Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI **ISIN TECHNICAL REPORT**

Poma

A Lightweight Framework for Distributed Processing

Author(s)

Amos Brocco[1]
Vanni Galli[2]

Technical Report Number

2018-2

Date

January 22, 2018

[1] amos.brocco@supsi.ch
[2] vanni.galli@supsi.ch

Abstract

Real-time data processing frameworks are concerned with continuous flows of information which need to be processed as fast as possible. In order to cope with large amounts of data, parallel and distributed solutions are employed to partition the processing task and execute analysis and transformation steps on multiple threads or on a multitude of computing devices. In this regard we introduce Poma, a lightweight framework for parallel and distributed processing using modular pipelines of composable modules, which provides both a development kit and a simple job management middleware for distributed operation.

1 Introduction

Processing and analyzing information in the realm of big data and the internet of things introduces several computational challenges related to the number of sources and the variety and velocity of information [1]. Stream processing [2] tackles the problem of continuously analyzing and transforming big amounts of data as fast as possible (possibly in real-time) using different computing nodes. In contrast to dedicated high performance computing systems, distributed stream processing frameworks can be easily deployed on a variety of computing devices, even embedded platforms (such as the NVIDIA Jetson[3]) or off-the-shelf commodity hardware. These technologies are thus suitable for a multitude of problems where horizontal scalability is preferred over a vertical approach. [3], both for large scale big data processing as well as in more marginal scenarios. In this regard, we introduce a novel middleware and framework which aims at providing a lightweight solution for implementing parallel and distributed data processing systems. Our focus is on modularity and flexibility through composability: computational tasks are divided into multiple steps and implemented as dynamic modules which can be rearranged depending on the available resources and the deployment scenario.

2 Related work

Nowadays there exist several frameworks and systems for supporting distributed processing of real-time data. The most influential research and development activities take place in the field of big data analytics, hence this brief review of other works related to our project will first pay attention to this area. The most known distributed computing framework is probably Apache Hadoop [4], which implements the MapReduce programming paradigm and supports parallel batch processing of large amounts of data, typically stored on a distributed filesystem. Unfortunately, this approach is not suited for real-time stream processing and for low-latency applications. In particular, with MapReduce each processing step stores its (intermediate) result on the distributed storage, worsening the overall computing performance. Higher throughputs can be achieved by performing all the processing in-memory, as implemented by Apache Spark [5]. To overcome the limitations of batch processing alternative solutions have also been developed over the years. Apache Flink[6] provides a framework for unified batch and stream processing, with low latency and high throughput. Apache Storm [7] is a cluster solution which allows for processing unbounded streams of data. Similarly Apache Kafka Streams [4] handles real-time data feeds using Kafka [8] as messaging layer. These platforms can be used to build robust applications to transform information or react to events, however the downside resides in their complexity and relatively large footprint. In contrast, the framework

[3] http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html
[4] https://kafka.apache.org/documentation/streams/

presented in this paper provides a lightweight solution for parallel and distributed processing, drawing inspiration from the pipeline concept, which is also used in specific solutions such as GStreamer [9].

3 Poma Framework Concepts

Poma is a framework for building data processing pipelines made of composable modules, as well as a lightweight middleware to support distributed execution (Figure 1). The framework and all supporting tools are written in C++ and only depend on Boost [5] and ZeroMQ [6]. The source code of the framework[7] is released under a 3-Clause BSD license. Pipelines are configured by means of JSON files which describe data flow graphs: these files are loaded by a component called *Loader* which instantiates the required modules and starts executing the pipeline. In a distributed setup, a *Deploy* tool is used to partition the pipeline and submit JSON configurations to a *Service* component installed on each node: the latter subsequently starts a *Loader* to manage the execution of a portion of the pipeline. The *Service* component acts as a job manager, allowing for multiple pipelines to be executed concurrently on the same machine. The *Loader* also comes as library which can be linked to other programs.

3.1 Data processing modules

Pipelines describe a data processing flow composed of dynamically loadable modules written in C++. Each module implements a procedure used to react to incoming data and eventually submit processing results to other modules down the pipeline. Data exchanged between modules is organized into packets, which contains a JSON metadata tree and a custom payload whose type can be freely defined by the user (for example, for video processing purposes the payload might be an image). The data flow (path of each packet) is determined by links between modules, which are described in the pipeline configuration file. To allow for dynamic routing of data packets, each link is associated with a *named channel*: depending on the behavior of each module, data packets can be send to distinct channels, allowing for different processing tasks to be carried out.

