Towards Complete Tracking of Provenance in Experimental Distributed Systems Research

Tomasz Buchert, Lucas Nussbaum, Jens Gustedt

Towards Complete Tracking of Provenance in Experimental Distributed Systems Research

Tomasz Buchert[1], Lucas Nussbaum[1], and Jens Gustedt[2]

[1] Inria, Villers-lès-Nancy, France
Université de Lorraine, LORIA, France
CNRS, LORIA - UMR 7503, France
{tomasz.buchert, lucas.nussbaum}@loria.fr
[2] Inria, Villers-lès-Nancy, France
Université de Strasbourg, France
CNRS, ICube - UMR 7357, France
jens.gustedt@inria.fr

Abstract. Running experiments on modern systems like supercomputers, cloud infrastructures or P2P networks became very complex, both technically and methodologically. It is difficult to re-run an experiment or understand its results even with technical background on the technology and methods used. Storing the provenance of experimental data, i.e., storing information about how the results were produced, proved to be a powerful tool to address similar problems in computational natural sciences. In this paper, we (1) survey provenance collection in various domains of computer science, (2) introduce a new classification of provenance types, and (3) sketch a design of a provenance system inspired by this classification.

1 Introduction

Computers are becoming faster and more powerful, but our understanding is not advancing accordingly. On the contrary, since the systems become more and more complex one can argue that we know less and less about them. This discrepancy is disconcerting and calls for action in almost all domains of computer science.

Experimental research in distributed systems is especially exposed to this problem. The systems under study are complex, built from similarly complex software and hardware which interact in unexpected ways. Often the scale of experiments is large and even their execution is challenging, as is the understanding of a particular system or drawing scientifically valid conclusions. A platform may suffer from intermittent or fatal failures which should not go unnoticed. The number of relevant factors and the level of complexity has long surpassed our capacity to reason about such systems as a whole.

The complexity of the systems is not the only difficulty, however. Quite often the *description* of processes (including descriptions of scientific experiments) that run on them is complex, incomplete or even erroneous. This is another menace to *reproducibility* which is generally considered a hallmark of science.

Among the techniques that may improve both understandability and reproducibility of scientific research is *provenance*. Traditionally understood as information about origins and/or a chain of custody of a historical object, it has found another meaning in computing and science as a representation of origin and transformation of a given data object during computation. In this sense it is, in fact, a form of documentation. It improves understandability and reproducibility by tracking how the processes transform data and by capturing the context where these processes take place.

However, obtaining useful provenance information is not an easy task. The problems are conceptual (e.g., what should be tracked and to which level of detail?) and technical (e.g., how to store and query provenance information efficiently?). Collection of provenance may be in conflict with performance or even correctness of the system by inadvertently changing its behavior.

This paper makes three contributions. First, we analyze provenance collection techniques in computer science with the intention to improve their use in experimental distributed system research. Second, building upon the previous observations, we classify provenance into three distinct but interrelated types. Finally, we design a provenance system that follows this distinction and can provide answers to a range of queries.

The paper is structured as follows. In Section 2 we make our first contribution by making a thorough analysis of provenance in various domains. In Section 3, as another contribution, we propose a new classification into three types of provenance. Then, in Section 4, we make our third and last contribution by considering implications of this classification for experimental distributed systems research and sketch a design of a provenance system. Finally, we draw final conclusions and describe our future work in Section 5.

2 Provenance in computer science

In this section, we look at provenance as an object of study on its own, and then gradually narrow the domain and discuss its use and support in general computing, scientific workflows, control-flows and, finally, in experimental research in distributed systems.

2.1 General provenance

In this work, *provenance* is a collection of metadata associated with a run of a computational process that provides *any* kind of useful information as to how it was executed. This is a much broader term than *data provenance* which is a prevailing notion of provenance in computational life and earth sciences. Collection of data provenance is an active domain of research that meets much success in computational natural sciences [30].

