

Pattern Matching Against Distributed Datasets

Mark Jessop, Andy Pasley, Jim Austin

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

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

Abstract

Aero-engine vibration and performance data is downloaded each time an aircraft lands. On a fleet wide basis this process soon generates terabyte scale datasets. Given the large volume of data and the rate at which it is produced, there is a need to consider how the data is managed. In addition, there is a requirement for pattern matching against this engine data. The DAME project has investigated the use of Grid technology to support a fully distributed storage and pattern matching architecture. This paper describes how this technology can be used to solve the data management problem.

1.0 Introduction

The Distributed Aircraft Maintenance Environment (DAME) project [1] is concerned with diagnostics and maintenance, with particular interest in the field of Rolls-Royce aero-engines. The focus of the project has been the analysis of vibration data from aero-engines in order to undertake the diagnostic and prognostic processes. These engines typically produce around 1GB of vibration and performance data per flight. Therefore an aircraft carrying four engines will generate 4GB of data per flight. At the fleet level, terabytes of data are produced on a daily basis.

Through the use of an on-wing diagnostic capability, an abnormality may be detected in an aircraft's engine data. Part of the diagnostic process requires the searching for similar behaviour to the abnormality in engine data from other aircraft. The pattern match problem in this case is an instance of 'query by content'[8] which is a common data mining problem, and not unique to the application domain discussed here.

Pattern matching over engine data could be required to search against all the historical engine data stored within the DAME system. Due to the volumes of data and the rate at which it is produced, there is a requirement for careful data management and associated pattern matching.

2.0 Architecture

Given the amount of data involved, and the distributed nature of the data collection from aircraft, at airports, DAME has considered the distributed storage and search of data. The DAME data management system uses airports as the unit of distribution. All data that arrives at an airport is stored at that location and all processing and pattern match queries against that data are processed at that site. This results in data relating to any one engine being spread around the airports that it has visited.

The overall architecture is depicted in figure 1. Each airport node consists of four major elements, Pattern Match Control, Pattern Matching Services, a Data Repository and a Data Catalogue. All nodes share a single or federated meta data catalogue and it is this catalogue that contains the locations of the engine data. In the aero-engine scenario, the user interacts with the system via the Signal Data Explorer [4], a sophisticated tool for the manipulation of engine data and orchestration of searches. Any client application that understands the simple pattern match protocols that are described in this paper, may use the distributed system.

A pattern match is initiated from a single airport node that acts as the master node for that search. Any node can act as master and a client may use different nodes to initiate separate match operations. The master node distributes the search amongst all airport nodes, each of which performs a pattern match against the data stored

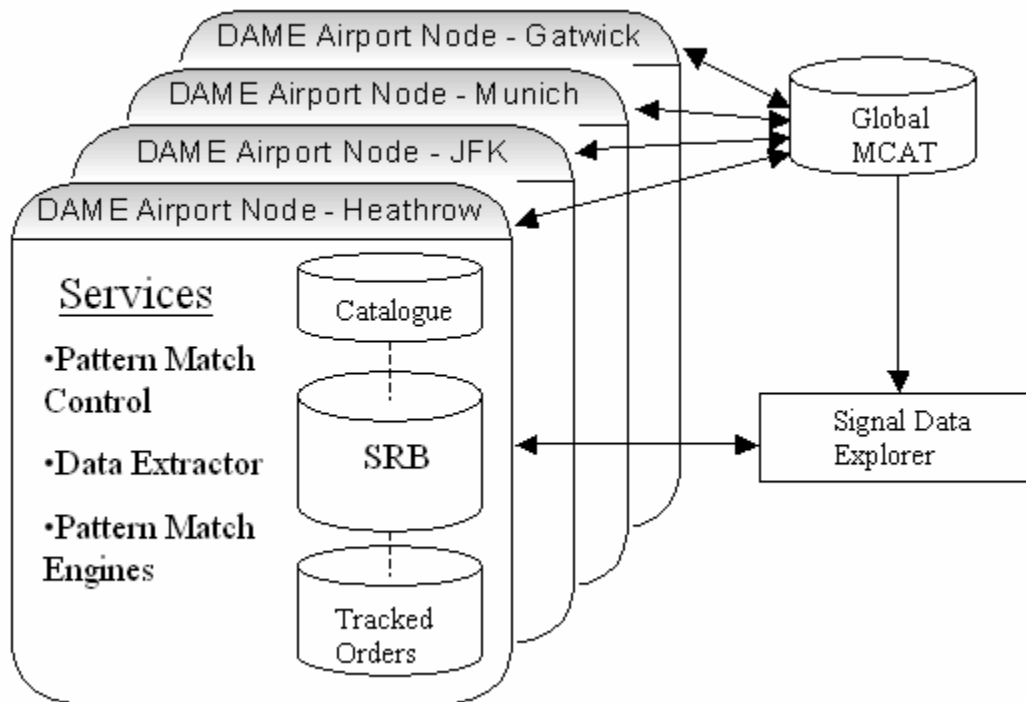


Figure 1: Distributed Architecture

in its local data repository. The individual results are then returned to the master node, where they are correlated and presented to users.

