Distributed Multi-Perspective Declare Discovery

Christian Sturm[1], Stefan Schönig[1], and Claudio Di Ciccio[2]

University of Bayreuth, Germany
{christian.sturm,stefan.schoenig}@uni-bayreuth.de
[2] Vienna University of Economics and Business, Austria
claudio.di.ciccio@wu.ac.at

Abstract. Declarative process models define the behaviour of processes by means of constraints exerted over their activities. Multi-perspective declarative approaches extend the expressiveness of those constraints to include resources, time, and information artefacts. In this paper, we present a fast distributed approach and software prototype to discover multi-perspective declarative models out of event logs, based upon parallel computing. The demo is targeted at process mining researchers and practitioners, and describes the tool through its application on a use case, based on a publicly available real-life bench-mark.

Keywords: Process Mining, Process Discovery, Multi-Perspective Process, Declarative Process, MapReduce

1 Overview and Significance to BPM

Automated process discovery is the field of process science embracing the approaches to extract the model of a process from event logs, namely datasets where the process executions are registered [1]. Declarative process discovery in particular returns declarative models, i.e., sets of constraints defining the conditions under which activities can or cannot be executed. Those constraints that show a high support in the event log, i.e., that are fulfilled in most of the registered cases, are considered as relevant and thus included in the final model [3,5]. DECLARE is a well-established declarative modelling language [4]. Upon the analysis of the event log pertaining to a mortgage emission process, e.g., a declarative process discovery algorithm can assert that after the occurrence of a "*Request loan*" activity, "*Check credit risk*" occurs in 100% of cases – using the classical DECLARE mining terminology [3], constraint $Response$(*Request loan*, *Check credit risk*) has a *support* of 100% because the *activation* (*Request loan*) is always followed by the *target* (*Check credit risk*). The main benefit of declarative models reportedly lies in their suitability to depict flexible, partially structured, knowledge-intensive processes [2]. Despite the ever-increasing availability of rich information backed by the events recorded by IT systems, only the sequential order and the class-names of the entries in the event logs have been taken into account by the vast majority of existing declarative process discovery algorithms. Among the undesirable consequences thereof is the partial understanding of the process, due to at least two reasons. Firstly, no other perspective than the behavioural one can be considered, thus disregarding the role of resources, information artefacts, and time in the description of the process. Secondly, the analysis of a narrow portion of the information

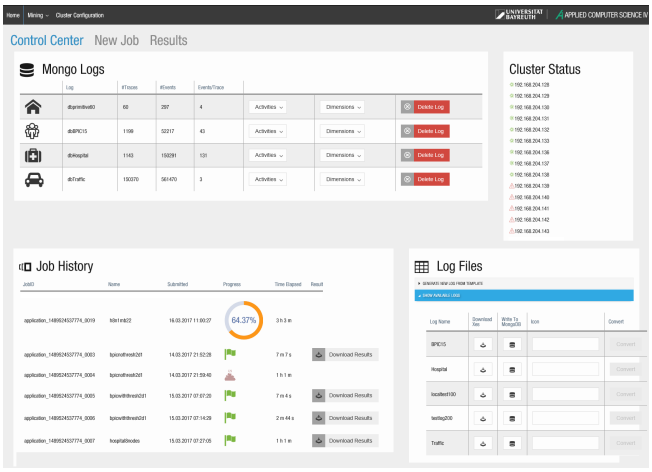

Fig. 1: MP-DECLARE Miner control centre with available logs and executed jobs

at hand can lead to a wrong estimation about the relevance of constraints. For example, it could be that in the example event log $Response(\textit{Check credit risk}, \textit{Grant loan})$ has a support of 60%, which is insufficient to be considered as a relevant information. The constraint would then be discarded. However, "*Grant loan*" always occurs when the data attribute "`credit score`" borne by the "*Check credit risk*" event is higher than 775: the support of $Response(\{\textit{Check credit risk}, (\texttt{credit score} > 775)\}, \textit{Grant loan})$ is 100%. This example clarifies the need to extend the analysis to multiple perspectives: When no other information than the activity name and the ordering are considered in the event logs, relevant facts about the execution of business processes is disregarded. Discovering multi-perspective declarative constraints is however a complex mining task that requires heavy computation and becomes practically intractable without resorting on highly efficient techniques.

In this demo paper, we present the Multi-perspective DECLARE (MP-DECLARE) MapReduce Miner, an efficient distributed mining framework for discovering MP-DECLARE models that leverages latest big data analysis technology and builds upon the distributed processing method *MapReduce*.

