Abstract—The paper describes the architecture for processing of Big Data in real-time based on multi-agent system paradigms. The overall approach to processing of offline and online data is presented. Possible applications of the architecture in the area of recommendation system is shown, however it is argued the approach is general purpose.

I. INTRODUCTION

Big Data is one of the most popular research topics nowadays. With the advances in parallel computation architectures more and more instances of the algorithms can be run concurrently. Systems such as Hadoop allow storage of huge amounts of unstructured information and processing them with algorithms such as MapReduce [1], optimized for parallel and asynchronous computations.

Also in the database systems we observe the shift of paradigms. In most practical applications of Big Data the traditional properties of ACID (Atomicity, Consistency, Isolation, Durability) are no longer applicable. As the CAP Theorem states [2], we can assure at the same time only two out of the following three properties namely:

- Consistency (all nodes see the same data at the same time)
- Availability (a guarantee that every request receives a response about whether it was successful or if it has failed)
- Partition tolerance (the system continues to operate despite arbitrary message loss or failure of part of the system)

There are several important applications of Big Data, which require new approaches in order to provide robust and usable solutions. Among the most popular there are: recommender systems [3], social network analysis [4], web mining, algorithmic trading, fraud detection etc. The diversity of these topics shows the importance of Big Data and imposes challenges on the tools and architectures, which have to be applicable in various scenarios.

This paper shows an architecture, which is designed for different implementations in which both real-time and offline Big Data processing is necessary. As an example of application a recommender system is shown.

II. MOTIVATION AND RELATED WORK

The work was motivated by the study on applications of Big Data processing in Allegro.pl, the largest e-commerce marketplace platform in Central Europe. The main challenge was to create an architecture, which would allow efficient processing of various data stored in the long term storage as well as captured in real-time in order to serve the results of such processing to the on-line users.

A. Big data

There has already been a lot of work done in the area of Big Data applications for Business Intelligence and analytics in particular in the e-commerce area [5]. Large vendors both in America (Amazon, eBay) as well as in Europe (Allegro) and other continents were able to build highly scalable platforms as well as analytical capabilities around them. Furthermore they analyze social content generated around to get even better insight into the client experience.

An important paradigm which is more and more present in the Big Data analysis is the concept of autonomous, intelligent and proactive agents. In the context of this work the notion of agent mining is especially interesting [6]. It combines methodologies, technology, tools and systems from the area of multi-agent technology, data mining and knowledge discovery, machine learning, statistics and semantic web in order to address issues that cannot be tackled by any single technique with the same quality and performance.

The multi-agent approach can also be applied to streaming data processing in the actors model. In [7] a S4 platform is presented, which allows distributed processing of the streams of data. The authors show how this can be applied to tune parameters in dynamic systems.

B. Recommender systems

The work on recommender systems has been a popular topic since the beginning of large commercial services on the Internet. Several approaches to this subject have been proposed [8] with Collaborative Filtering techniques being the most popular. Other approaches include content-based, knowledge-based and hybrid approaches.

Multi-agent systems have been often employed to the task of building recommender systems. In [9] a recommender system
of personalized TV contents, named AVATAR, is presented. A collaborative approach to recommendations combined with intelligent agents can be found in [10], [11].

III. REAL-TIME PROCESSING WITH BIG DATA

One of the main challenges, with respect to processing of very large sets of data, is handling of the real-time data streams. While both offline and online parts of data can be process independently, very often we need to provide answers to questions regarding online events based on the past history. The lambda architecture comes as an answer to these challenges.

A. The Lambda Architecture

The lambda architecture has been proposed by Nathan Marz [12]. It is based on several assumptions:
- fault tolerance
- support of ad-hoc queries
- scalability
- extensibility

The architecture as shown in the Figure 1 is composed of the following components:
- Batch Layer - responsible for managing the master dataset and for precomputation of batch views
- Serving layer - indexes the batch views for fast ad-hoc queries
- Speed Layer - serves only the new data, which has not yet been processed by the batch layer

The Batch layer can be implemented with the use of systems such as Hadoop. It is responsible for storing the imputable master data set. Furthermore, with the use of the MapReduce algorithms it contiguously computes views of this data available to the various applications.

