JUNIPER: Towards Modeling Approach Enabling Efficient Platform for Heterogeneous Big Data Analysis

Marcos Aurélio Almeida da Silva
Andrey Sadovykh
Alessandra Bagnato
SOFTEAM
20, avenue Victor Hugo
75016 Paris, France
{firstname.lastname}@softeam.fr

Alexey Cheptsov
High Performance Computing Center Stuttgart
Nobelstr. 19, 70569 Stuttgart, Germany
cheptsov@hlrs.de

Ludwig Adam
petaFuel
Münchner Straße 4
85354 Freising, Germany
ludwig.adam@petafuel.de

ABSTRACT
Big Data is a modern phenomenon that promises to bring unprecedented economical benefits. Hadoop-like MapReduce implementations has gained a well deserved popularity by providing an open-source data management solution running on commodity PC clusters and with a potential of Big Data scale. Nevertheless, there are many critical problems, for which solutions based on HPCs, FPGA-enabled nodes and providing real-time guaranties may offer a cost-efficient solution for data processing. JUNIPER is a European research projects that is carried out by an international consortium aiming at developing a Big Data analysis platform. In this article, we present an integral part of JUNIPER - a modeling approach, which helps abstracting the data processing stages and wrap the communication between them. This approach is also applied to specify the timing constraints. We illustrate our approach on a real-life application of credit card transaction processing developed by petaFuel.

Categories and Subject Descriptors
D.2.10 [**Design**]: Methodologies and representation.

General Terms
Documentation, Performance, Design.

Keywords
Big Data, Model-Driven Development, FPGA, HPC, Real-time, Java.

1. INTRODUCTION
Big Data is a challenging problem that opens a principally new dimension to the traditional data analysis applications with an enormous market potential [1][2]. The Big Data revolution has become a reality due to abundance of relatively affordable and simple tools for processing of large volumes of data and data streams. Apache Hadoop [3], Spark [4] and Storm [5] made it possible to run Map Reduce [6] and stream handling operations with simple APIs on commodity clusters, i.e. clusters built of desktop PCs. Nevertheless, more and more business cases are starting to require a better performance and real-time guarantees as discussed later in this paper. The latest advances in High-Performance Computing (HPC) and hardware acceleration with Field-Programmable Gate Array (FPGA) boards enable new architectures and tools for efficient processing of heterogeneous Big Data, while operating costs has been drastically reduced.

Juniper [7] is an European FP7 project that proposes to develop a programmable platform that combines advantages of the current state of the art in HPC, FPGA, real-time Java compilers, and real-time scheduling. This platform will deliver a leap in performance for variety of data processing problems, while preserving the operating costs and simplicity of programming interface.

One of the integral parts of the Juniper platform is its modeling language that helps design the data processing flow, align it with the real-time scheduler and automatically generate the communication code leveraging a high-performance communication library, such as OpenMPI [8], the standard communication mechanism in the HPC community.

In this article, we present our work in progress for the Juniper modeling language and engineering process, which we illustrate with a real-life case study from the financial sector. We start the paper with an overview of the Juniper approach. Further, we provide a detailed view on the modeling language and future perspectives.

2. OVERVIEW AND ILLUSTRATIVE EXAMPLE
Over the recent years, the MapReduce programming model and its corresponsing implementations gained a well-deserved popularity for parallel processing huge amounts of data on commodity PC clusters. The typical Big Data application, as depicted in Figure 1 would process the data coming from various sources in one or multiple stages through batch processing of various data sets. The data processing would usually result in an analytical data base that fits well for presenting the final results, for example, with a visualization kit.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

CEE-SECR '14, October 23 - 25 2014, Moscow, Russian Federation
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2889-0/14/10…$15.00
http://dx.doi.org/10.1145/2687233.2687252

