Performance Analysis of Apache Storm Applications using Stochastic Petri Nets

J. I. Requeno and J. Merseguer
Dpto. de Informática e Ing. de Sistemas
Universidad de Zaragoza, Spain
{nrequeno, jmerse}@unizar.es

S. Bernardi
Centro Universitario de la Defensa
AGM, Spain
simonab@unizar.es

***Abstract*—Real-time data-processing applications, such as those developed using Apache Storm, need to address highly demanding performance requirements. Engineers should assess these performance requirements while they configure their Storm designs to specific execution contexts, i.e., multi-user private or public cloud infrastructures. To this end, we propose a quality-driven framework for Apache Storm, that covers the following steps. The design with UML, using a novel profile for Apache Storm, allowing performance metrics definition. The transformation of the design into a performance model, concretely stochastic Petri nets. Last but not least, the simulation of the performance model and the retrieval of performance results.**

***Keywords*-Apache Storm; Performance analysis; Petri net;**

I. Introduction

The Apache Storm technology [1] is currently used by a large number of companies and products, such as in Twitter, Yahoo! or Flipboard. Storm helps for improving real-time analysis, the customization of searches, news and advertisements, and the optimization of a wide range of online services that require low-latency processing. Applications developed using the Storm technology are then very demanding in terms of performance and, definitely, they are also highly customizable by parameters that greatly impact their end to end performance.

A Capgemini research [2] shows that only 13% of organizations have achieved full-scale production for their Big Data applications. As a Big Data technology, the case of Storm is not different, we could even say that it is worse due to the youthfulness of the technology. Thus, there is now an urgent need for novel, performance oriented, software engineering methodologies and tools capable of dealing with the complexity of such a new environment.

In this paper, we present an approach for performance assessment of Apache Storm applications. In particular, we offer a modeling approach and a novel UML profile for a better performance characterization of a Storm design. We define transformations for these UML Storm designs into suitable models that are used for performance analysis, concretely stochastic Petri nets.

Our approach, with its corresponding tools and formalisms, can be applied to predict the behaviour of the application for future demands or to study the impact, in some performance parameters (e.g., response time, throughput or utilization), of highly varying workloads. Besides, the detection of performance bottlenecks using formal methods can be more effective and easier than in a real-world testbed. On the modeling side, the profiled-UML allows to work with the Apache Storm performance parameters in the very same model used for the workflow and deployment definitions. Moreover, the developer takes advantage of all the facilities provided by a UML software development environment. These reasons recommend the UML modeling, instead of doing it directly with the stochastic Petri net, that can be merely obtained by transformation.

Several approaches have been already presented in the literature for the modeling and performance assessment of stream applications [3], [4] or big data platforms [5], [6]. Some of these studies use variants of Petri nets, and they are applied in a generic context for stream processing [4] or distributed systems [7], [8]. Definitely, to the best of our knowledge, this is the first work devoted to the Apache Storm performance evaluation using formal methods.

The rest of the paper is organized as follows. Section II presents the basics on Apache Storm, focussing on the parameters that mainly affect the performance of an application. Section III presents our performance modeling approach for Apache Storm applications. Section IV details the transformation to get a performance model out of a Storm design. Section V is devoted to the validation of the approach. Finally, Section VI draws a conclusion and presents future work.

II. Storm and Performance

Storm is a distributed real-time computation system for processing large volumes of high-velocity data [1]. A Storm application is usually designed as a directed acyclic graph (DAG) whose *nodes* are the points where the information is generated or processed, and the *edges* define the connections for the transmission of data from one node to another. Two classes of nodes are considered in the topology. On the one hand, *spouts* are sources of information that inject *streams* of data into the topology at a certain *rate*. On the other hand, *bolts* elaborate input data and produce results which, in turn, are emitted towards other bolts of the topology. The notions of *tuples* and *messages* are equivalent. By default, a Storm

978-1-5386-1562-1/17 $31.00 © 2017 IEEE
DOI 10.1109/IRI.2017.64

application runs indefinitely until killed. Figure 1 represents the DAG of a Storm application made of two spouts and three bolts in a pipeline layout.

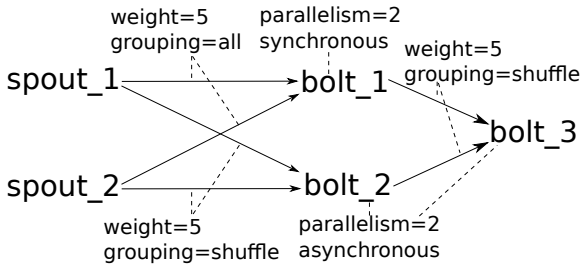

Figure 1. A Storm Application

A bolt is a generic processing component that requires *n* tuples for producing *m* results. This asymmetry is captured by the *weights* in the arcs of the DAG. They represent the number of tuples the next bolt requires for emitting a new message. Besides, different *synchronization* policies shall be considered. A bolt receiving messages from two or more sources can select to either a) progress, if at least a tuple from any of the sources is available (*asynchronously*), or b) wait for a message from all the sources (*synchronously*).

