its index records in the system, they must be continually refreshed. After a node leaves the system, either voluntarily or because of a failure, its index records will be removed when they expire.

Figure 2 shows how queries are resolved in PORDaS, with a structure mostly the same as for traditional distributed systems. The query processor can handle multiple queries at a time. We now give a brief overview of each query processing step, followed by a more detailed description of the query planning step.

The query and data manipulation language in the PORDaS prototype is a subset of SQL.

4.1. Query decomposition

In this step queries are decomposed from a textual representation into an algebra tree (see example in Figure 3).

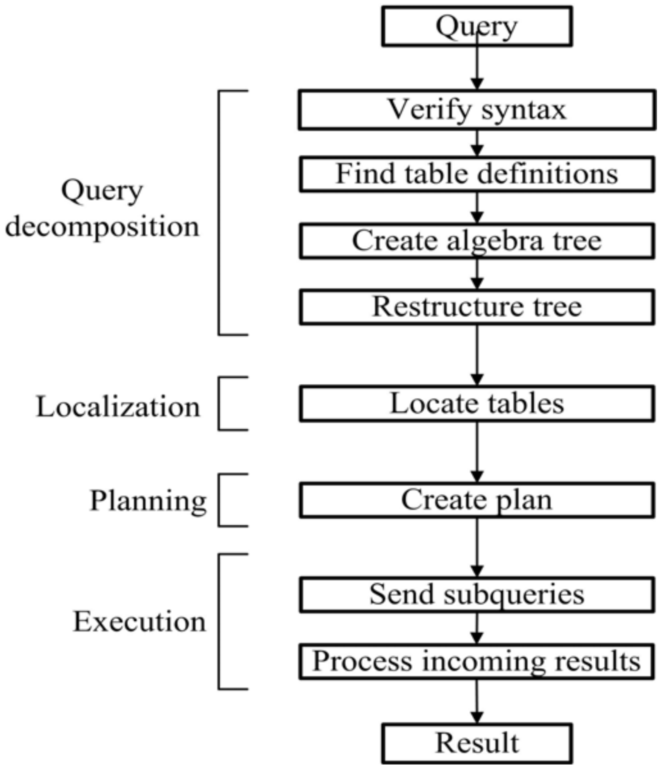

Figure 2. Query processing

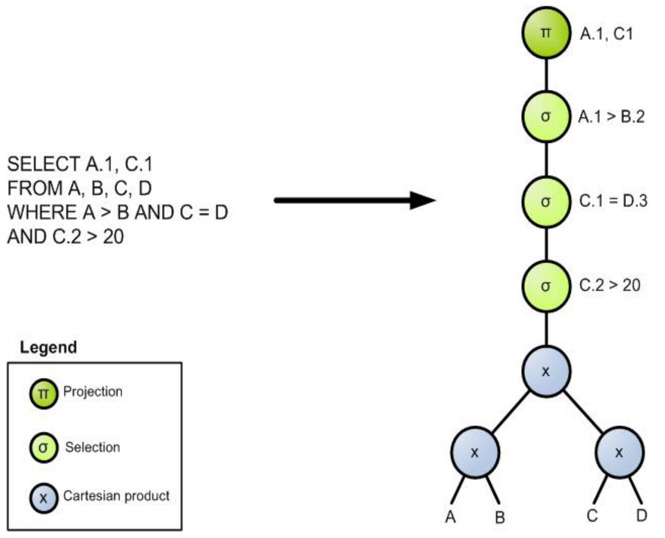

Figure 3. Decomposing an SQL query into an algebra tree

When a new query is submitted, first the query syntax is verified. Then the next task is to verify type correctness, which is making sure the relations and attributes referenced in the query actually exist. The operations in the query are checked against the type of each attribute as well, making sure they match. Before this can be done, the table definition for each table referenced in the query must be fetched. First, the local metastore is searched. If one or more table definitions are lacking, they must be fetched from the distributed index. This will delay the query until all table definitions have been found.

