

Noname manuscript No.
(will be inserted by the editor)

Quantitative Analysis of Apache Storm Applications: The NewsAsset Case Study

José I. Requeno · José Merseguer · Simona Bernardi · Diego Perez-Palacin · Giorgos Giotis · Vasilis Papanikolaou

Received: date / Accepted: date

Abstract The development of Information Systems today faces the era of Big Data. Large volumes of information need to be processed in realtime, for example, for Facebook or Twitter analysis. This paper addresses the redesign of NewsAsset, a commercial product that helps journalists by providing services, which analyze millions of media items from the social network in realtime. Technologies like Apache Storm can help enormously in this context. We have quantitatively analyzed the new design of NewsAsset to assess whether the introduction of Apache Storm can meet the demanding performance requirements of this media product. Our assessment approach, guided by the Unified Modeling Language (UML), takes advantage, for performance analysis, of the software designs already used for development. In addition, we converted UML into a domain-specific modeling language (DSML) for Apache Storm, thus creating a *profile* for Storm. Later, we transformed said DSML into an appropriate language for performance evaluation, specifically, stochastic Petri nets. The assessment ended with a successful software design that certainly met the scalability requirements of NewsAsset.

Keywords Apache Storm · UML · Petri nets · Software Performance · Software Reuse

José Ignacio Requeno, José Merseguer, Simona Bernardi and Diego Perez-Palacin
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza (Spain)
E-mail: {nrequeno,jmerse,simonab,diego}@unizar.es

Giorgos Giotis and Vasilis Papanikolaou
Athens Technology Center, ATC (Greece)
E-mail: {g.giotis, v.papanikolaou}@atc.gr

1 Introduction

Innovative practices for Information Systems development, like Big Data technologies, Model-Driven Engineering techniques or Cloud Computing processes have penetrated in the media domain. News agencies are already feeling the impact of these technologies (e.g., transparent distribution of information, sophisticated analytics or processing power) for facilitating the development of the next generation of applications. Especially, considering interesting media and burst events, which is out there in the digital world, these technologies can offer very efficient processing capabilities and can provide an added value to journalists.

Apache Storm (Apache, 2017a) is a free and open source distributed realtime computation system that can process a million tuples per second per node. Storm helps for improving real-time analysis, news and advertisements, the customization of searches, and the optimization of a wide range of online services that require low-latency processing. Today, the volume of information in Internet increases exponentially, especially that of interest for the media. For example, in the case of natural disasters, social or sportive events, the traffic of tweets or messages may rise up to 10 or 100 times with respect to the number of messages in a normal situation (Ranjan, 2014). Hence, applications developed using Apache Storm need to be very demanding in terms of performance and reliability.

This paper addresses, using Apache Storm, the redesign of NewsAsset, a commercial product developed by the Athens Technological Center (ATC, 2018). To this end, we apply a quality-driven methodology, that we already introduced in (Requeno et al., 2017), for the performance assessment of Apache Storm applications. For ATC the redesign also means to *reuse* coding of the

current version of the NewsAsset, then trying to impact only on the stream processing for leveraging Apache Storm. The simulation-based approach that we apply here is useful for predicting the behavior of the application for future demands, and the impact of the stress situations in some performance parameters (e.g., application response time, throughput or device utilization). Consequently, ATC gets, before reimplementing and deployment of the full application, a valuable feedback which saves coding and monetary efforts.

In particular, this paper extends the approach in (Requeno et al., 2017) with respect to the quality-driven methodology in different aspects. First, we improve the UML profile of the methodology for introducing a reliability characterization of Storm. Consequently, we convert UML into a DSML¹ for performance and reliability of Apache Storm applications. Second, we propose new transformations, into stochastic Petri nets (SPN), for some performance parameters of Storm not already addressed in (Requeno et al., 2017). Moreover, we introduce computation of reliability metrics by means of the UML profile. Consequently, our approach enables the performance and reliability assessment of Apache Storm applications. Finally, the application of the methodology to the NewsAsset case study has been useful to validate the approach in a real scenario and to assess ATC about its scalability.

On the modeling side, our DSML allows to work with the Apache Storm performance and reliability parameters in the very same model used for the workflow and deployment specifications. 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 SPN, that can be merely obtained by transformation.

Regarding the related work, (Ranjan, 2014) discusses the role of modeling and simulation in the era of big data applications and defends that they can empower practitioners and academics in conducting “what-if” analyses. (Singhal and Verma, 2016) develop a framework for efficiently set-up heterogeneous MapReduce environments and (Nalepa et al., 2015a,b) address the need of modeling and performance assessment in stream applications. More in particular, a generic profile for modeling big data applications is defined for the Palladio Component Model (Kroß et al., 2015). In (Kroß and Krcmar, 2016), the authors model and simulate Apache Spark streaming applications. Mathematical models for predicting the performance of Spark applications are introduced in (Wang and Khan, 2015). Some of these works use variants of the Petri nets, but they are ap-

plied in a generic context for stream processing (Nalepa et al., 2015b) or distributed systems (Samolej and Rak, 2009; Rak, 2015). Generalised stochastic Petri nets (Chiola et al., 1993), the formalism for performance analysis that we adopt here, have been already used for the performance assessment of Apache Hadoop MapReduce (et al., 2016). A recent publication uses fluid Petri nets for the modeling and performance evaluation of Apache Spark applications (et al., 2017). However, the work in (Requeno et al., 2017) was the first entirely devoted to the Apache Storm performance evaluation, combining a genuine UML profile and GSPNs, and the present work validates and extends it as aforementioned.

The rest of the paper is organized as follows. Section 2 introduces the NewsAsset case study. Section 3 recalls the basics on Apache Storm for performance and reliability and defines the DSML. Section 4 presents our performance modeling approach with focus on the case study. Section 5 is devoted to the performance analysis of NewsAsset. Finally, Section 6 draws a conclusion. Appendix A details the transformation to get performance models. Appendix B explains the computation of reliability metrics in a Storm design. Appendix C recalls basic notions of Generalized stochastic Petri nets (Chiola et al., 1993).

2 A Case Study in the Media Domain

Heterogeneous sources like social or sensor networks are continuously feeding the world of Internet with a variety of real data in a tremendous pace: media items describing burst events, traffic speed on roads, or air pollution levels by location. Journalists are able to access these data aiding them in all manner of news stories. It is the social networks like Twitter, Facebook or Instagram that people are using to watch the news ecosystem and try to learn what conditions exist in real-time. Subsequently, news agencies have realized that social-media content is becoming increasingly useful for news coverage and can benefit from this trend only if they adopt current innovative technologies that effectively manage such volume of information. Thus, the challenge is to catch up with this evolution and provide services that can handle the new situation in the media industry.

NewsAsset is a commercial product positioned in the news and media domain, branded by Athens Technology Center (ATC), a SME² located in Greece. NewsAsset suite constitutes an innovative management solution for handling large volumes of information offering a complete and secure electronic environment for storage, management and delivery of sensitive information

¹ Domain Specific Modeling Language.

² Small and medium-sized enterprise.

in the news production environment. The platform proposes a distributed multi-tier architecture engine for managing data storage composed by media items such as text, images, reports, articles or videos.

NewsAsset is built on the achievements of a EU-funded project, namely SocialSensor (et al., 2012), which managed to develop a framework for enabling real-time multimedia indexing and search in the Social Web. SocialSensor deployed an open source platform capable of collecting, processing, and aggregating big streams of social media data and multimedia.

As a commercial product, the NewsAsset is in continuous evolution, extending the original suite with new upgrades and functionalities. The NewsAsset needs to be constantly updated for handling efficiently the increasing volume of data. On the other hand, the NewsAsset application must continue satisfying a set of quality standards defined in terms of reliability and efficiency. Therefore, the system needs to adopt runtime scalability methods to manage temporal peaks of high computational demand (i.e., during the peaks of a bust event).

2.1 Objective of the Performance Analysis

The ATC team identified several functional and non-functional quality-driven requirements that are not addressed by the current status of the platform (Reqs., DICE Consortium, 2016). The ultimate goal is to modernize the existing commercial product NewsAsset. The main foreseen challenges are:

- Refactoring of the old-fashioned engine related to cloud processing and Big Data technologies,
- Reconfiguration of the obsolete architecture with respect to quality-driven metrics, and
- Managing the complexity real-time responsiveness for temporal peaks of high computational demand.

The part of the NewsAsset framework that potentially creates a major bottleneck, and therefore it must be redesigned, is the News Orchestrator application. The time behavior of the News Orchestrator application is quite critical since the analyzed information, regarding the trending topics extracted, should be indexed and exposed by the User Interface in real time, enhancing in this way the importance of the identified news topics. We will focus on this part in this work.

The idea is to re-engineer the architecture and introduce Big Data technologies where this is possible. Essentially, the core is replacing the batch processing, that is now the current approach, with stream processing. Then, the logical choice for the Big Data technology to accomplish this purpose is Apache Storm. It would definitely affect the accuracy and processing time of

the system. The goal is to optimize the existing processing time by means of not only minimizing the time slot duration to reflect real time processing but also by maximizing the crawling capacity of the social networks crawler, giving us the potential to collect and analyze as much social networks content as possible.

By identifying and analyzing quality-driven metrics we expect to detect bottlenecks in the architecture and redesign those critical parts by introducing more computational resources and/or adjusting configuration parameters of Apache Storm.

