Title: Scalability Requirements

Author: Workpackage 5 (Scalability)

Editor: John Ibbotson

Authors: John Ibbotson (IBM), Paul Groth (UoS), Simon Miles (UoS)

Reviewers: All project partners

Identifier: D5.1.1

Type: Deliverable

Version: 1.0

Date: 20th September 2005

Status: Public

Summary

Scalability has been studied for Provenance assertion generation (Submission) and Provenance data access (Query). Conceptually centralised, the Provenance Store will be distributed and its distributed instances will have to coordinate the storage of p-assertions. The Provenance Store will have to scale up with the quantity of p-assertions submitted, the number of actors being involved in workflow enactment, the number of actors constituting the Provenance store, and the number of workflows being enacted simultaneously. When dealing with large amounts of data (or very congested networks), we assume that p-assertions are submitted asynchronously. As far as scalability of queries is concerned, the problem is to make sure that, even when p-assertions are distributed at different locations, they can easily be queried or navigated.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

Members of the PROVENANCE consortium:

- IBM United Kingdom Limited United Kingdom
- University of Southampton United Kingdom
- University of Wales, Cardiff United Kingdom
- Deutsches Zentrum fur Luft- und Raumfahrt s.V. Germany
- Universitat Politecnica de Catalunya Spain
- Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutató Intézet Hungary

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

Foreword

This document has been edited by John Ibbotson (IBM) based on input from project partners.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

Table of Contents

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

1 Introduction

After introducing a logical architecture for provenance systems in [CG05] chapter 3, this document now discusses the distribution and scalability of the logical architecture. We have identified two forms of scalability that can be applied to the provenance logical architecture. We refer to these as *Architectural* and *Implementation* scalability.

We consider *Architectural Scalability* to be the scalability when considered from the point of view of the logical process architecture discussed in [CG05] chapter 1. This views the provenance store as a "black box" consisting of a set of interfaces and associated functionality. No consideration is given to how the internals of the "black box" are implemented. The *Architectural Scalability* will have an impact on any open standards the project proposes for the interfaces to a provenance store and is not influenced by any implementation technology.

Implementation Scalability breaks a provenance store into a set of components that are implemented using some computing technology. The technology is configured in such a way as to meet performance requirements such as numbers of connected actors, system response and p-assertion storage volumes. The *Implementation Scalability* issues are separate to the *Architectural Scalability* issues and do not impact any standards proposals developed by the project.

1.1 Overview of the Document

Section 2.0 identifies a set of use cases originating from the Requirements workpackage WP2. Section 3.0 discusses the Architectural Scalability issues and includes a set of recording patterns that identify communications between key architecture roles and explains how the data organisation adopted by the provenance store allows for data that is geographically distributed. Section 4.0 discusses the Implementation Scalability issues and describes a set of components that implement the provenance store together with clustering technology that can be used to implement a scalable provenance store.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

2 Provenance Scalability Use Cases

This section describes a set of provenance Store use cases derived from user and software requirements from Workpackage 2. The discussions refer to components of a provenance store identified in earlier sections.

2.1 Large P-Assertion Messages

Use Case: An actor is participates in a process that involves the movement of large amounts of data. Examples of this include the transfer of medical images as part of a Patient Healthcare Record. How does the provenance architecture deal with potentially large p-assertion messages being submitted to the provenance store? The p-assertions may be either interaction or actor state p-assertions. See requirement SC-1-1.

Description: This use case impacts the *message processor* and the messages that flow between the provenance store and recording or querying actor. The scalability issues illustrated by this use case relate to the potential network delays caused by transporting large messages and the potential creation of large objects within the *provenance store interface* as the data is processed.

There are several alternative solutions to this use case:

1. The provenance store can be located locally to the actor with either a remote or locally available *persistent store*.
    a. With a remote *persistent store*, the delay between sending the large message and the *provenance store interface* responding is minimised. There is still a possible network connection between the *provenance store interface* and the *persistent storage* which can result in associated delays.
    b. With a locally available *persistent storage*, the network delay is eliminated. However, the locally available *persistent storage* may not be part of a fully managed set of storage resources and its contents will have to be replicated with *persistent storage* optimised for an extended lifetime.
2. Use of SOAP Message Transmission Optimisation Mechanism [GM05] to reduce the p-assertion message size. This recommendation from the W3C provides a way of packaging non-XML data with a SOAP message for transmission. Transforming non-XML data to XML by base 64 encoding is extremely inefficient and results in an increase in message size. The recommendation is compatible with the SOAP processing model and may be used to provide an efficient transfer of non-XML data from an actor to the provenance store.
3. The actor does not record the large amount of data within the p-assertion. Instead, it stores the data within an application datastore and includes a reference to the data within the submitted p-assertion. This is the same use case as described in Section 3.1.2.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

2.2 References to Data within a P-Assertion

Use Case: An actor invokes a service to process a large amount of application data. Instead of sending the data, the actor provides a reference to the stored data and sends that in the invocation message. The processing actor uses that reference to access the data to be processed. At a later time, the provenance trace containing the reference is queried. What must the provenance architecture do to ensure that the reference is still resolvable?

See requirement SC-1-5a, SC-1-5b and SC-1-5c.

Description: This use case bridges the application data domain and the storage of p-assertions within a provenance store. The provenance logical architecture document discusses the identification of data items in order to query its provenance. Conversely, the data identifier allows the data to be identified following a query to a provenance Store. For this to be successful, the identification name must be resolvable within the application data store. This implies that the application data store supports a naming convention that is persistent throughout the data lifetime; in other words application data identifiers can always be resolved.