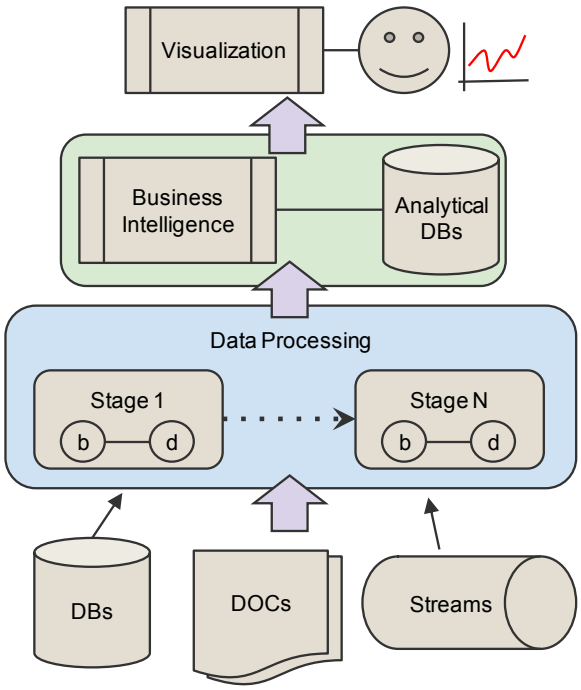

Figure 1. Typical Big Data Approach.

The data processing is often implemented with MapReduce and automatically managed by Hadoop clusters.

Hadoop automates the batch processing making the parallel processes to communicate over HDFS. Originally, it was designed for MapReduce tasks, but currently also used for data streams processing. Arguably Hadoop is suited for general audience, who can apply it for a large set of problems. Hadoop manages automatically redundancy and recovery of faulty cluster nodes. While this platform intends to provide the processing results as soon as possible and even sooner, it does not provide guarantees for the response time.

At the same time, there is also a large set of critical problems such as banking transaction processing and credit card payment fraud detections presented in this paper. For these problems the response time is of vital importance because of the threat of heavy financial loses. Moreover, there are also problems, which are better treated HPC clusters or with hardware accelerators. In our studies we previously showed that MapReduce type of processing can be more efficiently handled with HPC clusters due to significant gains in the interconnect time, parallel IO and efficient scheduling. We demonstrated earlier that using HPC for Big Data may reveal to be cost-effective for critical systems. Furthermore, recent FPGAs boards [9] provide highly affordable solutions for tasks such as image processing and patterns recognition. Finally, the progress in real-time scheduling [10] and Java Real-Time [11] make it possible to approach the response time guarantees problem.

JUNIPER, as it is depicted in Figure 2, proposes to combine the Hadoop platforms with HPC managed tasks and specific FPGA-enabled cluster nodes in order to extend the application area for Big Data technologies and address problems revealed in the domain of the critical systems.

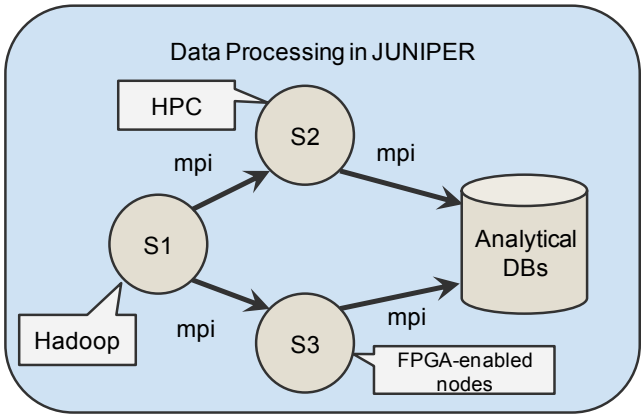

Figure 2. JUNIPER Approach for Data Processing

In JUNIPER, we propose a holistic approach, in which the stages processed with Hadoop communicate with HPC and FPGA-enabled stages over MPI - the most wide-spread HPC communication libraries. MPI offers a lot of possibilities to establish the communication between two (point-to-point) or a group (collective) of processes (compute nodes) as well as to organize a collective parallel I/O to a shared file system (MPI-IO). However, the use of MPI requires a decent knowledge of the underlying communication technology from the programmer and thus offers a high adoption barrier by the parallel applications.

