A Concept of Generic Workspace for Big Data Processing in Humanities

Jedrzej Rybicki Benedikt von St. Vieth Daniel Mallmann
Forschungszentrum Juelich GmbH
JSC
Juelich, Germany
Email: {j.rybicki, b.von.st.vieth, d.mallmann}@fz-juelich.de

***Abstract*—Big Data challenges often require application of new data processing paradigms (like MapReduce), and corresponding software solutions (e. g. Hadoop). This trend causes a pressure on both cyber-infrastructure providers (to quickly integrate new services) and infrastructure users (to quickly learn to use new tools). In this paper we present the concept of DARIAH Generic Workspace for Big Data Processing in eHumanities which alleviates the aforementioned problems. It establishes a common integration layer, thus enables a quick integration of new services, and by providing unified interfaces, allows the users to start using new tools without learning their internal details. We describe the overall architecture and implementation details of the working prototype. The presented concept is generic enough to be applied in other emerging cyber-infrastructures for humanities.**

***Keywords*-cyber-infrastructures for humanities; architecture; Big Data;**

I. Introduction

The European Commission supports emerging research infrastructures for a wide range of disciplines by funding the European Strategy Forum on Research Infrastructures (ESFRI). This body supports, among others, the digital humanities project DARIAH (Digital Research Infrastructures for the Arts and Humanities) [1]. DARIAH embraces a number of national initiatives also a Germany-based effort: DARIAH-DE, where the described work was conducted. It is safe to say that all those bodies support a vision of an infrastructure helping researchers to cope with the challenges that the digital humanities bring. One of the broadly acknowledged challenges is the one of Big Data i. e., data that is so large and complex that it exceeds the limits of the commonly used hardware and software systems. Big Data is sometimes characterized by its volume, velocity, and variety [2]. Clearly in case of humanities in DARIAH the variety of the data is given as it stems from various disciplines (linguistics, musicology, history, and many others). By combining the data sets from this different disciplines under the umbrella of a common cyber-infrastructure we observe a constant increase of the data volume (connected "small data pieces" become Big Data). To this end tools supporting efficient handling of large data sets must be established.

DARIAH has built a data Grid solution to provide the researchers with a place to store their data in a safe manner. It was primarily designed for bit preservation and the current solution is based on iRODS [3] with an HTTP-based interface towards the end-user [4]. Data storing and preservation is an important service, but not sufficient for research work in times of Big Data. Imagine a researcher who uploads a collection of documents to the storage service to assure their safety and wants to perform a simple quantitative analysis on the data set e. g. calculate word frequency or search for documents containing given terms. iRODS does not provide an efficient way of processing the data. In particular there is no easy way to execute parallel processing tasks, which become more and more important in the digital humanities, for instance for text processing [5]. Even such a simple quantitative analysis requires (beside some advanced programming skills) the sequential processing of each single object from a collection, resulting in a higher server load and long response times. On the other hand, there are emerging paradigms for efficient parallel processing of unstructured data (e. g. MapReduce [6] implemented in the Hadoop framework [7]) but they are not so strong in terms of data preservation. In this paper we describe the concept of a Generic Workspace for Big Data processing, which can be understood as an *active storage* that not only stores the data but also provides a means to efficiently process them.

A high-level goal of DARIAH is to provide generic easy-accessible services. This holds for both, storage and data processing. Thus our motivation is twofold: the users don't want to learn new tools and change their workflows, but rather prefer to stick to well-know principles and abstractions. The service providers, on the other hand, seek the ways of reducing the complexity and heterogeneity of their infrastructure. It is not feasible (from both user's and service provider's perspective) to add new independent, stand-alone components in the common infrastructure each time a new functionality is required. Having separate components, the users wishing to process the data would have to download them first from the common storage and then upload to a processing component (like Hadoop). Such manual operations are prohibitively complicated, time consuming, error-prone and inherently not scalable. The Generic Workspace hides the details of data movement, by establishing a common service layer for both storing and processing of the data and a uniform interface to interact with underlying resources.

