FORMAL VERIFICATION OF STORM TOPOLOGIES THROUGH D-VERT

Marconi, Francesco & M. Bersani, Marcello & Rossi, Matteo (2017)
1168-1174. 10.1145/3019612.3019769.

Presentation by: Mátyás Szántó

Budapest University of Technology and Economics, Dept. Of Control Engineering and Information Technology

Outline of this presentation

- **Objective of research**
- Background
  - DIA's
  - DICE
  - Storm Topologies – Apache Storm
- Modelling storm topologies
- Verification of storm topologies
- A use case

Objectives of the research

The paper introduces the methodology for the verification of Storm topologies based on formal validation of refined UML models. It introduces a verification tool, called D-VerT and the transformations needed for enabling the verification of UML models of Data Intensive Applications. It also shows an example of the application of the verification tool (D-VerT) on a generic Storm topology.

Outline of this presentation

- Objective of research
- **Background**
  - DIA's
  - DICE
  - Storm Topologies – Apache Storm
- Modelling storm topologies
- Verification of storm topologies
- A use case

Background I.

Data intensive applications

- DIA's
- Applications based on Big-Data technologies – computational systems that process huge amounts of diversified information usually produced by data sources with high throughputs.
- Users of DIA's: Twitter, Spotify etc. – Large amounts of data gathered from millions of users
- Important topic – quality assessment is a well-researched area

Background II.

DICE

- Data Intesive Cloud applications with iterative quality Enhancements
- European H-2020 research project
- The presented paper research has been conducted as a part of this DICE project
- DICE vision: The design of an application is decomposed into three distinct and consecutive phases, each one associated with a profiled UML diagram.

1. PIM – Platform indepent model: conceptual model of the application
2. PSM – Platform-specific model: this provides the architectrual schema of the application based on a specific data intensive technology
3. Deployment model

Background III.

DICE

- The DICE project approaches the assessment of DIAs by applying formal verification to the architectural models described through (metric) temporal logic.
- The goal of the analysis is to determine, through automated techniques, whether the behavior entailed by the architecture of the application conforms to specific properties over time.

Background IV.

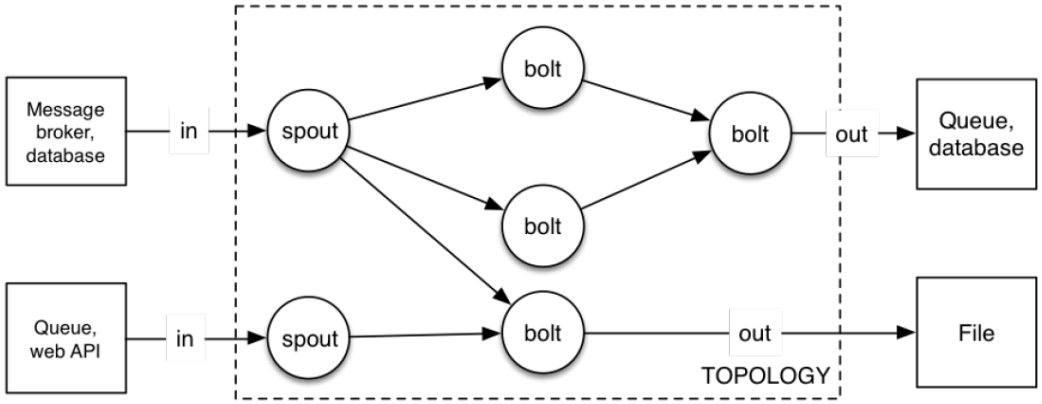
Storm topologies - Apache Storm

- Apache Storm is a stream processing system, developed and open sourced by Twitter in 2012, which allows real-time processing of large-scale streaming data on horizontally scalable systems through a parallel and distributed computation.
- The architecture of a Storm application is defined by means of a topology a directed graph, where nodes are of two kinds: computational nodes, which implement the logic of the application by elaborating information and producing an outcome; and input nodes, which bring information into the application from its environment.

Background V.

