A CKAN Plugin for Data Harvesting to the Hadoop Distributed File System

Robert Scholz, Nikolay Tcholtchev, Philipp Lämmel and Ina Schieferdecker
Fraunhofer Institute for Open Communication Systems (FOKUS), Berlin, Germany

Keywords: Smart Cities, Open Data, Distributed Processing, Hadoop, CKAN.

Abstract: Smart Cities will mainly emerge around the opening of large amounts of data, which are currently kept closed by various stakeholders within an urban ecosystem. This data requires to be cataloged and made available to the community, such that applications and services can be developed for citizens, companies and for optimizing processes within a city itself. In that scope, the current work seeks to develop concepts and prototypes, in order to enable and demonstrate, how data cataloging and data storage can be merged towards the provisioning of large amounts of data in urban environments. The developed concepts, prototype, case study and belonging evaluations are based on the integration of common technologies from the domains of Open Data and large scale data processing in data centers, namely CKAN and Hadoop.

1 INTRODUCTION

Data Processing is at the heart of future Smart Cities. Current Smart City projects often include the provisioning of a metadata storage which helps to keep track of the available datasets and offers a unified way of accessing information regarding belonging data, such as format, size and licensing. Smart City solutions utilize the data described in metadata catalogs and subsequently require the integration of big and diverse data on distributed systems. Currently, there seems to be a lack of attempts and research efforts concerning the seamless integration of such systems and the provisioning of a unified interface and data processing capabilities in direct association with a metadata catalog. As of now, required data(-sets) need to be manually collected, transferred onto the processing system and kept up-to-date, thereby forfeiting potential advantages offered by the aforementioned cataloging systems.

The *Comprehensive Knowledge Archive Network* (CKAN; CKAN Association, 2015) is one of the two most widely used metadata storage systems and a core component in various Smart City projects (Matheus and Manuella, 2014; Lapi et al., 2012; Marienfeld et al., 2012). CKAN's metadata catalog holds entries for resources from a diverse set of sources and is updated automatically on a periodic basis in order to ensure the topicality of the data. Its harvester (Mercader et al., 2012) extends the manual way of publishing of datasets by enabling the automatic inclusion of sources such as other metadata hubs or similar by means of harvester-plugins. *Hadoop* (The Apache Software Foundation [TASF], 2017a) is a popular and the currently most widely used open source framework for distributed storage and processing of big amounts of data.

This paper describes a novel concept for the integration of CKAN and the Hadoop Distributed File System (HDFS; Shvachko et al., 2010), which is the starting point for further storage and processing within other Hadoop subsystems. The concept builds on the CKAN platform and utilizes the core structure of a CKAN plugin. This concept and belonging prototype is denoted as *HdfsStorer* plugin and constitutes the key contribution of this work. It serves the purpose of integrating CKAN as a metadata store with the powerful capabilities of Hadoop in order to enable the efficient handling of large (open) data sets in urban environments.

As already mentioned, metadata catalogs such as CKAN, even though providing some features, can hardly satisfy the requirements - with respect to data storage - posed to a logically centralized data hub, but can instead deliver an entry point for its realization. Distributed file systems, such as the HDFS, are more suitable for big data storage.

The HDFS is the underlying distributed file system for storing data for the Hadoop data

Scholz, R., Tcholtchev, N., Lämmel, P. and Schieferdecker, I.
A CKAN Plugin for Data Harvesting to the Hadoop Distributed File System.
DOI: 10.5220/0006230200470056
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 19-28
ISBN: 978-989-758-243-1

processing system. Its inherent reliability and scalability with regard to storage capacity and computing resources renders the HDFS suitable for coping with the vast amounts of data encountered in a Smart City. Therefore, in the scope of this work, CKAN is extended by the possibility for uploading the data referenced in its metadata catalog to the HDFS. This extension - designed, developed and evaluated in the scope of German national and European projects related to Open Data and Smart Cities - is presented in the current paper and referred to as *HdfsStorer*.

The following paragraphs describe this extension in detail and outline its advantages and disadvantages. The second section encompasses a short review of already existing tools and is followed by section 3 that elaborates the functional and non-functional requirements for the plugin. Afterwards, the architecture of the plugin is described in section 4 and its performance is evaluated by means of a prototype (described in section 5) and a use-case scenario in section 6. Furthermore, a description of a possible real world Smart City application for the plugin is provided in section 7. The paper is concluded by a general discussion section and a summary of the contribution.

2 RELATED WORK

In addition to the HDFS, a multitude of other distributed file storage systems exists, such as Apache Cassandra (TASF, 2017b; Fan et al., 2015) - a distributed design with multiple entry points/where each node can also act as a master, or Open Stack Swift and Cinder (Rosado and Bernardino, 2014), which are distributed object and block storages respectively, abbreviated as OS-S/C (Open Stack – Swift and Cinder). Commercial solutions, such as the Ceph file system (CephFS; Weil et al., 2006) and the Amazon S3 cloud (Amazon.com, 2017) are also available. Furthermore, there are a variety of add-ons or pluggable systems that allow for the interoperability of the various associated technology stacks. These include OS Sahara (Thaha et al., 2014), formerly known as Savanna, which allows Hadoop processing engines to work on data stored on OS-S/C, as well as the CephFS Hadoop plugin (Red Hat, 2014), that makes data stored in the CephFS available to the above mentioned Hadoop processing engines. In this work we will focus on the HDFS as it offers the most straightforward entry point for further processing by a maximum amount of processing engines.

