AmbientDB: Complex Query Processing for P2P Networks

Caspar Treijtel

CWI
Kruislaan 413
1098 SJ Amsterdam
The Netherlands
C.Treijtel@cwi.nl

Abstract

This paper describes the project AmbientDB done at CWI which is the focus of my PhD track. The goal of the AmbientDB project is to provide a data management platform for multiple devices interconnecting with each other in ad-hoc ways. The platform is intended to be a crucial building block to *Ambient Intelligent (AmI)* applications. In this paper I will give an overview of AmbientDB project and present my PhD work associated with it in terms of what progress has been made so far and directions for further research.

1 Introduction

For several years now continuing advances in hardware have made handheld devices more powerful and attractive. They are likely to become ever more popular and will integrate more in our daily lives. The forthcoming of these devices leads to interesting opportunities for creating user-friendly *ambient intelligent* (AmI) applications that make the networked devices cooperate in a proactive manner. The AmI vision is being promoted by the IST Advisory Group (ISTAG) and aims to achieve proactive functionality in all kinds of electronic devices with intuitive human-computer interfaces [2].

One of the challenges for the development of AmI applications lies in the field of data sharing among devices. AmI applications, having proactive and intelligent functionality, should make use of available (meta)data and react to data changes in any ad-hoc connected device. Therefore we think that the AmI application will benefit from a transparent data storage and recall platform, where every device in the ambient environment may provide data and request data from other devices.

The development of an ambient DBMS is an ambitious goal and several problem domains can be identified that need investigation and development of novel database techniques. During my PhD track I will concentrate on the problem of providing complex querying facilities in heterogeneous P2P (peer-to-peer) environments. We believe the P2P paradigm to be suited to the ad-hoc behavior of connected devices. We assume the network to be heterogeneous in the sense that various classes of devices cooperate in the environment, from small resource poor devices to powerful, high bandwidth workstations.

This paper is organized as follows. Section 2 focuses on the problem of complex query facilities in a P2P network, where an architecture for a relational query processor is presented. Following, Section 3 describes work that we plan to do in the near future and contains some ideas on future research directions. Finally in Section 4 some concluding remarks are presented.

2 Complex query processing in P2P networks

The P2P paradigm has been a hot topic ever since the introduction of Napster [7] and Gnutella [3]. One of the subjects that has been studied for quite some time is item lookup queries in a P2P network, where systems based on Distributed Hash Tables (DHT) provide very scalable solutions for locating data objects based on keywords [9, 10]. So far little effort has been put in providing *complex* querying facilities over structured data (tabular, XML) in a P2P network. Harren et al. have investigated complex query facilities on top of a structured, DHT-based P2P system [5]. We focus on designing a relational query processor that functions in a P2P network that is not restricted to hashing data based on keywords [1].

The AmbientDB architecture for complex distributed query processing in an ad-hoc P2P network is shown in Figure 1. One of the modules of the architecture is a P2P networking layer. This layer should continuously provide an efficient *overlay* network topol-

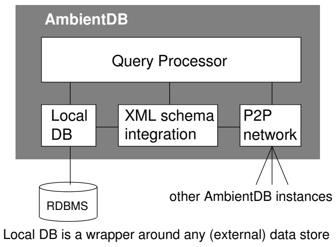
AmbientDB
Query Processor
Local DB
XML schema integration
P2P network
RDBMS
other AmbientDB instances
Local DB is a wrapper around any (external) data store

Figure 1: AmbientDB Architecture

data model, expressions
`Column(String name; int type)`
`Key(bool unique; List<Column> columns; Table table)`
`Table(String nme;List<Column> cols;List<Key> prim,forgn)`
`Expr(int type)`
`Expr::ConstExpr(String printedValue)`
`Expr::ColumnExpr(String columnName)`
`Expr::OperatorExpr(String opName, List<Expr>)`

