

Applying Process Mining on Scientific Workflows: a Case Study on High Performance Computing Data*

Zahra Sadeghibogar^[0000-0002-6340-9669], Alessandro Berti^[0000-0002-3279-4795],
Marco Pegoraro^[0000-0002-8997-7517], Wil M.P. van der
Aalst^[0000-0002-0955-6940]

Chair of Process and Data Science, RWTH Aachen University, Aachen, Germany
{sadeghi, a.berti, pegoraro, wvdaalst}@pads.rwth-aachen.de

Abstract. Computer-based scientific experiments are becoming increasingly data-intensive, necessitating the use of High-Performance Computing (HPC) clusters to handle large scientific workflows. These workflows result in complex data and control flows within the system, making analysis challenging. This paper focuses on the extraction of case IDs from SLURM-based HPC cluster logs, a crucial step for applying mainstream process mining techniques. The core contribution is the development of methods to correlate jobs in the system, whether their interdependencies are explicitly specified or not. We present our log extraction and correlation techniques, supported by experiments that validate our approach, enabling comprehensive documentation of workflows and identification of performance bottlenecks.

Keywords: High Performance Computing · SLURM · Scientific workflow · Process mining.

1 Introduction

A *workflow* is a description and automation of a process, in which data is processed by different logical data processing activities according to a set of rules. A *scientific workflow* is an ensemble of scientific experiments, described in terms of scientific activities with data dependencies between them [2]. Scientific workflows allow scientists to model and express the entirety of data processing steps and their dependencies. Fig. 1 shows an example of a scientific workflow depicted as a flow chart, where each task is associated with a command.

Scientific workflows, characterized by their massive data processing needs, are often automated and necessitate parallel processing on platforms such as cloud

* The authors gratefully acknowledge the German Federal Ministry of Education and Research (BMBF) and the Ministry of Education and Research of North-Rhine Westphalia for supporting this work/project as part of the NHR funding. Also, we thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

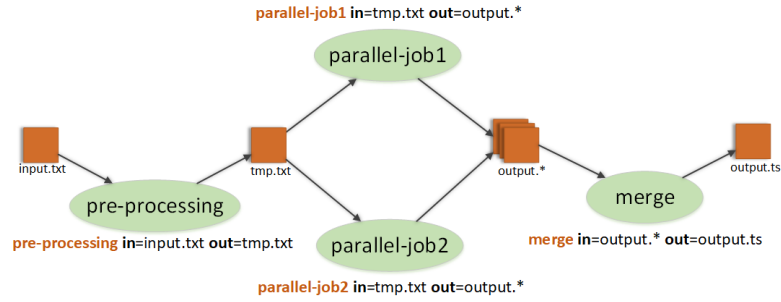


Fig. 1: Illustration of a Scientific Workflow: Starting with a single input file, the process involves pre-processing, parallel job execution, and merging outputs into a single final file.

or HPC clusters [2,9]. In this context, process mining emerges as an invaluable tool for workflow comprehension and detection of possible optimizations.

Over the past decades, there has been a growing interest in the field of process mining [1]. Process mining aims to extract information about processes from event logs, i.e., execution histories. This paper applies process mining to existing scientific workflows with the following goals:

- *Documentation of scientific workflows*: reporting which commands are executed and in which order. We pursue this goal by using process discovery techniques, one of the main branches of process mining [1]. Process discovery techniques assume that every record in the event log contains at least: (i) a reference to the executed activity, (ii) a reference to the identifier that associates an event with a particular execution of the process, and (iii) the timestamp at which the event occurred.
- *Detection of bottlenecks affecting the execution of scientific workflows*: We enrich the process model discovered in the previous step with the obtained performance results.

While the techniques proposed in this paper can be applied to any workflow system, we focus on the SLURM system to promote applicability. SLURM is a common choice for workflow management in HPC clusters, governing the RWTH HPC cluster¹, one of the most widely used platforms in the field.

