Diploma Thesis Exposé

Collecting Execution Statistics of Scientific Workflow on Hadoop YARN

Author: Hannes Schuh

Supervisors: Ulf Leser, Marc Bux, Jörgen Brandt

The goal of this diploma thesis is to extend a scientific workflow management system running on top of Hadoop with a database for storing statistical information persistently. Such statistics are useful for many use cases, for instance for providing a scheduler with estimates on the runtime of future tasks or for further applications like progress and time-remaining estimation.

1 Introduction

The development of computers, sensors and other technical instruments changed scientific research. Current scientific experiments often contain a whole range of computational activities, e.g: capturing and saving huge amounts of data in databases or flat files, analysis of data with software and data visualization [1].
The complexity of these data intensive experiments led to new challenges. Developing and maintaining the physical infrastructure to store the data and execute the analysis pipelines is expensive [2]. Data provenance is also an important aspect - scientists need to ensure the origin of the results and the underlying data. They need to share the knowledge and resulting information with the community in order that other scientists can repeat and review the experiments.

Scientific workflows provide a means for representing and managing such analysis pipelines. An activity of the analysis pipeline is encapsulated in a workflow step (task). A scientific workflow is a directed, acyclic graph (DAG), in which individual tasks are represented as nodes. Edges represent data dependencies between tasks. One task can be local software or a remote (web) service call which transforms input data to output data. Task execution is constrained by data dependencies [3]. An *abstract workflow* is a high-level workflow description. Scientists model an abstract workflow by specifying a set of individual tasks and the data dependencies between them [4]. Figure 2 shows an abstract workflow.

Scientific workflow management systems (SWfMS) are used to manage and execute scientific workflows. In addition, they often support scientists to record provenance information and statistics about the execution of the workflows. Provenance data is supposed to trace the flow of data through the workflow tasks to reason about the results. Statistics about the execution of workflows and tasks are useful not only for provenance questions: Precise information about historical executions of a workflow can help the SWfMS to execute the workflow faster or to present progress and time-remaining estimation [5].

Besides complex analysis pipelines which can be managed by SWfMS, scientists have to handle the massive volume of data which is captured by today's technical instruments. For instance, the Australian Square Kilometre Array of Radio Telescopes Project[1] or CERN's Large Hadron Collider[2] generate several petabytes of data per day. In bioinformatics, next-generation sequencing makes sequencing DNA more affordable and faster. A single sequencer is capable of generating more than 5 TB[3] per day and this technology is already provided by many sequencing labs [6, 7]. It is more and more difficult to process the increasing amount of data on a single computer in decent time. Parallelization and distribution provide means to handle such data analysis. In [2] Bux and Leser present an overview of the current SWfMS and the way they implement parallelization and scheduling. Scheduling a scientific workflow means assigning the individual tasks of an scientific workflow to the available physical resource considering dependencies between the tasks. The usual goal is to minimize overall job completion time but other goals are also possible [8]. Many of the most well-performing schedulers make use of runtime estimations (execution time, data transfer time) of each task on each worker node [3]. To get runtime estimations for a task the scheduler can use historical statistics of previous runs of that task.

2 Motivation

Apache Hadoop has been developed as a distributed data processing system and an open source implementation of Google's MapReduce programming model. YARN (Yet Another Resource Negotiator) is the next generation of Hadoop's parallel compute platform [9]. The architecture of YARN separates two major duties: resource management and job scheduling/monitoring. Figure 1 shows the shift from Hadoop 1.0 to 2.0. Similarly to Hadoop 1.0, file management is handled by the Hadoop Distributed File System (HDFS). HDFS breaks data in several blocks and stores them redundantly throughout the distributed compute environment.

