In-Memory Computing Enabling Real-time Genome Data Analysis

Matthieu-P. Schapranow* Franziska Häger* Cindy Fähnrich* Emanuel Ziegler† Hasso Plattner*

*Hasso Plattner Institute
Enterprise Platform and Integration Concepts
August–Bebel–Str. 88
14482 Potsdam, Germany
{schapranow|franziska.haeger|cindy.faehnrich|plattner}@hpi.uni-potsdam.de

†SAP AG
Dietmar-Hopp-Allee 16
69190 Walldorf, Germany
emanuel.ziegler@sap.com

***Abstract*—Latest medical diagnostics, such as genome sequencing, generate increasing amounts of "big medical data". Healthcare providers and medical experts are facing challenges outside of their original field of expertise, such as data processing, data analysis, or data interpretation. Specific software tools optimized for the use by the target audience as well as systematic processes for data processing and analysis in clinical and research environments are still missing. Our work focuses on the integration of data acquired from latest next-generation sequencing technology, its systematical processing, and instant analysis for researchers and clinicians in the course of precision medicine. We focus on the medical field of oncology to optimize the time spent on acquiring, combining, and analyzing relevant data to make well-informed treatment decisions based on latest international knowledge. We share our research results on building a distributed in-memory computing platform for genome data processing, which enables instantaneous analysis of genome data for the first time. For that, we present our technical foundation and building blocks of in-memory technology as well as business processes to integrate genome data analysis in the clinical routine.**

***Keywords*-Genome Data Analysis, Process Integration, In-Memory Database Technology, Precision Medicine, Next-Generation Sequencing, Alignment, Variant Calling.**

I. Introduction

We present our findings in providing specific software tools for clinicians and researchers in the course of precision medicine for integration of high-throughput genome data as source of diagnostic insights [1]. Precision medicine aims at treating patients specifically based on individual dispositions, e.g., genetic or environmental factors [2]. For that, researchers and physicians require a holistic view on all relevant patient specifics when making treatment decisions. Thus, the detailed acquisition of medical data is the foundation for personalized therapy decisions. The more fine-grained the available data is, the more specific the gained insights will be, but the complexity of data processing will rise as well. It requires tool support to identify the relevant portion of data out of the increasing amount of acquired diagnostic data [3].

Figure 1. Data processing steps involved in the analysis of genome data. Sequencing the samples results in chunks of DNA available in digital form. During alignment their position within the whole genome is mapped. Variant calling results in a list of differences of a fixed reference. The analysis obtains new insights based on the list of detected variants.

Figure 1 depicts the genome data workflow in the course of precision medicine. After a sample has been acquired, it is sequenced, which results in short chunks of Deoxyribonucleic Acid (DNA) in digital form. The DNA chunks need to be aligned to reconstruct the whole genome and variants compared to a reference, e.g., normal vs. pathologic tissue, are detected during variant calling. The analysis of genome data builds on the list of detected variants, e.g., to identify driver mutations for a medical finding [4].

Nowadays, Next-Generation Sequencing (NGS) devices are able to generate diagnostic data with an in-

creasing level of detail. In contrast to the first draft of the human genome, which involved thousands of institutes for more than one decade, modern NGS devices process a whole human genome within hours [5]. Nowadays, a sample of human tissue can be processed with more than 30x coverage in approx. 27 hours [6]. However, the increased level of detail results in additional data processing challenges. The following list summarizes selected data processing challenges, which are discussed in more detail in the remainder of our contribution.

- The sheer amount of generated DNA data is a challenge even for modern computer systems, e.g., per sequenced sample of a human tissue approx. 300-500 GB of raw data is generated digitalizing the human DNA with approx. 30x coverage.
- Raw DNA data needs to be processed prior to its analysis, which takes hours to days, i.e., alignment of DNA chunks to reconstruct a complete genome and identify variants compared to a known reference in the variant calling phase as depicted in Figure 1. Reducing the time for data processing would result in earlier start of data analysis.
- The availability of hundreds or thousands of individual cores in a modern computer cluster requires on the one hand the partitioning of the available data so that on the other hand specific algorithms can process this data in parallel.
- The analysis of genome data still involves big data, e.g., hundreds or millions of individual genetic variants. However, only a minority of these variants is connected to a certain disease, the majority of variants are not responsible for any malign changes. Thus, the analysis of genome data is an iterative and not a batch-oriented process. It consists of creating new hypotheses and their verification and requires software tools that support this kind of interactive analysis and exploration of genome data.

Figure 2 provides a comparison of costs for sequencing and main memory modules on a logarithmic scale. Both graphs follow a steadily declining trend, which facilitates the increasing use of NGS for whole genome sequencing and In-Memory Database (IMDB) technology for data analysis. Latest NGS devices enable the processing of whole genome data within hours at reduced costs [9]. The time consumed for sequencing is meanwhile a comparable small portion of the time consumed by the complete workflow. As a result, data processing and its analysis consume a significantly higher portion of the time and accelerating them would affect the overall workflow duration.

Our contribution focuses on how to optimize the time-consuming data processing and analysis aspects of the workflow by combining latest software and hardware

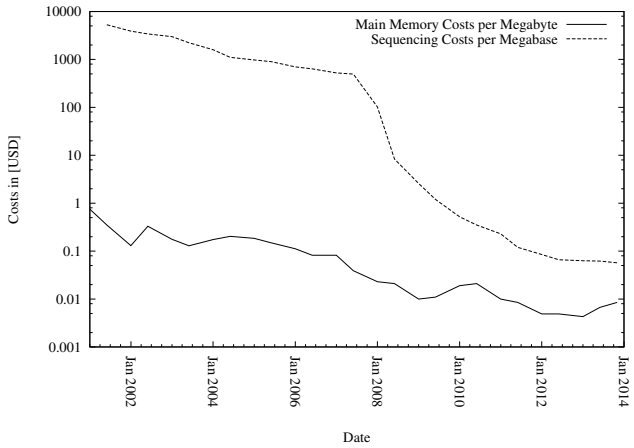


Figure 2. Costs for next-generation sequencing and main memory 2001-2014 adapted from [7], [8].

trends to create an integrated software system, which supports life science experts in their daily work.

The rest of this contribution is structured as follows: In Section II our work is set in the context of related work. We give a deeper understanding of in-memory computing in Section III and share concrete design decisions of our software architecture in Section IV. In Section V we outline our experiment setup and acquired results. An evaluation and discussion of our obtained results is given in Section VI while our work concludes in Section VII.

II. Related Work

The amount of related work in the field of genome data processing has increased in the last years. However, work focusing on the implementation of end-to-end processes and the improvement of scientific work is still rare. Thus, our work focuses on the integration of these aspects.

Pabinger et al. evaluated workflow systems and analysis pipeline tools [10]. They observed that existing tools either miss flexibility or the end-user needs specific know-how to install and operate them properly. We address this by introducing a combined system for modeling and execution of individual pipeline configurations without the need to adapt command line scripts as presented in Section IV-D.

Additionally, Pabinger et al. analyzed a variety of variant analysis tools and evaluated their functionality. For web-based tools they see a drawback in the required data preparation before the desired analysis can start, because "[...] files need to be packed, sorted and indexed before they can be used" [10]. We address time-consuming data transformations and preparations by replacing them by native database operations within our incorporated IMDB as outlined in Section IV.

Wandelt et al. observed in their evaluation a trend towards more and more cloud-based NGS data management solutions [11, Section 4.3]. We also believe that cloud-based software systems for processing and analysis of NGS data have advantages over local installations. For example, the setup, configuration, and operation of such systems requires trained personnel with specific bioinformatics background, which can be reduced by using cloud-based services. Cloud-based approaches also reduce costs for permanent local hardware resources, maintenance, and operation [12].

Wandelt et al. also identified the efficient mapping of workflow tasks in distributed computing environments and the adjustment of a given workflow to a dynamic environment as open issues. Our contribution addresses the modeling of workflows with a dedicated modeling notation as outlined in Section IV-D and their execution and resource allocation with a dedicated framework for scheduling as discussed in Section IV-G.

A first approach to distribute genome data analysis on a cluster of machines is Crossbow [13]. They use Hadoop for parallelization and built a pipeline that uses Bowtie for alignment and SOAPsnp for SNP detection [14], [15]. Their analysis pipeline took less than three hours on Amazon's Elastic Compute Cloud (EC2) with 320-cores distributed across 40 nodes for a 38x coverage genome. However, their approach was designed for a specific pipeline setup and requires extra work for adaptations, e.g., by adding additional variant calling algorithms. We enable users of our platform to adapt their pipelines individually using a graphical modeling notation as described in Section IV-D.

Our work contributes by providing a system architecture that combines processing and analyzing of genome data within a single system as outlined in Section IV. As part of our system architecture, we created a worker framework developed with the Python programming language, which enables integration of computing resources across platform and Operating System (OS) borders. Furthermore, our task scheduler controls the execution of a given workflow, e.g., prioritized processing of individual pipeline steps, as described in Section IV-G. It enables parallel data processing of multiple tasks as described in Section IV-E, e.g., to handle simultaneous user requests or tasks from multiple departments at the same point in time.

Galaxy, GenePattern, or Mobyle are selected related projects in the field of Reproducible Research Systems (RRS), which focus on enabling researchers to acquire, process, and document scientific data in a systematic, transparent, and reproducible way [16], [17], [18].

We address these fields with our platform as well as, e.g., among others by the following aspects:

- Graphical modeling and exchange of analysis workflows using a standardized modeling notation as described in Section IV-D,
- Enabling reproducible research results by sharing data and workflows with other users as described in Section IV-F, and
- Integration of latest international research databases using our annotation framework as described in Section IV-I.

III. Building Blocks of In-memory Computing

We refer to IMDB technology as a toolbox of IT artifacts to enable processing of enterprise data in real-time in the main memory of server systems [19]. Figure 3 depicts selected in-memory computing building blocks. The use of IMDB technology for genome data analysis is driven by the declining cost developments for NGS and main memory modules as described in Section II.

In the following, we outline selected building blocks of in-memory computing and their relevance for real-time analysis of genomic data in the context of our work.

A. Combined Column and Row Store

Historically, separate database systems for processing of analytical and transactional data evolved. The former store and process data in a row-oriented format, i.e., attributes of one record are stored side by side, while analytical database systems are optimized to scan selected attributes of huge data sets rapidly, e.g., by maintaining pre-aggregated totals.

