Large-scale Biological Meta-Database Management

Edvard Pedersen  
Department of Computer Science  
University of Tromsø  
Tromsø, Norway  
edvard.pedersen@uit.no

Lars Ailo Bongo  
Department of Computer Science  
University of Tromsø  
Tromsø, Norway  
larsab@cs.uit.no

Abstract— Up-to-date meta-databases are vital for the analysis of biological data. The current exponential increase in biological data is also exponentially increasing meta-database sizes. Large-scale meta-database management is therefore an important challenge for platforms providing services for biological data analysis. In particular, there is a need either to rerun an analysis with a particular version of a meta-database, or to rerun an analysis with an updated meta-database. We present our GeStore approach for biological meta-database management. It provides efficient storage and runtime generation of specific meta-database versions, and efficient incremental updates for biological data analysis tools. The approach is transparent to the tools, and we provide a framework that makes it easy to integrate GeStore with biological data analysis frameworks. We present the GeStore system, as well as an evaluation of the performance characteristics of the system, and an evaluation of the benefits for a biological data analysis workflow.

Keywords—bioinformatics; big data; hadoop; data-intensive computing; metagenomics

I. INTRODUCTION

Recent advances in scientific instruments, such as next-generation sequencing machines, have the potential of producing data that provides views of biological processes at different resolutions and conditions, opening a new era in molecular biology and molecular medicine [1]. Many of the data analysis techniques developed for analyzing such biological data integrate data from many experiments with metadata from multiple knowledge bases. The information in the meta-databases [2] is essential for understanding the biological content of the experiment data. For example, the results of DNA sequencing may not become truly useful before the UniProtKB [3] meta-database is used to map sequence bases to genes, the gene expression results are compared to results from other experiments, and the differences in expression values have been mapped to biological functions using the GO [4] meta-database.

The low cost of next-generation sequence machines and other biotechnological instruments have caused an exponential growth of biological data [5]. Analysis of all this data produces many results, which are added to meta-databases such as UniProtKB. Such meta-databases are frequently updated and therefore growing rapidly (Figure 1). For example, the August 2014 release of UniProtKB/TrEMBL contains 82,126,897 entries and is 194 GB in size. Compared to the previous July 2014 release, the number of entries increased by 3%, and 26% of the entries were updated. Each update may provide novel insights by reanalyzing old experiment data [6]. Updating experiment data with new meta-data is especially important for servers that provide search analysis services based on integrated data analysis [7].

For many analyses, it is also important that a specific meta-database version is used. For example, it is common to compare analysis results against “gold standard” results that are often calculated using a specific meta-database version. There are four main requirements for an infrastructure system that maintains large-scale biological meta-databases. First, the system should enable efficient methods for integrating biological compendium with new or updated meta-data, since the computational cost of the integration can be orders of magnitude larger than the cost of producing the data [8]. Second, multiple versions of the meta-database must be maintained in order to ensure repeatability of the analysis. Such repeatability is a cornerstone in the scientific process, but has often been hard to achieve in bioinformatics [9]. Third, the system must be transparent to data analysis tools since is not practical to implement and maintain modified versions of the many analysis tools used in biological data analysis [10]. Fourth, the system must easily integrate with biological data analysis frameworks to ensure

Figure 1: Number of entries in UniProtKB from July 2011 to August 2014.
adaptation in production systems.

Current popular biological data analysis frameworks such as Galaxy [9], Taverna [11], and Bioconductor [12] do not satisfy the first two requirements. Meta-data versions are manually maintained and specified by the analyst. In addition, meta-database updates typically require re-executing the analysis for each meta-data update. Such full updates increase the computational time and cost, often to the point where reanalysis is not done.

Incremental update systems [13] for large-scale data [14]–[19] solve the first two requirements. These systems maintain several versions of the experiment data compendia and meta-databases, and greatly reduce the cost of reanalysis by using incremental updates that limit 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 [20], MapReduce [21], or Spark [22]) or implementing ad-hoc scripts for input generation and output merging.

Data warehouse approaches for biological data, such as Turcu et al [23], may provide incremental updates for specific tools, but do not easily allow adding new tools, nor integrating with biological data analysis frameworks.

