Fusion Engineering and Design 96–97 (2015) 835–839

Contents lists available at ScienceDirect

Fusion Engineering and Design

journal homepage: www.elsevier.com/locate/fusengdes

Provenance metadata gathering and cataloguing of EFIT++ code execution

I. Lupelli[a,*], D.G. Muir[a], L. Appel[a], R. Akers[a], M. Carr[a], P. Abreu[b]

[a] CCFE, Culham Science Centre, Abingdon, Oxon OX14 3DB, UK
[b] Instituto de Plasmas e Fusão Nuclear, Instituto Superior Técnico, Universidade de Lisboa, 1049-001 Lisboa, Portugal

HIGHLIGHTS

- An approach for automatic gathering of provenance metadata has been presented.
- A provenance metadata catalogue has been created.
- The overhead in the code runtime is less than 10%.
- The metadata/data size ratio is about ∼20%.
- A visualization interface based on Gephi, has been presented.

ARTICLE INFO

Article history:
Received 30 September 2014
Received in revised form 19 February 2015
Accepted 6 April 2015
Available online 1 May 2015

Keywords:
Provenance
Metadata
Data Management

ABSTRACT

Journal publications, as the final product of research activity, are the result of an extensive complex modeling and data analysis effort. It is of paramount importance, therefore, to capture the origins and derivation of the published data in order to achieve high levels of scientific reproducibility, transparency, internal and external data reuse and dissemination. The consequence of the modern research paradigm is that high performance computing and data management systems, together with metadata cataloguing, have become crucial elements within the nuclear fusion scientific data lifecycle. This paper describes an approach to the task of automatically gathering and cataloguing provenance metadata, currently under development and testing at Culham Center for Fusion Energy. The approach is being applied to a machine-agnostic code that calculates the axisymmetric equilibrium force balance in tokamaks, EFIT++, as a proof of principle test. The proposed approach avoids any code instrumentation or modification. It is based on the observation and monitoring of input preparation, workflow and code execution, system calls, log file data collection and interaction with the version control system. Pre-processing, post-processing, and data export and storage are monitored during the code runtime. Input data signals are captured using a data distribution platform called IDAM. The final objective of the catalogue is to create a complete description of the modeling activity, including user comments, and the relationship between data output, the main experimental database and the execution environment. For an intershot or post-pulse analysis (∼1000 time slices, 65×65 grid, mpi execution $n = 8$ cores) of a typical MAST pulse, the overhead in the code runtime caused by the Provenance Metadata Gathering System is less than 10%, the metadata/data size ratio is about ∼20%, which we consider to be reasonable according to the present literature. A visualization interface based on Gephi for catalogue interrogation, will be presented.

1. Introduction and background

The European and UK [1,2] policy on open access requires that all publicly funded data, particularly data used in journal publications, is made publicly available. Journal publications, as the final product of research activity, are the result of extensive data analysis and complex modeling effort. It is of paramount importance, therefore, to capture the context and provenance[1] of the published data in order to achieve high levels of scientific reproducibility (both in the

* Corresponding author. Tel.: +447442211895.
E-mail address: ivan.lupelli@ccfe.ac.uk (I. Lupelli).

[1] In the context of this paper, *provenance* is a structured metadata that provides information about resources and activities involved in producing a piece of data, which can be used to form assessments.