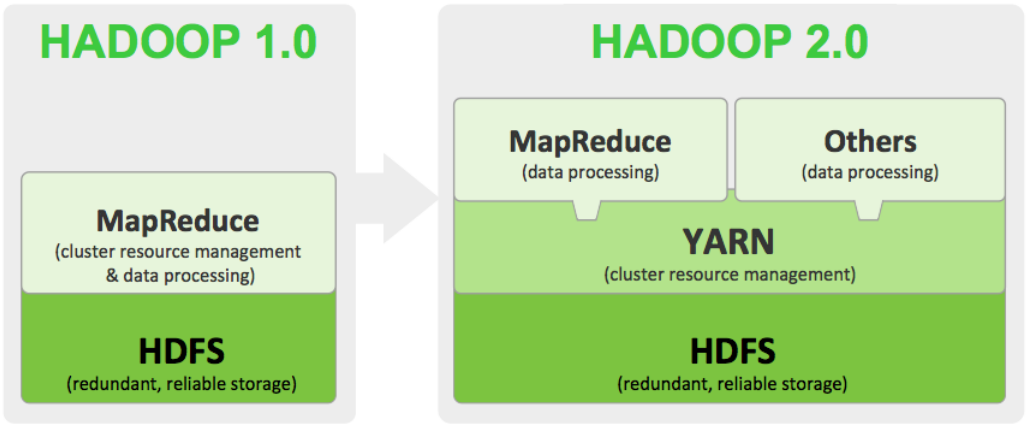

Figure 1: The YARN-based architecture of Hadoop 2.0. In this architecture MapReduce is just one of the applications running on YARN. (Image taken from http://hortonworks.com/hadoop/yarn/)

YARN's ResourceManager is responsible for resource management in the distributed compute environment. Applications in the system ask the ResourceManager for available physical resources and get assigned slots on worker nodes to run their computations. A per-application ApplicationMaster (AM) is responsible for the application's execution[4]. That involves negotiating resources from the ResourceManager, tracking their status and monitoring progress. Users can use different programming models and scheduling policies by implementing an ApplicationMaster running on

[1] www.ska.gov.au
[2] http://public.web.cern.ch/public/en/LHC/LHC-en.html
[3] http://blog.genohub.com/nextseq-500-and-hiseq-x-ten-services-coming-soon-to-genohub-com/
[4] http://hadoop.apache.org/docs/current2/hadoop-yarn/hadoop-yarn-site/YARN.html

top of YARN [9]. The MapReduce AM is provided as part of the Hadoop installation. In addition, there are already some frequently used AMs which implement a different programming model. For instance, *Apache Giraph*[5] which is a highly scalable iterative graph processing system.

2.1 The Hi-WAY AM for YARN

The research group of Knowledge Management in Bioinformatics at Humboldt-Universität zu Berlin, in cooperation with the Information and Communications Technology department of KTH Royal Institute of Technology Stockholm, develops a workflow management system on top of YARN. The SWfMS comprises an ApplicationMaster named Hi-WAY (Heterogeneity-incorporating Workflow ApplicationMaster for YARN), a scheduler (C3PO) and a workflow language (Cuneiform). Figure 2 shows an abstract workflow modeled and visualized in Cuneiform. Hi-WAY is capable of executing task parallel workflows. Task parallelism can be achieved by distributing the tasks comprising a scientific on parallel computation infrastructures like clusters or clouds [2]. Figure 3 shows a *concrete workflow* that has been executed by Hi-WAY. A concrete workflow is generated by mapping the abstract tasks to concrete methods.

2.2 Runtime statistics in Hi-WAY

Each invocation in Figure 3 traces its own statistical information and writes a log file into HDFS after the execution. Hi-WAY collects the log entries and concatenates them to a main log file during workflow execution. Statistics in the log file are not suited for easy access, since the file has no schema and is accessed sequentially. Applications have to parse the whole file and need to interpret the data even if they just want to query certain parts of the file. Hi-WAY does not use these statistics. Statistics in the current log files are only used by Cuneiform for the execution plan of the remaining tasks.

Hi-WAY collects its own statistics by monitoring the execution of the invocations, e.g. execution time and input file size. These statistics are only available in main memory and are lost after the execution of the workflow. Therefore Hi-WAY knows only the runtimes of the current execution and nothing about the previous ones. To determine future runtime estimates for a task, Hi-WAY make use of statistics about past invocations of that task on each worker node of the compute environment.

