Towards the Performance Analysis of Apache Tez Applications

José Ignacio Requeno, Iñigo Gascón, José Merseguer
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, Spain
{nrequeno,685215,jmerse}@unizar.es

ABSTRACT

Apache Tez is an application framework for large data processing using interactive queries. When a Tez developer faces the fulfillment of performance requirements s/he needs to configure and optimize the Tez application to specific execution contexts. However, these are not easy tasks, though the Apache Tez configuration will impact in the performance of the application significantly. Therefore, we propose some steps, towards the modeling and simulation of Apache Tez applications, that can help in the performance assessment of Tez designs. For the modeling, we propose a UML profile for Apache Tez. For the simulation, we propose to transform the stereotypes of the profile into stochastic Petri nets, which can be eventually used for computing performance metrics.

CCS CONCEPTS

• **Software and its engineering** → *Petri nets*; **Software performance**; *Unified Modeling Language (UML)*;

KEYWORDS

Apache Tez; UML; Petri nets; Software Performance

ACM Reference Format:
José Ignacio Requeno, Iñigo Gascón, José Merseguer. 2018. Towards the Performance Analysis of Apache Tez Applications. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering Companion, April 9–13, 2018, Berlin, Germany*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3185768.3186284

1 INTRODUCTION

Apache Tez [2] is a processing engine built atop Apache Hadoop YARN ecosystem, that supports interactive queries. Apache Tez simplifies the coding of data-centric workflows and significantly improves the performance of batch-processing large datasets with respect to MapReduce (i.e., Tez avoids the writing of temporary data into disk). Then, it is a new alternative to the original processing engine of Apache Hadoop [13, 16]. For this reason, Apache Tez is progressively replacing Apache Hadoop MapReduce as the main processing core for applications based on Apache Hive, Apache Pig or machine learning.

ICPE '18, April 9–13, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5629-9/18/04…$15.00
https://doi.org/10.1145/3185768.3186284

For leveraging the performance of the framework, Apache Tez developers customize their applications by several parameters, e.g., parallelism or scheduling. Certainly, this is not an easy task, so we argue that they need aids to prevent incorrect configurations leading to losings, monetary or coding. In our view, these aids should come early during application design and in the form of predicting the behavior of the Tez application for future demands (e.g., the impact of the stress situations) in performance parameters, such as response time, throughput or utilization of the devices.

Aligned with our work in the DICE project [1, 5], this paper presents the first steps towards the modeling and simulation of Apache Tez applications with high performance behavior. The DICE project is developing a complete DevOps quality-driven framework for designing, assessing and deploying data-intensive applications. In particular, the UML[1] [19] was the choice for design.

In this work, we have applied a methodology already used in DICE for integrating Apache Storm [14] in the framework. First, we deeply studied Apache Tez and developed several applications. Later on, we enhanced the applications performance by improving the designs and by leveraging the Tez parameters. As a result, we acquired knowledge to propose a novel UML profile for Apache Tez, oriented to performance modeling, that we present here. Finally, we have defined transformations for the Tez profile into stochastic Petri nets [10]. We consider that the resulting models are perfectly usable for an early performance assessment of the Tez applications.

Although the methodology is the same as in [14], differences between these two works are significant. First, Storm is a real-time processing framework, while Tez is an in-memory batch processing framework. Therefore, the parametrisation concepts offered by these frameworks are different, what also makes the profiles different. Consequently, the transformation patterns proposed for Storm could not be reused here for Tez.

Works in the literature for the modeling and performance assessment in big data platforms are discussed in [12]. For instance, a generic profile for modeling big data applications is defined for the Palladio Component Model [7]. Technology-specific profiles (e.g., Apache Spark [8] or Storm [14]) are also available; but none of them include extensions for Apache Tez. Generalized stochastic Petri nets (GSPNs [4]), the formalism for performance analysis that we adopt here, have been successfully used for the performance assessment of Apache Hadoop MapReduce [3], Storm [14] and Spark [6] applications. Nevertheless, GSPNs have not been used for the performance assessment of Apache Tez applications yet.