3 A DSML for Apache Storm

In the following, the main concepts of the Storm technology related to performance and reliability are revised. In fact, the Apache Storm framework offers configuration parameters, that when correctly tuned allow one to improve the performance and the reliability of the Storm applications. Consequently, these parameters are essential for the performance or reliability analysis of the Storm applications. Table 1 summarizes all these parameters.

Storm applications are directed acyclic graphs (DAG) with *nodes* and *edges*, see Figure 1. *Nodes* are the points where the information is generated or processed. They can be *spouts*, sources of information that inject *streams* of data at a certain *rate*, or *bolts*, that elaborate input data and produce results which, in turn, are emitted towards other bolts. The *edges* define the transmission of data from one node to another. By default, a Storm application runs indefinitely until killed.

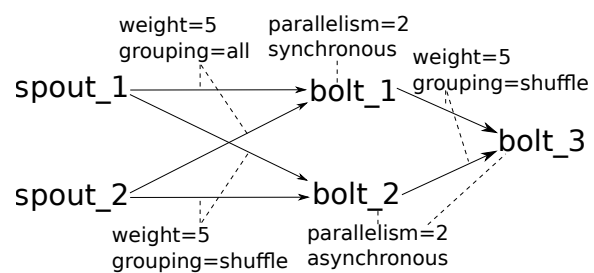


Fig. 1 A Storm Application

The computation of a bolt requires n tuples for producing m results. Such asymmetry is captured by the *weights* in the arcs of the DAG, that represent the number of tuples the next bolt requires for emitting a new message. The notions of *tuples* and *messages* are then equivalent. A bolt can select different *synchronization* policies. When it receives messages from two or more sources, it can 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*). The *parallelism* specifies the number of concurrent threads executing the same task (spout or bolt).

Specifically for the edges. Besides the *weights*, the *grouping* determines the way a message is propagated to and handled by the receiving nodes. By default, a message is broadcast 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* or *global*).

The Storm *Nimbus* is the cluster master node. It statically deploys 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. The Apache *Zookeeper* is a service used by the cluster to monitor, coordinate and recover the computational resources against failures.

Table 1 Storm Concepts for Performance and Reliability

#	Concept	Meaning
1.	<i>Spout</i> (task)	Source of information
2.	<i>Rate</i>	No. of tuples per unit of time produced by a spout
3.	<i>Bolt</i> (task)	Data elaboration
4.	<i>Weight</i>	No. of tuples required by a bolt
5.	<i>Asynchronous policy</i>	The bolt progresses when at least one input tuple is available
6.	<i>Synchronous policy</i>	The bolt progresses when all input tuples are available
7.	<i>Parallelism</i>	No. of concurrent threads per task
8.	<i>Grouping</i>	Tuple propagation policy (e.g., all)
9.	<i>Nimbus</i>	Deployment of tasks
10.	<i>Zookeeper</i>	Service for recovering computational resources against failures

3.1 A UML Profile for Apache Storm

A UML profile is a set of stereotypes, and corresponding tags, that extend the semantics of the UML diagrams when they are applied to the elements of a model. Here we present an extension of the UML profile for Apache Storm introduced in (Requeno et al., 2017), which is the first and only one in the literature devoted to Apache Storm, to the best of our knowledge. This profile extends UML with the Storm concepts identified in Table 1. Then, according to (Selic, 2007) and (Lagarde et al., 2007), we are creating a DSML for specifying the

properties that affect the performance and reliability of the Apache Storm applications designed with UML.

The Storm profile provides the stereotypes collected in Table 2. The attributes of the stereotypes may have multiplicity one (e.g., [1]) or multiple (e.g., [*] for unbounded arrays). For the concepts of *spout* and *bolt* we introduce the `<<StormSpout>>` and `<<StormBolt>>` stereotypes. They share the *parallelism*, i.e., number of concurrent threads executing the task, which is specified by the tag `parallelism`. Spouts also add the tag `avgEmitRate`, which represents the *rate* at which the spout produces tuples.

The concept of *stream* is captured by the `<<StormStreamStep>>` stereotype, which owns three tags:

- `numTuples` matches the *weight* concept. It represents the number of tuples required by the target bolt for emitting a tuple. A value $\$nT$ between 0 and 1 means that the next bolt produces $1/\$nT$ messages per input.
- `grouping` matches the *grouping* concept; and
- `probFields` is a n-dimensional 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 estimated or obtained at runtime experimentally, i.e., by tracing the messages grouped by the bolt thread. The dimension of the array depends on the parallelism of the target bolt.

Stereotypes `<<StormNimbus>>` and `<<StormZookeeper>>` are devoted to reliability analysis, so they are treated in Appendix B. The rest of the stereotypes in Table 2 are described in Section 4 in the context of the modeling.

Profile Inheritance

The standard MARTE³ profile provides a complete framework for quantitative analysis, while the DAM⁴ profile addresses the reliability analysis. In particular, the GQAM sub-profile of MARTE is specialized for performance analysis. These are the reasons why the Apache Storm profile inherits from MARTE and DAM.

The stereotypes for bolts, spouts and streams inherit from `MARTE::GQAM::GaStep`. So, they can be treated as computational steps and the bolts can use the `hostDemand` tag from `GaStep` for defining the task execution time.

MARTE also offers the NFPs and VSL sub-profiles. The NFP sub-profile aims to describe the non-functional

³ Modelling and Analysis of Real Time and Embedded Systems (OMG, 2011a).

⁴ Dependability Modelling and Analysis (Bernardi et al., 2011).

Storm concept	Stereotype	Applied to	Tag	Inheritance/Type
<i>Spout</i>	«StormSpout»	Action/Activity Node		MARTE::GQAM::GaStep
Parallelism Rate			parallelism avgEmitRate	NFP_Integer [1] NFP_Frequency [1]
<i>Bolt</i>	«StormBolt»	Action/Activity Node		MARTE::GQAM::GaStep
Parallelism Execution Time			parallelism hostDemand	NFP_Integer [1] NFP_Duration [1]
<i>Stream</i>	«StormStreamStep»	Control Flow		MARTE::GQAM::GaStep
Weight Grouping			numTuples grouping probFields	NFP_Real [1] {all, shuffle, global, field} [1] NFP_Real [*]
<i>Workstations</i>	«GaExecHost»	Device/Node	resMult	inheritance NFP_Integer [1]
	«GaCommHost»	Device/Node	capacity	inheritance NFP_DataTxRate [1]
<i>Nimbus</i>	«StormNimbus»	Device/Node		
<i>Zookeeper</i>	«StormZookeeper»	Device/Node		

Table 2 Apache Storm Profile

properties of a system, performance and reliability 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 useful for:

(i) specifying the input parameters in the model. An example:

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

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

(ii) specifying the metrics that will be computed for the model. An example:

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

This expression, applied to a resource, specifies that its *utilization* (4) is computed as a percentage of time (2), whose mean value (3) will be assigned to variable $\$util$ (1).

4 NewsAsset Performance Modeling

4.1 System Workflow Description

As explained in Section 2.1, the News Orchestrator is the part of the NewsAsset product that manages large volumes of information. In particular, the topic-detector, within the News Orchestrator, defines the workflow that exposes the system to handle the realtime Big Data information, hence the critical data-intensive computation part of the system. The main purpose of topic-detector is in fact the extraction of trending topics contained in items shared through social networks. By trending topics we refer to frequent features (n-grams, named entities or hashtags) that exhibit an abnormal increase on the current time slot compared to others.

The main workflow of the topic-detector is depicted in Figure 2. It has been refactored, from the current version, as an Apache Storm topology, for the sake of reusing the code pipelined forwardly. The backend of the system consists of several processes running continuously on a 24/7 basis as persistent loops.

In Figure 2 we observe that the input stream consists of two spouts that inject items in the topology: a) one waiting for incoming items from Twitter’s Streaming API, and b) one that listens to a Redis message broker following the Publish/Subscribe pattern. The first bolt in the topology extracts named entities from the text of the messages injected, for this procedure

