Datenbank-Spektrum manuscript No.
(will be inserted by the editor)

Data Management Challenges in Next Generation Sequencing

Sebastian Wandelt · Astrid Rheinländer · Marc Bux · Lisa Thalheim · Berit Haldemann · Ulf Leser

Received: date / Accepted: date

Abstract Since the early days of the Human Genome Project, data management has been recognized as a key challenge for modern molecular biology research. By the end of the nineties, technologies had been established that adequately supported most ongoing projects, typically built upon relational database management systems. However, recent years have seen a dramatic increase in the amount of data produced by typical projects in this domain. While it took more than ten years, approximately three billion USD, and more than 200 groups worldwide to assemble the first human genome, today's sequencing machines produce the same amount of raw data within a week, at a cost of approximately 2000 USD, and on a single device. Several national and international projects now deal with (tens of) thousands of genomes, and trends like personalized medicine call for efforts to sequence entire populations. In this paper, we highlight challenges that emerge from this flood of data, such as parallelization of algorithms, compression of genomic sequences, and cloud-based execution of complex scientific workflows. We also point to a number of further challenges that lie ahead due to the increasing demand for translational medicine, i.e., the accelerated transition of biomedical research results into medical practice.

1 Introduction

Modern biological research is driven by high throughput experiments, i.e., techniques which are able to uncover characteristics of thousands of entities in a single experiment. Examples of such techniques are gene chips, measuring the expression levels of all human genes in a given sample on a single silicon chip, or proteomics using mass spectrometry, which is able to detect several thousands of proteins in a sample. High throughput techniques are inevitable to obtain comprehensive views into the functioning of a cell, a tissue, an individual, or a disease. On the downside, they typically come along with high levels of noise which require complex statistical measures to cope with. Furthermore, they typically produce large amounts of data whose appropriate management can be a real challenge [44].

The most prominent high throughput technique is (genomic) sequencing. The goal of sequencing is to unravel the ordered sequence of nucleic acids that form the DNA of a given sample; in genomic sequencing, these sample sequences are entire chromosomes of living beings. A human chromosome has between 50 and 250 million nucleic acids. Revealing the sequence of such a long piece of DNA until today can only be achieved by first breaking it into many small overlapping pieces (called reads) which can be sequenced directly by appropriate devices. The fragmentation of a chromosome into small reads is best understood as a stochastic process, i.e., reads are essentially sampled at random from the chromosome. Heavy over-sampling is required to ensure that the entire chromosome is covered with sufficient probability. Sequencing projects striving for high quality typically over-sample by a factor of ten or more. Another important factor is the read length, which for many years used to be between 500 and 900 base pairs. Accordingly, sequencing even the shortest human chromosome typically requires the generation of app. 800,000 reads, a laborious and time-consuming undertaking as machines until the end of the 1990s were able to sequence only 40 – 100 reads in parallel. Obtaining the sequence of the original chromosome requires, in a second step, comparing all these reads

Department of Computer Science, Humboldt-Universität zu Berlin,
Unter den Linden 6, 10099 Berlin, Germany
Tel.: +49-30-2093-3902
Fax: +49-30-2093-5484
{wandelt,rheinlae,bux,thalheim,haldeman,leser}@
informatik.hu-berlin.de

to each other to compute their most probable order (see Figure 1). This process is called assembly [33].

The feasibility of this complex process was impressively demonstrated (and developed) in the Human Genome Project, in which hundreds of groups and labs worldwide cooperated to obtain the first entire, approximately three billion bases long genome of a human being [14]. This project lasted for roughly 15 years at an estimated cost of three billion USD. During the time of this project, sequences from other species, especially model organisms like mouse of fruit flies, were obtained as well. Furthermore, also molecules other than DNA were studied to tackle the complexity of cells, like partial or complete mRNA. Together, these projects led to an exponential growth of the international sequence databases Genbank, EMBL, and DDBJ.

The main operation performed (and still performed today) on these sequences is approximate matching. Accordingly, approximate matching of a given query sequence to a database of millions of other sequences has been one of the main challenges in genomic data management for a long time [2]. This problem attracted many researchers and has lead to a family of heuristic algorithms typically executed on large clusters.

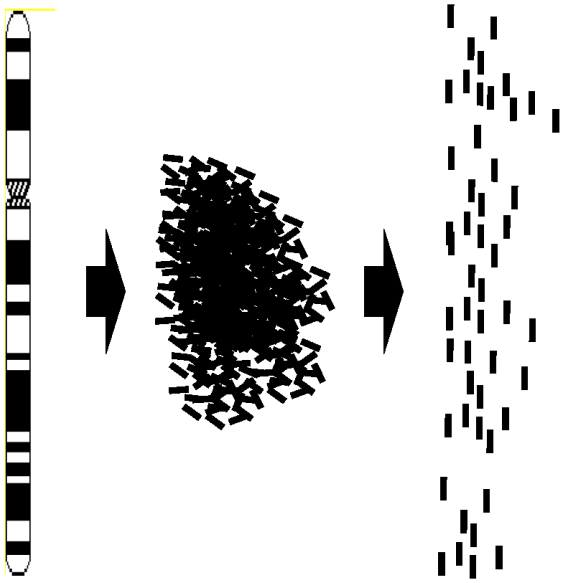

Fig. 1: Sequencing a chromosome. First, short parts of the chromosome are sampled in a stochastic process, yielding reads. To recover the original sequence, reads are compared all-against-all to find significant overlaps that are used to compute the most probable order. With NGS data, assembly usually is omitted, but reads are directly aligned against a known reference sequence from the same species.