Stream processing tools such as Kafka (Kreps, Narkhede and Rao, 2011) already provide the possibility of pushing streaming data (in contrast to batch data, which shall be the focus of this work) from a variety of sources onto the HDFS, where it will be available for long term usage. So far, there is also no integration with metadata portals that register and catalog such data streams. However, as the amount of data, which ultimately has to be manually transferred, is limited to access information about the stream - e.g. Kafka related information for accessing the data stream - such integration is also not of major importance.

To shift batches of data from one repository to another, appropriate protocols are needed. Ahuja and Moore (2013) pointed out a weakness regarding the consumption of computing resources by the Transmission Control Protocol (TCP) during the transfer of big amounts of data. Therefore Tierney et al. (2012) suggest the usage of the Remote Direct Memory Access protocol over Converged Ethernet (RoCE) for data transfer to/from and between repositories.

To the authors' best knowledge, the question of automated data import on the basis of metadata storage engines/catalogs has not been addressed yet. As an example, Khan et al. (2015) state that they use the data from the Bristol Open Data portal (Bristol City Council, 2015), but do not mention how it is being accessed and transferred to the utilized distributed file system. The current work aims at bridging this gap on the concept level, as well as on the level of case studies and prototype implementations.

3 REQUIREMENTS FOR THE HDFS-HARVESTER

Based on the inherent properties of big data itself and with regard to the presence of very diverse environments in which both CKAN and Hadoop find their applications - a similar diversity can also be found within the field of Smart Cities - the following main requirements for an integrating component/plugin can be derived:

Req. 1: Seamless integration of the *HdfsStorer* with CKAN: The user experience should not be disturbed by the plugin running in the background.

Req. 2: Timeliness of resources: Every time a new resource is created and provided, an existing resource updated or respectively deleted, these changes should be mirrored on the distributed file system (HDFS).

Req. 3: Network Usage Economization: Files on the HDFS should only be updated if the source data has changed, not upon every harvesting process. (CKAN checks periodically for changes in remote datasets).

Req. 4: Completeness: Files irrespective of their size or format should be uploaded to the file system. Especially, there should be no upper limit for file upload as has been the case with the CKAN-internal Filestore.

Req. 5: Backwards-compatibility: A way for importing datasets to the HDFS that have been registered with CKAN prior to the activation of the plugin should be available.

Based on these key requirements, the next sections proceed with devising the architecture of the *HdfsStorer* plugin, evaluating it based on a prototype, as well presenting the belonging case study and measurements.

4 ARCHITECTURE OF THE HDFS-HARVESTER

CKAN offers different points of entry for plugin development. These are given by *programming interfaces*. Mainly there are three important events to consider in the lifecycle of a data resource: 1) the creation of a new resource, 2) the update of an existing resource, and finally 3) the deletion of it. The interface, which is foreseen to handle these events, is the *IResourceController* within the Python based CKAN platform, offering the functions *after_create*, *after_update* and *before_delete*. The other functions provided are left out of consideration, as they won't have to be implemented.

There are certain peculiarities in CKAN with respect to metadata and dataset cataloging that have to be considered. In CKAN, a package generally describes a set of data and stores information about the dataset along with all given attributes and a list of the resources belonging to it. These resources describe single external files (actual data), referenced by an URL. Resources and packages, which are deleted within CKAN, remain within their respective PostgreSQL (Momjian, 2001) database and only a single attribute is changed, which prevents them from appearing in the catalog. A user with appropriate authorization can still access them. Only after their *purging* by a system administrator, either through the CKAN web interface, or from the corresponding database directly, resources and packages are fully removed. Unfortunately, there is no possibility for intercepting purge events (e.g. a function called "after_purge"), so that in the course of each deletion, i.e. on each call of the *before_delete* function, the belonging dataset file has to be fully removed from the HDFS storage. This procedure relates to another interesting aspect within CKAN given by the fact that package deletion does not automatically result in the deletion of the embedded resources, i.e. resources included/referred in this package/metadata. Therefore, complementary to the core *HdfsStorer* functionality, each package deletion event has to be intercepted by implementing the IPackageController interface and belonging embedded resources must be removed.

The description of the architecture is further refined in the next subsections by introducing a view on the components of *HdfsStorer* as well as on the process flows and interactions amongst these components.

4.1 Components and Dynamic Aspects

The general architecture of the plugin including a flow of operations – i.e. an enumeration of a sequence of operations – is illustrated in Figure 1. Thereby the *HdfsStorer* plugin is depicted as accommodated within the CKAN harvesting eco-system, since it essentially resembles a CKAN extension.