We present the GeStore system for large-scale biological meta-database management. It satisfies all four requirements listed above. GeStore provides an efficient transparent file based approach for incremental updates. It uses HBase to implement distributed data structures with efficient compression for multiple versions of large meta-databases. It uses Hadoop MapReduce for scalable parallel generation of specific database versions and increments. The transparent approach enables easy integration of GeStore with data processing frameworks, and does not require any changes to data analysis tools.

Our contributions are threefold:
1. We describe the design and implementation of a system for biological meta-database management that enables efficient transparent incremental updates for large-scale biological data analysis workflows.
2. We demonstrate how the approach can be integrated with biological data analysis frameworks with minimal changes to the framework code, and no changes to data analysis tools.
3. We present experimental evaluation of the performance, overhead and resource usage of the approach using a biological analysis workflows and real large-scale meta-databases.

Summarized our findings show that large-scale biological meta-databases can be efficiently maintained using data-intensive computing systems, and that our approach can easily be integrated with biological data analysis frameworks.

II. BACKGROUND

We provide a background describing biological data analysis implementation, configuration, and execution. We use one of our biological data analysis workflows as a case study. Additional examples are in [24], [25].

A. Data analysis workflows

A computer system for analyzing biological data typically consist of four main components: input data, meta-data, a set of tools in a workflow, and finally a data exploration tool (Figure 2). Biotechnology instruments such as short-read sequencing machines produce the input data. The data can also be downloaded from public repositories such as GEO [26], ArrayExpress [27] and Trace Archive [28]. There are hundreds of meta-databases with human or machine curated meta-data extracted from the published literature and analysis of experimental data [2]. 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 to 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 [10]. There are many libraries [9], [12], [29] with hundreds of tools, ranging from small, user-created scripts to large, complex applications.

B. Workflow managers

The analyst specifies, configures, and executes the workflow using a workflow manager. 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, meta-databases, output files, and possibly intermediate data.

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

1) Case study: GePan.

Our biology collaborators initially implemented GePan to run the Meta-pipe workflow described in section IV.A. GePan describes an analysis workflow as a set of shell scripts generated from a set of parameters specified by the user. These parameters specify the tools to run and their parameters. It will generate two kinds of shell scripts: 1) a set of job scripts, 2) a job submission script. GePan executes the analysis tools in a workflow in parallel on a compute cluster using the Open Grid Engine (OGE) or Torque.

![Figure 2: A biological data analysis workflow](image-url)
The job script created temporary directories, runs tools, saves results and deletes temporary files. The job submission script submits job scripts to the job manager and maintains dependencies between jobs.

GePan represents workflow managers written for a specific pipeline and a traditional HPC cluster. Such workflow managers are common since they are easy to implement, but they do not provide advanced features such as provenance data management.

C. Hardware platforms

The workflow manager typically executes the workflow on a fat server, high performance computing clusters, or a data-intensive computing cluster.

Workflow managers such as Galaxy [9] are typically run on a single server. There are two main advantages. First, most biological analysis tools can be used unmodified. Second, it is not necessary to distribute and maintain tools on a cluster. The main disadvantage is the lack of scalability, both concerning dataset size and the number of concurrent users.

Script based workflow managers, such as GePan, often execute the workflows on a high performance computing (HPC) cluster. Many biological data analysis tools can easily be run on such platforms by splitting the input (or meta-data) into many files that can be computed in parallel. The main advantage of using an HPC cluster is their availability. The main disadvantage is that the centralized storage system often becomes a bottleneck for production size datasets.

Systems such as Troilkatt [25] are designed to execute workflows on clusters built for data-intensive computing [30]. Compared to HPC clusters these have storage distributed on the compute nodes, and data processing systems that utilize such distributed storage. The main advantage is improved performance and scalability for I/O bound jobs. The main disadvantage is that to fully utilize such a platform applications may need to be modified [10], [24], [25]

III. GeStore

Our approach for large-scale meta-database management is based on the GeStore system [31]. We built GeStore to enable transparent incremental computations for unmodified file-based data analysis workflows. In addition, we use the same approach and mechanisms to generate specific versions of large meta-databases efficiently.

