Software Impacts 2 (2019) 100007

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

aFlux: Graphical flow-based data analytics

Tanmaya Mahapatra *, Christian Prehofer

Lehrstuhl für Software und Systems Engineering, Fakultät für Informatik, Technische Universität München, Boltzmannstraße 03, 85748 Garching b. München, Germany

ARTICLE INFO

Keywords:
Flow-based programming
Graphical pipelines
Mashup tools
Graphical Spark programming
graphical Flink programming

ABSTRACT

aFlux is a graphical flow-based programming tool designed to support the modelling of data analytics applications. It supports high-level programming of Big Data applications with early-stage flow validation and automatic code generation for frameworks like Spark, Flink, Pig and Hive. The graphical programming concepts used in aFlux constitute the first approach towards supporting high-level Big Data application development by making it independent of the target Big Data frameworks. This programming at a higher level of abstraction helps to lower the complexity and its ensued learning curve involved in the development of Big Data applications.

Code metadata

Current code version	0.0.3.0
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2019-6
Legal Code Licence	Apache Licence Version 2.0
Code versioning system used	GIT
Software code languages, tools, and services used	Java 8, Spring 4.3.9, Akka 2.5.1, MongoDB 3.4.4, Apache Tomcat 8.5.16
Compilation requirements, operating environments & dependencies	Maven & Java 8
If available Link to developer documentation/manual	https://aflux.org/images/pdfs/docs/aFluxDocs.pdf
Support email for questions	tanmaya.mahapatra@tum.de

1. Introduction

Data analytics has gained prominence in recent years. Nevertheless, developing Big Data applications is not a trivial task. Writing Big Data applications for frameworks like Spark [1], Flink [2], Pig [3], Hive [4] requires interaction with several libraries and APIs, and working with different data abstractions. Furthermore, developers might need to include additional libraries within an application to ensure its successful execution on Big Data clusters. This approach makes the process cumbersome and challenging with regard to quickly prototyping applications for performing exploratory data researches. Therefore, the learning curve associated with it is steep, and it requires a considerable amount of expertise to use Big Data analytics. Additionally, there is no support for end-user programming with Big Data, i.e. programming at a higher abstraction level.

We believe that one promising solution is to enable domain experts, who are not necessarily programmers, to develop the Big Data applications by providing them with domain-specific graphical tools based on the flow-based programming paradigm [5]. This conceptual approach for high-level Big Data programming involves the following (see Fig. 1 for illustration of these ideas):

1. Analysing target Big Data frameworks like Spark and Flink, extracting data abstractions and APIs which are compatible with the flow-based programming paradigm, i.e. *not supporting APIs requiring user-defined data transformation functions or supporting code-snippets during flow creation to interact with target framework internals.* Representing the selected APIs operating on the compatible data abstractions as modular, composable components. These components are independent of the execution semantics of flow-based programming tools and bundle a set of APIs invoked in a specific order to perform a specific data analytics operation.
2. Developing a generic approach to parse graphical flows making use of such modular components to generate native Big Data

* Corresponding author.
E-mail address: tanmaya.mahapatra@tum.de (T. Mahapatra).

https://doi.org/10.1016/j.simpa.2019.100007
Received 20 June 2019; Received in revised form 27 August 2019; Accepted 4 September 2019

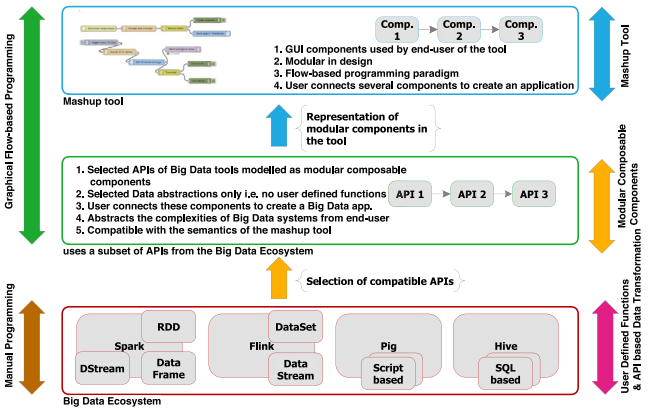

Fig. 1. Modular subset of Big Data systems compatible with flow-based programming paradigm, as in [6] (tool-agnostic).

programs and with support for early-stage validation so that the flow always yields a compilable and runnable Big Data program.

