Robust Aggregation in Peer-to-Peer Database Systems

Norvald H. Ryeng and Kjetil Nørvåg
Norwegian University of Science and Technology
Department of Computer and Information Science
Sem Sælands v. 7–9, 7491 Trondheim, Norway
{ryeng,noervaag}@idi.ntnu.no

ABSTRACT

Peer-to-peer database systems (P2PDBs) aim at providing database services with node autonomy, high availability and loose coupling between participating nodes by building the DBMS on top of a peer-to-peer network. A key feature of current peer-to-peer systems is resilience to churn in the overlay network layer. A major challenge in P2PDBs is to provide similar robustness in the data and query processing layer. In this paper we in particular describe how aggregation queries in P2PDBs can be handled in order to reduce the impact of churn on accuracy of results. We perform a formal study of data loss and accuracy of such queries, and describe new approaches that increase the accuracy of aggregation queries in P2PDBs under churn.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

Algorithms, Reliability, Performance

Keywords

Aggregation, distributed querying, peer-to-peer systems

1. INTRODUCTION

A key feature of current peer-to-peer routing mechanisms is resilience to *churn*, the effect of nodes constantly joining and parting from the network. Nodes leaving the network, either because of a planned shutdown or because of a node or network failure, will generally not interfere with the message passing capability of the network; network traffic is routed through other nodes until a disconnected node reconnects.

A major challenge in peer-to-peer database systems (P2PDB) is to provide similar robustness in the data and query processing layer. When a node is disconnected, the data stored at that node is also inaccessible. In some cases it may be possible for nodes to hand over their data before disconnecting, but in case of node and network failures, data may become inaccessible without warning. The failure rate of a large distributed system is such that the system cannot expect all nodes to be accessible at all times, so waiting for disconnected nodes to reconnect is not generally an option. Instead, when nodes fail, query processing has to be based on partial data.

The typical method for doing aggregation in P2PDBs is to use a reduction tree, as illustrated in Fig. 1. In this way, the nodes at the leaves of the tree start aggregating over their local database, and each leaf node sends its partial aggregates to its parent node. An intermediate-level node gathers partial results from its children and merges these results with the result from the local database. The result of the merge is sent upwards in the tree to the parent node. This passing of partial aggregates continues all the way to the root node, which merges and evaluates the partial aggregates to get the final result of the query.

The problem with this approach is that failure of internal nodes causes loss of data from all nodes below it in the hierarchy. Existing work propose to use replication of the aggregation process to counter this effect [5, 7]. In this paper we study in more detail the data loss in aggregation and show that costly replication is not necessarily the best way to improve the accuracy of results.

The analysis and techniques presented in this paper are applicable both for P2P systems based on distributed hash tables (e.g., Chord [15], CAN [10], Pastry [12]), as well as unstructured P2P-systems where tree-overlays can be created through flooding (e.g., Gnutella-like networks).

The main contributions of this paper are 1) a formal study of data loss in P2PDB aggregation queries, 2) new approaches that reduces the impact of churn on aggregation accuracy, and 3) an experimental study of the effect of various parameters in techniques used to reduce the impact of churn on aggregation accuracy.

The organization of the rest of the paper is as follows. In Section 2 we give an overview of related work. In Section 3 we perform a formal study of data loss and accuracy of query results in P2PDBs. In Section 4 we discuss techniques for reducing data loss. In Section 5 we present experimental results. Finally, in Section 6, we conclude the paper and

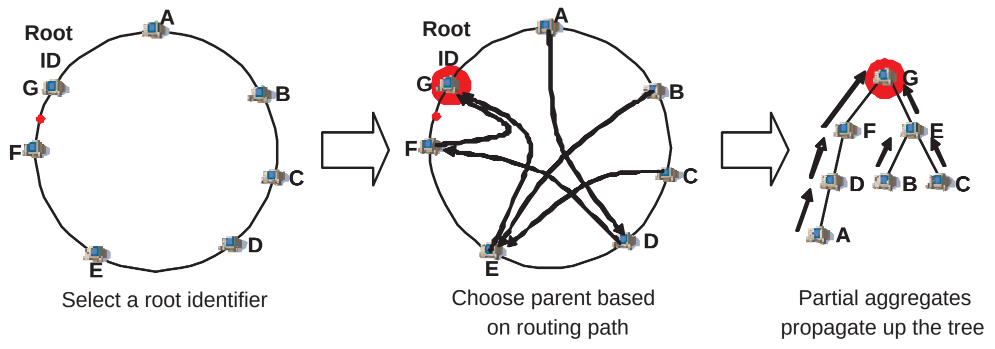
Root ID
A
G
B
F
C
E
D
Select a root identifier
Root ID
A
G
B
F
C
E
D
Choose parent based on routing path
G
F
E
D
B
C
A
Partial aggregates propagate up the tree

Figure 1: Hierarchical aggregation using a DHT.

outline issues for further work.

