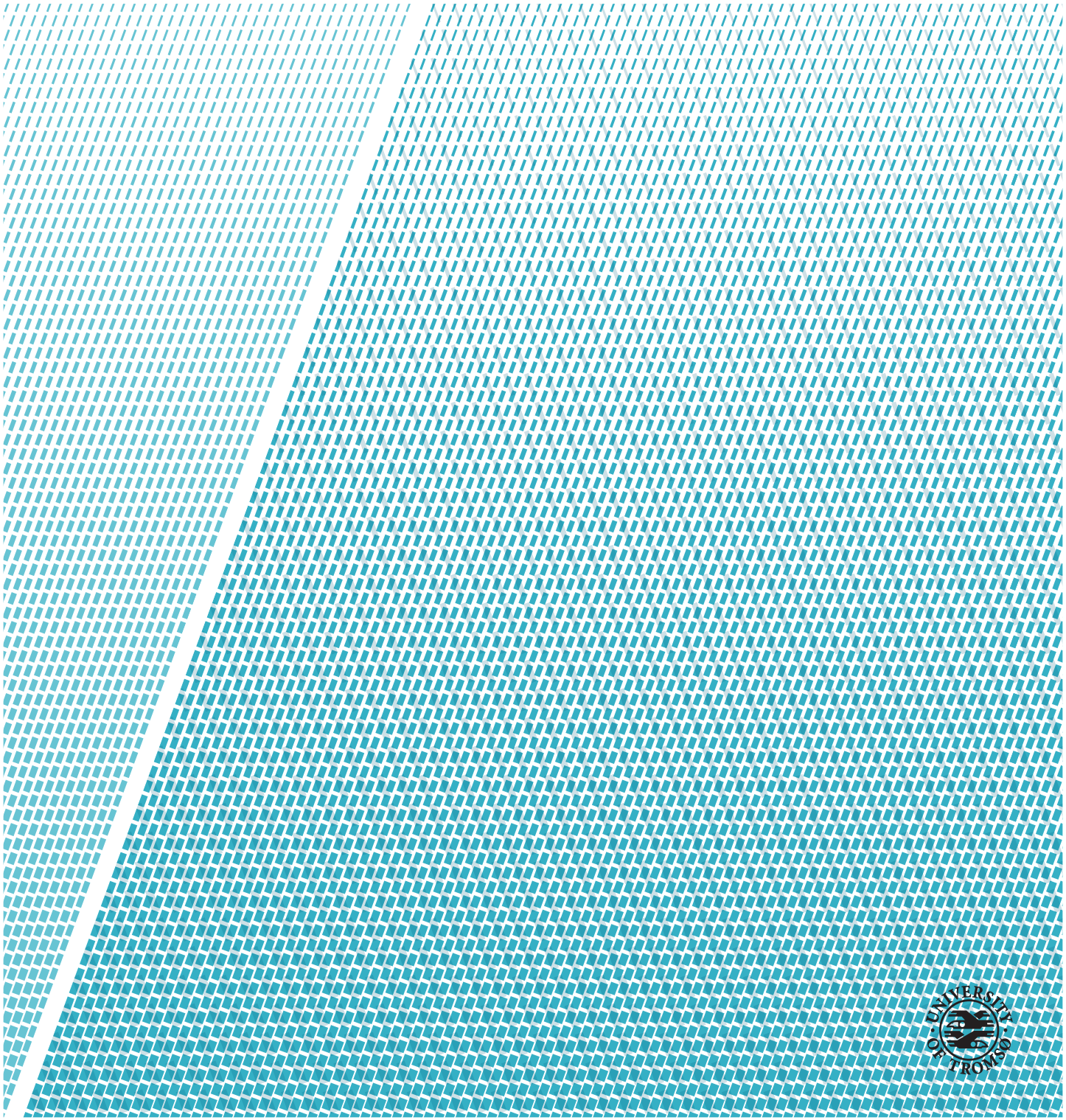
UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Department of Computer Science, Faculty of Science and Technology

# A Data Management Model For Large-Scale Bioinformatics Analysis

**Edvard Pedersen**

*A dissertation for the degree of Philosophiae Doctor*



# Abstract

Bioinformatics has seen an extreme data growth in later years due to the reduction in cost per megabase of sequencing, which today is around 1/400,000th of the cost in 2001. This reduction in cost enables new types of studies, such as searching for novel enzymes in marine environments using metagenomic approaches. However, it also leads to an increase in volume of data, which shifts overall cost from sequencing to analysis and data management. In addition, the data growth means that the analysis must move from personal computers to cluster, cloud and supercomputer infrastructure, which further complicates data management and processing.

This increase in data volume applies to both raw data produced as well as the size of reference databases. These reference databases are used in analysis to e.g. compare sequences to all the known sequences, so larger reference databases provide more accurate results. However, this increase in analysis accuracy also increases the volume of both input data and reference databases, which further increases analysis cost as well as the complexity of data management and processing.

In this dissertation, we examine the challenge of data management, particularly how existing bioinformatics analysis pipelines can reduce the runtime and hence the cost of analysis through a better data management approach. We provide the file-based distributed data materialization (FDDM) approach and realize it as the GeStore system to provide data management for real-world bioinformatics pipelines. The commonly used bioinformatics analysis frameworks do not provide efficient large-scale data management, in particular, updating analysis results with updated reference databases and reproducing previously computed results are costly and time-consuming. Technologies such as distributed databases and processing systems can efficiently process large amounts of data, but such systems are not straightforward to integrate with existing bioinformatics workflows since these workflows typically comprise legacy tools that are costly and time-consuming to port to new frameworks. Our approach bridges the gap between these by providing a simple file-based interface that makes it simple to integrate workflows using legacy tools with modern distributed databases and data processing frameworks.

We show the need for such a system through an evaluation of the tools of a bioinformatics pipeline that is provided as a data analysis service. Our results

show that the runtime of many of the most computationally intensive tools in the pipeline scale approximately linearly with input data size, so that runtime can be reduced by limiting the volume of data. We evaluate our implementation of the FDDM model using synthetic- and application benchmarks. Our results show that our implementation stores data efficiently with regards to storage space, and retrieves data quickly. We can therefore increase the speed of updates by up to 14 times. We integrate GeStore with three different workflow managers to demonstrate how popular workflow managers can easily use the FDDM approach.

*To Helene*

# Acknowledgements

I am deeply thankful to my advisors, Lars Ailo Bongo and Nils Peder Willassen, who have gone above and beyond in supporting my work.

In addition to my advisors, the work in this thesis is the product of the efforts of several people. Chapter 2 is a collaboration with the members of the Center for Bioinformatics: Espen Mikal Robertsen, Tim Kahlke, Inge Alexander Raknes, Aleksandr Agofonov, Giacomo Tartari and Erik Hjerde.

Further support has been given by the technical staff, department and my fellow students. In particular I would like to mention the members of the Biological Data Processing Systems lab: Bjørn Fjukstad, Einar Holsbø, Morten Grønnesby and Jarl Fagerli. Jon Ivar Kristiansen has been tireless in both providing technical support, as well as being a central administrator of our hardware resources.

Special thanks to Otto Anshus, Bjørn Fjukstad, Espen Mikal Robertsen and Inge Alexander Raknes for their invaluable comments and feedback on this thesis.

I would like to extend my thanks to my family, which has been an incredible support.

Thanks to the ELIXIR project for providing me with a large network of people to exchange ideas and opinions with, this has given me a much better understanding of the ramifications of this work.

# Contents

# List of Figures

# List of Tables

# Part I

# Thesis

# 1

# Introduction

The past few years have transformed biology into a data science [1], as new instruments are producing data that require more complex data analysis. In both commercial and life science fields, new data sources and instruments such as sequencing machines are producing rapidly increasing amounts of data, as seen in figure 1.1. This development has led to growth in the size of reference databases, which are collections of the current knowledge in bioinformatics, as well as input data. Data management and processing has become a major limitation of existing biological data analysis frameworks. As a result, more and more analysis projects have to expand or redesign their analysis pipelines to support efficient distributed large-scale data management and processing.

This development has led to the emergence of three core challenges in bioinformatics: (i) Data management that enables efficient processing of large amounts of data in a way that ensures reproducibility of experiments, while enabling researchers to update experiments when reference databases are updated without the cost of a full update. (ii) The need to use unmodified tools for analysis, since it would require a large effort in keeping modified tools updated. (iii) Efficiently utilizing infrastructures such as clusters, cloud and supercomputers to provide the required I/O throughput, elasticity and scalability to analyze data quickly and cost-efficiently.

Existing systems solve some of these challenges, the following systems represent data management approaches currently in use in bioinformatics. The Galaxy workflow manager [2] manages data sets, but does not support distribution of the data beyond network file systems (NFS), and does not

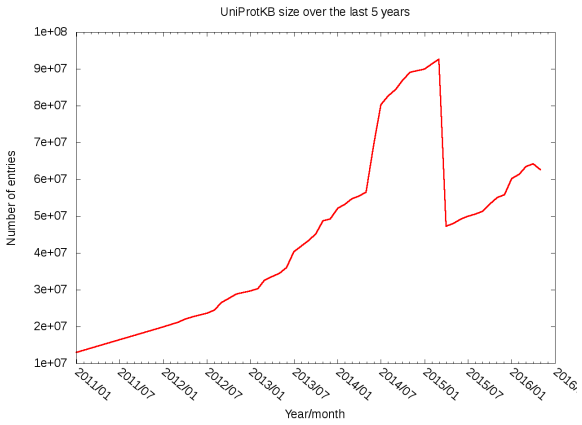



**Figure 1.1:** Growth of data in the UniProtKB reference database, the dip in early 2015 is due to the removal of redundant proteomes.

enable efficient distributed data management. The ADAM genomics pipeline [3] and the genomics pipeline described in [4], integrate the entirety of the pipeline with respectively the Spark [5] and MapReduce [6] frameworks. They also use distributed storage, but require re-implementation of the pipeline in their respective framework, which conflicts with the need to not modify legacy tools.

We propose the file-based distributed data materialization (FDDM) model for large-scale data management for biological data analysis frameworks. This model is centered around the idea of file-based on-demand generation of reference databases, input- and intermediate data. This model is based on five core ideas: (i) a file-based interface ensures that no modifications to tools are required; (ii) transparent incremental updates keep results up to date at low cost; (iii) the ability to generate any previous version of reference databases ensures reproducibility of analysis; (iv) simple query facilities enable filtering of reference databases to reduce data volume; (v) building on existing large-scale data processing systems allows this approach to utilize a wide range of hardware.

We have implemented this model in the GeStore system. We have integrated this system with three state of the art bioinformatics workflow managers. We have evaluated the performance of one of these workflow managers with and without GeStore. Our implementation provides up to a 14-time speedup in analysis time for updating results with unmodified tools using incremental updates. Query facilities allows researchers to tune reference databases to fit the analysis, reducing execution time. Versioning ensures that experiments can be performed with specific versions of reference databases.

Taking the long view, we also discuss the use of large-scale data management and processing systems in bioinformatics. As biological analysis grows in complexity and data volume, we believe that the file-based distributed data

materialization model used in GeStore will be increasingly important for efficient execution of production pipelines that provide up-to-date repeatable results. In the remainder of this thesis, we investigate this, framed by the thesis statement.

**Thesis Statement:** *A data management approach based on the file-based distributed data materialization model can be leveraged by existing bioinformatics pipelines to reduce runtime, keep results up to date and maintain reproducibility.*

## 1.1 Problems With Data Management in Bioinformatics

The current state of the art does not adequately solve the challenges of data management in bioinformatics.

Current popular biological data analysis frameworks such as Galaxy [2], Taverna [7, 8], and scripts using the packages in Bioconductor [9] require the user to manually maintaining and specifying reference database versions. In addition, reference database updates typically require re-executing the analysis for each metadata update. Such full updates increase the computational cost, often to the point where reanalysis is not done.

Incremental update systems [10] for large-scale data [11, 12, 13, 14, 15, 16] maintain several versions of the experiment data compendia and reference databases, and greatly reduce the cost of reanalysis by using incremental updates that limits the computation to new and updated data. However, they do not provide a transparent approach for adding incremental updates to existing biological analysis workflows. Instead, they require either porting applications to a specific framework (such as Dryad [17], MapReduce [6], or Spark [5]) or implementing ad hoc scripts for input generation and output merging.

Data warehouse approaches for biological data, such as Turcu et al [18], may provide incremental updates for specific tools, but do not easily allow adding new tools, nor integrating with biological data analysis frameworks

Data management in ad-hoc or script-based pipelines [19] often require manual work to update data such as reference database versions. This process is error-prone and requires a large effort from the pipeline maintainer.

A common issue is that the analysis of experimental data is inefficient due to lacking query facilities. For example restricting the query space to a specific biological kingdom or domain (e.g. human or marine) is not commonly supported in popular biological analysis pipelines [20, 21]. This leads to unneeded data being included in the analysis, which again leads to increased analysis time.

Summarized, the above systems all have one or more of the following issues:

1. *Manual maintenance:* Maintaining up-to-date compendia of reference databases, input files and intermediate data is time-consuming for the maintainer, in particular when the reference databases are frequently updated.
2. *Complicated integration:* Integration between a pipeline manager and the underlying data management system is work-intensive for the developer if large changes are required in the workflow management system.
3. *Inefficient filtering:* Filtering data to reduce the scope of analysis is frequently done through stand-alone filtering tools, which increases processing time due to increased data shuffling on disk.
4. *Requires changes to tools:* Tools must be modified to accommodate a new processing or storage framework, this leads to a sharp increase in the amount of work required, since each pipeline may use tens of different tools that are also updated periodically.

## 1.2 The File-based Distributed Data Materialization Model (FDDM)

To solve the above issues, we propose the file-based distributed data materialization (FDDM) model. It bridges the gap between legacy biological analysis tools and modern large-scale data management systems by leveraging these systems on the back end, while presenting a simple file generation interface to the workflow manager.

Previous systems are unsuited since commonly used workflow systems such as Galaxy use files as the primary abstraction for data, while large-scale data management systems such as Cassandra use tables as the primary abstraction for data. The FDDM bridges this gap by providing a simple interface to generate files from data in tables.

The features we have identified as particularly useful are

1. *Incremental updates of results:* Updating results through only updating the relevant subsets of data reduces computational costs, we do this through incremental updates.
2. *Filtering of data:* Removing unneeded data by only using specific data groups reduces computational cost. We do this through the filtering system.
3. *On-demand generation of versioned reference databases:* Reproducing experiments requires using the same version of reference databases. By

producing these on demand, experiments can be reproduced on demand.
4. *Provenance recording:* Conserving provenance is needed to ensure reproducibility, we do this through automatically and uniquely identifying and storing the data sets used in an experiment.
5. *Post-processing of data with legacy tools:* Incremental updates can introduce inaccuracies in the results due to a mismatch between the real reference data size and the incremental data size. We facilitate this through the plugin system.

## 1.3 Use of FDDM

We have integrated our implementation of FDDM, GeStore, with three workflow managers, using three distinct approaches.

**Direct Integration.** Where we use a command-line interface for GeStore to produce files for tools in the workflow manager. We used this approach for a script-based META-pipe workflow manager [20]. Direct integration provides transparency for the user while the workflow manager can utilize the features of GeStore fully, but requires larger changes to the workflow manager than the other two approaches.

**File System Integration.** Where we use an HDFS-like interface, so that GeStore can serve as a distributed file system. This approach was used for the Hadoop-based IMP pipeline [22]. This also enables the workflow manager to control GeStore through path- and file names, and is well suited to workflow manager that already use a distributed file system. The drawbacks of this approach is that it is difficult to fully utilize the features of GeStore without introducing further changes into the structure of the file storage, such as changing directory structures and naming schemes.

**Tool-based Integration.** Where we integrate GeStore into a workflow as a standalone tool executed before and after the other tools in the pipeline. This approach was used for the Galaxy workflow manager [2]. This allows the end-user full control over the use of GeStore, so that the use of the GeStore features can be tailored to each pipeline. The drawbacks of this approach is that the user is responsible for how GeStore is used, which requires the user to have knowledge of how to use GeStore for a pipeline.

## 1.4 Summary of Results

Our performance evaluation of a real-world pipeline with and without GeStore, shows the potential for large savings in analysis time, on the order of 14 times speedup for monthly updates, when compared to updating results by re-running

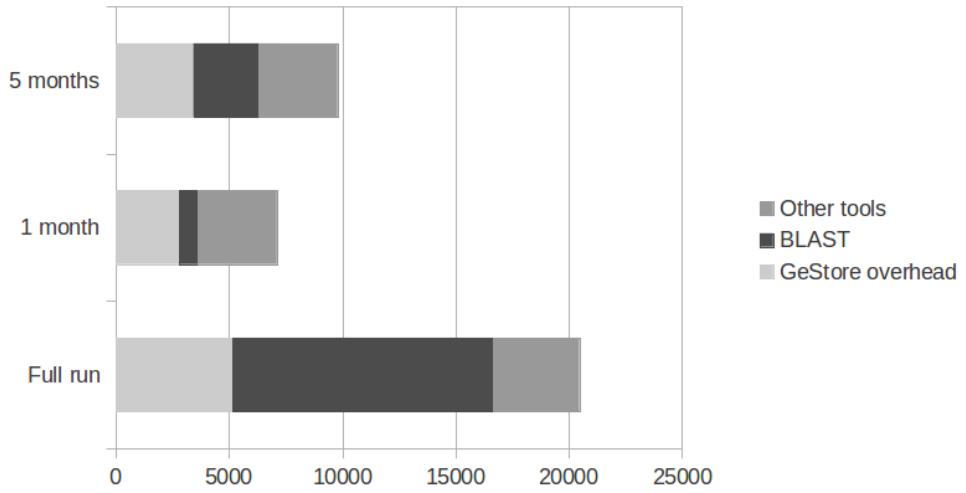



**Figure 1.2:** META-Pipe runtime for a full update, as well as a one-month and five-month incremental update. The focus here is on BLAST, which is the most computationally expensive part of META-Pipe in this configuration.

experiments fully when reference databases are updated (figure 1.2).