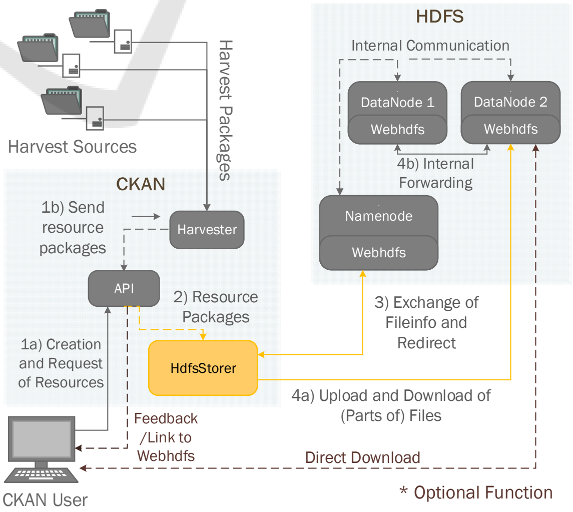

Figure 1: Plugin Architecture (highlighted in orange). Deletion pathway is not included.

Once a resource is created or updated with respect to its metadata in CKAN, the referenced data resource (file) needs to be uploaded to the HDFS - operations (1a), (1b), (2), (3) and (4a-b) in Figure 1. In turn, if a resource or package is deleted, the corresponding resource files have to be removed again. These files can be identified by means of the resource IDs, as for each ID a separate directory is created on the file system.

CKAN's native deletion events trigger API calls to the methods implemented by the *HdfsStorer* plugin. Upon creation, update and deletion of a resource in CKAN, the corresponding ID and a reference to the up-to-date file are carried along in the parameters. If an entire package is to be deleted in CKAN, only the package ID is given, requiring an additional lookup in a database table for resource identification and removal.

Communication and data exchanges - mainly operations (3-4) in Figure 1 - between the *HdfsStorer* plugin and the HDFS (in the upper right part of the figure) are managed over the WebHDFS - a Representational State Transfer-API for Hypertext Transfer Protocol/TCP-based manipulation of resources stored on the HDFS (TASF, 2013).

The employed WebHDFS operations are given in Table 1 and encompass functionalities such as 1) checking for the existence of a file or directory, 2) the access and download (of parts) of a file, 3) the creation of directories, 4) access to the location for writing a new file thereby overriding old resource files, 5) appending to a newly created resource file and 6) the deletion of resource files and folders. The order in the table corresponds to the order of the enumeration.

Table 1: List of WebHDFS operations.

Method	Operation	Fields	HTTP Return Type
GET	liststatus		200 (OK) + JSON
GET	open		200 (OK) + FILE
PUT	mkdirs		200 (OK) + JSON
PUT	createfile	data=' '	203 (redirect)
POST	append	data; content-type	200
DELETE	delete		200 + JSON

5 PLUGIN IMPLEMENTATION

In the following, the detailed description of the plugin implementation is provided, starting with the procedure for resource creation and updates.

5.1 Creation and Update of Resources

Before a new resource is created on CKAN - either by means of manually adding it through the web interface or by harvesting it from another source - the *before_create* methods found in all plugins classes implementing the *IResourceController* interface are called whereby the data (dictionary) structure holding information about the new resource is passed as parameter. This is followed by internal addition of the resource in CKAN and the call of the respective *after_create* methods. A similar path is followed during the update and deletion of resources.

The procedure implemented within the *HdfsStorer* plugin is illustrated in Figure 2. Upon the call of the after_create method implemented by the *HdfsStorer* plugin, a folder named after the ID of the resource is created on the HDFS. This folder can be found inside the resource storage folder, which has been previously created on the HDFS and specified in an additional parameter within the CKAN configuration. Subsequently, a new empty file with identical name to the remote resource file is created. For further appending of data, a redirect to the HDFS DataNode, on which the new empty file lies, is provided. Consequently, chunk by chunk of the original file is read and appended to the previously created file. By chunking and appending, even (larger than the machine's memory) files can be transferred - the biggest file transferred during testing had a size of roughly 30 GB. Therefore, **Requirement (4)** can be deemed as fulfilled. Should the file size exceed the specified HDFS block size, the remaining data is then automatically forwarded to a newly created block on a different DataNode. Data replication is also conducted on the fly in the background.

The concept of hashing plays an important role in the current circumstances. As only resource files should be updated, whose contents have actually changed since the last update, a hash check was implemented (on the right in Figure 2), satisfying **Requirement (3)**. Normally, a check sum is built on the basis of entire files.

For the computation of a checksum, the files normally have to exist locally on one machine. As it cannot be assumed that every data provider will also include appropriate checksums along with their files and the HDFS only provides a so called distributed checksum for files stored on it, both files would have to be downloaded to a single machine in order to calculate and compare their checksums. This would result in additional network traffic and thus requires further elaboration.