For task parallelization we use several techniques such as Yarn native to Hadoop as well as SLURM and TORQUE for HPC. Within a single node enabled with an FPGA, we apply a specific real-time scheduler reinforcing the response time guarantees.

In our work, we face a challenge to specify the whole set of data stages, data processing tasks and scheduling properties. We address this challenge through a UML-based modeling language, which helps to integrate Big Data in the engineering practices, hide the complexity of handling the communication between data processing stages and manage the schedulability analysis. Among other benefits, this approach allows to hide the complexity of MPI by generation of code wrappers and to help developers to master the efficient parallel data processing. In this paper we present this modeling approach.

Before going into details, we will present a motivating example from the financial sector in credit card payment fraud detection.

PetaFuel, a privately owned SME, provides financial services for online processing of prepaid credit cards transactions. One of the major problems the company faces is the detection of fraudulent transactions. Fraud detection limits costs caused by erroneously authorized transactions and is also highly important for card issuer reputation. Thus the transactions have to be thoroughly examined with the most sophisticated checks prior to the approval. However, the complexity of all of these checks is limited by specific time constraints imposed by the payment network provider. For example, MasterCard imposes 4 seconds as the time limit for a transaction examination. The current approval system checks basic facts such as transaction type, the amount cryptographic checks, but also specific patterns based on the historical data for the whole set of transactions. Currently, the amount of historical data analyzed is limited to 30GB due to technical constraints on data processing.

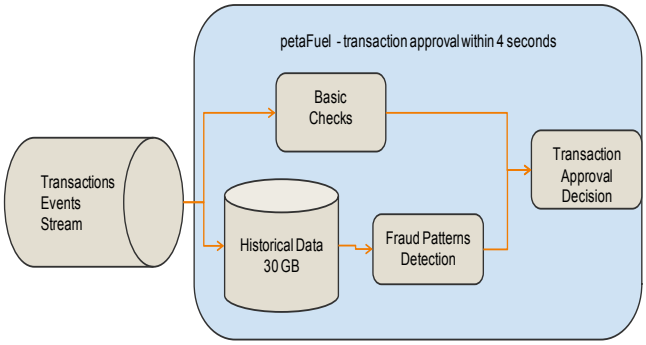

Figure 3. petaFuel Transaction Approval Process

It is in the best interest of petaFuel to enhance the approval system in order to cope with the ever growing number of transactions and more sophisticated fraud patterns. The JUNIPER platform and the modeling language are dedicated to help to build an efficient infrastructure for such problems.

In the sections below we present the modeling approach that helps to specify the data processing stages, define the communication layer and indicate the timing constrains.

3. MODELLING APPROACH

3.1 Concept

The modeling support embedded in JUNPER addresses the issues related to the programmability of large scale big data systems, keeping in mind other industrial related issues such as scalability, and real-time guarantees. In order to support that, a model-driven engineering approach is being developed, so that, modeling facilities could be used to ensure that the developed system is analyzable on a high-level and still consistent with the Java code that runs on the platform.

The main issues addressed by JUNIPER modeling approach are:

- Reducing the complexity, in terms of the necessary skills, in defining and document the architecture of big data systems;
- Facilitating the portability of big data systems, specially between database management systems;
- Making it possible for developers to estimate whether real-time constraints are likely to be met at design time.

This approach is based on the use of high-level UML based models that will then be used to analyze real-time properties of the system and to generate part of the code of big data real time applications.

