Report on the feasibility of parallelising preservation processes

Author
Martin Schenck (Technische Universität Berlin)

March 2013

This work was partially supported by the SCAPE Project. The SCAPE project is co-funded by the European Union under FP7 ICT-2009.4.1 (Grant Agreement number 270137).

Executive Summary

Preservation of digital media collections has become a scalability issue due to the large quantity of heterogeneous content. The amount of information has reached a point where distributed, shared nothing computing environments are necessary to enable processing in a timely manner. Hadoop is a system that abstracts from the hardware in a computing cluster and presents the user with a programming interface. The user can load data into the system and write programs that execute on the cluster, while not worrying about network transfers or data access. At the same time, governmental organizations like public libraries want to have an intuitive yet powerful graphical user interface to design preservation workflows that are supposed to run on Hadoop. Taverna offers such an interface as a domain-independent workflow management system. This deliverable discusses the feasibility of running preservation processes on Hadoop and the feasibility of automatically translating Taverna work flows into Hadoop programs. While it is certainly feasible, there is room for optimization.

Table of Contents

1 Introduction

The execution platform will provide a general means to facilitate the specification and evaluation of complex preservation operations over massive volumes of content. This work package is concerned with the design and implementation of a driver program, called PPL (for "Program for parallel Preservation Load"), for the optimized creation and execution of parallel algorithms for digital preservation actions.

Preservation in the SCAPE context is generally performed by public organisations such as national libraries. SCAPE aims to help enable these organisations to carry out preservation on a very large scale. In order to preserve huge volumes of digital material these libraries have acquired computing clusters as a platform for Hadoop[1]. Hadoop is a framework for distributed computing in shared-nothing environments running on commodity hardware. In order to run preservation tasks on these clusters, a number of steps must be taken:

1. Acquire and install the hardware
2. Install and configure Hadoop on the cluster
3. Transfer data onto the cluster
4. Develop preservation programs for Hadoop

The crucial step here is step 4: "Develop preservation programs for Hadoop". Usually, these programs are written in Java or another of the available higher level languages and they must use the MapReduce[2] paradigm. MapReduce consists of two processing main phases, i.e. map and reduce, for which the developer provides implementations. . Becoming accustomed to thinking in a MapReduce fashion is challenging, depending on the developer's programming experience and algorithmic thinking ability.

While it is not desirable to train the library employee's to writing programs in higher-level languages like Java or Pig[3] for MapReduce, it certainly is desirable to enable them to create parallelised workflows that execute efficiently on Hadoop. By abstracting to an even higher level, most of these underlying technologies are no longer visible to the user; instead users create workflows with an intuitive, easy to use, and powerful GUI – Taverna[4]. Teaching users how to use Taverna is a much easier task, than teaching them how to program Java.

One example of a common digital preservation scenario, *scannedconverting,* is converting scanned book pages to an open standard. The steps below decompose this scenario for execution on a Hadoop cluster.

1. Load images into Hadoop Distributed File System (HDFS)[5]
2. Perform automated image migration to the desired format

---

[1] http://hadoop.apache.org/

[2] http://research.google.com/archive/mapreduce.html

[3] http://pig.apache.org/, a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs

[4] http://www.taverna.org.uk/

[5] http://hadoop.apache.org/docs/r1.1.2/hdfs_design.html

3. Apply automated quality assurance processes to compare new images against the originals
4. Persist the process results within the institutions preservation system.

While steps 1 and 4 are also concerned with the transfer of files between storage locations, these steps are technically not part of the preservation processes of image conversion and quality assurance itself; steps 2 and 3 are however, and are distinct and consecutive steps. Quality assurance cannot be performed before the conversion has happened. These two steps (or an arbitrary number of steps for other scenarios) can be easily linked using the Taverna GUI. By enabling an automatic conversion from Taverna workflows to Hadoop programs, anyone who can create workflows can create distributed programs for scalable processing.

This work package is concerned with two major questions:

1. Is it feasible to run digital preservation processes on Hadoop?
2. How can we simplify the creation of workflows for Hadoop? The following sections provide initial answers to these questions. A naïve approach to converting and executing Taverna workflows on Hadoop is benchmarked and a possible optimization is presented.

2 The Translator (PPL)

This section presents the *Program for parallel Preservation Load* (PPL), its usage, and its functionality.

2.1 Overview

In order to enable scientists to easily scale Taverna workflows, the presented translator automatically generates a java class file. Essentially, the program takes a workflow as input and automatically generates a class that could be uploaded to and executed by Hadoop.

The resulting Hadoop execution is a linear list of MapReduce jobs produced from the arbitrarily complex input workflow. The required input for each individual job is either read locally (as provided by the Hadoop framework), or is fetched from other machines in the Hadoop distributed file system (HDFS), if required.

2.2 Usage

The translator can either be used as a compiled jar file, or can be built from source[6]. A command line interface is supplied; simply specify the location of the Taverna workflow file as a command line argument, and the translator will do the rest. Use -h or --help as program argument to get the help output.

2.3 The Translation Process

Figure 1 shows a simplified version of the translation process. PPL uses the SCUFL2 API to interpret workflow files created with Taverna. First the entire workflow is translated into a linear list of its activities. To achieve this, the translator starts at the workflow's output ports and follows the data links backwards and recursively. Each activity is checked to see whether it's already part of the linear list. If so, it is deleted from the linear list. After this check is complete the activity is appended to the end of the list. Recursion is performed when it encounters a workflow input port. The algorithm follows the data link to the source activity and performs the same steps there. Finally, the list is reversed and all of the activities and ports needed to produce data for the workflow output ports are part of the list. If an activity A depends upon activity B, activity B appears first in the list.

Next, each activity is translated individually from a template. For activities with multiple inputs, the Taverna user decides whether to combine these using a cross product or a dot product; the resulting Hadoop job also includes this logic. Since the execution of the activity must be done after the dot or cross product is created, activity logic is implemented in the reduce phase. In this manner map and reduce can be used to build the dot or cross product before invoking the activities (a dot product for example can be implemented as a reduce side join).

