Characterizing Workflow-based Activity on a Production e-Infrastructure Using Provenance Data

Souley Madougou\textsuperscript{a}, Shayan Shahand\textsuperscript{a}, Mark Santcroos\textsuperscript{a}, Barbera van Schaik\textsuperscript{a}, Ammar Benabdellkader\textsuperscript{a}, Antoine van Kampen\textsuperscript{a,b}, Silvia Olabarriaga\textsuperscript{a,∗}

\textsuperscript{a}Academic Medical Center, University of Amsterdam, The Netherlands
\textsuperscript{b}Swammerdam Institute for Life Science, University of Amsterdam, The Netherlands

Abstract

Grid computing and workflow management systems emerged as solutions to the challenges arising from the processing and storage of shear volumes of data generated by modern simulations and data acquisition devices. Workflow management systems usually document the process of the workflow execution either as structured provenance information or as log files. Provenance is recognized as an important feature in workflow management systems, however there are still few reports on its usage in practical cases. In this paper we present the provenance system implemented in our platform, and then use the information captured by this system during 8 months of platform operation to analyze the platform usage and to perform multilevel error pattern analysis. We make use of the large amount of structured data using the explanatory potential of statistical approaches to find properties of workflows, jobs and resources that are related to workflow failure. Such an analysis enables us to characterize workflow executions on the infrastructure and understand workflow failures. The approach is generic and applicable to other e-infrastructures to gain insight into operational incidents.

Keywords: Grid Computing, Scientific Workflow Management Systems, Provenance, Workflow Failure Analysis

1. Introduction

The advances in simulation and acquisition devices, while opening new opportunities, have also brought new challenges. The overwhelming amount of generated data that needs to be processed require computing infrastructure beyond traditional resources like desktop and individual clusters. Additionally, the challenging nature of the problems research teams are addressing requires them to collaborate with homologous teams in other cities, countries or even continents, involving sharing not only of knowledge but also of other resources such as acquisition, computing, and storage facilities [1].

New technologies have emerged to enable this new way of research, coined “e-science”. Among them are grid computing [2] and workflow management systems (WfMS) [3, 4]. Both technologies have been extensively used in practice in many scientific domains. Because WfMSs abstract complex computations and data dependencies on complex infrastructures, they need to document their execution in order to, for instance, facilitate troubleshooting and analysis. Especially in a scientific environment, the ability of the WfMS to document the process of the workflow execution with enough granularity to allow reproducibility of the scientific results and their auditing is of paramount importance [5]. Most of today’s WfMS either internally record workflow provenance (e.g., VisTrails [6], Kepler/pPOD [7], provenance trails [8] in Pegasus [9], and Taverna [10]) or provide structured data that can be used to build it such as in MOTEUR [11] and WS-VLAM [12]. Provenance can also include semantic information, in which case it can be used to answer high-level queries involving domain concepts like in Janus [13]. However, generally WfMSs implement system provenance by collecting workflow execution traces, which is very important in troubleshooting problems, especially given the high frequency of failures in distributed computing infrastructures (DCI) [14]. Despite the flexibility introduced by WfMS, running production workflows on DCIs still remains a difficult task because execution incidents are common place.

*Corresponding author

Email address: S.D.Olabarriaga@amc.uva.nl (Silvia Olabarriaga)
Optimization of the usage of grid resources is needed to increase the adoption of grid computing by a broader community. In practice, when running large data-intensive applications on production grid infrastructures, users experience various types of incidents such as bugs in the application program for certain combinations or types of inputs, data transfer timeouts, insufficient resources (e.g., memory) to run a job, hardware and software incompatibilities, etc. Usual approaches include manual examination of job log files or re-running the failing jobs. However, such approaches become very complex when a WfMS is used because these are exactly the details that it is meant to hide in the first place. Although WfMSs usually implement some automatic management of errors, in practice we see that detecting the cause of an error and acting upon it requires (much) human intervention.

Failure analysis on DCIs has received attention in various work. Townend et al. [15] describe a novel approach to multi-version design (MVD) fault-tolerant system in service-oriented applications. The approach has the particularity of using provenance of the services [16] to weight channels within the MVD. Using testbed applications, authors show how their provenance-aware MVD-based fault-tolerant system outperforms a traditional MVD system in terms of system dependability and avoidance of the pathological common-mode failure.

Data mining techniques have been used to help administrators of large production computing grids deal with failure and troubleshooting as described in [14]. The authors apply classification techniques to historical data about resources, users, and jobs on DCIs to infer what properties of machines, jobs, and the execution environment are correlated with success or failure.

