Experimental Evaluation of the Performance and Scalability of a Dynamic Distributed Federated Database

Graham Bent, Patrick Dantressangle, Paul Stone, David Vyvyan, Abbe Mowshowitz

***Abstract*— This paper presents the results of an experimental evaluation of the performance of a dynamic distributed federated database (DDFD). Databases of varying sizes up to 1025 nodes, have been evaluated in a test environment consisting of a 22 IBM Blade Servers plus 3 Logical Partitions of an IBM System P Server. The results confirm that by 'growing' databases, using biologically inspired principles of preferential attachment, that the resulting query 'execute time' is a function of the number of nodes (N) and the network latency (T_L) between nodes and scales as $O(T_L \log N)$. Distributing data across all the nodes of the database and performing queries from any single node also confirms that, where network bandwidth is not a constraint, the data 'fetch time' is linear with number of records returned.**

***Index Terms*— computer network performance, computer networks, database systems, dynamic distributed databases, Gaian database, dynamic graph construction, mobile ad hoc networks**

I. Introduction

DYNAMIC Distributed Federated Databases (DDFD) combine the principles of large distributed databases [1], database federation [2], network topology [3] and the semantics of data [4]. The resulting information management architecture is ideally suited to a range of applications that are required in support of the type of ad-hoc networked operations being considered by the ITA. One challenge in supporting such ad-hoc networked operations is that of providing efficient access to distributed sources of data, where the applications requiring the data do not have knowledge of the location of the data within the network. To address this challenge the DDFD adopts a 'Store Locally Query Anywhere' mechanism (SLQA), which provides for global access to data from any node in the database network. The DDFD also permits data from heterogeneous data sources to be accessed as a single federated database, enabling applications to perform database queries that can combine data from across these sources. Data access can be controlled using policy based mechanisms such as those being studied in Technical Area 2 of the ITA and this leads to new models of the management of policy based access control that are discussed in [5].

Our Previous work on DDFD has been focused on the theoretical characteristics of this type of database and has shown how biologically inspired principles of network growth, combined with graph theoretic methods, can be used to predict the expected query performance characteristics [6,7,8]. In [7] we also described an implementation of a DDFD which we termed the GaianDB. The GaianDB uses a principle of preferential attachment [9,1] to control how new database nodes are added to the network of existing database nodes. This attachment mechanism should result in a connected graph structure that has predictable properties that directly impact on database query performance. The purpose of this paper is to demonstrate that this is indeed the case. In the GaianDB, queries for data are performed from any node in the database network and are propagated using a controlled flood mechanism. Nodes that can return results in response to receiving a query return results to the querying node over the set of shortest path(s). The query performance can therefore be measured in terms of the time to perform the flood query (execute time) plus the time to fetch the actual results (fetch time) along these shortest paths. In this paper we investigate the actual performance characteristics of the GaianDB implementation of a DDFD, for database networks of various

Manuscript received May 8, 2009. Research was sponsored by the U.S. Army Research Laboratory and the U.K. 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 author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

Graham Bent is with IBM, Emerging Technology Services, Hursley Park, MP137, Winchester, Hants. SO21 2JN, UK (e-mail: GrahamBent@uk.ibm.com).

Patrick Dantressangle is with IBM, Emerging Technology Services, Hursley Park, MP137, Winchester, Hants. SO21 2JN, UK (e-mail: DANTRESS@uk.ibm.com).

Paul Stone is with IBM, Emerging Technology Services, Hursley Park, MP137, Winchester, Hants. SO21 2JN, UK (e-mail: STONEPD@uk.ibm.com).

David Vyvyan is with IBM, Emerging Technology Services, Hursley Park, MP137, Winchester, Hants. SO21 2JN, UK (e-mail: DRVYVYAN@uk.ibm.com).

Abbe Mowshowitz is with the Department of Computer Science, The City College of New York (CUNY), New York, NY 10031, USA (212-650-6161; fax: 212-650-6248; e-mail: abbe@ cs.ccny.cuny.edu).

sizes up to 1025 nodes and compare this with the theoretical predictions.

