Bioinformatics, 0(0), 2017, 1–3
doi: 10.1093/bioinformatics/btx354
Advance Access Publication Date: 5 June 2017
Applications Note

OXFORD

Phylogenetics

PASTASpark: multiple sequence alignment meets Big Data

José M. Abuín*, Tomás F. Pena and Juan C. Pichel

CiTIUS, Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain

*To whom correspondence should be addressed.

Associate Editor: Alfonso Valencia

Received on February 27, 2017; revised on May 2, 2017; editorial decision on May 29, 2017; accepted on June 1, 2017

Abstract

Motivation: One basic step in many bioinformatics analyses is the multiple sequence alignment. One of the state-of-the-art tools to perform multiple sequence alignment is PASTA (Practical Alignments using SATé and TrAnsitivity). PASTA supports multithreading but it is limited to process datasets on shared memory systems. In this work we introduce PASTASpark, a tool that uses the Big Data engine Apache Spark to boost the performance of the alignment phase of PASTA, which is the most expensive task in terms of time consumption.
Results: Speedups up to 10× with respect to single-threaded PASTA were observed, which allows to process an ultra-large dataset of 200 000 sequences within the 24-h limit.
Availability and implementation: PASTASpark is an Open Source tool available at https://github.com/citiususc/pastaspark
Contact: josemanuel.abuin@usc.es
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Multiple sequence alignment (MSA) is essential in order to predict the structure and function of proteins and RNAs, estimate phylogeny, and other common tasks in sequence analysis. PASTA (Mirarab *et al.*, 2015) is a tool, based on SATé (Liu *et al.*, 2012), which produces highly accurate alignments, improving the accuracy and scalability of other state-of-the-art methods, including SATé. PASTA is based on a workflow composed of several steps. During each phase, an external tool is called to perform different operations such as estimating an initial alignment and tree, computing MSAs on subsets of the original sequence set, or estimating the maximum-likelihood tree on a previously obtained MSA. Note that computing the MSAs is the most time-consuming phase, implying in some cases over 70% of the total execution time.

PASTA is a multithreaded application that only supports shared memory computers. In this way, PASTA is limited to process small- or medium-sized input datasets, because the memory and time requirements of large datasets exceed the computing power of any shared memory system. In this work we introduce PASTASpark, an extension to PASTA that allows to execute it on a distributed memory cluster making use of Apache Spark (Zaharia *et al.*, 2010). Apache Spark is a cluster computing framework that supports both in-memory and on-disk computations in a fault tolerant manner, using distributed memory abstractions known as Resilient Distributed Datasets (RDDs). PASTASpark reduces noticeably the execution time of PASTA, running the most costly part of the original code as a distributed Spark application. In this way, PASTASpark guarantees scalability and fault tolerance, and allows to obtain MSAs from very large datasets in reasonable time.

2 Approach

PASTA was written in Python and Apache Spark includes APIs for Java, Python, Scala and R. For this reason, authors use the Spark Python API (known as PySpark) to implement PASTASpark. The design of PASTASpark minimizes the changes in the original PASTA code. In fact, the same software can be used to run the unmodified PASTA on a multicore machine or PASTASpark on a cluster: If Python is used, the original PASTA is launched, while if the job is submitted through Spark, PASTA is automatically executed in parallel using the Spark worker nodes.

The PASTA iterative workflow consists of four steps or phases. In the first phase (P1), a default starting tree is computed from the input sequence set S. Afterwards, using the tree and the centroid decomposition technique in SATé-II, S is divided into disjoint sets $S_1, \ldots, S_m$ and a spanning tree T^* on the subsets is obtained. In the second phase (P2), the MSAs on each S_i are obtained. By default, MAFFT (Katoh *et al.*, 2005) is used in this step, but other aligners could be employed. The resulting alignments are referred as *type 1 subalignments*. Now, in the third phase (P3), OPAL (Wheeler and Kececioglu, 2007) is used to align the *type 1 subalignment* for every edge (v, w) in T^*, producing the so-called *type 2 subalignment*, from which the final MSA is obtained through a sequence of pairwise mergers using transitivity. Finally, in the fourth phase (P4), if an additional iteration is desired, FastTree-2 (Price *et al.*, 2010) is executed to estimate a maximum-likelihood tree on the MSA produced on the previous step, and the process is repeated using this tree as input.