2. Software prototype

aFlux [7–10] is a graphical flow-based programming tool (mashup tool) based on the *actor model* [11] to support the design of data analytics applications with concurrent execution semantics, thereby overcoming the prevalent architectural limitations in the state-of-the-art mashup tools [12–14]. A flow-based programming model with concurrent execution semantics is suitable for modelling a wide range of Big Data applications currently used in Data Science. Without the aforementioned semantics, designing a flow involving Big Data analytics would lead to components waiting to execute for a long time, as Big Data jobs usually take a long time to finish their execution, leading to the inefficient design of applications.

2.1. Conceptual approach

aFlux implements the *modular composable components* selected from Big Data frameworks like Pig, Hive, Spark and Flink as *actors* and *enables high-level Big Data programming* with *flow validation* and automatic *code generation*. The implemented components are available on the front-end as graphical components for the user to drag and create an application flow. The flow begins with components which read datasets, followed by a series of data-transformation components and ends with a data-output component. Every component has a set of properties which the user can configure on the front-end. The flow is parsed for correctness and internally represented as a directed acyclic graph (DAG). From the DAG, the native Big Data application is generated using an API-based code generation technique [15]. The conceptual approach for flow creation and automatic code generation, including some examples of supported components for flow creation, is illustrated in Fig. 2.

2.2. Implementation and working [9]

aFlux consists of a web application and an execution environment developed in Java and the Spring Framework.[1] The web application is composed of two main entities: the front-end and back-end, based on REST API. The front-end of aFlux (Fig. 3) provides a GUI for the creation of flow-based applications, while the back-end parses the user flow to generate native application code. The application can be executed in its internal execution environment or sent to an external environment. The front-end is based on React[2] and Redux[3] frameworks. Applications are created by dragging and dropping available graphical components from the left panel onto the canvas and wiring them. New components are loaded from *plug-ins* [16]. The application shows a console-like output in the footer, and the details regarding a selected component are shown on the right-hand side panel. The 'Application Header & Menu Bar' contains functionalities to control the execution of an application, like starting the execution, stopping the execution, saving the application etc. Using the aFlux front-end, a user can create a flow by wiring several components together. Fig. 4 illustrates a Pig graphical flow created on the front-end, its resultant generated code and its output after execution. An illustrative example of Flink flow creation and code generation has been explained via a video demonstration.

Table 1
Comparison of aFlux with the state-of-the-art, following [6].

Solutions	Target-framework support	Extensibility	Code generation
aFlux	Spark, Flink, Pig & Hive	Yes	Yes
QM-IConf [22]	Storm	No	Yes
Lemonade [23]	Spark	No	Yes
QryGraph [24]	Pig	No	Yes
SPSS Modeller [25]	None	No	No
Nussknacker [26]	Flink	No	No

2.3. Installation

aFlux uses Maven [17] for project management and can be installed from the Git [18] repository (codemetadata table lists all requisite information). It can be compiled[4] on any operating system including Windows, macOS and Linux if Java 8 development environment is present. Any Java integrated development environment (IDE) like Eclipse [19] or Intellij [20] can be used for importing and compiling, which generates a Web Application Resource or Web application ARchive (WAR) file. The WAR file is deployed inside Apache Tomcat, a web server which supports the execution of Java code. On start-up, the application requires a connection to a local MongoDB [21] instance in order to save the program flows created and the application configuration settings. All front-end graphical components are separate Maven projects within the same Git repository. They are compiled separately and loaded from the GUI of aFlux after it is up and running. The web application can be accessed from any standard web browser including Safari, Firefox, Opera and Google Chrome.

3. Comparison with the state-of-the-art

We compare and contrast aFlux with the existing solutions against these parameters: (i) *target-framework support*: if the tool supports multiple target frameworks, (ii) *extensibility*: if the approach can be/has been extended to other Big Data frameworks and (iii) *code generation*: if the user flow results in a final executable code or if the flow runs in the tool's internal environment. The results are summarized in Table 1.

[1] https://spring.io/.

[2] https://reactjs.org/.

[3] https://redux.js.org/.

[4] aFlux installation video: https://aflux.org/index.php/docs/5-aflux-getting-started-videos.

