Project Number 318763

D3.9 – Final Real-time Scheduling Advisor

Version 1.0
29 May 2015
Final

EC Distribution

Brno University of Technology

Project Partners: **aicas**, **HMI**, **petaFuel**, **SOFTEAM**, **Scuola Superiore Sant'Anna**, **The Open Group**, **University of Stuttgart**, **University of York**, **Brno University of Technology**

Project Partner Contact Information

aicas Fridtjof Siebert Haid-und-Neue Strasse 18 76131 Karlsruhe Germany Tel: +49 721 66396823 E-mail: siebert@aicas.com	**HMI** Markus Schneider Im Breitspiel 11 C 69126 Heidelberg Germany Tel: +49 6221 7260 0 E-mail: schneider@hmi-tec.com
petaFuel Ludwig Adam Muenchnerstrasse 4 85354 Freising Germany Tel: +49 8161 40 60 202 E-mail: ludwig.adam@petafuel.de	**SOFTEAM** Andrey Sadovykh Avenue Victor Hugo 21 75016 Paris France Tel: +33 1 3012 1857 E-mail: andrey.sadovykh@softeam.fr
Scuola Superiore Sant'Anna Mauro Marinoni via Moruzzi 1 56124 Pisa Italy Tel: +39 050 882039 E-mail: m.marinoni@sssup.it	**The Open Group** Scott Hansen Avenue du Parc de Woluwe 56 1160 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of Stuttgart Bastian Koller Nobelstrasse 19 70569 Stuttgart Germany Tel: +49 711 68565891 E-mail: koller@hlrs.de	**University of York** Neil Audsley Deramore Lane York YO10 5GH United Kingdom Tel: +44 1904 325571 E-mail: neil.audsley@cs.york.ac.uk
Brno University of Technology Pavel Smrz Bozetechova 2 61266 Brno Czech Republic Tel: +420 54114 1282 E-mail: smrz@fit.vutbr.cz	

Contents

List of Figures

Document Control

Version	Status	Date
0.1	Document outline	27 April 2015
0.2	Description of Scheduling Advisor components	13 May 2015
0.3	Making the document ready for internal review	22 May 2015
0.4	First draft for internal review	25 May 2015
0.5	Final modifications	28 May 2015
1.0	QA for EC delivery	29 May 2015

Executive summary

This document constitutes deliverable 3.9—*Final Real-time Scheduling Advisor* of work package 3 of the JUNIPER project.

The purpose of this deliverable is to describe the second part of the Scheduling Advisor, the *advisor component*, to show the results of experiments with two more testing applications, and to discuss the integration of the Scheduling Advisor into the JUNIPER project.

The *advisor component* is responsible for analysis of runtime monitoring data of the JUNIPER application deployed for production use. The analysis of the *advisor* may help application administrators to reveal a potentially dangerous behavior of the application and primarily helps application architect to fine tune or enhance the application according to the real runtime data and a knowledge about distributed systems, which is built into the advisor's plugins.

The baseline and evaluation criteria are defined in this document in order to evaluate the Scheduling Advisor. The baseline is established with respect to other distributed platforms and the JUNIPER platform without the Scheduling Advisor. The experiments then show that the advisor component satisfies the evaluation criteria and, along with that, the process of application tuning and redesign is outlined. The experiments were performed on two experimental applications and in both cases, the advisor component revealed significant issues. In current version of the advisor component, we focused on detection of three common issues, namely the data transfer overhead, the out of memory prediction, and the garbage collection frequency and duration. All of these issues may significantly affect the overall performance of JUNIPER applications, especially in the cases of continuous stream processing.

This deliverable also describes the integration of the Scheduling Advisor into the JUNIPER platform. The connections to other components of JUNIPER platform are presented together with exchange formats used for communication between Scheduling Advisor and the rest of JUNIPER platform.

The deliverable 3.8 [1] is prerequisite for reading the present document.

1 Introduction

As JUNIPER applications are distributed concurrent applications, they can be designed, implemented, and deployed on a JUNIPER platform infrastructure in various ways where some of these ways are more efficient than others. The goal of the Scheduling Advisor presented in this document is to measure the efficiency of JUNIPER applications in particular deployments, provide a list of advice concerning possible issues and their solutions related to the efficiency, and contribute to an optimal design, implementation, and deployment of the applications.

The work presented in this document refines the concept and introduces a final implementation of the novel real-time Scheduling Advisor previously described in Deliverable 3.8 "First Prototype of the Real-time Scheduling Advisor" [1]. Moreover, the work is related to other deliverables that are directly or indirectly affected by the Scheduling Advisor design and implementation, mainly, to Deliverable 5.4 "Specification of Model Transformation Chain" [3] and Deliverable 5.6 "Final Integrated MDE Environment" [2].

Scheduling Advisor is a tool composed of two partially independent components. First component is a heterogeneity aware scheduler, which was described in the Deliverable 3.8 [1]. Heterogeneity aware scheduler takes care of placing the JUNIPER application's components over the hardware platform. The other component is an advisor. The advisor component is responsible for analysis of runtime monitoring data of the JUNIPER application deployed for production use. The analysis of the advisor may help application administrators to reveal a potentially dangerous behavior of the application and primarily helps application architect to fine tune or enhance the application according to the real runtime data and a knowledge about distributed systems, which is built into the advisor's plugins.

In the JUNIPER project, the Scheduling Advisor tool is supposed to be a bridge between the developer or administrator and the hardware platform. It brings an automatic heterogeneity aware scheduling and a detection of potentially interesting or dangerous states in running application to the developer. Using the Scheduling Advisor, the development and design of distributed applications becomes easier while the application design and development is less dependent on the experience of developers in area of distributed computing.

1.1 Baseline and Evaluation

As a baseline for the evaluation we consider a distributed platform providing a standard set of tools from modeling to monitoring. Using such platform an architect models the application in modeling framework and developer implements the application following the model driven development. The application is then deployed on a hardware platform by administrators that must decide about the right placement of the application's components. All this requires a deep knowledge about either the application and the hardware. Another choice the administrators have is to use some standard scheduler, e.g., the round robin scheduler described in Deliverable 3.8 [1].

During the application lifetime, the administrators must monitor the performance of the application using the metrics of the infrastructure or metrics built into the application by developers. This again

requires a highly skilled and experienced architects, developers, and administrators. The schedulability analysis is performed on data tediously acquired after many manual re-deployments. Finally, the process of improvement or redesign of the application can be done only with help of manual analysis of monitoring data. This often means that the first changes are made in the design of metrics and data that are gathered during the application's run.

Evaluation criteria for the Scheduling Advisor are based on unsatisfying situation on the field of distributed applications' development and scheduling described in previous paragraphs. There are two main objectives of the Scheduling Advisor:

1. To improve the performance of distributed stream processing applications on hardware heterogeneous clusters.

2. To improve the basic monitoring of the distributed stream processing applications, to detect common problems of distributed applications, and to help solving these problems.

While the first evaluation criterion was discussed and fulfilled in the previous Deliverable 3.8 [1], the second criterion will be presented in this document.

2 Scheduling Advisor

The JUNIPER platform consists of multiple interconnected components where, in particular use cases, some of these components may be omitted. This allows a wide range of potential users to take the advantage of some or all of the possibilities that the JUNIPER platform offers. The Scheduling Advisor is one of the core components that helps with deployment itself, and later, with performance tuning and operations monitoring over the lifetime of JUNIPER applications.

As was firstly described in Deliverable 3.8 ([1]; see also Section 1), the Scheduling Advisor represents two components that closely cooperate through the monitoring data they produce and use.

The first component, which was widely described and benchmarked in Deliverable 3.8 [1], is *the heterogeneity aware scheduler*. This scheduler works in two phases. At first, it benchmarks components of a JUNIPER application on different hardware (HW) or virtual machine (VM) classes on the application's target hardware platform. The resulting monitoring data contain information about performance of each component (i.e., each JUNIPER program) on each HW and VM class. After the benchmarking phase and based on its results, the scheduler plans the best known placement of the JUNIPER programs in the way that overall performance of the JUNIPER application is nearly optimal.

The second component of the Scheduling Advisor is the *advisor component*. Whenever the application is deployed either for pre-production or production use, the monitoring data gathered over time may be exploited to further tuning of performance. The advisor component detects through the plugins different suspicious states and behaviors. Architecture of the advisor component will be described in Section 3.1 of this deliverable; a detailed description of the plugins currently available for the advisor is in Section 3.2.

2.1 Scheduling Advisor as a Part of the JUNIPER Project

In order to demonstrate the interconnection of the JUNIPER components, we will discuss the process of Scheduler Advisor's work and describe the connections to other parts of the JUNIPER platform. As the concept of the Scheduling Advisor was already described from different perspectives in Deliverable 3.8 [1], in Section 2.2 "Integration Scenario" and in Section 3.3 "Architecture and Prototype Implementation of the Scheduling Advisor", we only outline the main ideas here.

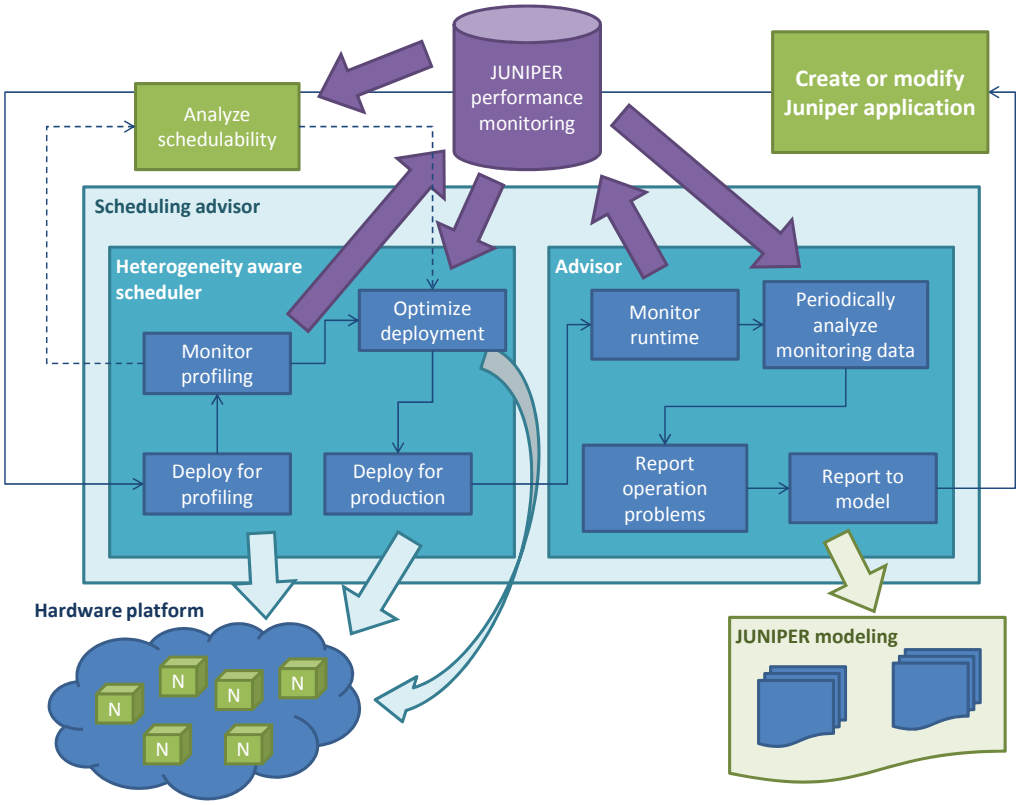

Figure 1: Tasks of Scheduling Advisor and adjacent components—thin arrows show sequence of tasks, thick arrows show dependencies of tasks and JUNIPER platform's components

Figure 1 depicts a part of the JUNIPER platform with tasks belonging to the Scheduling Advisor and other components. Thin arrows show the step by step process from a modeling to a production use of the application while thick arrows show dependencies between the steps and components of JUNIPER platform. An entry point of the diagram is the "Create or modify Juniper application" step.

The process may be briefly described in the following way: The development of a JUNIPER application starts by modeling. The application then has to be implemented and tested. After the first deployment of the application on its target platform and a few profiling runs, the schedulability analysis may be run to know if the application is able to meet defined real-time criteria. Later, the production deployment can start. During the production use of the application, different factors

affect the application performance and after some time the application may need modifications or maintenance.

The process consists of the following individual steps:

1. *Create or modify Juniper application*—Modeling, programming and testing of the new or modified application.

2. *Deploy for profiling*—To take advantage of the Scheduler Advisor's heterogeneity aware scheduler and of the Schedulability analysis, the profiling have to be run. Profiling step schedules each JUNIPER program to each hardware or virtual machine type. To achieve this, multiple redeployments may be made by the *Scheduling Advisor*.

3. *Monitor profiling*—This step generates a valuable data about profiling into the JUNIPER monitoring. This data is later used for heterogeneity aware scheduling and for schedulability analysis.

4. *Analyze schedulability*—Based on data stored to monitoring during the profiling deployment, schedulability analyser can determine if the JUNIPER programs are schedulable on the target platform.

5. *Optimize deployment*—Based on data stored to monitoring during the profiling deployment, the heterogeneity aware scheduler finds near optimal placement of JUNIPER programs over the target HW platform. This results in better performance of the JUNIPER application on heterogeneous clusters and better utilization of rare pieces of hardware (hardware classes) in the hardware platform.

6. *Deploy for production*—Whenever the right placement of the JUNIPER programs over the hardware platform is known, the production deployment can start. The placement is a result of heterogeneity aware scheduler or a decision of the administrator, developer or software architect.

7. *Monitor runtime*—During the runtime of the JUNIPER application, monitoring built into the JUNIPER program API takes care of the nonintrusive data gathering.

8. *Periodically analyze monitoring data*—To reveal potential problems of the running JUNIPER application,Scheduling Advisor periodically analyzes the monitoring data. Advisor's analyzers follow the plug-in architecture, which allows to easily implement additional diagnostics.

9. *Report operation problems*—Whenever Advisor detects a problem, explanatory messages are saved into the runtime log.

10. *Report to the model*—In case of application tuning or redesign, reports from the *advisor* may be displayed right in JUNIPER modeling where software architect can see, which components or interconnections cause problems. This way, the architect is able to precisely consider the right steps necessary to improve the application's design.

2.2 Schedulig Advisor Integration with JUNIPER Components

The Scheduling Advisor is integrated with number of other components of the JUNIPER platform. The advisor exploits the JUNIPER monitoring and at the same time the advisor writes into the monitoring a data important for other components. Deployment to the JUNIPER platform can be optimized by the Scheduling Advisor and monitoring data may be used in modeling by means of the advisor component.