abstract global algebra
`Select(Table t; Expr cond; List<Expr> result)`→`Table`
`Aggr(Table t; List<Expr> groupby, result)`→`Table`
`Order(Table t; List<Expr> orderby, result)`→`Table`
`Join(Table left,right;Expr cond;List<Expr> result)`→`Table`
`Project(Table t; List<Expr> key,result)`→`Table`
`Union(`T_1` t; List<Expr> key, result)`→`LT` # *create LT from DT/PT by distributed union*
`Partition(DT t; List<Expr> key)`→`PT` # *create PT from DT by finding duplicates*

concrete global algebra (T_1, T_2 ∈ {DT, PT})
$\mathtt{select}_{local}$(LT)→LT
$\mathtt{select}_{dist}$(T_1)→T_1
$\mathtt{aggr}_{local}$(LT)→LT
$\mathtt{aggr}_{dist}$(PT)→DT
$\mathtt{order}_{local}$(LT)→LT
$\mathtt{order}_{dist}$(T_1)→T_1
$\mathtt{join}_{local}$(LT,LT)→LT
$\mathtt{join}_{broadcast}$(LT,$T_1$)→$T_1$
$\mathtt{join}_{foreignkey}$($T_1$,X)→$T_1$
$\mathtt{join}_{dist}$(DT_1,DT_2)→LT
$\mathtt{project}_{local}$(LT)→LT
$\mathtt{project}_{dist}$(T_1)→T_1
$\mathtt{union}_{merge}$(T_1)→LT
$\mathtt{union}_{elim}$(T_1)→LT
partition(DT)→PT

Table 1: The Global AmbientDB Query Processing Model

ogy, with nodes joining and leaving the network at will. The *XML schema integration component* is embedded in the architecture to support both schema evolution and schema heterogeneity among nodes.

2.1 A relational query processor for AmbientDB

For the moment we make the following assumptions. First, we assume that during the execution of a single query, the network of participating nodes is stable. At the start of the execution of the query, a spanning tree is constructed where the query node is the root of the tree. Second, we assume the existence of a globally known database schema. We think that this assumption may be defended for application specific database schema needs. Third, the union of tables stored on the nodes in the network may contain tuple replicas, but they are considered to be consistent (thus we assume some data synchronization scheme to be present). These assumptions are meant to focus our work on distributed query processing in a heterogeneous P2P network. Future work, as well as parallel research tracks in AmbientDB will address some of the research issues that arise when loosening the assumptions (see also Section 3).

The model makes a distinction between three types of tables, which are *Local Table* (`LT`), *Distributed Table* (`DT`) and *Partitioned Table* (`PT`). We define a `DT` to be a table that is stored at multiple nodes, but not necessarily fully replicated. From the `DT` we can derive a partitioning scheme were all replicas are filtered out, leaving a set of disjunct partitions of the table. This we define as a `PT`, which has the advantage over a `DT` that some queries can be implemented more efficiently by broadcasting the query and letting each node calculate their local results. AmbientDB supports the on-line definition and computation of `PT`s, because on a set of ad-hoc connected nodes it may be uncertain beforehand which tuples are replicated where. See Figure 2 for a schematic view of the three types of tables.

In the AmbientDB platform, nodes may pose queries expressed in a standard relational algebra (see Table 1). In our definitions, we denote lists types (`List<Type>`), list instances (`<a,b,c>`) and list concatenation ($L_1|L_2$). In the table, the "abstract global algebra" contains the standard operators familiar in the relational model. Additionally it contains the operators `Union` and `Partition`, which take the union of sets of tuples and create a partitioning scheme, respectively. The translation of the global algebra queries is schematically shown in Figure 3 and will be described next.

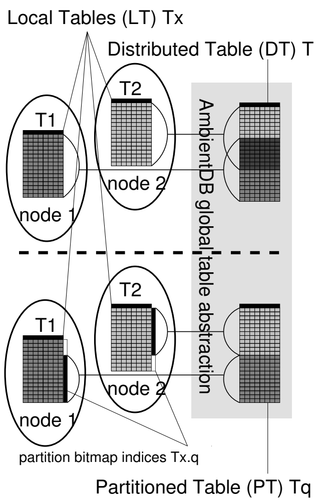
Local Tables (LT) Tx
Distributed Table (DT) T
T1
T2
node 1
node 2
AmbientDB global table abstraction
T1
T2
node 1
node 2
partition bitmap indices Tx.q
Partitioned Table (PT) Tq