GeStore consists of a runtime system that provides a plugin framework for incremental input file generation and output file merging, a framework for parsing and detecting changes in files, distributed data storage, and parallel processing. GeStore uses HDFS [30] and HBase [32] to efficiently store meta-databases, and Hadoop MapReduce [21] to efficiently generate a specific version of a meta-database. In addition, GeStore provides library functions and tools to add incremental updates to workflows, and client applications to administer the data maintained by GeStore.

GeStore exports two interfaces (Figure 3). The data feeder [33] interface is used by external programs that periodically download updated experiment and meta-data from public repositories and databases and insert it into GeStore. A workflow manager calls the workflow manager interface before running a workflow tool to generate incremental input files, and after running a tool to merge the resulting files with the compendia. The workflow manager uses the same interface to generate a specific version of a meta-database.

A. File Based Incremental Updates

GeStore uses a transparent file based approach to implement incremental updates by generating input and meta-data files for biological data analysis tools that only contain data for an incremental computation. The tool will then be run, as normal, but it will typically produce a partial result. GeStore then merges the partial result with previously produced results.

We have chosen a file-based approach since relatively few file formats are used by many genomics applications. 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 Basic Local Alignment Search Tool (BLAST [34]). It calculates a similarity score for all gene sequences in an input file by comparing each sequence to all sequences in the UniProtKB [3] meta-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 are 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 naïve diff will generate too many records. For example, BLAST results are not affected by the annotation record fields that are most frequently changed in the
the UniProtKB database. It is therefore necessary to write tool specific change detection code that finds changes only 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 [23]) that is incorrect for incremental updates. In such cases, the file merger tool must fix these values in the updated output files.

B. Storage

GeStore maintains versioned meta-database files, input files, and output files. GeStore uses the version information to generate incremental files, or a specific version of a meta-database. In addition, the version information is required to merge incremental update results with previously computed results. We use the Hadoop software stack for scalable storage and data processing.

There are two types of files maintained by GeStore: parsed and unparsed. Parsed files are stored as HBase tables, while unparsed files are stored unmodified in HDFS.

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 meta-database. 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 HBase table even if the file format or the parser code is changed.

We use the HBase timestamp mechanism to manage meta-database versions. The timestamp either represent the file generation date, release date or version of the meta-database. By storing updated data in timestamped HBase cells, we can efficiently compress a set of database versions using delta compression.

C. Processing.

GeStore provides four operations on the meta-database tables: create, update, get increment, and get version.

To create a table for a new meta-database, GeStore first checks if the table exists. If not, it will create an empty table, with one empty column family. The meta-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 meta-database. Each entry in the new meta-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 $T_{last run} - T_{current time}$. 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, and the record has a current EXISTS field, the relevant record fields are written to a file on HDFS.

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

D. GeStore Data Structures.

GeStore maintains three HBase tables with internal data structures. The updates table has a row with information about each update for every file. This includes the number of entries in the file and the version of the update. The runs table contains rows for each run of the workflow containing fields for 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. The files table contains lists of generated files. This table is used for cache lookups and for determining if a file can be generated from HBase, or if it is stored unparsed in HDFS.

E. Meta-Database Caching

GeStore implements a cache of previously generated meta-database files, since many workflows can share these. The files are stored in HDFS, and the filename is used to store information about how the files were generated. We store both big (multi-gigabyte) and small (megabyte) files, since the overhead of 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 the files table, 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 meta-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.
F. Plugin Framework

To add transparent incremental updates to a pipeline tool the workflow maintainer must implement: (i) a parser for each file type used by the pipeline tool, (ii) tool-specific file generator, and (iii) tool-specific incremental output file merger. In GeStore, these are implemented as a plugin, and managed by the GeStore plugin framework.

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

The GeStore plugin framework is responsible for executing the code for file management, parsing, generation and merging. The framework is invoked by a workflow manager or data feeder that respectively calls one of the two exported functions: generateFiles or mergeFiles. 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.

1) File parser.

The file parser must determine the structure of input files and meta-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 table format so it is usually easy to implement a converter.