http://dx.doi.org/10.1016/j.fusengdes.2015.04.016
0920-3796/

same lab or other institutions), transparency and internal/external data reuse. The consequence of the modern research paradigm is that High Performance Computing (HPC) and Data Management Systems (DMS), together with metadata cataloguing, have become crucial elements within the nuclear fusion scientific data lifecycle. Why is it difficult to reproduce a computational experiment exactly, especially years later or in a different lab [3]? The answer is related to the complexity of the codes, computing environments, and architectures. In addition, it is extremely difficult to capture the essential pieces of information during code execution with no human intervention and with an acceptable performance overhead, specially taking into account present day data generation rates and volumes. Scientific Workflow Management Systems, e.g. Taverna [4], Kepler [5], and VisTrails [6], have become increasingly popular as a way of specifying and executing data-intensive and computational-intensive analyses. All these systems provide automated tracking and storage of provenance information, but currently they are not widely adopted in present day nuclear fusion experiments and the provenance capture is limited to workflow level. Scientists frequently execute the workflow mainly in the form of scripts to drive the data analyses and run the scientific codes. For intershot analyses the task scheduler for the processing of diagnostic and plant data after each pulse usually calls those scripts, taking care of the code interdependencies [7]. The immediate needs of a model and an automatic tool for Provenance Gathering (PG), in particular for MAST-U and future machines like ITER, are the main motivation behind this research activity. This paper describes an approach to the task of automatically gathering and cataloguing provenance metadata during code execution, currently under development at Culham Center for Fusion Energy. The functional requirement of the Provenance Metadata Gathering System (PMGS) is the collection (with minimal human intervention and code instrumentation), cataloguing and retrieval of provenance metadata. The catalogue contains all the information needed to identify the full resources and metadata (e.g. input files, logs of experimental data access, version of the scripts and binaries, libraries, environmental variables and modules, users comments, annotations, tags, command line options, context of the simulation, etc.), enabling the reproduction of code execution, from data preparation to data exporting and publication. The content of the catalogue, based on the W7 [8,9] model of provenance, corresponds to answers to the questions "Who, What, When, How and Why" about resources and activities involved in producing the final outputs of the code execution. Provenance Metadata Gathering System can be thought of as the back-end of an electronic notebook for modeling and data analysis. The captured information is stored post hoc in a database and represented as a provenance graph (i.e. linked resources and activities). This approach is being applied to an axisymmetric equilibrium force balance code called EFIT++ [10,11] as a proof of principle test, but the ultimate goal is to keep the conceptual schema and provenance model as flexible as possible in order to follow the continuous evolution of codes and computational environments.

2. Approach

The strategy for the Provenance Metadata Gathering System is based on a combination of techniques for the collection of provenance metadata at different levels of granularity: (1) Wrapping (W): the code is wrapped in order to capture provenance information, thus avoiding code instrumentation. (2) Passive Monitoring (PM): the interaction between the code and its execution environment is traced, capturing the I/O patterns (read/write operations, network access, etc.). This approach involves no modification or instrumentation to either the code or its execution environment. When possible, each code resource is assigned a unique identifier (for example, the SHA-1 checksum for each file present in the filesystem and used by the code). (3) Use of Provenance Aware Libraries For Data Access and Management (PAL): libraries or tools that explicitly capture provenance information at the execution level are included in the code execution environment. As in Passive Monitoring, no modification of the code itself is required. The IDAM [12] server's provenance plugin is used for this specific purpose. A set of specific environmental variables is set and distributed across the execution environment in order to track all the data access related to the code workflow execution and its forked sub-processes. The IDAM server intercepts and logs all imported experimental data and provides the experimental data list used during the simulation and the attached provenance metadata.

The Provenance Metadata Gathering System utilizes all of these strategies, individually or in combination, depending on the specific task of the workflow.

3. Architecture

All of the software and libraries used to develop the system are entirely open source. Starting from the existing control scripts [7], the system is written in a combination of high-level dynamic programming and scripting languages (Perl and Python and Bash Unix Shell for the front-end, and PostgreSQL at the backend. IDAM is used for data access and management. An overview of the architecture is shown in Fig. 1. The code is initially wrapped with the Provenance Gathering Script and the scientist specifies the workflow (e.g. the sequence of preprocess functions and auxiliary codes, code input parameters, serial or parallel execution, post-processing functions). At the beginning of the simulation a Universal Unique IDentifier (UUID) of the specific run (RUN_UUID) is requested and sent back to the Provenance Metadata Gathering System by the IDAM server. In order to generate the UUID of the simulation and record the provenance of data access requests, the IDAM database cluster was extended with an IDAM server plugin. The IDAM plugin is called from the client application through the IDAM API and exposes the following new functional methods:

- <GET::UUID>: this method releases a new UUID for a specific simulation and owner. Example: idamGetAPI("provenance::get (owner = owner,class = class,title = title,description = description)" ,"MAST::");). The UUID is based on a Julian Date time stamp and a progressive sequence number: CCFE/Year/Ordinal Day/Sequence. The CCFE prefix is defined in the IDAM server start-up script and can have different values for each IDAM server installation.
- <PUT::UUID>: this method records the provenance of a data access request for a specified UUID. The provenance has the following components: Requested data objects, Requested data source, True data object, True data source, Data source UUID, IDAM Server log record string.