2. RELATED WORK

Although very popular for file-sharing applications and distributed computing, only a few P2PDBs and P2P data management systems have been realized so far. The most well-know systems include PIER, Piazza, APPA, and PeerDB:

- The Peer-to-Peer Information Exchange and Retrieval system (PIER) [5, 6] is a general-purpose query processor that executes relational queries in a DHT-based network. PIER does not implement persistent storage and relies on external data producers to insert and renew data.
- The Piazza system [4] is a peer data management system that mediates between heterogeneous schemas. Instead of requiring all nodes to share a common schema (as, e.g., PIER does), each node is allowed full autonomy in which data it wants to store and which schema to use.
- The APPA system [16], on the other hand, assumes that participating nodes will agree to a common schema description.
- PeerDB [9] also supports schema matching using agent technology to find relevant relations on other nodes.

Distributed aggregation queries are also performed in other variants of distributed computing systems and sensor network systems, for example:

- In Astrolabe [11], data is placed in a hierarchy of *zones*. Each zone stores and maintains aggregated data from its sub-zones, and at the lowest level, from *virtual leaf zones* constructed around single nodes.
- The Scalable Distributed Information Management System (SDIMS) [17] is an aggregation system similar to Astrolabe, but based on a DHT.
- The Tiny Aggregation Service (TAG) [8] is an aggregation service for wireless sensor networks. TAG is based on a hierarchical network which is also used as the reduction tree.
- The Cougar Project [3] also uses a hierarchical algorithm, but the approach is quite different from that of TAG. Queries are processed in a hybrid pull-push manner, where information is proactively updated at the second level view nodes, which then are queried by the root node.

In general, papers about aggregation queries in sensor networks are performed by best-effort algorithms where churn and accuracy are not taken into account. This contrasts to our work which makes adaption of parameters and choice of algorthm possible in order to achive high accuracy under churn. An exception to best-effort algorithms is the approach presented in [2], which is more robust to churn but at considerable cost for some aggregation operator types.

Related topics to churn-resistant aggregation are approaches to avoid/reduce the impact of cheaters/tampering and approximate query processing. For example, sensor networks are vulnerable to tampering, and networks could be attacked with the intent of giving erroneous answers to queries. In [13], algorithms are presented for aggregation despite compromised nodes.

When cost is more important than accuracy, approximate query processing can be employed. A sampling-based approach to aggregation query processing is proposed in [1].

In [14] the topic of accuracy in P2P aggregation was introduced.

3. DATA LOSS

In this section we investigate and formalize the notion of data loss in P2PDB aggregation queries using reduction trees. Formulas for evaluating the accuracy of query results are also presented.

3.1 Network Model

A peer-to-peer network $G = \langle V, E \rangle$ consists of a set V of nodes and a set E of network links between these nodes. Not all nodes and network links need to be fully functional at all times. Some nodes and network links may experience a failure, or may choose to part the network for a time. The total network is the network where V contains all nodes, both those currently in the network and those that are currently disconnected, and E contains all network links used by these nodes to communicate with each other. The coming and going of nodes that occurs in peer-to-peer networks is known as churn.

There are different types of events that generate churn in a peer-to-peer network. One is nodes that are joining the network or parting from it. When these events are planned, i.e., the node knows about the event in advance and may give warning to the network, we call them *voluntary.* A voluntary parting is thus when a node parts from the network in an organized way. Joins are always voluntary.

Other types of events are node and network failures. Failing nodes have no time to hand over data to other nodes or even tell the other nodes that it is failing. Data that are stored in a failing node are therefore lost until the node connects to the network again. A node may also be disconnected if its network link is disabled. In some cases a disabled network link may split the network into partitions, each partition fully functional, but with a reduced data set. A node that parts the network due to a node or network failure, parts *involuntarily.*

Due to churn, the network is split into an active network $G_a = \langle V_a, E_a \rangle$ and an inactive network $G_i = \langle V_i, E_i \rangle$. In the case of network partitioning into p partitions, there are p active networks $G_{a_1} \cdots G_{a_p}$ and one inactive network. The inactive network represents resources that could become available, but at the moment are inaccessible. The active and inactive networks are non-overlapping and $G_a \cup G_i = G$.

3.2 Processing of Aggregation Queries

The most practical solution to aggregation queries in a P2P system with focus on distributed processing is to use a reduction tree, as illustrated in Fig. 1. The nodes at the leaves of the tree start aggregating over their local database, and each leaf node sends its *partial aggregates.*

A partial aggregate is the information sufficient for creating a global aggregate value based on partial results from a number of sources. For example, assuming a grouped aggregate query for finding the average value of a column `c` in each group `g` of the relation `R`, i.e.:

```
SELECT g, AVG(c) FROM R GROUP BY g;
```

An example of a partial aggregate in this case is a 3-ary tuple consisting of a group identifier (a value from the `g` column of `R`), the sum so far for tuples within the group, and the number of tuples that have so far contributed to the result in the group.