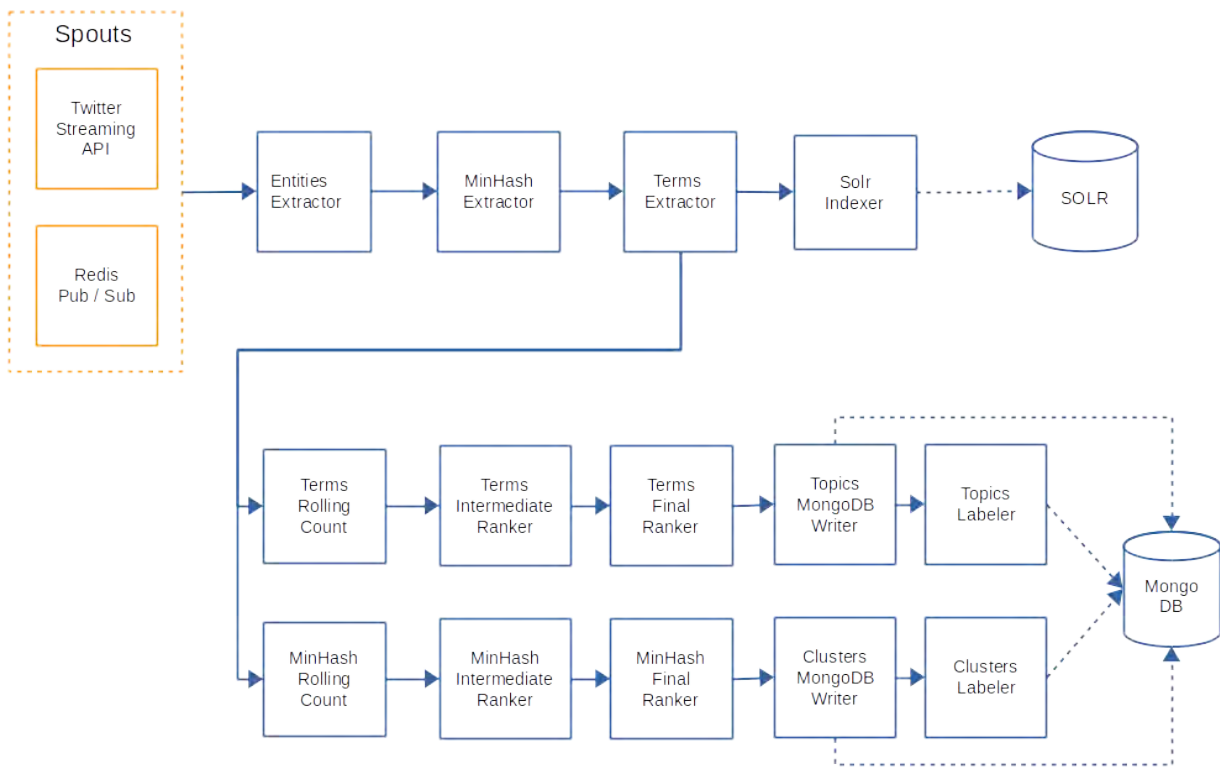


Fig. 2 Workflow of the topic-detector

Stanford NER extractor is used. The next bolt extracts a signature from the text based on the *MinHash* algorithm. Intuitively, items with similar text will have identical signatures. The next bolt extracts terms from the items i.e. n-grams and hashtags. The *Solr indexer* bolt used to index the items in a running instance of Solr database.

At that point, the topology splits into two parallel pipelines: the first one is used to extract trending topics based on trending terms, while the second is used to cluster items based on minhash signature. Both of these pipelines use the same sequence of bolts, but operate on a different field of the items (terms and minhash signature respectively). The first bolt, *TermsRollingCount-Bolt* performs rolling counts of incoming objects (terms or minhash). Rolling means that by using a sliding window the bolt keeps track of statistics of an object to the current time window compared to the previous ones. At the end of each time window, the bolt emits a rolling count tuple per object, consisting of the object itself, its latest rolling count, a metric that indicates how trending is the object and the actual duration of the sliding window (just for testing as the length of the windows is constant).

As each object can be emitted from multiple nodes in a distributed topology, the next two bolts aggregate and rank the objects, in a map-reduce fashion. The final ranking bolt emits a ranking of top objects (terms or minhash) for the current time window. Finally, each of the objects in the rank is stored in a MongoDB instance. The time stamp of the current window is also stored in order to keep track of the evolution of these objects over time.

4.2 UML Performance Design

Once the system workflow has been defined, we need to introduce the performance parameters, that will eventually enable a performance analysis of the system. For this reason, we defined in Section 3 a DSML for Apache Storm applications. In particular, we need to model: i) the Storm topology, i.e., the DAG; ii) the performance parameters, and iii) the deployment of the system, where the resources will be defined.

i) *Modeling the Storm topology*

For this purpose our approach proposes the UML activity diagram, which is interpreted as a DAG. It ba-

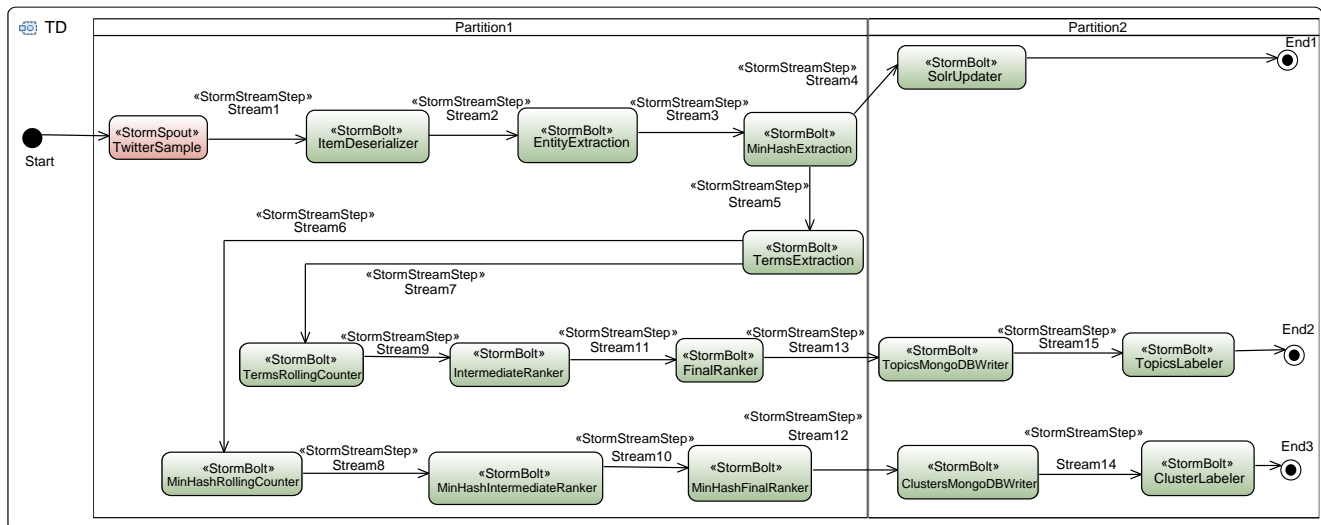


Fig. 3 Activity diagram for the topic-detector workflow, with profile annotations

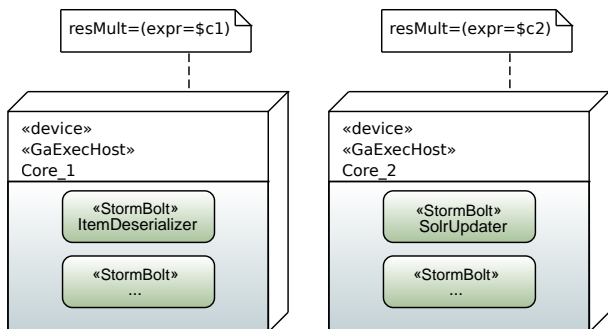


Fig. 4 Deployment diagram

sically mimics the workflow of the system, although the semantics we adopt is slightly different from the standard one of the UML activity diagram. In our approach, a UML activity diagram will always start with a set of initial nodes connected to the spout tasks since 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. 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). Figure 3 depicts the model for the topics-detector.

ii) Modeling the performance parameters

They represent input parameters and the metrics to compute and obviously are introduced in the model using the stereotypes and tags already defined in Table 2. In Figure 3, we observe that the activities and transitions have been stereotyped accordingly to their semantics. The tags that detail these stereotypes, i.e., the input parameters appear in Tables 3 and 4.

Table 3 introduces the variables associated to the tags of the spouts and bolts stereotypes in the UML activity diagram. The first column includes the level of parallelism (number of threads) for each spout (bolt) of the system. The second column shows the expected average execution time, in milliseconds, required by a spout (bolt) for creating (processing) a tuple. The total execution time depends on the number of input tuples needed for emitting an output.

Table 4, shows the grouping policies, weights and probabilities associated to the communication streams linking spouts and bolts. The weight corresponds to the number of messages consumed by the following bolt in order to generate a new message. The probabilities are associated to the *probFields* tag. The size of the array of probabilities depends on the parallelism of the target bolt (i.e., it shows the probability of receiving a tuple by the *i*-th thread).

iii) Deployment modeling

The bolts of the activity diagram are mapped to the computational resources where they will execute upon deployment. This modeling is of great importance since resources may turn into system bottlenecks. Figure 4 shows the deployment diagram, which complements the previous activity diagram. Each computational resource is stereotyped as *GaExecHost*, inherited from MARTE, and defines its resource multiplicity, i.e., number of cores. Table 5 defines variables for specifying the number of available cores in each workstation of the cluster.

Initializing the values of the model parameters The variables already defined in the Tables 3, 4 and 5, i.e., tags

of the stereotypes, are initialized according to the actual parameters of the running Storm application. The *parallelism* and *grouping* parameters are statically determined by the configuration of the Storm application when the DAG is defined for execution. Part of the *weight* parameters are also specified statically, while the rest has been dynamically extracted by monitoring. In most of the cases, the bolts need a single incoming tuple. The average execution *times* are also extracted from the Storm web monitoring platform. The values of the *probFelds* array are set according to observed distribution of the messages in the tasks receiving a *field* stream. Section 5 presents the actual configurations of the NewsAsset used for analysis purposes.

4.3 Complex Storm Applications

Naturally, more complex Storm topologies – such as diamonds or stars – can also be modeled using our approach. However, the topology is not enough when we want to represent complex behavior for tasks (i.e., spouts and bolts). So far we assumed a task to be a black box and, then, modeled by an action of the activity diagram. Nevertheless, UML offers nesting modeling capabilities of activity nodes, i.e., modeling the internals of an activity with another UML activity diagram. Consequently, we use this UML feature to allow the designer to model in detail the subset of tasks which are of interest for performance evaluation. These Storm tasks are just java programs that can be modeled with the usual interpretation of the activity diagram (e.g., sub-activities, choices, synchronizations, etc.). Moreover, this specification of the spout or bolt can be annotated using MARTE then gaining a detailed performance specification.

