Mario: Interactive Tuning of Biological Analysis Pipelines Using Iterative Processing

Martin Ernstsen[1*], Erik Kjærner-Semb[2†], Nils Peder Willassen[2], Lars Ailo Bongo[1]

[1]Dept. of Computer Science and Center for Bioinformatics, University of Tromsø, Norway
[2]Dept. of Chemistry and Center for Bioinformatics, University of Tromsø, Norway
martin.ernstsen@ksat.no, erikkj@imr.no,
nils-peder.willassen@uit.no, larsab@cs.uit.no

Abstract. Biological data analysis relies on complex pipelines for cleaning, integrating, and summarizing data before presenting the results to a user. Specifically, biological data analysis is usually implemented as a pipeline that combines many independent tools. During development, it is necessary to tune the pipeline to find the tools and parameters that work well with a particular dataset. However, as the dataset size increases, the pipeline execution time also increases and parameter tuning becomes impractical. No current biological data analysis frameworks enable analysts to interactively tune the parameters of a biological analysis pipelines for large-scale datasets. We present Mario, a system that quickly updates pipeline output data when pipeline parameters are changed. It combines reservoir sampling, fine-grained caching of derived datasets, and an iterative data-parallel processing model. We demonstrate the usability of our approach through a biological use case, and experimentally evaluate the latency, throughput, and resource usage of the Mario system. Mario is open-sourced at bdps.cs.uit.no/code/mario

Keywords: Iterative processing; interactive processing; biological data analysis; parameter tuning; provenance.

1 Introduction

Recent technological advances in instrument, computation and storage technologies have resulted in large amounts of biological data [1]. To realize the full potential for novel scientific insight in the data, it is necessary to transform it to knowledge through analysis. A computer system for analyzing biological data typically consist of three main components: the input data, a set of tools in a pipeline, and finally a data exploration tool (Fig. 1). Biotechnology instruments such as short-read sequencing machines produce the input data. The datasets can range in size from megabytes to many terabytes. A series of tools process the data in a pipeline where the output of one tool is the input to the next tool (Fig. 1). A specific biological data analysis project often requires a unique combination of tools. There are hundreds of available tools, ranging from small mall, user-created scripts to large, complex applications. Finally, the pipeline res-

[*]Now at Kongsberg Satellite Services AS, Tromsø, Norway
[†]Now at Institute of Marine Research, Bergen, Norway

In Proc. of 5th International Workshop on High Performance Bioinformatics and Biomedicine (HiBB'14)

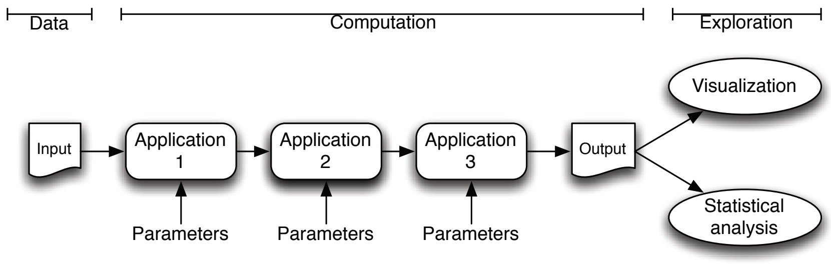

Fig. 1. A biological data analysis pipeline.

ults are exported to a data exploration tool for interpretation by biomedical researchers. An important and time-consuming part of bioinformatics analysis is pipeline setup and configuration. This includes selecting tools, and the best parameters for each tool. The parameters used may have a big impact on the quality and hence usability of the pipeline output, but the long processing time of existing biological analysis pipelines makes such parameter tuning impractical and time consuming.

We believe a system for interactive parameter tuning of biological data analysis pipelines should satisfy the following requirements: (i) the data processing should scale to the upcoming peta-scale datasets; (ii) when pipeline parameters are changed, the pipeline data processing should provide low end-to-end latency to enable quick updates of the pipeline output; (iii) the system should support unmodified pipelines and tools, since it is not practical to make and maintain changes to the source code of the many tools used in pipelines. The system should also handle input and output from each stage regardless of the tool data format; and (iv) the systems should manage provenance information [2] to ease parameter tuning and ensure result reproducibility.

