# Query Execution and Maintenance Costs in a Dynamic Distributed Federated Database

Paul D Stone, Patrick Dantressangle, Graham Bent, Abbe Mowshowitz, Andi Toce, Boleslaw K Szymanski

***Abstract*—The cost of query evaluation in a Dynamic Distributed Federated Databases (DDFD) depends on the topology connecting the database nodes together. Different topologies provide opportunities to adopt a variety of query optimisation strategies and topology also influences the efficiency of these strategies.**

**We describe a number of strategies to optimise join queries and then derive cost estimation formulae. We use these to visualize and explore the performance of join queries in different distributed topologies. We consider three candidate topologies; a random preferential attachment network and two hypercube based models. The costs of maintaining these topologies are also formulated and compared.**

**Costs formulas are defined at a coarse grained level, using a small number of parameters and derive only the dominant or average behaviours of the queries and topologies considered. This approach is intended to provide insights to likely optimisation candidates, leading to further refinement of the models.**

**We discover that query delegation to well connected hub nodes is beneficial in scale free networks and that knowing the location of the data is beneficial in all topologies. We ascertain that a semi-join strategy can be beneficial in distributed topologies when the join attributes are small, there is repetition of join attributes or the join result in s a small number of results. We show that the cost of scaling a network is better in the preferential attachment network, indicating that a hypercube is more suitable for smaller, stable networks or where the rate of queries is high.**

***Index Terms*—Database, Distributed, Federated, GaianDB, Network Topology, Query Optimisation**

## I. Introduction

The Gaian Database [1] is a Dynamic Distributed Federated Database (DDFD) which combines the principles of large distributed databases, database federation, and network topology in a dynamic, ad-hoc environment. This DDFD has been shown to be scalable for simple queries [2] but can be enhanced to optimise complex queries such as joins, aggregate functions and nested queries. In addition to identifying optimal query plans, it is useful to determine how to establish a network overlay topology to allow optimal performance of queries, since different query strategies and different network topologies are optimal in different situations (depending on the number of nodes in the network or the size or distribution of the data to be retrieved).

Reference [3] presents coarse grained models for select queries. This paper presents coarse grained query cost models for join queries between two tables and evaluation of a semi-join strategy. We consider different network topologies: a Preferential Attachment topology and a Hypercube topology with a Content Addressable Network overlay.

## II. Related Work

There is a long history of database query cost estimation to allow efficient query evaluation. Early on the System R database [4] implemented a system to generate and manipulate query access plans in a single database system. The distributed database R* [5] extended the approach to allow optimization of queries where data is located at multiple sites. A number of databases such as AmbientDB [6], TinyDB [7] and OGSA-DQP [8] provide strategies to optimise the evaluation of queries in peer to peer networks.

The GaianDB performs multi-level peer to peer query delegation and in this context, data being processed may need to be transferred in multiple steps between database nodes. This requires a different approach to query optimisation; one which considers the topology and distance between nodes as part of the cost evaluation.

Preferential attachment networks were first introduced by Barabasi and Albert [9] in 1999 which showed that a power law distribution of node degrees was a generic characteristic of many networks which dynamically grow and independently connect together. The diameter of scale free preferential attachment networks is shown [10] to be relatively small compared to the number of nodes and the diameter grows very slowly as the number of nodes increases. The slow growth of graph diameter enables any two nodes in a network to exchange messages quickly and efficiently.

Hypercube networks have been widely used in high performance computing for routing of data at control messages between processors to enable efficient parallel processing. A survey of hypercube algorithms is found in [11]. Hypercube networks have been adopted more recently for use in dynamic distributed peer to peer networks [12] and [13].

This paper combines the previous research on query optimisation with the work on network topologies to determine the impact of topology on query optimization in DDFDs.

## III. Network Topology Models

The networks must be considered to be dynamic; nodes will frequently join and leave the network and must be able to reconnect to more appropriate nodes as the utility of different nodes changes.

To form robust networks in a scalable way, the decision as to which existing nodes a new node connects should be made on a peer-to-peer basis without need for centralised management.

## *A. Hypercube Topology in a Content Addressable Network*

An n-dimensional hypercube consists of $2^n$ nodes, each of which is labelled by a binary string of length n. Two nodes are adjacent if their respective labels differ in exactly one bit. The imposition of a hypercube structure should provide advantages when routing queries to known destinations, since hypercube labelling provides an efficient mechanism for broadcasting messages.

A method for constructing and maintaining a Content Addressable Network (CAN) within a peer-to-peer network of labelled nodes is presented in [15]. This method provides the ability to determine where specific content exists within a network and to send a query to only these nodes of interest.

Instead of broadcasting a query to all nodes of a distributed database, we propose to use the Distributed Hash Table data storage mechanism of the CAN to keep details of which nodes have which Logical Tables, so a query can determine which nodes host a fragment of the table and target the query only to those nodes.

We propose the use of a CAN to maintain a Hypercube labelled network, and the Logical Tables contained therein.

