A Dynamic Distributed Federated Database

Graham Bent[#], Patrick Dantressangle[#], David Vyvyan[#], Abbe Mowshowitz[*] and Valia Mitsou[**]
[#]IBM, Emerging Technology Services, Hursley Park, MP137, Winchester, Hants. SO21 2JN, UK, [*]City College of NY (CUNY), Convent Ave. at 138th St., NY, NY 10031, USA; [**]Graduate Center, CUNY, 365 Fifth Avenue, New York , NY 10016, USA

Abstract-**This paper describes a new type of database architecture which we define as a dynamic distributed federated database (DDFD). Using biologically inspired principles of network growth combined with graph theoretic methods we describe how a DDFD can be developed and maintained. The DDFD uses a 'Store Locally Query Anywhere' mechanism (SLQA), which provides for global access to data from any vertex in the database network. To reflect this capability we have coined the name GaianDB for this type of database architecture. The main aim of the paper is to describe the principles of the DDFD approach and to describe a light weight implementation of a DDFD (<4Mb) based on the Derby database engine. We describe how this has been used to demonstrate a sensor network as a DDFD and how a DDFD can be used to perform distributed semantic search over a network of search engines.**

I. Introduction

Military, industrial and scientific data volumes continue to grow enormously in size and complexity, such that traditional transactional database architectures can no longer be used cost effectively to support the growing demand for information exploitation. The current trend is a move away from large monolithic database architectures to more distributed database technologies. In this paper we describe a new type of Dynamic Distributed Federated Database (DDFD) that combines the principles of large distributed databases [1], database federation [7], network topology [11] and the semantics of data [15], see Figure 1. The resulting database architecture is ideally suited to a range of applications that are required in support of networked operations.

The approach that we have taken differs from other current approaches to the distributed database challenge, such as the map-reduce [4] approaches being used in the Apache Hadoop project [8], in that we are seeking, as far as possible, to maintain a strict distributed relational database model. In this model data is stored in local database tables at any vertex in the network, and is accessible from any other vertex using Structured Query Language (SQL) like queries and distributed stored procedure-like processing.

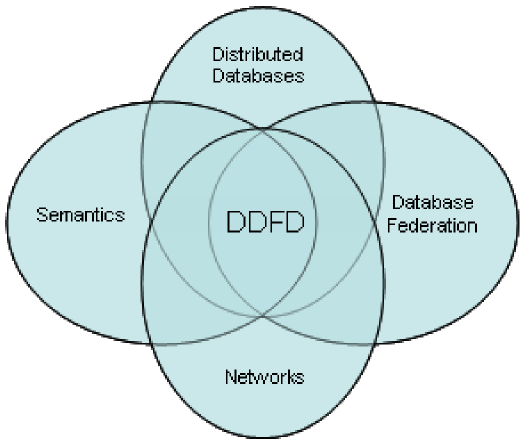

Figure 1. *DDFD's in relation to existing technologies*

One of the key aspects of our approach is the way in which the different vertices of the DDFD interconnect. Properties such as the degree of separation between any two vertices in the networks, and the degree distribution of connectivity at each vertex in the network have a significant impact on query performance, resilience and vulnerability. Using biologically inspired principles of network growth [11], combined with graph theoretic methods [9,1], we describe how a DDFD can be developed and maintained.

The 'Store-Locally-Query-Anywhere' (SLQA) provides a mechanism for global access to data from any vertex in the database network. To reflect this capability we have coined the name GaianDB for this type of database architecture.

II. DATABASE ARCHITECTURE

The federated distributed database concept that we have been investigating is illustrated in Figure 2. The database comprises a set of interconnected vertices (N1,N2…) each of which is a federated Relational Database Management System (RDBMS) engine. By federated we mean that the database engine is able to access internal and external sources as if it was one logical database. External sources may include other RDBMS such as Oracle, SQLServer etc. (the cylinder/drum icons in Figure 2) or any other data source such as flat files of data records (rectangles with bent corner in Figure 2).