To our knowledge, no existing system fulfills all these requirements. Pipeline managers such as Galaxy [3] do not scale to peta-scale datasets. Data-intensive computing systems such as MapReduce [4], [5] scale to peta-scale datasets, but are not suited for low-latency processing. Systems for iterative and interactive processing of large-scale data, such as Spark [6], Dremel [7], HaLoop [8], Naiad [9], and Nectar [10] require changes to the pipeline tool's source code.

We propose the Mario system. It fulfills all four requirements. Mario has a parallel shared-nothing architecture for computations and scalable storage. It combines reservoir sampling, fine-grained caching of derived datasets [10], and an iterative data-parallel processing model with low end-to-end latency. Mario provides transparent iterative processing and a storage model that is agnostic to the data types used by the tools [11]. It enables data provenance by storing detailed pipeline configurations associated with pipeline input-, intermediate-, and output data.

We contribute by providing a requirements analysis for biological analysis pipeline parameter tuning, and describing the design and implementation of Mario. We also contribute with an experimental evaluation of the performance and resource usage of Mario. In addition, we describe how we used HBase as storage backend for biological data, and we demonstrate the usability of Mario through a metagenomics case study.

2 Architecture and Design

We make the following assumptions that form the basis for the architecture and design of Mario: (i) input data can be split into parts with fine granularity; (ii) no intermediate pipeline stage requires access to the complete input data; and (iii) there is enough storage to hold all intermediate data.

The first two assumptions allow iterative processing of the dataset with inspection of output as the computation proceeds. This is our approach for achieving low end-to-end latency during pipeline tuning. The third assumption allows caching of intermediate data to avoid recomputing the full dataset after configuration changes.

The first assumption usually holds, since many genomic analysis tools either base the analysis on fine-grained parts such as sequences of nucleotides or gene expression values, or they use a summarization or machine learning algorithms on fine-grained parts. The second assumption is true for most algorithms that scale to large datasets. These typically split the data to independent parts, and distribute and process these independently on many nodes. In addition, tools that aggregate data from the full dataset can often easily be replaced by incremental versions that maintains a summary of the full dataset between iterations. The third assumption usually holds for clusters designed for data-intensive computing.

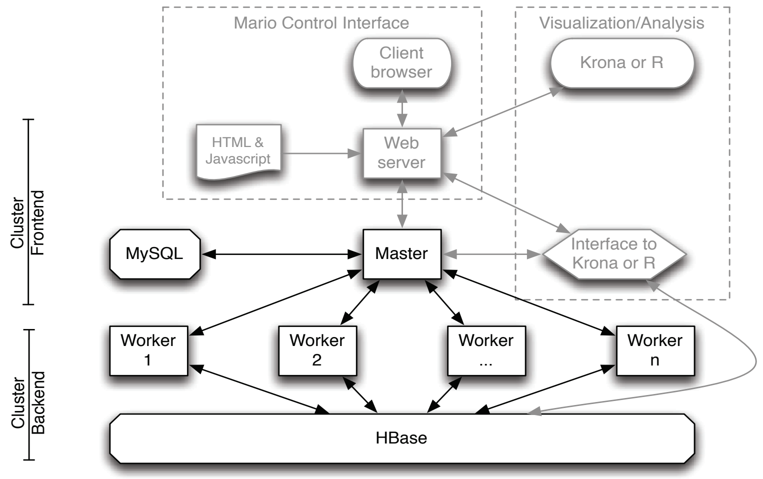

Fig. 2. Mario architecture. Greyed out parts are not implemented in current prototype.

The Mario architecture (Fig. 2) consists of four tiers: storage, logic/computation, the web server and the client/UI. The system runs on a cluster of computers, with the master process at the cluster frontend, and the workers at the compute nodes of the cluster. These are co-located with an HBase [12], [13] installation that has the HBase master at the frontend and the HBase region servers at the compute nodes. The web server and the MySQL server can be located on the cluster frontend, or on separate computers. The user runs the Mario controller and the data exploration tools on her computer.

To analyze a dataset using Mario, a user would first load the input data into HBase. She would then use a web interface such as Galaxy [3] to define the pipeline by specifying, for each pipeline stage: the tool to execute, the tool version, and the tool parameters. The configuration can specify that a dataset should be sampled with a given sample size. Mario uses a file-based approach for incremental processing [11], so it is not necessary to modify tool code. The user then starts the initial computation. As the computation proceeds, the user can change the parameters or tool used in a stage by sending an updated configuration to the master, which will start scheduling work with the new configuration. If the new configuration does not produce satisfactory results, Mario can restore a previous configuration by reading data for a previous configuration from the storage layer.