The deployment process of a JUNIPER application can be simplified by the Scheduling Advisor in the way that the advisor is able to prepare a placement of the JUNIPER application over its target hardware platform. This is done by modifying the *deployment-plan.xml* file generated by JUNIPER MDE Environment. The Scheduling Advisor will contain an easy to use shell scripts that will run all the profiling and redeployments for profiling for the user.

Gathering of data required by Schedulability Analyzer is being done automatically during the profiling phase of Scheduling Advisor. The data is gathered using the monitoring calls built into the JUNIPER API. There are multiple monitoring sensors that read data in different phases of JUNIPER program execution. The only metric required for schedulability analysis is the total execution time taken by JUNIPER program to process one piece of data received from another program. Scheduling Analyzer then analyzes this data using the JUNIPER monitoring API.

Along with the data used by the Schedulability Analyzer, Scheduling Advisor stores various different metrics about the running JUNIPER programs. These are for example: execution time per piece of data, communication time, different memory and garbage collection information, and other. Capturing of this data is completely transparent to JUNIPER programmer. Monitoring functions of the Scheduler Advisor are implemented in the JUNIPER monitoring library, and at the same place, there are methods for data retrieval from the JUNIPER monitoring back to the *advisor component* of the Scheduling Advisor. Aggregated data from the monitoring, to be analyzed by the *advisor component*, are then processed internally in the Scheduling Advisor.

For a seamless integration of the Scheduling Advisor into the JUNIPER platform, the results of the advisor component can be displayed right in the *JUNIPER MDE Environment*. The reason for this is mainly the easier process of JUNIPER application's modification and performance tuning by the architect. To achieve such interoperability, the advisor component uses a *XML exchange file* that can be loaded by JUNIPER MDE Environment. The advice's data is then displayed in the model connected to according objects in the model.

3 Advisor Component

This section describes the advisor component, which is the second part of the Scheduling Advisor (see Section 2). The advisor component analyzes monitoring data of a JUNIPER application looking up for potential issues related to the application performance and resource utilization in the JUNIPER platform. For each issue, the advisor component provides its detailed description (what is the issue, where is the issue and why is it important). Moreover, the list of issues found is described in an XML document of a defined format, so it can be imported and processed in other JUNIPER tools.

3.1 Architecture

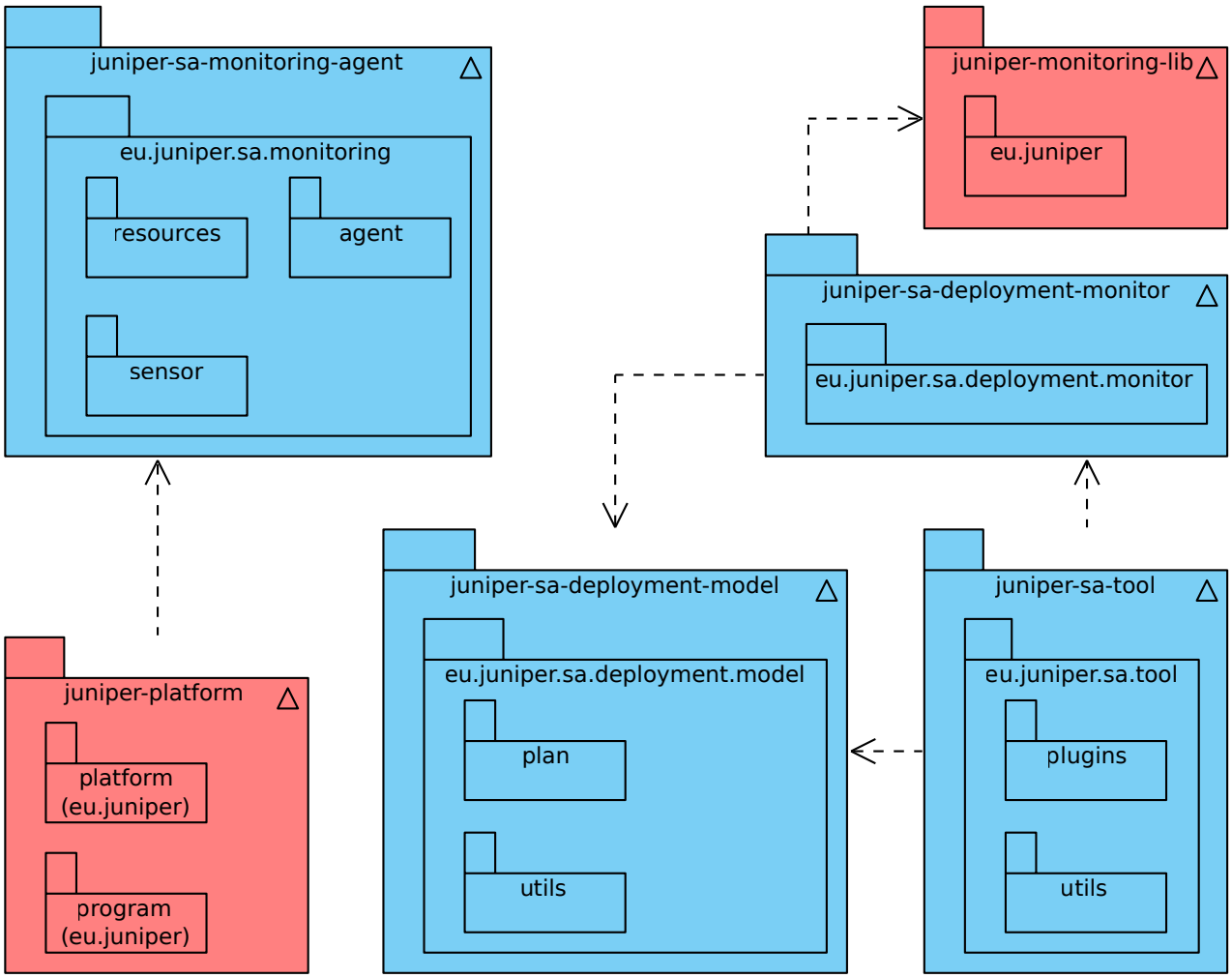

Figure 2: Modules of the Scheduling Advisor and their dependencies (blue modules are results of this deliverable, red modules are integrated results of other deliverables).

The architecture of the Scheduling Advisor's advisor component consists of several modules, which integrate different parts of the JUNIPER project (see Figure 2). These modules process inputs from or produce outputs to other JUNIPER tools related mainly to schedulability analysis and JUNIPER applications modeling and development. The main modules of the advisor component and their usages are the following:

1. *juniper-sa-deployment-model*—represents a deployment model of a JUNIPER application as a hierarchy of objects of predefined classes (for example, JUNIPER programs, program instances, data connections, MPI groups, etc.); loads the model from and saves it to an XML document describing a deployment plan (see Deliverable 5.6 [2]);

2. *juniper-sa-monitoring-agent*—monitors various runtime characteristics of JUNIPER programs and send them to a monitoring results storage (i.e., to a monitoring service of the JUNIPER platform or to a database connected via JDBC); the monitoring is performed by various monitoring sensors that allows to monitor, for example, CPU and memory usage (heap, non-heap, and swap memory), execution and communication times, garbage collection characteristics, and others;

3. *juniper-sa-deployment-monitor*—reads stored monitoring data from various sources (i.e., from a monitoring service of the JUNIPER platform or from a database connected via JDBC) and pass the data to other modules via an unified API;

4. *juniper-sa-tool*—analyzes a deployment model and processes the monitoring data, provides a list of advice based on the analysis and the processing above; implements a user interface of the Scheduling Advisor.

Moreover, the advisor component or more specifically its modules utilize existing software tools and libraries provided by the project partners, namely:

1. *juniper-platform*—that implements the JUNIPER platform model and runtime utilized by JUNIPER applications;

2. *juniper-monitoring-lib*—that implements methods to save monitoring data of JUNIPER applications to a monitoring service on the JUNIPER platform and later, to read them from the service for subsequent analysis by the Scheduling Advisor.

Finally, the modules of the advisor component require several auxiliary libraries, such as: Java Interface for Open MPI[1], Modelio Java Designer[2], and JuniperIDE Code Generation Helper described in Deliverbale 5.6.

[1] `https://www.open-mpi.org/faq/?category=java`
[2] `https://www.modeliosoft.com/en/modules/modelio-java-designer.html`

3.1.1 Monitoring the JUNIPER Programs for the Scheduling Advisor

By *juniper-sa-monitoring-agent* module, executions of JUNIPER programs can be measured and monitoring data can be reported to a central monitoring service. The monitoring is performed by various monitoring sensors that allows to measure, for example, CPU and memory usage (heap, non-heap, and swap memory), execution and communication times, garbage collection characteristics, and others. These sensors are integrated into the JUNIPER platform implementation classes. More specifically, method *run(String programJavaClass, String methodToRun)* of class *eu.juniper.platform.Rte* was modified to create and use a sensor for measuring a program execution by:

```
void run(String programJavaClass, String methodToRun) /* throws ... */ {
  // start monitoring agent and create sensor for the program
  juniperPlatform.monitoringAgent =
     MonitoringAgentFactory.createMonitoringAgentSingletonBySystemProperty
    (juniperPlatform.getApplicationModel().getName());
  juniperPlatform.programInstanceSensor =
     juniperPlatform.monitoringAgent.createProgramInstanceSensor
    (juniperPlatform.getMpiRank());
  /* ... */
  // indicate the program start point
  juniperPlatform.programInstanceSensor.programStarts();
  // run the program
  instanceMethod.invoke(programInstance, new Object[] {juniperPlatform});
  // indicate the program end point
  juniperPlatform.programInstanceSensor.programEnds();
  /* ... */
}
```

Moreover, method *public ArrayList<Object> transferData(String connectionName)* of class *eu.juniper.program.JuniperProgram* was modified to create and use a sensor for measuring a receive operation by:

```
public ArrayList<Object> transferData(String connectionName) {
  // load a data connection sensor associated with the connection of
     create a new one
  DataConnectionSensorInterface dataConnectionSensor =
     this.dataConnectionSensors.get(connectionName);
  if (dataConnectionSensor == null) {
    dataConnectionSensor =
       juniperPlatform.getMonitoringAgent().createDataConnectionSensor
      (this.myGlobalRank, connectionName);
    this.dataConnectionSensors.put(connectionName, dataConnectionSensor);
  }
  // indicate the receive start point
  dataConnectionSensor.receiveStarts();
  /* ... */
  // indicate the receive end point
```

```
  juniperPlatform.getProgramInstanceSensor().subtract
    (dataConnectionSensor.receiveEnds());
  /* ... */
}
```

Finally, a JUNIPER application utilizing the modified JUNIPER platform can enable and use the monitoring sensors by JVM system property *MonitoringAgentEnabled*, which can have one of the following values:

- an URL utilizing the HTTP or HTTPS protocol to connect to the central monitoring service of the JUNIPER platform and to report and store the monitoring data there,
- a JDBC URL to connect a database server to store the monitoring data in a database,
- a file path to store the monitoring data on a local disk as an SQL dump file (a list of SQL INSERT statements).

Finally, to avoid the monitoring whenever it is not required, without the *MonitoringAgentEnabled* system property defined, all the monitoring sensors will be disabled.

3.1.2 Scheduling Advisor Tool Plugin Architecture

The module called *juniper-sa-tool*, which performs analyses of deployment models and processing monitoring data to provide a list of advice, employs plugins to carry out individual analyses and related data processing and to generate different types of advice. This allows to extend the Scheduling Advisor without modifying and recompiling its source code.

The plugins are realized as Java classes that have to implement *AdvisorInterface* interface and belong to package *eu.juniper.sa.tool.plugins*. When processing the monitoring data, the Scheduling Advisor searches the available plugins and run each of them to perform particular analyses on the data and produce related advice if needed.

3.2 Scheduling Advice

In the advisor component, individual types of detected problems and related advice correspond to individual plugins of the *juniper-sa-tool* module. As the Scheduling Advisor is gradually utilized in use cases of the JUNIPER project partners, additional plugins are developed to address emerging problems and provide required advice.

3.2.1 Types of Scheduling Advice

In time of finalizing of this deliverable, the following plugins of the advisor component were available:

- AdvisorDataTransferOverhead,
- AdvisorOutOfMemoryPrediction,
- AdvisorGarbageCollectionPerformance.

These plugins are described individually in the following sections.

AdvisorDataTransferOverhead Plugin This plugin detects JUNIPER programs that spend the most of their time by data transfers instead of by computation (i.e., there is a longer communication time than a computation time where the communication time includes a time spend by synchronously receiving and deserializing of the data). This may indicate a very simple JUNIPER program with a high data transfer overhead or a JUNIPER program waiting for data most of the time. In the first case, the JUNIPER program should be merged with other JUNIPER programs to reduce unnecessary communication and increase the overall performance (in other words, the architecture of the JUNIPER application should be reorganized). In the second case, the data flows/streams in the JUNIPER application should be optimized to reduce waiting times.

AdvisorOutOfMemoryPrediction Plugin This plugin detects JUNIPER programs where the memory usage is growing over the time, and which may eventually run out of memory. The problem is detected by an application of the linear regression analysis to identify a linear trend in memory usage (the plugin analyses a heap and non-heap memory sizes and a swap file size). When detected, the problem may indicate potential memory leaks, which may eventually result into throwing OutOfMemoryError Java exceptions in the problematic JUNIPER programs or may cause performance issues due to swapping out memory pages of the JUNIPER programs.

AdvisorGarbageCollectionPerformance Plugin This plugin detects JUNIPER programs that spent significant time on garbage collecting (the garbage collection is a form of automatic memory management in Java). The plugin analyses a total time spent on garbage collection and a number of the garbage collections. Long garbage collections may affect negatively responsiveness of a JUNIPER application, which is critical in real-time stream processing of Big Data (any delay in processing of such data may result into data-loss issues).

3.2.2 Advice Output Format

Module *juniper-sa-tool* provides a simple user interface to call the advisor component on a given deployment plan and a given set of monitoring data. The main goal of the *juniper-sa-tool* module is to produce, based on the inputs above, a list of advice which is both human-readable and suitable for further processing by other JUNIPER tools, such as in JUNIPER IDE (see Deliverable 5.6 [2]). Through this module, the advisor component is expected to be integrated into other tools.