Our evaluation of GeStore, where we investigate the performance characteristics of individual operations in GeStore, shows that the system has low overhead, and fully utilizes the aggregate bandwidth in our cluster. It is held back primarily by the post-processing required for some legacy tools, such as converting reference databases to a binary format. In addition, we also demonstrate that the built-in caching functionality in GeStore can further reduce overhead.

Our evaluation of a real-world pipeline, where we evaluate the scalability on input data size of individual tools in the pipeline, shows that the largest contributors to the runtime of the pipeline are tools that scale well with regards to data size, and as such can benefit from using GeStore to reduce the volume of data.

Our integration of GeStore and three real-world workflow managers, where we have used three distinct approaches to enhance workflow managers with support for GeStore, shows that integration can involve few lines of code (60-300), and as such demonstrates that GeStore can be used easily in different workflow managers.

Together, these results show that the GeStore system for data management can be leveraged by existing bioinformatics pipelines to reduce runtime and keep results up to date.

## 1.5 Included Papers

In this section, I will give a short overview, and list my own contributions, to each paper included in this thesis in the order they are discussed. For each paper, I have contributed with writing.

### 1.5.1 Paper 1

| | |
|---|---|
| Title | META-pipe – Pipeline annotation, analysis and visualization of marine metagenomic sequence data. |
| Authors | Espen Mikal Robertsen, Tim Kahlke, Inge Alexander Raknes, Edvard Pedersen, Erik Kjærner Semb, Martin Ernstsen, Lars Ailo Bongo, Nils Peder Willassen. |
| Description | This paper describes the biological context, design and implementation of META-Pipe 1.0, and an experimental evaluation of the pipeline tools[1]. |
| My contribution | The experimental design, execution and result analysis, and presentation of META-Pipe performance. |
| Publication date | 14.04.2016 |
| Publication venue | Archived in arXiv. |
| Citation | [20] Robertsen, E.M., Kahlke, T., Raknes, I.A., Pedersen, E., Semb, E.K., Ernstsen, M., Bongo, L.A., Willassen, N.P.: Meta-pipe - pipeline annotation, analysis and visualization of marine metagenomic sequence data (2016) arXiv:1604.04103 |

### 1.5.2 Paper 2

| | |
|---|---|
| Title | Large-scale biological reference database management. |
| Authors | Edvard Pedersen, Lars Ailo Bongo. |
| Description | This paper describes the GeStore approach to data management and the design of the GeStore system, as well as providing an experimental evaluation [2]. |
| My contribution | I developed GeStore and did the experimental design, execution and evaluation of GeStore. |
| Publication date | In press. |
| Publication venue | Future Generation Computer Systems. |
| Citation | [25] E. Pedersen and L. A. Bongo, "Large-scale Biological reference database Management," in *Future Generation Computer Systems*, in Press. |

### 1.5.3 Paper 3

| | |
|---|---|
| Title | Integrating data-intensive computing systems with biological data analysis frameworks. |
| Authors | Edvard Pedersen, Inge Alexander Raknes, Martin Ernstsen, Lars Ailo Bongo. |
| Description | This paper describes three approaches for integrating GeStore with workflow managers. |
| My contribution | The integration between GeStore and several workflow managers. |
| Publication date | 06.03.2015 |
| Publication venue | 23rd Euromicro International Conference on Parallel, Distributed and Network-based Processing. |
| Citation | [26] E. Pedersen , I. A. Raknes, M. Ernstsen , and L. A. Bongo, "Integrating Data Intensive Computing Systems with Biological Data Analysis Frameworks," in *Proc. of 23rd Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2015, pp. 733–740. |

### 1.5.4 Paper 4

| | |
|---|---|
| Title | Data-intensive computing infrastructure systems for unmodified biological data analysis pipelines. |
| Authors | Lars Ailo Bongo, Edvard Pedersen, Martin Ernstsen. |
| Description | This paper summarizes our experiences using the Hadoop software stack in several projects, as well as a literature review which gives an indication of the use of large-scale data management systems in bioinformatics. |
| Mye contribution | I have contributed the experiences related to GeStore, as well as contributing to the literature review. |
| Publication date | 26.06.2014 |
| Publication venue | Computational Intelligence Methods for Bioinformatics and Biostatistics. |
| Citation | [27] L. A. Bongo, E. Pedersen, and M. Ernstsen, "Data-Intensive Computing Infrastructure Systems for Unmodified Biological Data Analysis Pipelines," in *Computational Intelligence Methods for Bioinformatics and Biostatistics*, ser. LNBI, vol. 8623, 2014. |

### 1.5.5 Paper 5

| | |
|---|---|
| Title | Big biological data management. |
| Authors | Edvard Pedersen, Lars Ailo Bongo. |
| Description | This non-peer reviewed book chapter gives an overview of large-scale data management for bioinformatics, challenges and approaches. |
| My contribution | I wrote this book chapter. |
| Publication date | In press. |
| Publication venue | Computational Intelligence Methods for Bioinformatics and Biostatistics. |
| Citation | [28] E. Pedersen and L. A. Bongo, *Resource Management for Big Data Applications*. Springer, 2016, ch. Big Biological Data Management. In Press. |

## 1.6 Dissertation Plan

This thesis is organized as follows. Chapter 2 gives an overview of the current state of the art and challenges of data management in bioinformatics. Chapter 3 describes our approach to data management for bioinformatics and the GeStore system. Chapter 4 details the different strategies used to integrate GeStore with different legacy workflow managers and execution environments. We conclude in chapter 5 and we finally outline future work in chapter 6.

# 2

# Biological Data Analysis Pipelines

In biology, data is typically analyzed by processing through a pipeline consisting of tools, where the output of each tool serves as the input for the next tool. These pipelines are created and managed in workflow managers, which also execute the pipelines on diverse execution environments, such as supercomputers or cloud infrastructure. Different types of data are used throughout the pipeline, such as input-, reference-, intermediate-, and output data. Visualizations of results, created from pipelines, are usually interpreted by domain-specific scientists [2]. We describe this approach briefly, with some details about the tools used in this process in relation to data management and processing.

We have implemented one such pipeline in the form of META-pipe [20][1], which we use as a case study. We describe our experiences in the development of this pipeline, and put it into a broader context within bioinformatics in relation to data management. The META-pipe pipeline is motivated by (i) providing a specialized analysis pipeline for marine metagenomics[2] for national and international users through the NeLS [29] and ELIXIR [30] projects, respectively; (ii) providing a pipeline that is used to create a marine reference

1. META-pipe was initially developed by Tim Kahlke, then improved by Espen Mikal Robertsen, Inge Alexander Raknes, Giacomo Tartari and Aleksandr Agafonov.
2. Note that META-pipe is not exclusively used for marine metagenomics, but in the interest of simplicity, we focus on marine metagenomics in this thesis.

database. These use cases require a scalable and optimized pipeline, therefore we have closely investigated the scalability and performance of tools, infrastructures and frameworks. The results of this evaluation are summarized in this chapter.

## 2.1 Background

A computer system for analyzing biological data typically consists of four main components: input data, reference databases (i.e. the databases used in the analysis), a set of tools in a workflow, and finally output data for interactive analysis. Biotechnology instruments such as short-read sequencing machines produce the input data. The input data can also be downloaded from public repositories such as GEO [31] and ENA [32]. There are hundreds of reference databases with human or machine curated meta-data extracted from the published literature and analysis of experimental data [33]. The datasets and databases 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 for the next tool. The data transformations includes file conversion, data cleaning, normalization, and data integration. A specific biological data analysis project often requires a deep workflow that combines many tools [4]. There are many libraries [2, 9, 34] with hundreds of tools, ranging from small, user-created scripts to large, complex applications.

To summarize, the typical analysis of biological data involves the following steps:

1. Retrieve the sample (e.g. collecting sediment from the sea floor).
2. Analyze the sample using specialized hardware (e.g. a sequencing machine).
3. Perform quality control on the data produced in analysis.
4. Run the data through a pipeline to produce the output data the researcher wants.
5. Interpret the output data.

### 2.1.1 Workflow managers

The analyst specifies, configures, and executes the analysis pipeline using a workflow manager [19]. The workflow manager provides a way of specifying the tools and their parameters, management of data and meta-data, and execution of the tools. In addition, a workflow manager may enable data analysis reproducibility by maintaining provenance data such as the version and parameters of the executed tools. It may also maintain the content of input data

files, reference databases, output files, and possibly intermediate data.

A workflow manager may comprise a set of scripts run on a specific platform, or a system that maps high-level workflow configuration to executable jobs for many platforms. There are also managers [2, 7] that provide a graphical user interface for workflow configuration, and a backend that handles data management and tool execution.

The two most popular of these are Galaxy [35, 2] and Taverna [7, 8]. Both provide intuitive interfaces, in which users can create their own pipelines as well as tune parameters in a graphical interface. The server for Galaxy and Taverna run on a single machine. If the lab does not use the fat server infrastructure exclusively, jobs that are executed through the workflow manager are executed on the infrastructure remotely.

In addition, many labs run pipelines through their own specialized workflow managers [19]. The complexity and feature sets of these workflow managers vary greatly. However, it allows the developers to provide a more streamlined interface that is tailored to their pipeline.

### 2.1.2 Data Management

In this chapter, the following definitions for different types of data are used: (i) Contextual data, which is information about the samples, such as where and when they were collected; (ii) Meta-data, which includes provenance information, and descriptions of machines and tools used, and (iii) Data, which includes the input data, intermediate analysis data, and output data.

Typically in bioinformatics, data is managed by workflow managers by packaging reference databases with tools, and versioning them together (such as in [36]). This allows provenance to be preserved, as the tool version and the reference database are uniquely identifiable, allowing the analysis to be repeated at a later date. Contextual information is generally managed by the user, with some exceptions (including the European Nucleotide Archive (ENA) [32] that collects contextual data when the researcher submits data).

The intermediate data is sometimes presented to the user as part of the optional output. Other times, the intermediate data is deleted as soon as it is used by the next step in the pipeline as a means to save storage space. This is a common tradeoff between how much storage space some results need, versus how long it takes to compute those results.

Output data as well as metadata such as tool and database versions are conserved for the user. It can be archived by the analysis service (such as ENA) to enable future replication and comparisons to new experiments. The policy for how long to keep this data, and if it is archived, is enforced by the workflow manager or administrator of the service.

## 2.2 Overview of Analysis Workflow

In this section, we give an overview of how data is typically collected, processed and analyzed in biology in the context of metagenomics, from sampling to visualization. We focus on the data management and analysis, and as such only give a brief overview of the other steps involved.

### 2.2.1 Sampling

For many metagenomics studies, data analysis starts by collecting samples that consist of soil, water or other media that contains living microorganisms directly from the environment.

There are several standards for meta-data that should be recorded to document metagenomic samples. M2B3 [37] gives an overview of the required metadata annotations for metagenomic samples, such as how to record position, collection equipment and environment.

### 2.2.2 Sequencing

Once the samples are collected, they are processed in the lab for isolation of DNA and construction of sequencing libraries. The DNA libraries are sequenced using sequencing machines, which range from small-scale sequencing machines such as the 454 Sequencing GS Junior which produces up to 35 megabases per run, to high-throughput machines such as the Illumina HiSeq X Ten, which produces up to 1.8 terabases per run.

The sequencing machines usually include a pipeline, which is executed to produce machine-readable FASTQ files, which contains the raw DNA sequence as well as quality information, from the huge number of image files used internally in the machine. These pipelines are vendor-specific, so the output from the machines is relatively uniform and easy to manage.

The sequences that are the output from modern sequencing machines tends to be quite short (such as the Illumina HiSeq X, which produces 2x150 base pair reads). The short read length may be a challenge when attempting to classify which gene or genome a sequence belongs to.

In addition to the output size and read length being different, each technology has its own peculiarities, such as "primer sequences", which may have to be removed from the resulting sequences before they can be analyzed. The quality of different parts of the reads may also vary, with some machines having a lower confidence for each base pair at the start or the end of the sequence.

### 2.2.3 Data Analysis

After the sequencing machines have generated the machine-readable reads, these files are used as input to an analysis pipeline, which consists of a sequence of tools, where the output of one tool acts as the input of the next. The META-pipe pipeline is described in detail in section 2.3.1.

The stages of the pipelines vary by the analysis used, and even within types of analysis. For example the EBI Metagenomics [38] pipeline differs from META-pipe in that it does not do assembly of reads into longer contigs before analysis.

### 2.2.4 Data Exploration

The results of the data analysis are interpreted by an analyst. The data is visualized in a way that a human can interpret through software such as METARep [39], Krona [40] and many others.

The visualization software is often integrated into the pipeline, so that the output of the pipeline is visualized directly in addition to the machine-readable analysis results.

Data interpretation and visualization is not a focus in this thesis, and as such will not be discussed as part of the performance metrics or other evaluations.

### 2.2.5 Data Archiving

When publishing results in biology, it is often a requirement that the data is accessible, as well as the metadata and the output data from the analysis pipeline [41].

The data used in the analysis is usually archived in repositories such as ENA [32] after analysis is complete. This includes both the raw data, as well as metadata and contextual data such as is described in section 2.1.2.

In this thesis, we have not focused on data archiving, since our focus is on data management during analysis.

## 2.3 META-pipe

META-pipe is a pipeline for analyzing and annotating metagenomics data. It is developed at the Center for Bioinformatics (SfB) at UiT - The Arctic University of Norway, as a part of the ELIXIR [30] project.

The goal of the META-pipe pipeline is to be the world-leading pipeline

for marine metagenomics analysis, both functional and taxonomic. META-pipe is currently deployed as a national service, and is being deployed as an international service.

Previous analysis resources have demonstrated their usability for metagenomics data analysis, including EBI-Metagenomics [42, 43], Metagenomics-Rapid Annotations using Subsystems technology (MG-RAST) [44] and Integrated Microbial Genomes and Metagenomes (IMG/M) [45]. However, these are not developed for the marine metagenomics domain and do not offer the extensive annotation options, flexibility and visualization needed to select interesting biological targets for further investigation. In particular, there is a need to produce full-length annotated genes from metagenomic assemblies. In addition, these are typically run on a server administered by a single organization, often resulting in scalability problems for free to use resources, or costly fees for pay to use services. There is therefore a need to develop a scalable pipeline for the marine metagenomics field. To ensure fast development, scalability to flagship projects, and easy deployment of the pipeline, it should utilize existing frameworks and infrastructure resources and services when possible.

To support many users and large-scale data, META-pipe utilizes supercomputing resources at the UiT - The Arctic University of Norway, as well as integrating with existing workflow managers to adapt the national infrastructure. In addition cloud systems are required to provide elasticity when faced with a large number of concurrent users and large datasets. Initial efforts were focused into integrating META-pipe with Norwegian infrastructure resources. We are integrating META-pipe with compute platforms (such as EGI Federated Cloud [46], EMBL Embassy Cloud [47], CSC cPouta [48]) and storage resources (such as EUDAT [49]) provided in the ELIXIR infrastructure. We use ELIXIR services such as AAI for authentication, data transfers to collaborate with other ELIXIR nodes on data storage, and tool registries to coordinate our service with others.

A prototype of the next version of META-pipe has already been created, but not published. This involves a major redesign to take advantage of more possible computing infrastructures at the same time. This has been done to increase the flexibility of the pipeline, as well as reducing the amount of work required to maintain compatibility with different computing infrastructures.

The prototype has been written in Scala and Java, utilizing the Spark framework for job execution and parallelism. It has not been used for evaluation in this thesis since it was not ready for use at the time the experiments were done.

### 2.3.1 Analysis Tools

In our work, we differentiate between four classes of tools: (i) data transformation tools that convert data from one file format to another, generate sequence

files from offset files, and other tasks which primarily involve reading and writing data. (ii) quality management tools, such as FastQC [50] and Prinseq [51], which help to evaluate the quality of data. (iii) data comparison tools, such as BLAST [52], HMMer [43, 36] and Priam [53], which compare the input data with reference databases to produce some output, such as finding the most similar annotated sequence to a given input sequence. (iv) data processing tools, which use a rule set to process the input data to produce a new data set, examples include assembling reads into longer contigs with e.g. Mira [54, 55] or MetaRay [56, 57] and predicting genes in a contig with tools including MetaGeneAnnotator [58, 59] or Glimmer [60, 61].

The tools used in META-pipe are summarized in table 2.1. The data processing tools that assemble reads into longer contigs used in META-Pipe are MetaRay and MIRA. The alternatives to these are somewhat limited, but include ABySS [62], and Spaler [63]. A limitation with these tools is that they require large amounts of memory (hundreds of GB), and are notoriously difficult to parallelize [3].