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

2) 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 coarse grained, where the contents of an entire file is compared, or fine grained where individual records are compared. For the latter, the change detection can use delta detection by taking 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 that specifies the parsers to use for each file format, and the fields to write to the input file using the associated file parser.

3) Output merger.

The output merger is responsible for merging the results of an incremental computation with previously generated output stored in GeStore. 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.

The output merger requires implementing a method that adjust any errors introduced by the incremental computation (e.g. adjusting e-values for BLAST).

G. Workflow Manager Integration.

The workflow manager must call GeStore to execute the plugin for a tool. GeStore provides a minimal interface with two functions that the workflow calls to request a specific meta-database version, one or more incremental update input files, and to merge the partial results with previously produced results. In addition, GeStore also provides a semi-POSIX file system interface, which can be used instead of file system calls in the workflow manager directly. This interface is more suited to low-level integration. This interface allows the workflow manager to use GeStore as a file system, including support for paths, movement, removing files, directory listing etc.

IV. PLUGINS AND INTEGRATION

In this section, we describe how we added incremental updates to the Meta-pipe metagenomics analysis workflow. Integration approaches for two additional workflows and workflow managers are described in [24].

A. Meta-pipe

Meta-pipe is a DNA sequence analysis workflow used by our biology collaborators to find novel commercially exploitable enzymes from marine microbial communities. These communities are not well explored, so a meta-database update can significantly change the analysis results. Meta-pipe takes as input assembled reads from environmental marine samples. The analyst uses tools such as METArep [35] to visualize and explore the workflow output results.

We use a smaller version of the full Meta-pipe workflow for our application benchmarks.

The version of Meta-pipe we use comprise the following tools:

- MetaGeneAnnotator (MGA) [36]: predicts genes in metagenomic sequences by looking for start sites and ribosomal binding sites.
- MGA-exporter: a Perl script that converts the MGA output files to the format used by the next stage.
- FileScheduler: is a python script that partitions and distributes the input data to the compute nodes.
- Protein BLAST (BLASTP) [34]: maps sequences to meta-data found in the UniProt Swiss-Prot and TrEMBL [3] databases.

B. Plugins.

To add incremental updates to Meta-pipe we implemented parsers for the four file formats used by the workflow: FASTA, UniprotKB meta-data, BLAST output, and MGA output. We also implemented plugins for the
BLAST tool. The BLAST plugin corrects incremental e-values as discussed in [23] during merge.

The file format plugins were 580 lines of Java code, and the tool plugins were 260 lines of Java code. The results show that file based incremental could be used for all tools in Meta-pipe, and there are relatively few lines of tool specific code.

C. Workflow manager.

GePan uses the key-value store interface exported by GeStore. We have modified the GePan script generation to replace file copy operations in the job scripts with GeStore calls. GePan sets the GeStore arguments at runtime without user input beyond an argument to GePan that specifies that GeStore should be used. This includes the workflow ID, tool to execute, tool input data, and the meta-databases to use. GePan can also determine if the file retrieved is a meta-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 should be treated as regular files.

In addition, the user may provide a filter to generate a subset of meta-database (for example for only one biological taxon).

The development effort for the integration is high. 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. We did not modify any of the Meta-pipe tools.

D. Database Version Use Case

A specific version of a meta-database can be retrieved by requesting it in the call to GeStore. In our integration, this is supported in Meta-pipe supplying a parameter when the user starts GePan.

V. EVALUATION

In our experimental evaluation of GeStore, we answer the following questions:

- What are the performance and resource usage characteristics of GeStore?
- How much can incremental updates using GeStore reduce workflow execution time?

We evaluate the first question to understand how to deploy GeStore in a production system, including how it will interfere with other applications running on the same system. The answer to the second question demonstrates the usefulness of GeStore for biological analysis workflows that must periodically update their analysis results.

A. Methodology

We characterize GeStore performance and resource usage using benchmarks for each of the GeStore operations exported to workflow managers. We also use the Meta-pipe workflow (described in section IV.A) as an application benchmark.

We report the average execution time of the benchmarks and the application benchmark. Each experiment is repeated 3 times. The standard deviation was less than 5% for all experiments and is therefore not reported.