The *juniper-sa-tool* module produces a list of advice in a human-readable plain-text format and in a structured XML format suitable for further processing. For example, the module executed based on a deployment plan and monitoring data of a sample JUNIPER MapReduce application with a single plugin *AdvisorDataTransferOverhead* produces an output starting with the following text:

```
*** loading and executing plugins from package eu.juniper.sa.tool.plugins
*** executing advisor AdvisorDataTransferOverhead with the following
   description:
This advisor detects JUNIPER programs with long data transfer times
   (i.e., a time spent on waiting for receiving data) and short
   computation. This indicate a very simple program with a high data
   transfer overhead (it should be merged with other programs) or a
```

```
   program waiting for data most the time (data flows/stream should be
   optimized in the Juniper application of the Juniper program).

Advice DataTransferOverhead: The instance with global rank 4 of Juniper
   program 'Reducer' running at cloud node with host name/IP address
   127.0.0.1 was receiving data in 0,495000 seconds of 0,598000 seconds
   of total execution time. That makes 82,775920 percentage of execution
   time spent by receiving data (the recommended maximum is 25,000000
   percentage).
```

Moreover, in this case, the advisor component produces the following XML document[3], which can be further processed in JUNIPER IDE:

```
<?xml version="1.0"?>
<sa:schedulingAdvice xmlns:sa="http://www.fit.vutbr.cz/homes/rychly␣
   /juniper/scheduling-advisor"
   xmlns:xmime="http://www.w3.org/2005/05/xmlmime"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.fit.vutbr.cz/homes/rychly␣
   /juniper-sa/xsd/scheduling-advisor-v4.xsd">
  <sa:advice xmlns:sa="http://www.fit.vutbr.cz/homes/rychly␣
     /juniper/scheduling-advisor" category="resource" severity="warning">
    <sa:problem>The <sa:objectRef attId="deployment-plan"
       objId="uuid_9cc9b1f3-1d86-46f4-8d78-3ad607831057"/> running at
       cloud node with host name/IP address 127.0.0.1 was receiving data
       in 0,495000 seconds of 0,598000 seconds of total execution time.
       That makes 82,775920 percentage of execution time spent by
       receiving data (the recommended maximum is 25,000000
       percentage).</sa:problem>
    <sa:sources>
      <sa:attachmentRef attId="deployment-plan"/>
    </sa:sources>
  </sa:advice>
  <!-- ... -->
  <sa:attachment attId="deployment-plan">
    <sa:anyXml xmime:contentType="application/xml">
      <application xmlns:sa="http://www.fit.vutbr.cz/homes/rychly␣
         /juniper/scheduling-advisor" name="Symbols_Counter">
        <!-- ... -->
        <DeploymentModel>
          <!-- ... -->
          <cloudnode hostipaddr="127.0.0.1" mpiglobalrank="4"
             sa:objId="uuid_9cc9b1f3-1d86-46f4-8d78-3ad607831057"/>
        </DeploymentModel>
      </application>
    </sa:anyXml>
```

[3]For demonstration purposes, just a fragment of the XML document is shown (for full outputs see Section 4 and appendices of this document.

```
  </sa:attachment>
</sa:schedulingAdvice>
```

4 Experimental results

To check the abilities of the *advisor component*, we made series of experiments reflecting different aspects of the advisor. Experiments were performed on the BUT's infrastructure using three nodes that run Ubuntu server (version 14.04), each with Intel Xeon E5-2630 v2 (Sandy Bridge-EP), 12 cores (24 virtual cores), 2.6 GHz, 15 MB of L3 cache and 64 GB of DDR3 RAM at 1333 MHz. Nodes were connected with Gigabit Ethernet. Experiments read data from NFS and pre-created RAM drives were used to hold the generated monitoring data to avoid the overhead of the monitoring in cases where all processing (i.e., all messages and computations) was monitored (in standard situation the monitoring data would be saved directly to the JUNIPER monitoring and the advisor would monitor just a fraction of data processed).

4.1 Financial Use Case (petaFuel)

The goal of the petaFuel application is to analyze a logs of financial transactions provided by the petaFuel company. By data analysis, the company is able to monitor the payment card transactions to avoid unauthorized operations. These statistics can be used to view trends, histograms and possibly detect unusual behavior of the system (like cyber-attacks). The application consists of four processing steps:

- Storage,
- Join,
- Unify,
- Write.

Storage program reads data from PetaFuel csv dumps and uniformly sends the data either to *Join* or *Unify*. There are eight types of csv dumps: messages, billings, chargebacks, product cards, redemptions, refunds, replenishments and transactions.

Unify program creates aggregated objects and sends them to Write. Every incoming object (billing, redemption, refund, replenishment) is converted to unified form containing: Event (billing, redemption, refund, replenishment), Timestamp, Product, Card id, Balance change.

Join program, alike the Unify, creates aggregated objects and sends them to the Write program. The only difference is that before the aggregated object can be created, information from two types of log messages need to be joined (messages with chargebacks and transactions with product cards).

Write program computes statistics from incoming aggregated objects and stores them to memory. Data is segmented at first to five-minute chunks, hours' statistics are computed from these chunks. Days', months' and years' statistics are computed consequently. Standard deviations and averages for particular events in time are available.

4.1.1 Monitoring

To deploy the petaFuel application for the first time, a default deployment plan produced by Juniper IDE tools has been used. The deployment plan defines four instances of *Storage*, four instances of

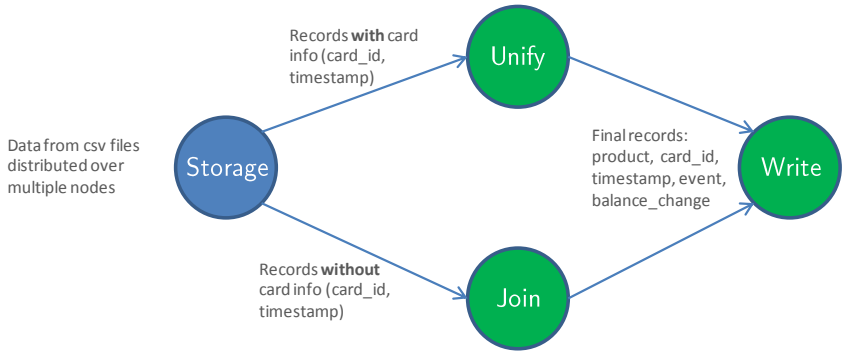

Figure 3: Topology of the experimental implementation of the petaFuel's financial use case

Join, four instances of *Unify*, and one instance of *Write* programs deployed on three infrastructure nodes: node 1 (147.229.8.104), node 2 (147.229.8.105), and node 3 (147.229.8.106) in such way that all *Storage* instances run on node 1 together with one instance of *Join* program, all but the one instances of *Join* program run on node 2 together with one instance of *Unify* program, and finally, all but the one *Unify* instances and one instance of *Write* programs run on node 3. An XML document with the deployment plan is attached in AppendixA.

For each program and each of its instances, the following measurements has been performed to obtain a monitoring data for further analysis:

- *the duration of a program execution* processing one set of input data[4] including receiving of the data but without sending the data to another programs (sending programs are waiting for the receiving programs, thus the sending time should not be included in the program execution time),
- *the duration of receiving of data* including the preparation for the processing in the receiving program—this usually means a deserialization of the data into valid Java objects,
- *the amount of memory* utilized by the program while processing one set of input data (more specifically, a heap memory, a non-heap memory, and a swap space utilization),
- *the count and duration of garbage collections* performed in the program while processing one set of input data.

The application has been run with the same data-set of about 1 GB using different message size when transporting the data between JUNIPER programs, namely for message size of 1 record from source data-set, of 10 records, of 100 records, and of 1000 records. The monitoring data have been gathered in each case. Part of the analyses involving memory consumption have been performed on the data from the runs with message size of 1000 records, and other part involving the inappropriate data granularity was performed on the data from runs with all message sizes. The monitoring data have been reported to a monitoring service of the JUNIPER platform.

[4]Input data is processed by programs of the application in sets (batches) of defined size (message size), the results are communicated between programs in the same sets (message sizes) too.

4.1.2 Analysis

The subsequent analysis of the monitoring data obtained, as was described in the previous section, has been performed by the advisor component of the Scheduling Advisor utilizing the following plugins:

- AdvisorDataTransferOverhead,
- AdvisorOutOfMemoryPrediction,
- AdvisorGarbageCollectionPerformance.

The full output of the analysis is described in AppendixB. The results obtained indicate several issues that are described and discussed in the following paragraphs.

Big data transfer overhead in the cases of the small message sizes The execution times of the whole application with different message sizes indicated that the message size has significant impact on an overall performance of the application. More specifically, the duration of the processing of the same data have been 523 seconds, 304 seconds, 252 seconds, and 238 seconds, for message sizes 1, 10, 100, and 1000 records, respectively (thus, the fastest execution is in the case of message size of 1000 records). The reason was exposed in the analysis performed by the *AdvisorDataTransferOverhead* plugin, which indicated an increasing time spent by communication (i.e., data transfers between communicating programs) with decreasing message sizes. For example, in the case of the *Write* program instance with rank 14[5], the communication took 4443 seconds of 4749 seconds of overall execution time of the Write instance (i.e., 93.5 % of the time was spent by the communication) for message size of 1 record, 494 seconds of 750 seconds (65.9 %) for message size of 10 records, 59 seconds of 232 seconds (25.5 %) for message size of 100 records, and 21 seconds of 178 seconds (12 %) for message size of 1000 records. Therefore, developers should be aware that it is necessary to process and communicate bigger messages to improve the efficiency by avoiding communication overhead. For example, the plain-text output of the *AdvisorDataTransferOverhead* plugin for the *Write* program instance and message size of 10 records, where the monitoring data have been gathered and reported for one in 100 executions, is the following:

```
Advice DataTransferOverhead: The instance with global rank 12 of Juniper
   program 'Write' running at cloud node with host name/IP address
   147.229.12.172 was receiving data in 4,940000 seconds of 7,502000
   seconds of its total execution time (averages are 0,000232 seconds
   for 21324 receives of data and 0,000704 seconds for 10662
   executions). That makes 65,849107 percentage of execution time spent
   by receiving data.
```

[5]The data transfer overhead is visible also in other instances of other programs, not only in the *Write* program instance with rank 14, which has been selected just for demonstration purposes (as it causes also the most significant issues as described later in this section).

Too frequent garbage collections The monitoring data of garbage collecting were processed by the *AdvisorGarbageCollectionPerformance* plugin. The plugin indicated that the *Write* program instance with rank 14 (the same as in the paragraph before) performed 639 garbage collections taking 1.43 seconds of 178 seconds of its total execution time (for message size of 1000 records). Compared to the second most frequent garbage collecting instance *Unify* with rank 9 (which performed just 30 garbage collections taking 0.099 seconds of 42 seconds of its total execution time), it is obvious that the *Write* program instance with rank 14 should be optimized. If it would not be optimized, it can cause performance issues—especially in the real-time processing. A subsequent inspection of of the *Write* program's source code reveals that the program creates many small short-lived objects for each processed message, which requires the frequent garbage collection. This is a common issue in Java applications with well-known solutions (e.g., to use object pooling, explicit null setting, or explicit garbage collection when possible). For example, the plain-text output of the *AdvisorGarbageCollectionPerformance* plugin for the *Write* program instance is the following:

```
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance
   with global rank 12 of Juniper program 'Write' running at cloud node
   with host name/IP address 147.229.12.172 was performed 639 garbage
   collections that took 1,430000 seconds in 177,956000 seconds of total
   execution time of the program (averages are 0,002238 seconds per
   garbage collection and 0,015694 seconds for the execution time). That
   makes 0,803569 percentage of execution time spent by garbage
   collections.
```

Out-of-memory prediction Finally, we applied the *AdvisorOutOfMemoryPrediction* plugin on the monitoring data related to the memory consumption by individual instances of the application programs. The analysis performed by this plugin indicated significant trends in increasing heap memory utilization in instances of *Join* program. This program keeps an internal storage of received data to perform a continuous data analysis, i.e., it creates and stores many long-lived Java objects that cannot be gathered by the garbage collector. In the case of batch processing those data is stored for limited time, so the increasing memory usage is not critical. However, in continuous stream processing, when the *Join* program instances will be running for a long (or unlimited) time, the continuously increasing memory usage will eventually cause Java out-of-memory exceptions. The *AdvisorOutOfMemoryPrediction* plugin indicated that the out-of-memory state may be reached in approximately 1.25 hours from the start of the application. For example, the plain-text output for the *Join* program instance is the following:

```
Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 7
   of Juniper program 'Join' running at cloud node with host name/IP
   address 147.229.12.153 has the memory usage growing over the time by
   the approximate rate of change 859929,198753 Bytes per second for the
   heap memory size (the recommended maximum is 0,100000 Bytes per
   second). This may result into OutOfMemoryError errors in the program
   on 2015-05-22 14:36:01.783 (that is 01:13:27.278 since the beginning
   of analyzed data on 2015-05-22 13:22:34.505; the heap memory is
   limited to 3737649152 Bytes). The sample linear regression model is
```

```
Y_{size_in_bytes} = -52298136,130751 + 859929,198753 *
X_{time_in_sec}.
```

4.1.3 Conclusion

For the petaFuel application, the advisor component of the Scheduling Advisor indicated several potential issues that should be fixed by its developers. Namely, the following remedies should be applied:

- Appropriate message size should be set, as the analysis indicated that there is a big communication overhead for small message sizes. In our case, a message size of 1000 records proved to be the most efficient. Bigger message sizes may result into minimal communication overhead, however, they may also limit parallelism in the application (programs would be processing big sets of data in each execution which may take time and delay subsequent processing and eventually decrease data-flow speed through the application).
- The *Write* program should be optimized with focus on memory management to limit frequency of automatic garbage collection which may cause performance issues in real-time processing.
- The *Join* program can cause out-of-memory Java exceptions in long application runs as it uses internal in-memory storage. The estimated time to out-of-memory state was estimated at 1.25 hours from the start of the application run. To use the application in the long runs, e.g., for continuous stream processing, the data in the *Join* programs should be stored in an external storage.

	Whole application		Slowest component (program Write, rank 12)					
Message size	Time total [s] (all data)	Gain over previous	Communication time total [s]	Computation total [s]	Communication % of computation	Communication per message avg. [s]	Computation per message avg. [s]	Communication % of computation
1	523		4443	4749	93,56%	0,000063	0,0001347	46,77%
10	304	72,04%	494	750,2	65,85%	0,000232	0,000704	32,95%
100	252	20,63%	59,09	231,45	25,53%	0,000262	0,002052	12,77%
1000	238	5,88%	21,356	177,956	12,00%	0,000942	0,015694	6,00%
	Overall gain:	119,75%						

Figure 4: Results of multiple runs of the petaFuel application with different message sizes; besides the total execution times of whole computation, the table contains accumulated and average times for the slowest component—*Write*

4.2 Twitter Demonstration Use Case (BUT)

The Twitter application performs natural language processing of tweets mentioning terms from computer gaming area. The goal is to learn what computer games, their particular installments and DLCs

are the most commonly discussed and whether such games are referred rather positively or negatively.
There are several steps of natural language processing performed in the following steps:

- Dump,
- Filter,
- Tokenize,
- SentenceSplit,
- PartOfSpeech,
- Gender,
- Lemma,
- Ner,
- Parse,
- Sentiment,
- Index.

