Recording and Using Provenance in a Protein Compressibility Experiment

Paul Groth, Simon Miles, Weijian Fang, Sylvia C. Wong, Klaus-Peter Zauner and Luc Moreau
School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
Tel: +44 23 8059 4487
Contact: pg03r@ecs.soton.ac.uk

Abstract

Very large scale computations are now becoming routinely used as a methodology to undertake scientific research. In this context, 'provenance systems' are regarded as the equivalent of the scientist's logbook for in silico experimentation: provenance captures the documentation of the process that led to some result. Using a protein compressibility analysis application, we derive a set of generic use cases for a provenance system. In order to support these, we address the following fundamental questions: what is provenance? how to record it? what is the performance impact for grid execution? what is the performance of reasoning? In doing so, we define a technology-independent notion of provenance that captures interactions between components, internal component information and grouping of interactions, so as to allow us to analyse and reason about the execution of scientific processes. In order to support persistent provenance in heterogeneous applications, we introduce a separate provenance store, in which provenance documentation can be stored, archived and queried independently of the technology used to run the application. Through a series of practical tests, we evaluate the performance impact of such a provenance system. In summary, we demonstrate that provenance recording overhead of our prototype system remains under 10% of execution time, and we show that the recorded information successfully supports our use cases in a performant manner.

1 Introduction

Very large scale computations are now becoming routinely used as a methodology to undertake scientific research: success stories abound in many domains, including physics (www.griphyn.org), bioinformatics (www.mygrid.org.uk), engineering (www.geodise.org) and geographical sciences (www.earthsystemgrid.org). In this context, 'provenance systems' are being regarded as the equivalent of the scientist's logbook for *in silico* experimentation. Provenance, which captures the documentation of the process that led to some data, is necessary to allow users to verify how results were achieved or to reproduce them: this is particularly crucial when results can be obtained only by *in silico* means and no other validation in the physical world is possible. Furthermore, as we design systems with provenance, new opportunities arise, in which added value is provided by provenance, for instance, to analyse and reason over previous experiments.

Specifically, we consider here a protein compressibility analysis application, which uses a mix of brute force computation, statistical methods, and guess work, in order to study the structure of protein sequences. Given its characteristics, such an application is an ideal candidate for use of Grid technology [4]. While studying this application, we have elicited several use cases for provenance (which have been compiled as part of a complete survey [11]). For instance, provenance is crucial to decide if two results were obtained by the same scientific process; to verify if semantically valid operations were performed; or to decide if a specific data item was used as input to a computation.

As far as provenance is concerned, the challenges are manifold. *(i)* From a conceptual viewpoint, the principled design of a notion of provenance must be related to execution in a meaningful manner and must contain sufficient information to satisfy the different use cases. *(ii)* From an engineering viewpoint, systems are becoming more and more complex and are typically assembled from multiple heterogeneous systems, each of them addressing specific parts of the computation. For instance, in the LCG Atlas experiment (atlas.web.cern.ch), Athena and VDT coexist, each capable, in their own way, of specifying components, compositions and their execution; likewise, applications, like the one studied in this paper, can consist of a mix of VDL workflows [5], shell scripts, and Web Services. Hence, the ques-

tion of inter-operability arises: how can we ensure that each sub-system can individually provide provenance data that can seamlessly be used to support use cases. Thus, protocols are necessary for provenance data to be recorded and to be retrieved. *(iii)* From a practical viewpoint, while it is expected that provenance recording may introduce some overhead, it should not hinder the progress of execution in any unacceptable way.

In summary, we are confronted with the following questions: what is provenance? how to record it? what is the performance impact for grid execution? what is the performance of reasoning? These are the questions that we address in this paper by presenting our preliminary evaluation of a practical deployment of a provenance architecture into a protein compressibility bioinformatics application. In answering these questions, our specific contributions are the following:

1. We define a notion of provenance that it is technology independent, but allows us to capture, in a systematic and structured manner, information about the scientific process that is being executed.

2. To promote inter-operability between systems, we introduce a provenance store, in which provenance documentation can be stored, archived and queried. We use this store to record provenance documentation that pertains to the scientific process rather than the execution environment being used.