Samak et al. [17] describe methods for guiding remediation to stochastic errors through predictions of the impact on application performance along with techniques for isolating systematic sources of failures. The approach identifies workflows with high failure probability, then looks at detailed failure patterns to identify problematic resources that should be avoided in subsequent runs. After assessing the classifier performance, its predictions are used on-line to preemptively restart problematic workflow runs.

A self-healing process to deal with the frequent operational incidents on DCIs [18] has been added to the MOTEUR workflow engine [19]. Workflow activity incidents leading to failure are modeled as states of a fuzzy finite state machine. The degrees of severity of the incidents are computed from metrics assuming that incidents have outlier performance. Incident degrees are later discretized into levels that are associated to actions like task or input replication, site blacklisting and activity stopping.

Previous approaches to workflow execution failure analysis were either simple error analysis or monitoring-like or fault-tolerant features directly implemented in the system. Our approach differs from these in several ways. We not only analyze the errors, but also provide a complete provenance of the workflow executions useful for other purposes such as establishing a data product lineage. Moreover, because we use a third-party WfMS, we prefer non-invasive approaches that do not require instrumentation of the WfMS and that can be adapted to other systems as well.

For the sake of using a common vocabulary, a grid faults taxonomy is proposed in [20]. This multilevel taxonomy classifies an incident according to eight viewpoints which are further subdivided into subtypes. These viewpoints are the following: the location is the component where the fault manifests itself; the originator is the component that causes the fault; the introduction time corresponds to the software life cycle phase where the fault was introduced; the duration is the time span of the fault existence; the severity of the fault; the consequences of the fault are the effects on the system and the caused behavior describes how the system behaves in the faulty state. For instance, a fault in the location viewpoint can be at resource level (e.g., a process, a node or the network), at middleware level (e.g., resource broker, scheduler), at application service layer (e.g., host or container) or at workflow layer (e.g., workflow specification or workflow engine). We use this taxonomy throughout this paper to classify our platform specific errors. For a complete description of the viewpoints and their subcategories, please refer to [20].

We have developed a system that collects provenance of workflow execution externally to the WfMS [11, 21]. This system facilitates the analysis of workflow activity and, in particular, the understanding of workflow execution problems without going into the application-specific details. In this paper we present the provenance system implemented in our platform, and then use the data collected by this system during 8 months of platform operation to analyze its usage and error patterns. We analyze the large amount of structured data using statistical approaches to find properties of workflows, jobs and resources that are related to workflow failures. Such an analysis enables us to characterize workflow execution on the infrastructure and understand workflow failures.

The paper is organized as follows. We first present the platform and the provenance system (Section 2),
and then we characterize the activity on this platform during a period of 8 months, in particular concerning failure (Section 3). Next we adopt a statistical approach to study failure in more detail for two applications in DNA sequence analysis and computational neuroscience (Section 4). Finally we discuss the results and present concluding remarks and future prospects.

2. Provenance system of the e-BioInfra platform

In this section we present a brief overview of the e-BioInfra platform and describe the implemented provenance system.

2.1. Overview of the e-BioInfra platform

The context of our study is the e-BioInfra platform [22], which enables scientists of the Academic Medical Center (AMC) of the University of Amsterdam to run large-scale data analysis on a grid infrastructure. Figure 1 shows its layered architecture. The bottom layer, grid fabric, consists of resources of the Dutch e-science grid provided by the BiG Grid project\(^1\). The layer above it contains grid middleware services also provided by BiG Grid. On top of these layers, generic services such as workflow management, monitoring and provenance services are provided. Finally, the e-BioInfra can be used from various user interfaces, among them the web-based e-BioInfra gateway[23].

![Figure 1: Overview of e-BioInfra architecture with focus on the provenance system (gray components at the right).](image-url)

On the e-BioInfra platform all grid computing activity takes place by executing workflows from the various user interfaces. The MOTEUR workflow management system [19] is used to enact workflows described using the GWENDIA language [24]. A MOTEUR workflow is a directed graph of processors representing computations and dependencies constraining their order of invocation. A processor description encompasses both the processing (binary code and command line parameters) and the input/output data. MOTEUR workflows accept input URIs and scalar values on their initial input ports and produce URIs for the generated output data. Throughout this paper, we also use the term components to refer to processors. Applications that perform data analysis are wrapped into MOTEUR workflow components using the generic application service wrapper (GASW) [25]), which is also responsible for staging data in and out of the worker node. The DIANE pilot job framework [26] is used by MOTEUR for resource mapping and task scheduling. DIANE adopts a master-worker model where the grid jobs (workers) contact the master to receive tasks that correspond to the actual processing units of the workflow. If the job fails, MOTEUR retries it up to a configurable number of times.

Both DIANE and MOTEUR write execution information into custom databases and log files. Our approach relies on these information sources to build, structure, link and store workflow execution traces into a provenance store.