To help the users to get to grips with the new services we decided to hide the underlying components from them and expose only their functionality through a well-know abstraction of a (distributed) file system. The Workspace offers the view of a hierarchical file system which contains two classes of files. Some of the files are ordinary files storing content, others we call *special files* model computation resources. Their content are results of processing. The users wishing to start processing have to simply create a special file. To get the results of such processing, they retrieve the content of the file. The corner stone of this approach is to use meaningful, declarative file names. For instance the creation of a special file named `wordCount` should trigger the computation of a word count of all documents in a directory where the file was created and the results should be written back into the file. In this *declarative* approach, users only define the result they are expecting to get from the system (like number of words) and not how the result should be derived. The mixture of the well-know abstraction of a file system with the declarative, meaningful file names allows the end-user to quickly integrate the new functionality into his workflow without even knowing how the actual processing is done and without learning the details of the new tool.

DARIAH infrastructure must serve a very heterogeneous user base. Not only in terms of disciplines they represent but also in terms of how they interact with the infrastructure. Some of them already use some kind of Virtual Research Environments (VRE) [8] but others do not. Thus, in our design, we aimed at bringing the processing close to the storage rather than to rely on a high-level VRE-like solution. By following this approach we can offer generic processing to both groups of users (accessing the generic infrastructure via an VRE or directly). To some extend this is an implementation of a more general principle of uniting the human- and computer-readable services in one common interface.

The rest of this paper is structured as follows. In Section II we present the high-level view of the proposed solution and define the main goals and functionalities for the system. In the next Section, we describe the concrete implementation of the Workspace. For the prototype two well-know products for storing and processing (iRODS and Hadoop) were selected. We discuss their strengths and weaknesses and explain how they can be combined to constitute the basis for a generic Workspace for Big Data Processing in Humanities. Section IV summarizes the results and put them in a broader perspective by referring to existing related work. We conclude our paper with a summary in Section V.

II. Architecture

As already explained the two main driving forces of our architecture were the urgent need to reduce the integration effort of adding new services into an existing cyber-infrastructure and the user-friendliness allowing for instant use of the new functionalities. In this section we explain the main architectural choices we made and how they correspond to the driving forces.

The user-friendliness is addressed mainly by a declarative approach and ReST-like architectural style. In short the declarative approach boils down to performing the actions by describing their results rather than by describing how they are to be implemented and conducted (as in case of an imperative approach). The well-known examples of declarative approaches are HTML (describing how the website should looks like and not how the rendering shall be done) or SQL (to create and manipulate the relations without describing how they are stored and processed in the RDBMS).

The natural remaining question is where and how the declarative definitions should be put? The answer is provided by the next architectural decision: to exploit a well-known abstraction of a file system. The idea bears some similarities with the ReST architectural style [9]. In short ReST proposes to use set of HTTP operations (like GET, POST, PUT) with a well-defined semantics [10] to manipulate resources identified by the URLs and interconnected by the hypermedia. In our case we use resources modeled as files and directories in a file system and users can manipulate those naturally by issuing commands like `open, close, read`. The abstraction together with a set of methods define a *uniform interface*. The definitions of the expected computation results should also be written in the common file system by using methods listed above. The rationale of using file system abstraction was that there was already an iRODS based distributed storage in DARIAH and it offered a file-system-like interface.

The architecture of our system is depicted on Figure 1. The users on the left hand side of the picture face a common DARIAH namespace and have tools to access and manipulate objects in it. Objects are identified by logical names. The integration layer provides the namespace and thus hides the actual physical resources on which the objects reside. It also distinguishes between the different classes of objects in the namespace: ordinary and special files. In case of an access to an ordinary file the logical name is translated to a physical path on the respective storage resource and depending on the command issued by the user either the content of a file is served back or modified. Some of the logical names, however, are mapped not on ordinary storage resources but on processing clusters. Access requests or modifications on such objects are intercepted by the integration layer and passed to the processing services (like Hadoop). The contract between the integration layer and the processing services states that the objects should have meaningful names following the declarative approach. The names are passed over to the processing services and interpreted by them. The name can also include some parameters required by the service to conduct the particular computation. The parameters are placed in brackets at the end of the file name