Provenance can be *prospective* (i.e., obtained via static analysis) and *retrospective* (i.e., obtained postmortem) [12,13]. Additionally, one may differentiate

by the level of abstraction that provenance provides [2]. Four levels of provenance can be distinguished: *L0* (abstract experiment description), *L1* (service instantiation), *L2* (data instantiation) and *L3* (run-time provenance) of increasing precision and decreasing level of abstraction. Formal approaches to the representation of provenance [9,25,27], and generic standards for interchange of provenance information[3] have been proposed.

Efficient storage and querying of provenance information (which may be voluminous) is another important aspect. Using efficient representation based on the type of provenance is a standard approach, among other techniques [4].

A common way to construct and evaluate provenance systems consists in defining *queries* that the provenance has to answer [9]. Such a use-case driven approach is common in the domain as is shown by provenance challenges evaluating capabilities of provenance systems [26].

Hierarchical logging, which is essential to our approach to track the provenance of experiments, is often used to provide a way of looking at series of events in a way that would be otherwise difficult with a linear representation. Recently, *systemd*[4] benefited from this approach to improve logging of Unix services.

From this overview we conclude that there are numerous aspects of provenance and fragmented initiatives to provide it. The lack of general provenance tracking is mainly due to different requirements imposed by different domains. In the next sections, we will observe how provenance collection is addressed in different domains of computer science. To this end, we turn to provenance in general computing, scientific workflows, control-flows and in distributed systems research.

2.2 Provenance in general computing

In this section, we explore how provenance is provided in a general context (programming languages, scientific computing, data analysis, etc.). Provenance in general computing is rarely addressed, at least explicitly. First, provenance collection always incurs overhead that may make it unfeasible to use (e.g., in high-performance computing). Moreover, each subdomain of computing calls for a different approach and therefore can be impractical and tedious to implement in each case. We will see that provenance is, with some exceptions, often addressed in *ad hoc* manner and in a very limited sense.

Some notions of provenance in database systems has been proposed [8], the most common describing relationship between a data source, a query and the results of query execution.

Software documentation is a form of prospective provenance information that explains how the software works (or should work). *Literate programming* [22] proposes to have a verbose, natural-language description interwoven with code in a single document. Similar initiatives have been proposed in the scientific context (e.g., *literate experimentation* [31]).

[3] http://www.w3.org/TR/prov-overview/
[4] http://freedesktop.org/wiki/Software/systemd/

A useful source of provenance is provided by *instrumentation* and *monitoring*. Deep instrumentation or monitoring may be intrusive for the execution, and change the behavior of the studied system or even cause it to malfunction.

The history of how experiments evolved over time is also a form of provenance and is generally provided by version control systems (Git, Subversion, Mercurial, etc.). These systems trace the content of individual files and have no semantic knowledge about the whole system. Many systems, programming languages being a prime example, offer therefore language-specific *software archives* that host code that can be referenced (e.g., PyPI for Python or Hackage for Haskell). Studies of large, language-agnostic software repositories have been done as well, showing interesting aspects of long-term software evolution (e.g., Debsources [7]). Some retrospective studies trace authorship of modern BSD systems, as far as 40 years into the past [32].

There are solutions that build on version control systems and aim at automated capture of experiment context for easier reproducibility of research [14]. More recently, researchers propose experimental workflows based on Git branching model *and* literate programming with Org-mode [33].

2.3 Provenance in scientific workflows

Scientific workflows describe the set of tasks needed to carry out a computational experiment [16]. Their role usually consists in carrying out the computation using a given infrastructure (e.g., a computational grid or virtual machines in a cloud), but without going into details about how exactly these operations are executed. Therefore scientific workflow systems provide a high-level abstraction of computing and are used even by non-technical researchers. The efficient mapping, scheduling and execution of scientific workflows is a vivid domain of research.

The standard representation of scientific workflows uses acyclic *data-flows* to describe transformations of input data in a structured way. The acyclic graph structure implies a natural way to collect data provenance by workflow systems. The data-centric nature of scientific workflows is a well-known fact: it has been observed that the *data preparation* (i.e., initial transformation of input data to useful representation) accounts for more than 50% of workflow structure [20].

The history of how experimental workflows evolved is rarely tracked, with some exceptions. For example VisTrails [18] enables to backtrack from a failed approach by storing historical changes in a tree.