A group of environmental variables is set for each provenance step of the workflow in order to enable passive monitoring. The workflow is then executed within the Provenance Gathering Script that captures provenance information implicitly in a set of temporary log files (formatted as CSV, Comma Separated Values). Data provenance is captured as a set of dependency edges between resources that are represented as nodes. The provenance metadata created during interactive activities are injected into the backend database comprising 25 tables and available to search and view at a later date by the user. The UUID of each code execution represents the primary key for all the provenance queries. For example, the "rundetails" table stores information about the code execution and the relationship with the main research objective, the experimental

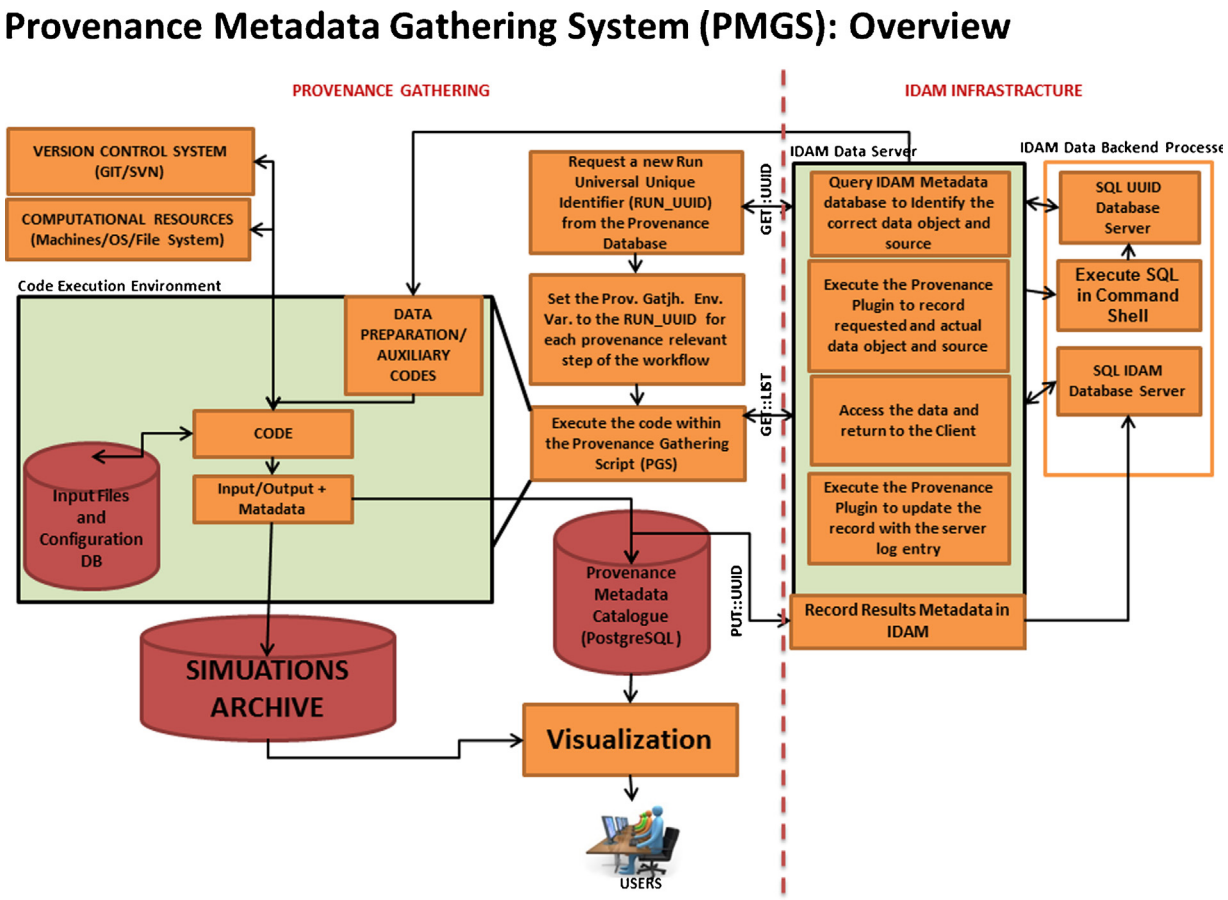

Fig. 1. Overview of the Provenance Metadata Gathering System (PMGS) prototype introduced in Section 2.

campaign or the name of the particular integrated modeling task. There are other tables in the schema for storing information about the computational environment (the version of kernel, environmental variables, modules, libraries), version control system (information about the version of the code, tags and hash, files to be committed, information about the developer), computational resources (characteristic of the machine, architecture), I/O patterns (system calls), command-line options passed to the scripts during the execution, input/output files and reference to experimental data, links with parent-child simulations, annotation and post-run comments. The database can be easily visualized with a visualization tool (see Section 5).

