Incremental workflow improvement through analysis of its data provenance

Paolo Missier
School of Computing Science
Newcastle University, UK
Paolo.Missier@ncl.ac.uk

Abstract

Repeated executions of resource-intensive workflows over a large number of runs are commonly observed in e-science practice. We explore the hypothesis that, in some cases, provenance traces recorded for past runs of a workflow can be used to make future runs more efficient. This investigation is an initial step into the systematic study of the role that provenance analysis can play in the broader context of self-managing software systems. We have tested our hypothesis on a concrete case study involving a Chemical Engineering workflow deployed on a cloud infrastructure, where we can measure the cost of its repeated execution. Our approach involves augmenting the workflow with a feedback loop in which incremental analysis of the provenance of past runs is used to control some of the workflow steps in subsequent executions. We present initial experimental results and hint at future improvements as part of ongoing work.

1 Introduction

Computing infrastructures for e-science are increasingly capable of tracing their data products through the complex processes, sometimes encoded as workflows, by means of which those products are created. Examples of such *provenance-aware* systems include the Taverna [11], Kepler [3], Pegasus [5], VisTrails [4], eScience Central [14], and Galaxy [13] workflow systems, amongst others. Despite growing interest in provenance management components, however, only a few case studies of "provenance in use" in concrete settings can be found in the literature (one such case is the use of provenance in Galaxy to enable *reproducible science* [10]). We are interested in exploring the potential for exploitation of large provenance corpora, accumulated by any of these systems during multiple workflow executions, to deliver added value to users (the scientists) as well as to infrastructure providers. More specifically, in this paper we focus on the common e-science scenario where the same resource-intensive workflow is repeateadly executed a large number of times on different inputs. In this setting, we begin to explore the hypothesis that analysis of the traces accumulated during past runs of a workflow may lead to more efficient runs of the same workflow in the future.

Our approch involves adding some form of *adaptive control* to an existing workflow, with provenance analysis at its core. In a sense, the idea of adding adaptivity to processes situates our work within the broader scope of *self-adaptive* systems, a term originally coined by IBM in 2001 [8] to denote software systems that can dynamically adapt their behaviour in response to changing conditions in the inputs or in their environment (changes in capacity of the infrastructure, for example). Since then, a number of adaptive systems have been built [7]. These systems are characterised by a common architectural pattern, namely the MAPE-K loop (Monitor, Analyse, Plan, Execute, Knowledge), depicted for instance in [9]. In this pattern, an element of the system is coupled with an *autonomic manager* which can monitor the element's behaviour and react to certain events by acting upon the element itself or its environment. For example, such a manager could monitor a web server's response time and decide to increase the pool of active servers to prevent the performance from degrading.

Our scenario differs slightly from the standard self-adaptive system setting, in that it involves repetitions of the same process over time, rather than a long-lived system. Despite this difference, however, our approach can still be described in terms of the MAPE-K loop, where the autonomic manager consists of a provenance collector (the Monitor) and analysis component (the Analyser), as well as a new *recommender* task that is added to the workflow (the Executor). In turn, the recommender is informed by a Knowledge component that encapsulates an encoding of all past provenance traces.

This is where the similarity with traditional self-

adaptive systems ends, however, because adaptive approaches generally assume that changes that require reaction occur in some predictable way over time. In contrast, in our setting future inputs to the same workflow may be completely unrelated to past *recent* inputs, whilst they may instead be similar to input datasets observed at any time in the remote past. For the same reason, past research in the area of *self-adjusting computation* [2] is only superficially attractive here, as it is based on the assumption that some portion of past computations can be salvaged when its corresponding inputs change slightly. Instead, the hope to encounter again inputs that have been observed at some arbitrary point in the past suggests that our problem can be cast in terms of the *Case Based Reasoning* (CBR) framework. CBR [1] is based on the principle that a collection of previously solved problems, which are stored in the *case base*, can be leveraged to tackle a new, but similar problem.

Following this idea, in this paper we describe a case study where a Chemical Engineering workflow (Sec. 2), deployed on the eScience Central infrastructure, is executed a large number of times (over 10,000 in our initial experiments, with more input datasets becoming available) and provenance traces are generated for each run. We present our CBR-based approach in Sec. 3. Our case base at any point in time consists of an encoding of the entire collection of traces up to that point. As a new execution progresses, its intermediate data products at some critical point in the workflow become available as part of a new provenance trace.These are matched against existing cases, in order to predict the effectiveness of some of the workflow tasks that follow. The *recommender* uses the analysis to predict the effectiveness of those tasks, and to selectively enable or disable some of them. As part of our initial experiments (Sec. 4), we show how this adaptive loop can lead to more efficient computations in the long term. However, our early results indicate that more work is needed to fine-tune the case base and reasoning components of the system, as discussed at the end of Sec. 4.