Matching is particularly challenging during assembly, because here millions of reads have to be compared to each other. Such problems could only be tackled by very few groups in the world as they required hundreds of machines running in parallel. However, until recently only very few groups in the world had the need to perform such assemblies as only large, international consortia were able to produce the necessary data in the first place. Storing genomic sequences and performing simple operations, like the approximate matching of a single query sequence mentioned above, was not so much of a problem, as the total amount of existing data, though growing fast, for a long time did not exceed a terabyte. Consequently, research in genomic data management somewhat declined in the early 2000s, as modestly expensive servers were capable of performing most of the required analysis in acceptable time.

This situation has changed dramatically in the last three to four years, which brought a new class of sequencing devices producing sequences at an unprecedented scale. These so called next-generation sequencing (NGS) machines still follow the same principles outlined above. However, NGS machines are capable of sequencing millions to even billions of reads in parallel, leading to the production of several terabytes of raw signal data, corresponding to several hundreds of gigabytes of sequence data, in one week on a single machine [10]. Hundreds of such machines already have been put in production worldwide. Sequencing thus becomes a commodity which is not only carried out by large consortia any more, but also by single hospitals or research institutions. The scale has shifted dramatically and at increasing pace: While a few years ago the most ambitious international sequencing project was the 1000 Genome Project, aiming at sequencing large parts of the genomes of 1,000 humans around the world, in 2011 the UK10K project announced to sequence 10,000 individuals. The International Cancer Genome Consortium (ICGC) recently started to sequence 500 individuals twice (one healthy, one disease sample) for 50 different types of cancer, which yields 50,000 genomes in total [23]. ”Just” sequencing already has lost its status as research but became a commercial service offered by dozens of companies around the world at falling prices.

The leaps in sequencing technology have been far greater than the advances in computing (see Figure 2) [41], which lead to a series of novel or re-appearing challenges in genomic data management. In this survey, we focus on three such challenges.

First, NGS reads are many, but short. Assembling a genome from NGS reads is highly complex if not impossible. However, since humans at the DNA level are more than 99% identical, ”sequencing” an individual today can be undertaken by generating these short reads and mapping them to a known human reference genome. This process is called read alignment or read mapping; in a nutshell, the task is

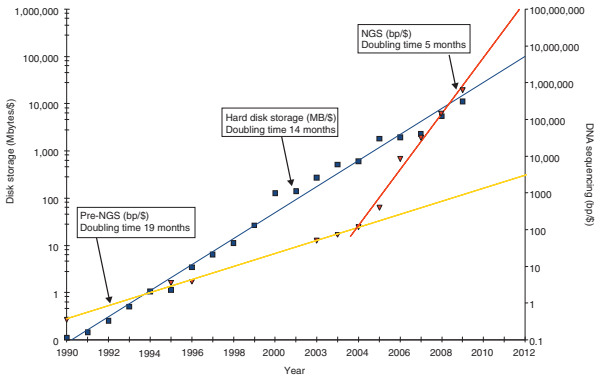

Fig. 2: Development of cost of storing one byte on hard disk versus cost for sequencing one base pair (bp). Clearly, sequencing becomes cheaper than storing sequences. Image from [49], BioMed Central.

to map hundreds of millions short strings onto a reference string of length three billion in the presence of errors. Efficiently solving this task requires advanced data structures and highly parallel algorithms which we will present in Section 2.

Second, simply storing or transmitting genomes becomes a challenge when the numbers rise from one (as in the HGP) to several thousands (as in the ICGC). The net amount of space necessary to store 1000 human genomes is three terabytes, but storing real sequences requires much more space due to attached quality information and other meta data. Furthermore, sequences often need to be transferred to special infrastructures to allow for their analysis (see next point), which is severely hampered by bandwidth limitations. Accordingly, sequence compression has become a hot topic which we briefly review in Section 3.

Third, the type of questions to the genomes have changed. While for a long time most data analysis was performed on a single or a few genomes only, recent research areas such as personalized medicine or translational medicine require the analysis of hundreds of genomes in a single "run", i.e., using a complex pipeline of several tools working in serial or in parallel on those sequences to produce a certain result. Such pipelines are often modelled as scientific workflows, and efficiently executing complex scientific workflows on large data sets has become an area of intense research which we discuss in Section 4.

2 Read Mapping

A fundamental task in NGS projects is the mapping of sequence reads to a known reference genome (so-called *"read alignment"*). Aligning sequences to each other is important in various scenarios, e.g., for finding structural, functional, or evolutionary relationships between sequences, or to identify highly conserved regions in a genome [36]. In this section, we introduce the problem of read alignment, describe basic algorithms, and review state-of-the-art tools which can be executed in a Cloud. We conclude with a discussion of open challenges.

2.1 Problem definition and base algorithms

The purpose of read mapping algorithms is to report all near-exact matches for the given reads in a known reference sequence. More formally, read mapping algorithms search for each given read r all 4-tuples (c,s,p,e), where c is the chromosome, s the positive or negative strand, and p the relative start position of r on c, such that r matches a substring of c with e errors. In most cases, the number of maximum allowed errors varies between one and ten percent of the read length. Depending on read type and application, allowed errors are either only mismatches, or insertions and deletions (indels) of bases as well.

Basic read alignment algorithms which consider only mismatches ground on Hamming distance computation and can be found in linear time. Algorithms that allow gaps in the alignment usually use some form of dynamic programming, and solving them exactly typically requires $O(m*n)$ operations, where m is the length of the genome and n is the length of the read being aligned.

Given these estimates and considering the amount of data which is being produced by NGS devices, the most challenging aspect is to design algorithms that achieve sublinear alignment speed. This is usually achieved by one of the following two means. First, one string is indexed; in read mapping, this is the reference. Frequently used index structures are variants of suffix trees [51] or hashing of q-grams [1]. Second, the problem is not solved exactly but only "sufficiently" good. Here, most algorithms build upon the *"seed-and-extend"* paradigm, exploiting the fact that an alignment which allows at most e mismatches must contain at least one exact match (*"seed"*) of a substring of the read of length $\lfloor \frac{n}{e+1} \rfloor$ [4]. A softer, faster, yet less accurate formulation is that in the core of every alignment with few errors there usually is an area where both strings match exactly.