Similarly, details of the underlying platform are also rarely collected. Since the premise of scientific workflow systems is to abstract the details of the computing platform away and still obtain qualitatively equivalent results, this is understandable. This type of provenance can be still useful, for example for debugging, but is not essential.

Examples of scientific workflow systems include Kepler, Pegasus, Taverna, and VisTrails; see [34,36] for surveys of scientific workflow systems. The details of provenance support in scientific workflow systems are explored thoroughly [19].

2.4 Provenance in control-flows

For the purpose of this article, we define *control-flows* as workflows consisting of a set of activities that are performed under causal, temporal and spatial constraints to achieve a specific goal, such as, in the context of this article, the collection of experimental results. This definition is closely related to the one of *business processes* in Business Process Modeling (BPM) [23]. The differences between control-flows and data-flows are studied quite extensively [3,24], including the expressiveness of both formalisms [10]. The most important distinction is that data-flows are data-centric, contrary to control-flows.

The provenance collection in BPM and control-flows does not seem to be very much explored yet. This may be due to mentioned difficulties and due to proprietary nature of many BPM systems. To our knowledge, this article may be the first to explore provenance tracking in control-flows to a larger extent.

2.5 Provenance in experimental distributed systems research

Research in distributed systems developed a wide range of methods to tame the complexity associated with experimentation. These methods can be grouped into four methodologies: simulation, benchmarking, emulation and *in-situ* experiments [21]. The *in-situ* methodology, which we focus on in this work, consists in running a real system on a real platform, and arguably requires the most extensive provenance coverage among all methodologies.

There are various solutions that control executions of *in-situ* experiments on real platforms (e.g., Plush [1], OMF [29]). The support for provenance tracking in these tools is almost nonexistent [5].

Recording and subsequently restoring the state of platform configuration is another aspect of provenance addressed to some extent by system configuration management tools like Puppet, Chef or Salt. NixOS [17] takes a more generic approach of declarative and stateless description of full system.

3 New classification of provenance

As has been observed above, there are many different ways to provide provenance information and methods differ between domains. In this section, we will observe that for any form of computation executed on a system like grid, cloud, or any computing platform, the general provenance can be split into 3 different types. More precisely, we will show that apart from the *provenance of data* two other types of provenance exist: the *provenance of description* and the *provenance of process*, and that all three are useful and even necessary for a complete provenance system. Although this article concentrates on the domain of experimental distributed systems research, the discussion in this section is general and applies to scientific workflow systems and even outside the scientific context.

To explain the existence of these three types of provenance, we make the following observations about *entities* that are present in an arbitrary computation

Table 1. Summary of the three proposed provenance types their position in existing classifications. L2 provenance is not present in most distributed research experiments.

Name	Entities	Moment of collection ([13])	Level ([2])
Data	Results, monitoring data, platform configuration	Retrospective	L3 (and L2, if present)
Description	Experiment description, platform specification	Prospective	L0 and L1
Process	Runtime information	Retrospective	L3

on a computing platform. First, from an abstract point of view, there are two elements necessary to run it: its abstract *description* and a physical platform. The platform consists of physical *machines*, network *equipment*, installed *software* and other details. The execution of the given computation may have useful *runtime* information and may produce, receive or transform arbitrary *data*. All these objects can be separated into 3 different classes (see Table 1 for a summary).

Note that L2 provenance (data instantiation) is not covered, since experiments we focus on do not take raw data as input. However, if need be, L2 provenance can be classified under the provenance of data.

Provenance of data is information on how data objects were created and transformed during the execution of the given computation. This is a type of provenance that is largely synonymous with provenance itself due to its successful application in scientific workflow systems. Moreover, data provenance is *implied* by the structure of a data-flow, that is, its interpretation and representation is derived from the original structure of a data-flow.
According to the existing classifications (see [2,13]), the provenance of data is of *retrospective* and *runtime* (L3) type. It may also cover L2 provenance (data instantiation), however it is not the case in our domain.

Provenance of description is information on how the description of the computation evolved as a function of time and how its constituents came to be. This provenance type is a form of documentation, but has multiple other uses. In particular it may track dependencies of the computation, as well as authorship information, among others. We will see that in our proposed approach even more features are present (see Section 4.2).
This provenance is *prospective* and covers the levels L0 and L1 of provenance.