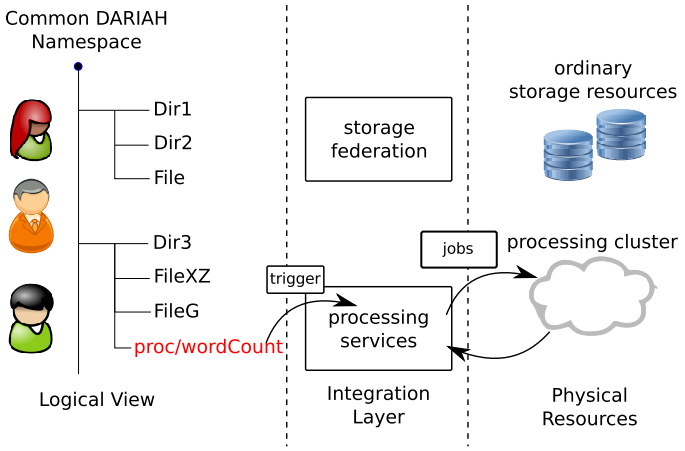

Figure 1. Architecture overview

(similarly to the convention known from some programing languages). The results are put into the special files and can be read from them. To make the differentiation between normal files and computation objects easier for both users and the developers of the integration layer it was agreed that the special files should follow a simple naming convention e g.: they should contain a common segment in their logical name: `*/proc/*`. Examples of the meaningful, declarative names within the convention are as following:

- `/home/user/proc/wordCount`
- `/path/proc/contains(foo)`
- `/coll1/proc/writtenBy(Goethe,Rilke)`

A nice side-effect of making the results of processing available as objects with logical names in a common namespace is that it is easy to re-use and share such results. Existence of a special file `proc/wordCount` in a common collection indicates that the word count has been already calculated for this collection and can be reused.

With the declarative approach we aim at providing initial set of usable functions for data processing but it should also be possible for advanced users to define and share their processing functions. In other words it should be possible to add new classes of special files.

Also the integration of the new services is made much easier by establishing a common integration layer. Typically in a distributed environment the integration of a new service means that it has to be "connected to" other existing services. Only after the integration steps are completed it is possible for the user to seamlessly use the infrastructure. To make such connections easier, developers usually define Application Programming Interfaces (API) but still some glue code must be added to the new service to handle the APIs of the remote services. The existing infrastructure components must be extended as well to become aware of the newly added service, and this usually requires a connection to the new API. Therefore the integration effort in an infrastructure comprised of n components can be approximated as $O(n^2)$ (connecting all pairs of components). Such an additional effort might lead to a situation in which new services are reluctantly added (to avoid the expensive changes) and the whole infrastructure becomes inflexible or even obsolete. The Workspace helps in keeping the infrastructure flexible and ready for the new challenges (like Big Data). This goal is achieved by the substantial reduction in the integration effort: In the Workspace model, the new service needs only be integrated into the integration layer. In other words it has to become responsible for a part of the common namespace and can be unaware of the existence of other services. There is also no need for the new service to offer any user interface as it is integrated into common namespace and the end-users already have a means to manipulate its content. The communication between end-users and new services is conducted via common namespace, the same hold for the communication between the services. Thus the namespace can be also viewed as a substitute of a common API.

III. Implementation

Previous section presented the rationale behind the architecture of the Workspace and justified its most important design principles. In this section we will present the experiences gathered while implementing a working prototype of the system. It was decided early on that DARIAH will use iRODS for bit preservation. As a processing candidate for our prototype we decided to use Hadoop due to its popularity, proliferation and broad acceptance in the world of Big Data. The experience gathered from building a prototype shows that the architecture is generally applicable and can be used to integrate other products as well.

We will first present the main features of iRODS and discuss its inapplicability for Big Data processing. Subsequently we elaborate on a processing paradigm which is quite well-established in the Big Data processing environment: MapReduce. We will focus on an existing open

source framework implementing the paradigm: Hadoop. Due to space limitations we only present main features of both products, for more details the readers are referred to [3], [7]. Finally we will show how both example technologies were integrated to provide a working prototype of a DARIAH Workspace.