The next step is creating the initial algebra tree. The final tree is either bushy or linear. The first part of the tree is always the same for both types. First, the projections are defined as the root, followed by each selection in the query. PORDaS builds left-deep trees, i.e., the tree is always extended along the leftmost path of each operator, with only base relations as the right child of the operators (see Figure 4). Cartesian products are added as the left child of its parent. If there are joins in the query, these are added as the last part of the tree, in the same fashion as cartesian products.

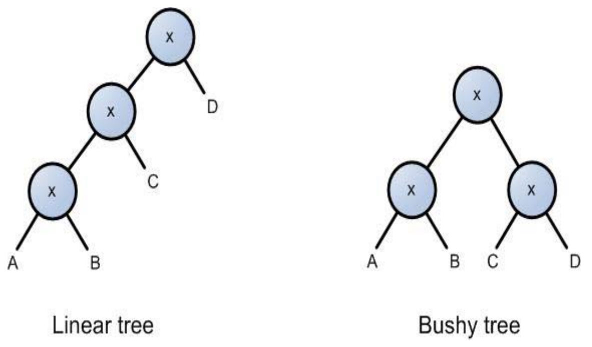

Figure 4. Classes of algebra trees

When building a bushy tree, the goal is to balance the tree as much as possible, catering for parallel execution. This is done by maximizing the number of cartesian products and joins with two operators as children.

Restructuring the tree is performed to achieve a better tree. The projections and selections in the algebra tree are pushed as far down as they can be, and in doing so, the size of intermediate results will be reduced during execution.

4.2. Localization

Localization means finding every site that has a table referenced in the query. These are found by querying the distributed index. Because of the potentially volatile nature of the P2P network, caching is not used to improve the localization process. Nodes that store one of the tables in the query might have left or joined the network since the last time they were queried.

4.3. Planning

When the locations of every table in the query are found, a plan for the execution can be devised. The planner uses heuristics to create plans; these plans are either centralized or distributed.

A centralized plan is a plan where the required base relations are fetched to the initiating site so that the operators in the query can be resolved locally. In the spirit of reducing the amount of network traffic, any selection and projection on base relations are executed at the remote sites before the streaming of results is started. Figure 5 (left) gives an example of a centralized plan. The dotted lines indicate network communication. It is important to note that the data fetched from a base relation in the figure may When the locations of every table in the query are found, a plan for the execution can be devised. The planner uses heuristics to create plans; these plans are either centralized or distributed. include contacting one or more sites. When the locations of every table in the query are found, a plan for the execution can be devised. The planner uses heuristics to create plans; these plans are either centralized or distributed. When the locations of every table in the query are found, a plan for the execution can be devised. The planner uses heuristics to create plans; these plans are either centralized or distributed. When the locations of every table in the query are found, a plan for the execution can be devised. The planner uses heuristics to create plans; these plans are either centralized or distributed.

In a distributed plan, the responsibility for executing the query tree is distributed among the set of nodes that store tables referenced in the query. If statistics about each table was available, like cardinality, maximum and minimum values, this information could be used to restructure the tree in a beneficial way. As this is not the case, predefined rules are used instead. The rule is best explained by an illustration, see Figure 5 (right). It gives an example of a conversion from an algebra tree to a distributed plan. The first rule is to always calculate cartesian products at the initiating node, which avoids sending too much data over the network. It is conceivable that in certain cases it might be better to distribute cartesian products as well, but it is assumed to not be the average case.

The second rule is to delegate the resolution of joins to one of the owners of the leftmost table in the join tree. The choice is arbitrary, it could have been any of the owners. The chosen owner will receive the entire join subtree. If this subtree has any more joins in it, these joins are delegated in the same manner as well. In the figure this can be seen as the second join under the cartesian product is delegated to two separate nodes.