The scope of the provenance project is the management of provenance p-assertions and not the application data to which they refer. Therefore it is assumed that the application database management system (DBMS) is resilient and allows application data identifiers to be resolved.

2.3 Multiple Actors making Submissions

Use Case: A provenance store is shared by a large number of actors that submit interaction and actor state p-assertions to it. How does the architecture deal with many actors submitting p-assertions simultaneously?

See requirement SC-1-3.

Description: This use case impacts the *message processor* of the provenance store. Each p-assertion message received results in a thread of execution within the provenance store. The number of concurrent threads that can be managed by the runtime, and hence the number of actor requests that can be handled concurrently, is dependent on the design of the provenance store runtime software and the performance of the hardware it is running on.

Providing a solution for this use case relies on a combination of two scaling techniques:

1. **Scaling up** increases the performance of runtime software by increasing the performance of the hardware that is providing the runtime environment.
2. **Scaling out** provides multiple instances of the runtime software on separate processors to allow more connections to be made to the single network *endpoint*. This technique is known as *clustering* and is discussed as an implementation technique later in this document.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

For this use case, the logical architecture is unchanged and scalability is introduced as part of the physical implementation of the architecture.

2.4 Multiple Actors Querying at the same time

Use Case: A provenance store is shared by a large number of actors that want to submit queries. How does the architecture deal with many actors querying a provenance Store simultaneously?

See requirement SC-1-4.

Description: This use case has identical issues to the one described in section 2.3.

2.5 Management of large amount of P-Assertions

Use Case: A *persistent store* associated with a provenance store contains large numbers of p-assertions resulting in potentially Terabytes of data. How does the architecture support the management of large amounts of p-assertion data?

See requirement SC-1-6.

Description: The architecture should treat the *persistent store* as a separate, closed component accessible by published interfaces. In most cases, the stateful resource will be implemented using a DBMS. Overall sizing of a provenance aware system will establish the expected volumes of p-assertions to be handled by the DBMS and a suitable configuration for the DBMS to support the large volumes must be deployed. The mechanism for managing large volumes of data in a DBMS is well known and outside the scope of the provenance project.

2.6 Degradation of Performance whilst Submitting P-Assertions

Use case: Requirements state that adding provenance capability to an existing or newly designed system must not degrade its performance. The performance metrics listed in the requirements include:

1. Responsiveness of a user interacting via a GUI
2. End to end execution time of a workflow
3. Volume of provenance p-assertions as a percentage to application data

See requirements CR1-1-A, CR1-1-B, CR1-1-C, CR1-1-D, CR-1-2-A, CR-1-2-B and CR-1-2-C.

Description: Issues 1 and 2 identified in this use case may be addressed by using an asynchronous messaging model connect to the provenance store. The asynchronous messaging model does not block any application processing and this, together with good practice for overall system design, will minimise the overall impact of a provenance capability on the system.

Issue 3 may be addressed through system design. The contents of an interaction or actor state p-assertion message are determined by the system designer; constrained by their schemas. Therefore the p-assertion message sizes themselves are not dictated by the provenance architecture.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

2.7 Large Result Sets

Use Case: An actor submits a query to a provenance store using the query interface. The result of the query contains a large number of results in the returned set. This has a potential negative impact on the performance of the provenance store. How does the architecture deal with returning large result sets from a query?

See requirement SC-1-2.

Description: Most information retrieval systems allow restrictions (such as the number of results to be returned) to be placed on submitted queries. The provenance query API should allow this feature and include operations for iterating through the returned result set. For large result sets, deferred presentation of results or redirection of result sets to a localised file systems are possible solutions. An implementation of the architecture should support these different solutions.

2.8 Access to P-Assertions for Query

Use Case: An actor is submitting p-assertions to a provenance store. At the same time, another actor attempts to query the same provenance store. The architecture should support the dynamic processing of p-assertions, i.e. recorded provenance data should be instantly accessible for queries even if a recording session is still in progress

See requirements TR-4-1 and TR-4-2.

Description: Use cases 2.3 and 2.4 address the issue of large numbers of actors recording and querying a provenance store. Each p-assertion record and query is independent of each other and each creates a separate thread of execution within the provenance store. Concurrent processing requests by the provenance store support the implementation of this use case.

2.9 Submission of Repeated P-Assertions

Use Case: How can an actor, involved in a long iteration in a workflow, avoid submitting the same p-assertions at every iteration step? Examples of this may include the actor functionality and data flow information. Information could be registered elsewhere (say, in a registry such as grimoires), possibly later consolidated into the provenance store, but the actor needs a mechanism to declare that information is to be found elsewhere.

Description: Loops within a workflow will result in the same or partial p-assertion data being repeatedly submitted; albeit with different time stamps. This may result in extremely large amounts of p-assertion data being managed by a provenance store. To reduce this volume, the repeated data can be stored by reference. The reference would consist of an identifier that refers to the p-assertion data which is stored either in a registry or within the provenance store. To support this, the p-assertion schema would have to include suitable identifiers for the complete or partial tree of the p-assertion message. Changes to the schema to support references within the p-assertion would be required.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

2.10 P-Assertions and databases

Use Case: Provenance has been studied intensively in database systems. In some specific cases (specialised query language, etc), the provenance of some data **within the database** can efficiently be reconstructed. In this case, if an actor is such a kind of database, the database will **not** submit actor state p-assertions describing data links etc, because these can be computed when and as required. Hence, p-assertions must indicate that useful information can be obtained from the database at provenance query time.