Seed-and-extend algorithms typically first find such exact matches (called seeds) by comparing the read against the indexed reference. Next, seeds are elongated both to the left and the right to find full alignments with at most e mismatching bases (*"extend"*). Such methods are particularly well suited for short NGS reads (i.e., reads shorter than 100 base pairs) as these seldomly contain many errors or large gaps. For a comprehensive review of these tools see [50,33].

As sequencing technology is evolving, the length of the NGS reads increases significantly (*"long reads"*). Compared to short reads, long reads have a higher chance to be error-prone. Particularly, insertions or deletions appear in much

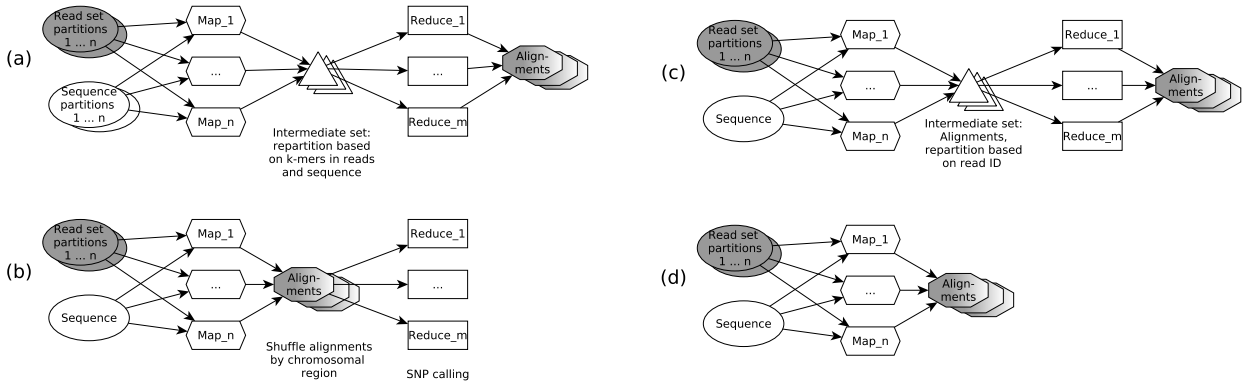

Fig. 3: Data partitioning strategies of Cloud-based read alignment tools: (a) CloudBurst, (b) Crossbow, (c) SeqMapReduce, (d) CloudAligner.

larger numbers than for short reads [32]. Thus, algorithms for long read alignment should be able to deal with larger gaps. Existing short read mapping tools are not well-suited for this tasks since they only run efficiently for ungapped alignments or when a very small number of gaps is allowed. Their performance quickly degrades when the read length increases or when more errors are allowed [32]. BLAST [1] and BLAT [25] are very popular tools for long read alignment that were initially designed for mapping comparatively few reads produced with the Sanger-Coulson [43] method. It has been shown that both tools do not scale well to larger data sets produced by NGS devices [42]. State-of-the-art approaches for long read alignment use various techniques for accelerating the alignment process, such as creating memory efficient genome indexes [16,32,28], multi-threaded execution [35], and gapped seeds [54].

2.2 Cloud-based solutions

To further scale up read mapping, as a reaction to the ever increasing degree of parallelism in sequencing technology, recently also approaches using a cluster of independent machines (a cloud) gained some popularity. Most of these build upon the *MapReduce* [11] programming paradigm for processing *embarrassingly parallel* [17] problems on huge data sets. The main feature of MapReduce is that it effectively and robustly manages the parallelization and distribution of input data and user code on a large number of compute nodes. Clearly, read alignment is an embarrassingly parallel problem, since each read can be aligned individually to the reference without any dependencies on other reads. Thus, MapReduce-based alignment tools have become popular and we briefly review the most prominent tools:

CloudBurst [46] is a seed-and-extend based tool that performs short read alignment on Amazon EC2[1] using Hadoop[2]. The alignment algorithm is modeled after Rmap [47] and aligns reads allowing mismatches only. In the Map stage, CloudBurst computes *k*-mers both for the reads and the reference sequence. In an intermediate stage, those *k*-mers occurring both in the reference and in the reads are shuffled and grouped together. Finally, the alignment is computed for all matching *k*-mers in the Reduce stage (see Figure 3 (a)).

Crossbow [29] performs read mapping and SNP calling[3]. It provides a web interface and scripts for executing Bowtie [30] on a Cloud platform, such that Bowtie aligns independent subsets of the read data set to the entire reference sequence in the Map stage (cf. Fig. 3 (b)). In an intermediate stage, all computed alignments are grouped by chromosomal region and finally, SNP calling is performed in the Reduce stage.

SeqMapReduce [34] is based on an in-memory seed-and-extend algorithm, using early filtering and late alignment pair emission to accelerate execution time. As displayed in Figure 3 (c), SeqMapReduce computes the alignments using Map and builds the final result set in the Reduce phase by grouping all alignments by read ID. Significant performance improvements of SeqMapReduce were reported over CloudBurst, which are attributed to the early filtering of reads and the avoidance of the massive I/O operations performed by CloudBurst during the shuffling phase. Unfortunately, there are currently no executables accessible.

CloudAligner [37] supports the alignment of both short and long reads. It also uses the seed-and-extend idea but only uses Mappers for aligning small parts of the read set to the reference (see Figure 3 (d)), avoiding any costly shuffling or repartitioning of the data. CloudAligner has been reported to be significantly faster than other Cloud-based tools while achieving comparable accuracy.