The case study represents one instance of a general workflow pattern that lends itself well to our adaptive optimisation, making our early results interesting beyond the narrow scope of the example. Optimisation of this class of workflows is especially important when the tasks are deployed on cloud nodes, where their execution incurs a monetary cost. Ultimately, our goal is to offer users well-defined trade-offs between cost of execution and the accuracy / completeness of the results.

2 QSAR and the DiscoveryBus workflow

One of the computational problems at the forefront of current Chemical Engineering research involves establishing correlations between the structure of molecular compounds and some of their associated activities, including toxicity, solubility, etc. The approach followed in the OpenQSAR project[1] involves training multiple machine learning schemes, using a dataset consisting of compounds that are representative of a family of structurally homogeneous molecules, for instance `Wombat_SID2353_AID1_human_thrombin`. This learning activity produces one or more models that are trained to predict molecular activity of new compounds that belong to the same family, based on a selection of their structural features. The experts' experience suggests that different learning schemes, including linear regression, neural nets, and decision trees amongst others, perform differently on different compound families, in terms of their resulting predictive power. Thus, the chemical engineer's strategy is to apply multiple schemes to an input dataset, and then select those models that perform best on that particular dataset. In OpenQSAR, this strategy is encoded as a workflow, called DiscoveryBus, which scientists can apply systematically to a large number of compound families [6]. This makes DiscoveryBus an examplar of a class of highly repetitive workflows that include a number of resource-intensive tasks, i.e., the learning schemes. An abstract rendering of one run of the workflow (denoted by the superscript i), is shown in Fig. 1.

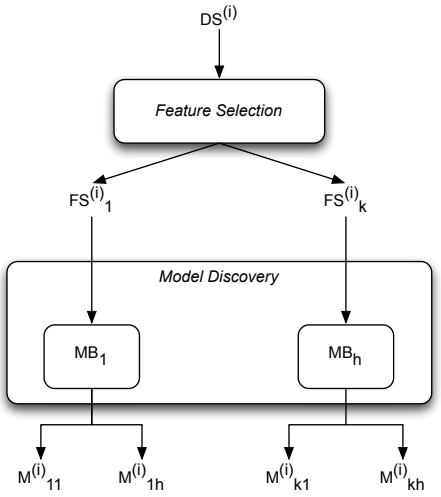

Figure 1: Sketch of Discovery Bus workflow. $DS^(i)$ denotes one about 10,000 data series. The number k of selected ferature sets varies with i. In the current implementation, there are only $h = 4$ model builders, leading to kh models for each input data series.

[1] `http://www.openqsar.com/`. QSAR stands for " Quantitative structure-activity relationship".

The i-th run of DiscoveryBus takes as input a target activity type a, such as `toxicity`, with associated domain values, e.g. `low`, `mid`, `high` and a data series $DS^{(i)}$, consisting of specific molecules. The workflow generates a potentially large number of models. Two sources of uncertainty contribute to this proliferation. Firstly, as mentioned, the workflows computes a set of H predictive models, $M_1, M_2 \ldots M_H$, such that each M_i is capable of predicting the activity value for property a of a new compound d that belongs to the same family $DS^{(i)}$. Each of the H models is generated using a different learning scheme, each implemented by a different *Model Builder* task, $MB_1 \ldots MB_H$ ($H = 4$ in the current implementation, while additional builders may added at a later time).

The second factor is the choice of features vectors used as input to each of the builders. Each builder operates on a set of features extracted from the raw input $DS^{(i)}$. A large number of features can potentially be extracted from the molecular structure (over 4,000 features are documented in the literature). To reduce this space, task *Feature Selection* first generates the features, and then selects subsets of independent features by correlation analysis. This task, however, generates a set of $k^{(i)} > 1$ candidate combinations of features, $FS_1^{(i)} \ldots FS_k^{(i)}$, rather than a single set of features, with each combination potentially leading to useful models. Each $FS_j^{(i)}$ includes the actual vectors for each $d \in DS^{(i)}$, and for a specific subset of features. This results in the generation of $k^{(i)}H$ models for each run. Associated to each model is a collection of performance metrics, which ultimately are summarized into a simple accept/reject value q for the model, referred to as the *quality* of the model (this can be generalised in the future to a more complex quality domain).