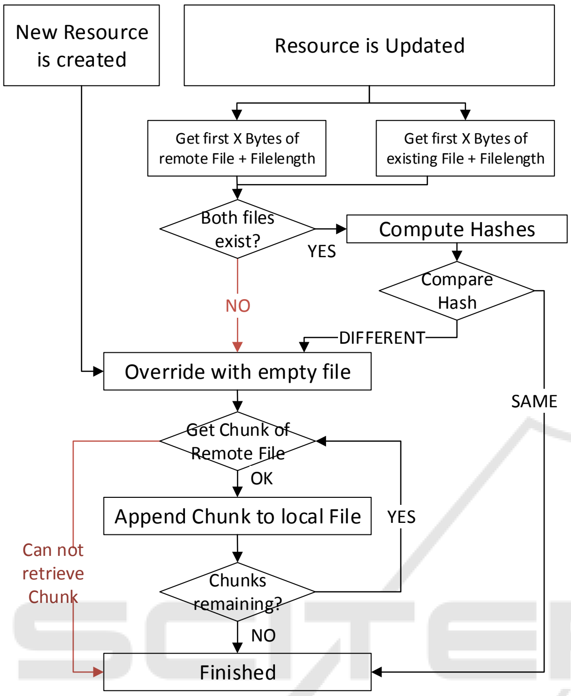

Figure 2: Upload of new or updated Resources to the HDFS through the Plugin. Possible occurring errors are shown in red.

Depending on the nature of the files implied in the application context, one of the two following approaches can be taken instead: 1) the comparison is skipped entirely and on every update the resource file is uploaded anew to the HDFS or 2) comparison of the files is done on the basis of a partial checksum. For the (CKAN-)*HdfsStorer* plugin the second approach (partial checksum check) was taken in anticipation of the large bodies of data needed for the realization of Smart City solutions. It has to be noted, that the filename is not taken into consideration for the creation of this partial checksum, as simple renaming of a file does not necessarily come with a change in the contents of a file. A difference in file size is a good indicator for differences in the contents of a file and therefore the file size makes up one constituent for the hash key. Files like logs, which do constitute a big part of what has to be processed on big data engines, are usually changed by either appending to the back or front of a file. Usually, this would also trigger a change in file size, and thus be detectable by looking at the file size only. Given the special case of a constant size log - such as for example logs only saving data for the X-recent days, or only saving X-number of entries - this does not hold anymore. By chunking a specific number of bytes at either the front or the back of a file (or both) and including these as further ingredients into the hash key, changes of such files could be detected. If the number of bytes is high enough, the entire checksum of smaller files (e.g. configuration files or images) is computed within the presented approach.

The downside is that files with no difference in file size after update, which are larger than the defined chunk size, with static header (beginning of the file) and footer (last bytes of the file) will still remain undetected. These files are not suited very well for distributed processing, as they are usually hard to split. Furthermore, the very rare case of having only minor differences in a large splittable, identically sized file remains undetected. As usually individual items from a big dataset are only of minor importance to the final result (after processing the entire dataset) and the more single items are changed, the more likely a difference in file size can be detected, this drawback can be deemed acceptable in the majority of cases.

5.2 Parallel Upload of Data

In order to not negatively influence the performance of the CKAN system and thus conform to **Requirement (1)**, the decision was taken to not have the data uploads to the HDFS run in parallel, as - depending on the specific setting - many concurrent uploads will likely use up the entire bandwidth of the server. This should be noted when setting the repetition period for harvesting, as a single harvesting circle might be slowed down markedly and might result in an ever-increasing queue of harvesting jobs and thus outdated data. The setup of a second CKAN server only for the purpose of harvesting with periodical synchronization of its database with the main CKAN server could allow for parallel data upload without infringing on **Requirement (1)**. Similar setups are already under development and employment.

5.3 Deletion of Resources

The deletion of a single resource on CKAN results in the call of the *before_delete* function of the *IResourceController* interface implemented by the plugin. Package deletion in CKAN does not result in the deletion of the respective resources and thus does not result in the call of any functions from the *IResourceController* interface. Therefore, package deletion has to be intercepted by the (CKAN-) *HdfsStorer* plugin by implementing the *after_delete* function from the *IPackageController* interface. After retrieval of the corresponding resource IDs in both of these two function, the deletion of the specific directories on the HDFS and their contents is triggered through the WebHDFS API. This ensures **Requirement (2)**.

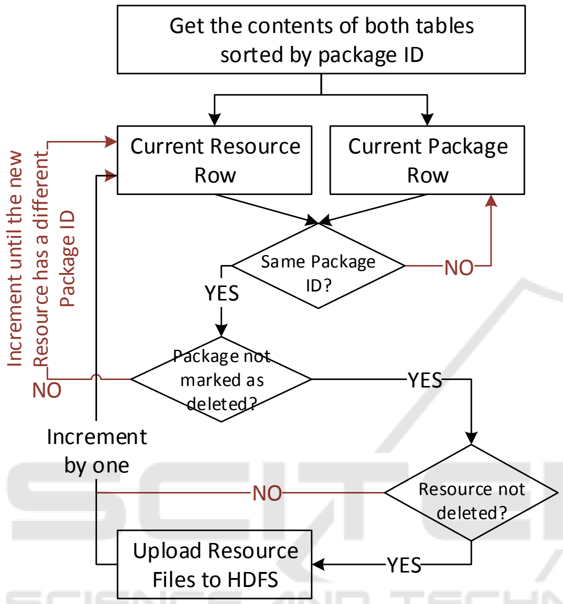
Get the contents of both tables sorted by package ID
Current Resource Row
Current Package Row
Same Package ID?
NO
YES
Package not marked as deleted?
YES
NO
Increment until the new Resource has a different Package ID
Increment by one
Resource not deleted?
NO
YES
Upload Resource Files to HDFS

Figure 3: Import of already existing Resources to the HDFS.