Provenance of process constitutes metadata that document details of how the execution of the computation progressed. In particular, it includes information on how it behaved in time (e.g., *when* parts of it executed) and in space (e.g., which machines were involved during execution). This kind of information is useful to understand the inner workings of the computation and the system, and resolve problems when they happen. Additionally, it documents the execution for reproducibility purposes. The provenance of process is implied by the control-flow structure, just like data provenance is implied by the structure of scientific workflows (see Section 4.3).

The provenance of process is *retrospective* and of *runtime* (L3) type.

One can argue, that all the mentioned supplementary types of provenance can be considered like any other data that, by definition, is tracked by the provenance of data. There are nevertheless a few reasons that warrant such a distinction.

First, the provenance of description operates at a higher level than the others. Indeed, the information it provides would not normally require the execution of the given computation, unlike the others. Second, as we will see in Section 4, different types of questions can be posed for each type of provenance. The presented distinction allows for more efficient storage and access, as well as more appropriate representation and visualization. Finally, provenance information is difficult to query without structured data and may be overwhelming both conceptually and in terms of resource requirements. For this reason, virtually every approach models provenance information in one way or another.

4 Design of a provenance system

In Section 3 we introduced a new classification of provenance into three types. This section proposes a design of a system that takes as an assumption the control-flow structure of experiment description. Our decision is dictated by promising results obtained while running large-scale, challenging experiments represented as business processes [6]. To represent experiments, the BPM workflow patterns [35] are used, extended with *experimental patterns* that include parallel execution of commands, failure handling, etc. We have shown previously that under sensible assumptions large-scale experiments can be run robustly.

First, let us explicitly state assumptions guiding our discussion and design:

1. The experiment follows *in-situ* methodology in distributed systems research.
2. The experiment description is a control-flow based on workflow patterns.
3. The data processing does not constitute a large fraction of the experiment execution. If it is not the case then it would be reasonable to use a scientific workflow system instead.

For such an experiment, one can ask a question about requirements of a prospective provenance system. We take the following approach: (1) we find *entities* that can be distinguished in the experiment, (2) we define *questions* that can be asked about them.

The first part has been already done in the previous section. As for the second step, we start with a general principle that for a given entity X the two following pieces of information describe fully its provenance: (1) the origin of X and (2) the logical, spatial and temporal context of X.

For the objects stored as the provenance of experiment data, the prospective questions ask which activities created the given datum (logical), which physical nodes or equipment was involved (spatial) and when that datum was collected (temporal). In the case of the provenance of experiment description this leads to questions about authorship of code (logical), about the dependencies between

Fig. 1. High-level workflow description of the exemplary experiment (modified BPMN notation). Each runtime is sequentially installed and evaluated with Linpack benchmark. Note that some parts of the workflow refer to external repositories.

Table 2. Summary of provenance types, the objects and relations they are concerned about and examples of queries that one may ask for each of them.

Type of provenance	Examples of queries
Data	*Which node produced the highest benchmark result?* *What was the runtime system configuration of the nodes?*
Description	*What are the dependencies of the experiment?* *Are there newer versions of modules?* *Who is the author of the activity X?*
Process	*What is the Gantt diagram of the experiment?* *What is the critical path of the experiment?* *What are the failure rates of activities?*
Data & Description	*Did the system specification reflect reality?*
Data & Process	*What activities executed at the node X?*
Description & Process	*Who authored a change that caused the experiment to fail?*
All types	*Who is the author of a module that produced the result X?*

modules (logical) and about the changes to the experiment in time (temporal; there is no spatial context, however). Finally, in the context of the provenance of experiment process this information reduces to details on how the experiment activities executed with respect to each other and the experiment description (logical), where they executed (spatial) and when they executed (temporal).

This analysis leads to examples of questions presented in Table 2. In the following sections, a design of a provenance system is presented. It consists of three subsystems, each capturing one type of the defined provenance. We observe that the natural representation for the provenance of data, the provenance of description and the provenance of process is: by a directed graph, by a rooted acyclic graph, and by a hierarchical tree, respectively. We illustrate the discussion with an abstract example of performance evaluation of MPI runtimes with Linpack benchmark (see Figure 1 for its workflow description).