4.4. Execution

The execution phase begins with the initiating node requesting data from all base relations in the query. If the

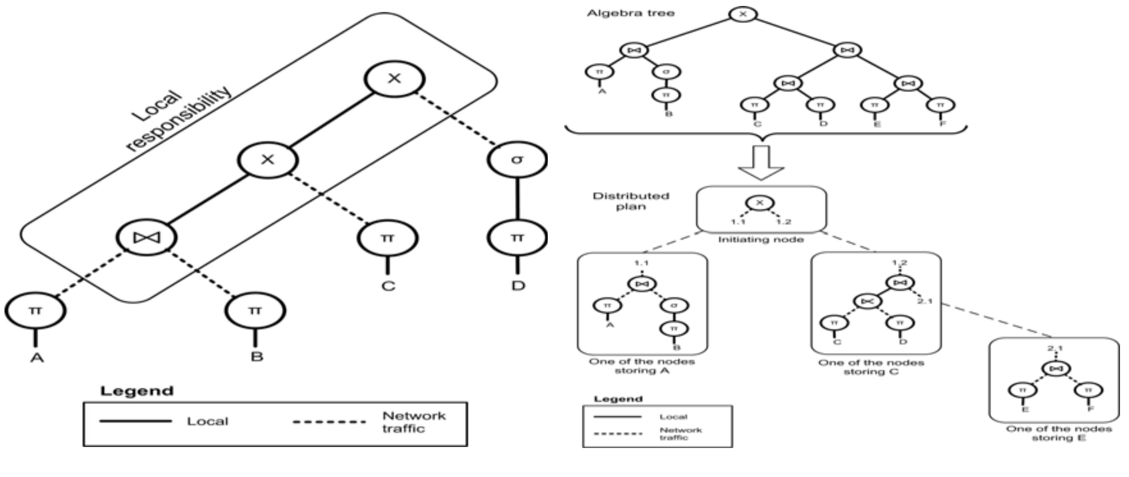

Figure 5. Centralized plan (left) and transformation from an algebra tree to a distributed plan (right)

query plan is a distributed query plan subtrees are delegated to other nodes. The execution is pipelined, which means that processed tuples are sent up the tree as soon as possible. This avoids having to store temporary caches with intermediate results. To know where a tuple belongs at the receiving end, all tuples sent across the network are tagged with an identifier. The identifier uniquely specifies the query and the point in the query tree where the tuple is expected. This can seen in the plan in Figure 5. For instance, the children of the cartesian product are tagged as 1.1 and 1.2. At the sites responsible for those parts of the query, every resulting tuple is tagged with 1.1 or 1.2, respectively.

A query can fail if any of the nodes it relies on goes down. There are several possible strategies for coping with failure. The simplest strategy is chosen for PORDaS, where a failed query times out and returns a failure message.

5. Experimental results

In this section study the different query planning variants. The experiments were conducted on a cluster of 36 computers, each having a 3 GHz Pentium 4 and 1 GB RAM. The experiments were performed by a test application that simulated the activity of a regular PORDaS node. We now give an overview of configurable parameters, measurements, test application, tests, and results.

5.1. Parameters

The test server could create different tests by changing these parameters:

• Number of tables at each node: Having more tables in the local database means the node has to maintain more state in the DHT. This interprets into more traffic due to the maintenance mechanism.

• Number of tuples in a table: The number of tuples in a table restricts the maximum amount of data that can be returned from a query.

• Length of test: In order to make the collected data less susceptible to variances due to the randomness of the query process, the length of the test can be increased. In effect, the sample size is increased so that the standard error is decreased.

• Number of nodes: If there were enough available computers, varying the number of nodes could indicate how PORDaS would scale to a larger network. Since there relatively few computers in the test environment, it has little purpose to alter this value. It will be kept at 36 for all the tests.

• Request interval: The time between two consecutive requests defines how often a new query will be created. It can either be defined as a constant or be randomly distributed by a Poisson distribution.