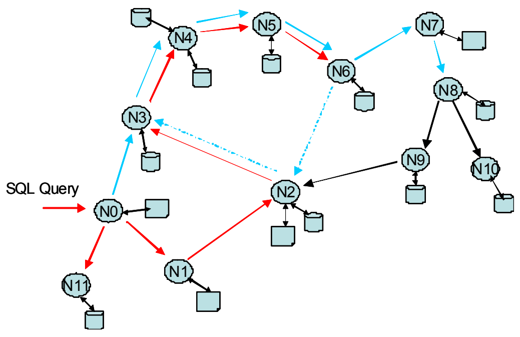

Figure 2. *The Gaian Database Concept*

Data can be stored at any vertex in the network of database vertices, with the table in which it is stored being available to other vertices through a logical table mapping which is implemented as a Virtual Table Interface (VTI) [5]. Queries can be performed at any vertex requesting information from any other vertex in the network. The query propagates through the network (currently a flood query) and result sets are returned to the querying vertex. The vertices manage the forward routing of queries so as to minimise the number of copies of the same query. Thus a query will flood through all paths of the database and result sets will be received from vertices that can satisfy the query, with null results (i.e. table found with no data) or zero result sets (no table found along the route) being returned. Each vertex that forwards the query is responsible for processing the results obtained from vertices to which the query was forwarded, leading to a distributed aggregation of results (where this is possible e.g. union operation).

The GaianDB concept can be used to store and access many types of data, we have been investigating the concept of a distributed sensor network in which sensors store data in a database table or file. The idea of treating a sensor network as a distributed database is not new [17,6]; however the approach to query propagation and the use of the SQLA approach is significantly different. The concept is illustrated in Figure 3, in which sensors of different types store data in local tables. Knowing the sensor schema, an application requiring data queries the network from any vertex using predicates to limit which sensor data are returned. For example, a sensor's location or type may be defined and queried as a predicate to limit which vertices return data. The model is significantly different from the publish/subscribe model being investigated in [3] since we do not attempt to stream data from a sensor but only to request data from the network.

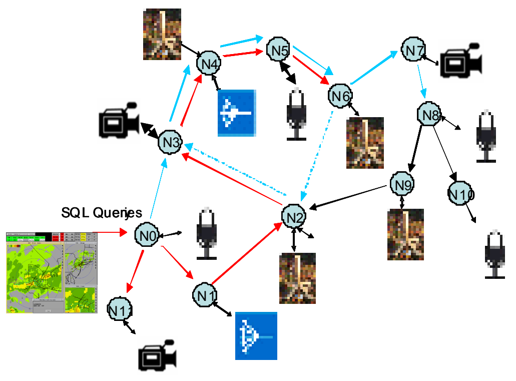

Figure3. *The Gaian Database Concept Sensor Network as a Gaian Database*

III. CURRENT IMPLEMENTATION

Our current implementation of a GaianDB has been built on top of the Derby database engine [5]. The choice of Derby was dictated by its availability as an 'open source' Java RDBMS that implements a VTI. It also has the additional advantage of having a small software footprint. The implementation of a GaianDB using the Derby RDBMS allows for the possibility of vertices automatically connecting to other vertices. The federation capability is realized in Derby by extending Virtual Table Interface (VTI) facility to accommodate data sources other than the Derby tables (Figure 4).

The following lists some of the main features of the implementation:

- Extended federation capabilities using VTIs for heterogeneous federation (RDBMS, Files, in-memory tables) with semi-automatic schema resolution;
- Re-propagation of queries from vertex to vertex;
- Loop detection/handling – no duplicates of queries or data;
- Depth first or breadth-first query propagation across network (multithreaded) with control over depth of query propagation;
- Distributed management of logical table definitions;
- Error tolerance: producing partial results when errors occur;
- Warning when incomplete result sets are returned;
- Integration with Messaging & Service Orientated Architecture (SOA) environments;
- Performance enhancement by enabling federated data to be cached in memory;
- Annotation of logical tables (e.g. providing vertex identity, network partitioning and query routing using a proprietary method call 'constant columns');
- Forward routing of queries through vertices that do not contain the logical table definition specified in the query(to minimize error handling costs);
- Distributed localised joins (as opposed to network joins) using sub-queries;
- Implementation of an automatic connection protocol (described in Section III).

