Implementing a Volunteer Notification System Into a Scalable, Analytical Realtime Data Processing Environment

Jesko Elsner, Tomas Sivicki, Philipp Meisen, Tobias Meisen, and Sabina Jeschke
Institute of Information Management in Mechanical Engineering
IMA/ZLW & IfU - RWTH Aachen University
{ Jesko.Elsner, Tomas.Sivicki, Philipp.Meisen, Tobias.Meisen, Sabina.Jeschke } @ima-zlw-ifu.rwth-aachen.de

Abstract - **The pace at which next-generation Internet of Things networks, consisting of wirelessly distributed sensors and devices, are being developed is speeding up. More and more devices produce data in automated manners and the demand of smartphones and wearable devices is continuously increasing. With respect to volunteer notification systems (VNS), the resulting vast amounts of data can be utilized for profiling and predicting the whereabouts of people that, combined with machine learning algorithms, complement artificial intelligence (AI)-based decision systems. Hence, VNS benefit from keeping pace with the current developments by using the corresponding data streams in order to improve decision making during the volunteer selection process. In emergency scenarios, the velocity, low latency and reaction times of the system are essential, which results in the need of online stream-processing and real-time computational solutions. This paper will focus on a basic concept for implementing a VNS approach into a scalable, fault-tolerant environment that uses state-of-the-art analytical tools to process information streams in real-time as well as on demand, and applies machine learning algorithms for an AI-based volunteer selection. This work concentrates on leveraging open source Big Data technologies with the aim to deliver a robust, secure and highly available enterprise-class Big Data platform. Within the given context, this work will furthermore give an insight on state-of-the-art proprietary solutions for Big Data processing that are currently available.**

Keywords - Volunteer Notification System; Internet of Things; Big Data; Stream Processing; Machine Learning

I. INTRODUCTION

As we are moving towards the Internet of Things (IoT), the number of sensors that are deployed around the world, and devices supporting various different sensory technologies, is growing at a rapid pace [1]. These sensors and devices continuously (and automated) generate high amounts of data. However, in order to add value to the collected raw data, further processing is required that will help understanding the meaning and correlations within.

Bundling the accumulated data into a so called real-time information pipeline does enable scalable real-time query / in-stream processing technologies [2] and regular batch processing, which is currently supported by various state-of-the-art Big Data analytical environments, as will be discussed later. To a given problem (query), the introduced approach will process both persisted as well as real-time data to generate results, which can be further processed instantaneously or stored for subsequent processing. Various machine learning extensions on top of the basic environment do furthermore provide possibilities for extensive profiling and learning approaches that are based on the collected data, whereas the resulting decisions are generated near real-time, enabling a scalable volunteer selection architecture within the application scenario of a Volunteer Notification System (VNS), as primary introduced in [3].

Hence, this paper is going to provide an insight of the various technologies that can be efficiently used in order to create a scalable, reliable and fault tolerant environment as architectural base for a reasonable VNS implementation.

A. *Structure*

Section I will continue by introducing the various terminologies that are used throughout this work, whilst Section II will discuss the state-of-the-art with respect to the (Big Data) domain specific technologies and analytical frameworks. Section III will give detailed insights on the basic implementation approach and the corresponding concepts and methods, discussing the scalability effects (of the most problematic system components) of the underlying technologies in comparison. The last section, Section IV, will present a brief conclusion on the elaborated approach and shortly discuss those proprietary solutions and standards that are currently well established in the industry.

B. *Volunteer Notification System*

A VNS is an approach to integrate laypersons and medically trained volunteers into emergency medical services (EMS). By tracking the users' location, and in case of a medical emergency, a VNS aims to alarm those potential voluntary first-aiders who can arrive on scene fast enough to provide the most urgent measures until professional EMS arrive at the victims location.

Whilst the volunteer selection process can be efficiently enhanced by an AI-driven selection system [4], rather than merely using the last known location of a volunteer, this general approach is greatly limited by the input data stream and the available processing power. Thus, in order to provide a technical solution for the basic research questions in regards to an intelligent VNS, the scope of this work will focus on providing a solution in which the supported input data - that is generated by a multitude of devices - ideally is limitless and the computational power will be matter of theoretically seamless scalability.