Our approach intends to address these issues as follows:

- The use of models based on standards (UML [12], [13] and MARTE [14]) and the generation of Java code following the JUNIPER approach is expected to reduce the complexity in defining, understanding and documenting the architecture of JUNIPER applications;
- The use of high-level models, that are the less tied as possible to any specific platform or database management system, is expected to improve the portability of big data systems.
- The integration of the modelling language with a schedulability analysis approach is expected to make it possible for developers to produce estimates of the real-time characteristics of systems under development. The model-driven approach is also going to make sure that the analysed system effectively corresponds to the implemented Java code.
- The integration of the modelling approach with standard software engineering practice tools. In the specific context of modelling based software engineering, the JUNIPER approach will integrate to document generators, model transformation tools, analysers and development tools such as Integrated Development Environments.

The JUNIPER programming model is based on a paradigm of data streams being transmitted and processed on a HPC infrastructure. It is based on the concepts on Application, Program, Communication Streams (to and from the external world) and Communication Channels (internal to the application). Processing nodes of a given application are called Programs, they receive data from incoming streams, perform some computation on it and then generate a new stream that should be processed by other programs or eventually transmitted to the external world. Low level concerns such as real-time scheduling, computation off-loading to FPGA's, and high performance communication between programs are dealt with by the underlying JUNIPER platform and are orthogonal to the modeling approach. Finally, the programming model is considered to be *semi-static*, since even though its structure is fixed, programs can be multiplied in order to handle peaks in load.

3.2 UML Based Language for Describing JUNIPER Applications

The two challenges underlying the use of MDE approaches for handling the problem of designing multi-cloud applications are: (i) choosing a modeling language that is high-level enough to abstract from the concepts in different programming languages, and (ii) low level enough to allow for code generation. We chose the Unified Modeling Language (UML) as modeling language since its object-oriented roots have been shown to be useful to model a wide range of problems while still serving as basis for code generation.

It's main drawback is however in the complexity of its specification, and therefore the steep learning curve that it represents to developers. Our approach to countering this drawback consists in selecting a **subset** of UML suitable to address both challenges. The subset selected in this paper starts with the subset of the language usually reused by code generation tools [15] [16] [17] [18] [19].

This section presents the concepts that will be supported by the modeling approach. It contains however extra concepts to represent, on a high level, the hardware platform on which the application will be deployed along with the real-time constraints and properties associated to each hardware and software element. These concepts were included to facilitate the schedulability analysis and the future support to the generation of deployment code from these models. Among other real-time analysis tools we also support MAST [20].

We divide the modeling concepts into three groups: the modeling of the **hardware platform**, the modeling of the **application architecture** and the modeling of the **allocation** of parts of the application to parts of the hardware platform. The main design principle followed is that the metamodel, and consequently the UML/MARTE subset, should be as small as possible. It should also still allow for code generation and schedulability analysis.

This was done to avoid the complexity of the proposed approach to hamper the adoption and use of the modelling support.

Program instances communicate by means of the MPI communication framework. This allows us to generate the code for Java programs including MPI communication and to perform schedulability analysis by modeling the use of different kinds of resource (CPU, disk, communication, etc.) by the different threads of the program instances.

Table 1 presents the mapping from the modeling concepts on the JUNIPER programming model into UML/MARTE concepts.

Generally speaking, we map the hardware related concepts onto UML object diagram concepts, the application modeling related concepts onto UML class diagram concepts and allocation related ones onto a mix of elements that may appear in different UML diagrams, like, object diagrams, class diagrams and activity diagrams.

More details on the proposed mapping follow:

- **Hardware platform modelling concepts:**
  - **UML concepts:** The hardware platform will be represented by a set of UML **InstanceSpecification**s (representing cloud nodes and cloud disks) in a UML **Package** (representing the whole platform).
  - **MARTE concepts:** Cloud Nodes are represented as **HwProcessors**; the number of CPUs of each node will be represented by the **nbCores** attribute from MARTE **HwProcessors**. The Cloud Disk concept is represented by the **HwDrive** stereotype.
- **Application architecture concepts:**
  - **UML concepts:** The application architecture will be represented by a set of Classes (representing JUNIPER programs) in a class diagram. The associations between these classes (here represented by their Ports) will represent the communication between the programs. Programs and non-pre-emptive software regions may be represented either as **Classes** or **InstanceSpecifications**. In the first case it can be used to represent the allocation of Program instances to non-pre-emptive software regions and communication channels. In the second case, it can be used to represent the allocation of Programs to Cloud Nodes and Cloud Disks. The multiplicity of the **InstanceSpecification**s will be used to represent a **ProgramGroup**.
  - **MARTE concepts:** Juniper programs are active entities and therefore they are represented by the MARTE **RtUnit** stereotype. The **RtSpecification** and **ResourceUsage** stereotypes are used to represent the real-time characteristics of a program. Its **occKind** attribute is used to represent the periodic properties of a Task and a Channel. Finally, non-pre-emptive software regions correspond to MARTE **SwMutualExclusionResource**s.
- **Application behavior modelling concepts:**
  - **UML concepts:** Tasks will be represented by **Action**s on an activity diagram owned by a Program. The use of resources will be represented by **Comments** associated to model elements.

MARTE concepts: Tasks correspond to MARTE RtActions and resource usage are modeled by means of MARTE ResourceUsages. The priority attribute of a RtSpecification is also used to represent the priority of a Task.

Table 1. Subset of UML and MARTE reused in JUNIPER for modeling big data real-time applications

Modelling concept	UML Metaclass	MARTE Stereotype
Hardware platform modeling concepts		
Hardware Platform	Package	-
CloudNode	InstanceSpecification	HwComputingResource
CloudNode. numberOfCPUs	InstanceSpecification	HwProcessor.nbCores
CloudDisk	InstanceSpecification	HwDrive
Application architecture modelling concepts		
Application	Component	-
Program	Class, InstanceSpecification	RtUnit
ProgramGroup	InstanceSpecification	-
ProgramGroup. minSize	InstanceSpecification. MultiplicityMin	-
ProgramGroup. maxSize	InstanceSpecification. MultiplicityMax	-
Channel	Port	-
Channel.name	Port.name	-
Channel.datatype	Port.type	-
Channel ementSize	-	ResourceUsage. msgSize
Channel.burstSize	-	RtFeature. specification. occKind.burst.burstSize
Channel.period	-	RtFeature. specification. occKind.periodic.period
Channel. timeBound	-	RtFeature. specification. relDl
Application behavior modelling concepts		
Task	Action	RtAction
Task.priority	-	RtFeature. specification.priority
ResourceUsage Profile	Comment	ResourceUsage
NonPreemptive SwRegion	InstanceSpecification, Class	SwMutialExclusionResource

4. CASE STUDY

In this In this section we present the implementation of the petaFuel case study described in Section 2 using the Juniper platform following the Modeling approach we described on the preceding section.

This case study was implemented by means of the code generation capabilities of an alpha version of the JUNIPER Modeling Environment being developed as a module to the Modelio Open Source modeling environment[1]. Softeam already implemented the subset of UML and MARTE used in JUNIPER for modeling the hardware platform, the application architecture and the application behavior of big data real-time applications within the Modelio Modelling tool. Modelio allows the user to use all the OMG MARTE standard concepts from the MARTE 1.2 standard specification and it guides the specific JUNIPER concert through a menu interface

The generated code has been deployed on the HPC environment provided to the JUNIPER project by BonFIRE Foundation, established in December 2013 to operate the BonFIRE multi-site Cloud testing facility [22]. Actual anonymzed data from financial processing systems, accounting for more than 7 million events. Evaluating the performance of the implementation was deemed as out of the scope of this case study.