For quality assessment, FastQC is currently used as a manual step in META-Pipe. We would like to automate this step, as well as add quality management to additional stages of the pipeline. This is a challenge since we are not able to detect all errors with FastQC or similar software, due to the highly heterogeneous data quality requirements for different analysis types. We are currently investigating alternatives and research in this area.

The tools contributing most to the execution time of the pipeline are the comparison of input data to reference data. This is done with the Basic Local Alignment Search Tool (BLAST) [52], PRIAM [53] and InterProScan [36]. These tools generally scale very well, and therefore we are not actively searching for replacements of these.

Finally, the locally developed data transformation tools, which have already been described, are simple enough to be replaced by simple Scala code in the next version of META-Pipe, not much effort has been put into finding alternatives for these.

It should be noted that the parallelizability of the tools vary greatly, and for us this has dictated the choice of tools.

### 2.3.2 Workflow Manager

The design of META-pipe 1.0 builds on the design of the GePan pipeline [64]. This design has been extended to incorporate more tools, as well as run on HPC infrastructure.

The pipeline is started through the Galaxy workflow manager interface by

3. The new version of META-pipe does not use a parallel assembler, since we have not found one which performs well enough.

| *Tool name* | *Type* | *CPU* | *RAM* | *Network* | *Disk* |
|---|---|---|---|---|---|
| Mira | Processing | High | High | None | Low |
| Ray | Processing | High | High | High | Low |
| MetaGeneAnnotator | Processing | Low | Low | None | Low |
| Glimmer | Processing | Low | Low | None | Low |
| BLAST | Data comparison | High | Low | None | Low |
| Priam | Data comparison | Low | Low | Low | Low |
| HMMer | Data comparison | High | Low | None | Low |
| Data transformation | Data transformation | High | Low | High | High |

**Table 2.1:** META-Pipe tools.

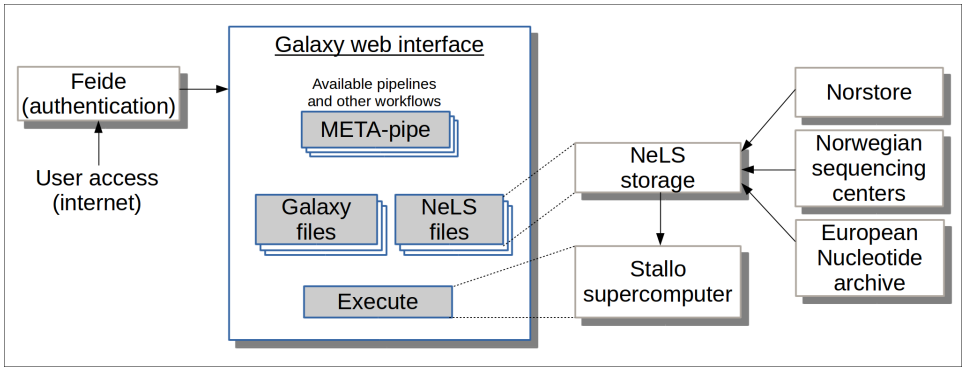



**Figure 2.1:** META-pipe tool architecture

the user. META-pipe runs the pipeline by generating scripts for each tool in the pipeline. These scripts manage the data, as well as executing the tools. These scripts are sent to the computing infrastructure (in this case Stallo). The data is then copied back to the Galaxy instance, and visualized to the user by way of Metarep and Krona.

Galaxy was chosen as the common user interface in the Norwegian e-Infrastructure for Life Sciences (NeLS) project, since we believe it is the most popular, and hence most familiar, workflow manager for bioinformatics. In addition, some of the partners in the NeLS project already had previous experience using Galaxy for the Lifeportal [65].

## 2.4 Hardware Infrastructures

As the tools described in section 2.3.1 have heterogeneous requirements for efficient operation, the choice of hardware infrastructure can have an impact on the performance and cost of an analysis pipeline, which is vital due to the data growth in bioinformatics. In this section we present a short overview the

| *Infrastructure* | *Elasticity* | *Cost* |
|---|---|---|
| Fat server | None | Up-front |
| Lab Cluster | Long-term | Up-front |
| Supercomputer | Immediate | Varies |
| Cloud | Immediate | Pay-as-you-go |

**Table 2.2:** Comparison of hardware infrastructures. Elasticity refers to the speed of scaling out.

most popular hardware infrastructures used for biological data analysis, this is summarized in table 2.2.

### 2.4.1 Fat Server

A fat server is a term used to describe a high-performance single machine, which may have more RAM, disk space and CPUs than a desktop computer.

Many tools in bioinformatics were originally designed to run on a single machine, and as such are optimized around a single high-performance machine. A typical task for this type of machine is assembly of reads, which is difficult to parallelize efficiently across multiple computers and has very large memory requirements (hundreds of gigabytes).

The advantages of the fat server are that the up-front cost can be relatively low, and jobs can run for longer than on a personal computer.

The disadvantages of this approach is that it is not possible to scale out to more machines to respond to increasing demands on the infrastructure. In addition, the machine must be maintained, so the aggregate costs can grow to be relatively large over time.

### 2.4.2 Lab Cluster

A lab cluster consists of multiple machines in a local area network (LAN), typically commodity machines, which allows more aggregate performance than a single machine.

The cluster usually uses consumer-grade components, such as relatively low-speed (gigabit) Ethernet, consumer CPUs and a modest amount of RAM.

The advantages of this approach are that the aggregate performance can be an order of magnitude higher than a fat server, or allow multiple jobs to run at the same time.

The disadvantages are the high up-front costs, as well as maintenance costs. In addition, the hardware may not be ideally suited to each tool, as some tools may require large amounts of RAM or a high-speed interconnect.

### 2.4.3 Supercomputers

Supercomputers have been used by the scientific community for a long time, including bioinformatics, chemistry, physics, astronomy and meteorology.

Supercomputers are generally built up by a huge number of relatively low-powered nodes, with a high-bandwidth and low-latency interconnect. Storage is usually centralized, and accessed over a high-speed network connection.

The advantages of this approach are the availability of a large number of nodes without a large up-front cost for the lab using it. In addition the interconnect enables communication-intensive jobs.

The disadvantages are that not everyone, particularly outside of academia, has access to a supercomputing cluster. In addition, the queue times for jobs may make smaller jobs impractical. Data-intensive jobs may also be a poor fit since the storage or networking infrastructure may not scale. In addition, changes to the hardware and software stack may be impractical to do for a researcher.

### 2.4.4 Cloud Computing

Cloud computing is an alternative to the lab cluster and even supercomputers in later years. Software-as-a-Service platforms are common, and the use of Infrastructure-as-a-Service is on the rise.

There are several types of cloud computing services, with the most attractive approaches for scientists being commercial, such as Amazon EC2 [66], Microsoft Azure [67] and Google Cloud Platform [68], and academic efforts such as CSC cPouta [48]. The infrastructure on offer is highly diverse, with Amazon offering instances such as the X1, which is similar to a fat server, to the relatively small T2 instances, which are low-performance virtual machines.

Commercial clouds store large-scale biological datasets (as in Amazon AWS [69]), and provide the compute resources for analyzing the datasets (e.g. Amazon EC2 [66]). Cloud services such as EC2 run virtual machines provided by the user in very large data-centers. The user only pays for the resources used.

The advantages of elasticity and minimal wait time for running jobs, as well as no up-front costs make the cloud an attractive alternative. In addition, the ability to tailor the cluster to an application through different virtual machine images and instance types means that a wide range of tools can be used.

These attractive properties are tempered by high long-term cost for production-type jobs, and being locked into a particular underlying software infrastructure based on what the cloud provider provides.

### 2.4.5 Infrastructure Used by META-Pipe

META-pipe was originally designed to run on a lab cluster. Currently, it has been expanded to run on supercomputer infrastructure, and integration with cloud platforms has begun.

As the use of META-pipe has increased, our 12-node commodity lab cluster proved too small to provide sufficient performance for multiple concurrent users. A quota on the local supercomputer (Stallo) was available for us, we decided to integrate META-pipe with Stallo, which provides higher performance.

To increase the elasticity of the service to the levels required to support large metagenomic projects, multiple infrastructures are currently being explored, such as three different cloud providers (Amazon Web Sevices, CSC cPouta and EMBL Embassy Cloud). The plan is to extend the supercomputer support with the ability to commission cloud resources on demand.

## 2.5 META-pipe Performance

To motivate the need for the FDDM model, we conduct a performance analysis of META-pipe. The two main questions we want to answer are: (i) which tools in the pipeline scale well with regards to data size; (ii) to what extent do these tools contribute to the overall runtime of META-pipe.

META-pipe scales to multiple nodes by splitting the input data into the same number of chunks as there are nodes. An evaluation of the tools in META-pipe running on more nodes is therefore an evaluation of the scalability of the tool with regards to data size. We also run experiments with different cutoffs from the assembly, which produces different input sizes for the rest of the pipeline, to verify this assumption.

### 2.5.1 Methodology

For the evaluation, we run META-pipe on the Stallo Supercomputer. We used 32, 64 or 128 of the 18.144 cores. Our jobs allocated one core per process (32, 64, or 128). We cannot choose the nodes allocated for a job or the number of cores allocated per node, but we assumed that the process to node mapping does not influence the execution time of our jobs. This assumption does not always hold as discussed below. When a job is submitted to the queue, it is blocked until the queue system allocates the requested resources. This is done in a semi-round-robin system, with additional priority given to smaller jobs, and for users that have few jobs running.

For the evaluation we use the “Muddy” (European Nucleotide Archive sample accession: SAMEA3168559) marine metagenomic sediment sample

| *Tool* | *128 cores* | *32 cores* | *Scalability* |
|---|---|---|---|
| Ray | 45% | 21% | Sublinear |
| MGA | 0% | 0% | No |
| MGA Exporter | 5% | 3% | No |
| FileScheduler | 0% | 0% | No |
| InterProScan | 15% | 33% | Linear |
| Priam | 0% | 0% | Linear |
| BLASTp | 13% | 31% | Linear |
| Annotator | 13% | 6% | Sublinear |
| Exporter | 9% | 5% | No |

**Table 2.3:** Tool-by-tool overview of scalability and impact on total runtime of pipeline on 128 and 32 cores on the medium dataset. Total runtime is 38 hours for 128 cores, and 63 hours for 32 cores.

from the Barents Sea. This dataset is representative for medium sized (9 GB) and high complexity marine metagenomics dataset that we expect most of META-pipe users will analyze. Note that the performance and hence scalability of the tools may depend on the input data complexity, size, quality, and the sequencing technology used to generate sequence data. Complex datasets which are rich in terms of unique organisms present and abundance, and that have large size and low quality base calling, will increase memory usage and affect performance in de Bruijn graph assemblers such as MetaRay [56]. Different sequencing technologies will also contribute with their respective sequencing traits, such as average read length and unique sequence quality flaws.

We evaluated the scalability of functional analysis tools with respect to dataset size by choosing a different cutoff length for the results from MetaRay. The "Medium" dataset, which we have used for the scalability experiments, has a cutoff of 300 nucleotides, and a size of 21 MB. "Small" has a cutoff of 400 nucleotides, resulting in an input size of 12 MB. "Large" has a cutoff size of 250 nucleotides, resulting in a 34 MB input file.

For the MetaRay experiments, the input data was stored on the shared file system on Stallo. For the META-pipe experiments, the input data was loaded from Galaxy.

Note that we only vary the input data size from the assembly. However, as the analysis tools using references databases compare the sequences in the input data to the entries in the reference database, a reduction in reference database size has the same effect as a reduction in input data size.

| *Infrastructure* | *Nodes* | *CPU time* | *Walltime* |
|---|---|---|---|
| Cloud (EC2) | 20 (m4.2xlarge, 80 cores, 8 GB RAM per core, high-bandwidth interconnect) | 615H | 7H41M |
| Cluster (ICE2) | 9 (36 cores, 8 GB RAM per core, gigabit interconnect) | 277H | 7H42M |
| Supercomputer (Stallo) | 10 (80 cores, 16 GB RAM per core, infiniband interconnect) | 918H | 11H28M |

**Table 2.4:** MetaRay performance on cloud, lab cluster and supercomputer infrastructure (Number for EC2 are based on 1 run). CPU time is the aggregate time spent per CPU, while walltime is the time from start to finish.

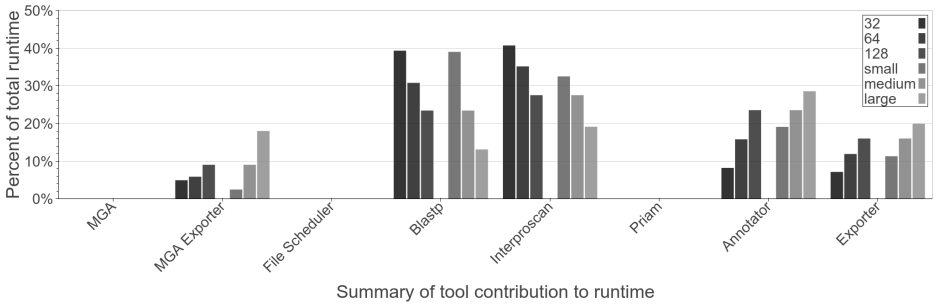
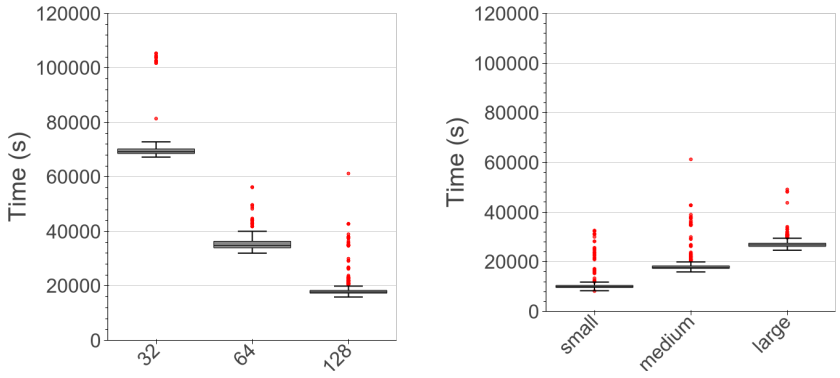
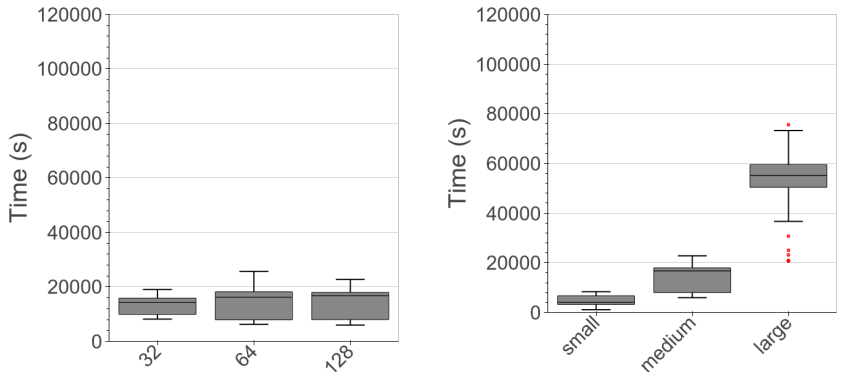



**Figure 2.2:** Contribution to walltime of META-pipe tools.




**(a)** Number of CPUs **(b)** Data set size

**Figure 2.3:** BLAST scaling, showing the characteristics of a linear scaling tool.




**(a)** Number of CPUs **(b)** Data set size

**Figure 2.4:** Annotator scaling, showing the characteristics of a sublinear scaling tool.

### 2.5.2 Results and Discussion

The tools we use in META-pipe are listed in table 2.3, and an illustration of their contribution to runtime can be seen in figure 2.2. An example of a tool that scales well (BLAST) can be seen in figure 2.3, and one that scales poorly (the annotator) can be seen in figure 2.4.

Our results show that the two largest bottlenecks in the pipeline are the assembly of raw reads into longer contigs (45% of runtime for 128 cores) and the locally developed data processing tools (27% of runtime for 128 cores).

These data processing tools (the two exporters and the annotator) contribute to 50% of the functional analysis execution time using 128 cores (when not including the assembly step). In addition, with increasing dataset sizes the computation time exponentially increases. However, this is due to an inefficient implementation. We re-implemented these tools (but did not integrate these with the other tools in the pipeline), and reduced the execution times on the medium dataset with 128 cores to respectively 1 second for the MGA exporter, and 5 seconds for the Annotator and Exporter that we combined into a single tool running on a single node (the re-implemented tools are available at `https://github.com/emrobe/META-pipe`). The performance of these prototypes leads to these tools being less than 1% of the total runtime combined. META-Pipe 2.0 includes an implementation with performance similar to the prototypes for these stages.

The assembly step is more complex to optimize. The main issues here are that single-machine assembly requires large amounts of memory, and the multi-machine implementations do not scale linearly. We have run some preliminary experiments with MetaRay on Amazon EC2, Stallo and our lab cluster, the results can be seen in table 2.4. Note that the numbers for EC2 are from a single run (but we have not observed significant variance), while the others are the

average of three runs.

Several promising alternative assemblers are emerging, such as Spaler [63], as well as a project under development at UiT - The Arctic University of Norway (unpublished), which aim to provide scalable assembly. These projects are not available for use yet, and have therefore not been evaluated.