Storm topologies - Apache Storm

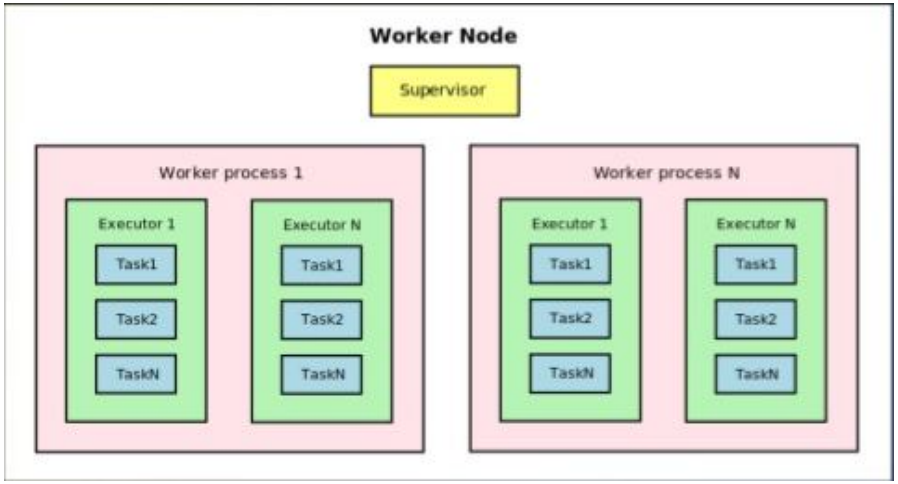
- Directed graph
- Tuples: Means of data transport
  Strings of characters
- Nodes:
  - **Spouts**: stream soruces, get data from external systems such as data brokers (Kafka, RabbitMQ, Kestrel)
  - **Bolts**: tranform data streams to new output streams to be proceessed by following bolts.
  - **Connections** are statically defined



Background VI.

Storm topologies - Apache Storm

- Bolts usually perform operations, such as filtering, join, functions, database interaction, which are combined through the topology to perform complex transformations.
- The Storm runtime leverages the computational powers of distributed clusters
  - Deployed topology is composed of a master node and worker nodes.
  - Worker processes – JVM's
  - Executors are spawned
  - Tasks can execute a bolt or a spout.



- Communication among workers and executors is managed through a multi-level queuing system. Each executor has its own input queue and output queue, the tuples are read and emitted on them.

Outline of this presentation

- Objective of research
- Background
  - DIA's
  - DICE
  - Storm Topologies – Apache Storm
- **Modelling storm topologies**
- Verification of storm topologies
- A use case

Modeling storm topologies I.

Abstracting a formal model of Storm topologies

- The authors identified the relevant aspects of the executions of a generic topology, and some suitable assumptions that allow us to generate models that can be practically managed by state-of-the-art formal verification tools in a reasonable amount of time.
- The formal verification assessment of a Storm topology aims at checking for the existence of bolts that cannot process the incoming stream of tuples on time, thus causing a monotonic growth of the size of their queues.
- In the paper, the behavior of the relevant features and parameters of spouts and bolts is extracted by reverse-engineering the Java interfaces of the Storm API.

Modeling storm topologies II.

Abstracting a formal model of Storm topologies

- The following assumptions were made during the formal model creation:
  - Topologies are assumed to run on a single worker process and each executor runs a single task, which is the default Storm configuration of the runtime.
  - Each bolt has a single receive queue for all its parallel instances and no sending queue, while the workers' queues are not represented.
  - The contents of tuples is not modeled and therefore the size of queues is represented by the number of tuples they contain.
  - The external sources of information abstracted by the spouts are not represented.
  - For each component, the duration of each operation or the permanence in a given state has a minimum and a maximum time.

Modeling storm topologies III.

CLTLoc formulae

- Constrained Linear Temporal Logic over clocks
- Formulae for describing the behaviour of the nodes:

$$\mathtt{add}_j \wedge \neg\mathtt{take}_j \wedge \neg\mathtt{startFail}_j \Rightarrow (\mathrm{X}\mathtt{q}_j = \mathtt{q}_j + \mathtt{r}_{\mathtt{add}_j}) \quad (1)$$

$$\mathtt{take}_j \Rightarrow (\mathrm{X}\mathtt{q}_j = \mathtt{q}_j + \mathtt{r}_{\mathtt{add}_j} - \mathtt{r}_{\mathtt{process}_j}) \quad (2)$$

  - Where: j – node, add – add to queue, take – take from queue, q – size of queue, r - receive

- For the time measurement and and to impose timing contraints between events, the following formula is introduced for each node:

$$\mathtt{process} \wedge \mathtt{emit} \Rightarrow (t_{\mathtt{phase}} \geq \alpha - \epsilon) \wedge (t_{\mathtt{phase}} \leq \alpha + \epsilon) \quad (3)$$

  - Where alpha is the parameter of the bolt and represents the time of one tuple processing.