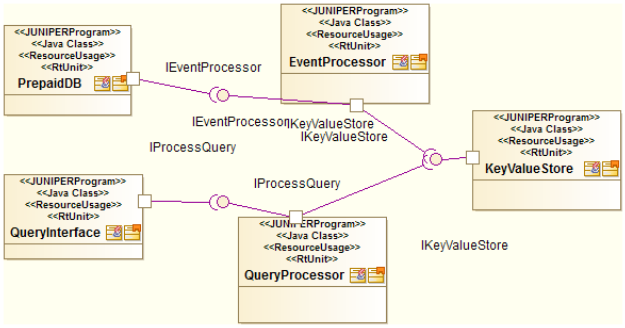

Figure 4. Case study architecture.

Figure 4 and Figure 5 show the case study architecture model and hardware platform models.

The objective of this study was to provide an early validation on the modeling environment and of its code generation approach, which allows MPI communication code to be handled automatically by the generated code. Five JUNIPER programs were implemented within the petaFuel JUNIPER Software Platform, as we can see from Figure 4. The PrepaidDB and QueryInterface programs receive respectively external events and queries and transform them into Java/MPI calls to other juniper programs. The EventProcessor aggregates events and routes them to data store. The KeyValueStore represents a NoSQL key-value store. The QueryProcessor processes queries received from the external world by the QueryInterface and using data from the store, computes responses to these queries.

Figure 5 describes the hardware platform model for our experiment. In this model, two cloud nodes host the aforementioned Juniper programs. The first one (with IP address ending in 81) hosts the QueryProcessor, the PrepaidDB, the QueryInterface and the EventProcessor programs. The other one hosts the KeyValueStore. We also model the CPUs and disks present in each node. This will allow our schedulability analysis tools to detect resource usage bottle necks at design time.

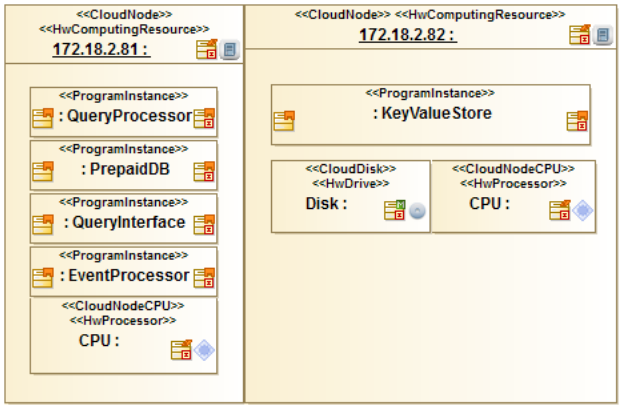

Figure 5. Deployment model.

By means of the Modeling Environment (for designers) and of Eclipse (for developers) code was generated and implemented for this case study. Figure 6 shows a snippet of the generated code. Besides the business level code (i.e. processing inputs and computing data to be sent to other programs), notice that references are made to MPI API for communication and to a JUNIPER API for communication. All these lines of code are generated automatically by the modeling environment, the developers only need to provide the business level code of the application. In this example, the implementation of the process method of the IEventProcessor interface is provided by developers, the rest of the code is automatically generated by the modeling environment.

```
public class EventProcessor {
  public static final int RANK = 1;
  public static IEventProcessor iEventProcessorImpl = new IEventProcessor() {
    @Override
    public void process(Event event) {
      String key = getKeyFromTimestamp(event.getTimestamp());
      String value = keyValueStoreIKeyValueStore.find(key);
      if (value == null) {
        keyValueStoreIKeyValueStore.put(key, "1");
      } else {
        int count = Integer.parseInt(value);
        keyValueStoreIKeyValueStore.put(key, ""+(count+1);
      }
    }
    ...
  };
  ...
  public static void main(final String[] args) {
    MPI.Init(args);
    ...
    MPI.Finalize();
  }
```

Figure 6. Example of generated code.

Our initial experiments with petaFuel case study have shown that code generation is effective for abstracting on MPI communication aspects of processing streams on HPC infrastructures. The next steps of the JUNIPER project consist in managing data models and storage on the programming model level. Initial experiments and findings have been published here [21].