5 Experimentations

The UML design, presented in Subsection 4.2, has been first calibrated and validated using monitored data from the deployed Storm topic-detector application (Subsection 5.1). Then, simulation experiments have been conducted with the validated model to predict the scalability of the application (Subsection 5.2).

The experiments in this section have been carried out using the performance model in Figure 11. Appendix A details how an Apache UML design is transformed in a performance model.

5.1 Model Calibration and Validation

To calibrate and validate the UML design we have deployed a prototype of the Storm application under different workload assumptions, where the emission rate of the spouts ranges in the interval $[10 - 80]$ tuples/sec.. We then monitored performance indexes (i.e., bolt utilizations, total number of tuples generated by the spout and produced by the topic-detector) for a total period of two hours per scenario.

Each scenario has been launched and monitored three times for removing environmental noise. The monitoring has been carried out by a Linux script shell that periodically polls the Storm Web UI through the REST API. The execution environment where the NewsAsset application was launched consists of a set of four workstations for running the computations and basic Storm services (e.g., Nimbus, Zookeeper) and a single workstation for hosting the databases (MongoDB and Solr). The workstations dedicated to running the computations are virtual AMD CPUs (4x2.60GHz) with 4GBytes of RAM. The workstation dedicated to hosting the databases is a virtual AMD CPU (2x2.60GHz) with 4GBytes of RAM. All workstations are deployed on the Flexiant cloud (Flexiant, 2017) and they have a Gigabit ethernet and Ubuntu Linux (version 14.04) OS.

Independent sets of monitored data have been used to calibrate and validate the model. In the calibration phase, we have set the values of the performance input parameters that were not explicitly provided in the configuration of NewsAsset, i.e., the host demands of the bolts (Table 6, third column), the probabilities for the field grouping policy streams and the number of input tuples of some of the bolts (Table 7, items in bold font).

In particular, from the monitored data, we observed that some input parameters were linearly dependent on the spout emission rate. This is the case of the input streams of the two bolts *TermsRollingCounter* and *MinHashRollingCounter* (Figure 5), due to an ad-hoc implementation of a time window which enables to collect the tuples generated by the *TermsExtraction* bolt and send them in bunches to the mentioned bolts.

The host demands of the bolts accessing the databases are also linearly depend on the emission rate of the spout, as shown in Figure 6. The database writing scales linearly with respect to the emission rate of the spout because of the increasing size of the sliding window that accumulates tuples in bolts *TermsRollingCounter* and *MinHashRollingCounter*.

The calibrated model has been validated using the *paired-t* approach in Averill (2015), based on the computation of the confidence interval for the mean of the

Table 3 Configuration of spouts and bolts

	parallelism	Expected execution time (ms)
TwitterSample	\$pTwitterSample	\$tTwitterSample
ItemDeserializer	\$pItemDeserializer	\$tItemDeserializer
EntityExtraction	\$pEntityExtraction	\$tEntityExtraction
MinhashExtraction	\$pMinHashExtraction	\$tMinHashExtraction
SolrUpdater	\$pSolrUpdater	\$tSolrUpdater
TermsExtraction	\$pTermsExtraction	\$tTermsExtraction
TermsRollingCounter	\$pTermsRollingCounter	\$tTermsRollingCounter
IntermediateRanker	\$pIntermediateRanker	\$tIntermediateRanker
FinalRanker	\$pFinalRanker	\$tFinalRanker
TopicsMongoDBWriter	\$pTopicsMongoDBWriter	\$tTopicsMongoDBWriter
TopicsLabeler	\$pTopicsLabeler	\$tTopicsLabeler
MinhashRollingCounter	\$pMinHashRollingCounter	\$tMinHashRollingCounter
MinhashIntermediateRanker	\$pMinHashIntermediateRanker	\$tMinHashIntermediateRanker
MinHashFinalRanker	\$pMinHashFinalRanker	\$tMinHashFinalRanker
ClustersMongoDBWriter	\$pClustersMongoDBWriter	\$tClustersMongoDBWriter
ClustersLabeler	\$pClustersLabeler	\$tClustersLabeler

Table 4 Configuration of streams

Connection from	to	Stream_Id	grouping	numTuples	ProbFields (max 1.0)
TwitterSample	ItemDeserializer	Stream1	shuffle	\$ntStream1	
ItemDeserializer	EntityExtraction	Stream2	shuffle	\$ntStream2	
EntityExtraction	MinHashExtraction	Stream3	shuffle	\$ntStream3	
MinHashExtraction	SolrUpdater	Stream4	shuffle	\$ntStream4	
MinHashExtraction	TermsExtraction	Stream5	shuffle	\$ntStream5	
TermsExtraction	MinHashRollingCounter	Stream6	field	\$ntStream6	[\$prStream6_1, ..., \$prStream6_n]
TermsExtraction	TermsRollingCounter	Stream7	field	\$ntStream7	[\$prStream7_1, ..., \$prStream7_n]
MinHashRollingCounter	MinHashIntermediateRanker	Stream8	field	\$ntStream8	[\$prStream8_1, ..., \$prStream8_n]
TermsRollingCounter	IntermediateRanker	Stream9	field	\$ntStream9	[\$prStream9_1, ..., \$prStream9_n]
MinHashIntermediateRanker	MinHashFinalRanker	Stream10	global	\$ntStream10	
IntermediateRanker	FinalRanker	Stream11	global	\$ntStream11	
MinHashFinalRanker	ClustersMongoDBWriter	Stream12	shuffle	\$ntStream12	
FinalRanker	TopicsMongoDBWriter	Stream13	shuffle	\$ntStream13	
ClustersMongoDBWriter	ClustersLabeler	Stream14	shuffle	\$ntStream14	
TopicsMongoDBWriter	TopicsLabeler	Stream15	shuffle	\$ntStream15	

Table 5 Configuration of Workstations

Workstation	resMult (nCores)
Core_1	\$c1
Core_2	\$c2

differences between two sets of N observations. In particular, for each performance index of interest, we have collected two sets of independent observations: one set X includes the values of the performance index monitored by running the deployed application under different workload assumption (within the interval range [10 – 80] tuples/sec.); and the other set Y includes the values of the performance index obtained via simulation

of the calibrated model under the same workload assumptions. Then, for each performance index, we have constructed a 95% confidence interval for the sample mean of the difference $\bar{\mu} = \bar{\mu}_X - \bar{\mu}_Y$ with the aim of evaluating its statistical and practical significance.

Table 8 summarizes the validation results for the performance indexes of interest (one per column). Concretely, the utilization of the bolts belonging to the first

	parallelism	emit rate (tuples/s)	host demand (ms)
TwitterSample	1	$x \in [10 - 80]$	
ItemDeserializer	4		0.665
EntityExtraction	4		21.683
MinhashExtraction	4		13.048
SolrUpdater	1		16.895
TermsExtraction	2		10.441
TermsRollingCounter	4		10.127
IntermediateRanker	4		10.731
FinalRanker	1		16.308
TopicsMongoDBWriter	1		$11.5x + 24.8$
TopicsLabeler	1		$4.9x + 82.2$
MinhashRollingCounter	4		10.212
MinhashIntermediateRanker	1		10.219
MinHashFinalRanker	1		19.600
ClustersMongoDBWriter	1		$7.5x + 76.6$
ClustersLabeler	1		$5.6x + 66.7$

Table 6 Configuration of Spout and Bolt Elements: parallelism, emit rate/host demand

Connection from	to	Stream.Id	grouping	numTuples	ProbFields ($\sum_i p_i = 1$)
TwitterSample	ItemDeserializer	Stream1	shuffle	1	
ItemDeserializer	EntityExtraction	Stream2	shuffle	1	
EntityExtraction	MinHashExtraction	Stream3	shuffle	1	
MinHashExtraction	SolrUpdater	Stream4	shuffle	10	
MinHashExtraction	TermsExtraction	Stream5	shuffle	0.2857	
TermsExtraction	MinHashRollingCounter	Stream6	field	4x + 3	[0.16, 0.24, 0.27, 0.33]
TermsExtraction	TermsRollingCounter	Stream7	field	9x + 1	[0.12, 0.12, 0.27, 0.49]
MinHashRollingCounter	MinHashIntermediateRanker	Stream8	field	59	[1.0]
TermsRollingCounter	IntermediateRanker	Stream9	field	27	[0.18, 0.26, 0.27, 0.29]
MinHashIntermediateRanker	MinHashFinalRanker	Stream10	global	1	
IntermediateRanker	FinalRanker	Stream11	global	3	
MinHashFinalRanker	ClustersMongoDBWriter	Stream12	shuffle	0.06	
FinalRanker	TopicsMongoDBWriter	Stream13	shuffle	0.06	
ClustersMongoDBWriter	ClustersLabeler	Stream14	shuffle	1	
TopicsMongoDBWriter	TopicsLabeler	Stream15	shuffle	1	

Table 7 Configuration of Streams: number of tuples and probability fields

	Utilization					Throughput ratio ($X_{End_i}/X_{TwitterSample}$)		
	Item Deserializer	Entity Extraction	MinHash Extraction	Solr Updater	Terms Extraction	End1	End2	End3
$\bar{\mu}$	0.0906	-0.3378	-0.2826	-1.2475	-1.3607	-2.553E-3	-1.523E-4	-2.258E-4
$\bar{\sigma}$	0.1750	1.1474	1.4331	15.5080	1.3761	3.104E-2	1.946E-4	1.899E-4
lb	0.0130	-0.8467	-0.9181	-8.1246	-1.9709	-1.632E-2	-2.386E-4	-3.100E-4
ub	0.1682	0.1710	0.3529	5.6296	-0.7504	1.121E-2	-6.604E-5	-1.416E-4
ϵ (%)	19.93	3.50	6.34	11.57	8.84	1.67	12.27	16.41

Table 8 Validation of the UML model

part of the workflow, and the ratio of the tuples produced by each branch of the *News Orchestrator* workflow with respect to the input stream generated by the spout *TwitterSample*. In the UML model, the latter indexes correspond to the throughput ratios $\frac{X_{End_i}}{X_{TwitterSample}}$.