4. Overhead evaluation for the EFIT++ plasma axisymmetric equilibrium code

A key issue in evaluating the feasibility of the Provenance Metadata Gathering System is its performance overhead (i.e. the impact on code run time when the Provenance Metadata Gathering System is enabled). This is especially relevant for parallel execution, since the I/O patterns or the number of system calls per second can drastically reduce the performance of the code. The main concerns are the consumption of computational resources required to collect provenance (gathering) and the overhead for storing provenance (metadata injection time and database size). During the run the Provenance Metadata Gathering Script writes provenance metadata additional logs in human readable format (CSV); the interactions to the backend database and the metadata injection are performed at the end of the simulation with no practical impact on the performance of system.

In order to evaluate the impact of Provenance Metadata Gathering System on the performance of code execution, we ran a set of EFIT++ shots on the Fusion Unix Cluster at Culham Science Center. EFIT++ is a machine-agnostic code that calculates the axisymmetric equilibrium force balance in tokamaks consistent with the experimental data. The standard set of static, self-described and hierarchically organized input files, for machine description and references to the input signals, are stored in the internal Input File and Configuration Database included in the VC repository (see Fig. 1).

A set of auxiliary codes and preprocess functions (mainly for the calculation of induced current in passive structures and for the generation of the machine-dependent input files) are executed during the workflow. The code execution is typically strongly-pulse dependent, especially for advanced post-pulse analyses with internal constraints (kinetic profiles, polarimetry, MSE), in which the quality of the reconstruction is directly related to the quality of the input data. In this case, the user needs to frequently modify the code parameters (basis functions, weights, numerical controls of the solver) in order to get good results. Furthermore, the code uses a considerable set of input signals (~220 for a standard magnetic-only run).

In order to maintain the consistency of the experimental database (when a code needs to be modified or there is a calibration update) the scheduler may have to be rerun frequently, in post-pulse reprocessing mode. For codes strongly linked with the experimental database, it is particularly important to have a system that understands the interdependencies and provides information about the code-to-environment (in case of upgrading a library by the system administrator), code-to-code and code-to-experiments relationships. The scaling of the Provenance Metadata Gathering

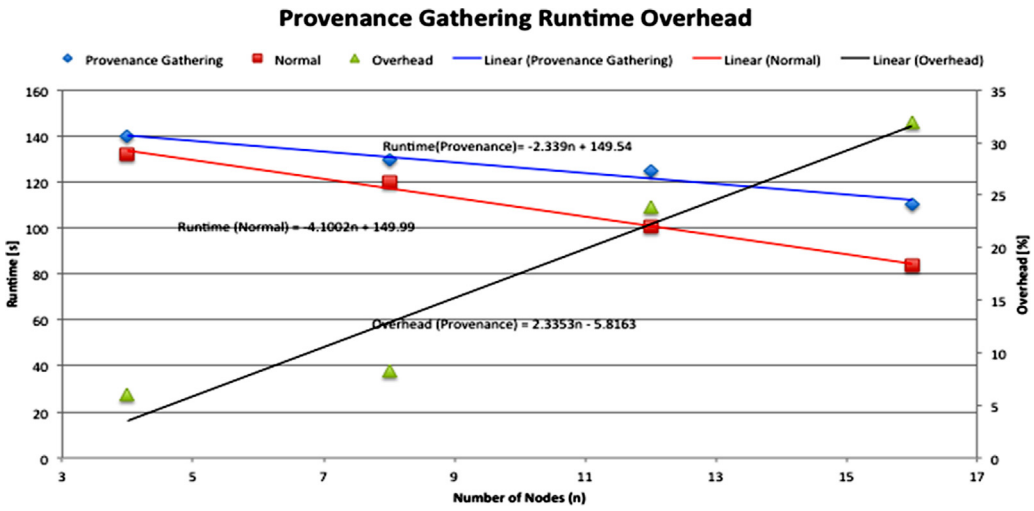

Fig. 2. Provenance Gathering runtime overhead for EFIT++ MPI execution for a typical intershot discharge (magnetic only ~1000 time slices, grid 65 × 65): normal code runtime (red), code runtime with Provenance Metadata Gathering System enable (blue) and overhead (green) (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article) .