4.1 Provenance of experiment data

As observed before, capturing and storing the provenance of data is a challenging task with many difficulties. However, due to our special use case (i.e., control-flow based experiments in distributed systems research) we were able to make

a few assumptions. In particular, since the data transformation does not constitute a significant portion of experiments we are interested in, we can store the provenance of data (and data itself) in mostly unstructured way. As a result, we propose a simple key-value store, e.g., BerkeleyDB [28]. Distributed key stores may be preferred if high-availability or scalability is requested (e.g., Dynamo [15]). Objects stored as the provenance of data may reference each other, although we do not optimize for queries involving these relations. Abstractly, data provenance is an *arbitrary directed graph*, that presumably is sparse.

Each datum in the store has its *name*, *type*, *value* and optional *annotations*. The *name* is an identifier (not necessarily unique) of the given data artifact in the context of the current experiment run. Its *type* defines a group of objects it belongs to (e.g., nodes, results) and can be used to optimize queries by narrowing them down to a subset of elements in the store. The *value* of the datum is its raw value, it may be, for example, the result of a benchmark on a node. Finally, optional annotations are used to link the data to related entities, for example to the node that the result comes from, or a timestamp when this data object was stored. They may also link to other types of provenance.

The data stored must be explicitly marked as such. There are two reasons behind: (1) contrary to data-flows, in control-flows it is not explicitly known what *constitutes* data, (2) it narrows down the scope of what is collected and improves performance. Nevertheless, some elements of data provenance can be collected automatically, the configuration of the platform, for example.

The data provenance collected in the exemplary MPI experiment consists of benchmark results (*benchmark* type), and runtime configuration of nodes (*node* type). The benchmark results point to nodes that participated in the execution and to instances of activities that produced them (see Section 4.3).

4.2 Provenance of experiment description

In this section, we provide a design for a representation of experiments that traces the provenance of the experiment description. By *experiment description* we mean the workflow of actions executed as experiment.

Our solution to this problem is inspired by software engineering, more precisely, by (1) a version control system (Git) to track evolution and authorship of the description and (2) a module system based on programming languages (Go) to improve reproducibility, document dependencies and facilitate collaboration. Note that creative use of Git branching model is nothing new [14,33].

However, as Git tracks content at the level of files, it does not meet all our needs. For that reason we propose a simple, yet powerful and easy to use module system that is built on top of it. It adds an additional layer that tracks dependencies of experiments. The approach is modular and ensures reproducibility and consistency of the experiment description, while remaining easy to use.

The experiment and its modules is tracked in a Git repository, and *tags* represent its evolution. Dependencies of the experiment follow the same scheme and are referenced as a pointer to a *tag* in another *repository*. It implies that a pair (*repository*, *tag*) unambiguously defines the experiment description with

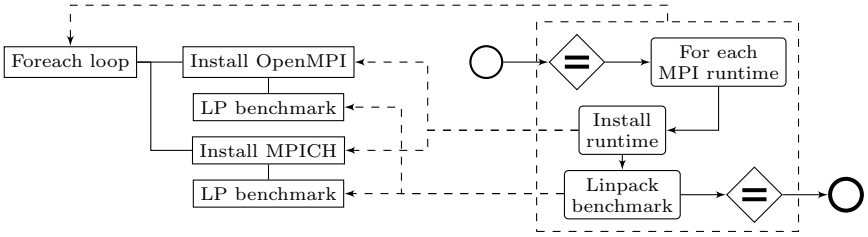

Fig. 2. Example of a mapping between the control-flow and its log. Nesting of workflows implies the hierarchical structure of the log, as is shown with dashed edges.

its all transitive dependencies. Cycles in the dependency graph are forbidden, hence the provenance of description is a *directed, acyclic graph*.

In our exemplary experiment, the main workflow references well-defined versions of modules with MPI support (module *MPI*) and Linpack support (module *Linpack*). Moreover, the latter depends on the former to launch itself.