A Storm application is also configurable by the parallelism of the nodes, the stream grouping and the scheduling. The *parallelism* specifies the number of concurrent threads executing the same task (spout or bolt). The *stream grouping* determines the way a message is propagated to and handled by the receiving nodes. By default, a message is broadcasted to every successor of the current node. Once the message arrives to a bolt, it is either redirected randomly to any of the multiple internal threads (*shuffle*), copied to all of them (*all*) or copied to a specific subset of threads according to some criteria (e.g, *field*, *global*, etc.).

Finally, the Storm *scheduling algorithm* deploys statically the spouts and bolts to the computational resources of the cluster, at the beginning of the execution. 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.

In summary, a Storm framework is highly configurable by various parameters that will influence the final performance of the application (see Table I).

III. Modelling Storm Applications with UML

Our modelling approach for Storm applications is oriented to performance evaluation and, initially, it uses UML diagrams. At least, we need to model the Storm topology, i.e., the DAG, the performance parameters already identified in Section II and the deployment. Therefore, we will work with *activity diagrams* complemented with *deployment diagrams*.

Table I
Storm Concepts for Performance

#	Concept	Meaning
1.	*Spout* (task)	Source of information
2.	*Rate*	No. of tuples per unit of time produced by a spout
3.	*Bolt* (task)	Data elaboration
4.	*Weight*	No. of tuples required by a bolt
5.	*Asynchronous policy*	The bolt progresses when at least one input tuple is available
6.	*Synchronous policy*	The bolt progresses when all input tuples are available
7.	*Parallelism*	No. of concurrent threads per task
8.	*Grouping*	Tuple propagation policy (e.g., all)
9.	*Scheduling*	Deployment of tasks

A. UML Diagrams for Storm

Figure 2 shows the UML activity diagram for the example of the Storm topology in Figure 1. A UML activity diagram for Storm will always start with a set of initial nodes connected to the spout tasks because they are the sources of information responsible of inserting tuples in the topology. The rest of the tasks (i.e., bolts) will follow according to the Storm *synchronization policy* declared for them. In particular, when a given bolt is declared as *synchronous* then we collapse all its incoming edges into a join node (bar). See, for example, **bolt_1** in Figure 2 that was declared as synchronous in Figure 1. This means that **bolt_1** will not progress until all incoming tuples are ready. When, instead, a bolt is *asynchronous* then we collapse all its incoming edges into a merge node (diamond). See, for example, **bolt_2** in Figure 2 that was declared as asynchronous in Figure 1. This means that **bolt_2** will progress when the very first tuple arrives.

In our approach, a UML activity diagram is interpreted as the DAG of a particular Storm topology. The semantic is different from the standard one of a UML activity diagram. In our case, the actions (rounded rectangles representing spouts and bolts) are tasks that continuously process streams according to the characteristics of the Storm technology. The standard UML semantic considers the actions as tasks that finalize once they have processed the input data. Besides, the arcs connecting activities do not represent a logical succession of actions, as in standard UML, but a communication channel between two tasks (i.e., spouts and/or bolts).

B. A UML Profile for Storm

Once the Storm topology and synchronization policies have been represented, we still need to address the rest of concepts in Table I. We decided to convert them into stereotypes and tags, which are the extension mechanisms offered by UML. Therefore, we devised a UML profile for Storm. A UML profile is a set of stereotypes that can be applied to UML model elements for extending their semantics [9], [10]. In our case, we are extending UML with the Storm concepts that impact on performance.