3 Provenance-driven incremental process improvement

The history of about 10,000 actual runs performed in the context of OpenQSAR, each including $H = 4$ builders and $k^{(i)} = 25$ feature sets on average, has been captured into a provenance database. This translates into a total of about 1 million generated models over time. What makes DiscoveryBus interesting as a testbed for our adaptive process improvement experiment is that *less than 10%* of these models turns out to be acceptable, i.e., exhibits sufficient predictive power to be used reliably. Our objective is therefore to reduce the number of generated models, while still generating the majority of the good models that exist in the models space. Each run i has a set of provenance graphs associated to it, one for each builder $h : 1 \ldots H$ and for each set of $k^{(i)}$ feature sets. Using OPM notation [12], each graph fragment is of the form:

$$M_{jh}^{(i)} \overset{WasGeneratedBy}{\rightarrow} MB_h \overset{used}{\rightarrow} FS_j^{(i)} \quad (1)$$

$$FS_j^{(i)} \overset{WasDerivedFrom}{\rightarrow} DS^{(i)} \quad (2)$$

Crucially, rather than performing a *post hoc* analysis of the entire provenance corpus, we have used this collection to *simulate* an incremental learning process in which only the provenance graphs accumulated up to run i are used to steer run $i+1$ of the DB computation, in an adaptive fashion. Furthermore, note that we are going to need a complete provenance trace (as opposed to the simpler input/output dependencies), as our predictors of model builder quality are the FS, which are intermediate results observed half-way through the computation.

Our approach is designed to be applicable to the generic workflow pattern shown in Fig.2, in which n *experts* are simultaneously consulted using the same input dataset, and their response is then combined (task *results merge*). In the DiscoveryBus case study the steps upstream from the pattern generate the feature sets, the selection step simply distributes them to the experts, which represent model builders, and the merge task filters out the unacceptable models. We add a new *recommender* step to the workflow, a task designed to improve the efficiency of the process, measured as the ratio of the number of good models to the total number of generated models. The recommender takes as input a feature set FS and returns a list of builders, sorted in priority order according to the decision logic described in detail below. Such logic is based upon a representation of the provenance graph fragments that accumulate at the end of each run, which are captured and stored by the new *online provenance analysis* task.

3.1 Time-accuracy trade-offs

In this enhanced version of the workflow, the expert selection task invokes the experts in the order suggested by the recommender. Furthermore, the *number* $1 \leq n \leq H$ of builders that are invoked is a configurable parameter that provides a way to tune the efficiency/accuracy tradeoff offered by the recommender: a greedy selector will only try the first builder ($n = 1$) and is prepared to miss out on good models that can potentially be produced by the builders that follow. This selector is optimistic, as it assumes good recommendations, in the sense that for any input FS it expects to find the good models, if any, by using the first builders in the list. A prudent selector, on the other hand, is prepared to try more of the H builders, in turn, thus ensuring that no good models will be missed (high accuracy), but possibly at the cost of exploring the entire space of models.

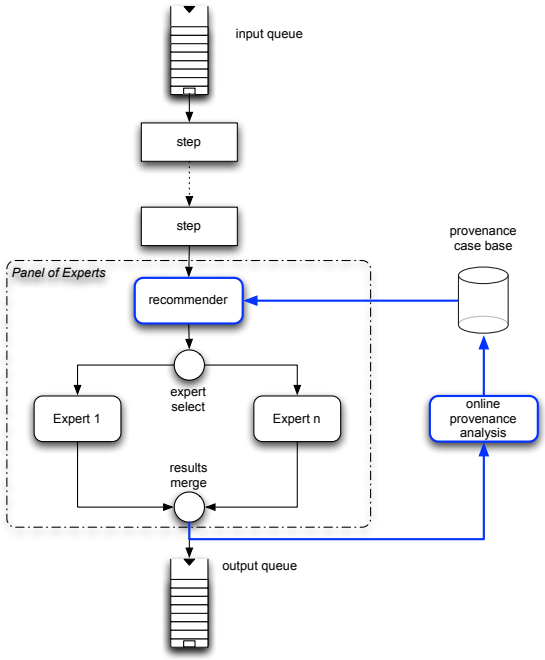

Figure 2: Sketch of a provenance-enhanced workflow including a recommender task for steering a "expert consult and results merge" pattern