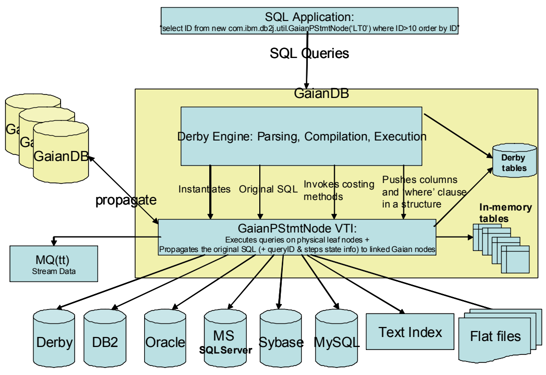
SQL Application:
'select ID from new com.ibm.db2j.util.GaianPStmtNode('LT0') where ID>10 order by ID'
SQL Queries
GaianDB
Derby Engine: Parsing, Compilation, Execution
Derby tables
GaianDB
propagate
Instantiates
Original SQL
Invokes costing methods
Pushes columns and 'where' clause in a structure
In-memory tables
GaianPStmtNode VTI:
Executes queries on physical leaf nodes +
Propagates the original SQL (+ queryID & steps state info) to linked Gaian nodes
MQ(tt)
Stream Data
Derby
DB2
Oracle
MS SQLServer
Sybase
MySQL
Text Index
Flat files

Figure 4 Schematic of Gaian Database Vertex

The database has been tested on the following RDBMS databases:

- Derby 10.3
- DB2 8.x,9.x
- ORACLE 9.x,10.x
- SQLServer 8
- MySQL 5

VTI interfaces have also been defined for:

- MQtt Microbroker
- CSV files (excel spreadsheets or data exported from other data sources)
- REST api (e.g. Omnifind Yahoo Edition Text Index [12])

The GaianDB Derby SQL engine can leverage the complete SQL ANSI syntax including but not limited to:

- DISTINCT
- INNER and OUTER JOINS
- GROUP-BY
- HAVING

We also support the full set of Derby functions:

- Aggregation (sum, average, count)
- substring
- substring(CLOB)
- trim
- cast
- like

The following data types are fully supported:
CHAR, VARCHAR, LONGVARCHAR, DECIMAL, NUMERIC, INT, BIGINT, SMALLINT, FLOAT, REAL
BINARY, VARBINARY, LONGVARBINARY, CLOB, BLOB ,DATE, TIME, TIMESTAMP

IV. VERTEX CONNECTIVITY

The efficiency of this database concept is in part determined by how the vertices of the database are connected together. One approach [9] seeks to minimise the distance between any two vertices in the database (Graph diameter) whilst retaining some control over the number of connections at each vertex (vertex degree). We have also investigated ways of constructing the network through a process of graph combining operations [10], which seek to minimise the resulting network diameter and minimise the vulnerability of the network to attack from the removal of vertices and edges. These techniques are particularly relevant when initially separate GaianDB networks are to be joined together.

Our initial approach to the problem was to make use of a network topology that is formed as a scale free random graph. Scale free graphs exhibit a number of desirable properties such as a predictable bound on the graph diameter and the distribution of vertices and edges.

The dynamic construction of such networks requires that when a new vertex is added to the graph *m* new edges are added one at a time to existing graph. One approach to the creation of scale-free random graphs is to add *m* edges e_i from a vertex *u* to vertex v_i is made with a probability proportional to the degree of v_i. ($1 \leq i \leq m$). According to Bollobas & Riordan (2004) [1], 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 $m > 1$ the graph diameter scales as
O{log(N) / log log(N)}. Typically $m = 2$ or 3.

If a physical network topology was to be created in this way, then each connecting vertex would be required to first establish communication with the network. The vertex would then query all vertices to determine their degree and then determine probabilistically with which vertices to establish connections. This requires multiple queries to be performed and an (inefficient) serial computation to be performed by the connecting vertex.