2.3 Log Data in Cuneiform

Each entry in the log represents a tuple, which consists of eight fields. A field can be a value (integer or string) or a complex type, e.g. a list of key-value pairs specified in JSON format. The log file of an invocation contains statistics and provenance information about the execution of a workflow task. The first seven fields are used for classification of the log entry. There are six types of possible log entries:

- invoc-stat : Statistics about the invocation run.
- invoc-stagein : All staged in files and their sizes.
- invoc-stageout : All staged out files and their sizes.
- invoc-output : The values of all output variables.
- invoc-stdout : The content of the standard output channel.
- invoc-stderr : The content of the standard error channel.

Figure 4 shows examples for three of these types of log entries of an invocation. Execution statistics, stagein and stageout information are the most important for the scheduler. No single field is a primary key. However, the invocation signature and the type together constitute a composite key.

[5] http://giraph.apache.org/

The size of the statistic traces depends on the workflow task and data input. For example, the workflow from Figure 2 and 3 produces a main log file of 1.5 MB. The SWfMS can run hundreds of workflows simultaneously, assuming that the cluster or cloud has appropriate capabilities. In these cases, the SWfMS can generate several GB of statistical data per day.

3 Goal of this Thesis

The goal of this diploma thesis is to extend the architecture of Hi-WAY by adding a database which saves statistical information persistently and independently from the AM. The database is supposed to supply the scheduler with latest and historical statistics about task executions as well as data transfer times on different worker nodes. Furthermore the information can be used for comparison between workflow executions, to answer provenance questions and to provide workflow progress estimation.

4 Approach

The important question of how to save the data persistently will be investigated in the diploma thesis. There are some requirements which narrow down the potential technologies we focus on. The requirements are determined by Hi-WAY, since the AM will use the database for scheduling.

4.1 Requirements

- **Low response time**: Hi-WAY waits for the statistics before assigning the invocation to a node. During this time the invocation can't run, which is why this time span needs to be as short as possible. SELECT statements to the statistic database should result in an negligible overhead for the AM. Fast SELECT statements are the main requirement.

- **Timeliness, Actuality**: Runtime estimates can be unreliable or quickly outdated in shared compute environments. In such settings the utilization of resources is unsteady and can change dynamically at runtime [11]. That means that older statistics are not reliable anymore, since the utilization of the cluster can already be different to the time the statistic was obtained. It is important that the current invocations write their statistic into the database as soon as possible, otherwise the AM can't use latest statistics. Each invocation is supposed to save their statistics into the database after their execution on the worker node. The need for actuality implies a kind of garbage collection for old and outdated log entries.

- **Read/Write ratio**: Once stored in the database, log entries won't have to be altered, the writes are append-only. Each time the scheduler determines which task to assign to a compute resource with available capacity, it reads parts of the log entries written by all tasks on this node. Therefore we can assume that there is a workload with many reads and just a few writes. The decision for a database technology should be focused on a system which provides fast reads. Fast writes or updates are substantially less important for our use case.

4.2 Persistence Technology

SQL databases has been the dominating persistence technology for many years. The challenges which have emerged in the last few years changed this situation. Many Web 2.0 startups started their business without SQL databases in order to address the needs of modern applications. The increasing volumes of data and the need for easy horizontal scaling on distributed compute environments have brought RDBMS to their limit. A number of specialized solutions for such use cases have emerged in the last few years. Non-relational databases have proven their usability here [12, 13].

4.2.1 SQL Database

The relational data model has been utilized for many years. Relational Database Management System (RDBMS) like Oracle database[6], Microsoft SQL Server[7] or MySQL [8] are the most popular database systems[9]. An RDBMS stores data in the form of tables with rows and columns. Due to the relational model, it is easy to add, update or delete data in an RDBMS. The ACID (Atomicity, Consistency, Isolation, Durability) properties guarantee that database transactions are processed reliably. The data can be easily related by foreign keys and joins, therefore the same database can be viewed or reassembled in many different ways [12]. Relational databases are well-known, mature and flexible, hence one of the databases to investigate will be an SQL database. The SQL database will be used as the baseline for further evaluations.
The relational data model suffers from a number of drawbacks. One of these drawbacks is the difference between the relational model and the in-memory object model. Mapping objects and classes of an object oriented concept to a relational data model leads to a set of conceptual and technical difficulties, which are called the *impedance mismatch*. For this reason, developers usually use an object-relational mapper to bridge the gap between objects and relations automatically [14]. The strict schema of a table can be another drawback, especially if the structure of data often changes or if the data is unstructured, this leads to high complexity and schema migrations. The rich feature set of SQL databases can be drawback as well. Join operations and transaction lead to inefficient and complex operations in distributed environments.