An intermediate level node gathers partial aggregates from its children and merges these results with the result from the local database. The result of the merge is sent upwards in the tree to the parent node. This passing of partial aggregates continues all the way to the root node, which merges and evaluates the partial aggregates to get the final result of the query.

3.3 Causes for Data Loss

Queries can only access nodes in V_a, the active network. Data in nodes in V_i are inaccessible. If a node $v \in V_i$ remains inaccessible throughout the query, the only way data stored in v may be accessible is through a replica stored at another node in V_a. However, v may suddenly join the network and make its data accessible again. The opposite may also happen. A node in V_a may part the network, voluntarily or involuntarily, during query processing. Depending on whether it has processed the query or not, its data may also be inaccessible to the query.

There are three events that may occur during query processing that can affect the total database $\mathcal{B}(V_a)$ of the active network: a node may join the network, a node may part voluntarily, and, finally, a node may part involuntarily.

When joining a network, v may bring new data to the network. If $\mathcal{B}(v) = \emptyset$, $\mathcal{B}(V_a) \cup \mathcal{B}(v) = \mathcal{B}(V_a)$, and the result of the query should be the same as if v had not joined. Nodes with no data may occur if nodes that part voluntarily hand off their data to other nodes before parting, and when new nodes are introduced to the system. If $\mathcal{B}(v) \neq \emptyset$, the total database has changed, and the result of ongoing queries may be affected. If $\mathcal{B}(v) \cap \mathcal{B}(V_a) = \mathcal{B}(v)$, all data in v are already present in the database, and duplicate insensitive aggregation functions are not affected. Since data are never lost when nodes join, the problem is limited to informing the newly joined nodes of ongoing queries and let them take part in processing these.

Nodes that part voluntarily may hand off data to other nodes before they part. This would be the natural behavior in a peer-to-peer database system. In file sharing systems nodes usually take data with them when they part, but in these systems data are usually heavily replicated, so the total database of files is not affected. However, aggregates, e.g., count of nodes that contain certain files, may change.

When nodes part involuntarily, data are lost if it is not replicated on other nodes still in V_a. Also, due to the use of reduction trees, failing nodes may contain aggregated data from other nodes. These *shadow nodes* are part of the active network, but due to the failure of another node, their data are lost to the querying process.

3.4 Importance of Nodes

To investigate the consequences of involuntary parting and shadow nodes, we introduce the concept of *importance* of a node. The importance of a node v is the amount of the total database v is responsible for. In a reduction tree, leaf nodes are only responsible for their own data, so if v_l is a leaf node, its importance is

$$\mathcal{I}(v_l) = \frac{|\mathcal{B}(v_l)|}{|\mathcal{B}(V_a)|}. \tag{1}$$

Its parent node, v_i, is an internal node with C children, $v_1, v_2, \ldots, v_C$, and its importance is

$$\mathcal{I}(v_i) = \frac{|\mathcal{B}(v_i)|}{|\mathcal{B}(V_a)|} + \sum_{c=1}^{C} \mathcal{I}(v_c). \tag{2}$$

The root node is in the end responsible for the whole database, making its importance

$$\mathcal{I}(v_{root}) = \frac{|\mathcal{B}(V_a)|}{|\mathcal{B}(V_a)|} = 1. \tag{3}$$

This is natural, since if the root node fails, all data are in its shadow, and the whole query result is lost.

In a DHT we assume a uniform distribution of tuples, so the size of the local database and hence the importance of each leaf node is the same. The formulas for importance are simplified and become

$$\mathcal{I}(v_l) = \frac{1}{|V_a|}, \tag{4}$$

$$\mathcal{I}(v_i) = \frac{1}{|V_a|} + \sum_{c=1}^{C} \mathcal{I}(v_c), \tag{5}$$

$$\mathcal{I}(v_{root}) = \frac{|V_a|}{|V_a|} = 1. \tag{6}$$

Generally, the importance of a node at depth h in an aggregation tree of height H is

$$\mathcal{I}_h = \frac{k^{H-h+1} - 1}{k - 1} \cdot \mathcal{I}(v_l), \tag{7}$$

where k is the degree, i.e., the number of child nodes at each level.

3.5 Expected Data Loss

Using the notion of importance, we can calculate the expected data loss caused by a single node failure. The data loss consists of both the local database of the failing node and of all nodes in its shadow, which is summed up in the importance number for that node.