3. In a series of practical experimentation, we evaluate the performance impact of such a provenance architecture. In summary, we demonstrate that provenance recording overhead of our prototype architecture can remain under 10% of execution time, and we show that the information recorded is sufficient to support our use case in a performant manner.

In the rest of the paper, we introduce our bioinformatics Grid application in Section 2 and its use cases for provenance in Section 3. We then discuss shortcomings of existing provenance systems in Section 4 and define our notion of provenance and its architectural realisation in Section 5. This is followed the evaluation of the provenance architecture in Section 6. We then analyse our results and discuss future work in Section 7 before concluding the paper.

2 Protein Compressibility

In this section, we explain the biology and process of our bioinformatics case study.

Biology Proteins are the essential functional components of all known forms of life; they are linear chains of typically a few hundred building blocks taken since 2 milliards years from the same set of about 20 different amino acids. Protein sequences are assembled following a code sequence represented by another polymer (mature mRNA). This polymer is produced by splicing certain pieces (the exons) of a molecular copy of the coding region of a gene on the DNA, while discarding other pieces (the introns) of the copy. During and following the assembly, the protein will curl up under the electrostatic interaction of its thousands of atoms into a defined but agile shape of typically 5–8 nm size. The resulting 3D-shape of the protein determines its function.

The linear structure of protein (amino acid) sequences is of considerable interest for predicting which sections of the DNA encode for proteins and for predicting and designing the 3D-shape of proteins. For comparative studies of the structure present in an amino acid sequence, it is useful to determine the textual compressibility of the sequence. Compression exploits context-dependent correlations within the sequence. The fraction of its original length to which a sequence can be loss-lessly compressed is an indication of the structure present in the sequence. In general, no practical compression method can discover all the structure in a sequence. Actual compression of a sequence can only yield a lower bound on its compressibility. For the same reason, the compressibility values are also relative to the applied compression method [9]. Methods that are good at discovering structure are computationally expensive; initial investigations of protein compressibility indicated that it is indeed difficult to discover structure in protein sequences [13]; however, recently, progress has been made by grouping amino acids [14]: if the compression of the sequences serves only to quantify structure and decompression is not intended, the sequences can be recoded with a reduced alphabet. In an amino acid sequence, for instance, each amino acid symbol is replaced by a symbol representing a group of amino acids. Compression is then applied to the recoded sequence. The results of this experiment can, for example, be used to determine the amino acid groupings that maximise compressibility.

Workflow To focus the discussion, we consider the main workflow of the comparative sequence compressibility experiment, shown in Figure 1. It starts with the selection of a sequence sample, which sample may be composed from several individual sequences to provide enough data for the statistical methods employed by the compression algorithms (**Collate Sample**). This sample is then recoded with a given group coding (**Encode by Groups**). The recoded sequence is then compressed with compression algorithms, e.g., gzip, bzip2 or ppmz, to obtain the length of the compressed sequence. Random permutations of the sequence (**Shuffle**) are also compressed to provide a standard for comparison. This standard removes the influence of two factors from the calculation of compressibility: the

particular data encoding used to represent the groups, and the non-uniform frequency of groups. From the results, a compressibility value is obtained for the sample sequence that is relative to both the compression method and group coding employed. The variability in the compressed length of the permuted sequences leads to a distribution of compressibility values (**Collate Sizes**). The workflow entails a sufficient number of compressions of permuted sequences to estimate the standard deviation for the compressibility (**Average**).