We have developed an efficient distributed method for adding a vertex to the network that does not require any queries to be issued from the new vertex. In this method the new vertex establishes communication with the network and announces its presence to the other vertices using some broadcast mechanism. Vertices respond to the broadcast in a time that is inversely proportional to a local fitness function F(*v*) and the joining vertex creates edges to the first *m* vertices to respond.

The current implementation of the GainDB is a Java application and therefore we use the Java Database Connectivity protocol (JDBC) to connect to the other databases and vertices. This means that we are currently limited to using TCP/IP for the communication protocol. Using this protocol we have implemented the broadcast mechanism using an IP multicast. The fitness function F(*v*) is equal to the degree of the vertex. To determine the maximum period of time t_v that an existing vertex will wait before attempting to connect to the new one is given by:

$$t_v = t_0 / F(v)$$

where t_0 is a constant used by every vertex.

The actual delay time t_d is determined by randomly selecting a time between 0 and t_v. Vertices with the higher degree preferentially respond faster than vertices of lower degree.

As vertices connect to the network they record the *m* edges by which they connect. If one of these edges is subsequently lost, the vertex to which it is incident is responsible for re-establishing a new edge using the broadcast protocol. When any of the other edges that the vertex maintains are lost then no action is taken since these are the responsibility of other vertices.

We are also continuing to investigate other fitness functions and mechanisms for inter-connecting the database vertices. Several different types of graphs have been studied in connection with modelling networks as graph decompositions. Among them are Cayley graphs, hypercubes and Hamming graphs (generalization of hypercube). We are seeking to develop an efficient algorithm to decompose large networks into partial joins. Knowing the diameters of the partial join components and the way they are interconnected would allow for computing the diameter of the network [9]. The inverse problem of building a network by combining pieces by partial joins is also interesting if it is possible to engineer or guide the interconnections. One obvious observation is that a complete interconnection pattern would give the lowest diameter. Other interconnection patterns may also turn out to be important.

V. QUERY MECHANISMS

In our database model a query propagates to all vertices that can satisfy the query (flood query) and then aggregates the results back along a path to return a single result set at the querying vertex. Since we do not know where the sources of data that match the query are located, a flood query of this type will always produce the shortest delay (if we ignore the overhead of generating the flood query in the first instance). The technique provides insurance against vertex or link failure since there are multiple paths to the data, a property that is highly desirable for the sorts of networks that we are considering.

Once the path(s) to the data are determined in response to a query from a given vertex, it is possible to route future queries along the same paths. We are undertaking a study in which we compare the theoretical prediction of performance against the actual performance achieved by implementing this type of persistent query mechanism. We are also investigating algorithms to minimise the number of paths traversed where the same query is being issued from different vertices.

As noted earlier the GaianDB has the ability to perform distributed queries over a network of sensors. To demonstrate this we have used the temperature, battery and accelerometer sensors that are available on many laptops. Each laptop is provided with an application that monitors these sensors and publishes the measurements into a local GaianDB database table. This table is then accessible from all other vertices in the network. Each vertex can now query its local logical table and obtain the data from all other vertices.

Figure 5 shows a graphical representation of a snapshot of the state of the sensors from a network of 16 sensors. We have also developed a method for gathering distributed statistics on each query and corresponding result-set. These statistics can be used to 'explain' how queries and results are propagated through the network of vertices.

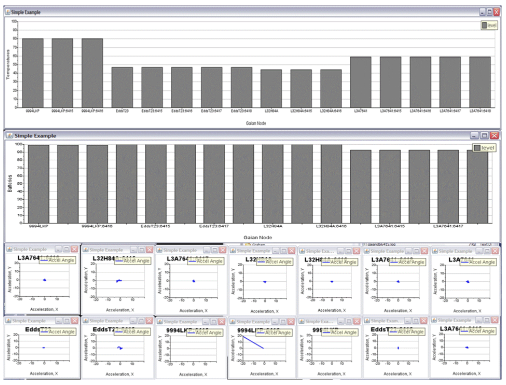

Figure 5. Display of current laptop sensor values (accelerometers, temperature and battery level from a 16 sensor GaianDB.