Our results show that a large portion of non-assembly compute time scales well with regards to data set size, which indicates that reducing the amount of data reduces compute time.

### 2.5.3 Experiment Summary

Summarized, we find that for our experiment, three large classes of tools emerge (i) assembly, which is responsible for 45% of the runtime (ii) BLAST and InterProScan, which are responsible for 28% of the runtime, and (iii) the data processing tools, which are responsible for 27% of the runtime.

Out of these, the BLAST and InterProScan stages scale with regards to data size. Assembly does not scale well, and the assembler has been replaced with a faster, non-distributed assembler in the current implementation of META-pipe. The data processing tools do not scale, but have been replaced with tools that reduce their runtime to seconds.

These results indicate that reducing the input data can reduce the computational cost of META-pipe runs.

## 2.6 Related Work

Several pipelines exist which explore similar ideas in terms of parallelism and performance as META-pipe, such as ADAM [3] and the deep analysis pipeline described in [4]. These systems leverage distributed processing and data storage frameworks to improve scalability of bioinformatics pipelines. However, these systems require changes to the pipeline tools.

Galaxy provides the CloudMan [70] extensions to execute jobs on cloud infrastructure, such as Amazon EC2. There are also many systems that can be used to run biological data processing pipelines such as SLURM [71], Open Lava [72] and Condor [73]. MapReduce [6] and similar application frameworks such as Spark [5], are an alternative both for distributing tools for execution, as well as distributed processing. These systems provide a different programming model, scale well to very large datasets, and they handle load balancing and fault tolerance. However, these are typically not integrated with workflow managers, and hence require an integration approach as described in this thesis. This integration can be simplified through the use of an adapter such as Hadoop-Galaxy [74].

Illumnia offers the infrastructure BaseSpace [75], which is a cloud-based platform for next-generation sequencing data management and analysis. Users can store and share sequencing data, and simplify and accelerate data analysis via the integrated web-based interface. Independent labs can also set-up and monitor their sequencing runs in real time on their Illumina instruments. However, it does not offer extensive analyses and annotation of full-length genes.

Another cloud infrastructure is Oxford Nanopore's Metrichor solution [76], which appears to be similar in functionality to BaseSpace.

We have also built infrastructure systems to reduce the resource usage of META-pipe in computer science research projects, but these are currently not used by the production version of META-pipe. Mario [77] is a system designed to interactively tuning pipeline tool parameters using fine grained iterative processing of the META-pipe data. COMBUSTI/O [78] is a framework for workflow creation, and it used a simplified version of META-pipe in the evaluation.

## 2.7 Summary

In this chapter, we have given an overview of biological data analysis. We have also presented META-pipe, and performed a performance evaluation of individual tools in META-pipe. The results show that there is a need for reducing analysis time by reducing input data size, but this requires a new data management solution, which we investigate in the next chapter.

# 3

# GeStore

In this chapter, we will describe the File-based Distributed Data Materialization (FDDM) model and the implementation of the FDDM model in the GeStore system [24, 25]. We also evaluate the performance of GeStore. In section 3.1 we describe the motivation for the design choices made in GeStore, and the requirements for such a system. In section 3.2 we describe the design of the GeStore system. In section 3.3 we describe the interfaces that GeStore exports. In section 3.4 we provide an experimental evaluation of the performance of GeStore. In section 3.5 we describe related work, and in section 3.6 we draw conclusions from the results.

## 3.1 Motivation and Requirement Analysis

Through our work on, and discussions around, pipelines and workflow managers such as META-pipe [20], EBI Metagenomics [42], IMP [22] and Galaxy [2], we observed that the data management in many pipelines consisted of passing files between tools. However, the version of reference databases and other files were hidden from the user or had to be manually maintained outside the pipeline execution. Reference databases were also not updated very often, as this would create issues with provenance (for example in multi-part analysis jobs, the already-finished analysis would have to be re-analyzed). In addition to infrequent updates and the need for manual maintenance of reference databases and files, the filtering of data was often handled through

ad-hoc processes, such as maintaining a large number of reference databases for different biological groups. Combined these issues lead to more work for the user due to manual maintenance, less frequent updates due to provenance concerns, and longer analysis runtime due to lacking filtering systems.

To solve these problems, we believe a system should fulfill the following requirements:

1. Transparently update results when reference databases are updated, so that results can be kept up to date for less computational cost than a full re-analysis.
2. Produce different versions of a reference database on demand, so that multi-part analysis jobs can be done with a consistent set of reference databases.
3. Generate smaller reference databases from larger ones on demand, so that computational demands can be reduced when only a subset of the reference databases are required.
4. Do not require changes to tools, so that the cost of keeping tools up to date does not increase.
5. Do not require large changes to workflow managers or pipelines, so that integrating the system with existing workflow managers is not prohibitively costly in terms of development time.
6. Conserve provenance, so that the version of a reference database used can be retrieved later.
7. Store data efficiently, so that the storage costs of the reference databases does not grow uncontrollably.
8. Scale with number of computers, so that it is possible to scale out when needed.

To our knowledge, and as discussed in section 2.6, no existing system fulfils these requirements.

## 3.2 Design

In this section, we give an overview of the architecture and design of GeStore. GeStore comprises processing, storage and plugin frameworks. The plugins in the plugin framework use the data processing and storage frameworks to produce data for the pipeline manager on demand, and similarly to add data to the storage framework when data is fed into the system. This is shown in figure 3.1.

Summarized, the solutions to the above requirements are:

1. *Transparently update results*: is solved by automatically determining the

previous run by inspecting provenance data, to determine a version of the reference database to produce. (described in detail in section 3.2.4)
2. *Produce different versions on demand*: is solved by generating reference databases from the collection of data on demand using distributed processing and a distributed database. (described in detail in section 3.2.3)
3. *Smaller reference databases*: is solved by enabling the use of filters on the reference database results. (described in detail in section 3.2.3)
4. *No changes to tools*: is solved by using the file-based approach. (described in detail in section 3.2.1)
5. *Small changes to workflow managers*: is solved by the interface design of GeStore. (described in detail in section 3.3)
6. *Conserve provenance*: is solved by recording requests made by pipelines in a database. (described in detail in section 3.2.4)
7. *Store data efficiently*: is solved by the distributed database storage design. (described in detail in section 3.2.2)
8. *Scale with number of computers*: is solved by using scalable processing and storage systems. (described in detail in section 3.2.2)

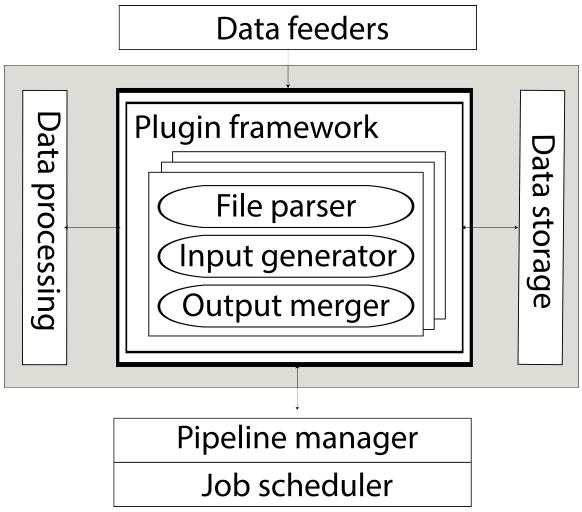



**Figure 3.1:** GeStore architecture.

### 3.2.1 File-based Data Management Model

GeStore uses the FDDM to implement reference database versioning and incremental updates by generating input and meta-data files used by analysis tools that only contain data for a specific period. For example, a reference database for an incremental update may only contain the entries changed in the period. The tool will then be run, as normal, but it will typically produce a

| *File format* | *Tools* |
|---|---|
| FASTA | BLAST, InterProScan, Priam, Annotator, MetaGeneAnnotator, Glimmer. |
| FASTQ | MetaRay, FastQC, Mira. |
| HMM | InterProScan, Priam. |

**Table 3.1:** File formats used as input to META-pipe tools

partial result in case of incremental updates. The result from this tool are then merged with previously produced results.

We have chosen a file-based approach since many genomics applications use relatively few file formats (as seen in table 3.1). It is therefore feasible to implement parsers that support most file formats and therefore most tools. In addition, most file formats are simple and structured, which makes it easy to write parsers for each format. We also believe that many bioinformatics tools can use the file-based approach since bioinformatics applications are often parallelized using a data-parallel approach. Hence, a subset of the data can be computed separately, as in an incremental update.

One example of an analysis tool that is well suited for incremental updates is the widely used BLAST tool [52]. It calculates a similarity score for all gene sequences in an input file by comparing each sequence to all sequences in the UniProtKB [79] reference database. We can implement an incremental update of the results each time UniProtKB is updated by generating an incremental version of the database that only contains the entries that have changed since the last update.

The simplest approach to file generation is to compare all records in two versions of a file to find the new, deleted, and updated records. However, since most tools do not use all record fields, a naive diff will find too many changes. For example, BLAST results are not affected by the annotation record fields that are most frequently changed in the UniProtKB database. It is therefore necessary to write tool specific change detection that only detects changes in the significant fields. In addition, it may be necessary to handle new, updated and deleted records differently. For example, record deletions may require finding and discarding associated records in the output data.

The simplest approach to merge result files is to append the incremental updates to existing result files. However, some output record fields may contain values aggregated over the full dataset. For example, the BLAST output data contains a field, e-value, which is incorrect for incremental updates [18]. In such cases, a tool-specific merger must correct these values in the updated output files.

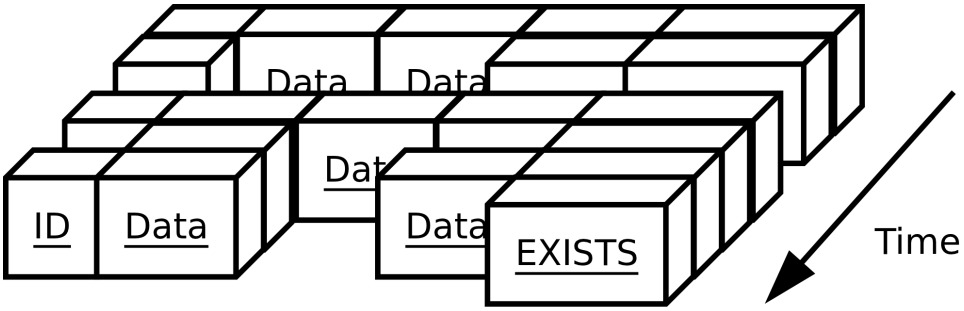



**Figure 3.2:** How data is stored in HBase

### 3.2.2 Storage

GeStore maintains versioned reference database files, input files, and output files. GeStore uses the version information to generate incremental files, or a specific version of a reference database. In addition, the version information is required to merge incremental update results with previously computed results. We use the Hadoop software stack, consisting of HDFS [80], HBase [81], and MapReduce [6], for scalable storage and data processing.

There are two types of files maintained by GeStore: parsed and unparsed. Parsed files are stored as database tables, while unparsed files are stored unmodified on the distributed file system. The unparsed files are files that are difficult to parse and/or do not need incremental updates, such as files that are always completely updated, or not at all.

For file types that have a parser implemented, GeStore splits the data into entries and entry fields. The entries are stored as rows in HBase, and the fields as columns. All entries are stored in the same HBase column family. The only required column in the schema is a unique ID for each row, which the plugin uses to generate a row key. In addition, GeStore generates an EXISTS column, which describes if a given entry exists in the current version of the reference database, so that removing entries is supported. The remaining columns are file-format specific. HBase is designed such that new columns can easily be added to a table. GeStore uses this flexibility to enable reuse of an old HBase table even if the file format or the parser code is changed, so that e.g. if a reference database adds a new field to their schema, this can be reflected in GeStore without influencing previous versions.

We use the HBase timestamp mechanism to manage reference database versions. The timestamp either represent the file generation date, release date or version of the reference database. By storing updated data in timestamped HBase cells, we can efficiently compress a set of database versions using delta compression as seen in figure 3.2. In addition, HBase uses the Snappy compression algorithm to compress the tables.

### 3.2.3 Operations

GeStore provides four operations on the database tables used for reference databases and input, intermediate and output data: create, update, get version, and get increment.

To create a table for a new database, GeStore first checks if the table exists. If not, it will create an empty table, with one empty column family. The database plugin will later add columns for each of the parsed database fields as described above.

To update an existing table with new meta-data, GeStore first finds the correct table to use, then updates or adds new rows using a parallel job that executes the parser for the specific database. Each entry in the new database is compared to the entry in the previous version by comparing the corresponding HBase row. If there are no changes no updates are made, except to the EXISTS field. If one or more fields have changed, the column in the row are updated with the new data for the field and with the current timestamp. If new fields are added, a new column is added to the table with the new data and timestamp. If the row is deleted the EXISTS column is not updated.

To generate an incremental update a parallel job is executed that scans the table for the timeframe $T_{lastrun} - T_{currenttime}$. For each record in the scan, the fields that are relevant for the specified output are selected. If there are updates to one or more of these fields, the record has a current EXISTS field, and the entry matches the regular expression used for filtering if one is supplies, the relevant record fields are written to a file on HDFS.

To generate a specific version of the database we use the above approach with the timeframe $T_{first} - T_{specified}$.

### 3.2.4 Reference Database Caching and Internal Data Structures

GeStore implements a cache of previously generated reference database files, since many workflows can share these. The files are stored in HDFS, and the file-name is used to store information about how the files were generated. We store both big (multi-gigabyte) and small (megabyte) files, since the overhead from starting a job and doing the processing when generating both is large.

The filename consists of a file ID, the time range for the file content, and the regular expression used to select entries in the file. In addition, the filename may contain plugin-specific parameters, a workflow run ID, and a task ID. This information is stored in the file name so that a single file is uniquely identified by the file name without looking it up in a meta-data table. This filename is stored in a GeStore data structure, along with information about the plugin used to generate the file, and if the data is stored in HBase.

When a workflow manager requests a reference database, GeStore first

generates a filename as described above. It uses the filename to lookup in the files table. If a matching filename is found, that file is returned from the HDFS cache. Otherwise, a new file is generated and the files table is updated with a reference to the new file. GeStore does not limit the cache size. The oldest files in the cache can automatically be deleted by e.g. a cron job.

In addition to the data structure used to implement the cache, GeStore has two additional internal data structures. The first has information about each update for each file maintained by GeStore. This includes the number of entries in the file and the version of the update, and is used to determine which version of data is the newest and providing the post-processing step with information about the real size of the reference database in the case of an incremental update. The second contains the files accessed by each workflow tool execution. This table is used to identify which files a workflow used, when they were used, and how they were generated. This is useful for automatically determining which update a particular pipeline run needs when generating incremental updates, as well as conserving provenance information.

## 3.3 Interfaces

GeStore exports two interfaces; the command-line interface and the Java interface. The interfaces of GeStore allow integration with different workflow managers.

The command-line interface is designed to supports workflow managers which request files using a shell. It is generally used by script-based workflow managers. This interface comprises two basic operations, *get* and *put*. The parameters for these operations are detailed in table 3.2 and 3.3.

The Java interface is modeled after the HDFS Java interface, and as such is even simpler in terms of possible arguments, but is more limited in terms of functionality. The required information is inferred from a filename. This interface is described in table 3.4.

Other interfaces such as REST were considered. However, we found that the file-based approach using the command-line and Java interface were the closest match to how the workflow managers accessed data.

| *Parameter* | *Required* | *Description* | *Default* |
|---|---|---|---|
| File | Yes | The name of the file | None |
| Run ID | No | The ID of the run | None |
| Task ID | Yes | The ID of the task | None |
| Time stop | No | The maximum timestamp for the data | MAX_INT |
| Time start | No | The minimum timestamp for the data | 0 |
| Regex | No | A regular expression which must match a given field for the data | None |
| Full run | No | If set, will generate a full data set, even if a previous execution of the run has generated a file | False |
| Format | No | The format of the output, if not the default | None |
| Split | No | Split the output into NUM files | None |
| Copy | No | If false, will not copy files to local disk, and instead return a list of HDFS paths for files | True |

**Table 3.2:** Parameters for GeStore get operation.

| *Parameter* | *Required* | *Description* | *Default* |
|---|---|---|---|
| File | Yes | The name of the file | None |
| Path | Yes | The path to the file on the local disk | None |
| Run ID | No | The ID of the run | None |
| Task ID | No | The ID of the task | None |
| Timestamp | No | The timestamp for the data | Current system time |
| Format | No | The format of the input | None |
| Quick add | No | If true, will return as soon as the job has started | False |

**Table 3.3:** Parameters for GeStore put interface.

| *Action* | *Parameters* | *Description* |
|---|---|---|
| rm | filename | Remove file/directory |
| rmr | filename | Recursive rm |
| mkdir | directory | Create directory |
| getFileSize | file | Get the number of bytes for a file |
| ls | directory | Get a listing of directories |
| lsr | directory | Get a listing of directories and files |
| getFile | file | Returns a HDFS path for the file |
| getFiles | files | Returns HDFS paths for multiple files |
| putFile | local file, remote path | Copy local file to GeStore |
| putFiles | local files, remote path | Copy local files to GeStore |

**Table 3.4:** GeStore Java interface.

### 3.3.1 Plugin Framework

To use GeStore to maintain data used by a pipeline tool, the workflow maintainer must implement: (i) a parser for each file type used by the tool; (ii) tool-specific file generator; and (iii) tool-specific incremental output file merger (if incremental updates will be used). In GeStore, these are implemented as a plugin, and managed by the GeStore plugin framework. The plugin framework uses MapReduce jobs to do the processing required to add data to the system and retrieve it, as well as doing change detection, data verification and merging. MapReduce is used since the files can be very large, and hence efficient parallel processing is required.