C. The Internet of Things paradigm

The IoT paradigm proposes that everyday objects will be globally accessible over the Internet or other adequate network structures. Opposite to the Internet world, things with a physical shape usually belong to resource-challenged environments where energy, data throughput, and computing resources are scarce.

The focus of typical IoT activities lies on establishing connectivity at a certain protocol level to enable truly distributed machine-to-machine (M2M) applications. In the general protocol specification, the devices must communicate with each other (D2D). A device's data then must be collected and forwarded to the server infrastructure (D2S), whereas the server infrastructure will share the various device data (S2S), possibly providing it back to devices, analytical environments, people and any other subscriber for a specific type of data.

In regards to a VNS, the specific machines are handheld or wearable devices and corresponding servers. Hence, a device-to-server (D2S) infrastructure and a protocol that will secure this communication environment against data loss and eavesdropping, fulfills the basic requirement in the context of a VNS approach. A communication protocol of this type is the commonly used MQ Telemetry Protocol (MQTT) [5]. As device-to-device communication is not necessarily needed within a VNS approach, a pub/sub messaging system similar to a push notification system as lightweight as MQTT offers a suitable approach to fulfil the systems' communication requirements. A more in-depth view about MQTT and similar pub/sub systems will be discussed in Section II.

D. Big Data in the context of a VNS

In a data-driven society, massive amounts of data are being collected from people, sensors, algorithms and of course, the Web itself; storing it in conventional database systems (i.e., online transaction processing) or data warehouses (i.e., online analytical processing) that itself conform to an additional layer on top of single or multiple databases. The term Big Data describes the challenge for handling this continuously increasing data, whereas mainly three reasons posture the arising difficulties: the sheer volume, the velocity (how fast new data is continuously produced) and the variety of different data-types. For some time, an additional challenge has been observed; the so called veracity, which describes the challenge to exclude uncertainty and inconsistency within the collected data.

The VNS must handle these challenges gracefully and overcome the resulting difficulties with scalability and reliability in terms of the technologies that are being implemented. In general, the system approach that is to be illustrated in the upcoming sections of this work will be able to handle large amounts of continuously generated input data and will furthermore be able to detect faulty (i.e., inconsistent) information in an online matter.

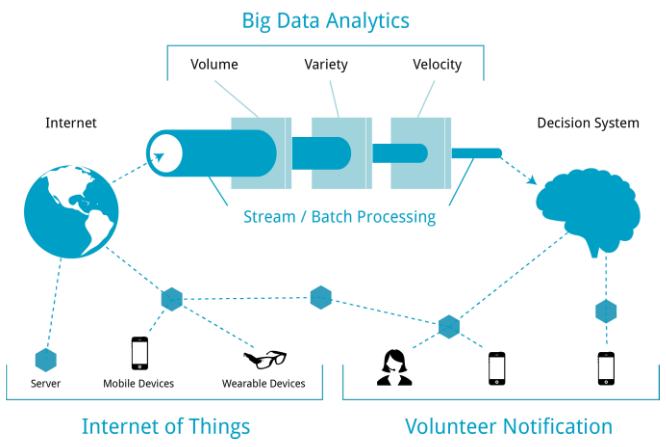

Figure 1. Big Data Analytics within a VNS

E. Stream Processing

As computer systems are creating ever more data at increasing speeds, Hadoop-style batch processing has awakened engineers to the value of big data analysis, whereas the current trend is focusing on the demand for real-time processing. In essence, people do not only want all of their data analyzed, but they want it done as soon as possible, which is driving the current Big Data research trend towards so called high-velocity data [7]. Exemplary use cases within this context are real-time analytics, machine learning, and new generation of decision support and fraud detection systems [8].

The desire to extract real-time insight from high-velocity data led to the creation of so called Stream Processing Engines. These engines include open source projects, such as Twitter's Storm [9], Apache Spark [10] and LinkedIn's Samza [11] as well as proprietary solutions, such as Amazon Kinesis [12] or Google's BigQuery [13]. These engines provide functionalities for routing, transforming and analyzing streams of data at high-velocity for a specified time window or near real-time (depending of the velocity and volume of streamed data chunks). The classical approach in this context would instead store the real-time data in order to apply data warehousing techniques for batch-processing in a subsequent matter. Figure 1 illustrates the conceptual coherence of the IoT paradigm and real-time Big Data Analytics within the context of an intelligent volunteer selection system.

