Efficient and Robust Database Support for Data-Intensive Applications in Dynamic Environments

Jon Olav Hauglid, Kjetil Nørvåg, and Norvald H. Ryeng

Dept. of Computer Science, Norwegian University of Science and Technology
Trondheim, Norway
E-mail: {joh,noervaag,ryeng}@idi.ntnu.no
URL: http://research.idi.ntnu.no/dascosa/

***Abstract*— Requirements from new types of applications call for new database system solutions. Computational science applications performing distributed computations on Grid networks with requirements for efficient storage and query solutions are now emerging. For this purpose we have developed DASCOSA-DB, a P2P-based distributed database system, which in addition to providing location-transparent storage and querying, also includes novel features like efficient partial restart of queries and redistribution of query operators in the context of failure, dynamic refragmentation and allocation, and distributed semantic caching. In this demo, the novel features will be demonstrated, combined with a more general description of the architecture and demonstration of the distributed query processing capabilities.**

I. Introduction

Requirements from new types of applications call for new database system solutions. Computational science applications performing distributed computations on Grid networks with requirements for efficient storage and query solutions are now emerging.

While Grid *computing* has gained maturity through the recent years, management of data in Grid systems is less mature. Data storage and access is still mostly file oriented, and it is in general left to users to manage files and their locations as needed. Although some support has emerged for metadata management, more advanced database services is still only the proposal stage, examples are the OGSA-DAI and OGSA-DQP frameworks [2], which are service-based, with little cooperation between sites.

The goal of our research is a reliable *Database Grid*, with location-transparent storage, i.e., users/applications do not have to care about where data is stored and where queries are processed. The aim is sites cooperating on data storage and processing while retaining autonomy, i.e., a Grid-wide database system. A sub-goal (but maybe just as important in the long term!) is to help in making computational engineering society believe in databases!

What we provide is a SQL-accessible distributed database system supporting traditional features, but in addition also novel features intended to make the system more suitable for challenging dynamic environments:

- Automatic management of fragment location and replication.
- Efficient handling of query failures through a new partial restart technique.
- Distributed semantic caching.

In the reminder of the paper we will 1) present the background for our project, 2) give an overview of the architecture and implementation of DASCOSA-DB, with emphasis on three of the novel contributions offered by DASCOSA-DB, and 3) outline our three planned demonstrations.

II. Background

DASCOSA-DB can be considered somewhere in between a traditional distributed/federated database system and P2P DBMS. We will in this section describe some issues that forms the background for some of the design decisions in the development of DASCOSA-DB, as well as describing some related work.

Each DASCOSA-DB site has a large degree of local autonomy, but a high degree of cooperation (for example during query execution) is possible. Unlike a typical P2P setting, the participants in a Grid will have knowledge of each other, for example, which universities or companies are participating. Because of this, users can also be expected to be more "well-behaving" compared to a P2P network where most users can be assumed to only gain without giving if given the possibility. Note that although the participating organizations can be known, individual machines as such does not need to be known, i.e., each site will only know a few other sites.

It can be expected that in our application area, large and long-running queries involving many computers will be frequent. Since the probability of failure increases with query duration and number of computers involved, failure of individual sites (or connection to sites) during execution of a query can be frequent, and there is also a certain probability of double-failures, where also the restarted query fails. In some cases there can also be a deadline on data delivery, for example in combination with simulations/observation of physical processes. Thus, complete restart of a query should be avoided. It is also desirable that the handling of query failures should be completely transparent from applications. This contrasts to the traditional case where a query is aborted and has to be restarted.

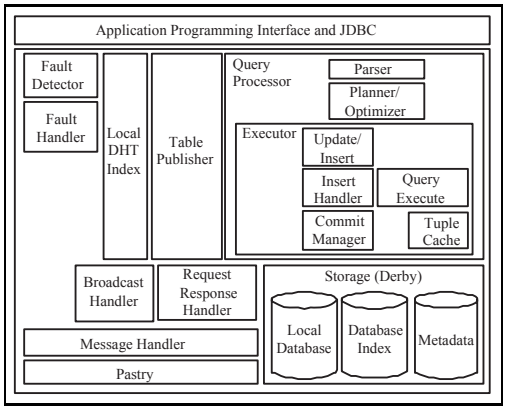

Fig. 1. High-level overview of the architecture of DASCOSA-DB.

A typical access pattern for many computational science applications is to first read an amount of initialization data, then sporadic queries and updates during computation, and finishing with writing a possibly large amount of result data to the database. The different sites will in general be in different phases wrt. access pattern. Global queries, for example from previous results, might be performed during execution. In many cases, strict transaction consistency and isolation is not critical. As a result, there is little need for optimizing on concurrency. However, since there can be large amounts of data created, this data should as much as possible be stored locally. In order to cope with this access pattern which is dynamic both with respect to site and data, dynamic fragmentation and replication is needed.