[6] https://github.com/schenck/taverna-to-hadoop

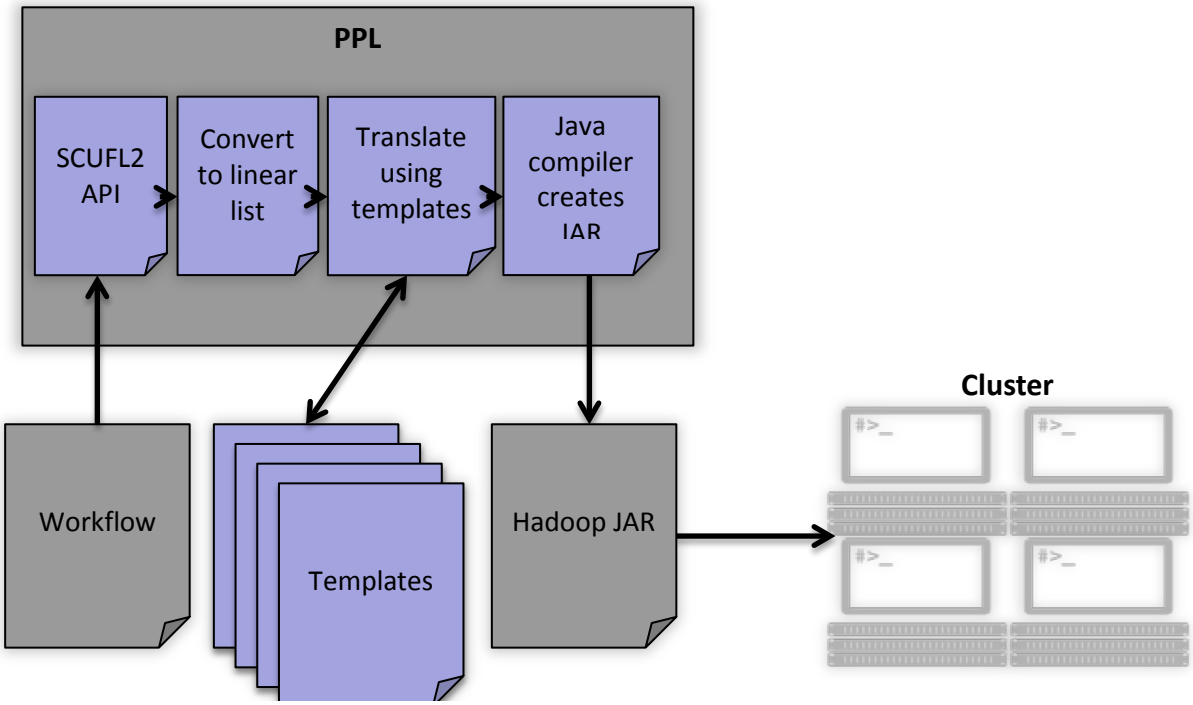

Figure 1: Compiler overview.

An activity in the Taverna workflow that is to be translated can have any number of input ports and any number of output ports. The data location of outputs from output ports is chosen based on the input ports it feeds in to. In turn, the reader that reads the data for those input ports looks in those locations and reads the data. The location consists of a prefix defined by the user, the activity's name, and the input port's name.

Finally, Java creates a JAR that can be executed on Hadoop. This JAR includes all dependencies, so that they are available to Hadoop.

2.4 Architecture

This section describes the Java classes that are part of the PPL, shown in Figure 2. The class `TavernaToHadoopMain` handles initialization and triggers the translation process. `Config` and `FileUtils` are helper classes for configuration and reading/writing files respectively. `TavernaToHadoopConverter` preforms the translation.

The translator uses java code templates to create the final MapReduce class. `TemplateTranslator` converts these general templates into Java code by replacing placeholders and organizing imports required for the execution. To put individual map and reduce classes for the respective activities in the Taverna workflow into the final class, the `WorkflowManager` reads the workflow using the SCUFL2 API[7]. It converts the workflow into a linear list of `ActivityConfigs`, each representing the execution of a Taverna activity. `ActivityConfig` is sub-classed for each type of activity. There are templates for each of activity type - sub-classes need to supply a way of translating those into java source code.

[7] https://github.com/myGrid/scufl2

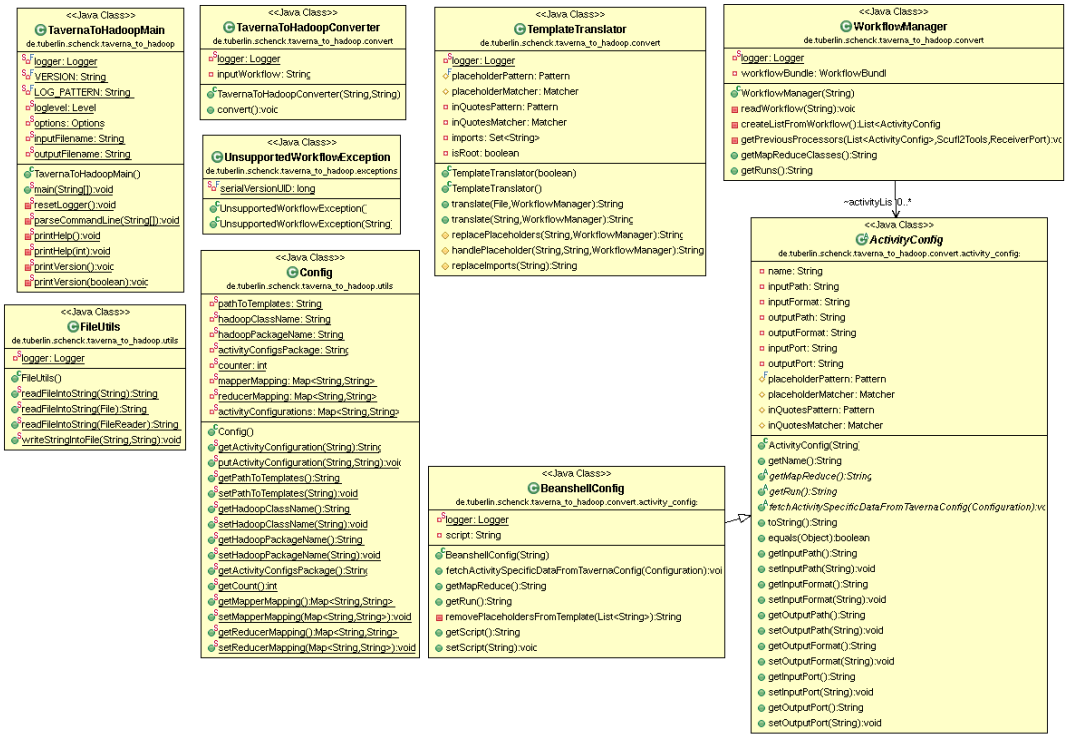

Figure 2: Class diagram of the translator.