The issue in examining the logs obtained from a given workflow system is the absence of a clearly defined case identifier that groups events associated with the same execution. To apply process mining to these logs, it is necessary to study the correlation between tasks that are running on the HPC cluster. Fig. 2 shows an overall view of our approach. The RWTH HPC cluster is observed periodically, and an input log is generated. Based on how users execute their jobs on SLURM [8], we propose two different approaches to assign case IDs to events. Finally, we obtain an event log on which process mining techniques can be applied.

¹ <https://help.itc.rwth-aachen.de/service/rhr4fjjuttff/>

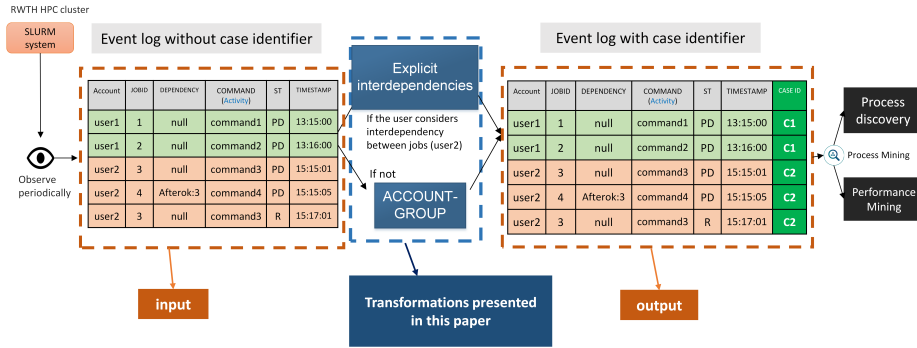


Fig. 2: An overall view of the proposed approach

The remainder of this paper is organized as follows. Sec. 2 reviews some related works. Sec. 3 shows some technical notions on how the SLURM system is implemented and which information is available to eventually form an event log. Sec. 4 explains our approach to apply process mining techniques to the scientific workflows running on SLURM-based HPC clusters. Sec. 5 introduces some analyses of the event log extracted from the SLURM system of RWTH Aachen University. Finally, Sec. 6 concludes the paper.

2 Related Work

Many studies have analyzed HPC behavior starting from data collected about the running jobs. In [4], an extension of miniHPC is proposed, to enable job-level monitoring to interpret anomalous behaviors such as load imbalance, CPU and I/O anomalies, or memory leaks. A framework for monitoring, analyzing, and predicting jobs running on PBS-based job scheduler HPCs is defined in [5]. The monitoring module captures data about the topology of in-use nodes while a job is running. This provides a deeper understanding of how the job is distributed across the HPCs node network. In [3], a software stack for center-wide and job-aware cluster monitoring to influence node performance is described.

Process mining techniques have been used to analyze scientific/business workflow logs. In [9], a technique to mine scientific workflows based on provenance (i.e., the source of information involved in manipulating artifacts) is proposed. In [7], Scientific Workflow Mining as a Service (SWMaaS) is presented to support both intra-cloud and inter-cloud scientific workflow mining.

A limitation of [4,3,5] is that they examine the jobs regardless of their interdependencies. Moreover, in [7], it is assumed that the data source already contains all the necessary information to apply process mining, ignoring the situations in which no case notion is defined. This paper aims to introduce event correlation methods applicable to event data extracted from scientific workflows.

3 Preliminaries

We will focus on analyzing event data from the popular SLURM platform for HPC computing. Hence, in this section, we present some technical notions. To interact with SLURM, we have a set of possible commands. The most essential commands are listed here [8]:

- **srun**: runs a single job. We need to create a **srun** script, which can then be submitted on SLURM for real-time execution.
- **sbatch**: submits one or more **srun** commands for later execution on SLURM.
- **squeue**: reports the states of the running jobs. This command helps us to extract a log for process mining purposes.

3.1 Execution of a Single Job on SLURM

Any script runs on SLURM as a job. As mentioned above, the execution of a job on SLURM could be easily done with **srun** and **sbatch** containing one single **srun** command. Understanding the sequential stages a job undergoes for execution, and the data that can be extracted for each job running in the SLURM queue is valuable [8].