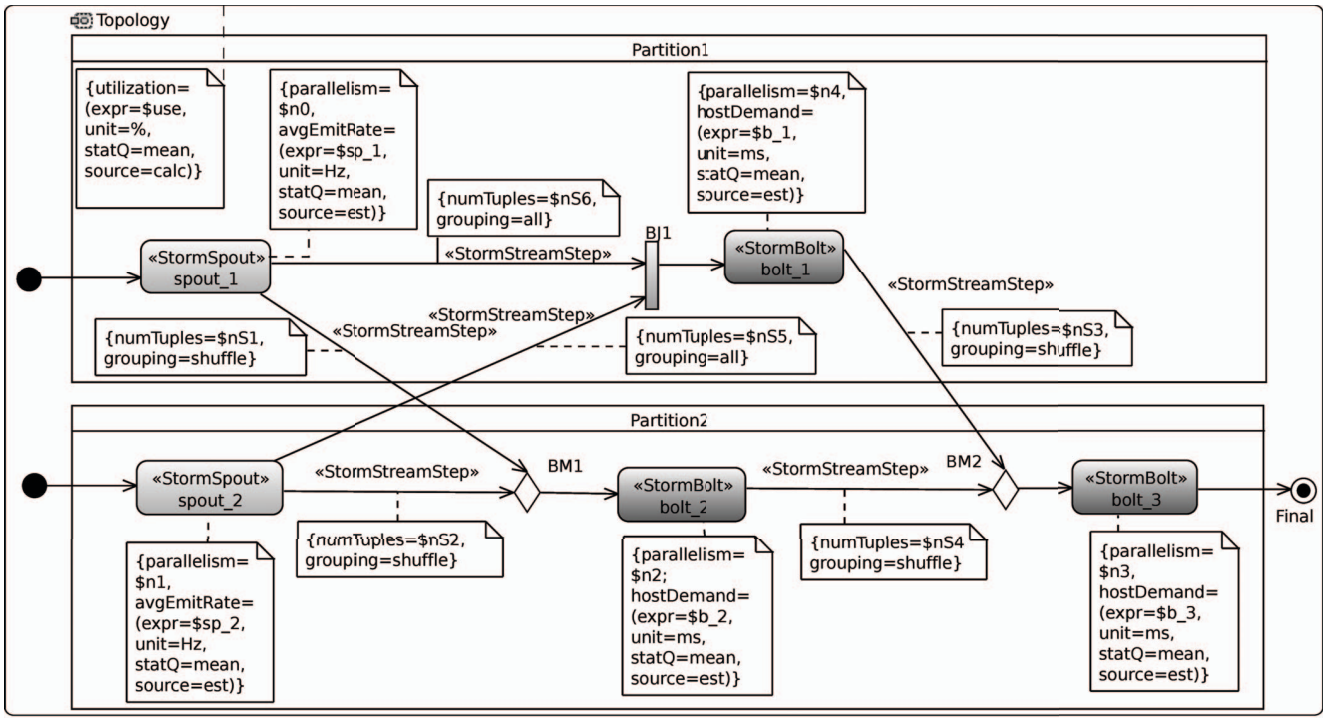

Figure 2. An Activity Diagram for Storm with Profile Annotations

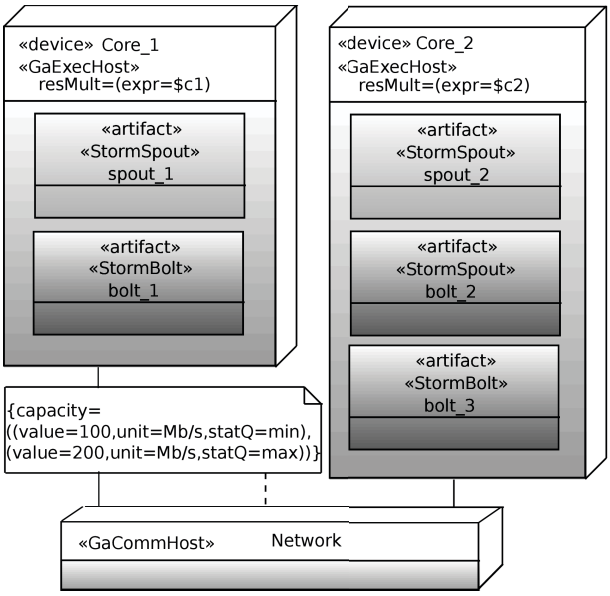

Figure 3. A Deployment Diagram for Storm

The Storm profile heavily relies on the standard MARTE profile [11]. This is because MARTE offers the GQAM sub-profile, a complete framework for quantitative analysis, which is indeed specialized for performance analysis, then perfectly matching to our purposes. Moreover, MARTE 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 latter, 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 Storm-profiled models with two main goals: (i) to specify the input parameters of the model and (ii) to specify the performance metric(s) that will be computed for the model (i.e., the output results). An example of a VSL expression for a host demand tagged value of type `NFP_Duration` is:

```
expr=$b1, unit=ms, statQ=mean, source=est
  (1)       (2)       (3)         (4)
```

This expression specifies that **bolt_1** in Figure 2 demands $\$b_1$ (1) *milliseconds* (2) of processing time, whose mean value (3) will be obtained from an estimation in the real system (4). $\$b_1$ is a variable that can be set with concrete values during the analysis of the model.

Another interesting VSL expression is the definition of the performance metric to be calculated, the utilization in the example of Figure 2 :

```
expr=$use, unit=%, statQ=mean, source=calc
  (1)       (2)       (3)         (4)
```

This expression specifies that we want to calculate (4) the `utilization`, as a percentage of time (2), of the whole

system or a specific resource, whose mean value (3) will be assigned to variable `$use` (1). Such a value is obviously obtained from the performance model.

