Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015)
Krakow, Poland

Jesus Carretero, Javier Garcia Blas
Roman Wyrzykowski, Emmanuel Jeannot.
(Editors)

September 10-11, 2015

Multilevel parallelism in sequence alignment using a streaming approach

Beat Wolf[1,2], Pierre Kuonen[1], Thomas Dandekar[2]

[1] University of Applied Sciences Western Switzerland, HES-SO//Fribourg
[2]University of Würzburg//Germany
beat.wolf@hefr.ch

Abstract

Ultrascale computing and bioinformatics are two rapidly growing fields with a big impact right now and even more so in the future. The introduction of next generation sequencing pushes current bioinformatics tools and workflows to their limits in terms of performance. This forces the tools to become increasingly performant to keep up with the growing speed at which sequencing data is created. Ultrascale computing can greatly benefit bioinformatics in the challenges it faces today, especially in terms of scalability, data management and reliability. But before this is possible, the algorithms and software used in the field of bioinformatics need to be prepared to be used in a heterogeneous distributed environment. For this paper we choose to look at sequence alignment, which has been an active topic of research to speed up next generation sequence analysis, as it is ideally suited for parallel processing. We present a multilevel stream based parallel architecture to transparently distribute sequence alignment over multiple cores of the same machine, multiple machines and cloud resources. The same concepts are used to achieve multithreaded and distributed parallelism, making the architecture simple to extend and adapt to new situations. A prototype of the architecture has been implemented using an existing commercial sequence aligner. We demonstrate the flexibility of the implementation by running it on different configurations, combining local and cloud computing resources.

Keywords Ultrascale systems, NESUS, Template

I. Introduction

In the field of bioinformatics the advance of next generation sequencing (NGS) technologies increased the amount of data produced at a very high speed. They produce the sequencing data at a higher speed, with longer sequences that have increasingly better quality. The amount and speed at which the data is produced increased much faster than the capacities of computers evolved during the same time. This makes it challenging for sequence laboratories to analyse the produced data in a reasonable amount of time. While in the beginning of DNA sequencing the produced data could still be analysed by hand, the amounts of data produced today for one sample can range from 10 giga base pairs in whole exome sequencing to hundreds in the case of whole genome sequencing. Powerful computing infrastructures are needed to analyse this type of data.

Ultrascale computing presents itself as a possible solution to many of the issues faced in bioinformatics. But to benefit from the possibilities of ultrascale computing, many algorithms and tools used need to be adapted to work in such a distributed environment. While much effort is spent to distribute and parallelize various tools, they are often tied to a specific environment, be it a grid or a cloud. In this paper we look at the issue of sequence alignment and create a generic way to distribute the workload over a heterogeneous distributed system. This work can be used as the basis of future work to bring bioinformatics data processing to ultrascale systems.

Sequence alignment is an active topic of research, with various approaches taken to solve the issue. The basic idea of sequence alignment is to find the best position of a sequenced read on an already known reference genome. The complexity of the task comes from the size of the reference genome, the amount of sequences to be aligned and the fact that the sequenced genome does not perfectly match the reference genome. As an example, the he human reference genome is 3 Giga base pairs long. Millions of sequences with a typical length between 100 and 1000 base pairs are then aligned against this reference. The differences between those sequences and the reference can range from simple single nucleotide mismatches to complicated insertions or deletions.

Various approaches exist to the problem of sequence alignment, an overview of the different techniques can be found at [1]. While the different alignment tools approach the issue from different angles, they do all have in common that they find the optimal placement of a single sequence on the reference genome. They do so by looking only at one sequence at a time, without any expectations of the order at which the sequences are aligned. This makes sequence alignment an ideal candidate for parallel processing. Every sequenced read can be processed independently and in any order. The order in which they are input for the alignment and their output order also does not matter. In fact most alignment algorithms make use of multi-core systems through multithreading to speed up the alignment. Some, like the recently released tool nvBowtie, even use GPUs to make use of the thousands of cores of a GPU to accelerate sequence alignment. The use of parallel processing does not stop here, but also extends to distributed systems. Examples of such tools are Crossbow [3], Cloudburst [4], ScalaBLAST [5] or custom frameworks like the one presented in [6].