If complete data of a single row needs to be accessed, storing data in a row format is advantageous. For example, the comparison of two customers involves all of their database attributes, such as inquirer's name, time, and content need to be loaded. In contrast, columnar stores benefit from their storage format when only a subset of attributes needs to be processed. For example, adding up the gender ratio of patients treated in a certain period of time only involves the attributes date and gender, but the remainder, such as name and birth date, are not required. Using a row store for this purpose would require processing of all attributes, although only two of these attributes are required. Therefore, a columnar store benefits from accessing only relevant data.

Combining column and row stores improves any kind of analytical queries while keeping transactional response times low. In our case, the use of columnar stores supports the comparison of multiple genomes to identify common mutations in the blink of an eye.

B. Complete History

Keeping the complete history of values even after individual data points have been updated or changed is the purpose of the insert-only or append-only technique. Insert-only is a data management approach that stores

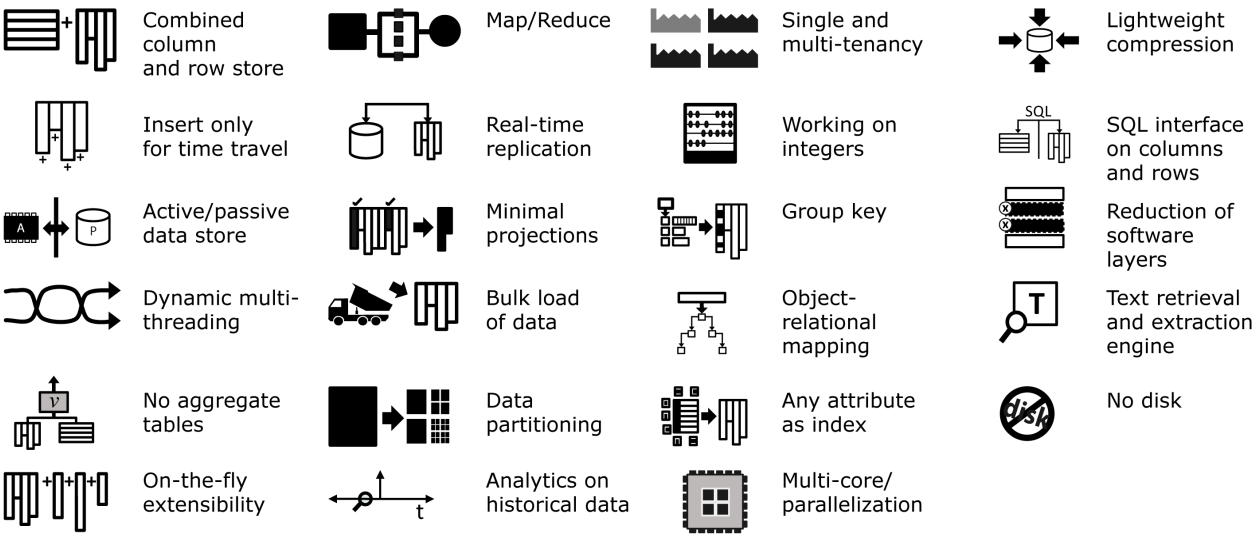


Figure 3. Selected in-memory computing building blocks.

data changes as new entries. Traditional database systems support four operations for data manipulation, i.e., insert, select, delete, and update of data. The latter two are considered as destructive operations since the original data is no longer available after their execution [20, Section 7.1]. In other words, it is neither possible to detect nor to reconstruct the values for a certain attribute after their execution since only the latest value is persistently stored in the database. Insert-only database tables enable storing the complete history of value changes and the latest value for a certain attribute [3]. This is the foundation of worldwide bookkeeping systems guaranteeing transparency.

Insert-only can also be used to trace decisions, e.g., in course of incident analysis. For example, consider a Clinical Information System (CIS) that is used to store latest decisions on medical dosages. If you directly replace the current value by a new value for the dosage, it is impossible to track when a patient received what dosage of a certain drug. Nowadays, updated dosages are stored as a new entry with a dedicated timestamp when they were applied. Using insert-only does not require this workaround and you can easily update the current dosage while the IMDB keeps a complete history of all changes. Thus, the IMDB is capable to reconstruct the global database state for any point in time using a specific database query.

C. Lightweight Compression

Lightweight compression techniques refer to a data storage representation that consumes less space than its original pendant [19]. A columnar database storage layout, as used in IMDBs, supports lightweight compression techniques, such as run-length encoding, dictionary encoding, and difference encoding [21]. Typically, values of a database attribute are within a limited subset of the attribute's full data domain, e.g., `male` and `female` for the gender type. The lightweight compression technique dictionary encoding, for example, maps all unique values to a uniform format, e.g., `male=1` and `female=2`.

The application developer can apply this technique during design time. However, IMDBs automatically perform lightweight compression optimized for the specific data to store. As a result, there is no longer an explicit need to map data from a human-readable format to an optimal storage representation since it is done transparently by the IMDB. Thus, the time to create new applications is reduced, the maintainability of the application code is improved since the source code is easier to understand, and any data stored in the database benefits from this kind of optimization without the need for explicit consideration in the application's code by the software developer.

For example, the International Code of Diseases (ICD) is identical for patients suffering from the same disease. Instead of storing the ICD redundantly in the database, dictionary compression stores it once and maps it to a smaller integer representation. Thus, only the corresponding integer value is stored in the database and all queries are rewritten to use the integer representation instead. The original representation is replaced just before the result set is returned to the client. As a result, the database executes all operations on compressed data without the need for explicit decompression, which improves cache-hit ratio since more compressed data fits into the same amount of cache memory [19].

D. Parallel Data Processing

Latest computer systems consist of multiple cores per individual Central Processing Unit (CPU), which is referred to as multi-core architecture [22, Chap. 2]. Additionally, a single server system can be equipped with multiple CPUs multiplying the amount of available computing cores, which is referred to as multi-CPU architecture [22, Chap. 2]. The hardware of a single computer system is designed to perform multiple processing tasks simultaneously. However, to use all available computing resources most efficiently software needs to incorporate specific instructions to explicitly make use of parallelization features, e.g., when adding up multiple values using the Parallel Addition (PADD) instruction [23, Chap. 5.3].

Parallelization can be applied to various locations within the application stack of software systems – from within the application running on an application server to query execution in the database system. For example, multiple clinical departments access the data of a single patient simultaneously. Processing multiple queries can be handled by multi-threaded applications, i.e., the application does not stall when dealing with more than one query at the same time. OS threads are a software abstraction that needs to be mapped to physically available hardware resources [24, Chap. 2].

A CPU core is comparable to a single worker on a construction area. If it is possible to map each query to a single core, the system's response time is optimal. Query processing also involves data parallelization, i.e., the database needs to be queried in parallel, too. If the database is able to distribute the workload across multiple cores, a single server works optimal. If the workload exceeds physical capacities of a single system, multiple servers or blades need to be installed for distribution of work to reach optimal processing behavior. From the database point of view, data partitioning supports parallelization since multiple CPU cores even on multiple servers can process data simultaneously [25, Chap. 6].

This example shows that multi-core architectures and parallelization depend on each other while data partitioning forms the basis for parallel data processing.

E. Data Partitioning

We distinguish between vertical and horizontal data partitioning [26].

Vertical partitioning refers to rearranging individual database columns. It is achieved by splitting columns of a database table in two or more sets of columns. Each of the sets can be distributed individually, e.g., on separate databases servers. This technique can also be used to maintain the same database column in different ordering to achieve better search performance for mixed workloads while guaranteeing high-availability of data [27]. Key to success of vertical partitioning is a thorough understanding of data access patterns. Attributes that are accessed in the same query should be located in the same partition since identifying and joining additional columns result in additional query processing overhead.

In contrast, horizontal partitioning addresses long database tables and how to divide them into smaller chunks of data. As a result, each portion of the database table contains a disjoint subset of the complete data. Splitting data into equivalent long horizontal partitions is used to support parallel search operations across all data of a database table and to improve scalability [19].

The identification of CpG Islands (CGIs) is a concrete application example: CGIs are known to represent unstable chemical compounds [28]. Their identification requires a full scan of the genome database table to identify positions where the bases cytosine and guanine are direct neighbors. Applying a horizontal partition per chromosome for the genome table enables scanning of all chromosomes in parallel. Furthermore, applying horizontal partitioning to each of the chromosome database tables enables processing of each individual chromosome by individual CPU resources, e.g., CPU cores.

F. Active and Passive Data

We distinguish two categories of data: active and passive. We refer to active data when it is frequently accessed and updates are expected to occur on regular basis, e.g., data of patients currently treated in a hospital. In contrast, passive data is neither updated nor accessed regularly. It is purely used for analytical and statistical purposes or in exceptional situations where specific investigations require this data. For example, tracking events of a certain pharmaceutical product that was sold five years ago can be considered as passive data. Firstly, from the business' perspective, the pharmaceutical can be consumed until the best-before date, which is reached two years after its manufacturing date. When the product is handled now, five years after it's manufacturing, it is not allowed to sell it any longer. Secondly, the product was most probably sold to a customer four years ago, i.e., it left the supply chain and is typically already used within its best-before data. Therefore, the probability that details about this pharmaceutical are queried is very low. Nonetheless, the tracking history is conserved and no data is deleted in conformance to legal regulations. As a result, the passive data can still be accessed but with a higher latency than active data. Thus, passive data results in a classification of data stores. For example, passive data can be used for reconstructing the path of a product within the supply chain or for a financial long-term forecast.

Dealing with passive data stores involves the definition of a memory hierarchy including fast, but expensive, and

slow, but cheap memory. A possible storage hierarchy is given by: memory registers, cache memory, main memory, flash storages, Solid State Disks (SSDs), Serial Attached SCSI (SAS) hard disk drives, Serial Advanced Technology Attachment (SATA) hard disk drives, and magnetic tapes [3].

Thus, active data that needs to be accessed in real-time can be separated from passive data that is ready for archiving. When data is moved to a passive data store, it frees fast accessible data stores, e.g., main memory.

To distinguish between active and passive data, rules for migration of data from one store to another need to be defined. We refer to them as data aging strategy or aging rules. We consider the process of aging, i.e., the migration of data from a fast to a slower medium as background task, which is performed regularly, e.g., once a month or once a week. Since this process involves reorganization of the entire database, it should be performed only during times of low database access, e.g., at night or on weekends.

G. Text Search and Text Mining

We distinguish the following categories of data:

- Structured data sources: We define structured data as data stored in a format, which can be used for automatic processing by computers. Examples for structured data are ERP data stored in relational database tables, tree structures, and arrays.
- Unstructured data sources: We define unstructured data as the opposite of structured data, which cannot be processed automatically, e.g., all data that is available as raw documents, such as videos or pictures. In addition, any kind of unformatted text, such as freely entered text in a text field, textual documents, or spreadsheets, are considered as unstructured data unless a machine-readable data model exists for automatic interpretation, e.g., a possible semantic ontology.