[1] http://http://aws.amazon.com/ec2/
[2] http://hadoop.apache.org/

[3] SNP calling attempts to predict which of the disagreements between reference and query sequences are due to Single Nucleotide Polymorphisms.

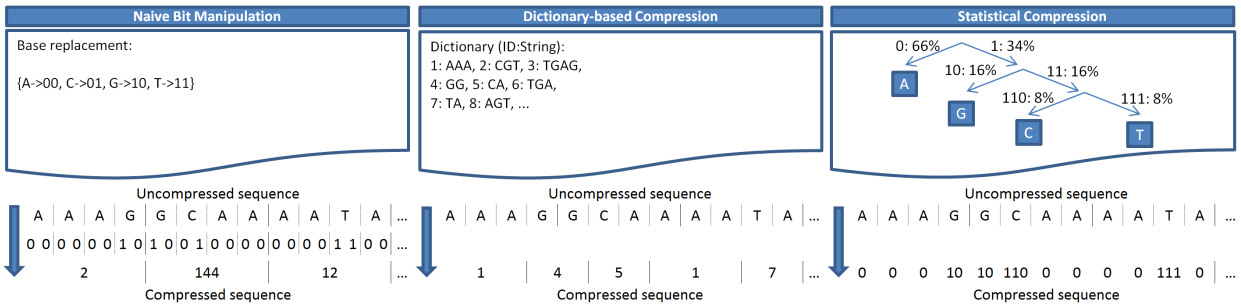

Fig. 4: An example of compression with standard compression schemes.

2.3 Open challenges

Although much research has already been carried out to increase the performance of read mapping tools, scalability remains an open challenge. It is still not clear whether state-of-the-art parallel or distributed read aligners can process the amount of data produced in large sequencing projects in a reasonable amount of time (and space).

Finding alignments exhibiting large gaps demands special algorithms as usual heuristics typically produce unsatisfying accuracy when faced with such data. Such data especially appears in transcriptome projects sequencing mature mRNA. The majority of genes in eukaryotic organisms contain some non-coding stretches of DNA called introns. The genes' transcripts undergo a process called splicing, where these introns are excised (and sometimes also other parts which we ignore here). Introns can be several hundred thousand nucleotides long; thus, aligning back a sequenced mRNA to a genome has to deal with exceptionally large gaps. Another area where large gaps appear is cancer research, because cancerous cells often exhibit a high level of genomic instability leading to significant genomic rearrangements. Both of these problems are highly active areas of research.

Another open question is how to integrate quality scores into read mapping algorithms. All sequencing machines output quality scores along with each base, indicating the probability of this particular base being correct. Using these quality scores during read mapping is known to improve mapping accuracy [48], but is not possible with current tools for large-scale read mapping.

With an ever-increasing number of read mapping software packages, it is quite a challenge to pick the best one for a particular sequencing project, and to gauge the quality of the resulting alignment. This is further complicated by frequent updates to the software packages, which may change the performance in terms of both running time and alignment quality. Although some papers appeared that compared the performance of different tools, a widely accepted benchmark against which read mapping software could be evaluated has yet to emerge [22].

3 Sequence Compression

The growth of genome data makes it more necessary than ever to carefully think about economical storage and transmission. Storing a single human genome requires three gigabyte of space; accordingly, 50.000 genomes require 150 terabyte. While storing such amounts of data is possible, transmitting them, for instance to use cloud resources for their analysis (see next section), is almost impossible. In fact, large quantities of sequences today are usually shipped by hard disc before analysing them on large clusters. It is an ongoing research challenge how to encode/compress biological sequences in a way such that they can be 1) economically stored and transmitted and 2) accessed easily.

3.1 Traditional compression algorithms

Traditional sequence compression schemes can be separated into three groups. *Naive bit manipulation* schemes exploit encodings of two or more symbols into one byte [6,52]. *Dictionary-based* or substitutional schemes replace long repeated substrings by references to a dictionary, which is usually built at runtime [3,26]. *Statistical* or entropy encoding schemes derive a probabilistic model from the input. Based on partial matches of subsets of the input, this model predicts the next symbols in the sequence [15]. High compression rates are possible if the model always indicates high probabilities for the next symbol, i.e. if the prediction is reliable. One example compression for each group is shown in Figure 4. The compression ratio of these compression schemes is 3:1 – 6:1.

3.2 Referential compression schemes

Referential compression schemes recently emerged as a new compression scheme [7,19,27]. The key idea is to encode

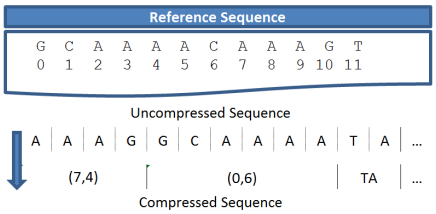

Fig. 5: Compression with a referential algorithm.

an input sequence with respect to an external (set of) reference sequence(s). The main differences to dictionary-based algorithms are that reference sequences are usually fixed beforehand and reference sequences are not being added to the compressed file. Properly fixing the reference sequence is very important, since biological sequences often undergo several revisions and many different versions are available online. Given that the reference sequence is evolutionary close and available to the decompressor, referential compression schemes allow for compression rates of 400:1 or higher.

One example for a referentially compressed sequence is shown in Figure 5. There exist two matching subsequences with respect to the reference. The interval match $(7,4)$ indicates that the current input matches the reference for four symbols starting at position seven. In addition, the short sequence *TA* is stored in a raw manner, since there exists no good match in the reference sequence for *TA*.