In this paper we explore a possible architecture which uses parallelism on multiple levels. It is used to distribute the sequence alignment over multiple cores, multiple computers in the same network and cloud resources. While several of the previously mentioned tools support some of those approaches, none combine all of them at once. A stream based approach is used to distribute the workload over all the different systems. The way a remote computer and a local thread are integrated into the alignment process is very similar in this architecture. A prototype of the system has been implemented in a commercial NGS data analysis software, GensearchNGS [9]. The proposed architecture is verified by measuring its scalability over multiple heterogeneous computing nodes and the flexibility of the design is discussed. While the sequence alignment algorithm distributed in this example is a custom one used by GensearchNGS, the architecture and design of the distribution is not restricted to that algorithm.

II. Methods

The implementation of the distributed sequence aligner uses Java 6+ as the programming language. The communication with remote installations is done through DIRMI, a replacement for the by default in Java integrated remote method invocation library RMI. DIRMI was chosen over RMI as it allows for bidirectional connections to remote objects, making it possible to work behind a NAT while using remote computing resources.

The developed sequence aligner can be run locally using the multiple available cores of a machine and distributed over multiple computers with the software installed and running. For this prototype the installed software is GensearchNGS. Additionally, the user can also dynamically launch remote cloud instances of the aligner. The integration of cloud resources, for the moment using the Amazon AWS EC2 cloud service, is done through the generic Java clouds library jclouds. It allows for easy integration of cloud instances on various clouds, making it possible to launch and destroy cloud computing instances from inside Java. The aligner which is distributed is based on the aligner used in GensearchNGS [7]. The aligner uses a hash based index for the initial seed detection. The seeds are evaluated used a custom heuristic algorithm with the final alignment being performed by the Gotoh [2] algorithm, ideally suited for gapped alignment. While for this implementation of the distributed alignment we use this specific algorithm, the architecture could also be applied to other existing alignment algorithms.

The dataset used to test the prototype comes from the genome comparison & analytic testing project (GCAT) [8], which provides standardized

datasets to test different NGS data analysis software. The illumina-100bp-pe-exome-150x dataset has been used, which is available on the GCAT website (http://www.bioplanet.com/gcat/). The dataset contains 45'038'905 read pairs with an average read length of 100 bp. For the benchmarks in this paper, only the first 5 million read pairs have been used.

III. Architecture

The architectural choices are based on a typical scenario. Small genetic research laboratories often do not have the required infrastructure to perform big sequence alignments. But what they do have is a modest computing infrastructure with multiple multi-core desktop systems used for data analysis. A common evolution for those laboratories is to start NGS data analysis using targeted sequencing for a specific set of genes. Later they expand to whole exome analysis and in the end doing whole genome sequencing. While targeted sequencing can be done using a modest desktop computer, moving to whole exome and especially whole genome sequencing data quickly brings laboratories to their limits. Thus the goal of the proposed architecture is to optimally use the existing resources but also provide the option to expand the infrastructure when needed to outside resources. Possible options to expand to outside resource are, depending on the laboratory policies, cloud computing services like Amazon.

The core concept of the architecture is based on the idea of using stream processing for sequence alignment. As shown by [6], using stream processing for DNA sequence alignment can greatly increase the total processing time. This comes mainly from that fact, that uploading the data to a remote computing resource and analysing it can be done at the same time, drastically reducing the overall analysis time.