Figure 2: LT, DT and PT

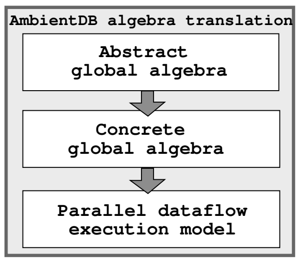
AmbientDB algebra translation
Abstract global algebra
Concrete global algebra
Parallel dataflow execution model

Figure 3: Translation of Algebra to Execution Model

	Dataflow Algebra Operators
n	**scan**(Buf b)$\rightarrow$Dataflow
s	**select**(Dataflow d; Expr cond; List<Expr> result)$\rightarrow$Dataflow
a	**aggr**(Dataflow d; List<Expr> groupBy,aggr)$\rightarrow$Dataflow
o	**order**(Buffer b; List<Expr> orderby,result)$\rightarrow$Buffer
j	**join**(Dataflow d$_l$, d$_r$; List<Expr> key$_l$,key$_r$,result) # merge-join on dataflows ordered on key
p	**project**(Dataflow d; List<Expr> key,result)$\rightarrow$Dataflow # outputs subset of input tuples such that all keys occur once
m	**merge**(Dataflow d$_l$,d$_r$; List<Expr> key)$\rightarrow$Dataflow # merges key-ordered dataflows, of ordered tuples # adds $t.\#nr$ sequence in merge chunk, and $t.\#cnt$ chunk size
t	**split**(Dataflow d;List<Buf$\times\!\!<b_1..b_n>$;List<Expr$\times\!\!<f_0..f_n>$)$\rightarrow$Dataflow # $\big(f_0(t) = true \rightarrow$ propagate t into pipeline$\big) \wedge \big(\forall i : 1 \leq i \leq n : f_i(t) = true \rightarrow$ copy t into $b_i\big)$

Table 2: Local query algebra operators

2.2 Concrete global algebra

The right side of Table 1 shows the "concrete global algebra," which consists of the abstract global algebra operators that are defined on specific types of tables (instantiated into `LT`, `DT` or `PT`). Additionally, the concrete operators are annotated with the keyword *local* or *dist*. A *local* operator is only performed at the query node whereas for a *dist* operator the query is propagated throughout the network.

Not all combinations of instantiations into `LT`, `DT` and `PT` are defined for the operators. The set of concrete operators supported provide a rich set of query functionality, where operators can be viewed in isolation or in the context of a more complex query plan.

2.3 Parallel dataflow execution

The implementation of the concrete global operators is based on each node's local algebra dataflow operators as listed in Table 2. The local algebra uses a dataflow execution model [11], with streams of tuples processed "in-network" throughout the spanning tree, analogous to the aggregation of tuples in a sensor network [6]. Our model is based on the assumption that all streams of tuples are ordered. This makes an efficient implementation of the algorithms possible, minimizing space requirements.

Each concrete operator is implemented by one or more *waves* working in parallel, in which streams of tuples flow from the query node to descendant nodes or vice versa. A *wave-plan* consists of a graph of dataflow query operators depicted in a graphical formalism. Each wave-plan is executed by a separate thread and can be cleanly encapsulated using the Volcano iterator model [4]. That is, each dataflow operator has multiple input dataflows, but only one output dataflow such that they can be pipelined in recursive function calls. Other outputs are written to buffers (queues). Such buffers may be local tables (`LT`), network connection buffers or be used for inter-wave communication.

In Figure 2.3 some of the concrete global algebra operators are depicted in the wave formalism. Each operator in the figure is executed in a single wave-plan. A distributed query originates a query flood which is indicated by the hexagon. An interesting characteristic of the operators is that they can be interconnected to result in more complex operators.

Two important operators from the dataflow algebra are the `merge` and the `split` operators. The `merge` operator merges two dataflows into one, augmenting each tuple with extra information that can be used by other operators like the `select` and the `split` operator. Specifically, for each tuple an extra attribute is added specifying the number of replicas (`#cnt`) and the sequence number (`#seqnr`) for this tuple (thus ranging from 1 to `#cnt`). The `split` operator copies tuples that satisfy specified rules to specific buffers which may be read by other operators.

