Real time big data processing

Ilia Sheiko

Bachelor thesis

Czech Technical University in Prague
Faculty of Electrical Engineering

2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Sheiko** Jméno: **Ilia** Osobní číslo: **452825**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Zpracování velkých dat v realtime režimu

Název bakalářské práce anglicky:

Real time big data processing

Pokyny pro vypracování:

Seznam doporučené literatury:

1. Saurabh G. ; Shilpi S. 2017. Practical Real-time Data Processing and Analytics, 1 edition. Packt Publishing. ISBN: 978-1787281202
2. Byron E. 2014. Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data. 1 edition. John Wiley & Sons. ISBN 978-1118837917
3. Kalavri V. ; Hueske F. 2019. Stream Processing with Apache Flink. 1 edition. O´Reilly Media, Inc. ISBN 978-1491974285
4. Friedman E. ; Tzoumas K. 2016. Introduction to Apache Flink. 1 edition. O´Reilly media, Inc. ISBN 978-1491977132
5. Garillot F. ; Maas G. 2017. Stream Processing with Apache Spark. 1 edition. O´Reilly Media, Inc. ISBN 978-1491944240
6. Stopford B. 2018. Designing Event-Driven Systems. 1 edition. O´Reilly Media, Inc. ISBN: 978-1492038252
7. Shapira G. ; Narkhede N. ; Palino T. 2017. Kafka: The Definitive Guide. 1 edition. O´Reilly Media, Inc. ISBN: 978-1491936160
8. Kleppmann M. 2016. Making Sense of Stream Processing. 1 edition. O´Reilly Media, Inc. ISBN: 978-1492042563

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Marek Sušický, katedra počítačů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **18.04.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **19.02.2021**

Ing. Marek Sušický podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

Abstract

The purpose of this bachelor thesis is to understand principles of the events stream processing using big data technologies, analyse requirements on a system that provides services of this processing via a user-friendly interface, find and examine an open-source solution that fulfill these requirements, prove this fulfillment, learn to use, develop and build it.

Key words: complex event processing, event stream processing, real time, big data, Flink, Nussknacker

Cílem této bakalářské práce bylo porozumět principům proudového zpracování událostí v kontextu velkých dat, zanalyzovat požadavky na systém, poskytující služby tohoto zpracování prostřednictvím uživatelsky přívětivého rozhraní, vyhledat a prozkoumat open-source řešení, které tyto požadavky splňuje, prokázat toto naplnění, naučit se systém používat a rozvíjet.

Klíčová slova: zpracování komplexních událostí, proudové zpracování událostí, reálný čas, velká data, Flink, Nussknacker

Declaration / Prohlášení

Hereby, I declare that I have composed the presented paper independently on my own and without any other resources than the ones indicated.

Prohlašuji, že jsem predloženou práci vypracoval samostatne a že jsem uvedl veškeré použité informacní zdroje v souladu s Metodickým pokynem o dodržování etických principu pri príprave vysokoškolských záverecných prací.

In Prague / V Praze 24. 5. 2019

Acknowledgements / Poděkování

I thank all the close people for support; I am glad you are with me, and I am with you. Thank Ing. Marek Sušický for material comments, incredible benignity and the opportunity to work on such project with such technologies. I will not forget you.

Děkuji všem blizkým lidem za podporu, děkuji, že jste se mnou a já jsem s vámi. A děkuji pánovi Ing. Marku Sušickýmu za věcné připomínky, neuvěřitelnou dobrotivost a možnost pracovat na podobném projektu s podobnými technologiemi. Nezapomenu na vás.

Contents

1 Introduction

In this section I describe the motivation of the work, it's purpose and general way to reach it.

1.1 Motivation

Nowadays, there are many areas of systems where people expect immediate processing of different data. They demand lower latency in interaction with interfaces, as actual as possible time-sensitive information and the immediate reaction of automated systems on different events. Also, the amount of processed information is getting higher every year.

A particular type of explicit performance demanding systems are "complex event stream processing" systems that receive a continuous flow of related time-sensitive data from several sources, evaluate them and produce any outputs in the required time. Big customers like banks, logistics, mobile and trade companies are often interested in significant investments in such systems for their needs. Of course, they expect a high quality of the product, long-time support and acceptable usability.

Recently, I have been working for a major Czech company Profinit EU Ltd. (later in the text "The Company") that provides different information technology services like custom software development, enterprise integration, BI/DWH and Big Data development etc.

Generally, The Company needs a system for the fast processing of various abstract events. The system should receive information, enrich it with data from different stores, evaluate it and product any outputs: send emails, store logs, send something to other systems etc. Inputs, outputs, additional data sources and rules for evaluation have to be easily configured without actual programming and console using by business men. The system should be prepared to compute a significant amount of data with very low latency. Also, the system should be extensible and durable to data overload and crushes. Probably, the most crucial point is that the system has to bring benefits by its selling to customers. Precise requirements defined in the "Requirements" section.

1.2 Purpose

The purpose of this work is to find an open-source system that fulfill the company's requirements, can be sold to customers and its evolving is cheaper then a completely own solution. If such system will not be found I should design it.

I will analyze the business and technology domains of the given problem: it should be understood where and how are real-time systems may be used, what are they made of. Then I will determine particular business and system requirements of the needed software. After that I will define recherche request and use it in web sites like Google and GitHub. Found systems that fulfill major requirements will be inspected and deployed in a respective environment. Then

I will test systems by some prepared use-cases, choose methods for usability and stress tests and perform them. Based on gotten information I will choose the most proper system, circumstantially describe it's problems, analyze possibilities and needs of it's involving and maybe prepare a plan of the further steps.

2 Problem analysis

In this section I succinctly summarize the main topics needed to understand the problem area and their relations with some abstract requirements on the possible system.

2.1 Main definitions

At first, there should be defined the most basic concepts like “event", “complex event processing", “real time", “big data", “open source" etc.

2.1.1 Event processing

According to Cambridge English dictionary, event is "anything that happens, especially something important or unusual"[2]. In business area events are important in business sense. Complex events are events consist from some parts, other events, and/or information from several sources.

Complex event processing (CEP) consists of identifying and correlating the most significant events from their variety, analysing the positive or negative impact and taking proper actions in real time. It requires appropriate event monitoring, reporting, registration and filtering.

Processing complex events refers to changes (and their rapidity) of:

- Values
- States
- Behavior

and others.

Let's examine an example of non-business complex processes. Imagine these three related events happening to a human:

1. The human's nose becoming stuffy or runny. (a change of state)
2. The human becomes feeling fatigue. (a change of state)
3. The human sneezes and/or coughs more then often. (a change of behavior)

The human may feel fatigue because he worked hardly and had not rest for significant time, he may sneeze and cough because he cuts or roasts very hot peppers, his nose may be cold because of allergy. Being apart and/or not very

hard these events doesn't mean that the human's highest priority should be taking care of his own health.

But we can create a formula related to the events' intensity, define categories for each indicator and points per them. Depending on the output sum of points enriched by information of the human's bill we can consider what to do, to work or not to work, call to a doctor, come to a hospital or heal independently by drugs (or herbs). Also, if there is only fatigue but not sneezing and problems with a nose, there should be considered another possibilities.

CEP systems and technologies are intended for automatic evaluation of such events via communicating with different data sources, enriching input data by additional storage, implementing of rules given by an user and production of a proper output.

When a system consumes a continuous flow, known as stream, of events that must be processed it becomes an event stream processing (ESP) system. For such systems exist special technologies oriented on processing streams of data. Historically ESP technologies were not a subset of CEP intended ones, but now it is difficult to separate them. The main difference is that normally CEP (and not all of ESP) systems have crucial requirements for the latency of operations.

In this work I use CESP definition a few times as Complex Event Stream Processing (system or processing) when streaming property is essential.

2.1.2 Real time

Real time is understood as a quantitative value that may be measured by a real physical clock. In contrast, logical time is qualitative value characterizing a relational order of operations.

According to the Oxford English Dictionary, real time processing systems are systems that have strict requirements for the timing. In other words, these systems are oriented on the results production in a limited time.

