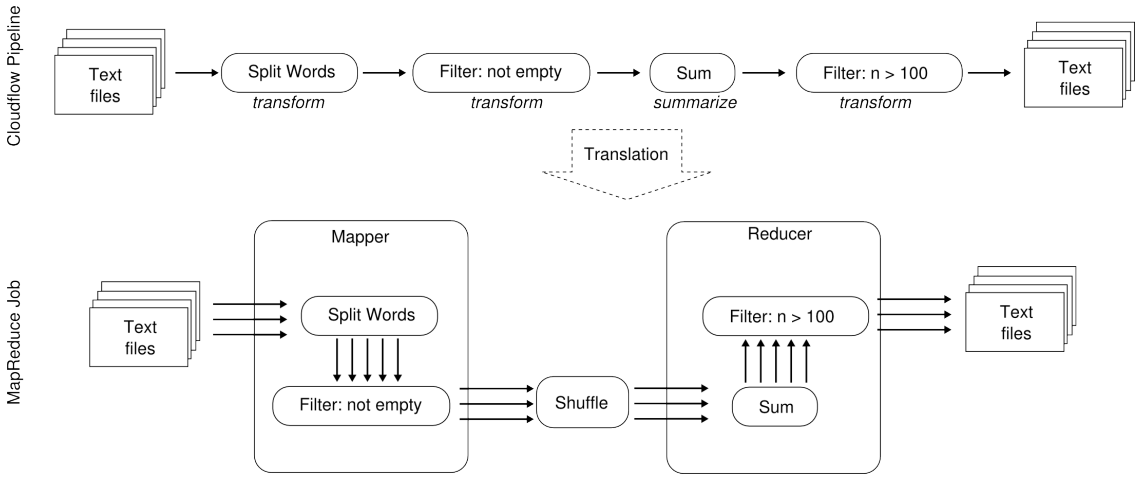

Figure 1. Cloudflow translates the operation sequence automatically into an executable MapReduce job

D. Pipeline Execution

Before the execution, Cloudflow checks the compatibility of input and output records of consecutive operations. This ensures that only valid and executable pipelines are submitted to a Hadoop cluster.

If the pipeline is executable and valid, then the operation sequence is translated into an execution plan, that decides if an operation is executed in the *map* or in the *reduce* phase. Based on this plan, Cloudflow creates one or more MapReduce jobs and configures them to execute the user-defined operations in the correct order. In this translation step, Cloudflow tries to minimize the number of MapReduce jobs by combining consecutive *transform*-operations and by executing all *transform*-operations after a *summarize*-operation in the same reducer instance (see Figure 1).

For additive *summarize*-operations (e.g. sum), Cloudflow takes advantage of Hadoop's combiner functionality. The idea of this improvement is to combine the key/value pairs that are generated by all map tasks on the same machine, into fewer pairs. Thus, the number of pairs that are transferred between mapper and reducer are minimized, which results in a positive effect on the network bandwidth since useless communication is avoided.

IV. CLOUDFLOW FOR BIOINFORMATICS

Cloudflow provides a variety of already implemented utilities, which facilitate the creation of pipelines in the field of Bioinformatics (especially for NGS data in Genetics). For that purpose, we implemented, based on HadoopBAM [7], several record types and loader classes in order to process FASTQ, BAM and VCF files. Moreover, we created several operations and filters for the analysis of biological datasets (see Table I for an overview of all currently implemented operations and filters).