Description: The description of this use case is outside the scope of this document.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

3 Architectural Scalability

In this section, we discuss the issues relating to the *Architectural Scalability* of a provenance store. We identify a set of recording patterns that can be used to influence the design of a provenance store. Given these patterns, we can describe how actors within a workflow can each record their p-assertions in different provenance stores. These provenance stores may be in different domains for reasons of distribution and/or security. The set of p-assertions that documents the generation of some application data may be distributed throughout different provenance stores. We describe the advantages of this architectural approach and propose a linking mechanism that allows an actor using the provenance query interface to reconstruct a complete provenance trace.

3.1 Recording Patterns

We need to distribute the provenance stores in which documentation of a single execution is recorded, as there can be a large amount of data in a large amount of assertions from a large set of actors deployed in a large number of organisations, each with their own security domain, privacy requirements etc. The requirement for recording process documentation in distributed provenance stores, such that all documentation related to one execution can be retrieved again, presents a developer with several deployment problems. Therefore, one aim of the scalability architecture is to present a set of deployment patterns that address these problems.

A pattern based approach is discussed by Groth [G05] and began with Alexander in the field of architecture [AIS77, Ale79] and was later promoted as useful for software systems by Gamma et al. [GHJV95]. According to Alexander [AIS77], a pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that one can use this solution a million times over, without ever doing it the same way twice. A pattern then describes a solution to a common design problem; the solution described must strike a balance between being concrete enough to be applicable and abstract enough so that it can be applied to a range of similar problem situations. Patterns allow us to present a solution that any developer can use to integrate p-assertion recording into their SOA-based application. We now describe a set of patterns for p-assertion recording. The format of these patterns is as follows:

Name	A short name of the pattern that reflects the solution.
Diagram	A diagram that shows the pattern visually. Diagrams have a common visual appearance. Provenance stores are labeled and denoted by a 3D cylinder. Actors are denoted by boxes. A single message exchange is denoted by a line with an arrow head. The arrow denotes the direction of the message flow. Dotted lines follow the same convention but denote multiple message exchanges.
Context	The situation in which the pattern applies and why this pattern exists.
Problem	Describes the problem that the pattern solves providing more detail as to when the pattern should be applied.
Solution	A description of how to apply the pattern including the interactions between actors and any properties an actor is expected to have in order to function in the pattern.

3.1.1 Separate Store

Name	SeparateStore. See Figure 1.
Diagram	

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

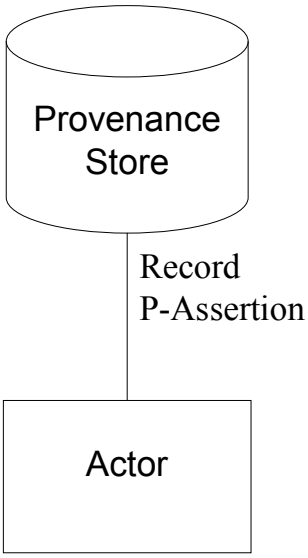

Figure 1 SeparateStore Pattern

Context Application actors want to make available information about their interactions and associated state. This pattern exists because querying actors want to know how application actors have interacted in the past in order to produce a piece of data. To know how application actors performed, these application actors must make available information about their actions.

Problem An application actor, A, may be involved in a large number of interactions over its lifetime and cannot retain all the process documentation itself. Likewise, querying actors would like to access information about A's previous message exchanges and state even when A is not available. For example, A may have been shutdown, moved or be under repair.

Solution A separately deployed store is introduced to retain information about an application actor's interactions and state. In Chapter 2, we termed that store a provenance store. An actor records assertions (p-assertions) into a provenance store. Therefore, the actor does not have to retain the information. A provenance store should have the following properties:

1. It should be available in a long term manner in comparison to the application actors that submit p-assertions to it. This property allows p-assertions recorded by an application actor to be accessed after the application actor has become unavailable.
2. It should provide a well defined interface for the recording of p-assertions by an application actor.
3. It should provide a query mechanism to retrieve p-assertions, which makes the p-assertions available to querying actors.
4. It should provide a management mechanism to manage the stored p-assertions.

3.1.2 Context Passing

Name ContextPassing. See Figure 2.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