In the following, we outline selected features of text search that can be incorporated by IMDB technology.

Fuzzy search handles a specified level of fuzziness in search queries automatically, e.g., typing errors. It blows up the pool of words that are searched for by inverting pairs of letters or scrambling them. With these methods additional words can be found that are stored in a wrong format in the data set to search in. This is very helpful if humans added the data stored in the database, e.g., people, who search for terms, as well as people, who create textual content may add misspelled data. For example, a doctor's letter can contain various descriptions for the same result, e.g., "carcinoma", "karzinom", or "carzinoma". Fuzzy search helps to identify these entities as relevant for the same search query.

Synonym search addresses the challenge that different words can have identical meanings. These synonyms can be used in various contexts, but the search query typically only contains a single representation. For example, the medical abbreviation "ca." and "carcinoma" can be considered as synonyms. However, "ca." can also be the abbreviation for "circa". In other words, synonyms have individual meanings per context. To keep track of them, abbreviations should be considered by their probability in the active application context.

Entity and feature extraction refers to the identification of relevant keywords and names of entities from documents. This is comparable to tagging in online web blogs when certain additional meta information is associated to a document. Entity extraction can be customized by dictionaries and individual extraction rules. In this context, dictionaries are lists of entities with an assigned entity type that enable the database to recognize the listed entities in unstructured text. A dictionary contains one or more entity types, each of which contains any number of entities. Each entity in turn contains one standard form name and any number of synonyms. Extraction rules, define the entities of a specific type using a formal syntax. Such syntax allows formulating patterns that match tokens by using a literal string, a regular expression, a word stem, or a word's part-of-speech.

IV. High-Performance In-Memory Computing Platform

Figure 4 depicts the software system architecture of our high-performance in-memory computing platform with application, platform, and data layer as Fundamental Modeling Concepts (FMC) block diagram [29]. Our High-Performance In-Memory Computing Platform combines data from various data sources, such as patient-specific data, genome data, and annotation data within a single system. Thus, it enables flexible real-time analysis and combination of data in an interactive way for the first time. In the following, we share details about design decision and software components of our system.

A. Data Layer

The data layer holds all required data for performing processing and analyzing of genomic data. The data can be distinguished in the two categories master data and transactional data [30]. For example, human reference genomes and annotation data are referred to as master data, whereas patient-specific NGS data and Electronic Medical Records (EMR) are referred to as transactional data [31], [32]. Their analysis is the basis for gathering specific insights, e.g., individual genetic dispositions and to leverage personalized treatment decisions in course of precision medicine [2].

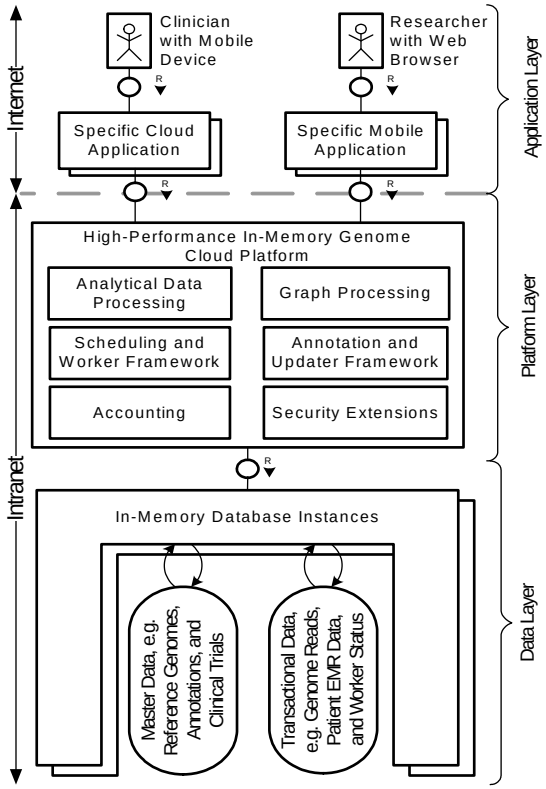


Figure 4. Our system architecture consist of application, platform, and data layer. Analysis and processing of data is performed in the platform layer eliminating time-consuming data transfer.

The actual step of analyzing the genetic data requires answering very specific questions. Thus, our application layer consists of specific applications to answer these questions. They make use of the platform layer to initialize and control data processing.

B. Application Layer

The application layer consists of special purpose applications to answer medical and research questions. You can access our cloud services online at http://www.analyzegenomes.com. We provide an Application Programming Interface (API) that can be consumed by various kinds of applications, such as web browser or mobile applications. Figure 4 depicts the data exchange via asynchronous Ajax calls and JavaScript Object Notation (JSON) [33], [34]. As a result, accessing data and performing analyses is no longer limited to a specific location, e.g., the desktop computer of a clinician. Instead, all applications can be accessed via devices connected to the Internet, e.g., laptop, mobile phone, or tablet computer. Thus, having access to relevant data at any time enhances the user's productivity. The end user can access our cloud applications via any Internet browser after registration. Selected cloud applications are our clinical trials search and our patient cohort

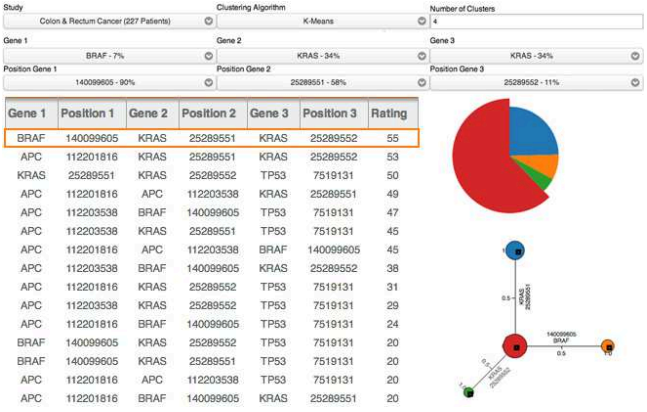
Figure 5. The patient-specific clinical trial search results based on the individual anamnesis of a patient. It extracts relevant entities from the free-text description of the clinical trial with the help of specific text-mining rules.

analysis, which are described in further details in the following [35].

1) Clinical Trials Application: Our clinical trials search assists physicians in finding adequate clinical trials for their patients. It analyses patient data, such as age, gender, preconditions, and detected genetic variants, and matches them with clinical trials descriptions from clinicaltrials.gov [36]. Furthermore, it incorporates details about the clinic a patient is treated in, e.g., to distinguish internal and external clinical trials to emphasize the link to colleagues from the same clinic. Our analysis incorporates more than 130,000 clinical trial descriptions, which are processed and ranked in real-time accordingly to the personal anamnesis of each individual patient. The ranked results are summarized on a single screen and provided to the researcher as depicted in Figure 5.

The clinical trials search incorporates the extraction of entities and features from the textual descriptions as described in Section III-G. We developed a set of specific dictionaries. For example, we use a dictionary for human gene identifiers with more than 120,000 gene names and synonyms and a dictionary for pharmaceutical ingredients with more than 7,000 entries. Our dictionaries incorporate a set of standardized vocabularies, e.g., the Metathesaurus Structured Product Labels (MTHSPL) of the Unified Medical Language System (UMLS) [37].

2) Patient Cohort Analysis: Figure 6 depicts our cohort analysis application. It enables researchers and clinicians to perform interactive clustering on the data stored in the IMDB, e.g., k-means and hierarchical clustering as shown in Figure 6 [38, Chap. 13]. Thus, they are able to verify hypotheses by combining patient and genome, and annotation data in real-time.



Gene 1	Position 1	Gene 2	Position 2	Gene 3	Position 3	Rating
BRAF	140099605	KRAS	25289551	KRAS	25289552	55
APC	112201816	KRAS	25289551	KRAS	25289552	53
KRAS	25289551	KRAS	25289552	TP53	7519131	50
APC	112201816	APC	112203538	KRAS	25289551	49
APC	112203538	BRAF	140099605	TP53	7519131	47
APC	112203538	KRAS	25289551	TP53	7519131	45
APC	112201816	APC	112203538	BRAF	140099605	45
APC	112203538	BRAF	140099605	KRAS	25289552	38
APC	112201816	KRAS	25289552	TP53	7519131	31
APC	112203538	KRAS	25289552	TP53	7519131	29
APC	112201816	BRAF	140099605	TP53	7519131	24
BRAF	140099605	KRAS	25289552	TP53	7519131	20
BRAF	140099605	KRAS	25289551	TP53	7519131	20
APC	112201816	APC	112203538	TP53	7519131	20
APC	112201816	BRAF	140099605	KRAS	25289551	20

Figure 6. Results of an interactive analysis of a cohort of 220 colon carcinoma patients using k-means clustering. It shows relevant combinations of genomic loci, such as gene KRAS on chromosome 12 at position 25,289,551, which are present in the majority of cohort members as depicted by the pie chart on the right.

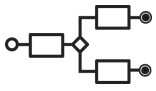
C. Platform Layer

The platform layer holds the complete process logic and consists of the IMDB system that enables real-time data analysis. We developed specific extensions that support processing of high-throughput genome data and enables real-time analyses. Thus, we established selected system components as follows:

- Graphical modeling of workflow and analysis pipelines to improve reproducibility,
- Parallel execution of pipeline model instances to enable high-throughput processing,
- Prioritized scheduling of jobs,
- Integration of existing tools in our system and development of highly optimized tools for IMDB technology, and
- Always up-to-date access to international knowledge databases.

In the following, we outline details about selected components and their relevance for our computing platform.

D. Modeling of Genome Data Processing Pipelines



Specific processing and analysis tasks need to be performed to identify genetic variants from raw DNA data acquired from sequencing devices. Nowadays, various software tools are used for each step of the processing and analysis workflow while researchers and clinicians use individual setups and parameters for their experiments. These setups are commonly implemented as a number of software scripts depending on each other. We refer to a concrete implementation of a processing and analysis workflow as Genome Data Processing Pipeline (GDPP).

In the following, we define a subset of a standardized modeling notation for the definition of GDPPs to improve maintainability, ease of modeling, and to establish a common understanding of the workflow. Another goal of our modeling approach is to enable external scientists and physicians to model their pipelines accordingly to their individual needs and have them executed on a central computer cluster.

A specific runtime environment for GDPPs enables the translation of models into executable code as described in Section IV-E.

