

A Scalable Service Architecture for Distributed Search

Mark Jessop, Andy Pasley, Jim Austin

Advanced Computer Architectures Group

Department of Computer Science, University of York, Heslington, York, YO10 5DD, UK.

[mark.jessop, arp, austin]@cs.york.ac.uk

Abstract

The “Grid” provides the ability to bring together diverse tools and data into a single virtual environment, increasing the availability to a user of distributed data that requires pattern matching and/or searching algorithms performed against the full dataset. A service-based solution is evaluated that provides a generic architecture for distributed search. The ability of the system to scale to data distributed across many locations is analysed and it is compared with proprietary solutions to estimate the performance overhead of the Grid protocols.

1 Introduction

The Distributed Aircraft Maintenance Environment (DAME) project [1] has demonstrated the use of the Grid to implement a distributed decision support system for deployment in maintenance applications and environments, with particular interest in the field of Rolls-Royce aero-engines. A core part of the diagnostic process supported by DAME is the searching against large distributed datasets for vibration patterns measured during engine operation, which provides the focus for this paper.

If an anomaly has been observed in an engine’s vibration data, a vibration signal of interest is identified. A vital part of the diagnostic process is a search against historical flight data using this signal of interest. The aim is to find vibration patterns that are ‘similar’ to the anomaly just observed. This is an example of a ‘query by content’ or ‘query by example’ problem, which is a significant area of data mining research [2, 3, 4]. It can be stated as finding a set of time-series sub-sequences R that are ‘similar’ to a query time series q :

$$R = \{x \in X \mid d(x, q) \leq \tau\}$$

where $d(x, q)$ is the distance between two patterns, x and q , τ is the maximum distance (using an appropriate metric) that a vibration pattern can be from the query and be considered a match. Where τ is varied to return a set of a fixed number (k) of matches in R , this is referred to as a k-nearest neighbour (k-NN) search.

The results of the search are a number of matches to the query pattern, each match relating to a specific engine flight. Further data associated with these

engine flights is available from other services within DAME (e.g. Service/Maintenance history) and is used to assist the diagnostic process [5].

The project has implemented a ‘proof of concept’ demonstrator system on the White Rose Grid [6], using a solution that separates the distributed nature of problem from the searching and pattern matching problem, thus providing re-usable generic components.

2 A Service Architecture

Traditional solutions to these types of problem move the data to the processing. The decision was taken to distribute the processing out to the separate data locations. The logic behind this reasoning is not discussed here but follows a similar pattern to that provided in [7], which for a more general range of problems recommends that to combine data from multiple sites “one should push as much of the processing to the data sources as possible in order to filter the data early”. The architecture was developed with this aim in mind.

The resulting service architecture is detailed in [8] and was previously presented to this forum. Every location where data is observed and stored is considered a ‘node’. For a DAME deployment, each airport data repository will be a node. For the purpose of this architecture definition, each node will be treated as a single resource. In practise, it is likely that each node will utilise many resources, ranging from high performance clusters, tape archives, desktop PCs and laptops. The key assumption made is that the communication bandwidth available between resources at a single node is significantly higher than that available between nodes.

The key component is the Pattern Match Controller (PMC) Service. An instance of the PMC resides at every data location (node) and is responsible for all inter-node communication. It is also the front-end service that receives requests from client applications. The PMC is a generic service that can receive and distribute search requests without understanding of the data or problem domain. It is implemented as a Java Grid service (using Globus Toolkit v3 α) and hosted within a Tomcat installation.

At each node the same suite of component services reside:

- The front-end service - PMC. This service receives search requests, propagates them to other nodes within the system, collates results and returns them to the client.
- A Pattern-Matching Service (PMS) that communicates solely with the other services/data located at that node. This service provides the implementation of the search functionality.
- A data management service that provides a single 'virtual' view of the distributed dataset(s).

The PMS is responsible for implementing the search across the data held at a node. It can use whichever appropriate pattern-matching and/or indexing algorithms it has available, as specified by the user. A PMS receives search requests from its local PMC and upon completion of the search sends the results back to the PMC its role in the search is then finished. Where there is no relevant data to search against at a node, the PMS simply returns an empty result set.