Outline of this presentation

- Objective of research
- Background
  - DIA's
  - DICE
  - Storm Topologies – Apache Storm
- Modelling storm topologies
- **Verification of storm topologies**
- A use case

Verification of Storm topologies I.

- The verification of DTSM models is done through their automatic translation into a set of CLTLoc formulae, which are then analyzed by the Zot bounded satisfiability checker
- Zot is fed the CLTLoc formulae capturing the application under design and the property to be checked concerning the unbounded growth of the queues of the nodes of interest.
- The tool produces one of two responses:
  - A trace of the modeled Storm topology|a counterexample corresponding to an execution of the application in which one of the queues grows in an unbounded manner in this case, the set of formulae is satisfiable (SAT)
  - OR the notification that the set of formulae is unsatisfiable (UN-SAT). This result increases our confidence that no major design flaws are present in the architecture of the Storm topology for what concerts its ability to process data in a timely manner.

Verification of Storm topologies II.

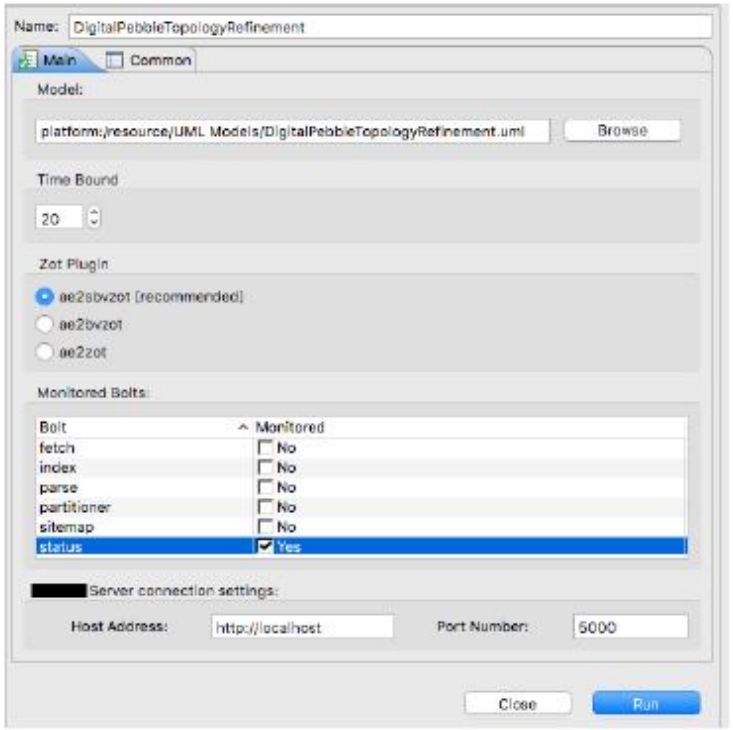
The D-VerT tool

- DICE Verification Tool
- D-VerT is structured as a client-server application:
  - The client component is an Eclipse plug-in. It allows users to de
    ne the design of the DIA under development, then, after providing some additional configuration information, to launch verification tasks and to retrieve their outcomes.
  - The server component is a web application written in Python. The D-VerT server exposes APIs to launch verification tasks and to obtain information about their status. Based on the needs of the user, the D-VerT server can be instantiated either on the local machine or on a remote server.
- D-VerT provides support for the analysis of the boundedness of bolts' queues.

Verification of Storm topologies III.

The D-VerT tool

- Run configuration dialog box
- Nodes of interest are specified
- Depth of the search
- Zot plug-in used
- The analysis allows for the detection of possible runs of the system leading to an unbounded growth in the size of at least one of the aforementioned bolts' queues. This corresponds to the presence in the topology of at least one bolt that is not able to manage the incoming flow of messages in a timely manner.