1) Requirements: We refer to the atomic unit of a GDPP as job. A job refers to a concrete script that can be executed to perform a specific task while activity refers to the abstract representation of a job in the process model. Thus, the most fundamental precondition for modeling of GDPPs is a representation of a number of jobs and their execution sequence. In order to allow reuse of a group of logically associated jobs in several pipelines, e.g., a specific combination of alignment algorithm and post-processing steps, the modeling system should support hierarchically nested inclusion of pipelines to form a new model.

Parallel data processing improves the execution time for the overall pipeline as described in Section IV-E. Therefore, the modeling approach should also support the explicit definition of activities that should be executed in parallel.

Some activities have a varying internal behavior or outcome depending on their defined input parameters. For example, an alignment job might support a dynamic parameter for the reference genome that is used for the alignment of chunks of DNA. Thus, modeling should support the definition of input parameters and the link to activities.

Additionally, the models should be stored in a standardized, machine-readable format, e.g., to ensure exchange and interpretation of models when sharing them across institutions.

We defined our GDPP modeling approach as a subset of Business Process Model and Notation (BPMN), which is a standardized and widely adopted modeling technique. In the following, we define the required subset and mapping to our GDPP modeling notation.

2) Business Process Model and Notation: The Business Process Management Initiative (BPMI) introduced BPMN standard in 2004. Since 2006, it is an official standard of the Object Management Group (OMG), which released BPMN version 2.0 in 2011. The actual work within a BPMN process is modeled by activity elements. They represent the atomic unit of a model that can be executed by either a human or a computer system. The logic of a BPMN flow is defined by so-called gateways, such as exclusive gateways representing a logical XOR and parallel gateways representing a logical AND. Each BPMN process is defined by a unique

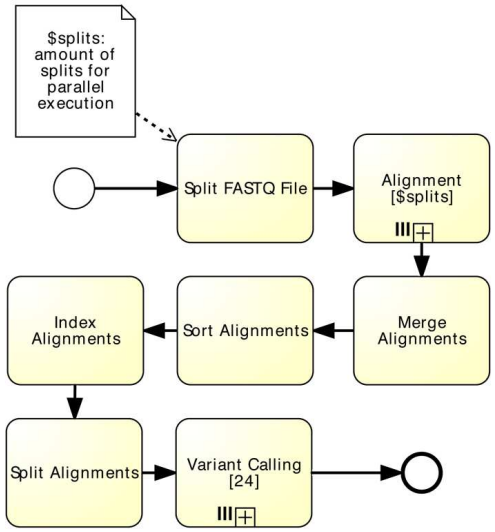


Figure 7. GDPP model of the general approach with file system as primary storage. The input FASTQ file is first split up for parallel processing during alignment. The outcome is merged again and prepared additional processing steps before it is split up per chromosome again for variant calling.

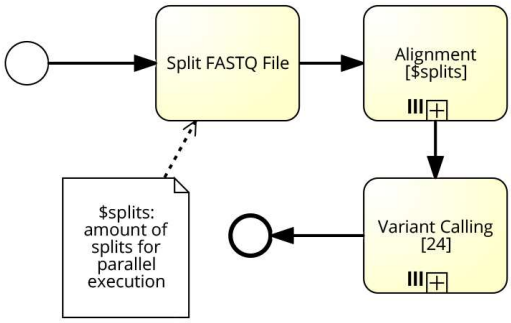


Figure 8. GDPP model incorporating an IMDB as primary storage. In contrast to using a file system as primary storage, the intermediate processing steps are not required anymore.

start event and at least one end event [39].

A widespread and well-defined XML-based representation of BPMN models is the XML Process Definition Language (XPDL). We incorporate the existing XPDL standard to store and exchange our GDPP models.

3) Hierarchy of Activities: GDPPs can be hierarchically nested to any level of depth. Any set of logically associated activities can be represented as a separate model containing a sub process model. Sub process activities are used as placeholders in the invoking process. An example is shown in Figure 7, which contains a sub process named `Alignment[$splits]` that is depicted in Figure 10. The names of the sub process and the corresponding process model are automatically replaced during runtime based on their name. Figure 10 depicts the concrete sub process for `Alignment[$splits]` in-

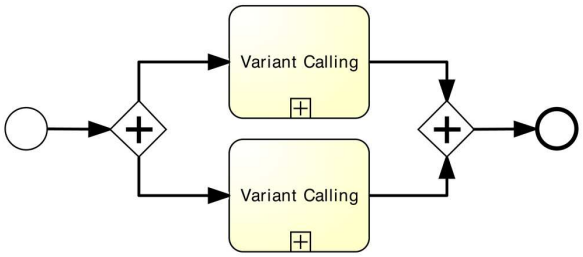


Figure 9. GDPP model using parallel gateways to perform variant calling two-times parallel.

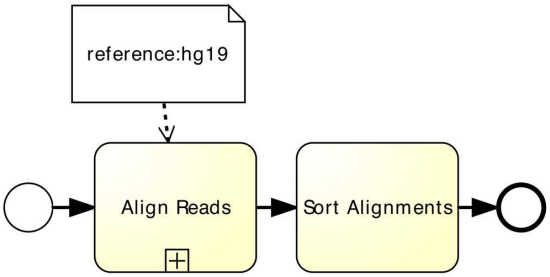


Figure 10. Modeling of parameters to acquire from end-users as input for an activity prior to its execution. Here, the reference to use during alignment is set to hg19.

cluded in the model depicted in Figure 7. The alignment step consists of multiple jobs to be performed, e.g., transformation steps or sorting the alignment results. In addition, each alignment algorithm modeled in our GDPP has its distinct sub process, which encapsulates further necessary transformation steps to receive output in the standard format.

4) Parallel Processing of Activities: Parallel execution of activities in BPMN can be defined as follows.

- Parallel multiple instances are modeled as an activity with three vertical lines at the bottom as depicted in Figure 7 for sub processes alignment and variant calling. The parallel multiple instance is executed as often as defined by the number defined in square brackets following the activity's name, e.g., variant calling is executed 24 times in parallel.
- Parallel gateways are an alternative way of modeling parallel workflows. They are used when no specification of quantity of parallelization exists. When a parallel gateway is signaled, all outgoing edges of the gateway are signaled as well. When the gateway consists of multiple incoming edges, the gateway only signals once all incoming edges were activated. Thus, the sequence flow can be split in two or more parallel strands and resynchronized if needed. Figure 9 illustrates an example for parallel gateways. This time, variant calling is executed by two activities in parallel.

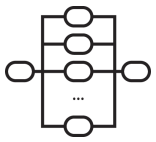
5) Parameters and Variables: We distinguish between parameters and variables as follows. Parameters are set during design time of the GDPP model and cannot be changed afterwards. Variables are placeholders that are assigned at the latest point in time just prior to the execution of a GDPP model instance.

BPMN defines data objects for modeling of specific input parameters of activities [40]. A parameter is stored in a data object labeled as the parameter's name followed by its value separated by a colon. The parameter's name matches the input variable name of the corresponding activity. A data object can be associated to one or multiple activities.

We added support for variables in our GDPP models by using a specific data object identified by a dollar sign (`$`) followed by the variable's name. Figure 7 depicts the use of variables in a GDPP model, i.e., the variable `splits` has to be set to a concrete value prior to the execution of the concrete GDPP model instance.

Parameters and variables can be assigned to multiple activities. For example, Figure 8 depicts multiple usage of parameter `splits`. On the one hand, it is required by the first activity to know how many splits to create. On the other hand, the parameter defines the amount of alignment sub processes that will be executed in parallel.

E. Parallel Execution of Genome Data Processing Pipelines



In context of precision medicine the aspect of high-throughput processing and analysis becomes essential to leverage a clinical solution. Thus, we focus on parallel execution of GDPPs and for that, we designed specific functionality within our platform.

A distributed set of computing nodes each running multiple workers forms our worker framework. Each worker is directly connected the IMDB database landscape to access their local portion of the database content. Relevant details about tasks that need to be executed are added to the tasks database table by the scheduler. Once a worker starts processing of a concrete task, it updates the current status of the task within the database. Incorporating the database for these purposes reduces the complexity of the individual worker code since specific exception handling can be processed by the database, e.g., using built-in database locks can prevent concurrent start of the identical task.

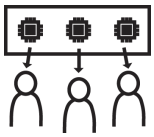
All workers and the scheduler use a specific communication protocol to exchange short messages between each other's, e.g., to reduce idle times. On the one hand, workers can exchange relevant status information about the load of a certain node and updated jobs. On the other hand, the scheduler sends a wakeup signal to all workers to inform about jobs ready to be executed. For further details about the scheduler component, please refer to Section IV-G.

F. Fair Use of Resources and Accounting

Processing and analyzing data consumes resources of our platform, such as computing time or hard disk storage. Thus, we have integrated a fine-grained accounting functionality to ensure fair usage of provided services and resources. The atomic measurement unit for any kind of service on our platform is called gene point. On the one hand, users can spend gene points on platform services or to access data provided by other users. On the other hand, users can acquire gene points by providing services or data to other users. This mechanism guarantees fair resource allocation for all users and encourages active data exchange. Furthermore, it builds the foundation for sharing intellectual property and enables compensation [3, Chap. 5].

The prioritized scheduling of jobs is the key concepts to implement fair use and accounting within our platform as discussed in the following.

G. Prioritized Task Scheduling



We created a single scheduler component coordinating the execution of multiple GDPPs. Thus, it enables resource allocation and distribution of workload across our cluster of worker machines. The scheduler stores its internal state permanently within the IMDB, e.g., for global communication, logging, and for maintaining statistics. We implemented specific scheduling algorithms optimized for throughput that analyze the complete execution history of all former runs in order to process shortest GDPP instances first.

The scheduler node is responsible for managing all aspects from reading the GDPP models to scheduling all relevant activities and linked jobs.

Every scheduling decision is persisted in the database prior to its executed, i.e., the database provides a consistent transaction log, which enables controlled recovery in case of a system failure.

We implemented specific scheduling policies to optimize scheduling decisions depending on various aspects. For example, we incorporate the Shortest Task First (STF) scheduling policy to minimize turnaround time and maximize throughput [24, Section 2.4.2].

Our STF scheduling policy is adapted to estimate the remaining execution time of all waiting tasks whenever a scheduling decision needs to be taken. The incorporated IMDB technology guarantees that the estimation can be processed in real-time and does not delay decision making significantly [3, Chap. 3].

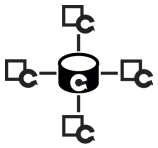
The developed scheduler component is very generic and can easily be adapted to fit individual requirements, e.g., to prioritize the execution of tasks from a department or to keep a processing reserve for the very important users. Furthermore, individual scheduling policies can be developed to change the behavior of the scheduling system. Each scheduling policy can incorporate various input data, e.g., details about the overall system load provided by the load balancer as described in Section IV-H.

