Query Delegation and Hill Climbing Algorithms for DDFD Join Queries

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

Abstract— Previous research into the Gaian Database has identified that the cost of executing Join queries in a DDFD network is lower at “Hub” nodes, which tend to have a lower average path length to the remaining nodes. This paper defines a low-cost algorithm to delegate queries to nodes with higher vertex degree, thus reducing the cost of query evaluation. The benefit of adopting this algorithm is evaluated by analysing simulated networks.

Index Terms— Database, Distributed, Federated, GaianDB, Query Optimisation, Query Delegation, Hill Climbing.

I. INTRODUCTION

The Gaian Database [1] is a dynamic, distributed federated database which combines the principles of large distributed databases, database federation, and network topology in a dynamic, ad-hoc environment.

The GaianDB has been shown to be scalable for simple queries [2] but can be enhanced to optimise complex queries such as joins. In addition to identifying optimal query plans, we can determine how to establish a network overlay topology to allow optimal performance of queries.

This paper presents a protocol to allow delegation of queries between nodes in a DDFD, defines a hill climbing algorithm based on the vertex degree of each node at its neighbours. The ability of this hill climbing algorithm to find a low cost location to perform a database query is assessed.

II. QUERY DELEGATION

Coarse grained models [3] have shown that in a network formed by preferential attachment, the vertex degree distribution of the nodes varies. There is a strong correlation between nodes with a high vertex degree and a low average path length. For both Join queries and Aggregate queries the cost of evaluating a query is proportional to the average path length from the evaluating node to all other nodes. This is illustrated by considering the Gaian Database shown in Fig 1.

Assume that we want to evaluate a query which will require 100 data items from each node:

If we evaluate a query at node A, it can get the data from its two neighbours (B, D) in one network hop. It has 4 nodes which are 2 network hops away (C, E, F, G), 2 nodes which are 3 hops away (H, I) and one node which is 4 hops away. In total, data has to be passed 20 hops (2 x 1 hop + 4 x 2 hops + 2 x 3 hops + 1 x 4 hops).

Alternatively, if we evaluate a query at node F, it can get the data from its six neighbours (B, C, D, G, H, I) in one network hop. The remaining 3 nodes are all just 2 network hops away (A, E, J) so in total, data has to be passed 12 hops (6 x 1 hop + 3 x 2 hops).

This shows that in this example, delegating a query from node A to node F can result in a bandwidth saving of 40% to retrieve the set of data under consideration. This is useful for queries such as join operations and aggregate functions where the set of data processed is reduced down to a far smaller set of “matches” or “aggregate values” which are then returned to the delegating node.

In general we model the network as an undirected graph $G = (N, E)$ where $N$ is the set of vertices corresponding to the nodes of the network, and $E \subset N \times N$ is the set of edges corresponding to the direct link between nodes in the network. The $i$-th neighbourhood $N_i(u)$ of vertex $u$ is defined by: $N_0(u) = \{u\}$, and for $i>0$ $N_i(u) = \{v \in N | d(v,u) \leq i\}$ where $d(v,u)$ is the distance (i.e., length of a shortest path) between $v$ and $u$ in the graph.

The set of vertices whose distance to $u$ is exactly $i$ is denoted by $S_i(u)$, defined by $S_i(u) = N_i(u) - N_{i-1}(u)$ for $i > 0$.

Suppose that each node $u$ has the volume $V(u)$ of data to send to evaluate a query. The optimum node to evaluate the query is at the vertex which satisfies the following:

$$\min_{u \in N} \sum_{i=0}^{\infty} i^* \sum_{v \in S_i(u)} V(v)$$

If we assume that for each node $V(u)$ is constant $V$ then

$$\sum_{v \in N_i(u)} V(v) = V^* |S_i(u)|$$

so the problem simplifies to:

$$\min_{u \in N} \sum_{i=0}^{\infty} i^* |S_i(u)|$$

The processing of a query in a DDFD should be implemented to consist of an initial delegation phase and a subsequent evaluation phase. In the delegation phase, the database nodes are able to pass the query between themselves to determine the best location for evaluating the query. Once a node decides that there is no benefit in delegating the query further then it becomes responsible for evaluating the query, which is effected in the same manner as in the current implementation of the Gaian Database.

Fig 2 shows a flow chart illustrating a query delegation process for queries in a DDFD.

Each node in the chain of delegation follows this flow chart. At any point in time, a single node is deemed to be the “responsible delegate”, whose role is to find a suitable node...
to coordinate the evaluation of the specified query.

This responsible node will evaluate its own fitness to evaluate the query and then based on its knowledge of other nodes will judge whether other nodes will be “fitter” for evaluating the query. Fitness in this context involves estimating the cost of query evaluation with the knowledge of the network available.

