Data-intensive computing infrastructure systems for unmodified biological data analysis pipelines

Lars Ailo Bongo[(1)], Edvard Pedersen[(1)], Martin Ernstsen[(2)*]

(1) Department of Computer Science and Center for Bioinformatics,
University of Tromsø, Tromsø, Norway
larsab@cs.uit.no, edvard.pedersen@uit.no

(2) Kongsberg Satellite Services AS, Tromsø, Norway
martin.ernstsen@ksat.no

Keywords: data-intensive computing, biological data analysis, flexible pipelines, infrastructure systems.

Abstract. Biological data analysis is typically implemented using a flexible data analysis pipeline that combines a wide array of tools and databases. These pipelines must scale to very large datasets, and therefore often require parallel and distributed computing. There are many infrastructure systems for data-intensive computing. Currently, most biological data analysis pipelines do not leverage these systems. An important challenge is therefore to integrate existing biological data analysis frameworks with data-intensive computing infrastructure systems.

We give an overview of data-intensive computing infrastructure systems, and describe how we have leveraged these for: (i) scalable fault-tolerant computing for large-scale data; (ii) incremental updates to reduce the resource usage required to update large-scale data compendium; and (iii) interactive data analysis and exploration. We provide lessons learned and describe problems we have encountered during development and deployment. Our results show that even unmodified biological data analysis tools can benefit from infrastructure systems for data-intensive computing.

1 Scientific Background

Recent advances in instrument, computation, and storage technologies have resulted in large amounts of biological data. To realize the full potential for novel scientific insight in the data, it is necessary to transform the data to knowledge through data analysis and interpretation.

Biological data analysis is typically implemented using a flexible data analysis pipeline that combines a set of tools and databases. Biological data analysis is diverse and specialized, so the pipelines have a wide range of resource requirements. Examples include the 1000 Genomes project [1] with a dataset of 260 TB analyzed on supercomputers or warehouse-scale datacenters; the many databases and servers built for a specific type of analysis; and simple analyses run using pre-defined Galaxy pipelines [2]. An important challenge when building data analysis tools and pipelines is therefore to choose a hardware platform and underlying data management and processing systems that satisfies the requirements for a specific data analysis problem.

A data analysis and exploration tool typically has an architecture with the following components:

- A front-end that provides the *user interface* used by the data analysts, including: web applications, pipeline managers, web services, and low-level interfaces such as file systems, and cloud APIs.

*Work done while at the University of Tromsø

- *Data analysis and interpretation services* including: specialized servers, search engines, R libraries such as BioConductor [3], and tool collections such as Galaxy Toolsheds [2].

- *Infrastructure system for data management* including: file systems, databases, and distributed data storage systems.

- *Infrastructure system for parallel and distributed computation* including: queuing systems, and the Hadoop software stack (http://hadoop.apache.org/).

- *Hardware platform*, such as virtual machines, dedicated servers, small clusters, supercomputers, and data warehouses.

In this extended abstract, we focus on the choice of data management and processing infrastructure systems. In our experience, most biological data analysis uses a POSIX file system in combination with a centralized database for data storage and management, and are run either on a single machine or on a small cluster with a queuing system such as Open Grid Engine (http://gridscheduler.sourceforge.net/). This platform has four main advantages. First, the file system-, and database interfaces are stable, and the technology is reliable. Second, many clusters for scientific computation are designed to use a network file system and to run jobs on a cluster using a queuing system. Third, the developers are familiar with these systems and interfaces. Fourth, there are already hundreds of tools implemented for this platform.

An alternative infrastructure must therefore provide better scalability, performance, or efficiency. Or, it must provide services not available on the standard platform. We give an overview data-intensive computing systems, and describe how these can be leveraged for biological data analysis. We describe how we used these systems to transparently extended flexible biological data analysis frameworks with data intensive computing services. We provide lessons learned, and describe the problems we have encountered during development and deployment of the extended pipelines.