2.2. Provenance System

The provenance system in use today has evolved from our previous work [21, 11]. The main motivation for its initial development was to capture information about the provenance of grid workflow executions in a structured manner to facilitate troubleshooting, among other uses. In particular, the goal was to link information available from various log files and software components (MOTEUR, DIANE) and provide context for errors observed when running workflows on the grid. We chose the Open Provenance Model (OPM) [27] to represent workflow provenance because it clearly defines what information to capture and how to represent it. Furthermore, OPM is a standard that facilitates interoperability with other provenance and workflow management systems, as well as compatibility with the new provenance specification of the W3C, PROV\(^2\), which

\(^1\)http://www.biggrid.nl

\(^2\)http://www.w3.org/TR/prov-primer
was released after we started development of our system.
As illustrated in Figure 1, the provenance system has three main components: Provenator, Provenance store and Retrieval interface.

The Provenator collects provenance information from various sources, including the database maintained by MOTEUR, log files and grid jobs standard output and standard error files. The Provenator is automatically executed after completion of a workflow execution. Collected information include both workflow-specific data (e.g., application and user names) and job-specific data (e.g., component name and execution site) with the proper links between workflow runs and jobs.

The collected information is mapped to OPM using the strategy described in Section 2.3. The provenance store is implemented as a relational database described in Section 2.4. The Retrieval interface implements selected queries based on use cases defined with users of the e-BioInfra platform [21], such as lineage of a data product or comparison of executions of the same workflow specification. The interface also manages how the retrieved information is presented to the user. These functions have been incorporated to the e-BioInfra gateway. SQL can also be used for more flexible queries, which was the approach adopted to perform the analysis presented in this paper (see Section 3).

2.3. Workflow to OPM Concept Mapping

The developed system is based on version 1.1 of the OPM specification, a standard originated from a community effort, the provenance challenge series [28]. OPM defines the provenance of an object as a causality graph enriched with annotations capturing further information pertaining to the execution. The graph has three types of nodes which are agent, artifact and process. An edge materializes a causal dependency between the source and the destination of the edge.

The workflow concepts are mapped onto OPM as explained in [11] and summarized below. Mapping a MOTEUR workflow to an OPM graph is straightforward, as both are defined as a graph. The inputs and outputs are defined as OPM artifacts. GASW processors (where the computation takes place) are defined as OPM processes. A processor is either a GASW task (grid job) or a bean-shell (embedded Java code executed on the MOTEUR server). A data dependency between two processors is materialized by a causal dependency between their corresponding OPM processes. An OPM agent is a contextual entity causing and managing a process execution; therefore, the scientist running the workflow and all e-BioInfra components (hardware and software) involved in the workflow execution are mapped to agents. OPM requires every graph to have at least one account, so we map a workflow run to an account of its corresponding graph.

We followed OPM as closely as possible, however the original model does not specify how to link retried grid jobs to their original grid job. When a given job is a retry of another one that failed previously, an annotation is added to it (RetryOf) and a link is created between the two jobs. Therefore, the generated provenance graph is, in fact, an extended OPM graph.

As an illustration, consider a genomics application that executes the BWA program [29] for sequence alignment of next generation sequencing data. For the sake of readability, let us take the simplified workflow in Figure 2. It contains one component (grid job) that requires a reference file and (a list of) files containing sequence data as inputs, and generates results of the alignment in two output files. Figure 3 represents the extended OPM graph for the execution of this workflow using one reference file (Reference.fasta.tgz) and two files with

---

3See documentation about the e-BioInfra gateway on http://www.bioinformaticslaboratory.nl/twiki/bin/view/EBioScience/EBioInfraGatewaySCIBUSMOTEUR.

4Burrows-Wheeler Alignment tool.
sequence reads (Read1.fastq.tgz, Read2.fastq.tgz). For each read file, a BWA grid job is created to align the sequences against the reference. Successful jobs generate two output files (Result1.bam, Result2.bam and Result1.bai, Result2.bai). The execution graph in Figure 3 shows that, actually, three BWA jobs were executed, however one of them has failed (BWA2) and was retried (BWA3).

The representation of the OPM graph in Figure 3 reads similarly to the workflow representations: inputs are on top, outputs on the bottom\(^5\). The graphical representations for the inputs, outputs and processes are labeled with the values they are given during the execution. For files, those values correspond to URLs, and scalar values otherwise. Edges linking inputs/outputs to processes are labeled by their OPM type (causal dependency), with the role played by the input/output in the process execution in parenthesis. When a job is a retry of a failed one, an annotation is added to it (folded-corner rectangle in the figure).