Figure 6 illustrates a method for visualizing the statistics by showing the paths followed by a flood query and the route by which data are returned in response. In the figures, dotted arrows indicate that no results were returned because the query had already reached the vertex at an earlier time via another route. The red arrows indicate paths where the query arrived by a shorter path but where the same query had already arrived by an alternate longer path. The numbers indicate the number of rows returned along each path.

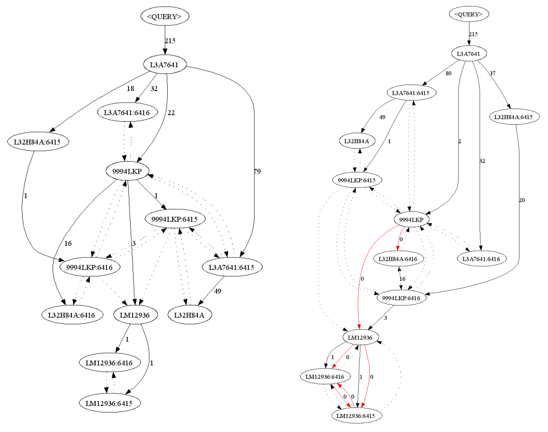

Figure 6 . Graphs showing the routing of consecutive queries and the paths over which results are obtained across a 12-vertex Gaian Database.

It is interesting to note that the paths followed to deliver results vary dramatically from query to query. This indicates how the different loadings within the network can influence the query routing.

Our investigations to determine how different sensor processing algorithms can be implemented using the distributed network paradigm has currently been limited to using distributed SQL to aggregate and filter. We have begun work on the use of distributed stored procedures to provide sensor processing services.

We have also continued our investigations into the characteristics of the inter-network and have developed a simulation tool for creating networks using different connection protocols.

IV NON-SENSOR APPLICATION – DISTRIBUTED SEMANTIC SEARCH

The GaianDB concept makes no assumptions about the type of data that are being federated and therefore it has many applications other than as a distributed sensor database. As a part of the International Technology Alliance (ITA) program, an investigation into semantic web technologies for information representation is being undertaken [13]. This investigation has studied techniques for extracting structured information from unstructured text, making use of the Apache Unstructured Information Management Architecture (UIMA) [16]. The extracted structured information can be stored in a common relational database structure based on the Resource Description Framework (RDF) triple concept [14]. A search over the extracted triples possibly combined with a corresponding 'key word' search of a text index provides a powerful way of extracting semantic information.

To demonstrate distributed semantic search using a GianDB we have created a VTI interface to the text index of a proprietary document search engine. The search engine we have chosen is the Omnifind Yahoo Edition [12] which contains an embedded UIMA pipeline. The UIMA pipeline has been configured to extract specified entities (e.g. people, places, organizations) and relationships between entities (e.g. person A belongs to organization B) and to store these in a GaianDB as triples.

The architecture of a semantic query database associated with a single vertex is illustrated in Figure 7.

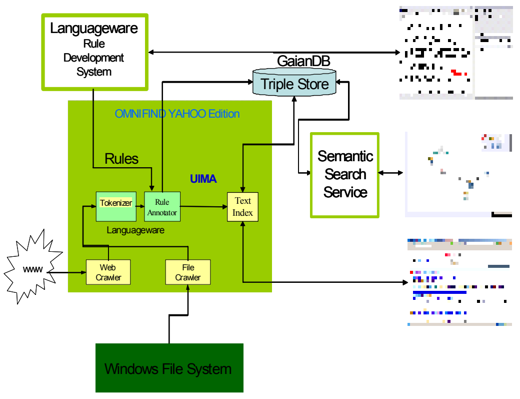

Figure 7 Semantic Query Database Architecture

Connecting these semantic query databases enables in a GaianDB allows the system at each vertex to crawl and index different document collections. A semantic query issued at any vertex retrieves results from across all the interconnected triple stores and text indices as shown in Figure 8.