The main characteristic of the real time systems is latency. It may be understood as response time (delay time) to and external event, the time between receiving any input by a system and producing a computed output. Deadline is a critical maximum latency allowed to an operation.

Real time systems are derived into the following categories by requirements for latency:

1. Hard - latency above a deadline is a major system failure.
2. Firm - certain (small) part of operations with latency above deadline is acceptable, but computed result loses entire relevance.
3. Soft - certain part of operations with latency above deadline is acceptable, computed result partially loses relevance.

Normally real time processing mistakenly understood as a processing nearly without any latency.

Methods and technologies used for real time computing and processing are described in the subsection “Technology Domain".

2.1.3 Big data

Big data is a widely interpreted definition of information technologies and methods intended for horizontally scalable storing and processing immense amount of data and these data themselves. Horizontal scaling is system capability escalation via increasing of processing or memory units' count opposite to increasing the efficiency of them - vertical scaling.

Three “V" are traditionally distinguished as defining characteristics for big data:

1. Volume - physical amount of data. Normally "big" means terabytes and petabytes of them. Or more.
2. Velocity - data growth rate and required processing speed.
3. Variety - opportunity to simultaneously process various structured, unstructured and semi-structured data.

Also categorize additional “V -aspects" such as:

- Viability - latent correlation dependencies of data within an object (e.g. complex event).
- Value - economical and/or science effectiveness of a data unit.

and others.

The most common big data solutions and technologies, especially intended for real time event processing systems, are described in the subsection “Technology Domain".

2.1.4 Open source

Open source software is software with published non compiled source code and a licence that allows to:

- Study used algorithms and technologies.
- Take a part in the development of the software itself.
- Use the code in own development with limitations of license.
- Ensure the software quality, safety and compliance with the stated documentation.

There are few widely used open-source licences that will be described lately.

2.2 Business domain

To analyze possible requirements on the solution, in this subsection I describe the common using of CEP/CESP systems in business area, special business traits of such systems and how they may be used. The recherche is already oriented on the given business target described in the introduction.

2.2.1 Common business CESP using

Initially CEP/CESP systems were used mainly in financial industry, but now they may be implemented in every area with continuous event production. For example, fraud detection, logistics, marketing, analytics. Mostly such systems are used in larger business companies to hold and control sophisticated processes. Let's examine closer fraud detection in mobile telecommunication area to understand the key points and pattern of the possible CESP system using.

As any widely used information communication channel, mobile may be an object of eminently damaging frauds. There are few simple examples that may be detected by CESP:

- Spam. Sending of advertising messages without agreement of the recipient is normally illegal. In addition, they may have an extortionate nature, for example, promise to receive a prize in the lottery with requisition to pay a small organization fee or taxes. Occasions of mass sending of unpleasant messages happen quite often and decrease the veracity of the mobile operator and may cause more unpleasant consequences. Additionally, a person with incoming calls paid in favour of that person, may do short calls to different phone numbers and drop them without answer with expectation that somebody call back. In order to prevent such events may be controlled counts of outgoing messages/calls and unique recipients per time unit.

- SIM cloning. There are several techniques of remote hacking and cloning of SIM cards, as well as requiring physical access of the cloned card, including special software and hardware. These events may be detected by monitoring of the SIM's location changes, being in two places simultaneously, unusual banking transactions, mass password recovery requests. If there are a lot of uncommon events happening the holder of the SIM may be requested for a person identification via secret question, alternative ID or something. Also if a card changed its location in impossible time, it may be immediately blocked.

- SIM swap. Via various techniques of social engineering a fraudster can obtain the information required for a SIM card recovery and call the operator for that. Prevention of mass malicious aftermaths demands a specific control in minimally several days after the card changing. The control principles may be similar to those illustrated in the description of SIM cloning.

These examples of fraud primarily cause damage to the customers of the company, but this damage is urgent and mobile companies are interested in happiness of their customers. If a customer loses a trust to a company he change it. In addition, loud scandals can cause a fall in stocks of the company. So, let's conclude, that mobile companies care about their customers.

So, how may a company use a CESP system to prevent fraud damage?

2.2.2 Business example of CESP using

An authorized person/people of the company creates a process for the singular event detection, adds channels of input data incoming, that cause a computation, maybe enriches them with additional data, sets proper evaluation rules and determine, what should happen under certain conditions. Let's analyze the variables of this description in business sight.

- How the rules, inputs, outputs and other elements of processes may be set? Ideally, it should be the fast, easy, but effective way in intention to reduce cost of the work. Graphical user interface is desirable where it is possible.
- Who may be that authorized people? At first, internal workers of the company who understand the business and the problem well. Probably, they will be technically educated, otherwise why should they control large and serious information processes? But ideally they should not be required to have strong knowledge of concrete programming languages or technologies. Also, it may be a combination of business and technical men. May be it will be a hired person with high cost. Anyway, their time that can be saved, must be saved until it brings benefits.
- Which of the inputs that cause processing there may be? Other systems sending messages mainly.
- Which of the outputs there may be? SMS, emails, logs, a notification on the screen, other systems that may perform any operation in case of a happened event.
- Which additional sources may require a user? Databases and another systems eventually.
- Which rules there can be? Different combinations of if/else/then structures. Also they would need an easy way to aggregate available data.
- What should happen if condition of the rule is positive? May be sent a message to an another system that perform a defined reaction on the detected event, also may be sent SMS/email to a stakeholder or a system operator.
- Should be done something after the process creation? It should be tested. In case of work in enterprise, it should be tested in a safe environment with test data and outputs.

Let's return to the fraud detection area and imagine a use case of a CEP system using, for example, a creation of a rule for SMS spam detection.

An authorized person:

1. Creates a process.

2. Defines a synthetic input, may be special visible flow of realistic singular SMS data purposed by the company for the tests. Input data is a set of the sender's ID and recipient's ID. Synthetic input must have some known spam events.

3. Defines a request for the SMS count of the given sender for the last 12 hours.

4. Defines a rule with request: if the gotten count of SMS is greater then 1000, then the system makes a request for the count of unique SMS recipients for this sender.

5. Defines a rule: if the count of unique recipients greater then 100, then the system writes to logs the senders ID.

6. Saves the process.

7. Deploys the process. Leaves it for the 12 hours.

8. Returns to the test and stops the process.

   (a) If results in logs are correct, then connects real data input and output. Input would be a data stream of all SMS or their defined share. Outputs are a system, that blocks a phone number, and a system that sends him a notification.

   (b) Until results in logs are not correct improves defined rules and retest the process

This example has many possible technical problems, should not provide a business successful, but in the very primitive form it would be something like that. A more realistic example is described in the next subsection "Technology domain"2.3

2.3 Technology domain

2.3.1 Big data

Remember, big data technologies resolve such problems as:

1. Growing amount of stored and processed information - volume.

2. Growing speed of data generation and required processing - velocity.

3. Processed information heterogeneity - variety.

via methods and technologies oriented on escalation of memory or processing units' count - horizontal scaling.

Horizontal scalable systems are based on distributing of singular servers' work to groups of them - clusters, where machines (nodes) work together. They may do an independent part of the common work or do identical operations.

If nodes replicate each other, the operation they performed is durable against node crushing until there is a functioning replica. If they are able to effectively compute independent dividable parts of the bigger task, then the task and it's computation may be scaled near to infinitely. A good model of such system and a base of many big data techniques is MapReduce.

MapReduce computing consists in three phases:

1. Map. A node obtains a part of the whole input data and prepares it - makes a map of <key, value1>.

2. Shuffle. A node obtains each map <key1, value1> from the Map with a defined key1 phase, combines their values to the list and sends to the next phase a map of <key, list(value1)>

3. Reduce. A node obtains maps of <key, list(value1)>, reduces the list(value1) to value 2 and sends pairs of <key, value2> to the output.

The final output is a map of <key, value2> pairs.

Map and Reduce phases may be scaled infinitely, Shuffle phase is limited by the count of the possible keys - there cannot be more nodes, then keys.

There are described the main big data technologies important for the project. All of them are open-sourced and founded by the Apache software foundation.