Client applications, such as the Signal Data Explorer, are able to poll the master node for results at any point in time to gain a snapshot view of the current search status. This also allows them to retrieve a partial set of match results for presentation to users and to get an idea of how a search is progressing. The master node maintains search results for a user specified period of time, usually many hours, before automatically cleaning up. Pattern match results are available to any user until this timeout expires.

2.1 Pattern Match Control

The Pattern Match Controller, PMC, is the front-end service for pattern matching operations across the data held at a single node. It accesses a catalogue of what searchable data is held at that node and is responsible for coordinating searches across remote nodes in the system.

In any practical system, failure is a possibility. If any PMC nodes were to fail, any search operations will not include results from these nodes. The result set will indicate the numbers of participating, failed and completed nodes in any single pattern match operation. This gives users an idea of the amount of data covered

in any single search. Once failed nodes become available again, they are once more able to participate in searches.

2.2 Pattern Match Services

The Pattern Match Controller communicates with several other services. An Engine Data Extractor service extracts specific low-level features, known as tracked orders, from raw engine data into a specified format.

The pattern match engine use AURA [2] technology, which provides a high performance neural network based pattern matching engine. The results from AURA consist of a set of candidate matches that are processed using conventional pattern matching techniques to form a result set.

Although a specific pattern match engine is described here, the architecture has been designed to be generic. This allows for the AURA pattern match engine to be replaced or accompanied by other pattern matching methods.

2.3 Data Repository

The Data Repository at each node is responsible for storing all raw engine data along with any extracted and AURA encoded data. This service is provided by the

SDSC Storage Request Broker, SRB [3]. SRB is a tool for managing distributed storage resources from large disc arrays to tape backup systems. Files are entered into SRB and can then be referenced by logical file handles that require no knowledge of where the file physically exists. A Meta Data Catalogue is maintained which maps logical handles to physical file locations. Additional system specified meta-data can be added to each record. This means that the catalogue can be queried in a data centric way, e.g. engine serial number and flight date/number, or via some characteristic of the data, rather than in a location centric fashion.

SRB can operate in heterogeneous environments and in many different configurations from completely stand-alone, such as one disc resource, one SRB server and one MCAT to completely distributed with many resources, many SRB servers and completely federated MCATs. In this configuration a user could query their local SRB system and yet work with files that are hosted remotely.

When data is requested from storage, it is delivered via parallel IP streams, to maximise network through-put, using protocols provided by SRB.

2.4 Data Catalogue

The Data Catalogue maintains a list of all the tracked orders and AURA encoded searchable data held at a single node. A PMC uses this information to determine if it needs to participate in a search request. It is expected that this service can be implemented by extending the meta data held in the SRB MCAT.

3.0 Implementation

The hub of the distributed pattern match architecture described in this paper is the PMC service and it is through this service that the power of the architecture is derived. In order to realise this power, the physical manifestation needs to be simple. This simplicity is encapsulated within an API, see figure 2, that describes the interfaces between client applications, PMC services and Pattern Match services. One of the aims of DAME is to build a generic framework for diagnostics. The API has been designed to support this behavior by not referring to any specific data types or locations. The API only specifies how components should interact with the pattern match architecture. This means that application specific components can be built and configured as required by particular implementations.

As a result of this generic behaviour, figure 2 includes a “Search Constraints Builder” component that has not been mentioned up to this point. This tool allows complex pattern match queries to be built. It talks in application specific terms and generates generic pattern match queries as outputs.

At no point is the implementation constrained to any single hardware or software platform. In any deployment it is likely that each component and/or node may be hosted on completely different platforms.

3.1 System Initialisation

All nodes in the system are represented by a PMC service that can behave as a completely stand-alone pattern matching entity, capable of matching against data stored at its location. The first PMC in a system is configured as a single entity, with no knowledge of any other nodes.

As new PMC nodes are added, each one is configured with the address of one other known PMC. When one of these nodes starts up it contacts the known PMC and requests a list of all nodes in the system. It then contacts each one in turn to register itself. From this point on all searches will include the new node.