For example, a typical quality control pipeline for VCF files can be implemented by simple combining of several built-in operations. First, we apply predefined filters to discard variations that are monomorphic, marked as duplicates, or are Insertions or Deletions (InDels). For all records passing the filters, Cloudflow applies a *summarize*-operation that calculates the call rate for each variation (see Listing 2).

```
Class CallRateCalc extends Transformer {
  public void transform(VcfRecord record) {
    VariantContext snp = record.getValue();
    float call = callRate(snp);
    emit(new FloatRecord(snp.getID(), call);
  }
}

pipeline.loadVCF(input)
        .filter(MonomorphicFilter.class)
        .filter(DuplicateFilter.class)
        .filter(InDelFilter.class)
        .transform(CallRateCalc.class)
        .save(output);
```

Listing 2. VCF Quality Control Pipeline using Cloudflow

V. DEPLOYING PIPELINES AS A SERVICE

Cloudgene [6] is a web-based platform to create and execute workflows consisting of Hadoop MapReduce, Apache Pig and command line-based programs. It can be seen as an additional layer between Hadoop MapReduce and the end user that hides the complexity of the MapReduce framework. Therefore, Cloudgene is the perfect candidate to provide Cloudflow pipelines as a service. Such pipeline can be integrated into the workflow platform by utilizing Cloudgene's plugin interface. No adaptation to the source code is needed, while only a simple plain text file including a header, input parameters, output parameters and the definition of the workflow itself need to be created. When launching Cloudgene, the manifest file is loaded and the client interface is automatically rendered using information from the file. As Cloudgene supports different technologies, it is possible to parallelize the calculations using Cloudflow and to visualize the results using R.

Cloudflow requires a compatible MapReduce cluster for executing pipelines. CloudMan [8] makes it possible to easily procure and configure a functional data analysis platform on a cloud infrastructure. The procured platform delivers a scalable cluster-in-the-cloud and a data analysis environment preconfigured with a number of applications. With its ability to be launched and managed via a web browser on a number of clouds, customized as necessary, and easily shared with collaborators, CloudMan makes it

TABLE I. CURRENTLY SUPPORTED DATA FORMATS AND OPERATIONS

Data Format		Pipeline Operation	Description
Fastq	Split	`split()`	Find pairs (for paired-end reads)
	Filter	`filter(LowQualityReads.class)`	Filters reads by quality
		`filter(SequenceLength.class)`	Filters reads by sequence length
	Other	`findPairedReads()`	Detects read pairs
		`align(referenceSequence)`	Aligns sequences against a reference (using jBWA for alignment)
BAM	Split	`split()`	Creates fixed size chunks (e.g. 64 MB)
		`split(5, BamChunk.MBASES)`	Creates logical chunks (e.g. 5MBases)
	Filter	`filter(UnmappedReads.class)`	Filters unmapped reads
		`filter(LowQualityReads.class)`	Filters reads by map.quality
	Other	`findVariations()`	Finds variations in aligned reads (using samtools)
VCF	Split	`split()`	Creates fixed size chunks (e.g. 64 MB)
		`split(5, VcfChunk.MBASES)`	Creates logical chunks (e.g. 5MBases)
	Filter	`filter(MonomorphicFilter.class)`	Filters monomorphic site
		`filter(DuplicateFilter.class)`	Filters duplicates
		`filter(InDelFilter.class)`	Filters inDels
		`filter(CallRateFilter.class)`	Filters by call rate
		`filter(MafFilter.class)`	Filters by MAF
	Other	`checkAlleleFreq(reference)`	Allele frequency check with external reference (e.g. 1000 genomes)

possible to readily utilize cloud resources in a research environment. The approach on how Cloudgene and CloudMan can be combined efficiently have already been demonstrated [9].

VI. EVALUATION

To evaluate our approach, we implemented three different Bioinformatics data-analysis pipelines using Cloudflow and integrated them into Cloudgene. The results of our experiments demonstrate that Cloudflow has only a minimal overhead in the execution time compared to an identical pure MapReduce implementation. However, the performance of Cloudflow is better than Apache Crunch's (see Figure 2).