2.1 Storage Layer

The primary component of the storage layer is HBase, where Mario stores input-, intermediate-, and output data. Intermediate and output data can be stored in multiple versions resulting from the use of different parameters to pipeline stages. The HBase storage architecture is demonstrated to scale to peta-scale datasets [13]. We chose HBase since it is widely used for interactive computing systems, and because it is well suited for storing and accessing the data needed by Mario since it provides low-latency random reads and writes, and efficient storage and compression of sparse data structures. To reduce long-time storage requirements, Mario can delete intermediate data and perform an HBase major compaction when an analysis project is completed.

Mario also uses a MySQL database to store the different pipeline configurations, including parameters to each stage, used for computing the intermediate and output data stored in HBase. These represent a history of configurations, so that Mario users can find cached data computed in a previous configuration. In addition, Mario stores metadata about HBase tables, and tool specifications in the database. The latter includes tool versions and allowed parameters.

2.2 Logic and Computation Layer

This layer contains a single master server, and multiple worker processes. The user would interact with the master through a web client. When starting a job, the master will distribute work to the workers. All workers operate independent of another, processing separate parts of the dataset in parallel. The master provides each worker with the current configuration of the pipeline and the HBase row key of the data to be processed. The master can also query HBase for the location of the HBase region server responsible for a row, and assign the row to a Mario worker located on the same server. This will improve data locality and therefore reduce network traffic.

The master retrieves the row keys from HBase, but does not retrieve the data stored under each key or perform any processing. The reservoir sample is stored in memory as a list of row keys. This sample is the main source of memory usage in the master. However, assuming 20-byte key length, a large sample of four million keys will only consume 80MB of memory.

The worker processes wait for messages from the master server. When receiving a message, the worker reads the data associated with the keys in the message from HBase. The pipeline stages process the data, and write intermediate and final output into HBase. When a worker has completed its work, it sends a message to the master. This enables the master to adjust work distribution to the capacity of the workers, and to notify visualization tools that new results are available. The worker processes may be CPU and memory intensive depending on the computation done in each tool.

2.3 Web Server, and the Visualization and Analysis Interface

The web server hosts the Mario control application and forwards requests from this application to the master server. The web server also forwards data generated by a *visualization and analysis (VA) interface* to data exploration tools.

A data exploration tool uses Mario's VA interface to retrieve results from HBase, either periodically or when notified of the presence of new results by the Mario master. To integrate Mario with a visualization system such as Krona [14] or METAREP [15], an interface must be implemented that updates the data structures used by the exploration tool. For Krona, this involves generating an XML-file of the organism hierarchies found in the data. For METAREP, a search engine data structure must be updated (refer to [16] for details). For integration with end-user statistical analysis, the interface can be implemented in for example R.

3 Implementation

In this section, we describe the implementation of data storage in HBase, and the reservoir sampling algorithm. Additional implementation details are in [16].

3.1 HBase Tables

Mario stores data in HBase tables. A table consists of rows that are identified by row keys. Each row contains cells that store data. The cells are uniquely identified by a key consisting of column family, column, and timestamp. The key is stored, together with each cell, in a byte array within an immutable file that is lexicographically ordered by row key. HBase allows adding columns dynamically. Mario uses column names generated at runtime to provide a mapping between the data in the column and the pipeline configuration used to generate that data.

The HBase table for an analysis pipeline has two column families: *in* and *out*. The unique row key is dataset dependent, and can for example be the line number in the input file that contains the data value, or a sequence ID in a FASTA file. The input data is stored in the *in* column family before the pipeline is executed. Output from pipeline stages are stored in the *out* column family. HBase column names are pipeline execution version numbers that combine a unique ID for each tool, the version of the stage tool configuration, and the version of the tool that produced the input data. Since only a small subset of rows are processed to test each configuration, most columns will be empty for most of the rows. HBase is ideally suited for storage of such sparse data.

Fig. 3 shows a version tree for an example three-tool pipeline where the user has modified tool parameters three times. The top row shows the column names for the initial versions of each stage of the pipeline. The second branch is the result of changing the parameters of the first tool of the pipeline, but leaving the other two tools unchanged. Even if only the first stage is changed, the version numbers of the downstream stages are incremented to create columns for storing the data based on the output from the new first stage. In the same way, the lower branch in Fig. 3 is the result of changing the second stage of the pipeline. The resulting HBase table will have eight columns in the *out* family.