Any PMC within the system can receive a search request. A search request consists of the query pattern to be searched for in the stored data and additional information including the number of results required (i.e. the value k in a k -Nearest Neighbour search). The node that receives the request becomes the 'master' node for that search. All other nodes in the system are referred to as 'slaves' for that search. Subsequent requests may be received by other nodes, each node acting as the master for search requests they have received.

It is the responsibility of the master PMC to replicate the search request to all slave PMCs. The search request is also passed to the PMC's local PMS. The master PMC then retains a results set (initially empty) which is updated as the local PMS or slave PMCs return results. At any time the current results set may be requested, which the master PMC returns to the calling client.

Slave PMCs have a simpler responsibility. After receiving a search request from a master PMC, the request is passed to its local PMS. Once the search is completed, the PMS passes the results for that node to the slave PMC instance. The slave PMC then passes the results to the master PMC and its role in the search is finished.

3 Evaluation

The aim was to devise an efficient and scalable solution for distributed search. An investigation was carried out into the performance of the architecture and the number of nodes it could realistically utilise. A simple theoretical model has been identified for estimating the search time based upon the workload and number of nodes data is distributed across.

The DAME demonstrator has been used to implement timing experiments and provide measurements for various parameters of the model. The use of the Globus framework is compared to two alternative native implementations of the PMC:

- In C++ as a web service (using gSOAP [9]).
- In C++ using raw sockets for the communication.

From this, the overhead of Java and the Grid protocols is measured and its effect on the scalability of the architecture estimated.

3.1 Modelling the System

The system implements behaviour similar to the manager-worker model described in [10]. To ease the modelling of its performance (with regard to search time, efficiency etc) the following assumptions are made:

- Each node has the same specification, and the data is distributed equally among them.
- The pattern-matching algorithm implemented, is fully parallel, i.e. where the search problem is split equally between n nodes, each node completes its search in $1/n$ time in comparison to a single node.
- That every node is an equal distance from the node selected as the 'master' with identical network connections.
- Clients ask for no more than 100 results from a search.

The first two assumptions are not as restrictive as might first be thought. Query by Content and many other search problems can be implemented in a fully parallel manner. Where the data is not distributed evenly it is possible to vary the specification of the nodes so that each node can complete a search against

its data in similar times. Under such circumstances, the analysis in the following section still holds.

The model developed in this section investigates the performance in performing a single search. Under the condition that there are no previous searches requiring processing, the search time is:

$$\text{Search Time} = \frac{\text{Total Work (in time)}}{\text{Nodes}} + \text{Network Overhead}$$

The overhead is made up of two components a parallel part and a serial (or Amdahl) part. The parallel part has been implemented in parallel and remains constant as the number of nodes changes. The serial part must be carried out for each node and is proportional to the number of nodes utilised. The search time t can therefore be written as

$$t = \frac{w}{n} + on + c, \text{ where}$$

- w = total amount of 'work' to be done, the time to process the search across all data.
- n = number of nodes.
- o = serial part of the overhead, the time incurred sending search requests to slave nodes.
- c = a constant, the parallel part of the overhead that is processed concurrently.

c includes the decoding of the search request by the slave PMC, all processing and network time to send the request to the PMS. After the search is complete, it includes any preparation of the results that is required and their delivery back to the slave PMC.

More influential to the search time is o . This is the network time to send requests to the slave nodes. At present it also includes the preparation of the search request (the majority of this is encoding into SOAP) as this is repeated for every slave node. Ideally, the encoding would be done once but this is not possible within the Globus framework. It should also include the time taken to merge the results with the overall result set, but after a few measurements this was considered a negligible influence relative to the network time and discounted.

The replication of the search request to all nodes requires $n-1$ messages. Some slaves receive requests before others and can initiate their search earlier. Under a reasonable assumption, by the time that the last PMC receives its results, all other nodes will have already sent their results to the master node and merged them. Therefore a total number of n communications between nodes is required extra to the basic search time.

The behaviour of such a system is easy to analyse. In the case where $c = 0$ and $o \geq w/2$, the search time drops initially as nodes are added. There is an optimal number of nodes where the quickest search is obtained, adding additional nodes at this point increases the search time. Eventually sufficient overhead can be introduced that the search time exceeds that of using a single node. The optimum number of nodes is $n = \sqrt{w/o}$ which provides a speed up of $n/2$ over a single node.