2.4. Provenance Schema

The schema of the provenance database is shown in Figure 4. It is composed of twelve tables, most of which correspond to OPM concepts (prefix OPM). Since the OPM specification does not define any recommendations for provenance storage, we chose to adopt a relational database due to the maturity, popularity and efficiency of this technology. The Provenator builds OPM entities as Java objects and Hibernate\(^6\) is used as an object-to-relational mapping (ORM) tool to make the Java objects persistent into the relational schema. Tables AgentAccount, ArtifactAccount, DependencyAccount and ProcessAccount are association tables imposed by the mapping. The OPMDependency table contains all objects of type causal dependency (Used, WasGeneratedBy, WasControlledBy, WasTriggeredBy, WasDerivedFrom). Most ORM tools map objects of the same class to the same table. By harnessing polymorphism, some ORM tools are able to map objects of different classes to the same table, as long as they are of the same type. The objects can be unambiguously extracted by using the actual class name as discriminator. Our provenance system leverages this feature to optimize the schema. Furthermore, we took into account two performance-to-compatibility trade-offs in the OPM implementation. The first trade-off is related to Role, which we implement as simple string instead of an object to avoid additional query overhead. The second trade-off concerns time information about events, which OPM defines as an interval (start and stop times) for generality, but in our case they are implemented as timestamps.

3. Understanding workflow activity on the e-BioInfra

The e-BioInfra platform is used by different users, for different applications and in different settings. Collecting usage information and understanding this varied activity is challenging. In particular, it is difficult to identify problems that could be affecting the overall activity and to distinguish these from specific cases of applications or users. In this section we analyze the activity at the e-BioInfra based on workflow provenance information. Our goal is to characterize the platform usage in terms of applications, users, resources and failures. We first describe the dataset used in the experiments, and then we present an analysis of activity at the workflow level and at the task level (jobs).

3.1. Provenance dataset

The dataset used in this study comprises information collected during actual e-BioInfra operation (Decem-
ber 2011-July 2012). It includes workflow runs performed with various goals: testing of the infrastructure, development of new workflows, and data analysis experiments, mostly in DNA sequencing and computational neuro-science applications. These workflows were started manually by different users from different user interfaces, including the e-BioInfra gateway [23]. From this set, we removed incomplete runs. For example, when the workflow could not start, or when some serious error caused the workflow engine to abort its execution, because in such cases the reconstruction of provenance is not reliable due to missing data. The subset of the provenance store considered in this study contains nearly 394 MB of provenance information about 1848 workflow runs, 70 different applications, 77 components and 36033 jobs. A large amount of details are available for these workflow runs, however in this study we concentrate on the following subset.

At the workflow level, the following information was used:

1. application name (string identifying the abstract workflow, unique);
2. name of the user who executed the workflow;
3. start time of the workflow run, which is the workflow submission time;
4. end time of the workflow run, which is the end time of the last job;
5. number of retried jobs: MOTEUR retries jobs up to $N$ times, where $N$ is configurable by the user for each workflow run;
6. final status (successful or failed): a workflow fails when some tasks cannot be successfully completed during its execution, even after the task has been retried a given number of times by MOTEUR.

For each workflow task executed as a grid job, the following information was used:

1. application name to which the job belongs to;
2. component name (string identifying the program executed by the task, which can be reused in different workflows);
3. final status of the job: successful or failed with the error codes explained below;
4. computing resource (coined “site”) and hostname;
5. start time which is when the job starts to execute;
6. end time of the job which is when the job completes or gets aborted or killed;
7. error type derived from the job exit code.

[Figure 4: The relational schema of the provenance store.]
The errors detected at the job level fall into four categories derived from different viewpoints of the faults taxonomy [20]:

**application error:** this error code is returned whenever the component execution is not successful: for instance, exception in Java or C++ code, non zero return value from C code or a script, etc. This corresponds to *resource level fault* with the resource being a process.

**input download error:** the middleware cannot move a file from some external storage to a worker node. This corresponds to a *FileTransferIn* fault.

**result upload error:** the middleware cannot move a file from a worker node to some external storage, corresponding to a *FileTransferOut* fault.

**unknown error:** any error that cannot be attributed to the above types, corresponding to a default return value. This value is returned, for example, when a job is aborted, or GASW crashes, or DIANE agent is killed.

From the diverse applications and users of the platform, we focus on the most used applications and the most active users in the remainder of the paper. For a sake of readability of the figures, we have used abbreviations for the names of those applications and users. The abbreviations are described in Table 1 for the users and Table 2 for the applications. We do not describe all active users but only those with outlier behavior regarding workflow execution.