Hadoop - a platform, framework and a set of utilities for distributed systems developing and running. Consists in four modules:

1. Hadoop Common - the main and joining set of software libraries and utilities used for other modules and related projects

2. Hadoop distributed file system or HDFS - is a file system purposed to store large files via distributing of them between nodes by blocks. All blocks in HDFS except for the last one per file have the same size (normally 64MB) and can be placed on several nodes. The block size and replication ratio (the number of replica nodes where a block may be saved) is configured. Files in HDFS are binary and are read consequentially, not via random access. That may grant a much greater reading speed for the cost that files can be written on the physical disk only once (modification is not supported and files may be completely rewritten only), and processes cannot write one file simultaneously.

   Deployed HDFS consists in:

   - Name node that stores file system metadata and information about block allocation, responsible for opening and closing files, directories manipulating.
   - Group of data nodes that store file blocks, responsible for writing and reading singular blocks.

   Administrative functions are available from the command line interface.

3. Yet Another Resource Negotiator or YARN - module intended to control cluster resources and plan tasks. Under the control of YARN can be computed both MapReduce programs and any other distributed applications that support the appropriate software interfaces.

4. Hadoop MapReduce - a Hadoop implementation of MapReduce paradigm, a platform for programming and performing MapReduce calculations.

Also there are many significant technologies based on Hadoop. For the project may be important such database platforms:

Apache HBase - a non-relational distributed database platform. Tables in HBase consist in rows and columns. Columns are not mandatory per rows, every row may have any combination of the columns. Columns may be grouped in “column families". Rows have unique keys and sorted by them in the table. Data are versioned and stored in cells, addressed by a combination of the row's key, column and version. Data are stored in HDFS, in bytes, so, every serializable data type may be stored. Also row's cells with the different column types may be stored on the separate nodes.

HBase engine provides create (put), read (of the row by key and scanning through rows by defined parameters) and delete operations. Update may be performed via put some data in the existing row that causes rewriting of the whole block of data. Joins are not supported.

Apache Hive - a database management system intended to write, query and aggregate data in Hadoop environment via SQL-like commands. Designed mainly for warehousing tasks - analysis of constant data in massive amount.

Now let's inspect opportunities for big data stream processing.

2.3.2 Streams

There are two ways which big data systems use to process terabytes and petabytes of data per day: in stream and in batch.

Batch processing consists in (distributed) computing of the larger amounts of data that are already available. It is the way to process data where processing has no sense without all data set (for example any "per day" analysis) and there are not high requirements for latency. Of course, a system must to have enough of memory resources.

Stream processing systems is purposed for strongly defined operations on the "infinitely" incoming data, where requirements for latency are important. The idea consists in creating an entity which architecture fully oriented on continuous determined data consuming, computing and output producing, on the strictly defined operation. That grants to the entity a significant increase of processing velocity at the cost of the it's flexibility.

There are few famous open-source streaming platform based on Hadoop.

At first, must be renown Apache Kafka - a distributed message transfer platform. The main purpose of Kafka is reliable data transmission between systems. The main entity in Kafka is topic, a queue/pipeline of incoming and outgoing binary messages. Data come from "producers", outgo to "consumers".

Consumer may be organized to consumer groups. Kafka cluster guarantees that every received message will be consumed by exactly one of the consumers from every consumer group and only once. Messages received by a Kafka cluster may be replicated between nodes to ensure the system's durability against node crushing. Undelivered messages are cached in file system. In addition, Kafka has its own API that allows to develop stream processing beans in standart Java and Scala programs without separate cluster for that needs. But there are better opportunities such as Apache Spark and Flink.

Apache Spark and Apache Flink are both big data computing frameworks. Both of them are intended to distributively process massive amount of data with perfect performance. Both of them can consume data from Kafka, perform on them aggregation operations, filters, mapping etc. I addition both of them support stateful processing - internal saving and using of the intermediate results (states) of any computing for a defined messages window. But there are two essential differences:

1. Spark is a batch processing platform and stream processing is simulated via micro-batch processing of data sets. Flink processes true streams (but may combine input data to batches) that may grant significant decrease of latency.

2. Spark stores states as a dataset, Flink stores them as a map. Spark scans the whole dataset for a specific state, Flink directly uses its key, that grants much of the performance if a state store is large. Also Flink has a connector for state storing in a database - RocksDB.

I conclude, that for an abstract CESP processing system should be rather used Flink by reason of high requirements on latency and expectation of massive state data. But it can be considered.

Flink processes have the form of "jobs" - jar packaged Java programs that runs and regulates a Flink cluster. Each Job has own data flow with input, outputs, hold states in "state backend", custom variables.

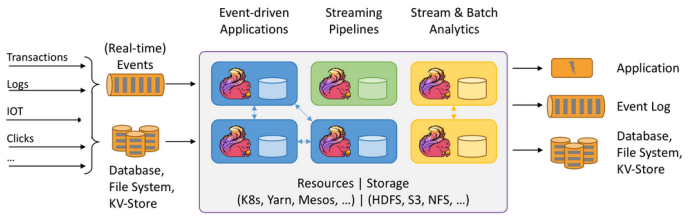

Figure 1: Flink architecture picture from the project web [24]

2.3.3 Licenses

There are the the most commonly used licenses in open source world:

- MIT
- Apache 2.0
- GPL (GPLv3, GPLv2, LGPL, Affero GPL)

All of them permit commercial and patent use, but GPL licenses require to hold the same licence in derivative works.

2.3.4 Other essential technologies

There must be renowned some additional technologies.

Kerberos - a network authentication protocol. Based on symmetric key cryptography. A master (authentication) server authenticates a client and grant him authorization tickets for using other system servers' services. Kafka and Flink have built-in support of Kerberos for creating secure data channels, Hadoop/HDFS and another systems accessing.

InfluxDb - a time related database. May be used by Flink to store metrics.

Grafana - dashboard system for different metric collection services

RocksDB - Key/value database, may be used by Flink to store state backends

2.3.5 Extended example of CESP using

Return to the problem of the fraud detection area: it is needed to detect SMS spam events with an assumptions, that at least 1000 SMS addresed to at least 100 unique recipients is a reliable attribute of such events and there is a Hive database with data about SMS' senders, recipients and time of the sending.

An user:

1. Assures Kerberos authentication of the Kafka and Flink clusters if needed.
2. Creates a Flink job.
3. Defines the input - subscription on a Kafka topic with test data. Input data are JSONs with a sender's ID.
4. Defines state for SMS count storing with 12 hours time window keyed by the sender's ID. Defines the state update and include the variable "SMSCount" into the data flow.
5. Defines a filter: IDs with SMSCount greater then 1000 continue flow.
6. Defines a request to Hive database with SMS data for the unique recipients sent for last 12 hours by the sender. Includes this varirable named "recipientsCount" to the flow.
7. Defines a filter: IDs with recipientsCount greater then 100 continue flow.
8. Defines a test output, for example, just logs of IDs in filesystem.
9. Tests the job on Flink cluster. Improves it.

10. Sets real input Kafka data stream.

11. Sets a producer of the IDs to a Kafka topic as an output.

This example may have technical problems as well as business ones. For example, if a person sends a lot more then 1000 SMS to a lesser count of unique recipients then 100, the system makes a lot of useless requests to Hive, and it can be resolved by a timeout state or rules redefinition.

Now with studied technology and business domains can be determined real requirements of the project.

3 Requirements

In this section I describe the method of the requirements gathering and defined requirements on the final solution and a system that I search in this work. Remember, the general purpose is to find an open-source CESP system with a simple process creating which is not complete now, but can be finished by the Company and profitably sold.

3.1 Introduction to requirements

For each requirement is defined its identifier, type and content. Types categorization are following:

- Business (BR) - non-technical requirements, major changes that should do the project's solution, specific targets for the general business purpose reaching. General business purpose here is resolving abstract problems that may be resolved by CESP.
- System - requirements describing the result system, specifically:
  - Functional (FR) - the functions, that should have the system
  - Non-functional (NFR) - quality and limitations, requirements on usability, reliability, performance, security, extensibility, used technologies, licences etc.
  - Transitional (TR) - requirements on the solution deploying - data migration, user training etc.
- Requirements for the searched system (SSR) - because The Company would prefer to make a solution based on an open-source system, I independently describe requirements for that system. They are mostly about which of the system requirements should be already implemented, what will cost to implement others.

