KLAST: fast and sensitive software to compare large genomic databanks on cloud

I. Petrov[1], S. Brillet[1], E. Drezen[1], S. Quiniou[2], L. Antin[2], P. Durand[2] and D. Lavenier[1]
[1]INRIA/IRISA/GenScale, Campus de Beaulieu, 35042 Rennes cedex, France
[2]Korilog, 4, rue Gustave Eiffel, 56230 Questembert, France

Abstract - *As the genomic data generated by high throughput sequencing machines continue to exponentially grow, the need for very efficient bioinformatics tools to extract relevant knowledge from this mass of data doesn't weaken. Comparing sequences is still a major task in this discovering process, but tends to be more and more time-consuming. KLAST is a sequence comparison software optimized to compare two nucleotides or proteins data sets, typically a set of query sequences and a reference bank. Performances of KLAST are obtained by a new indexing scheme, an optimized seed-extend methodology, and a multi-level parallelism implementation. To scale up to NGS data processing, a Hadoop version has been designed. Experiments demonstrate a good scalability and a large speed-up over BLAST, the reference software of the domain. In addition, computation can be optionally performed on compressed data without any loss in performances.*

Keywords: Cloud computing – bioinformatics – Intensive sequence comparison – High Throughput Sequencing

1 Introduction

The extraordinary progresses of sequencing technologies currently lead to generate very huge amount of data. Processing these data are performed by specializing bioinformatics pipelines that depend of the biological question to answer. Such pipelines include many specific bioinformatics tools but also – most of the time - intensive sequence comparison steps to extract similarities between raw sequencing data and fully annotated DNA or protein databanks. Comparing large sequence datasets to genomic banks (DNA or protein) can thus be extremely time-consuming, especially if a minimum of sensitivity is required.

Speeding up the comparison process can be done using different directions. A very efficient heuristic is the seed-extend approach that first detects similar *seeds* (i.e. short words) between query and genomic databank, and then performs left and right extensions to generate relevant alignments. BLAST family software [1], RAPSEARCH [14] or DIAMOND [2] are based on this concept. Another possibility is to count the number of identical short words in common between two sequences and to decide whether it's worth to continue the search according to a threshold value. USEARCH [5] follows this interesting strategy.

The exploitation of internal parallelism of modern processors is also a convenient way to gain performances: processors now include several cores that can be simultaneously activated through multithreading. They also integrate powerful instruction sets allowing basic comparison operations to be simply vectorized. SSEARCH [6] and SWIPE [11] makes an intensive use of this technique to produce a very efficient fine-grained parallelization of the Smith and Waterman algorithm based on dynamic programming [13]. Algorithm optimization and parallelization techniques can of course be mixed together to improve efficiency.

KLAST has been primarily designed to compare a query set of sequences against a DNA or protein databank. It follows the BLAST strategy by proposing all the possible query/database combinations: DNA/DNA, DNA/protein, protein/DNA, protein/protein, DNA-translated/DNA-translated. Performances of KLAST are obtained by a new indexing scheme, an optimized seed-extend methodology, and a multi-level parallelism implementation (multithreading and vectorization). Tuning the seed-extend heuristic of KLAST allows the users to precisely define the sensitivity/speed tradeoff.

Compared to BLAST (the gold standard of search alignment tools), and considering equivalent sensitivity, speed-up ranges from 5 to 10 according to the nature of the datasets, and to the amount of data to process. For very large problems, this speed-up is far from negligible. It can save million hours of computation and significantly reduce the number of nodes in a cloud infrastructure.

The KLAST cloud implementation highlights this aspect. Similar to existing BLAST cloud solutions [8][10] our implementation provides efficient scalability thanks to the nature of the sequence comparison problem that is an embarrassingly parallel problem. Distributing the computation on a cluster, especially for this specific problem, is thus straightforward and doesn't present any theoretical difficulties. On the other hand, scalability can be limited by data accesses, I/O transfers or sequential sections of the algorithms. Cloud implementation must consequently be

highly optimized on these points to maintain a good scalability.

The KLAST cloud implementation also addresses the problem of managing large data files. The genomic banks represent hundred of Giga bytes of data. They have to be stored near the hardware computing resources to make them available when a job is run. They also have to be efficiently routed through the cloud network to the computing nodes. The first point requires consequent space storage. The second point can be critical for I/O bound problems. Working on compressed data, as proposed in the KLAST cloud implementation, is a way to release the pressure when tackling big data domains.