The repositories containing the workflows can be hosted at social software repositories like GitHub, which offer useful features supporting collaboration, innovation, knowledge sharing and community building [11].

4.3 Provenance of experiment process

In this section we start with analysis of control-flows (based on business processes) and observe that their structure maps to a hierarchical log. We then use this structure as a representation of the process provenance.

Just like scientific workflows imply the structure of data provenance, the structure of an experiment represented as a control-flow implies a form of process provenance. From a high-level point of view, a BPM-like data-flow can be defined either as an *activity* (a basic, atomic action) or as a *pattern* (e.g., a sequence of activities). Other patterns have been defined in the literature [35].

In the example in Figure 2, we see the same workflow that was shown in Figure 1, and associated structure of the provenance of process. In particular, subworkflows map deeper into the log hierarchy. We see therefore that the natural representation for the provenance of process is a *hierarchical tree*.

Each log entry has some predefined data recorded: a timestamp when its execution started, timestamp when its execution finished and the node where it was executed. The log entries annotate their log with relevant information, such as data produced by this activity or a pointer to the source code. The log can be stored in the key-value store used to store the provenance of data.

5 Conclusions and future work

This paper made three contributions. First, we analyzed the provenance in different disciplines of computer science. Then we observed that provenance can be

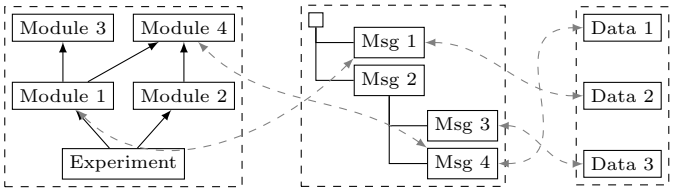

Fig. 3. Interactions between 3 types of provenance. The central role is occupied by the process provenance. It refers two-sidedly to the description provenance and to the data provenance, with no direct links between them.

split into three different types: the provenance of data, the provenance of description and the provenance of process. Finally, we designed a provenance system for distributed systems research that can capture all of them (see Figure 3).

Currently, we focus on the implementation. We also plan to enhance the presented design: in particular, we want to consider a formal model of provenance and verify that all useful queries can be answered within its framework.

References

1. Albrecht, J., et al.: Planetlab application management using plush. ACM SIGOPS Operating Systems Review 40, 33–40 (Jan 2006)
2. Barga, R.S., et al.: Automatic capture and efficient storage of e-science experiment provenance. Conc. and Comp.: Practice and Experience 20(5), 419–429 (2008)
3. Barker, A., et al.: Scientific workflow: A survey and research directions. In: Parallel Proc. and Applied Math., LNCS, vol. 4967, pp. 746–753. Springer (2008)
4. Biton, O., et al.: Querying and managing provenance through user views in scientific workflows. In: Proc. of the 24th Intern. Conf. on Data Eng. pp. 1072–1081. ICDE '08, Washington, DC, USA (2008)
5. Buchert, T., et al.: A survey of general-purpose experiment management tools for distributed systems. Future Generation Computer Systems (2014)
6. Buchert, T., et al.: A workflow-inspired, modular and robust approach to experiments in distributed systems. In: The 14th Intern. Symp. on Cluster, Cloud and Grid Comp. Chicago, Illinois, USA (May 2014)
7. Caneill, M., et al.: Debsources: Live and historical views on macro-level software evolution. In: Proc. of the 8th Inter. Symp. on Empirical Soft. Eng. and Meas. pp. 28:1–28:10. ESEM '14, New York, NY, USA (2014)
8. Cheney, J., et al.: Provenance in databases: Why, how, and where. Found. Trends databases 1(4), 379–474 (Apr 2009)
9. Cohen, S., et al.: Towards a model of provenance and user views in scientific workflows. In: Proc. of the Third Inter. Conf. on Data Integr. in the Life Sciences. pp. 264–279. DILS'06 (2006)
10. Curcin, V., et al.: Scientific workflow systems - can one size fit all? In: Biomedical Engineering Conf. pp. 1–9 (Dec 2008)
11. Dabbish, L., et al.: Social coding in github: Transparency and collaboration in an open software repository. In: Proc. of the ACM Conf. on Computer Supported Cooperative Work. pp. 1277–1286. CSCW '12, New York, NY, USA (2012)

