Hindawi Publishing Corporation
Computational Biology Journal
Volume 2013, Article ID 707540, 12 pages
http://dx.doi.org/10.1155/2013/707540

Research Article

SCBI_MapReduce, a New Ruby Task-Farm Skeleton for Automated Parallelisation and Distribution in Chunks of Sequences: The Implementation of a Boosted Blast+

Darío Guerrero-Fernández,[1] Juan Falgueras,[2] and M. Gonzalo Claros[1,3]

[1] *Supercomputación y Bioinformática-Plataforma Andaluza de Bioinformática (SCBI-PAB), Universidad de Málaga, 29071 Málaga, Spain*
[2] *Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, 29071 Málaga, Spain*
[3] *Departamento de Biología Molecular y Bioquímica, Universidad de Málaga, 29071 Málaga, Spain*

Correspondence should be addressed to M. Gonzalo Claros; claros@uma.es

Received 21 June 2013; Revised 18 September 2013; Accepted 19 September 2013

Academic Editor: Ivan Merelli

Current genomic analyses often require the managing and comparison of big data using desktop bioinformatic software that was not developed regarding multicore distribution. The task-farm SCBI_MapReduce is intended to simplify the trivial parallelisation and distribution of new and legacy software and scripts for biologists who are interested in using computers but are not skilled programmers. In the case of legacy applications, there is no need of modification or rewriting the source code. It can be used from multicore workstations to heterogeneous grids. Tests have demonstrated that speed-up scales almost linearly and that distribution in small chunks increases it. It is also shown that SCBI_MapReduce takes advantage of shared storage when necessary, is fault-tolerant, allows for resuming aborted jobs, does not need special hardware or virtual machine support, and provides the same results than a parallelised, legacy software. The same is true for interrupted and relaunched jobs. As proof-of-concept, distribution of a compiled version of Blast+ in the SCBI_Distributed_Blast gem is given, indicating that other blast binaries can be used while maintaining the same SCBI_Distributed_Blast code. Therefore, SCBI_MapReduce suits most parallelisation and distribution needs in, for example, gene and genome studies.

1. Introduction

The study of genomes is undergoing a revolution: the production of an ever-growing amount of sequences increases year by year at a rate that outpaces computing performance [1]. This huge amount of sequences needs to be processed with the well-proven algorithms that will not run faster in new computer chips since around 2003 chipmakers discovered that they were no longer able to sustain faster sequential execution except for generating the multicore chips [2, 3]. Therefore, the only current way to obtain results in a timely manner is developing software dealing with multicore CPUs or clusters of multiprocessors. In such a context, "cloud computing" is becoming a cost-effective and powerful resource of multicore clusters for task distribution in bioinformatics [1, 2].

Sequence alignment and comparison are the most important topics in bioinformatic studies of genes and genomes. It is a complex process that tries to optimise sequence homology by means of sequence similarity using the algorithm of Needleman-Wunsch for global alignment, or the one of Smith-Waterman for local alignments. Blast and Fasta [4] are the most widespread tools that have implemented them. Paired sequence comparison is inherently a parallel process in which many sequence pairs can be analysed at the same time by means of functions or algorithms that are iteratively performed over sequences. This is impelling the parallelisation of sequence comparison algorithms [5–9] as well as other bioinformatic algorithms [10, 11].

In most cases, the parallelised versions need to be rewritten from scratch, including explicit parallel programming

related to communication and synchronisation [12]. This makes programming software for distributed systems a very challenging task [13], and important long-running data processing scripts for bioinformatics remain unparallel. Hence, it should be desirable to have a flexible, general-purpose framework for distribution that could (i) take advantage of the existing scripts and/or binaries without requiring any source code modification, (ii) be used for distributing new bioinformatic algorithms, (iii) transfer data in the most secure form when secure connections cannot be established, and (iv) exploit the total computational power of any multicore computing system, allowing for parallelisation among cores and distribution between computers.

2. Related Work

Native threads is a satisfactory approach in compiled computer languages (for example Jrpm [14], a Java runtime machine for parallelising loops in sequential Java programs automatically), but it may not be fully implemented in scripting languages. But there are efficient, dedicated computer languages such as ErLang and Scala [15, 16], which offer programmable solutions for specific concurrent models. Although being quite efficient, its main disadvantage is that its use requires whole code rewriting, making embarrassingly parallel task regions or orchestrating communication as well as synchronisation, which is reserved only for skilled programmers. Moreover, the resulting parallel/distributed code remains bonded to the software version that is adapted.