We use the Ganglia Monitoring System [37] 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 Hadoop servers, and eight 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 nodes have a 1-gigabit Ethernet interconnect. We assume the cluster size and setup is realistic for a small cluster in a production environment. We also believe that it is realistic to dedicate such a small cluster for meta-database management.

The software used is 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.

B. Add Meta-Databases

We first measure the time and resource usage of adding a new meta-database to GeStore. We assume new meta-databases are rarely added, and that it can be done in the background. We are therefore primarily interested in the resource usage during the operation.

We first downloaded the November 2014 release of the UniProtKB meta-database from URL using FTP. The database is stored as a file compressed using gzip. The

![Figure 4: Megabytes received per second for each node in the cluster when adding a 240GB meta-database to GeStore](image)
compressed file size is 43GB. The file was uncompressed on the cluster frontend. The uncompressed file size is 240GB.

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 89 million parsed entries to an empty HBase table.

The GeStore add takes 191 minutes (TABLE I.), while downloading and unpacking the data takes respectively 62 and 33 minutes. We consider this acceptable, due to our assumption that this is a background operation.

The frontend has low load during download, and close to 100% load on a single core during decompression. These will therefore not interfere with other jobs on the cluster. In addition, we could use dedicated data feeder nodes if needed.

The GeStore-specific part of the operation is limited by network bandwidth. The CPU load is around 50% on all cores on all compute nodes for the MapReduce stage of the retrieval, and much less for the copy to HDFS.

Figure 5 shows the MB/s received by each node in the cluster when adding a meta-database to GeStore. The MB/s sent is similar. The network throughput is about 80% of the maximum aggregate throughput when stressing the network fully with netperf. This means that network-intensive jobs are expected to be more interfered with than CPU-intensive jobs while adding meta-databases to GeStore.

We consider this operation to scale well as more nodes are added to the cluster, since our workload depends on the aggregate network bandwidth.

| Add 2014 11 UniProtKB | 191 minutes |
| Add 2014 09 UniProtKB | 182 minutes |
| Update to 2014 10 UniProtKB | 144 minutes |
| Retrieve UniProtKB and generate BLAST database | 80 minutes |
| Retrieve cached results | 12 minutes |

C. Update Meta-Database

We also measured the time to update an existing meta-database in GeStore with new data. We assume that meta-databases are updated at most weekly and that this can be done in the background.

We first added the September 2014 release of UniProtKB to GeStore (84.5 million entries) and then updated it with entries that were updated in the October release (37 million of 87 million entries). The MapReduce job compares each entry in the downloaded meta-database with the old entries in HBase, and then puts updated entries to the existing HBase tables.

The GeStore update meta-database time is 144 minutes. This is about 25% faster than adding a new meta-database. This can be due to less HBase put operations, or that the puts are balanced across more region servers. We assume the scalability and interference with other jobs is similar to the add operation.

D. Retrieve a Meta-Database Version

Retrieving a specific meta-database version is a frequent operation that is in the critical path for the workflow analysis time. It is therefore important that this operation has a low execution time. In addition, we may want to overlap this operation with other jobs on the cluster.

We measured the time to retrieve the November 2014 version of the UniProtKB meta-database from GeStore (240 GB uncompressed, 89 million rows); including the time to convert the database to a BLAST-compatible format (the resulting file is 32 GB). GeStore runs a MapReduce job, where the map stage retrieves the relevant fields of each row, and a single reduce stage gathers the entries and writes these to a single file. This file is transferred to a single node, and formatted with the formatdb tool. Finally, the resulting file is copied to the HDFS cache.

The retrieval time is 80 minutes (TABLE I.). The MapReduce job contributes 31 minutes (39%) to the execution time (Figure 6). In the MapReduce job, the last mapper completes after 14 minutes. The reducer execution is partially overlapped with the mappers. MapReduce job performance is limited by HBase reads. The map part scales well, but the overall scalability is limited by the plugin specific parts (Figure 6 shows the CPU usage of all the nodes when generating the meta-database).

The plugin-specific parts: formatdb, copy to HDFS and copy to local disk, contribute with respectively 45%, 10% and 6% to the runtime. These, and the single reducer, dominate the execution time and limits scalability, but are necessary for legacy biological tools that require the database to be stored in a single file on network- or local file systems.