II. State Of The Art

A. Pub / Sub Messaging Systems

Publish-subscribe is a messaging pattern in which occurring messages are not sent directly to a target receiver but rather published to a channel. Subscribers have the option to subscribe themselves on specific topics or channels and hence express their interest on receiving specific messages. The result is a lose coupling between publisher and subscriber, as they are unaware of each other.

In many pub/sub systems, publishers post messages to an intermediary message broker or event bus, and subscribers register subscriptions with that broker, letting the broker perform any type of necessary filtering. Pub/sub Messaging Systems allow implementation of a device-to-device, device-to-server and server-to-server interface, as have been introduced earlier.

The MQTT protocol on the other hand is a lightweight messaging protocol that uses a publish/subscribe architecture to deliver messages over low bandwidth or unreliable networks with a low footprint. Compared to a classical REST/HTTP implementation [14], MQTT imparts various advantages for the use within mobile applications, such as faster response times, higher throughput, higher messaging reliability, lower bandwidth usage and lower battery consumption

In this context, Apache Kafka [15] is a publish/subscribe log for integrating data between applications, stream processing, and Hadoop data ingestion. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. The design is heavily influenced by transaction logs to prevent data corruption and/or loss. On the server side, Apache Kafka will be used to create a pipeline between the MQTT broker cluster and the Hadoop/Spark environment to persist and stream process data; it will be managed by Apache Zookeeper for scalability and reliability purposes.

An alternative to MQTT in a proprietary environment are Amazon SNS, Amazon SQS as well as Amazon Kinesis, which are all capable of real-time streaming/distributing data between applications merely within Amazon Web Services (AWS) [16].

B. *The Apache Hadoop Ecosystem*

Apache Hadoop [17] is an open source software project that enables the distributed batch processing of large data sets across clusters of commodity servers. It is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance. Hadoop is supplemented by an ecosystem of Apache projects, such as Pig, Hive and Zookeeper and many more, which extend the value of Hadoop and improves its usability. The core part of Hadoop is the Hadoop file system (HDFS) which comprises two major components: namespaces and block storage service. The namespace service manages operations on files and directories, such as creating and modifying files and directories, whilst the block storage service implements the actual data node cluster management, resulting block operations and replication.

Hadoop was often criticized [18] [19] for its open-source implementation of the MapReduce model [20] based on so called JobTrackers, which due to its problematic structure have be resolved with the implementation of Apache YARN [21] and MapReduce 2 in the scope of Hadoop 2.x. YARN is a resource manager that is based on separating the processing engine and resource management capabilities of MapReduce as it was implemented in Hadoop’s original approach. YARN is often called the operating system of Hadoop because it is responsible for managing and monitoring workloads, maintaining a multi-tenant environment, implementing security controls, and managing high availability features of Hadoop. One crucial advantage of YARN in the context of using the Hadoop ecosystem for the VNS implementation is that is allows multiple processing models to be implemented on top of HDFS, thereby allowing Apache Spark to fit into the Hadoop Ecosystem [22]. The resulting flexible architecture allowed companies as Amazon and Google to create cloud computing platforms (e.g., Amazon EMR and Google's Cloud Platform) which implement enterprise-features out of the box and give a transparent in-depth cost overview.

C. *Apache Spark*

Apache Spark is a cluster computing platform similar to Hadoop designed to be fast and of general-purpose. Spark extends the popular MapReduce model to efficiently support more types of computations, including interactive queries and stream processing. One of the main features that Spark offers, is the ability to run even huge computational queries fully in memory (split over various clusters), reaching performance gains of up to 100 times compared to general Hadoop MapReduce implementations under specific circumstances. However, the system itself is also faster than MapReduce when running merely on disc operations.