3 Future work

The query processor described in Section 2 is based on algorithms that provide correct answers under the assumptions taken and thus can be seen as a starting point for performing query processing in an ad-hoc manner. A lot of improvements, additions and fine-tuning will be necessary in order to face the complexity of the real world and thus provide a lot of interesting research questions. In the next sections I will discuss a research plan that should bring us somewhat closer to the goals of the AmbientDB project. On a short term, building a prototype for AmbientDB and defining experiments are part of the research agenda. Some thoughts on research directions may be useful for a more long term planning.

3.1 Prototyping AmbientDB

Currently we are working on a first prototype implementation in Java that implements all operators in the algebras. The goal of this prototype is twofold; first of all we want the prototype to serve as a starting point for coming to a reference implementation of the P2P query processor. Second, we want the prototype to serve as a tool for running simulations that actually make use of our defined algorithms. Figure 5 contains a schematic overview of the prototype, where the AmbientDB query processor is able to process its own data from the local data store, connect to other

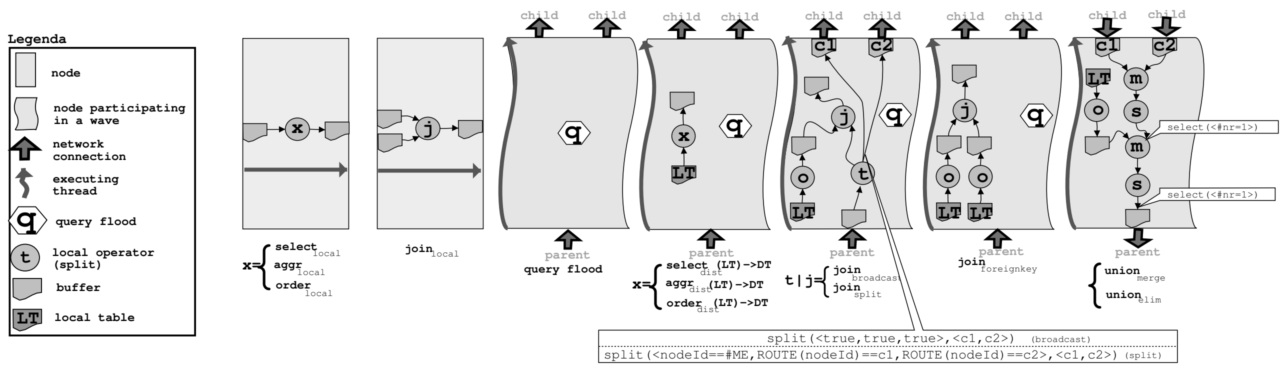
Legenda
node
node participating in a wave
network connection
executing thread
q
query flood
t
local operator (split)
buffer
LT
local table
x
x= select local / aggr local / order local
j
join local
child
child
q
parent
query flood
x
LT
x= select dist (LT)->DT / aggr dist (LT)->DT / order dist (LT)->DT
c1
c2
j
o
t
LT
t|j= join broadcast / join split
o
o
LT
LT
join foreignkey
LT
m
s
select(<#nr=1>)
union merge / union elim
split(<true,true,true>,<c1,c2>) (broadcast)
split(<nodeId==#ME,ROUTE(nodeId)==c1,ROUTE(nodeId)==c2>,<c1,c2>) (split)

Figure 4: Wave-plans for some of the concrete global algebra operators

AmbientDB peers (assuming a network layer is available) and connect to a network simulator for running simulations on different topologies, simulating network congestions, etc.