• Number of concurrent queries: Setting the number of concurrent queries means halting when a certain number of queries are running. When the network is congested, setting this value low can avoid further worsening of traffic.

• Number of active nodes and sharing nodes: An active node periodically requests, directed at sharing nodes' data, with the predefined request interval. A sharing node is a node whose table definitions are distributed among the active nodes.

• Type of Query Processor: Queries are resolved either in a centralized or a distributed manner. Section 4 in the PORDaS chapter gives an explanation.

• Type of query trees: The trees produced in the planner can be either bushy or left deep.

• Result size: The size of the final result can be constrained to a maximum value. A standard deviation can be supplied to get different sized queries. By setting this value, very large and very small results can be avoided.

• Number of tables in a query: The number of tables referenced in a query can be set to get a certain control over the number of nodes involved in a query. A standard deviation can be defined to get variation.

• Number of joins in a query: Setting this number decides the number of joins a query.

The parameters of the test application are summarized in Table 1, and are chosen to simulate a probable usage pattern under medium load. The time between requests is Poisson distributed with a given mean value. The queries will be delayed if there are more concurrent queries than the maximum number of concurrent queries allowed. The actual databases have been kept small in order to be sure PORDaS is tested and not the local database system (i.e., Derby).

5.2. Measurements

The tests measured:

• Response time: The response time of a query is measured as the time from the query is sent until the last tuple is received. The maximum, minimum and average response times are recorded.

• Throughput: Throughput is measured as the number of queries processed per second, and is calculated as the number of responses received divided by the length of the test.

• Number of tuples received: The number of tuples received says something about the amount of data transferred through the network.

• Index size: Knowing the index size for each of the nodes in the system enables the calculation of the distribution of index records.

5.3. Test application

The tests are performed by a test application which simulates the activity of a regular PORDaS node. The test application connects to a centralized server to ease the administration and collection of statistics.

The test application initially connects only to the test server. The server then sends parameters such as the number of tables each node should store and the number of tuples in each table. The test server can order nodes to join the DHT at any time, the first node that joins will create a new ring. When a new simulation starts, each node will receive simulation parameters and an overview of which tables it can include in its queries. The test application will generate queries at a specified rate while the simulation is running. There is no communication between the test server and the nodes during this step. Nodes send their statistics to the test server after the simulation is done. It is possible to start new tests with different parameters without restarting the system. One should note that some parameters, such as database size, cannot be changed from one simulation to another without restarting the entire system.

The communication between the nodes and the test server is done by a single TCP connection per node. This will allow nodes to communicate with the test server even if they are behind a firewall that blocks incoming connections (the connection is an outgoing connection). Last years testenvironment relied on Java RMI and required a connection in each direction to be initialized. This made it impossible to test PORDaS on UNIX clusters.

Parameter	Value	Parameter	Value
Tables per node	8	Tuples per node	2000
# of nodes	36	Simulation length	10 minutes
Request intensity	2 seconds	# of concurrent queries	5
Joins in query	3 (deviation 2)	# of tables in query	4 (deviation 2)
Maximum result size	100000 (deviation 90000)		

Table 1. Parameters for test application

5.4. Experiments

This section presents the contents and purpose of the two tests that were planned.

Many of the parameters in the two tests were common, summarized in Table 1. The reasoning behind the setting of parameters was to simulate a medium load and a probable usage pattern. Databases are kept small since the desire is to test PORDaS and not Derby. The time between requests is Poisson distributed with a given mean value. The queries will wait if there are more concurrent queries than the maximum number of concurrent queries allowed.

Test 1: In the first test, all nodes were sharing and active. The purpose was to compare every permutation of type of algebra tree and type of planner, which means that 4 simulations were be run. The interesting data in this case are the number of started queries versus the number of finished queries, and the response times for each permutation.