The next section briefly describes the KLAST methodology and provides a performance overview compared to other software. Section 3 is dedicated to the cloud implementation of KLAST with Hadoop. Section 4 concludes the paper.

2 KLAST SOFTWARE

KLAST is issued from PLAST, a sequence comparison software previously developed for protein alignments only [9]. It has been extended to DNA comparison by adapting the ORIS algorithm [7] to the double indexing seed scheme. Furthermore, for nucleotides search, it includes a new filtering step strategy that makes KLAST very competitive to process NGS data.

The global KLAST strategy, for protein or DNA sequences, is based on the following steps:

1. *Seed indexing*: In this step, the query and the subject bank are both indexed. For performance purpose, the two indexes completely fit into the computer memory. However, for very large banks, a bank splitting is automatically performed.
2. *Ungap alignment search*: This step acts as a filter to limit the search space. The idea behind this filtering step is that if no similarity is found in the immediate neighborhood of the seed, then the probability to find a significant alignment in this region is low, and won't necessitate further processing.
3. *Gap alignment search*: The ungap alignments calculated on the previous step are extended to include gap errors. This is done by dynamic programming technique over a restricted search area.
4. *Alignment sorting*: The multi-threaded implementation of the steps 2 & 3 generate alignments in a random order. They need to be reordered before to be displayed.

Depending of the nature of the data (protein or DNA), the methodologies used in step 1 and 2 are different. The two next subsections explain the way indexing and ungap search are implemented according to the data type.

2.1 Seed indexing

Protein sequences: A subset seed model is used [12]. Such seeds are more convenient than standard seeds for indexing purpose [9]. The protein index is an array that stores all the positions of the seeds in the banks.

DNA sequences: A conventional seed model is used (words of N consecutive nucleotides). By default, a seed size of 11 is used. Unlike the protein index that only store the seed positions, the DNA index also memorizes neighborhood information: 3-mers that are present in the left and right neighborhood of 20 nucleotides are tagged into two 64-bit set vectors.

2.2 Ungap alignment search

As both banks are indexed in the same way, the ungap alignment search step consists in performing a loop over all possible seeds. For a given seed, the indexes of the two banks provide two lists of positions from which an all-vs-all computation is done. More precisely, each element of one list is compared to all elements of the other one. In this step, gap errors are not allowed.

Protein sequences: A search over a fixed size area near the seeds is performed. This predefined search region allows computation to be parallelized with SIMD instructions. 16 8-bit alignment scores are simultaneously computed.

DNA sequences: This step is split into two tasks. First, logical operations between the two 64-bit vectors embedded in the index are performed to determine the number of identical 3-mers. If it overcomes a threshold value, then an ungap search is launched using the ORIS algorithm. This algorithm computes a score by a left and right extension starting from the seed, but returns only an alignment if this extension doesn't use words smaller than the seed. See [7] for a complete description of the algorithm.

2.3 Parallelism

The KLAST algorithm exhibits three levels of parallelism. The first one is linked to the capacity of KLAST to split the banks into chunks of data that are fully indexed into the computer memory. These features have two main advantages:

1. Huge computer memory configurations are not required to process large genomic banks. The computation is performed sequentially on each chunk of data.

2. Pieces of banks can be dispatched and independently processed over a grid or a cluster infrastructure. The merging step is cheap and doesn't penalize the overall performances.

The second level of parallelism comes from the double indexing approach. A large number of seeds can be analyzed

simultaneously using the multithreading possibilities of today multi-core processors. The programming model is a producer /consumer model. One thread manages the overall computation and request many threads to compute ungap and gap alignments.

The last level of parallelism is the use of the SIMD paradigm for very "regular" computation. For protein sequences it is intensively used in the ungap and gap alignment search. The ungap step parallelizes 16 computations of scores simultaneously. The gap search step, which is more complex, run only 8 score computations in parallel.

2.4 Performances

This section provides a brief overview of KLAST performances, both in terms of quality and speed-up compared to other similar software. In the context of NGS data, we have compared a subset of a RNA-seq dataset (50K Illumina sequences) from a microbiome project with a Human proteome databank (71,338 proteins). Experiments have been conducted on a 2.67 GHz Intel Xeon E5640, 8 cores, 48 GBytes of RAM, Debian 4.6.3-13 Linux version. The following software, which handle the comparison of DNA sequence with protein sequences, are considered: BLAST, UBLAST, DIAMOND and KLAST.