The Serving layer is responsible for serving the views computed by the batch layer. This process can be facilitated by additional indexing of the data in order to speed up the reads. An example of a technology used typically to do this job is Impala, which is easily integrated with Hadoop used in the Batch layer.

Finally, the role of the Speed layer is to compute in real-time the data that has just arrived and has not yet been processed by the batch layer. It serves this data in the form of real-time views, which are incremented as the new data arrives and can be used together with batch views for the complete view of the data.

B. Other approaches

Several other approaches to processing of Big Data in real-time have also been proposed. In [13] the following 8 requirements for real-time data processing are proposed:
- keep the data moving
- query using SQL on Streams (StreamSQL)
- handle stream imperfections
- generate predictable Outcomes
- integrate stored and streaming data
- guarantee data safety and availability
- partition and scale applications automatically
- process and respond instantaneously

Also other approaches to the subject of real-time or near real-time processing of Big Data can be found in the literature [14], [15].

IV. ARCHITECTURE FOR MULTI-AGENT BIG DATA ANALYSIS

Presented approach for processing Big Data in a real-time manner is trying to solve the main problem of analyzing large, continuously growing data sets. Lambda architecture is one of the newest method which has lately gained a lot of popularity, mainly because of its simplicity and use of a well-known tools for processing the data. Three easily recognizable layers for batch, speed and service is making clear division of components functionality. Moreover for each of this layer, there is wide selection of solutions for implementation to choose. Most of them are available on market for years and are known for the reliability.

Despite the fact, that the recalled architecture is presented as a simple and a clear one, still many decisions and lots of integration work needs to be done. The architecture gives only guidelines on how the system should be designed and which parts should it consist of. This gives freedom in the choice of existing solutions for a specific job. Still, interaction between batch, service and speed layers has to be handled properly. Furthermore, even in a single layer a few components must interact together using different protocols and methods of communication. In the Big Data environments these means, that we have to cope with the integration of distributed data processing systems where the scalability and reliability should be taken into account.

The lambda architecture for processing Big Data can be modeled as a heterogeneous multi-agent environment. Tree distinct layers with different characteristics exists, between which multiple components have to interact with each other. This communication can be simplified using the multi-agent system approach. Each agent is responsible for specific task in data processing, e.g.: receiving data, aggregating result, etc. Agents are autonomous and distributed. Cooperation between agents is done using message passing. All agent are communicating in the same manner. Therefore, the integration is simplified.
The Figure 2 presents the multi-agent big-data processing architecture for Big Data processing using multi-agent systems. There are no changes to the main concept. In MAS approach there are still batch, speed and serving layers. Where the batch layer is creating aggregates - Batch Views - from the all data. The speed layer is only incremental - Real-Time Views - for new, non-archived data. The service layer which brings both offline and online calculated data (views) for solving specified problems, e.g. analytic query, new credit decision, music recommendation etc.

The input data is processed by the system as a data stream. Depending on the domain, it can be a stream of: page views, user transactions, system logs, diagnostic events etc. Stream - as data series - is collected by the Stream Receiver Agent. This agent is responsible for simple data pre-processing like: filtering, data format change, serialization to objects etc. After that, every data event in the stream is passed to the Archiver Agent and the Stream Processing Agent. Both of these agents are in control of handling new data in the batch and the speed layer respectively.