2.5 Data Preparation

In order to be able to create a dot product in Hadoop the input data needs prior preparation. Hadoop splits large input files into small chunks and hands individual chunks to tasks as input. Each distinct task only knows the position in the complete data as a byte offset. Since each line can have any length, a byte offset does not allow for the calculation of a line number. To enable the creation of a dot product, the line number needs to be known for pairing entries from multiple sources on the same lines.

To create the dot product, the data needs to have the line number as the first value on each line, followed by a separator, and the data:

```
1  image01
2  image02
3  ...
```

This can be carried out when loading the data into HDFS and only needs doing once. Many jobs can then be executed on this prepared data, so, making the data preparation overhead (against the execution) less significant.

2.6 Extensibility

It is possible to extend the converter for any Taverna activity. The class for the conversion needs to be a sub-class of `ActivityConfig` and it must implement the abstract methods defined in the super-class. The name of the class needs to equal the name of the activity extended by the term `Config`. The PPL searches for classes and templates that match the activity names within the workflow. Translation of the templates for the activity must be provided in the methods

`getMapReduce()` and `getRun()`. This enables anyone to implement configurations for any arbitrary Taverna activity. As a result, even Taverna plug-ins can be supported to be executed on Hadoop.

2.7 Sustainability

Sustainability of the tool will be given by a number of factors. First of all, the entire code is hosted publicly on GitHub[8]. This means that anyone at any given time can correct or enhance the program. The entire source code is written in Java, one of the most popular high-level programming languages. Within the source code exists exhaustive information as JavaDoc. JavaDoc enables other programmers to understand the source code more quickly and find information about any class or method in a very structured way.

Finally, the entire code is licensed under the Apache license[9]. This means, that anyone can edit the source code in whichever way they please. None of these changes must be returned to the owner. Other software using this software does not have to be licensed under the Apache license. Any code under the Apache license can also be distributed freely. For more information see the online version of the license[10].

[8] https://github.com/schenck/taverna-to-hadoop

[10] http://www.apache.org/licenses/LICENSE-2.0.html

3 Executing Taverna Workflows on a Hadoop Cluster

This section presents initial results of running "compiled" Taverna workflows on a Hadoop cluster, i.e. not just executing a Taverna Workflow on a Taverna runtime engine on each node – the workflow is "compiled" to a MapReduce job first.

3.1 Overview

A Taverna workflow has been translated into a native Hadoop job using the *Program for parallel Preservation Load* (PPL or translator, see section 2), with the resulting jar package copied to the Hadoop cluster. Subsequently, Hadoop was invoked to execute the jobs contained within the jar. The process was benchmarked in order to compare the time taken by the automatically compiled program compared to that taken by the manually written program, which is a hand coded translation from Taverna to Hadoop.

3.2 Benchmark

Runtimes of the three versions of the Hadoop program were recorded and compared. The cluster consisted of six physical Hadoop worker machines, each with 16 2GHz cores and 30GB of RAM. Hadoop was configured so that each worker ran at most 15 concurrent mapper tasks and eight concurrent reducer tasks, resulting in a possible maximum of 138 concurrent tasks executing on the cluster.

Both jobs received identical inputs: a file of approximately 6GB containing almost 80 million lines of random text. Time taken to generate the file and transfer it to the cluster is not considered here, just the time taken to run the MapReduce tasks.

A simple Taverna workflow was created for benchmarking, consisting of only two beanshells[11]. The first processed the workflow input removing all characters after the first space. The second appends the letters "bench" to each line. The automatically compiled program was generated using the PPL translator, which places the beanshell logic was contained in the reducer step rather than the mapper, so that dot or cross products could be created in the map step, when applicable(see section 2.3 for more details). This experiment required neither dot nor cross products so this is of little concern. However, since the cluster was configured to run twice as many map tasks than reduce tasks, the decision to place logic into the reduce phase increases runtime for all experiments.

The manually created version was somewhat simpler. Exact knowledge of what the program was supposed to do meant the MapReduce job could be written more efficiently. For example, the manually created program does not use beanshells and has no reduce step. The logic was kept in two map jobs to aid comparability with the two jobs generated for the automatically compiled program. Finally, a third, manually created program contains all these steps inside one map class.

All three programs were run multiple times and the average runtimes determined. Figure 3 shows the results of the benchmarks.