There were used two sources of requirements: my own brainstorming and the Company as an imaginable customer (specifically the tutor of this work). Also, there must be defined the stakeholders of the final system:

- Regular process master - a main creator and holder of the processes. Must not be an expert in programming and console using. Understands business.
- Programming specialist - a secondary creator and holder of the processes. Also controls processes' quality in technical sight. Is educated in programming, ideally, knows the language in which the solution is written, understands mechanisms of the memory using. Educated in Hive.
- Admin - deploys and holds all the systems. May be a programming specialist.
- Business owner - has money and a business problem to be solved.

3.2 Business requirements

BR-1 - as a process master, I need to create, run, read, update, copy and delete processes.

BR-2 - as a process master, I need a simple process management tool for process manipulating.

BR-3 - as a process master, I need possibilities for the simple creating of inputs, outputs, switches, filters and custom variables in processes.

BR-4 - as a process master, I need arithmetic, logic and aggregation operators in context of any data manipulation.

BR-5 - as a process master, I need possibilities for database queries and timestamp evictable values creating with the help of a programming specialist.

BR-6 - as a process master, I need universal input channel. I may need following inputs/triggers of processing:

- Clicks in another systems
- Transactions
- Visited pages
- Banners in the process management tool
- Messages to another systems

BR-7 - as a process master, I need following outputs:

- SMS
- Push notification
- Email
- Banners in the process management tool
- Messages to another systems

BR-8 - as a process master, I may need additional process elements.

BR-9 - as a process master, I need to be educated how to work with processes.

BR-10 - as a process master, I need the processes to work in real time.

BR-11 - as an admin, I need to control process access rights.

BR-12 - as an admin, I need to have a simple method of inputs and outputs definition and tuning.

BR-13 - as a system user, I need to see the evidence of the processes work flow: errors, count of the computed events, metrics about the system performance.

BR-14 - as a business owner I want to minimize the cost of the possible solution's extension.

3.3 System requirements

For the reason that the final system form is not defined, the system requirements are very abstract comparatively to the normal project ones.

3.3.1 Functional requirements

FR-1 - The system will allow users to create, read, update, copy and delete processes in form of Flink jobs.

FR-2 - The system will allow users to run and stop processes.

FR-3 - The system will allow users to create Kafka inputs in the processes.

FR-4 - The system will allow users to create following outputs of the processes:

- Kafka
- Email

FR-5 - The system will allow users to define settings for Kafka connection outside of the processes.

FR-6 - The system will allow users to create filters, switches and custom variables in processes.

FR-7 - The system will support arithmetic, logic and aggregation operators in context of any data manipulation.

Operators:

- Arithmetic: + - * /
- Logic: or, and, not, equal
- Aggregation: Sum(), Avg(), Max(), Min()

FR-8 - The system will allow users to make time related values in the system.

FR-9 - The system will allow users to make queries to Hive databases.

FR-10 - The system will allow users to define Hive databases' settings outside of the processes.

FR-11 - The system will allow users to inspect metrics of the process work via an open-source dashboard.

FR-12 - The system functions will require authentication and authorization via login and password.

3.3.2 Non-functional requirements

NFR-1 - The system will provide its main functions via a web application.

NFR-2 - The system will provide process construction functions via GUI with parameterization and/or pseudocode. Database queries may be written in its native language.

NFR-3 - The system will inform a user about made technical errors in processes.

NFR-4 - A regular Flink node will be able to handle 250 regular events per second with latency less then 1 second.

NFR-5 - Process services will be implemented in modular form.

NFR-6 - The system back end will be written in Scala v.2.11 or newer, Java v.1.8 or newer or Python v.2.7 (the latest version of the Python, supported by Flink). Java and Scala are preferable.

NFR-7 - The web application will support following browsers: Edge v.18, Firefox v. 66, Chrome v.74, Safari v.12.1, Opera v.58 and newer.

NFR-8 - Low-level settings as access rights, Kafka and database predefined connections may be defined through the file system of the hardware where the system will be running.

NFR-9 - The system will support Kerberos authentication.

3.3.3 Transitional requirements

TR-1 - The solution will include a manual.

TR-2 - May be required additional agreed services.

3.3.4 Requirements for the searched system

SSR-1 - Following system requirements must be already realized: FR-1, FR-2, FR-3, FR-6, FR-7, NFR-1, NFR-2, NFR-4, NFR-5, NFR-6, NFR-8

SSR-2 - The system should be licensed under the Apache 2.0 or MIT licenses.

SSR-3 - The project must be fit to be finished.

SSR-4 - The project is developing and/or has viable community.

4 Solution

In this section I describe how did I search the needed solution and what did I found.

4.1 Solution search

As was defined, I had to find a system, that would already have realized functions of creating processes from much less abstract parts then Flink has itself, already has a usable user interface, available for a commercial use and is fit for developing.

Search requests in Google, GitHub and GitLab like “Complex event system", “open source CEP system", “Event stream processing" and similar ones gave stream platforms like Flink and Spark themselves only. But the request "Flink GUI" gave many articles about a Polish open source project named "Nussknacker" (a "nutcracker" in German) developing by a company Touk.

4.2 Solution overview

Nussknacker is an open-source real time processing tool with a flowchart GUI for process constructing licensed under Apache 2.0. It is based on Flink processing engine, has input and output connectors to Kafka. It has implemented base blocks of processing as switch, filter and value definition and wide API for services creating with a few examples of their using. The 1 of the April 2019 was released a new version 0.0.11, commits on GitHub are frequent, so a project looks viable. Site documentation seems to be quite poor, videos with using representation look credibly so I decided to inspect this solution.

Nussknacker system consists in:

- UI - a standalone web application intended for process designing in form of flow charts and accessing them.
- Engine - It’s main purpose is to perceive and interpret desined process as a Flink job and deploy it on a Flink cluster. Also provides API for custom services, utils, inputs and outputs creating to use them in the flow.
- Generic model - jar archive compilating all used services, inputs etc. Also may use its own data model, utilities and many other things.

There are three version available to run:

- Demo - there is a Docker based demo with all the parts running on that. Also it includes Graphana, Kibana, ElasticSearch and InfluxDb for metric collecting and a Flink node.
- Official release archive - a built system.
- Source code - Available to be built via sbt wrapper - a build tool for Java and Scala.

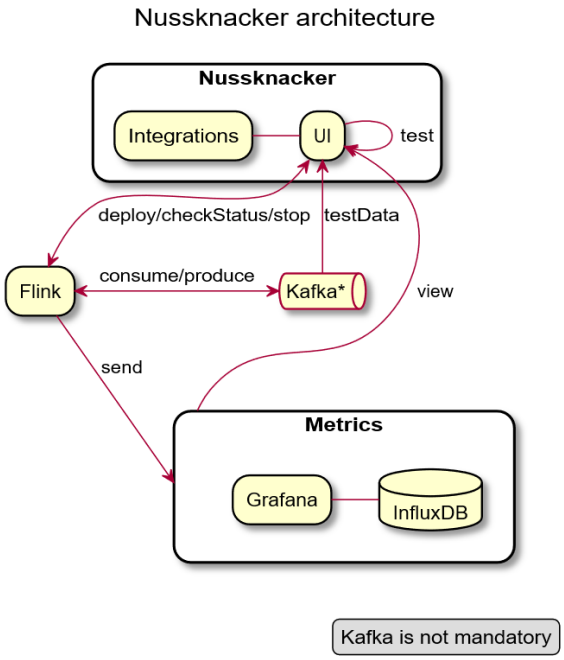

Figure 2: Nussknacker architecture picture from the project web[29]

A built system requires a Generic model jar and there must be configured all the connections (as Flink, non-mandatory Kibana and Kafka) in a configuration file.