The ratio w/n is referred to in [10] as the 'loading factor'. As w and n increase, if $w \gg n$ then the system maintains a high efficiency and will not reach a point where the addition of nodes is associated with an increase in the search time. This seems intuitive, the larger the problem the more nodes it can be distributed across before the benefit is outweighed by the communication overhead.

Given good estimates for w and o , it is easy to decide the optimum number of nodes that should be used. However, this architecture has been designed for problems where the distribution of the data cannot be controlled and moving data incurs a large cost (which may be insurmountable in the case of legal barriers). The ability of the architecture to scale to many nodes is mostly dependent upon providing a low serial overhead o .

3.2 Performance

The model described in the previous section, incorporates a value for the serial part of the overhead of using this Grid-enabled service architecture. The ability of the system to scale to a large number of nodes depends heavily upon this value. This section provides the results of timing experiments designed to measure this overhead and explores the additional cost of using the Grid framework instead of providing a proprietary solution designed solely for speed. These measurements are then used to model the performance of the system with regard to the time taken to perform a search.

3.3 Alternative Implementations

The PMC has been developed within the Globus framework and installed on several White Rose Grid nodes. It was expected that the use of Globus protocols had increased the communication overhead over what would be encountered by other implementations. To measure the impact of this on the performance and scalability of the system, two alternative native implementations of the PMC were created for comparison.

- As a native C++ Web Service using gSOAP [9].

- Using a proprietary encoding and raw sockets for communication. The socket implementation was designed to be as efficient as possible at the expense of the flexibility provided by Grid and web services.

For the socket solution, it was possible to develop the PMC so that the encoding of the request was only carried out once. This moved the encoding from the serial to parallel part of the communication overhead, reducing search times. Also, it was no longer necessary to encode the request using SOAP, providing smaller data packets and therefore reducing network time.

3.4 Overhead Measurements

For the purpose of measuring the serial overhead all three PMC implementations were hosted at each node. The specifications of these machines are detailed in Table 1.

The PMC instances in Sheffield and Leeds were used to invoke a search (from master to slave using the NodeSearch API call) at York PMC. The time taken to perform this operation was timed. For the Grid and web service versions, invoking a search was considered an atomic operation, and included all encoding/decoding and network connection/transmission time. For the raw socket implementation, the time to encode the request was discounted due to it belonging to the parallel part of the overhead.

| | York | Leeds | Sheffield |
|-----------------------|-----------------------|-----------------------|---------------------|
| Machine Type | Sun V880 | Sun V880 | PC |
| Processor Type | 900 Mhz UltraSparcIII | 900 Mhz UltraSparcIII | 1 Ghz Intel Celeron |
| Processors | 8 | 5 | 1 |
| RAM | 24GB | 15GB | 128MB |
| OS | Solaris 9 | Solaris 9 | Linux Red Hat 8.0 |

Table 1: Node Specifications

This process was repeated three times in succession for each client, with the entire process being repeated two more times throughout the day. The results are shown in Table 2, which includes an overall time for each implementation. These measurements allow comparison between the different implementations. They show that the use of Globus provides a much larger communication overhead than the alternate solutions. It is expected that this is due to the parsing of XML in Java, and a similar overhead could be expected from any Java implementation.

It should be highlighted here that all these implementations do not incorporate the Globus security framework [11]. It is expected that including security to the Globus service would add significantly to the overhead measured. Alternative implementations would also take a performance hit from the use of security mechanisms.

| Implementation | Globus Grid Service | | C++ Web Service | | Raw Socket | |
|-----------------------|----------------------------|------------------|------------------------|------------------|-------------------|------------------|
| Site | Leeds | Sheffield | Leeds | Sheffield | Leeds | Sheffield |
| Time (ms) | 1296 | 1013 | 11 | 30 | 4 | 9 |
| | 1392 | 1245 | 9 | 20 | 4 | 9 |
| | 1181 | 1152 | 9 | 21 | 4 | 10 |
| | 1105 | 1025 | 10 | 22 | 5 | 10 |
| | 1251 | 1235 | 9 | 20 | 4 | 9 |
| | 1199 | 1179 | 9 | 20 | 4 | 9 |
| | 1305 | 1182 | 10 | 21 | 5 | 11 |
| | 1185 | 1315 | 9 | 21 | 4 | 9 |
| | 1235 | 1015 | 9 | 20 | 4 | 9 |
| Average | 1195 | | 15.556 | | 6.833 | |