H. Load Balancing

The overall system load of all computing nodes incorporated by our computing platform depends on running jobs and their assignment to individual nodes. This becomes especially important if we assume a computing pool that consists of a heterogeneous hardware. Thus, we implemented a load balancer that incorporates the current system status of available worker nodes. The configuration of all attached worker nodes, e.g., how many workers are running on each of them or how many CPU cores are available, is stored in the configuration database table.

The detailed view of the load balancer can be incorporated by the scheduler component during its decision-making process as described in Section IV-G. For example, the scheduler still can postpone the execution of long-running jobs when short-running jobs are available.

I. Annotation Framework

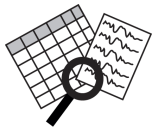


We consider the use of latest international research results as enabler for evidence-based therapy decisions [41]. Our annotation framework is the basis for combining international research results. It periodically checks all registered Internet sources, such as public FTP servers or web sites, for updated and newly added versions of annotations, e.g., database exports as dumps or characteristic file formats, such as Comma-Separated Values (CSV), Tab-Separated Values (TSV), and Variant Call Format (VCF) [42]. If the online version is newer than the locally available version, the new data is automatically downloaded and imported in the IMDB to extend the knowledge base.

The import of new versions of research databases is performed as a background job without affecting the system's operation. We import new data without any data transformations in advance. Thus, data becomes instantaneously available for real-time analysis [43], [44].

For example, the following selected research databases are regularly checked by our annotation framework: National Center for Biotechnology Information (NCBI), Sanger's Catalogue Of Somatic Mutations In Cancer (COSMIC), University of California, Santa Cruz (UCSC) [45], [46], [47].

J. Combined Search in Unstructured and Structured Data Sources



A significant amount of today's medical data is encoded in the form of unstructured natural language [48]. Scientific publications and patents, medical reports, as well as comments, keywords, or descriptions in database records use natural language to store information [48]. We consider this unstructured data as a substantial part of the world's medical knowledge. However, comprehension, analysis and searching of unstructured data are still challenging compared to structured data, such as experiment results or genomic variant data. So far, research to extract information either from structured or unstructured medical data does not investigate the advantages that can be gained by combining results from both sources.

For example, a physician could receive information contained in scientific publications that perfectly match her or his patient's current diagnosis. Additionally, researchers face the challenge to identify relevant information sources within a tremendously short timespan. With the help of our IMDB technology, we enable researchers to identify relevant data from structured and unstructured data sources. Our specific database extensions recognize relevant entities within text documents and extract them automatically.

This builds the foundation for applications that incorporate data from both worlds: structured and unstructured data. An example application that builds on the combined search in structured and unstructured data is our clinical trials search as described in Section IV-B1, which incorporates unstructured textual information from clinical trial descriptions and doctor letters as well as structured information obtained from the genetic biomarkers of a specific patient.

K. Development of Tools for In-Memory Computing vs. Integration of Existing Tools

Existing tools for genome data analysis can be directly integrated in our GDPPs. For that, we implemented a new job that invokes the corresponding tool via command line and adapted the pipeline model as described

in Section IV-D. This strategy facilitates an easy integration of new tools into our framework without caring for distribution and scheduling. However, this improves integrated algorithms only to a limited extent, e.g., since these tools still access data from files located on disk storage. Loading large files from disk into main memory for processing still consumes additional processing time prior to data analysis. In addition, the majority of tools do not exploit computational resources fully to optimize runtime performance. Analysis tools specifically designed for our in-memory computing platform benefit in terms of parallelization, compression, in-memory storage, and distribution across multiple machines.

1) Benefits of Optimized Tools for In-Memory Computing: Developing specific tools for genome data analysis as built-in functionality of our in-memory computing platform results in reduced setup and configuration in addition to improved data processing. Furthermore, the results of the data processing are directly available in our IMDB. Thus, you can apply any analysis tools directly to the results without the need for any data preparation. The different building blocks presented in Section III come along with further advantages that accelerate fast data processing as follows.

Lightweight Compression: Sequencing data consumes huge amounts of storage capacities, i.e., up to hundreds of GB per single human genome. Therefore, it is crucial to apply data compression techniques, e.g., as currently done in genome data analysis by converting the raw data from Sequence/Alignment Map (SAM) format into its Binary Alignment Map (BAM) format [49]. Our incorporated IMDB technology applies lightweight compression to genomic data in a transparent way. Thus, data remains in a human readable format, e.g., during database queries, while the storage footprint is automatically reduced.

Column Orientation: Storing data as complete tuples in adjacent blocks, i.e., row-wise, is advantageous if the complete data of a single row has to be accessed, for instance by comparing two complete table entries to each other. However, most often algorithms only require access to particular attributes of a data record for analysis. For example, when filtering read alignments in the first phase of variant calling, only data quality indicators need to be accessed such as mapping or base quality scores. If all records were stored row-wise as it is the case for files, all attributes need to be processed although only two of them might be relevant for computation. Storing data in columnar format, i.e., storing complete columns in adjacent blocks facilitates direct access only to relevant attributes and avoids cache misses [50], [51].

Multi-Core and Parallelization: When creating optimized analysis tools for our in-memory computing platform, we can profit from already existing functionality to apply parallelization and full exploitation of CPU resources. Thus, we can focus on optimizing algorithms instead of implementing resource management.

Data Partitioning: Regarding the optimization of analysis tools, in- and output data can easily be horizontally partitioned according to chromosomes and even chromosome regions to distribute data and it's processing to computing nodes. When working with current tools, distribution and selection of relevant data is a time-consuming task that is carried out by specific tools, e.g., SAMtools [49]. With data partitioning, the search space for accessing data of a particular chromosome or region reduces to only a small part of the original data set. This improves search operations and facilitates better scalability of our optimized algorithms.

2) Alignment on In-Memory Computing Platform: Our genome data alignment algorithm optimized for in-memory technology was designed with the following requirements in mind:

- Use available main memory for faster index structures that allow accelerating the lookup process,
- Minimize the cache miss ratio since cache misses are known to be a major cause for bad performance on modern hardware architectures, and
- Optimize parallel code execution, e.g., by minimize the need for process synchronization and avoid writing to shared data structures requiring locking.

We used a k-mer based index structure to find alignment position candidates and filled the unmatched gaps using heuristics optimized for low error rates and a variant of the Needleman-Wunsch and Smith-Waterman algorithms [52], [53].

Structure of the Index: The commonly used index structures based on the proposed techniques by Ferragina and Manzini (FM-index) have a very low memory impact, but require at least two cache misses on average per nucleotide even if only perfect matches are required [54]. k-mer-based indexes are less error tolerant since they usually require the whole k-mer to match the reference genome. They are much faster than FM-indexes though since complete k-mers can be identified by a single access to the index structure. The price to be paid is that the algorithm has to find sufficiently many perfectly matching k-mers to get a strong signal. FM-index-based algorithms are more tolerant in theory as deviations from the reference can be included into the search. But due to the large search space that has to be covered in order to achieve this tolerance, their efficiency on long reads is limited.

For example, 100 base pairs are nowadays standard and already cause major problems to this approach. Thus, additional assumptions need to be made, which are comparably restrictive as k-mer-based indexes.

While longer reads cause a stronger problem to FM-index search they help with finding sufficiently many k-mers without difference to the reference genome. Recent advances in sequencing quality improve this further and make k-mer indexes more attractive, which is the reason we prefer this approach.

Finding Optimal Positions: By comparing multiple k-mers from the read, the algorithm usually obtains many index hits per read of which some are true positives and others false positives. k-mers with too many hits are not very decisive and blow up the search space beyond their meaningful use. Therefore, k-mers with more than 256 hits are ignored. The search space can also be narrowed down by choosing longer k-mers, which is done whenever too few k-mers with few hits were found.

The possible matching positions from all k-mers are being matched with each other depending on the k-mer position within the read and the distances of their matches on the reference genome. The distance furthermore gives a lower limit for the number of insertions resp. deletions required for the match. The missing k-mers indicate that mismatches have appeared even if distances match. From this information, an optimistic score can be computed and the alignments are further processed starting with the best score. When each alignment is finalized, the actual score is known and can be compared to the optimistic score of the next alignment candidate. If the optimistic score is worse than the best actual score found, the alignment is done as no better alignment can be found anymore.

Filling the Gaps: To get a perfect alignment, all differences between the reference genome and the read need to be found similar to computing the edit distance, but with probabilistic scores for each deviation. They are traditionally computed by dynamic programming algorithms, which scale quadratic in read length. This problem is dramatically reduced by focusing on filling only the gaps in between the k-mer hits, .

Extending the matching parts into the gap until a difference compared to the reference genome is detected makes further improvements. If the remaining part is only a single nucleotide long both on the reference genome and on the read, it must be a substitution. If the gap on either the reference genome or the read is completely closed, it must be an insertion or deletion, respectively. Therefore, many of the differences can be resolved without the need for more complex algorithms due to lower error rates.

For the final step, we use a dynamic programming algorithm optimized for Single Instruction, Multiple Data (SIMD) based on the Needleman-Wunsch and Smith-Waterman algorithms adjusted to our boundary conditions [52], [53]. This allows alignments restricted on both ends by known k-mers and half open gaps for which a k-mer has only been found on one end.

Improvements for Mate-paired Reads: If a correlation in the alignment position on the reference genome is known beforehand (mate-pairing) the alignment finalization can be skipped for all positions that do not fulfill this correlation. Also the optimistic scores can already be computed for both reads simultaneously achieving a better estimate of the best alignment. This strategy excludes many possible matches at a very early stage. Thus, it reduces the amount of expensive full alignments and improves accuracy.

3) Variant Calling on In-Memory Computing Platform: We designed a variant calling approach to identify Single Nucleotide Polymorphisms (SNPs) that is directly executed within our in-memory computing platform. With respect to that, our approach has to meet the following requirements:

- Use available main memory capacity to store and process read alignments while eliminating access to slower file systems,
- Direct access to specific read alignment attributes without the need to traverse the complete data record, and
- Apply compression strategies to reduce memory footprint and to improve processing throughput.

Our SNP calling is divided in data preparation and genotype calling. We achieve parallelization by executing the algorithm in a MapReduce-like fashion, where the processing steps correspond to map phases. For each of them, data is split up into smaller subsets and processed in parallel [55]. After that, the result sets are merged in a reduce phase to be then again split up for the next map phase.