On the other hand, the nodes (actions, joins and merges) in the UML activity diagram are grouped by partitions (e.g., Partition1 and Partition2, in Figure 2). Each partition is mapped to a computational resource in the UML deployment diagram following the *scheduling* policy defined for the topology. Figure 3 shows the deployment diagram, which complements the previous activity diagram. Each computational resource is stereotyped as *GaExecHost* and defines its resource multiplicity, i.e., number of cores. The deployment also allows one to know which messages exchanged by tasks can introduce network delays, i.e., tuples exchanged between cores in different physical machines, which is of importance for the eventual performance model. Therefore, we use the *GaCommHost* stereotype. Both stereotypes are inherited from MARTE GQAM.

Apart from the two aforementioned stereotypes, the Storm profile also provides genuine stereotypes (see Table II) for representing those parameters not already addressed, i.e., concepts *1–4*, *7* and *8* in Table I. *Bolts* and *spouts* have independent stereotypes because they are conceptually different, however they both inherit from MARTE::GQAM::GaStep stereotype since they are computational steps. Moreover, they share the *parallelism*, or number of concurrent threads executing the task, which is specified by the tag `parallelism`. On the other hand, the spouts add the tag `avgEmitRate`, which represents the *rate* at which the spout produces tuples. Finally, the bolts use the `hostDemand` tag from GaStep for defining the task execution time.

The Storm concept of *stream* is captured by the *StormStreamStep* stereotype. It also inherits from the MARTE::GQAM::GaStep stereotype, which enables to apply it to the control flow arcs of the activity diagram. The stereotype has three tags: `numTuples` and `grouping` match the *weight* and *grouping* concepts, respectively; the `probFields` is an array of reals that is used when the type of `grouping` is equal to *field*. The array specifies the probabilities p_i that a message, transmitted through the *StormStreamStep*, arrives to the threads t_i of the target bolt. The value of p_i can be obtained at runtime experimentally, i.e., by tracing the messages grouped by the bolt thread.

IV. Transformation of the UML Design

For evaluation of the already defined metrics (e.g., utilization or throughput), we need to transform the Storm design into a performance model. We choose as target performance model Generalized Stochastic Petri Nets (GSPN) [12]. Petri nets are suitable for modelling software systems (see Appendix VII). In the following, we propose a set of original *transformation patterns*; each pattern takes as input a part of the Storm design and produces a GSPN subnet. These patterns have been used to implement a model-to-model transformation (M2M) [13] that automatically generates the GSPN model. The correctness and compositionality of the transformation patterns are validated experimentally in Section V.

Table II
Storm Profile Extensions

Storm concept	Stereotype	Tag
Bolt	≪StormBolt≫	
Exec. Time		hostDemand
Spout	≪StormSpout≫	
Rate		avgEmitRate
Parallelism		parallelism
Stream	≪StormStreamStep≫	
Weight		numTuples
Grouping		grouping
		probFields
Scheduling	≪GaExecHost≫	resMult
Partition		
	≪GaCommHost≫	capacity

A. Activity Diagram Transformation

Figures 4–5 show the patterns for the activity and deployment diagrams. For each figure, the left hand side presents the input of the pattern, i.e., the UML model elements, possibly stereotyped with the Storm profile. The right hand side indicates the corresponding GSPN subnet. For an easier understanding of the transformation, we depicted: a) text in bold to match input and output elements; b) interfaces with other patterns as dotted grey elements, because they actually do not belong to the pattern.

Patterns *P1* and *P2* map spout and bolt tasks, respectively. Both spout and bolt subnets become structurally similar when the bolt subnet is merged with a *P3–P5* pattern. The subnet consists of two places, a timed transition, an immediate transition, and a set of arcs. Places p_{A1} and p_{A2} represent, respectively, the idle state and the processing state of incoming messages. The place p_{A1} is marked with as many tokens as the *parallelism* tagged-value associated to the task denotes (**$n0**). The rate of the timed transition is equal to either the emission rate (**$rate**) of the spout or the inverse of the host demand of the bolt (1/ **$time**). The timed transitions have an infinite server semantics because the production of tuples is already constrained by the number of available threads (tokens) defined by the parallelism. The immediate transition in the spout subnet does not have source places because it models the continuous arrival of new messages.