12. Davidson, S.B., et al.: Provenance in scientific workflow systems. IEEE Data Eng. Bull. 30(4), 44–50 (2007)
13. Davidson, S.B., et al.: Provenance and scientific workflows: challenges and opportunities. In: In Proc. of ACM SIGMOD. pp. 1345–1350 (2008)
14. Davison, A.: Automated capture of experiment context for easier reproducibility in computational research. Computing in Science and Eng. 14(4), 48–56 (2012)
15. DeCandia, G., et al.: Dynamo: Amazon's highly available key-value store. SIGOPS Oper. Syst. Rev. 41(6), 205–220 (Oct 2007)
16. Deelman, E., et al.: Workflows and e-science: An overview of workflow system features and capabilities. Future Generation Computer Systems 25(5), 528–540 (2009)
17. Dolstra, E., et al.: Nixos: A purely functional linux distribution. In: Proc. of the 13th Intern. Conf. on Func. Prog. pp. 367–378. ICFP '08 (2008)
18. Freire, J., et al.: Managing rapidly-evolving scientific workflows. In: Prov. and Annot. of Data, LNCS, vol. 4145, pp. 10–18. Springer (2006)
19. Freire, J., et al.: Provenance for computational tasks: A survey. Computing in Science & Engineering 10(3), 11–21 (2008)
20. Garijo, D., et al.: Common motifs in scientific workflows: An empirical analysis. Future Generation Computer Systems 36(0), 338 – 351 (2014)
21. Gustedt, J., et al.: Experimental methodologies for large-scale systems: a survey. Parallel Processing Letters 19(3), 399–418 (2009)
22. Knuth, D.E.: Literate programming. The Computer Journal 27(2), 97–111 (1984)
23. Ko, R.K.L.: A computer scientist's introductory guide to business process management (bpm). Crossroads 15(4), 4:11–4:18 (Jun 2009)
24. Ludäscher, B., et al.: Scientific workflow management and the kepler system. Concurrency and Computation: Practice and Experience 18(10), 1039–1065 (2006)
25. McPhillips, T., et al.: Scientific workflow design for mere mortals. Future Generation Computing Systems 25(5), 541–551 (May 2009)
26. Moreau, L., et al.: Special issue: The first provenance challenge. Concurrency and Computation: Practice and Experience 20(5), 409–418 (2008)
27. Moreau, L., et al.: The open provenance model core specification (v1.1). Future Generation Computer Systems 27(6), 743 – 756 (2011)
28. Olson, M.A., et al.: Berkeley db. In: Proc. of the Annual USENIX Technical Conf. pp. 43–43. ATEC '99, Berkeley, CA, USA (1999)
29. Rakotoarivelo, T., et al.: Omf: a control and management framework for networking testbeds. ACM SIGOPS Operating Systems Review 43(4), 54–59 (Jan 2010)
30. Simmhan, Y.L., et al.: A survey of data provenance in e-science. SIGMOD Rec. 34(3), 31–36 (Sep 2005)
31. Singer, J.: A literate experimentation manifesto. In: Proc. of the 10th SIGPLAN symposium on new ideas, new paradigms, and reflections on programming and software. pp. 91–102. ONWARD '11, ACM, New York, NY, USA (2011)
32. Spinellis, D.: A repository with 44 years of unix evolution. In: Proc. of the 12th Working Conf. on Mining Soft. Repos. pp. 13–16. IEEE (2015)
33. Stanisic, L., et al.: An effective git and org-mode based workflow for reproducible research. SIGOPS Oper. Syst. Rev. 49(1), 61–70 (Jan 2015)
34. Talia, D.: Workflow systems for science: Concepts and tools. ISRN Soft. Eng. (2013)
35. Van Der Aalst, W.M.P., et al.: Workflow patterns. Distrib. Parallel Databases 14(1), 5–51 (Jul 2003)
36. Yu, J., et al.: A taxonomy of scientific workflow systems for grid computing. SIGMOD Record 34, 44–49 (Sep 2005)