### Table 1: Description of some users.

<table>
<thead>
<tr>
<th>User</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxy</td>
<td>Corresponds to all e-BioInfra gateway users</td>
</tr>
<tr>
<td>u1</td>
<td>develops, tests and runs production workflows</td>
</tr>
<tr>
<td>u2,u9</td>
<td>execute simple workflows to test the infrastructure</td>
</tr>
<tr>
<td>u3</td>
<td>expert who develops and tests workflows</td>
</tr>
<tr>
<td>u4</td>
<td>user who runs workflows developed by others</td>
</tr>
<tr>
<td>u11</td>
<td>scientist testing the platform</td>
</tr>
</tbody>
</table>

3.2. **Analysis at the Workflow Level**

In this section we analyze the provenance data to understand success and failure of workflows in the e-BioInfra regarding different applications and different users.

Figure 5 shows the successful and failed workflow runs of the applications that have been executed at least 15 times (11 applications in total after excluding test workflows). Success rate ranges from 12% for *a1* up to 96% for *a5*. Note that some of the most failing applications are also the most executed ones. As currently a workflow cannot be resumed, in practice, the users need to resubmit the complete workflow for the same inputs or a subset of them. Note also that applications *a2*, *a6*, *a10* and *a11* correspond to evolving generations of workflows implementing DNA sequence alignment using BWA. The success rate of *a6*, *a10* and of *a11* is clearly higher than *a2*, confirming the success of the strategy used to improve the workflows along time described in [31]. We observed that one of the most executed components of *a1* often failed during the data staging phase. *a3* mostly fails because of wrong input files and data staging issues. Besides data staging incidents, *a5* usually runs smoothly on most sites. *a7* involves both data- and compute-intensive stages. *a8* and *a9* are simple workflows including no memory nor compute-intensive steps, so they have high success rate. In particular because of the compute-intensive nature of some workflow components, we observed lots of sudden termination of components execution due to memory faults, DIANE agent timeouts or for unknown reasons.

Figure 6 shows the number of failed and successful runs started by users who executed at least 10 workflows in the analyzed period (12 users in total). User `proxy` is linked to a robot proxy certificate, a credential used to run workflows from the e-BioInfra gateway. The workflows executed with robot credentials normally corre-
Table 2: Description and abbreviation of the used applications.

<table>
<thead>
<tr>
<th>Application</th>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blast</td>
<td>a1</td>
<td>Alignment of DNA sequences using BLAST(^\text{a}) [30]</td>
</tr>
<tr>
<td>bwaone</td>
<td>a2</td>
<td>Alignment of DNA sequences using BWA with one component</td>
</tr>
<tr>
<td>bwasplit</td>
<td>a6</td>
<td>Alignment of DNA sequences using BWA with data splitted and processed in parallel and partial results merged</td>
</tr>
<tr>
<td>bwasplit2</td>
<td>a10</td>
<td>Same as a6 but with additional input sequence files</td>
</tr>
<tr>
<td>bwasplit2ss</td>
<td>a11</td>
<td>Same as a10 but with site selection for the split and merge steps</td>
</tr>
<tr>
<td>diffgenomes</td>
<td>a3</td>
<td>Counts and clusters homologous genes to find related strains</td>
</tr>
<tr>
<td>dtiatlas</td>
<td>a4</td>
<td>Wraps Diffusion Tensor Imaging software</td>
</tr>
<tr>
<td>dtiprep</td>
<td>a5</td>
<td>Pre-processing step to a4</td>
</tr>
<tr>
<td>freesurfer</td>
<td>a7</td>
<td>Wraps freesurfer(^\text{b}) for reconstruction of the brain’s cortical surface from MRI</td>
</tr>
<tr>
<td>pileupvarscanannovar</td>
<td>a8</td>
<td>Determines variants from alignment files and annotates them</td>
</tr>
<tr>
<td>samtoolsflagstat</td>
<td>a9</td>
<td>Gives basic statistics about alignment files</td>
</tr>
</tbody>
</table>

Figure 6: Successful and failed workflow runs for 12 users, sorted by number of runs. Bars indicate the number of workflows (see left axis). The line indicates the success rate (\%, see right axis).

Figure 7 presents the job performance (success/failure) with respect to computing clusters located on distributed sites. Although there could be more clusters per site,
we treat the clusters individually because their operation and administration policies can differ within the same site. Most sites (or clusters) have a fairly high success rate (above 70%), with the exception of two sites, s11 and s12, where roughly every other job fails. These two sites had bad connectivity during the analyzed period as will be shown in Figure 8.

Figure 8: Distribution of error types for all runs (pie chart) and per grid site (bars).

Next, we analyzed the errors, their types and how they were distributed across sites. In the pie chart in Figure 8 we see that data staging errors (input download and output upload) dominate the landscape (around 70%), followed by application errors. The bars in Figure 8 show the error type distribution per site. Whereas all error types occur in most sites in similar proportions to the pie chart, they show different patterns in some sites. Sites s11, s12 and s16, for instance, display almost exclusively data staging errors. We discovered a large number of connection timeouts on s11, which suggested that the site had bad connectivity. As of s12 and s16, it has been confirmed from the logs that there was a configuration problem on those sites during the analyzed period which made data upload fail.