Outline of this presentation

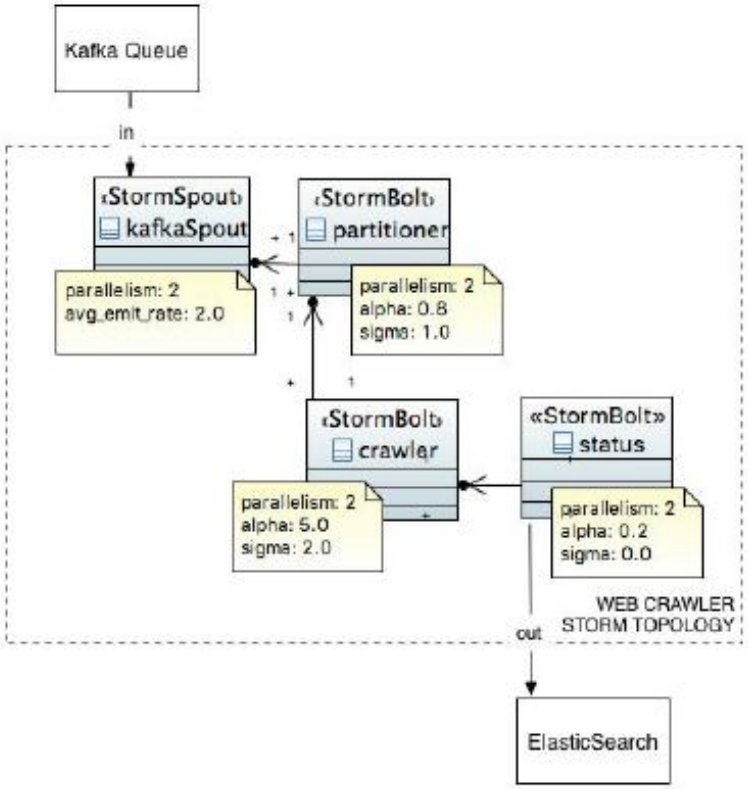
- Objective of research
- Background
  - DIA's
  - DICE
  - Storm Topologies – Apache Storm
- Modelling storm topologies
- Verification of storm topologies
- **A use case**

A use case I.

Setup of the use case

- The use case is taken from the open source project StormCrawler.
- Suppose we want to create a web crawler application to efficiently fetch, parse and index web resources of our interest. Given the dynamic nature of the web, this kind of task can be formulated as a streaming problem, where the input is a continuous stream of URLs that need to be processed by the streaming application with low latency, and the output is represented by the resulting indexes.
- For input node a Kafka Queue, and for output node an ElasticSearch node is used.

A use case II.



Storm topology model of the use case

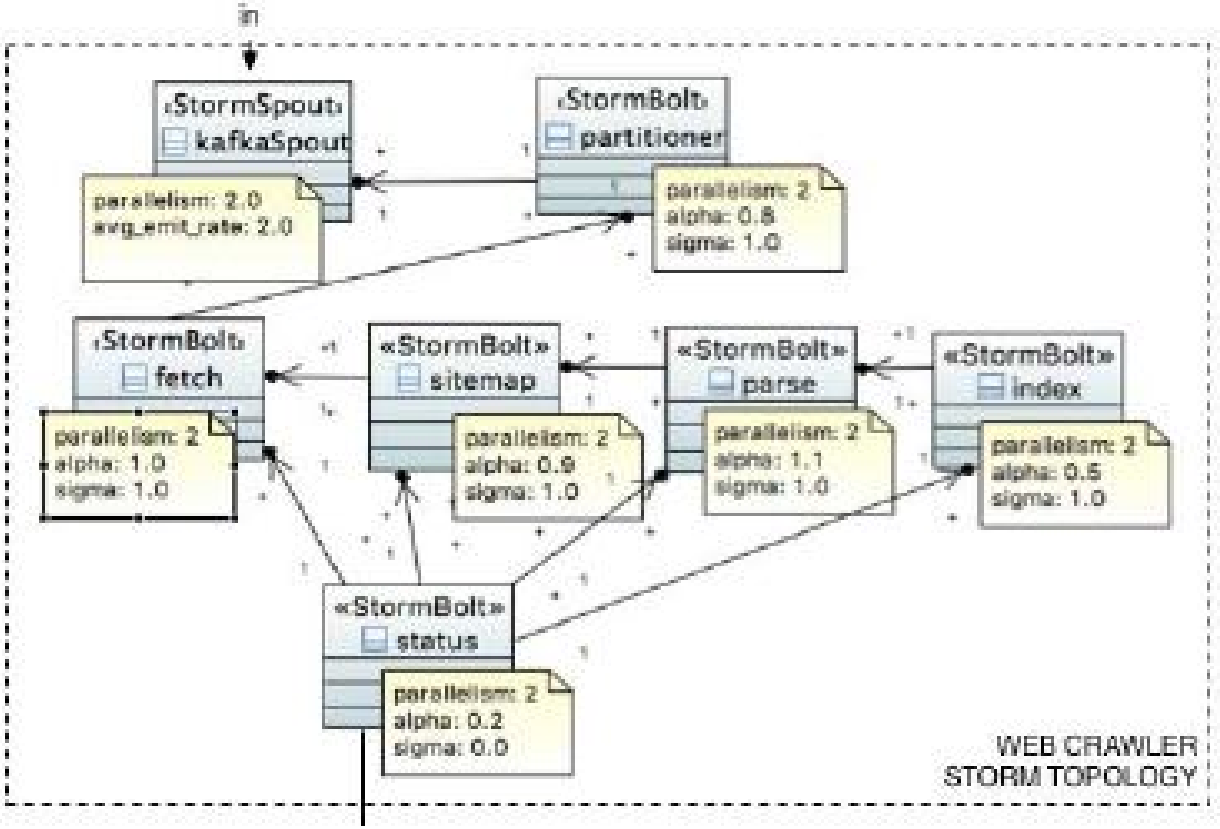
- The partitioner bolt partitions the incoming URLs, while the crawler bolt performs many operations such as resource fetching, metadata parsing and content indexing. The status bolt at the end of the chain indexes the URL, metadata and its status.
- Each of these topology components can be executed by an arbitrary number of parallel threads, and is characterized by the (average) execution time (time needed to perform its task) and by the (average) number of tuples emitted with respect to the number of tuples received as input. These aspects are specified as parameters in the UML class diagram.