The main functions of the Nussknacker step by step:

- On the main window of the web application may be created processes with defined unique names. On the same window is showed running status of the processes and placed links for process editing.
- Processes may become a subprocess of an another process (does not work yet) or be archved. Archived processes may be restored.
- Processes have multiple versions. Any saving of the process creates its new version.
- After the process creating opens a window with the process editor.
  - The central surface of the editor is used by flowchart representing the process. Every box of the chart represents any operation in the flow and has it own editor opening by a click on the box.

- Mostly boxes have input and output. They can be connected by mouse dragging from the output of the one box to the input of another one. There appears an arrow representing data flow.
- The most of operation use Expression fields for any evaluation values definition. It uses main parts of the Spring expression language (SpEL) intended for manipulating querying an object graph at the runtime. Supports logic logical, arithmetical and aggregating operations in the Java similar form. In Nussknacker aggregation functions do not work yet. Also in expressions may be used own operations compiled in generic model.

  There are following operation initially installed:
  - JSON and Avro Kafka inputs. Subscribes on a specified in its box topic and starts a flow with a value "input". A process may have only one input.
  - JSON and Avro Kafka outputs. Produces a specified by an expression value to a specified topic. A process may have many outputs. Application may have another inputs implemented and Kafka connectors using is not mandatory. But each process must have at least one input and one output.
  - Variable - includes a specified by an expression value with the specified name in the flow.
  - Filter - passes the flow if specified expression returns True.
  - Split - splits the flow on the several parts and sends processing values to the every branch. Branches cannot be merged.
  - Switch - splits the flow on the several parts and sends defined by processing values to the every branch. Branches cannot be merged. Switches' output connected with inputs of another by a special connection arrow with its own editor where is defined condition for the flow passing like in filters.
  - Demo custom services - in different versions of Nussknacker there are different services that may have expression fields (or some of them), may include into the flow a specified value or to do anything else. Services like that may be implemented by someone and compiled in a used generic model.
- In the right part of the editor are placed functions for the whole process access. Process run and stop, save, compare current version to another one, import or export it. Process run, stop and save operations show proper info banners. There are also important operations such as undo, redo (does not work), delete a box and duplicate it.
- In the left part of the editor there are windows with the available boxes grouped by using type and version chart. There is an opportunity to restore any saved version.

- From the main window or process editor may be accessed Grafana metrics if it is installed.

- In addition there may be an Admin console for used and unused services observing.

That is the most most important Nussknacker's functions. Looks almost ideally for our requirements. Meanwhile in the practice everything was not so fine.

5 Solution inspection and using

In this section I describe how did I build up a Nussknacker system, studied to work with it, develop, assure fulfillment of the requirements.

5.1 Build and deployment

For my needs I could use one of the Company's CentOS servers with two 2.4 GHZ CPU, 100 GB of hard memory and 8 GB RAM.

At first, I deployed the Docker demo that distributes Touk. Also I installed Kafka, created some test topics and moved simple python consumer and producer from another project I participated. It started without any problems, but GUI was very unstable, boxes in the process editor often became uncontrollable, after a box removing an arrow that was connected to it sometimes stayed connected to nothing etc. Also it has some custom services in its generic model that had own data model. I did not know how they work and have not come to anything more useful then it was shown in "Quickstart" page of the Nusknacker's web.

At the same time I tried to build the system from the official source code v 0.0.10. There were prepared several scripts for sbt wrapper, but it did throw incomprehensible errors.

Then I was building up a system based on a official 0.0.10 release. I installed Flink of the version officially supported by Nussknacker on my machine and connected it with the Kafka server and Nussknacker. This version was much more stable and reliable. I studied a bit how does Flink work, how to create processes in Nussknacker, tried to understand, what is capable to do in further perspective, and tested what can be interpreted by expressions.

Then the Company offered me to use its Kafka cluster secured by Kerberos. I accepted. Due to lack of documentation, I spent two weeks in trying to understand how to connect Nussknacker and its Kafka connectors to the Kerberos authenticating server. Flink could be connected strictly from its configuration, but Kafka connectors should be configured from the application that uses them. Sadly, Touk had not experience of this task and could not help. The solution was easy and elegant: in addition to standard operations and simple Flink connecting there should be pasted four rows of settings in the Nusscnacker's configuration file. Also I found a Kafka consumer and producer that were able to authenticate via Kerberos (Python Kafka engines do not support keytabs).

Later, after Nussknacker v.0.0.11 release I returned to the attempts to build the system from the code. For inexplicable reasons the same operations gave the positive results with the code from the master branch of the Nussknacker repository, and still gave negative results with incomprehensible errors. After that I just use the master version.

But meanwhile I had already started examining of the engine's API, tried to compile a functional generic model and develop my own services.

5.2 Code inspection and developing

The main problem of the whole project code is evident: it almost has no documentation. The code is quite readable even though I had not experience with Scala before, but it almost has no commentaries. And it is still not clear, what the different parts of the project.

I learned to compile generic sources quite fast. There was a choice of the language: Scala and Java, I chose Scala because of the interest to the language, elegance of the examples and the bigger count of them.

I thought, that Nussknacker do not use State backends and cannot store values in context of a time window. Then after several days of code inspecting I realized, that it actually do use them in transformers that extend an abstract transformer named timestamp evictable. In one of the accessible generics models there was an implementation of this transformer. It stored for a defined time and returns an array of them. I included this transformer in my generic model and test it. But processes with that did not work. Nussknacker logs gave me nothing, Flink logs informed me only that there is an error. I wasted three weeks in attempting to understand which error it is, I inspected Nussknacker code, Flink code, experimented with the values given to the transformer. Finally, I decided to control the version of Flink, but there was not information about Flink version in "Breaking changes". And then I control demo version of the system. There was a newer version.

After I updated Flink, its logs began to inform me, that there is an error with time interpretation. I knew, that time field requires string values, but wrote there the time of value storing in milliseconds like '1000'. But installed interpreter required something like '2 seconds" or '5.5 hours'. And proper documentation would help me to avoid such errors.

And after that there was an incomprehensible null pointer exception in abstract timestamp evictable function. After some weeks I resolved this error by copying of method that contained this error to an implementation of the transformer. Then I created an abstract class that store numbers mapped to strings in defined time and include in the flow a specific number computed by an abstract function. Finally, I developed and tested implementations of these abstract classes for the numbers' sum, average, count and difference between the first and the last numbers in a time window.

Due to lack of time, I had to finish the code inspection and implementation part of the work and start its testing (and writing this thesis).

5.3 Usability testing

ISO defines usability as "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use."[1] Also, we can understand usability as a conflux of following parameters:

- Learnability: opportunities to easily understand how to use the system.

- Efficiency: opportunities to use the system's services quickly.
- Memorability: opportunities to remember how to use the system without relearning.
- User error protection: how does the system protect a user from making mistakes and helps to recover from an error state.
- Satisfaction: how does a user contended with the system's interface.

In this project usability testing is mainly needed to test the interface of the jobs creating. Also it can be helpful in the general system developing. As the testing methods were chosen:

- Heuristic evaluation - assurance if the system follows the convention of usability via the conventional criteria.
- Simplified testing with a user - inspection how does a user from the target group use the system.