During the recent years, a large number of research papers on topics related to indexing and querying structured data in P2P network has been published. Of particular interest in the context of our work are a number of other projects aiming at providing *relational* data access through P2P technology like PIER [6], PeerDB [7], Hyperion [10], and APPA [1]. Also other distributed storage systems like, e.g., Bigtable [3] provide distributed storage of data but no query capability. We will in this paper focus on features in DASCOSA-DB not available in other systems.

III. Overview of DASCOSA-DB

In this section we give an overview of 1) the architecture of DASCOSA-DB with particular emphasis on the its novel partial restart techniques, adaptive fragment management and distributed semantic caching, and 2) the implementation of the current version of DASCOSA-DB. For more information about details of DASCOSA-DB we refer to the project's web page.[1]

A. Architecture

The global data model used in DASCOSA-DB is based on the relational model. A table can be stored in its entirety on one site, or it can be horizontally fragmented over a number

[1] http://research.idi.ntnu.no/dascosa/

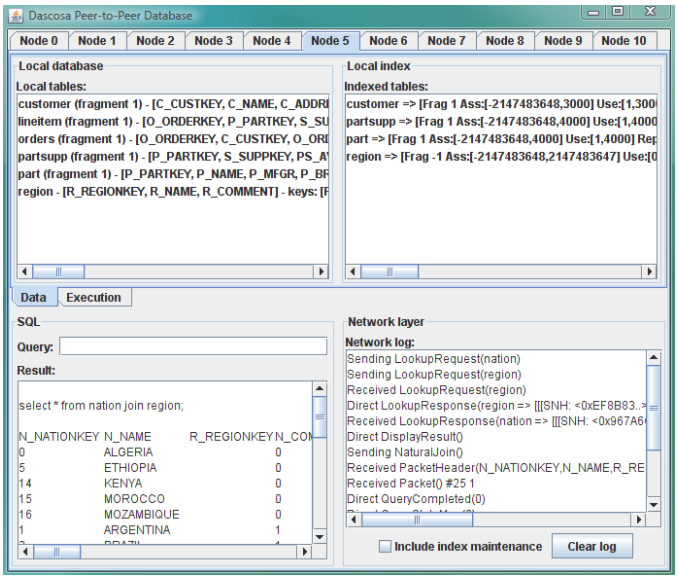

Fig. 2. Screenshot from the DASCOSA-DB system monitoring tool.

of sites. In order to improve both performance as well as availability, fragments can be replicated, i.e., fragments of a table can be stored on more than one site.

In some approaches to P2P databases, every tuple is individually indexed in the P2P system (typically using a DHT *put* operation). In the case of large amounts of data and updates, and with queries more complex than single lookup operations, such solutions are not scalable. In DASCOSA-DB, we instead use the DHT to index data of larger granularity (fragments), i.e., the DHT can be considered as a highly reliable and distributed catalog. The main task of the DHT-part of the system is to provide a mapping between a table name and the sites storing the fragments of the table. Information about local fragments is regularly published to the DHT.

The overall architecture of DASCOSA-DB is illustrated in Fig. 1. As can be seen from the figure, DASCOSA-DB can be viewed as middleware on top of a DHT and local database system.

Query processing is in DASCOSA-DB performed quite similarly to traditional databases, where SQL is transformed into a distributed execution plan (i.e., query operators annotated with the site where they are going to be executed). After planning, query execution begins by transmitting the algebra tree from the initiator site to the different sites involved. Choice of sites is based on location of fragments and total reduction of communication cost.

B. Novel Features

We now give a brief description of 3 novel features of DASCOSA-DB: 1) partial query restart, 2) dynamic fragment management, and 3) distributed semantic caching.

Partial Restart

As mentioned above, the probability of failure during a query increases with the number of sites involved in the query

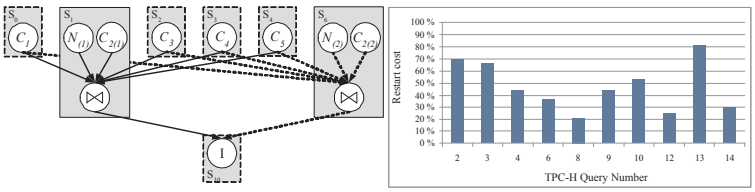

Fig. 3. Query and performance under churn. Replica j of fragment T_i is denoted $T_{i(j)}$.