If other nodes are potentially fitter then the responsible node, then the responsible node will attempt to delegate the evaluation of the query to candidate nodes that it considers fitter. The query delegation request from the responsible delegate to a candidate will contain the query and may also include the responsible node’s fitness to allow the delegate node to compare fitness.

Those candidate nodes may actually determine that they are not fitter, or may be too overloaded with current queries so may choose to reject the delegation request.

Eventually, a node will determine that it is fittest, or find that its candidates reject further delegation requests, so it actually starts the evaluation of the query, determines the query results and returns these back via the delegation chain to the initial requester of the query.

The Client application injecting the query into the Gaian Database Network is deemed to be delegating the query to the first Gaian Database node and this node must always accept the delegation.

III. HILL CLIMBING ALGORITHM

The delegation mechanism outlined in Section II relies on each vertex having an assessment of relative fitness and cost of query evaluation at that location. We consider cost to be a measure of the total network bandwidth required to evaluate the query, and so fitness is inversely related to the cost of the query.

We show in Section V.A that the degree of a vertex u is inversely related to the sum of the path lengths to every other vertex in the graph. In DDFD networks this in turn is proportional to the cost of retrieving all data in the network when data in uniformly distributed over the vertices. So from any vertex, we should delegate join and aggregate queries to those vertices in the network which have the highest degree.

A design principle of the dynamic distributed federated database is that it is undesirable to require each vertex to have global knowledge of all other vertices. If each vertex requires knowledge of others then the amount of data to communicate increases as O(n²) and algorithms requiring this exchange of information are inherently un-scalable.

An alternative approach to determining candidate delegates is to perform a “hill climbing” algorithm, where each vertex attempts to delegate the query to its fittest neighbour:

1. The vertex determines its degree.
2. The vertex determines the degree of each adjacent vertex.
3. If any adjacent vertex has a degree higher than this vertex, the vertex delegates the query to the vertex with the highest degree.
4. If no adjacent vertex has degree higher than this vertex (or all higher degree candidate delegates reject the query) then evaluate the query from the current vertex.

![Fig 3. Hill climbing query delegation example](image)

The Hill Climbing algorithm is illustrated in Fig 3. A query is injected into vertex J, which has degree 2. It has two adjacent vertices: H and I with degrees 5 and 4 respectively. The query is delegated to vertex H as it is the adjacent vertex with the highest degree. Vertex H has 5 adjacent vertices: E, F, G, I and J. Of these, F has the highest degree: 6 so the H delegates the query to vertex F. F determines that none of its adjacent vertices has a higher degree than F itself, so the query is not delegated further, but is evaluated at this vertex. The evaluation of the query is performed by flooding the query to all vertices, including H and J. The result set of the query is determined at F, which then returns the results to the vertex J via H.

As can be seen, the query is delegated one step at a time, always to a fitter vertex and is evaluated at a vertex which has the largest degree of all its neighbours.

Note that in Fig 3, vertices D and F each have the same degree 6, but the cost of evaluating a query at these vertices will be different. The degree is a fitness measure which is directly related to the query cost, but is not an absolute measure of query cost. Vertex D has 2 neighbours at
distance 2 and 1 neighbour at distance 3. The sum of distances from D to all other vertices is 13 (a 35% saving over vertex A) as opposed to vertex F where the sum of distances to all other vertices is 12 (40% saving over vertex A).

Note that Vertex C is connected to vertices D and F which have the same degree. In this situation, the Hill Climbing algorithm evaluated at vertex C will choose to delegate the query equally in a random manner to the vertices D and F.

Fig 4 exhibits the same graph as in Fig 3, with arrows between vertices showing how a query is delegated by the hill climbing algorithm.

Fig 4. Hill climbing query delegation graph for example graph

A solid directed arrow indicates that the vertex at the start of the arrow will delegate a query to the vertex at the arrow’s endpoint.

Vertices which can delegate equally to multiple neighbours (because of equal fitness) are connected with dashed arrows.

Vertices which are local maxima (none of their neighbours has a higher degree) have no outgoing arrows: D and F in Fig 4 for example. These are the vertices at which the hill climbing algorithm terminates and at which the queries will be evaluated.

A. Suboptimal terminal vertices.

With complex graphs it is possible to have vertices which are local maxima, i.e. they have the highest degree of all of their neighbours, but they may have a smaller degree than that of another vertex in the graph. This is shown in Fig 5. Vertices O and P each are local maxima, having 5 edges each so the hill climbing algorithm will terminate at these vertices, and queries will be evaluated at O and P. However vertices M and N each have 10 edges and M or N would be better locations to evaluate a join or aggregate query.