Diagram

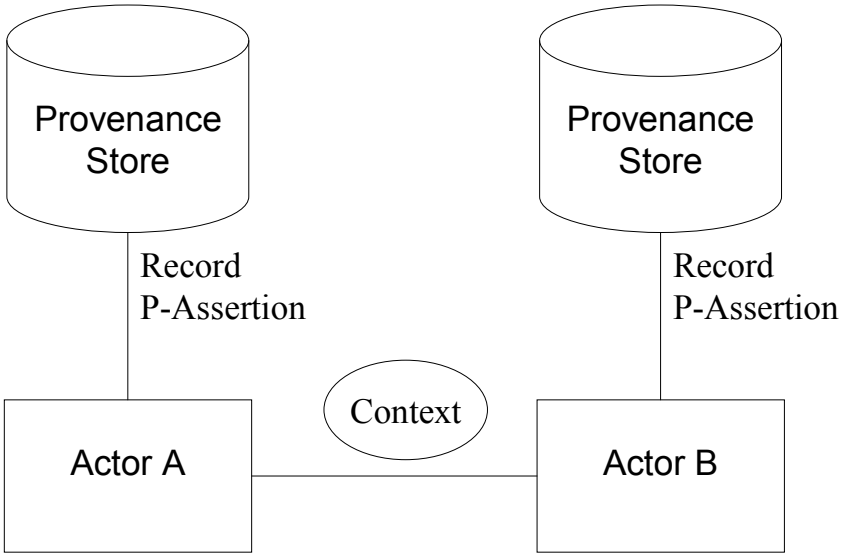

Figure 2 ContextPassing Pattern

Context	Two application actors, A and B, exchange a message. A and B record p-assertions about this interaction in two provenance stores (see the pattern SeparateStore). Both actors record these interaction p-assertions because they want their view of that interaction to be recorded. This allows other actors to determine if A's and B's views of the interaction concur. Likewise, A and B may want to record actor state p-assertions within the context of the interaction.
Problem	The p-assertions that A and B record need to be identified as being the documentation for the same message exchange. Otherwise, the actors' views of the message exchange cannot be associated with one another and it becomes difficult to determine if the recorded p-assertions are documentation for the same interaction.
Solution	The client actor in the interaction must generate the appropriate identifiers (IDs) to identify the interaction. It must then pass a context containing those IDs to the service actor. Both actors use these IDs to record their p-assertions in their respective provenance stores. The p-assertions for the interaction can be matched by the IDs generated by the client actor. One method of passing this context is by attaching it to the message being passed between the client and service actors. This is just one method; application actors may use any method appropriate to pass the context information. Beyond passing IDs, application actors may use a context to pass other information relevant to provenance.

3.1.3 Shared Store

Name SharedStore. See Figure 3.

Diagram

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

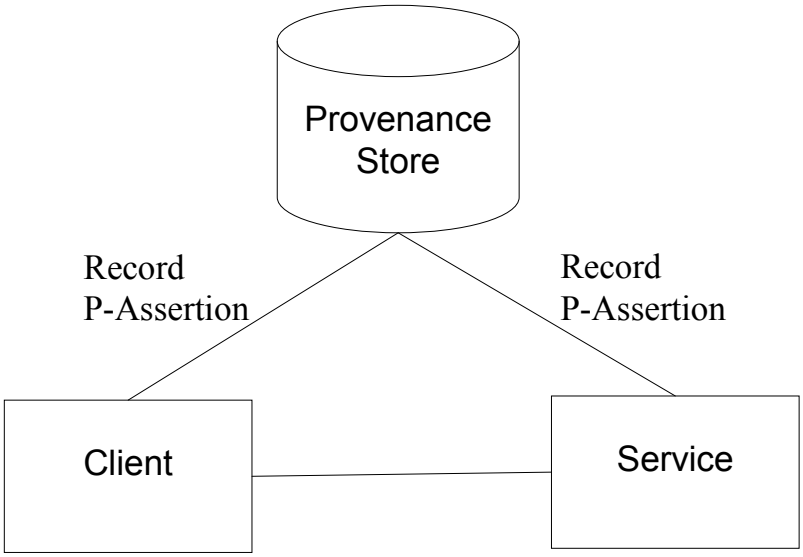

Figure 3 SharedStore Pattern

Context	Actors record p-assertions into p-assertions into provenance stores following the SeparateStore and ContextPassing patterns.
Problem	The SeparateStore and ContextPassing patterns may lead developers to believe that for every application actor there is a corresponding store. However, developers may not want to deploy a provenance store for every application actor, especially when the number of application actors is large. Also, in order to retrieve the provenance of a result each provenance store must be contacted resulting in slower query performance.
Solution	Application actors are allowed to record p-assertions in a shared provenance store. All p-assertions pertaining to one interaction from a particular actor should be stored in the same provenance store. This keeps the record of an interaction in one place allowing for faster queries. The SharedStore pattern clarifies the way in which SeparateStore and ContextPassing can be applied. Both SeparateStore and ContextPassing are agnostic as to what provenance store an actor may use to record its p-assertions. The pattern emphasizes that actors can record their p-assertions in any store they choose and provenance stores may hold p-assertions from multiple actors. SharedStore does not prescribe how many stores there should be and which provenance stores should be shared. This is left to the developer applying the pattern. SharedStore allows developers to determine the distribution of provenance stores that fits their application.

3.1.4 Pattern Application

These patterns show how p-assertions can be recorded in provenance stores by actors. The documentation of process can be recorded for an entire system by applying a selection of these patterns to every actor in a system. For example, Figure 4 shows a system with a client initiator, a workflow enactor and a service all recording p-assertions about their request and response interactions. SeparateStore, ContextPassing and SharedStore have all been applied multiple times in this case.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

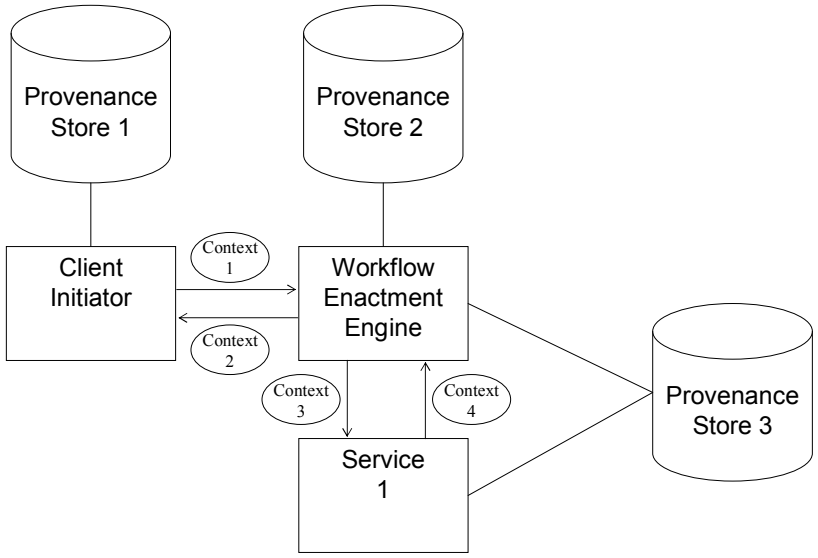

Figure 4 A system in which SeparateStore, ContextPassing and SharedStore have been applied multiple times

These recording patterns allow for the flexible logical deployment of provenance stores to aid scalability. The patterns can be applied to any number of interacting actors using any number of provenance stores in order to record passertions.