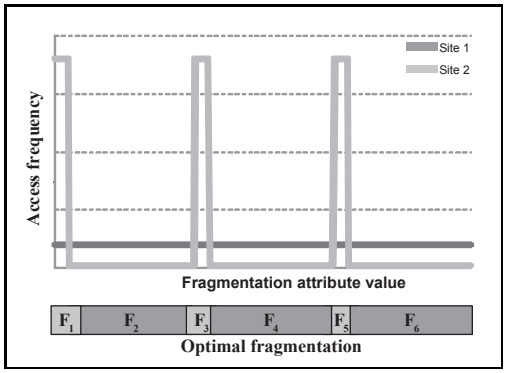

Fig. 4. Example access pattern, and desired fragmentation and allocation.

and with longer duration of queries and/or higher churn rate (unavailable sites). Traditionally, only failure during update transactions has been considered, and failure of queries has been handled by complete query restart. While this is an appropriate technique for small and medium-sized queries, it can be expensive for very large queries, and in some application areas there can also be deadlines on results so that complete restart should be avoided. While in some cases various checkpoint-restart techniques have been employed to avoid complete restart of operations, these techniques have been geared towards update/load operations, and in many cases implies that a query will be delayed until the failed site is online again.

An alternative to local checkpointing and complete restart is a technique supporting *partial restart*. In this case, unfinished subqueries from failed sites can be resumed on new sites after failures, and utilizing partial results already produced before the failure (both results generated at non-failing sites as well as results from failing sites that have already been communicated to non-failing sites). In DASCOSA-DB we have integrated a new technique for partial restart that compared to previous approaches like [11] 1) reduces query execution time compared to complete restart, 2) incurs minimal extra network traffic during recovery from query failure, 3) employs decentralized failure detection, 4) supports non-blocking operators, 5) handles recovery from multi-site failures, and 6) avoids duplicate tuples by deterministic delivery of tuples from base relations and operators..

An example of partial restart is illustrated in Fig. 3. To the left is illustrated 6 sites, each storing fragments of the *customer* and *nation* tables of the TPC-H benchmark (all sites store fragments of *customer*, there is a replica of fragment 2 on both site S_1 and S_5, and the single fragment of *nation* is stored on both site S_1 and S_5). Assume then that the simple query

```
SELECT * FROM nation JOIN customer
```

is issued from site S_{10}, resulting in the query tree in Fig. 3 (site S_1 was selected for the join operator during planning to minimize network traffic as it has a fragment of both involved tables). Then assume that site S_1 fails sometime during the query. The failure is discovered by site S_{10} which select site S_5 as replacement site. The particular challenges that have been solved in our approach relates to failure detection, selection of replacement site, and restart of the various relational algebra operators. In Fig. 3 is illustrated the restart cost for some selected TPC-H queries, compare to complete restart. As can be shown, the cost of restart is considerably reduced using partial restart.

Dynamic Fragment Management

Traditionally, table fragments in distributed database systems have been fragmented and replicated based on fixed value ranges or rules defined by database administrators. In DASCOSA-DB, fragments and replicas are created and migrated automatically by the system based on the access pattern, aiming at keeping the amount of accesses to remote sites as low as possible. The ranges of fragments are not fixed, so that fragments can be split and coalesced automatically. In this way, the system will be able to efficiently adapt to changing workloads.

An example of what we aim at with our approach is illustrated in Fig. 4, where the figure illustrates the access pattern to a database table from two sites. Site 1 has a uniform distribution of accesses, while Site 2 has an access pattern with distinct hot spots. In this case, a good fragmentation would be 6 fragments, one for each of the hot spot areas and one for each of the areas between. A good allocation would be the fragments of the hot spot areas (F_1, F_3, and F_5) allocated to site 2, with the other fragments (F_2, F_4, and F_6) allocated to site 1. Using our approach this access pattern will be detected, and fragmentation and allocation performed accordingly. Note that if the access pattern changes later, this will be detected and fragments reallocated in addition to possible repartitioning.

In DASCOSA-DB, access statistics is used to detect pattern. For each fragment, a set of histograms is maintained, and each histogram represents statistics about accesses from one particular remote site. Each bucket in a histogram represents a value subrange of the fragment, and contains an estimate of the number of accesses to the actual interval the bucket covers. At regular times, the histograms are analyzed, and it is determined through the use of cost functions whether the overall cost would be lower if a fragment were located on another site. It can also be detected if a subinterval of a fragment is heavily accessed from a remote site. In this case, it can be decided that the fragment should be split into 3 fragments, and the fragment containing the interval heavily accessed from the remote site is migrated to the remote site. In order to avoid too

many fragments, fragments which meets in the value interval can be coalesced when they end up at the same site. I.e., a fragment resulted from a split and then migrated might actually be coalesced with another fragment when it is received at the remote site (whether this is performed or not is also based on considerations of potential replicas of the fragments). Contents in histograms are regularly expired, so that they only contain recent statistics and only represent remote sites that have recently accessed the fragment.