The rest of the paper is organized as follows. Section 2 recalls the main concepts of Apache Tez related to performance. Section 3 presents the UML profile for Apache Tez. Section 4 presents designs for Apache Tez applications. Section 5 details the transformation

[1] Unified Modeling Language

that we propose to get an analyzable performance model out of a Tez design. Finally, Section 6 draws a conclusion and presents future work.

2 TEZ CONCEPTS FOR PERFORMANCE

#	Concept	Meaning
1.	*Vertex*	Node for processing and transforming data
2.	*Parallelism*	No. of tasks executed by each *Vertex*
3.	*MinSrcFraction*	Fraction of source tasks that must be completed before scheduling the first task for the current vertex
4.	*MaxSrcFraction*	Fraction of source tasks that must be completed before scheduling all the remaining tasks on the current vertex
5.	*VirtualCores*	No. of virtual cores assigned to each task of a *Vertex*
6.	*Memory*	Memory size assigned to each task of a *Vertex*
7.	*Edge*	Connection between a producer and consumer *Vertex* in the DAG
8.	*Edge Scheduling*	Condition for stating the execution of the target *Vertex*
9.	*Data Source*	Lifetime of the data generated by the previous vertices in the *Edge*
10.	*Data Movement Type* (dmt)	Policy of data movement in a *Edge*
11.	*Application Master*	Coordinator of the DAG execution

Table 1: Tez concepts that impact in performance

Apache Tez [2], developed within the Apache Hadoop ecosystem, was born as an evolution of MapReduce to solve or mitigate its main limitations. Based on YARN, Apache Tez offers a complete distributed computation framework for processing very large volumes of information.

Programming in Tez is easier and more expressive than in Hadoop MapReduce: a Tez application is organized as a direct acyclic graph (DAG) according to the application data-flow. The application starts with a map operation followed by a sequence of one or more reduces, instead of dividing the logic of a program in a strict alternation of map and reduce phases. Apache Tez provides a rich API that allows modeling precisely, and with few code, the movement of data inside the program. More important, Tez usually gets a better performance than Hadoop MapReduce for equivalent applications [13, 16]. Tez achieves this speedup by keeping in memory the intermediate results of a MapReduce phase instead of storing them temporarily in hard disk.

Based on the DAG, a Tez application executes a succession of **Vertices**, which process the data, connected with **Edges**, which transfer data between vertices according to some characteristics and movement policies. Then, both the information and the computations are distributed among all the computational resources of the cluster. Once the DAG is specified, the Tez framework is responsible for the automatic distribution and coordination of all the workflow; which makes it transparent to the end user. In addition, Tez dynamically optimize some configuration parameters of the application according to the CPU and network load of the cluster.

Vertices are responsible for running the logic of the application. A vertex is internally divided in multiple **parallel** subtasks inside the Tez framework. Each task executes the same method defined in the Tez Vertex but applying it over a smaller chunk of the dataset. By default, the resource manager allocates a maximal number of virtual cores and memory for each Vertex.

An edge connects two vertices in Tez. It acts as an intermediate message buffer with extended functionalities. The configurable properties of an Edge are: a) the scheduling, b) the data source or persistence, and c) the type of data movement from one node to another. The **scheduling** indicates when the execution of the internal tasks of the target vertex should start. For instance, *sequential* means that the execution of the target vertex starts after the completion of every task of the source vertex, while *concurrent* means that the target vertex starts once it receives the first data emitted by the source vertex.

Next, the **data source** specifies the lifetime/reliability of the data generated by the source vertex: the output is still available when the producer exists (*persisted*), with explicit guarantee that the data is stored in a file and won't be lost (*persisted-reliable*); or the output is only available while the producer is running (*ephemeral*). The type of *scheduling* imposes restrictions to the selection of *persistence* (e.g., ephemeral is only compatible with concurrent scheduling).

Finally, the **data movement** describes, with finer details, how the data is sent from the producer to the consumer. The edge routes the data produced by task i to the task i of the consumer (*one-to-one* policy), copies the output of task i to all the tasks in the consumer (*broadcast*), or scatters the data in shards and the consumer i gathers the i-th portion from every source (*scatter-gather*).