As it was stated in Mirarab *et al.* (2015), the most time-consuming phase in PASTA is the computation of MSAs using MAFFT (P2). Due to this, PASTASpark focuses on parallelizing this step. In the original PASTA, P2 is parallelized using Python multiprocessing. As MAFFT requires a file as input, the computed subsets S_i have to be stored in files from which each PASTA process calls the aligner using the Python class Popen. By default, PASTA creates so many processes as cores are available in the machine. This procedure has several important limitations: it only works on shared memory machines, it implies storing to disk a large amount of data, which could be a bottleneck, and, finally, it prevents MAFFT to run in parallel taking advantage of its OpenMP implementation.

To overcome these limitations, PASTASpark creates an in-memory RDD of key-value pairs, in case it detects that the Spark engine is running. In particular, the key is an object that includes information about the aligner and the required parameters to run it, while the value is an object containing a subset S_i. This RDD is automatically distributed by Spark to the worker nodes in the cluster, receiving each worker a slice of the RDD. Then, a map transformation is applied to the RDD. As a consequence, the input data are stored to each local disk of the worker nodes, and the aligner is invoked to process each local file using the Popen class. Notice that the amount of data stored to any of the local disks is divided by the number of workers. We must also highlight that the storing process is done in parallel, which reduces significantly the I/O cost. Besides, if the worker nodes are multicore machines, the aligner software could run in parallel using several threads, which is also an important advantage of PASTASpark over the original PASTA application. The output of the map transformation is a new RDD that contains the *type 1 subalignments*. The resulting RDD is collected by the Spark Driver in order to continue the PASTA workflow. In this way, P2 is performed by the workers, while P1, P3 and P4 are executed by the Spark Driver. More details about PASTA and PASTASpark can be found in the Supplementary Material.

Table 1. Main characteristics of the input datasets

Tag	Name	No. of sequences	Avg. sequence length	Size
D1	16S.B.ALL	27 643	1371.9	184.1 MB
D2	50k RNASim	50 000	1556	591.8 MB
D3	200k RNASim	200 000	1556	3.4 GB

3 Results and discussion

Input datasets from the original PASTA publication are used to test PASTASpark performance. A summary of their main characteristics is shown in Table 1. Note that a starting tree is available for all the considered datasets, so its calculation in P1 is avoided. The experimental evaluation was carried out considering two different clusters, CESGA and AWS (see description in the Supplementary Material).

Table 2 shows the execution times of PASTA and PASTASpark when running on the CESGA cluster (PASTA can only run on a single 8-core node). Important improvements were observed when using PASTASpark with different number of workers (cores). Note that the table displays between brackets the percentage of time spent in the alignment phase (P2), which was the one parallelized by PASTASpark. Those values are essential to understand the corresponding speedups achieved by PASTASpark with respect to original PASTA (see Fig. 1a and b). In these figures, we have used the Amdahl's law (Amdahl, 1967) to estimate the theoretical maximum speedup achievable by PASTASpark. This law states that if a fraction s of the work for a given problem cannot be parallelized, with $0 < s \leq 1$, while the remaining portion, $1 - s$, is p-fold parallel, then the maximum achievable speedup is $1/[s + (1 - s)/p]$. In our particular case, P1, P3 and P4 phases in PASTA are multithreaded, so they could not be, in theory, considered sequential code. However, the execution time of P1 and P3 is really small with respect to P2 and P4, so without losing precision we can consider their execution time as a constant. On the other hand, the scalability of P4 (FastTree-2) is poor and it does not scale beyond 1.5–1.7× using three or more threads. Therefore, as a valid approximation for the current implementation of PASTA we consider P1, P3 and P4 as sequential processes. Red lines in Figure 1a and b show the Amdahl's law theoretical maximum speedup applied to D1 and D2 datasets. It can be observed that PASTASpark is close to the upper limit, obtaining speedups up to 10× when using 64 workers (cores).

Finally, Figure 1c displays the performance results of PASTASpark running on the AWS cluster using a different number of computing nodes. Each node in this system consists of 16 cores. It is worth to mention that PASTASpark is able to process an ultra-large dataset of 200 000 sequences (D3) within the 24-h limit using only eight AWS nodes.