Data Preparation: During data preparation data is assembled and prepared for subsequent genotype calling. It comprises the data extraction and grouping phase.

The goal of the data extraction phase is to reduce the overall amount of data to process by filtering out irrelevant and low-quality data. During data extraction, we identify the sequences of a read alignment that are relevant for SNP calling. For that, we first eliminate reads that are not sufficient for processing, e.g., because necessary information is missing or data quality does not meet user-defined thresholds. Afterwards, we identify the relevant read alignment sequences. We receive the information about what parts of a read alignment are involved in single substitutions, insertions, or deletions from each read alignment's individual CIGAR attribute [56]. As we concentrate on the identification of SNPs in our approach, sequences of a read that are involved in insertions will be filtered from our data. The aim of the data-grouping phase is to rearrange the read alignment data for efficient genotype calling afterwards. The output received from the data extraction phase is

used to group all bases from relevant sequences of a read alignment according to the positions in the genome they have been aligned to. This way, each position in the genome has assigned four "piles" for the distinct bases Adenine (A), Cytosine (C), Thymine (T), and Guanine (G) that comprise information about base occurrences. At this time, we also extract relevant information about the data quality, e.g., base quality scores, as this information builds the computational basis for genotype calling in the subsequent processing step.

Genotype Calling: The goal of the genotype-calling step is to derive a concrete genotype for each particular position in the genome that is covered by the base pileups. For the actual computation, we apply a statistical model that is sensitive to input data quality. This includes that we have to compute the probability for each possible genotype, i.e., ten different genotypes for the diploid human organism, at a particular position in the genome. The genotype with the highest probability will be called in the end. To calculate the probability of a genotype, we apply a Bayesian framework as proposed by Nielsen et al. with two components called prior probability and genotype likelihood [57].

The prior probability of a genotype is its general chance to occur regardless of the given data. In our computations, we do not assign a unique prior probability to all genotypes. Instead, we distinguish genotypes according to their zygosity, i.e., homo- or heterozygous, and reference equality and make use of the assumptions stated by Li et al. [15].

The genotype likelihood of a genotype is its chance to occur with regard of the given data. We compute this value from all occurrences of a genotype at a particular position and incorporate the bases' quality scores. A base quality score indicates how likely the sequencing machine has detected a base correctly. We incorporate this value in our computations because the read alignments produced by those sequencing machines are error-prone up to one percent of the data [58], [59]. Thus, we give stronger weights to bases with a higher probability to be correct and downgrade low-quality bases.

After genotype calling, each derived genotype owns a quality value, which indicates the likelihood of the called genotype. All genotypes that differ from the reference genome and have a quality value that matches the user-defined threshold make up the set of emitted SNP calls.

V. Benchmarks

The aim of all conducted benchmarks was to minimize the overall execution time for a single GDPP run, i.e., to use the maximum of available computing resources and achieve highest throughput. Furthermore, we aim to compare selected alignment algorithms regarding their efficiency for varying file sizes. In the following, we share insights about our benchmarks conducted on our in-memory-based processing and analysis platform.

TABLE I. EXPERIMENT CONFIGURATIONS.

Experiment	Split Size	Primary Storage
A	1	File System
B	1	In-Memory Database
C	25	File System
D	25	In-Memory Database

TABLE II. DATA SET SPECIFICATIONS.

Data Set	Size [Gbp]	Size [GB]	Reads [Billion]
1	0.5	1.2	5.4
2	1.0	2.4	10.8
3	1.9	4.8	21.7
4	3.9	9.6	43.4
5	7.9	19.2	86.8
6	15.8	38.5	173.7
7	31.6	78.0	345.8

A. Setup

All benchmarks were performed on a computer cluster consisting of 25 identical computing nodes with a total of 1,000 cores provided the Future Service-Oriented Computing Laboratory at the Hasso Plattner Institute [60]. We incorporated this hardware setup to demonstrate the scalability of our in-memory computing platform. It should not be interpreted as minimum hardware resources required operating our contribution. Each of the nodes was equipped with four Intel Xeon E7-4870 CPUs running at 2.40 GHz clock speed, 30 MB Intel Smart cache, interconnected by 6.4 GT/s Quick Path Interconnect (QPI), and 1 TB of main memory capacity [61]. Each CPU consisted of 10 physical cores and 20 threads running a 64-bit instruction set. All computing nodes were equipped with Intel 520 series SSDs of 480 GB capacity combined using a hardware raid for local file operations [62]. The average throughput rate of the local SSDs was measured with 7.6 GB/s cached reads and 1.4 GB/s buffered disk reads. All nodes were interconnected via a Network File System (NFS) using dedicated 10 Gb/s Ethernet links and switches to share data between nodes.

Instead of using generated test data, we incorporated real NGS data for individual measurements, i.e., FASTQ files from the 1,000 genomes project [5]. We used the FASTQ file of patient HG00251 for our benchmarks, which consumes 160 GB of disk space, consists of approx. 63 Gbp, approx. 695 M reads with 91 bp individual read length, forming an average 20x coverage.

We implemented two GDPPs for our benchmarks. The first corresponds to the commonly followed approach of using a file system as primary storage. The distinct execution steps are modeled as GDPP notation in Figure 7. On the contrary, the second pipeline as

shown in Figure 8 uses an IMDB as primary storage, i.e., intermediate results are stored in our in-memory computing platform.

Our pipelines contain distinct parts for alignment and variant calling that are parallelized, e.g., by splitting up the input data. The alignment step is comprised of the alignment algorithm itself and file transformation and processing steps due to differing output formats per alignment algorithm. Furthermore, the GDPP using a file system as primary storage contains additional processing steps between alignment and variant calling, which are necessary to split up data per chromosome. These steps are not required for the IMDB-optimized GDPP since alignment results are already imported into the database and chromosome-wise splits are implemented using native database operations.

1) Burrows Wheeler Aligner: We used Burrows Wheeler Aligner (BWA) version 0.6.2 as alignment algorithm reference [63]. BWA was configured to use a maximum of 80 threads, which relates to the maximum available hardware resources of our benchmark infrastructure. The algorithm's output is a SAI file that needs to be converted to the SAM format. Therefore, we added format transformation directly after alignment to receive alignments in SAM format. For the GDPP models of Exp. A and C, we additionally have to carry out a transformation from SAM into BAM format and to sort the resulting BAM file as preparation for merging.

2) HANA Alignment Server: The second part of our benchmarks was performed on a GDPP integrating our own alignment algorithm as described in Section IV-K2. It is implemented directly within our in-memory computing platform, i.e., it can directly access native database operations. This algorithm was configured to use a maximum of 80 threads and emits alignments either in SAM or BAM format. As a result, additional format transformations, e.g., from SAM to BAM, are no longer required for both of our pipelines.

B. Experiments

We designed our benchmarks to compare the impact of the incorporated storage system and the level of parallelization on the overall execution time. Each of the experiment categories was conducted for the alignment algorithms BWA and HANA alignment server to evaluate the impact of the overall execution time for a specific GDPP. Exp. A and B were executed on a single computing node, while Exp. C and D were executed on 25 computing nodes to evaluate the impact of a fully parallelized execution environment as outlined in Table I. In addition, we derived subsets of the input data as shown in Table II.

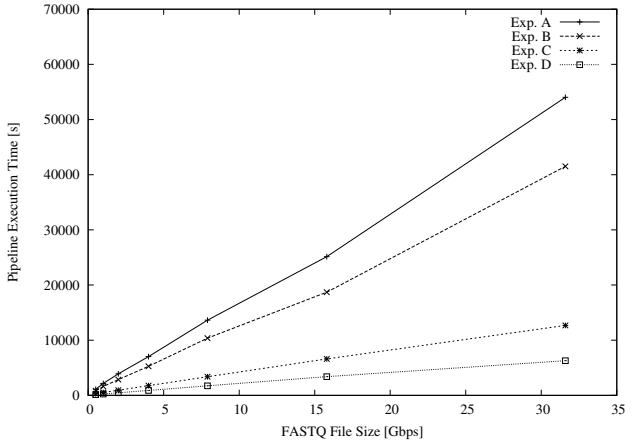


Figure 11. BWA: Development of overall execution times for varying file sizes and experiment setups.

C. Results

In the following, we present our obtained benchmark results. For each alignment algorithm, we measured the overall pipeline execution times t_x for Exp. x and derived the relative advance of execution time for Exp. x compared to Exp. A as $R_x = \frac{t_A - t_x}{t_A}$.

Table III shows the overall pipeline execution times incorporating BWA as alignment algorithm. The measured execution times indicate that the use of the IMDB as primary storage for intermediate results is beneficial for all selected file sizes. This pipeline optimization results in a reduction of the overall runtime by at least 25 percent on average.

Exp. C and D as shown in Table III document the impact of the parameter `splits`, i.e., the number of distributed computing nodes used for parallel execution as introduced in our pipeline models. Parallel execution of selected pipeline steps reduces the overall execution time by at least 74 percent on average for BWA. Additional improvement can be achieved by using the IMDB as primary storage. Figure 11 illustrates execution time behavior for GDPP with BWA as alignment algorithm. It clearly shows the improvements originating from parallelization and main memory as primary storage medium. For a GDPP using BWA as alignment algorithm, execution time can be reduced by at least 87 percent on average.

Table III shows the overall pipeline execution times for BWA and HANA alignment server. Execution times develop similarly to the results obtained for BWA pipelines. Table III depicts that runtime improves up to 50 percent when using an in-memory database as primary storage, and up to 75 percent when distributing the pipeline across 25 computing nodes.

Comparing the overall execution times amongst the different alignment algorithms used, we can see that

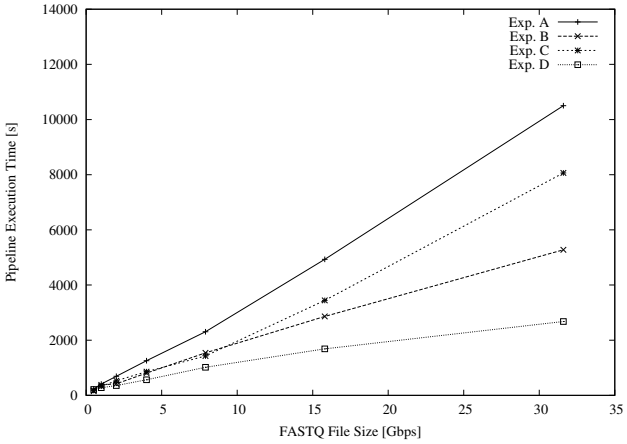


Figure 12. HANA alignment server: Development of overall execution times for varying file sizes and experiment setups.