In a full, perfectly balanced reduction tree where each node is of degree k, the probability of a random failing node to be a node at depth h, is

$$\Pr(h) = \frac{k^h}{|V_a|}. \tag{8}$$

The expected data loss caused by a single node failure is

$$\mathcal{L}_{\mathrm{H}} = \sum_{h=0}^{H} \Pr(h) \cdot \mathcal{I}_h. \tag{9}$$

3.6 Accuracy

The expected data loss, $\mathcal{L}$, is closely related to the accuracy of the query result, but accuracy depends not only on how much data is lost, but also on which data are lost. Some tuples may be more important than others, and some queries and aggregation functions can tolerate more data loss than others. One way of measuring accuracy is to look at the distance from the ideal result, i.e., the result of the query if the system was not subject to churn during query processing.

For the aggregation functions *count*, *sum* and *avg*, we simply define the distance function as

$$d_{count}(r, r_i) = d_{sum}(r, r_i) = d_{avg}(r, r_i) = \frac{r - r_i}{r_i}, \tag{10}$$

i.e., the percentage of deviation from the ideal result.

The *min* and *max* functions should behave similarly, and the distance between the actual and ideal result should be comparable between these two functions. The definition of distance given for *count*, *sum* and *avg* will result in large distances for small deviations from the ideal answer of the *min* function, while the same deviation will result in a short distance for the *max* function. By defining the distance function as

$$d_{min}(r, r_i) = d_{max}(r, r_i) = \frac{r - r_i}{|D_{value}|}, \tag{11}$$

where D_{value} is the domain of the value attribute, the distance measure should be comparable for these two functions.

The definition of the ideal result, however, is not so straight forward. In a system without churn, all nodes would be connected and all data would be present in the system at all times. Given a network $G = \langle V, E \rangle$, the ideal result of a query Q can be defined as the result of executing Q on all data residing on nodes in V, i.e., the result of Q executed on all data in the total network.

Another definition may be to consider the active network $G_a = \langle V_a, E_a \rangle$ at a given time, e.g., at the start of query execution. The ideal result is then defined as the result of executing Q on all data residing on nodes in V_a at that time.

Yet another approach is to look at the nodes $V_q \subseteq V_a$ that have actually received the query. The ideal result is then the result one would get if none of V_q fails during query execution.

Whichever definition of ideal result is chosen, it should be possible to calculate the expected accuracy under a certain churn level based on the formulas for expected data loss given in 3.5. In these calculations one should notice that, e.g., the *min* and *max* functions are very sensitive to loss of tuples with extreme values, but tolerate much more loss of other tuples, so the distribution of values should be taken into account.

4. FIGHTING DATA LOSS

In this section we first look at the current method of preventing data loss on node failure, i.e., replication. We then present a two new approaches to increasing accuracy: 1) importance-based replication (IB-replication) and 2) increasing node degree.

4.1 Replication

The current approach to fighting data loss (as suggested by, e.g., [5] and [7]) is replication. This is easily done by creating several independent reduction trees, replicating the whole aggregation process. To run a parallel aggregation processes a complete trees have to be created, as illustrated in Fig. 2(b) for $a = 2$. The query is executed in each of these trees. Different trees can be constructed from the routing

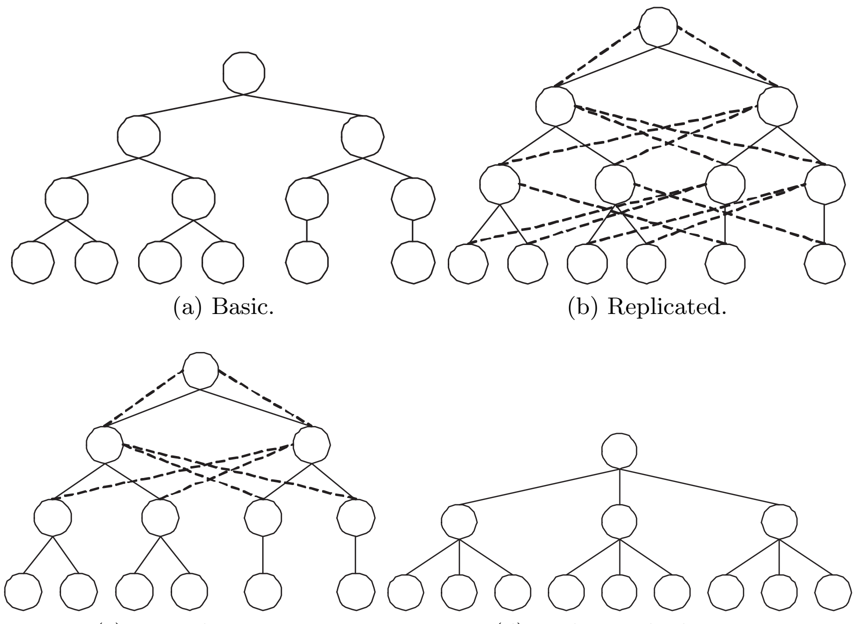
(a) Basic. (b) Replicated. (c) IB-replication. (d) Higher node-degree.