5.3.1 Heuristic evaluation

Usability heuristics are the principals, that should follow user interfaces. Heuristic evaluation helps to detect evident problems and may offer a way to resolve them. Heuristics published by Jakob Nielsen in 1994 represent the current convention. The whole following list is important and should not be interpreted. That is a quote: [3]

1. *Visibility of system status: The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.*
2. *Match between system and the real world: The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.*
3. *User control and freedom: Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.*
4. *Consistency and standards: Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.*
5. *Error prevention: Even better than good error messages is a careful design that prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.*

6. *Recognition rather than recall: Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.*

7. *Flexibility and efficiency of use: Accelerators unseen by the novice user may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.*

8. *Aesthetic and minimalist design: Dialogues should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.*

9. *Help users recognize, diagnose, and recover from errors: Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.*

10. *Help and documentation: Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.*

During the work with Nussknacker the following UI problems (UIP) were found:

UIP-1 - If a process fails, system informs only that "the process maybe failed". Further information may be found in Flink and Nussknacker logs. Heuristics: 1, 9

UIP-2 - Nussknacker logs are unavailable in web client even for an admin. Heuristics: 7, 9

UIP-3 - If a Flink job cannot be constructed or uploaded Nussknacker may do not show any message. Heuristics: 1, 9

UIP-4 - A user must remember names all of the values in the flow to use them elsewhere. Heuristic: 6

UIP-5 - It is impossible to copy a service in a flow. Heuristic: 7

UIP-6 - It is impossible to copy a process. Heuristic: 7

UIP-7 - A user may close a changed service without saving and the system does not ask if he wants discard the changes. Heuristic: 5

UIP-8 - Service informs a user about made mistakes in the service editor only after he saved and closed the editor. Heuristic: 5

UIP-9 - Process properties and save, import/export and archive functions are available in process editor only. But there is not possibility to rename the process. Heuristic: 4

UIP-10 - If a new version of the process is saved and deployed when an old version was running, the system show a notification, that the process was deployed, but version control tree says, that old version may be running and the main window shows a red circle in a status of the process. In addition the process become uncontrollable. Heuristics: 1, 3, 5

UIP-11 - There is not a process status info in the process editor. Heuristics: 1, 7

UIP-12 - A process may be run and stopped only via its editor. Heuristic: 1

UIP-13 - The system only one manual is on the Nussknacker web-site and it is incomplete. Heuristic: 10

The main problems here are definitely UIP1 and UIP3 for me as a developer, for a support engineer and for a customer. It must be fixed. Also must be written a manual. But there were not found any enormous problems for a regular user until do hot happen any mistakes and exceptions.

5.3.2 Introduction to testing with a user

The most full usability tests with the user should be performed in a special laboratory with multimedia recording hardware, have special phases of the pre-tests and post-tests, have special tests to pick the test users, records form the main test may be watched and analyzed. But I decided to make it simpler.

I invent a few business cases of the system using, let several users (people from the target group) go through the official manual and implement processes for my business cases, inspect how is it going and ask users about their feelings and thoughts, what they did like and did not, what would they suggest to change or improve. The target group was defined as a people that have not seen Nussknacker, have any experience with information management and they are not experts programming, but understand how it works.

5.3.3 Tasks preparation

In the Company context, the project has a further perspective in work with the Prague integrated transport (PID) company. We already have some their data (roughly 160 data sets in form of 50MB JSON bathes) about buses, metro and trams movement: their coordinates, next stations, delays, type, time of measurement and less important ones. I decided to design several test processes (TP) with them.

- TP1 - If the current delay (delay of arrival at the next station) is much bigger then previous delay, then should be sent an alert.

- TP2 - If at the defined time the delay significantly increased, then should be sent an alert.
- TP3 - There should be sent specific messages based for following events:
  - The transport is steadily and significantly late. Should be sent an alert.
  - The transport is steadily and quite late. Should be sent a warning.
  - The transport is exceptionally and significantly late. Should be sent a warning.

TP2 and TP3 require holding of the delay states mapped to the transports' identifiers. They are implemented by me. Also, I implemented a python script to make the batch data sets possible to be effortlessly sent in stream and adapted my Kafka producer to them.

5.3.4 Test

For the test I found four people from the target group: they are students of the Prague School of Economics, two of them are Bachelors, one is a Master, the last one finishes bachelor study. They had several subjects in the university about programming, but they do management mainly. One of them has serious interest in information management, event processing and business intelligence. In addition, they are my friends.

The testing was performed with each test user, one by one. I explained each of them the rules of the testing, let them go through the manual of Nussknacker and review the system, explained the purpose of the my timestamp evictable transformers, then introduced them to the test data, answered their specifying questions about them. Then I let them examine the business cases, clarify, that Kafka output with the word "Warning" and transport ID is a needed warning. After that users started trying to implement processes. All of them understood how to implement TP1 and had problems with the TP2 and TP3. But the test was not about successful process implementation, but about system usability. There were found following usability points:

1. All users liked chart flow structure of the processes.
2. There is not evident how to use logical operators. 3 users tried to use “and", “not" and “or" instead of "&&", “!" and “||".
3. One user (the Mgr.) was not satisfied with the GUI, called it "old looking and clumsy". Others were neutral-positive about that.
4. All users would like to have a well-describing manual about expression writing and services using. Sophisticated services like timestamp evictable transformers should be explained better.
5. A drop, the icon of the sincs (outputs), did not represent an output for all users.

6. All users "Align" function, that aligns the chart.
7. One user said he would like to be able to copy services.
8. Two users asked, how to copy a process.
9. One user said he would like to see details of a service without opening its editor.
10. One user said he would like to have a visible choice between possible operators in expressions or at least auto-complete function.
11. All users said that the system is usable, one of them meanwhile would not like to use it in practice until major improvements.

The testing assures that the system that Nussknacker is usable enough to work with it, but it is a hard question how should be implemented needed major improvements.

5.4 Stress testing

The first and main question is what should actually be tested. The system's processing is performed by Flink its reliability and efficiency ant it does not require an assurance. But its using is influenced by Nussknacker's implementation of the Apache Flink API. But there are not so much of services implemented to create a process complicated enough to represent something. But I can use Nussknacker's API, implement some custom services, maybe a hardcoded connection to a Hive database and enrich input Kafka stream by it's data. But why would I test my own code if it is needed to verify the system's durability and performance?

I decided to implement a few services and construct some simple processes to test requirements on them. Especially, NFR - "A regular Flink node will be able to handle 250 regular events per second with latency less then 1 second". Harder processes remain untested, but their efficiency will depend on things that will be implemented later. Also, as the latency I understood here as the time difference between the source operator and sinc operator.

5.4.1 Test method and preparation

The intention was to design and implement a few realistic processes, test them on a cluster with a high load of data and inspect latency/throughput. I decided to use the processes from usability testing with a user.

The test processes in more technical sight are following:

- TP1 - If delta between the current delay and the previous one is more then N, then should be sent an alert.
- TP2 - If delta between the current delay and the oldest one on a time window, then should be sent an alert.

- TP3 - There should be sent specific messages based on average delay and measurements count on a defined time window. There are following events:
  - If the average delay is bigger then N and the measurements count is bigger then M, then should be sent an alert.
  - If the average delay is bigger then N and the measurements count is not bigger then M, the transport is exceptionally and significantly late, then a specific warning should be sent.
  - If the average delay is between K and N, and the measurements count is bigger then M. K is less then N, but is still undesirable, then a specific warning should be sent.

To test a real Flink node performance I moved it on a separate server, same as the main one. For needs of metrics collection I installed Grafana and InfluxDB on the main server and set up dashboards for throughput and latency metrics with the help of Grafana settings JSON in Nussknacker demo. Kafka cluster is still external.

In addition, for the test I created a dataset with 666255 messages in total volume 298 MB.

5.4.2 Test

I implemented the designed processes8 and launched the data stream through them. TP1 and TP2 were not stressful for the system, but TP3 results are much more interesting8.

Test	Max throughput (events per second)	ma latency (ms)
TP1	14260	27
TP2	17860	172
TP3	1947	920

I consider the results as positive.

6 Conclusion

The main goal of the work was to find an open-source solution that would fulfill the most of the Company's requirements in real time big data event processing tool. To understand what definitely the Company requires, I studied what is event, how events may be processed, what is stream and how may be events processed in a streams, big data principles, real time meaning, related business areas and technologies, and designed use cases of complex event processing. Then I analyzed requirements for an online CESP tool with a user-friendly UI. Based on specified requirements examined only one found open-source solution, learnt to use, develop and build it. To prove fulfillment of usability requirements I inspect the system through a prism of heuristics, performed a usability test with 4 people from the target group. Then to prove fulfillment of durability and speed requirements I performed a stress test on synthetic business data set.

Finally, I conclude, that the only found solution fulfill the most of requirements. It has terrible lack of documentation, its GUI is not perfect, there are problem with building, but the solution have a great potential and even almost without documentation it is usable. GUI should be fixed, code should be documented, problems in the building definitely may be solved with some sbt wrapper studying. It is much cheaper, then a completely own solution.