II. Theoretical Predictions

The predicted efficiency of this network database in terms of execution and fetch time is in part determined by how the nodes of the database are connected together. We consider the database network as a connected graph G with each database node as a graph vertex. The eccentricity $\varepsilon(v_i)$ of a graph vertex v_i is the maximum graph distance between v_i and any other vertex v_j of G i.e. the "longest shortest path" between any two graph vertices (v_i , v_j) of the graph. The maximum eccentricity is the graph diameter G_d. The minimum graph eccentricity is the graph radius G_r. We define the size of G as the number of vertices N and the number of connections at each vertex as the vertex degree δ_i $(1 < i \leq N)$.

Since we propagate the query to every node in the database network, our approach to minimise query execution time is to minimise $\varepsilon(v_i)$ of G whilst retaining some control over the distribution of δ . Clearly a trivial approach to minimising $\varepsilon(v_i)$ is to have all database nodes connect to all other nodes directly. However this approach would not scale for large N, since it would require N connections per node. Our initial approach to the problem was to consider a network topology that is formed as a scale free random graph [9]. Scale free graphs exhibit a number of desirable properties such as a predictable bound on the graph diameter and on the distribution of vertices and edges. We have also investigated ways of constructing the network through a process of graph combining operations [8], which seek to minimise the resulting network diameter and minimise the vulnerability of the network to attack from the removal of vertices and edges.

The dynamic construction of such networks requires that when a new vertex v_n is added to the graph m new edges are added one at a time to the existing graph. One approach to the creation of scale-free random graphs is to add edges e_{ni} from vertex v_n to an existing vertex v_i with a probability proportional to the degree of v_i. This is termed preferential attachment. In a previous paper we reported that according to Bollobas & Riordan (2004) [11], if m = 1 the resulting network has a graph diameter that scales as O{log N}, where N is the number of vertices in the network. Whilst for m >1 (typically m =2 or 3) the upper bound on the graph diameter scales as O{log(N) / log log(N)}. Whilst a preferential attachment strategy of this type does result in a small graph diameter, it does so with a vertex distribution P(δ) that exhibits a power law distribution of the form:

$$P(\delta) = C\,\delta^{-\tau} \qquad (1)$$

Where C and τ are constants. This type of distribution results in a relatively few vertices with high degree and many vertices with low degree. Using unconstrained preferential attachment would therefore still require some database nodes to support large numbers of connections which is not desirable both in terms of supporting connections to other nodes and in having to process high volumes of queries and routing data through these nodes. It has also been shown by Albert, Jeong and Barabasi [12] that by the removal of the high degree nodes the network rapidly fragments leaving it susceptible to attack.

We therefore investigate a dynamic construction strategy that uses preferential attachment but where the maximum number of connections at any vertex is limited to some value MAX[δ]. In this case we find that for the database networks that we investigate in this study (typically N<2000) that the dynamic construction of networks in this way results in graphs in which the maximum graph diameter scales as A + B ln(N), where A and B are constants.

III. Simulation Results

To determine the effect of our dynamic graph construction strategy diameter we performed a Monte Carlo simulation to determine the maximum graph diameter and minimum graph radius for graphs of varying sizes between 10 and 1025 nodes for different values of the cut off parameter MAX[δ] and the number of edges added for each new vertex m. In this paper we present the results for MAX[δ] = 10 and m = 2, i.e. each new vertex connects to 2 existing vertices with the proviso that no vertex may exceed 10 connections. Figure 1 illustrates the results of the simulation.