3.1.5 Linking

By the definition of the p-assertion recording patterns, an actor is allowed to record its p-assertions in any provenance store. This means that the documentation of process that led to a result can exist across any number of provenance stores. There are several benefits in allowing documentation to be recorded across multiple stores: the elimination of a central point of failure, the spreading of demand across multiple services and the ability for provenance stores to exist in different network areas (for example, one provenance store may be behind a firewall whereas another is not). In general, allowing p-assertions to be recorded across multiple stores increases the flexibility and scalability of systems recording p-assertions. This scalability and flexibility are key to allowing these patterns to be applied in the large scale, open distributed systems that we consider.

Given that the p-assertions representing the provenance of a result may be spread across multiple stores, there must be some mechanism to retrieve these p-assertions in order to validate, visualise or replay the represented process. To facilitate such a retrieval mechanism, we introduce the notion of links. Intuitively, a link is a pointer to a provenance store.

We assume that there is a resolution function that maps provenance store identities to their physical location. Thus, an actor can contact a provenance store given the store's identity. We define a link as a pair of identities respectively called source and destination. Links are unidirectional: the source identity identifies the location from which the link departs, whereas the destination identity points at

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

the location where the link arrives. The source identity is always the identity of the provenance store in which the link is stored.

Links are used in two instances. The first instance is when a client's view of an interaction and a service's view of the same interaction as identified by a shared interaction ID are stored in two different provenance stores. It is necessary for them to record a link, which we refer to as a View Link that points to the provenance store where the opposite party recorded their p-assertions. Hence, the client in an interaction records a link to the provenance store that the service used to record p-assertions for the given interaction, and vice-versa. This allows querying actors to navigate from one provenance to the other in order to retrieve both views of an interaction. We note that links point to provenance stores only, not particular pieces of data in the provenance store. View Links are dependent on the two actors involved in an interaction informing each other of the provenance stores they each use. They can do so by using the ContextPattern. A provenance store identity may be recorded as part of an interaction p-assertion or an actor state p-assertion. It is up to the provenance store implementation to extract the provenance store identity from the context and create a View Link. The second instance is when an application actor records a relationship passertion. In this case, the application actor may record which provenance store the data being related is stored in. Again the link only points to the provenance store and not a particular piece of data in the store.

Both View Links and links within relationship p-assertions allow data and p-assertions stored across provenance stores to be retrieved by querying actors.

3.2 Data Staging

A provenance store that manages p-assertions over an extended lifecycle requires extensive management and data storage features of the kind provided by a commercial database management system. These are typically provided within an organisation on managed servers administered centrally. In most cases, actors will communicate directly to a provenance store whose interfaces provide connections to the data management infrastructure.

There are, however, cases where systems designers may choose to include data staging in their design. Data staging allows a provenance store to be placed closer to the recording actor for reasons of performance. An example of this might be where the documentation of execution is being recorded for a workflow running on a computing cluster with high speed networking support between the clustered processors. Slower speed networks connect the cluster to other servers which include a managed provenance store. Communicating directly with the managed provenance store would impact the performance of the clustered workflow.

For this use case, we identify the requirement for a provenance store to be able to export its p-assertions in a format that can be imported into another provenance store allowing locally staged p-assertions to be moved to a remote managed provenance store independently of the recording interface. This may be achieved in either of two ways:

1. The provenance management interface includes import and export operations to allow p-assertions to be moved between provenance stores.
2. The provenance management interface allows an alternative protocol such as GridFTP to be used to transfer p-assertions from one provenance store to another.

Data staging may also be used to overcome performance bottlenecks when recording large p-assertion messages. However, two alternatives to recording large amounts of p-assertion data are now discussed.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

3.2.1 By-Value versus By-Reference recording

Section 2.2 identifies a use case where references to data are included in a p-assertion rather than including the data by-value. Examples of this use case are where interaction p-assertions include large amounts of application data or where the data in a service request or response is private. Examples of private data may include patient records in a healthcare application. The patient records are usually held within a database that contains high levels of access security and the system designers may wish to have a lower level of security for the provenance store.

This use case can be supported by including application data references as optional elements within the p-assertion message schema. Chapter 6 of the provenance logical architecture [CG05] discusses the links between application data items and their associated provenance information. It identifies a set of naming conventions and patterns that can be used to link application data and provenance for querying. These conventions may be used within the p-assertion recording messages to support the by-reference use case.

For by-reference recording of data within a p-assertion, the lifetime of the application data has to be equivalent to that of the provenance store. To resolve a subsequent query to the provenance store, the querying actor also has to have access rights to the application data referred to by the p-assertion. An alternative approach would be for the referenced data to be managed by the provenance store. This approach however requires the provenance store to be at least as secure as the application data store. This scenario would not support the healthcare example cited earlier in this section.

The recommended approach for this scenario would be for the application data to be managed separately from the provenance store and references placed within the p-assertion message to link the two p-assertion and application data records.