The rows of the table indicate: the difference sample mean $\bar{\mu}$ (first row), the sample standard deviation of the difference $\bar{\sigma}$ (second row), the lower and up-

per bounds of the 95% confidence interval⁵ (third and fourth rows, respectively), and the maximum relative error $\max(\frac{lb}{\bar{\mu}_X}, \frac{ub}{\bar{\mu}_X})$, where $\bar{\mu}_X$ is the sample mean of the index computed from the monitored data (fifth row).

From the validation results, we observe that all the sample means, but the one related to the utilization of the *ItemDeserializer*, are negative meaning that the UML model produces an underestimation of the con-

⁵ The Student t-distribution with N-1=21 degrees of freedom has been used.

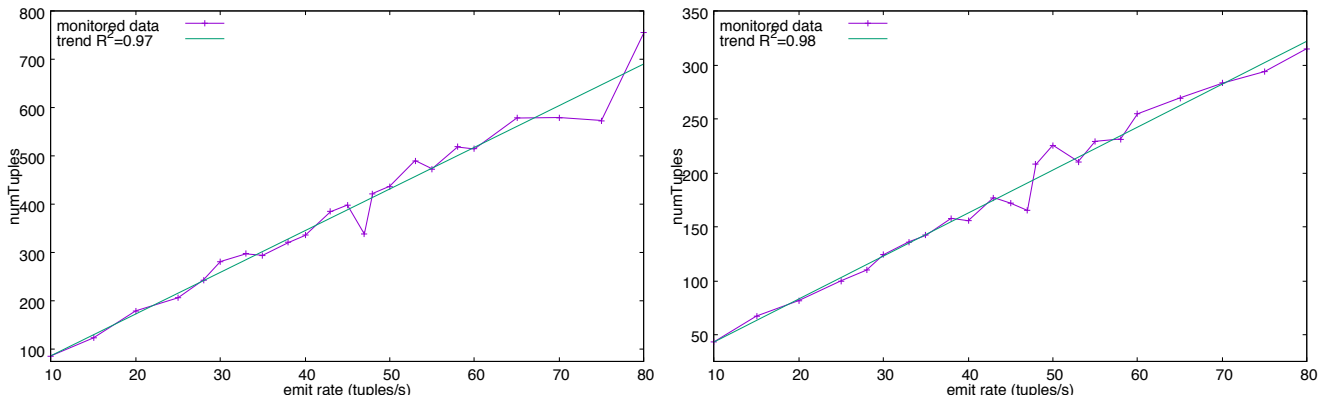


Fig. 5 Input streams: *TermsRollingCounter* (left), *MinHashRollingCounter* (right).

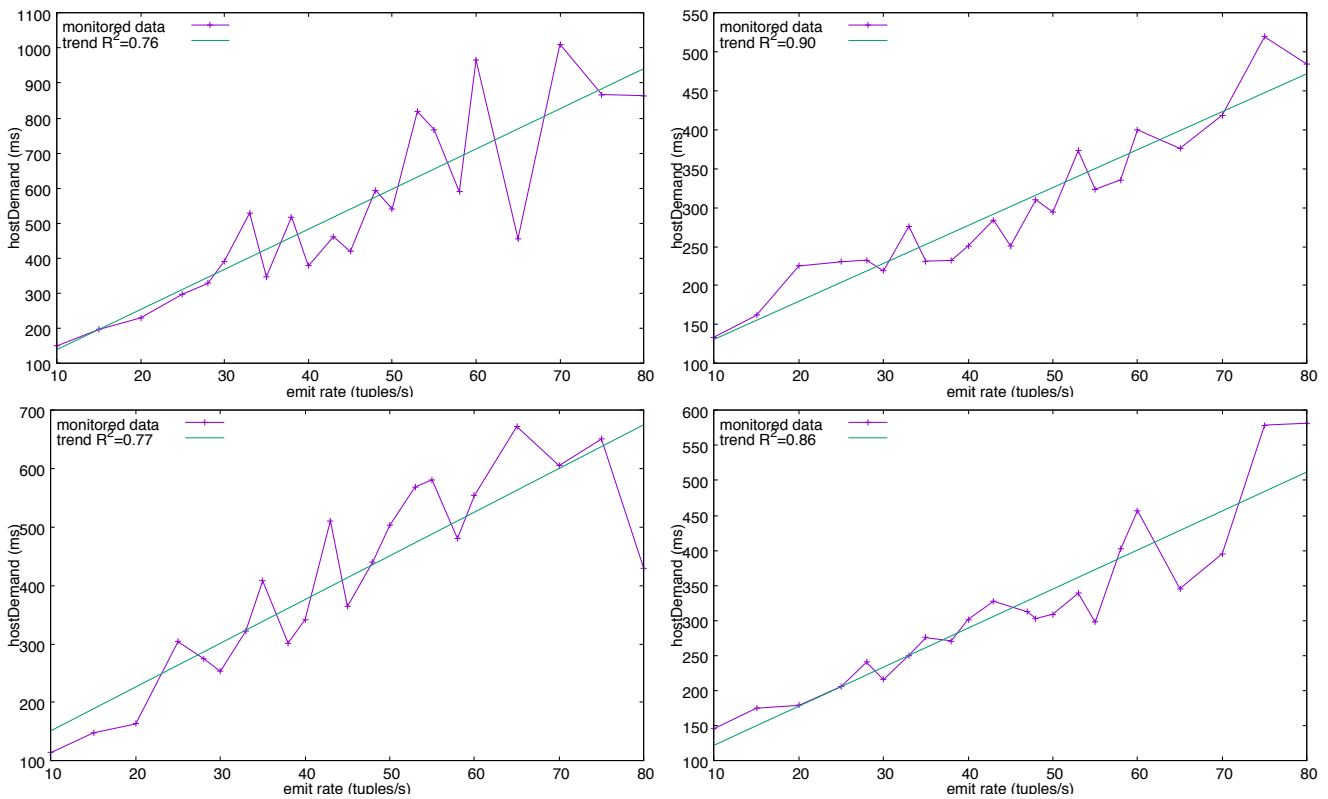


Fig. 6 Host Demand: *TopicsMongoDB* (top-left), *TopicsLabeler* (top-right), *ClusterMongoDB* (bottom-left), *ClusterLabeler* (bottom-right).

sidered performance indexes with respect to the deployed application. However, analyzing the confidence intervals, we can infer that the differences between the monitored values and the simulated ones for the utilization of the *EntityExtraction*, *MinHashExtraction*, *SolrUpdater* and the throughput ratio of the first branch of the workflow (*End1*) might be caused by sampling fluctuation and they are not statistically significant (i.e., $0 \in [lb, ub]$). Although the differences for the rest of the performance indexes could be statistically significant

(i.e., $0 \notin [lb, ub]$), they are not considered significant in practice by the domain experts.

5.2 Performance Predictions

The UML model, calibrated and validated under different workload assumptions of the NewsAsset, has been used to assess the performance requirements identified by ATC (Reqs., DICE Consortium, 2016):

- Scalability of the current topology. The software architects are interested in figuring out whether the

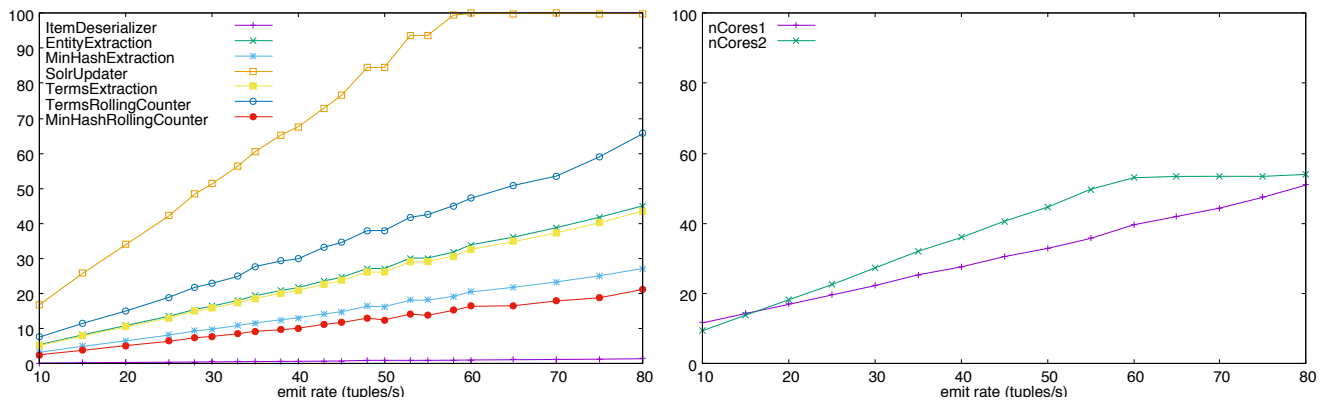


Fig. 7 Utilization (%) of bolts on the left, and utilization (%) of cores on the right.

system is linearly scalable and it scales out rather than up, meaning that performance problems can be solved by simply adding more cores. This goal implies the detection of system bottlenecks in order to identify the type of resources that need to be increased.