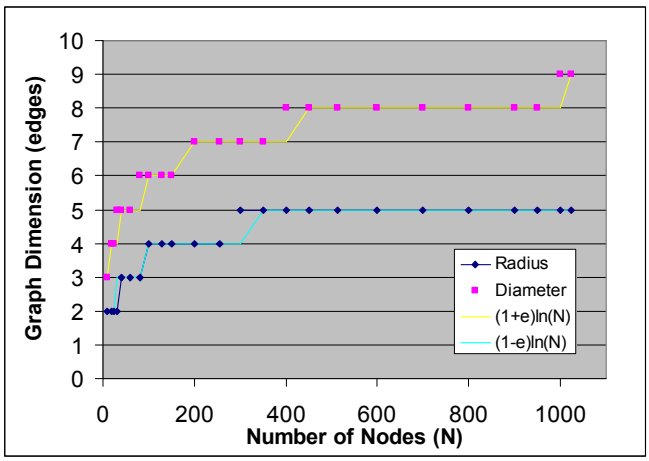

Figure 1. Measured and predicted bounded graph diameter and radius using preferential attachment with m=2 , MAX[δ] = 10 and e= 0.24.

The results show that the bounds on graph diameter and radius scale logarithmically. Interestingly, they obey the following relationships:

$$G_d = \text{nint}\,[\,(1+e) * \ln(N)\,] \qquad (2)$$

$$G_r = \text{nint}\,[\,(1-e) * \ln(N)\,] \qquad (3)$$

Where e is a constant (=0.24 in Figure 2), nint is the nearest

integer. This scaling is similar to that predicted in [11] for m = 1 and large N but the reason for this scaling for graphs of smaller size requires further investigation.

IV. Experimental Test Environment

The Test environment is shown in Figure 2. The environment comprises 22 IBM Blade Servers distributed over two Blade Centres, plus 3 Logical Partitions of an IBM System P Server. These were used to host the database nodes. An additional Blade Server was used to configure the network of GaianDB Nodes, to inject queries, and to measure the performance of the resulting distributed database network. These machines were connected by a Gigabit Ethernet LAN as shown in Figure 2.

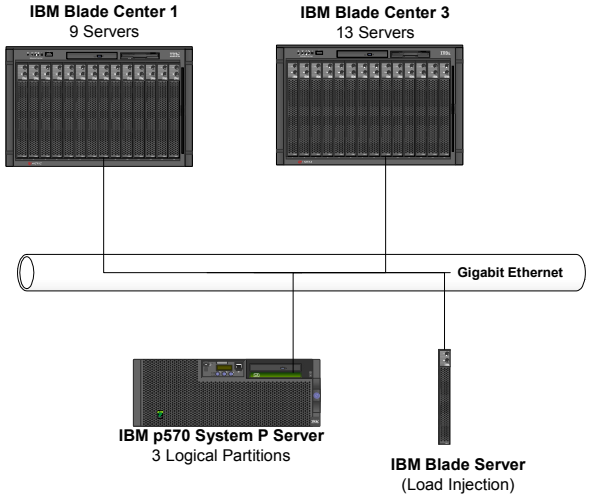

Figure 2. Experimental Test Harness comprising 22 blades and an IBM P-Series server

"Blade Center 1" comprised 9 Blade Servers each having 2 hyper-threaded 3200 GHz processors and 5 Gigabytes of RAM. "Blade Center 3" comprised 13 Blade Servers, each of which has 2 hyper-threaded 3800 GHz processors and 5 Gigabytes of RAM. The Operating System installed on all blades is Red Hat Enterprise Linux AS release 4.

In addition 3 Logical Partitions (LPAR) were configured on the IBM System P Server. Each LPAR has 4 power 5 processors (1900 Ghz speed) and over 20 Gb of RAM. The Operating System installed is AIX 5.3.

Our GaianDB implementation is based on the Java open source Derby database [13]. The flood query propagation from any node to its connected neighbours is done in parallel. Data fetch operations use a records buffer which is shared between all connections (execution threads) to prevent blocking on database i/o batch-fetch operations. This optimises the throughput of each database node.

Each Server in the Test Environment was configured to run up to 41 GaianDB nodes giving a maximum of 1025 nodes in the whole environment. Each node runs in its own Java Virtual Machine (JVM) with 64 Mb of memory allocated to each JVM and each configured on a separate TCP/IP port. GaianDB's of various sizes were grown using the connection method described in [7]. In this method, each time a node is added to the network it generates an IP multicast which existing nodes respond to by offering connections to the joining node. The nodes that have the highest 'fitness' respond preferentially faster and the joining node accepts the first two connections to be offered. The fitness measure used in the series of experiments presented is based on the number of connections that a node already has, up to a limit of 10 nodes when the fitness function drops to zero.