## *B. GaianDB Preferential Attachment Topology*

The GaianDB implements a preferential attachment algorithm so an entering node is more likely to connect to a node which already has a large number of connections. The aim of this approach is to reduce the diameter of the network. The number of hops between nodes is reduced to minimise the total time required to perform the broadcast-style queries of the prototype Gaian Database.

Each node that joins a GaianDB network broadcasts a request for connections and any existing, networked node responds with a delay that is depends on the number of connections that it already has; the more connections a responding node has, the shorter the delay will be.

The GaianDB is a scale free graph and experimental results confirm the expected average path length for such a graph (see Lewis [14] ) with N nodes to be approximately $\log_2 N/2$. The average path length in a hypercube is also $\log_2 N/2$.

Broadcasting a Gaian Query to all nodes results in 2N messages being sent. The average vertex degree in the GaianDB converges to 4 since each new vertex joining the network is connected to 2 existing ones. In other random graphs, the average vertex degree may not converge to a fixed value as the network grows.

## IV. Join Query Cost Models

For joins, we need to find the number of fragments of each table. We also need to consider the number of join predicates between the tables

Execution of a Join Operation gives the opportunity for data reduction – a large amount of data is considered as part of the join operation, but only a subset of that data will be found to match the join predicate. This data reduction gives us the opportunity to execute the join operation on a node close to the source of the largest amount of data, thus reducing the total volume of network traffic incurred.

We consider the cost of executing joins operators with a range of strategies.

- **Baseline** – 2 tables, all data is retrieved to the coordinating node from which the join query originates, which then performs the join.
- **Delegate to Center** - similar to the Baseline case, a coordinating node is chosen as the one with the smallest eccentricity in the network, all data from 2 tables is retrieved to the coordinating node from which the join query originates, which then performs the join. This is not a valid approach in a Hypercube network as all nodes will have the same eccentricity.
- **Delegate to a Hub** - A coordinating node is chosen as the one with the largest number of connections in the network, all data from 2 tables is retrieved to the coordinating node from which the join query originates, which then performs the join. This is not a valid approach in a Hypercube network as all nodes will have the same eccentricity.
- **Delegate to the Center of the Tables** – If we can determine which nodes in the join contain data then we may be able to calculate a central node and evaluate the query from this node.
- **Delegate to the Center of Data** – If we can determine which nodes in the join contain data and also the cardinality of the data that needs to be retrieved then we may be able to calculate a central node, considering the amount of data at each node and the distance that it would need to be communicated, so minimizing the total amount of data transferred for the query.

One consequence of delegating queries to other nodes is the increases possibility of overloading particular nodes with too many queries, causing unnecessary bottlenecks. To resolve this, the delegation mechanism should allow a node to refuse a delegation request, so the query will be executed at the original node.

Subsequent research should extend these strategies to model non-uniform data distribution models

## *A. Baseline*

The Baseline case is equivalent in terms of bandwidth to 2 select queries, all data from both logical tables is retrieved to the querying node and then the join operator processes the data here:

Definitions

N = the number of nodes in the network,

$LT_x$ = proportion of fragments of the Logical table named X,
$PL_N$ = Average path length in a network with N nodes. In most cases we will use the average path length to consider the average cost of communication between nodes without having to consider the specific distribution of nodes.
SLL = Size of a logical table lookup message and response (bytes per network step) in a CAN
SQ = Size of a query message and standard (no data) response in bytes,
SQR = size of data results per logical table fragment in bytes,

*Hypercube topology with Content Addressable Network:*

Cost (Baseline Join, HyperCAN) = $2*(PL_N * SLL + N * LTx * PL_N * SQ + N * LTx * PL_N * SQR)$

*Hypercube topology:*

Cost (Baseline Join, Hypercube) = $2*(N*SQ + N*LTx*PL_N*SQR)$

*Gaian topology:*

Cost (Baseline Join, Gaian) = $2*(2*N*SQ +N*LTx*PL_N* SQR)$

## B. *Delegate to Center*

We can reduce the costs of accreting all the input data for the join processing by choosing a central node. Then we need to transmit the join results back to the originating querying node. This may be beneficial if the result set is smaller than the original. If the join is to result in a Cartesian product (or an expansion of the data set) then the join should be performed at the originating node.

Empirical calculations made on (a small set of) Gaian topologies with up to 1025 nodes show that the average path length from a central node is approximately 14% lower than the average path length for any given node.

Definitions
SJR = size of total data results for the Join query
$PLC_N$ = Average path length from a central node in a network with N nodes. A central node is deemed to be one with the smallest eccentricity in the network.

Cost (Delegate to Center, Gaian) = $2*(2*N*SQ + N*LTx*PLC_N* SQR) + PLC_N *SJR$

It is clear that this is advantageous where SJR is small but when is this strategy more efficient than the Baseline case?