[11] http://www.beanshell.org/

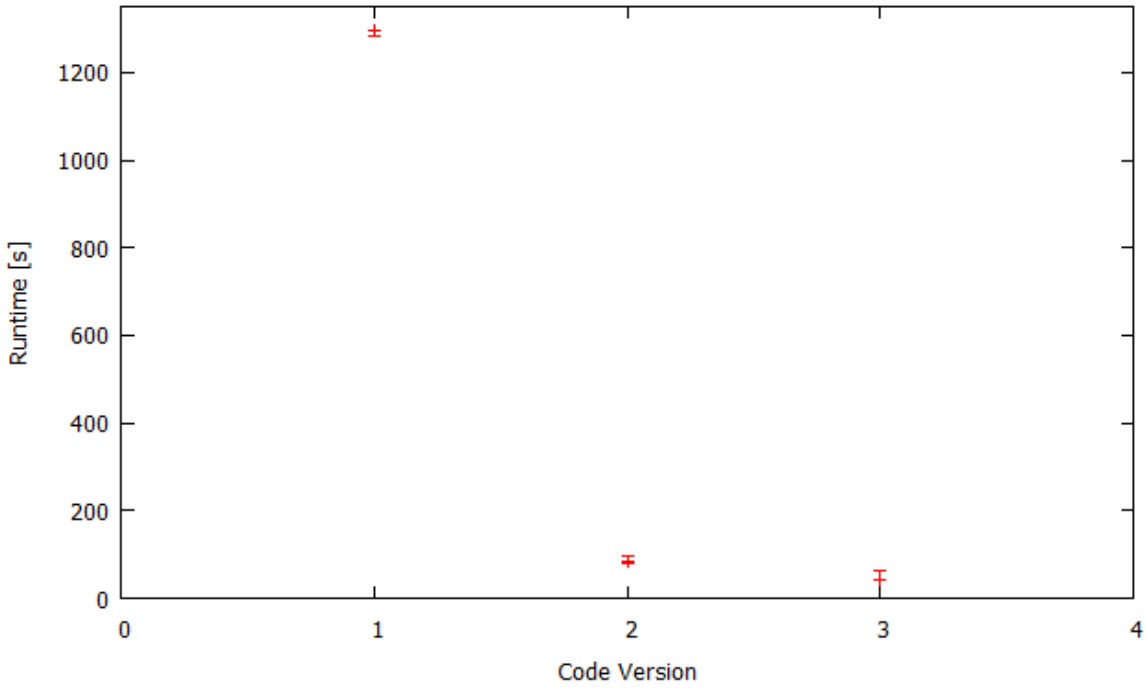

Figure 3: Runtimes of different programs (code versions). 1: automatic compilation, 2: manual program, 3: all in one map class

There are a couple of main factors which cause the differences in runtimes seen above. Probably the most significant fact is that the automatically compiled version employs beanshells to execute the logic. Every `reduce()` call creates a new beanshell which has to parse the script. Another factor is the low number of reducers in comparison to mappers. What is not a factor, however, is algorithmic structure of the programs. All comply with standard map reduce code conventions and coding styles.

A possible way to increase performance might be to create just a single beanshell per `reduce` class, not per call of `reduce()`. This would mean that the beanshell script is parsed fewer times potentially reducing runtime. Another possibility is to move the logic into a mapper instead of a reducer to have more concurrent tasks running; the downside of this is that creating dot and cross products would require an additional MapReduce phase before executing the actual logic of the activity.

3.3 Conclusion

A simple Taverna workflow was created and was then automatically converted to a Hadoop program using the PPL translator. Then a map reduce job was written that performed the same task, but more efficiently, because it was bespoke developed for the particular task. Opposed to that will be the possibility to convert any Taverna workflow. Automatic optimization is a lot more difficult than creating programs to perform simple tasks by hand. Currently automatic compilation is much less efficient than developing the map reduce jobs manually.

It is certainly possible to run Taverna workflows on a Hadoop cluster. The drawback of using a beanshell interpreter is negligible as soon as the workflow consists of other activities, specifically local tool invocations, which will spawn local tools running on the nodes and are reported to be more efficient. In that case, logic cannot be moved into the MapReduce java program and the overhead of using Hadoop is potentially lower, e.g. if the local tools are written as highly optimized C programs. However, there is room for optimization.

4 Optimization for Scalable Preservation

In the context of preservation, availability of main memory is a concern. Compared to common Hadoop jobs which often work on lines of `Strings`, data input to individual preservation tasks can be much larger, for example, a single image or audio/video data. In order to make the execution of preservation tasks on Hadoop more feasible, memory management optimization is proposed.

4.1 Introduction

The amount of data generated by data intensive preservation applications calls for distributed shared-nothing environments in order to be able to cope. In these environments, although each node is independent and self-sufficient, hardware and data is shared by multiple users or jobs. An example is the data stored at Google's data centres, which is used to generate a page rank for the stored pages, calculate ad revenues and distribution, and many more jobs simultaneously. An occurring problem in these environments is resource distribution. How should resources be split among users to be fair and at the same time guarantee good resource utilization and run times?

One resource that is always used is main memory. Many parts of the system can be considered memory consumers, i.e. they need a sufficient amount of main memory to work. Examples are buffers for network activity or sorting, and user defined functions (UDFs) running on the system. Static memory allocation by these individual components, can significantly degrade the system's performance, e.g., if a UDF allocates all available main memory, a subsequent UDF starting while the first one is still running will crash because there is no memory available for it. Another example is the execution of a UDF on heterogeneous data on multiple nodes. A fixed memory size does not consider the heterogeneity of the data and thus will probably be larger than necessary for almost all running instances. In a multi-tenant system this results in performance degradation for all users.

MapReduce is a paradigm for reliable, distributed computing on big data that incorporates UDFs in two phases: map and reduce. It was originally proposed by Dean and Ghemawat from Google in 2008 [3].

Stratosphere[12] is a system that goes beyond map and reduce by introducing an advanced parallel programming model, which allows for more second order functions.

Apache Hadoop[13] is an open source, multi-tenancy framework that implements the MapReduce paradigm [26]. Big companies like Facebook or Yahoo adopted the use of the Apache Hadoop project. Its popularity led to a lot of research that has been done in the field.

For example, Morton et al. try to estimate the progress of a running Hadoop job [15]. However, they assume knowledge about cardinality of data and a homogeneous distribution, which is not always the case. Multiple groups worked on capturing provenance in order to be able to debug jobs and the system [9] [17]. Nova is a system that runs continuous workflows on Hadoop, as opposed to the usual batch processing [16]. Stubby optimizes the execution of Hadoop workflows by enumerating a sub space of all possible execution plans and choosing the cheapest one according to a cost model, much like a query optimizer [12]. Following the trend of cloud computing, systems were developed to improve performance on elastic clouds, rather than clusters static in size. Kambatla et al. run performance analysis on a small part of the data to determine the best configuration for the cloud according to a model and a database of previously running jobs [8]. Zaharia et al. improve the scheduling of Hadoop jobs in the cloud [29]. As opposed to the standard scheduler, which assumes a

[12] https://www.stratosphere.eu/ accessed on 2013-02-04

[13] http://hadoop.apache.org accessed on 2013-02-04

homogeneous cluster environment, Zaharia et al. created the LATE scheduler, which is optimized for heterogeneous environments. Aria optimizes Hadoop's execution in the cloud by adding automatic resource allocation and de-allocation to have jobs meet soft deadlines in execution time automatically [22]. Systems like Starfish [7] optimize Hadoop job execution on multiple levels, i.e. workloads, workflows, and jobs. Aria and Starfish also optimize job execution by tuning Hadoop job parameters, e.g. degree of parallelism, according to a performance model. Tian et al. created a cost model to predict run times of jobs for different configurations, which allows them also to automatically tune Hadoop configurations to reduce run time [21].