3.2 Graphical data flow editor

To support the creation and editing of data flows we provide a simple graphical tool (Figure 2) which enables the user to compose pipelines by dragging and dropping objects on a canvas. The editor relies on a dynamically generated file to obtain a list of all existing modules and their configuration options, hence additional components developed by the user are seamlessly integrated.

3.3 Parallel execution

By default each pipeline operates in a synchronous way: when a data packet is forwarded to another node the sender remains blocked until the processing task has been completed, introducing large latencies. In order to process data in an asynchronous way, the user can employ *Buffer* modules at different stages of the data flow: buffers free the sender as soon as a packet has been queued. Dividing the pipeline into different parts by means of buffers achieves a primitive form of parallel execution, which is commonly referred to as *pipelining*, since each part is executed in a different

[5] http://www.boost.org/
[6] http://zeromq.org/
[7] https://github.com/slashdotted/PomaPure

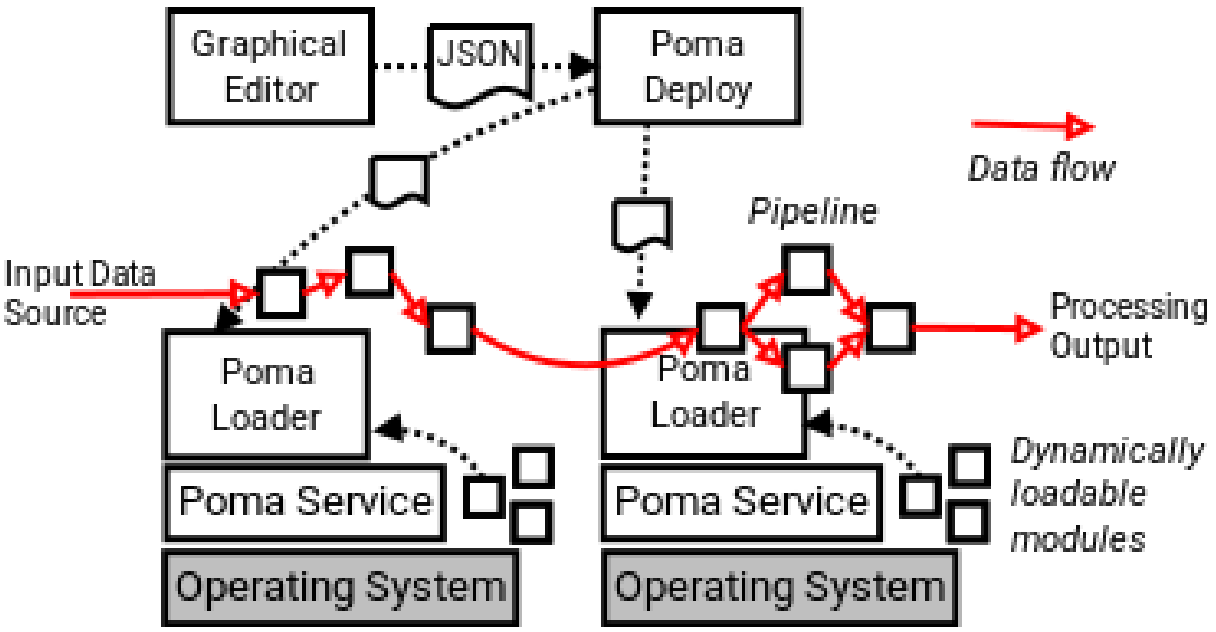

Figure 1: Overview of the framework

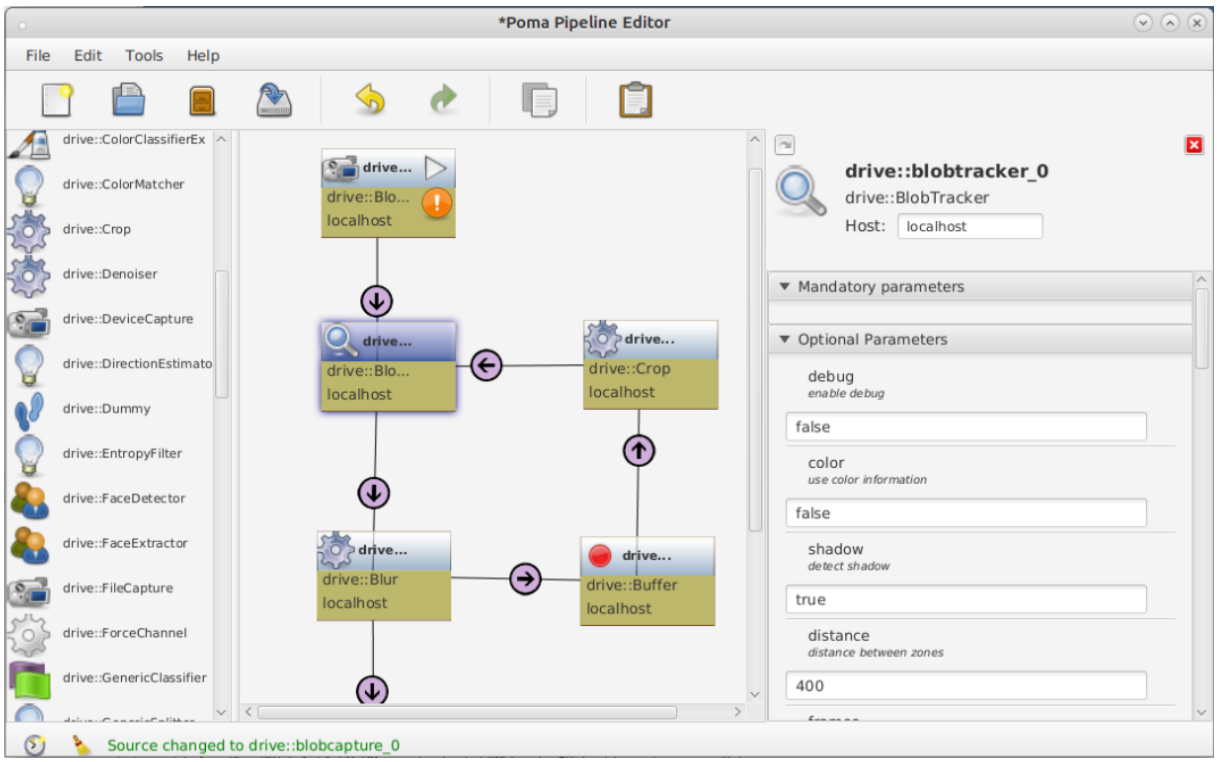

Figure 2: Graphical tool for editing the data flow

thread. Specific sections of a pipeline can also be executed on multiple threads by means of the *ParExecutor* and the *Joiner* modules, which implement the fork-join paradigm. Finally, if the data to be processed originates from several source modules it is possible to assign each of them to a

different thread using a *ParProcessor* module.

3.4 Distributed execution

Poma also supports distributed execution of a pipeline, for example on a cluster of computing devices. Each node requires the *Service* and *Loader* components and dynamically loadable modules used by the pipeline. Data communication between modules on different machines relies on ZeroMQ and is mainly transparent to the user: only serialization and deserialization procedures suitable for the data type contained into each packet need to be implemented. Assigning sections of a pipeline to a different machine is achieved by setting an *host* attribute in the pipeline configuration file. One of the hosts is used to deploy and control the distributed pipeline.