4. Statistical approach to error analysis

From the analysis performed in Section 3, we conclude that workflow executions fail a lot and that we can roughly categorize the error occurrences by type. However, we still don’t have more details about the causes of errors, for example, whether a particular workflow component fails more often on a particular site, or whether a particular error type occurs more often in some applications. Such information would be useful to give hints about how to solve a problem, for example to find out that the problem is transient and that rerunning the workflow the next day is the best approach. However, typically a thorough troubleshooting requires manual inspection of the log files of the individual jobs, which is a daunting task, or even impractical when too many jobs are run and retried. Moreover, it can only be performed by a developer with sufficient information about all details of the workflow implementation, the WfMS and the infrastructure. This is not the case of the e-BioInfra operators, who are the first to detect and support the investigation of errors occurred during workflow execution. Therefore it would be useful to use the provenance information as a first analysis step in troubleshooting to identify the major error causes.

In this section we explore a statistical approach to identify the major error and success patterns based on the information in the provenance store. The motivation for such an approach is the variety of the data, which makes the analysis very complex when performed in a case-by-case fashion.

4.1. Classification method

Pattern classification [32] consists of learning a target function, also called classification model, that maps each attribute set, or record, to one of a set of predefined category or class labels. A training dataset is used to build the model, which then can be used as an explanatory tool to distinguish between objects of different classes, as well as a prediction tool for unlabeled objects. In our case we are interested in the explanatory potential of classification models to understand the error patterns of grid jobs executed via the e-BioInfra platform.

There are several approaches to classification. Our first attempts were based on clustering [32], however the results were not enlightening. First of all, we found it difficult to prepare the attribute values properly to compensate for their heterogeneity. Secondly, the interpretation of the clusters in such a large feature space was not straightforward. We then investigated tree-based classification methods, which are popular in data mining for tasks of similar complexity. These methods use a form of recursive partitioning in which the training dataset is split into subsets using the most discriminating attribute. The splitting continues until a stopping criterion is met. Generated trees are particularly interesting to explain the classification model generated from the training data in an intuitive way. Conditional inference tree is a relatively new tree-based method that estimates
In this study we used the R package, in particular the ctree function, which implements a conditional inference tree derivation. As this model avoids overfitting, there is no need for explicit pruning or cross-validation of our trees. The essential arguments to ctree are (1) the definition of the target and input attributes, (2) a training dataset, which provides the values for those attributes, and (3) controls, such as the stopping criterion. The target attribute, also called the response variable, defines the class for a given record.

The resulting tree is plotted using a routine of the same R package – see example in Figure 10. The most discriminating attributes appear at the top levels of the tree. Improvements were implemented to the plotting function to enhance readability of the trees, in particular the position of edge labels.

4.2. Experiments

The classification method was applied to different sets of workflow runs using two response variables: the job final status (completed successfully or error) and the error type (application, input, output and unknown). All jobs are included in the training dataset in the first case, whereas only the failed jobs are included in the second case. The stopping criterion for node split was set to 0.999, such that only splits with high confidence (p-value < 0.001) are performed. This value was set after experimentation: smaller values generate larger trees with too many fine grained nodes, hampering its interpretation.

We initially considered all available job attributes and all the workflows from Figure 5, but the generated trees were too big and too hard to interpret, therefore of limited usefulness in practice. We also experimented with subsets of the attributes and workflow runs to bring the trees to an interpretable size. We studied two types of workflow runs with high incidence on the infrastructure operation: successful runs with only a few job retries and failed runs with high number of retried jobs. Our goal was to understand why the workflow execution recovers in the first case and not in the second.

Next, we considered workflow runs per application and per user, and finally we used sets of well-known workflow runs of two applications. The first set corresponds to an application for DNA sequence alignment with BWA that was implemented by three different workflows (a2, a10 and a11). These workflows were executed under controlled conditions during the period of one month, and a thorough analysis of errors has been manually performed in this dataset by an advanced user (bioinformatician) – details of this study are presented in [31]. The second set is composed of production runs of a7 started by one end-user (neuroscientist) from the e-BioInfra gateway. These runs have also frequently failed and were briefly analyzed manually by the workflow developer and the e-BioInfra operators. A summary of the datasets is shown in Table 3. The manual findings about these two sets of workflow runs helped us to test and evaluate the value of the results obtained with the classification approach.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Runs</th>
<th>Jobs</th>
<th>Failed Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Set</td>
<td>178</td>
<td>2372</td>
<td>968</td>
</tr>
<tr>
<td>Second Set</td>
<td>24</td>
<td>1574</td>
<td>472</td>
</tr>
</tbody>
</table>

4.3. Results

We carried out a large number of analyses involving different combinations of the above datasets and job attributes. It turned out that some attributes are not useful because they are correlated (failed jobs and retried jobs, resource and hostname), and others are a consequence of the error, and not its cause (e.g. job duration). After much experimentation, we found that the most useful attribute set is: application name, final status of the job, error type, component name and execution site. This generates small trees that are compact and summarize well the intuition built from extensive troubleshooting during the e-BioInfra operation.