1.1 The MP-DECLARE MapReduce Miner Tool

Our distributed MP-DECLARE mining tool resorts on parallel distributed computing, and is based upon the Apache Hadoop software framework, MapReduce system for parallel processing of large data sets, and NoSQL data storage. The Apache Hadoop software library[1] is a framework that allows for the distributed processing of large data sets across clusters of computers. Hadoop is designed to be highly scalable. The clusters consist of one master node and multiple *name nodes*, i.e. slaves. Name nodes execute computing tasks through the MapReduce system, which is in charge of the

[1] `http://hadoop.apache.org/`

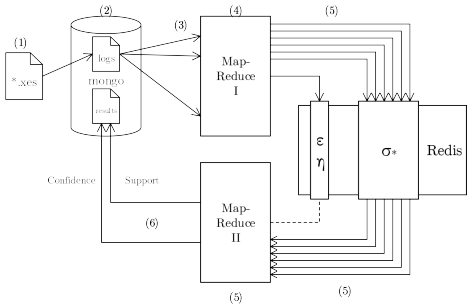

Fig. 2: Implementation architecture of the MP-DECLARE MapReduce Miner

distribution of computing tasks over the nodes, autonomously providing for redundancy and fault tolerance. MapReduce jobs are conventional Java programmes, whose classes have to implement specific interfaces. The input data is divided into chunks (*split*) each assigned to one name node. When a slave has completed its task on a split, it can handle the next input split. In our implementation, data are stored in two distinct repositories: *(i)* A *MongoDB* instance storing the event log data and the results from the discovery procedures, and *(ii)* a key-value store called *Redis*, which stores the temporary intermediate results of the discovery algorithm. Redis is used as a connector between the MapReduce jobs. It holds the whole data in main memory so as to provide faster data seek times.

The user can provide the input event log and configure the mining parameters via a web front-end, shown in Fig. 1. Fig. 2 illustrates the workflow of the application through numbers in brackets showing the communication sequence among the software components. In (1) and (2), the *pre-processing* phase takes place, wherein our application turns an XES event log into a MongoDB database serving as input format for the MapReduce job. Each entry represents one trace with all data attributes. In (3), the *multi-dimensional mapping*, Hadoop transfers the whole event log into shuffled splits as input for the Mapper. In (4), the *reducing* phase, arithmetical operations are performed to compute three numerical values for the computation of the relevance and reliability metrics of constraints, resp. support and confidence. Support denotes the number of cases in which the constraint was not violated in the event log. Confidence scales it by the number of traces in which the constraint was activated. In the example of $Response($*Check credit risk*, *Grant loan*$)$, confidence is thus computed as the support scaled by the number of distinct traces in which *Check credit risk* occurs. The computation of the three variables to calculate the values of support and confidence varies according to the constraint template under consideration and adapts the technique first introduced in [3,5]. They are computed for every constraint template (e.g., $Response$), every combination of activities (e.g., *C* and *G*, abbreviations for *Check credit risk* and *Grant loan*) and all occurring distinct values of the selected data attributes (e.g., x, for `credit score`). σ counts the number of events that trigger the constraint and lead to its satisfaction in the traces (*C* with some $x > 775$ followed by *G*); η sums up the number of occurrences of the activation events (*C* with any x); ϵ counts the number of

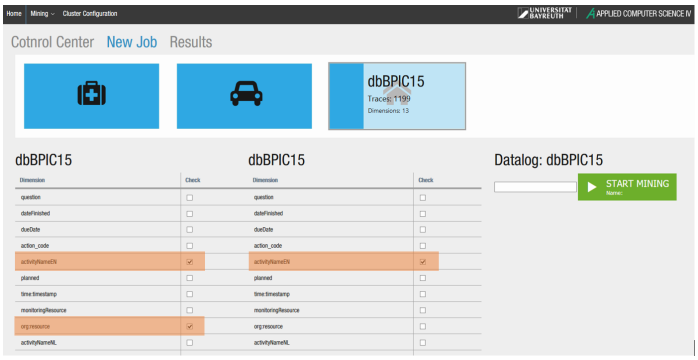

Fig. 3: User interface to start a new mining job

distinct traces where the activation events (*C* with any x) occur. Consider an example event log split consisting of three traces with events labelled a.o. as the aforementioned *C*, *G*, and *R* (*Request loan*):
$\mathbf{t}_1 = \langle R, \ldots, (C, x = 790), \ldots, G, \ldots, R, \ldots, (C, x = 550), \ldots \rangle$;
$\mathbf{t}_2 = \langle R, \ldots, (C, x = 550) \ldots, R, \ldots, (C, x = 550), \ldots \rangle$;
$\mathbf{t}_3 = \langle R, \ldots, (C, x = 790), \ldots, G, \ldots, R, \ldots, (C, x = 470), \ldots \rangle$. In the example event log, dots represent additional events not labelled as either of *C*, *G*, or *R*. Given the event log provided above, the constraint $Response(\{(C, x{=}790\}, G)$ gets associated to $\sigma = 2$, $\eta = 2$ and $\epsilon = 2$. For $Response(\{C, x{=}550\}, G)$, instead, $\sigma = 0$, $\eta = 3$ and $\epsilon = 2$.