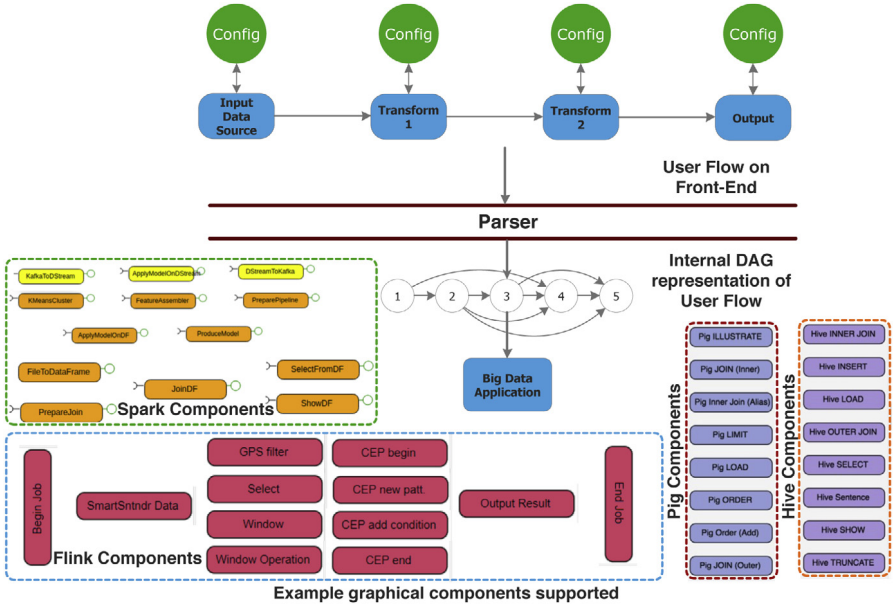

Fig. 2. Conceptual approach for flow creation and code generation in aFlux. This figure is based on the conceptual approach diagram discussed in [7].

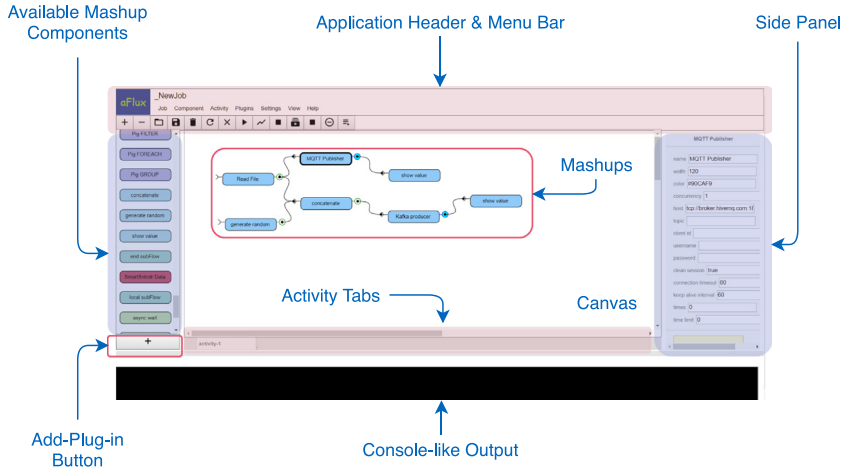

Fig. 3. Graphical user interface of aFlux, as in [9,10,16].

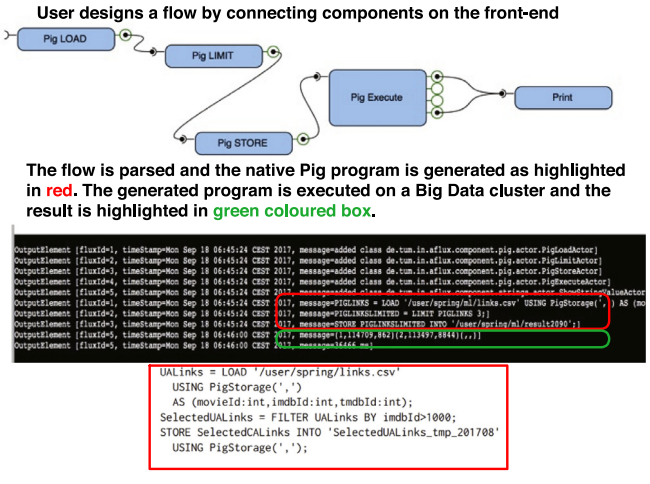

Fig. 4. Example of a Pig flow creation in aFlux.