The delegation strategy is beneficial when:

Cost (Delegate to Center, Gaian) < Cost (Baseline Join, Gaian):

$2*(2*N*SQ + N*LTx*PLC_N* SQR) + PLC_N *SJR < 2*(2*N*SQ + N*LTx*PL_N* SQR)$

$PLC_N * SJR < 2*(2*N*SQ + N*LTx*PL_N* SQR) - 2*(2*N*SQ + N*LTx*PLC_N* SQR)$

$PLC_N * SJR < 2* N*LTx * (PL_N* - PLC_N) * SQR$

$SJR < 2*N*LTx * (PL_N - PLC_N)/PLC_N * SQR$

For the "typical" join, 500,000 bytes of data are transferred to contribute towards the join (2*N*LTx * SQR = 500,000). In such a case, this strategy is beneficial when SJR < 81,395 bytes.

## C. *Delegate to Hub*

A hub node is defined to be a node whose degree is equal to the maximum vertex degree in the network. This strategy involved delegation of a query to a hub node in the expectation that the average path length to the hub node is lower than the average path length from a non-hub node. This approach is valid in the case of a Preferential Attachment topology, but not in the case of a hypercube topology, where each node has the same vertex degree.

The cost formula in this case is the same as for delegation to the center, the difference being the average path length of a node from the Hub node:

$PLH_N$ = Average path length from a hub node in a network with N nodes. A hub node is deemed to be one with the highest vertex degree in the network.

Cost (Delegate to Center, Gaian) = $2*(2*N*SQ + N*LTx *PLH_N* SQR) + PLH_N *SJR$

Calculations made on (a small set of) Gaian topologies with up to 1025 nodes show that the average path length from a hub node is ~ 17% lower than the average path length for any given node. We derived a formula in IV.B showing that the delegation strategy is beneficial when:

$SJR < 2*N*LTx * (PL_N - PLC_N)/PLC_N *$ or the "typical" join, 500,000 bytes of data are transferred to contribute towards the join. In such a case, this strategy is beneficial when SJR < 102,409 bytes.

## D. *Delegate to Center of Tables*

If we know where the tables that contribute towards our join query are then we can estimate the "center point" of the tables. If we perform our query at this central node then we can benefit from a shorter "average path length" to the data nodes.

$PLCT_{N-LTx}$ = Average path length from the node which is the center of the tables in a network with N nodes, where either of the logical tables in the join is present at a proportion (LTx) of the nodes. This approach can be used in the HyperCAN topology as it is possible to ascertain the addresses of nodes with the specific logical tables referenced in the join, and to calculate the central point of these nodes.

A Monte Carlo simulation [18] was performed to find the value of $PLCT_{N\text{-}LTx}$ . The graph in Fig. 3 plots the benefits of querying from a calculated table center; the benefits reduce as the proportion of nodes participating in the query increases. Also, the benefits are lower for higher dimension hypercubes.

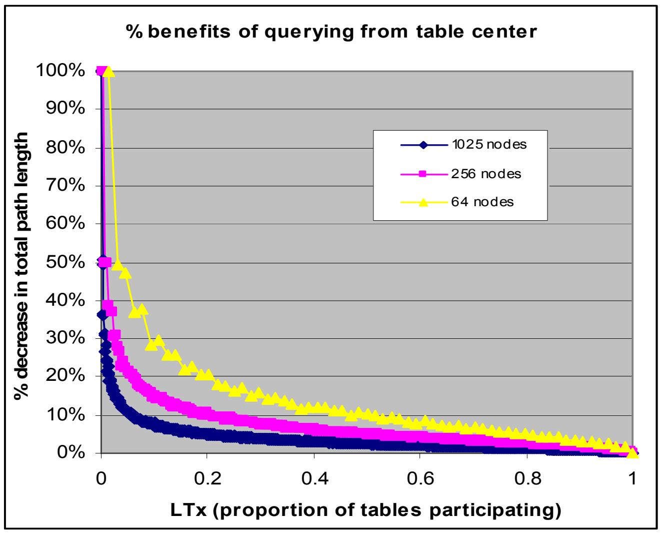

Fig. 3. Average Path length benefits evaluating a join query from a central node in HyperCAN topology

## E. *Delegate to Center of Data*

If we know where the tables are and how much data each will contribute then we can estimate the “center of data” of the tables – In a hypercube, this could give advantages for a handful of tables or when the data is skewed and predominantly at one node and it’s near neighbours.

## F. *Join Query Comparison*

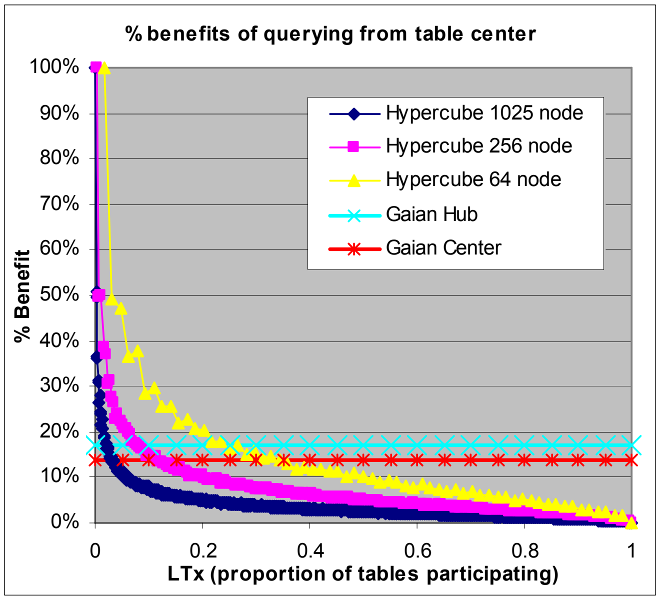

Fig. 4. Average Path length benefits querying from a central node for various topologies