Dump is downloading data from prepared twitter dumps and emits them to the Filter. These dumps were taken from Twitter Streaming API and every record contains name of the author, timestamp and text of the particular tweet. Four hundred keywords most often occurring in the area of computer gaming were used to filter the tweets.

Filter ensures that the given tweet is really from area of computer gaming with the help of regular expressions and retrieves expressions most probably referring to a computer game. Filtered data are forwarded to tokenizer and indexer.

Tokenize step takes the text of a tweet and outputs its tokens. The result is then sent to the sentence splitter.

SentenceSplit extracts a list of sentences used in particular tweet from list of tokens.

PartOfSpeech tags words (tokens) with a part-of-speech categories. After extracting a part-of-speech information about the tokens, we have enough information to proceed with further classification and extraction tasks in parallel. Therefore the data from this step is sent to *Gender*, *Lemma*, *Ner* and *Parse* processing steps.

Gender extractor uses predefined classifier and with certain probability extracts the gender of tweet's author. When classification is uncertain, no data is extracted. Result of this extraction will be stored in the index.

Lemma extracts keywords of a tweet. Good keywords from a tweet are extracted with the help of a part-of-speech information. Basic forms (lemmas) of nouns, verbs and foreign words are taken as keywords of the tweet. Extracted keywords will be indexed along with tweets.

Ner step extracts named entities (namely persons) from the tweet. Result is sent to the indexing too.

Parse step extracts parses of a sentence. This information is sent to sentiment analysis. The Parse step is necessary for further processing

Sentiment is the last classification step. It extracts sentiment information about the tweet, namely, whether it is rather positive, negative or just neutral. Results are sent to index too.

Index is the final step of the entire processing. This is the place where all the information about the tweet is being put together. When the natural language processing is completed, the tweet is indexed in *Apache Lucene*. As there can be multiple indexes, prospective query must use all these indexes and sort their results before retrieving desired information to a user.

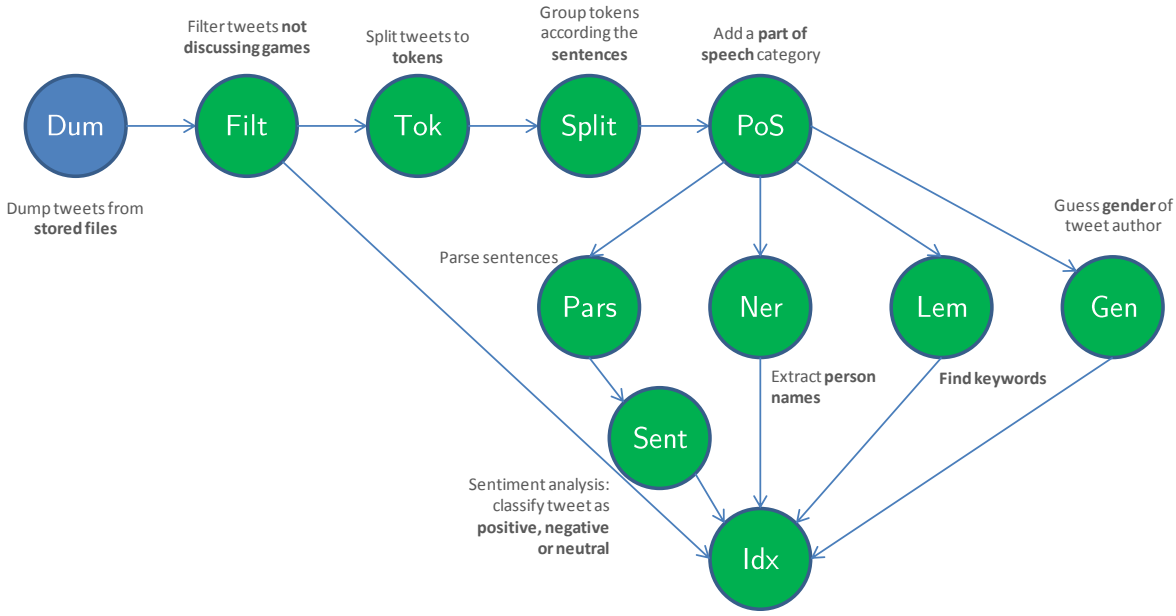

Figure 5: Topology of the BUT's experimental application for processing of the stream of tweets; each circle represents one component of the application; Dump (Dum), Filter (Filt), Tokenize (Tok), SentenceSplit (Split), PartOfSpeech (PoS), Gender (Gen), Lemma (Lem), Ner, Parse (Pars), Sentiment (Sent), Index (Idx)

4.2.1 Monitoring

To deploy the Twitter Use Case application for the first time, a default deployment plan produced by Juniper IDE tools has been used. The deployment plan defines four instances per each of programs, namely *Dump*, *Filter*, *Gender*, *Index*, *Lemma*, *Ner*, *Parse*, *PartOfSpeech*, *SentenceSplit*, *Sentiment*, and *Tokenize* programs. The instances are deployed on three infrastructure nodes: node 1 (147.229.8.104), node 2 (147.229.8.105), and node 3 (147.229.8.106), in such way that all instances of *Dump*, *Filter*, *Gender*, *Index*, *Lemma*, and *Ner* programs run on node 1, all instances of *Parse* and *PartOfSpeech* programs run on node 2, and all instances of *SentenceSplit*, *Sentiment*, and *Tokenize* programs run on node 3 of the infrastructure. An XML document with the deployment plan is showed in AppendixC.

For each program and each of its instances, the same set of measurements has been performed to obtain a monitoring data for further analysis—the same as was performed in the case of the petaFuel use case application described in Section 4.1.1. Analogously, the application has been run for different message sizes, namely for the message size of 1 record, of 10 records, of 100 records, and of

1000 records from original data set and the monitoring data have been gathered in each case, as was described in Section 4.1.1.

4.2.2 Analysis

The subsequent analysis of the monitoring data obtained has been performed by the advisor component of the Scheduling Advisor utilizing the following plugins:

- AdvisorDataTransferOverhead,
- AdvisorOutOfMemoryPrediction,
- AdvisorGarbageCollectionPerformance.

The full output of the analysis is attached in AppendixD. The results obtained indicate several issues that are described and discussed in the following paragraphs.

Big data transfer overhead in the cases of small message sizes The execution times of the whole application with different message sizes set indicated that, in this case, the message size has not significant impact on an overall performance of the application (contrary to the case of the petaFuel use case analyzed in Section 4.1.2). More specifically, the duration of the processing of the same data set has been 723 seconds, 577 seconds, 602 seconds, and 593 seconds, for message sizes 1, 10, 100, and 1000 records, respectively. The execution times were nearly the same for all message sizes. The only exception is the message size of 1 record, which was the slowest in execution (note that the execution with message size of 1 record was the slowest one also in the other use case analyzed in Section 4.1.2). However, for further analyses, we select the message size of 10 as it has the shorter execution time than a message size of 1 record.

The *AdvisorDataTransferOverhead* plugin indicated that in programs *Index*, *Gender*, *Lemma*, *SentenceSplit*, *Sentiment*, and *Tokenize*, the duration of their communication (i.e., data transfers between communicating programs) took nearly 99 % of the whole execution time of these programs. The reasons are different for program *Index* and for the rest of programs listed above. The *Index* program is receiving data from five directions and merging them to create an index of annotated twitter messages, so it performs five times more communication than other programs[6] (that takes much more time than the indexing itself which follows the receiving). Contrary to that, other programs, namely *Gender*, *Lemma*, *SentenceSplit*, *Sentiment*, and *Tokenize*, receive one message per execution and perform very simple or fast algorithms, so the communication took much more time than the algorithms that process the data (for example, the *SentenceSplit* program just splits a sentences to its words). The programs not listed above, especially, the *Parse* and *Filter* programs, perform complicated and time-consuming algorithms per a received message, so their communication to execution times ratio is lower (about 38 % in the case of the *Parse* program and about 10 % in the case of the *Filter* program). For example, an instance of the *Index* program and an instance of the *Sentiment* program were analyzed as follows:

[6]For the context of the *Index* program in the architecture, see Figure 5.

```
Advice DataTransferOverhead: The instance with global rank 13 of Juniper
   program 'Index' running at cloud node with host name/IP address
   147.229.8.104 was receiving data in 518,957000 seconds of 519,204000
   seconds of its total execution time (averages are 0,036064 seconds
   for 14390 receives of data and 0,072062 seconds for 7205 executions).
   That makes 99,952427 percentage of execution time spent by receiving
   data.

Advice DataTransferOverhead: The instance with global rank 38 of Juniper
   program 'Sentiment' running at cloud node with host name/IP address
   147.229.8.106 was receiving data in 534,994000 seconds of 541,245000
   seconds of its total execution time (averages are 0,181724 seconds
   for 2944 receives of data and 0,367195 seconds for 1474 executions).
   That makes 98,845070 percentage of execution time spent by receiving
   data
```

Finally, results of the analysis performed by the *AdvisorDataTransferOverhead* plugin indicate that the *Index* program performed about 14000 receive operations per instance while other programs performed about 3000 receives per instance (the average number of receives per instance, see the example above). Moreover, as the communication took 99 % of the execution time in the *Index* program instances, five times more receives in the *Index* instances make these instances run about 3-4 times longer than others and slow down the whole application (other program instances must wait for the *Index* instances). Therefore, as a part of the experiment, we modified the deployment plan to increase the *Index* program instances from 4 to 12 and to decrease the number of instances of programs *Gender*, *Lemma*, *SentenceSplit*, *Sentiment*, and *Tokenize* that were quite fast. This modification of the deployment plan resulted in decrease of the execution time of the whole application in total of 25 % (processing the same amount of data took 577 seconds before and 437 seconds after the modification). After the modification, analysis of the *Index* program instance (the same instance as in the example above) showed the reasonable improvement in the average number of receives per instance:

```
Advice DataTransferOverhead: The instance with global rank 13 of Juniper
   program 'Index' running at cloud node with host name/IP address
   147.229.8.104 was receiving data in 398,307000 seconds of 398,573000
   seconds of its total execution time (averages are 0,084138 seconds
   for 4734 receives of data and 0,167891 seconds for 2374 executions).
   That makes 99,933262 percentage of execution time spent by receiving
   data.
```

Too frequent garbage collections The most frequent garbage collecting was performed during executions of the *PartOfSpeech* program where the garbage collection took also the longest time. More specifically, there were 2 garbage collections performed in about 2 seconds that is about 8 % of a total execution time of the program instances. The reason lays again is many short-lived object created in the *PartOfSpeech* program to extract the part-of-speech information about each of processed tokens. The similar situation, that is creating of many short-lived object when processing many tokens, was detected in executions of the *Tokenize* program where 1 garbage collecting was performed in each of the executions which took 0.4 seconds. However, in both cases, the frequency

of and the time spent by the garbage collecting are reasonable and the programs do not need to be fixed. For example, the *PartOfSpeech* program with rank 28 was analyzed as follows:

```
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance
   with global rank 28 of Juniper program 'PartOfSpeech' running at
   cloud node with host name/IP address 147.229.8.105 was performed 2
   garbage collections that took 2,476000 seconds in 30,025000 seconds
   of total execution time of the program (averages are 1,238000 seconds
   per garbage collection and 0,020650 seconds for the execution time).
   That makes 8,246461 percentage of execution time spent by garbage
   collections.
```

Out-of-memory prediction In the case of the out-of-memory prediction, the most critical instances were those of programs *Filter* and *SentenceSplit* that allocated their heap memory by approximately 25 megabytes per second and 35 megabytes per second, respectively. Such fast allocation would cause the out-of-memory exceptions in about 15 minutes from the start of the application. However, a risk of the out-of-memory exceptions is quite low in this case as these memory allocations are usually very short. Both programs allocate memory quickly at the beginning of their executions and then they perform long computation without any additional massive allocations, in the case of the *Filter* program, or perform a simple computation and exits with freeing all the memory, in the case of the *SentenceSplit* program. For example, the plain-text output for one of the *Filter* program instances is the following:

```
Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 4
   of Juniper program 'Filter' running at cloud node with host name/IP
   address 147.229.8.104 has the memory usage growing over the time by
   the approximate rate of change 25191512,187807 Bytes per second for
   the heap memory size (the recommended maximum is 0,100000 Bytes per
   second). This may result into OutOfMemoryError errors in the program
   on 2015-05-26 11:50:45.636 (that is 00:18:28.124 since the beginning
   of analyzed data on 2015-05-26 11:32:17.512; the heap memory is
   limited to 28631367680 Bytes). The sample linear regression model is
   Y_{size_in_bytes} = 716025141,730374 + 25191512,187807 *
   X_{time_in_sec}.

Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global
   rank 4 of Juniper program 'Filter' running at cloud node with host
   name/IP address 147.229.8.104 has the memory usage growing over the
   time by the approximate rate of change 108748,469364 Bytes per second
   for the non-heap memory size (the recommended maximum is 0,100000
   Bytes per second). This may result into OutOfMemoryError errors in
   the program. The sample linear regression model is Y_{size_in_bytes}
   = 30381994,359782 + 108748,469364 * X_{time_in_sec}.
```

4.2.3 Conclusion

For the Twitter Use Case application, the *advisor component* of the Scheduling Advisor indicated several potential issues that should be fixed by its developers. Namely, the following remedies should be applied:

- Number of the *Index* program instances should be increased to decrease the number of receives per instance. This optimization resulted in decreasing the execution time of the whole application of 25 %.
- The *PartOfSpeech* program should be optimized with focus on memory management to limit the frequency of automatic garbage collection, which may cause the performance issues in real-time processing.
- Amount of data processed by the *Filter* and *SentenceSplit* programs should be limited to keep their executions short, otherwise they can cause out-of-memory Java exceptions.

5 Summary and Conclusions

Performance analysis and optimization of a distributed stream processing application running on the JUNIPER platform is not a simple task. The Scheduling Advisor aims at helping with such analyzes and optimization by better program placement based on resource to program suitability, by the *heterogeneity aware scheduler component*, and at collecting and analyzing monitoring data to detect potential issues, by the *advisor component*. As the heterogeneity aware scheduler component was already described in Deliverable 3.8 [1], this document focuses mainly on the *advisor component* and the integration of the Scheduling Advisor into the JUNIPER platform.