Patterns *P3*, *P4* and *P5* map the reception of a stream of tuples by a bolt. In *P3* the source of the stream is only one task, whereas in *P4* and *P5* there are multiple sources. In particular, the pattern *P4* represents the synchronous case and the pattern *P5* is the asynchronous one. In *P3–P5* subnets, the interface transition t_A refers to the transition

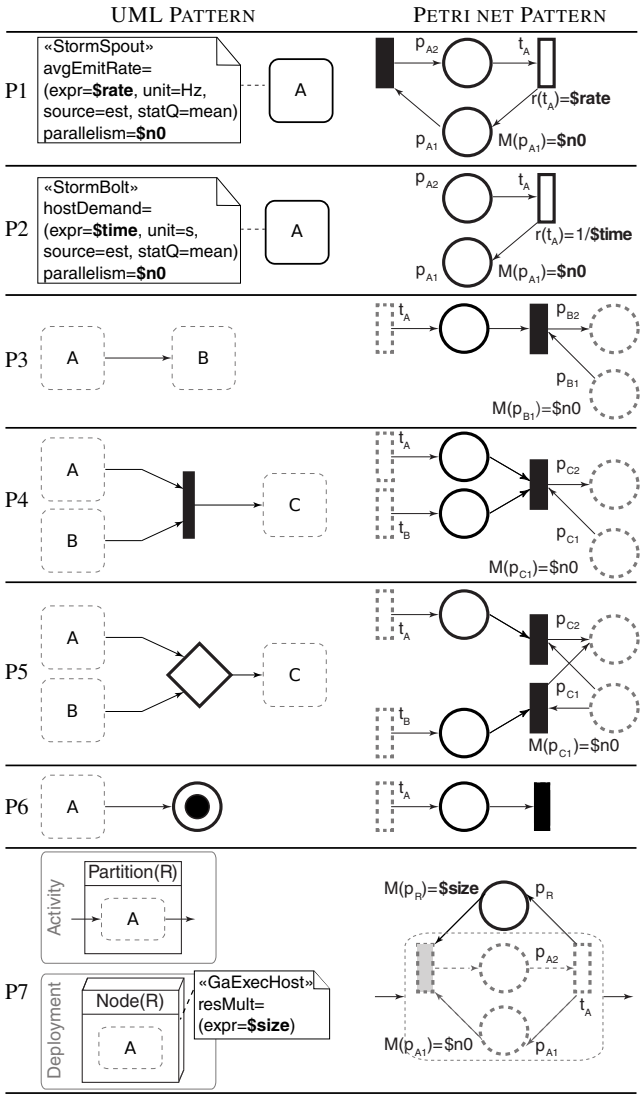

Figure 4. Transformation Patterns for Storm I

in *P2* with the same name. Pattern *P6* maps the final node to a transition without output places. This is a sink transition that represents the end of the stream processing and could potentially act as interface with subsequent systems (i.e., injecting tuples in another Storm application).

Patterns *P8–P11* detail the transformation of the *numTuples* and *grouping* tagged-values of a given stream step. Therefore, these patterns refine patterns *P3*, *P4* and *P5*. The *numTuples* indicates the number of input tuples that the receiving bolt requires for producing a message. Then, such a value is mapped to the weight of the arcs a_2 (*P8* subnet), a_1 (*P9–P10* subnets), and a_i (*P11* subnet).

Additionally, the *grouping* defines how the stream should be partitioned among the next bolt's threads. If the grouping is set to *all*, every thread of the receiving bolt will process a copy of the tuple, then the weight of the arc a_1 in the GSPN subnet is equal to the *parallelism* of the bolt **B** (*P8*). Otherwise, only one thread of the receiving bolt will process the tuple, therefore, the weight will be set to the default value (i.e., 1). If the grouping policy is *shuffle*, the target execution thread of **B** is selected randomly among the idle ones (*P9*). In the case of *global* policy, the entire stream goes to the bolt's thread with the lowest id (*P10*). The initial marking of place p_{BG}, in the GSPN subnet, is set to a single token for restricting the access to just one thread. Finally, the *field* grouping policy divides the outgoing stream of **A** by value (*P11*) and all the messages having the same value are sent to the same threads of the receiving bolt. The transformation creates a GSPN subnet with **n** basic subnets, where **n** is the number of different stream values. This number is limited by the number of parallel threads (*parallelism* tagged-value) in **B**. When an incoming message arrives to the receiving bolt, it is redirected to one of the basic subnets according to the probabilities $\mathbf{\$prob}_i$ assigned to the immediate transitions t_{B1_i}.

The rest of the Apache Storm grouping policies are not considered neither by the Profile nor the transformation yet. They are variants of the previous ones and both the profile and the transformation can be adapted accordingly. For instance, the *partial key* grouping is a *field* grouping that balances the stream load between two downstream tasks instead of a single one. The *local* and *none* groupings are equivalent to a *shuffle* grouping, but they prioritize stream connections among spouts and bolts inside the same computational node. Finally, the *direct* grouping specifies an explicit connection between threads of **A** and **B**.