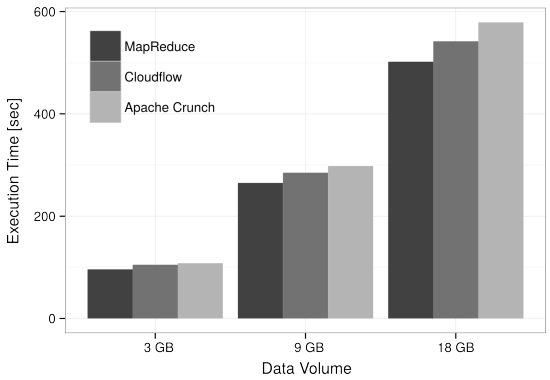

Figure 2. Execution Time of a Cloudflow pipeline, an Apache Crunch pipeline and a pure MapReduce implementation of WordCount

A. Preprocessing and Mapping

When working with NGS data, the quality of raw data needs to be checked before a successful subsequent downstream analysis (e.g. read mapping/alignment) can be achieved. The overall goal of alignment is to dock the vast amount of short reads, mostly in the FASTQ file format, to a reference genome. Factors such as read errors, insertions or deletions of bases must be considered that finally results in the most accurate genome position for each read.

The goal of the following pipeline is the alignment of paired-end data to a reference genome. Therefore, the FASTQ data is loaded using the FastqLoader, which creates records for each sequence. The records are then filtered by an average base quality of 30. Numerous other quality metrics (such as sequence length or C/G content) can be filtered as well. Since paired-end reads are used, read pairs are detected using a predefined *transform*-operation. The aligner step is implemented as a *summarize*-operation that calls a parallelized version of BWA-MEM [10]. This has been achieved by using JNIs (https://github.com/lindenb/jbwa). Similar to BWA-MEM 99,100 reads are aligned to the user-specified reference genome in one batch. Aligned reads are saved in HDFS in the BAM file format (see Listing 3.A).

B. Variation Calling

After data has been aligned and cleaned (e.g. removing duplicates, quality recalibration), the next step of NGS pipelines is the detection of reliable variants that can be used e.g. in association studies. A widely used pipeline for variant detection is GotCloud, developed at the Center of Statistical Genetics (University of Michigan) and utilized in the 1000 Genomes (1000G) project.

In this example pipeline, the aim is to find variations without a statistical model by implementing a simple counting approach of the four bases. This is only possible when using high coverage data. Therefore, in the first step the BAM file (created in the previous pipeline) is loaded and chunked in user-specified splits (e.g. 5 MBases). Variations are then detected for each chunk by counting

the occurrences of A, C, G, T on each position. Finally, the detected variations are stored in VCF files (see Listing 3.B).

C. Genome-Wide Association Studies

Many genome-wide association studies (GWAS) have identified associations between various phenotypes and common sequence polymorphisms, which might play a role for disease development. Technologies like microarrays have made it possible to measure millions of single nucleotide polymorphisms (SNPs) of one individual simultaneously and for low cost. Since the costs of microarrays is much lower than next-generation sequencing (NGS), it is today the cheapest method to genotype large-scale population studies. Such datasets are combined with collected phenotypes (e.g. diseases and measured data) in order to detect if one of these variations has a high impact on the value of a phenotype. Since the size of such datasets grows rapidly, a parallelization at the data-level is necessary to analyze the data in appropriate time.

```
//A. Preprocessing and Mapping
pipeline.loadFastq(input)
        .filter(LowQualityReads.class, 30)
        .findPairedReads()
        .align(refSeq)
        .save(output);

//B. Variation Calling
pipeline.loadBam(input)
        .split(5, ChunkSize.MBASES)
        .groupByKey()
        .findVariations(refSeq)
        .save(output);

//C. Genome-Wide Association Study
pipeline.loadText(input)
        .split(1000, ChunkSize.LINES)
        .execute(SnpTestExecutor.class).
        .filter(FilterHeader.class)
        .filter(FilterInvalidSnps.class)
        .save(output);
```