We described the architecture, inputs and outputs, and experimental results of the advisor component. The experiments were performed on two experimental applications and in both cases, the advisor component revealed significant issues. In current version of the advisor component, we focused on detection of three common issues, namely the data transfer overhead, the out of memory prediction, and the garbage collection frequency and duration. All of these issues may significantly affect the overall performance of JUNIPER applications, especially in the cases of continuous stream processing. Following the problems detected and their descriptions provided by the advisor component, we enhanced the performance of the petaFuel use case application by 119 % and the BUT's experimental Twitter processing application by 40 %.

While the first evaluation criterion "*To improve the performance of distributed stream processing applications on hardware heterogeneous clusters.*" was already satisfied in Deliverable 3.8 [1], considering the improvement in the tested applications, we can summarize that the second evaluation criterion for the Scheduling Advisor: "*To improve the basic monitoring of the distributed stream processing applications, to detect common problems of distributed applications, and to help solving these problems.*" was successfully satisfied by the experiments taken on our two test applications.

The advisor component can be easily extended by plugins to detect another types of issues in analyzed JUNIPER applications. We expect to further extend the advisor during its continuous integration into the JUNIPER project (and its application on another JUNIPER use cases) as various types of issues will be emerging in the future.

References

[1] BUT. First prototype of the real-time scheduling advisor. Deliverable 3.8, 2014.

[2] SOFTEAM. Final integrated MDE environment. Deliverable 5.6, 2015.

[3] SOFTEAM. Specification of model transformation chain. Deliverable 5.4, 2015.

A Deployment Plan for the Financial Use Case (petaFuel)

The following deployment plan was used to deploy the Financial Use Case application of petaFuel as it is described in Section 4.1.1.

```
<?xml version="1.0"?>
<application name="BUT_Streams">
  <ProgramModel>
    <program javaclass="cz.vutbr.fit.Write" name="Write"/>
    <program javaclass="cz.vutbr.fit.Storage" name="Storage"/>
    <program javaclass="cz.vutbr.fit.Unify" name="Unify"/>
    <program javaclass="cz.vutbr.fit.Join" name="Join"/>
  </ProgramModel>
  <GroupModel>
    <mpigroup name="group_node_Storage">
      <member mpiglobalrank="0" mpilocalrank="0" programName="Storage"/>
      <member mpiglobalrank="1" mpilocalrank="1" programName="Storage"/>
      <member mpiglobalrank="2" mpilocalrank="2" programName="Storage"/>
      <member mpiglobalrank="3" mpilocalrank="3" programName="Storage"/>
    </mpigroup>
    <mpigroup name="group_node_Join">
      <member mpiglobalrank="4" mpilocalrank="0" programName="Join"/>
      <member mpiglobalrank="5" mpilocalrank="1" programName="Join"/>
      <member mpiglobalrank="6" mpilocalrank="2" programName="Join"/>
      <member mpiglobalrank="7" mpilocalrank="3" programName="Join"/>
    </mpigroup>
    <mpigroup name="group_node_Unify">
      <member mpiglobalrank="8" mpilocalrank="0" programName="Unify"/>
      <member mpiglobalrank="9" mpilocalrank="1" programName="Unify"/>
      <member mpiglobalrank="10" mpilocalrank="2" programName="Unify"/>
      <member mpiglobalrank="11" mpilocalrank="3" programName="Unify"/>
    </mpigroup>
    <mpigroup name="group_node_Write">
      <member mpiglobalrank="12" mpilocalrank="0" programName="Write"/>
    </mpigroup>
  </GroupModel>
  <CommunicationModel>
    <dataconnection name="connection_group_node_Unify_group_node_Write"
        receiverMpiGroup="group_node_Write" sendingGroup="group_node_Unify" type="symmetric"/>
    <dataconnection name="connection_group_node_Join_group_node_Write"
        receiverMpiGroup="group_node_Write" sendingGroup="group_node_Join" type="symmetric"/>
    <dataconnection name="connection_group_node_Storage_group_node_Unify"
        receiverMpiGroup="group_node_Unify" sendingGroup="group_node_Storage" type="symmetric"/>
    <dataconnection name="connection_group_node_Storage_group_node_Join"
        receiverMpiGroup="group_node_Join" sendingGroup="group_node_Storage" type="symmetric"/>
  </CommunicationModel>
  <DeploymentModel>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="0"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="1"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="2"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="3"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="4"/>
    <cloudnode hostipaddr="147.229.8.105" mpiglobalrank="5"/>
    <cloudnode hostipaddr="147.229.8.105" mpiglobalrank="6"/>
    <cloudnode hostipaddr="147.229.8.105" mpiglobalrank="7"/>
    <cloudnode hostipaddr="147.229.8.105" mpiglobalrank="8"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="9"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="10"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="11"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="12"/>
  </DeploymentModel>
</application>
```

B Results of the Analysis for the Financial Use Case (petaFuel)

While the execution and monitoring of the petaFuel application were performed several times with different sizes of messages for individual processing and communication, the most important results are those with message size of 1000 records (other sizes have been used just for detailed analysis of data transfer overhead as described in Section4.1.2). The output of the advisor component in a human-readable plain-text is the following:

```
*** processing deployment plan deployment_plan_athena4_pcknot_pcmaliulin.xml
*** openning/creating JDBC database for monitoring results cache
    jdbc:h2:/tmp/eu.juniper.sa.tool.Advisor.e5711e04-9aac-4b13-b5d8-5e7fa9304bd7;COMPRESS=TRUE
*** importing metrics from petafuelrun1.merged.1000.gz
*** loading and executing plugins from package eu.juniper.sa.tool.plugins
*** loading and setting advisor plugin
    eu.juniper.sa.tool.plugins.AdvisorGarbageCollectionPerformance

*** executing advisor plugin AdvisorGarbageCollectionPerformance with the following description:
This advisor detects Juniper programs that spent much time on garbage collecting. Long garbage
    collections may affect negatively responsiveness of a Juniper application which is critical in
    real-time stream processing of Big Data (any delay in processing of such data may result into
    data-loss issues).

Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 12 of Juniper
    program 'Write' running at cloud node with host name/IP address 147.229.12.172 was performed
    639 garbage collections that took 1,430000 seconds in 177,956000 seconds of total execution
    time of the program (averages are 0,002238 seconds per garbage collection and 0,015694 seconds
    for the execution time). That makes 0,803569 percentage of execution time spent by garbage
    collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 9 of Juniper
    program 'Unify' running at cloud node with host name/IP address 147.229.12.172 was performed 30
    garbage collections that took 0,099000 seconds in 42,046000 seconds of total execution time of
    the program (averages are 0,003300 seconds per garbage collection and 0,040821 seconds for the
    execution time). That makes 0,235456 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 11 of Juniper
    program 'Unify' running at cloud node with host name/IP address 147.229.12.172 was performed 32
    garbage collections that took 0,121000 seconds in 63,539000 seconds of total execution time of
    the program (averages are 0,003781 seconds per garbage collection and 0,060170 seconds for the
    execution time). That makes 0,190434 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 7 of Juniper
    program 'Join' running at cloud node with host name/IP address 147.229.12.153 was performed 17
    garbage collections that took 0,298000 seconds in 167,665000 seconds of total execution time of
    the program (averages are 0,017529 seconds per garbage collection and 0,050320 seconds for the
    execution time). That makes 0,177735 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 6 of Juniper
    program 'Join' running at cloud node with host name/IP address 147.229.12.153 was performed 16
    garbage collections that took 0,282000 seconds in 167,154000 seconds of total execution time of
    the program (averages are 0,017625 seconds per garbage collection and 0,050166 seconds for the
    execution time). That makes 0,168707 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 10 of Juniper
    program 'Unify' running at cloud node with host name/IP address 147.229.12.172 was performed 31
    garbage collections that took 0,076000 seconds in 49,503000 seconds of total execution time of
    the program (averages are 0,002452 seconds per garbage collection and 0,046569 seconds for the
    execution time). That makes 0,153526 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 5 of Juniper
    program 'Join' running at cloud node with host name/IP address 147.229.12.153 was performed 16
    garbage collections that took 0,239000 seconds in 167,702000 seconds of total execution time of
    the program (averages are 0,014938 seconds per garbage collection and 0,050331 seconds for the
    execution time). That makes 0,142515 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 8 of Juniper
    program 'Unify' running at cloud node with host name/IP address 147.229.12.153 was performed 29
    garbage collections that took 0,059000 seconds in 48,016000 seconds of total execution time of
    the program (averages are 0,002034 seconds per garbage collection and 0,045170 seconds for the
    execution time). That makes 0,122876 percentage of execution time spent by garbage collections.
```

```
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 3 of Juniper
    program 'Storage' running at cloud node with host name/IP address 147.229.8.104 was performed 9
    garbage collections that took 0,054000 seconds in 218,757000 seconds of total execution time of
    the program (averages are 0,006000 seconds per garbage collection and 0,116298 seconds for the
    execution time). That makes 0,024685 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 2 of Juniper
    program 'Storage' running at cloud node with host name/IP address 147.229.8.104 was performed 9
    garbage collections that took 0,048000 seconds in 220,751000 seconds of total execution time of
    the program (averages are 0,005333 seconds per garbage collection and 0,117358 seconds for the
    execution time). That makes 0,021744 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 0 of Juniper
    program 'Storage' running at cloud node with host name/IP address 147.229.8.104 was performed 9
    garbage collections that took 0,047000 seconds in 224,963000 seconds of total execution time of
    the program (averages are 0,005222 seconds per garbage collection and 0,119598 seconds for the
    execution time). That makes 0,020892 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 1 of Juniper
    program 'Storage' running at cloud node with host name/IP address 147.229.8.104 was performed 9
    garbage collections that took 0,043000 seconds in 216,380000 seconds of total execution time of
    the program (averages are 0,004778 seconds per garbage collection and 0,115035 seconds for the
    execution time). That makes 0,019872 percentage of execution time spent by garbage collections.
*** loading and setting advisor plugin eu.juniper.sa.tool.plugins.AdvisorOutOfMemoryPrediction

*** executing advisor plugin AdvisorOutOfMemoryPrediction with the following description:
This advisor detect Juniper programs where the memory usage is growing over the time by detecting a
    linear trend in memory usage (the linear regression analysis). This may indicate potential
    memory leaks and eventually result into performance related issues and OutOfMemoryError errors
    in a Juniper programs.

Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 4 of Juniper program 'Join'
    running at cloud node with host name/IP address 147.229.8.104 has the memory usage growing over
    the time by the approximate rate of change 697664,267920 Bytes per second for the heap memory
    size (the recommended maximum is 0,100000 Bytes per second). This may result into
    OutOfMemoryError errors in the program on 2015-05-23 00:42:17.036 (that is 11:19:42.531 since
    the beginning of analyzed data on 2015-05-22 13:22:34.505; the heap memory is limited to
    28631367680 Bytes). The sample linear regression model is Y_{size_in_bytes} = 178852382,676363
    + 697664,267920 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 4 of Juniper program
    'Join' running at cloud node with host name/IP address 147.229.8.104 has the memory usage
    growing over the time by the approximate rate of change 13373,765433 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 15816901,638517 + 13373,765433 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 6 of Juniper program 'Join'
    running at cloud node with host name/IP address 147.229.12.153 has the memory usage growing
    over the time by the approximate rate of change 877058,223252 Bytes per second for the heap
    memory size (the recommended maximum is 0,100000 Bytes per second). This may result into
    OutOfMemoryError errors in the program on 2015-05-22 14:34:38.108 (that is 01:12:03.603 since
    the beginning of analyzed data on 2015-05-22 13:22:34.505; the heap memory is limited to
    3737649152 Bytes). The sample linear regression model is Y_{size_in_bytes} = -54402824,344177 +
    877058,223252 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 6 of Juniper program
    'Join' running at cloud node with host name/IP address 147.229.12.153 has the memory usage
    growing over the time by the approximate rate of change 12261,104566 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 13918176,331781 + 12261,104566 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 3 of Juniper program
    'Storage' running at cloud node with host name/IP address 147.229.8.104 has the memory usage
    growing over the time by the approximate rate of change 10676,523368 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 16512877,856674 + 10676,523368 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 7 of Juniper program 'Join'
    running at cloud node with host name/IP address 147.229.12.153 has the memory usage growing
    over the time by the approximate rate of change 859929,198753 Bytes per second for the heap
    memory size (the recommended maximum is 0,100000 Bytes per second). This may result into
    OutOfMemoryError errors in the program on 2015-05-22 14:36:01.783 (that is 01:13:27.278 since
```

```
    the beginning of analyzed data on 2015-05-22 13:22:34.505; the heap memory is limited to
    3737649152 Bytes). The sample linear regression model is Y_{size_in_bytes} = -52298136,130751 +
    859929,198753 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 7 of Juniper program
    'Join' running at cloud node with host name/IP address 147.229.12.153 has the memory usage
    growing over the time by the approximate rate of change 11692,556351 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 14074420,281653 + 11692,556351 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 2 of Juniper program
    'Storage' running at cloud node with host name/IP address 147.229.8.104 has the memory usage
    growing over the time by the approximate rate of change 10814,145565 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 17013845,638706 + 10814,145565 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 10 of Juniper program
    'Unify' running at cloud node with host name/IP address 147.229.12.172 has the memory usage
    growing over the time by the approximate rate of change 8292,241152 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 13893891,065452 + 8292,241152 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 11 of Juniper program
    'Unify' running at cloud node with host name/IP address 147.229.12.172 has the memory usage
    growing over the time by the approximate rate of change 8077,788977 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 13917082,193309 + 8077,788977 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 8 of Juniper program
    'Unify' running at cloud node with host name/IP address 147.229.12.153 has the memory usage
    growing over the time by the approximate rate of change 8749,749953 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 13897840,839726 + 8749,749953 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 1 of Juniper program
    'Storage' running at cloud node with host name/IP address 147.229.8.104 has the memory usage
    growing over the time by the approximate rate of change 14467,076909 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 16248031,626854 + 14467,076909 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 9 of Juniper program
    'Unify' running at cloud node with host name/IP address 147.229.12.172 has the memory usage
    growing over the time by the approximate rate of change 7943,309935 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 13918832,349910 + 7943,309935 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 0 of Juniper program
    'Storage' running at cloud node with host name/IP address 147.229.8.104 has the memory usage
    growing over the time by the approximate rate of change 9960,684255 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 16825373,048864 + 9960,684255 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 5 of Juniper program 'Join'
    running at cloud node with host name/IP address 147.229.12.153 has the memory usage growing
    over the time by the approximate rate of change 724393,291070 Bytes per second for the heap
    memory size (the recommended maximum is 0,100000 Bytes per second). This may result into
    OutOfMemoryError errors in the program on 2015-05-22 14:49:25.042 (that is 01:26:50.537 since
    the beginning of analyzed data on 2015-05-22 13:22:34.505; the heap memory is limited to
    3737649152 Bytes). The sample linear regression model is Y_{size_in_bytes} = -36829269,777905 +
    724393,291070 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 5 of Juniper program
    'Join' running at cloud node with host name/IP address 147.229.12.153 has the memory usage
    growing over the time by the approximate rate of change 12043,446667 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 13958321,580510 + 12043,446667 * X_{time_in_sec}.
*** loading and setting advisor plugin eu.juniper.sa.tool.plugins.AdvisorDataTransferOverhead
```

```
*** executing advisor plugin AdvisorDataTransferOverhead with the following description:
This advisor detects Juniper programs with long data communication times (i.e., a time spent on
    waiting for receiving data) and short computation. This indicate a very simple program with a
    high data transfer overhead (it should be merged with other programs) or a program waiting for
    data most the time (data flows/stream should be optimized in the Juniper application of the
    Juniper program).

Advice DataTransferOverhead: The instance with global rank 5 of Juniper program 'Join' running at
    cloud node with host name/IP address 147.229.12.153 was receiving data in 167,285000 seconds of
    167,702000 seconds of its total execution time (averages are 0,025118 seconds for 6660 receives
    of data and 0,050331 seconds for 3332 executions). That makes 99,751345 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 6 of Juniper program 'Join' running at
    cloud node with host name/IP address 147.229.12.153 was receiving data in 166,731000 seconds of
    167,154000 seconds of its total execution time (averages are 0,025035 seconds for 6660 receives
    of data and 0,050166 seconds for 3332 executions). That makes 99,746940 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 4 of Juniper program 'Join' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 185,492000 seconds of
    185,984000 seconds of its total execution time (averages are 0,027852 seconds for 6660 receives
    of data and 0,055818 seconds for 3332 executions). That makes 99,735461 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 7 of Juniper program 'Join' running at
    cloud node with host name/IP address 147.229.12.153 was receiving data in 167,157000 seconds of
    167,665000 seconds of its total execution time (averages are 0,025099 seconds for 6660 receives
    of data and 0,050320 seconds for 3332 executions). That makes 99,697015 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 8 of Juniper program 'Unify' running at
    cloud node with host name/IP address 147.229.12.153 was receiving data in 47,593000 seconds of
    48,016000 seconds of its total execution time (averages are 0,022428 seconds for 2122 receives
    of data and 0,045170 seconds for 1063 executions). That makes 99,119044 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 11 of Juniper program 'Unify' running at
    cloud node with host name/IP address 147.229.12.172 was receiving data in 62,813000 seconds of
    63,539000 seconds of its total execution time (averages are 0,029797 seconds for 2108 receives
    of data and 0,060170 seconds for 1056 executions). That makes 98,857395 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 10 of Juniper program 'Unify' running at
    cloud node with host name/IP address 147.229.12.172 was receiving data in 48,807000 seconds of
    49,503000 seconds of its total execution time (averages are 0,023000 seconds for 2122 receives
    of data and 0,046569 seconds for 1063 executions). That makes 98,594025 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 9 of Juniper program 'Unify' running at
    cloud node with host name/IP address 147.229.12.172 was receiving data in 41,367000 seconds of
    42,046000 seconds of its total execution time (averages are 0,020120 seconds for 2056 receives
    of data and 0,040821 seconds for 1030 executions). That makes 98,385102 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 12 of Juniper program 'Write' running at
    cloud node with host name/IP address 147.229.12.172 was receiving data in 21,356000 seconds of
    177,956000 seconds of its total execution time (averages are 0,000942 seconds for 22672
    receives of data and 0,015694 seconds for 11339 executions). That makes 12,000719 percentage of
    execution time spent by receiving data.

*** writing the list of advice into XML file petafuelrun1.merged.1000.advice.xml
*** removing database files with monitoring results cache
```