Cached databases are retrieved in 12 minutes, which is the time it takes to copy a 32 GB meta-database from HDFS to local disk.

We believe the time to generate databases is reasonable since they are typically used by workflows with execution times ranging from several hours to days. It is also possible to cache many of the generated meta-database versions since many tools in the same analysis may use these.

Except for the mapper part of the job and when copying to and from HDFS, this operation will not interfere with other CPU- or network-intensive jobs running on the cluster, as large parts of the load are limited to a single node. The
retrieve can therefore be overlapped with execution of another workflow.

E. Create an Incremental Meta-Database

We measured the time to create a meta-database for an incremental update. We assume many workflows will use GeStore for incremental updates. This operation is also in the critical path for analysis time.

We measured the time to generate an incremental database that contains the UniProtKB entries that were changed between the September 2014 and October 2014 releases. The MapReduce job is similar to the one described in the previous section, except that only updated fields are written to the final output file. The resulting incremental database is 1 GB in size and has 2.7 million entries.

The incremental database is generated in 9 minutes (TABLE II.). Figure 7 shows the CPU use while generating the database. The single-node reducer is less of an issue in this case, since the amount of data to write is smaller and hence, the runtime is mostly overlapped with the mappers. Formatdb still contributes the most to the runtime. However, the map stage is more significant, since it does more work for comparisons.

The cached version of this database is retrieved in 26 seconds, since it is relatively small.

The scalability of this use case is similar to the scalability of the full version use case. Formatdb still contributes most to the runtime.

The interference of this use case is also similar to the full use case, but the CPU use is higher during the mapping stage. Therefore, CPU-intensive jobs will be more perturbed.

F. Create and Split Meta-Database

In the experiments in the previous sections, the performance of the retrieve meta-database operations were limited by the need to format and write the meta-database to a NFS or local file system. In this section, we measure retrieve time for tools that can utilize data parallelism and HDFS. For these, GeStore splits the generated meta-database, and does not copy the files to local disk.

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

The split reduces the generation time to 9 minutes (TABLE III.), compared to the 55 minutes it takes to generate the same file using a single reducer and copying the results to local disk (a five-fold speedup). The cached version can be read directly from HDFS, and therefore incurs no overhead. The split also scales well, since there is no single node bottleneck.

These results show the advantage of GeStore if used by biological 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.

| TABLE II. | GENERATINGINCREMENTALMETA-DATABASES |
|-----------|-------------------------------------|
| Get incremental UniProtKB, and generate BLAST database | 9 minutes |
| Get cached incremental UniProtKB | 26 seconds |

| TABLE III. | CREATE AND SPLITMETA-DATABASE |
|------------|-------------------------------|
| Get file | 55 minutes |
| Get cached file | 10 minutes |
| Generate split file on HDFS | 9 minutes |
| Get HDFS path of generated files | 2 seconds |

G. Application benchmarks

To evaluate the benefits of GeStore incremental updates for a real-world biological analysis workflow, we used Meta-pipe (described in IV.A). Meta-pipe scales linearly with respect to input data and meta-database sizes. Hence, a reduction in meta-database will reduce the execution time approximately linearly with the size of the meta-databases used. If this reduction is more significant than the GeStore overhead for generating the meta-databases, we will get a reduction in execution time.

We measure the execution time of Meta-pipe. We use a small (15 MB) input, and measure the time it takes to do MGA, mga exporter, fileScheduler and BLAST. We do not include the steps of the workflow after BLAST. We use the 2014_09, 2014_10 and 2015_01 versions of UniProtKB. A larger input file would increase runtime linearly, so a small input was chosen to finish the benchmarks in a timely manner. With a large input file, the overhead of GeStore compared to workflow execution time would be lower.

GeStore adds an overhead of 132 minutes when generating a meta-database, and 26 minutes when the meta-database is cached (TABLE IV.). 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 13-fold speedup compared to a full re-analysis. These results show that incremental updates through GeStore can provide large benefits to biological analysis workflows.
H. Discussion

In our benchmarks, we use a small 10-node cluster; we observed that the UniProtKB plugin limits scalability. However, when using a plugin that can take advantage of the parallelism of the system, we see much higher potential for scalability. Where data-parallel plugins can be used, a larger cluster would result in performance benefits.