Despite the vast research on optimization that has been done, the field of main memory consumption has not been touched as of yet. And that regardless of the fact that wrong configuration of Hadoop jobs' memories by individual users can severely decrease cluster performance. It is possible for a user to define configurations regarding resource utilization on a per job basis. But that is not fine-grained enough. Because data stored in the Hadoop Distributed File System (HDFS) can be arbitrary and therefore very heterogeneous, different tasks of a single job have different optima in configuration.

The goal of this work package will be the improvement of Hadoop's robustness for preservation tasks through automatic resource management, in this case, predominantly focussing on memory as a resource. Three main concepts are proposed:

- Consumer Manager
  - A new scheduler that handles all memory consumers.
- Cost Model
  - A model to calculate the required memory for memory consumers.
- Memory Broker
  - Is aware of available memory and will give memory to consumer manager. The amount of memory is requested from the model.

4.2 Related Work

Although memory management in Hadoop has not been approached yet, a lot of expertise exists in the individual fields of this research. In [5], Graefe describes what robust query processing should yield: "great performance every time instead of exceptional performance in exceptional circumstances." This concisely describes the goal of this work package. The occasions when a user chooses optimal resource allocation are very exceptional. By removing this resource configuration responsibility from the user, the aim is to achieve an overall performance improvement; even when the cost model does not choose the optimal amount of heap memory, the plan shall still be executed with sufficiently good performance.

Graefe et al. also published a paper on how to visualize the robustness of query execution [6]. There, they show how to create diagrams that show how well plans perform. These might help identify optimal resource configuration for Hadoop as well. TritonSort is a system that balances resources when sorting large data sets, e.g. 100TB. Their goal was to create a system that can sort datasets while finding a beneficial compromise between speed, resource utilization, and cost [17]. While TritonSort is an independent system, its successor, Themis, improves MapReduce job performance [19]. Because a lot of MapReduce jobs are I/O-bound, it minimizes the number of I/O operations to a total of two disk reads and two disk writes (if the data does not fit into memory).

Furthermore, there has been research around DB2 that is concerned with memory utilization and self-managing systems. Lightstone et al. share their experience on making the system more autonomic during the *DB2 Autonomic Computing project* [10]. Another publication by Lightstone et al. focuses on

the optimization of concurrency by adapting lock memory dynamically [11]. Finally, Storm et al. propose adaptive memory through a cost-benefit analysis and control theory [20]. Their memory controller constantly checks whether system performance can be improved through a change in memory distribution.

Related work exists for the three main components of proposed work, as described in the following subsections.

4.2.1 Consumer Manager

White gives a very good overview over all consumers of the Hadoop system, how much memory they need, and how Hadoop schedules jobs on individual nodes [26]. Yang et al. show how they choose the correct resources for the execution of certain user functions on a global grid scale [27][28]. Some of their approaches might be adaptable to choose the right nodes in a Hadoop cluster of heterogeneous nodes. This is the case, for example, when Hadoop is executed on an elastic cloud such as Amazon EC2[14].

4.2.2 Cost Model

For the cost model, two main topics are relevant: memory models and code analysis.

- Memory Models
  - A general overview over the Java memory model is given by Manson et al. [14]. Manegold et al. identify memory access patterns and provide cost functions. These functions take into account the access level in the memory hierarchy [13]. A paper by Weikum et al. gives an overview of self-tuning methods for memory management issues. The issues range from traditional caching to exploiting distributed memory. For web-based systems, they also discuss the problem of speculative prefetching [25]. Also, this work might be a little remote, but Upadhyaya et al. present a system which optimizes shared data access and makes the benefiting users pay for their improvement [22].
- Code Analysis
  - Through the use of static or dynamic code analysis, certain properties for a UDF can be derived. One possible property is the worst case execution time [4]. Knowing the worst case execution time will help the scheduler. But even more, if the time could be determined as a function of the maximum available heap space, code analysis could be used to assume good values for the UDF's heap (see above [13]). For a more general approach to execution time prediction, Vrchoticky wrote a good overview [24]. However, he assumed real-time programs and access to the compiler. Hadoop, on the other hand, should run any program compiled with any Java compiler.

4.2.3 Memory Broker

Spear and Gardner give a detailed description of how an apparatus could broker memory resources [1]. Since Hadoop is a multi-tenant system, execution of UDFs is basically a multi-user query execution. Davison et al. present a framework that manages resources for such executions [2]. They consider largely varying resource requirements, which is also the case for Hadoop.

[14] http://aws.amazon.com/en/ec2/ accessed on 2013-02-04

4.3 Resource Management

The following sub sections explain two systems and how they handle memory management: Hadoop and Stratosphere.

4.3.1 State of the Art

Hadoop's current state of the art allows any user to request as much memory as they desire. Even more than is physically available, which will result in a crash of the task. This approach is not feasible in a multi-tenant environment, because *a)* a user often does not know how much memory their UDF's need; and *b)* Hadoop cannot guarantee reliability, as machines might crash when users request too much memory.

Figure 4 shows the proportion of memory needed on every Hadoop worker node, when using the standard configuration [26]. The box sizes within the figure are scaled according to their consumption. The standard configuration for the data node is 1GB. It runs as an HDFS client on the node.