4.2.2 NoSQL

The term *NoSQL* encompasses a number of recent non-relational databases, which have some common characteristics but are still very different. A database system referring to NoSQL does not use the relational model, runs well on clusters, has no strong schema (schemaless) and is mostly an open-source project [14]. Example Projects are Riak[10], Oracle's Berkley DB[11], MongoDB[12] or Cassandra[13]. NoSQL databases are commonly classified by their data model: Key-Value, Document, Column Family and Graph databases [13–15]. Prof. Dr.-Ing. S. Edlich give an overview about current NoSQL databases on[14].
Graph databases are out of focus because they are specialized for handling data whose relations are well represented as a graph, such as social network data. All other systems have an aggregate-oriented data model. An *aggregate* is a collection of data with a key that is stored as a unit [14]. A log entry of an invocation can be seen as an aggregate. An aggregate can save complex types of values whereas a row in an RDBMS consists of a tuple of single values (which are related). An aggregate is the unit of work for a NoSQL database. Therefore data which is accessed together should be stored within one aggregate. Aggregates are useful to manage data storage over clusters, since they are a suitable unit for distribution [14]. Aggregate-oriented databases differ in the way they store data physically and this affects the way a user can access the data.

- *Key-Value* databases are simple hash tables where data is stored and accessed by a key. Behind a key there can be an arbitrary aggregate which is opaque to the system. It is not possible to query a part of the aggregate.

- *Document* databases are key-value databases whose value can be interpreted by the system. The value is a structured document, e.g. key-value pairs in JSON. A document database provides more query features than a key-value database.

[6] http://www.oracle.com/us/products/database/overview/index.html
[7] http://www.microsoft.com/en-us/sqlserver/default.aspx
[8] http://www.mysql.com/
[9] http://db-engines.com/en/ranking
[10] http://basho.com/riak/
[11] http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/index.html
[12] http://www.mongodb.org/
[13] http://cassandra.apache.org/
[14] http://nosql-database.org/

- *Column-Family* databases are motivated by Google Big Table and the need for a distributed storage system which can scale to a very large size [16]. Column-family stores are designed for distributing huge data sets with different kinds of attributes on large clusters [15]. That comes at the cost of a complex data model and less query features. Even if the SWfMS generates several GB of statistical data per day, the log data that we want to store cannot be described as a very large data set. We have a constantly small number of attributes and easy horizontal scaling is not important in our use case. Therefore, Column-Family stores are out of focus as well.

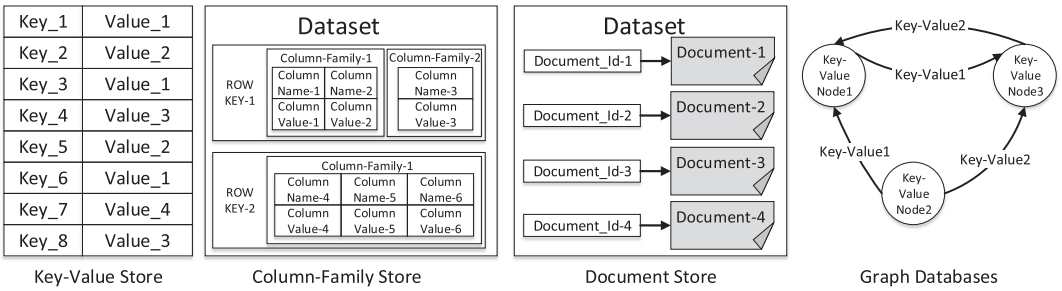

Figure 5: NoSQL data models [13]. Key-Value databases are simple hash tables where data is stored and accessed by a key. Document databases are key-value databases whose value can be interpreted by the system. Column-Family databases are key-value pairs (Column) which are grouped into a column family. Logical related data is stored in a row which consists of several column families and a row key. Graph databases are specialized on handling heavily linked data.