3.2.2 Record-Once versus Record-Many

Section 2.9 identifies a use case where the same (or similar) p-assertion message may be recorded multiple times leading to potentially large amounts of redundant information within the provenance store. This use case may be supported by implementing a strategy commonly used by compression algorithms. In compression algorithms, a catalogue of common patterns that appear in the data is created. Each entry in the catalogue has an identifier which is typically smaller than the pattern it identifies. To compress the data, when a particular pattern is identified it is replaced by its catalogue identifier which results in a compressed version of the data. Decompression of the data is performed by replacing the catalogue identifiers within the data with the matching patterns from the catalogue which is attached to the compressed data.

For the provenance recording protocol, a p-assertion is assigned a unique identifier. This is similar to the catalogue identifier described above. If an actor records a subsequent p-assertion that is the same as a previously recorded one, it may choose to replace the contents of the p-assertion message with the previous unique identifier thereby reducing the size of the p-assertion message. If there are multiple p-assertion messages to be recorded, it may repeatedly replace the p-assertion message with the unique identifier. Querying the p-assertion will include a lookup stage that replaces the recorded identifier with the contents of the message that matches the identifier. This is analogous to the decompression of data described above.

The previous description describes the replacement of a complete p-assertion message by its identifier. This mechanism can be extended by including identifiers at roots of sub-trees within the p-assertion message schema. This allows replacement at the sub-tree level within a p-assertion message in cases where only part of a p-assertion message is repeatedly recorded.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

3.3 Architectural Scalability Recommendations

The following recommendations are made for *Architectural Scalability*:

1. The provenance query interface should support the retrieval of large result sets. This will provided by caching of the results within the provenance store. A set of interface operations will allow actors to iterate and retrieve the cached results in response to a query request.
2. The provenance recording interface should support the linking between application data and p-assertions using the by-reference model. P-assertion message schemas will allow references to application data to be included in instance messages. The provenance store will not be responsible for managing the application data lifecycle.
3. The provenance recording interface should support the recording of repeated p-assertions by identifying the message and its sub trees with an identifier. These identifiers may be used in subsequent p-assertions to replace the repeated information.
4. The provenance management interface should support the import and export of p-assertions allowing provenance store to exchange their contents. Options for providing this support are:
    a. Explicit import and export operations on the management interface
    b. Operation(s) allowing provenance stores to communicate via another protocol (e.g. GridFTP)
5. The provenance store recording interface should support the SOAP MTOM recommendations for non-XML data.

3.4 Use Cases satisfied by Architectural Scalability

Architectural Scalability will satisfy the following Use Cases:

Large Result Sets
This is supported by *Architectural Scalability* recommendation 1.

References to Data within a P-Assertion
This is supported by *Architectural Scalability* recommendation 2.

Submission of Repeated P-Assertions
This is supported by *Architectural Scalability* recommendation 3.

Large P-Assertion Messages
This is supported by *Architectural Scalability* recommendations 4 and 5

3.5 Conclusions

This section has discussed the *Architectural Scalability* of the provenance architecture. This provides an external view of the logical architecture which will influence the draft open standards developed as part of the project. It has identified a set of recording patterns that can be used to distribute p-assertions between distributed provenance stores and proposed a linking mechanism for a complete set of p-assertions relating to application data to be assembled following a query to the provenance store. It has also discussed the use of data staging for use cases where large p-assertion messages are recorded.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

4 Implementation Scalability

4.1 Components of a Provenance Store

This section identifies components of the provenance logical architecture and assesses their impact on the scalability of the overall architecture. In particular we address components of the provenance store and their impact on overall system scalability.

The provenance logical architecture classifies the actors involved in the provenance lifecycle according to their role in a provenance system. Briefly, the responsibilities of each role are as follows.

- An *application actor* is responsible for carrying out the application's business logic.
- A *provenance store* is responsible for making persistent, managing and providing controlled access to recorded p-assertions.
- An *asserting actor* is an actor that creates p-assertions about an execution.
- A *recording actor* is an actor that submits p-assertions to a provenance store for recording.
- A *querying actor* is an actor that issues provenance queries to a provenance store.
- A *managing actor* is an actor that interacts with the provenance store for management purposes.

Central to the provenance architecture is the notion of a provenance store which is a service designed to store and maintains a provenance representation beyond the lifetime of an application. Such a service may encapsulate at its core the functionality of a physical database, but also provides additional functionality pertinent to the requirements of the provenance architecture. In particular, the provenance store's responsibility is to offer long-term persistence of p-assertions.

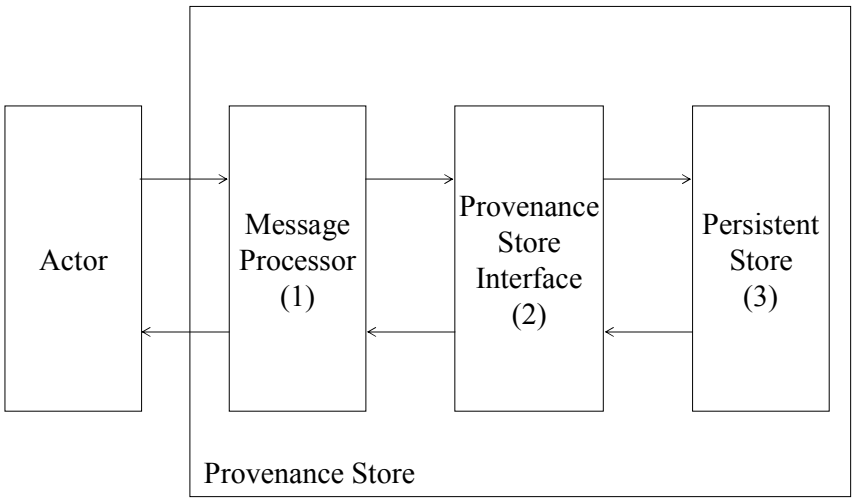

Figure 5 Components of a Provenance Store

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

We can further divide the provenance store into three components that jointly provide an implementation of the provenance store role. These are:

1. A *message processor*. This component receives request messages from actors and returns response messages to them. It may support one or more transport protocols such as HTTP, SMTP or IIOP. The term *endpoint* is often applied to this component since it is made available to the distributed computing fabric at an advertised network address. Other responsibilities of the *message processor* include translating the message into a format that is understandable by the *provenance store interface*. For example, this may include converting an on-the-wire XML representation into a collection of Java objects and dispatching them to the provenance store interface.
2. A *provenance store interface*. This component implements the operations specified by the provenance store external interface. It receives translated request messages from the *message processor* and implements the functionality of the provenance recording, query and management interfaces. This component sits between the *message processor* and the *persistent store*.
3. A *persistent store*. This component stores and maintains p-assertions beyond the lifetime of a Grid application. Such a component encapsulates the functionality of a physical database; in particular it has the responsibility is to provide long-term persistence of p-assertions.

4.2 Managing State

A typical interaction between an actor and a provenance store consists of a single session. An actor sends a request to the *message processor* which forwards it to the *provenance store interface*. The *provenance store interface* may then access the *persistent store* to record or query p-assertions. The response to the actor's request then flows in the reverse direction resulting in the results of the requested operation being returned to the actor. This pipeline is fundamental to the operation of the provenance store; equally it is the cause of bottlenecks in system performance and its design and implementation must be carefully considered.

The role of a provenance store is to persistently manage a set of recorded p-assertions. The management of p-assertions is *stateful*. By this we mean that a recorded p-assertion can be subsequently retrieved from a provenance store for later analysis or processing. The term *state* in this context refers to the last-known or current status of an application or a process; in this case, the actor. The terms *maintaining state* and/or *managing state* refer to keeping track of the condition of the process. The provenance store, as far as the actor is concerned, has to be *stateless*. By this we mean that any request to the provenance store is processed without any knowledge of any previous requests. This is the model used by the World Wide Web which is intrinsically stateless because each request for a new Web page is processed without any knowledge of previous pages requested. Some commentators consider this is to be one of the drawbacks to the HTTP protocol. Because maintaining state is extremely useful, programmers have developed a number of techniques to add state to the World Wide Web. These include server APIs, such as NSAPI and ISAPI, and the use of cookies.

However, for a scalable architecture, a *stateless* provenance store is required. We can identify two components of state that must be considered to achieve an overall stateless design. These are *application state* and *connection state*.

Application state is represented by the set of p-assertions recorded by actors for subsequent retrieval via the provenance query interface. The *persistent store* component of the provenance store has the responsibility for managing these p-assertions throughout the provenance lifecycle. Typically this

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

component will be implemented using a database management system (DBMS) that can manage the p-assertions over an extended period of time.

Connection state only exists for the lifetime of a single thread of execution within the provenance store and consists of a session established between the requesting actor and the provenance store. The session exists to allow a response to the request to be returned to the actor. Once the response has been returned, the session may be torn down. This ensures that in terms of connection, a request by an actor is not dependent on any previous requests; its importance will be discussed later when clustering as an implementation technology is introduced.

Application state is passed between an actor and a provenance store through the schemas of request and response message instances. Examples of application state information may include:

- Identifiers for *persistent stores*
- P-assertion headers and provenance context headers
- Security tokens used by the provenance store to authenticate actors

In summary, for scalable provenance architecture, the provenance store has to appear stateless in terms of connections when viewed by a requesting actor. *Application state* is held within the *persistent store* component of the provenance store. References to the *application state* are contained within the request and response messages exchanged between an actor and the provenance store.

4.3 Scalability through Clustering

The purpose of the provenance logical architecture is to describe the architecture in terms that are independent of any physical implementation. This section discusses how the components of a provenance store may be mapped to an implementation infrastructure that meets the scalability requirements Use Cases from section 2 that relate to actor connections and response.

The scalability strategy referred to as scaling-out provides for resources to be made available to support large numbers of connections to an application server used to implement a provenance store. The strategy is most commonly realized using clustering technology. When clustering is implemented, a *Load Balancing* component is added to a set of application servers that allows the servers to be viewed as a single logical network endpoint. Each application server will contain a replicated instance of a provenance store. This is illustrated in Figure 6.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