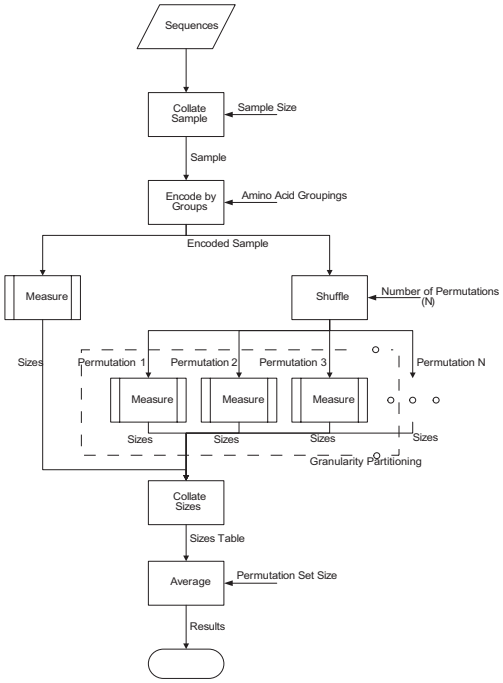

Figure 1. Compressibility Workflow

The Measure sub-workflow, shown in Figure 2, is comprised of the following steps. **gzip/ppmz Compression:** The input sample is compressed using the gzip or ppmz algorithm. **Measure Size:** The sample and its two compressed forms are measured to determine their respective sizes. **Collate Sizes:** The size data output from all Measure Size steps are collated into a single table.

3 Provenance Use Cases

In this section, we introduce some use cases that we have identified for the Protein Compressibility Experiment. For a more complete survey of provenance use cases and technical requirements, we refer the reader to [11].

Use case 1 *A bioinformatician, B, downloads sequence data of microbial proteins from the database RefSeq and runs the compressibility experiment. B later performs the same experiment on the same sequence data, again downloaded from RefSeq. B compares the two experiment results and notices a difference. B determines whether the difference was caused by the algorithms used to process the sequence data having been changed.* □

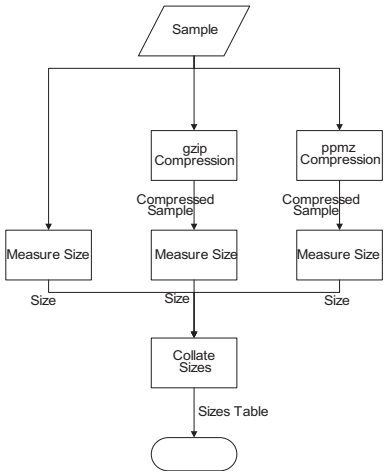

Figure 2. Measure Workflow

For instance, with use case 1, B could observe that the compression algorithms used in the compressibility workflow were configured differently in two runs of the experiment, and so produced differing results.

Use case 2 *A bioinformatician, B, performs an experiment on a FASTA sequence encoding a protein. A reviewer,* R*, later determines whether or not the sequence was in fact processed by a service that meaningfully processes protein sequences only.* □

The principle of amino acid group encoding described in Section 2 also applies to other types of sequences; for instance, in a nucleotide sequence, each codon triplet can replaced with a symbol representing a group of codons. A fundamental part of the experiment presented here is for symbols for the amino acids making up a protein sequence to be replaced by symbols for the groups to which they belong (i.e., the **Encode by Groups** activity in Figure 1). If a *nucleotide* sequence was accidentally used at this stage rather than an amino acid sequence, there would be no error in running the workflow because the symbols used for nucleotides are a subset of those used for amino acids. However, the experiment results would not be meaningful, because the groups used and workflow results compared against would all be particular to amino acids. In this case, we can say that, while the workflow is *syntactically* correct, it is *semantically* incorrect, which is what use case 2 aims to discover.

In order to support such use cases, a provenance system needs to provide the following features. It needs not only to identify which scripts are being invoked at every step of the execution, but also to make a copy of these, so as to be able to detect changes in their content from one execution to the other. It needs to identify scripts to sufficient detail that their intended semantics can be retrieved and compared. Finally, it needs to maintain a link between the inputs and the outputs of each workflow run in an accurate manner: it should be possible to determine which inputs were used to produce which output unambiguously from the provenance documentation, even if multiple workflows were run simultaneously.

4 The Challenge of Recording Provenance

In this section, we describe the approach adopted to implement the protein complexity workflow. We then discuss the challenges that such an application presents for integrating a provenance system: in particular, we analyse limitations of existing provenance systems.

This workflow offers great potential for massive parallelism given the vast number of input sequences and the large number of permutations to be considered. Thus, we have decided to adopt VDT (the Virtual Data Toolkit, www.cs.wisc.edu/vdt), which offers good possibility of scheduling over the Grid through the use of Condor. However, very quickly, our application has turned out to be a heterogeneous system, involving multiple technologies to run and compose applications (or services), as we now explain.