Typically, jobs pass through several states in the course of their execution. There are a total of 24 possible states for a job, however, three states are most common. In the *PENDING (PD)* state, the job awaits resource allocation; in the *RUNNING (R)* state, it is currently allocated, and in the *COMPLETING (CG)* state, the job is undergoing completion.

The SLURM scheduling queue contains all the information about running jobs. To view this information we use the **squeue** command. The most important features of the jobs that have been used in our study are listed in Table 1. These features could be extracted with the **squeue -o "%a %i %E %o %t %g"** command on the SLURM system. This command shows the list of jobs in the SLURM scheduling queue along with their account, job ID, declared dependency, executed command, status, and project ID information [8].

Table 1: Extracted features of running jobs on SLURM system [8].

Column title	Description
ACCOUNT	Account associated with the job.
JOBID	An unique value as job identifier.
DEPENDENCY	Specify the dependencies of the job on other jobs. This job will not begin execution until these dependent jobs are complete. In the case of a job that cannot run due to job dependencies never being satisfied, the full original job dependency specification will be reported. A value of NULL implies this job has no dependencies.
COMMAND	The command to be executed.
ST	Jobs typically pass through several states in the course of their execution. The typical states are <i>PENDING</i> , <i>RUNNING</i> , <i>SUSPENDED</i> , <i>COMPLETING</i> , and <i>COMPLETED</i> . <i>ST</i> is the compact state of the job.
GROUP	Group name of the job. The project ID is reported as GROUP in SLURM.

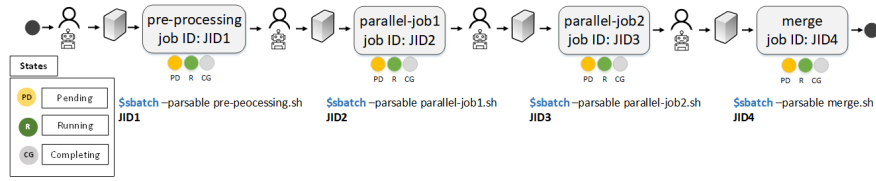


Fig. 3: Execution of a sequence of jobs without explicit interdependencies.

3.2 Execution of a Sequence of Jobs on SLURM

To explain how to run a series of jobs (sequence of scripts) on the SLURM workflow system, we will go through an example. Consider a user who wants to run four scripts on SLURM, *pre-processing* as the first one, then *parallel-job1* and *parallel-job2*, which can be executed in parallel but must be executed after the *pre-processing* script, because they need its output. Finally, the *merge* script needs the output of the two parallel jobs for its execution. The user can run this sequence of jobs on SLURM in two ways: either manually (without explicit interdependencies) or automatically (with interdependencies).

Execution of a Sequence of Jobs without Explicit Interdependencies: In this case, the user runs the jobs manually—without declaring the interdependencies between jobs—and after submitting each job waits for its execution to be completed; then, executes the next job (Fig. 3). In this case, each job is executed as independent, and only the user knows that some of these jobs are logically dependent on each other.

Execution of a Sequence of Jobs with Explicit Interdependencies: In this scenario, the user uses the SLURM dependency management system and submits all jobs at once with correct inter-dependencies on the SLURM system, as shown in Fig. 4. Here, the user uses the `sbatch` command. This command is used to submit a job script for later execution using the `--dependency` option. The script typically contains one or more `srun` commands to launch parallel tasks. In this case, the user does not need to wait for the outputs of a single job, but can wait for the execution of all the tasks and retrieve the final results at completion (Fig. 4).

Table 2 shows the output of the `squeue` command where the user has declared explicit interdependencies between jobs. As one may see, the `DEPENDENCY` column in the `PENDING` state has a non-empty value. Conversely, when users manually initiate jobs, the `DEPENDENCY` column in the `PENDING` state remains empty, denoted as (*null*) values.

4 Approach

The input of most process mining algorithms is an event log, which contains at least a case, an activity, and a timestamp as attributes for each event. The majority of algorithms presume that the event data is fully accessible and has a clearly defined case notion.