- Performance of alternative architectures. The software quality engineers are interested in measuring the impact of different architecture alternatives based on performance and cost.

In the following paragraphs, we describe the performance experiments (parameter configuration and measures to compute) to assess the performance goals.

5.2.1 Scalability of the current topology

To assess the first performance goal, we have conducted a bottleneck analysis considering both the hardware (i.e., the cluster CPUs) and software resources (i.e., bolts). During the calibration and validation of the UML model for different spout emission rates, we already detected a potential bottleneck in the *SolrUpdater* bolt. In particular, the *SolrUpdater* becomes a bottleneck when the emission rate is higher than 60 tuples/sec. (see Figure 7 on the left), whereas the utilization⁶ of the other bolts is lower than 60%. Besides, the utilization of the cluster CPU's is always less than 53% for the workstations hosting the spout and bolts; and less than 55% for the workstation hosting the databases (see Figure 7 on the right).

Therefore, unlike the expectation of the software architects, the design does not scale out, since the bottleneck is not a hardware resource. Figure 8 confirms that the increase in the number of cores (workstation *Core_2*) does not improve the system performance. In particular, for a given emission rate, the utilization of

⁶ Figure 7 does not show the utilizations of the bolts that are below 5%.

the *SolrUpdater* bolt remains constant. Indeed, the limiting factor for the maximum production rate of the topic-detector is the parallelism of the bolt *SolrUpdater* (software bottleneck). Therefore, we initially recommended to ATC the redesign of the algorithms of this bolt module to achieve the performance objective, that did not turn out to be an easy task.

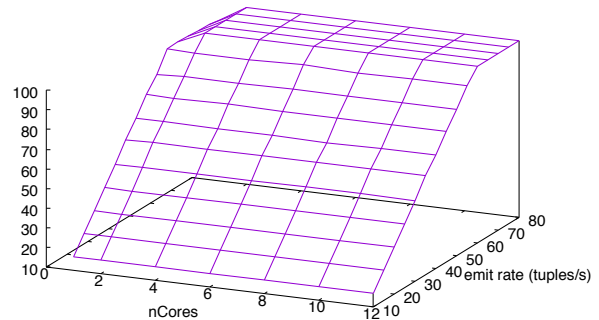


Fig. 8 Utilization (%) of *SolrUpdater*, with one thread, under different number of cores and workload.

5.2.2 Performance of alternative architectures

The objective of the second experiment is to evaluate the impact of different architecture alternatives of the News Orchestrator, based on the Apache Storm performance features. In particular, we aimed at reaching a practical solution for the bottleneck detected in the bolt *SolrUpdater*.

For this purpose, we considered the basic configuration of the UML model used in the calibration and validation phases (Tables 6 and 7). In order to analyze the sensitivity of the performance indices to the Storm configuration, the varying parameters are the level of parallelism of the *SolrUpdater* ([1–10] threads) and the

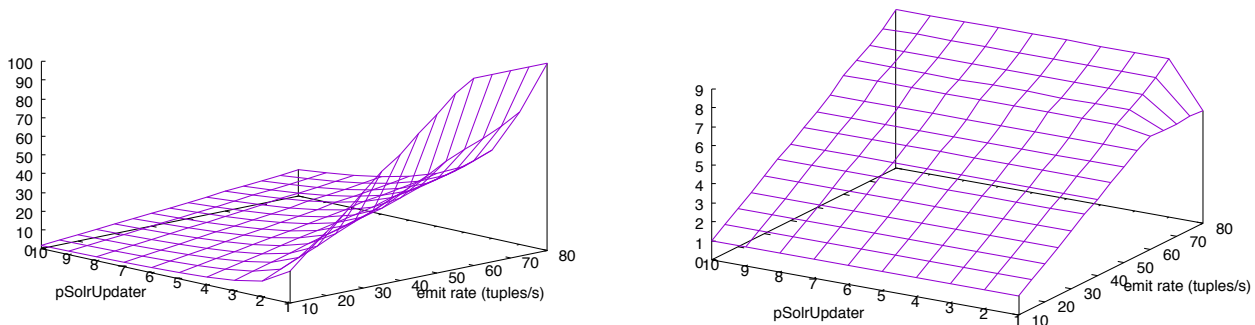


Fig. 9 Utilization (%) of *SolrUpdater* on the left, and throughput (tuples/ms) of *SolrUpdater* on the right.

emission rate ($[10 - 80]$ tuples/s). Figure 9 shows the utilization (left plot) and the throughput (right plot) of the *SolrUpdater*. In particular, considering high spout emission rates (that is, an emission rate greater than 60 tuples/s), we can observe that the utilization of the *SolrUpdater* is reduced by at least a 30% when just two threads are set for the bolt and it is drastically reduced by a 90% for 10 threads. On the other hand, two threads for the bolt are sufficient to increase the throughput of the first branch of the topic-detector workflow and to avoid the system saturation under the considered workload assumption (for more than one thread, the throughput increases linearly with respect to the increase of the emission rate).

Therefore, the ATC software quality engineers should consider this alternative configuration to guarantee an optimal throughput under high workloads.

For the sake of offering alternative choices to the ATC engineers, we carried out additional experiments. A similar sensitivity analysis was conducted considering the second potential bottleneck for high workloads identified in the first experiment, that is, the bolt *TermsRollingCounter* (see Figure 7, left plot). The goal was to figure out how to change the Storm configuration in order to improve also the throughput of the second branch of the topic-detector workflow, i.e., the throughput of *End2* in the activity diagram in Figure 3. In this case, the level of parallelism of *TermsRollingCounter* ranged from 4 to 12 threads (in the basic configuration, the level of parallelism of the bolt was set to 4). Figure 10 shows the utilization (left plot) of the *TermsRollingCounter* and the throughput of *End2* (right plot). For high workloads (i.e., emission rate greater than 60 tuples/sec.), the utilization of the bolt is reduced by a half when the number of threads is doubled and it is reduced by two-thirds when the number of threads is tripled. Nevertheless, as shown by the plot on the right, the increase of the level of parallelism of this

bolt does not improve the throughput of *End2*, which remains basically constant. Thus, increasing the level of parallelism of the *TermsRollingCounter* is not a good solution if the aim is increasing the throughput of *End2*.

6 Conclusion

The work in (Requeno et al., 2017) presents, and validates synthetically, an approach for the performance analysis of Apache Storm applications. Now, we have extended the approach for addressing also the reliability analysis in the same context. Therefore, this work has presented a quality-driven methodology, for Apache Storm applications, that can be used by practitioners as a blueprint for development. In our view, an important contribution is that we have applied the methodology to a commercial product, working together with the engineers of the Athens Technological Center (ATC), which means a real validation of the methodology.

ATC needed to refactor the NewsAsset application in order to satisfy highly demanding performance requirements, in the context of processing large volumes of data in realtime from the social network. The results of the performance assessment helped to modernize, redesign and correctly tune the core of NewsAsset using Apache Storm. Several issues, bottlenecks and bad configurations were identified, but not initially supposed, some of them have been reported in this paper.

We also consider important that the methodology is currently usable by practitioners since we have developed a complete framework that automates it, the DICE Simulation tool (DICE Consortium, 2017). This tool includes the Apache Storm profile, the transformation of the Apache Storm designs into GSPN and the computation of performance metrics (utilization, throughput and response time) and reliability metrics (availability and MTTF).

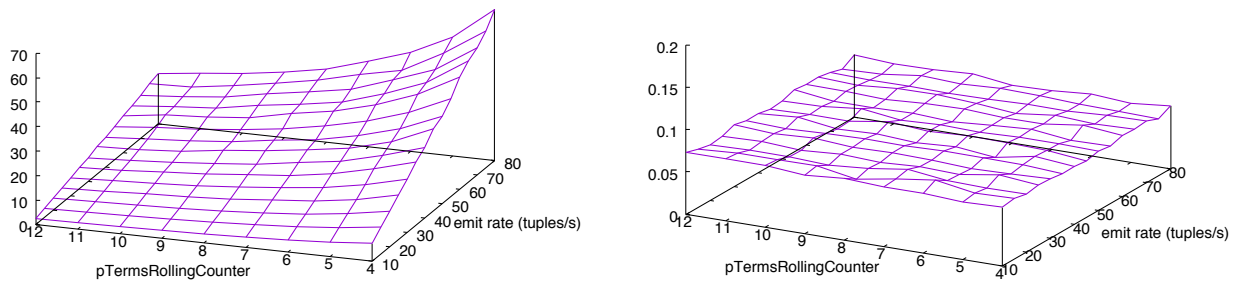


Fig. 10 Utilization (%) of *TermsRollingCounter* on the left, and throughput (tuples/ms) of *End2* on the right.