As can be seen, the Delegate to Center of a join offers the strongest benefits when a small number of nodes are participating; these benefits reduce as the total number of nodes increases, or as the proportion of nodes contributing data increases. The Delegate to Center and Delegate to Hub strategies offer a more constant benefit independent of the number of nodes participating in the join query. Moreover, the Delegate to Hub strategy offers the most efficient data transfer costs when a large proportion of nodes participate in the join.

## V. Semi Joins

To reduce the cost of joins, semi-joins can be used. The approach is as follows: to join tables S and R, ship a projection of R on it’s joining attribute(s) to a site, here we can either pull all of, or a portion of the other table, then perform the join processing and then retrieve the full data of the matched sets.

We have the option to retrieve the full data, or a projection of just the joining columns for each table in a join.

Fig 1 shows two Logical Tables, LT0 and LT1 to be used for join processing. The shaded zones illustrate subsets of the data which may be used for join processing.

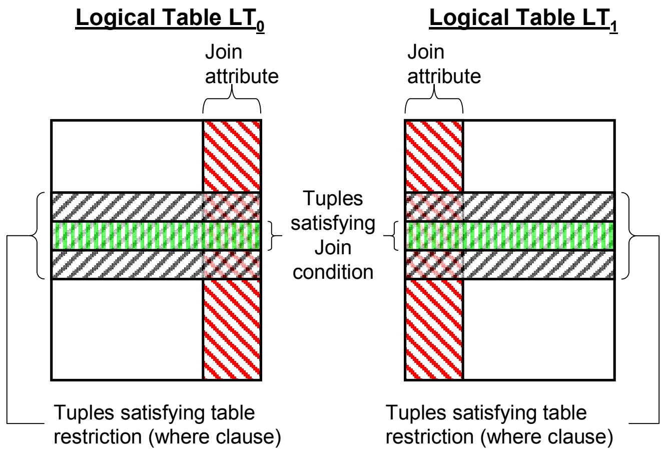
Logical Table $LT_0$
Logical Table $LT_1$
Join attribute
Join attribute
Tuples satisfying Join condition
Tuples satisfying table restriction (where clause)
Tuples satisfying table restriction (where clause)

Fig 1. Divisions of Logical tables for join and semi-join processing

Fig 2 shows the various subsets of data incorporated in join processing and their defined size parameters, which are as follows:

$SLT_x$ = the size of all the data from table $LT_x$

$SLT_xR$ = the size of data from table $LT_x$ following the initial restriction of the table data (according to query predicates)

$SLT_xPR$ = the size of data from table $LT_x$ following the restriction of the table data according to query predicates and projection of the remaining data to just the Join Attribute(s)

$SLT_xJPR$ = the size of data from table $LT_x$ following the evaluation of the join query and projection of the remaining data to the Join Attributes.

$SLT_xJR$ = the size of all tuple attributes from table $LT_x$ following the evaluation of the join query and restriction according to query predicates.

We define the scale factors between the different subsets of the table relations as follows:

$$SLT_xPR * JAP * JAD = SLT_x ,$$

where JAP is the proportion of the size of the join attributes to the size of the overall tuple, andJAD is the average duplication of the join attributes within the restricted dataset.

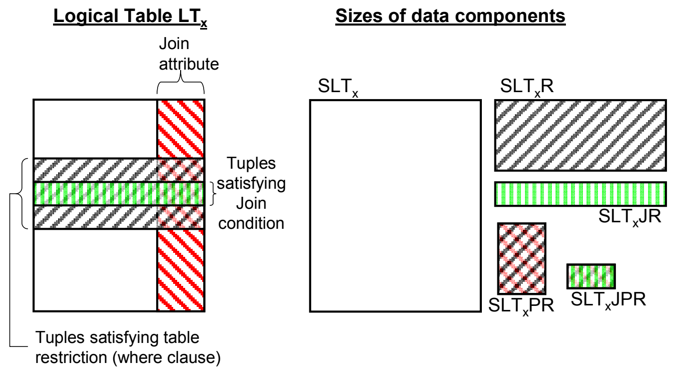

Fig 2. Sizes of table subsets for join and semi-join processing

If we choose to take the semi join strategy to stage the data fetching for both tables in the join then we have the processing sequence as shown in Fig. 3:

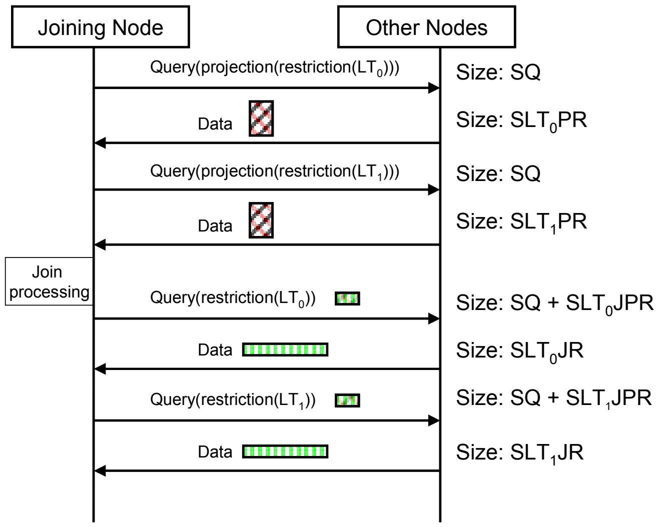

Fig. 3. Processing sequence for semi-join processing

In contrast, if we do not adopt the semi-join strategy and retrieving all data back to the join site for both tables in the join then we have the processing sequence as shown in Fig 4:

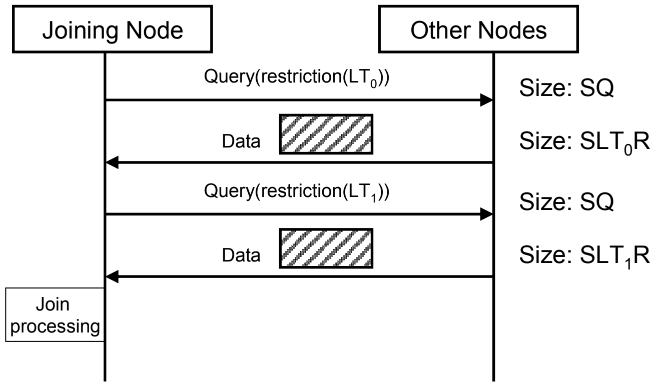

Fig 4. Processing sequence for join processing

The retrieval of each table's data is independent from the other table; In this case we can decide whether to adopt a semi-join strategy on either table independently according to the costs of retrieving the data from that table:

*GaianDB topology*

Cost (Join retrieval of $LT_x$, Gaian) = Cost (Select, Gaian) = $2*N * SQ + N*LTx * PL_N * SLT_xR$

Cost (SemiJoin retrieval of LTx, Gaian) = $2*N * SQ + N*LTx * PL_N * SLT_xPR + 2*N*(SQ+ SLT_xJPR) + N*LTx * PL_N * SLT_xJR$

The SemiJoin strategy for a particular table is beneficial in the Gaian Topology when the following condition is true:

$SLT_xR > SLT_xPR + (2/ (LTx*PL_N))*(SQ+ SLT_xJPR) + SLT_xJR$

So for a semi-join to be beneficial, the size of the join attribute data ($SLT_xPR$) plus the size of the join-matching tuples data ($SLT_xJR$) must be less than the total size of the restricted data ($SLT_xR$). Additionally in the semi join there is an overhead of propagating an additional query containing the keys of the join-matching tuples ($SQ+ SLT_xJPR$).

*Hypercube topology*

Cost (Join retrieval of $LT_x$, Gaian) = Cost (Select, Gaian) = $N * SQ + N*LTx * PL_N * SLT_xR$

Cost (SemiJoin retrieval of LTx, Gaian) = $N * SQ + N*LTx * PL_N * SLT_xPR + N*(SQ+ SLT_xJPR) + N*LTx * PL_N * SLT_xJR$

The SemiJoin strategy for a particular table is beneficial in the Hypercube Topology when the following condition is true:

$SLT_xR > SLT_xPR + ((SQ+ SLT_xJPR)/ (LTx*PL_N)) + SLT_xJR$

So for a semi-join to be beneficial, the size of the join attribute data ($SLT_xPR$) plus the size of the join-matching tuples data ($SLT_xJR$) must be less than the total size of the restricted data ($SLT_xR$). Additionally in the semi join there is an overhead of propagating an additional query containing the keys of the join-matching tuples ($SQ+ SLT_xJPR$).

*HyperCAN topology*

Cost (Join retrieval of $LT_x$, HyperCAN) = Cost (Select, HyperCAN) = $PL_N * SLL + N * LTx * PL_N * SQ + N * LTx * PL_N * SLT_xR$

Cost (SemiJoin retrieval of LTx, HyperCAN) = $PL_N * SLL + N * LTx * PL_N * SQ + PL_N * N * LTx * SLT_xPR + N * LTx * PL_N * (SQ+ SLT_xJPR) + N * LTx * PL_N * SLT_xJR$

The SemiJoin strategy for a particular table is beneficial in the HyperCAN Topology when:

$SLT_xR > SLT_xPR + SQ+ SLT_xJPR + SLT_xJR$

So for a semi-join to be beneficial, the size of the join attribute data ($SLT_xPR$), plus the size of a query message (SQ), plus the size of the join-matching attributes ($SLT_xJPR$) plus the size of the join-matching tuples data ($SLT_xJR$) must be less than the total size of the restricted data ($SLT_xR$).