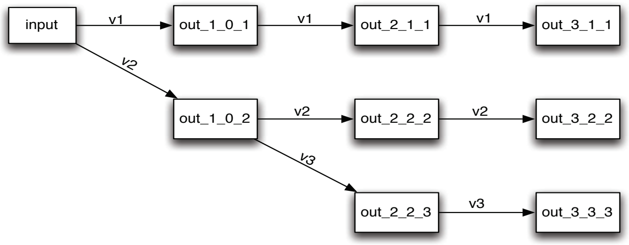

Fig. 3. Data version with HBase column names

3.2 Reservoir Sampling

The Mario master server uses the reservoir sampling algorithm described in [17] to produce a random sample of elements from an HBase table. The algorithm does one pass through the table keys, and requires the generation of one random number per key. It guarantees that each element in the table has equal probability of being in the sample. The single-pass property of reservoir sampling make this technique well suited for sampling large datasets where performance is I/O limited. If weighted sampling is required other single pass algorithms such as [18] could be implemented without a significant I/O performance overhead.

4 Evaluation

The goal of the experimental evaluation is to: (i) demonstrate the usefulness of Mario for biological analysis pipeline tuning; (ii) validate the suitability of HBase as a storage backend for an iterative, interactive system; and (iii) validate the architecture and the design choices made for the Mario prototype.

4.1 Methodology

We used a nine-node cluster, where each compute node has an 8 core Intel Xeon 3.6GHz CPU, 32GB DRAM, and 2 x 2TB disks. The network was 1Gbps Ethernet using a single switch. The operating system was CentOS 6.31. We used the Cloudera Hadoop CDH4.3.0, with HBase v0.94.6 and Hadoop v2.0.0. We used ZeroMQ v3.2.4 for communication. The HBase master server had a JVM with 4GB of memory. HBase regionservers allocated 12GB of memory. Unless otherwise noted, we measured the elapsed wall time. We repeated all experiment five times, and report average time and sample standard deviation.

4.2 Use Case: Taxonomic Classification of Metagenomics Samples

To demonstrate the usefulness of Mario for biological analysis pipeline tuning, we analyze the results of a taxonomic classification in metagenomics samples (details are in [19]). The rarefaction curve in Fig. 4 shows that most of the phyla are identified with only 6-8 million reads and that most of the classes and orders are found with 10-12 million reads. However, even with 2 million reads 75% of the orders are discovered. This shows that processing a small subset of the data provides initial biological insight.

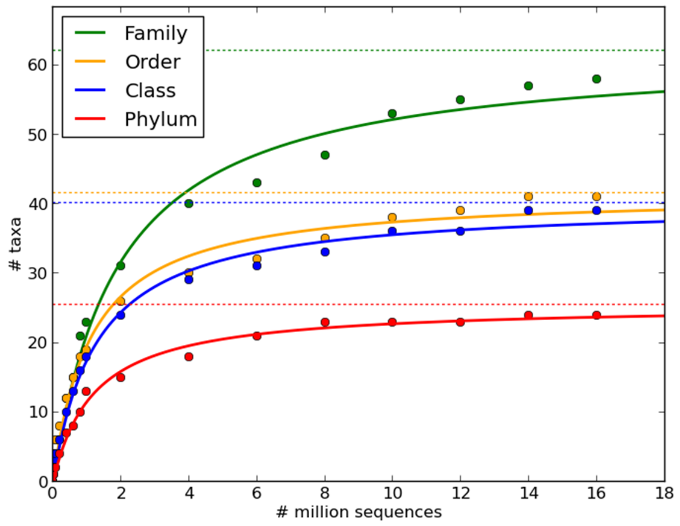

Fig. 4. Rarefaction curves for analyzing species richness and sample completeness of different taxonomic ranks. Figure is from [19], which provides additional experiment details.

4.3 HBase as Storage Backend for Biological Data.

We first evaluate HBase read and write performance for biological datasets. Since Mario relies on HBase for storage, these results provide a lower bound for the end-to-end latency and throughput of Mario data processing.

Mario will execute jobs that access both large and small amounts of data. Earlier evaluations of the Google BigTable/ HBase design [13] have demonstrated its scalability for datasets with millions of rows. We therefore focus on the latencies of reading and writing small amounts of data to and from HBase. We generated a synthetic dataset representative of FASTA files, and BLAST [20] tabular output. The FASTA files have random nucleotide sequences, ranging in length between 100 and 5000 bases. IDs were random 15 character strings. The BLAST output is similar to the output generated using the –m 8 option.