Following the ACID principles, RDBMS guarantee that a database transaction is processed reliably. A transaction is a set of data manipulation tasks that brings a database from one consistent state to another consistent state [17]. NoSQL systems have a different focus: In order to achieve better performance, availability and data distribution, they sacrifice consistency [14].

Aggregate-oriented NoSQL databases do not support transactions over multiple aggregates but provide atomic manipulation of a single aggregate at a time. Since an aggregate is the unit of work with the database it is very important to design a well-suited aggregate. Developers have to decide which data is accessed by the application as a unit. Naturally different applications manipulate the same data in different ways, which makes it difficult to design one suitable aggregate for different use cases. Evidently, NoSQL databases lack the flexibility provided by SQL databases.

4.3 Choosing a certain System

The statistic collector which will be implemented in this diploma thesis consists of two major parts, a lightweight logical layer and a data store. The logical layer is only responsible for saving log entries into the data store. It will run on each worker node and is called right after an invocation has finished. The data store is responsible for saving the statistics persistently and for answering queries from Hi-WAY. The investigation of different database systems, with focus on the requirements outlined above, will show which technology is suitable.

4.3.1 SQL

MySQL is the most used open-source database[15]. In combination with *memcached*[16] it is a good choice for applications requiring high scalability[17]. Memcached is an open-source, distributed memory caching system designed to speed up performance by caching data and objects in RAM. Many of the largest web sites like Facebook, YouTube, Yahoo, and Wikipedia deploy memcached

[15] http://db-engines.com/en/ranking
[16] http://memcached.org/
[17] http://highscalability.com/blog/2010/2/26/mysql-and-memcached-end-of-an-era.html.

and MySQL. MySQL Cluster has been part of the MySQL release since 2004. It works with a distributed layer called NDB. MySQL Cluster uses a 'shared nothing' architecture and distributes data over multiple servers. MySQL and memcached will be investigated in this diploma thesis as well as MySQL Cluster.

4.3.2 NoSQL

There are several surveys which compare NoSQL systems over a number of dimensions [13, 15, 18]. Besides data model the most important aspects are: Query capabilities, consistency, distribution model and performance. All surveys recommend systems for a certain use case and point out that NoSQL is a large and expanding field. Therefore, choosing a suitable data store for a given use case scenario is a challenging task [13].
It is most important to find a suitable data model for the given data to avoid unnecessary complexity due to the transformation or mapping of tasks. The data model is closely related to the query capabilities of a database. Therefore queries which should be supported by the database have to be taken into account.
Choosing a suitable NoSQL database involves trading between high performance through partitioning and load balanced replica servers, high availability supported by asynchronous replication and strict consistency [15].

All surveys state that key-value stores should be used for very fast and simple operations like storing session data or shopping cart data. Document stores should be used for applications dealing with data that can be easily interpreted as documents or for storing items of similar nature that may have different structures, e.g. content management systems or event logging. Due to their flexible data model, document stores offer more query possibilities.
A Key-Value database would be a reasonable choice, but fast SELECT statements is not the only requirement. A certain flexibility in accessing the data for different use cases like time estimation would be desirable. According to the recommendations in the literature [13–15, 18] a document database provides a reasonable trade-off between flexibility and scalability. The log data is perfectly suitable for a document database. Each message type can be interpreted as a document with a key. Programming complexity will be low because the data structure used in the log can be saved without transformation. Document databases combine the access performance of a key-value database with more query features.

DB-Ranking[18] mention that the most popular document databases are MongoDB[19], Apache CouchDB[20] and Couchbase[21]. While MongoDB and CouchDB are mature NoSQL databases, Couchbase was founded in 2011 by the former chief of Apache CouchDB.

Couchbase merged with an open source distributed key-value database called Membase to built a database which combines the indexing and querying capabilities of a document database with the high performance of a key-value data base[22]. Documents in Couchbase are stored as JSON objects. Couchbase provides low-latency read and write operations with linearly scalable throughput due to built-in caching [19]. According to latest benchmarks Couchbase performs very well not only in a heavy read scenario like the one described in Section 2 [19, 20]. In addition, Couchbase has several distribution possibilities which enable better read performance through load balancing. Couchbase will be the NoSQL database which is investigated in this diploma thesis.