In the end, the **Application Master** is in charge of initializing, launching and coordinating the execution of the Tez DAG. It is linked to a single Tez DAG and it is responsible of monitoring and controlling the total amount of running tasks during the life cycle of the application. The application master deploys the tasks to the computational resources of the cluster according to a scheduling algorithm. Schedulers are customizable by the programmer, although some priority schedulers are preconfigured. Complex schedulers may take into account the available computational resources and the software requirements (memory and CPU consumption) for defining an optimal distribution of the tasks.

For instance, the *fractionality* indicates when the tasks of the current vertex are scheduled with respect to the execution state of the previous vertices in the graph. The property **MinSrcFraction** specifies, in case of a Edge with a scatter-gather connection, the fraction of source tasks that must be completed before scheduling the first task of the current vertex. Otherwise, the property **MaxSrcFraction** specifies the fraction of source tasks that must be completed before scheduling all the remaining tasks on the current vertex. The number of tasks ready for scheduling on the current vertex scales linearly between min-fraction and max-fraction.

In summary, a Tez framework is highly configurable by various parameters that will influence the final performance of the application.

3 A UML PROFILE FOR TEZ

The objective is to define a *domain specific modeling language* (DSML) for Apache Tez. The DSML, oriented to the modeling and performance analysis, will allow to specify the concepts summarized by the previous section. Since the DICE project uses UML, then we propose a UML profile, that according to Selic and Lagarde [9, 15] is a way offered by this standard language for creating a DSML. Concretely, a UML profile is a set of stereotypes, and corresponding tags, that extend the semantics of the UML diagrams when are applied to the elements of the model. The Tez profile has been implemented for the Papyrus Modeling environment in Eclipse and it can be downloaded from [17].

The Tez profile inherits from the standard MARTE profile [11] since it offers a framework for quantitative analysis, the GQAM sub-profile. The latter is specialized for performance analysis, and it offers the NFPs and VSL sub-profiles. The NFP sub-profile aims to describe the non-functional properties of a system, performance in our case. The VSL sub-profile provides a concrete textual language for specifying the values of metrics, constraints, properties, and parameters related to performance, in our particular case. VSL expressions are used in Tez-profiled models with two main goals: (i) to specify the values of the NFP in the model (i.e., input parameters such as the expected execution time of a task) and (ii) to specify the metric/s that will be computed for the current model (i.e., output results such as the application response time).

Apart from the inherited MARTE stereotypes, the Tez profile provides stereotypes and tags for representing the Tez concepts identified in Table 2. The stereotype **TezVertex** represents a Vertex. This stereotype inherits from MARTE::GQAM::GaStep and is applied to UML opaque actions. In addition to the inherited tags, the stereotype owns five specific tags:

- *parallelism*, which corresponds to the number of concurrent tasks per operation. Each task has an associated execution time denoted by the tag *hostDemand*, inherited from GaStep.
- *minSrcFraction* (*MaxSrcFraction*), which matches with the fractionability properties presented in Table 1, and, finally
- the number of *virtualCores* and *memory* assigned per Vertex.

The stereotype **TezEdge** also inherits from MARTE::GQAM::-GaStep, but it is applied to UML control flows. The tag dataMovement, based on the DataMovementType enumerable, defines the three different kind of policies supported by Tez (i.e., *DataMovementType={one-to-one, scatter-gather, broadcast}*). Current versions of Apache Tez only support sequential scheduling and, consequently, persisted or persisted-reliable data sources. For that reason, these properties are deactivated in the TezEdge stereotype. We will introduce them in the profile when the next Tez releases implement these features.

The stereotype **TezScenario** represents the application master that coordinates the Apache Tez application. The tags *amVirtualCores* and *amMemory* are respectively the amount of computational cores and memory resources assigned by the application master to each vertex of the current application in the case that a Vertex does not specify them by itself. A maximum of *maxTaskparallelism* tasks can simultaneously run in parallel. The tags *taskMemory* and *taskVirtualCores* correspond to the resources assigned to the application master for controlling the Tez application. The stereotype *TezScenario* inherits from MARTE::GQAM::GaScenario. It gathers the rest of the contextual information of the application; for instance, the specification of the response time or throughput metrics to be computed by the performance model. The *TezScenario* stereotype is applied to a system scenario, e.g., a UML activity diagram.