System overhead for the MPI execution in shown in Fig. 2. For a simple intershot or post-pulse analysis (~1000 time slices, 65 × 65 grid, $n = 8$ cores) of a typical MAST pulse, the overhead is less than 10% (~10 s). This is acceptable and aligned with the results of other similar systems [13]. When we increase the node number, the I/O pattern and the network performance became important, and the overhead reaches 30%. This needs to be optimized, in particular for the massive parallel applications and next generation applications for ITER. The relevant provenance of the code results has been captured comprehensively. The metadata/data size ratio for a typical run is about ~20%, which we consider to be reasonable. Note that for a typical EFIT++ single shot scheduler/chain-1 run, the archive storage space needed is ~400 MB; advanced equilibrium reconstructions with the full set of internal constraints, or high time resolution reconstructions can generate much larger output files of ~1.5 GB per shot.

5. Visualization

The provenance data is annotated in the catalogue as a directed acyclic graph. Nodes represent a resource in a particular state. Edges represent the dependency relationships between the nodes. Many provenance visualization tools are based on the graph description language DOT, like GraphViz, to draw node-link diagram layouts. However GraphViz has poor support for clustering or high-level grouping and generates static graphs. The only way to manipulate an updated arrangement of the data is to regenerate a new image. In order to query the database and to easily

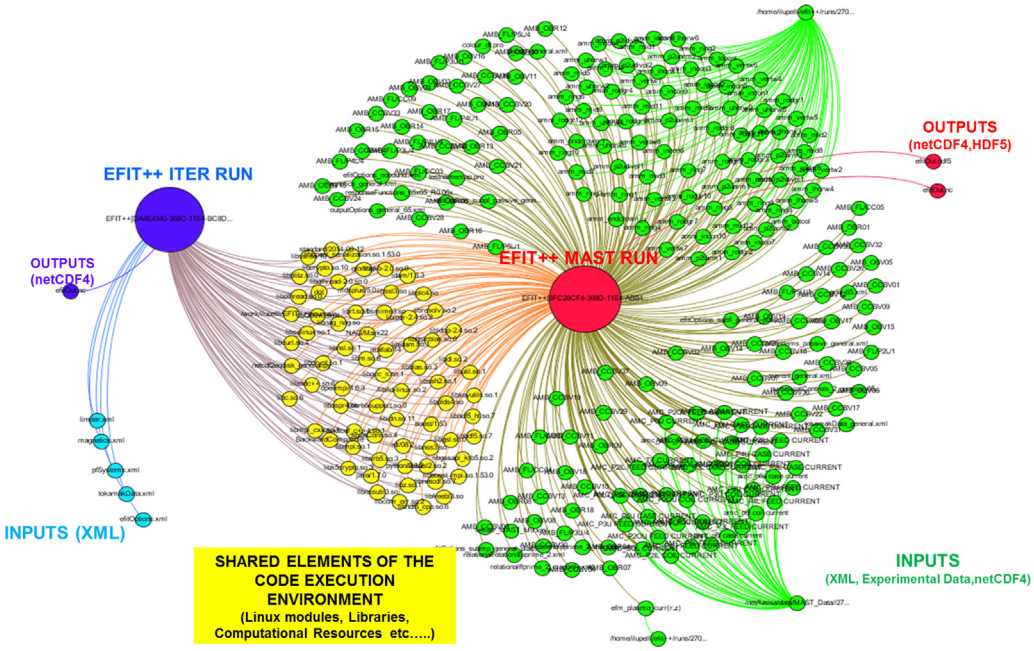

Fig. 3. Example of the provenance metadata visualization for two different EFIT++ runs for MAST (big red circle) and ITER (big blue circle) executed in the same cluster: The essential components of the simulation, input (small green circles for MAST and small light blue circles for ITER represent the set of configuration xml files and/or different data object enclosed in netCDF4 files), outputs (netCDF/HDF5, small red circles for MAST and small blue circles for ITER) and execution environment (yellow circles represent the shared elements of the execution environment, like libraries, linux environmental modules) are shown. The user can interact with nodes and edges, make selections and explore their properties and metadata.

visualize, explore and process the amount of nodes in a typical provenance metadata collection (∼200 nodes and ∼500 edges for a basic EFIT++ run) the interactive visualization and exploration platform, Gephi [14] has been used. Gephi is an efficient large-scale node-link graph layout tool. It offers a variety of improved design algorithms of reasonable algorithmic complexity, sufficient clustering and grouping functionality. The user can view and explore the summary of all available system data, filter subsections, see quantitative or qualitative values of attributes, and make statistics. An example of the visualization of the provenance metadata for two different EFIT++ runs (for a MAST and an ITER shots) executed on the same cluster is shown in Fig. 3.