The plugins parse and store data. The (unmodified) tool does the data analysis. Typically, only a few tens of lines of code must be written, since many plugins can reuse parsers and file mergers written for other plugins. In addition, the framework provides a library of parsers for known file formats, and libraries for parsing, change detection, and merging of files. The framework also takes care of efficient data storage, low overhead file parsing, file generation, and merging. It is therefore easy to implement a plugin.

#### File Parser

The file parser must determine the structure of input files and reference database files used by a tool. Only one parser must be implemented for each file format, so it is likely that parsers already exists for the file formats used by a tool. The parser must also convert the file data into the HBase table format used by GeStore. Most biological data is in a table format so it is usually easy to implement a parser.

The file parser interface consists of six methods that must be implemented,

| *Method* | *Description* |
|---|---|
| boolean addEntry( String entry ) | Creates an entry based on a string |
| Put getPartialPut( Vector<String> fields, Long timestamp ) | Returns a Put containing the specified fields with the given timestamp |
| boolean sanityCheck( String type ) | Returns True if the entry contains enough information for the given type |
| String[] get( String type, String options ) | Returns an array of strings in the specified format. |
| Vector<String> compare( *Entry ) | Compares this entry with another, returns list of updated fields. |
| String[] getRegexes() | Returns two regular expressions which determine the start and end of an entry in the input file format |

**Table 3.5:** Methods that must be implemented in file parser plugins.

| *Method* | *Description* |
|---|---|
| FileStatus[] process( HashTable<String, String> params, FileSystem fs) | Returns a list of the files that are produced |

**Table 3.6:** Methods that must be implemented in file generator plugins.

listed in table 3.5. These: (i) provide regular expressions that define the start and end of an entry in the file; (ii) split an entry into columns; (iii) compare two versions of an entry; (iv) check if an entry contains all elements required by the tool; (v) generate a HBase Put object; and (vi) generate output in other formats. Every field of the input file is parsed and added to HBase by the plugin, even if only a few fields are used in the analysis.

### File Generator

The file generator class is responsible for generating the input and meta-data files used by a tool. It must detect changes in input data and meta-data. The change detection can be course grained, where the contents of an entire file is compared, or fine grained where individual records are compared. For the latter, the change detection may take into account the structure of the file. In particular, the change detection is efficient and easy to implement if the data is stored in HBase tables as described above.

The file generator requires implementing one method (shown in table 3.6) that specifies the parsers to use for each file format, and the fields to write to

the input file using the associated file parser.

#### Output Merger

The output merger is responsible for merging the results of an incremental computation with previously generated output stored in GeStore. GeStore executes output merging similarly to reference database updates. However, the merge is application-dependent, and hence requires application specific knowledge to understand how an incremental computation may influence the results and how to fix any resulting errors. To fix errors a tool-specific method must be implemented in the plugin.

## 3.4 Evaluation

In this thesis, we focus on questions related to deployment of GeStore in a production system. In particular, we want to answer the following three questions: (i) What are the performance and resource usage characteristics of GeStore? (ii) How does the overhead of GeStore compare to alternative approaches for biological meta-data management? (iii) How does a real-world pipeline perform when using GeStore for incremental updates?

We evaluate the first question to understand how to deploy GeStore in a production system, including identifying areas for optimization, understanding the scalability of the GeStore operations, and how GeStore perturbs other applications running concurrently on the same system. The answer to the second question demonstrates the usefulness of GeStore meta-data management for biological data analysis. The answer to the third question illustrates the real-world use of the GeStore system.

### 3.4.1 Methodology

We characterize GeStore performance and resource usage using benchmarks for each of the GeStore operations We use the META-pipe workflow (section 2.3) as an application benchmark. In addition, we have implemented ad hoc tools for Meta-pipe meta-data management.

We report the average execution time of the benchmarks. Each experiment is repeated 3 times. The standard deviation is only reported when it exceeds 5%.

We use the Ganglia Monitoring System [82] to measure CPU load, memory usage, network traffic, and disk I/O during benchmark execution. The experiments were run on a 10-node cluster. It has one front-end node with

and NFS server, one node with HDFS namenode and HBase master server, and eight HDFS/HBase/MapReduce data/compute nodes. Each node has 32 GB of DRAM, a 4-core Intel Xeon E5-1620 CPU with two-way hyper-threading, 4 TB local disk, and a 2 TB disk used for NFS. The cluster has a 1-gigabit Ethernet interconnect. We assume the cluster size and configuration is realistic for a small cluster in a production environment.

The software used is Oracle Java 1.7.0, Cloudera 4.6.0 (HBase 0.94.6, HDFS 2.0.0, MapReduce 2.0.0, ZooKeeper 3.4.5). In addition, the UniProtKB plugin uses formatdb 2.2.25, which is part of the legacy BLAST package.

HBase is configured to use a heap size of 4 GB for the master server and 12 GB for the region servers. HDFS is configured with a replication factor of three, block size of 128 MB, and heap size of 1 GB for the NameNode and DataNodes. HBase is configured to use Snappy compression, and has a maximum of 32 write-ahead log files. Client scan caching is set to 100.

The data used are the UniProt reference databases[1] from late 2014, and a metagenomic sample from the Yellowstone National Park [83][2]. GeStore version 0.2 is used[3], as well as a modified version of GePan[4].

### 3.4.2 Add and Update Reference Databases

We first measure the time and resource usage of adding a new reference database to GeStore, and for updating an existing database. We assume new reference databases are rarely added, and that reference databases are updated at most weekly. Both operations are therefore background operations, and we are therefore primarily interested in their resource usage, as long execution time for these operations does not directly impact pipeline runtime. Also note, that a GeStore merge is executed similarly to an update, so the results for an update are similar to those of a merge operation.

We download the September 2014 release of UniProtKB (41 GB, gzip compressed), and decompress it on the frontend (231 GB). We measure the time of copying the files to HDFS, and then running a MapReduce job that reads and parses the HDFS files, and puts the 84.5 million parsed entries to an empty HBase table. To update the reference database, we updated it with entries that were updated in the October release (37 out of 87 million entries). The MapReduce job for the update reads and parses the HDFS file, reads old entries by scanning the HBase table, compares each old and new entry, and puts updated entries to HBase.

The time to add the UniProtKB reference database to GeStore is 182 minutes

1. Available at `http://www.uniprot.org`
2. Data available at `http://metagenomics.anl.gov/linkin.cgi?metagenome=4443749.3`
3. Available at `http://github.com/EdvardPedersen/GeStore`
4. Available at `http://github.com/EdvardPedersen/GeStoreGePan`

| *Operation* | *Time* |
|---|---|
| Add 2014_09 UniProtKB | 182 minutes |
| Update to 2014_10 UniProtKB | 144 minutes |
| Retrieve UniProtKB | 36 minutes |
| Retrieve cached UniProtKB | 12 minutes |
| Retrieve incremental UniProtKB | 5 minutes |
| Retrieve cached incremental UniProtKB | 26 seconds |

**Table 3.7:** GeStore add, update, and retrieve operation execution times.

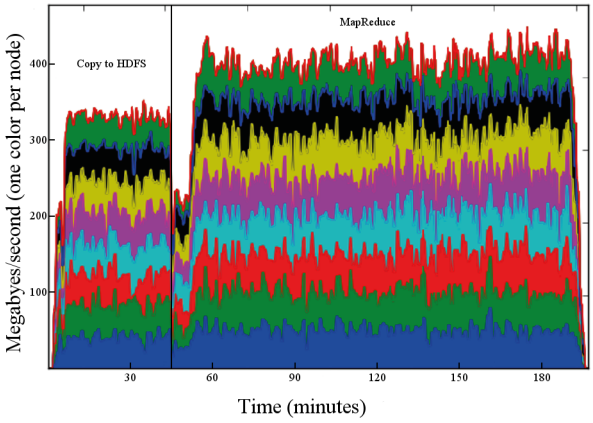



**Figure 3.3:** Bandwidth usage while adding a new reference database to GeStore

(Table 3.7). Updating the reference database is 21% faster (144 minutes). In addition, the time to download and decompress the database are respectively 52 and 33 minutes. We believe the update operation is faster, even if it requires scanning 84.5 rows from HBase for two reasons: First, there are fewer entries put to the HBase table. Second, the reference database is (mostly) cached in memory on the HBase region servers and the updates are more evenly distributed among the cluster nodes.

The resource usage of the add and update operations are similar. For both, the performance is limited by the network (80% of the maximum aggregated bandwidth of the interconnect), while CPU utilization is about 50% over all eight cores. We therefore believe the performance is limited by the HBase (and HDFS) operations. Performance may be improved by better tuning of these operations, for example by optimizing client-side buffering of put operations. Performance will also improve by disabling or relaxing the write-ahead-log (WAL) for HBase. However, this increases the chance of table corruption, which may lead to data loss. The scalability of the add and update operations is similar to the scalability of HBase read and write intensive jobs. The results also show that there are CPU cycles available on the cluster for a computation-intensive

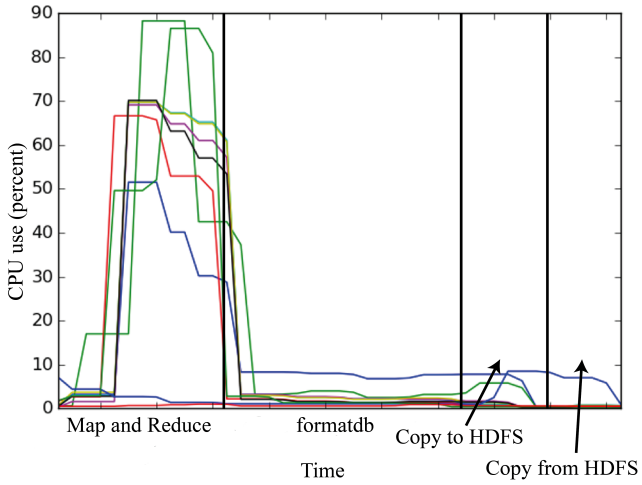



**Figure 3.4:** CPU use for generating a full reference database from scratch

job run concurrently with these operations (many biological data analysis tools are computation-intensive).

### 3.4.3 Retrieve Reference Databases

We evaluate the overhead and resource usage of retrieving an existing reference database from GeStore, and saving it on a local file system to be used by a non-distributed analysis tool. This is a common operation for analysis pipelines with legacy analysis tools. Since the retrieve contributes to pipeline execution time, it should have a low overhead.

We first measured the time to retrieve the November 2014 version of the UniProtKB reference database (240 GB uncompressed, 89 million entries) from GeStore (table 3.7). GeStore will first run a MapReduce job with a large number of map tasks that retrieve the relevant fields for each entry (HBase row), and a single reduce task that writes the output to a single file. The total time to retrieve the reference database is 36 minutes, in which the mappers are done after 14 minutes, and the reducer runs an additional 17 minutes. The overhead is acceptable for two reasons: (i) legacy analysis tools often have a step that must be run sequentially. For example, to convert the retrieved UniProtKB reference database to a BLAST-compatible format using the formatdb tool (resulting file is 32 GB), requires an additional 36 minutes, with an additional 8 minutes to copy the file to HDFS. (ii) the total pipeline execution is often several hours or more.

Second, we retrieved an incremental version of UniProtKB that contains the entries updated between the September 2014 and October 2014 releases

(in total 2.7 million entries, resulting in a file size of 1 GB). The incremental database is generated in 9 minutes. The speedup is due to the much smaller resulting file size and hence reducer execution time.

The cached version of the full UniProtKB takes 12 minutes to retrieve, and the incremental version takes 26 seconds. We believe many pipeline executions can use the cached reference databases in a production system.

The maximum network utilization is about 20%. The CPU utilization differs between the full and incremental update (Figure 3.4) since there is more processing per byte of data in the incremental case. Figure 3.4 also shows how the legacy tools (formatdb, copy to HDFS and copy to local disk), as well as the single reducer stage, dominate the execution time. These execute sequentially and hence limit scalability. In the next section, we evaluate retrieve for tools that do not have this limitation. The incremental update utilizes most of the cluster resources and is therefore not suited to execute concurrently with another job. However, the execution time is short. The longer executing retrieve for a specific version is well suited to overlap with another job, especially during the reducer execution.

### 3.4.4 Retrieve and Split Reference Database

In the experiments in the previous sections, the performance of the retrieve operation was limited by the need to format and write the reference database to a NFS or local file system. In this section, we measure reference database retrieve time for biological data analysis tools that can utilize the high aggregate disk bandwidth on a data-intensive computing platform by reading splits directly from a distributed file system and computing on these in parallel.

We initialized an HBase table with a 50 GB FASTA file with 150 million entries (sequences). The file is generated as in section 3.4.3, but we do not run the formatdb tool, and the output is split among 20 reducers that each write their split directly to the HDFS cache.

| *Operation* | *Time* |
|---|---|
| Retrieve FASTA | 55 minutes |
| Retrieve cached FASTA | 10 minutes |
| Retrieve and split FASTA | 9 minutes |
| Get HDFS path of cached FASTA file | 2 seconds |

**Table 3.8:** Execution time for retrieve and split for the FASTA reference database.

The execution time is reduced from 55 to 9 minutes (Table 3.8). The cached version can be read directly from HDFS, and therefore incurs no overhead. Since there is no bottleneck due to a single reducer, the retrieve scales well.

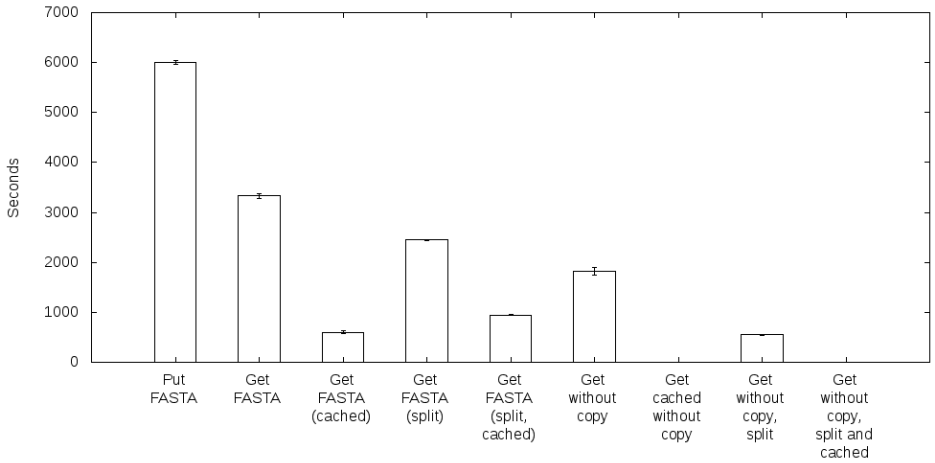



**Figure 3.5:** Walltime for different operations on FASTA reference databases.

### 3.4.5 Space Usage

In this section, we quantify the storage overhead of GeStore. We measure the storage space used by the UniProtKB reference database when compressed using gzip, uncompressed, and in when stored in HBase with delta and Snappy compression and 3x replication.

The disk space used by GeStore is comparable to storage of the compressed reference databases (Table 3.9). The overhead for a single version in HBase is high. With additional versions, the space usage first decreases, then it increases roughly with the size of the compressed version of each new version of UniProtKB. We believe the storage overhead is acceptable, since low-cost distributed storage is used.

| *Versions* | *Compressed on disk* | *Uncompressed on disk* | *In HBase* |
|---|---|---|---|
| 2014-03 | 23 GB | 133 GB | 306 GB |
| 2014-03 to 2014-04 | 47 GB | 268 GB | 240 GB |
| 2014-03 to 2014-05 | 71 GB | 405 GB | 210 GB |
| 2014-03 to 2014-06 | 99 GB | 568 GB | 234 GB |
| 2014-03 to 2014-07 | 133 GB | 757 GB | 273 GB |

**Table 3.9:** Aggregate size of UniProtKB on disk and in HBase using snappy and delta compression with a replication factor of three.

### 3.4.6 Comparison to Ad Hoc Approaches

In this section, we compare GeStore performance to ad hoc approaches for biological meta-data management. We do not compare GeStore performance

against other distributed meta-data management systems, since the previous sections have shown that performance of the GeStore operations is largely dependent on the underlying distributed storage system. Instead, we compare against the non-distributed meta-data storage approach used by most biological data analysis pipelines. These however, require ad hoc implementations of many reference database management problems.

We will focus on the BLAST and *annotation* stage of Meta-pipe. These stages requires a BLAST database and annotation database on each of the compute nodes for parallel execution. It is therefore necessary to generate these reference databases and then replicating them to all nodes. We cannot store the databases on NFS, since the many small file reads by the pipeline significantly increases pipeline execution time. We have implemented a Python script[5] that extracts the FASTA version of UniProtKB (release 2014_11), uses formatdb to generate the BLAST database, and puts the database into a SQLite database used by the annotation tool. Finally, we distribute the resulting files to the computation nodes using rsync with deflate compression enabled.