2 Materials and Methods

There are several specialized infrastructure systems developed for data-intensive computing. Many of these were initially developed and deployed at web-scale companies such as Google, Yahoo, Facebook, and Twitter, and then later implemented as open source systems. There are also many new systems under development in both academia, the open source community, and the industry. We provide a short description of the features provided by these systems, and how we have leveraged these features for biological data analysis. To save space, we limit our description to the most used systems and omit many emerging systems, and systems that provide a traditional file system or SQL interface.

Data intensive computing systems are often built on a distributed file systems such as Hadoop Distributed File System (HDFS) [4] that provide reliable storage on commodity component hardware and high aggregate I/O performance. A HDFS cluster co-locates storage and computation resources to avoid the overhead of transferring large datasets over the network to and from the computation nodes. For large systems, it reduces cost since high-volume network attached storage is expensive. The main advantage of HDFS for biological data analysis is that the architecture is demonstrated to scale to peta-scale datasets and it is widely used for data-intensive computing. The main disadvantage is that HDFS does not provide a traditional file system interface, so it is necessary to either rewrite the many data analysis tools that use a POSIX file system interface, to incur an overhead for moving data between HDFS and a local filesystem, or incur the performance overhead of third-party library such as (http://wiki.apache.org/hadoop/MountableHDFS). In addition, it is not yet a common

platform in scientific computing, so it may be necessary to purchase and build a new cluster with storage distributed on the compute nodes.

MapReduce [4] is a widely used programming model and infrastructure system for data-intensive parallel computation. It provides fault-tolerant computation on a HDFS-like file system, and makes it easy to write scalable applications since the system handles data partitioning, scheduling, and communication. Biological data analysis applications, especially for next-generation sequencing data, have already been implemented using MapReduce ([5] provides examples). The main advantage of MapReduce is that it scales to peta-scale datasets. Also, most cloud platforms provide a MapReduce interface. The main disadvantage is that the MapReduce programming model may not be suited for all applications. It is a low-level programming model so many higher-level programming models have been developed to make it easier to write data analysis programs (but to our knowledge, these are not often used for biological data processing).

HBase (http://hbase.apache.org/) is a column based storage system that provides in-memory caching for low latency random data access and efficient compression. Biological data analysis applications can use HBase to store data accessed interactively, to implement custom data structures, or to structure data for more efficient compression. Compared to relational databases it is more difficult to implement queries in HBase. It is therefore common to a system built on top of HBase to execute queries. HBase also does not provide ACID properties, so these must also be implemented by another system.

An alternative for low-latency query processing is Spark [6]. It offers a richer programming model than MapReduce, including iterative operations. It is well suited to implement machine learning algorithms, and interactive data analytics. However, Spark is based on the Scala programming language, which may be unfamiliar to many developers. In addition, compared to the systems discussed above the Spark codebase is still immature, but it is now a top-level Apache project.

3 Results

We have extended several data-intensive computing systems to provide services for biological data analysis. In this section, we answer the following questions:

1. Why did we choose a particular system?
2. What problems did the system solve?
3. What where the main limitations of the systems for our use?
4. What were the lessons learned during development and deployment?

We have used three clusters for development and deployment of our systems over a period of five years. All were built for data-intensive computing with storage distributed on the compute nodes. We chose the Hadoop software stack, including HDFS for distributed storage. Hadoop is probably the mostly used data-intensive computing platform, and it has a very active development community. By using Hadoop, we have benefited from improvements to the infrastructure systems, and the addition of new infrastructure systems such as Spark to the ecosystem. We will give detailed achievements, issues, and lessons learned below.

3.1 Troilkatt

Troilkatt is a system for batch processing of large-scale collections of gene expression datasets. We use Troilkatt to process data for the IMP integrated data analysis tool [7]. We built Troilkatt in order to scale our gene expression dataset integration pipeline to process all datasets for several organisms in NCBI GEO. The pipelines comprise tools

Table 1: Our use of data-intensive computing systems for biological data processing

System	Problem	Solution	Issues
Troilkatt	Scale integrated analysis pipeline	HDFS: scalable storage MapReduce: I/O intensive pipeline processing	Memory management
GeStore	Incremental updates for analysis pipelines	HBase: data structures for generating incremental updates MapReduce: I/O intensive pipeline processing	Hadoop MapReduce job startup time
Mario	Tuning of analysis pipelines	HBase: sparse data structure, low-latency reads and writes	Performance tuning HBase

for data cleaning, transformation, and signal balancing of these datasets. The integrated data compendium for organisms such as human have ten thousands of datasets. The raw data, final results, and intermediate data in a compendium use tens of terabyte of storage space.