B. Deployment Diagram Transformation

Pattern *P7* (Figure 4) illustrates the modifications introduced in the GSPN model by the profile extensions in the deployment diagram. The Storm tasks are first logically grouped into partitions in the activity diagram, later they are deployed as artifacts and mapped to physical execution nodes (*GaExecHost* stereotype) in the deployment diagram. In particular, *P7* maps the *GaExecHost* to a new place p_R in the GSPN, with an initial marking that represents the number of computational cores of the node (*resMult* tagged-value). The addition of such place restricts the logical concurrency, that is the number of threads of the Storm tasks, to the number of available cores. The pattern corresponds to the acquire/release operations of the cores by the spouts and bolts.

C. Performance Model and Implementation

Figure 6 shows the final GSPN model for the Storm design in Figures 2 and 3. It has been obtained by applying the patterns and combining the subnets through the interfaces. The image of the GSPN model has been simplified for readability purposes.

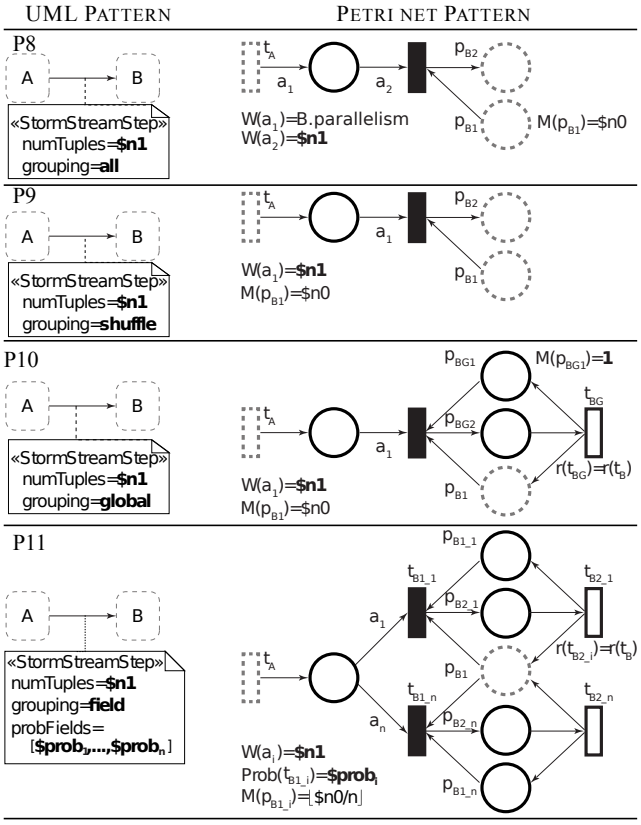

Figure 5. Transformation Patterns for Storm II

The **spout_1** has parallelism \$n0 and the **spout_2** has parallelism \$n1. The **bolt_1** requires \$nS6 messages from **spout_1** and \$nS5 messages from **spout_2** with *all* grouping policy. The **bolt_2** requires \$nS1 messages from **spout_1** or \$nS2 messages from **spout_2** with *shuffle* grouping policy. Finally, the **bolt_3** requires \$nS3 messages from **bolt_1** or \$nS4 messages from **bolt_2** with *shuffle* grouping policy.

The Storm profile, the transformation patterns, and the evaluation of performance metrics have been implemented for the Papyrus Modelling environment in Eclipse. In particular, they are completely integrated within the DICE Simulation plugin for Eclipse.

The transformation of the UML models to stochastic Petri nets, as well as the evaluation of the performance metrics, are fully automatized and they are transparent to the end user. Firstly, the transformation uses QVT-MOF 2.0 [14] to obtain a Petri Net Markup Language file (PNML) [15], an ISO standard for XML-based interchange format for Petri nets. A trace file is created during the M2M transformation. This file links the elements of the Petri net with the source components of the UML. It helps for the identification of the items in the Petri net that the tool needs to inspect during the performance analysis. Later on, Acceleo [16] has been used to implement a model-to-text (M2T) transformation from PNML into a GSPN tool specific format, concretely for the GreatSPN tool [17].

The UML Profile for Storm is published inside the DICE Profile [9] and can be downloaded from [18]. The transformation of UML profiled models into Petri nets is implemented in the DICE Simulation tool [19]. The code of the transformation and the DICE Simulation tool are publicly available. They can be downloaded from [20].