Table 2. Execution time (hours) for D1 and D2 using the CESGA cluster

		No. of workers (cores)				
		1	8	16	32	64
D1	PASTA	35.5	7.2 [69.5%]	—	—	—
	PASTASpark		7.4 [70.3%]	5.4 [59.3%]	4.5 [51.1%]	3.7 [40.6%]
D2	PASTA	100.5	18.7 [74.8%]	—	—	—
	PASTASpark		20.9 [77.5%]	14.4 [67.4%]	11.6 [59.5%]	9.5 [50.5%]

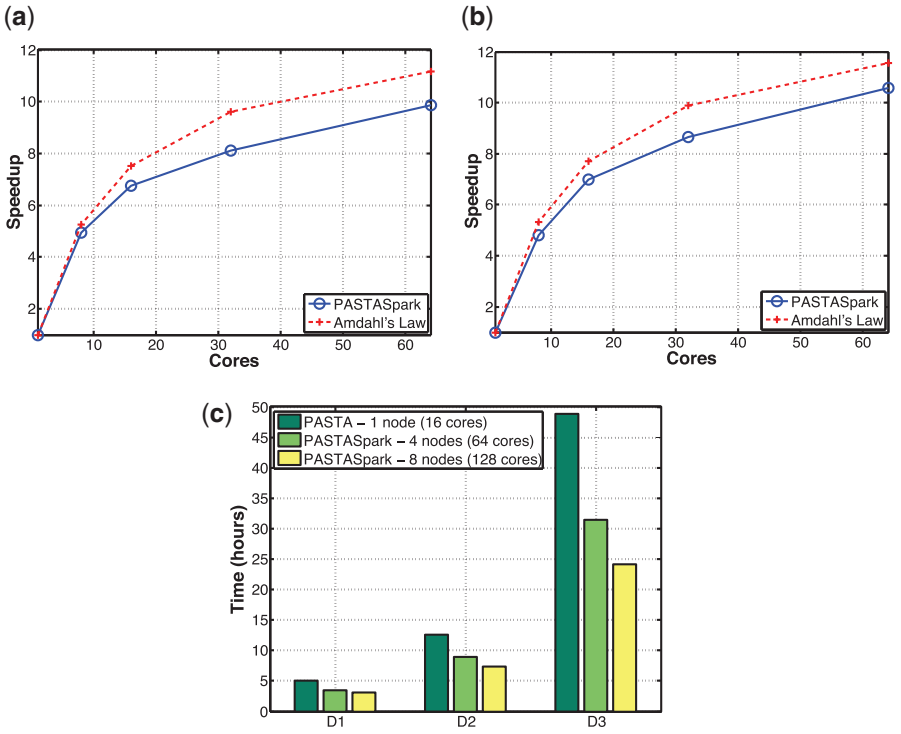

Fig. 1. Speedup considering D1 (**a**) and D2 (**b**) datasets on the CESGA cluster, and execution times on the AWS cluster (**c**)

Funding

This work was supported by MINECO [TIN2016-76373-P, TIN2014-54565-JIN]; Xunta de Galicia [ED431G/08]; European Regional Development Fund; and Amazon Web Services Cloud Credits for Research program.

Conflict of Interest: none declared.

References

Amdahl,G.M. (1967) Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the AFIPS'67*, pp. 483–485.

Katoh,K. *et al.* (2005) MAFFT: improvement in accuracy of multiple sequence alignment. *Nucleic Acids Res.*, **33**, 511–518.

Liu,K. *et al.* (2012) SATé-II: very fast and accurate simultaneous estimation of multiple sequence alignments and phylogenetic trees. *Syst. Biol.*, **61**, 90–106.

Mirarab,S. *et al.* (2015) PASTA: ultra-large multiple sequence alignment for nucleotide and amino-acid sequences. *J. Comput. Biol.*, **22**, 377–386.

Price,M. *et al.* (2010) FastTree2 – approximately maximum-likelihood trees for large alignments. *PLoS One*, **5**, e9490.

Wheeler,T.J. and Kececioglu,J.D. (2007) Multiple alignment by aligning alignments. *Bioinformatics*, **23**, i559–i568.

Zaharia,M. *et al.* (2010) Spark: cluster computing with working sets. In: *Proceedings of the USENIX Conference on Hot Topics in Cloud Computing*, p. 10. USENIX Association, Boston, USA.