A semi-join strategy is more likely to be beneficial when any of the following conditions are true:

- The join attribute is a small proportion (size wise) of the entire tuple.
- There is a lot of repetition of the join attributes in the dataset.
- The number of rows expected to "match" the join condition is a small proportion of the overall data set.

These factors are independent of the network topology considered. However the overhead of the additional query, and the distances required to retrieve data do change the exact point at which the strategies have equal cost.

There is likely to be a high correlation between the costs of joins and semi-joins at various nodes but it is unclear whether the same site that is optimal for a join also optimal for a semi-join. The choice of network topology is concerned with reducing the average path length between nodes, whereas the join/semijoin choice is concerned with reducing the amount of data to be retrieved, so these two factors should be relatively independent. However the amount of data at each node will modify the center of data calculation, and reducing the data fetched for a join will emphasise the query overhead.

## VI. Topology Maintenance Cost

Now we consider the cost of a maintaining the topologies as nodes join and leave the network. We disregard the physical layout of the network, considering all nodes to be connected by a one hop path. This model can be later extended to consider more specific physical networks.

We define the following parameters:
RL = rate at which nodes leave the network (nodes per minute)
RJ = rate at which nodes join the network (nodes per minute)

For each network topology we derive the following costs:
$CJ_n$ = Cost for joining network which has n nodes (network bandwidth in bytes)
$CL_n$ = Cost for leaving network which has n nodes (network bandwidth in bytes)
$BC_n$ = Background cost to maintain network which has n nodes (bytes per minute)

Total Bandwidth Cost = RJ * $CJ_n$ + RL * $CL_n$+ $BC_n$

*Hypercube topology with Content Addressable Network:*

Definitions

SCJReq – Size of a CAN Join request message (bytes)
SCJResp – Size of a CAN Join response message (bytes)
SCJNot – Size of a CAN Join notification message (bytes)
SDBJ – Size of a Derby Database connection (bytes)
SK – Size of a Key distribution message (bytes)
NLT – Average Number of Logical tables per node
AVD – Average Vertex Degree for node = $\log_2 N$

**Cost for joining network** - the process to join a Hypercube topology which has a CAN is derived from the KZCAN protocol for use in Kautz Graphs [17]. The Kautz graph is a directed graph, and a hypercube graph is undirected. Because of this, there is no distinction between parent and child connections in a hypercube network. When the KZCAN algorithms talk about sending messages to parents, or child connections, this is equivalent to a hypercube node sending a message to all neighbour nodes. Sibling connections are maintained as described in [17] to allow complementary nodes to be easily found when nodes leave the network.

The protocol is as follows:

1 – Assign Labels: The Joining node obtains its label by label extension (one node donates a portion of its label to be shared).

Cost = $PL_N$ * SCJReq + $PL_N$ * SCJResp

2 – Distribute Keys: The donating node sends the relevant portion of its keys (on average half) to the joining node.

Cost = $PL_N$ * (NLT/2) * SK

3 – Update Connections: Each neighbour of the donating node is informed of the new labels of the joining node and of the donating node. The down sibling of the donating node is notified, as is the new sibling of the joining node.

Cost = (AVD + 1) * SCJNot + 2 * AVD * SDBJ

4 – Update Resource Keys: Both the joining node and the joined node need to publish the location of their Logical Tables to the CAN.

Cost = 2 * NLT * $PL_N$* SK

Total Cost:

$CJ_n$ (HyperCAN) = PL * (SCJReq + SCJResp + 2.5 * NLT * SK) + AVD * (SCJNot + 2 * SDBJ) + SCJNot

Definitions

SCLReq – Size of a CAN Leave request message (bytes)
SCLResp – Size of a CAN Leave response message (bytes)
SCLNot – Size of a CAN Leave notification message (bytes)
SCConn – Size of a Connections notification message (bytes)

**Cost for leaving the network** - The process to leave a Hypercube topology which has a CAN is derived from the KZCAN protocol for use in Kautz Graphs [17]. The protocol is as follows:

1 – Find Substitute Label – The simplest case occurs if the up or down sibling has a complimentary label to which the leaving label can be merged. In this case, the intention to leave will be communicated to the sibling:

Cost = SCLReq + SCLResp

The following costs consider this "worst case" scenario where there are three active nodes – two that need to be merged, to free up one to inherit the leaving node's label.

2 – Redistribute Keys – One of the complementary nodes sends its DHT keys to its paired complementary node. The leaving node sends its DHT keys to the substitute complementary node.

Cost = 2 *NLT * $PL_N$ * SK

3 – Update Connections: The leaving node sends its connection information (of neighbours and siblings) to the substitute node. The substitute node will also send its connection information to its paired complementary node.