Existing referential compression schemes differ in various ways. One issue is encoding of the compressed blocks; here, especially Golomb or Fibonacci codes are used frequently [20,8]. Efficiently finding long matches in the chosen reference sequence is another challenge. It can be addressed by indexing the reference (see Section 3.1), but due to the size of these index structures, they often do not fit into main memory. Therefore, the access time is restricted by the hard disk. One solution is to conduct a local search in the neighbourhood of previous matches. This strategy has a biological foundation: Often two parts of a genome might only be different by few bases (base insertion, base removal, or base mutation). Whenever finding an appropriate match

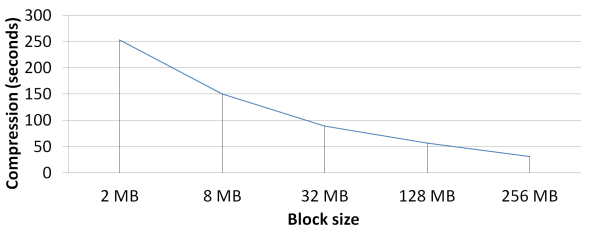

Fig. 6: Time required for referential compression of a human genome split into blocks of different size.

in the neighbourhood, one can avoid consulting the index structure, although it might yield longer matches and therefore better compression. Another solution is to split up the input and reference into blocks of fixed size and handle each of these blocks separately. Besides having lower main memory requirements, this strategy also allows for parallelization of compressor and decompressor implementations.

In Figure 6, compression time for different block sizes and seven compression threads is shown using an algorithm we are currently working on (manuscript in preparation). At a block size of 2 megabyte, which yields a main memory usage of 50 megabyte in total, the compression of a human genome takes 250 seconds on an off-the-shelf laptop (Intel core i7 3610m). Using a block size of 256 megabyte, which needs roughly 6 gigabyte of main memory, the compression needs only 31 seconds. If we load the index structures into main memory beforehand, then compression can be further speed up to only 7 seconds for compressing an entire human genome.

3.3 Open challenges

The main challenges for sequence compression are scalability and compression rates. Regarding scalability, the question is still open how an optimal compression can be obtained in short time. An optimal referential compression is the one with the least space requirements, which requires to solve complex optimization problems in order to balance length of referential matches and length of raw sequences in between. To the best of our knowledge, no solution to this problem is known, nor is it known how close current methods come to this (theoretical) optimum. In any case, compression rate must be balanced with compression speed.

Another open challenge occurs as long as not a single, but a set S of thousands of sequences should be stored in compressed form. The higher the compression rates (and speed), the more similar reference and to-be-compressed sequences are. The question now is to find the one sequence s from S which is most similar to all other sequences, making s the best candidate to be used as reference. Heuristics for finding a good reference sequence can be based on k-mer hashing. High similarity of k-mers indicate high potential for compression with respect to the reference. However, at genome scale, k should be chosen higher than 15, in order to avoid too many random matches.

Another open problem is read compression. While genome compression typically only considers the sequence itself, read compression also must take quality scores into account (see Section 2.3). The compression rate of reads is dominated by the compression of these quality scores, since these scores have a higher entropy than the base symbols. Future research will have to investigate how quality scores are actually used in practice and which resolution of scores

is necessary. Lossy compression schemes are likely to play a key role.

Finally, a largely unexplored question is how to analyze compressed sequences directly, instead of decompressing them before any usage. If 1000 genomes should be compared together, little is gained by compressing them if they all need to be decompressed again before analysis. Therefore, there is a need for string search algorithms that can efficiently make use of the existing index structure of a reference sequence and referentially compressed files.

Scientific workflows have gained increased interest during the last years in computational biology (see next section). The integration of referential compression and string search into these workflows is one further open challenge.

4 Managing Analysis Pipelines

NGS technology promises biologists and clinicians insights into masses of individual genomes at reasonable speed and affordable cost. Due to the diversity of research questions emerging from this potential, hundreds of analysis methods have been proposed and are employed in practice. In this section, we discuss some typical research questions and how analysis pipelines are used to solve them. We also outline the programming paradigm of scientific workflows, which has been established as a common way of modeling these analysis pipelines, and discuss the problem of executing them in parallel on large clusters to tackle BIG data sets.

4.1 Analysis pipelines

The output generated by high-throughput sequencing machines is usually available in FASTQ format. FASTQ files comprise sets of DNA reads, each with their respective identifier, base sequence and quality scores for each base call. Quality scores are mostly utilized to assess and filter low quality reads prior to further analysis.

Reconstructing the underlying genome from the remaining short reads constitutes the first major step of most NGS analysis pipelines. How this genome reassembly is being approached largely depends on whether a well-studied and closely related genome is available as reference. If a reference genome is present, reads can be mapped onto this reference (see Section 2). Otherwise, reads have to be assembled (see Figure 1). Both techniques – an assembly more so than an alignment – are prone to error and computationally demanding. A plethora of tools has been developed for both approaches, yet no standards have been established. Since all of these proposed applications produce different results, a common practice to increase alignment quality involves running several alignment applications in parallel and merging the resulting alignments in a final step.

Further steps in the analysis pipeline mostly depend on the research question at hand. Subsequent to reference alignment, detection of variants is a common goal. A variant denotes a base that is different between the reference and the newly sequenced read, hinting towards a mutation with potential consequences for the organism. Besides these single nucleotide variants (SNVs), smaller indels[4] or larger structural variants are also important. Identified variants undergo quality assessment, filtering and characterization of functionality and/or specificity. This aggregated information can, for instance, be utilized to determine associations between a disease and mutations [31]. See Figure 7 for an abstract workflow consisting of reference alignment, variant calling, and disease-gene association.

Other common research questions include the discovery of differential expressions or splice variants in transcriptome sequences (RNA-seq), the determination of bindings of proteins to DNA to elucidate regulatory relationships between genes and transcription factors, or the study of evolutionary relationships between species or individuals. All these problems boil down to a series of computationally demanding base algorithms operating on sequences or derived information.