While compression methods such as gzip or ppmz can run directly from the command line, other advanced compression methods may be available as Web Services. Likewise, sequences provided as inputs are typically obtained by some form of remote procedure call to a remote database. Given that for a typical sample, compression takes of the order of 100ms, we have partitioned the processing of permutations into scripts that provided a sufficient granularity of computation (the order of 15 minutes) in order to offset the overhead of grid scheduling and file transfer (cf. the dashed box in Figure 1). Consequently, this specific application relies on a variety of methods to run and compose computations: binary executables, shell scripts, Web Services and VDT/Dagman workflows.

Heterogeneity is not specific to this application but is also present, e.g., in the LCG Atlas experiment, in which Athena and VDT coexist, or in myGrid, in which Web Services based workflows, plans for distributed queries and high-throughput cluster-based bioinformatics services are all seamlessly integrated. Such heterogeneity presents a number of challenges for recording provenance. A number of bespoke provenance systems have been designed for specific applications, but their lack of open-ness prevents the plugging of external execution environments. For instance, some application specific provenance support is provided for Geographical Information Systems [10]. In database systems, provenance study has focused on the data lineage problem [2], which determines the source data used to produce a data item. Buneman *et al.* [1] refine the concept of data lineage into "why" and "where" -provenance for which they derive a formal model they apply to both relational and XML databases. In our case, data is not necessarily contained in databases, and processing over such data is not necessarily expressed as database queries.

Given that multiple parts of our application can run under the control of different runtime systems, at separate locations, some mechanism is required for all these parts to contribute provenance data, so that it can be used seamlessly by programs that support use cases. However, no system supports such open-ness. For instance, while Chimera [5] provides for domain-independant provenance recording for VDL scripts, it offers no mechanism for submitting provenance information, which would have not been produced under the control of VDT. Likewise, provenance generated by myGrid is directly generated by the Taverna/Freefluo enactment engine, but no provision exists for other entities to submit provenance data [6]. Additionally, while Chimera and myGrid provide for provenance recording of activities they regard as atomic, finer grained recording may be required to follow the *actual* scientific process being executed.

Arbitrary pieces of data (such as scripts themselves) may have to be submitted to support use cases of section 3. Chimera records information that is useful for debugging (such as invoked commands, inputs and outputs), but does not allow arbitrary data to be submitted; alternatively, Chimera's virtual data catalog could be used to store arbitrary metadata stored, but it offers no generic mechanism for submitting metadata about a specific command execution. Likewise, myGrid allows metadata to be added to provenance traces, since they are encoded in RDF, but, again, no open mechanism is readily available to submit such metadata.

Finally, the different activities in a workflow should typically be grouped in different ways, with each grouping providing a well understood semantics. For instance, a workflow run is usually referred to as a "session", while a sequential succession of activities as a "thread". Such groupings are essential to analyse dependencies of activities while reasoning over provenance.

5 Provenance Architecture

In this section, we overview our proposed architecture for a provenance system. We take the broad view that grids are typically designed using a service-oriented approach; by

service, we do not intend to restrict ourself to a specific service technology (e.g, Web Services), but we rather mean to refer to components that take some inputs and return some outputs. Such services are brought together, usually via a workflow definition, to solve a given problem: hence, with such a broad definition, we see that BPEL, WSFL, VDL, Dagman's DAGs or Gaudi are all workflow frameworks capable of expressing the composition of services.

Based on this widespread design pattern, we have developed a categorisation of provenance for service-oriented architectures (SOA). In SOAs, a service is typically invoked by clients, but these may themselves act as services for other clients; hence, we will use the term *actor* to denote either a client or a service in a SOA. In general, the provenance of some data is the documentation of the process that led to the data. We refer to a given element of the documentation of process as a p-assertion [An asseration, by an actor, pertaining to the provenance of some data.].