The concept of stream processing was not only used to integrate the remote computing resources, but to connect all parts of the system together, including local alignment threads. The core system, as shown in figure 1, is composed of 3 main elements: The `Data reader`, the `Sequence aligner` and the alignment `Writer`. For this implementation we choose to use `BlockingQueues` to communicate between the different elements. It

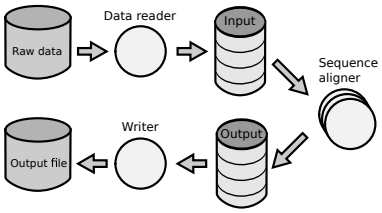

Figure 1: Overall stream based aligner architecture, connecting the different system elements through queues

is this choice of using Queues for the communication between the different parts that made the architecture very flexible. The `Data reader` reads the input data provided by the sequencer, usually in the FASTQ file format, and puts the individual reads on the Queue to be aligned. The different local threads implementing the `Sequence aligner` take those sequences from the queue and align them against the reference sequence. After having aligned a sequence, the `Sequence aligner` puts the created alignment in a different queue, the one containing the finished alignments. The alignment `Writer` takes the aligned sequences from the output queue and writes them into an alignment file, typically a SAM or BAM file. All the different elements of the system are run in their own thread.

Using queues to communicate between the different parts of the aligner makes it extremely flexible and scalable. This is shown in figure 2 which expands the basic local parallelization to work in a distributed environment. The addition of `Sequence aligner` and `Distributed client` allow to easily distribute the work to a remote machine. In fact, `Sequence aligner` and `Distributed client` can run simultaneously on the same machine, performing some alignment work locally while distributing other. It is even possible to have a `Distributed server` which in turn distributes the alignment workload further with its own `Distributed client`. This flexibility allows for interesting configurations which adapt to various real life situations discussed later.

To be able to connect to a remote machine, all that is required is that the remote machine is running the `Distributed server` object, accessible by the client machine. In the context of this prototype every user

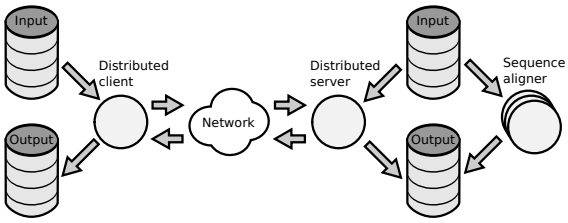

Figure 2: Integration of remote computing resources

of the GensearchNGS application has the option to let other users in the same network connect to his installation and use it to perform sequence alignment. The different installations discover each other using a custom automatic service discovery protocol. When starting the sequence alignment, the different local sequence alignment threads are created, and the remote `Distributed server` objects are automatically connected.

This concept has been expanded to include cloud computing. This has been done through the integration of the JClouds library, which allows to initialize cloud resources on demand. When starting the sequence alignment, the user is provided with the choice of how many cloud instances of the sequence aligner he wants to execute. Depending on his choice, a certain number of cloud resources are started, running a preconfigured image with the software preinstalled and executed upon booting. Once connected to the `Distributed server` object, there is no cloud specific code to perform the alignment. The system only sees an object that takes raw sequences from the input queue and puts aligned sequences back on the output queue.

The previously described possibility to create `Distributed server` objects which in turn distribute the workload further using `Distributed client` allows to handle particular limitations certain cloud providers have. In certain cloud environments it is desirable to only start a single cloud instance with a public IP address. Any additional cloud resource is then instantiated in a private network inside the cloud, only accessible from the outside through the public instance. Instead or additionally to a local sequence aligner in the `Distributed server` object, the `Distributed server` can run multiple `Distributed client` objects. This allows the public cloud instance to route the incoming raw sequencing stream from the outside to the multiple instances in the private network.

Another interesting side effect of this architecture is that it allows to perform sequence alignment on machines which are normally not powerful enough to handle the alignment. Aligning against the human reference sequence using the GensearchNGS aligner requires approximatively 5 GB of RAM. Especially older desktop systems do not always have that amount of RAM. The discussed architecture makes it possible to launch the sequence alignment locally on those machines, but without creating local `Sequence aligner` objects.