Finally, the **GaExecHost** stereotype from MARTE is used for showing the resources in the Tez cluster where the operations are run. The tags *resMult* and *memSize* match with the number of available CPUs and RAM in the device or cluster. This stereotype is used in the UML deployment diagram to define the computational devices.

4 A TEZ MODEL

This section summarizes steps to create a Tez-profiled UML model that describes the main configuration of a Tez application. Specifically, we focus on an *activity diagram* complemented with a *deployment diagram*. The goal is to annotate the UML models adequately, so that they can be later transformed to performance models in order to compute performance metrics. The transformation to a performance model is based on patterns, presented in Section 5.

Figure 1 introduces the UML activity diagram of a toy example application in Tez; a word counter. We consider the UML activity diagram as the DAG of the Tez application. Then, it shows a workflow with a sequence of *vertices* for processing data. Vertices are connected by *edges* that specify a data movement policy. The arcs do not represent causal relationships between actions, such as in standard UML, but communication channels. In particular, the workflow consists of:

- Two initial vertices, *Tokenizer1* and *Tokenizer2*, that read two data sources (e.g., files) word by word, separately,
- two intermediate vertices, *Summation1* and *Summation2*, that count the words propagated by the previous vertex, and
- a final vertex, *Grouping*, that composes the results of both paths and writes the final count in a single output sink.

The Tez application manages two data sources and, therefore, the activity diagram have two initial nodes (one per data source). Initial nodes correspond to the initialization of a data source (i.e., loading of a input file). Then, every path executes a set of operations, and eventually the pipelines merge, as it happens in the *Grouping*, which means the combination of several datasets into a single one. Finally, the Tez application finishes in a final node. Multiple final nodes are also allowed, maximum one per pipeline.

The UML activity diagram is stereotyped as a *TezScenario*, which captures the assigned virtual resources and parallelism of the Tez application, as explained in Section 3. The activity nodes are grouped by a UML partition (e.g., Partition1 in Figure 1).

Figure 2 depicts the deployment diagram, which complements the previous activity diagram to represent the resources where the application executes. Each partition in the activity diagram is mapped to a computational resource, which is stereotyped as *GaExecHost* to define its own resource multiplicity, i.e., number of cores and memory.

Finally, the VSL from MARTE enables the specification of the performance metrics that must be computed for the Tez-profiled UML diagrams. The metrics can be the throughput and response time of the Tez application, and the utilization of the resources.

Tez Concept	Stereotype	Applied to	Tag	Type
Vertex	≪TezVertex≫	Action/Activity		
Parallelism		Node	parallelism	NFP_Integer
Min/MaxSrcFraction			min/maxSrcFraction	NFP_Real
VirtualCores/memory			vCores/memory	NFP_Integer
			hostDemand	NFP_Duration
Edge	≪TezEdge≫	Control Flow		
DataMovementType			dataMovement	DataMovementType[1]
Application Master	≪TezScenario≫	Activity Diagram	maxTaskparallelism	NFP_Integer
			amVirtualCores/amMemory	NFP_Integer
			taskVirtualCores/taskMemory	NFP_Integer