Table 1 reports the results. The Align column represents the number of alignments found by the software. Two alignments are considered as identical if the two sequences overlap at 80% [4]. The Hit column is the number of query-subject pairs reported without any control on the alignment boundaries. It just tells that a specific DNA sequence has a significant match with a specific protein. This type of information can be sufficient to answer many biological questions.

	Align	Hit	Exec. Time
BLAST	2934080	465376	2539 sec.
UBLAST	308826	221551	65 sec.
DIAMOND	968886	402211	44 sec.
KLAST	2902727	463934	283 sec.

Table 1: Nucleic/Protein search (e-value = 10-3)

From a quality point of view, BLAST and KLAST generate similar results. Differences come from the search methodologies that are not exactly identical. But both rely on heuristics and are statistically equivalent [9]. KLAST is however much faster (speed up = 9). UBLAST and DIAMOND are very fast but only 10% and 35% of the alignments are reported, respectively.

If less sensitivity is required, KLAST can be tuned to reach DIAMOND sensitivity. The tuning is performed by considering only a subset of seeds, from 100% to 1%. In that case, the execution time of KLAST significantly decreases. DIAMOND is generally 50% to 100% faster but requires computers with a large memory. As a matter of fact, DIAMOND speed comes from a sophisticated – but costly – indexing scheme that absolutely needs to fit into memory

3 KLAST on CLOUD

3.1 Parallelization Strategy

The problem of finding alignments between two sets of sequences is embarrassingly parallel. If N and M are respectively the size (in terms of number of sequences) of the query set and the reference bank, then we have to solve NxM independent problems. N and M can be very large (a few tens of millions), leading to 10^{14} to 10^{15} elementary tasks that could be ultimately processed in parallel. As an example, a NGS data set may represents 10^8 sequences of length 100 (10 GB) and the non-redundant uniprot protein bank contains 92 x 10^6 proteins (~35 GB).

The strategy, here, is to process independently chunks of sequences, and to run KLAST on these data. The query set and the reference bank are thus split into packets of a few mega bytes. Each run of KLAST generates a list of sorted alignments, which are pushed to the storage system.

The final step is to reorder the lists of alignments. Actually, depending of the downstream processing, this task may not be essential. Hence, it is optionally done when reading back the results.

3.2 Hadoop Implementation

The task of processing a large number of independent chunks of data can be solved efficiently using a Map-Reduce approach [3]. The most famous open source implementation of this approach is Hadoop (http://hadoop.apache.org/). The Hadoop strategy allows very good job scheduling on a large number of nodes to be done very efficiently in regard of many aspects: physical location of the data, good utilization of the hardware resources, High Availability (HA), dynamic modification of available nodes, etc.

Here, we have to process NxM chunks of data, each chunk being a set of sequences that do not have exactly the same size. Furthermore, for a specific job, Hadoop manages only the split of a single file among all the cluster nodes. Thus, our strategy is the following:

- The subject bank is managed by the Hadoop splitting mechanism. N is the number of chunks.
- The query bank is split independently into M sub query banks, and M Hadoop jobs are launched.

As Hadoop splits data into chunks of identical size, an additional step is required when the data is being read. It

consists of a special treatment of the first and the last sequence in each chunk, so that they are exactly read once, even when a part of any of them is placed in one split and another part of it is placed in the next chunk. One solution, for a chunk, is to skip the first partial sequence and ensure that it is read as part of the previous chunk. In that way, all sequences are considered. Hadoop provides internal features to deal with this kind of data adjustment.

As previously mentioned, a Hadoop job handles the splitting of only one single file. For that reason, the query bank is cut into sub banks of nearly identical size. It is then possible to force Hadoop to place specific files on each node, and to run a Hadoop job for each query split. In this way, both the subject and the query are split and all the pairs of query and subject chunks are processed independently. In order to improve further the nodes utilization, two jobs are started at the same time on each node. In that way, the I/O and scheduling times for one comparison are used for calculations of the others.

Klast computes an e-value for each alignment. This e-value depends of the size of the subject bank. The e-value computation based on a reduced section of the subject bank will lead to incorrect statistical information. To avoid such a situation, and to output identical results compared to a sequential execution, the real size of the subject bank is considered through the use of a specific parameter of KLAST that is set automatically by our implementation.