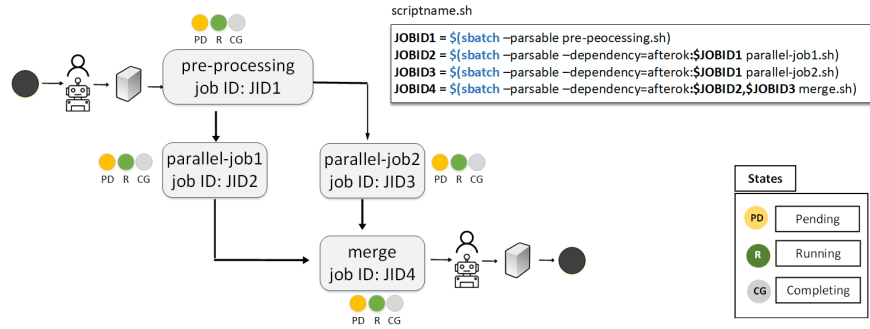


Fig. 4: Execution of a sequence of jobs with explicit interdependencies.

However, we cannot assume that we know the complete historical log, because of privacy issues and the required administrative privileges on the target workflow system. Instead, we aim to observe it for a limited amount of time as described in Sec. 4.1. However, this poses technical challenges. For instance, the observation interval may be too long, making it difficult to capture job information during rapid status changes.

Moreover, since the SLURM log does not contain any explicit case notion, in Sec. 4.2 we describe *event correlation* to assign a case to the different events and allow for process mining analyses.

4.1 Register SLURM events

In order to extract an event log from the system, we perform the following operations periodically (we refer to this as *observing the system*):

1. Connect to the access node of the SLURM system
2. Observe the status (e.g., *PENDING*, *RUNNING*, *COMPLETING*) of the current jobs using the `squeue` SLURM command.
3. For each of the listed jobs (rows of the log file), one of the following situations occurs:
 - The *JOBID* is new: register an event related to the creation of the job.
 - The *JOBID* already exists, but the status has changed: register an event related to the status change.
 - The *JOBID* already exists, and the status has not changed: do nothing.

All the features mentioned in Table 1 are recorded for each job. Our log (as illustrated in the sample SLURM log in Table 2) includes a *TIME* column denoting the event observation time recorded by the script monitoring the SLURM system, and the *COMMAND* values are derived from the executed file path by extracting and using only the file name (the last part of the path).

Table 2: Sample SLURM Log

ACCOUNT	JOBID	DEPENDENCY	COMMAND	STATE	TIME	GROUP
userA	JID1	(null)	pre-processing.sh	PD	13:34:09	G1
userA	JID2	afterok:JID1 (unfulfilled)	parallel-job1.sh	PD	13:34:09	G1
userA	JID3	afterok:JID1 (unfulfilled)	parallel-job2.sh	PD	13:34:09	G1
userA	JID4	afterok:JID2 (unfulfilled), afterok:JID3 (unfulfilled)	merge.sh	PD	13:34:09	G1
userA	JID1	(null)	pre-processing.sh	R	13:35:00	G1
userA	JID1	(null)	pre-processing.sh	CG	13:49:10	G1
userA	JID2	(null)	parallel-job1.sh	R	13:52:09	G1
userB	JID5	(null)	Import.input.sh	PD	13:52:32	G2
userB	JID6	afterok:JID5 (unfulfilled)	Main.calculation.sh	PD	13:52:33	G2
userB	JID7	afterok:JID6 (unfulfilled)	Export.output.sh	PD	13:52:33	G2
userA	JID3	(null)	parallel-job2.sh	R	13:54:13	G1
userA	JID3	(null)	parallel-job2.sh	CG	14:12:10	G1
userB	JID5	(null)	Import.input.sh	R	14:12:12	G2
userA	JID2	(null)	parallel-job1.sh	CG	14:38:10	G1
userB	JID5	(null)	Import.input.sh	R	14:39:32	G2
userA	JID4	(null)	merge.sh	R	14:51:30	G1
userA	JID4	(null)	merge.sh	CG	14:53:30	G1
userB	JID6	(null)	Main.calculation.sh	R	14:54:12	G2
userB	JID6	(null)	Main.calculation.sh	CG	14:58:32	G2
userB	JID7	(null)	Export.output.sh	R	14:59:10	G2
userB	JID7	(null)	Export.output.sh	CG	15:10:10	G2

4.2 Event Correlation

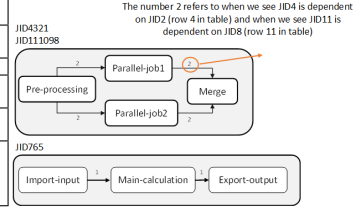
Let us now obtain case IDs from SLURM. We extract a case ID with different techniques, depending on whether the jobs were executed with or without explicit interdependencies.