[1]DataMovementType={one-to-one, scatter-gather, broadcast}

Table 2: Tez Profile

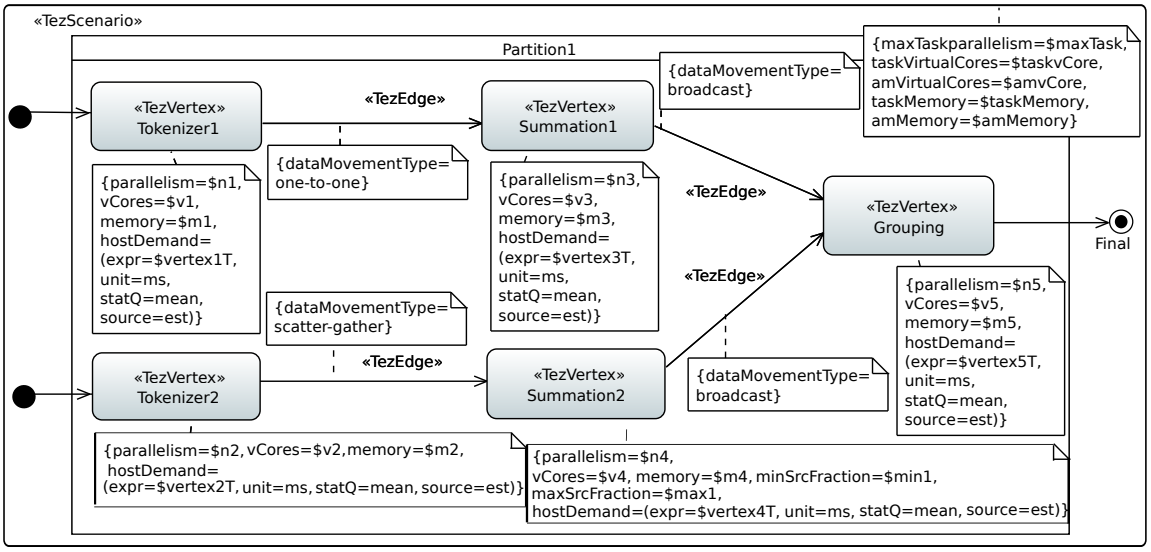

Figure 1: Tez activity diagram for a Word Counter example application

Figure 2 illustrates the VSL annotation for computing the utilization of the device. The throughput and response time should be specified in the UML activity diagram, through the *TezScenario* stereotype, so to guarantee future integration of the Tez profile in the DICE Simulation tool [18].

5 FROM A TEZ DESIGN TO A PERFORMANCE MODEL

UML models, while useful for different purposes, can not be used on their own for computing performance metrics. Hence, we need a proper performance model, Generalized Stochastic Petri Net (GSPN) [10] in our case.

We propose a preliminary set of original *transformation patterns*, that will be applied to the UML-profiled Tez models. Figures 3 and 4 present these patterns. They take as input a part of the Tez design, first column in the Figures, and produces a GSPN subnet, second column. For an easier understanding of the transformation, we depicted in the Figures: a) text in bold to match input and output elements; b) interfaces with other patterns as dotted grey elements, then they actually do not belong to the pattern.

Pattern *P1* presents the transformation of the *TezScenario* stereotype. Place p_{ini} is initialized with a single token to start executing the scenario. Place $p_{maxtask}$ restricts the maximum number of tasks running in parallel in the application, i.e., maxTaskparallelism. Place $p_{maxtask}$ is combined with Pattern *P7* for controlling the access to physical resources. The attribute taskVirtualCores (taskMemory) is used in Pattern *P3* in case that the Vertex omits the specification of vCores (memory). Finally, the rest of the configuration for the application master (amVirtualCores/amMemory) is not transformed to a Petri net in this version of the transformation.

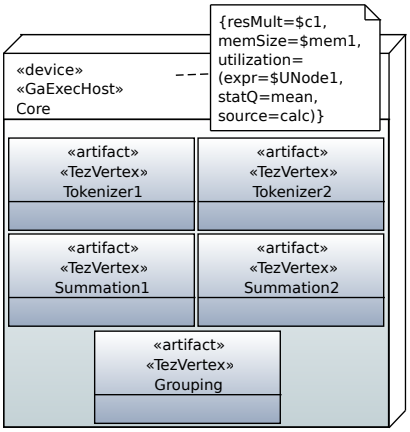

Figure 2: Tez deployment diagram

Pattern *P2* presents the transformation of an initial node in UML. Multiple initial nodes are supported: each one creates an independent path emerging from p_{ini}. The subnet consists of a set of places and immediate transitions that act as intermediate elements for gluing patterns *P1* and *P3*.