All PMC nodes maintain a list of all other PMCs that are known to them via the registration process. If a node fails, its entry is kept in each node list maintained by each PMC. This allows search results to reflect the fact that the entire dataset is not being searched against. If a node is to be permanently removed, a de-registration process is invoked.

The PMC service is implemented as a Java GT3 Grid service [5] and hosted within a Tomcat 4.1.24 [6] installation.

3.2 Data Storage

As aircraft land, engine data is downloaded into the system. The raw data is stored directly into SRB on a local resource. Each location stores engine data under a logical file structure based on node location, engine type, engine serial number and flight information. Since all data is visible to any user regardless of their location, this allows users to see where data is coming from, and provide some physical location information where it is required. This path is then used as the location for all extracted tracked orders, to keep raw and processed data together.

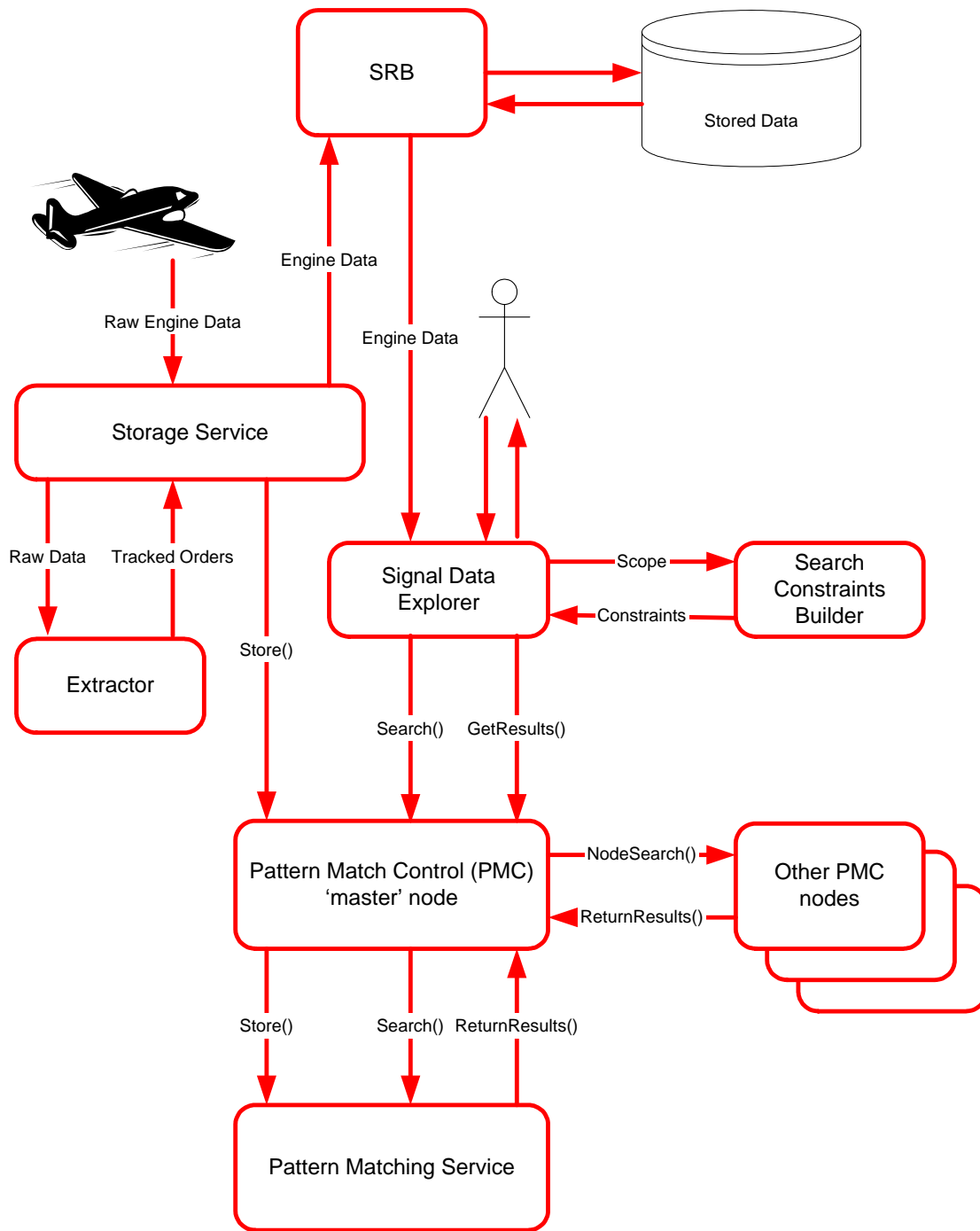


Figure 2:Implementation

The tracked orders are passed to the PMC service for storage. This allows the PMC to maintain a catalogue of what data is held at that location, allowing it to know whether it needs to participate in pattern match operations. The catalogue itself is stored within the SRB MCAT.