3.2 Recommender algorithm

The recommender follows the general *Case Base Reasoning* paradigm [1], based on the principle that a collection of previously solved problems, which are stored in the *case base*, can be leveraged to tackle a new but similar problem. The new problem may not find an exact match in the case base, but rather, a similarity function is defined to compute the best possible match. Normally the existing solution(s) are likely to require an adaptation step rather than being directly applicable. In our setting, one case consists of one provenance graph fragment as defined in (1). A case base consists of a set of mappings of the form $FS \rightarrow QM$, where FS is a feature set that has been observed in at least one run, and QM is a *quality matrix* that encodes the success history of each model builder $MB_1 \ldots MB_H$ in the workflow when it is applied to FS:

$$QM_{FS}[MB_h] = \langle G, B \rangle$$

where G (resp. B) is the number of times MB_h has been observed to produce a good (resp. bad, i.e., $q = 0$) model when applied to input FS. The builders' success rate $\frac{G}{G+B}$ in the context of QM_{FS} induces the desired partial order. The matrices are updated by the *provenance analysis* task whenever a new model is generated, and consulted by the recommender whenever a new feature set is generated, as follows.

Thus, in this setting each case is encoded as a quality matrix, and the case base is indexed by the set of FS. In the initial version of our algorithm, we match each FS representing a new case *exactly* to one entry in the case base, i.e., lookup into the case base is by set equality. In the next section we discuss improvements to this baseline algorithm, in which set equality is replaced by a more general set similarity function. Each quality matrix is updated and retrieved as follows.

Update. When a new provenance fragment is recorded during run i, i.e., for each $FS_j^{(i)}$ computed by *Feature Selection* and for each new model generated by one of the builders MB_h, the corresponding $QM_{FS_j^{(i)}}$ is updated (or created, if this is the first occurrence of $FS_j^{(i)}$):

$$QM_{FS_j^{(i)}}[MB_h] = \begin{cases} \langle G+1, B \rangle & \text{if } \langle M_{jh}^{(i)}, 1 \rangle \overset{WGBy}{\rightarrow} MB_h, \\ \langle G, B+1 \rangle & \text{if } \langle M_{jh}^{(i)}, 0 \rangle \overset{WGBy}{\rightarrow} MB_h. \end{cases}$$

Lookup. The set of all QM is indexed by feature set. Every time a new FS is generated, the recommender simply looks up QM_{FS} and returns all the builders, partially ordered by their success rate. It is then up to the *expert selector* to decide how many of those builders will be invoked, depending on its time/accuracy setting. Note that we allow the recommender to provide a *null* response when the FS is used for the first time. This is important in practice, as the space of FS that are found in the provenance database is rather sparse, so that the majority of FS is only used once (making its QM useless). We will return to this point in the next section.

4 Experimental setup and early results

We now present early experimental results obtained by using this baseline recommender algorithm. To test its effectiveness, we have used the historical provenance database produced by OpenQSAR to "play back" the history of the runs and thus simulate the incremental refinement of the quality matrices associated with each FS over the entire runs history, as described in the previous section. For this, the entire provenance database is scanned in the natural order of its runs. Each run i is broken down into provenance records of the form $\langle FS_j^{(i)}, MB_h, M_{jh}^{(i)}, q \rangle$ for some j, h as in (1) For each such record:

1. the recommender provides a list of builders, based on the current content of the case base *prior to adding this record to the case base* (because in reality the outcome of this builder is not known to the recommender at this stage);

2. if $q = 1$, then a hit for the recommender is recorded if any of the first (in the partial order sense) n builders is MB_h, and a miss is recorded otherwise. Here n is the trade-off parameter from Sec.3.1. If $q = 0$, no hit or miss are recorded and the recommender's success rate is unchanged.

3. $QM_{FS_j^{(i)}}[MB_h]$ is updated in the case base and this outcome becomes available to the recommender.