Pattern *P3* presents the transformation of a Tez Vertex. The subnet consists of two places and a timed transition. Place p_{A1} controls the pool of virtual cores running the method defined in vertex A. Anther place should be created for controlling the virtual memory allocated for the vertex, but we have abbreviated the pattern for simplicity. Place p_{A2} represents the execution phase. The access to this place, see pattern *P7*, is controlled by p_R to ensure the existence of physical resources.

The maximum number of concurrent tasks is specified by *parallelism*. The number of allocated virtual cores and memory impacts on the number of concurrent tasks running in p_{A2}. Each task takes \$time units for processing an input on average. t_A follows an infinite server semantics and different distributions can be modeled. Finally, the result is sent to the next Vertex through Pattern *P4*. The properties minSrcFraction and maxSrcFraction are used in the case of a *scatter-gather* connection with the following vertex.

Patterns *P4–P5* show the concatenation of two vertices. Patterns *P8–P10*, refine the previous patterns depending on the dataMovementType of the *TezEdge* stereotype. In these patterns, the weights in the arcs of the Petri net represent the number of tasks executed by each vertex.

The messages exchanged by Vertex A and Vertex B are implicit in the structure of the Petri net. For instance, Pattern *P8* shows the broadcast configuration. All the messages created by every task of Vertex A are copied and propagated to every task of Vertex B.

A one-to-one data movement policy imposes that both vertices share the same parallelism (Pattern *P10*). The messages generated by the i-th task in Vertex A must go to the i-th task in Vertex B. Consequently, Pattern *P10* must create n paths for distinguishing each one of the i-th task, with n equal to the parallelism of both vertices. Places PB1_i ensure that task i in Vertex B is processing a single message from task i in Vertex A at a time.

Pattern *P9* corresponds to a scatter-gather movement policy. It is structurally similar to Pattern *P10* but it removes the restrictions