Test 2: The purpose of the second test was to study and isolate the effect of executing queries in parallel. This was achieved by having one active, non-sharing node query the rest of the nodes, which were sharing and inactive. The only task of the nonactive sites in the system was resolving the queries the one active node gave them. Two tests were run. The first test had the query processor make linear trees and used a centralized execution strategy. The second test used bushy trees and a distributed execution strategy.

5.5. Results

This section presents and comments the results of the tests.

Test 1

Figure 6 shows a comparison of all permutations of type of query planner and type of algebra tree with respect to the number of started and finished queries. In both cases using centralized plans the loss of queries is minimal, while in the distributed case, the loss is noticeable. With a linear query tree, the loss is over 13%. The circumstances indicate that the reason for the loss is that queries timed out before results were received. The time-out threshold was set to two minutes. By looking at the maximum response times for the queries that did finish, these are close to this limit.

The same figure also shows a distinct difference between the centralized and distributed simulations. The number of started queries is much lower in the distributed case. The reason for not starting more queries is because of the limit on the number of concurrent queries. Waiting for a query to be executed has the effect of lowering the throughput.

Figure 7 and figure 8 show the response times for all permutations of query planner and algebra tree. Times are given for when the first, tenth, hundredth and so on tuple was received. The series in the graphs are divided into categories based on the size of the result.

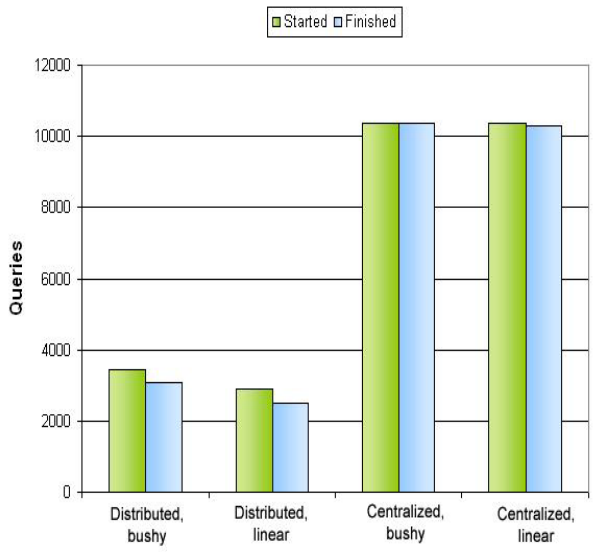

Figure 6. Started versus finished queries

Similar for all the simulations is that for the category of results of size 0-100 tuples, there is a decrease in response time from when the first to the tenth tuple is received. This means there are queries with less than 10 tuples in the result that take a long time to execute. The reason why it takes more time to execute might be that the queries are more complex, containing more joins and selections. The size of the result does not say much about the complexity of the query.

In the distributed, bushy case and in the centralized, bushy case, the category of queries with results in the range of 250-500 tuples have high response time compared to the others. This series is based on scarce data, with only 11 queries. The maximum response time for this series is high, with over 104 seconds. The next range has maximum response times of 11 seconds, so the anomalous range is likely to consist of outliers.

In both the distributed cases, the largest range of results has significantly lower response time than many of the lower ranges. The same goes for the two second highest and third highest ranges as well. It is probable to assume the explanation lies with the way queries are formed by the simulator. Basically, the maximum result size is determined in advance for each query created. To get smaller queries, joins are added to constrain the size. When large queries occur, they have many cartesian products[2]. Cartesian products are always executed locally, which means that smaller parts of the query is delegated to others, which again translates to less network traffic.

[2] The number of tables in a query is determined per query, and if there are few tables available, the only way to get big results is through cartesian products.

Both the centralized simulations have a noticeable artifact. The series with results in the range of 1000-2500 tuples have a terrible response time for the 1000th tuple. It is hard to say why this is so, as the statistics gathered is too coarsely grained to identify the source of error.

When comparing the centralized and the distributed simulations, it is obvious that the centralized plans perform much better. The second test was designed to study this closer.