It is important to note that our approach for fragment maintenance is different from previous approaches (e.g., [9]) that are based on analyzing SQL queries, while in our approach accesses at tuple level are considered. Also the possibility of managing higher degree of dynamics as well as migration strategies distinguish our techniques from previous approaches.

Distributed Semantic Caching

In semantic caching, results from selected queries as well as their query descriptions are kept. These results can be applied to reuse results from previous queries, thus reducing the total query cost. In large-scale distributed database systems, using a central central server with complete knowledge of the system will be a serious bottleneck and single point of failure. In DASCOSA-DB this problem is reduced by distributing the caching knowledge over several sites. In addition, decisions about what to cache and cache replacement is performed automatic and site-autonomous.

C. Implementation

The implementation of DASCOSA-DB has been performed in two steps: first, a simple proof-of-concept prototype was developed [8]. The current version is a completely re-implemented version, and acts as a middleware on top of Apache Derby [4] running as the local DBMS on each site. The catalog service for indexing tables is implemented using the FreePastry DHT [5]. DASCOSA-DB is 100% Java, which gives easy deployment, platform independence, and can also be embedded in other software if desired.

In order to facilitate easy interactive access to the system as well as study configuration, distribution of data, and query execution, we have implemented a monitoring tool, cf. the screenshot in Fig. 2. As can be seen in the figure, queries can be entered and information about local tables as well as information about remote tables (which is part of the responsibility of the site as participant in the DHT) can be found. Both static statistics as well as per-query statistics can easily be viewed.

IV. Demonstration

Our demonstration will illustrate both the general distributed database aspects of DASCOSA-DB, as well as more novel aspects like partial restart. In the demonstration we will for convenience run a number of DASCOSA-DB-instances on one or two machines. The demonstration will use a dataset generated by the TPC-H data generator [12].

A. Overall and Distributed Queries

In this demonstration we will give an overview of DASCOSA and its basic features. We will show how both simple and complex SQL queries are executed in the system, and demonstrate how fragmentation and replication aspects are automatically handled by the system.

B. Partial Restart

In this demonstration we will demonstrate how DASCOSA-DB work in the context of failing sites during query execution. The effect of partial restart will be demonstrated by queries not failing, queries failing and not employing partial restart (i.e., complete restart), and employing our partial restart approach. We will in this context also show how the system automatically selects new sites that will complete the work of failed sites, and how replication can make restart possible even when sites storing base tables fail during a query.

C. Distributed Semantic Caching

In this demonstration we will show how DASCOSA-DB utilizes cached subqueries in order to improve performance in the context of repeated queries or queries that contains subqueries of previous queries.

V. Future Work

Although we now have a working prototype of a distributed database system, there is no lack of remaining challenges. Improving query optimization in the context of adaptive fragments and replication is an obvious goal. Another important issue in our context is more automatic handling schema heterogeneity, where the plan is to employ ontology-based methods. We also intend to study how equivalents of partial restart and adaptive fragmentation can be employed in the context of XML data.

Acknowledgments: The authors would like to thank other previous and current participants in the DASCOSA project: Eirik Eide, Odin H. Standal, and João Rocha. Supported by grant #176894/V30 from the Norwegian Research Council.

References

[1] R. Akbarinia, V. Martins, E. Pacitti, and P. Valduriez. Design and implementation of Atlas P2P architecture. In *Global Data Management*, 2006.
[2] M. N. Alpdemir et al. OGSA-DQP: a service for distributed querying on the Grid. In *Proceedings of EDBT'2004*, 2004.
[3] F. Chang et al. Bigtable: A distributed storage system for structured data. In *Proceedings of OSDI'2006*, 2006.
[4] Apache Derby, `http://db.apache.org/derby/`.
[5] FreePastry, `http://freepastry.org/`.
[6] R. Huebsch et al. Querying the internet with PIER. In *Proceedings of VLDB'2003*, 2003.
[7] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. PeerDB: A P2P-based system for distributed data sharing. In *Proceedings of ICDE'2003*, 2003.
[8] K. Nørvåg. DASCOSA: database support for computational science applications. In *Proceedings of GLOBE'06*, 2006.
[9] J. Rao, C. Zhang, N. Megiddo, and G. M. Lohman. Automating physical database design in a parallel database. In *Proceedings of SIGMOD'2002*, 2002.
[10] P. Rodríguez-Gianolli et al. Data sharing in the Hyperion peer database system. In *Proceedings of VLDB'2005*, 2005.
[11] J. Smith and P. Watson. Fault-tolerance in distributed query processing. In *Proceedings of IDEAS'2005*, 2005.
[12] TPC-H, `http://www.tpc.org/tpch/`.