As future steps I consider Hive studying with intention to create an enricher service, studying of Flink details and implementation of the common useful services. Also I plan to contribute the services that I have already developed.

7 Literature and web sources

[1] Saurabh G.; Shilpi S. 2017. Practical Real-time Data Processing and Analytics, 1 edition. Packt Publishing. ISBN: 978-1787281202

[2] Byron E. 2014. Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data. 1 edition. John Wiley & Sons. ISBN 978-1118837917.

[3] Kalavri V.; Hueske F. 2019. Stream Processing with Apache Flink. 1 edition. O'Reilly Media, Inc. ISBN 978-1491974285

[4] Friedman E.; Tzoumas K. 2016. Introduction to Apache Flink. 1 edition. O'Reilly Media, Inc. ISBN 978-1491977132

[5] Garillot F.; Maas G. 2017. Stream Processing with Apache Spark. 1 edition. O'Reilly Media, Inc. ISBN 978-1491944240

[6] Stopford B. 2018. Designing Event-Driven Systems. 1 edition. O'Reilly Media, Inc. ISBN: 978-1492038252

[7] Shapira G.; Narkhede N.; Palino T. 2017. Kafka: The Definitive Guide. 1 edition. O'Reilly Media, Inc. ISBN: 978-1491936160

[8] Kleppmann M. 2016. Making Sense of Stream Processing. 1 edition. O'Reilly Media, Inc. ISBN: 978-1492042563

[9] NIELSEN, J. Usability engineering. Boston: AP Professional, 1993. ISBN isbn0-12-518406-9.

[10] Jean J. Labrosse, et al. Chapter 8. DSP in Embedded Systems // Embedded Software. — Newnes, 2007. — 792 p. — ISBN 978-0-7506-8583-2.

[11] O. Etzion and P. Niblett, "Event Processing in Action", Manning Publications, 2010.

[12] https://www.wired.com/insights/2013/05/the-missing-vs-in-big-data-viability-and-value/

[13] https://www.quora.com/How-is-stream-processing-and-complex-event-processing-CEP-different

[14] https://www.sciencedirect.com/science/article/pii/S2212017315002364

[15] https://medium.com/stream-processing/what-is-stream-processing-1eadfca11b97

[16] https://www.confluent.io/blog/apache-flink-apache-kafka-streams-comparison-guideline-users/

[17] https://dzone.com/articles/streaming-in-spark-flink-and-kafka-1

[18] https://jobs.zalando.com/tech/blog/apache-showdown-flink-vs.-spark/?gh_src=4n3gxh1

[19] https://resources.whitesourcesoftware.com/blog-whitesource/top-open-source-licenses-trends-and-predictions

[20] https://wikipedia.org

[21] https://opensource.org/licenses/

[22] https://ci.apache.org/projects/Kafka

[23] https://ci.apache.org/projects/Spark

[24] https://ci.apache.org/projects/flink

[25] https://ci.apache.org/projects/flink/flink-docs-stable/dev/stream/python.html

[26] https://web.mit.edu/kerberos/

[27] https://grafana.com/

[28] https://www.influxdata.com/

[29] https://touk.github.io/nussknacker/

[30] Nielsen, Jakob (1994). Usability Engineering. San Diego: Academic Press. pp. 115–148. ISBN 0125184069.

[31] https://docs.spring.io/spring/docs/4.3.10.RELEASE/ spring-framework-reference/html/expressions.html

8 Citations

[1] Usability. PORTAL ISO 25000 [online]. Copyright © 2019 iso25000.com retrieved May 23, 2019, from: https://iso25000.com/index.php/ en/iso-25000-standards/iso-25010/61-usability

[2] EVENT | meaning in the Cambridge English Dictionary. Cambridge Dictionary | English Dictionary, Translations & Thesaurus [online]. Copyright © Cambridge University Press, retrieved May 23, 2019, from https://dictionary.cambridge.org/dictionary/english/event

[3] 10 Heuristics for User Interface Design: Article by Jakob Nielsen [online]. Copyright © Jakob Nielsen, retrieved May 23, 2019, from http://www.useit.com/papers/heuristic/heuristic_list.html

Attachment A - Abbreviations used

- DWH - Data Warehouse
- API - Application Programming Interface
- CEP - Complex Event Processing
- ESP - Event Stream Processing
- CESP - Complex Event Stream Processing
- UI - User Interface
- GUI - Graphical User Interface
- SQL - Structured Query Language
- ID - Identifier
- JSON - JavaScript Object Notation
- SpEL - Spring Expression Language
- RAM - Random Access memory