V. Experimental Results

We investigated the query performance of a GaianDB for networks of varying size, up to 1025 nodes using the test harness. Figure 3 shows an example network configuration of 1025 nodes, created using preferential attachment with MAX[δ] = 10 and m = 2.

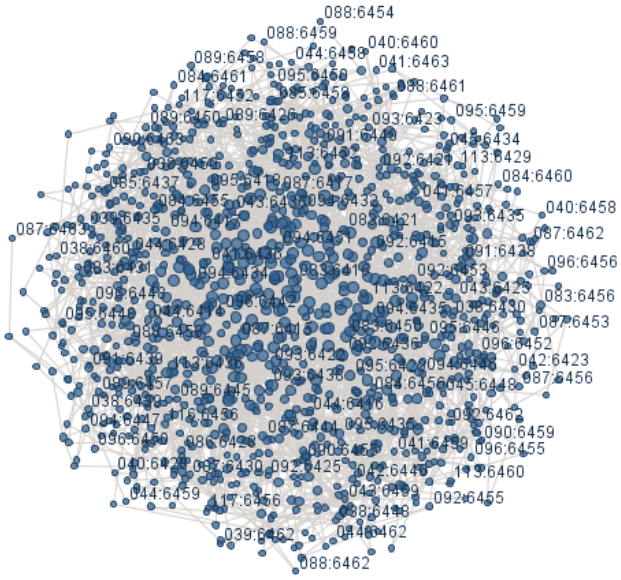

Figure 3 . Illustration of a 1025 Node DDFD – the size of vertex indicates the vertex degree and the numbers are the unique node identifiers.

In these investigations query performance was measured in terms three parameters:

A) the time to propagate a query to all of the nodes in the database, as a function of the number of database nodes (N);

B) the time to fetch data from across the nodes of the database to a single node, as a function of the volume of data;

C) the time to fetch data from across the database to multiple nodes concurrently querying, as a function of the number of nodes concurrently querying.

A. *Query Propagation Time*

When a query is performed at any node in the database network the query is propagated to the immediately connected nodes, which in turn propagate the query to their respective neighbours. If the same query arrives at a node via multiple paths then only the first instance of the query is propagated. This means that the query propagates one extra hop beyond the graph eccentricity of the node from where the query was injected. Figure 4 illustrates this effect for a node with a vertex degree of 9 and an eccentricity $\varepsilon(v) = 1$ where each node returns 1250 rows of data. We refer to the extra hops that result as 'cross-talk'. Various strategies for reducing the cross-talk are currently being investigated.

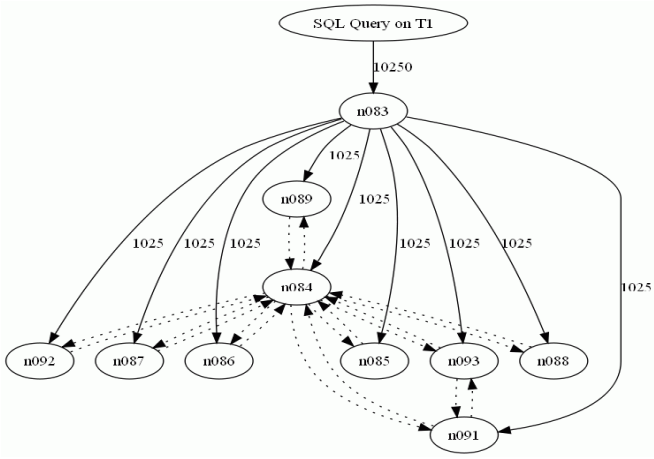

Figure 4. Query propagation for a single layer showing effect of cross-talk on path length.