A *de facto* standard model used in scientific high-performance computing analysis is the Message-Passing Interface (MPI) [17], whose most widely implementations are pyMPI (http://pympi.sourceforge.net/) that requires explicit parallel coding, OpenMP [18], and a set of compiler directives and callable runtime library routines that enables shared-memory parallelism. OpenMP includes a set of synchronisation features, since programmers are responsible for checking dependencies, deadlocks, race conditions, etc. There is also the R library `Rmpi`, which is a wrapper for porting MPI to R with the same pros and cons of MPI.

True parallelisation/distribution frameworks can also be achieved by means of MapReduce [19] and its most widely distributed implementation, Hadoop [20]. A promising, new resource is YARN [21], which introduces a generic scheduling abstraction that allows multiple parallelisation/distribution frameworks (for example, Hadoop and MPI) to coexist on the same physical cluster. Researchers can also find Condor [22], a specialised, full-featured workload management system for compute-intensive jobs. It is easy to use, but provides sub-optimal solutions. The BOINC platform [23] is a distributed sequence alignment application that offers the aggregation of the available memory of all participating nodes, but it suffers from communication overhead.

Parallelisation libraries for R language, besides `Rmpi`, are `SPRINT` [24] and `pR` [25] packages, whose their main advantage is that they require very little modification to the existing sequential R scripts and no expertise in parallel computing; however, the master worker suffers from communication overhead, and the authors recognise that their approach may not yield the optimal schedule [25]. Other parallelisation libraries are `snow` and `nws` that provide coordination and parallel execution facilities.

More general-purpose tools, such as bag-of-tasks engines for multicore architectures and small clusters, have also been developed in Python, such as PAR [26]; its main disadvantage is that it is hard to put in practice and is only available for small clusters. There is also FastFlow [27], a C++ pattern-based programming framework for parallel and distributed systems; although it simplifies the task of distributed software, it must be compiled on every machine and seems more appropriate for skilled programmers in C++. It has been argued that abstractions are an effective way of enabling nonexpert users to harness clusters, multicore computers and clusters of multicore computers [13]. Although abstractions can enable the creation of efficient, robust, scalable, and fault tolerant implementations and are easy for nonexpert programmers, they are specialised to a restricted class of workloads and its customisation to produce general-purpose tools is not trivial.

The comparison of nucleotide or protein sequences from the same or different organisms is a very powerful tool in the study of gene and genomes for finding similarities between sequences to infer the biological function and structure of newly sequenced genes, predict new members of gene families, decipher genome organisation, and explore evolutionary relationships. Blast is the algorithm of choice for such analyses, and its performance has been continuously improved, particularly from the arrival of high-throughput sequencing. That is why it is becoming a critical component of genome homology searches and annotation in many bioinformatics workflows. Improvements in its execution speed will result in significant impact in the practice of genome studies. Therefore, important efforts have been invested in accelerating it for different computers systems (to cite a few, mpiBLAST [6, 12], CloudBLAST [28], AzureBlast [29], GPU-Blast [30], and scalaBLAST 2.0 [31]). These Blast parallelisations require computer expertise to produce and adapt a particular Blast code and are tightly bonded to the software version included in the parallelised/distributed code [31]. It has been reported [9] that most MPI- and GPU-based Blast are only adequately optimal and generalizable to perform the batch processing of small amounts of sequence data on small clusters. Therefore, there is room for a distributed, flexible, easily-upgradeable version of Blast.

It can be inferred that distributed algorithms are becoming a real need in present bioinformatics research in order to adapt legacy and new software to multicore computing facilities. This paper describes SCBI_MapReduce, a new task-farm skeleton for the Ruby scripting language [32] that gathers the requirements presented in the Introduction, and simplifies the creation of parallel and distributed software to researchers without skills in distributed programming. Even if customisation could appear more complicated than using existing libraries for parallelisation—such as OpenMP [18], BOINC [23], or R libraries such as `Rmpi` or SPRINT [24]—it is as simple as the parallelisation of R code using `pR` [25]. In contrast to these libraries, SCBI_MapReduce is not constrained only to parallelisation (not allowing distribution nor grid

computing) and is able to extract the complete distribution capabilities of any computer system. While other systems like MPI [17] cannot deal with node failure, SCBI_MapReduce, like Hadoop [20] and FastFlow [27], includes implementation of error handling and job checkpointing methods. Moreover, it gathers additional features for task-skeletons such as encryption, compression on-the-fly, and distribution in chunks. As a proof-of-concept of use with legacy software, the SCBI_Distributed_Blast gem was developed to distribute the widely used Blast+ [4] application. This gem is not bonded to the Blast+ version included in it, since any Blast+ binary of the scientist computer can be used.