Figure 2: Variants of aggregation trees. In the case of replication, solid lines denote one of the replica trees, and the dashed lines the other replica. Note that in general, a node participating in replica trees do not have to be in the same part of the tree in the different trees (as is the case in the figures above), it can for example be leaf node in one tree but function as a higher-level node in a replica tree. Note also that in practice, the node degrees of the aggregation trees will be much higher than in the figures.

path by selecting different destination identifiers for the tree generation message.

When aggregation is done, the algorithm is presented with a set of a complete aggregation results from which it may select one. However, this still leaves room for some unnecessary data loss, as the results for one group may be better in one replica, while the results for another group is better in another replica.

The solution is to pick the best results from each replica of the final result. In general, a result is better than another if it is based on the information from more tuples, i.e., the result with the highest count is the best. This selection of the best result may be done on the final result as a whole, but also on individual groups in the results. For each group, the algorithm selects the result from the replica with the best result.

The best result of the *count* function is the maximum result among the replicas. Similarly the best result for the *sum* function over positive integers is the highest sum. The *max* and *min* aggregation functions behave similarly, choosing the maximum and minimum among the replicas for each group. When computing the *avg* function, it is hard to tell which result is the best, but again, choosing the result with the highest tuple count will statistically be the better choice.

The major drawback with replication is the increased number of messages sent. An a-replication results in a times the communication costs of the original algorithm. While even doubling the costs is expensive, selecting $a = 3$ or $a = 4$ would seriously degrade the scalability of the system.

4.2 Importance-based Replication

As noted above, replication of the complete tree results in considerably more expensive query execution. This cost may be decreased by only replicating nodes of a certain importance, with respect to Equation (7). This would result in a replication of the nodes that will have a big impact on the system if they fail, while the large number of lesser important nodes are kept at $a = 1$, i.e., unreplicated. Such a tree is illustrated in Fig. 2(c). Note that in Fig. 2(c) replication is only omitted at the lowest level, while in a higher tree more than one level might be without replication.

Importance-based replication requires the tree generation algorithm to know at which level of the hierarchy the node is placed. This information could be included in the parent node's response to the tree generation message. The replication scheme could be arranged to have several levels of replication, e.g., three copies of the immediate children of the root, two copies of their children and only one copy of other nodes. Since most nodes are placed at the leaf level or the level above, this would greatly reduce the cost of replication while still replicating the most important nodes. I.e., the result is no significant extra cost (the replicated communication and processing at upper part is very low compared to the total number of nodes in the tree participating in the processing), but accuracy comparable to full replication.

4.3 Increasing Node Degree

Replication is prohibitively costly for large systems, and other approaches should be followed if possible. Based on the formulas in Section 3.4, we propose that more attention is paid to other parameters, especially the degree of nodes. A tree with larger node-degree is illustrated in Fig. 2(d).

Using trees based on routing paths, the degree is decided by the size of the routing tables and the routing strategy used by the DHT algorithm. A large routing table that allows the node to do long jumps in the DHT space will result in a broad tree with few tiers, while a small routing table or an algorithm that only does short jumps will result in a deeper tree.

This hierarchy results in low communication costs, but the drawback is that nodes become more valuable towards the top. If one of the higher-level nodes fail, a lot of data will be lost. This effect is documented by Li *et al.* [7].

The importance of a node depends not only on its level in the hierarchy, but also on the degree. This is evident in Equations (2), (5) and (7). If the tree is broad, each of the higher level nodes are less important. A pathological case is when the degree is 1, i.e., the hierarchy is a linked list where the importance is increased by one for each level of the hierarchy. If one node is lost, all information from nodes below it in the hierarchy is lost. Assuming a uniform failure model, this hierarchy would on average lose half of the information each time a node fails.

If the hierarchy is a binary tree, the two nodes below the coordinator node are each responsible for one half of the information in the tree. Instead of losing half the information on average, it is the worst case loss. If the degree is three or four, the expected data loss is still lower. In the extreme, the degree is equal to the number of nodes, in which case the distributed aggregation degenerates into centralized aggregation.

If routing paths are used to build the hierarchy, the degree and depth of the hierarchy are decided by DHT algorithm parameters. The size of the routing table is one of the factors that decide how many hops a message needs to get to the root, and therefore also the depth of the tree. The properties of the hierarchy are also dependent on the routing policy of the network, e.g., if the DHT algorithm always routes messages in a greedy manner or if it takes other aspects into consideration. The degree of nodes could be changed either by modifying the routing principles of the algorithm, or by disconnecting the tree generation from the routing path.

In [7], Li *et al.* describe a tree construction protocol that takes the degree of internal nodes as a parameter. Such functions could be used to generate trees independent of routing paths, thus deciding in each case the desired degree of the tree.