In practice, the simulator records the success rate of an expert selector that follows the recommendations. We generally expect to observe an increase in recommendation accuracy as the sequence of runs progresses. Importantly, however, we must distinguish the *net* success rate, i.e., the number of number of hits as a fraction of the number of recommendations *given*, from the *gross* success rate, as a the number of hits over the total number of recommendations *requests* received. The latter include the requests for which the recommender abstained, not having sufficient information in the quality matrix to provide a meaningful list of builders. Net accuracy is reported in Fig. 4, for the four possible values of the trade-off parameter n (`max-attempts`). As we can see, the net success rate stabilises between 40% and 80% quite early on in the runs sequence, depending on the value of n, and it is predictably higher when we allow for more of the builders down the list to be used. Gross success rate, on the other hand, is disappointingly low (Fig.4). The main reason for this low rate is that the vast majority of the FS generated by the workflow are only used once, as the histogram in Fig. 5 illustrates, making the corresponding quality matrices uninformative.

Our ongoing work is currently focused on improving the gross success rate, by increasing the density of the FS space. We hope to achieve this by incrementally clustering over the space of FS (clustering is incremental because new FS are added with each new run). We can then replace naïve set equality with a similarity metric, such as the Jaccard set similarity $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$. In this improved setting, one quality matrix is associated to each cluster, rather than to each individual FS. We are specifically experimenting with hierarchical clustering, which provides control over the density of the FS space and is therefore ideal for exploring the trade-offs between cluster density and effectiveness of the corresponding quality matrices.

5 Summary and Extensions

In this paper we have described our initial exploration into the hypothesis that provenance of workflow-generated data can play an important role in the context of self-managing systems. We are focusing specifically on the application of data mining and automated learning from a large but structurally very simple corpus of provenance traces, observed through a large number of runs of the same workflow. We have presented one initial application of this idea, by showing that provenance traces can be encoded into a case base that can be used for the incremental improvement of a repetive process. Our case study is focused on a resource-intensive Chemical Engineering workflow. A provenance-driven recommendation step is added to the workflow in order to prune the space of potential solutions that the workflow needs to explore during any given run. Our initial experimental results suggest that intermediate data products found in the provenance traces make for potentially interesting predictors of output quality, however more work is needed to exploit them effectively.

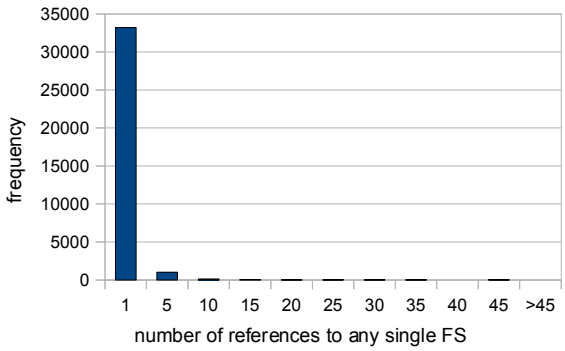

Figure 5: The vast majority of each FS are only used once, making for a poor case base when exact FS match is used for case lookup.

References

[1] AAMODT, A., AND PLAZA, E. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications 7*, 1 (1994), 39–59.

[2] ACAR, U. A. Self-adjusting computation: (an overview). In *Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation* (New York, NY, USA, 2009), PEPM '09, ACM, pp. 1–6.

[3] ALTINTAS, I., BARNEY, O., AND JAEGER-FRANK, E. Provenance Collection Support in the Kepler Scientific Workflow System. In *IPAW* (2006), pp. 118–132.

[4] CALLAHAN, S. P., FREIRE, J., SANTOS, E., SCHEIDEGGER, C. E., SILVA, C. T., AND VO, H. T. VisTrails: visualization meets data management. In *Procs. SIGMOD* (2006), pp. 745–747.

[5] DEELMAN, E., SINGH, G., SU, M.-H., BLYTHE, J., GIL, Y., KESSELMAN, C., MEHTA, G., VAHI, K., BERRIMAN, G. B., GOOD, J., LAITY, A. C., JACOB, J. C., AND KATZ, D. S. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming 13*, 3 (2005), 219–237.