At its core, the Spark Engine itself is responsible for scheduling, distributing, and monitoring applications consisting of many computational tasks across many worker machines powered by a high-level structure of components. These components are designed to interoperate closely, supporting a library-like combination of the various data representations (graphs, matrices, SQL like queries). Spark revolves around the concept of a resilient distributed dataset (RDD), which is a fault-tolerant collection of elements that can be operated in parallel. There are currently two types of RDDs: firstly parallelized collections, parallelizing an existing collection in your driver program, and secondly by referencing a dataset in an external storage system supported by Hadoop (e.g., the local file system, HDFS, Cassandra, Amazon S3). This allows Spark to interoperate with various stable established solutions in order to efficiently focus on problems regarding the introduced big data challenges. A recent cloud service that is entirely based on Spark and runs on AWS has been introduced by Databricks (who also drove the adoption of the Apache Spark ecosystem) in 2014. It allows developers to create scalable computing clusters running on Apache Spark for data analysis, machine learning and similar use cases.

This work will incorporate Apache Spark and its core components as the main cluster computing platform to overcome weaknesses of classical Hadoop architectures and to support the incorporation of the various proprietary solutions, such as Amazon Web Services and the Databricks Cloud Platform.

D. *Data Streaming & Processing*

LinkedIn's Kafka was designed to support not merely the distribution of data, but also to provide the infrastructure primitives that will enable real-time data processing. Samza on the other hand provides elastic, fault-tolerant processing as being layered on top of real-time feeds. A simple analogy in respect to the batch domain is described by Kafka taking the role of HDFS while Samza relates to MapReduce.

While this architecture scales horizontally due to its MapReduce nature, speed is an important factor which needs to be considered. A combination of Apache Kafka with various Spark components (i.e., Spark SQL, MLlib and Streaming Processor) will result in a more reliable, vertically and horizontally scalable high-velocity architecture. The lack of security options within Kafka and Samza are an important criteria for using Spark's Security implementation and an integrated secure tunnel between Kafka and the corresponding MQTT brokers.

In terms of security, scalability and reliability a commercial solution with Amazon Kinesis and Amazon Elastic MapReduce provides leverage to these problems, including the high-velocity implementation of Spark components, which replicates the scenario in a more enterprise-ready fashion.

As the fault tolerance plays an additional key role for a successful scalable VNS implementation, Apache Cassandra [23] is the state-of-the-art database system in combination with Spark technologies; highly robust and fault tolerant. It protects against data loss or corruption by replicating blocks of data to multiple nodes and supporting replication between geographically distributed nodes. Amazon and Google offer similar enterprise ready data stores, such as Amazon Redshift [24], Amazon DynamoDB [25] and Google Cloud Datastore [26], whilst a general comparison between the Cassandra File System (CFS) and HDFS is given in [27].

E. *Webinterfaces & API*

Responsive web design architecture and supporting the HTML5 specification, esp. Websocket support [28], is efficiently incorporated by implementing Nginx [29] as a high-performance HTTP server for both, static web data as well as proxy requests to an underlying Node.js [30] runtime environment running server-side applications. Node.js applications are entirely written in JavaScript, whereas Express.js constitutes an adaptable MVC framework [31]. Node.js is characterized to be fast (due to event based architecture), offer high throughput, support high amounts of concurrent connections, support clustering and generally has a very low resource footprint. Offering advanced scalability, load balancing, health checks and some additional features, the Nginx Inc. released an enterprise version under the label: Nginx+ [32]. Node.js in this context enables the implementation of simple server applications as well as the requirements in respect to APIs.

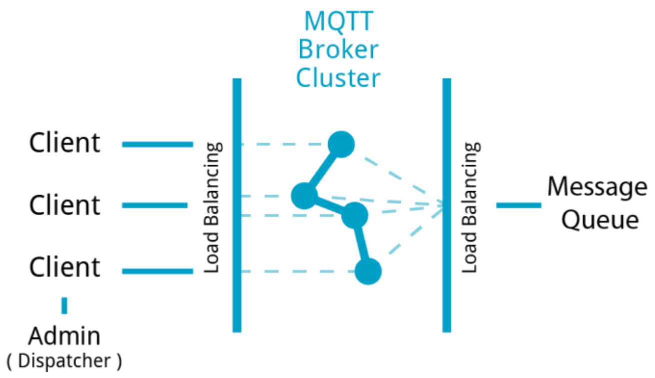

Figure 2. Realtime Data Pipeline

III. SYSTEM ARCHITECTURE

This section will illustrate the main strategy that incorporates the introduced technologies into a general system architecture that conforms to the requirements of an enterprise application.