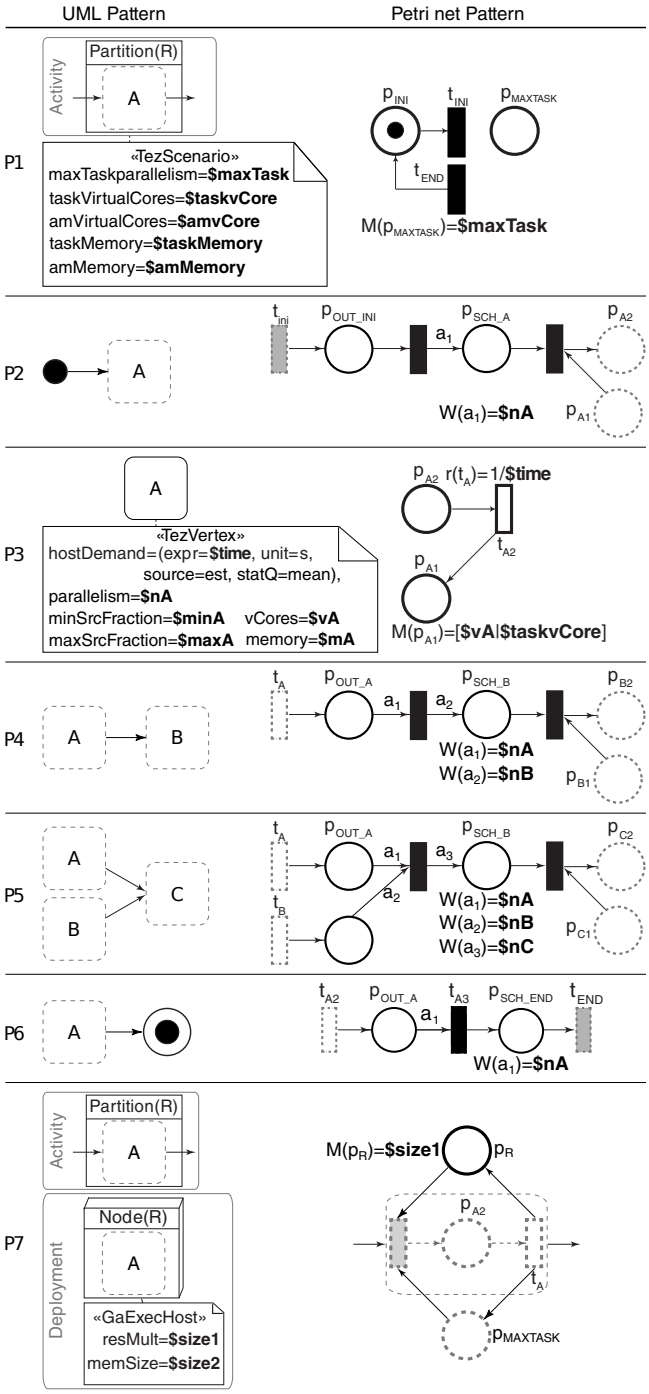

Figure 3: Transformation patterns I

imposed by places PB1_i. Besides, Pattern *P9* adds conditions in the immediate transitions of the Petri net for controlling the scheduling and pre-launching of tasks (min/maxSrcFraction properties).

Tez operations are first logically grouped into partitions in the activity diagram. Later on, they are deployed as artifacts in the

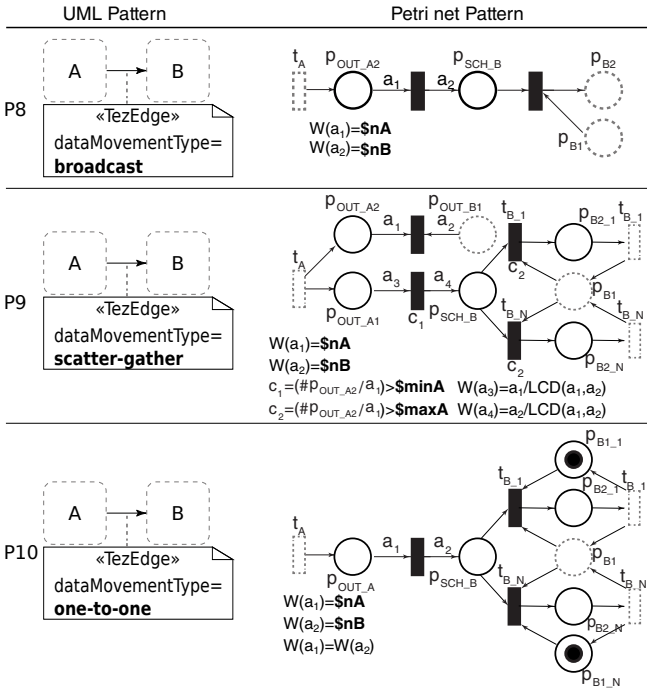

Figure 4: Transformation patterns II

deployment diagram and mapped to physical execution nodes. Pattern *P7* maps the *GaExecHost* stereotype to a new place p_R with an initial marking representing the number of computational cores of the node, *resMult*. The addition of such place restricts the logical concurrency (number of tasks of the Tez application) to the number of available cores. A similar place to p_R should be added for representing the memory size of the node, *memSize*.

Finally, a GSPN model is obtained by applying the patterns and combining the subnets through the interfaces. Pattern *P1*, together with *P6*, get a closed Petri net which enables steady-state analysis. The performance metrics of a Tez application are calculated as:

- *Response time* = 1 / average(t_{end}), where average(t_{end}) is the mean throughput of the transition.
- *Throughput* = average(t_{end}).
- *Utilization*= 1 - [average(p_R)/$\$size$1], where average($p_R$) is the mean number of tokens in p_R during the simulation.

6 CONCLUSION

Frameworks for developing applications in the Big Data context, like Apache Tez, are very useful for managing extremely large data volumes. However, such frameworks are complex enough for developers to leverage all the potential they offer. Moreover, they are young and very few experiences have been reported by practitioners.

This work introduces a domain-specific modeling language for Apache Tez, which guides the engineer to manage the key performance concepts of the framework. We have implemented the Tez profile for the Papyrus Modeling environment in the Eclipse platform [17].

Besides, we have proposed transformation patterns to get a performance model from the Tez models; specifically, stochastic Petri nets. Following this proposal, engineers can asses the performance impact of hosting their services in specific servers or clusters (e.g., parametrizing number of cores or CPU demands), by predicting response times and throughputs of the application or the utilization of the devices.