A. About iRODS

iRODS is a data management middleware developed by Data Intensive Environments (DICE) group. iRODS provides data management functions required in a distributed environment such as a file transfer service, data replication and metadata management [11]. Files in iRODS are organized into hierarchical collections (in a way similar to file systems), offering a logical namespace for files stored on different (distributed) storage resources. Currently many different types of storage resources are supported: UNIX and Windows file systems, but also secondary storage solutions like HPSS. It is also possible to integrate new types of storages (by writing drivers). The typical deployments of iRODS comprise of one or multiple servers maintaining storage resources and exactly one metadata server (called *iCAT*). The later component is responsible for logical-to-physical mapping for stored data objects and user access management. To improve fault tolerance iRODS can be configured to replicate the data objects across multiple storage resources.

A unique feature of iRODS is its ability to use user-defined rules for implementing data management policies. They follow event-condition-action model. iRODS defines a list of events (like file ingestion). The data administrator can use those to define actions (like file format verification) executed under certain conditions (e. g. ingested file has given extension). Each event triggers iRODS *rule engine* to execute corresponding rules. In this model it is possible to implement even complex data management policies. It is worth mentioning here that the term *policy* is a broad one and it is usually understood as a "statement of intent". User-defined rules define ways of achieving the intended effects. Syntactically, the rules are composed of calls to *microservices* and other rules. Microservice is defined as the smallest unit of work in iRODS that embodies single, autonomous operation. iRODS software package brings a rich set of microservices that can be reused to compose own rules. It is also possible to implement own microservices but this require C programming skills and recompilation of the server to make the new functionalities available. An example of simple iRODS rule that makes a copy of a given directory (`/zone/monitored/`) to pre-defined target location (`/zone2/safe-copy`) each time a new file is created and respective iRODS action is triggered (`acPostProcForPut`) is shown on Listing 1.

```
acPostProcForPut{
  ON($objPath like "/zone/monitored/*") {
    msiSplitPath($objPath,*collection,*fileName);
    msiCollRsync(*collection,"/zone2/safe-copy
        ","","IRODS_TO_IRODS",*status);
    msiWriteRodsLog("Backup of *collection done (
        status=*status)");
  }
}
```

Listing 1. Example policy

Beside the "global rules" triggered by the system-wide events there are also user-defined rules. Such rules are executed manually (via command-line tools) and not each time a given combination of action-condition occurs. The user-defined rules use the same syntax as the global rules and can be used to implement data processing workflows. Two obvious and severe limitations of user-defined rules when it comes to analyzing larger amounts of data are following. First of all iRODS does not provide support for parallel execution of rules and microservices. Thus to analyze the content of a file, a microservice has to read it sequentially and the performance is upper bounded by the disk throughput. Secondly iRODS does not provide a convenient way of creating the microservices. The source code must be upload to the server and the server must be recompiled. This limitations renders rules almost unusable for researchers to do processing.

B. About Hadoop

Hadoop is an open-source framework that facilitates scalable, distributed processing of large data sets. It leverages clusters of computers to store and process the data in a robust fashion. The robustness is not achieved by the high-end hardware but rather by the redundancy and effective handling of failures. Two main component of Hadoop are distributed file system (typically HDFS) and a framework for job scheduling and resource management.

Large data sets stored in a distributed fashion on HDFS can be efficiently analyzed using simple programming models. Most prominent among them is the MapReduce. To understand the strengths (and weaknesses) of Hadoop it is worth discussing the computation with MapReduce in a more detailed fashion. The first prerequisite for efficient processing is the availability of the data in the HDFS. HDFS stores files as a sequence of blocks of the same size (except the last one). The blocks are distributed across the resource servers (*storage nodes*) within a cluster. To improve the fault tolerance, blocks are replicated. The replication factor for each file can be defined either upon the file ingest or globally. The distribution of the file segments across the storage nodes in cluster enables quicker access to files' content. The operation is not bounded by the throughput of a single resource since the reading can be done simultaneously on different storage nodes.

To process the data stored in the HDFS MapReduce jobs can be used. In the first step *map* function transforms job input into key-value pairs. Map divides the input data into

smaller portions of work and act on them. Afterwards the framework sorts the intermediate results by the key and passes them to the *reduce* function which produces the final output (again in form of key-value pairs). Reduce combines the many results from the map step into a single output, hence the name. Since the input data and usually the intermediate results are handled as files and stored in a distributed fashion in the HDFS it is possible to split map and reduce functions into smaller *tasks* and execute them in parallel on storage nodes where the data resides. The data locality improves the performance and reduces the network load by avoiding data movement.