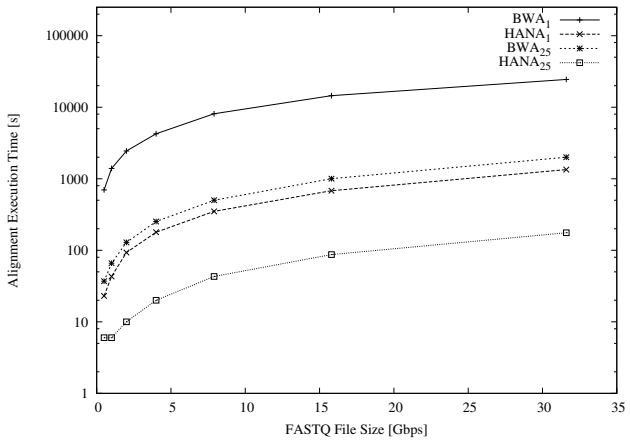


Figure 13. HANA and BWA alignment: Development of execution times for varying file sizes and split sizes.

runtime performance drops significantly when using HANA alignment server as alignment algorithm. From the alignment algorithms used, the pipelines applying HANA alignment server show best runtime performances throughout all runs, up to 85 percent faster than with BWA.

Figure 13 shows execution times for alignment with BWA and HANA for different file and split sizes. t_{Aln_1} and $t_{Aln_{25}}$ describes average processing times consumed by the alignment algorithm only with split sizes 1 and 25, respectively. Performing read alignment with BWA takes absolutely longer than HANA alignment server. We derived the speedup factor $S_{x:y} = \frac{t_x}{t_y}$ for BWA and HANA alignment server when applying parallelization. Performing BWA alignment in parallel on 25 nodes results in a speedup factor of up to 21x, which means the process has a great parallel portion that benefits from additional computing resources. HANA alignment server with 25 nodes results in a speedup factor of up to 9x. This can be explained by the very short execution time HANA alignment server consumes, i.e., there is a significant higher sequential portion of code that reduces the speedup.

For split size 1, HANA alignment server brings a relative runtime improvement of 97 percent on average as listed for $t_{1_{Aln}}$ in Table III. Relating these numbers to our overall execution times in Table III, the portion of alignment compared to the overall pipeline execution time is significantly reduced, e.g., to approx. five percent for HANA alignment server compared to approx. 20 percent for BWA alignment both processing the second-largest file size in Exp. D. In contrast when executing GDPPs with HANA alignment server instead of BWA at a split size of 25, the relative improvement rates drops below 90 percent, especially for the smallest file size with a relative improvement of 74 percent.

TABLE IV. INTERSECTION OF RESULT SET FROM HANA ALIGNMENT SERVER AND BWA. IN TOTAL, HANA ALIGNMENT SERVER CREATES APPROX. 16M MORE ALIGNMENTS THAN BWA WHILE LEAVING ONLY HALF THE READS UNALIGNED.

	Total (#)	Aligned (#)	Unaligned (#)
BWA∩HANA	274,785,274	254,223,773	20,561,501
BWA	329,338,990	286,491,783	42,847,207
HANA	345,805,881	324,241,424	21,564,457

Table IV shows result set intersections of read alignment output from BWA and HANA alignment, respectively. We concentrate on a quantitative analysis of our obtained results since the selected benchmark data was taken from the 1,000 genome project [5]. Thus, it is no gold standard available that could be used to validate obtained alignment results.

The result set produced by BWA contains less read alignments than HANA alignment server, which includes the exact amount of read alignments from the input file listed as data set 7 in Table II. Both alignment algorithms share the majority of joint read alignments, i.e., approx. 274M reads. From them, a small proportion is made up from unaligned reads, i.e., reads for which the alignment algorithm could not find a suitable position in the genome. BWA leaves more than 42M reads unaligned, which is about twice as much as HANA alignment server. Thus, HANA maps a total of 324M read alignments, i.e., about 18M more read alignments than BWA with only 286M.

VI. Evaluation and Discussion

Our conducted benchmarks verify two hypotheses. Firstly, the usage of the IMDB as primary storage system is beneficial for integrating established alignment algorithms, such as BWA, as well as optimized alignment algorithms for IMDB technology, such as HANA

TABLE III. COMPARISON OF INDIVIDUAL EXECUTION TIMES FOR BWA (I) AND HANA (II) ALIGNMENT (R = RELATIVE IMPROVEMENT, S = SPEED UP).

	I	II	I	II	I	II	I	II	I	II	I	II	I	II
Size [Gbp]	0.5		1.0		2.0		4.0		7.9		15.8		31.6	
$t_A[s]$	1,100	231	2,159	409	3,900	690	7,029	1,256	13,626	2,305	25,147	4,931	54,034	10,503
$t_B[s]$	808	153	1,622	377	2,860	421	5,259	806	10,364	1,542	18,707	2,861	41,520	5,276
$t_C[s]$	283	178	520	330	943	529	1,761	860	3,377	1,427	6,609	3,443	12,673	8,064
$t_D[s]$	130	212	245	278	470	355	893	566	1,733	1,016	3,387	1,685	6,275	2,676
$R_B[\%]$	27	34	25	8	27	39	25	36	24	33	26	42	23	50
$R_C[\%]$	74	23	76	19	76	23	75	32	75	38	74	30	77	23
$R_D[\%]$	88	8	89	32	88	49	87	55	87	56	87	66	88	75
$t_{1_{Aln}}[s]$	699	23	1,395	43	2,446	93	4,250	179	8,099	350	14,504	680	24,417	1,342
$t_{25_{Aln}}[s]$	37	6	66	6	129	10	252	20	501	43	1,001	87	2,000	176
$R_{1:25}[\%]$	95	74	95	86	95	89	94	89	94	88	93	87	92	87
$S_{1:25}$	19x	4x	21x	7x	19x	9x	17x	9x	16x	8x	14x	8x	12x	8x
$R_{1_{I:II}}[\%]$	97		97		96		96		96		95		95	
$R_{25_{I:II}}[\%]$	84		91		92		92		91		91		91	
$S_{1_{I:II}}$	30x		32x		26x		24x		23x		21x		18x	
$S_{25_{I:II}}$	6x		11x		13x		13x		12x		12x		11x	

alignment server. Secondly, our platform supports the parallel execution of intermediate process steps across multiple computing nodes, which results in an additional performance improvement compared to the execution on a single computing node.

We observed the best relative improvement for GDPP using the IMDB as primary data storage and BWA as alignment algorithm with at least 74 percent on single computing node and up to 89 percent on 25 computing nodes. Thus, the overall pipeline execution time with BWA as alignment algorithm correlates to the number of base pairs contained in the FASTQ file in a linear way. However, improvements when using 25 nodes is still below our expectation of a factor 25 due to the use of traditional tools, e.g., SAMtools, which partially operate in a single threaded way.

Relative improvement observed for pipelines using HANA alignment server remain below the numbers achieved by pipelines using BWA. For the three largest file sizes, we achieve relative improvements between 33 and 42 percent on a single computing node and between 55 and 66 percent on 25 computing nodes. For this pipeline, runtime limitations from third-party tools, such as SAMtools, apply. Using HANA alignment leads to a significant reduction of overall execution times. Thus, it results in a detrimental shift of the ratio between time overhead needed for setting up parallelization, i.e., splitting and merging, and the time improvement due to parallelization. For example, the time needed for merging intermediate results in GDPPs using file storage as primary storage medium increases up to 10x when distributing pipeline execution across 25 computing nodes. It shows almost no impact for BWA alignment as the majority of time is spent on alignment. In contrast, HANA alignment reduces the proportion of time needed for alignment significantly, so that the impact of remaining operations, e.g., merging of partial results, on the overall runtime duration increases. In addition, HANA alignment reaches its minimum execution time of approx. 6 seconds when operating on the small benchmark files, leading to worse relative improvement rates compared to BWA.

As a result, GDPPs using HANA alignment have a worse speedup compared to BWA alignment while the overall execution time is significantly smaller because of its near optimal use of parallelization. In addition, pipeline execution times reach a lower boundary that reduces relative performance improvements for the three smallest files. For example, the time needed for alignment of the smallest benchmark data set drops below ten seconds on a single node.

Scaling factors for the overall execution time across all experiments and file sizes indicate a constant and predictable system behavior of our system for varying input file size. Thus, we are able to predict execution time, which helps to supervise the correct system functionality, e.g., to detect broken computing resources. We do not elaborate on costs emerging when using the cluster as it remains available during pipeline execution to other users that can start their own pipelines or use one of our other applications, e.g., patient cohort analysis.

Furthermore, our results stress the benefits of using an IMDB for operating on intermediate results of the pipeline execution. The pipeline optimized for the IMDB no longer uses individual tools operating on files for specific process steps, such as sorting, merging, and indexing. These operations are directly performed as an integral operation of the incorporated IMDB without the need to create intermediate files in the file system at all.

VII. Conclusion

In our contribution, we shared details about building blocks of in-memory computing and proofed the applicability of the IMDB technology for genome data

processing and its real-time analysis. For that, we conducted expressive benchmarks, which underline that our computing platform improves the overall runtime by enabling a) scale-out, b) seamless integration of existing tools, such as BWA, and c) development of specific algorithms, such as HANA alignment and variant calling, which are directly embedded as core components in the incorporated IMDB.

We shared insights in specific system components of our technology stack, such as task scheduling, worker and annotation framework, which build the foundation for consistent and scalable high-throughput data processing. To enable reproducible genome data analysis, we shared details about our GDPP modeling notation for processing and analysis pipelines based on BPMN.

Ultimately, we linked our technology building blocks to concrete requirements for specific applications in the context of precision medicine and clinical research, e.g., cohort analysis and search in unstructured clinical trial documents. These applications are the results of interdisciplinary cooperation with researchers, clinicians, and medical experts. As a result, we were able to monitor improvements in the daily working routine of these target audiences by providing them our cloud applications.

Our future work focuses on integration of additional tools and services into our in-memory computing platform to further support researchers and clinicians in the course of precision medicine. In addition, our future research will focus on patients to support them in explore latest international medical knowledge about critical diseases and possible treatments, such as cancer disease.

References

[1] M.-P. Schapranow, F. Häger, and H. Plattner, "High-Performance In-Memory Genome Project: A Platform for Integrated Real-Time Genome Data Analysis," in Proceedings of the 2nd Int'l Conf on Global Health Challenges. IARIA, Nov 2013, pp. 5–10.

[2] K. Jain, Textbook of Personalized Medicine. Springer, 2009.