The tree generated by existing algorithms can be estimated by considering the height of the tree to be the maximum length of the routing path, i.e., the maximum number of hops when doing a lookup. Assuming that all nodes have the same degree and that the tree is completely balanced, the degree can be calculated. The connection between number of nodes, N, degree, k, and height h of the tree is given by

$$N = \frac{k^{h+1} - 1}{k - 1}.$$

The Chord algorithm has a high-probability upper bond of $O(\frac{1}{2} \log_2 N)$, where N is the number of nodes. Experiments show that the routing path length for a Chord network of 10,000 nodes varies from 2 to 11 [15], with an average of approximately 5. Assuming that all nodes have the same degree, and that the tree is completely balanced and full, a network of 10,000 nodes with maximum path length 5 has a degree of approximately 6.

In CAN, the degree depends on the number of dimensions. Experiments in [10] show that the number of hops in a 4-dimensional CAN of approximately 130,000 nodes, the path length is approximately 5, which should give a degree of approximately 10.

These low numbers indicate that more attention should be paid to the degree of nodes in the reduction tree. From Section 3.4 we see that the degree is directly related to the importance of nodes, and hence, the expected data loss.

5. EXPERIMENTS

In this section we compare full replication and varying node-degrees by simulating aggregation by reduction trees in a DHT. Results from importance-based replications are omitted as their performance will be close to full replication (although at a very much lower cost).

5.1 Network Model

The simulated network system is a DHT network experiencing different churn levels. The focus is on the results of the queries, not on the number of messages sent between nodes or other network metrics. This allows for some simplifications in the network model.

The first assumption that is made, is that nodes that leave the network without failure, i.e., in a planned, organized way, will hand over data and ongoing queries to other nodes before disconnecting. This can be done either by transferring data and queries to other nodes, or by entering a state where the node completes all ongoing queries, but does not accept new queries. When a node no longer has active queries or data, it can leave the network. This assumption is one on the behavior of the software system, not of the network structure.

Based on this assumption, the network (without node failures) is modeled as of constant size, i.e., the number of nodes joining is equal to the number of nodes leaving the network.

Node failures are assumed to occur after the node has received all messages, but before it sends any messages. This means that failed nodes are not discovered by the network, and that all messages to failed nodes are lost. The justification for this assumption is that nodes that discover failed nodes can take actions to overcome this problem, e.g., update its routing tables and route messages through a different node.

Since some cases of node failure are supposed to be discovered and handled otherwise, the number of node failures for the simulations should be lower than in the corresponding real-world situation.

The simulations are run on a network of 10,000 nodes. 10% of the nodes fail during query processing. This is a fairly high number, chosen to show how the algorithms perform under the bad conditions.

5.2 Data and Query Model

The data model is that of a relational database consisting of a single 100,000 tuple relation which is distributed over all nodes. Since the network is based on a DHT, a uniform distribution of tuples is assumed.

The aggregation functions studied in the experiments are the standard SQL functions *sum*, *count*, *avg*, *min* and *max*.

The tuples have 3 attributes: *key*, *group* and *value*. The *key* attribute is a unique value which is used as the primary key for the tuple. When aggregating, the results are grouped by the *group* attribute, and the *value* attribute is used as parameter to the aggregation function. All values are positive integers. The *value* and *group* attributes are chosen randomly from their domains, using a uniform distribution.

5.3 Algorithm Implementation

The generated reduction trees are completely random, and not based on any DHT. This design choice allows the simulator to construct trees with different properties, so that the effect of changing the degree can be studied.

When doing replication, all replicas of the result are examined to pick out the best result for each function over each group, as described in Section 4.1. Replication is done by generating new reduction trees.

5.4 Metrics

The results are compared to an ideal result aggregated over $\mathcal{B}(V_a)$, and the accuracy of the query result is calculated using the formulas defined above. As a result, the accuracy of *min* and *max* cannot be compared directly to the accuracy of the other aggregation functions.

Each query is run 10,000 times, and the mean and inter-quartile range is used when discussing the results of the experiments.

5.5 Results

First we look at the effect of replication. Fig. 3 shows the accuracy for *avg*, *count* and *max* functions. The results for *sum* and *min* functions are similar to those of the *count* and *max* functions, respectively, and are not reproduced in the figure. The figure shows the mean value, and lines extend to the first and third quartile to show distribution density.

We see that the results of the *count* function are more inaccurate than those of *avg* and *max*. For *avg* this is the result of the DHT distributing values randomly. The loss of one node does not result in systematic data loss, so the average value is not severely affected by data loss.

The *max* function depends on a specific value being present in the accessible dataset. If this value is not lost, the function will return the correct answer. If there are more than one tuple with this value, the function is even more resilient to data loss.