Word counting could be implemented in MapReduce as following. In the map step the input text would be tokenized and pairs of word and value of 1 would be emitted. The framework would sort the map results by the key (i. e. by the word) and pass all the key-value pairs in form of $< word, 1 >$ to the reduce step. It is guaranteed that all the pairs with the same key value would be processed by the same reducer task. For different keys (words), however, different reducing tasks can be used. Reducing function can simply iterate through the input map and sum the `1`'s, emitting current word and the sum. Finally the outputs of single reduce tasks will be merged and made available to the user.

The Hadoop framework monitors storage nodes and processing tasks and takes care of the failures seamlessly e. g. by re-spawning failed tasks so that the end-user can use service despite the failures. Also the data integrity is periodically checked and the detected errors (with help of checksums) are corrected by overwriting from replicas. Hadoop and HDFS, however, do not provide convenient tools for defining data management policies and thus are less suited for long time archiving of the data.

C. About Pig

Since writing native MapReduce jobs require programming in Java against Hadoop API, there is a number of projects aiming at lowering this hurdle. One example is Apache Pig [12], [13] providing a high-level, declarative language for expressing data analysis programs. Pig offers an interactive shell and a possibility to execute pre-defined scripts. Programs written in the high-level Pig language are translated into MapReduce jobs and submitted to the cluster. Example of a simple Pig script to calculate word count is presented on Listing 2. The readers can notice the similarity between Pig and SQL.

```
data = LOAD 'path/file.txt' USING TextLoader();
token = FOREACH data GENERATE FLATTEN(TOKENIZE($0)
    ) AS word;
words = FILTER token BY word MATCHES '\\w+';
gr = GROUP words BY word;
c = FOREACH gr GENERATE COUNT(words) AS cnt, gr;
res = ORDER c BY cnt;
STORE res INTO 'path/output.dat';
```

Listing 2. Example of a Pig script

D. Integration

Concrete implementation of the Workspace is depicted on Figure 2. In our experiment we used iRODS in version 3.2, Hadoop in 1.0.4 and Pig in 0.10. The prototype uses iRODS to establish common integration layer exposing file system like interface to the user and abstracting both storage and processing resources.

The first step of adding efficient processing functionality into iRODS was to integrate the HDFS as a storage resource. The rationale behind this step was simple: efficient MapReduce processing is only possible when the data are distributed. As already explained iRODS supports different types of physical storages, unfortunately there is no support for HDFS. Thus it was necessary to write a new storage driver. We decided to implement the driver by using Universal Mass Storage System driver interface offered by iRODS. This interface is usually used to access tape systems and other storage technologies with sequential access to files. Since iRODS offers random access to files, the resources offering sequential access can only be used as *compound resources* together with so called disk cache. Before any operation on the content of a file is performed it is staged to the cache (method `stageToCache`), after the operation the file is moved back to archive (`syncToArch`) to save storage space. Such a movement can be handled transparently by iRODS, with help of rules using microservice `msiSysReplDataObj`. The actual driver implementation was quite simple. We only had to write a limited set of methods with self-explanatory names like `syncToArch`, `stageToCache`, `mkdir`, `chmod`, `rm`, `mv`, `stat`. Due to the space limitation we only present how `syncToArch` method, that moves the data from cache to HDFS, can be implemented (remaining methods are trivial as well). We decided to use the standard HDFS client tool provided by Hadoop. It could be also possible to use C library and write a native iRODS HDFS driver.

```
syncToArch () {
 echo "syncToArch $1 $2" >> /tmp/univDriver.log
 hadoop fs -copyFromLocal $1 $2
 return
}
```

Listing 3. Implementation of `syncToArch` method

After we made the data available on the distributed storage, the next step was to enable processing. Let us first discuss a simple use case and then extend it further. As already explained the processing in the Workspace should be triggered by creation of a special file. For instance the creation of a file called `/my/collection/proc/wordCount` would trigger the process of counting words across all documents in `/my/collection`. We already have shown iRODS's feature that detects a creation of files. Similarly to the policy from Listing 1 we use the pre-defined iRODS