We define two types of p-assertions *interaction p-assertions* and *actor state p-assertions*. In a SOA, interactions are, fundamentally, a client invoking a service. These interactions can be documented by recording interaction p-assertions about the inputs and outputs of the various services involved in generating a result. The second type of p-assertion we have identified is an *actor state p-assertion*, which is the documentation provided by an actor about its internal state in the context of a specific interaction. Actor state documentation is extremely varied: it can include anything from the workflow that is being executed to the amount of disk and CPU a service used in a computation. We note that both interaction p-assertions and actor state p-assertions are independent of the actual service technology used to run the application.

Provenance Architecture In order to support the capture and querying of provenance, we have specified a provenance architecture that takes into account a broad range of use cases [11]. Central to this architecture is the notion of a provenance store, which is designed to store and maintain provenance beyond the life of a Grid application. To allow distributed application parts to record provenance documentation in a store, we introduce PReP, the Provenance Recording Protocol, which specifies the messages that actors can asynchronously exchange with the provenance store in order to record their interaction and actor state p-assertions [7]. Beyond just preserving provenance, stores can also be queried to retrieve stored documentation, to be processed and reasoned over. A crucial aspect of this architecture is the clear organisation of the stored documentation around the fundamental categorisation of interaction and actor state p-assertions. In order to further structure these p-assertions in a manner that relates to execution, PReP and the underpinning provenance model also support a notion of group, which identifies well-specified associations of interactions such as sessions or threads.

While PReP identifies *how* the documentation of process should be recorded, it lets the implementor decide *when* to do so. This flexibility allows implementors to customise recording according to the application's needs: when the application requires provenance to be used immediately as execution occurs, p-assertions may be submitted synchronously with execution; alternatively, when provenance is used after application completion, then p-assertions may be recorded asynchronously so as to reduce recording overhead. We exploit the latter strategy in our implementation of the protein compressibility experiment.

PReServ Provenance Recording for Services (PReServ) is an Open Source Java based Web Services implementation of PReP, available for download from `www.pasoa.org`. The package includes a provenance store, client APIs and XML schemas for storing data in and retrieving data from the store implemented as a Web Service. PReServ runs as a servlet inside a Java Servlet container. For the evaluations, Apache Tomcat 5.0 was used.

Figure 3 shows the layered design of the PReServ store. Normally, a SOAP message is sent to PReServ to either record or query provenance. Based on the port that the message was sent to, the SOAP Message Translator strips off the HTTP and SOAP Headers and passes the contents of the SOAP body to an appropriate PlugIn, which must conform to the schemas distributed with PReServ. For example, the Store PlugIn handles messages in order to record data in the provenance store. The selected PlugIn then makes calls on the backend data store to generate an appropriate response for the received message. Currently, PReServ comes with in-memory, file system and database backends. Each of these backends implements the same API, the Provenance Store Interface. This abstraction makes it easy to integrate new backend stores without having to change already developed PlugIns and provides an API that maps directly to the PReP protocol specification. We note that all evaluations make use of a database backend based on the Berkeley DB Java Edition (www.sleepycat.com) as the database.

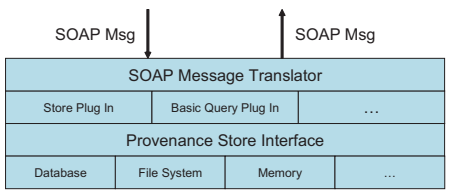

Figure 3. PReServ Layers

6 Evaluation

In this section, we present our preliminary evaluation of the PReServ-based provenance architecture. For this, we analyse the performance of the protein compressibility application with and without recording and of the different use cases.

Beforehand, we examine benchmarks of the PReServ Web Service itself, while running on a Windows XP PC with a Pentium P4, 2.8 Ghz, 1.5 GB RAM. It takes approximately 18 ms round trip to record one pre-generated message in PReServ. These tests were conducted with both the client and server running on the same host.

For our preliminary evaluation, we deployed VDT on a Redhat Linux 9.1, running under a VMWare virtual machine, itself running on a Windows XP PC with a Pentium P4, 2.8 Ghz, 1.5 GB RAM. The Provenance Store PReServ was deployed on a separate Windows XP PC, with the same hardware. PCs were connected by a 100Mb local ethernet. In the following section, we discuss this preliminary testing configuration and the significance of our results.