Attachment B - Test processes

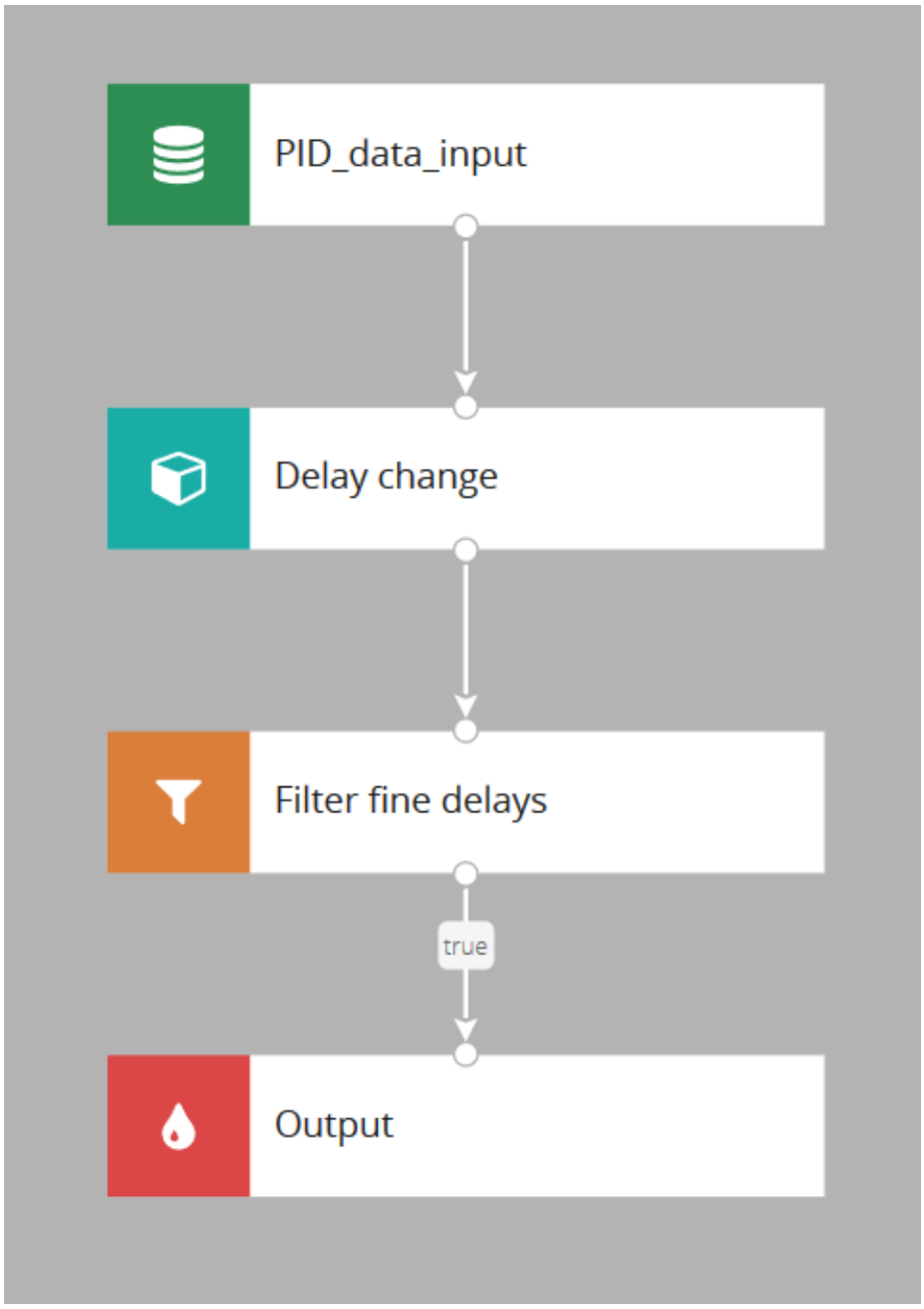

Figure 3: TP1

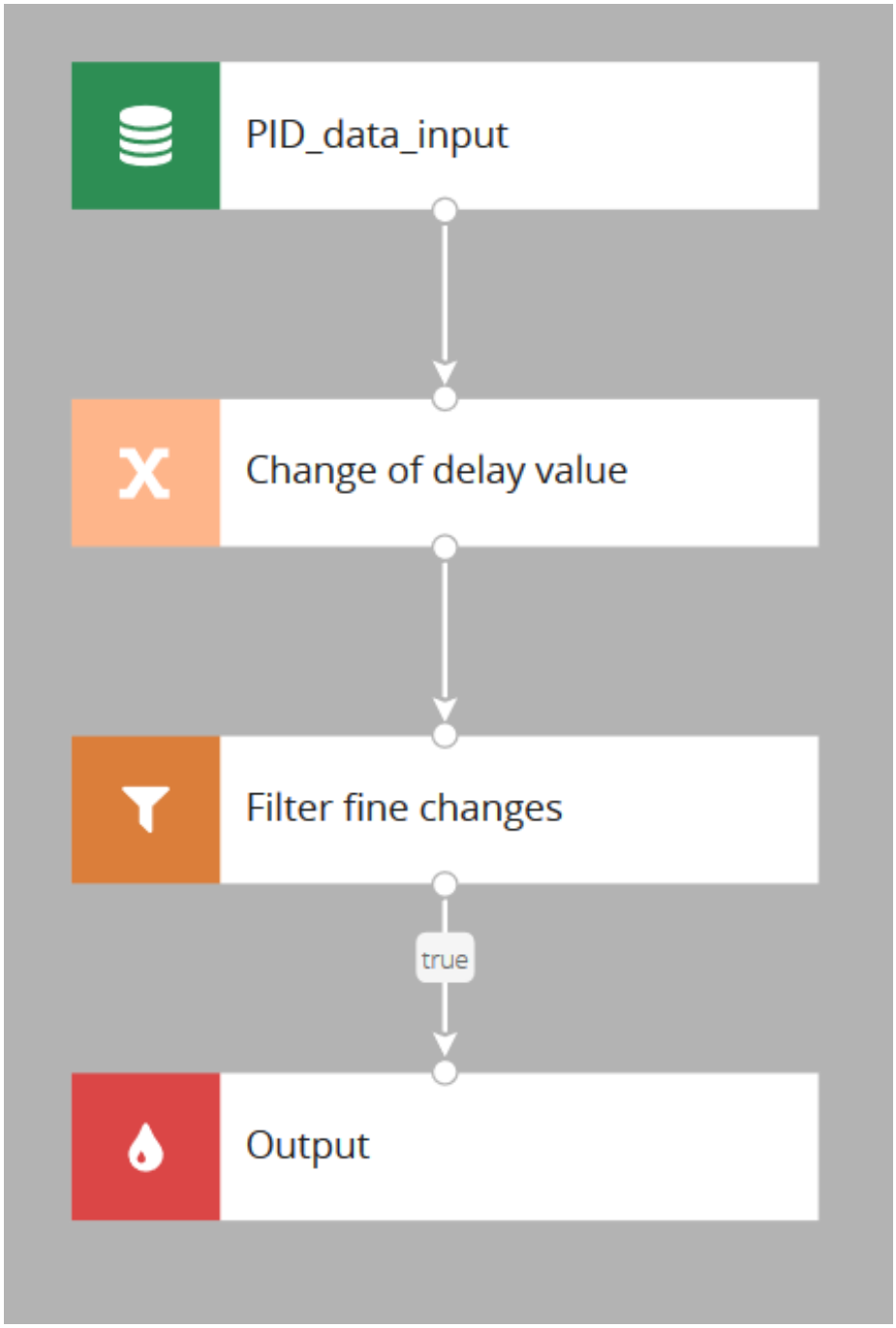
PID_data_input
Change of delay value
Filter fine changes
true
Output

Figure 4: TP2

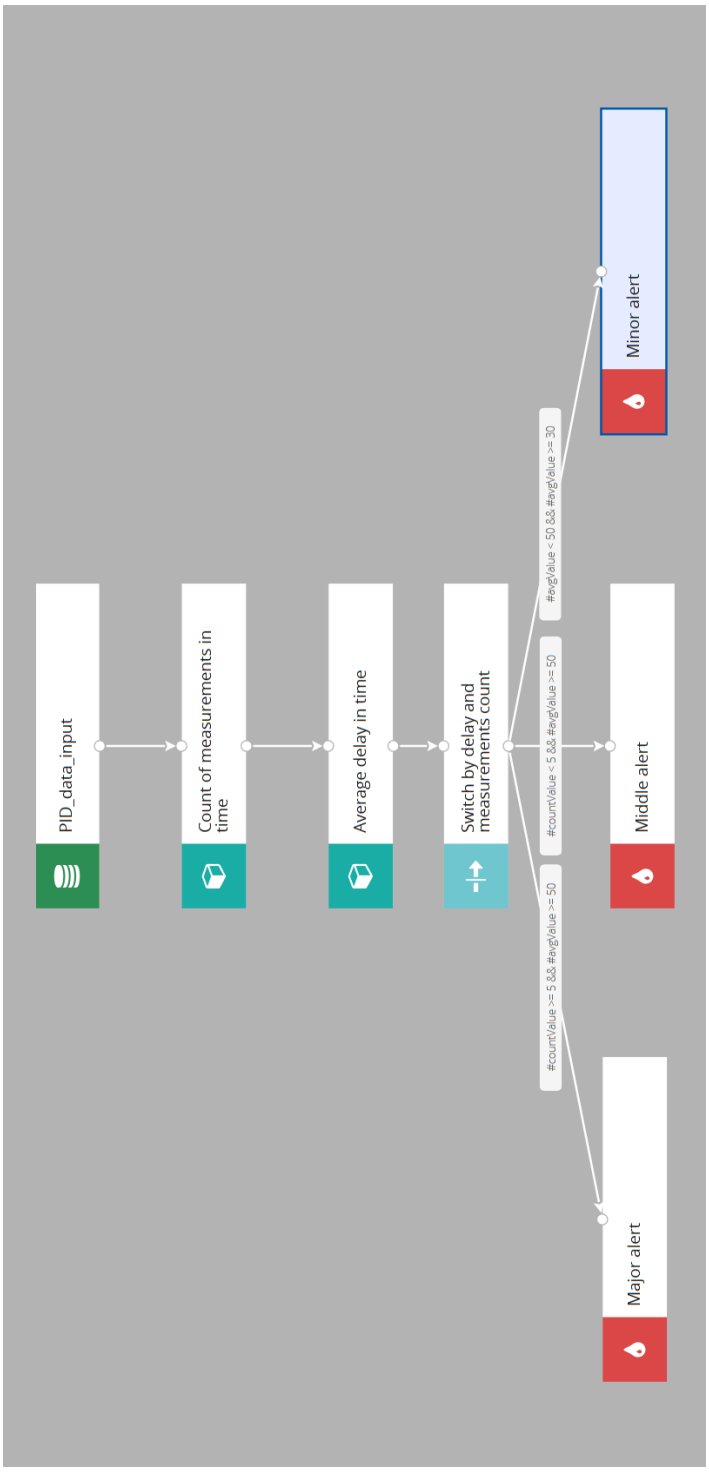
PID_data_input
Count of measurements in time
Average delay in time
Switch by delay and measurements count
#countValue >= 5 && #avgValue >= 50
#countValue < 5 && #avgValue >= 50
#avgValue < 50 && #avgValue >= 30
Major alert
Middle alert
Minor alert

Figure 5: TP3

Attachment C - stress test charts

Figure 6: Throughput of the TP3

Figure 7: Latency of the TP3