The actual computation is performed in the Hadoop mappers. They execute a KLAST command with the appropriate arguments and save the output on the Hadoop distributed file system. If special merging of the results is needed, it is performed in a single Reducer task.

Having the databases in raw fasta format may take a considerable amount of space storage and may also take time during data transfers. To make these aspects less constraining as possible, data can be uploaded in a compressed format. When this option is selected, bz2 is currently used for storing the data. Fortunately, this is a splitable format that can benefit of the parallel cloud environment. We are looking into adding other splitable archive formats, as bz2 is computationally expensive in decompress mode. As Hadoop can support a number of archive formats this is not expected to present any difficulty.

3.3 Experimentation

Tests have been performed on the IFB (French Institute of Bioinformatics) cloud. The virtual CPU cores are simulated by qemu and report a working frequency of 2.6 GHz. 2 or 4 core nodes configuration have 8 GB RAM, whereas 8 core configuration have 16 GB RAM. The OS is CentOs 6.5.

We first tested the scaling of KLAST by comparing the full yeast proteome (6640 proteins, 3.2 x 10^6 amino acids) with the NCBI nr protein database (release 67, ~61.3 x 10^6 sequences, 18.7 x10^9 amino acids).

Different nodes/CPU configurations have been tried. Figure 1 plots the speed-up of KLAST in the 2 CPU/node configurations ranging from 1 to 40 nodes. The 4 or 8 CPU/node configurations provide similar performances (same scaling and same execution time: ~1h30).

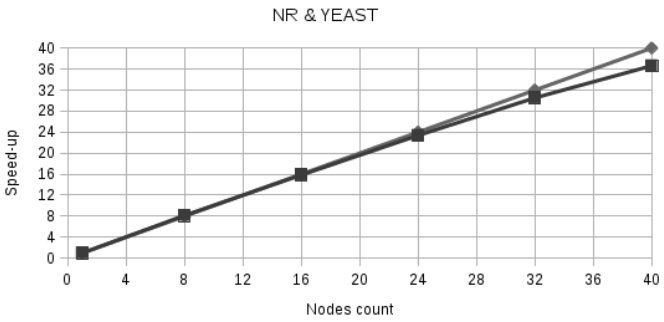

Figure 1: KLAST speed-up on the IFB cloud with 2 CPU per node. The Yeast proteome (6640 proteins is compared with the NCBI nr protein databank (61.3 millions of proteins)

As it can be seen, the speed-up is nearly linear, allowing an execution time of 60 hours on one 2 CPU node to be lower down to 1h30 with 40 nodes. Further experimentations with a higher number of nodes would be necessary to better analyze the scalability of the KLAST Hadoop implementation.

We also compare the parallel BLAST and KLAST Hadoop implementation. Instead of running KLAST, BLAST is simply launched with the same splitting parameters. The following table reports the execution times. On average, KLAST is about 7 times faster and the speed-up tends to increase with the number of nodes.

#nodes	10	20	40
BLAST	35h35	19h40	10h58
KLAST	5h08	2h45	01h29
Speed-up	6.9	7.2	7.4

On this specific experiment, if the data compression mode is activated, the overall processing time remains the same. Actually, the time for transferring the data is balanced out by the time for decompressing the data. The great advantage is that a lot of space is saved without any loss in performances. The size of ncbi nr bank is 36.3 GB. Its compressed version with bz2 is 13.2 GB. Hence, on that example, 23 GB is saved.

4 Conclusions

KLAST is a bank-to-bank comparison software designed and optimized to process large genomic and metagenomic data sets. Like BLAST, KLAST is based on the *seed-and-extend* heuristic strategy, but includes various improvements that favour NGS data processing. It also exploits all parallel resources of modern computers (multi-cores and aggressive use of SSE instruction set) that allow execution time to be significantly reduced.

A KLAST Hadoop version has been designed to tackle large sequence comparison problems, i.e. problems requiring millions hours of CPU time. The current implementation, over concurrent approaches, mainly bring two advantages:

- *Significant speed up* compared to the gold standard of the domain (BLAST) and a good scalability. An average speed up of 7 if often measured if equivalent sensibility to BLAST is wanted. If a loss of sensibility is accepted, then much high speed up is achieved (X50). For huge instances of genomic sequence comparison problems, and from an economic point of view (in a cloud context), the efficiency of KLAST may save a lot of time and money.