4.2 Parallelizing scientific workflows

Managing these often complex, intertwined and long pipelines of algorithms is a challenge that led to the development of scientific workflow management systems [12]. Scientific workflows are high-level compositions of sequential and concurrent data processing tasks, whose topology is defined by data interdependencies. They differ from business workflows mostly in the absence of any control structures; instead, only the data dependencies determine the order in which tasks may be executed. Most scientific workflow management systems provide advanced capabilities for designing, executing, and monitoring workflows. Galaxy [18] and Taverna [39] are two examples with a strong affinity for the field of bioinformatics. See Figure 8 for a typical Galaxy workflow in next-generation sequencing.

Due to ever-increasing amounts of data, the computational effort required to execute a given scientific workflow is more and more becoming a critical issue, leading to an increased interest of the community in approaches towards parallelization and distributed execution of scientific workflows. One can differentiate three types of parallelism in scientific workflows: task parallelism, pipelining and data parallelism. Task parallelism is achieved when different tasks on parallel branches of the workflow are distributed over several independent compute nodes. Pipelining denotes a form of parallelism in which sequential data processing tasks

[4] insertions and deletions

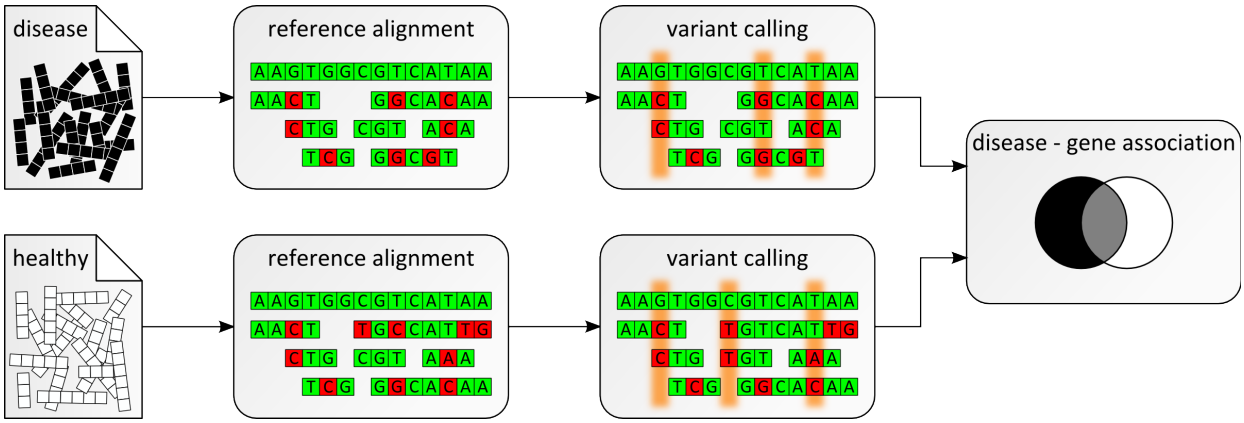

Fig. 7: An abstract workflow consisting of reference alignment, variant calling, and disease-gene association.

are executed simultaneously on different fragments of input data, i.e., the data flows through the workflow similar to oil in a pipeline. In data parallelism, input data is split in advance and the entire workflow (or parts thereof) is replicated for each data fragment. Both pipelining and data parallelism require the scientific workflow to be embarrassingly parallel [17].

In next-generation sequencing, the most time-consuming steps of the analysis pipelines are assembly, read alignment, and variant calling. For the most part, these problems are embarrassingly parallel and qualify for data parallel execution. It would therefore be desirable to execute analysis pipelines in a highly scalable, data parallel and distributed compute environment. While most scientific workflow management systems, like Taverna, provide pipelining functionality by default, they usually assume a monolithic environment and cannot take advantage of data or task parallelism. Systems stemming from cluster or grid schedulers, like Pegasus [13], are capable of detecting and exploiting task parallelism, but often do not support pipelining or data parallelism. Finally, current dataflow engines like PACT/Nephele [5,53] or PIG/HADOOP [40,55] concentrate on exploiting data parallelism, but are restricted in the patterns of parallelism that can be exploited (i.e., they usually map all pipelines into series of map / reduce tasks). A system that would be capable of using all three types of parallelism and does not hamper the topology of analysis pipelines still does not exist.

4.3 Open challenges

The proliferation of cloud computing technology has made highly scalable compute resources readily available and affordable for the end user. The usage of (public) cloud resources for execution of scientific workflows has therefore become a major topic of interest in recent years [21,24]. However, using a cloud of (usually virtual) machines efficiently for scientific workflows raises several questions that are still mostly unexplored.

First, the question of how to get input (and output) data to (and from) the cloud constitutes a severe challenge when trying to use clouds for BIG data analysis. One solution for NGS data could be compression (see Section 3); another solution is that cloud providers offer pre-configured images containing important sequence data like reference genomes. For instance, users of EC2 can mount the entire Genbank database from any image. Clearly, the latter option does not help if novel sequences are to be analyzed. Thus, the seamless integration of compression / decompression algorithms into scientific workflows is an important yet open issue.

Second, the problem of efficiently mapping workflow tasks onto heterogeneous distributed compute nodes – such as virtual machines in a cloud – is still not solved satisfactorily. Different types of parallelism may be exploited. As NGS data is huge, data transfer times must be taken into account when considering which tasks to execute on which machines. Ideally, a workflow scheduler would be able to continuously adjust the execution of a given workflow to a dynamic environment, in which bandwidth, availability of memory, and speed of assigned nodes change with high frequency, as this is exactly the situation in most public cloud environments [56]. On top, an ideal scheduler would also be able to use the elasticity offered by public clouds. Effectively utilizing elasticity in distributed workflow execution is a challenge that is not addressed adequately by any of the current systems.