The *count* function, however, depends on every tuple being present, and is thus much more vulnerable in case of node failures. Unlike the other functions, which come much closer to an accurate result even when not replicated, *count* (and *sum*) clearly show improvement when replicated.

From the results, we see that the accuracy can be increased somewhat by replicating the process, but that there is little to gain by increasing to three or four complete processes. Replication is a costly solution, and if there is little to gain in terms of accuracy, it may not be worth the cost. Also, the *avg*, *min* and *max* functions prove to be quite accurate to start with, whereas the *count* and *sum* functions show that some method to fight data loss is needed. In the rest of the experiments, only single replication is used, so the results compared are from simple hierarchical aggregation (H) and replicated hierarchical aggregation (RH).

Fig. 4 shows the results of varying the degree of nodes in the reduction tree from 10 to 1,000 (using steps 10; 50; 100; 1,000). We can see that the accuracy of *count* queries climb quite steeply from 10 to 100, i.e., from 0.1% to 1% of the number of nodes in the system, but that accuracy does not increase much beyond this number. The replicated algorithm (RH) performs better than the non-replicated algorithm (H), but for low node degrees, there is more to gain by increasing the degree than by replicating.

Due to implementation details, the replicated algorithm actually performs worse for low node degrees when computing the average, as can be seen in Fig. 5. The reason for this peculiar result, is that the simulator computes the result of *avg* from the results of *sum* and *count*.

The algorithm chooses the best result for *count* and *sum* separately, and only computes *avg* in the end. The result is that the two components of the partial aggregate for *avg* are chosen from different replicas which generally do not contain the same tuples, and the result of the query becomes more inaccurate. This shows that it is important to choose all elements of the partial aggregates from the same replica, even though the single parts may be more accurate by themselves in different replicas.

We also see that there is little to gain by increasing the degree when computing *avg* queries. The distribution becomes somewhat denser, but not much.

When computing the maximum value, the results in Fig. 6 show that the node degree is important, but that there is more to get from simple replication. For the parameters used in our simulations it is always better to replicate than to increase the degree of internal nodes.

The results from the experiments show that the different aggregation functions react differently to varying node degrees. The *count* function has much to gain from increasing the

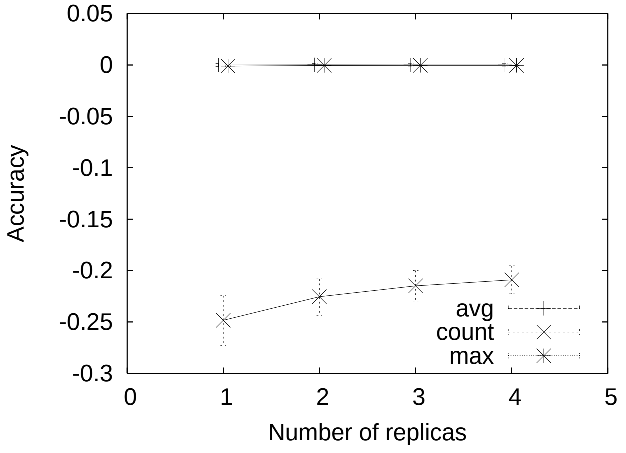

Figure 3: Accuracy of aggregation functions with different number of replicas.

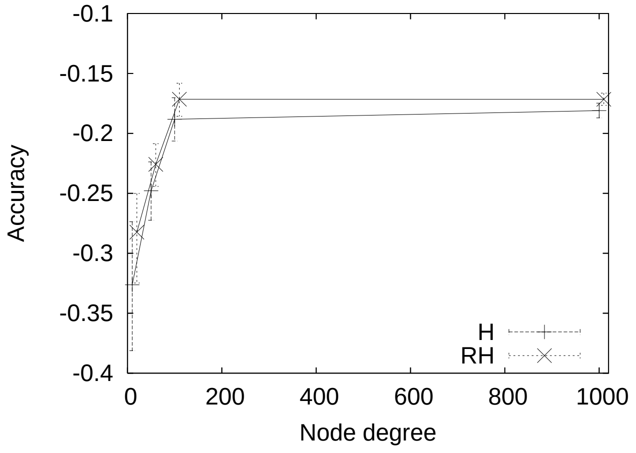

Figure 4: Accuracy of the count function with different node degrees.

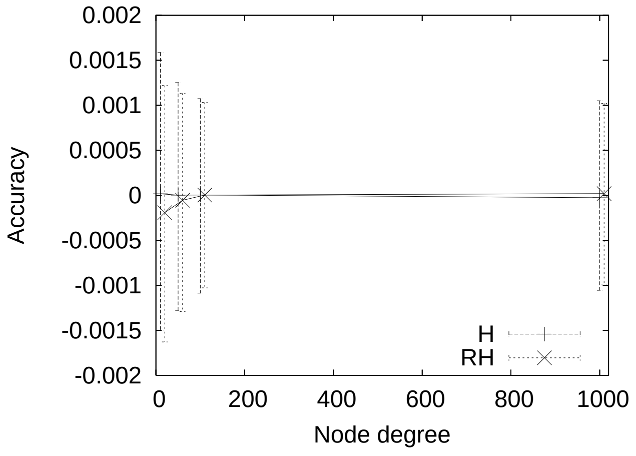

Figure 5: Accuracy of the avg function with different node degrees.