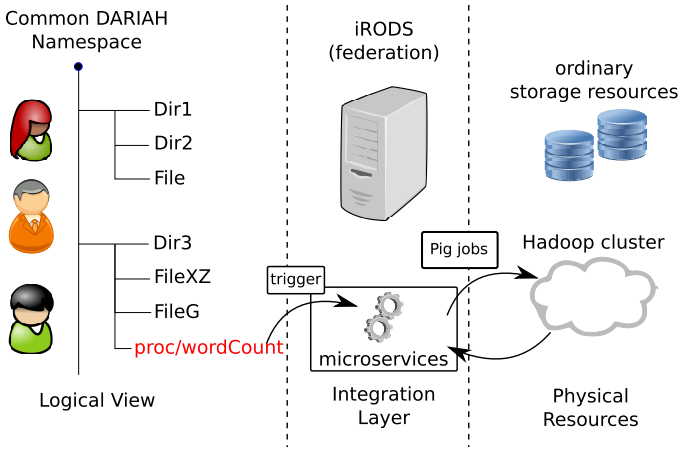

Figure 2. Concrete implementation

action `acPostProcForPut` and define additional condition:

```
acPostProcForPut {
 ON($objPath like "*/proc/*" ) {
    ...
 }
}
```

Listing 4. Rule to trigger processing (details omitted)

Listing 2 shows a Pig script that can be used for the word calculation. The script, however, has a predefined input and output (`/path/file.txt` and `/path/output.dat` respectively). This had to be changed. Pig scripts can take external parameters, they are marked by `$` in the script. We agreed on a convention to call the input of a Pig script `$input` and output `$output`. The input of the script can be extracted from the name of the special file, the output should be directed to a temporary directory first and then moved to overwrite the special file after the computation is completed. Now both parts have to be put together and the creation of the special file should trigger the execution of the Pig script. iRODS provides a microservice that can be convenient here: `msiExecCmd` can be used to call batch scripts stored on the iRODS server. It was quite simple to write a short batch script (called *executing script*) that takes the logical and physical name of the special file as parameters, start a Pig *processing script* and move its results back to overwrite the special file. Surprisingly the most challenging task was the last one. iRODS stores the file size in the iCAT database. Since the process of overwriting changes the size of the special file, the iCAT content has to be updated. This was done by issuing direct SQL query to the iCAT database. Listing 5 shows how the executing script runs the processing script.

```
/usr/bin/pig -p input="$input" -p output="$output"
     -l /var/pig.log $PIG_SCRIPTS/$scriptName.pig
```

Listing 5. Fragment of the executing script

Let us now extend the simple example presented above to show more advanced features of the Workspace. The Pig script for word counting produces output in form presented on Listing 6, where the first column contains the number of occurrences of the given word from the second column.

```
1775    the
1040    of
730     in
677     and
457     to
343     was
334     a
331     und
248     die
223     he
...
```

Listing 6. Processing results

For the sake of discussion let us assume that the researcher is only interested which words occurred more often than 200 times and less than 300 times. It is a simple filtering job that should analyze the first column of the file. First challenge to tackle is the need to provide more parameters to a Pig script than just input and output. In Section II we mentioned that the parameters should be part of the logical name of the special file. This is certainly true for the input and output parameters that can be derived from the full path of the file. But it is also possible to include more parameters in the full name. Particularly for the filtering job it would be reasonable to encapsulate the filtering criteria in the file name like this `/path/proc/filter(coll1>200 AND coll1<300)`. Within this convention it is quite simple to write a Pig script for the required analysis that allow for flexibility in terms of filtering (e. g. change of the filtering boundaries). The script is presented on Listing 7.

```
A = LOAD '$input' USING PigStorage('\t')  AS (
    coll1,coll2);
B = FILTER A BY ($params);
STORE B INTO '$output';
```

Listing 7. Pig script for filtering

The executing script that is used for starting the job must be extend as well to pass not only the input and output parameters but also the filtering conditions to Pig. It must also determine which script to run (either `wordCount` or `filter`). This is done based on the name of the special file.