C Deployment Plan for the Twitter Demonstration Use Case (BUT)

The following deployment plan was used to deploy the Twitter Demonstration Use Case of BUT as it is described in Section 4.2.1.

```
<?xml version="1.0"?>
<application name="BUT_Streams">
  <ProgramModel>
    <program javaclass="cz.vutbr.fit.source.Dump" name="Dump"/>
    <program javaclass="cz.vutbr.fit.process.Filter" name="Filter"/>
    <program javaclass="cz.vutbr.fit..process.Tokenize" name="Tokenize"/>
    <program javaclass="cz.vutbr.fit.process.SentenceSplitter" name="SentenceSplit"/>
    <program javaclass="cz.vutbr.fit.process.POSTagger" name="PartOfSpeech"/>
    <program javaclass="cz.vutbr.fit.process.Gender" name="Gender"/>
    <program javaclass="cz.vutbr.fit.process.Lemma" name="Lemma"/>
    <program javaclass="cz.vutbr.fit.process.Ner" name="Ner"/>
    <program javaclass="cz.vutbr.fit.process.Parser" name="Parse"/>
    <program javaclass="cz.vutbr.fit.process.Sentiment" name="Sentiment"/>
    <program javaclass="cz.vutbr.fit.process.Index" name="Index"/>
  </ProgramModel>
  <GroupModel>
    <mpigroup name="group_node_Dump">
      <member mpiglobalrank="0" mpilocalrank="0" programName="Dump"/>
      <member mpiglobalrank="1" mpilocalrank="1" programName="Dump"/>
      <member mpiglobalrank="2" mpilocalrank="2" programName="Dump"/>
      <member mpiglobalrank="3" mpilocalrank="3" programName="Dump"/>
    </mpigroup>
    <mpigroup name="group_node_Filter">
      <member mpiglobalrank="4" mpilocalrank="0" programName="Filter"/>
      <member mpiglobalrank="5" mpilocalrank="1" programName="Filter"/>
      <member mpiglobalrank="6" mpilocalrank="2" programName="Filter"/>
      <member mpiglobalrank="7" mpilocalrank="3" programName="Filter"/>
    </mpigroup>
    <mpigroup name="group_node_Gender">
      <member mpiglobalrank="8" mpilocalrank="0" programName="Gender"/>
      <member mpiglobalrank="9" mpilocalrank="1" programName="Gender"/>
      <member mpiglobalrank="10" mpilocalrank="2" programName="Gender"/>
      <member mpiglobalrank="11" mpilocalrank="3" programName="Gender"/>
    </mpigroup>
    <mpigroup name="group_node_Index">
      <member mpiglobalrank="12" mpilocalrank="0" programName="Index"/>
      <member mpiglobalrank="13" mpilocalrank="1" programName="Index"/>
      <member mpiglobalrank="14" mpilocalrank="2" programName="Index"/>
      <member mpiglobalrank="15" mpilocalrank="3" programName="Index"/>
    </mpigroup>
    <mpigroup name="group_node_Lemma">
      <member mpiglobalrank="16" mpilocalrank="0" programName="Lemma"/>
      <member mpiglobalrank="17" mpilocalrank="1" programName="Lemma"/>
      <member mpiglobalrank="18" mpilocalrank="2" programName="Lemma"/>
      <member mpiglobalrank="19" mpilocalrank="3" programName="Lemma"/>
    </mpigroup>
    <mpigroup name="group_node_Ner">
      <member mpiglobalrank="20" mpilocalrank="0" programName="Ner"/>
      <member mpiglobalrank="21" mpilocalrank="1" programName="Ner"/>
      <member mpiglobalrank="22" mpilocalrank="2" programName="Ner"/>
      <member mpiglobalrank="23" mpilocalrank="3" programName="Ner"/>
    </mpigroup>
    <mpigroup name="group_node_Parse">
      <member mpiglobalrank="24" mpilocalrank="0" programName="Parse"/>
      <member mpiglobalrank="25" mpilocalrank="1" programName="Parse"/>
      <member mpiglobalrank="26" mpilocalrank="2" programName="Parse"/>
      <member mpiglobalrank="27" mpilocalrank="3" programName="Parse"/>
    </mpigroup>
    <mpigroup name="group_node_PartOfSpeech">
```

```
    <member mpiglobalrank="28" mpilocalrank="0" programName="PartOfSpeech"/>
    <member mpiglobalrank="29" mpilocalrank="1" programName="PartOfSpeech"/>
    <member mpiglobalrank="30" mpilocalrank="2" programName="PartOfSpeech"/>
    <member mpiglobalrank="31" mpilocalrank="3" programName="PartOfSpeech"/>
  </mpigroup>
  <mpigroup name="group_node_SentenceSplit">
    <member mpiglobalrank="32" mpilocalrank="0" programName="SentenceSplit"/>
    <member mpiglobalrank="33" mpilocalrank="1" programName="SentenceSplit"/>
    <member mpiglobalrank="34" mpilocalrank="2" programName="SentenceSplit"/>
    <member mpiglobalrank="35" mpilocalrank="3" programName="SentenceSplit"/>
  </mpigroup>
  <mpigroup name="group_node_Sentiment">
    <member mpiglobalrank="36" mpilocalrank="0" programName="Sentiment"/>
    <member mpiglobalrank="37" mpilocalrank="1" programName="Sentiment"/>
    <member mpiglobalrank="38" mpilocalrank="2" programName="Sentiment"/>
    <member mpiglobalrank="39" mpilocalrank="3" programName="Sentiment"/>
  </mpigroup>
  <mpigroup name="group_node_Tokenize">
    <member mpiglobalrank="40" mpilocalrank="0" programName="Tokenize"/>
    <member mpiglobalrank="41" mpilocalrank="1" programName="Tokenize"/>
    <member mpiglobalrank="42" mpilocalrank="2" programName="Tokenize"/>
    <member mpiglobalrank="43" mpilocalrank="3" programName="Tokenize"/>
  </mpigroup>
</GroupModel>
<CommunicationModel>
  <dataconnection name="connection_group_node_Dump_group_node_Filter"
      receiverMpiGroup="group_node_Filter" sendingGroup="group_node_Dump" type="symmetric"/>
  <dataconnection name="connection_group_node_Filter_group_node_Tokenize"
      receiverMpiGroup="group_node_Tokenize" sendingGroup="group_node_Filter" type="symmetric"/>
  <dataconnection name="connection_group_node_Filter_group_node_Index"
      receiverMpiGroup="group_node_Index" sendingGroup="group_node_Filter" type="symmetric"/>
  <dataconnection name="connection_group_node_Tokenize_group_node_SentenceSplit"
      receiverMpiGroup="group_node_SentenceSplit" sendingGroup="group_node_Tokenize"
      type="symmetric"/>
  <dataconnection name="connection_group_node_SentenceSplit_group_node_PartOfSpeech"
      receiverMpiGroup="group_node_PartOfSpeech" sendingGroup="group_node_SentenceSplit"
      type="symmetric"/>
  <dataconnection name="connection_group_node_PartOfSpeech_group_node_Gender"
      receiverMpiGroup="group_node_Gender" sendingGroup="group_node_PartOfSpeech"
      type="symmetric"/>
  <dataconnection name="connection_group_node_PartOfSpeech_group_node_Lemma"
      receiverMpiGroup="group_node_Lemma" sendingGroup="group_node_PartOfSpeech"
      type="symmetric"/>
  <dataconnection name="connection_group_node_PartOfSpeech_group_node_Ner"
      receiverMpiGroup="group_node_Ner" sendingGroup="group_node_PartOfSpeech" type="symmetric"/>
  <dataconnection name="connection_group_node_PartOfSpeech_group_node_Parse"
      receiverMpiGroup="group_node_Parse" sendingGroup="group_node_PartOfSpeech"
      type="symmetric"/>
  <dataconnection name="connection_group_node_Parse_group_node_Sentiment"
      receiverMpiGroup="group_node_Sentiment" sendingGroup="group_node_Parse" type="symmetric"/>
  <dataconnection name="connection_group_node_Sentiment_group_node_Index"
      receiverMpiGroup="group_node_Index" sendingGroup="group_node_Sentiment" type="symmetric"/>
  <dataconnection name="connection_group_node_Gender_group_node_Index"
      receiverMpiGroup="group_node_Index" sendingGroup="group_node_Gender" type="symmetric"/>
  <dataconnection name="connection_group_node_Lemma_group_node_Index"
      receiverMpiGroup="group_node_Index" sendingGroup="group_node_Lemma" type="symmetric"/>
  <dataconnection name="connection_group_node_Ner_group_node_Index"
      receiverMpiGroup="group_node_Index" sendingGroup="group_node_Ner" type="symmetric"/>
</CommunicationModel>
<DeploymentModel>
  <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="0"/>
  <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="1"/>
  <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="2"/>
  <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="3"/>
  <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="4"/>
  <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="5"/>
  <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="6"/>
```

```
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="7"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="8"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="9"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="10"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="11"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="12"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="13"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="14"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="15"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="16"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="17"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="18"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="19"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="20"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="21"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="22"/>
    <cloudnode hostipaddr="147.229.8.104" mpiglobalrank="23"/>
    <cloudnode hostipaddr="147.229.8.105" mpiglobalrank="24"/>
    <cloudnode hostipaddr="147.229.8.105" mpiglobalrank="25"/>
    <cloudnode hostipaddr="147.229.8.105" mpiglobalrank="26"/>
    <cloudnode hostipaddr="147.229.8.105" mpiglobalrank="27"/>
    <cloudnode hostipaddr="147.229.8.105" mpiglobalrank="28"/>
    <cloudnode hostipaddr="147.229.8.105" mpiglobalrank="29"/>
    <cloudnode hostipaddr="147.229.8.105" mpiglobalrank="30"/>
    <cloudnode hostipaddr="147.229.8.105" mpiglobalrank="31"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="32"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="33"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="34"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="35"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="36"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="37"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="38"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="39"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="40"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="41"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="42"/>
    <cloudnode hostipaddr="147.229.8.106" mpiglobalrank="43"/>
  </DeploymentModel>
</application>
```

D Results of the Analysis for the Twitter Demonstration Use Case (BUT)