5 Expected Results

This diploma thesis will investigate two database systems (MySQL and Couchbase) with different data models and evaluate their response times to Hi-WAY. Both technologies can be used on a

[18] http://db-engines.com/en/ranking
[19] http://www.mongodb.org/
[20] http://couchdb.apache.org/
[21] http://www.couchbase.com/
[22] http://www.couchbase.com/couchbase-vs-couchdb

single server or with distributed data, which will also be a test scenario. The absence of transactions in the classical sense and the limited query features of NoSQL databases makes them less flexible than SQL databases but they promise a better performance. Independent of the actual database technology there are some expected outcomes. Hi-WAY is supposed to be the main user of the statistics database and the technology should be suited to its needs. It is expected that Hi-WAY can use the database to get runtime statistics for every task of a workflow on each node in the compute environment. The information used by Hi-WAY is expected to be up to date and the overhead for the select statement should be negligible.

Additionally, the statistics are usable for the user of the workflows. They can compare several runs of the same workflow and find errors in workflow design. The provenance information in the database can be used to reason about results and to verify the underlying data. In the best case scenario the diploma thesis results in an application which provides time-remaining and progress estimation to the user of the SWfMS.

References

[1] Tony Hey, Steward Tansley, and Kristin Tolle. *The Fourth Paradigm: Data-Intensive Scientific Discovery*, volume 2. Mircrosoft Research, 2009.

[2] Marc Bux and Ulf Leser. Parallelization in Scientific Workflow Management Systems. *CoRR*, abs/1303.7195, 2013.

[3] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, Miron Livny, Luc Moreau, and Jim Myers. Examining the Challenges of Scientific Workflows. *Computer*, pages 24–32, 2007.

[4] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 2008.

[5] Kristi Morton, Magdalena Balazinska, and Dan Grossman. Paratimer: a progress indicator for mapreduce dags. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 507–518, New York, NY, USA, 2010. ACM.

[6] Elizabeth Pennisi. Will Computers Crash Genomics? *Science*, 331(6018):666–668, February 2011.

[7] Scott D. Kahn. On the future of genomic data. *Science*, 331(6018):728–729, 2011.

[8] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbree, Richard Cavanaugh, and Scott Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25–39, 2003.

[9] Vinod K. Vavilapalli. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings Symposium on Cloud Computing*, 2013.

[10] Li Bingshan and Suzanne M. Leal. Methods for Detecting Associations with Rare Variants for Common Diseases: Application to Analysis of Sequence Data. *The American Journal of Human Genetics*, 83(3):311–321, 2008.

[11] Marc Bux and Ulf Leser. Dynamiccloudsim: Simulating heterogeneity in computational clouds. *Int. Workshop on Scalable Workflow Enactment Engines and Technologies (SWEET'13), in conjunction with ACM SIGMOD Conference, New York, USA*, 2013.

[12] Nishtha Jatana, Sahil Puri, Mehak Ahuja, Ishita Kathuria, and Dishant Gosain. A survey and comparison of relational and non-relational database. *International Journal of Engineering Research & Technology*, 1, 2012.

[13] Katarina Grolinger, Wilson A. Higashino, Abhinav Tiwari, and Miriam A. M. Capretz. Data management in cloud environments: Nosql and newsql data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1):22+, 2013.

[14] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 1st edition, 2012.

[15] Robin Hecht and Stefan Jablonski. Nosql evaluation: A use case oriented survey. In *Proceedings of the 2011 International Conference on Cloud and Service Computing*, CSC '11, pages 336–341, Washington, DC, USA, 2011. IEEE Computer Society.

[16] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.

[17] Jim Gray. The transaction concept: Virtues and limitations (invited paper). In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7*, VLDB '81, pages 144–154. VLDB Endowment, 1981.

[18] A. B. M. Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics - classification, characteristics and comparison. *CoRR*, abs/1307.0191, 2013.

[19] Alexey Diomin and Kirill Grigorchuk. Benchmarking couchbase server for interactive applications. Altoros Systems, Inc http://www.altoros.com/, 2013.

[20] Denis Nelubin and Ben Engber. Ultra-high performance nosql benchmarking. Thumbtack Technology Inc., http://www.thumbtack.net/, 2013.