To evaluate the performance of tables stored on a single region server, versus regions on multiple servers, we use a table with only 200KB of data, and a table with approximately 500GB of data. The latter is large enough to fill the aggregated DRAM on the cluster (288GB). It is generally recommended to compress HBase tables [13]. We use the Snappy compression algorithm since it offers the highest encoding and decoding rate of the HBase compression algorithms. HBase also uses Bloom filters to avoid having to scan files for keys that are not in the file [13]. For all experiments, we enable the client scanner cache to reduce the number of RPC requests to HBase region servers. Disabling it significantly decreases performance. In addition, we use deferred log flushing. Mario accepts data loss since the data can easily be recomputed. The default write ahead logger gives orders of magnitude worse performance. We flush the HBase cache between each measurement to get the worst-case performance results. Informal experiments show that a warm cache improves performance by an order of magnitude. We spent a considerable amount of time tuning these and other parameters [16]. Others have reported similar efforts required even by experts to performance tune HBase installations [21].

Reading 200 000 rows of data (22MB) from an almost empty table, and a table larger than the DRAM size on the cluster takes respectively 11 and 13 seconds, regardless of compression and Bloom filter settings ([16] has additional details). Since we flush the HBase caches between each experiment, these represents worst-case results, but still we consider the read performance sufficient for Mario.

Writing 200 000 rows (22MB) into respectively an empty and populated HBase takes 10 and 7 seconds, regardless of compression setting ([16] has additional details). Writing to a populated table is faster since writes are distributed on multiple region servers. We consider the write performance sufficient for Mario.

4.4 Mario Overhead and Throughput

We measure the overhead added to the end-to-end latency of Mario. We define end-to-end latency as the time from a computation is started, until the first results are ready to be visualized. In addition, we measure the throughput of Mario, which we define as the amount of data that can be processed per time unit. We use a dummy four-stage pipeline using the Linux *cat* tool in each stage. Since cat has no computation, it represents an I/O intensive application. For more computation intensive applications, Mario overhead will contribute less to the wall clock execution time, and the throughput will be reduced.

To emulate a user tuning pipeline parameters, each experiment consists of three executions of the dummy pipeline with the same input data (FASTA-type generated as described above). Before the second execution, the last two stages are marked as modified. Before the third execution, the modifications are reverted.

Table 1. Time top process one row (one nucleiotode sequence).

	Avg (ms)	SD
Execution 1	68	6.4
Execution 2	7	1.4
Execution 3	6	1.1

Table 2. Time to process 200 000 rows (500MB).

	2 Workers		4 Workers		8 Workers	
	Avg (ms)	SD	Avg (ms)	SD	Avg (ms)	SD
Execution 1	11699	766	11784	1871	10817	1224
Execution 2	7432	288	8279	2497	8302	1377
Execution 3	7689	483	8406	2334	8563	2096

Table 3. Time to process 10 000 row sample of 200 000 rows using eight workers.

	Avg (ms)	SD
Execution 1	10777	242
Execution 2	55	6
Execution 3	38	13

The Mario overhead is 68ms for the first stage, and 7 and 6 ms for the second and third execution (Table 1). The reduced latencies in executions 2 and 3 are due to the data being cached in HBase after the first execution. In the second execution, the worker detect that data exists for the first two stages, and only executes cat for the last two pipeline stages. In the third execution, the pipeline result are read directly from the cache without executing any pipeline tools. However, there is no significant performance difference between these executions. The results demonstrate that the end-to-end latency added by Mario is very small.

To measure the throughput of Mario, we use a 200 000 row (500MB) dataset, which is approximately the size of a two million short read dataset, and which is large enough to provide initial biological insight (section 4.2). The throughput is limited by task distribution, and does not improve with more workers (Table 2). The 500MB dataset will fit in the cache of a single HBase region server. The Mario master will therefore read row keys from an in-memory cache, but still be limited by the time it takes to iterate over the sequence of keys and transmit task messages to workers. For a pipeline with more computationally intensive jobs, additional workers will improve throughput. The results demonstrate that Mario provides high throughput data processing.