Table 2: Results of measuring the serial part of the overhead

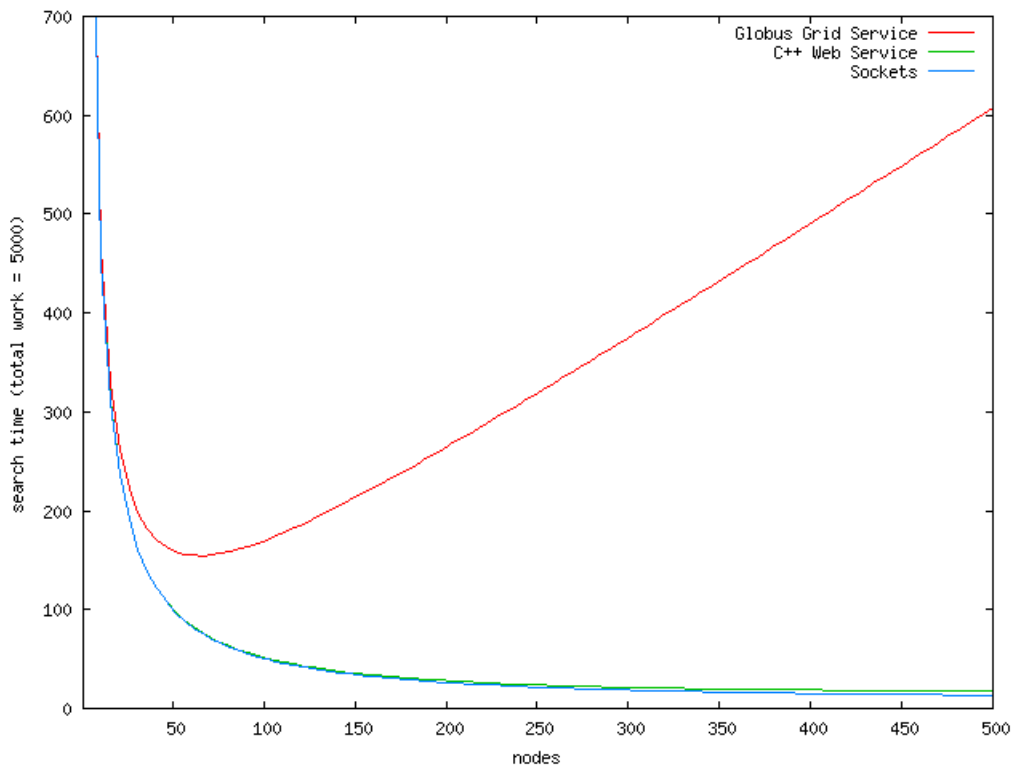


Figure 1: The search time for various PMC implementations for a static workload as the number of nodes is varied.

3.5 Search Performance

Using the estimates for the overhead given in the previous section, the search time for particular problem sizes can be modelled. For all subsequent analysis $w = 5000$ is used. Exact values of the total work required depend upon data volumes, available hardware and algorithms used to implement the searching, but this is considered a realistic value that DAME will require. *Figure 1* shows the search times as the number of nodes is increased for all three frameworks investigated. It can be seen that initially the search time falls for all implementations. The Globus implementation quickly reaches a point (at 65 nodes) where the overhead exceeds the time spent ‘working’ and the search time begins to increase. Although not shown in *Figure 1*, in contrast the C++ Web Service and Socket implementations achieve their lowest search times at approximately 560 nodes and 860 nodes respectively. Where the data is distributed in this many locations the difference between the Globus implementation and the others is extremely large. This difference is likely to make the Globus implementation exceed time requirements where the other implementations would be considered viable.