We also noticed that trees generated for sets of runs with too distinct workflows in them are not useful. In such cases site and application prevail as discriminating attributes, occluding other attributes that are specific for each of the workflows (e.g. component name). For example, the tree generated for all the runs in the first set is large and not informative, whereas the individual trees corresponding to the workflows a2, a10 and a11 are valuable as illustrated below. A selection of the most relevant classification trees and findings are included as supplemental material\(^\text{10}\). Some examples of classification trees are presented below to illustrate the results and potential of this approach.

\(^{10}\)see attached documents.
4.3.1. Successful vs Failed Runs
For jobs of successful workflow runs with very few (0 or 1) job retries, we found the application to be the most influential attribute. One interesting finding depicted in Figure 9 was that runs of the most failing applications such as a1 and a4 a2 (see also Figure 5) do not usually retry jobs when they successfully complete, which suggests that the job execution should be stopped and/or restarted after the very first retries. Conversely, a5, the most successful application, does retry some jobs, for instance, after they failed on misconfigured sites.

4.3.2. First Set
Figure 10 shows the classification tree of all jobs executed by the BW A application implemented via a single component workflow (a2). It shows that the most influential attribute in the success of a job is the execution site. Indeed, all jobs executed on the sites on the right branch have failed. In only two sites, on the left branch, the jobs have some chance to successfully complete. Figure 11 presents more details about the same set of runs as in Figure 10 considering only failed jobs. It clearly shows that different groups of sites display different error patterns. On most sites, jobs fail for unknown reason, as shown in tree node 2. The error pattern displayed by jobs in node 4 is dominated by result upload errors, which were caused by a middleware misconfiguration of site s16 (based on manual analysis of logs). Application errors are the most frequent in three sites (s1, s10, s2) in node 5. These sites have the largest clusters of the grid infrastructure and implement strict memory usage policy by killing any non-complying jobs (e.g., using above 2GB of memory). A manual analysis of the logs showed that most of these failing jobs display “out of memory exceptions” because the BWA component in the a2 workflow uses a too aggressive memory approach. One last observation is that s4 does not appear in Figure 11, which means none of the jobs running on it has failed and the failure rate in the left hand branch (node 2) of Figure 10 is due to s7.
In the rightmost branch we observe that, irrespective of components, are less successful on the same sites (node 6). The collected workflow provenance information was analyzed to identify patterns in the data. Figure 12 shows that application and download errors in components ImportInput and steps1to14 shape the error pattern. The manual analysis has revealed that application errors are linked to ImportInput and download errors to steps1to14, however due to the strict stopping criterion for tree splitting adopted, no further node splitting could take place.

5. Discussion

In this paper, we presented our approach to collect detailed provenance of grid workflow executions based on logging information generated by a particular workflow management system. In this case, MOTEUR and DIANE were used because they currently form the backend of the e-BioInfra. However, the approach could be adapted to other systems as well. The architecture of the system is designed to confine changes to the Provenator, as long as the provenance specification is kept. When the WfMS or its log messages change, the Provenator needs to be adapted, and the mapping into OPM graph needs to be revised to comply with possible differences in the available information and concepts in the new system. Changing the provenance standard (e.g. to PROV), however, would require modifying the whole system, including the PLIER API, the Provenator, and the relational schema. Nevertheless, we expect such a modification to be straightforward because usually standards, including OPM, only change incrementally. Although PROV for instance introduces new concepts and some changes in data models, these actually rename or map to OPM concepts and can be easily incorporated into the system. Finally, although we chose a relational schema, we realize the advantages of introducing a semantic layer on top of the current system provenance. Semantic Web standards will enable to publish our results as Linked Open Data after proper data anonymization.

The collected workflow provenance information was used to analyze failure and success of workflows executed on the e-BioInfra during a relatively large period of time. The availability of structured information enabled a thorough analysis based on various combinations of attributes and subsets of the data using straightforward queries to the provenance store. Although the results presented in this study have been performed via programmatic interface (SQL), it would be straightforward to implement a GUI for selected pre-defined
queries, along the lines of the already existing provenance interface available on the e-BioInfra gateway\(^{11}\).