5.4 Backwards Compatibility

In order to fulfil **Requirement (5)**, which refers to the handling of legacy CKAN datasets, another module was developed. This module is basically in charge of reading the data in the *CKAN Datastore* (Winn, 2013; CKAN Association, 2017) and pushing it to HDFS thereby taking care of consistency. The process flow within this module is illustrated in Figure 3. The resource information for each resource registered within CKAN is read from the internal PostgreSQL-database and thereupon uploaded to the HDFS. In order to allow for fast recovery of recently deleted packages and resources, CKAN does not remove them directly from the database, but only marks them as deleted. These are excluded from uploading to the HDFS.

6 PROOF OF CONCEPT

In order to exemplify the feasibility and the workflow of the plugin, a use case was defined based on an algorithm from the field of Machine Learning on two different processing engines for classification of data points. The usage of two different systems is intended to be representative of the variety of paths the data can take once it has been imported to the HDFS (due to free choice of processing engines). Thereby, the *HdfsStorer* is the enabler for such evaluations by allowing large scale Big Data and Open Data to be integrated and efficiently used in the scope of Smart Cities. Building an application that takes as input a dataset, which is linked to in the CKAN-catalog, essentially consists of three major steps elucidated and exemplified in turn.

6.1 Import of the Dataset(S) to the HDFS by Means of the HdfsStorer

The used input dataset (Alinat and Pierrel, 1993) held entries about phoneme properties (such as place of articulation) and the corresponding classification of those into phoneme classes. It was split into a training and an evaluation part, stored in different files. Once the files are registered in CKAN, they are automatically transferred to the HDFS by the plugin.

6.2 Selection of the Appropriate Processing Engine and Program Logics

Both standard Hadoop MapReduce and its more flexible in-memory counterpart Spark (TASF, 2016) were used separately to train Artificial Neural Networks (ANN) on the training set. A standard backpropagation algorithm was employed for that purpose, the details of which can be found in Liu, Li and Miao (2010). The trained ANNs then served to classify the data points of the evaluation set.

6.3 Job Execution and Result Retrieval

Jobs were executed through the command line. The resulting classifications, based on the *HdfsStorer* data uploads, can be retrieved from the HDFS filesystem or the command console respectively. As the target of this work is not to evaluate the classification quality of different implementations, only the training step will be considered in the following, in order to exemplify the performance of two key approaches (Hadoop MapReduce and Spark).

Execution time was measured as the time difference between job application submission and job termination. The illustration of Spark and Hadoop execution times in Figure 4 indicates that for the current data set - imported over CKAN and the HdfsStorer plugin - not only Spark execution times are significantly shorter, whilst varying with the number of ANN iterations, but also rise more slowly than Hadoop execution times, probably due to its much lower overhead for each iteration.

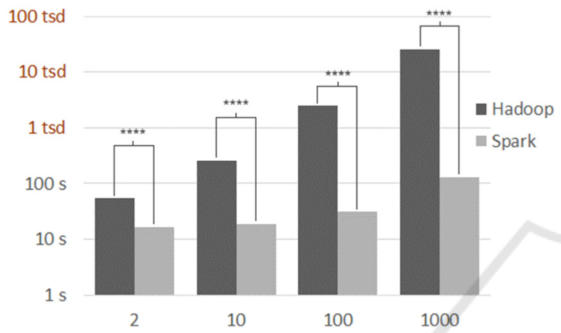

Figure 4: Comparison of Spark and Hadoop Execution Times. Four stars indicate a significant deviation of the means with a p-value less than 0.0001.

Moreover, Figure 5 depicts the mean memory usage of both Hadoop and Spark during idle and program execution with 100 iterations on top of the open dataset that was imported over the *HdfsStorer* plugin and CKAN. The vertical axis shows the memory usage in Mb.

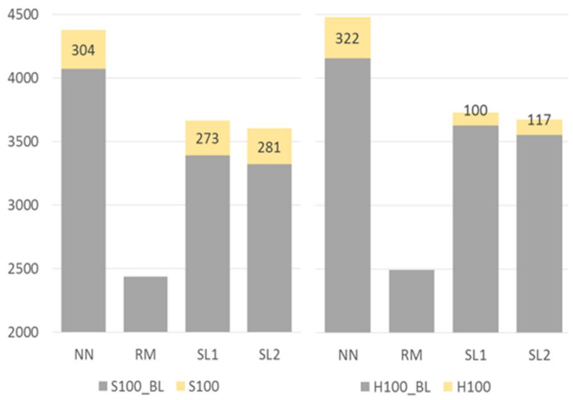

Figure 5: Hadoop (H) and Spark (S) Memory Usage during idle ("Baseline” BL) and work intensive Periods shown for Neural Network Training with 100 Iterations. NN: NameNode/MasterServer, RM: ResourceManager, SL1/2: Slaves.