4.5 Sampling

To measure the overhead of sampling, we use the experiment setup described in the previous section. The Mario master creates a sample size of 10 000 keys from the 200 000 row dataset, and distributes the samples to eight workers. Our results show that the execution time with sampling (Table 3) is similar to the time without sampling (Table 2). This shows that the sampling algorithm has very low overhead, since it in addition to executing the sampling algorithm must put the samples in a local in-memory array, and transmit the keys in the array to the workers. The results for executions 2 and 3 (Table 3), demonstrate that transmitting the samples to the workers is also cheap. To conclude, Mario provides efficient reservoir sampling of datasets.

4.6 Mario Resource Usage

Mario servers are co-located on cluster work nodes with the bioinformatics applications doing the analysis. It is therefore important that Mario does not perturb these by using excessive CPU, memory, or network bandwidth. We use Ganglia to measure the aggregated resource usage of the Mario, HBase and HDFS servers. We initialized an HBase table with one million FASTA rows (2.5GB), and executed the dummy pipeline described in section 4.4 on eight Mario workers.

Our results show a peak load of 1.3 (of 8.0) for CPU load. This CPU usage is for processing I/O, and may be overlapped with tool I/O waiting time. The peak network usage is 30MB/s and aggregated usage of 900Mbps, both well within the capacity of our Gigabit Ethernet. In addition, most bioinformatics analysis tools are not communication intensive. The Mario servers have a small memory footprint. However, Mario relies on HBase region servers for which we had to allocate 12GB JVM heap space to get good performance. We believe the memory usage is acceptable since current compute cluster nodes typically have at least 32GB of DRAM.

5 Related Work

Frameworks such as Hadoop MapReduce [4], [5] have widely been adopted for large-scale data-intensive computing, including genomics data processing [22]. However, these frameworks are not well suited for low-latency data processing.

Our approach for tuning a data analysis pipeline was inspired by the Pig Pen debugging environment for Pig programs [23]. Pig Pen provides developers with a small dataset created by sampling from the real dataset, and by generating data that is similar to the real data. However, Pig is typically not used for biological data analysis.

The Galaxy [3] pipeline managers is similar to Mario in that the user can compose a pipeline from existing applications, but Galaxy supports the creation of more general execution graphs, as opposed to Mario's linear pipeline model. A linear model makes it easier to implement iterative data processing. In addition, Mario provides automatic parallelization.

Distributed systems for iterative, incremental, and interactive processing of large-scale datasets include systems designed for a specific application domain (such as Mahout [24] for machine learning), or programming model (such as HaLoop [8] for Hadoop MapReduce, and Dremel [7] for SQL queries). These are not suitable for Mario's data processing model. Instead general approaches and systems such as Spark [6], or Naiad [9] could be used to implement Mario processing. We also believe Mario should be integrated with a cluster resource management systems such as Mesos [25] to facilitate sharing of a cluster with other interactive and batch processing jobs.

6 Conclusion and Future Work

We have presented Mario, a system for iterative and interactive processing of large-scale biological data. Our approach allows users to interactively tune biological analysis pipelines, which is vital to find the tools and parameters that give insight to a particular biological dataset and problem. By quickly computing a result for a subset of the data that still gives meaningful biological insight, the user can overcome the limitation of the long-processing time of existing large-scale data analysis pipelines. Low-latency updates to pipeline results allows pipeline developers to make more parameter changes in order to improve the quality of the analysis results.

Our approach uses reservoir sampling to reduce the amount of data processed to achieve biological meaningful results. We enable interactive tuning by combining fine-grained caching of derived datasets, and an iterative, data-parallel processing model. This allows low-latency calculation of the first results, and in addition high-throughput incremental computation of the remaining results. Mario also offers integrated data provenance by storing detailed pipeline configurations associated with pipeline input, intermediate, and output data.

Although we applied Mario for a dummy pipeline, we believe it can be integrated with pipeline managers such as Galaxy [3], and visualization systems such as Krona [14] or METAREP [15]. Mario has an interface that allows writing a module that can aggregate and present data in the format required by visualization tools. Mario uses the GeStore [11] approach for adding transparent updates to pipelines implemented in for example Galaxy. We plan to improve the performance of Mario by implementing locality-aware scheduling by handling stragglers. We believe the Spark system [6] is well suited to solve these problems. We also plan to implement the Mario controller interface, and to add support for management of datasets for multiple pipelines.