We observe that the resource usage of GeStore depends on both the plugin used, and the operation performed. Adding meta-databases to GeStore puts heavy load on the network, while retrieving databases puts more load on the CPU.

The overhead of GeStore is small when we consider the runtime of a typical workflow. However, workflows with very short runtime GeStore incremental updates will not reduce execution time. However, these may still benefit from meta-database versioning.

GeStore provides many benefits to users, in the form of enabling incremental updates for unmodified tools, using simple operations. Other benefits we have not examined in detail in this paper are the provenance recording and data management services provided by GeStore.

There are several avenues for optimizing the performance of GeStore. It may be possible to better tune HBase, HDFS and MapReduce parameters. One example is to relax 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 meta-databases to GeStore will be a relatively infrequent background job, we consider the additional execution time to be less important than in usage scenarios where this operation type is frequent.

| TABLE IV. APPLICATION BENCHMARKS, GEPan WITH GeSTORE |
|-------------------------------------------------------|
| Without GeStore | 833 minutes |
| With GeStore   | 965 minutes |
| With GeStore, cached DB | 859 minutes |
| 1-month incremental update | 61 minutes |
| 4-month incremental update | 99 minutes |

VI. RELATED WORK

Systems and frameworks for incremental updates on large scale datasets include Incoop [15], Percolator [16], Nectar [14], DryadInc [17], Marimba [38], HaLoop [19] and Spark [22]. In Percolator, the programmer implements a system specific incremental program using respectively event-driven mini transactions and stateful primitives. Incoop, Nectar, DryadInc, and HaLoop use data dependency graphs of Dryad [20] or MapReduce programs to automatically replace the input data for a computation with previously calculated results. GeStore combines these two main approaches; a programmer implements file generators and mergers for unmodified programs. GeStore is independent of the programming model and job management system, so the applications can be executed using Spark, Dryad, MapReduce [21], or other models.

Simple change detection is supported by tools such as Unix diff, delta encoding compression systems [39], 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 [23] 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.

We chose to focus on GePan, since it is the workflow manager in our production. Other workflow managers for scientific workflows include Galaxy [9] and Taverna [11].

We evaluated GeStore using the locally developed Meta-pipe workflow. An alternative is the JCVI metagenomics analysis pipeline [41] or the EBI metagenomics analysis pipeline [42].

VII. CONCLUSIONS AND FUTURE WORK

We proposed an approach for efficient management of large-scale biological meta-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 using a real metagenomics analysis workflow and real data. Our findings show that large-scale biological meta-databases can be efficiently maintained using data-intensive computing systems, and that out approach can easily be integrated with biological data analysis frameworks.

We plan to deploy GeStore on our production systems and use it to produce data for tools such as IMP [31].

ACKNOWLEDGMENTS

Thanks to Espen Robertsen and Tim Kahlke for help with the GePan workflow, Jon Ivar Kristiansen for maintaining our cluster, Nils Peder Willassen for his biological insights and Martin Ernstsen for his input on tuning HBase.

REFERENCES

[1] S. C. Schuster, “Next-generation sequencing transforms today’s biology,” Nat. Methods, vol. 5, no. 1, pp. 16–18, 2008.
[2] 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.
[3] M. Magrane and U. Consortium, “UniProt Knowledgebase: a hub of integrated protein data,” Database J. Biol. databases curation, vol. 2011, 2011.
[4] M. Ashburner et al., “Gene Ontology: tool for the unification of biology,” Nat. Genet., vol. 25, no. 1, pp. 25–29, 2000.
[5] S. D. Kahn, “On the Future of Genomic Data,” Science (80- ), vol. 331, no. 6018, pp. 728–729, Feb. 2011.
[6] R. Sandberg and O. Larsson, “Improved precision and accuracy for microarrays using updated probe set definitions,” BMC Bioinformatics, vol. 8, no. 1, p. 48, Jan. 2007.