These suboptimal terminal vertices can each have their own catchment basin. In Figure 5, queries injected at vertex Q will end up either at O or P, and queries injected at S have a chance to end up at N, O or P.

The likelihood of suboptimal terminal vertices has been identified by simulation and detailed in Section V.D of this paper.

IV. VERTEX DEGREE METRIC DISTRIBUTION

To allow the hill climbing algorithm to operate, each vertex must know the degree of its neighbours. This information can be obtained in a number of ways including the following:
[1] Determine the degrees of adjacent vertices at query time,
[2] Communicate the degrees of adjacent vertices periodically.

The Gaian Database currently executes a small query periodically to test that each of its connections is still valid. This connection checking query could be extended to communicate the degree of adjacent vertices, thus allowing the degree of each adjacent vertex to be communicated with low additional cost, piggy-backing on the existing connection checking.

V. EVALUATION OF HILL CLIMBING EFFECTIVENESS

In this section we address the efficiency of the Hill climbing algorithm to delegate queries to vertices which have relatively low total distances from all other vertices. By extension, this allows us to assess the reduction in cost of join and aggregate function queries. As the hill climbing algorithm has been shown to fail to reach the absolute best vertex in some cases, the rate of success, and total distance improvement are considered.

We produced a Java application capable of simulating the formation of Graphs according to the Gaian Database growth algorithm. We used this simulator to grow multiple instances of graphs and evaluate the performance of the Hill climbing algorithms in terms of the improvement of
Average Path Length. Graphs where the maximum degree per vertex is capped or uncapped were simulated to compare results. Connectivity in this simulation is modelled in such a way that each vertex is capable of a direct connection to each other vertex, so a vertex can chose to connect to any other vertex in the graph and will chose to do so according to the fitness of that vertex.

A. Correlation between Vertex Degree and Path Length

Fig 6 shows a graph plotting the degree from simulated graphs against the Average Path Length (APL) realised at that vertex.

APL is defined as the sum of distances from a vertex to all other vertexes in the graph, divided by the total number of vertices in the graph:

$$APL(u) = \frac{\sum d(v, u)_{v \in N}}{n}$$

This is shown for different graph sizes and for each size there is a strong correlation between the vertex degree and the APL. The correlation coefficient is shown on the graph for each graph size and varies from -0.967 for a 10 vertex graph to -0.817 for a 1000 vertex graph.

In both capped and uncapped graphs, hill climbing in small graphs (10 vertices) gives an improvement of 29%.

In uncapped graphs this initially improves as the graphs grow, reaches a maximum of 34% at around 200 vertices, and then slowly degrades to 32.4% at 5000 vertices.

In graphs with a cap of 10 edges per vertex, the hill climbing improvement gradually degrades as the graph size grows. The improvement is 14.8% at 1000 vertices and 12.5% at 5000 vertices. This degradation in benefit is caused by the increase of number of vertices in the graph which have the maximum degree, but different eccentricities. This algorithm does not distinguish between them, so we are less likely to find a vertex with the lowest eccentricity.

B. Average Path Length Improvement

Fig 7. shows the simulated results of performing the Hill Climbing algorithm for Gaian Networks of varying sizes from 10 vertices up to 5000 vertices. In some results, the maximum vertex degree was capped at 10 edges (as is the case with the current Gaian Database) and other results allowed any vertex to have an unlimited number of edges.

C. Number of vertices improving and worsening

Fig 9. Shows the number of vertices in the graphs which improve by reducing their Average Path Length, stay the same (most probably because the vertex is the highest of all
its neighbours) or in some cases, the query may “climb” to a vertex with a worse (higher) Path Length. Note that the lines showing the worsening vertices are plotted on a different scale, the values of which are shown on the right hand side of the graph.

Fig 9. Number of vertices improved with Hill Climbing

It can be seen that capped graphs (each vertex limited to 10 edges) have a lower ratio of vertices improving: 88% as opposed to 97% in uncapped graphs. Capping a graph results in an accumulation of vertices with the maximum degree and so it is more likely that vertices will not improve in a capped graph.

The proportion of vertices which result in a worse path length when the query is delegated is also higher in capped graphs than uncapped. This starts off at zero in small graphs and at 5000 vertices, climbs to 0.77% in capped graphs and 0.43% in uncapped graphs.

The vertices that have the same path length after hill climbing are those which are local degree maxima, so the hill climbing algorithm does not delegate in these cases. Fig 10 shows the percentage of vertices which are local degree maxima for different sizes of graphs.

For 1000 vertex graphs which are capped, there are 112 vertices (11.2%) which are degree maxima. Of these, 92 are optimal (with 10 edges) and 20 are suboptimal.

For 1000 vertex graphs which are uncapped, there are 23 vertices (2.3%) which are degree maxima. Of these, 1 is optimal and 22 are suboptimal.