The leaving node, substitute node and the paired complementary node each notify neighbours and sibling nodes of the connection changes. Neighbours of the substitute node and the leaving node will need to establish a connection with the new nodes.

Cost = 2 * $PL_N$* SCConn + 3 * (AVD) * SCLNot + 2 * AVD * SDBJ

4 – Update Resource Keys: The leaving node, substitute node and the paired complementary node publish the location (or absence) of their Logical Tables to the CAN.

Cost = 3 * NLT * $PL_N$* SK.

Total cost of leaving the network:

$CL_n$ (HyperCAN) = SCLReq + SCLResp + $PL_N$*(2 *NLT*SK + 2*SCConn + 3*NLT*SK) + AVD*(3*SCLNot + 2*SDBJ)

**Background cost** – each connection is regularly checked to ensure that it is still a viable connection, each node regularly republishes the content of its logical tables to mitigate against nodes leaving without a controlled handover protocol.

$BC_n$ (HyperCAN) = AVD * n * SDBC * RCC + n * NLT * SK * $PL_N$ * RLTR

*Gaian topology:*

**Cost for joining network** – the joining node broadcasts a join request message and the majority of the nodes receiving the request (n nodes) will respond. The joining node chooses the fastest two responses to the join request and establishes a database connection to each node.

<u>Definitions</u>

SGJReq – Size of a Gaian Join request message (bytes)
SGJResp – Size of a Gaian Join response message (bytes)
SDBJ – Size of a Gaian Database connection (bytes)

$CJ_n$ (Gaian) = SGJReq + n * SGJResp + 4 * SDBJ

(The 4* multiplier represents the costs for 2 bi-directional calls).

**Cost for leaving network** – there is no direct cost of a node leaving a Gaian network, but there a chance that the node leaving is one that other nodes have connected to, in which case those nodes are forced to get another connection. Each node has an average vertex degree of 4 (2 incoming and 2 outgoing connections), so when a node leaves the network, 2 connections need to be re-established.

$CL_n$ (Gaian) = 2 * SGJReq + 2 * n * SGJResp + 2 * SDBJ

**Background cost** – each Gaian connection is regularly checked to ensure that it is still a viable connection:

SDBC – Size of a Gaian connection check (bytes)
RCC – the Rate of Gaian connection checking (bytes per minute)
$BC_n$ (Gaian) = 2 * n * RCC * SDBC

## VII. Topology Maintenance Comparison

We have taken representative values of the specified variables and then varied them to see which topologies have the lowest cost using these cost formulae. These values have been determined by monitoring real life traffic from the Gaian Database.

Fig. 5 shows the cost of maintaining the network algorithm according to the algorithms outlined above. Assuming that all the edges incident to every node must be checked, then 2N checks are required in a GaianDB and N*Log N checks in a hypercube. The cost of background connection checking increases more rapidly in the case of the Hypercube topology.

Fig. 6 shows the costs excluding the regular background connection checking. We can see that the scalability of the Hypercube topology is better than the Gaian topology.

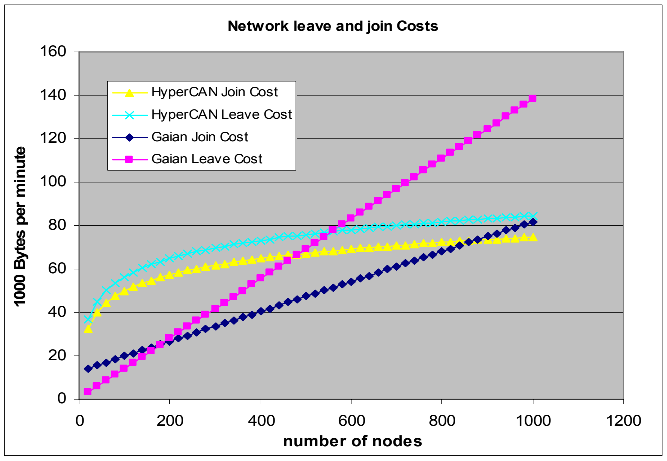

Fig. 5. Total Network Maintenance cost for different topologies, varying the number of nodes.

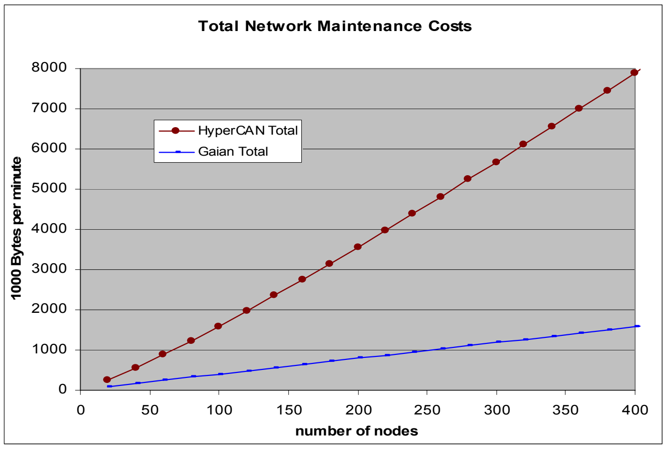

Fig. 6. Network Join and leave cost for different topologies, varying the number of nodes.