Analysis of the network structure was performed each time nodes were added and for each configuration the eccentricity from every node was measured and the graph radius and diameter determined. To investigate the effect of the controlled flood query in each configuration, a fixed delay of 50ms was introduced prior to each node executing a query. This time can be considered to be the equivalent of a network latency time T_L between the database nodes. In addition there is a measured process delay (T_p) of 3.9ms for each GaianDB node to process the query. Thus the predicted maximum (T_{max}) and minimum times (T_{min}) to execute the flood query are:

$$T_{max} = (G_d + 1)(T_L + T_p) \quad (4)$$

$$T_{min} = (G_r + 1)(T_L + T_p) \quad (5)$$

with the predicted execute query time from any node (T_v) being:

$$T_v = (\varepsilon(v) + 1)(T_L + T_p) \quad (6)$$

These predicted maximum and minimum times are shown in Figure 5 for networks of varying size up to 1025 nodes. Also shown in Figure 5 are the measured average execution times, obtained by querying 0 records from the same node in each database configuration, and the predicted execution time, calculated by measuring the eccentricity of the query node in each database configuration. The result shows that this selected node is a central node for some configurations but is never a peripheral node. Most importantly the results shows that query execution time does follow expected logarithmic scaling i.e. O(logN) and that:

$$T_v = \text{nint}[1 + B * \ln(N) * (T_L + T_p)] \quad (7)$$

where the constant B depends on $\varepsilon(v)$.

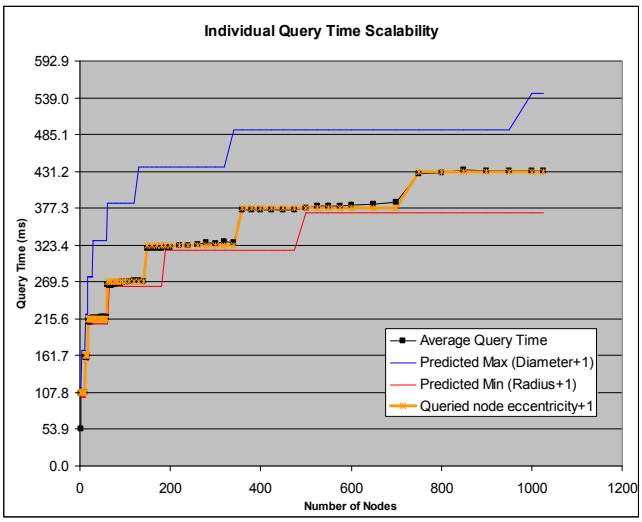

Figure 5. Average Query Execute Time with T_L=50ms process delay T_p= 3.9ms for database size up to 1025 nodes.

Figure 6 shows the individual measured execution times, for databases up to 100 nodes. This shows variability in the process time T_p with the majority of times clustered around the predicted value based on the query node eccentricity and then a number of outliers with higher processing time. Analysis of the higher values show that these are highly correlated with 'garbage collection' events in the Java Virtual Machines that run the individual Gaian Database nodes. The results confirm that, under conditions where network bandwidth is not a constraining factor, the GaianDB is highly scalable, with a predictable query execution time.

(3

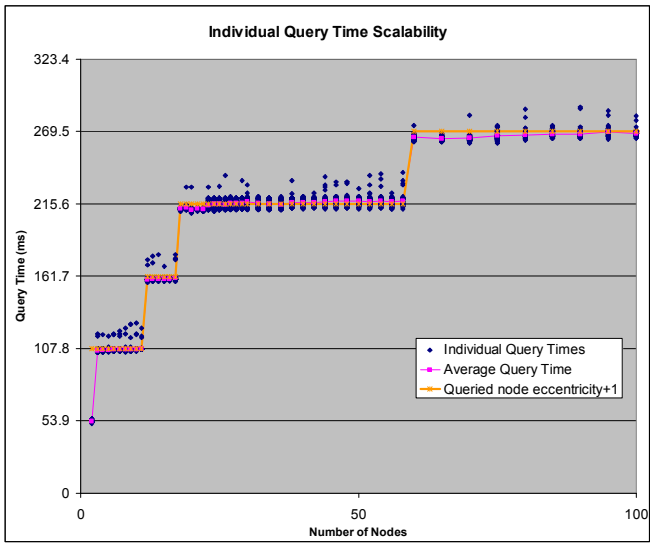

Figure 6 Individual Query Execute Time with T_L=50ms process delay T_p= 3.9ms for database size up to 200 nodes.