Recording Evaluation The purpose of this evaluation is to benchmark the overhead of recording p-assertions in a real scientific application. To this end, we executed the protein compressibility workflow of Figure 1, with sets of sequences, which when collated constituted samples of about 100Kb. For each sample, we considered an increasingly large number of permutations.

As indicated in Section 4, we grouped the execution of 100 permutations into a single script to increase the granularity of the activities to be scheduled by Condor. In order to provide provenance for the scientific experiment, both interaction and actor state p-assertions were recorded, for every single activity of the measure workflow (Figure 2), for every permutation (and not just for every script directly scheduled par Condor).

Figure 4 plots the overall execution time (measured by the time difference between the last and first activities in the protein complexity workflow), for increasing number of permutations, and for different configurations of p-assertion recording: *(i)* without recording p-assertions, *(ii)* with asynchronous recording, in which all p-assertions are accumulated locally in a file before being shipped to PReServ after execution, *(iii)* with synchronous recording by direct Web Service invocation of PReServ, and *(iv)* with synchronous recording with extra information being recorded as actor state p-assertions (such as script provenance to support use case 1). Our observations are as follows: *(i)* overall, the different execution times remain linear (each plot has a correlation coefficient greater than 0.99) with the number of permutations to be processed; *(ii)* accumulating p-assertions to be submitted asynchronously has an overhead over no provenance recording; *(iii)* asynchronous recording has an overhead smaller than synchronous recording; *(iv)* overall, the overhead of asynchronous performance recording remains less than 10%.

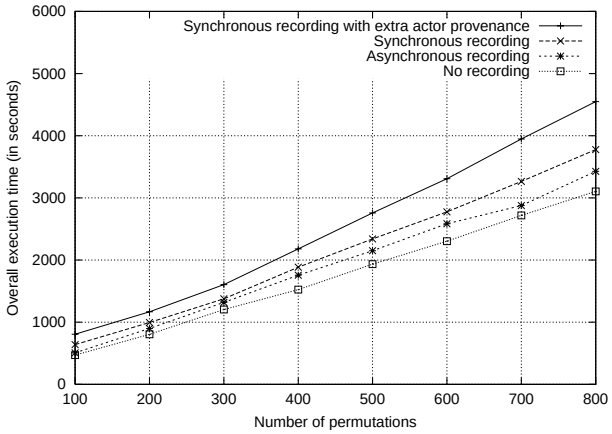

Figure 4. Recording Provenance

Like a scheduler requires a granularity coarse enough to offset the overhead of automatic scheduling, automatic recording of p-assertions has an acceptable cost if the granularity of activities is coarse enough. In this experiment, the run of a workflow for one 100Kb sample with 1 permutation takes approximately 4.5s and; each permutation involves the creation of 6 records and their submission (averaged over a long running workflows).

Execution Comparison In this section, we evaluate the performance of comparing the provenance of two data results as described in use case 1. After a set of workflow runs, each analysing one sample, the provenance store contains records of the service interactions. Each interaction record is made up of a record of the invocation message that occurred in the workflow, and actor state p-assertions containing the script for the service in that interaction. We categorise the (contents of the) scripts that workflow activities have used, so that the bioinformatician can determine whether the results of one workflow run differed from another due to a change in algorithm or configuration. The script contents are around 100 bytes each and are recorded in PReServ as actor state p-assertions. Categorisation is performed by querying each activity in the provenance store for actor state p-assertions containing the script and creating a mapping from each set of exactly equivalent scripts to the sessions (groups denoting workflow runs) in which that script is used for a given service.

As we analyse all activities in the provenance store, the time taken to perform the categorisation is dependent on the size of the store. In Figure 5 (which we use to display the

performance of both use cases), we plot the time to query the store for all relevant actor state p-assertions and perform a full comparison against the number of interaction records contained in the store. We observe a linear behaviour (the plot has a correlation coefficient greater than 0.99) with the size of the store; on average, it takes about 15ms to retrieve a script (through one store invocation) and map it.