As immediate future work, we want to validate our approach using a real case study. Also, we plan to automatize and integrate the transformation patterns and performance calculators within the DICE Simulation tool [18].

7 ACKNOWLEDGEMENTS

This work has received funding from: the EU H2020, grant agreement No.644869 (DICE), the Spanish MINECO project CyCriSec [TIN2014-58457-R], the UZCUD2017-TEC-09 project, and the Aragon Government Ref. T27 - DISCO research group.

REFERENCES

[1] 2017. DICE H2020 Website. (2017). http://www.dice-h2020.eu/
[2] Apache. 2017. Apache Tez Website. (2017). http://tez.apache.org/
[3] Danilo Ardagna et al. 2016. Modeling Performance of Hadoop Applications: A Journey from Queueing Networks to Stochastic Well Formed Nets. In *Proceedings of the 16th International Conference on Algorithms and Architectures for Parallel Processing*. Springer, Cham, 599–613.
[4] Giovanni Chiola et al. 1993. Generalized Stochastic Petri nets: A Definition at the Net Level and its Implications. *IEEE Transactions on Software Engineering* 19, 2 (1993), 89–107.
[5] G. Casale et al. 2015. DICE: Quality-driven Development of Data-intensive Cloud Applications. In *Proceedings of the Seventh International Workshop on Modeling in Software Engineering (MiSE '15)*. IEEE Press, NJ, USA, 78–83.
[6] Eugenio Gianniti et al. 2017. Fluid Petri Nets for the Performance Evaluation of MapReduce and Spark Applications. *ACM SIGMETRICS Performance Evaluation Review* 44, 4 (2017), 23–36.
[7] Johannes Kroß et al. 2015. Modeling Big Data Systems by Extending the Palladio Component Model. *Softwaretechnik-Trends* 35, 3 (2015).
[8] Johannes Kroß and Helmut Krcmar. 2016. Modeling and Simulating Apache Spark Streaming Applications. *Softwaretechnik-Trends* 36, 4 (2016).
[9] François Lagarde et al. 2007. Improving UML profile design practices by leveraging conceptual domain models. In *Proceeedins of the 22nd IEEE/ACM International Conference on Automated Software Engineering*. ACM, Atlanta, 445–448.
[10] Marco Ajmone Marsan et al. 1994. *Modelling with Generalized Stochastic Petri nets* (1st ed.). John Wiley & Sons, Inc., New York, NY, USA.
[11] OMG. 2011. UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems, Version 1.1. http://www.omg.org/spec/MARTE/1.1/. (2011).
[12] Rajiv Ranjan. 2014. Modeling and Simulation in Performance Optimization of Big Data Processing Frameworks. *IEEE Cloud Computing* 1, 4 (2014), 14–19.
[13] Kritwara Rattanaopas. 2017. A performance comparison of Apache Tez and MapReduce with data compression on Hadoop cluster. In *Proceedings 14th International Joint Conference on Computer Science and Software Engineering*, IEEE (Ed.). 1–5.
[14] José Ignacio Requeno et al. 2017. Performance Analysis of Apache Storm Applications using Stochastic Petri Nets. In *Proceedings of the 5th IEEE International Workshop on Formal Methods Integration*, IEEE (Ed.). 1–8.
[15] Bran Selic. 2007. A Systematic Approach to Domain-Specific Language Design Using UML. In *Proceedings of the 10th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*. IEEE Computer Society, 2–9.
[16] Rupinder Singh and Puneet Jai Kaur. 2016. Analyzing performance of Apache Tez and MapReduce with Hadoop multinode cluster on Amazon cloud. *Journal of Big Data* 3, 1 (2016), 19.
[17] The DICE Consortium. 2017. Apache Tez Profile. (2017). https://github.com/dice-project/DICE-Profiles.
[18] The DICE Consortium. 2017. DICE Simulation Tool. (2017). https://github.com/dice-project/DICE-Simulation/.
[19] UML2 2011. Unified Modeling Language: Infrastructure. (2011). Version 2.4.1, OMG document: formal/2011-08-05.