A use case III.

System of the use case

- The formal analysis on the initial topology design helped us to detect an unbounded increase in the queue of the crawler bolt.
- This outcome from the tool enabled the authors to review the topology structure, and to decide for the decomposition of the crawler bolt in a series of bolts, each of them performing a subtask of the original bolt (fetch, sitemap, parse and index).
- The refined version of the topology, aims at lightening the load on the core crawling phase by pipelining the main operations and by directly updating the status bolt with partial results computed by the new bolts.

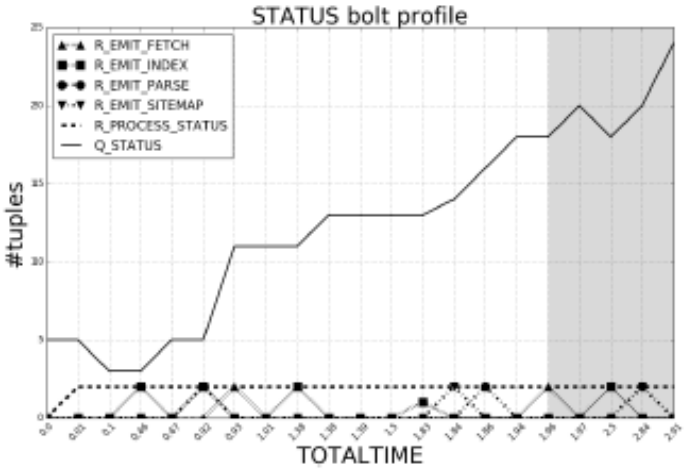
A use case IV.


in
«StormSpout»
kafkaSpout
parallelism: 2.0
avg_emit_rate: 2.0
«StormBolt»
partitioner
parallelism: 2
alpha: 0.8
sigma: 1.0
«StormBolt»
fetch
parallelism: 2
alpha: 1.0
sigma: 1.0
«StormBolt»
sitemap
parallelism: 2
alpha: 0.9
sigma: 1.0
«StormBolt»
parse
parallelism: 2
alpha: 1.1
sigma: 1.0
«StormBolt»
index
parallelism: 2
alpha: 0.5
sigma: 1.0
«StormBolt»
status
parallelism: 2
alpha: 0.2
sigma: 0.0
WEB CRAWLER
STORM TOPOLOGY

A use case V.

System of the use case

- After the refactoring the tool revealed another unwanted run of the system, this time showing a growing trend in the queue of the status bolt. This bolt, subscribing to the streams of the four newly-created bolts, needs a further refinement to avoid excessive loads in its input buffer.



- Increasing the parallelism level of the status bolt to 4 helped improving the design so that no counterexample was found by D-VerT within the selected search bounds (UNSAT result).
- Execution times for the verification vary significantly depending on the topology configuration, ranging from the 98 seconds of the second analysis to the 2130 seconds of the third analysis (UNSAT result).

Thank you for your attention!