The creation of the SQLite database is comparable to a GeStore add, since they both make the database accessible from all nodes. The GeStore add is 191 minutes, and the ad hoc script execution time is 315 minutes. Write FASTA, formatdb and copying the BLAST database to the nodes is comparable to a GeStore retrieve, which takes 80 minutes for GeStore, and 206 minutes for the ad hoc script. We have not optimized the ad hoc script. However, we believe such unoptimized scripts are common in biological data analysis. In addition, optimizations to the GeStore operations will benefit all analysis tools using GeStore, while optimization of an ad hoc script typically only benefits a single pipeline.

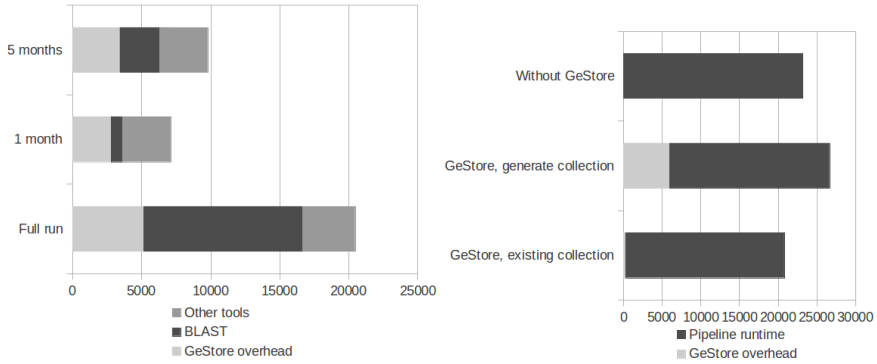
We have not found a realistic use case for an ad hoc implementation of a reference database update and generation of an incremental reference database. For these, existing biological data analysis frameworks typically require generating a new database and then re-executing the full pipeline.

In summary, we believe that GeStore offers better performance and more features than are realistic to implement in ad hoc tools.

| | *Ad hoc Script* | *GeStore* |
|---|---|---|
| Write FASTA | 63 minutes | |
| Formatdb | 34 minutes | |
| Copy BLAST DB to nodes | 109 minutes | |
| *Total* | *206 minutes* | *80 minutes* |
| Create SQLite DB | 142 minutes | |
| Copy SQLite DB to nodes | 173 minutes | |
| *Total* | *315 minutes* | *191 minutes* |

**Table 3.10:** Ad hoc scripts vs. corresponding GeStore operations.

5. Available at `http://github.com/EdvardPedersen/SimpleMetaManager`




**(a)** Improvements in runtime of META-pipe with GeStore

**(b)** Overhead compared to runtime of META-pipe with GeStore

**Figure 3.6:** Integration performance results

### 3.4.7 Application Benchmarks

We measure the execution time of Meta-pipe using a small (15 MB) input file. Only the MGA, MGA exporter, fileScheduler and BLAST were run. We use the 2014_09, 2014_10 and 2015_01 versions of UniProtKB. With a larger input file, the overhead of GeStore compared to workflow execution time will be even lower. In addition, the query feature of GeStore allows the analysis to be done on smaller sections of UniProtKB where appropriate (e.g. fungi are around 10% of UniProtKB, so a query for fungi will produce a reference database of around 1/10th the size of not using a query).

| *Operation* | *Time* |
|---|---|
| Full update without GeStore | 833 minutes |
| Full update with GeStore | 965 minutes |
| Full update with GeStore, existing DB | 859 minutes |
| 1-month incremental update | 61 minutes |
| 4-month incremental update | 99 minutes |

**Table 3.11:** Application benchmarks for Meta-Pipe

GeStore adds an overhead of 132 minutes when generating a reference database, and 26 minutes when the reference database is cached from an earlier run (Table 3.11). Incremental updates are done in 61 minutes for the 1-month update, and 99 minutes for the 4-month update. A 1-month incremental update has a 14-fold speedup compared to a full re-analysis. These results show that incremental updates through GeStore can provide large benefits to production biological analysis workflows.

### 3.4.8 Discussion

The results show that communication between nodes is a large part of the execution time for our benchmarks, in addition to the processing which is not integrated into the tool plugin.

We also see that adding reference databases to GeStore suffers from the overhead imposed by HBase and HDFS, as the data duplication saturates the network infrastructure we use. Further optimization is possible (e.g. disabling the WAL), but we have decided not to perform these due to the possibility of increasing the incidence of undetected data corruption.

In addition, our application benchmarks show that production pipelines can benefit from GeStore, resulting in lower runtimes for updating results.

#### Case Study: Marine Metagenomics Reference Database

As a case study of the benefits of this type of solution, we use an ongoing project at the Center for Bioinformatics in Tromsø as an example. We are creating a marine metagenomics reference database, which contains a database of manually annotated genomes and genes, as well as an automatically annotated database which contains sequences which have been compared with the manually annotated database (similar to how it is done in UniProt).

Our estimates place the cost of the initial creation of the automatically annotated database at around 1 million NOK (based on prices from Amazon EC2 Ireland for m4.2xlarge instances at 0.53 USD per hour, and extrapolated from the experiments done with META-pipe, where assembly accounts for approximately half of the processing time on a 9 GB sample, and requires 154 node-hours, extrapolated to around 10 TB of marine metagenomic data in ENA [84]). Keeping this reference database updated and correct will cost around 1 million NOK per month, since the entire analysis has to be re-done every month to account for new data.

An alternative approach is to only update the reference database with new data when new data becomes available. This will cost around 50 thousand NOK per month (assuming a data growth of around 5% for both reference data and experimental data). However, after one year, over half of the reference database will no longer be fully annotated with the best-matching data using this approach.

The third option is to use the incremental update approach, which will cost around 100 thousand NOK per month since new sequences have to be compared against the full reference database, and all experimental data has to be compared to new sequences in the reference database. Using this approach, the results will maintain their correctness.

Summarized, incremental updates allow a large reduction in cost for maintaining reference databases in metagenomics.

## 3.5 Related Work

Management of large-scale data relies on a robust storage system, we have chosen to use HDFS [80] and HBase [81] due to the close relationship with the Hadoop framework, but other alternatives include Cassandra [85], MongoDB [86] and BlobSeer [87]. We have compared HBase and Cassandra for our use case in section 3.5.1 and found minimal differences in performance. We have also considered alternatives to HDFS such as GPFS [88], but have chosen HBase due to the close integration with the Hadoop ecosystem. In addition, we need to perform some processing on the reference databases to implement the features of GeStore, we have chosen MapReduce [6], but similar systems such as Spark [5], Nectar [11] and Dryad [17], provide many of the same benefits. A more traditional system with e.g. pNFS [89] and MySQL [90] are also an option, but these also require a distributed processing framework to fully utilize the cluster, and require further efforts to integrate with the distributed processing framework.

The simple query facilities in GeStore are limited to simple filtering on entries, more complete query languages such as those used in Google Dremel [91] or Apache Drill [92] would enable more queries that are more complex.

These processing systems have also been extended to support incremental processing, through systems like Incoop [12], Percolator [13], Marimba [93] and DryadInc [14]. In addition, data aggregation systems such as in [94] extend the processing systems to support processing and management of general data types. We have used many ideas from these systems when designing GeStore.

Simple change detection is supported by tools such as UNIX diff, delta encoding compression systems [95], and version management systems such as CVS. However, the change detection in these do not take into account the complex inter-file relationships found in genomic datasets.

GeStore extends the work in [18] by providing a framework and libraries to implement the necessary pre and post processing of data moved between a data warehouse and genomic analysis tools. This makes it easier to add additional support for additional genomic analysis tools as we have demonstrated by implementing incremental updates for a complete metagenomics analysis workflow.

Recent systems similar to GeStore such as [96], which provides versioning and on-demand retrieval of specific reference database versions for Galaxy users, demonstrate the growing need for GeStore functionality.

### 3.5.1 Comparison of Structured Data Storage Systems

In addition to the experiments done with SQLite in section 3.4.6, we have also compared HBase and Cassandra for a typical use case in bioinformatics

to evaluate which is the higher-performing structured data storage system for our use case.

The microbenchmark uses the SPROT part of UniProtKB. We measure the time it takes to add this data to respectively HBase and Cassandra, and then retrieve it. We use a parallel implementation where 8 compute nodes each insert 18th of the data. The rows are prefixed with the node hostname, such that these later can be retrieved by a process running on the node. We use a replication factor of three, so most requests can be served locally. However, there are still some remote data accesses.

Our results shows that for retrieves HBase and Cassandra have similar performance (respectively 9.6 sec and 9 sec for 547085 entries). Cassandra is however much faster than HBase for inserts (58 vs. 161 seconds). However, although we could implement the GeStore operations in Cassandra, we prefer HBase due to our experience with the Hadoop system, and the close integration with MapReduce. In addition, for GeStore retrieve performance is more important than insert performance as discussed above.

### 3.5.2 Experiences using Hadoop

The Hadoop stack is vital for GeStore, here we describe our experiences in using the Hadoop stack for GeStore, and other related projects in our lab.

HDFS is used for caching generated files, as well as files that do not have an appropriate plugin. HBase is used for data, for file formats that have a plugin and ZooKeeper is used to prevent multiple nodes from starting MapReduce jobs at the same time.

Our experiences with the Hadoop stack is that the configuration is very important for performance [77]. In addition, start-up times for MapReduce jobs are significant for small tasks, so for smaller jobs it may be beneficial to not start a MapReduce job. We have also observed that the Hadoop stack has been able to withstand failure of nodes.

## 3.6 Summary

In this chapter we have described the GeStore approach to data management for bioinformatics workflows.

We proposed an approach for efficient management of large-scale biological reference databases. The approach is designed for production systems where biological analysis workflows are periodically executed to analyze large-scale datasets, often by updating existing analysis results with new meta-data. We presented the design and implementation of the GeStore system, including a framework for implementing plugins that enable transparent incremental

updates. We demonstrated the feasibility of our approach and provided an experimental evaluation of our system for reference database management using a real metagenomics analysis workflow and real data. Our findings show that large-scale biological reference databases can be efficiently maintained using data-intensive computing systems, and that our approach can easily be integrated with biological data analysis frameworks, replacing ad hoc solutions. We have also characterized the performance and resource usage of reference database management operations and provided insight into how GeStore can be deployed on a production system. We have demonstrated how GeStore can reduce execution time for updates in a real production pipeline. Finally, we have discussed how GeStore can be used to reduce costs in one of our current projects.

# 4

# Integration

In this chapter, we describe three approaches that we have used to integrate GeStore with three workflow managers in section 4.1, in order to demonstrate that it is possible to utilize modern data management systems with legacy workflow managers. In addition, we survey the literature in bioinformatics to determine how widely used large-scale data management infrastructure systems are in bioinformatics in section 4.2.

## 4.1 Workflow Manager Integration

For a data management system such as GeStore, the integration with workflow managers is necessary to support legacy pipelines and tools. This integration must be done without much overhead, and easily enough that it is feasible to integrate the systems without a complete redesign of the either GeStore or the workflow manager.

The FDDM model makes this integration simple. However, some parameters and information are required from the pipeline to decide which type of file to generate, the version of the file, if it should be incremental, and query parameters.

### 4.1.1 The three approaches

For integrating GeStore with workflow managers, we have utilized three distinct approaches based around three workflow managers. We have used tool-based, direct and file system integration. We believe that the three integration approaches are applicable to most legacy workflow systems in bioinformatics. The file system integration uses the file system interface to GeStore, while the tool-based and direct integration use the command-line interface.

#### Tool-based Integration

Galaxy [2] is probably the most popular pipeline manager for biological data processing. Galaxy represents a big and complex pipeline manager with 165.000 lines of code. We have integrated GeStore with Galaxy by implementing a Galaxy tool wrapper for GeStore. The tool wrapper executes a script that calls GeStore to read files and write result files.

The user must add the GeStore tool to her pipeline, and specify parameters including a unique pipeline-ID, the file ID, incremental file retrieval, and the file type. In order to specify these parameters the user must know the GeStore API, and understand the features and limitation of GeStore. We believe this is too complicated for ordinary Galaxy users, but still useful for power users that use Galaxy to specify and tune pipelines for large datasets or production pipelines.

The approach is easy to implement, and does not require any changes to the large Galaxy codebase. We implemented the GeStore tool using the tool API provided by Galaxy. It is up to the user to specify the GeStore parameters. She can therefore configure GeStore to introduce minimal overhead. We created a new tool wrapper of 51 lines of XML, where GeStore was the tool being wrapped. This tool could then be used in pipelines designed in Galaxy.

The main benefit of this approach is the clear separation between GeStore and the pipeline manager, and hence no changes are required to the pipeline manager. The main disadvantage is manual configuration of GeStore operations. This approach is suited for complex pipeline managers where it is not possible, or practical, to modify the code.

#### Direct Integration

We developed GeStore to provide incremental updates to script-based pipelines such as META-pipe. META-pipe implements a custom pipeline manager called GePan that executes the analysis tools in parallel on a compute cluster using the Open Grid Engine (OGE) or Torque. GePan represents pipeline managers written for a specific pipeline and a traditional HPC cluster. GePan generates

shell scripts that choose reference databases and file format conversions based on tools and biological domains specified by the user.

GePan describes an analysis pipeline as a set of shell scripts generated from a set of parameters that specify the tools to run and their parameters. It will generate two kinds of shell scripts: (i) a set of job scripts; and (ii) a job submission script.

For each step in the pipeline, GePan will generate a script to run the specified tool with the correct parameters. The job script will read a task-ID and job-ID from the environment and pass it to the specified tool as appropriate. The job script will also create directories and delete intermediate files.

The job submission script is a shell script that submits jobs to the job manager. The script consists of a series of commands to OGE that submits each tool in the pipeline to the job manager with the appropriate dependencies.

GePan uses the command-line interface exported by GeStore. We have modified the GePan script generation to replace file copy operations in the job scripts with GeStore calls. GeStore method arguments are set at runtime. This includes the pipeline ID, tool to execute, tool input data, and the reference databases to use. GePan can also determine if the file retrieved is a reference database, input data or intermediate data. In addition, GePan has information about filters to use and other user-supplied parameters. GePan can specify for which files GeStore should generate incremental versions, which files need additional parameters for increment generation, and which files are regular non-incremental files.

The integration requires very small modifications to the pipeline specification. The user must set a "-g" parameter in GePan to specify that GeStore calls should replace file systems operations. In addition, the user may provide a filter to generate a subset of reference database (for example for only one biological taxon).

However, the development effort for the integration is high, since the developer needs to have extensive knowledge of how GeStore will be used in different parts of the pipeline. All file accesses are from scripts generated by GePan, hence GePan must be modified to replace these file accesses with GeStore calls. In total, we added about 300 lines of code to the 14.000 line GePan codebase, but we did not modify any META-pipe tools.

GeStore incurs an overhead for small files (a few megabytes), for which the time to generate incremental updates is larger than the reduction in execution time. To avoid this overhead the pipeline manager can set a file size threshold for files managed by GeStore.

The main advantage of this approach is that the incremental updates are hidden from the user, and that GePan can directly use the full feature set of GeStore. However, the integration requires more development time and a detailed knowledge of the GePan pipeline manager. This integration approach is therefore best suited for small, specialized pipeline mangers that the developer knows in detail.

| *Approach* | *User effort* | *Developer effort* | *Missing features* | *Performance* |
|---|---|---|---|---|
| Key-value | Low | High | Low | High |
| External tool | High | Low | Low | High |
| File system | Low | Moderate | High | Medium |

**Table 4.1:** Summary of integration approaches.

### File System Integration

The Troilkatt system [22] is a system for batch processing pipeline tools that analyze a large-scale compendium. Troilkatt represents specialized pipeline managers designed for data-intensive computing. It manages the setup of tool execution including the specification of input and meta-files for a tool and the management of tool output files. A tool may read and write the files directly from HDFS. Alternatively, if the file requires a POSIX file system interface, Troilkatt automatically copies the files to and from a local file system before and after tool execution. We designed Troilkatt to manage files in several file systems. It therefore provides a common file system interface. The interface was implemented for POSIX file systems and for HDFS. To integrate GeStore with Troilkatt we chose to implement the file system interface for GeStore.

To use GeStore, the user sets a field in the pipeline configuration XML file. The user does not need to have a detailed knowledge of GeStore nor Troilkatt.

The required development effort is moderate. A detailed knowledge of the internals of Troilkatt is not required to implement the interface. However, the interface is complex and the documentation may be lacking. The implementation was 318 lines of code, alongside the existing HDFS and local file system interfaces. These changes were restricted to a single new component, where the HDFS calls were instead handled by the file system interface of GeStore.

The main benefit of this approach is that it does not require changes to core Troilkatt code. However, there are two disadvantages: (i) GeStore must infer the required information for its parameters from file-, and pathnames. These may not always have all required information, such as pipeline IDs, the type of file, or the analysis tool to execute; and (ii) we must parse file pathnames to automatically infer the file type and pipeline ID. If the information is ambiguous, GeStore must use the non-incremental full-file format and can therefore not provide the execution time reduction of incremental updates.

### 4.1.2 Discussion

We have described the three main approaches to integration we have used. These are summarized in table 4.1.

In tool-based integration, GeStore is added as a separate tool in a workflow manager. This approach has several advantages: (i) it is very easy to implement; (ii) it allows the end-user full control over the data management; and (iii) all features of GeStore can be fully utilized. However, the main disadvantage is that the user has to be familiar with GeStore, and has to insert calls to GeStore between every tool of the pipeline that uses GeStore.