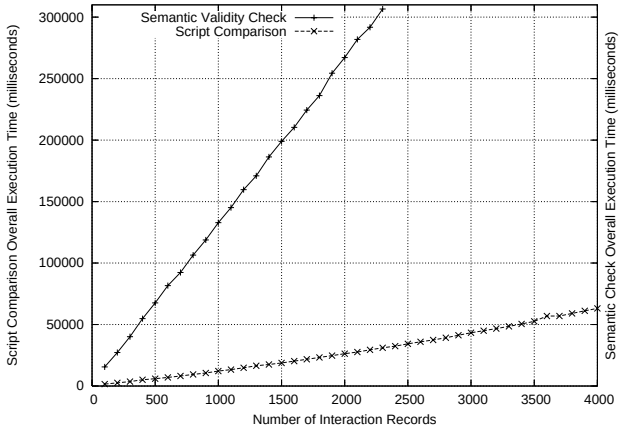

Figure 5. Execution Comparison and Semantic Validity

Semantic Validity In this section, we evaluate the performance of semantically validating a workflow execution as described in use case 2. We note that some of this validation could be performed statically by analysing the workflow script. However, this script may not always be available, or it may be expressed in a language for which we do not have a static analyser, or it even may not lend itself to a static analysis because runtime (or post-runtime in this case) analysis is required.

To support such a use case, we introduce a registry that contains semantic information for the different workflow activities. To this end, we use the Grimoires registry (www.grimoires.org), an extension of the UDDI registry, designed to support semantic annotations of service descriptions [12]. The registry provides an interface that supports metadata publication and metadata-based service discovery.

Practically, each workflow activity is described by a WSDL interface: we use here the abstract part of a WSDL interface to characterise the type of inputs or outputs taken by services[1]. Each message part (whether input or output) of each service operation is annotated by some metadata identifying its semantic type, which we have expressed in an ontology fragment for this specific application. The process of semantically validating an execution is as follows. Given a provenance trace for an execution that led to some data, the semantic type of each service output (obtained from interaction p-assertions and metadata stored in the registry) is verified to be equal to the semantic type of the service input it is fed into.

In terms of deployment, the registry, the provenance store and the semantic validator were all deployed on different PCs (with mentioned hardware characteristics), communicating over 100Mb ethernet. Figure 5 plots the overall execution time for semantic validation against the number of records in the provenance store. Again, we observe a linear behaviour (the plot has a correlation coefficient greater than 0.99), though a much higher time is needed for this experiment compared to the previous. For each interaction, we perform one call to PReServ and 10 to Grimoires; as seen from Figure 5, the slope of the semantic validity plot is about 11 times higher than the one for script comparison.

7 Discussion and Future Work

Our deployment of the provenance architecture in a bioinformatics application exhibits interesting properties. In the most optimised case, p-assertion recording may require just a few milliseconds to prepare a record to be temporarily stored in a file and submitted asynchronously to PReServ. Not surprisingly, if such processing is negligible compared to the average granularity of a workflow activity, p-assertion recording only brings a small overhead to overall application performance, which is offset by the very valuable benefits that it offers. Besides optimising recording, static analysis of workflows would be useful to prepackage some of the p-assertions to be recorded, leaving less to perform at runtime.

Virtual machines were adopted for virtualising Grid deployment, which is an approach that has been used by a number of other authors. [8, 3]. While some application slowdown was observed by running over VMWare, we note that p-assertion recording itself also suffers a similar slowdown. Hence, we conjecture that our results remain valid if similar benchmarks are run natively on a physical machine.

Our preliminary evaluation of the protein compressibility experiment was performed on a single machine. Given that workflows are highly parallel and expressed in VDT, we anticipate they should run on large scale clusters and grids. In such a context, PReServ may become a bottleneck when handling p-assertion submission requests. To combat such scalability concern, we are undertaking the design of a distributed version of PReServ, which would allow parallel submissions into several provenance store instances; additionally, documentation recorded in different stores should

[1] In this application, we only use an interface for *describing* a service; if we were to *invoke* a service, then a specific binding would be required, so that command line executables could be invoked by a framework such as WSIF (ws.apache.org/wsif).

be cross-linked to allow navigation; a facility is also required to consolidate data into a single provenance store.