Listing 3. Complete NGS pipeline using Cloudflow

The parallelization of the association analysis was realized by splitting the list of markers into chunks. In detail, the mapper splits all input SNPs into chunks with a fixed number of SNPs (e.g. 1000). Then, the reducer executes the linear regression model for each chunk by using SNPTest. Finally, the reducer collects the results and merges them into a single file. The corresponding Cloudflow pipeline loads the text input file and automatically creates records for each line. On these records we apply the *split* operation, which creates chunks containing a fixed number of lines. For the execution of the SNPTest program, we can implement a special operation called *BinaryExecutor,* which enables us to write the chunks automatically to the POSIX file system. In the next step, we can use this file as the input file for SNPTest. After the execution the operation creates text records for each line of results (see Listing 3.C).

VII. Conclusion

Cloudflow's overall aim is to simplify the development of complex MapReduce pipelines by abstracting the map and the reduce function from end users. Therefore, operations need only be written once and can be re-used for future MapReduce usage. The major advantage of Cloudflow lies in the provision of validated operations, especially in the area of genetics, and its extensibility. Combining Cloudflow with CloudMan (cluster orchestration) and Cloudgene (Hadoop workflow system) allows users to use Hadoop without a deeper knowledge of the internal MapReduce concepts and could yield to a boost of Hadoop in genetics.

Acknowledgment

This work was, in part, supported by the "Scalable Big Data Bioinformatics Analysis in the Cloud" grant from the Croatian Ministry of Science, Education, and Sport and the Austrian Federal Ministry of Science and Research (BMWF) and by the FP7-PEOPLE programme grant 277144 (AIS-DC).

References

[1] V. Marx, "Biology: The big challenges of big data," *Nature*, vol. 498, pp. 255–260, 2013.

[2] S. Yazar, G. E. C. Gooden, D. A. Mackey, and A. W. Hewitt, "Benchmarking undedicated cloud computing providers for analysis of genomic datasets.," *PLoS One*, vol. 9, no. 9, p. e108490, Jan. 2014.

[3] A. Schumacher, L. Pireddu, M. Niemenmaa, A. Kallio, E. Korpelainen, G. Zanetti, and K. Heljanko, "SeqPig: simple and scalable scripting for large sequencing data sets in Hadoop.," *Bioinformatics*, vol. 30, no. 1, pp. 119–20, Jan. 2014.

[4] H. Nordberg, K. Bhatia, K. Wang, and Z. Wang, "BioPig: a Hadoop-based analytic toolkit for large-scale sequence data.," *Bioinformatics*, p. btt528–, Oct. 2013.

[5] C. Chambers, A. Raniwala, F. Perry, S. Adams, R. R. Henry, R. Bradshaw, and N. Weizenbaum, "FlumeJava," *ACM SIGPLAN Not.*, vol. 45, no. 6, p. 363, May 2010.

[6] S. Schönherr, L. Forer, H. Weißensteiner, F. Kronenberg, G. Specht, and A. Kloss-Brandstätter, "Cloudgene: A graphical execution platform for MapReduce programs on private and public clouds," *BMC Bioinformatics*, vol. 13, no. 1, p. 200, 2012.

[7] M. Niemenmaa, A. Kallio, A. Schumacher, P. Klemelä, E. Korpelainen, and K. Heljanko, "Hadoop-BAM: Directly manipulating next generation sequencing data in the cloud.," *Bioinformatics*, p. bts054–, Feb. 2012.

[8] E. Afgan, B. Chapman, and J. Taylor, "CloudMan as a platform for tool, data, and analysis distribution.," *BMC Bioinformatics*, vol. 13, no. 1, p. 315, Jan. 2012.

[9] L. Forer, T. Lipic, S. Schonherr, H. Weisensteiner, D. Davidovic, F. Kronenberg, and E. Afgan, "Delivering bioinformatics MapReduce applications in the cloud," in *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on*, 2014, pp. 373–377.

[10] H. Li, "Aligning sequence reads , clone sequences and assembly contigs with BWA-MEM," vol. 00, no. 00, pp. 1–3, 2013.