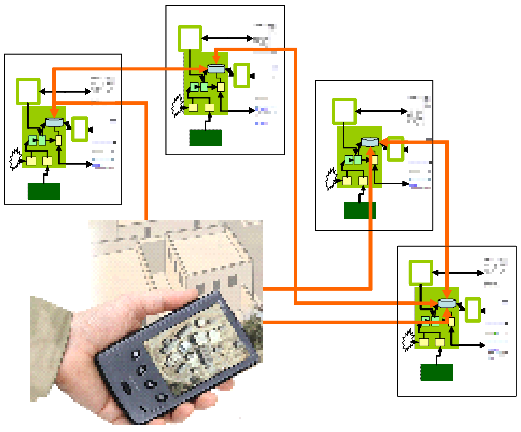

Figure 8 Semantic Query Vertex Architecture

Key to the efficient processing of these distributed queries is the ability to instruct the GaianDB to perform local join operations using the sub-query mechanism referred to in section III. We are currently investigating other types of distributed triple store using the RDF query language SPARQL rather than SQL and using ontologies and rule based methods to perform distributed inference across a GaianDB.

VI. DISCUSSION AND CONCLUSIONS

The paper introduces a new concept of a dynamic distributed federated database (DDFD) which can be used in support of a range of applications required for networked operations. The key innovation is that data stored at any location in the DDFD network can be accessed from any other location.

The DDFD model has a number of advantages over current approaches to the integration of distributed data. It is capable of integrating data from a wide range of heterogeneous sources using a relational database model. It does so by creating a common virtual interface that can be used to map queries to the data structures of any specific source. A noteworthy advantage of this system is that application requiring access to the different data sources need only interact with a single database process.

In our current implementation of a DDFD we have demonstrated the applicability of the model to a networked sensor database and to a distributed semantic search application. The software required to run the system at a given location is approximately 4Mb (excluding the Java Virtual Machine) , which allows the system to be used on a wide range of devices.

A number of outstanding challenges need to be addressed. The scalability of the GaianDB model in real networks needs to be demonstrated. In addition some types of query operations cannot be efficiently performed in a distributed environment. This is a consequence of the limitations of the structured query languages (SQL) typically used in relational databases. Extensions of SQL that allow for taking full advantage of the distributed environment are needed.

Security and policy are also major challenges for further development of the GaianDB model in particular distributed encryption schemes based on identity management seems a promising approach to the problem of secure communication in this context.

The Gaian database approach to processing information in a network of sensors is radically different from traditional approaches such as the messaging paradigm. Information from sensors in the GaianDB model is made accessible by storing it in databases which are queried by those who require the information. Accomplishing this method of distributed sensor information processing requires new methods of managing queries to balance the traffic flow across the network.

We have begun to address these challenges and have developed promising avenues for resolving them in future research.

VI. REFERENCES

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

[2] 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.

[3] F. Bergamaschi, et al. “A Distributed Test Framework for the Validation of Experimental Algorithms Using Real and Simulated Sensors” ACITA, September 2007.

[4] **J.** Dean, S. Ghemawat, “MapReduce: Simplifed Data Processing on Large Clusters”, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

[5] Derby Database http://db.apache.org/derby/ (last accessed 3rd May 2008)

[6] R. Govindan et al. “The Sensor Network as a Database” - ICSI (2002)

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

[8] HadoopMapReduce http://hadoop.apache.org/core/ (last referenced 3rd May 2008)

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

[10] A. Mowshowitz, V. Mitsou and Graham Bent, “Models of Network Growth by Combination”, to be published in ACITA, September 2008.

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

[12] Omnifind Yahoo Edition, http://omnifind.ibm.yahoo.net/ (last accessed)

[13] Project 12, International Technology Alliance, http://www.usukita.org.

[14] Resource Description Framework (RDF), http://www.w3.org/RDF/ (last accessed 3rd May 2008)

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

[16] Unstructured Information Management Architecture, http://incybator.apache.org/UIMA, (last accessed 3rd May 2008)

[17] Yong Yao et al., “The Cougar Approach to In-Network Query Processing in Sensor Networks”, SIGMOID Record, Vol. 31, No 3, September 2002

ACKNOWLEDGEMENT OF SUPPORT AND DISCLAIMER

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 authors 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.