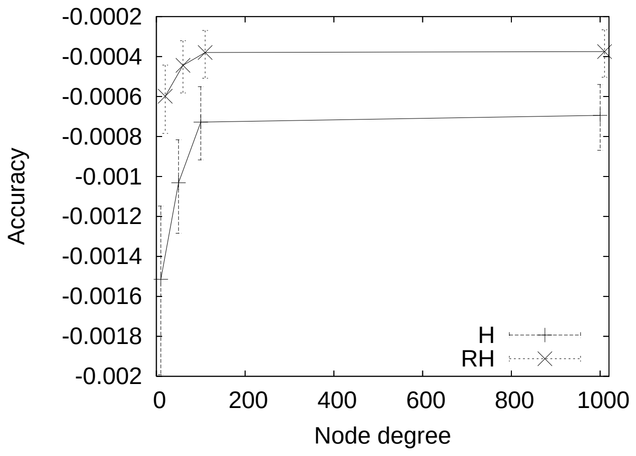

Figure 6: Accuracy of the max function with different node degrees.

node degree before replicating the process, but *max* gains more from replication than from increasing the degree. The results also show that the two techniques can be combined to increase accuracy further. The *avg* function is quite accurate to begin with, and does not seem to react much to either method.

If we compare the results to the estimated node degrees of trees based on routing paths in Chord and CAN given in Section 4.3, we see that the simulation results indicate that trees based on current DHT implementations are too narrow, and that accuracy could be increased by generating broader trees.

6. CONCLUSION AND FUTURE WORK

We have performed a formal study of data loss in aggregation queries and described the different events that may affect the result of such queries. The focus has been on the uncontrollable events, i.e., node and network failure, and how algorithms can be adapted to provide accurate answers in a setting where node failures are common.

Based on this analysis, we have proposed new approaches to increasing accuracy. Instead of just replicating the whole aggregation process, which until now has been the suggested solution, we proposed two alternatives based on importance-based replication and the degree of internal nodes in aggregation trees.

Our experiments showed that these new approaches in some cases may be more efficient in increasing accuracy than the costly replication, and also that these two methods may be combined to increase accuracy further. The simulations also indicate that there is much to gain from increasing the node degree from that of current implementations.

Several open problems remain. The query processor should be able to use statistics to predict which algorithm and which parameters would suit the query best. This could be combined with a requested level of accuracy to find the most efficient aggregation method to achieve the requested accuracy.

The data and accuracy loss of other relational operations should also be studied, so that queries do not suffer unnecessarily from data loss. When planning query execution, the methods chosen for each operation should be selected to achieve the requested accuracy. This requires a formal study of data loss and functions for predicting accuracy of relational operations.

7. REFERENCES

[1] B. Arai, G. Das, D. Gunopulos, and V. Kalogeraki. Approximating aggregation queries in peer-to-peer networks. In *Proceedings of ICDE'2006*, 2006.

[2] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. In *Proceedings of SIGMOD'2004*, 2004.

[3] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao. The Cougar project: A work-in-progress report. *SIGMOD Rec.*, 32(4):53–59, 2003.

[4] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza peer data management system. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):787–798, 2004.

[5] R. Huebsch, B. N. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi. The architecture of PIER: an internet-scale query processor. In *Proceedings of CIDR*, pages 28–43, 2005.

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

[7] J. Li, K. Sollins, and D.-Y. Lim. Implementing aggregation and broadcast over distributed hash tables. *SIGCOMM Comput. Commun. Rev.*, 35(1):81–92, 2005.

[8] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.

[9] 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.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of SIGCOMM'01*, 2001.

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

[12] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of Middleware'2001*, 2001.

[13] S. Roy, S. Setia, and S. Jajodia. Attack-resilient hierarchical data aggregation in sensor networks. In *Proceedings of SASN*, 2006.

[14] N. Ryeng and K. Nørvåg. Accuracy of aggregation in peer-to-peer DBMSs. In *Proceedings of DBISP2P'2007*, 2007.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of SIGCOMM'01*, 2001.

[16] P. Valduriez and E. Pacitti. Data management in large-scale P2P systems. In *Proceedings of VECPAR'2004*, 2004.

[17] P. Yalagandula and M. Dahlin. A scalable distributed information management system. In *Proceedings of SIGCOMM '04*, 2004.