- *Efficient storage* through the direct use of compressed data. Cloud services dedicated to the bank-to-bank comparison have to propose a large panel of public banks. Having the banks in a compressed format save both storage space and communication bandwidth within a cluster.

Short-term perspectives are to increase scalability measurements. We are currently limited by the IFB cloud infrastructure that is still in its starting phase, but that should progressively grow to 10,000 cores in the next 2 years.

Longer-term perspectives are to split the reference banks in a much clever way. The current splitting generates raw chunks of data independently of any knowledge that could be extracted from the genomic banks. Actually, banks content a lot of redundancy coming from the orthologous nature of the sequences. On the other hand they contain a huge diversity that makes the search computationally intensive: a query often matches with a very tiny fraction of sequences of the reference bank, meaning that exploring the entire search space could be avoided.

Similarly, in the case of a NGS query bank, a pre-processing step to exploit the coverage redundancy would be an efficient way to reduce the complexity of the problem. A solution is to assemble the NGS data into contigs and then to perform a comparison between these contigs and the reference bank. In that way, the number of query sequences could be significantly reduced, and similar computation avoided.

KLAST AVAILABILITY

KLAST is jointly developed by the INRIA/IRISA GenScale research team and the KORILOG Company through a common research lab (KoriScale Lab). A free academic version is available and can be downloaded at: http://koriscale.inria.fr/klast-download

ACKNOWLEDGMENT

The authors would like to thank the IFB (Bioinformatics French Institute) for the availability of computing resources. *Funding*: This work is supported by the ANR (French National Research Agency), ANR-14-CE23-0001-01 and by the Brittany Region (KoriPlast2 project).

5 References

[1] Altschul S.F. et al. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res., 25, 3389–3402.

[2] Benjamin Buchfink, Chao Xie & Daniel H. Huson, Fast and Sensitive Protein Alignment using DIAMOND, Nature Methods, 12, 59–60 (2015) doi:10.1038/nmeth.3176

[3] Dean J., Ghemawat S., (2005) MapReduce: Simplified Data Processing on Large Clusters, OSDI'04

[4] Drezen E, Lavenier D., (2014) Quality metrics for benchmarking sequences comparison tools, Advances in Bioinformatics and Computational Biology, LNCS 8868, pp.144-153

[5] Edgar R.C. (2010) Search and clustering orders of magnitude faster than BLAST, Bioinformatics 26(19), 2460-2461.

[6] Farrar M., (2007) Striped Smith–Waterman speeds database searches six times over other SIMD implementations, Bioinformatics (2007) 23 (2): 156-161.

[7] Lavenier D., (2008) Ordered Index Seed Algorithm for Intensive DNA Sequence Comparison, IEEE International Workshop on High Performance Computational Biology, Miami, Florida, USA

[8] Matsunaga A., Tsugawa M., Fortes J., (2008) Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications eScience, 2008. eScience'08. IEEE Fourth International Conference

[9] Nguyen V.H., Lavenier D., (2009) PLAST: parallel local alignment search tool for database comparison, BMC Bioinformatics, vol 10, no 329

[10] O'Driscoll A. et al., (2015) HBLAST: Parallelised sequence similarity – A Hadoop MapReducable basic local alignment search tool, Journal of Biomedical Informatics, Volume 54, April 2015, Pages 58–64

[11] Rognes T., (2011) Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation, BMC Bioinformatics 12, 221

[12] Roytberg M, Gambin A, Noe L, Lasota S, Furletova E, Szczurek E, Kucherov G., (2009) On subset seeds for protein alignment, EEE/ACM Trans Comput Biol Bioinformatics 2009, 6(3): 483-494.

[13] Smith T.F., Waterman, M. S. (1981). Identification of Common Molecular Subsequences, Journal of Molecular Biology 147: 195–197

[14] Zhao Y., Tang H. and Ye Y., (2012) RAPsearch2 : A fast and memory efficient protein similarity search tool for next generation sequencing data, Bioinformatics (2012) 28 (1): 125-126

[15] Sul, S-J., Tovchigrechko A., (2011) Parallelizing BLAST and SOM algorithms with MapReduce-MPI library, 2011 IEEE International Parallel & Distributed Processing Symposium