6. Conclusions and future work

An approach to the task of automatically gathering and cataloguing provenance metadata, currently under development and testing at Culham Center for Fusion Energy, has been described. The approach is being applied to a machine-agnostic code that calculates the axisymmetric equilibrium force balance in tokamaks, EFIT++, as a proof of principle test. The full provenance of the code results has been captured comprehensively. From this test we conclude that the overhead of enabling the Provenance Metadata Gathering System is acceptable for a small number of nodes. The provenance metadata recorded in the relational database can be easily visualized and used to locate data, to enable data reproducibility, and to support compliance with the Open Data access policy. Further developments are planned: to extend the schema (including the link with CCFE's publication database), reduce the overhead, and apply this approach to different codes (e.g. Locust GPU). Furthermore, the visualization of the provenance metadata will be enabled in a web interface, together with the next generation of experimental session leading/logging tools, currently under development at CCFE, based on Drupal technology.

Acknowledgments

This work was part-funded by the RCUK Energy Programme and by the European Union's Horizon 2020 research and innovation programme.

References

[1] Directive 2003/98/ec of the European Parliament and of the Council of 17 November 2003 on the re-use of public sector information.
[2] http://www.epsrc.ac.uk/about/standards/researchdata/.
[3] P. Buneman, S. Khanna, W.C. Tan, Data provenance: some basic issues, in: Foundations of Software Technology and Theoretical Computer Science: 20th Conference New Delhi, India, December 13–15, Springer-Verlag GmbH, 2000, pp. 87–89.
[4] S. Bowers, B. Ludascher, Actor-oriented design of scientific workflows, in: Conceptual Modeling – ER, Springer, 2005, pp. 369–438.
[5] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, Taverna: a tool for the composition and enactment of bioinformatics workflows, Bioinformatics 20 (1) (2004).
[6] J. Freire, C.T. Silva, S.P. Callahan, E. Santos, C.E. Scheidegger, H.T. Vo, Managing rapidly-evolving scientific workflows, in: IPAW, 2006.
[7] D. Dodt, Improved framework for the maintenance of the JET intershot analysis chain, Fusion Eng. Des. 88 (2013) 79–84, Contents.
[8] S. Ram, J. Liu, A new perspective on the semantics of data provenance, in: Proceedings of International Workshop on the Role of Semantic Web in Provenance Management, Washington, DC, October, 2009.
[9] D.P. Schissel, G. Abla, S.M. Flanagan, M. Greenwald, X. Lee, A. Romosan, Automated metadata, provenance cataloguing and navigable interfaces: ensuring the usefulness of extreme-scale data, Fusion Eng. Des. 89 (5 May) (2014) 745–749.
[10] L.L. Lao, H.E. St. John, Q. Peng, J.R. Ferron, E.J. Strait, T.S. Taylor, Fusion Sci. Technol. 48 (2005) 968.
[11] L.C. Appel, Equilibrium reconstructions on multiple tokamaks, Nucl. Fusion (2013) (to be submitted to).
[12] D.G. Muir, L. Appel, N.J. Conway, A. Kirk, R. Martin, H. Meyer, et al., MAST's Integrated Data Access Management system: IDAM, Fusion Eng. Des. 83 April (2-3) (2008) 406–409.
[13] K.-K. Muniswamy-Reddy, U. Braun, D.A. Holland, P. Macko, D. Maclean, D. Margo, Layering in provenance systems, in: Proceedings of the 2009 USENIX Annual Technical Conference (USENIX 09), June 14–19, San Diego, California. Berkeley, CA: USENIX Association 2009, 2009.
[14] M. Bastian, S. Heymann, M. Jacomy, Gephi: an open source software for exploring and manipulating networks, in: Association for the Advancement of Artificial Intelligence, 2009.