[1] http://www.modelio.org

5. RELATED WORKS

Languages such as SoaML [23], SoaMF [24] and CloudML [27] are used to define the high level architecture of cloud applications. These languages however are not intended to support nor big data nor real-time contexts. They do not allow neither the representation of the data types used in the communication between high level components nor the real-time constraints to be respected by the message passing interfaces.

Data integration tools such as Pentaho [29] and Yahoo! Pipes [30] offer visual editors that allow one to describe the partitioning of data in different data stores. In both categories of work, the data structures are considered but the architecture of the application and its real-time constraints are not represented.

Another group of languages uses the existing UML deployment diagrams to model the physical distribution of data besides using UML class diagrams to define the architecture of the application. As an example we have the work of S. Lujan-Mora and J. Trujillo [28]. They define a UML profile that they can use to specify the deployment of Data Warehouses, which can be considered a former kind of private clouds, and could be extended to potentially represent multi-cloud big data deployment. This approach however is not cloud specific and doesn't takes into account real-time constraints.

There are however UML profiles, such as MARTE (Modelling and Analysis of Real-Time and Embedded systems) [14] allow the representation of real time constraints on any UML based system.

In the JUNIPER project, we decided to provide an approach based on UML and MARTE. This stems from the fact that they are standards for the representation of data models, application architecture and deployment and real-time constraints respectively. The main problem of reusing these languages is that they are too expressive, including many concepts that are not necessary for big data real-time applications. Our approach consists in reusing only a subset of both languages and extending this subset, by means of the UML profile mechanism, to account for big data specific concepts.

6. CONCLUSIONS

In this article we presented a modeling approach involved in the solution by the JUNIPER platform for critical systems requiring a Big Data processing and response time guarantees.

The modeling approach is required in order to abstract from the data processing flow, specify the timing constraints and help in implement the communication mechanisms between data processing stages. This methods helps JUNIPER to propose an efficient infrastructure combining the power of Hadoop, HPC and FPGA-enabled nodes in order to face Big Data challenges of the modern critical systems.

This work for developing the JUNIPER platform is currently ongoing. We implement the petaFuel case study in order to evaluate our approach. The final results will be reported in further papers.

7. ACKNOWLEDGMENTS

The work presented in this paper is co-funded by European Commission's Framework Program 7 in the frame of the Juniper project. We express our gratitude to all our colleagues in the JUNIPER project for very fruitful collaboration.

8. REFERENCES

[1] Manyika, James, et al. "Big data: The next frontier for innovation, competition, and productivity." (2011). McKinsey Global Institute. Available: http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation

[2] Christian, Hagen, et al. "Big Data and the Creative Destruction of Today's Business Models ", AT Kerney, Available: http://www.atkearney.fr/strategic-it/ideas-insights/article/-/asset_publisher/LCcgOeS4t85g/content/big-data-and-the-creative-destruction-of-today-s-business-models/10192

[3] Hadoop, Apache. "Hadoop." (2009). Available: http://hadoop.apache.org

[4] Zaharia, Matei, et al. "Spark: cluster computing with working sets." Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. 2010.

[5] Yang, Wenjie, et al. "Big Data Real-Time Processing Based on Storm." Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on. IEEE, 2013.

[6] J. Dean and S. Ghemawat. "Mapreduce: simplified data processing on large clusters." Commun.,51,1:107–113, 2008.

[7] JUNIPER FP7 project web-site. Available: http://www.juniper-project.org/

[8] E. Gabriel. Open MPI. "Goals, concept, and design of a next generation MPI implementation". In Proc., 11th European PVM/MPI Users' Group Meeting, pp. 97-104, Budapest, Hungary, September 2004.

[9] Y. Chan, I. Gray, A. Wellings, and N. Audsley. "Exploiting multicore architectures in big data applications: The JUNIPER approach." In Proceedings of MULTIPROG 2014 : Programmability Issues for Heterogeneous Multicores, 2014.