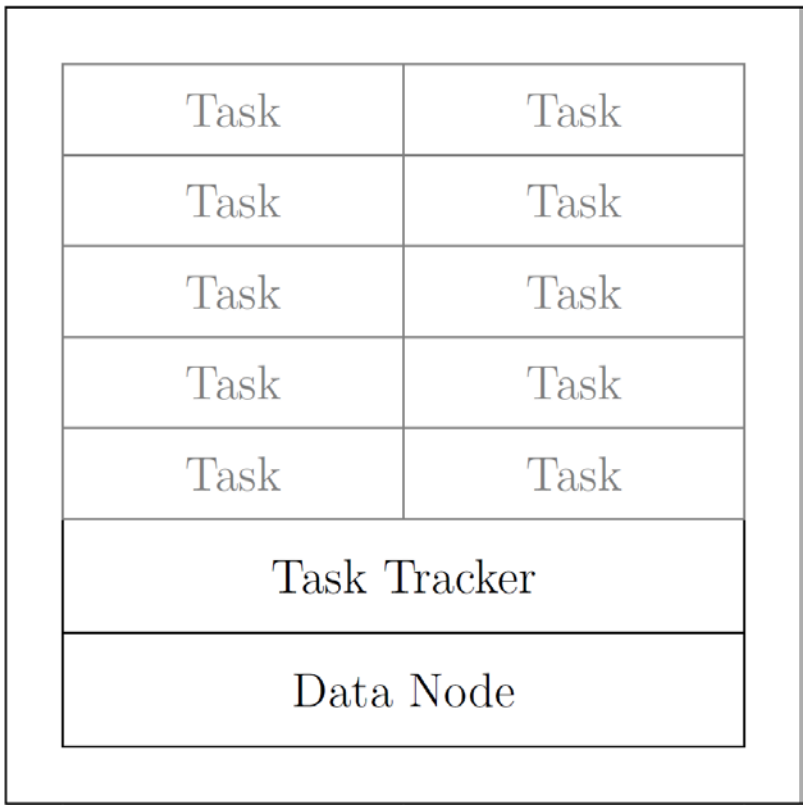

Figure 4: Hadoop memory consumers.

The common configuration for the task tracker is also 1GB. The task tracker runs as a Hadoop job client. It takes requests from the master node to run tasks on this machine. Finally, up to a certain amount of map or reduce tasks run on the machine. Usually, a map or reduce task has a maximum heap space of 200MB. If a maximum of ten tasks per node is configured, Hadoop needs a total of 4GB of available memory on each node. Besides data node, task tracker, and task slots, no significant memory consumer needs to run on a node in order to provide all of Hadoop's functionality. Stratosphere currently only allows one job to run at any given time. Consequently, a complex memory management system is unnecessary. However, as soon as multiple jobs can run concurrently, some kind of memory broker needs to be available to the system.

4.3.2 New Approach

A possible solution is a memory management system incorporating three major components, namely *a)* a consumer manager that manages consumer's memory requests per node; *b)* a memory broker that manages the available memory and gives memory to users; and *c)* a cost model for memory requirements for UDFs and possibly other consumers. A consumer in this case means any part of the system that requires memory, e.g. a UDF, a job tracker, or the HDFS client. Figure 5 shows how the system processes a memory request in four steps:

1. The consumer manager wants to create a new JVM and needs a valid configuration. A simple reason could be the creation of a new task on a machine. It asks the memory broker of that node for a configuration.
2. The memory broker manages the memory, but does not know which configuration individual JVMs require to run optimally. It requests the requirements for a certain task from the cost model.
3. The cost model uses either a learning model or code analysis to specify minimum and optimum configurations for the JVM. This is by far the most crucial part of the entire proposal. Figure 6 displays possible solutions for the creation of a cost model, and assumes that the same job will run multiple times. The *learning approach* would store intermediate results in a database. After a UDF is submitted, the model checks the database for whether information is already available. An algorithm determines a new configuration for the UDF depending on the possibly available information from the database. After execution was successful or unsuccessful, the newly gained knowledge is saved in the database. The *code analysis* approach however would try to predict performance estimates by static or dynamic code analysis. After a UDF is submitted, the database is checked and the code is analysed if the database does not store any previously gained knowledge. According to the cost model, the code is executed with a specific configuration. Success, failure, or more detailed data on performance might be written to the database, if it contains new information.
4. As soon as the required configuration is available, the memory broker tells the consumer manager, which can in turn start the JVM. The consumer manager will observe all running JVMs and how close heap space usage is to its maximum. The manager might decide to restart JVMs with different configurations and report to the cost model. In that case, the entire procedure could restart: the manager asks the broker for memory and so on. The manager might also be able to pause and migrate JVMs by serializing their states to save time, instead of restarting JVMs and their tasks.

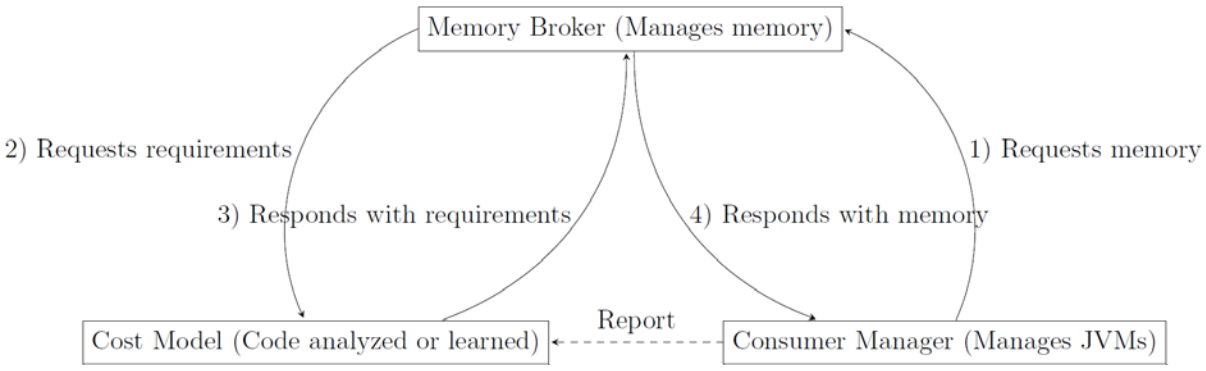

Figure 1: Flow of memory requests and responses.

The consumer manager runs on every node and schedules jobs. It might very well be necessary to change the standard Hadoop scheduler as well. A possibly easy, effective change could be a switch from task slots to fractions and multiples of task slots. That way, every task occupies a fraction or a multiple of a standard task, which means the scheduler knows when a node is completely occupied, e.g. by a task taking up all available task slots and memory. It could still happen that a task needs to wait on a node after being scheduled by Hadoop, because there are not enough slots available. An alternative would be the new YARN framework[15]. The new YARN framework decouples MapReduce logic and MapReduce management.