While the execution and monitoring of the BUT application were performed several times with different sizes of data-sets for individual processing and communication, the most important results are those with data-set size set to 10 records (other sizes have been used just for detailed analysis of data transfer overhead as described in Section4.1.2). The output of the advisor component in a human-readable plain-text is the following:

```
*** processing deployment plan deployment_plan_network2.xml
*** openning/creating JDBC database for monitoring results cache
    jdbc:h2:/tmp/eu.juniper.sa.tool.Advisor.60714ad8-1e3d-4df4-a203-258c30bb7e42;COMPRESS=TRUE
*** importing metrics from 10.merged.gz
*** loading and executing plugins from package eu.juniper.sa.tool.plugins
*** loading and setting advisor plugin eu.juniper.sa.tool.plugins.AdvisorDataTransferOverhead

*** executing advisor plugin AdvisorDataTransferOverhead with the following description:
This advisor detects Juniper programs with long data communication times (i.e., a time spent on
    waiting for receiving data) and short computation. This indicate a very simple program with a
    high data transfer overhead (it should be merged with other programs) or a program waiting for
    data most the time (data flows/stream should be optimized in the Juniper application of the
    Juniper program).

Advice DataTransferOverhead: The instance with global rank 12 of Juniper program 'Index' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 519,064000 seconds of
    519,262000 seconds of its total execution time (averages are 0,036046 seconds for 14400
    receives of data and 0,072020 seconds for 7210 executions). That makes 99,961869 percentage of
    execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 15 of Juniper program 'Index' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 517,321000 seconds of
    517,550000 seconds of its total execution time (averages are 0,035073 seconds for 14750
    receives of data and 0,070081 seconds for 7385 executions). That makes 99,955753 percentage of
    execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 13 of Juniper program 'Index' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 518,957000 seconds of
    519,204000 seconds of its total execution time (averages are 0,036064 seconds for 14390
    receives of data and 0,072062 seconds for 7205 executions). That makes 99,952427 percentage of
    execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 10 of Juniper program 'Gender' running
    at cloud node with host name/IP address 147.229.8.104 was receiving data in 535,919000 seconds
    of 536,393000 seconds of its total execution time (averages are 0,189773 seconds for 2824
    receives of data and 0,379344 seconds for 1414 executions). That makes 99,911632 percentage of
    execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 8 of Juniper program 'Gender' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 534,242000 seconds of
    534,729001 seconds of its total execution time (averages are 0,183968 seconds for 2904 receives
    of data and 0,367764 seconds for 1454 executions). That makes 99,908926 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 14 of Juniper program 'Index' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 520,653000 seconds of
    521,142000 seconds of its total execution time (averages are 0,037376 seconds for 13930
    receives of data and 0,074716 seconds for 6975 executions). That makes 99,906168 percentage of
    execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 9 of Juniper program 'Gender' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 534,240000 seconds of
    534,745000 seconds of its total execution time (averages are 0,188644 seconds for 2832 receives
    of data and 0,377112 seconds for 1418 executions). That makes 99,905562 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 11 of Juniper program 'Gender' running
    at cloud node with host name/IP address 147.229.8.104 was receiving data in 533,656000 seconds
    of 534,168000 seconds of its total execution time (averages are 0,181887 seconds for 2934
    receives of data and 0,363627 seconds for 1469 executions). That makes 99,904150 percentage of
    execution time spent by receiving data.
```

```
Advice DataTransferOverhead: The instance with global rank 16 of Juniper program 'Lemma' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 531,409000 seconds of
    532,405000 seconds of its total execution time (averages are 0,184645 seconds for 2878 receives
    of data and 0,369469 seconds for 1441 executions). That makes 99,812924 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 18 of Juniper program 'Lemma' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 531,011000 seconds of
    532,104000 seconds of its total execution time (averages are 0,182478 seconds for 2910 receives
    of data and 0,365205 seconds for 1457 executions). That makes 99,794589 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 17 of Juniper program 'Lemma' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 533,786000 seconds of
    534,962001 seconds of its total execution time (averages are 0,181436 seconds for 2942 receives
    of data and 0,363179 seconds for 1473 executions). That makes 99,780171 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 19 of Juniper program 'Lemma' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 536,090000 seconds of
    537,329000 seconds of its total execution time (averages are 0,193954 seconds for 2764 receives
    of data and 0,388243 seconds for 1384 executions). That makes 99,769415 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 33 of Juniper program 'SentenceSplit'
    running at cloud node with host name/IP address 147.229.8.106 was receiving data in 27,088000
    seconds of 27,175000 seconds of its total execution time (averages are 0,009565 seconds for
    2832 receives of data and 0,019164 seconds for 1418 executions). That makes 99,679853
    percentage of execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 32 of Juniper program 'SentenceSplit'
    running at cloud node with host name/IP address 147.229.8.106 was receiving data in 26,831000
    seconds of 26,918000 seconds of its total execution time (averages are 0,009246 seconds for
    2902 receives of data and 0,018526 seconds for 1453 executions). That makes 99,676796
    percentage of execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 34 of Juniper program 'SentenceSplit'
    running at cloud node with host name/IP address 147.229.8.106 was receiving data in 26,550000
    seconds of 26,647000 seconds of its total execution time (averages are 0,008891 seconds for
    2986 receives of data and 0,017824 seconds for 1495 executions). That makes 99,635982
    percentage of execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 35 of Juniper program 'SentenceSplit'
    running at cloud node with host name/IP address 147.229.8.106 was receiving data in 27,146000
    seconds of 27,259000 seconds of its total execution time (averages are 0,009786 seconds for
    2774 receives of data and 0,019625 seconds for 1389 executions). That makes 99,585458
    percentage of execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 39 of Juniper program 'Sentiment'
    running at cloud node with host name/IP address 147.229.8.106 was receiving data in 536,156000
    seconds of 542,149999 seconds of its total execution time (averages are 0,188787 seconds for
    2840 receives of data and 0,381259 seconds for 1422 executions). That makes 98,894402
    percentage of execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 36 of Juniper program 'Sentiment'
    running at cloud node with host name/IP address 147.229.8.106 was receiving data in 534,664000
    seconds of 540,752996 seconds of its total execution time (averages are 0,183230 seconds for
    2918 receives of data and 0,370125 seconds for 1461 executions). That makes 98,873978
    percentage of execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 37 of Juniper program 'Sentiment'
    running at cloud node with host name/IP address 147.229.8.106 was receiving data in 535,603000
    seconds of 541,735000 seconds of its total execution time (averages are 0,191835 seconds for
    2792 receives of data and 0,387507 seconds for 1398 executions). That makes 98,868081
    percentage of execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 38 of Juniper program 'Sentiment'
    running at cloud node with host name/IP address 147.229.8.106 was receiving data in 534,994000
    seconds of 541,245000 seconds of its total execution time (averages are 0,181724 seconds for
    2944 receives of data and 0,367195 seconds for 1474 executions). That makes 98,845070
    percentage of execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 41 of Juniper program 'Tokenize' running
    at cloud node with host name/IP address 147.229.8.106 was receiving data in 21,318000 seconds
    of 21,719000 seconds of its total execution time (averages are 0,007496 seconds for 2844
    receives of data and 0,015252 seconds for 1424 executions). That makes 98,153690 percentage of
    execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 40 of Juniper program 'Tokenize' running
    at cloud node with host name/IP address 147.229.8.106 was receiving data in 20,722000 seconds
```

```
    of 21,128000 seconds of its total execution time (averages are 0,007205 seconds for 2876
    receives of data and 0,014672 seconds for 1440 executions). That makes 98,078379 percentage of
    execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 42 of Juniper program 'Tokenize' running
    at cloud node with host name/IP address 147.229.8.106 was receiving data in 20,796000 seconds
    of 21,580000 seconds of its total execution time (averages are 0,007276 seconds for 2858
    receives of data and 0,015080 seconds for 1431 executions). That makes 96,367006 percentage of
    execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 43 of Juniper program 'Tokenize' running
    at cloud node with host name/IP address 147.229.8.106 was receiving data in 21,208000 seconds
    of 22,024000 seconds of its total execution time (averages are 0,007273 seconds for 2916
    receives of data and 0,015085 seconds for 1460 executions). That makes 96,294951 percentage of
    execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 20 of Juniper program 'Ner' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 467,125000 seconds of
    538,643000 seconds of its total execution time (averages are 0,167069 seconds for 2796 receives
    of data and 0,384745 seconds for 1400 executions). That makes 86,722560 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 21 of Juniper program 'Ner' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 461,590000 seconds of
    536,200000 seconds of its total execution time (averages are 0,161057 seconds for 2866 receives
    of data and 0,373659 seconds for 1435 executions). That makes 86,085416 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 22 of Juniper program 'Ner' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 461,479000 seconds of
    536,980000 seconds of its total execution time (averages are 0,157609 seconds for 2928 receives
    of data and 0,366289 seconds for 1466 executions). That makes 85,939700 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 23 of Juniper program 'Ner' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 459,796000 seconds of
    537,972000 seconds of its total execution time (averages are 0,158332 seconds for 2904 receives
    of data and 0,369994 seconds for 1454 executions). That makes 85,468389 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 31 of Juniper program 'PartOfSpeech'
    running at cloud node with host name/IP address 147.229.8.105 was receiving data in 15,289000
    seconds of 26,939000 seconds of its total execution time (averages are 0,005338 seconds for
    2864 receives of data and 0,018786 seconds for 1434 executions). That makes 56,754148
    percentage of execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 29 of Juniper program 'PartOfSpeech'
    running at cloud node with host name/IP address 147.229.8.105 was receiving data in 15,119000
    seconds of 27,031000 seconds of its total execution time (averages are 0,005242 seconds for
    2884 receives of data and 0,018720 seconds for 1444 executions). That makes 55,932078
    percentage of execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 30 of Juniper program 'PartOfSpeech'
    running at cloud node with host name/IP address 147.229.8.105 was receiving data in 15,531000
    seconds of 29,345000 seconds of its total execution time (averages are 0,005465 seconds for
    2842 receives of data and 0,020622 seconds for 1423 executions). That makes 52,925541
    percentage of execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 28 of Juniper program 'PartOfSpeech'
    running at cloud node with host name/IP address 147.229.8.105 was receiving data in 15,054000
    seconds of 30,025000 seconds of its total execution time (averages are 0,005184 seconds for
    2904 receives of data and 0,020650 seconds for 1454 executions). That makes 50,138218
    percentage of execution time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 27 of Juniper program 'Parse' running at
    cloud node with host name/IP address 147.229.8.105 was receiving data in 221,001000 seconds of
    540,289999 seconds of its total execution time (averages are 0,076683 seconds for 2882 receives
    of data and 0,374421 seconds for 1443 executions). That makes 40,904144 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 25 of Juniper program 'Parse' running at
    cloud node with host name/IP address 147.229.8.105 was receiving data in 215,594000 seconds of
    542,647000 seconds of its total execution time (averages are 0,078057 seconds for 2762 receives
    of data and 0,392369 seconds for 1383 executions). That makes 39,730064 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 24 of Juniper program 'Parse' running at
    cloud node with host name/IP address 147.229.8.105 was receiving data in 200,292000 seconds of
    541,286000 seconds of its total execution time (averages are 0,068359 seconds for 2930 receives
```

```
    of data and 0,368975 seconds for 1467 executions). That makes 37,002989 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 26 of Juniper program 'Parse' running at
    cloud node with host name/IP address 147.229.8.105 was receiving data in 192,679000 seconds of
    539,898000 seconds of its total execution time (averages are 0,065986 seconds for 2920 receives
    of data and 0,369287 seconds for 1462 executions). That makes 35,688037 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 4 of Juniper program 'Filter' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 1,033000 seconds of
    5,248000 seconds of its total execution time (averages are 0,000347 seconds for 2980 receives
    of data and 0,003517 seconds for 1492 executions). That makes 19,683689 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 7 of Juniper program 'Filter' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 0,502000 seconds of
    4,745000 seconds of its total execution time (averages are 0,000167 seconds for 3004 receives
    of data and 0,003155 seconds for 1504 executions). That makes 10,579557 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 6 of Juniper program 'Filter' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 0,362000 seconds of
    4,671000 seconds of its total execution time (averages are 0,000120 seconds for 3022 receives
    of data and 0,003087 seconds for 1513 executions). That makes 7,749946 percentage of execution
    time spent by receiving data.
Advice DataTransferOverhead: The instance with global rank 5 of Juniper program 'Filter' running at
    cloud node with host name/IP address 147.229.8.104 was receiving data in 0,345000 seconds of
    4,652000 seconds of its total execution time (averages are 0,000115 seconds for 3010 receives
    of data and 0,003087 seconds for 1507 executions). That makes 7,416165 percentage of execution
    time spent by receiving data.
*** loading and setting advisor plugin eu.juniper.sa.tool.plugins.AdvisorOutOfMemoryPrediction

*** executing advisor plugin AdvisorOutOfMemoryPrediction with the following description:
This advisor detect Juniper programs where the memory usage is growing over the time by detecting a
    linear trend in memory usage (the linear regression analysis). This may indicate potential
    memory leaks and eventually result into performance related issues and OutOfMemoryError errors
    in a Juniper programs.

Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 4 of Juniper program
    'Filter' running at cloud node with host name/IP address 147.229.8.104 has the memory usage
    growing over the time by the approximate rate of change 25191512,187807 Bytes per second for
    the heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program on 2015-05-26 11:50:45.636 (that is 00:18:28.124
    since the beginning of analyzed data on 2015-05-26 11:32:17.512; the heap memory is limited to
    28631367680 Bytes). The sample linear regression model is Y_{size_in_bytes} = 716025141,730374
    + 25191512,187807 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 4 of Juniper program
    'Filter' running at cloud node with host name/IP address 147.229.8.104 has the memory usage
    growing over the time by the approximate rate of change 108748,469364 Bytes per second for the
    non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
    into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 30381994,359782 + 108748,469364 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 33 of Juniper program
    'SentenceSplit' running at cloud node with host name/IP address 147.229.8.106 has the memory
    usage growing over the time by the approximate rate of change 35260350,719867 Bytes per second
    for the heap memory size (the recommended maximum is 0,100000 Bytes per second). This may
    result into OutOfMemoryError errors in the program on 2015-05-26 11:45:31.014 (that is
    00:13:13.502 since the beginning of analyzed data on 2015-05-26 11:32:17.512; the heap memory
    is limited to 28631367680 Bytes). The sample linear regression model is Y_{size_in_bytes} =
    652189767,226191 + 35260350,719867 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 33 of Juniper program
    'SentenceSplit' running at cloud node with host name/IP address 147.229.8.106 has the memory
    usage growing over the time by the approximate rate of change 101916,201744 Bytes per second
    for the non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may
    result into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 30015788,907701 + 101916,201744 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 7 of Juniper program
    'Filter' running at cloud node with host name/IP address 147.229.8.104 has the memory usage
    growing over the time by the approximate rate of change 25919255,567001 Bytes per second for
    the heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result
```

into OutOfMemoryError errors in the program on 2015-05-26 11:50:15.878 (that is 00:17:58.366 since the beginning of analyzed data on 2015-05-26 11:32:17.512; the heap memory is limited to 28631367680 Bytes). The sample linear regression model is Y_{size_in_bytes} = 680914986,680025 + 25919255,567001 * X_{time_in_sec}.

Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 7 of Juniper program 'Filter' running at cloud node with host name/IP address 147.229.8.104 has the memory usage growing over the time by the approximate rate of change 107207,563624 Bytes per second for the non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result into OutOfMemoryError errors in the program. The sample linear regression model is Y_{size_in_bytes} = 30469096,890765 + 107207,563624 * X_{time_in_sec}.

Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 6 of Juniper program 'Filter' running at cloud node with host name/IP address 147.229.8.104 has the memory usage growing over the time by the approximate rate of change 25872737,604439 Bytes per second for the heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result into OutOfMemoryError errors in the program on 2015-05-26 11:50:17.679 (that is 00:18:00.167 since the beginning of analyzed data on 2015-05-26 11:32:17.512; the heap memory is limited to 28631367680 Bytes). The sample linear regression model is Y_{size_in_bytes} = 684489279,131399 + 25872737,604439 * X_{time_in_sec}.

Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 6 of Juniper program 'Filter' running at cloud node with host name/IP address 147.229.8.104 has the memory usage growing over the time by the approximate rate of change 100017,397278 Bytes per second for the non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result into OutOfMemoryError errors in the program. The sample linear regression model is Y_{size_in_bytes} = 30227219,182431 + 100017,397278 * X_{time_in_sec}.

Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 35 of Juniper program 'SentenceSplit' running at cloud node with host name/IP address 147.229.8.106 has the memory usage growing over the time by the approximate rate of change 35216956,153826 Bytes per second for the heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result into OutOfMemoryError errors in the program on 2015-05-26 11:45:32.213 (that is 00:13:14.701 since the beginning of analyzed data on 2015-05-26 11:32:17.512; the heap memory is limited to 28631367680 Bytes). The sample linear regression model is Y_{size_in_bytes} = 644409639,812841 + 35216956,153826 * X_{time_in_sec}.

Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 35 of Juniper program 'SentenceSplit' running at cloud node with host name/IP address 147.229.8.106 has the memory usage growing over the time by the approximate rate of change 107852,298691 Bytes per second for the non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result into OutOfMemoryError errors in the program. The sample linear regression model is Y_{size_in_bytes} = 29950099,188930 + 107852,298691 * X_{time_in_sec}.

Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 34 of Juniper program 'SentenceSplit' running at cloud node with host name/IP address 147.229.8.106 has the memory usage growing over the time by the approximate rate of change 38793657,335887 Bytes per second for the heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result into OutOfMemoryError errors in the program on 2015-05-26 11:44:19.566 (that is 00:12:02.054 since the beginning of analyzed data on 2015-05-26 11:32:17.512; the heap memory is limited to 28631367680 Bytes). The sample linear regression model is Y_{size_in_bytes} = 620230268,004137 + 38793657,335887 * X_{time_in_sec}.

Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 34 of Juniper program 'SentenceSplit' running at cloud node with host name/IP address 147.229.8.106 has the memory usage growing over the time by the approximate rate of change 101548,242340 Bytes per second for the non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result into OutOfMemoryError errors in the program. The sample linear regression model is Y_{size_in_bytes} = 29894620,733395 + 101548,242340 * X_{time_in_sec}.

Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 5 of Juniper program 'Filter' running at cloud node with host name/IP address 147.229.8.104 has the memory usage growing over the time by the approximate rate of change 24657879,498275 Bytes per second for the heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result into OutOfMemoryError errors in the program on 2015-05-26 11:51:10.595 (that is 00:18:53.083 since the beginning of analyzed data on 2015-05-26 11:32:17.512; the heap memory is limited to 28631367680 Bytes). The sample linear regression model is Y_{size_in_bytes} = 691931247,868077 + 24657879,498275 * X_{time_in_sec}.

Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 5 of Juniper program 'Filter' running at cloud node with host name/IP address 147.229.8.104 has the memory usage growing over the time by the approximate rate of change 103581,857571 Bytes per second for the non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may result into OutOfMemoryError errors in the program. The sample linear regression model is Y_{size_in_bytes} = 30356419,551213 + 103581,857571 * X_{time_in_sec}.

```
Advice OutOfMemoryPrediction_HeapMemory: The instance with global rank 32 of Juniper program
    'SentenceSplit' running at cloud node with host name/IP address 147.229.8.106 has the memory
    usage growing over the time by the approximate rate of change 36472397,062788 Bytes per second
    for the heap memory size (the recommended maximum is 0,100000 Bytes per second). This may
    result into OutOfMemoryError errors in the program on 2015-05-26 11:45:05.046 (that is
    00:12:47.534 since the beginning of analyzed data on 2015-05-26 11:32:17.512; the heap memory
    is limited to 28631367680 Bytes). The sample linear regression model is Y_{size_in_bytes} =
    637541291,455761 + 36472397,062788 * X_{time_in_sec}.
Advice OutOfMemoryPrediction_NonHeapMemory: The instance with global rank 32 of Juniper program
    'SentenceSplit' running at cloud node with host name/IP address 147.229.8.106 has the memory
    usage growing over the time by the approximate rate of change 102028,026860 Bytes per second
    for the non-heap memory size (the recommended maximum is 0,100000 Bytes per second). This may
    result into OutOfMemoryError errors in the program. The sample linear regression model is
    Y_{size_in_bytes} = 29901067,757512 + 102028,026860 * X_{time_in_sec}.
*** loading and setting advisor plugin
    eu.juniper.sa.tool.plugins.AdvisorGarbageCollectionPerformance

*** executing advisor plugin AdvisorGarbageCollectionPerformance with the following description:
This advisor detects Juniper programs that spent much time on garbage collecting. Long garbage
    collections may affect negatively responsiveness of a Juniper application which is critical in
    real-time stream processing of Big Data (any delay in processing of such data may result into
    data-loss issues).

Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 30 of Juniper
    program 'PartOfSpeech' running at cloud node with host name/IP address 147.229.8.105 was
    performed 2 garbage collections that took 2,617000 seconds in 29,345000 seconds of total
    execution time of the program (averages are 1,308500 seconds per garbage collection and
    0,020622 seconds for the execution time). That makes 8,918044 percentage of execution time
    spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 28 of Juniper
    program 'PartOfSpeech' running at cloud node with host name/IP address 147.229.8.105 was
    performed 2 garbage collections that took 2,476000 seconds in 30,025000 seconds of total
    execution time of the program (averages are 1,238000 seconds per garbage collection and
    0,020650 seconds for the execution time). That makes 8,246461 percentage of execution time
    spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 29 of Juniper
    program 'PartOfSpeech' running at cloud node with host name/IP address 147.229.8.105 was
    performed 2 garbage collections that took 1,438000 seconds in 27,031000 seconds of total
    execution time of the program (averages are 0,719000 seconds per garbage collection and
    0,018720 seconds for the execution time). That makes 5,319818 percentage of execution time
    spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 43 of Juniper
    program 'Tokenize' running at cloud node with host name/IP address 147.229.8.106 was performed
    1 garbage collections that took 0,432000 seconds in 22,024000 seconds of total execution time
    of the program (averages are 0,432000 seconds per garbage collection and 0,015085 seconds for
    the execution time). That makes 1,961497 percentage of execution time spent by garbage
    collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 41 of Juniper
    program 'Tokenize' running at cloud node with host name/IP address 147.229.8.106 was performed
    1 garbage collections that took 0,411000 seconds in 21,719000 seconds of total execution time
    of the program (averages are 0,411000 seconds per garbage collection and 0,015252 seconds for
    the execution time). That makes 1,892352 percentage of execution time spent by garbage
    collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 31 of Juniper
    program 'PartOfSpeech' running at cloud node with host name/IP address 147.229.8.105 was
    performed 1 garbage collections that took 0,507000 seconds in 26,939000 seconds of total
    execution time of the program (averages are 0,507000 seconds per garbage collection and
    0,018786 seconds for the execution time). That makes 1,882030 percentage of execution time
    spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 42 of Juniper
    program 'Tokenize' running at cloud node with host name/IP address 147.229.8.106 was performed
    1 garbage collections that took 0,400000 seconds in 21,580000 seconds of total execution time
    of the program (averages are 0,400000 seconds per garbage collection and 0,015080 seconds for
    the execution time). That makes 1,853568 percentage of execution time spent by garbage
    collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 26 of Juniper
    program 'Parse' running at cloud node with host name/IP address 147.229.8.105 was performed 13
```

```
    garbage collections that took 2,172000 seconds in 539,898000 seconds of total execution time of
    the program (averages are 0,167077 seconds per garbage collection and 0,369287 seconds for the
    execution time). That makes 0,402298 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 27 of Juniper
    program 'Parse' running at cloud node with host name/IP address 147.229.8.105 was performed 14
    garbage collections that took 1,921000 seconds in 540,289999 seconds of total execution time of
    the program (averages are 0,137214 seconds per garbage collection and 0,374421 seconds for the
    execution time). That makes 0,355550 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 24 of Juniper
    program 'Parse' running at cloud node with host name/IP address 147.229.8.105 was performed 14
    garbage collections that took 1,824000 seconds in 541,286000 seconds of total execution time of
    the program (averages are 0,130286 seconds per garbage collection and 0,368975 seconds for the
    execution time). That makes 0,336975 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 25 of Juniper
    program 'Parse' running at cloud node with host name/IP address 147.229.8.105 was performed 12
    garbage collections that took 1,451000 seconds in 542,647000 seconds of total execution time of
    the program (averages are 0,120917 seconds per garbage collection and 0,392369 seconds for the
    execution time). That makes 0,267393 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 37 of Juniper
    program 'Sentiment' running at cloud node with host name/IP address 147.229.8.106 was performed
    4 garbage collections that took 0,881000 seconds in 541,735000 seconds of total execution time
    of the program (averages are 0,220250 seconds per garbage collection and 0,387507 seconds for
    the execution time). That makes 0,162626 percentage of execution time spent by garbage
    collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 38 of Juniper
    program 'Sentiment' running at cloud node with host name/IP address 147.229.8.106 was performed
    3 garbage collections that took 0,780000 seconds in 541,245000 seconds of total execution time
    of the program (averages are 0,260000 seconds per garbage collection and 0,367195 seconds for
    the execution time). That makes 0,144112 percentage of execution time spent by garbage
    collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 21 of Juniper
    program 'Ner' running at cloud node with host name/IP address 147.229.8.104 was performed 10
    garbage collections that took 0,672000 seconds in 536,200000 seconds of total execution time of
    the program (averages are 0,067200 seconds per garbage collection and 0,373659 seconds for the
    execution time). That makes 0,125326 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 20 of Juniper
    program 'Ner' running at cloud node with host name/IP address 147.229.8.104 was performed 10
    garbage collections that took 0,621000 seconds in 538,643000 seconds of total execution time of
    the program (averages are 0,062100 seconds per garbage collection and 0,384745 seconds for the
    execution time). That makes 0,115290 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 22 of Juniper
    program 'Ner' running at cloud node with host name/IP address 147.229.8.104 was performed 11
    garbage collections that took 0,613000 seconds in 536,980000 seconds of total execution time of
    the program (averages are 0,055727 seconds per garbage collection and 0,366289 seconds for the
    execution time). That makes 0,114157 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 23 of Juniper
    program 'Ner' running at cloud node with host name/IP address 147.229.8.104 was performed 11
    garbage collections that took 0,586000 seconds in 537,972000 seconds of total execution time of
    the program (averages are 0,053273 seconds per garbage collection and 0,369994 seconds for the
    execution time). That makes 0,108928 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 39 of Juniper
    program 'Sentiment' running at cloud node with host name/IP address 147.229.8.106 was performed
    2 garbage collections that took 0,507000 seconds in 542,149999 seconds of total execution time
    of the program (averages are 0,253500 seconds per garbage collection and 0,381259 seconds for
    the execution time). That makes 0,093517 percentage of execution time spent by garbage
    collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 36 of Juniper
    program 'Sentiment' running at cloud node with host name/IP address 147.229.8.106 was performed
    2 garbage collections that took 0,418000 seconds in 540,752996 seconds of total execution time
    of the program (averages are 0,209000 seconds per garbage collection and 0,370125 seconds for
    the execution time). That makes 0,077300 percentage of execution time spent by garbage
    collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 14 of Juniper
    program 'Index' running at cloud node with host name/IP address 147.229.8.104 was performed 4
    garbage collections that took 0,394000 seconds in 521,142000 seconds of total execution time of
    the program (averages are 0,098500 seconds per garbage collection and 0,074716 seconds for the
    execution time). That makes 0,075603 percentage of execution time spent by garbage collections.
```

```
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 13 of Juniper
    program 'Index' running at cloud node with host name/IP address 147.229.8.104 was performed 4
    garbage collections that took 0,384000 seconds in 519,204000 seconds of total execution time of
    the program (averages are 0,096000 seconds per garbage collection and 0,072062 seconds for the
    execution time). That makes 0,073959 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 12 of Juniper
    program 'Index' running at cloud node with host name/IP address 147.229.8.104 was performed 4
    garbage collections that took 0,358000 seconds in 519,262000 seconds of total execution time of
    the program (averages are 0,089500 seconds per garbage collection and 0,072020 seconds for the
    execution time). That makes 0,068944 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 15 of Juniper
    program 'Index' running at cloud node with host name/IP address 147.229.8.104 was performed 3
    garbage collections that took 0,337000 seconds in 517,550000 seconds of total execution time of
    the program (averages are 0,112333 seconds per garbage collection and 0,070081 seconds for the
    execution time). That makes 0,065114 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 18 of Juniper
    program 'Lemma' running at cloud node with host name/IP address 147.229.8.104 was performed 1
    garbage collections that took 0,169000 seconds in 532,104000 seconds of total execution time of
    the program (averages are 0,169000 seconds per garbage collection and 0,365205 seconds for the
    execution time). That makes 0,031761 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 17 of Juniper
    program 'Lemma' running at cloud node with host name/IP address 147.229.8.104 was performed 1
    garbage collections that took 0,156000 seconds in 534,962001 seconds of total execution time of
    the program (averages are 0,156000 seconds per garbage collection and 0,363179 seconds for the
    execution time). That makes 0,029161 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 19 of Juniper
    program 'Lemma' running at cloud node with host name/IP address 147.229.8.104 was performed 1
    garbage collections that took 0,155000 seconds in 537,329000 seconds of total execution time of
    the program (averages are 0,155000 seconds per garbage collection and 0,388243 seconds for the
    execution time). That makes 0,028846 percentage of execution time spent by garbage collections.
Advice GarbageCollectionDelays: The Java Hotspot JVM of the instance with global rank 16 of Juniper
    program 'Lemma' running at cloud node with host name/IP address 147.229.8.104 was performed 1
    garbage collections that took 0,149000 seconds in 532,405000 seconds of total execution time of
    the program (averages are 0,149000 seconds per garbage collection and 0,369469 seconds for the
    execution time). That makes 0,027986 percentage of execution time spent by garbage collections.

*** writing the list of advice into XML file 10.merged.advice.xml
*** removing database files with monitoring results cache
```