References

- Apache. Apache Storm Website, 2017a. URL <http://storm.apache.org/>.
- Rajiv Ranjan. Modeling and Simulation in Performance Optimization of Big Data Processing Frameworks. *IEEE Cloud Computing*, 1(4):14–19, 2014.
- ATC. Athens Technology Center Website, 2018. URL <https://www.atc.gr/default.aspx?page=home>.
- José Ignacio Requeno, José Merseguer, and Simona Bernardi. Performance Analysis of Apache Storm Applications using Stochastic Petri Nets. In *Proceedings of the 5th International Workshop on Formal Methods Integration*, 2017.
- Rekha Singhal and Abhishek Verma. Predicting Job Completion Time in Heterogeneous MapReduce Environments. In *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium Workshops*, pages 17–27. IEEE, 2016.
- Filip Nalepa, Michal Batko, and Pavel Zezula. Model for Performance Analysis of Distributed Stream Processing Applications. In *Proceedings of the 20th International Conference on Database and Expert Systems Applications*, pages 520–533. Springer, 2015a.
- Filip Nalepa, Michal Batko, and Pavel Zezula. Performance Analysis of Distributed Stream Processing Applications Through Colored Petri Nets. In *Proceedings of the 10th International Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pages 93–106. Springer, 2015b.
- Johannes Kroß, Andreas Brunnert, and Helmut Krcmar. Modeling Big Data Systems by Extending the Palladio Component Model. *Softwaretechnik-Trends*, 35(3), 2015.
- Johannes Kroß and Helmut Krcmar. Modeling and Simulating Apache Spark Streaming Applications. *Softwaretechnik-Trends*, 36(4), 2016.
- Kewen Wang and Mohammad Maifi Hasan Khan. Performance prediction for Apache Spark platform. In *2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), and 2015 IEEE 12th International Conference on Embedded Software and Systems (ICSS)*, pages 166–173. IEEE, 2015.
- Slawomir Samolej and Tomasz Rak. Simulation and Performance Analysis of Distributed Internet Systems using TCPNs. *Informatika*, 33(4), 2009.
- Tomasz Rak. Response Time Analysis of Distributed Web Systems using QPNs. *Mathematical Problems in Engineering*, 2015, 2015.
- Giovanni Chiola, Marco Ajmone Marsan, Gianfranco Balbo, and Gianni Conte. Generalized Stochastic Petri nets: A Definition at the Net Level and its Implications. *IEEE Transactions on Software Engineering*, 19(2):89–107, 1993.
- Danilo Ardagna et al. Modeling Performance of Hadoop Applications: A Journey from Queueing Networks to Stochastic Well Formed Nets. In Jesus Carretero et al., editor, *Proceedings of the 16th Int. Conf. on Algorithms and Architectures for Parallel Processing*, pages 599–613, Cham, 2016. Springer. ISBN 978-3-319-49583-5.
- Eugenio Gianniti et al. Fluid Petri Nets for the Performance Evaluation of MapReduce and Spark Applications. *ACM SIGMETRICS Performance Evaluation Review*, 44(4):23–36, 2017.
- Sotiris Diplaris et al. SocialSensor: sensing user generated input for improved media discovery and experience. In *Proceedings of the 21st International Conference on World Wide Web*, pages 243–246. ACM, 2012.
- DICE Consortium. Requirement Specification. Technical report, European Union’s Horizon 2020

- research and innovation program, 2016. URL <http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2015/08/D1.2.Requirement-specification.pdf>.
- Bran Selic. 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*, pages 2–9. IEEE Computer Society, May 2007.
- François Lagarde, Huáscar Espinoza, François Terrier, and Sébastien Gérard. Improving UML profile design practices by leveraging conceptual domain models. In R. E. Kurt Stirewalt, Alexander Egyed, and Bernd Fischer, editors, *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), Atlanta (USA)*, pages 445–448. ACM, November 2007.
- OMG. UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems, Version 1.1, Juny 2011a. URL <http://www.omg.org/spec/MARTE/1.1/>.
- Simona Bernardi, José Merseguer, and Dorina C. Petriu. A Dependability Profile within MARTE. *Software & Systems Modeling*, 10(3):313–336, 2011.
- Flexiant. Flexiant Cloud Orchestator Website, 2017. URL <https://www.flexiant.com/>.
- M. Law Averill. *Simulation Modeling and Analysis*. McGraw-Hill, 2015.
- DICE Consortium. DICE Simulation Tool, 2017. [https://github.com/dice-project/DICE - Simulation/](https://github.com/dice-project/DICE-Simulation/).
- DICE Consortium. Storm profile, Oct., 2016. URL: <https://github.com/dice-project/DICE-Profiles>.
- OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1, January 2011b. URL <http://www.omg.org/spec/QVT/1.1/>.
- ISO. Systems and software engineering – High-level Petri nets – Part 2: Transfer format. ISO/IEC 15909-2:2011, International Organization for Standardization, Geneva, Switzerland, 2008.
- Dipartimento di informatica, Università di Torino. GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets, Dec., 2015. URL: <http://www.di.unito.it/~greatspn/index.html>.
- Armin Zimmermann. *Modelling and Performance Evaluation with TimeNET 4.4*, pages 300–303. Springer International Publishing, Cham, 2017.
- Apache. Apache Zookeeper Website, 2017b. URL <https://zookeeper.apache.org/>.
- Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.*, 1(1): 11–33, 2004. doi: 10.1109/TDSC.2004.2. URL <https://doi.org/10.1109/TDSC.2004.2>.
- Marco Ajmone Marsan, G. Balbo, Gianni Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri nets*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

A Transformation of a UML Design to a Performance Model

For evaluating performance metrics (utilization, throughput and response time), we need to transform the UML Apache Storm design into a performance model. We choose as target performance model Generalized Stochastic Petri Nets (see Appendix C). In the following, we recall the *transformation patterns* proposed in Requeno et al. (2017). Each pattern takes as input a part of the Storm design and produces a GSPN subnet.

A.1 Activity Diagram Transformation

Tables 9–11 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 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 *num-Tuples* and *grouping* tagged-values of a given stream step. Therefore, these patterns refine patterns $P3$, $P4$ and $P5$. The

Table 9 Transformation Patterns for Storm-profiled UML Activity Diagrams (Node Elements)

	UML PATTERN	PETRI NET PATTERN
P1		
P2		
P3		
P4		
P5		
P6		

$numTuples$ 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 $\$prob_i$ assigned to the immediate transitions t_{B1_i} .

A.2 Deployment Diagram Transformation

Pattern $P7$ (Table 11) 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.

A.3 Performance Model and Implementation

Figure 11 shows the final GSPN model for the Storm design in Figures 3 and 4. 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. For instance, the input (output) arcs of the transitions that acquire (release) tokens from (to) the resource places *Core_1* and *Core_2* are shown as broken arcs.

The UML Profile for Apache Storm can be downloaded from DICE Consortium (Oct., 2016). The transformation of

Table 10 Transformation Patterns for Storm-profiled UML Activity Diagrams (Stream Policies)

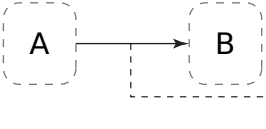
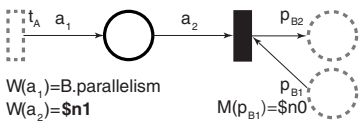
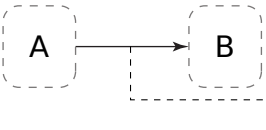
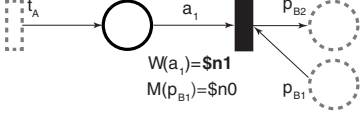
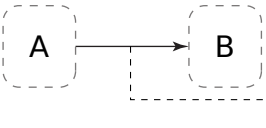
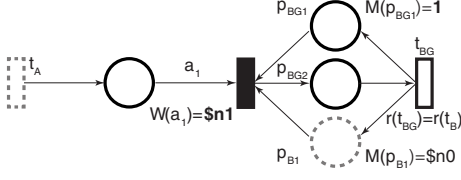
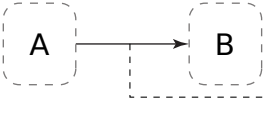
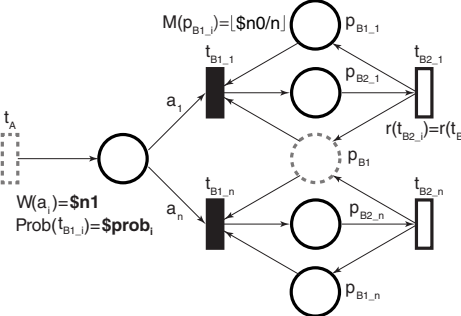
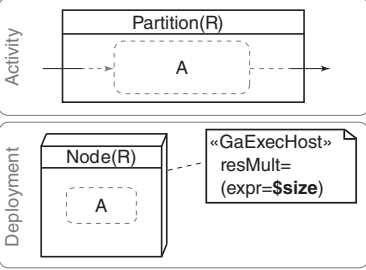
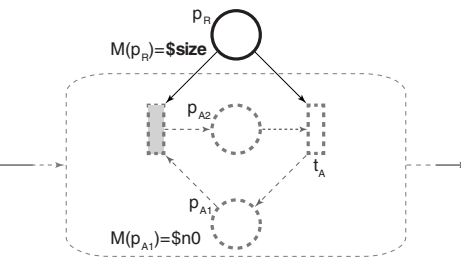
	UML PATTERN	PETRI NET PATTERN
P7	 <pre> «StormStreamStep» numTuples=\$n1 grouping=all </pre>	 <p> $W(a_1)=B.parallelism$ $W(a_2)=\$n1$ $M(p_{B1})=\$n0$ </p>
P8	 <pre> «StormStreamStep» numTuples=\$n1 grouping=shuffle </pre>	 <p> $W(a_1)=\$n1$ $M(p_{B1})=\$n0$ </p>
P9	 <pre> «StormStreamStep» numTuples=\$n1 grouping=global </pre>	 <p> $W(a_1)=\$n1$ $M(p_{BG1})=1$ $M(p_{B1})=\$n0$ $r(t_{BG})=r(t_B)$ </p>
P10	 <pre> «StormStreamStep» numTuples=\$n1 grouping=field probFields=[\$prob_1,...,\$prob_n] </pre>	 <p> $W(a_1)=\$n1$ $Prob(t_{B1,i})=\$prob_i$ $M(p_{B1,i})= n0/n$ $r(t_{B2,i})=r(t_{B1,i})$ </p>