We sought after an extensible solution for the Workspace and it is to expect that sooner or later the researchers' needs would not be satisfied by the two scripts presented so far and they will look for ways of defining own processing scripts. This functionality is easily enabled by storing the Pig scripts in ordinary iRODS collections. The more advanced users could then upload their Pig scripts to a iRODS directory. The scripts must only follow the convention of flexible input and output paths and parameter passing (if required). The whole processing workflow does not change much. First a creation of a special file with name in form `collection/proc/function(parameters)` is detected by iRODS. Subsequently this logical name is passed to the executing script that extracts the function name (which is mapped on the Pig script name) and job parameters. The appropriate Pig script is fetched from the iRODS collection and executed with extracted parameters. In the last step the result is written back to the special file. The final solution is extensible and future-proof, it allows to easy extend the set of processing function, run them on new data by all the users of Workspace and share the results of the computation.

IV. Related Work

In the previous sections we presented the idea, architecture, and a prototype of the Generic Workspace for Big Data Processing in Humanities. The Workspace can be seen as an *active storage* i. e., a storage that not only stores the data but also provides a means for efficient processing of the data. Let us compare the presented approach to some previous work in the field.

Our concept is clearly related to the idea of Virtual Research Environments. An example of VRE is TextGrid [14] (although it is much more on the organizational level than just a tool). TextGrid comprises two main components a web front-end and a repository (TextGridRepo) that abstracts underlying storage resources and services. Usually the VREs provide high-level, complex services (like XML editing or text comparison tools). The Workspace does not fit into this category. It can be seen as a service of underlying middelware that is exposed as a service to the user to enable efficient processing before the ingest of the data for preservation in the TextGridRepo or further processing with high-level tools. There are also plans for migrating the TextGrid repository infrastructure to Fedora Commons [15] and iRODS, that would allow to even reuse some parts of our prototype.

The digital humanities workbench [16] provides a wide collection of useful text processing tools as on-line services. Making the existing tools available without the need of installing them locally is a central concept of all kinds of VREs. The authors of [8] presented a general-purpose Virtual Research Environments that facilitates the integration of information resources and tools for supporting research activities. We believe that our Workspace could easily be integrated into VREs as a tool that provides efficient processing of large amounts of data. Most of the VREs foresee the possibility to extend the installations by adding new storage resources. Since the Workspace provides a file system interface it can be integrated into a VRE.

There are already examples of the application of MapReduce in digital humanities [17]. The motivation of the authors were the performance problems of the tool called Text Analysis Portal for Research (TAPoR) which is a web-based application that provides a suite of text-analysis tools to scholar and researcher in the digital humanities. The authors identified that the problem was caused by the Ruby-based backend services that performed the actual processing. They decided to migrate the services to Hadoop and achieved substantial performance improvements. This work shows that the Hadoop-based services can be successfully used to solve problems in digital humanities. The main architectural difference between this work and our solution is the application of the declarative approach in the Workspaces. We have shown that our approach has serious implications with regard to the user-friendliness and the costs of integration of new services.

Despite its high popularity and broad application, it becomes clear that MapReduce is not always the right tool and has its limitations [18], [19]. Our work was motivated by the experience of cyber-infrastructure providers in a digital humanities project. The rationale behind the Workspace is to enable quick integration of new services into an emerging infrastructure. We used Hadoop MapReduce and Pig to build a working prototype but the same approach can be used to integrated alternative processing solutions. Especially while we use a declarative approach the user is not even aware how the actual computation is done. Thus changes or additions of new services are seamless to the end-users. Adding new services would be an interesting avenue for further research.

The work [20] discusses generic processing environments, the solution, however, strives for general applicability of the remote method execution pattern. The discussion focusses on the low level technologies, which is out of scope for our work.

Admittedly the user interface was not a primary concern of our work. Given the heterogeneous user-base of DARIAH it would be hard to come up with an interface that could be suitable and acceptable for all users. We decided to use the well-known abstraction of a file system as a primary low-level interface to the Workspace. The decision was motivated by the presence of such a file system offered by iRODS. The abstractions of the web and file system resources, however,

are quite close to each other, thus it would be easy to build an HTTP-based ReST interface above the file-system-based Workspace in the future. This might become a relevant topic given the DARIAH efforts to provide an HTTP-based user interface to storage resources [4] and common portal [21].