Interactive tuning of biological analysis pipelines is vital for the development of solutions to future large-scale data analysis problems in the biological community. We have demonstrated the usability of our approach for pipeline parameter tuning, and we have experimentally evaluated the latency, throughput, and resource usage of the Mario system. We believe the Mario approach and system are useful not only for biological analyses, but potentially also for other data analysis disciplines.

Acknowledgements. Thanks to Jon Ivar Kristiansen for help setting up HBase, and to Bjørn Fjukstad, Giacomo Tartari, and Einar Holsbø for comments to the paper.

References

1. S. D. Kahn, “On the Future of Genomic Data,” Science (80-.)., vol. 331, no. 6018, pp. 728–729, Feb. 2011.
2. R. Bose and J. Frew, “Lineage retrieval for scientific data processing: a survey,” ACM Comput. Surv., vol. 37, no. 1, pp. 1–28, Mar. 2005.
3. J. Goecks, A. Nekrutenko, and J. Taylor, “Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.,” Genome Biol., vol. 11, no. 8, Jan. 2010.

4. “Hadoop homepage,” 2014. Available: http://hadoop.apache.org/.
5. J. Dean and S. Ghemawat, “MapReduce: a flexible data processing tool,” Commun. ACM, vol. 53, no. 1, Jan. 2010.
6. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing,” in Proc. of NSDI’12, Usenix, 2012.
7. S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, “Dremel: interactive analysis of web-scale datasets,” in Proc. VLDB Endow. 2010, vol. 3, no. 1–2, 2013.
8. Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, “The HaLoop approach to large-scale iterative data analysis,” VLDB J., vol. 21, no. 2, pp. 169–190, Mar. 2012.
9. D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, “Naiad: a timely dataflow system,” in Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles - SOSP ’13, 2013, pp. 439–455.
10. P. K. Gunda, L. Ravindranath, C. A. Thekkath, Y. Yu, and L. Zhuang, “Nectar: automatic management of data and computation in datacenters,” in Proc. of OSDI'10, Useinx, 2010.
11. E. Pedersen, N. P. Willassen, and L. A. Bongo, “Transparent incremental updates for Genomics Data Analysis Pipelines,” in Proc. of HiBB 2013, LNCS, vol 8374, 2014.
12. Apache HBase.. Available: http://hbase.apache.org/.
13. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “BigTable: A Distributed Storage System for Structured Data,” ACM TOCS, vol. 26, no. 2, pp. 1–26, Jun. 2008.
14. B. D. Ondov, N. H. Bergman, and A. M. Phillippy, “Interactive metagenomic visualization in a Web browser.,” BMC Bioinformatics, vol. 12, no. 1, p. 385, Jan. 2011.
15. J. Goll, D. Rusch, D. M. Tanenbaum, M. Thiagarajan, K. Li, B. A. Methé, and S. Yooseph, “METAREP: JCVI Metagenomics Reports - an open source tool for high-performance comparative metagenomics.,” Bioinformatics, vol. 26, no. 20, pp. 2631–2, Aug. 2010.
16. M. Ernstsen, “Mario - A system for iterative and interactive processing of biological data,” Master's thesis, University of Tromsø, 2013.
17. L. Sidirourgos, M. Kersten, and P. Boncz, “Scientific discovery through weighted sampling,” in 2013 IEEE International Conference on Big Data, 2013, pp. 300–306.
18. P. S. Efraimidis and P. G. Spirakis, “Weighted random sampling with a reservoir,” Inf. Process. Lett., vol. 97, no. 5, pp. 181–185, Mar. 2006.
19. E. Kjærner-Semb, “Exploring Bioinformatic Software for Taxonomic Classification of Metagenomes,”. Master thesis, University of Tromsø, 2013.
20. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic local alignment search tool.,” J. Mol. Biol., vol. 215, no. 3, pp. 403–10, Oct. 1990.
21. M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, “MapReduce and parallel DBMSs: friends or foes?,” CACM, vol. 53, no. 1, p. 64, Jan. 2010.
22. R. C. Taylor, “An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics.,” BMC Bioinformatics, vol. 11 Suppl 1, p. S1, Jan. 2010.
23. A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, “Building a high-level dataflow system on top of Map-Reduce: the Pig experience,” in Proc. of VLDB Endowment, vol. 2, no. 2, 2009.
24. Mahout homepage, 2014. Available: https://mahout.apache.org/.
25. B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, “Mesos: a platform for fine-grained resource sharing in the data center,” in Proc. of NSDI’11, Usenix, 2011.