Case ID Extraction with Explicit Interdependencies: We utilize this technique when the user has specified the inter-dependencies among jobs. This declaration allows the inclusion of the *DEPENDENCY* column in the extracted log, indicating the jobs on which the current job depends. Note that the *DEPENDENCY* column for the job lists only the dependencies that have not been completed yet. Thus, the *DEPENDENCY* list would be naturally empty for a job that is in the *RUNNING* state.

To implement this method, a Directed Acyclic Graph (DAG) is generated for each chain of connected jobs in *PENDING* state by utilizing the *JOBID*, and *DEPENDENCY* columns. The vertices are job IDs and the edges show dependent job IDs, and then a unique case ID will be assigned to all of the connected job IDs as shown in Fig. 5b. In the table in Fig. 5, we observe that JID2 and JID3 are dependent on JID1, and JID4 depends on both JID2 and JID3. Based on these dependencies, we assign case ID JID4321, as indicated in the *CASE ID* column of the table. Different cases will be assigned to different discovered connected components. For instance, JID111098 is assigned to another execution of the same chain of commands as JID4321.

	ACC	JID	COMMAND	DEP	TIME	ST	CASE ID
1	userA	JID1	pre-processing.sh	(null)	13:35:00	PD	JID4321
2	userA	JID2	parallel-job1.sh	afterok:JID1 (unfulfilled)	13:52:09	PD	JID4321
3	userA	JID3	parallel-job2.sh	afterok:JID1 (unfulfilled)	13:52:11	PD	JID4321
4	userA	JID4	merge.sh	afterok:JID2 (unfulfilled), afterok:JID3 (unfulfilled)	13:54:13	PD	JID4321
5	userB	JID5	Import.input.sh	(null)	14:12:12	PD	JID765
6	userB	JID6	Main.calculation.sh	afterok:JID5 (unfulfilled)	14:51:30	PD	JID765
7	userB	JID7	Export.output.sh	afterok:JID6 (unfulfilled)	14:54:12	PD	JID765
8	userA	JID8	pre-processing.sh	(null)	14:56:15	PD	JID11098
9	userA	JID9	parallel-job2.sh	afterok:JID8 (unfulfilled)	14:59:10	PD	JID11098
10	userA	JID10	parallel-job1.sh	afterok:JID8 (unfulfilled)	15:10:10	PD	JID11098
11	userA	JID11	merge.sh	afterok:JID9 (unfulfilled), afterok:JID10 (unfulfilled)	16:05:17	PD	JID11098

(a)



(b)

Fig. 5: Case ID extraction with explicit interdependencies by studying *JOBID* and *DEPENDENCY*. ACC stands for ACCOUNT, DEP for DEPENDENCY, JID for JOBID, and ST for STATE. We use these shorter forms to make the table more compact.

Case ID Extraction without Explicit Interdependencies: In this case, we do not have explicitly defined job dependencies; therefore, we need to use the attributes at the event level to determine correlations and dependencies between the jobs. We use a combination of the following two attributes in order to define the case identifier:

- The *account* executing the job: it is reported as *ACCOUNT* in SLURM.
- The *group* of the given job: the project ID is intended to group the jobs belonging to the same project. The status should be empty or default if the user does not call the command with the project ID. The project ID is reported as *GROUP* in SLURM. So, whenever we execute the same scripts several times, the project ID is reported as the same *GROUP* in SLURM.