[3] H. Plattner and M.-P. Schapranow, Eds., High-Performance In-Memory Genome Data Analysis: How In-Memory Database Technology Accelerates Personalized Medicine. Springer-Verlag, 2014.

[4] I. Bozic et al., "Accumulation of Driver and Passenger Mutations during Tumor Progression," Proceedings of the National Academy of Sciences of the United States of America, vol. 107, no. 43, Oct. 2010, pp. 18 545–50.

[5] The 1,000 Genomes Project Consortium, "A Map of Human Genome Variation from Population-scale Sequencing," Nature, vol. 467, no. 7319, Oct. 2010, pp. 1061—1073.

[6] Illumina, "HiSeq 2500 Sequencing System," http://res.illumina.com/documents/products/datasheets/datasheet_hiseq2500.pdf [retrieved: May 30, 2014], Jan 2014.

[7] National Human Genome Research Institute, "DNA Sequencing Costs," http://www.genome.gov/sequencingcosts/ [retrieved: May 30, 2014], Apr 2013.

[8] J. C. McCallum, "Memory Prices (1957-2013)," http://www.jcmit.com/memoryprice.htm [retrieved: May 30, 2014], Feb 2013.

[9] W. J. Ansorge, "Next-Generation DNA Sequencing Techniques," New Biotechnology, vol. 25, no. 4, 2009, pp. 195–203.

[10] S. Pabinger et al., "A Survey of Tools for Variant Analysis of Next-generation Genome Sequencing Data," Brief Bioinform, Jan. 2013.

[11] S. Wandelt et al., "Data Management Challenges in Next Generation Sequencing," Datenbank-Spektrum, vol. 12, no. 3, 2012, pp. 161–171.

[12] V. Fusaro, P. Patil, E. Gafni, D. Wall, and P. Tonellato, "Biomedical cloud computing with amazon web services," PLoS Comput Biol, vol. 7, no. 8, Aug 2011, p. e1002147.

[13] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg, "Searching for SNPs with Cloud Computing," Genome Biol, vol. 10, no. 11, 2009, p. R134.

[14] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and Memory-efficient Alignment of Short DNA Sequences to the Human Genome," Genome Biol, vol. 10, no. 3, 2009, p. R25.

[15] R. Li et al., "SNP Detection for Massively Parallel Whole-Genome Resequencing," Genome Research, vol. 19, no. 6, 2009, pp. 1124–1132.

[16] J. Goecks, A. Nekrutenko, and J. T. The Galaxy Team, "Galaxy: A Comprehensive Approach for Supporting Accessible, Reproducible, and Transparent Computational Research in the Life Sciences," Genome Biology, vol. 11, no. 8, Aug 2010, p. R86.

[17] M. Reich et al., "Gene Pattern 2.0," Nat Genet, vol. 38, no. 5, May 2006, pp. 500–501.

[18] B. Néron et al., "Mobyle: A New Full Web Bioinformatics Framework," Bioinformatics, vol. 25, no. 22, Nov 2009, pp. 3005–11.

[19] H. Plattner, A Course in In-Memory Data Management: The Inner Mechanics of In-Memory Databases, 1st ed. Springer, 2013.

[20] M.-P. Schapranow, "Transaction Processing 2.0," Master's thesis, Hasso Plattner Institute, 2008.

[21] P. Svensson, "The Evolution of Vertical Database Architectures – A Historical Review," in Proceedings of the 20th Int'l Conf on Scientific and Statistical Database Management. Springer-Verlag, 2008, pp. 3–5.

[22] A. Vajda, Programming Many-Core Chips. Springer, 2011.

[23] A. Clements, Computer Organization & Architecture: Themes and Variations. Cengage Learning, 2013.

[24] A. S. Tanenbaum, Modern Operating Systems, 3rd ed. Pearson Prentice Hall, 2009.

[25] J. M. Hellerstein and M. Stonebraker, Readings in Database Systems, 4th ed. MIT Press, 2005.

[26] S. S. Lightstone, T. J. Teorey, and T. Nadeau, Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and more. Morgan Kaufmann, 2007.

[27] J. M. Hellerstein, M. Stonebraker, and J. Hamilton, Architecture of a Database System, Foundation and Trends in Databases. now Publishers, 2007, vol. 1.

[28] M. Gardiner-Garden and M. Frommer, "CpG Islands in Vertebrate Genomes," Mol Biol, vol. 196, no. 2, July 1987, pp. 261–282.

[29] A. Knöpfel, B. Grone, and P. Tabeling, Fundamental Modeling Concepts: Effective Communication of IT Systems. John Wiley & Sons, 2006.

[30] T. K. Das and M. R. Mishra, "A Study on Challenges and Opportunities in Master Data Management," Int'l Journal of Database Mgmt Syst, vol. 3, no. 2, May 2011.

[31] The Genome Reference Consortium, "Genome Assemblies," http://www.ncbi.nlm.nih.gov/projects/genome/assembly/grc/data.shtml [retrieved: May 30, 2014].

[32] A. Rector, W. Nolan, and S. Kay, "Foundations for an Electronic Medical Record," Methods of Information in Medicine, 1991, pp. 179–186.

[33] A. T. Holdener, AJAX: The Definitive Guide, 1st ed. O'Reilly, 2008.

[34] D. Crockford, "RFC4627: The application/json Media Type for JavaScript Object Notation (JSON)," http://www.ietf.org/rfc/rfc4627.txt [retrieved: May 30, 2014], July 2006.

[35] M.-P. Schapranow, "Apps of analyze genomes," http://we.analyzegenomes.com/apps/ [retrieved: May 30, 2014], March 2014.

[36] U.S. National Institutes of Health, "Clinicaltrials.gov," http://www.clinicaltrials.gov/ [retrieved: May 30, 2014], 2013.

[37] U.S. National Library of Medicine, "Unified Medical Language System (UMLS)," http://www.nlm.nih.gov/research/umls/ [retrieved: May 30, 2014], Jul 2013.

[38] S. Krawetz, Bioinformatics for Systems Biology. Humana Press, 2009.

[39] M. Weske, Business Process Management - Concepts, Languages, Architectures. Springer, 2007.

[40] M. Owen and J. Raj, "BPMN and Business Process Mgmt," http://www.omg.org/bpmn/Documents/6AD5D16960.BPMN_and_BPM.pdf [retrieved: May 30, 2014], 2003.

[41] M.-P. Schapranow, H. Plattner, and C. Meinel, "Applied In-Memory Technology for High-Throughput Genome Data Processing and Real-time Analysis," in System on Chip (SoC) Devices in Telemedicine from LABoC to High Resolution Images, 2013, pp. 35–42.

[42] The 1000 Genomes Project Consortium, "VCF (Variant Call Format) Version 4.1," http://www.1000genomes.org/wiki/Analysis/Variant+Call+Format/vcf-variant-call-format-version-41 [retrieved: May 30, 2014], Oct. 2012.

[43] A. Bog, K. Sachs, and H. Plattner, "Interactive Performance Monitoring of a Composite OLTP and OLAP Workload," in Proceedings of the International Conference on Management of Data. Scottsdale, AZ, USA: ACM, 2012, pp. 645–648.

[44] F. Färber et al., "SAP HANA Database: Data Management for Modern Business Applications," SIGMOD Rec., vol. 40, no. 4, Jan. 2012, pp. 45–51.

[45] National Center for Biotechnology Information, "All Resources," http://www.ncbi.nlm.nih.gov/guide/all/ [retrieved: May 30, 2014].

[46] S. A. Forbes et al., "The Catalogue of Somatic Mutations in Cancer: A Resource to Investigate Acquired Mutations in Human Cancer," Nucleic Acids Research, vol. 38, 2010.

[47] L. R. Meyer et al., "The UCSC Genome Browser Database: Extensions and Updates 2013," Nucleic Acids Research, 2012.

[48] M. Krallinger, A. Valencia, and L. Hirschman, "Linking Genes to Literature: Text Mining, Information Extraction, and Retrieval Applications for Biology," Genome Biology, vol. 9, supplement 2, 2008, p. S8.

[49] H. Li et al., "The Sequence Alignment/Map Format and SAMtools," Bioinformatics, vol. 25, no. 16, 2009, pp. 2078–2079.

[50] M. Stonebraker et al., "C-store: A Column-oriented DBMS," in Proceedings of the 31st International Conference on Very Large Data Bases. VLDB Endowment, 2005, pp. 553–564.

[51] G. P. Copeland and S. N. Khoshafian, "A Decomposition Storage Model," in ACM SIGMOD Record, vol. 14, no. 4. ACM, 1985, pp. 268–279.

[52] S. B. Needleman and C. D. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins," Mol Biol, vol. 48, no. 3, Mar. 1970, pp. 443–53.

[53] T. F. Smith and M. S. Waterman, "Identification of Common Molecular Subsequences," Journal of molecular biology, vol. 147, no. 1, Mar. 1981, pp. 195–7.

[54] P. Ferragina and G. Manzini, "Opportunistic Data Structures with Applications," in Proceedings of the 41st Annual Symposium on Foundations of Computer Science. IEEE, 2000, pp. 390–398.

[55] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in Proceedings of the 6th Symposim on Operating Systems Design and Implementation, 2004, pp. 137–150.

[56] A. Stabenau et al., "The Ensembl Core Software Libraries," Genome Research, vol. 14, no. 5, 2004, pp. 929–933.

[57] R. Nielsen et al., "Genotype and SNP Calling From Next-generation Sequencing Data," Nature Reviews Genetics, vol. 12, no. 6, 2011, pp. 443–451.

[58] D. R. Bentley et al., "Accurate Whole Human Genome Sequencing Using Reversible Terminator Chemistry," Nature, vol. 456, no. 7218, 2008, pp. 53–59.

[59] M. Margulies et al., "Genome Sequencing in Microfabricated High Density Picoliter Reactors," Nature, vol. 437, no. 7057, 2005, pp. 376–380.

[60] Hasso Plattner Institute, "Future SOC Lab," http://www.hpi.uni-potsdam.de/forschung/future_soc_lab.html [retrieved: May 30, 2014], Feb 2014.

[61] Intel Corporation, "Intel Product Quick Reference Matrix," http://cache-www.intel.com/cd/00/00/47/64/476434_476434.pdf [retrieved: May 30, 2014], Apr 2011.

[62] ——, "Intel Solid-State Drive 520 Series," http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/ssd-520-specification.pdf [retrieved: May 30, 2014], Feb 2012.

[63] H. Li and R. Durbin, "Fast and Accurate Short Read Alignment with Burrows-Wheeler Transformation," Bioinformatics, vol. 25, 2009, pp. 1754–1760.