5 Conclusion

NGS has dramatically increased the amount of data that must be handled by current genome projects. This trend has lead to a number of challenges that need to be addressed by the

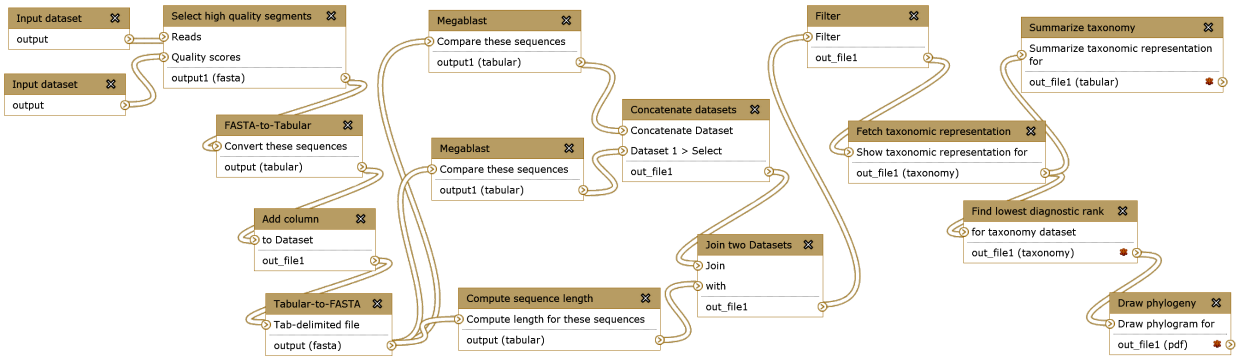

Fig. 8: A generic Galaxy workflow for performing a metagenomic analysis on NGS data. A metagenome analysis compares the genomes of species in one environment to the genomes of species in another environment to find environment-specific genes.

research community, some of which we highlighted in this paper. Note that the situation soon will become even worse (or even more challenging): First, the scope of sequencing projects will grow and grow due to the falling prices of sequencing. Second, it is expected that within the next two or three years a third generation of sequencing machines will become available [45]. Several development routes are followed; they all have in common that the speed of sequencing and the length of reads will increase drastically. The ”100 dollar genome” most likely is only a few years away.

There are also further challenges we did not discuss in this review. For instance, meta data management for thousands of genomes must be carefully designed, to not loose important data associated to a genome. Another issue is the integration of large genomic data sets with other types of information, like function or interaction of genes. A particularly hard problem is that of data privacy. Genomic data is highly personal and sensitive. What's more, anonymization or pseudonymization of sequencing data is not simply a matter of dissociating the donor's name from the data, since the data itself can potentially identify the donor. In a research context, probands of genomic studies may want to be assured that they retain some form of control over this sensitive personal data. In a clinical context, genomic data may be regarded as personal health information, making its protection imperative and even mandated by law [38]. This severely limits the use of publicly accessible cloud-based read mapping services and also puts commercial services to sequencing into question. Possible solutions include the establishment of non-public ”walled” cloud-based solutions with strict and reliable access control, or the development of cloud-based read mapping that does not require transmission of the actual read sequence to the public cloud [9].

Acknowledgements Astrid Rheinländer is funded by the Deutsche Forschungsgemeinschaft through the *Stratosphere* project. Marc Bux is funded by the Deutsche Forschungsgemeinschaft through the *SOAMED* research unit. Berit Haldemann is funded by the Bundesministerium f. Bildung und Forschung through the project *Prositu*.

References

1. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.
2. S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–3402, 1997.
3. D. Antoniou, E. Theodoridis, and A. Tsakalidis. Compressing biological sequences using self adjusting data structures. In *Information Technology and Applications in Biomedicine*, 2010.
4. R. A. Baeza-Yates and C. H. Perleberg. Fast and practical approximate string matching. In *Proceedings of the Third Annual Symposium on Combinatorial Pattern Matching*, CPM '92, pages 185–192, London, UK, UK, 1992. Springer-Verlag.
5. D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke. Nephele / PACTs : A Programming Model and Execution Framework for Web-Scale Analytical Processing Categories and Subject Descriptors. *Proceedings of the 1st ACM symposium on Cloud computing*, 2010.
6. R. K. Bharti, A. Verma, and R. Singh. A biological sequence compression based on cross chromosomal similarities using variable length lut. *International Journal of Biometrics and Bioinformatics*, 4:217–223, 2011.
7. M. C. Brandon, D. C. Wallace, and P. Baldi. Data structures and compression algorithms for genomic sequence data. *Bioinformatics*, 25(14):1731–1738, July 2009.
8. X. Chen, S. Kwong, and M. Li. A compression algorithm for DNA sequences. *Engineering in Medicine and Biology Magazine, IEEE*, 20(4):61–66, 2001.
9. Y. Chen, B. Peng, X. Wang, and H. Tang. Large-scale privacy-preserving mapping of human genomic sequences on hybrid clouds. In *Proceeding of the 19th Network & Distributed System Security Symposium*, 2012.
10. G. T. Chiang, P. Clapham, G. Qi, K. Sale, and G. Coates. Implementing a genomic data management system using irods in the wellcome trust sanger institute. *BMC Bioinformatics*, 12:361, 2011.
11. J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107, 2008.