## VIII. Conclusions

These results allow the quantification and comparison of the network bandwidth costs associated with Queries in a DDFD (such as the GaianDB). The costs of different query strategies in a preferential attachment network and a hypercube network are modelled, allowing the selection of a suitable query strategy within a given network topology, or allowing the choice of network topology if the query workload is known as the network is constructed.

For Join queries, it is possible to improve the query cost by evaluating a query at a central or well connected node. This reduces the total number of legs that query data has to pass before the data can be combined and reduced. The random preferential attachment topology shows a better overall percentage benefit when delegating to a central or well connected node in larger networks, or where there is a high proportion of nodes contributing data to the Join. A semi-join strategy is likely to be beneficial when any of the join attribute constitutes a small proportion of the entire tuple, there is considerable repetition of the join attributes in the dataset, or if the number of rows expected to "match" the join condition is a small proportion of the overall data set.

For a network with N nodes, the connection checking bandwidth scales with O(NlogN) for the HyperCAN topology and O(N) for the Gaian random preferential attachment topology. Thus scaling in this situation is better in the random preferential attachment topology.

## IX. Next Steps

Query cost issues requiring further investigation include join partitioning schemes, joins with skewed data distributions, data replication, query delegation mechanisms, distribution of information about the data at a node and its location.

The maintenance and distribution of statistics on the Logical Tables will be investigated e.g. the amount and distribution of data at each node. We will compare the cost incurred communicating the statistics and the benefits realised with their use.

## Acknowledgement

Research was sponsored by US Army Research Laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon

## References

[1] G Bent, P Dantressangle, D Vyvyan, A Mowshowitz, V Mitsou, "A Dynamic Distributed Federated Database", "*Proc. 2nd Ann. Conf. International Technology Alliance*" 2008.
[2] G. Bent, P. Dantressangle, P. Stone, D. Vyvyan, A. Mowshowitz, "Experimental Evaluation of the Performance and Scalability of a Dynamic Distributed Federated Database", "*Proc. 3rd Ann. Conf. International Technology Alliance*" 2009.
[3] P. Stone, P Dantressangle, G. Bent, A Mowshowitz, A. Toce, B. Szymanski, "Relational Algebra Coarse Grained Query Cost Models for DDFDs", "*Proc. 4th Ann. Conf. International Technology Alliance*" 2010.
[4] P.G. Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, T.G. Price, "Access Path Selection in a Relational Database Management System", "*SIGMOD '79*" 1979
[5] L. F. Mackert, G.M. Lohman, "R* Optimizer Validation and Performance Evaluation for Distributed Queries", "*VLDB '86*"
[6] C. Treijtel, "AmbientDB: Complex Query Processing for P2P Networks", "Proceedings of the VLDB 2003 PhD Workshop"
[7] S Madden, M. Franklin, J. Hellerstein, W Hong, "TinyDB: An Acquisitional Query ProcessingSystem for Sensor Networks", "ACM Transactions on Database Systems Volume 30, Issue 1 (March 2005)"
[8] M. Alpdemir, A. Mukherjee, A. Gounaris, N. Paton, P. Watson, A. Fernandes, D. Fitzgerald, "OGSA-DQP: A Service for Distributed Querying on the Grid" "Advances in Database Technology - EDBT 2004"
[9] A. Barabasi, R. Albert, "Emergence of scaling in random networks", *Science 286*. 1999
[10] B. Bollobas, O. Riordan. "The diameter of a scale free random graph", *Combinatorika 24*, 2004
[11] S. Lakshmivarahan and Sudarshan K. Dhall,. "Ring, torus and hypercube architectures/algorithms for parallel computing. "*Parallel Computing 25(13-14)*" 1999
[12] E. Anceaume, R. Ludinard, A. Ravoaja, and F. Brasileiro. "Peercube: A hypercube-based p2p overlay robust against collusion and churn",. "SASO '08: Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems"
[13] A. Toce, A. Mowshowitz, P. Stone, P. Dantressangle, G. Bent, "HyperD: A Hypercube Topology for Dynamic Distributed Federated Databases", "*Proc. 5th Ann. Conf. International Technology Alliance*" 2011
[14] T. Lewis, ""Network Science, Theory and Applications", 2009.
[15] P. Fraigniaud, P. Gauron, "D2B: a de Bruijn Based Content-Addressable Network", 2006.
[16] Z Wang, E Bulut, B K Szymanski, G Bent, A Mowshowitz, P Stone, "An Energy Efficient Dynamic Location Server Hierarchy for Mobile Ad Hoc Networks", 2010
[17] J Yu, J Song, H Liu, L Zhao, B Cao, "KZCAN: A Kautz Based Content Addressable Network", 2007
[18] Metropolis, N. and Ulam, S. "The Monte Carlo Method.", "*Journal of the. American. Statistical. Association. 44*", 1949