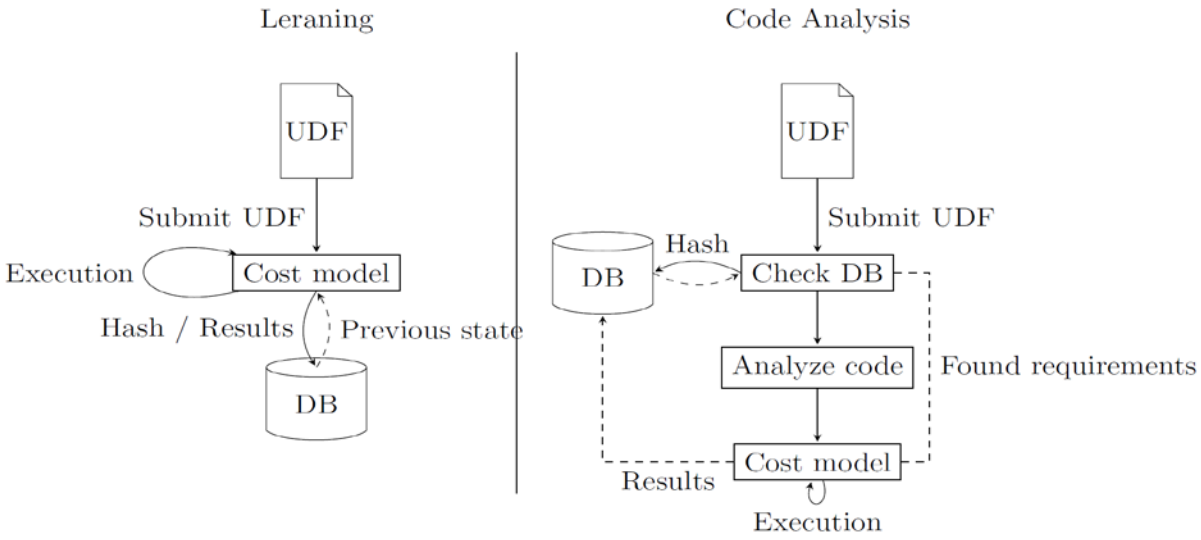

Figure 3: Cost model creation possibilities.

An open question is how the consumer manager should schedule these jobs, and other jobs that are being scheduled by Hadoop, on that node when task slots are available. Or, indeed, whether the consumer manager should block the slots even if the task is not being executed yet. Also of interest is the issue of scheduling smaller tasks while a large task is waiting. Doing that would reduce overall run time of the jobs, but might lead to the large task never being executed because slots are always in use by smaller tasks. A weighting of waiting tasks could help determine when to stop scheduling smaller tasks in favour of larger ones.

4.4 Evaluation

Evaluations can be done for every individual part of the proposal as well as for the combination of all parts, although it might be hard to evaluate the robustness of the system. Additional information of interest is, for example, run time or overall memory consumption (and their comparison).

[15] http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html accessed on 2013-03-01

5 Conclusion

It is shown to be feasible to parallelize Taverna workflows. The implementation of a prototype for the automatic translation from Taverna to Hadoop has shown that the process of parallelizing such workflows (and therefore preservation processes embedded as workflows) can easily be automated to an extent that anyone can create parallel preservation processes with ease using a graphical user interface. The automatic translator at the same time opens the possibility for automatic optimization of parallel programs.

A possible optimization strategy has been proposed that is specific to the preservation community. It considers the distinct type of data that needs to be processed by the system and how to increase efficiency for that type.

6 References

[1] Beverly Hills CA Daniel S. Spear and Woodbury MN Philip B. Gardner. Method and apparatus for Brokering memory resources, 02 2000.

[2] Diane L. Davison and Goetz Graefe. Dynamic resource brokering for multi-user query execution. In Michael J. Carey and Donovan A. Schneider, editors, SIGMOD Conference, pages 281-292. ACM Press, 1995.

[3] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107-113, 2008.

[4] Christian Ferdinand and Reinhold Heckmann. Ait: Worst-case execution time prediction by static program analysis. In Renè Jacquart, editor, Building the Information Society, volume 156 of IFIP International Federation for Information Processing, pages 377-383. Springer US, 2004.

[5] Goetz Graefe. Robust query processing. Data Engineering, International Conference on, 0:1361, 2011.

[6] Goetz Graefe, Harumi A. Kuno, and Janet L. Wiener. Visualizing the robustness of query execution. CoRR, abs/0909.1772, 2009.

[7] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F.B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In Proc. of the Fifth CIDR Conf, 2011.

[8] K. Kambatla, A. Pathak, and H. Pucha. Towards optimizing hadoop provisioning in the cloud. In Proc. of the First Workshop on Hot Topics in Cloud Computing, 2009.

[9] Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu. Perfxplain: debugging mapreduce job performance. Proc. VLDB Endow., 5(7):598-609, March 2012.

[10] Sam Lightstone, Guy M. Lohman, Peter J. Haas, Volker Markl, Jun Rao, Adam J. Storm, Maheswaran Surendra, and Daniel C. Zilio. Making db2products self-managing: Strategies and experiences. IEEE Data Eng. Bull., 29(3):16-23, 2006.

[11] Sam S. Lightstone, Chris Eaton, Yun Han Lee, and Adam J. Storm. Optimizing concurrency through automated lock memory tuning in db2. Data Engineering, International Conference on,0:1154-1163, 2007.

[12] Harold Lim, Herodotos Herodotou, and Shivnath Babu. Stubby: a transformation-based optimizer for mapreduce workflows. Proc. VLDB Endow., 5(11):1196-1207, July 2012.

[13] Stefan Manegold, Peter Boncz, and Martin L. Kersten. Generic database cost models for hierarchical memory systems. In Proceedings of the 28th international conference on Very Large Data Bases, VLDB '02, pages 191-202. VLDB Endowment, 2002.

[14] Jeremy Manson, William Pugh, and Sarita V. Adve. The java memory model. In Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL '05, pages 378-391, New York, NY, USA, 2005. ACM.

[15] K. Morton, A. Friesen, M. Balazinska, and D. Grossman. Estimating the progress of mapreduce pipelines. In Data Engineering (ICDE), 2010 IEEE 26th International Conference on, pages 681 -684, march 2010.