3.1. Overview of impact

aFlux impacts end-users as well as researchers in the following way: (i) it supports graphical flow-based application development, thereby enabling non-experts to quickly prototype Big Data applications [8] and (ii) the graphical programming concepts used in aFlux is the first approach to support high-level Big Data application development by making it independent of the target Big Data frameworks, which is a significant improvement achieved over the existing state-of-the-art. It has been used to support frameworks like Spark [7], Flink [9], Pig and Hive, which demonstrates (a) the *extensibility* of the approach and (b) the *generalizability* of the code generation technique. The high-level graphical programming approach *abstracts the complexities* of Big Data application development from end-users, thereby lowering the associated learning curve and enabling less skilled Big Data programmers to adopt Big Data analytics. The main research contributions have been published in three peer-reviewed publications [7–9]. Recently, there have been many commercial solutions aimed at enabling less-skilled Big Data programmers to quickly prototype Big Data applications using Spark and Flink via flow-based programming. Examples include Stratio Sparta 2.0 [27] and StreamAnalytix [28]. The research contributions

and results from open-sourced aFlux will pave the foundation for further research in this area and will significantly help less-skilled Big Data users in an academic environment adopt Big Data analytics.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors are grateful to Shilpa Ghanashyam Gore, Federico Alonso Fernández Moreno and Ioannis Varsamidakis who actively contributed in the development of plug-ins for aFlux. Furthermore, we would like to acknowledge the inputs and suggestions received from Dr. Ilias Gerostathopoulos during various stages of this work.

This paper uses figures, tables and text from the Ph.D. dissertation titled "High-level Graphical Programming for Big Data Applications" of Tanmaya Mahapatra to be published in Technical University of Munich, 2019. All relevant inclusions from the dissertation have been cited appropriately.

This work was supported by the German Research Foundation (DFG) and the Technical University of Munich within the Open Access Publishing Funding Programme.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.simpa.2019.100007.

The illustrative example showcases how easy it is to create Flink applications graphically from aFlux, by abstracting the code generation from the end-user. The example uses real data from the city of Santander, Spain, offered as open data via public APIs [29]. In this example, the user need not have programming skills or familiarity with Flink framework. The user just needs to know how to use aFlux from the end-user perspective (i.e. how to drag and drop graphical components, etc.) and have some very basic knowledge of what Flink can do, from a functionality point of view rather than from a developer point of view. The flow created is designed to analyse traffic patterns in a particular area of the city, so as to help decision makers in the city make the appropriate calls.

References

[1] Apache, Spark - Unified Analytics Engine for Big Data, 2014. URL https://spark.apache.org. [Online; Accessed 20 June 2019].

[2] Apache, Flink: Stateful Computations over Data Streams, 2014. URL https://flink.apache.org. [Online; Accessed 20 June 2019].

[3] Apache, Pig, 2008. URL https://pig.apache.org/. [Online; Accessed 20 June 2019].

[4] Apache, The Apache Hive data warehouse software, 2010. URL https://hive.apache.org. [Online; Accessed 20 June 2019].

[5] J.P. Morrison, Flow-Based Programming, 2nd Edition: A New Approach to Application Development, CreateSpace, Paramount, CA, 2010.

[6] T. Mahapatra, High-level Graphical Programming for Big Data Applications, Technische Universität München, München, 2019.

[7] T. Mahapatra, I. Gerostathopoulos, C. Prehofer, S.G. Gore, Graphical Spark Programming in IoT Mashup Tools, in: 2018 Fifth International Conference on Internet of Things: Systems, Management and Security 2018, pp. 163–170, http://dx.doi.org/10.1109/IoTSMS.2018.8554665.

[8] T. Mahapatra, C. Prehofer, I. Gerostathopoulos, I. Varsamidakis, Stream Analytics in IoT Mashup Tools, in: 2018 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2018, pp. 227–231, http://dx.doi.org/10.1109/VLHCC.2018.8506548.

[9] T. Mahapatra, I. Gerostathopoulos, F.A. Fernández, C. Prehofer, Designing Flink Pipelines in IoT Mashup Tools, in: Proceedings of the 4th Norwegian Big Data Symposium, NOBIDS, vol. 2316, no. 03, 2018, pp. 41–53, URL http://ceur-ws.org/Vol-2316/paper3.pdf.