B. *Data Fetch Time*

To determine the time to fetch data from across the distributed database network each node of a 1025 node database was configured to store 1025 rows of data with no index on the database tables. SQL queries were then launched from a single node to select the same number of rows from all nodes varying from 1 row per node to 1025 rows per node (i.e. 1025 to 1.05M rows). Further measurements were performed to determine the time taken to fetch the same number of rows from a database with just one node containing all 1.05M rows both with and without an index. The results are presented in Figure 7 which shows that, following the initial execution time (in this case 485ms), the fetch time for the 1025 node database is linear with the number of rows selected with an average fetch time of 4.2µS per row.

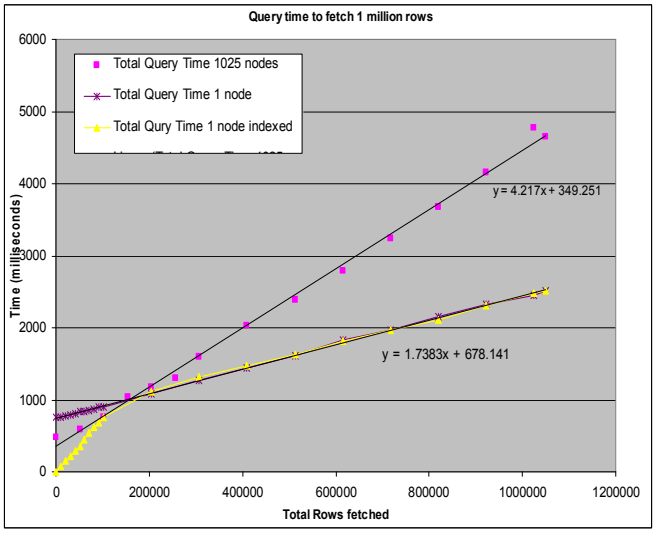

Figure 7 Query Fetch Time and Execute time as a function of the number of rows fetched per node from a 1025 node database.

The result obtained from querying a single node, with no index, also show a linear fetch time with an average of 1.7µS per row. Above 18000 records being fetched, querying from a single node is faster than the distributed database but below this value the distributed query is faster due to the smaller amounts of data being fetched from each individual node because smaller table scans are required on each node. With indexing, the fetch time from a single node drops dramatically below 20 thousand records and is always faster than the distributed database with no indexing. An interesting result, based on this observation, is that when fetching relatively few records, the distributed database acts as an index itself, cancelling the requirement to index the smaller tables on each node.

C. *Concurrent Query Time*

To demonstrate the effect of concurrent queries in the experimental test bed we investigate both the effect on query execution time and data fetch time. To obtain these measurements it was found to be necessary to restrict the size of the database network to 500 nodes each storing 1024 records. Larger networks were affected by processor contention, as an increasing number of Java objects are created per server when performing concurrent queries and this results in increased levels of 'garbage collection'. This is an issue with the sharing of database nodes on the same processor and would not be an issue where nodes are running on individual processors.

To determine query execution time (i.e. querying 0 records), queries were injected into the database at randomly chosen nodes and the query was then repeated 30 times concurrently on all chosen nodes. Figure 8 shows the average execution time for up to 40 concurrent querying nodes being injected at randomly selected nodes in the database.