[16] C. Olston, G. Chiou, L. Chitnis, F. Liu, Y. Han, M. Larsson, A. Neumann, V.B.N. Rao, V. Sankarasubramanian, S. Seth, et al. Nova: continuous pig/hadoop workflows. In Proceedings of the 2011 international conference on Management of data, pages 1081-1090. ACM, 2011.

[17] Hyunjung Park, Robert Ikeda, and Jennifer Widom. Ramp: A system for capturing and tracing provenance in mapreduce workflows. In 37th International Conference on Very Large Data Bases (VLDB). Stanford InfoLab, August 2011.

[18] Rasmussen, G. Porter, M. Conley, H.V. Madhyastha, R.N. Mysore, A. Pucher, and A. Vahdat. Tritonsort: A balanced large-scale sorting system. In Proceedings of the 8th

USENIX Symposium on Networked Systems Design and Implementation (NSDI 2011), volume 11, 2011.

[19] Alexander Rasmussen, Vinh The Lam, Michael Conley, George Porter, Rishi Kapoor, and Amin Vahdat. Themis: an i/o-efficient mapreduce. In Proceedings of the Third ACM Symposium on Cloud Computing, SoCC '12, pages 13:1-13:14, New York, NY, USA, 2012. ACM.

[20] Adam J. Storm, Christian Garcia-Arellano, Sam Lightstone, Yixin Diao, and Maheswaran Surendra. Adaptive self-tuning memory in db2. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim, editors, VLDB, pages 1081-1092. ACM, 2006.

[21] F. Tian and K. Chen. Towards optimal resource provisioning for running mapreduce programs in public clouds. In Cloud Computing (CLOUD), 2011 IEEE International Conference on, pages 155-162. IEEE, 2011.

[22] P. Upadhyaya, M. Balazinska, and D. Suciu. How to price shared optimizations in the cloud. Proceedings of the VLDB Endowment, 5(6):562-573, 2012.

[23] Verma, L. Cherkasova, and R.H. Campbell. Aria: automatic resource inference and allocation for mapreduce environments. In Proceedings of the 8th ACM international conference on Autonomic computing, pages 235-244. ACM, 2011.

[24] Vrchoticky. The basis for static execution time prediction. PhD, Technical University of Vienna, 1994.

[25] G. Weikum, A.C. Koenig, A. Kraiss, and M. Sinnwell. Towards self-tuning memory management for data servers. 1999.

[26] T. White. Hadoop: The Definitive Guide. O'Reilly Media, 2012.

[27] Chao-Tung Yang, Chuan-Lin Lai, Po-Chi Shih, and Kuan-Ching Li. A resource broker for computing nodes selection in grid computing environments. In Hai Jin, Yi Pan, Nong Xiao, and Jianhua Sun, editors, Grid and Cooperative Computing - GCC 2004, volume 3251 of Lecture Notes in Computer Science, pages 931-934. Springer Berlin Heidelberg, 2004.

[28] Chao-Tung Yang, Po-Chi Shih, and Kuan-Ching Li. A high-performance computational resource broker for grid computing environments. In Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on, volume 2, pages 333 - 336 vol.2, March 2005.

[29] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In Proceedings of the 8th USENIX conference on Operating systems design and implementation, pages 29-42, 2008.

7 Appendix

7.1 Template for Beanshell Execution on Hadoop

```
<%@ requires imports =
"java.io.IOException,org.apache.hadoop.io.Text,org.apache.hadoop.mapreduce.Reducer,org.apache.hadoop.mapre
duce.lib.output.MultipleOutputs,bsh.EvalError,bsh.Interpreter" %>
  public static class <%= configName %>BeanshellReduce extends Reducer<Text, Text, Text, Text> {
    private Interpreter interpreter = new Interpreter();
    private String script = <%= script %>;
    private Text newValue = new Text();
    private String valueString;
    private String port;
    private MultipleOutputs<Text, Text> mos;

    /* (non-Javadoc)
     * @see org.apache.hadoop.mapreduce.Reducer#reduce(java.lang.Object, java.lang.Iterable,
org.apache.hadoop.mapreduce.Reducer.Context)
     */
    @Override
    protected void reduce(Text key, Iterable<Text> values,
        Context context) throws IOException, InterruptedException {
      // Empty value string
      valueString = "";

      // Clear interpreter first
      try {
        interpreter.eval("clear();");

        for(Text value : values) {
          valueString = value.toString();

          port = valueString;
          // Is within an output folder of a previous activity
          if(valueString.indexOf("-r-") != -1) {
            port = valueString.substring(0, valueString.lastIndexOf("-r-")) + "/";
          }

          port = getPortFromInput(port.substring(port.lastIndexOf(",") + 1));

          valueString = valueString.substring(0, valueString.lastIndexOf(","));
          interpreter.set(port, valueString);
        }

        interpreter.eval(script);

        <%= multipleOutputsWrite %>
      } catch (EvalError e) {
        System.err.println("Could not evaluate beanshell: " + e.getMessage());
        e.printStackTrace();
      }
    }

    /* (non-Javadoc)
     * @see org.apache.hadoop.mapreduce.Reducer#setup(org.apache.hadoop.mapreduce.Reducer.Context)
     */
    @Override
    protected void setup(Context context) throws IOException,
        InterruptedException {
      super.setup(context);
      mos = new MultipleOutputs<Text, Text>(context);
    }

    /* (non-Javadoc)
     * @see org.apache.hadoop.mapreduce.Reducer#cleanup(org.apache.hadoop.mapreduce.Reducer.Context)
     */
    @Override
    protected void cleanup(Context context) throws IOException,
        InterruptedException {
```

```
    super.cleanup(context);
    mos.close();
  }

  /**
   * Get the folder name that corresponds to the output port name.
   *
   * Removes everything behind the last slash (including) and then before the last dash (including).
   * E.g. /this/is/a/path/portname/part0
   * becomes /this/is/a/path/portname and finally
   * portname
   *
   * @param path the path of the file
   * @return the output port name
   */
  private String getPortFromInput(String path) {
    // Remove everything behind last slash (including)
    path = path.substring(0, path.lastIndexOf("/"));
    // Remove everything before last slash (including)
    path = path.substring(path.lastIndexOf("/") + 1);

    // Remove activity name in front
    path = path.substring("<%= configName %>".length());

    return path;
  }

}
```