3.6 Hierarchical Organisation

Under the current model of communication between nodes, the overhead is proportional to the number of nodes used. An alternative is to organise the nodes in a hierarchy, allowing concurrency in the delivery of

requests to all nodes and the merging of results at several nodes before they are delivered into a full result set at the ‘top’ of the tree. Theoretically, the complexity of the search time could be reduced from $\Theta(w/n + n)$ to $\Theta(w/n + \log n)$. However, this is dependent upon the network topology and organising a suitable hierarchy.

Organising nodes within the architecture described here, would introduce additional complexity to the system, especially if maintaining the property of no single point of failure. At present, each node has two simple roles it must be able to perform, master or slave, that require only knowledge of which other nodes must be communicated with. Under hierarchical organisation, each node would need to be aware of the hierarchy, which would be heavily influenced by the network topology and geography. Functionality such as allowing any node to receive search requests could be difficult to provide efficiently. Current research is evaluating different ways a hierarchy could be implemented.

Under the assumption that organising nodes in a hierarchy does not add any further overhead to the required processing and communication, the performance of the system can be estimated. *Figure 2* shows how hierarchical organisation under the different frameworks compares to the best system reported in *Figure 1*, a network of ‘peers’ communicating via sockets. It shows that as the number of nodes increases a hierarchical network implemented using Globus will eventually (if more than 1200 nodes are used) outperform that of a peer

network. Where the other frameworks are implemented using a hierarchical structure they continue to outperform the Globus implementation, but show less improvement over it. The use of hierarchies has reduced the influence of the overhead on search times, making the use of Globus technology a much more realistic possibility.

4 Conclusion

A service-based architecture for searching and pattern matching has been detailed. It provides generic and heterogeneous functionality and has been designed to provide a robust and scalable method of searching large scale distributed data sets. The distributed nature of the data is hidden from users and developers of client applications.

The performance of the system has been modelled and the effect of using Grid services evaluated. The results showed that the use of Grid services restricts the performance and scalability of the distributed architecture. The use of a hierarchy has been identified as a future extension to this work that could improve the scalability and reduce the influence of the Grid/Java overhead.

Acknowledgements

This project was supported by a UK eScience Pilot grant from EPSRC (GR/R67668/01), as well as support from our industrial collaborators Rolls-Royce, Data Systems and Solutions, and Cybula Ltd.