12. E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.
13. E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, and Others. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.
14. C. Dennis and R. Gallagher, editors. *The Human Genome*. Palgrave Macmillan, 2002.
15. M. Duc Cao, T. I. Dix, L. Allison, and C. Mears. A simple statistical algorithm for biological sequence compression. In *Proceedings of the 2007 Data Compression Conference*, pages 43–52, Washington, DC, USA, 2007. IEEE Computer Society.
16. P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proc. Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 390–398, Los Alamitos, CA, USA, 2000. IEEE Computer Society.
17. I. Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Parallel programming / scientific computing. Addison-Wesley, 1995.
18. J. Goecks, A. Nekrutenko, J. Taylor, and T. Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010.
19. S. Grabowski and S. Deorowicz. Engineering relative compression of genomes. *CoRR*, abs/1103.2351, 2011.
20. S. Grumbach and F. Tahi. A new challenge for compression algorithms: genetic sequences. *Inf. Process. Manage.*, 30(6):875–886, Oct. 1994.
21. C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, and J. Good. On the Use of Cloud Computing for Scientific Workflows. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 640–645, 2008.
22. M. Holtgrewe, A.-K. Emde, D. Weese, and K. Reinert. A novel and well-defined benchmarking method for second generation read mapping. *BMC Bioinformatics*, 12:210, 2011.
23. T. J. Hudson, W. Anderson, A. Artez, A. D. Barker, C. Bell, R. R. Bernabe, M. K. Bhan, F. Calvo, I. Eerola, D. S. Gerhard, et al. International network of cancer genome projects. *Nature*, 464(7291):993–998, 2010.
24. G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling. Data Sharing Options for Scientific Workflows on Amazon EC2. *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–9, 2010.
25. W. J. Kent. BLAT – the BLAST-like alignment tool. *Genome Research*, 12(4):656–664, 2002.
26. S. Kuruppu, B. Beresford-Smith, T. Conway, and J. Zobel. Iterative dictionary construction for compression of large dna data sets. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 9(1):137–149, Jan. 2012.
27. S. Kuruppu, S. J. Puglisi, and J. Zobel. Relative lempel-ziv compression of genomes for large-scale storage and retrieval. In *Proceedings of the 17th international conference on String processing and information retrieval*, SPIRE'10, pages 201–206, Berlin, Heidelberg, 2010. Springer-Verlag.
28. B. Langmead and S. L. Salzberg. Fast gapped-read alignment with bowtie 2. *Nat Meth*, 9(4):357–359, Apr. 2012.
29. B. Langmead, M. Schatz, J. Lin, M. Pop, and S. Salzberg. Searching for snps with cloud computing. *Genome Biol*, 10(11):R134, 2009.
30. B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.
31. B. Li and S. M. Leal. Methods for Detecting Associations with Rare Variants for Common Diseases: Application to Analysis of Sequence Data. *The American Journal of Human Genetics*, 83(3):311–321, Sept. 2008.
32. H. Li and R. Durbin. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.
33. H. Li and N. Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Brief Bioinform*, 11(5):473–483, 2010.
34. Y. Li and S. Zhong. Seqmapreduce: software and web service for accelerating sequence mapping. In *Proceedings of the 9th International Conference for the Critical Assessment of Massive Data Anaysis*, CAMDA 2009, 2009.
35. Y. Liu and B. Schmidt. Long read alignment based on maximal exact match seeds. *Bioinformatics*, ECCB 2012 special issue, 2012.
36. D. W. Mount. *Bioinformatics: sequence and genome analysis*. CSHL Press, 2004.
37. T. Nguyen, W. Shi, and D. Ruden. CloudAligner: A fast and full-featured MapReduce based tool for sequence mapping. *BMC Research Notes*, 4(1):171+, 2011.
38. U. D. of Health and H. Services. Ocr privacy brief: Summary of the hipaa privacy rule. *HIPAA Compliance Assistance*, 2003.
39. T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–54, 2004.
40. C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.
41. E. Pennisi. Will computers crash genomics? *Science*, 331(6018):666–668, 2011.
42. E. Rivals, L. Salmela, P. Kiiskinen, P. Kalsi, and J. Tarhio. mpscan: fast localisation of multiple reads in genomes. In *Proc. 9th International Workshop on Algorithms in Bioinformatics (WABI)*, volume 5724 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2009.
43. F. Sanger, S. Nicklen, and A. R. Coulson. Dna sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences of the United States of America*, 74(12):5463–5467, 1977.
44. E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan. Computational solutions to large-scale data management and analysis. *Nat Rev Genet*, 11(9):647–657, 2010.
45. E. E. Schadt, S. Turner, and A. Kasarskis. A window into third-generation sequencing. *Human molecular genetics*, 19(R2):R227–R240, Oct. 2010.
46. M. C. Schatz. Cloudburst. *Bioinformatics*, 25(11):1363–1369, June 2009.
47. A. D. Smith, W.-Y. Chung, E. Hodges, J. Kendall, G. Hannon, J. Hicks, Z. Xuan, and M. Q. Zhang. Updates to the RMAP short-read mapping software. *Bioinformatics*, 25(21):2841–2842, Nov. 2009.
48. A. D. Smith, Z. Xuan, and M. Q. Zhang. Using quality scores and longer reads improves accuracy of solexa read mapping. *BMC Bioinformatics*, 9, 2008.
49. L. D. Stein. The case for cloud computing in genome informatics. *Genome Biol*, 11(5):207, 2010.
50. C. Trapnell and S. L. Salzberg. How to map billions of short reads onto genomes. *Nature biotechnology*, 27(5):455–457, 2009.
51. N. Välimäki, W. Gerlach, K. Dixit, and V. Mäkinen. Compressed suffix tree—a basis for genome-scale sequence analysis. *Bioinformatics*, 23(5):629–630, Feb. 2007.
52. G. Vey. Differential direct coding: a compression algorithm for nucleotide sequence data. *The Journal of Biological Database and Curation*, 2009, 2009.

53. D. Warneke and O. Kao. Nephele : Efficient Parallel Data Processing in the Cloud Categories and Subject Descriptors. *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, 2009.
54. D. Weese, A. Emde, T. Rausch, A. Döring, and K. Reinert. RazerS – fast read mapping with sensitivity control. *Genome Research*, 19(9):1646–1654, 2009.
55. T. White. *Hadoop: The definitive guide*. Yahoo Press, 2010.
56. M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving MapReduce Performance in Heterogeneous Environments. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, pages 29–42, 2008.