We have a many-to-many relationship between accounts and groups. All the jobs executed by an account under a given group are therefore related to the same project. We can use *ACCOUNT-GROUP* as case ID; in this case, we are certain that the jobs executed by the same user under the same project are all collected. In this technique, we generate a unique case ID per each unique combination of *ACCOUNT* and *GROUP*, as shown in Table shown in Fig 6a. The parallel relationship between *Parallel-job1* and *Parallel-job2* has been discovered based on their occurrence in rows 2, 4 and 8, 10, which show they can be executed in any order.

In this method, we may consider only the account instead of considering the combination of group and account, but the advantage of considering also the group is that the control flow of different projects of the same account is not combined. This technique also has limitations, considering loops where they shouldn't be, due to the inability to differentiate between consecutive executions of the same command related to different experiments. As a result, the preci-

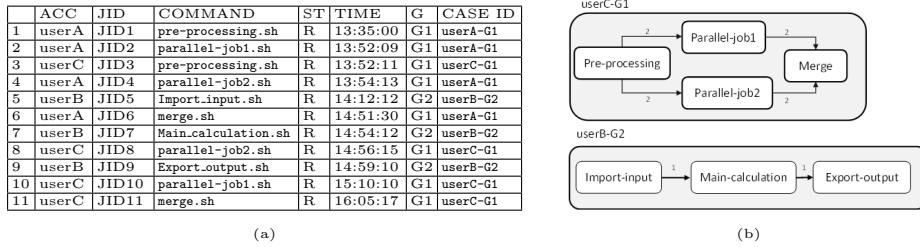


Fig. 6: Case ID extraction without explicit interdependencies by studying *ACCOUNT* and *GROUP*. ACC stands for ACCOUNT, JID for JOBID, ST for STATE and G for GROUP. We use these shorter forms to make the table easier to read and understand.

Table 3: Some event log statistics extracted from the SLURM system. The system was observed in a time interval from 2022-12-07 11:51:45 to 2022-12-09 10:49:07.

Number of events	81632
Number of unique submitted jobs	17997
Number of accounts	123
Percentage of accounts who submitted jobs with explicit interdependencies	0.06%
Percentage of jobs defined with explicit interdependencies	0.02%
Average number of allocated CPUs per job	13.71
Average amount of allocated RAM per job	5G

sion is significantly reduced, because many different behaviors and command sequences are allowed by the resulting model.

5 Experiments

In our previous work, we developed SLURMminer, a tool specifically designed for mining and analyzing process models from SLURM job logs [6]. The experiments presented in this paper utilize SLURMminer. SLURMminer was applied to the SLURM system at RWTH Aachen University multiple times, resulting in the extraction of an event log². Table 3 presents key statistics derived from the event log.

Given the variety of research areas (including physics, chemistry, biology, and computer science) and executed purpose-specific scripts, generating a comprehensive process model containing the behavior of all the accounts proved impractical. Instead, we focused on individual account process models. These process models show the scientific workflows executed by a single user/research group. Moreover, the process model is annotated with performance information on the arcs, allowing for the detection of paths with high execution time (bottlenecks), and therefore fulfilling the second goal of finding root causes of performance problems.

² For more details on the RWTH HPC cluster, visit: <https://help.itc.rwth-aachen.de/service/rhr4fjjuttff/>. A sample event log can be downloaded at: <https://www.ocpm.info/hpc.log.csv>.