[10] Dario Faggioli, Giuseppe Lipari, and Tommaso Cucinotta. "Analysis and implementation of the multiprocessor bandwidth inheritance protocol." Real-Time Systems, 48(6):789–825, 2012.

[11] Basanta-Val, Pablo, and Marisol García-Valls. "Resource management policies for real-time Java remote invocations." Journal of Parallel and Distributed Computing 74.1 (2014): 1930-1944.

[12] OMG, "UML: OMG Unified Modeling Language (OMG UML) Superstructure, Version 2.4.1," 2001.

[13] T. Weilkiens. "Systems Engineering with SysML/UML: Modeling, Analysis, Design." Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[14] OMG, "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems," 2011.

[15] "ArgoUML Code Generation Window," [Online]. Available: http://argouml.tigris.org/tours/classgen.html. [Accessed 14 January 2014].

[16] MagicDraw UML Editor [Online]. Available: http://www.nomagic.com/products/magicdraw.html.

[17] “IBM,” [Online]. Available: https://www-304.ibm.com/support/docview.wss?uid=swg21259513. [Accessed 14 January 2014].

[18] “Modelio,” [Online]. Available: http://www.modeliosoft.com/en/modelio-store/modules/generators/java-designer-open-source.html.

[19] “Uml to Java Generator 2.0.2,” [Online]. Available: http://marketplace.eclipse.org/content/uml-java-generator#.UtPwpfRDvfk. [Accessed 14 January 2014].

[20] MAST - Modeling and Analysis suite for Real-Time Applications. Available: http://mast.unican.es/

[21] M. A. A. d. S. a. A. Sadovykh, “Multi-cloud and Multi-data Stores - The Challenges Behind Heterogeneous Data Models,” in *CLOSER, Special Session on Multi-Clouds*, Barcelone, 2014.

[22] BonFIRE project, Available: http://www.bonfire-project.eu/innovation [Accessed 14 July 2014].

[23] OMG, “Service oriented architecture Modeling Language (SoaML),” 2009.

[24] B. Michael, “Introduction to Service-Oriented Modeling,” in *Service-Oriented Modeling: Service Analysis*, Wiley & Sons.

[25] “Chef,” [Online]. Available: http://www.opscode.com/chef/.

[26] "Puppets," [Online]. Available: https://puppetlabs.com/.

[27] "CloudML," [Online]. Available: http://cloudml.org/.

[28] J. T. Sergio Luján-Mora, “Physical Modeling of Data Warehouses Using UML Component and Deployment Diagrams: Design and Implementation Issues,” *Database Management,* pp. 12-42, 2006.

[29] "Pentaho," [Online]. Available: http://www.pentaho.com/

[30] "Yahoo Pipes," [Online]. Available: http://pipes.yahoo.com/pipes/

[31] C. Batini, S. Ceri and S. B. Navathe, Conceptual Database Design, an Entity-Relationship Approach, Benjamin and Cummings Publ. Co., 1992.

[32] "ArgoUML DB generator." [Online]. Available: http://argouml-db.tigris.org/.

[33] Rational, “The UML and Data Modeling,” [Online]. Available: http://bit.ly/15hxqgj.

[34] "Modelio SQL Designer," [Online]. Available: http://www.modeliosoft.com/en/modules/sqldesigner.html.

[35] D. Gorni, “UML Data Modeling Profile,” 2002. [Online]. Available: http://bit.ly/VYJVGw.

[36] D. Silingas and S. Kaukenas, “Applying UML for Relational Data,” 2004. [Online]. Available: http://bit.ly/VkrNtw.

[37] “Relational Persistence for Java and .NET,” [Online]. Available: http://hibernate.org.

[38] Oracle, “Java™ Persistence 2.0, JSR 317”.

[39] “Pig Latin Reference Manual,” [Online]. Available: http://pig.apache.org/docs/r0.7.0/piglatin_ref1.html. [Accessed 14 July 2014].