A. *Realtime Data Pipeline*

Within a VNS, the data that is to be analyzed is generated by individual mobile or wearable devices. As illustrated in Figure 2, clients publish their data to a server which is connected to a message broker, which is responsible for broadcasting the received messages to the corresponding subscribers. Whilst a standard MQTT broker solution is lightweight and performant for a limited amount of connected clients (due to limits in the port range), a horizontally scalable approach will have to balance the various connections between multiple instances (load balancing) residing on different machines. As clients generally subscribe to specific topics in order to achieve push-like notifications, horizontal scaling will result in brokers having different information and topic structures.

To solve this problem, the various brokers (i.e., nodes) need to be connected with each other and share their message structure and permissions, forming a cluster of machines that can be scaled at will. Modern systems, such as RabbitMQ [33] and Apache’s ActiveMQ [34], support the application of efficient clustering. Mirroring the message queues between all machines will allow the subscribers to connect to any existing node while still having access to the whole cluster. Established commercial projects that support scalable messaging systems and efficient load balancing for MQTT connections are: HiveMQ [35], CloudAMQP [36] and CloudMQTT [37].

B. *Load Balancing*

Since most standard load balancing approaches, such as Amazon's Elastic Load Balancer, only support Round Robin (RR) and Session Sticky Algorithms, they are not sufficient for balancing MQTT clients or applications between brokers. The already introduced commercial Nginx+ solution supports various advanced load balancing strategies [38], but even the open-source standard Nginx version can be extended with additional functionalities by incorporating the programming language LUA and a TCP-proxy module to support the programmatic injection of algorithms that can filter requests of clients and balance connections between brokers with high performance. This added functionality enables a distinguished consideration of the various active brokers in order to terminate obsolete sessions, run additional scripts for scaling the cluster, and perform regular health checks on running instances.

C. *Ad-Hoc / Online Computation*

As described in [39], ad-hoc computation on message brokers is efficiently achieved by combining Apache Kafka with Apache Sparks infrastructure; since Kafka efficiently persists the message queue on a data store (e.g., Cassandra or HDFS) while Apache Spark handles workloads both in real-time as well as by batch processing. Kafka is guaranteed to deliver reliable message durability and a fault-tolerant near real-time computation with Spark Streaming [40]. At this point, one might argue about missing security measures within Apache Kafka [41].

Whereas various other messaging platforms (e.g., RabbitMQ) support the persistence of incoming data on data stores, they are usually not performant enough or simply not optimized for processing environments such as Kafka, which itself is very robust in throughput of messages and during read/write operations [42]. Whilst the Apache Spark libraries provide methods for connecting to MQTT brokers and streaming data, the underlying communication has to be implemented manually. In contrast, Kafka can be implemented as a complementing stream processing layer between the MQTT cluster and Apache Spark [43].

Within a VNS, the streamed data will mainly consist of location data of individual volunteers and case update data. Thus, stream processing will be applied to regulate updates concerning a specific case in real-time; deriving decisional, predictive or anomaly detection results. However, an efficient volunteer selection, based on accumulated profile data, will mostly be computed in batches, as discussed in the upcoming section.

As data store, Apache Cassandra constitutes a high performance scalable database with linear scaling that secures an enterprise-ready solution for this work. Similar, proprietary options are Amazon DynamoDB and Amazon RedShift, whereas HDFS would partly limit the performance of Spark and other NoSQL data stores [44].

D. *Batch/ Offline Computation*

Batch processing on big amounts of accumulated data is commonly implemented based on Hadoop clusters. Within a VNS, finding the most reasonable candidates for an ongoing medical emergency – within a minimum time interval – hereby constitutes a batch processing problem with an increasing (raw) data size over time. Location based data will be analyzed in order to compute behavioral patterns of volunteers; this can be done on a regular basis (iterative) based on batch processing of the acquired location data and in combination with various machine learning algorithms. The results will be available for additional real-time computations, whereas details for an AI-driven volunteer selection discussed in [45].