The pipeline processing is well suited for data-intensive systems since most pipeline tools are I/O intensive. The data is stored in HDFS. We use MapReduce for parallel processing and HBase for meta-data storage. We chose MapReduce since the processing must scale to several tens of terabytes of data. Since the datasets can be processed independently we use one Mapper task per dataset for each pipeline tool, and hence can process all datasets in parallel. MapReduce is well suited for many of the tools, since these process one row in a gene expression table at a time.

We achieved system that efficiently execute pipelines for processing integrated compendia datasets. We did not have to implement data communication between tasks, nor data locality aware mapping of tasks to compute nodes. Our main issue was large in-memory data structures in two pipeline tools. Hadoop MapReduce is run in a JVM, so the maximum heap size must be set at system startup time. The memory usage of the largest tasks therefore limits the number of tasks that can be run in parallel on each node. To achieve a good trade-off between memory usage and parallelism, we had to divide the expression datasets by their memory usage and processes similarly sized datasets together in a MapReduce job.

Since all datasets can be processed in parallel, and most organisms have hundreds or thousands of datasets, the parallelism in the pipeline exceeded the available compute resources even on the biggest 64-node cluster. We can therefore efficiently utilize a much larger cluster. We also used up all storage on the clusters, and therefore have to periodically delete raw data downloaded from public repositories. A full update of the dataset therefore requires re-downloading tens of terabytes of data from public repositories. We also have to share the clusters with other non-MapReduce jobs, and will therefore benefit from a cluster management system such as Mesos [8].

3.2 GeStore

GeStore [9] is a system for adding transparent incremental updates to biological data processing pipelines. We use GeStore to incrementally update large-scale compendia, such as the IMP compendia described in the previous section. GeStore is integrated with Troilkatt and Galaxy. We built GeStore since the processing time for a full compendium update can be several days even on a large computer cluster, making it impractical to frequently update all compendia. GeStore adds incremental updates to unmodified pipeline tools by manipulating the input and output files of the tools. A plugin framework provides services for parsing common genomic data file formats, implementing tool-specific input file generators, and output file mergers.

GeStore must detect changes in, and merge updates into, compendia that can be tens of terabytes in size. GeStore must also maintain and generate incremental updates of meta-databases, such as UniProt (http://www.uniprot.org/). These can be tens of gigabytes in size. We use HBase for data storage and MapReduce for generating input files and merging output files. HBase provides high-throughput random data accesses required for efficient change detection and merging. GeStore stores meta-database entries as HBase rows, and updates to these entries by creating a new version of an HBase table cell. The timestamps enable efficient table scans to find entries that have changed in a time period and hence are part of an incremental update. In addition, the flexible schema of HBase tables is used to reduce the work required to maintain plugins when file structures or databases changes, allowing several years of database versions to be stored in the same HBase table.

We achieved up to 82% reduction in analysis time for dataset updates when using GeStore with an unmodified biological data analysis pipeline ([9] has additional experimental results). We found HBase to be well suited for the data management requirements of GeStore. File generation and merging scales to large datasets since we use MapReduce for data processing. In addition, we reduce storage space requirements by storing multiple meta-databases versions in HBase instead of storing all versions separately.

The overhead introduced by GeStore is high for incremental updates of small datasets, since the startup time of Hadoop MapReduce jobs is tens of seconds. A system with lower startup time, such as Spark, will significantly reduce this overhead.