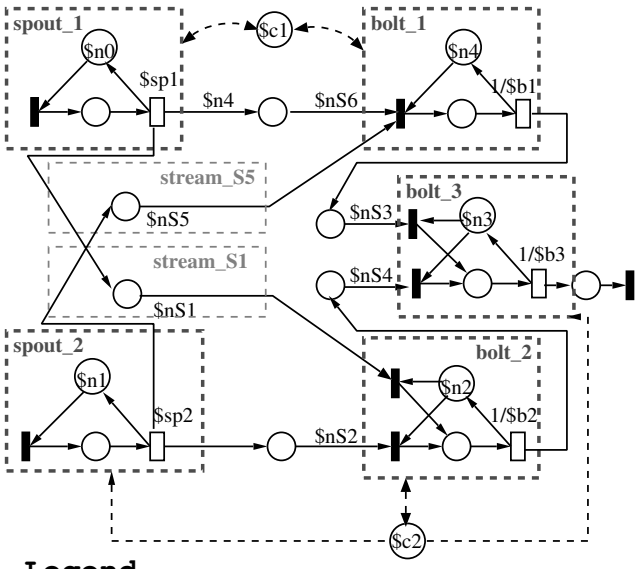

Figure 6. GSPN for the design in Figs. 2 and 3

V. Validation of the Performance Model

This section addresses the validation of the transformation patterns we have proposed and implemented. To this aim, we first applied the transformation to get automatically the GSPN model in Figure 6 from the UML models in Figures 2 and 3. Then, we analysed the GSPN using event driven simulation techniques to estimate the performance metrics. Later, we deployed the Storm application, specified in Figures 2 and 3, and we compared the estimated results, obtained via simulation, to the results measured by monitoring the Storm application in operation.

The aforementioned deployment was distributed in a cluster with two workstations. All the workstations were characterized by Intel(R) Core(TM) i7-6700 CPUs (3.40GHz) with 8 cores, 32GBytes of RAM, a Gigabit ethernet and Ubuntu Linux OS (version 14.04).

Emission rate or host Demand (ms)				%Utilization (%Relative error)		
$\$sp_i$	$\$b_1$	$\$b_2$	$\$b_3$	bolt_1	bolt_2	bolt_3
20	100	100	100	97,6 (2,45)	100 (9,91)	39,0 (3,71)
30	100	100	100	100 (2,91)	100 (4,48)	43,2 (6,05)
40	100	100	100	100 (1,00)	99,5 (0,40)	38,1 (4,50)
50	100	100	100	83,1 (4,61)	78,0 (2,07)	33,1 (2,42)
100	100	100	100	39,4 (0,06)	38,7 (3,27)	17,0 (4,18)
20	20	30	40	41,2 (2,90)	56,4 (5,85)	32,0 (0,26)
30	20	30	40	26,9 (0,68)	40,1 (0,46)	20,8 (2,86)
40	20	30	40	21,2 (6,05)	29,9 (0,19)	16,6 (4,05)
50	20	30	40	16,5 (3,39)	24,7 (3,21)	12,8 (0,09)
100	20	30	40	9,5 (15,7)	11,9 (0,95)	7,3 (12,61)

Table III
RESULTS OF THE EXPERIMENTS

The parameters of the Storm configuration were as follows. The rates of the spouts and the host demands of the bolts were parameterized ($\$sp_i$ and $\$b_i$), in the GSPN model and in the real application. The parallelism for the spouts and bolts was set to 2, respectively. The number of input tuples required for the bolts were set to 5 for all bolts (cf., arc weights $\$nS\_i$ for i=1..6, in Figure 6).

The performance metric of reference for the validation was the utilization. In particular, we considered the utilization of each bolt, i.e., the percentage of time that the threads associated to a bolt are active and processing tuples.

Concerning the performance analysis of the GSPN model, we used the event-driven simulator of the GreatSPN tool [17] (confidence level of 99% and accuracy of 3%). Analytical solvers and structural analysers are also integrated into the GreatSPN tool. They can be invoked for studying the properties of the GSPN. In the GSPN model (see *P2*, Fig. 4), the utilization is the mean number of tokens in the place p_{A2} divided by the initial marking of the place p_{A1}. On the other hand, the Apache Storm monitoring platform provided us with the result of the bolts utilization.

Table III shows the utilization of each bolt measured by monitoring the Storm application and, in parenthesis, the relative error with respect to the result estimated by simulation of the GSPN model. The utilization estimated by simulation of the GSPN model is not provided for space limitation reason. Each row of the table represents a different emission rate (spouts) and host demand (bolts).

For all cases but one, the relative error is lower than 15%. Accordingly, we can consider that the GSPN estimations are quite good. From the experiments, we get the following insight. When the spouts insert tuples at a low rate (e.g., every 100 ms) and the bolts execute low time-consuming functions (e.g., **$b_1** with execution time of 20 ms), the bolt threads will be idle most of the time (e.g., 9.5% utilization of **bolt_1** in Table III). Conversely, some bolts will be saturated (e.g., 100% utilization of **bolt_2** in Table III) in case of high production rate of the spouts (e.g., every 40 ms or less) with respect to the bolts execution times (e.g., 100 ms).

VI. CONCLUSION

The paper presents a novel approach for the modeling and performance analysis of Storm applications. The goal is to guide software engineers for increasing the quality of their systems during the design. For example, they can assess the performance impact on a specific server or cluster, by predicting response times, throughputs or utilizations. The experimental results confirm the feasibility of the approach.