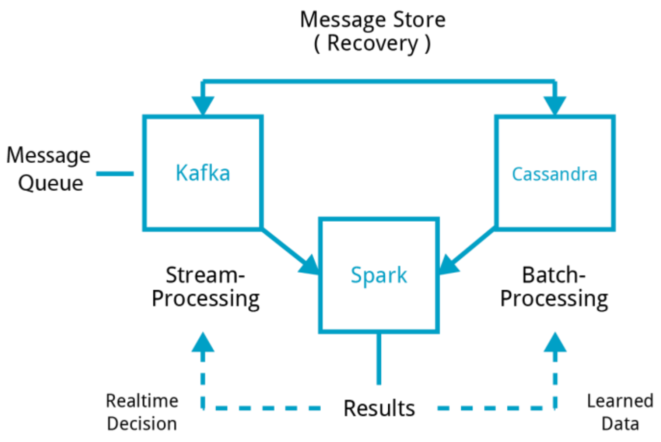

Figure 3. Real time Data Processing

Apache Spark can accomplish both tasks of on- and offline computation quite reliable and fast allowing the results to be stored in data stores or be directly accessible via API or MQTT subscribed topics. While real-time per-case computation and live updates would amass resource consumption it would be possible but unfeasible and unnecessary. With a proper modelling of data stores direct API access allows fast updates of a case without the need of costly computations. Behavioral patterns can be learned after an emergency scenario, as well due to the systems structure.

Figure 3 illustrates the general architecture for a real-time data processing environment, as has been discussed within this section. Nowadays, Amazon EMR, Google Cloud Platform and Databricks deliver the technologies needed for a successful computation environment for similar use cases and allow different services like data stores or real-time computation ecosystems to be fully implemented on commodity hardware.

IV. CONCLUSION

This work illustrated details on how to implement a VNS into a distributed analytical environment with high velocity data support. Scalability and reliability is hereby achieved by utilizing merely open-source software solutions without relying on any commercially driven software or proprietary cloud solutions. While security and special solutions for load balancing and regulating the corresponding environments cannot be guaranteed by open-source Apache software alone, new Big Data challenges arise continuously and more open-source projects are being incubated or upgraded; hopefully solving both, newer as well as older challenges that were formally limited to enterprise solutions.

ACKNOWLEDGMENT

This paper is based on work done in the INTERREG IVa project EMuRgency (www.emurgency.eu). The project is partially financed through the European Regional Development Fund (ERDF) and co-financed by several regions and partners of the EMuRgency consortium.

REFERENCES

[1] T. Danova, [online], http://read.bi/1hwjWTr [accessed 08/10/2014].

[2] M. Stonebraker, U. Cetintemel, and S. Zdonik, “The 8 requirements of real-time stream processing”, ACM SIGMOD Record, Vol. 34, No. 4, 2005, pp. 42-47.

[3] J. Elsner, M. T. Schneiders, M. Haberstroh, D. Schilberg, and S. Jeschke, “An Introduction to a Transnational Volunteer Notification System Providing Cardiopulmonary Resuscitation for Victims Suffering a Sudden Cardiac Arrest”, eTelemed2013, April 2013, pp. 59-64.

[4] J. Elsner, P. Meisen, S. Thelen, D. Schilberg, and S. Jeschke, “A Basic Concept for an AI Driven Volunteer Notification System for Integrating Laypersons into Emergency Medical Services”. International Journal On Advances in Life Sciences, Vol. 5, No 3&4, 2013, pp. 223–236.

[5] J. Elsner, M. T. Schneiders, D. Schilberg, and S. Jeschke, “Determination of Relevant First Aiders within a Volunteer Notification System“, Med@Tel 2013, Luxemb., pp. 245-249.

[6] MQTT Protocol 3.1.1. Spec., [online], http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html [accessed 02/10/2014].

[7] C. Aggarwal, “Data Streams: Models and Algorithms”, Series: Advances in Database Systems, Vol. 31, Kluwer, 2007.

[8] J. Taylor, “Real-Time Responses with Big Data”, 2014.

[9] Apache Storm [online], https://storm.incubator.apache.org/ [accessed 10/12/2014].

[10] Apache Spark [online], https://spark.apache.org/ [accessed 10/14/2014].

[11] Apache Samza [online], http://samza.incubator.apache.org/ [accessed 10/14/2014].

[12] Amazon Kinesis [online], http://aws.amazon.com/kinesis/ [accessed 10/14/2014].