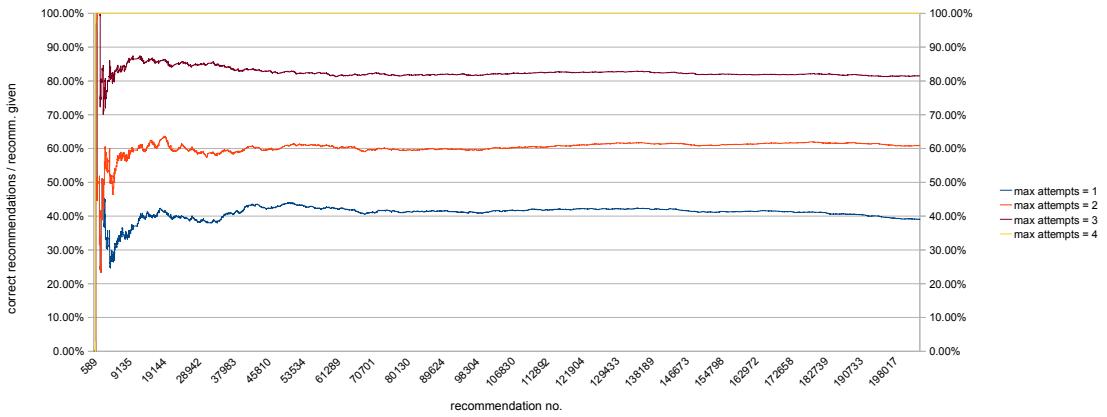

Figure 3: Increase in incremental net accuracy

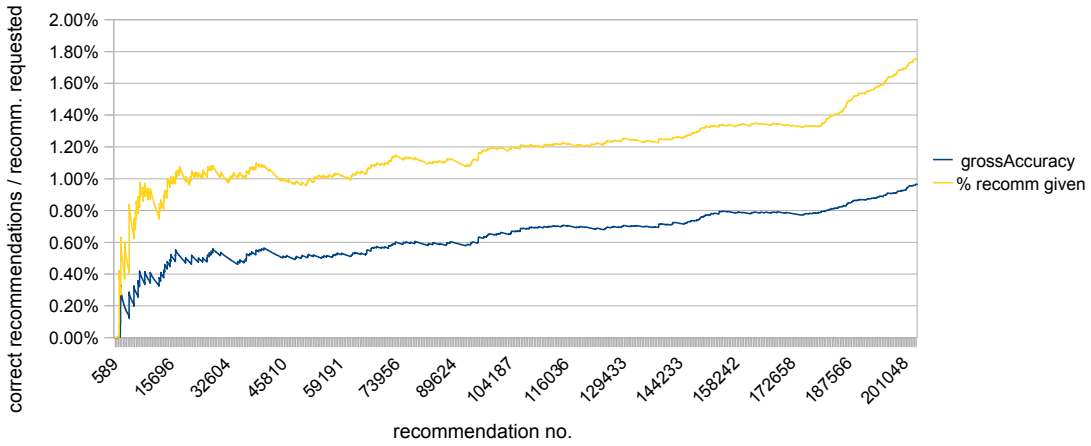

Figure 4: Increase in incremental gross recommendation rate

[6] HIDEN, H., WATSON, P., WOODMAN, S., AND LEAHY, D. e-Science Central: Cloud-based e-Science and its application to chemical property modelling. Tech. rep., CS-TR-122, School of Comp. Science, Newcastle University, 2011.

[7] HUEBSCHER, M. C., AND MCCANN, J. A. A survey of autonomic computing - degrees, models, and applications. *ACM Computing Surveys CSUR 40*, 3 (2008), 1–28.

[8] IBM. An architectural blueprint for autonomic computing. Tech. rep., IBM, 2011.

[9] KEPHART, J. O., AND CHESS, D. M. The Vision of Autonomic Computing. *IEEE Computer 36* (2003), 41–50.

[10] MESIROV , JILL, P. Accessible Reproducible Research. *Science 327* (2010).

[11] MISSIER, P., PATON, N., AND BELHAJJAME, K. Fine-grained and efficient lineage querying of collection-based workflow provenance. In *Procs. EDBT* (Lausanne, Switzerland, 2010).

[12] MOREAU, L., CLIFFORD, B., FREIRE, J., FUTRELLE, J., GIL, Y., GROTH, P., KWASNIKOWSKA, N., MILES, S., MISSIER, P., MYERS, J., PLALE, B., SIMMHAN, Y., STEPHAN, E., AND VAN DEN BUSSCHE, J. The Open Provenance Model — Core Specification (v1.1). *Future Generation Computer Systems 7*, 21 (2011), 743–756.

[13] NEKRUTENKO, A. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology 11*, 8 (2010), R86.

[14] WATSON, P., HIDEN, H., AND WOODMAN, S. e-Science Central for CARMEN: science as a service. *Concurrency and Computation: Practice and Experience 22*, 17 (2010), 2369–2380.