All use cases have to be revisited in the light of a distributed version of PReServ, since queries potentially need to be run over multiple distributed stores. Finally, the role of the provenance store is to record p-assertions data, to support provenance queries, but also to act as a long term storage for provenance: support for curation of provenance data is therefore also required.

8 Conclusion

Large scale applications typically make use of multiple technologies to compose complex computations, which all have to contribute information about their execution. In this paper, we have proposed a technology independent way of characterising provenance, consisting of interactions between services, internal information about these services, and interaction groupings reflecting the order of execution. Additionally, we have introduced the idea of a protocol to submit p-assertions, allowing the multiple components of a computation to submit p-assertions, despite being distributed and possibly relying on different technologies. We then presented PReServ, a realisation of this protocol as a Web Service, which offers a persistent storage for provenance, an API to record provenance data and an API to query it. We have deployed PReServ in a bioinformatics application to study protein compressibility, which we made run under the Virtual Data Toolkit VDT; this application presents interesting use cases for provenance, which we have shown to be supported by our provenance architecture. We then studied the performance of provenance recording and reasoning. Our conclusion is that when an application is composed of activities with a high enough average granularity, the cost of recording p-assertions is largely offset by its benefits.

9 Acknowledgements

This research is funded in part by PASOA project (EPSRC Grant GR/S67623/01) and EU Provenance (IST 511085). The semantic validity use cases make use of the Grimoires Registry, funded by the Grimoires (EPSRC Grant GR/S90843/01) and myGrid (EPSRC Grant GR/R67743/01) projects. Thanks to Jens-S. Voeckler for his support for the VDT system. The Protein Compressibility Experiment is part of on ongoing investigation into the complexity of evolved sequences by Stefan Artmann and one the authors (KPZ).

References

[1] P. Buneman, S. Khanna, K.Tajima, and W. Tan. Archiving scientific data. In *Proc. of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM Press, 2002.

[2] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.

[3] R. Figueiredo, P. Dinda, and J. Fortes. A case for grid computing on virtual machines. In *Proceedings of the 23rd Internatinal Conference on Distributed Computing Systems (ICDCS 2003)*, 2003.

[4] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman Publishers, 1998.

[5] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, Edinburgh, Scotland, July 2002.

[6] M. Greenwood, C. Goble, R. Stevens, J. Zhao, M. Addis, D. Marvin, L. Moreau, and T. Oinn. Provenance of e-science experiments - experience from bioinformatics. In *Proceedings of the UK OST e-Science second All Hands Meeting 2003 (AHM'03)*, pages 223–226, Nottingham, UK, Sept. 2003.

[7] P. Groth, M. Luck, and L. Moreau. A protocol for recording provenance in service-oriented grids. In *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04)*, Grenoble, France, Dec. 2004.

[8] K. Keahey, K. Doering, and I. Foster. From sandbox to playground: Dynamic virtual environments in the grid. In *Proceedings of the 5th International Workshop in Grid Computing (Grid 2004),*, Pittsburgh, PA, Nov. 2004.

[9] K. Lanctot, M. Li, and E. h. Yang. Estimating dna sequence entropy. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 409–418, San Francisco, California, Jan. 9–11, 2000. ACM.

[10] D. Lanter. Design of a lineage-based meta-data base for gis. *Cartography and Geographic Information Systems*, 18(4):255–261, 1991.

[11] S. Miles, P. Groth, M. Branco, and L. Moreau. The requirements of recording and using provenance in e-science experiments. Technical report, University of Southampton, 2005.

[12] S. Miles, J. Papay, T. Payne, M. Luck, and L. Moreau. Towards a protocol for the attachment of metadata to service descriptions and its use in semantic discovery. *Scientific Programming*, pages 201–211, 2005.

[13] C. Nevill-Manning and I. Witten. Protein is incompressible. In J. Storer and M. Cohn, editors, *Proc. Data Compression Conference*, pages 257–266, Los Alamitos, CA, 1999. IEEE Press.

[14] G. Sampath. A block coding method that leads to significantly lower entropy values for the proteins and coding sections of haemophilus influenzae. In *Proceedings of the Computational Systems Bioinformatics (CSB'03)*. IEEE Computer Society, 2003.