In direct integration, GeStore is used in the code for the workflow manager, and replaces the file management of the workflow manager. This approach allows GeStore to be used efficiently, since the workflow manager has full knowledge of which features can be used at any time. It is also invisible for the end user. The disadvantage of this is that it requires relatively deep modifications to the source of the workflow manager.

The file system integration relies on the file system interface to GeStore. It is relatively simple to implement, and is invisible to the end user. However, it relies on hints from the file paths to enable features from GeStore. This means that the level of features used can be adjusted gradually, where the initial integration is relatively simple, while utilizing advanced features requires more effort. The disadvantage to this approach is that either there must be changes in the naming scheme for files used by the pipeline, or it will be difficult to fully utilize GeStore's features.

## 4.2 Use of Data-intensive Computing Systems in Bioinformatics

We believe that data-intensive computing systems such as MapReduce are not widely used in bioinformatics. To gauge the extent of use of data-intensive computing systems in bioinformatics, we performed a literature review in BMC Bioinformatics[1] including articles until November 2014.

To answer this question we count the number of articles mentioning MapReduce and Hadoop. The MapReduce design [97] was published in 2004, the open-source Hadoop MapReduce project [97] was started in 2005, and Hadoop became popular around 2008. Since then, an increasing number of articles mention MapReduce and Hadoop each year; especially in the last two years

1. We also examined Genome Biology and the Web Server and Database special issues of Nucleic Acids Research, but these journals have few infrastructure articles compared to BMC Bioinformatics

| *Year* | *Hadoop* | *MapReduce* |
|---|---|---|
| 2009 | 1 | 1 |
| 2010 | 4 | 7 |
| 2011 | 4 | 7 |
| 2012 | 4 | 8 |
| 2013 | 4 | 15 |
| 2014 | 7 | 14 |
| Total | 24 | 52 |

**Table 4.2:** Number of articles per year for keywords MapReduce and Hadoop (many articles are in both results). The year 2014 does not include articles published in November and December.

(Table 4.2). The results indicate that data-intensive computing systems are increasingly being used for biological data processing.

### Use of Specific Systems in Bioinformatics

In this section, we examine papers describing specific data-intensive computing systems. We want to find the systems that are used for biological data processing, and what the systems are used for. We search for specific keywords and manually filter the returned results to exclude papers that do not describe the system. We set *Include* to *All* article types, since the number of *Software* papers is low for some searches.

We first searched for "Hadoop" since it is the most widely used data-intensive computing platform. The search returned 24 papers. Of these, the software in 14 papers use systems in the Hadoop stack, three articles describe virtual machine images or provisioning systems that include Hadoop, the remaining only mention Hadoop in related work. The Hadoop systems are used for search, integrated analysis, data integration, machine learning, and distributed analysis of different data types

We also searched for the Hadoop alternative Azure [67]. There are six articles discussing Azure, of which the software in two use Azure in combination with MapReduce, two articles propose to use Azure for processing and storage, one describes a virtual machine provisioning systems, and one is a review article.

We then searched for names of the data-intensive computing systems described in section 2. For MapReduce we found 54 articles. We refined the MapReduce search by limiting the search to Title+Abstract+Text in order to remove articles that just mention MapReduce in the citations. Of the remaining 27 articles, 12 describe software that use Hadoop MapReduce and three that use the MapReduce programming model. Most of these articles are also in

the Hadoop search results (discussed above). We found four HBase articles; of which two [98, 99] describe software that uses HBase as a storage backend for sequencing data (the remaining two mention HBase in related work). We found two Spark papers. One described how they used Spark to implement distributed processing of next-generation sequencing data [100]. Cassandra, Hive, and Mahout are not used by software in any articles, but are discussed in the related work in two articles. We did not find Impala in any articles.

We also searched for the Pig [101], Cascading [102], Drill [92], and Storm [103] systems, but these return many false positives. We therefore refined the search to: "Pig Apache", "Cascading SQL", "Drill Apache", and "Storm Apache". Pig and Cascading where both discussed in the related work in two articles. We did not find any articles for Drill and Storm.

The above results show that MapReduce is the most popular system for data-intensive biological data processing, and that most MapReduce tool implementations use the Hadoop stack. We also found a few articles where HBase and Spark were used. MapReduce, Spark, and HBase are all core systems. We did not find any Bioinformatics articles that described tools that use higher-level systems. This may suggest that the services and abstractions provided by these are not well suited for biological data. The systems that were not mentioned in any articles (Impala [104], Drill, and Storm) are all recently developed, and may therefore have been unstable when the tools were implemented.

### Updates to the Review

Since our literature review was done in 2014, we have attempted to update the results with more recent articles.

At the time of writing, we find 7 new papers when searching for Hadoop in BMC Bioinformatics, 8 new papers for MapReduce, two additional papers for Spark, two additional papers for HBase and no additional articles for the other terms used.

Of the 13 unique articles found, 5 use MapReduce, Spark, HDFS, or HBase. 4 are compatible with Hadoop or the MapReduce paradigm. 2 mention MapReduce or Hadoop as related work, and 2 mention MapReduce or Hadoop as future work.

These results are not fully compatible with the results in the literature review, due to changes to the search algorithm used in BMC Bioinformatics, resulting in differences in the number of articles found when searching for the terms used in the review.

### 4.2.1 Discussion

Our literature review shows that big data management systems are not yet widely used in bioinformatics, but the trend seems to be towards increased adaption.

It should be noted that since the review was done, new papers have appeared such as ADAM [3] and the deep analysis pipeline described in [4], but these were published at primarily computer science conferences.

## 4.3 Conclusion

In this chapter, we have demonstrated three approaches for integrating data-intensive computing systems with legacy bioinformatics workflow managers and pipelines. We have shown the tradeoffs involved in the different approaches.

Despite the relative ease of this integration, our literature review shows that large-scale data management and processing systems are not widely used in bioinformatics. We do however observe a trend towards more adaption. This indicates that there is a need for data management and processing systems in bioinformatics.

# 5

# Conclusions

The current data growth in biology has large ramifications for data management in bioinformatics.

This thesis demonstrates that a data management approach based on the file-based distributed data materialization model can be leveraged by existing bioinformatics pipelines to reduce runtime, keep results up to date and maintain repeatability.

Through our analysis of META-pipe in chapter 2, we show that the execution time of a large fraction of the tools used in bioinformatics pipelines can be reduced by reducing the amount of data processed. This means that a system that can reduce the amount of data used by these tools can also reduce the runtime of the pipeline. Our analysis of the GeStore implementation of the FDDM in chapter 3, shows that this model can be used to generate reference databases quickly, and that we can reduce the amount of data generated for the tools used in pipelines. Our integration of GeStore with three classes of workflow managers, described in chapter 4, demonstrates that the GeStore implementation of the FDDM can be used by several existing bioinformatics pipelines.

Combined, these results show that the FDDM approach can be leveraged to increase the performance and reduce the cost, of running pipelines, in particular when updating experimental results.

## 5.1 Lessons Learned

Through the design of the FDDM, and the implementation of the FDDM in the GeStore system, some key lessons have been learned which we believe are applicable more generally.

*Integration* is a key factor in the performance and usability of middleware systems such as GeStore. We have explored multiple approaches to the integration, and found several feasible approaches, which have different tradeoffs in terms of user effort, integration effort, and richness of features that can be used by the workflow manager. If tools are difficult to use, they will not be used by users, and if they are too difficult to integrate against, they will not be used by pipeline creators.

*Performance* is one of the key reasons to adopt a system such as GeStore. We have found that many tools in bioinformatics pipelines have computational requirements that scale with data size, enabling us to increase performance drastically by reducing the volume data. Performance is important since the demands of researchers and customers include performance, there is not much demand for an analysis service that completes analysis in months and years.

*Cost* reduction is the most significant motivator of this work, although closely related to performance. The reduction in cost is for some cases directly related to number of CPU hours used in analysis, since the number of CPU hours used determines how many machines are needed and for how long. Since GeStore is able to reduce the number of CPU hours used, the cost is also similarly reduced. Cost is a significant limitation of many projects in bioinformatics in terms of the cost of hardware, software and VMs as well as the human cost in terms of support, and manual maintenance of data.

## 5.2 Availability

GeStore is open-source, and available at `https://github.com/edvardpedersen/GeStore`. Also available at the same website is a user guide, as well as a preview VM for easy setup.

META-pipe is open-source, and available at `https://github.com/emrobe/META-pipe`. In addition, the service is available at `https://galaxy-uit.bioinfo.no` (note that a FEIDE user account is required to access the service).

# 6

# Future Work

In this chapter, we outline future work.

## 6.1 GeStore Improvements

GeStore improvements include investigating more easy to use query opportunities, such as extending the regular expression queries with an integration step to the workflow manager that rewrites simpler queries to GeStore-compatible queries, so that it is simpler to write queries for the end-user.

Another feature which is needed to support additional tools would be plugins for additional file formats. One specific area of improvement is to implement an improved plugin for the file format used to store hidden markov models (HMM) used by many analysis tools, such as InterProScan, so that small changes are marked as insignificant. The HMM data consists of an array of floating point numbers, and even small changes in the data used to generate the HMM can lead to small changes in a large number of entries in the reference database. These changes are currently marked as significant, even if the results are not impacted by the changes.

There are several areas for optimizing the performance of GeStore. We may better tune HBase, HDFS and MapReduce parameters. For example by relaxing the rules for updating the write-ahead log in HBase, which is a tradeoff between performance and fault tolerance. Since we work from assumptions that adding large reference databases to GeStore will be a relatively infrequent

background job, we have not investigated the optimization opportunities for that stage of GeStore in great depth.

Recent pipelines such as ADAM [3] and META-pipe 2.0 use Spark for data processing. For GeStore, Spark may provide better performance for small files by reducing startup time, but requires changes to the code related to retrieval and addition of data.

## 6.2 Deployment Challenges and Opportunities

The marine reference database described in section 3.4.8 is a project that requires a GeStore-like approach for frequent updates. Current estimates are that 4.32 million CPU hours are needed for the initial release. For subsequent releases, using incremental updates will significantly reduce costs for each update. We therefore plan to use the FDDM approach to handle this use case.

Other workflow managers and pipelines such as ADAM [3] and Taverna [8] are possible future integration targets for GeStore. They do not currently solve the problem which GeStore is designed to solve; increasing performance by reducing data volumes processed by tools. It would be of particular interest to examine additional modern workflow managers that use large-scale data intensive frameworks such as Spark and HDFS to store and process data for tools without using the local file system. For these we can reduce overhead by reducing the reliance on local files.

## 6.3 Quality Control and Error Detection

Error detection and quality control is vital to avoid producing erroneous results. We have started to investigate quality and error detection within the data management system, but we have not yet moved beyond early prototyping. This approach is based on the idea of combining legacy tools such as FastQC [50] with heuristics in a framework integrated with the data management system to enable techniques such as sampling, and testing ranges of data. In addition, we plan to combine automatic error detection with visualization techniques to do quality control. The early results show that this type of error detection has potential to be beneficial to users. By detecting errors as early as possible, processing time can be saved by aborting the pipeline and informing the user of the issues with the data set.

# Bibliography


[1] M. Baker, "Next-generation sequencing: adjusting to data overload," *Nat Meth*, vol. 7, no. 7, pp. 495–499, Jul. 2010.

[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, no. 8, p. R86, Jan. 2010.

[3] F. A. Nothaft, M. Massie, T. Danford, Z. Zhang, U. Laserson, C. Yeksigian, J. Kottalam, A. Ahuja, J. Hammerbacher, M. Linderman, M. J. Franklin, A. D. Joseph, and D. A. Patterson, "Rethinking data-intensive science using scalable analytics systems," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015, pp. 631–646.

[4] Y. Diao, A. Roy, and T. Bloom, "Building highly-optimized, low-latency pipelines for genomic data analysis," in *Proc. of 7th Biennial Conference on Innovative Data Systems Research*, 2015.

[5] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark : Cluster Computing with Working Sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, p. 10.

[6] J. Dean and S. Ghemawat, "MapReduce," *Communications of the ACM*, vol. 51, no. 1, p. 107, Jan. 2008.

[7] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, "Taverna: a tool for the composition and enactment of bioinformatics workflows." *Bioinformatics*, vol. 20, no. 17, pp. 3045–54, Nov. 2004.

[8] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble, "The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud," *Nucleic Acids Research*, vol. 41, pp. W557–61, Jul 2013.

[9] 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, no. 10, p. R80, Jan. 2004.

[10] Y. A. Liu, S. D. Stoller, and T. Teitelbaum, "Static caching for incremental computation," *ACM Trans. on Programming Languages and Systems*, vol. 20, no. 3, pp. 546–585, May 1998.

[11] 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 the 9th Symposium on Operating Systems Design and Implementation*. USENIX, 2010, pp. 1–8.

[12] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquini, "Incoop : MapReduce for Incremental Computations," in *Proc. of the 2nd ACM Symposium on Cloud Computing*. ACM Press, 2011, p. 7.

[13] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications," in *Proc. of the 9th Symposium on Operating Systems Design and Implementation*, Google, Inc. USENIX, 2010, pp. 1–15.

[14] L. Popa, M. Budiu, Y. Yu, and M. Isard, "DryadInc: reusing work in large-scale computations," *Proc. of the 2009 conference on Hot topics in cloud computing*, p. 21, Jun. 2009.

[15] D. Logothetis, C. Olston, B. Reed, K. C. Webb, and K. Yocum, "Stateful bulk processing for incremental analytics," in *Proc. of the 1st ACM symposium on Cloud computing*. New York, New York, USA: ACM Press, Jun. 2010, p. 51.

[16] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "The HaLoop approach to large-scale iterative data analysis," *The VLDB Journal*, vol. 21, no. 2, pp. 169–190, Mar. 2012.

[17] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, p. 59, 2007.

[18] G. Turcu, S. Nestorov, and I. Foster, "Efficient Incremental Maintenance of Derived Relations and BLAST Computations in Bioinformatics Data Warehouses," in *Data Warehousing and Knowledge Discovery*, ser. Lecture Notes in Computer Science. Springer, 2008, vol. 5182, pp. 135–145.

[19] J. Leipzig, "A review of bioinformatic pipeline frameworks," *Briefings in Bioinformatics*, 2016.

[20] E. M. Robertsen, T. Kahlke, I. A. Raknes, E. Pedersen, E. K. Semb, M. Ernstsen, L. A. Bongo, and N. P. Willassen, "Meta-pipe - pipeline annotation, analysis and visualization of marine metagenomic sequence data," 2016, arXiv:1604.04103 [cs.DC].

[21] S. Hunter, M. Corbett, H. Denise, M. Fraser, A. Gonzalez-Beltran, C. Hunter, P. Jones, R. Leinonen, C. McAnulla, E. Maguire, J. Maslen, A. Mitchell, G. Nuka, A. Oisel, S. Pesseat, R. Radhakrishnan, P. Rocca-Serra, M. Scheremetjew, P. Sterk, D. Vaughan, G. Cochrane, D. Field, and S.-A. Sansone, "EBI metagenomics–a new resource for the analysis

and archiving of metagenomic data," *Nucleic Acids Research*, vol. 42, pp. D600–6, Jan 2014.

[22] A. K. Wong, C. Y. Park, C. S. Greene, L. A. Bongo, Y. Guan, and O. G. Troyanskaya, "Imp: a multi-species functional genomics portal for integration, visualization and prediction of protein functions and networks," *Nucleic Acids Research*, vol. 40, pp. W484–90, Jul 2012.

[23] E. Pedersen, "Sequence assembly in the cloud, on the grid and in the basement," in *Abstract Book from NBS 2016*, 2016.

[24] E. Pedersen, N. P. Willassen, and L. A. Bongo, "Transparent Incremental Updates for Genomics Data Analysis Pipelines," in *Proc. of HiBB 2013 – 4th Workshop on High Performance Bioinformatics and Biomedicine*, ser. LNCS, vol. 8374. Springer, 2014, pp. 311–320.

[25] E. Pedersen and L. A. Bongo, "Large-scale biological meta-database management," *Future Generation Computer Systems*, 2016, in Press.

[26] E. Pedersen, I. A. Raknes, M. Ernstsen, and L. A. Bongo, "Integrating Data-Intensive Computing Systems with Biological Data Analysis Frameworks," in *Proc. of 23rd Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2015, pp. 733–740.

[27] L. A. Bongo, E. Pedersen, and M. Ernstsen, "Data-Intensive Computing Infrastructure Systems for Unmodified Biological Data Analysis Pipelines," in *Computational Intelligence Methods for Bioinformatics and Biostatistics*, ser. LNBI, vol. 8623, 2014.

[28] E. Pedersen and L. A. Bongo, *Resource Management for Big Data Applications*. Springer, 2016, ch. Big Biological Data Management.

[29] Norwegian Bioinformatics Platform, "Norwegian e-infrastructure for life sciences," 2016, [accessed 18-April-2016]. [Online]. Available: https://nels.bioinfo.no

[30] Elixir, "Elixir excelerate," 2016, [accessed 18-April-2016]. [Online]. Available: https://www.elixir-europe.org/excelerate

[31] R. Edgar, M. Domrachev, and A. E. Lash, "Gene Expression Omnibus: NCBI gene expression and hybridization array data repository." *Nucleic Acids Res.*, vol. 30, no. 1, pp. 207–210, 2002.