Due to the comparably small size of the data set, expected tendencies (such as the much higher memory usage by Spark) in the absolute RAM usage statistics have not been evident. The difference between idle and work intensive periods is greater for both slaves in the Spark deployment than that of the slaves in the Hadoop deployment, indicating the stronger memory dependence of Spark during processing. Hence, we see on how *HdfsStorer* can enable the efficient evaluation of various distributed processing engines for various **real** word datasets and scenarios in Smart Cities and urban environments.

This evaluation gives an idea of the importance of choosing the right processing engine for the efficiency of the overall application. The usage of the HDFS thereby enables the free choice of the processing engine, as it is the basis for a wide range thereof. The evaluation can be done on a wider basis or can be targeting specific datasets or scenarios in Smart Cities. This setup is overall enabled by the *HdfsStorer* plugin, designed and prototyped in the current work.

7 A SMART CITY SCENARIO USING HADOOP AND CKAN

After having shown the general workflow of the plugin and evaluated its principal applicability, it is also possible to envision its working in the context of a more complex scenario. For that purpose, we envision the scenario that the public transport system of a forthcoming Smart City is to be streamlined and optimized. This includes schedule improvements following the dynamic identification of peak traffic hours and the possibility for both, delay and occupancy prediction integrated with trip planning, thus providing a better travel experience to passengers This can help, as an example, to answer questions such as “Will the airport express bus normally arrive on time and should I worry about fitting in along with my baggage? What about the next bus? Is it worth to wait another ten minutes?”.

The data required for this particular scenario is composed of two different classes: static and real-time. The current schedule and past occupancy and punctuality statistics along with the past road and weather condition records are spread over different data stores as static data and their location and other information is indexed in the CKAN-catalog. Periodical harvesting through CKAN keeps this catalog up-to-date. Information about the current weather, road and traffic conditions and the amount of passenger measured by sensors inside the transport vehicle are provided as streaming data.

Integration of these two types of data in Hadoop is sketched in Figure 6. The *HdfsStorer* plugin serves hereby as the intermediate for importing static data to the HDFS according to information provided in the CKAN-catalog (such as location of the original dataset). Thereupon, different processing engines, that can make use of the files stored on the HDFS and data streams provided through message brokers, integrate all this information and thus allow for said enhancements.

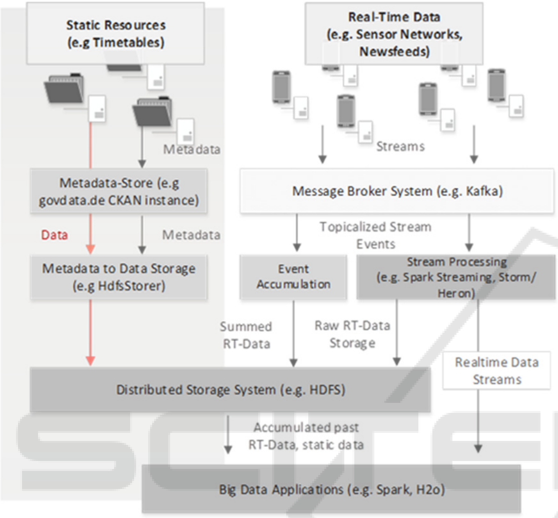
Static Resources
(e.g Timetables)
Real-Time Data
(e.g. Sensor Networks, Newsfeeds)
Metadata
Streams
Metadata-Store (e.g govdata.de CKAN instance)
Message Broker System (e.g. Kafka)
Data
Metadata
Topicalized Stream Events
Metadata to Data Storage (e.g HdfsStorer)
Event Accumulation
Stream Processing (e.g. Spark Streaming, Storm/ Heron)
Summed RT-Data
Raw RT-Data Storage
Realtime Data Streams
Distributed Storage System (e.g. HDFS)
Accumulated past RT-Data, static data
Big Data Applications (e.g. Spark, H2o)

Figure 6: Distributed Integration of Static and Real-Time Data from different Sources.

A full implementation of a similar procedure can be found e.g. on the H2O.ai github-page (H2O.ai, 2016). Therein Spark is used in combination with the H2O extension (=Sparkling water) to create a flight arrival delay prediction system based on past data and current weather information. In contrast to our scenario, the used dataset is already believed to be existent on the HDFS right from the beginning, e.g. by prior manual transfer. We extend such legacy solutions by a more convenient way for data import to a distributed filesystem, making use of a widely applied data cataloging system (i.e. CKAN) in the Smart City context.