The approach is unique in some aspects. We have introduced a new UML profile for Storm, which captures the concepts needed for performance evaluation. We have proposed transformation patterns to get a performance model. We have used Generalized Stochastic Petri Nets as the formalism for performance analysis. Last but not least, we have integrated the approach (profiling, UML-to-GSPN transformation and analysis via event-driven simulation) in a publicly available tool [20].

VII. APPENDIX: PETRI NETS

A GSPN is a Petri net with a stochastic time interpretation, therefore suitable for performance analysis purposes. A GSPN model is a bipartite graph of two types of vertices: places and transitions. Places are graphically depicted as circles and may contain tokens. A token distribution, namely a marking, represents a state of the modelled system. The dynamics are governed by the transition enabling and firing rules, where places represent pre- and post-conditions for transitions. In particular, the firing of a transition removes (adds) as many tokens from its input (output) places as the weights of the corresponding input (output) arcs. Transitions can be immediate, those that fire in zero time; or timed, those that fire after a delay which is sampled from a (negative) exponentially distributed random variable.

In our Storm performance model, places represent the intermediate steps of the stream processing. Transitions represent the execution of Storm tasks and fire when certain conditions are met (e.g., the synchronous or asynchronous reception of messages in a bolt) or the associated time delay has elapsed (e.g., production of a processing result from input tuples by a bolt). Besides, tokens represent either the messages sent between tasks or the resources of the application in case of multi-threading.

ACKNOWLEDGMENT

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

REFERENCES

[1] Apache Storm. URL:http://storm.apache.org/.

[2] M. Colas *et al.*, "Cracking the Data Conundrum: How Successful Companies Make Big Data Operational," Capgemini consulting, Tech. Rep., 2014, URL: https://www.capgemini-consulting.com/cracking-the-data-conundrum.

[3] F. Nalepa *et al.*, "Model for Performance Analysis of Distributed Stream Processing Applications," in *Procs. 20th DEXA*. Springer, 2015, pp. 520–533.

[4] ——, "Performance Analysis of Distributed Stream Processing Applications Through Colored Petri Nets," in *Procs. 10th MEMICS*. Springer, 2015, pp. 93–106.

[5] R. Ranjan, "Modeling and Simulation in Performance Optimization of Big Data Processing Frameworks," *IEEE Cloud Computing*, vol. 1, no. 4, pp. 14–19, 2014.

[6] R. Singhal and A. Verma, "Predicting Job Completion Time in Heterogeneous MapReduce Environments," in *Procs. 30th IEEE IPDPS*. IEEE, 2016, pp. 17–27.

[7] S. Samolej and T. Rak, "Simulation and Performance Analysis of Distributed Internet Systems using TCPNs," *Informatica*, vol. 33, no. 4, 2009.

[8] T. Rak, "Response Time Analysis of Distributed Web Systems using QPNs," *Mathematical Problems in Engineering*, 2015.

[9] "DICE Consortium Tech. Reports. Design and Quality Abstractions [21]."

[10] A. Gómez *et al.*, "Towards a UML Profile for Data Intensive Applications," in *Procs. 2nd QUDOS*, 2016, pp. 18–23.

[11] OMG, "UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems, Version 1.1," URL: http://www.omg.org/spec/MARTE/1.1/, Object Management Group, Juny 2011.

[12] M. A. Marsan *et al.*, *Modelling with Generalized Stochastic Petri nets*, 1st ed. NY, USA: John Wiley & Sons, 1994.

[13] S. Kent, "Model Driven Engineering," in *Procs. 3rd iFM*, ser. LNCS, vol. 2335. Springer, 2002, pp. 286–298.

[14] OMG, "Meta Object Facility (MOF) 2.0 Query/-View/Transformation Specification, Version 1.1," Object Management Group, January 2011. [Online]. Available: http://www.omg.org/spec/QVT/1.1/

[15] ISO, "Systems and software engineering – High-level Petri nets – Part 2: Transfer format," Geneva, Switzerland, ISO/IEC 15909-2:2011, 2008.

[16] The Eclipse Foundation & Obeo, "Acceleo," 2015, URL: https://eclipse.org/acceleo/.

[17] Dip. di informatica, Universitá di Torino, "GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets," 2015, URL: http://www.di.unito.it/~greatspn/index.html.

[18] DICE Consortium, "DICE Profile," 2017, URL: https://github.com/dice-project/DICE-Profiles.

[19] "DICE Consortium Tech. Reports. Transformations to Analysis Models [21]."

[20] DICE Consortium, "DICE Simulation Tool," 2017, URL:https://github.com/dice-project/DICE-Simulation/.

[21] ——, "Technical Reports," 2016-2017, URL: http://www.dice-h2020.eu/deliverables/.