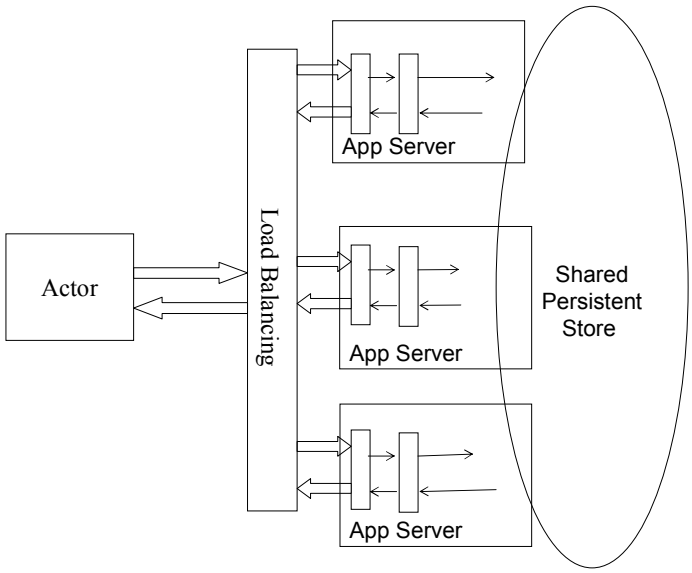

Figure 6 Clustered Containers

The Load Balancer allows connection requests from an Actor to be dynamically re-directed to one of any application servers within the cluster. These application servers are deployed on different physical machines each with its own IP address within the cluster. Two popular methods of load balancing in a cluster are *DNS round robin* and *hardware load balancing*. DNS round robin provides a single cluster logical name, returning any IP address of the nodes in the cluster. This option is inexpensive, simple, and easy to set up, but it does not provide any server affinity or high availability. In contrast, hardware load balancing solves the limitations of DNS round robin through virtual IP addressing. Here, the load balancer shows a single IP address for the cluster, which maps the addresses of each machine in the cluster. The load balancer receives each request and rewrites headers to point to other machines in the cluster. If any machine in the cluster is removed or fails, the changes take effect immediately. The advantages of hardware load balancing are server affinity and high availability; the disadvantages are that it's expensive and complex to set up.

There are many different algorithms to define the load distribution policy, ranging from a simple round robin algorithm to more sophisticated algorithms used to perform the load balancing. Some of the commonly used algorithms are:

- Round-robin
- Random
- Weight-based
- Minimum load
- Last access time
- Programmatic parameter-based (where the load balancer can choose a server based upon method input arguments)

Load-balancing algorithms affect statistical variance, speed, and simplicity. For example, the weight-based algorithm has a longer computational time than the other algorithms. Clustering is supported in many proprietary and open source servers.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

A provenance store that is easily clusterable must have no affinity with any of the actors or applications that connect to it. To achieve this, the service must be stateless apart from handling any connection state as discussed in a previous section. All state information that relates to a connecting actor or application must be stored in the associated stateful resources as described in previous sections. This is compatible with the provenance logical architecture where the stateful information from an actor is contained in the p-assertion schema. Connection state is sufficient to ensure that a response to a request is returned via the correct connection thread.

A further condition is that the stateful resources must be shared between all runtime containers within the cluster. Since there must be no affinity between an actor and the server it is connected to, all state must be accessible by all containers hosting instances of the provenance store within the cluster. For example, a workflow enactment engine that is recording many p-assertions to a provenance store will have all the submissions stored within the same persistent store irrespective of which container instance the submission requests are processed by.

The provenance architecture will also make use of a number of system "middleware" services. Examples of such services will include naming and security authorisation and authentication. To ensure that there are no bottlenecks in overall system performance, these service implementations must also be scalable. These services will include commercial or open source database management systems to implement the *persistent store* component of the provenance store.

4.4 Implementation Scalability Recommendations

The following recommendations are made for *Implementation Scalability*:

1. Provenance actors should have no affinity to a provenance store. Actors communicate with the single published network endpoint of the provenance store. This recommendation ensures that a provenance store can be scaled using clustering technology.
2. Implementations of the provenance recording, query and management interfaces should be clusterable.
3. The provenance store persistent store component should be implemented using a proprietary or open source database management system.

4.5 Use Cases satisfied by Implementation Scalability

Implementation Scalability will satisfy the following Use Cases:

Multiple actors making submissions
This is supported by *Implementation Scalability* recommendation 2.

Multiple actors querying at the same time
This is supported by *Implementation Scalability* recommendation 2.

Management of large amount of P-Assertions
This is supported by *Implementation Scalability* recommendation 3.

Degradation of Performance whilst Submitting P-Assertions
This is supported by *Implementation Scalability* recommendation 2.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

Access to P-Assertions for Query
This is supported by *Implementation Scalability* recommendation 1.

4.6 Conclusions

This section has discussed how scalability use cases can be supported through the implementation of a provenance store. It considers how the three components of a provenance store may be implemented and what computing technologies would be appropriate to support a given set of scalability use cases.

The solutions discussed in this section do not impact the external or *Architectural Scalability* issues that would influence a set of open standards the project may propose.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

5 Conclusion and Future Work

In this document, we discussed scalability issues that were relevant in the context of provenance. The scalability architecture for the provenance store is then presented along with an explanation of the functionality of its constituent components including an illustration of the interaction of these various components in managing state within the provenance store. We present a set of use cases from the Requirements workpackage discussed with respect to the provenance store components. Finally, we identify a set of recording patterns between actors and the provenance store.

5.1 Future Work

Deliverable D5.2.1 of the provenance project will be a further draft of this document. In this later draft we will address the following issues:

1. A mapping of the provenance architecture to a set of Grid and Web Services standards to support a scalable implementation.
2. Scalability for provenance queries.
3. A discussion of the middleware technologies to support scalable deployment of the architecture.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme

Appendix A References

[AIS77] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. A Pattern Language. Oxford University Press, 1977.

[Ale79] Christopher Alexander. The Timeless Way of Building. Oxford University Press, 1979.

[CG05] Chen L et al. An Architecture for Provenance Systems. Internal document for Provenance project.

[G05] P. Groth. Provenance in Large Scale, Open, Distributed Systems, July 2005. Available from http://www.ecs.soton.ac.uk/~pg03r/mypapers/PaulMiniThesisFinal.pdf

[GM05] M. Gudgin et al. SOAP Message Transmission Optimization Mechanism, W3C Recommendation, January 2005. Available from http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns. Addison-Wesley Professional, 1995.

The PROVENANCE project receives research funding from the European Community's Sixth Framework Programme