Current test implementations use a single Postgress [7] data base. With this set-up each airport's SRB installation is statically configured to use the known

MCAT. Real world implementations would use a production standard database, such as Oracle or DB2.

3.3 Searching

A client application formulates a pattern match query and chooses one PMC to act as master to process that query. The client may always use the same PMC, but is free to use any in the system. The query consists of a pattern to be matched against stored data and the

number of results that the user wants. The client application can also specify a measure of speed vs. accuracy for the search, a timeout period for the results set and a minimum score that represents the match quality that the results must meet.

When a pattern match query arrives at a master PMC node, it is passed down to the local search engine, and fanned out to all other PMCs. The master then waits for each PMC to respond with a result set. Those PMCs that contain no relevant data simply return an empty set.

As stated above the user is allowed to query the current search status at any point until the query and results are destroyed. Each result set consists of a set of SRB data handles with associated match scores and sorted from best to worst. The set will contain up to the maximum requested number of results.

As each PMC returns its results, the master PMC keeps and updates an order list of results ready to be returned to the client. Only those results that meet the client requirements are kept.

Since the result set consists of SRB logical paths, the client application can access SRB directly. This means that no data is passed around and the client application does not need to know, or be concerned with where the data is actually located. SRB's logical view allows PMCs and pattern match services to operate on a distributed data set, while other DAME services can treat the data as being held at a central repository.

Each API call places very little load on the network infrastructure and only when the user requests a specific match result does any data transfer take place. The transfer is managed by SRB which is best placed to efficiently transmit data.

4.0 Future Work

Once the implementation and testing process has been completed, there is a requirement to produce performance benchmarks. This would be in terms of comparing the performance of multi node searches against single nodes and attempting to assess the through-put of the system. This would include the number of realistic concurrent users.

It would be useful to assess the simple peer to peer nature of the system and try to ascertain at what point it breaks down, i.e. how many nodes and users can be supported before more time is spent in inter-node communication than in doing real work. The results of

this should lead to a set of peer to peer models for differing performance requirements.

The use of a single MCAT provides a single point of failure that would incapacitate the entire distributed system. To alleviate this, a federated MCAT would be used, allowing each node to use their own local MCAT. In large systems this would provide a performance benefit as all data requests would be made through local servers. However, this would require an efficient MCAT synchronization process.

As stated above a single MCAT provides a single point of failure for the entire system. Therefore, there is need to determine the performance benefits and issues of hosting many SRB zones, i.e. meta data catalogues.

5.0 Summary

In addition to the generic and heterogeneous properties of the architecture and API described in this paper, the architecture has been designed to provide a robust and scalable method of searching large scale distributed data sets.

The architecture exhibits graceful degradation in the face of node failures. When nodes are not able to contribute to search operations, users still receive results from the remaining airports and are made aware of the search coverage.

The addition of extra airports into the system does not place any additional work load onto existing nodes and this makes the system highly scalable.

7. 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.

References

[1] Jackson, T., Austin, J., Fletcher, M., Jessop, M. Delivering a Grid enabled Distributed Aircraft Maintenance Environment (DAME), In: Proceedings of the UK e-Science All Hands Meeting 2003, Nottingham, UK. ISBN: 1-904425-11-9

[2.] J. Austin, J. Kennedy, and K. Lees, The advanced uncertain reasoning architecture. In weightless neural network Workshop, 1995.

[3] Storage Request Broker at <http://www.npaci.edu/DICE/SRB/>.

[4] Laing, B., Austin, J., A Grid Enabled Visual Tool for Time Series Pattern Match, In: Proceedings of the UK e-Science All Hands Meeting 2004, Nottingham, UK.

[5] The Globus Toolkit at <http://www.globus.org>.

[6] Jakarta project at <http://jakarta.apache.org/>.

[7] Postgress at <http://www.postgresql.org/>.

[8] Agrawal, R., Faloutsos, C., Swami, A. Efficient Similarity Search in Sequence Databases. In Proceedings of the 4th International Conference on Foundations of Data Organisation and Algorithms (FODO), pages 69-84, Chicago 1993. Springer Verlag.