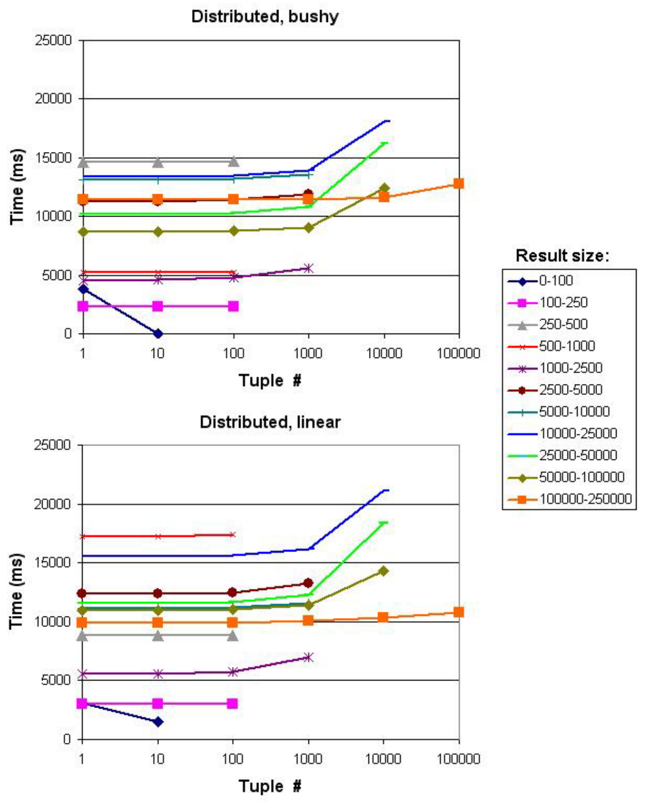

Figure 7. Average response times for distributed query plans

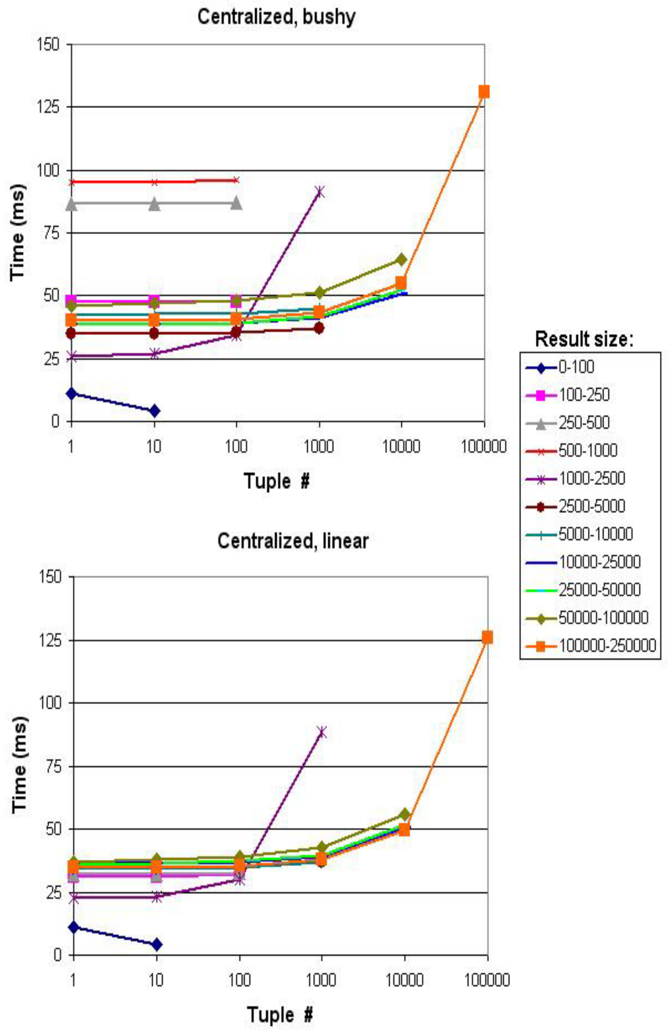

Figure 8. Average response times for centralized query plans

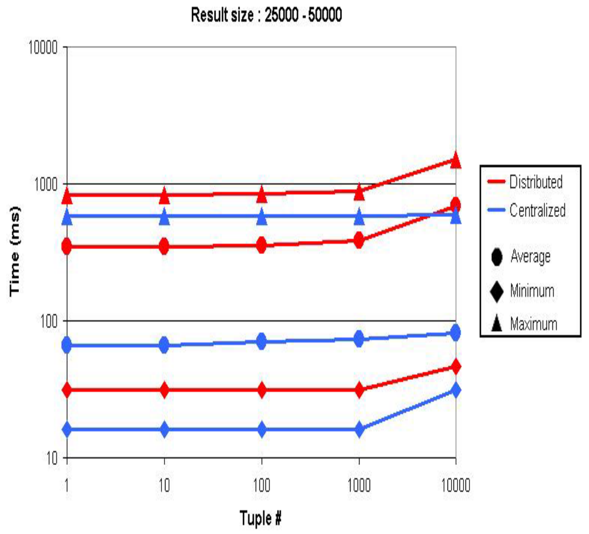

Figure 9. Comparison of distributed and centralized query processing

Test 2

Figure 9 shows a comparison of the minimum, average and maximum response times observed for queries with a result-size between 25.000 and 50.000 tuples. The distributed execution strategy performs consistently worse than the centralized strategy.

Discussion

PORDaS was able to conduct the simulations without node failures. This does not prove the correctness or scalability of PORDaS, but it shows that the system is stable enough to handle moderate work loads between a modest number of peers.

Both tests show that the centralized execution strategy is better than the distributed. The distributed execution strategy has the advantage of executing operators in parallel, but failed since the joins always had a huge selection rate. This caused the distributed execution strategy to generate a lot more network traffic than the centralized strategy. The distributed strategy could have worked better in comparison to the centralized strategy if the joins had lower selection rates.

The tests show the importance of choosing the correct heuristics when processing queries without an optimizer. A better heuristic would probably be to only perform equi-joins in a distributed fashion by using an operator like the pipelining hash join or hash ripple join. The other joins could be centrally performed with the current join algorithm.

The test application also deserves some commenting. Classifying queries by the size of the result does not reveal much about the complexity of the queries. Generating queriesat random did also make it difficult to identify specific problems. It would probably have been better to use a small set of pre-defined queries to get better control over the workload. More extensive logging of events could help identify problem areas, but probably at the expense of the performance.

6. Conclusions and further work

In this paper, we have given an overview of the P2P DBMS PORDaS, and described some aspects of query processing and query planning in this system. We have also presented some results from an experimental evaluation of different query planning variants in PORDaS.

Future work will be in two directions: 1) large-scale experiments and 2) extended functionality. Experiments are planned to be performed both on larger clusters and Grids in Norway, as well as using PlanetLab (http://www.planet-lab.org/). Extended functionality that will be implemented includes replication, enhanced indexing, and a further development of aspects of query optimization in the context of P2P DBMSs.

References

[1] Abdallah, M., Le, H. C. (2005). Scalable range query processing for large-scale distributed database applications. *In: Proceedings of PDCS'2005.*

[2] Akbarinia, R., Martins, V., Pacitti, E., Valduriez, P. (2006). Design and implementation of Atlas P2P architecture. In *Global data management.*

[3] Apache Derby, *http://db.apache.org/derby/.* (2007).