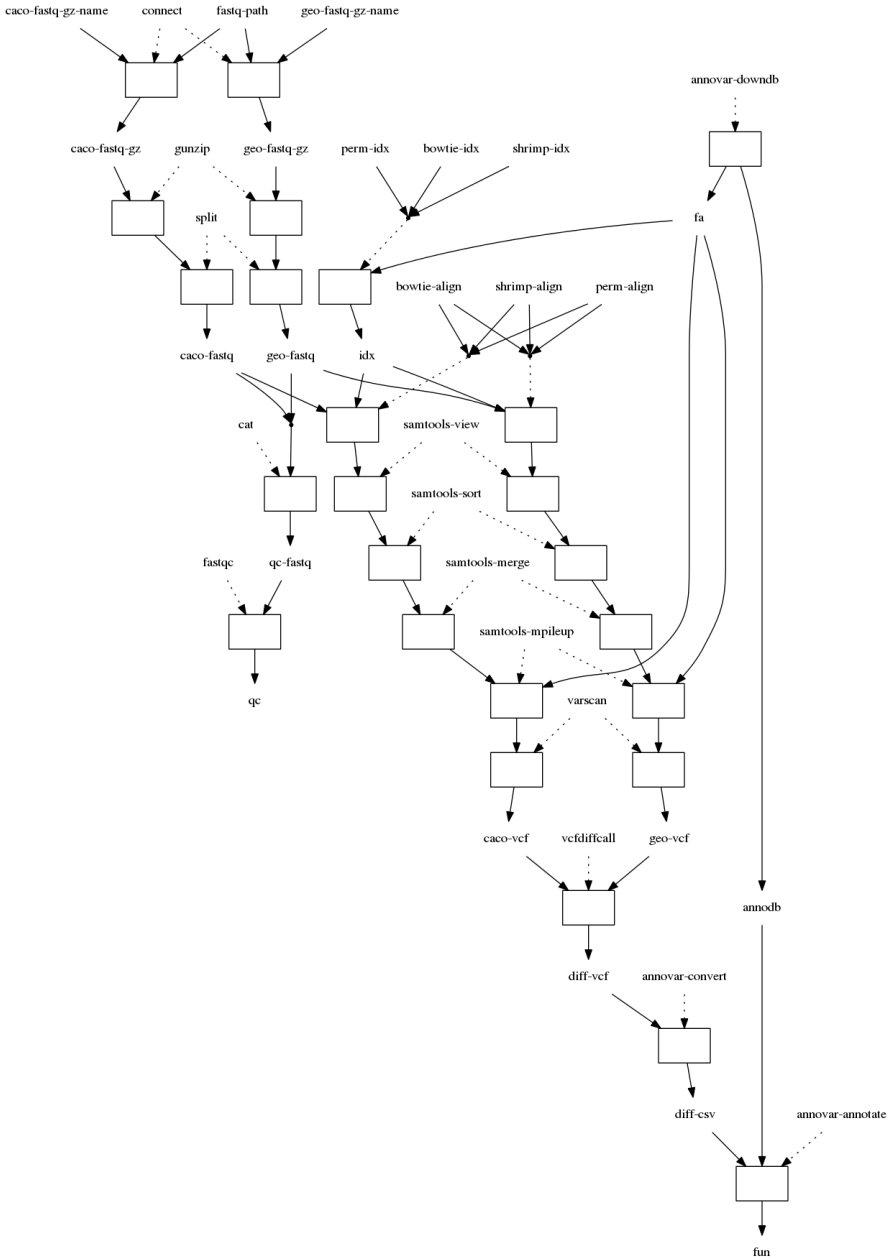

Figure 2: An abstract bioinformatics workflow which processes genomic sequencing data [10]. The workflow is modeled and visualized by Cuneiform. Boxes correspond to individual tasks named by dotted arrows. The solid arrows show data dependencies between tasks. It is possible to declare variables for intermediate results.