IV. Results & Discussion

To demonstrate the flexibility and performance of our architecture, we tested the prototype using multiple configurations. To do this, four typical configurations have been tested. The first one tested the performance using a single laptop. The second configuration added a second desktop computer located in the same network to speed up the calculations The third configuration expanded upon the second one by adding one instance of an Amazon AWS EC2 virtual machine. The fourth configuration uses no local alignment on the laptop which starts the alignment process, but offloads all of the alignment to two instances in the Amazon AWS EC2 cloud. In this configuration, the laptop only does the work of reading the raw data and saving the aligned sequences in the output file. The configurations of the tests have been chosen to represent a typical scenario in a small genetics laboratory. Figure 3 shows how the different computers used for the four configurations are connected.

The work laptop is equipped with an Intel Core i7-3520M dual core CPU, clocked at 3.6 Ghz with 8 GB of RAM. The desktop computer uses an Intel Core I7 870 with 4 cores clocked at 3.6 Ghz, also with 8GB of RAM. The Amazon AWS EC2 cloud instances of the aligner are launched on a virtual server of type c3.xlarge, which has a CPU of type Intel Xeon E5-2680 with 4 cores and 7.5 GB of RAM. The laptop and the desktop computer are connected with a 1 Gb/s switch, and both of them are connected to the internet with a

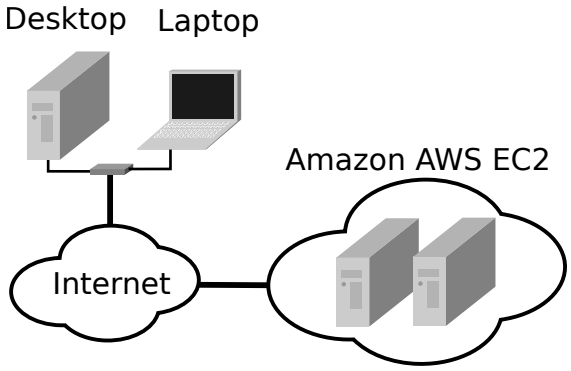

Figure 3: Topology of all elements used in the benchmark configurations.

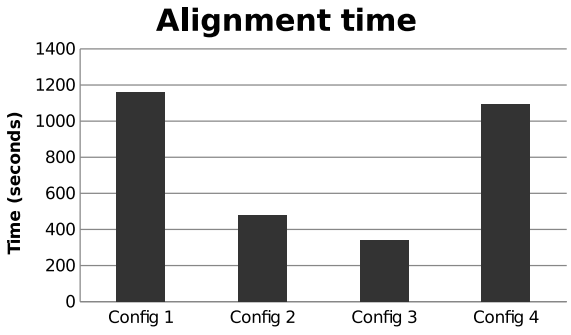

Figure 4: Alignment times on all 4 example configurations

100Mbs (up and down) connection.

Figure 4 shows the benchmark times for the distributed alignment on the different configurations. The measured times contain only the time required for the sequence alignment, not including initialization times which is about 3 minutes. Those initialization times contain the time required to load the reference sequence into memory as well as starting the cloud instances.

The raw times for the different configurations settings were: 1'163 seconds for configuration one, using only the laptop. 478 seconds for configuration 2, which used the laptop and the desktop computer. 339 seconds for the configuration 3, using the laptop, the desktop computer as well as one cloud instance. Configuration 4 finished in 1'096 seconds, using the laptop as a base station but doing all alignment work on two cloud instances.

Looking at the benchmark results we can conclude that the proposed architecture adapts well to all tested configurations. In the various configurations the different systems were well saturated, without slower systems badly affecting the overall performance. Load balancing is indeed a side-effect of the chosen design where every computing resource, be it a local thread or a remote computer, takes sequences to align out of a common queue. As long as I/O speeds permit, every computing resource is provided with the amount of work needed to saturate it.