8 DISCUSSION

This work described a possible realization of data import on the basis of the CKAN metadata catalog to the HDFS. The resulting CKAN plugin suits well the requirements of many use cases that can be encountered in a Smart City or during research. According to the type of application, different ways of realization are suitable for data import. Furthermore, the structure of the implicated systems has been described in detail and an overview about current research efforts in the field of ICT for Smart Cities and the state-of-the-art in the field of distributed processing has been provided, along with some intuition about possible future developments and improvements therein. The choice of the HDFS as target has been majorly motivated by the fact that a multitude of popular processing systems can make use of it. In the following, some prospects are discussed, which are enabled through the HDFS import of data, facilitated by the developed plugin. Furthermore, the results of the current work are briefly summarized and related to the possible research directions.

CKAN is currently one of the two most employed metadata storage systems. A multitude of Smart City initiatives embrace the idea of creating a unified publically accessible portal, where mostly various governmental entities, but also other stakeholders, are publishing their data. This is done with the hope that once the data is available, private citizens and companies will use it, in order to realize their own ideas and make use of that portal, where mostly various governmental entities, but also other stakeholders, are publishing their data. New business models are created and the public benefits emerge through better services based on the published data.

For certain applications it is advantageous to process the vast amounts of provided data in a distributed fashion. For this purpose, the user would normally have to accesses the web portal and search for each dataset. In order to make use of the resources provided therein, he or she has to download them separately from the referenced websites. After this, those files have to be uploaded in turn to the distributed file system in order to enable their processing. This way of accessing big data sets has some major drawbacks: It is likely that the resource size exceeds local file system storage capabilities (and thus resource transfer wouldn't work at all) and it also takes quite some time going through the whole process manually.

The current (CKAN-)*HdfsStorer* plugin was developed from the perspective of a CKAN instance operator. Making use of the CKAN plugin interface structure for data import has the following advantages: There is no need for writing a separate client for handling requests to the CKAN API and response parsing. The CKAN harvesters possess rich extensions for harvesting other non-CKAN sources (e.g. other metadata catalogs or specialized formats) and the data described in there can be easily imported without the need for further solutions.

Additionally, operators of already existing CKAN instances are provided with a convenient way of going over from metadata-only storage to having access to the full set of data, accumulated from a multitude of different sources. This could be one simple way for the creation of a Smart City Data Hub on which centralized data processing could then take place. Coupling the data import with CKAN has the further advantage of ensuring the timeliness of the data. A more elaborate data upload logic could also contribute towards streamlining this process.

Aside of the usage in the context of a Smart City, also other areas could benefit from the combined power of CKAN and Hadoop: The field of computational linguistics uses big corpora of text for language research and as the basis for speech production and translation engines (Wuebker, Ney and Zens, 2012). The creation of big corpora consumes a lot of time and poses a great challenge especially for smaller research groups as they usually do not have access to sufficient sources. Therefore, a few corpora are re-used repeatedly. This is a useful for the comparison of different solutions to a specific problem, as the number of confounding variables is decreased, but has the drawback that only a small and possibly not representative subset of a language is looked at. More resources and corpora would mean a better abstraction of the results and avoidance of possible biases. Ad-hoc creation of new specialized corpora could give rise to insights into the characteristics of situative (e.g. newspaper articles, law texts, search queries, chat logs or email correspondence) and group-specific (e.g. youth, elderly, the scientific community, cross cultural communications) language usage. CKAN instances provide an entry point for finding a high number of diverse resources and the metadata allows for easy identification of textual material and its classification according to situation and types/classes/groups of data. By means of harvesting a specific set of CKAN instances, and possibly also other user defined sources, the desired type of corpus can be accumulated on the HDFS in a time efficient manner by even a small group of people. Within the HDFS this corpus can then be readily processed by means of the distributed nature of Hadoop. According to the specific application, these non-annotated corpora can either be used directly or further pre-processed. The example of machine translation making use of aligned corpora has already been mentioned previously. The alignment of corpora can also be done on the HDFS grace to the availability of governmental documents in different languages, as there are often multiple official languages in a single administrative union (Steinberger et al., 2014).

The above described prospects outline possible further developments on top of the CKAN-*HdfsStorer* plugin. Some of these directions will be pursued in the course of emerging and running national and international projects regarding the topic of Urban Data Platforms.

9 SUMMARY & CONCLUSIONS

The current paper presented our recent work on the integration of metadata harvesting and data importing within Smart Cities. The concepts are exemplified based on two widely used systems – CKAN for metadata aspects and HDFS/Hadoop for enabling the distributed processing of Big/Open Data. The paper presents an architecture for such a component integrating the harvesting processes of CKAN and the storage of data to HDFS for further usage by different processing engines.

The architecture and concepts were prototyped and various evaluations were conducted which illustrated the benefits of the proposed solutions. A special section discusses the various application areas of our component and thus points to potential future developments and applications (e.g. Linguistics, Public Transportation) within urban environments.

REFERENCES

Ahuja, S. P. and Moore, B. 2013. State of big data analysis in the cloud. Network and Communication Technologies, 2(1), 62.