Note that the degree maximal vertices will be those at which queries will be evaluated, so in capped graphs, queries will be evaluated at 11.2% of the vertices and in uncapped graphs, queries will be evaluated at 2.3% of the nodes.

D. Suboptimal terminal vertices

Fig 11 shows the proportion of vertices that achieve a suboptimal result from the hill climbing algorithm. This is the case when there is a vertex degree somewhere in the graph which is higher than the local maximum reached by hill climbing. In large graphs, there is a higher rate of suboptimal results in uncapped graphs, 25% in graphs of 5000 vertices as opposed to 7% in capped graphs.

Fig 12 shows the impact of suboptimal terminal vertices by comparing the Average Path Lengths from the Hill Climbing end vertex with that from the global optimum vertices, i.e. the vertices with the highest degree in the whole graph.
It can be seen from Fig 12 that in capped graphs, there is a small gap between the optimal vertices and hill climbing vertices, but a larger gap in uncapped graphs. This is caused by the higher proportion of sub-optimal terminal vertices shown in Fig 11. Note that the gap between the two algorithms does not increase proportionally as the graph grows.

Table 1 gives the APL improvement figures for 1000 vertex graphs, comparing capped and uncapped graphs and optimal vertices against hill climbing algorithm end points.

```
<table>
<thead>
<tr>
<th></th>
<th>Optimal Vertices</th>
<th>Hill Climbed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capped</td>
<td>15.4%</td>
<td>14.8%</td>
</tr>
<tr>
<td>Uncapped</td>
<td>37.3%</td>
<td>33.2%</td>
</tr>
</tbody>
</table>
```

Although the Hill Climbing algorithm can result in suboptimal results relative to the global optimum, the cost of performing the Hill Climbing is lower than the cost of determining the location of global optima.

The cost of performing the Hill Climbing algorithm is proportional to the diameter of the graph; log(N), so the complexity is O(log(N)).

The global optima can be determined by performing a flood query to all vertices in the graph. This is simple to perform in a Gaian Database, but the cost is proportional to the number of vertices in the graph, so the complexity is O(N).

So the Hill Climbing algorithm cuts the cost of finding good vertices by O(N/log(N)) while the good vertices found are a little worse than the global optimum (empirically 4% less saving in capped graphs and 10% less saving in uncapped graphs).

### E. Impact of non-uniform data distribution

It can be seen from these simulation results that the Hill Climbing algorithm is effective for delegating a query to a vertex where lower overall network traffic is incurred when retrieving uniformly distributed data.

However, if data is clustered in specific locations in the graph, hill climbing based purely on vertex degree could potentially delegate a query to a vertex further from the location best placed for retrieving all the data.

A fitness function could be defined which considers the amount of relevant data proximate to each vertex. This will be more costly to determine and so there would be a larger overhead of maintaining this delegation strategy.

### VI. Conclusions

It can be seen from these simulation results that the Hill Climbing algorithm is effective for delegating a query to a vertex where lower overall network traffic is incurred when evaluating queries retrieving uniformly distributed data.

The cost of performing the Hill Climbing algorithm is low compared to determining the global optimum location and delivers a large proportion of the delegation benefit.

Delegation using this Hill Climbing algorithm would have benefits for queries such as Queries and Aggregate functions where data from across the network needs to be brought to a single location to process, resulting in a reduced result set.

The overall improvement in capped and uncapped networks is shown for various network sizes in Table 2.

```
<table>
<thead>
<tr>
<th>Size</th>
<th>Capped</th>
<th>Uncapped</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>28.75%</td>
<td>29.83%</td>
</tr>
<tr>
<td>50</td>
<td>24.15%</td>
<td>33.27%</td>
</tr>
<tr>
<td>100</td>
<td>21.21%</td>
<td>33.77%</td>
</tr>
<tr>
<td>500</td>
<td>16.17%</td>
<td>33.49%</td>
</tr>
<tr>
<td>1000</td>
<td>14.80%</td>
<td>33.23%</td>
</tr>
<tr>
<td>5000</td>
<td>12.47%</td>
<td>32.40%</td>
</tr>
</tbody>
</table>
```

The algorithm considered used the degree of each vertex as the metric determining the fitness of a vertex. This is effective in networks with evenly spread data. In networks with clustered data, the effectiveness of this algorithm is expected to worsen.

It appears that uncapped networks provide better opportunities for query delegation than capped networks, however there is a risk that vertices with greater numbers of connections will become overloaded and become bottlenecks in the system, reducing the overall query throughput. An analysis needs to be performed to determine the optimum capping strategy to allow delegation (and a lower diameter in general) without causing bottlenecks.

### VII. 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