[4] Bauer, D., Hurley, P., Pletka, R., Waldvogel, M. (2004). Bringing efficient advanced queries to distributed hash tables. *In: Proceedings of the 29th annual IEEE international conference on local computer networks (LCN'04).*

[5] Bernstein, P. A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I. (2002). Data management for peer-to-peer computing : A vision. *In: Proceedings of the fifth international workshop on the web and databases (WebDB'2002).*

[6] Boncz, P., Treijtel, C. (2003). AmbientDB: relational query processing in a P2P network. *In: Proceedings of DBISP2P'2003.*

[7] Braumandl, R., Keidl, M., Kemper, A., Kossmann, D., Kreutz, A., Seltzsam, S., et al. (2001). ObjectGlobe: ubiquitous query processing on the Internet. *VLDB Journal*, 10 (1) 48-71.

[8] FreePastry, *http://freepastry.org/.* (2007).

[9] Harren,M., Hellerstein, J.M., Huebsch, R., Loo, B. T., Shenker, S., Stoica, I. (2002). Complex queries in DHT-based peer-to-peer networks. In *Electronic proceedings for the 1st international workshop on peer-to-peer systems (IPTPS'02), available at http://www.cs.rice.edu/conferences/iptps02/.*

[10] Huebsch, R., Hellerstein, J. M., Lanham, N., Loo, B. T., Shenker, S., Stoica, I. (2003). Querying the internet with PIER. *In: Proceedings of VLDB'2003.*

[11] Ives, Z. G., Khandelwal, N., Kapur, A., & Cakir, M. (2005). ORCHESTRA: rapid, collaborative sharing of dynamic data. *In: PRoceedings of CIDR'2005.*

[12] Jimenez-Peris, R., Patino-Martýnez, M., Kemme, B. (2006). Enterprise Grids: challenges ahead. In *Proceedings of int. workshop on high-performance data management in grid environments 2006, http://vecpar.fe.up.pt/2006/programme-hpdg/papers/1.html.*[13] Kossmann, D. (2000). The state of the art in distributed query processing. *ACM Computing Surveys*, *32* (4) 422-469.

[14] Maniatis, P., Roussopoulos,M., Giuli, T., Rosenthal, D. S. H., Baker,M., Muliadi, Y. (2003). Preserving peer replicas by rate-limited sampled voting. *In: Proceedings of the 19th ACM SOSP.*

[15] Ng, W. S., Ooi, B. C., Tan, K.-L., Zhou, A. (2003). PeerDB: A P2P-based system for distributed data sharing. *In: Proceedings of the 19th international conference on data engineering.*

[16] Nørvåg, K. (2006). DASCOSA: database support for computational science applications. *In: Proceedings of Globe'06.*

[17] Ntarmos, N., Triantafillou, P. (2004). SeAl: managing accesses and data in peer-to-peer data sharing networks. *In: Proceedings of HDMS.*

[18] Ntarmos, N., Triantafillou, P., Weikum, G. (2006). Counting at large: Efficient cardinality estimation in internet-scale data networks. *In: Proceedings of the 22nd International conference on data engineering (ICDE'06).*

[19] OGSA-DAI. (2007). *Open grid services architecture data access and integration, http://www.ogsadai.org.uk/.*

[20] Renesse, R. van, Birman, K. P., Vogels,W. (2003). Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, *21*(2), 164-206.

[21] Sattler, K., Rösch, P., Buchmann, E., Böhm, K. (2004). A physical query algebra for DHT-based P2P systems. In *Proceedings of WDAS'2004.*

Author biographies

Kjetil Nørvåg is a Professor in the Department of Computer and Information Science at the Norwegian University of Science and Technology. He received a Dr.Ing. degree in computer science from the Norwegian University of Science and Technology in 2000. He has been a visiting researcher at INRIA in Paris, Athens University of Economics and Business, and Aalborg University. His major research interests include temporal database systems, information retrieval, Grid databases, and P2P data management. He has published more than 60 papers in international refereed conferences and peer reviewed journals.

Eirik Eide received his MSc from the Department of Computer and Information Science at the Norwegian University of Science and Technology in 2006.

Odin Hole Standal received his MSc from the Department of Computer and Information Science at the Norwegian University of Science and Technology in 2006.