V. Conclusion

We presented a generic concept of Workspace for Big Data Processing in Humanities. Our work was inspired by the experiences collected during the establishing of a distributed, service-oriented cyber-infrastructure in DARIAH-DE. We proposed a solution that fulfills two main design goals: it is user-friendly and reduces the burden of integrating and providing new services. The working demonstrator of the concept helps users to process large amounts of data by employing efficient data processing based on Hadoop and Pig, two emerging data processing products, often applied to deal with the challenges of Big Data. The two products were used together with a well-established data Grid solution: iRODS. The paper presents a provider-centric view on enabling Big Data processing tools, which is the first step of their application in digital humanities. As long as the tools are not available there is little chance that they will be used to solve the actual research problems.

Acknowledgements

The work has been supported by DARIAH-DE, which is partially funded by the German Federal Ministry of Education and Research (BMBF), fund number 01UG1110A-M and by EUDAT, funded by the European Union under the Seventh Framework Program, contract number 283304.

References

[1] DARIAH: Digital Research Infrastructures for the Arts and Humanities. [Online]. Available: http://www.dariah.eu/

[2] D. Laney, "3D Data Management: Controlling Data Volume, Velocity and Variety," Gartner, Tech. Rep., 2001.

[3] A. Rajasekar, R. Moore, C.-Y. Hou, C. A. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert, P. Tooby, and B. Zhu, *iRODS Primer: Integrated Rule-Oriented Data System*, ser. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2010.

[4] D. Tonne, J. Rybicki, S. E. Funk, and P. Gietz, "Access to the DARIAH bit preservation service for humanities research data," in *PDP '13: 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2013, pp. 9–15.

[5] M. Sarwar, M. Alexander, J. Anderson, J. Green, and R. Sinnott, "Implementing MapReduce over language and literature data over the UK National Grid Service," in *ICET '11: 7th IEEE International Conference on Emerging Technologies*, 2011, pp. 1–6.

[6] D. Jeffrey and G. Sanjay, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[7] T. White, *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.

[8] T. Blanke, L. Candela, M. Hedges, M. Priddy, and F. Simeoni, "Deploying general-purpose virtual research environments for humanities research," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1925, pp. 3813–3828, 2010.

[9] R. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), Internet Engineering Task Force, Jun. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[11] B. Zhu, R. Marciano, R. Moore, L. Herr, and J. P. Schulze, "Digital repository: preservation environment and policy implementation," *International Journal on Digital Libraries*, vol. 12, no. 1, pp. 41–49, 2012.

[12] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *SIGMOD' 08: ACM International Conference on Management of Data*, 2008, pp. 1099–1110.

[13] Apache Pig Project. [Online]. Available: http://pig.apache.org/

[14] H. Neuroth, F. Lohmeier, and K. M. Smith, "TextGrid – virtual research environment for the humanities," *International Journal of Digital Curation*, vol. 6, no. 2, pp. 222–231, 2011.

[15] Fedora commons repository software. [Online]. Available: http://www.fedora-commons.org/

[16] A. Bia, "The digital humanities workbench," in *INTERACCION '12: 13th ACM International Conference on Interacción Persona-Ordenador*, 2012, p. 50.

[17] H. Vashishtha, M. Smit, and E. Stroulia, "Moving text analysis tools to the cloud," in *SERVICES-1: 6th IEEE World Congress on Services*, 2010, pp. 107–114.

[18] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *SIGMOD' 09: ACM International Conference on Management of Data*, 2009, pp. 165–178.

[19] A. Rowstron, D. Narayanan, A. Donnelly, G. O'Shea, and A. Douglas, "Nobody ever got fired for using Hadoop on a cluster," in *HotCDP '12: ACM 1st International Workshop on Hot Topics in Cloud Data Processing*, 2012, pp. 1–5.

[20] B. Dumant, F. Horn, F. D. Tran, and J.-B. Stefani, "Jonathan: an open distributed processing environment in Java," *Distributed Systems Engineering*, vol. 6, no. 1, p. 3, 1999.

[21] DARIAH Portal. [Online]. Available: https://portal-de.dariah.eu/