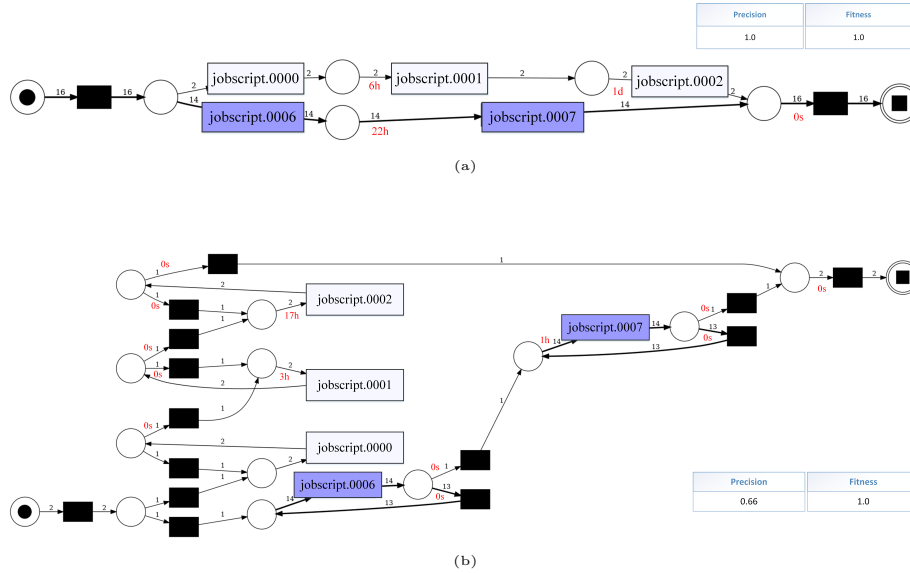


Fig. 7: The discovered process models for the account *jara0180* considering (a) explicit and (b) implicit interdependencies.

To highlight different execution paradigms, we focus on two accounts:

- *jara0180*: contains computations performed on a funded research project (Molecular dynamics simulations of P2X receptors).
- *thes1331*: contains scientific experiments performed for an MSc thesis.

The executions carried out by *jara0180* take advantage of explicit interdependencies (since HPC expertise is involved). Therefore, for *jara0180* we were able to develop a meaningful process model, as depicted in Fig. 7a. In this figure, we observe two distinct chains of commands. The first chain comprises commands from *jobsript_0000* to *jobsript_0002*, while the second chain includes the remaining commands related to two different projects and corresponding to 30 distinct cases in the event log. We could still obtain a model from the data (contained in Fig. 7b) without considering these interdependencies (considering the *ACCOUNT* and *GROUP* values lead to two distinct cases). However, this model is less precise because it relies solely on the temporal order of command execution, where every event belongs to the same case in the event log.

The executions performed by *thes1331* are defined without explicit interdependencies. The case extraction approach, relying on explicit interdependencies, leads to the assignment of a unique case ID to each execution (seven distinct cases). Consequently, the model depicted in Fig. 8a exhibits concurrency among all the executed commands, rendering it highly imprecise. For *thes1331*, it is more appropriate to focus on the models discovered without considering the explicit interdependencies (contained in Fig. 8b), which shows the temporal order of execution of the commands.

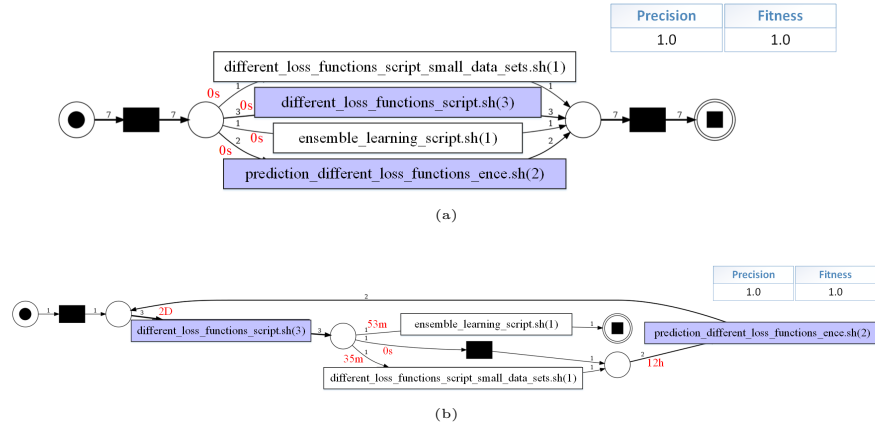


Fig. 8: The discovered models for the account *thes1331* considering explicit (a) and implicit (b) interdependencies.

The discovered models offer insights by visualizing control flow and execution frequency, aiding users in identifying bottlenecks for improvement. Table 3 shows that only a small fraction of HPC users submit jobs with explicit interdependencies, crucial for identifying connected jobs. Without these dependencies, resulting models may be imprecise, with events either belonging to different cases or all belonging to the same case.