Opportunities for optimizing the algorithms in terms of network bandwidth consumption are certainly available. First, some initial optimizations can be realized by gathering statistics on tuple locations for determining the set of tuples that participate in the query. Using these statistics, algorithms may use heuristics based on gathered information to choose which tuples participate in the query. This can be done on a per-query basis. Second, the algorithms as they are defined currently do not use any knowledge on data locations in the network. Obviously a lot of network bandwidth may be spared when some knowledge on data locations is available. Third, the streams of tuples may be compressed, significantly reducing network bandwidth. Fourth, when we find that a nearly complete answer to our query suffices, we can define a time-to-live (TTL) counter of to be visited hops to reduce bandwidth consumption. This will be based on the assumption that for a given network with uniformly distributed tuples, we can estimate the number of nodes to visit to account for a minimum percentage (e.g. 95%) of available results. On a short term we want to run simulations to gain more insights in this problem domain. Currently we are investigating the use of the ns-2 simulator [8] to embed it in our simulations.

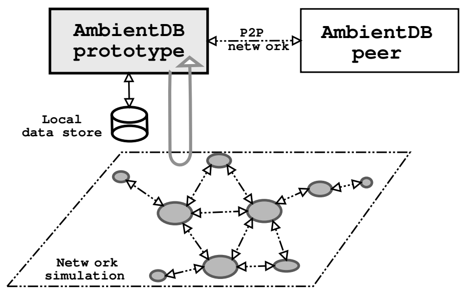

Figure 5: AmbientDB prototype

3.2 Experiments

Due to the complexity of the ambient environment we want to workout a number of specific user environment scenarios. Not only will this be valuable for keeping in touch with real world applications but also create opportunities for focusing on subproblems in a wealth of tunable variables. Already the AmbientDB project is involved in a cooperation with the home mobile consumer electronics division of Philips Research, to gain more insight in the requirements for the AmbientDB platform in light of real world applications.

The scenarios also allow us to define costmodels and benchmarks that can be used to experiment with all kinds of possible strategies for the query processor. Examples of application scenarios may be a shared agenda application running on a set of PDAs, or a multimedia home environment setting where multimedia data (images, sound) are stored in the user's home.

3.3 Future research directions

In our model we have assumed the network to be stable during the execution of a single query. This assumption may not hold in a real world scenario and therefore the query processor and the P2P network layer have to work closely together to cope with the dynamics of the network. The network layer should try to provide a network topology that suits the needs of queries being executed. For example, the construction of the (overlay) spanning tree for execution of a query may be done in multiple ways, where not all possibilities will be equally efficient.

We acknowledge the fact that in the AmI application domain users make their own choices regarding data placement. Despite of this the AmbientDB platform will benefit from proactive data replication and caching of query results at different levels of the architecture. The network layer may proactively cache efficient routes through the network with a certain expiration time. A dynamic load balancing scheme is needed at the data level. The query processor may cache query results for query intensive nodes, and data may be replicated for improving data availability.

4 Conclusions

In this paper I have described the AmbientDB project which is the subject of my PhD track. We have designed a framework for a relational query processor that is able to operate in a P2P network of nodes. The framework may serve as a starting point for improving and fine-tuning the implementation of database operators. I have outlined a research agenda to get closer to the realization of the goals of AmbientDB, being the development of a prototype and defining experiments in terms of real world scenarios. For future research there are many tunable parameters may lead to some interesting research problems.

References

[1] P. A. Boncz and C. Treijtel. AmbientDB: relational query processing in a P2P network. Technical Report INS-R0306, CWI, Amsterdam, The Netherlands, June 2003.

[2] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.C. Burgelman. *Scenarios for Ambient Intelligence in 2010.* European Commission, Brussels, 2001.

[3] Gnutella, http://www.gnutella.com/, July 2003.

[4] G. Graefe. Encapsulation of Parallelism in the Volcano Query Processing System. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 102–111, Atlantic City, New Jersey, United States, 1990. ACM Press.

[5] M. Harren, J.M. Hellerstein, R. Huebsch, B.T. Loo, S. Shenker, and I. Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. In *First International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, USA, March 2002.

[6] S. Madden, M.J. Franklin, J.M. Hellerstein, and W.Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, December 2002.

[7] Napster, http://opennap.sourceforge.net/, July 2003.

[8] The Network Simulator – ns-2, http://www.isi.edu/nsnam/ns/, July 2003.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[10] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[11] A.N. Wilschut and P.M.G. Apers. Dataflow Query Execution in a Parallel Main-Memory Environment. *Distributed and Parallel Databases*, 1(1):103–128, 1993.