σ, η and ϵ are pipelined through the Redis store (5) and served as input for the following *post-processing* phase (6), where the support and confidence values are computed for every constraint to be finally registered in the MongoDB instance with the final results. Please notice that the values of σ, η and ϵ of a constraint gathered on single splits can be solely summed up to merge the measurements in a way that represent the entire event log. Consider, e.g., the following second split: $\mathbf{t}_4 = \langle R, \ldots, (C, x = 550), \ldots \rangle$;
$\mathbf{t}_5 = \langle R, \ldots, (C, x = 700), \ldots, R, \ldots, (C, x = 790), \ldots, G, \ldots \rangle$;
$\mathbf{t}_6 = \langle R, \ldots, (C, x = 790), \ldots, G, \ldots \rangle$. From the second split, $Response(\{(C, x{=}790\}, G)$ gets associated to $\sigma = 2$, $\eta = 2$ and $\epsilon = 2$ again. $Response(\{C, x{=}550\}, G)$, gets assigned with $\sigma = 0$, $\eta = 1$ and $\epsilon = 1$. Overall, then, the values amount to $\sigma = 4$, $\eta = 4$ and $\epsilon = 4$ for $Response(\{(C, x{=}790\}, G)$ and $\sigma = 0$, $\eta = 4$ and $\epsilon = 3$ for $Response(\{C, x{=}550\}, G)$.

Support and confidence values are computed according to the technique explained in [3,5]. The example constraint $Response(\{C, x = 790\}, G)$, e.g., has a support of 100% because each activation of the constraint, *C* with $x = 790$, leads to a fulfilment (the occurrence of *G* afterwards), i.e., σ divided by the number of occurrences of $(C, x = 790)$ is $\frac{4}{4} = 1$. The confidence value amounts to the support scaled by ϵ over the number of traces, i.e., $1 \cdot \frac{4}{6}$, hence 75%. $Response(C, x = 550, G)$, in contrast, has a support and confidence both of 0%, because no occurrence of *C* with $x = 550$ is eventually followed by *G*.

Example Application. We illustrate an example usage of the presented MP-DECLARE MapReduce Miner on three real-life event logs: *(i)* a log pertaining to the building permit

applications in Dutch municipalities[2]; *(ii)* an event log that records the treatment of patients from a large Dutch hospital;[3] and *(iii)* a log of a Road Traffic Fine Management Process.[4] We first load each of the event logs given as a XES file into a MongoDB. Having loaded the XES file into the MongoDB database, the event log is available in the *New Job* section shown in Fig. 3. The tool is configured to discover the conditions under which a certain constraint is fulfilled in the log. The user can select the data attributes that should be considered for the MP-DECLARE mining. Columns refer to the activities involved in the constraint. In Fig. 3, the checked attributes specify that the activity names are taken into account for both activities. Additionally, the *org:resource* attribute is analysed for the first one, e.g., the activation of the $Response$ constraint. The application sends the command to the Hadoop cluster to start the job with the given parameters. Afterwards, the distributed discovery is started and the user can check the progress in the Control Center (Fig. 1). The support and confidence values are stored in the MongoDB for further processing and the graphical representation of models.

For the evaluation, a cluster with 17 nodes (3 GB RAM and 1 CPU each) was used. The system was tested against each of the three log files. The discovery task respectively took less than 30 minutes for log *(i)*, about 4 hours for log *(ii)* and less than 15 minutes for log *(iii)*. Keep in mind that due to the high scalability the cluster size has an enormous influence on the performance. Thus, our distributed mining approach is extremely future-proof when the size of input log data increases more and more.

2 Conclusion, Maturity and Additional Resources

In this demo, we present an efficient and scalable tool for the discovery of MP-DECLARE models based on the distributed processing method MapReduce. The approach represents another step into the direction of integrating process science and data science. For future work, we plan a.o. to provide automated techniques that cluster together data attribute value ranges within the mined MP-DECLARE constraints. Further screenshots, the application results as well as a screencast illustrating the mining procedure are accessible on-line at `http://mpdeclare.kppq.de`.

References

1. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive Processes: Characteristics, requirements and analysis of contemporary approaches. J. Data Semantics 4(1), 29–57 (2015)
3. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM TMIS 5(4), 24:1–24:37 (2015)
4. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: IEEE International EDOC Conference 2007. pp. 287–300 (2007)
5. Schönig, S., Di Ciccio, C., Maggi, F.M., Mendling, J.: Discovery of multi-perspective declarative process models. In: ICSOC. pp. 87–103 (2016)

[2] DOI: `10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1`
[3] DOI: `10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54`
[4] DOI: `10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5`