In the batch processing the new data is first written to the Data Store, e.g. Hadoop Distributed File System [16]. The Data Store has to handle large data sets and store all events in the system. Holding every single event from data stream allows to run computation for a selected time period from the history. The computations are coordinated by the Batch Driver Agent. This agent is created for running specific jobs, while the actual work is done by its slave agents - Batch Worker Agents. Each worker agent processes part of the data, in order to successfully produce the output of the job - Batch Views. The Batch Views are all kinds of aggregates that need to be produced from the stored data. This is a general overview of the batch processing, which can be easily implemented in a distributed processing clusters like YARN[17] or Mesos[18].

The same events from the incoming data are processed by the speed layer. Here, a Stream Processing Agent is the first point of contact. The Processing Agent routes every event to appropriate Real-Time Worker Agent where the actual work is done. The result are represented by the Real-Time Views which are the online updated aggregates. Due to the speed consideration, this views are fast in-memory data sets prepared for the quick online access.

Both the Batch Views and the Real-Time Views are created for a specific use case. This use case problem is solved in the serving layer. Request from the outside world is handled by a dedicated Service Agent. The particular problem e.g. a new bank credit decision, is solved by the appropriate types of agent. For every new request the Service Agent is created. To solve the given problem and prepare the response, Service Agent is collecting necessary data. Historical data are provided by a precomputed Batch Views. To access this data a Batch Aggregator Agent is used. This agent query appropriate Batch Views.

A similar processing is done to collect fresh online data. Real-Time Aggregator Agent is preparing data from Real-Time Views data sets. Both batch and real-time views are combined together to present the whole picture of the data. After collecting all required data from the aggregators agents the response is created. In this point, when request is served and response is send back to the client, a life-cycle of Service Agent ends.

Depending on an infrastructure and the system domain, the online layer can hold views for the different time periods. It can vary from days to seconds. The main idea behind the batch and the speed layers is to work together to present a coherent view for the data at the service layer. The first thing which can be noticed in the proposed MAS architecture is that all communication in the system is only between the agents. Every single task presented in the lambda architecture in previous section is encapsulated inside an autonomous agent. This results in simplified integration and distributed computation.

Moreover, in the presented approach it is strongly recommended to use the same event representation in a both processing: batch and online. Despite the differences in the infrastructure a data schema can be the same (for lambda the most common ones are Hadoop for batch and Storm for online processing). What is more, when agents are fine grained and designed for a single responsibility then it can be reused in speed and batch layer. For example: the same calculation done in Batch Worker Agent and Real-Time Worker Agent can be done by the same agent implementation.

V. APPLICATIONS

The lambda architecture is showing how the batch and stream processing can work together solving most of the modern Big Data problems. The batch Big Data processing is well known for the most of analytic tasks like segmentation, ad-hoc queries for data exploration, etc. However, the general movement in Big Data processing is towards more responsive, real-time systems where results are available near online. The lambda architecture and the proposed multi-agent solution is trying to cope with a real-time and batch processing in a reasonable fashion. The proposed MAS approach can by applied whenever the Big Data analysis need quick response.
time and the query results scheduled by batch windows are not enough.

One of the attractive new areas, where the support of Big Data processing can results in a better customer experiences, are recommendation engines. For a long period of time Internet search engines were used for finding products, media content and any other kind of interesting information. However, in a still growing information flood, environment tools for better user profiling, web personalization and new product recommendation are required. Internet search engines have ceased to be convenient tools for most users. The recommendation systems, instead of forcing users for active exploration, are trying proactively to suggest new interesting content. The better user is profiled in the recommendation system - the better content is targeted. Therefore Big Data solutions comes handy.

Collaborative filtering is one of the most widely adopted and successful recommendation methods[19]. Unlike other recommendation approaches no expert knowledge and explicit rules are needed. System is characterizing users and recommended items implicitly from the previous interactions - events. Thus it is becoming a good choice for e-commerce[20] systems and online website personalization[21]. Popularity of collaborative filtering methods has led to new algorithms and methods for representing user preferences, like matrix factorization techniques[3].