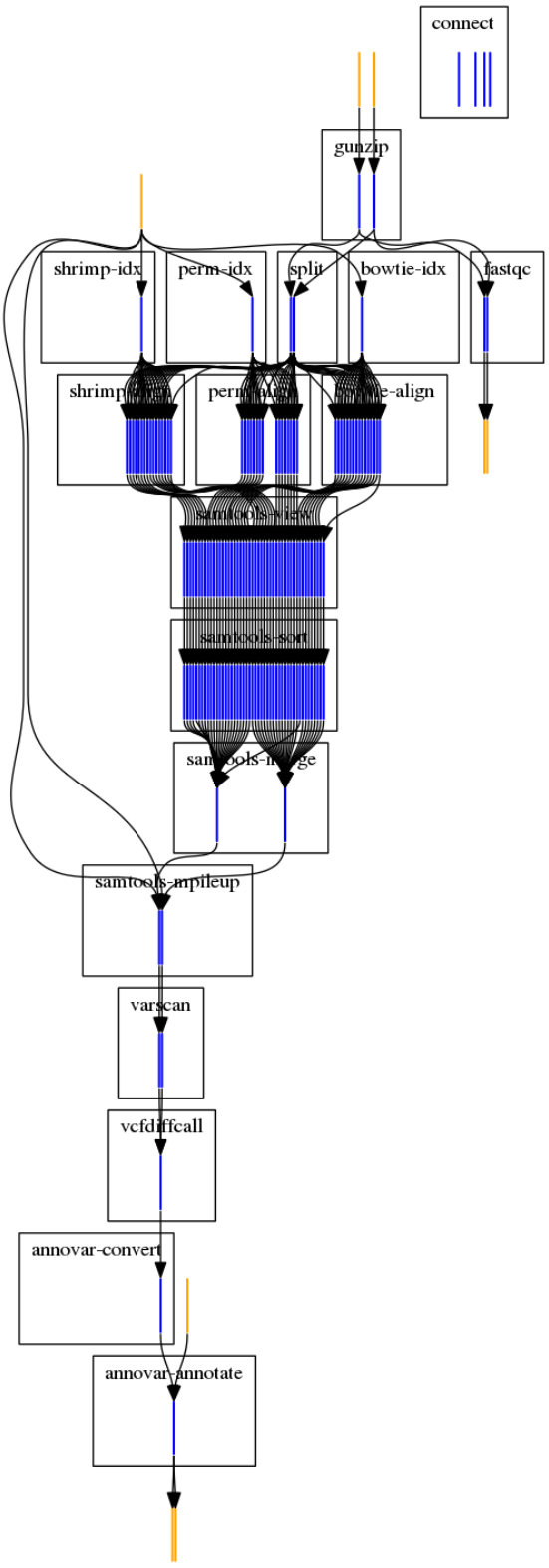

Figure 3: Invocation graph of the abstract workflow modeled and visualized by Cuneiform. Each blue line in is an executed invocation of a task on a worker node. Data dependencies are shown by black lines between tasks. Orange lines denotes input and output files.

	Statistics	StageIN	StageOUT
Timestamp	1383818054361	1383817921406	1383817924747
RUN ID	b9d6132b-0f9e-47aa-b4d1-f3e2395920c7	b9d6132b-0f9e-47aa -b4d1-f3e2395920c7	b9d6132b-0f9e-47aa -b4d1-f3e2395920c7
Workflow name	variant-call07	variant-call07	variant-call07
Task signature	31	31	31
Invocation Signature	433115866	433115866	433115866
Task name	fastqc	fastqc	fastqc
Message	invoc-stat:	invoc-stagein:	invoc-stageout:
	{"realTime":3.35,	{"reads_/geo.1.0002.fastq":5107533}	{"433115866_0_0":214234}
	userTime:4.02,		
	sysTime:0.13,		
	maxResidentSetSize:728240,		
	avgResidentSetSize:0,		
	avgDataSize:0,		
	avgStackSize:0,		
	avgTextSize:0,		
	nMajPageFault:0,		
	nMinPageFault:59160,		
	nSwapOutMainMem:0,		
	nForcedContextSwitch:26,		
	nWaitContextSwitch:504,		
	nIoRead:0,		
	nIoWrite:992,		
	nSocketRead:0,		
	nSocketWrite:0,		
	nSignal:0}		

Figure 4: Three log entries: Each entry belongs to an invocation (invocation signature), the invocation is an instance of a task (task signature or name), a task is part of a workflow (workflow name) which has a UUID that identifies the workflow run, that produced the tuple.