[13] Google BigQuery [online], https://cloud.google.com/bigquery/ [accessed 10/14/2014].

[14] Wilde, Erik, “Putting things to REST”, School of Information, Series: Recent Work, November 2007.

[15] Apache Kafka [online], http://kafka.apache.org/ [accessed 10/15/2014].

[16] Amazon Webservices such as SNS or SQS [online], http://aws.amazon.com/sns/ [accessed 10/15/2014].

[17] Apache Hadoop [online], http://hadoop.apache.org/ [accessed 10/14/2014].

[18] J. Polo, "Big Data Processing with MapReduce", Big Data Computing, Oktober 2013, pp. 295-313.

[19] M. Stonebraker et al. “MapReduce and parallel DBMSs: friends or foes?”, Communications of the ACM, Vol. 53, No.1, 2010, pp. 64-71.

[20] R. Lämmel, "Google’s MapReduce programming model Revisited" Science of computer programming, Vol. 70, No. 1, January 2008, pp. 1-30.

[21] V. K. Vavilapalli et al. "Apache hadoop yarn: Yet another resource negotiator", SOCC’13, ACM New York, 2013, pp. 1-16.

[22] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets", HotCloud'10, USENIX Association 2010, pp. 10-10.

[23] Apache Cassandra [online], http://cassandra.apache.org/ [accessed 10/15/2014]

[24] Amazon RedShift [online], http://aws.amazon.com/redshift/ [accessed 10/15/2014]

[25] Amazon DynamoDB [online], [accessed 10/15/2014], http://aws.amazon.com/dynamodb/

[26] Google Cloud Datastore [online], [accessed 10/15/2014], https://cloud.google.com/datastore/

[27] Datastax Corporation, "Comparing the Hadoop Distributed File System (HDFS) with the Cassandra File System (CFS)", White Paper by Datastax, August 2013.

[28] I. Fette and A. Melnikov, "The websocket protocol", IETF, December 2011.

[29] W. Reese, "Nginx: the high-performance web server and reverse proxy", Linux Journal, Issue 173, September 2008.

[30] R. Fielding et al. "Hypertext transfer protocol –HTTP/1.1", RFC Editor, USA, 1999.

[31] C. Le and X. Yang, "Research of applying MVC pattern in distributed environment", Computer Engineering, Vol. 32, No. 19, 2006, pp. 62-64.

[32] Nginx+ [online], http://nginx.com/products/technical-specs/ [accessed 10/16/2014].

[33] RabbitMQ [online], http://www.rabbitmq.com/ [accessed 10/16/2014].

[34] ActiveMQ [online], http://activemq.apache.org/ [accessed 10/16/2014].

[35] HiveMQ [online], http://www.hivemq.com/ [accessed 10/16/2014].

[36] CloudAMQP [online], https://www.cloudamqp.com/ [accessed 10/17/2014].

[37] CloudMQTT [online], http://www.cloudmqtt.com/docs.html [accessed 10/17/2014].

[38] A. Piórkowski, A. Kempny, A. Hajduk, and J. Strzelczyk, "Load balancing for heterogeneous web servers", Computer Networks, Springer 2010, pp. 189-198.

[39] Real time Analytics with Apache Kafka and Apache Spark [online], http://bit.ly/1Djbstn, [accessed 10/17/2014].

[40] M. Zaharia et al. "Discretized streams: Fault-tolerant streaming computation at scale", Proc. of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP’13, 2013, pp. 423-438.

[41] Samza / Kafka Security [online], http://bit.ly/1pl550b [accessed 10/17/2014]

[42] J. Kreps, N. Narkhede, and J.Rao, "Kafka: A distributed messaging system for log processing", NetDB 2011.

[43] R. Ranjan, "Streaming Big Data Processing in Datacenter Clouds", Cloud Computing, Vol. 1, No. 1, IEEE, 2014.

[44] J. Zollman, “NoSQL Databases”, Proceedings of the NetDB, 2011, [online], http://sewiki.iai.uni-bonn.de/_media/teaching/labs/xp/2012b/seminar/10-nosql.pdf

[45] J. Elsner, Dissertation, “An AI Driven Volunteer Selection System”, Aachen 2015, (unpublished)