A user-friendly interface for freely browsing the provenance database is desirable, for instance using the approach described in [35], however such advanced interfaces are still a subject of research.

We took a top-down approach in the analysis of e-BioInfra activity, first looking at the patterns at the workflow level, and then at the job level. This first approach was to filter workflow runs, and their corresponding grid jobs, using various attributes, e.g., per application, per user, per site. General statistics about these groups of runs or jobs allowed us to better understand the general activity taking place on the infrastructure. In general we observed low efficiency due to a large number of errors and retries. The reasons for this low efficiency are found in an unfortunate combination of factors such as (the absence of) site selection to run the experiments and to retry jobs, malformed or unavailable inputs to run the workflow, and resource overload. Further actions are needed to address these factors both at the technical and training level for workflow development and execution.

With this first analysis, however, we could not identify by eye any significant “pattern” of activity, in particular of failure and success, because of the heterogeneity of the data and the complexity of the infrastructure. We therefore applied a pattern recognition algorithm, namely classification tree. In general, we observed that classification trees are useful for providing explanations of job success and failure patterns by making explicit the job attributes that most contribute to the job final status. As such, they could become an invaluable tool for first step workflow incident troubleshooting. However, care must be taken to grow the right-sized tree because big trees are not only hard to interpret, they are also hard to read. For instance, in order to make readable the reasonably small trees shown in this paper, we had to implement some R callback routines to improve the original plotting function. Classification trees present a dilemma: small, they are easy to interpret but poor in information, large they are hard to interpret but potentially rich in information. Additionally, they depend on the data and the feature space and their interpretation is close to an art. In our case, we had to reduce the input data by selecting the workflow runs and/or the most relevant or intuitive features.

After extensive analysis of a large number of trees generated with various combinations of attributes and sets of workflow runs, we concluded that the most valuable classification trees for troubleshooting are generated from a homogeneous set of workflow runs. This is in fact not limiting, because it reflects the typical scenario in which such an analysis would take place. For example, a workflow developer would like to optimize a certain workflow implementation, or a platform operator would like to investigate why a given application is failing. Although we confirmed our results for a limited number of cases as reported here, the value of this approach needs to be further validated by others. Moreover, the performance of the classifier has not been objectively evaluated, which would require extensive confirmation of results via manual or custom inspection of the job logs. Considering that the trees are meant to roughly guide a more thorough and detailed analysis of some workflow activity, we consider the current results satisfactory. Finally, the visualization of the trees also calls for improvement in both interactivity and parameterization, which would be very important for its exploitation by the actual users.

Another aspect that could be enhanced is the error type classification, which is currently limited to four types only. A richer and hierarchical error taxonomy as described in [20] would greatly improve both analysis and troubleshooting. However this would require instrumentation of the workflow system and of the application components, violating our approach to consider these as black boxes from the provenance system perspective. Finally the analysis presented in this paper, which is based on structured provenance data, could not have been easily done based only on log files. The provenance data enables linking jobs to the applications, their components and the input files. It also enables the identification of retried jobs. For example, we could not have had component names in Figure 12 and Figure 13 without the proper linking provided by causal dependencies.

6. Conclusion

In this paper we showed how provenance data can be collected from an existing system and then used to characterize workflow-based activities on an e-infrastructure. We analyzed usage and failure patterns, firstly at workflow run level for the most executed applications and the most active users of the platform, and then at the task level, showing the global performance per site in our virtual organization. Finally, we further investigated the causes of failure by analyzing job properties concerning failure and error type distribution. In all cases, the execution site shows as the most influential attribute for the success or failure of a job, indicating

\(^{11}\)http://www.bioinformaticslaboratory.nl/twiki/bin/view/EBioScience/EBioInfraGatewaySCIbus
that better site selection, or alternatively site blacklist- ing, could improve the efficiency of the infrastructure significantly.

We discussed how this work can be extended and how classification trees can be used as a valuable tool for troubleshooting workflow execution incidents. Our provenance system also annotates whether a job is a retry of another job during workflow execution trace collection, and this information can be used to study the job retry mechanism in an operational context. We observed that the mechanism is non-clairvoyant, as the workflow engine often re-submits a job to a site where it has consistently failed. Making the mechanism clairvoyant by using provenance would increase the efficiency of workflow executions on the platform. Finally, during the analysis, one recurrent question was to know the maintenance status of a given site in a given period of time. As currently this information is not available in the provenance, we had to get it from external sources when available. To address this issue, we plan to extend the system with tools designed to provide such information.

Acknowledgements

This work is financially supported by the AMC-ADICT innovation fund, the BiG Grid programme funded by the Netherlands Organisation for Scientific Research (NWO) and the COMMIT project "e-Biobanking with imaging for healthcare". We thank Perry Moerland for their suggestions in improving the text.

References