[32] R. Leinonen, R. Akhtar, E. Birney, L. Bower, A. Cerdeno-Tárraga, Y. Cheng, I. Cleland, N. Faruque, N. Goodgame, R. Gibson, G. Hoad, M. Jang, N. Pakseresht, S. Plaister, R. Radhakrishnan, K. Reddy, S. Sobhany, P. T. Hoopen, R. Vaughan, V. Zalunin, and G. Cochrane, "The European nucleotide archive," *Nucleic Acids Res.*, vol. 39, no. SUPPL. 1, 2011.

[33] X. M. Fernández-Suárez, D. J. Rigden, and M. Y. Galperin, "The 2014 Nucleic Acids Research Database Issue and an updated NAR online Molecular Biology Database Collection." *Nucleic Acids Res.*, vol. 42, no. Database issue, pp. D1–6, Jan. 2014.

[34] J. E. Stajich, D. Block, K. Boulez, S. E. Brenner, S. A. Chervitz, C. Dagdigian, G. Fuellen, J. G. R. Gilbert, I. Korf, H. Lapp, H. Lehväslaiho, C. Matsalla, C. J. Mungall, B. I. Osborne, M. R. Pocock, P. Schattner, M. Senger,

L. D. Stein, E. Stupka, M. D. Wilkinson, and E. Birney, "The Bioperl toolkit: Perl modules for the life sciences," *Genome Research*, vol. 12, no. 10, pp. 1611–1618, 2002.

[35] D. Blankenberg, G. Von Kuster, N. Coraor, G. Ananda, R. Lazarus, M. Mangan, A. Nekrutenko, and J. Taylor, "Galaxy, a web-based genome analysis tool for experimentalists," *Current protocols in molecular biology*, vol. 0 19, pp. Unit–19.1021, Jan 2010.

[36] P. Jones, D. Binns, H.-Y. Chang, M. Fraser, W. Li, C. McAnulla, H. McWilliam, J. Maslen, A. Mitchell, G. Nuka, S. Pesseat, A. F. Quinn, A. Sangrador-Vegas, M. Scheremetjew, S.-Y. Yong, R. Lopez, and S. Hunter, "Interproscan 5: genome-scale protein function classification," *Bioinformatics*, vol. 30, pp. 1236–40, May 2014.

[37] P. ten Hoopen, S. Pesant, R. Kottmann, A. Kopf, M. Bicak, S. Claus, K. Deneudt, C. Borremans, P. Thijsse, S. Dekeyzer, D. M. Schaap, C. Bowler, F. O. Glöckner, and G. Cochrane, "Marine microbial biodiversity, bioinformatics and biotechnology (m2b3) data reporting and service standards," *Standards in Genomic Sciences*, vol. 10, no. 1, pp. 1–10, 2015.

[38] European Bioinformatics Institute, "EBI Metagenomics," [accessed 18-April-2016]. [Online]. Available: https://www.ebi.ac.uk/metagenomics/

[39] J. Goll, D. B. 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 (Oxford, England)*, vol. 26, no. 20, pp. 2631–2, Oct. 2010.

[40] B. D. Ondov, N. H. Bergman, and A. M. Phillippy, "Interactive metagenomic visualization in a web browser," *BMC Bioinformatics*, vol. 12, no. 1, pp. 1–10, 2011.

[41] Oxford Journals, "Instructions to authors," 2016, [accessed 10-September-2016]. [Online]. Available: https://cloud.google.com

[42] S. Hunter, M. Corbett, H. Denise, M. Fraser, A. Gonzalez-Beltran, C. Hunter, P. Jones, R. Leinonen, C. McAnulla, E. Maguire, J. Maslen, A. Mitchell, G. Nuka, A. Oisel, S. Pesseat, R. Radhakrishnan, P. Rocca-Serra, M. Scheremetjew, P. Sterk, D. Vaughan, G. Cochrane, D. Field, and S.-A. Sansone, "EBI metagenomics–a new resource for the analysis and archiving of metagenomic data." *Nucleic Acids Res.*, vol. 42, no. Database issue, pp. D600–6, Jan. 2014.

[43] A. Mitchell, H.-Y. Chang, L. Daugherty, M. Fraser, S. Hunter, R. Lopez, C. McAnulla, C. McMenamin, G. Nuka, S. Pesseat, A. Sangrador-Vegas, M. Scheremetjew, C. Rato, S.-Y. Yong, A. Bateman, M. Punta, T. K. Attwood, C. J. A. Sigrist, N. Redaschi, C. Rivoire, I. Xenarios, D. Kahn, D. Guyot, P. Bork, I. Letunic, J. Gough, M. Oates, D. Haft, H. Huang, D. A. Natale, C. H. Wu, C. Orengo, I. Sillitoe, H. Mi, P. D. Thomas, and R. D. Finn, "The InterPro protein families database: the classification

resource after 15 years," *Nucleic Acids Research*, vol. 43, pp. D213–21, Jan 2015.

[44] F. Meyer, D. Paarmann, M. D'Souza, R. Olson, E. M. Glass, M. Kubal, T. Paczian, A. Rodriguez, R. Stevens, A. Wilke, J. Wilkening, and R. A. Edwards, "The metagenomics rast server - a public resource for the automatic phylogenetic and functional analysis of metagenomes," *BMC Bioinformatics*, vol. 9, p. 386, 2008.

[45] V. M. Markowitz, I.-M. A. Chen, K. Chu, E. Szeto, K. Palaniappan, Y. Grechkin, A. Ratner, B. Jacob, A. Pati, M. Huntemann, K. Liolios, I. Pagani, I. Anderson, K. Mavromatis, N. N. Ivanova, and N. C. Kyrpides, "Img/m: the integrated metagenome data management and comparative analysis system," *Nucleic Acids Research*, vol. 40, pp. D123–9, Jan 2012.

[46] EGI, "EGI Federated Cloud," 2016, [accessed 18-April-2016]. [Online]. Available: https://www.egi.eu/infrastructure/cloud/

[47] EMBL-EBI, "Embassy cloud," 2016, [accessed 18-April-2016]. [Online]. Available: http://www.embassycloud.org

[48] CSC, "cPouta," 2016, [accessed 18-April-2016]. [Online]. Available: https://research.csc.fi/cpouta

[49] EUDAT, "Eudat," 2016, [accessed 18-April-2016]. [Online]. Available: https://www.eudat.eu

[50] S. Andrews, "Fastqc a quality control tool for high throughput sequence data," feb 2016, [accessed 18-April-2016]. [Online]. Available: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

[51] R. Schmieder and R. Edwards, "Quality control and preprocessing of metagenomic datasets," *Bioinformatics*, vol. 27, pp. 863–4, Mar 2011.

[52] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[53] C. Claudel-Renard, C. Chevalet, T. Faraut, and D. Kahn, "Enzyme-specific profiles for genome annotation: Priam," *Nucleic Acids Research*, vol. 31, pp. 6633–9, Nov 2003.

[54] B. Chevreux, T. Wetter, and S. Suhai, "Genome sequence assembly using trace signals and additional sequence information," feb 2016, [accessed 18-April-2016]. [Online]. Available: http://miraassembler.sourceforge.net/docs/DefinitiveGuideToMIRA.html

[55] B. Chevreux, T. Pfisterer, B. Drescher, A. J. Driesel, W. E. G. Müller, T. Wetter, and S. Suhai, "Using the miraest assembler for reliable and automated mrna transcript assembly and snp detection in sequenced ests," *Genome Research*, vol. 14, pp. 1147–59, Jun 2004.

[56] S. Boisvert, F. Raymond, E. Godzaridis, F. Laviolette, and J. Corbeil, "Ray meta: scalable de novo metagenome assembly and profiling," *Genome Biology*, vol. 13, p. R122, 2012.

[57] S. Boisvert, F. Laviolette, and J. Corbeil, "Ray: Simultaneous assembly of reads from a mix of high-throughput sequencing technologies," *Journal*

*of Computational Biology*, vol. 17, pp. 1519–33, Nov 2010.

[58] H. Noguchi, T. Taniguchi, and T. Itoh, "MetaGeneAnnotator: detecting species-specific patterns of ribosomal binding site for precise gene prediction in anonymous prokaryotic and phage genomes." *DNA Research*, vol. 15, no. 6, pp. 387–96, Dec. 2008.

[59] H. Noguchi, J. Park, and T. Takagi, "Metagene: prokaryotic gene finding from environmental genome shotgun sequences," *Nucleic Acids Research*, vol. 34, no. 19, pp. 5623–5630, 2006.

[60] A. L. Delcher, D. Harmon, S. Kasif, O. White, and S. L. Salzberg, "Improved microbial gene identification with glimmer." *Nucleic Acids Research*, vol. 27, pp. 4636–41, Dec 1999.

[61] S. L. Salzberg, A. L. Delcher, S. Kasif, and O. White, "Microbial gene identification using interpolated markov models." *Nucleic Acids Research*, vol. 26, pp. 544–8, Jan 1998.

[62] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol, "Abyss: A parallel assembler for short read sequence data," *Genome Research*, vol. 19, no. 6, pp. 1117–1123, 2009.

[63] A. Abu-Doleh and U. V. Catalyurek, "Spaler: Spark and GraphX based de novo genome assembler," in *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, Oct 2015, pp. 1013–1018.

[64] T. Kahlke, "Analysis of the vibrionaceae pan-genome," Ph.D. dissertation, UiT - The Arctic University of Norway, 2013.

[65] UiO, "Lifeportal," 2016, [accessed 18-April-2016]. [Online]. Available: https://lifeportal.uio.no

[66] Amazon, "Elastic compute cloud," 2016, [accessed 18-April-2016]. [Online]. Available: https://aws.amazon.com/ec2/

[67] Microsoft, "Azure," 2016, [accessed 18-April-2016]. [Online]. Available: https://azure.microsoft.com

[68] Google, "Google cloud platform," 2016, [accessed 18-April-2016]. [Online]. Available: https://cloud.google.com

[69] Amazon, "Simple storage service," 2016, [accessed 18-April-2016]. [Online]. Available: https://aws.amazon.com/s3/

[70] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor, "Galaxy Cloudman: delivering cloud compute clusters," *BMC Bioinformatics*, vol. 11, p. S4, 2010.

[71] A. B. Yoo, M. A. Jette, and M. Grondona, *Job Scheduling Strategies for Parallel Processing: 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ch. SLURM: Simple Linux Utility for Resource Management, pp. 44–60.

[72] The OpenLava Project, "Openlava - open source job scheduler," 2016, [accessed 18-April-2016]. [Online]. Available: http://www.openlava.org

[73] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor-a hunter of idle workstations," in *Proc. of 8th International Conference on Distributed*

*Computing Systems*, Jun 1988, pp. 104–111.

[74] L. Pireddu, S. Leo, N. Soranzo, and G. Zanetti, "A Hadoop-galaxy adapter for user-friendly and scalable data-intensive bioinformatics in galaxy," in *Proc. of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, ser. BCB '14. New York, NY, USA: ACM, 2014, pp. 184–191.

[75] Illumina, "Basespace sequence hub," 2016, [accessed 18-April-2016]. [Online]. Available: https://basespace.illumina.com

[76] Oxford Nanospore, "Metrichor," 2016, [accessed 18-April-2016]. [Online]. Available: https://metrichor.com/

[77] M. Ernstsen, E. Kjærner-Semb, N. P. Willassen, and L. A. Bongo, *Euro-Par 2014: Parallel Processing Workshops: Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part I*. Cham: Springer International Publishing, 2014, ch. Mario: Interactive Tuning of Biological Analysis Pipelines Using Iterative Processing, pp. 263–274.

[78] J. Fagerli, "COMBUSTI/O," Master's thesis, UiT - The Arctic University of Norway, 2016.

[79] M. Magrane and U. Consortium, "UniProt Knowledgebase: a hub of integrated protein data." *Database*, vol. 2011, 2011.

[80] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies*, no. 5, pp. 1–10, 2010.

[81] Apache, "Apache HBase," [accessed 18-April-2016]. [Online]. Available: http://hbase.apache.org

[82] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, Jul. 2004.

[83] D. Bhaya, A. R. Grossman, A.-S. Steunou, N. Khuri, F. M. Cohan, N. Hamamura, M. C. Melendrez, M. M. Bateson, D. M. Ward, and J. F. Heidelberg, "Population level functional diversity in a microbial community revealed by comparative genomic and metagenomic analyses." *The ISME journal*, vol. 1, no. 8, pp. 703–713, 2007.

[84] A. Mitchell, F. Bucchini, G. Cochrane, H. Denise, P. t. Hoopen, M. Fraser, S. Pesseat, S. Potter, M. Scheremetjew, P. Sterk, and R. D. Finn, "Ebi metagenomics in 2016 - an expanding and evolving resource for the analysis and archiving of metagenomic data," *Nucleic Acids Research*, vol. 44, pp. D595–603, Jan 2016.

[85] Apache, "Cassandra," [accessed 18-April-2016]. [Online]. Available: http://cassandra.apache.org

[86] "Mongodb," [accessed 18-April-2016]. [Online]. Available: http://mongodb.org

[87] B. Nicolae, G. Antoniu, L. Bougé, D. Moise, and A. Carpen-Amarie, "Blobseer: Next-generation data management for large scale infrastructures,"

*Journal of Parallel and Distributed Computing*, vol. 71, no. 2, pp. 169 – 184, 2011.

[88] F. Schmuck and R. Haskin, "Gpfs: A shared-disk file system for large computing clusters," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, ser. FAST '02. Berkeley, CA, USA: USENIX Association, 2002.

[89] Panasas, "Parallel Network File System," [accessed 18-April-2016]. [Online]. Available: http://pnfs.com

[90] Oracle, "MySQL," [accessed 18-April-2016]. [Online]. Available: http://www.mysql.com

[91] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: interactive analysis of web-scale datasets," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 330–339, 2010.

[92] M. Hausenblas and J. Nadeau, "Apache drill: Interactive ad-hoc analysis at scale," *Big Data*, vol. 1, no. 2, pp. 100–104, May 2013.

[93] J. Schildgen, T. Jorg, M. Hoffmann, and S. Dessloch, "Marimba: A Framework for Making MapReduce Jobs Incremental," in *2014 IEEE International Congress on Big Data*. IEEE, Jun. 2014, pp. 128–135.

[94] V. Serbanescu, F. Pop, V. Cristea, and G. Antoniu, "Architecture of distributed data aggregation service," in *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, May 2014, pp. 727–734.

[95] F. Douglis and A. Iyengar, "Application-specific Delta-encoding via Resemblance Detection," in *Proc. of the USENIX Annual Technical Conference*, 2003, pp. 113–126.

[96] D. M. Dooley, A. J. Petkau, G. Van Domselaar, and W. W. L. Hsiao, "Sequence database versioning for command line and galaxy bioinformatics servers," *Bioinformatics*, vol. 32, no. 8, pp. 1275–1277, Dec. 2015.

[97] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation*, ser. OSDI'04, vol. 6. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[98] S. Killcoyne and A. del Sol, "Figg: Simulating populations of whole genome sequences for heterogeneous data analyses," *BMC Bioinformatics*, vol. 15, p. 149, 2014.

[99] B. D. O'Connor, B. Merriman, and S. F. Nelson, "Seqware query engine: storing and searching sequence data in the cloud," *BMC Bioinformatics*, vol. 11, p. S2, 2010.

[100] A. Roberts, H. Feng, and L. Pachter, "Fragment assignment in the cloud with express-d," *BMC Bioinformatics*, vol. 14, no. 1, pp. 1–9, 2013.

[101] 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," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1414–1425, Aug. 2009.

[102] Driven, "Cascading," 2016, [accessed 18-April-2016]. [Online]. Available: http://www.cascading.org

[103] Apache, "Storm," 2016, [accessed 18-April-2016]. [Online]. Available: http://storm.apache.org

[104] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi, L. Kuff, D. Kumar, A. Leblang, N. Li, I. Pandis, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-Milne, and M. Yoder, "Impala: A Modern, Open-Source SQL Engine for Hadoop," in *CIDR*. www.cidrdb.org, 2015.

# Part II

# Collection of publications

# /7

# Papers

## 7.1 Paper 1

Robertsen, E.M., Kahlke, T., Raknes, I.A., Pedersen, E., Semb, E.K., Ernstsen, M., Bongo, L.A., Willassen, N.P.: Meta-pipe - pipeline annotation, analysis and visualization of marine metagenomic sequence data (2016) arXiv:1604.04103

## 7.2 Paper 2

E. Pedersen and L. A. Bongo, "Large-scale Biological reference database Management," in *Future Generation Computer Systems*, in Press.

## 7.3 Paper 3

E. Pedersen , I. A. Raknes, M. Ernstsen , and L. A. Bongo, “Integrating Data Intensive Computing Systems with Biological Data Analysis Frameworks,” in *Proc. of 23rd Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2015, pp. 733–740.

## 7.4 Paper 4

L. A. Bongo, E. Pedersen, and M. Ernstsen, "Data-Intensive Computing Infrastructure Systems for Unmodified Biological Data Analysis Pipelines," in *Computational Intelligence Methods for Bioinformatics and Biostatistics*, ser. LNBI, vol. 8623, 2014.

## 7.5 Paper 5

E. Pedersen and L. A. Bongo, *Resource Management for Big Data Applications*. Springer, 2016, ch. Big Biological Data Management. In Press.