Table 11 Transformation Patterns for Storm-profiled UML Deployment Diagrams

	UML PATTERN	PETRI NET PATTERN
P11	 <pre> «GaExecHost» resMult= (expr=\$size) </pre>	 <p> $M(p_R)=\$size$ $M(p_{A1})=\$n0$ </p>

the UML models to GSPN, as well as the evaluation of the performance metrics, have been automatized in the DICE Simulation tool (DICE Consortium, 2017), they are publicly available. The transformation uses QVT-MOF 2.0 OMG (2011b) to obtain a Petri Net Markup Language file (PNML) ISO (2008), an ISO standard for XML-based interchange format for Petri nets. Later on, a model-to-text (M2T) transformation from PNML into a GSPN tool specific format, concretely for the GreatSPN tool (Dipartimento di informatica, Università di Torino, Dec., 2015), has been performed. Other tools, such as TimeNet (Zimmermann, 2017), also could be used, although not integrated in the framework yet.

B Computation of Reliability Metrics

The DICE Simulation tool (DICE Consortium, 2017), used in this work, also computes reliability metrics. In particular, the availability and mean time to failure for UML Apache Storm designs stereotyped using the Apache Storm profile presented in Section 3.

Many of the technologies covered for data-intensive applications (DIAs), such as Apache Storm, are designed as fault-tolerant, which means that their failures are internally handled and are not visible to users. Therefore, the metrics are calculated from the properties of the resources used by the technologies, rather than from the activities executed by

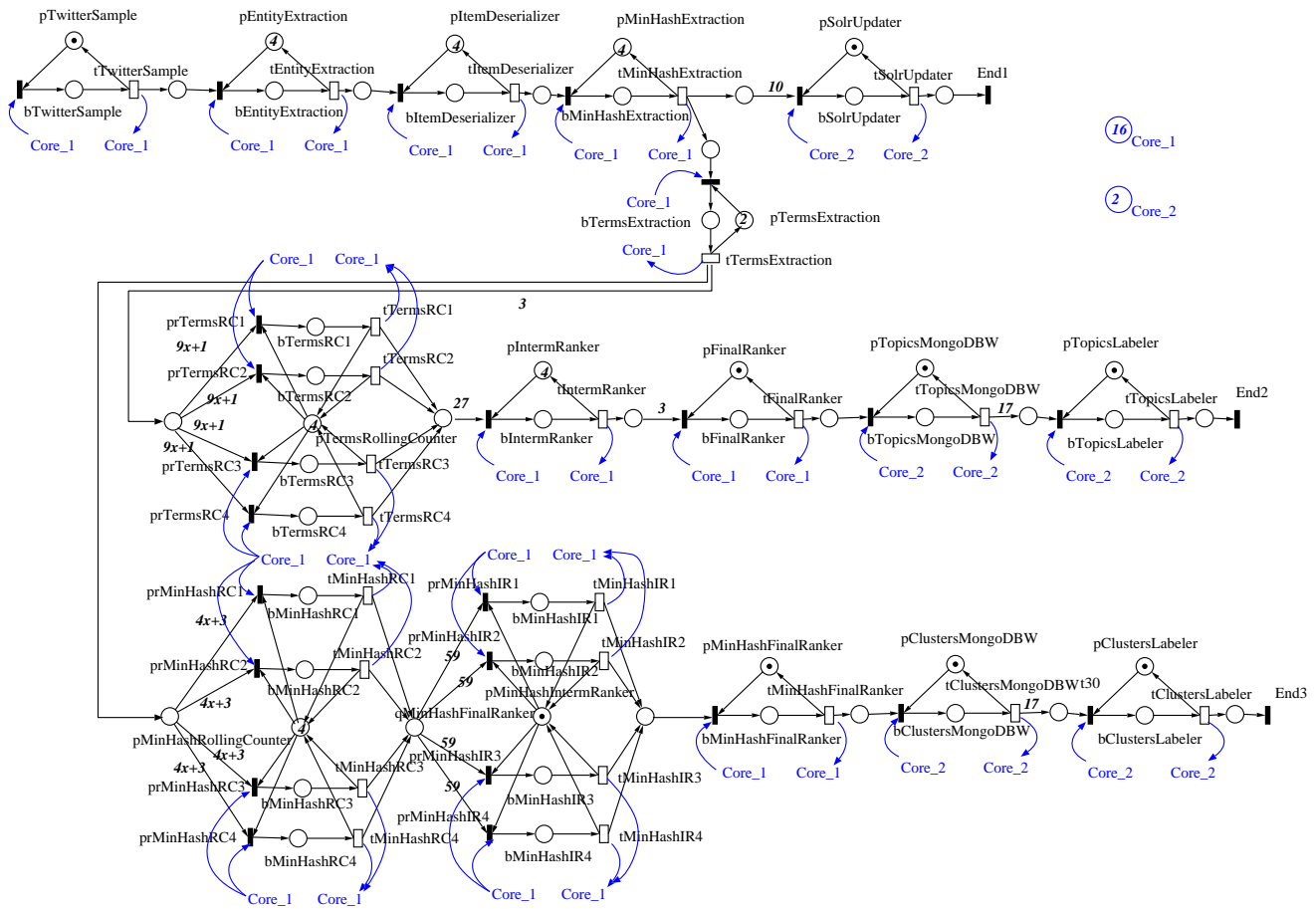


Fig. 11 GSPN for the design in Fig. 3 and 4.

the applications. The following paragraphs detail how each metric is computed.

B.1 Computation of the Mean Time to Failure

Apache Storm applications are fault-tolerant and the progress of the computation of elements in the input streams is managed and stored by a differentiated software, Zookeeper (Apache, 2017b). Therefore, for the failure of a Storm application, the important concept to consider is the failure of the cluster of Zookeeper nodes, rather than the failure of spouts and bolts. In this case resources that execute Zookeeper service have to be stereotyped with `<<StormZookeeper>>` to point out their functionality and with `<<DaComponent>>`, inherited from the DAM profile, to specify their multiplicity with the “resMult” attribute and the estimated MTTF of the resource with the “failure” attribute.

To compute the MTTF, the DICE Simulation tool traverses all Nodes and Devices represented in the Deployment Diagram of the application looking for the element stereotyped as `<<StormZookeeper>>`. When it finds this element, it gets from it the information in its `<<DaComponent>>` stereotype regarding the number of resources used for the Zookeeper cluster and the estimated MTTF of each of them. Finally, the computation of the global MTTF considers that the Storm DIA takes place when all the resources dedicated to execute Zookeeper cluster have failed.

B.2 Computation of the availability

The availability of a system is defined as the readiness for correct service (Avizienis et al., 2004). We compute the percentage of time that a system is available, i.e., the percentage of time that the system is ready for executing a correct service for its users (steady state availability). It is defined as the mean time that the system is working correctly between two consecutive failures (i.e., MTTF) with respect to the Mean Time Between two consecutive Failures (called MTBF). In turn, MTBF is defined as the mean time of correct service until a failure plus the Mean Time to Repair (MTTR). Therefore, the steady state availability is calculated using the formula:

$$Availability = \frac{MTTF}{MTTF + MTTR} \cdot 100.$$

We opted to offer a computation of the system availability when users choose to use preemptable resources, such as Amazon AWS EC2 spot instances. Users need to fill the a) expected amount of time that a preemptable resource will be granted for utilization, and b) the expected amount of time required to choose an alternative affordable set of resources, boot them, set up and configure the technology (i.e., repair time of the application). This information is provided by the attributes “failure” and “repair” of the `<<DaComponent>>` stereotype. Information a) is filled into field MTTF of “failure” attribute and information b) is filled into field MTTR

of “repair” attribute. The computation of the availability is computed from these two values.

C Petri Nets

A GSPN (Generalized Stochastic Petri Net, Marsan et al., 1994) is a Petri net with a stochastic time interpretation. Therefore a GSPN is a modeling formalism suitable for performance analysis purposes. A GSPN model is a bipartite graph, consisting of two types of vertices: places and transitions.

Places are graphically depicted as circles and may contain tokens. A token distribution in the places of a GSPN, namely a marking, represents a state of the modeled system. The dynamic of the system is 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. Immediate transitions are graphically depicted as black thin bars while timed ones are depicted as white thick bars.