The way remote computing resources are handled, be it in the local network or in the cloud, also permits for any amount of computers to fail. As long as at the computer launching the alignment and at least one computing resource which performs the alignment keep running, the alignment will successfully finish. This is achieved by keeping a local copy of every sequence sent to a remote computer. This local copy is only deleted once a successful alignment has been received for the sequence. If the remote computer disconnects, all the local copies of the sequences which have been sent to him but for which no alignment has been found yet, are put back in the input queue. They are then recovered and aligned by another computing resource which is still running.

V. Future work

While the developed prototype nicely shows the ability of the architecture to adapt to various situations and distribute the workload over all systems in the example configurations, there are still issues to be addressed. The first issue being the bandwidth required to distribute the workload over multiple computers, especially if they are located in a remote cloud. While the amount of data sent to the remote resource has already been minimized as much as possible, including efficient encoding of DNA sequences, not every possible optimization has yet been done to reduce the bandwidth requirements. A home internet connection will quickly saturate, putting an upper limit to the cloud instances that can be used simultaneously. While the internet connection of a genetic laboratory is usually higher than a standard home internet connection, the exact bandwidth requirements and limitation still need

to be evaluated and optimized. Once this step is done, the architecture will be compared to other distributed aligners like the previously mentioned ones [3, 4, 5, 6].

The second issue is the one of data security and confidentiality in distributed systems. In many data laboratories the data privacy rules restrict or forbid the usage of remote resources for any patient data. This is currently an active topic of research and no particular security measures were taken to improve the data security in this prototype. The two main features related to datasecurity and are planned to be implemented are: encrypting all data sent to and from the cloud and pooling multiple samples to be aligned simultaneously.

VI. Conclusion

We presented a generic architecture for stream based multilevel parallel alignment. The architecture uses the same concepts to distribute the alignment over multiple cores on one system and over multiple computers. The implemented prototype is able to adapt to various real life situations, using locally available computing resources or extend them using cloud resources. The effortless combination of the different distribution methods is a unique feature of this prototype, showing the potential for bioinformatics software to optimally use existing infrastructure and extend it if needed. While the implemented prototype has been integrated into a commercial NGS data analysis software, GensearchNGS, the proposed architecture is applicable to other alignment algorithms. The current implementation and its source code is not publicly accessible, but we have plans to release it at a later date under the GNATY (GensearchNGS Analysis Tools librarY) project. GNATY is in the process of being published and is free of access. Future work will include the optimization of the architecture as well as addressing the issue of datasecurity in the cloud related to DNA sequencing data.

References

[1] Li, H., & Homer, N. (2010). *A survey of sequence alignment algorithms for next-generation sequencing*, Briefings in Bioinformatics, 11(5), 473-83.

[2] O. Gotoh, *An improved algorithm for matching biological sequences*, Journal of Molecular Biology 162, 705-708, 1982.

[3] B. Langmead, M. C. Schatz, J. Lin, M. Pop and S. L. Salzberg, *Searching for SNPs with cloud computing*, Genome biology, 10:R134, 2009.

[4] M. C. Schatz, *CloudBurst: highly sensitive read mapping with MapReduce*, Bioinformatics, 25, 11, 1363-1369, 2009.

[5] C. S. Oehmen and D. J. Baxter, *ScalaBLAST 2.0: rapid and robust BLAST calculations on multiprocessor systems*, Bioinformatics, 29, 6, 797-798, 2013.

[6] S. A. Issa *et al.*, *Streaming Support for Data Intensive Cloud-Based Sequence Analysis*, BioMed research international, vol. 2013, Art.no. 791051, 2013.

[7] B. Wolf, P. Kuonen and D. Atlan, *Distributed DNA alignment, a stream based approach*, Doctoral Workshop on Distributed Systems, Bern, Switzerland, Proc., 39-41, 2012.

[8] G. Highnam *et al.*, *An analytical framework for optimizing variant discovery from personal genomes*, Nature Communications, 6, Art.no. 6275, 2015.

[9] B. Wolf *et al.*, *DNAseq Workflow in a Diagnostic Context and an Example of a User Friendly Implementation*, BioMed Research International, vol. 2015