In the collaborative filtering algorithms the more data is gathered about the users interaction, the more precisely user can be profiled and better recommendation can be. Users using web and mobile application leave trace of theirs action - events. The simplest form of user events for the web site is a click-stream. Collecting data from a click-stream is the most widely used method for web mining and analysis techniques in Big Data projects. From the collected users event collaborative filtering algorithms can implicitly [22] predict user preferences based on the others users interactions.

Collecting large amount of users events as the online stream is the first difficulty to cope with in collaborative filtering for the large scale application. After the data is gathered, user preferences should be prepared for serving online recommendation. Events generated by the users are converted into the rating model for specific recommendation items. Then, such prepared data can be used for collaborative filtering offline model generation. For the item-to-item version of the algorithm this operation applies to preparation of an item similarity matrix.

For building large scale online recommendation system with Big Data analytic stack, the multi-agent architecture presented in previous section can be used. In the recommendation systems new data stream include: items to recommend, users data and all events generated by users e.g. explicit item rating or page view. The problem to solve for the system is to provide best online recommendation in a timely manner for every user. A simplified MAS architecture for collaborative filtering recommendation system is presented on diagram 3. As a framework for agent based system implementation Akka[23] solution can be used. Akka is one of the most popular actor system framework prepared for java and scala language. The actor model has been designed for parallel computation based on multi-agent systems paradigm [24].

In the presented recommendation system all communication with the external systems can be done using JSON-REST endpoints. All services should be implemented as an agents. The data about items, user and new events are collected respectively by: Item Service Actor, User Service Actor and Event Service Actor. The Akka framework supports full asynchronous communication between actors and easy message passing pattern. For REST services we proposed a spray.io server. The server is build on top of Akka and simplifies handling HTTP protocol in actor environment. After receiving data on HTTP endpoint, messages are passed to Receiver Actor which stores data stream in simple queue system[25] in common format (e.g. Avro). A new data is then send to both: batch and speed layer.

The batch layer is mainly Hadoop ecosystem. New data are handled by Archiver Actor which stores data to HDFS - main offline data storage. The data saved in HDFS is processed by the batch framework in order to produce Batch Model. For the collaborative filtering recommendation, batch model consists of a prepared user rating model and calculated item-to-item similarity matrix. This Batch Model is produced in offline using YARN computation cluster on which a Spark[26] jobs are run. The Spark framework for a communication and coordination of its Driver and Workers using as well the Akka actor system framework. The Batch Model could be stored in Cassandra NoSQL database for further processing and fast access.

In the speed layer, new data is received by the Akka actors in the Spark Streaming framework [27]. An architecture of the Spark Streaming is very similar to its offline equivalent. The Driver as master coordinator as well as Workers actors exists. But in each worker there is a receiver actor of new data. This allows to easily scale the processing. All is run on top of Mesos cluster[18]. Online streaming processing produce Online Model: an up-to-date user preferences (based on user events) and new items with rating. Here, the same rating model as in the batch layer should be used. Online Model is held in memory databases e.g. Redis.

The data computed by batch and speed layer is prepared in
order to handle a new recommendation requests. The request in JSON-REST format is handled by a Recommendation Service Actor on spray.io server. Just like any other request to the system. But for this requests processing is somehow different. The request is being passed to a Recommendation Engine where a Recommender Actor is created. Its responsibility is to carry the process of recommendation. The Recommender Actor is collecting necessary data from Batch Model Actor and Online Model Actor. Based on request data context, precalculated item similarities and online updated user preferences new recommendation are generated for the user. All process ends when JSON response is send with a set of new recommended items to the user.

VI. CONCLUSION

The paper presents a multi-agent architecture for processing of Big Data. The approach is based on the lambda architecture proposed before. It has been shown how autonomous agents can enhance the architecture and provide capabilities for robust processing of data in real-time.

Possible applications of the proposed architecture to recommender systems have been discussed. The article argues that the approach is very well suited for such a task, where on-line events from users visiting an online site have to be combined with the offline data on users and items from the past.

REFERENCES