3.3 Mario

Mario [10] is a system for interactive iterative data processing. We have designed Mario for interactive parameter tuning of biological data analysis pipeline tools. For such interactive parameter tuning, the pipeline output should quickly be visible for the pipeline developer. Mario combines reservoir sampling, fine-grained caching of derived datasets, and a data-parallel processing model for quickly computing the results of changes to pipeline parameters. It uses the GeStore approach for transparently adding iterative processing to unmodified data analysis tools.

Mario must efficiently produce random samples from a stream for reservoir sampling, implement a cache of fine-grained pipeline tool results, and implement parallel pipeline stage processing. We use HBase as storage backend due to its low-latency random read and write capability, its ability to efficiently store sparse data structures, and its scalability. Mario stores all intermediate data records produced during pipeline execution in HBase, and uses the cached data to quickly find the data records that must be updated when pipeline tool parameters are changed. Mario also uses HBase for data provenance and single-pass reservoir sampling.

We achieved a system for iterative parallel processing that adds less than 100ms of overhead per pipeline stage, and that does not add significant computation, memory, or storage overhead to compute nodes (additional results are in [10]). We found HBase to be very well suited for efficiently storing and accessing the sparse data-structures used by Mario.

Our main issue was to configure HBase to achieve the required performance. We chose a configuration where HBase region servers allocate 12GB DRAM on the cluster nodes, and we traded reliability for improved write latencies by keeping write-ahead logs in memory (and periodically flushing these to disk). The latter is acceptable for Mario since the intermediate data stored in HBase can be recomputed at low cost, if required.

4 Conclusion

We have transparently extended flexible biological data analysis frameworks to utilize data intensive computing infrastructure systems. Our results show that even unmodified biological data analysis tools can benefit from these for: (i) scalable fault-tolerant computing for large-scale data; (ii) incremental updates in order to reduce the resource usage required to update large-scale data compendium; and (iii) interactive data analysis and exploration.

We have identified several limitations of the infrastructure systems we used. However, by using new systems recently added to the Hadoop ecosystem we can remove many of these limitations. We will therefore continue using these systems to extend biological data processing pipelines with new services for making data analysis more efficient and for improving the quality of the analysis results.

5 Acknowledgements

Thanks to Einar Holsbø, Giacomo Tartari and Bjørn Fjukstad for their comments and insights while writing this extended abstract.

References

[1] L. Clarke, X. Zheng-Bradley, R. Smith, E. Kulesha, C. Xiao, I. Toneva, B. Vaughan, D. Preuss, R. Leinonen, M. Shumway, S. Sherry, and P. Flicek, "The 1000 Genomes Project: data management and community access.," *Nature methods*, vol. 9, pp. 459–62, May 2012.

[2] J. Goecks, A. Nekrutenko, and J. Taylor, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.," *Genome biology*, vol. 11, p. R86, Jan. 2010.

[3] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. H. Yang, and J. Zhang, "Bioconductor: open software development for computational biology and bioinformatics.," *Genome biology*, vol. 5, p. R80, Jan. 2004.

[4] J. Dean and S. Ghemawat, "MapReduce - MapReduce simplified data processing on large clusters," *Communications of the ACM*, vol. 51, p. 107, Jan. 2008.

[5] 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.

[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 *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, (Berkeley, CA, USA), pp. 2–2, USENIX Association, 2012.

[7] A. K. Wong, C. Y. Park, C. S. Greene, L. A. Bongo, Y. Guan, and O. G. Troyanskaya, "IMP: a multispecies functional genomics portal for integration, visualization and prediction of protein functions and networks.," *Nucleic acids research*, vol. 40, pp. W484–90, July 2012.

[8] 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," *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, p. 22, 2011.

[9] E. Pedersen, N. Willassen, and L. A. Bongo, "Transparent incremental updates for genomics data analysis pipelines," in *Euro-Par 2013: Parallel Processing Workshops*, vol. 8374 of *Lecture Notes in Computer Science*, pp. 311–320, Springer Berlin Heidelberg, 2014.

[10] M. Ernstsen, *Mario. A system for iterative and interactive processing of biological data*. Master thesis, University of Tromsø, Nov. 2013.