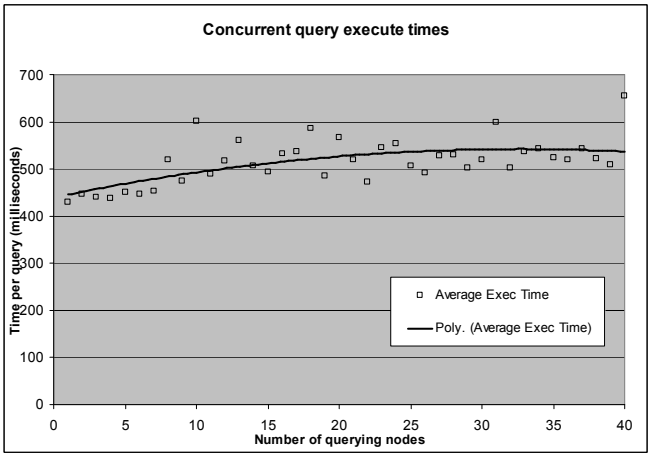

Figure 8 Query execution times as a function of the number of concurrent querying nodes for a database of 500 nodes.

The predicted execution times from equations (4) and (5) where for the 500 node Graph G_r = 6 and G_d = 8 and Tp =

4mS, Tl = 50 ms, giving a range from 378 – 486ms. Below 8 concurrent queries the average execution time falls within this range. Detailed analysis of the results shows that above this level concurrent queries increasing amounts of Java garbage collection occur increasing the average value and hence the measured execution timing increase above that predicted from network propagation time alone.

To determine query fetch time the same randomly selected nodes were used with concurrent queries selecting either all 1025 rows or 100 rows from all 500 nodes in the network. The results are shown in Figure 9 for the two cases and confirm that the average query time is a linear function of the number of concurrent queries, which is proportional to amount of data being fetched across the network.

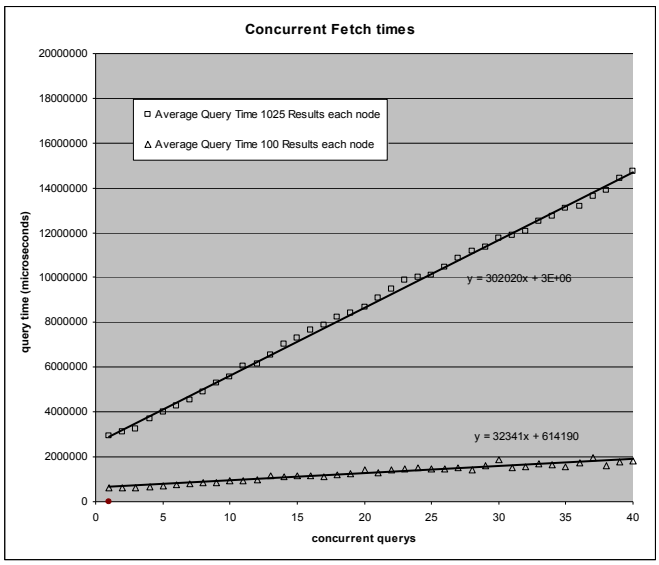

Figure 9 Query fetch time as a function of the number of concurrent queries for a 500 node database. Results are shown for concurrently fetching 1025 rows from each node and 100 rows from each node.

This result can be better understood by considering the average query fetch time as a function of the total number of rows being fetched across the network as a whole. This is presented in Figure 10 for a single node case and for the two concurrent query cases and then for the concurrent query cases respectively.

When selecting 500 thousand records from the single node all of the data flows back to the node along the shortest paths. On average each of the nearest neighbours processes this number of rows divided by the average vertex degree. Similarly the second neighbours (on the next step out) have a further reduction in the number of rows to be processed and so on, out to the furthest neighbours. If the amount of data being brought back to the same querying node is increased, then all the workload of all nodes increase in proportion to the increase in data being fetched. However, each time a new concurrent query is added, the workload on each node increases in proportion to the relative position of the node to the new querying node. On average this will be a linear increase as the number of rows fetched increases but it will always be less than fetching the extra number of records from a single node (red line in figure 10). Further work is required to quantify this effect in detail.