[10] T. Mahapatra, C. Prehofer, aFlux: Flow-based Programming for Big Data, 2019. URL https://aflux.org. [Online; Accessed 20 June 2019].

[11] G. Agha, Actors: A Model of Concurrent Computation in Distributed Systems, MIT Press, Cambridge, MA, USA, 1986.

[12] T. Mahapatra, I. Gerostathopoulos, C. Prehofer, Towards Integration of Big Data Analytics in Internet of Things Mashup Tools, in: Proceedings of the Seventh International Workshop on the Web of Things, WoT '16, ACM, New York, NY, USA, 2016, pp. 11–16, http://dx.doi.org/10.1145/3017995.3017998.

[13] T. Mahapatra, C. Prehofer, Service Mashups and Developer Support, Digit. Mob. Platforms Ecosyst. (2016) 48–65, http://dx.doi.org/10.14459/2016md1324021.

[14] J. Mineraud, O. Mazhelis, X. Su, S. Tarkoma, A gap analysis of internet-of-things platforms, Comput. Commun. 89–90 (2016) 5–16, http://dx.doi.org/10.1016/j.comcom.2016.03.015.

[15] T. Stahl, M. Voelter, K. Czarnecki, Model-Driven Software Development: Technology, Engineering, Management, John Wiley & Sons, Inc., USA, 2006.

[16] F.A.F. Moreno, Modularizing Flink programs to enable stream analytics in IoT Mashup tools = Modularización de programas Flink para el análisis de datos en tiempo real en herramientas de Mashup para IOT, Telecomunicacion, 2018, URL http://oa.upm.es/52898/.

[17] Apache, Maven, 2004. URL https://maven.apache.org. [Online; Accessed 18 July 2019].

[18] Git, Git. distributed-is-the-new-centralized, 2005. URL https://git-scm.com. [Online; Accessed 18 July 2019].

[19] Eclipse, The platform for open innovation and collaboration, 2001. URL https://www.eclipse.org. [Online; Accessed 18 July 2019].

[20] JetBrains, Intellij IDEA: The java IDE for professional developers by jetbrains, 2001. URL https://www.jetbrains.com/idea/. [Online; Accessed 18 July 2019].

[21] MongoDB, The database for modern applications, 2009. URL https://www.mongodb.com. [Online; Accessed 18 July 2019].

[22] H. Eichelberger, C. Qin, K. Schmid, Experiences with the Model-based Generation of Big Data Pipelines, in: B. Mitschang, D. Nicklas, F. Leymann, H. Schöning, M. Herschel, J. Teubner, T. Härder, O. Kopp, M. Wieland (Eds.), Datenbanksysteme für Business, Technologie und Web (BTW 2017) - Workshopband, Gesellschaft für Informatik e.V., Bonn, 2017, pp. 49–56.

[23] W.d. Santos, G.P. Avelar, M.H. Ribeiro, D. Guedes, W. Meira Jr., Scalable and Efficient Data Analytics and Mining with Lemonade, Proc. VLDB Endow. 11 (12) (2018) 2070–2073, http://dx.doi.org/10.14778/3229863.3236262.

[24] S. Schmid, I. Gerostathopoulos, C. Prehofer, QryGraph: A Graphical Tool for Big Data Analytics, in: SMC'16, 2016.

[25] K. McCormick, D. Abbott, M.S. Brown, T. Khabaza, S.R. Mutchler, IBM SPSS Modeler Cookbook, Packt Publishing, 2013.

[26] Touk, Nussknacker. Streaming processes diagrams, 2019. URL https://touk.github.io/nussknacker/. [Online; Accessed 27 May 2019].

[27] Stratio, The definitive visual build tool for Apache Spark: Sparta 2.0, 2019. URL https://www.stratio.com/blog/apache-spark-visual-tool-sparta/. [Online; Accessed 20 June 2019].

[28] Streamanalytix, Self-Service Data Flow and Analytics, 2019. URL https://www.streamanalytix.com. [Online; Accessed 20 June 2019].

[29] Santander City Council, Santander Open Data - REST API Documentation, 2018. URL http://datos.santander.es/documentacion-api/. [Online; Accessed 01 June 2018].