4 Conclusion and future work

In this paper we presented Poma, a framework for real-time parallel and distributed data processing. In contrast to other processing frameworks, Poma is a lightweight customizable solution suitable for embedded computing devices. Processing is performed by dynamically loadable modules, whereas data flows are defined through simple JSON files which can be created and edited using a graphical tool. A real world deployment of Poma is currently carried out in the context of an automated video surveillance platform. Future work includes the development of profiling tools and the implementation of secure data transmission for distributed deployments.

References

[1] Jianqing Fan, Fang Han, and Han Liu. Challenges of big data analysis. *National Science Review*, 1(2):293–314, 2014.

[2] Henrique C. M. Andrade, Bugra Gedik, and Deepak S. Turaga. *Introduction to stream processing*, pages 33–74. Cambridge University Press, 2014.

[3] K. Bakshi. Considerations for big data: Architecture and approach. In *2012 IEEE Aerospace Conference*, pages 1–7, March 2012.

[4] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[5] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016.

[6] Ellen Friedman and Kostas Tzoumas. *Introduction to Apache Flink: Stream Processing for Real Time and Beyond*. O'Reilly Media, Inc., 1st edition, 2016.

[7] Jonathan Leibiusky, Gabriel Eisbruch, and Dario Simonassi. *Getting Started with Storm*. O'Reilly Media, Inc., 2012.

[8] Nishant Garg. *Learning Apache Kafka, Second Edition*. Packt Publishing, 2nd edition, 2015.

[9] Wim Taymans, Steve Baker, Andy Wingo, Ronald S. Bultje, and Stefan Kost. *GStreamer 1.8.3 Application Development Manual*. Samurai Media Limited, United Kingdom, 2016.