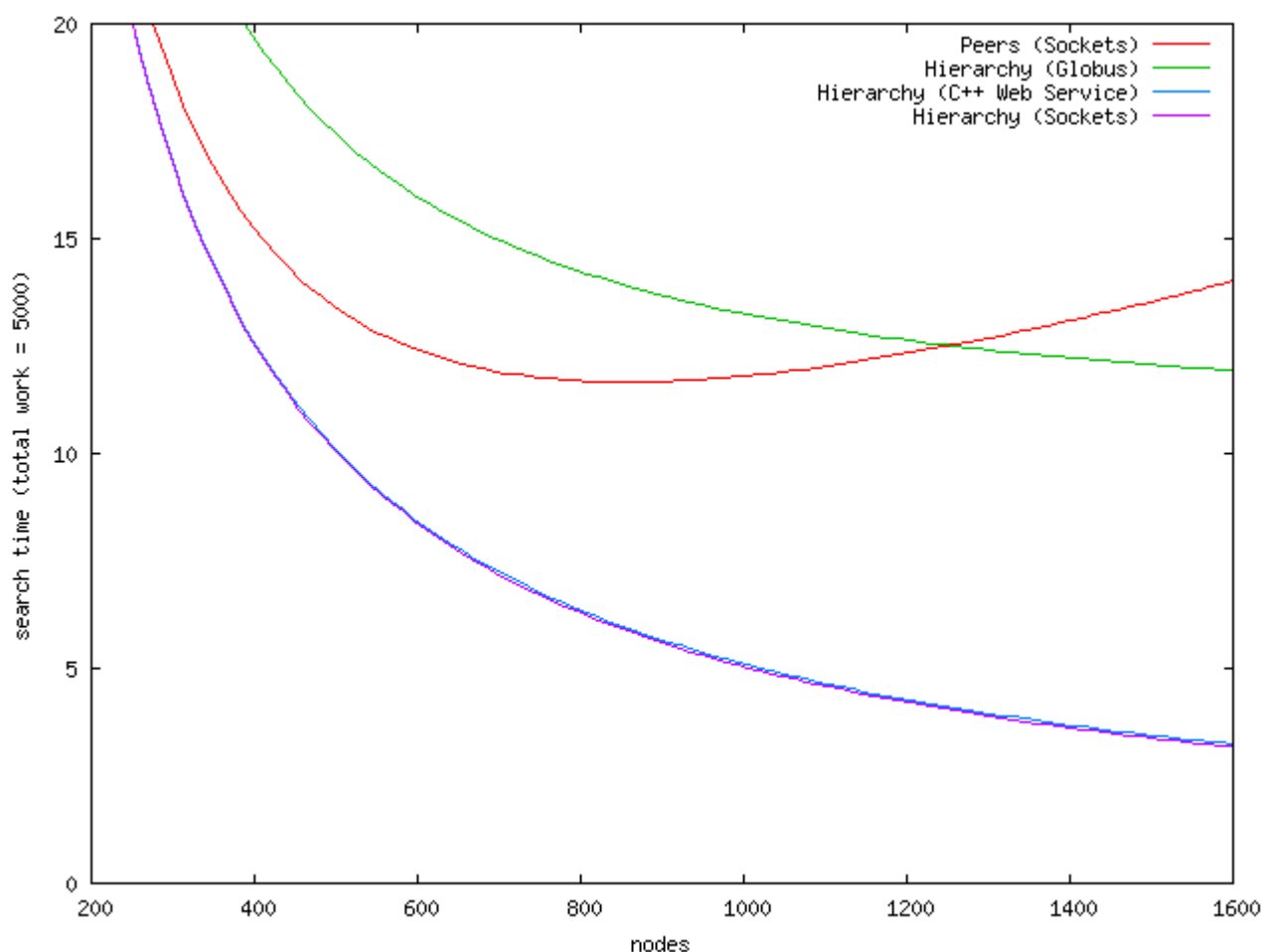


Figure 2: Performance of a hierarchical PMC network in comparison to a network of peers.

References

[1] Jackson, T., Austin, J., Fletcher, M. and Jessop, M., "Delivering a Grid Enabled Distributed Aircraft Maintenance Environment (DAME)", Proc. UK e-Science All Hands Meeting 2003, Nottingham, UK.

[2] Agrawal, R., Faloutsos, C. and Swami, A. 1993. Efficient Similarity Search in Sequence Databases. Proc. 4th Int. Conf. Foundations of Data Organization and Algorithms (FODO). pp. 69-84.

- [3] Keogh, E., Chakrabarti, K., Pazzani, M. and Mehrotra, S. 2001. Dimensionality Reduction for Fast Similarity Search in Large Time-Series Databases. *Knowl. Inf. Syst.*, vol. 3, no. 3, pp. 263-286
- [4] Keogh, E. and Kasetty, S. 2002. On the Need for Time-Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *Proc. 8th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*. pp102-111.
- [5] Ren, X., Ong, M. et al. 2004. Integrated Fault Diagnostics on the Grid. *Int. Conference on Engineering Complex Computer Systems (ICECCS'04)*. pp. 59-65.
- [6] Dew, P., Schmidt, J., Thompson, M. and Morris, P., "The White Rose Grid: Practice and Experience", *Proc. UK e-Science All Hands Meeting 2003*, Nottingham, U.K.
- [7] Gray, J. 2003. *Distributed Computing Economics*. Microsoft Research, San Francisco, California
- [8] Jessop, M., Pasley A. and Austin, A., "Pattern Matching Against Distributed Datasets", *Proc. UK e-Science All Hands Meeting 2004*, Nottingham, U.K.
- [9] van Engelen, R. and Gallivan, K. 2002. The gSOAP Toolkit for Web Services and Peer-to-Peer Computing Networks. In *Proceedings of IEEE CCGrid Conf.*
- [10] Almasi, G. and Gottlieb, A. 1994. *Highly Parallel Computing*. Benjamin/Cummings Publishing Company.
- [11] Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L. and Tuecke, S. 2003. Security for Grid Services. *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, IEEE Press, to appear June 2003.