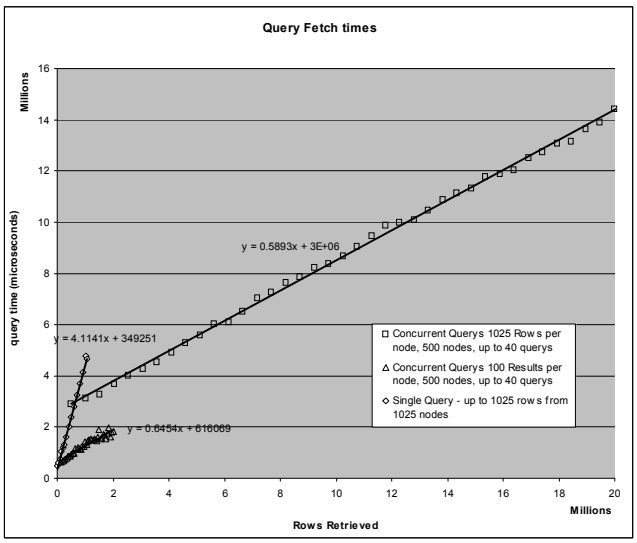

Figure 10 Query fetch time as a function of the number of concurrent rows of data being retrieved from across a 500 node database. Also shown for comparison are the results of fetching up to 1 million rows querying from a single database node reproduced from Figure 7.

The results show that when querying from a single node, retrieving more data takes longer than it does to retrieve it from a collection of nodes in the network. This demonstrates and quantifies the benefit of parallel data fetching

VI. Conclusion and Future Work

This paper has presented the results from an experimental evaluation of the performance and scalability of a Dynamic Distributed Federated Database. The results have shown that by considering the database as a connected graph that the database performance in terms of query execution and query fetch times can be predicted. Dynamic construction of a graph using preferential attachment with a limit on the vertex degree has shown both by simulation and experiment that the query execution time scales as $O(T_L \log N)$ and that query fetch time for data distributed across the database network is linear with the number of records returned . Concurrent query fetch time has been shown to be proportional to the total number of rows retrieved from across the distributed database.

The results confirm that this type of distributed database is highly scalable and that databases comprising very large numbers of nodes could be constructed with little impact on query performance. Further work is required to assess the impact of bandwidth constraints, consider different network construction algorithms and to provide a more robust mathematical analysis of concurrent query performance.

REFERENCES

[1] J. Becla, K. T Lim. “Report from 1st Workshop on Extremely Large Databases”, Held at Stanford Linear Accelerator Centre, Menlo Park, CA, USA, October 25, 2007.

[2] L. M. Haas, E. T. Lin and M. A. Roth, “Data integration through database federation”, IBM Systems Journal, Volume 41, Number 4, 2002.

[3] M. Newman, “The structure and function of complex networks”, SIAM Review, 45, 167–256, 2003.

[4] N. Shadbolt, W Hall,T Berners-Lee "The Semantic Web Revisited" IEEE Intelligent Systems 1541-1672/06 2006

[5] S.Carlo, D. Wood, P.Zerfos, D.Vyvyan, P.Dantressangle, G.Bent. “Technologies for Federation and Interoperation of Coalition Networks”, The 12th International Conference on Information Fusion, Seattle, July 2009.

[6] A. Mowshowitz and Graham Bent. “Formal properties of distributed databse networks”, ACITA, September 2007.

[7] G. Bent, P. Dantressangle, D. Vyvyan, A. Mowshowitz and V. Mitsou. “A dynamic distributed federated database.” Second Annual Conference of ITA, Imperial College, London, September 2008.

[8] A. Mowshowitz, V. Mitsou and Graham Bent, “Models of Network Growth by Combination”, Second Annual Conference of ITA, Imperial College, London, September 2008.

[9] M. Newman, S Strogratz, D Watts “Random graphs with arbitrary degree distributions and their applications” Physical Review E, Volume 64, 2001.

[10] L. Adamic, R Lukose, A Puniyani, B. Huberman. “Search in power-law networks” Physical Review E, Volume 64, 2001.

[11] B. Bollobas, and O. Riordan. “The diameter of a scale free random graph”, Combinatorika 24 (1), pp. 5-34, 2004.

[12] R. Albert, H, Jeong, A.-L Barabasi. “Error and attack tolerance of complex networks” Nature, Vol 406, July 2000.

[13] Derby Database http://db.apache.org/derby/ (last accessed 8rd May 2009).