6 Conclusion

In this paper, we propose an approach to extract and analyze process mining event logs of an HPC system (in particular, we focus on the SLURM system). While this is not the first application of process mining to HPC systems, existing techniques assume the case notion to be well-defined in the data source. This assumption is not satisfied by mainstream systems, and we propose two different case notions (using and not using explicit interdependencies). Moreover, we propose the *SLURMminer* as a tool to connect to the HPC system, extract an event log, and perform a process mining analysis. The analyses allow us to document the execution of scientific workflows for different accounts or research groups utilizing process models that are annotated with performance information (allowing us to detect bottlenecks). Therefore, we address our initial research question: How can process mining be applied to SLURM-based HPC clusters to document workflows and identify execution bottlenecks?

Our event logs are extracted from information that is publicly available in the SLURM system (including the command that is executed and the requested environment, i.e., the number of CPUs, RAM, and disk space required). However, we do not know the detailed content of the commands or have access to more advanced profiling options. This would require collaboration with the specific research groups operating in the HPC systems and availability to modify the

execution of scientific workflows to accommodate more detailed process mining analyses.

Our process mining analyses rely on a single account or research group. Since the naming schema of the commands is quite arbitrary, we could not identify shared logical steps (e.g., pre-processing, training of ML model, testing of the model) between different accounts; therefore, we could not produce a generic process model. This is indeed a limitation that could not be tackled without properly naming the commands executed on SLURM and without having insights about the commands.

Overall, our approach succeeds in extracting an event log for process mining purposes from the SLURM HPC system, and we can respond to our basic analytical goals. However, given the arbitrary execution styles and naming conventions, we could not produce more general analyses, which remain as a goal for future work.

References

1. van der Aalst, W.M.P., Carmona, J. (eds.): *Process Mining Handbook*, Lecture Notes in Business Information Processing, vol. 448. Springer (2022)
2. Deelman, E., Gannon, D., Shields, M.S., Taylor, I.J.: *Workflows and e-science: An overview of workflow system features and capabilities*. *Future Gener. Comput. Syst.* **25**(5), 528–540 (2009)
3. Dietrich, R., Winkler, F., Knüpfner, A., Nagel, W.E.: *PIKA: center-wide and job-aware cluster monitoring*. In: *IEEE International Conference on Cluster Computing, CLUSTER 2020, Kobe, Japan, September 14-17, 2020*. pp. 424–432. IEEE (2020)
4. Kunz, P.: *HPC Job-Monitoring with SLURM, Prometheus, and Grafana* (2022)
5. Pal, A., Malakar, P.: *MAP: A visual analytics system for job monitoring and analysis*. In: *IEEE International Conference on Cluster Computing, CLUSTER 2020, Kobe, Japan, September 14-17, 2020*. pp. 442–448. IEEE (2020)
6. Sadeghibogar, Z., Berti, A., Pegoraro, M., van der Aalst, W.M.P.: *Slurmmminer: A tool for SLURM system analysis with process mining*. In: *Demonstration & Resources Forum at BPM 2023 co-located with 21st International Conference on Business Process Management (BPM 2023), September 11th to 15th, 2023*. *CEUR Workshop Proceedings*, vol. 3469, pp. 97–101. CEUR-WS.org (2023)
7. Song, W., Chen, F., Jacobsen, H., Xia, X., Ye, C., Ma, X.: *Scientific workflow mining in clouds*. *IEEE Trans. Parallel Distributed Syst.* **28**(10), 2979–2992 (2017)
8. Yoo, A.B., Jette, M.A., Grondona, M.: *SLURM: simple linux utility for resource management*. In: *Job Scheduling Strategies for Parallel Processing, 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003, Revised Papers*. *Lecture Notes in Computer Science*, vol. 2862, pp. 44–60. Springer (2003)
9. Zeng, R., He, X., van der Aalst, W.M.P.: *A method to mine workflows from provenance for assisting scientific workflow composition*. In: *World Congress on Services, SERVICES 2011, Washington, DC, USA, July 4-9, 2011*. pp. 169–175. IEEE Computer Society (2011)