Alinat, P. and Pierrel, J. 1993. Esprit II project 5516 Roars: robust analytic speech recognition system.

Amazon.com. Amazon Simple Storage Service. Available at: https://aws.amazon.com/de/s3/ [Accessed on 20 Feburary 2016].

Bristol City Council. 2015. Bristol Open Data Portal. Available at: https://www.bristol.gov.uk/data-protection-foi/open-data [Accessed on 20 Feburary 2016].

CKAN Association. CKAN Overview. 2015. Available at: http://ckan.org [Accessed on 20 Feburary 2016].

CKAN Association. DataStore Extension. 2017. Available at: http://docs.ckan.org/en/latest/maintaining/datastore.html [Accessed on 20 Feburary 2016].

Fan, H., Ramaraju, A., McKenzie, M., Golab, W., & Wong, B. 2015. Understanding the causes of consistency anomalies in Apache Cassandra. Proceedings of the VLDB Endowment, 8(7), 810-813.

H2O.ai. AirlinesWithWeatherDemo. 2016. Available at: https://github.com/h2oai/sparkling-water/tree/master/examples/ [Accessed on 20 Feburary 2016].

Khan, Z., Anjum, A., Soomro, K., and Tahir, M. A. 2015. Towards cloud based big data analytics for smart future cities. Journal of Cloud Computing, 4(1), 1.

Kreps, J., Narkhede, N. and Rao, J. 2011, June. Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB (pp. 1-7).

Lapi, E., Tcholtchev, N., Bassbouss, L., Marienfeld, F. and Schieferdecker, I. 2012, July. Identification and utilization of components for a linked open data platform. In *Computer Software and Applications Conference Workshops (COMPSACW)*, 2012 IEEE 36th Annual (pp. 112-115). IEEE.

Liu, Z., Li, H. and Miao, G. 2010, August. MapReduce-based backpropagation neural network over large scale mobile data. In *2010 Sixth International Conference on Natural Computation (Vol. 4, pp. 1726-1730)*. IEEE.

Marienfeld, F., Schieferdecker, I., Lapi, E., and Tcholtchev, N. 2013, August. Metadata aggregation at GovData. de: an experience report. In *Proceedings of the 9th International Symposium on Open Collaboration* (p. 21). ACM.

Matheus, R. and Manuella, M. 2014. Case study: open government data in Rio de Janeiro City. Open Research Network.

Mercader, A. et al. 2012. ckanext-harvest - remote harvesting extension. Available at: https://github.com/ckan/ckanext-harvest [Accessed on 20 Feburary 2016].

Momjian, B. 2001. PostgreSQL: introduction and concepts (Vol. 192). New York: Addison-Wesley.

Red Hat, Inc. Using Hadoop with CephFS.. 2014. Available at: http://docs.ceph.com/docs/jewel/cephfs/hadoop [Accessed on 20 Feburary 2016].

Rosado, T. and Bernardino, J., 2014, July. An overview of openstack architecture. In *Proceedings of the 18th International Database Engineering & Applications Symposium* (pp. 366-367). ACM.

Shvachko, K., Kuang, H., Radia, S. and Chansler, R. 2010, May. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)* (pp. 1-10). IEEE.

Steinberger, R., Ebrahim, M., Poulis, A., Carrasco-Benitez, M., Schlüter, P., Przybyszewski, M., & Gilbro, S. 2014. An overview of the European Union’s highly multilingual parallel corpora. Language Resources and Evaluation, 48(4), 679-707.

Thaha, A.F., Singh, M., Amin, A.H., Ahmad, N.M. and Kannan, S., 2014, December. Hadoop in openstack: Data-location-aware cluster provisioning. In *Information and Communication Technologies (WICT), 2014 Fourth World Congress on* (pp. 296-301). IEEE.

The Apache Software Foundation. WebHDFS REST API. 2013. Available at: http://hadoop.apache.org/docs/r1.0.4/webhdfs.html [Accessed on 20 Feburary 2016].

The Apache Software Foundation. Apache Spark: Lightning-fast cluster computing. 2016. Available at: http://spark.apache.org/ [Accessed on 20 Feburary 2016].

The Apache Software Foundation. Hadoop Project Webpage. 2017a. Available at: http://hadoop.apache.org/ [Accessed on 20 Feburary 2016].

The Apache Software Foundation. Apache Cassandra. 2017b. Available at: http://cassandra.apache.org/ [Accessed on 20 Feburary 2016].

Tierney, B., Kissel, E., Swany, M. and Pouyoul, E. 2012. Efficient data transfer protocols for big data. In *E-Science (e-Science), 2012 IEEE 8th International Conference on* (pp. 1-9). IEEE.

Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. and Maltzahn, C. 2006, November. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation* (pp. 307-320). USENIX Association.

Winn, J. 2013. Research Data Management using CKAN: A Datastore, Data Repository and Data Catalogue. IASSIST Conference.

Wuebker, J, Ney, H and Zens, R. 2012. Fast and scalable decoding with language model look-ahead for phrase-based statistical machine translation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*.