3. Methods

3.1. Hardware and Software. Scripting code was based on Ruby 1.9 for OSX and SLES Linux. The computing facilities used were (i) a "x86" cluster, consisting of 80 x86_64 E5450 cores at 3.0 GHz with 16 GB of RAM every 8-core blade connected by an InfiniBand network and a PBS queue system; (ii) a "x86 upgraded" cluster, consisting of 768 x86_64 E5-2670 cores at 2.6 GHz with 64 GB of RAM every 16-core blade connected by an InfiniBand FDR network and a Slurm queue system; (iii) a homogeneous symmetric multiprocessing machine, consisting of a "SuperDome" of 128 Itanium-2 cores at 1.6 GHz with 400 Gb of RAM; and (iv) two x86 computers with 4 cores at 2.7 GHz using OSX and four x86 computers with 8 cores at 2.8 GHz using Linux and connected by gigabit Ethernet (GbE).

3.2. The Task-Farm Skeleton Design. Based on the well-known approach of MapReduce [21], which restricted it to trivial problems in which no communication between workers is needed (Figure 1(a)), SCBI_MapReduce takes a further step following a task-farm skeleton design (Figure 1(b)). This design does not need synchronous handling and entails asymmetry-tolerance [33]. It works launching one "manager" process to dispatch "tasks" to "workers" on demand. When a new worker is connecting, it automatically receives the input parameters and a data chunk (e.g., a group of sequences) from the manager. Since the skeleton contains the capability to take advantage of shared storage (lustre, nfs, ibrix, stornext, and even samba), common data for all workers (e.g., the subject database for Blast+) are not copied on every node, saving disk space and correspondingly diminishing the data transfer and starting delay for every worker. After task completion, the worker sends the results back to the manager. The results are then written to disk, and a new assignment is sent to the now idle worker. This cycle is repeated until the manager does not have any other data to process. As a result, it behaves as black-box where the only requirement is to code some predefined methods or calls. Any training in parallel programming or communication libraries is unnecessary, the resulting code apparently remaining sequential at the user level (see the appendix).

The number of workers is indicated by the user, although additional workers can be launched/stopped at any time. If workers do not use the full capacity of the cores, more workers than cores can be defined. Therefore, the task-farm design of Figure 1(b) using a stand-along manager—that is only dedicated to file opening, chunk construction, data saving, and worker coordination—avoids threading control, diminishes the idle times, and provides asymmetry tolerance.

3.3. Implementation of Other Relevant Features. Since no particular compiler technology is required, SCBI_MapReduce skeleton is prepared for workers to be executed simultaneously over a mixture of architectures (x86_64, PPC, ia64, i686) running on UNIX-like standalone machines, clusters, symmetric multiprocessing machines, and grids. Having a connection protocol based on TCP/IP, the skeleton can handle at the same time several interconnection networks (Ethernet, Gigabit, InfiniBand, Myrinet, optic-fiber with IP, etc.). Additionally, when network transfers are required, data encryption, as well as compression, can be enabled to guarantee data privacy. Any encryption and compression program installed on the user computer can be invoked.

Implemented input/output (I/O) operations have been optimised to diminish reading/writing overload. Optimisation consisted of (i) the use of EventMachine Ruby library for asynchronous I/O events for networked operations; (ii) the manager reads data from disk only once at the beginning of the job; (iii) the data, in the form of objects, are maintained in memory during the entire job to avoid further disk access; (iv) the data are split in memory by the manager into chunks of objects of customisable size; and (v) the results are written on disk only at the end of a task (see example in the code of the appendix). As a result, once the whole required software is installed on every worker, only the manager needs to have access to data on disk. However, the use of shared storage is optional. These features provide portability to SCBI_MapReduce implementations.

Fault-tolerance policy and basic error handling capability were included. These capabilities enabled the safe execution of SCBI_MapReduce over long periods when (i) execution exception occurs: it is reported to the manager, which can try to restart the faulty task instead of stopping the job; (ii) a worker fails: the manager redistributes the data to a new worker and launches the same task again; (iii) unexpected job interruption: since completed jobs are checkpointed to disk, when the user restarts the interrupted job, the manager is able to resume execution precisely at the object being processed when the interruption occurred; (iv) log file recording: an exhaustive log file can be tracked to find execution problems for debugging purposes; and finally (v) too buggy job: when a high error rate in workers is detected, the manager stops the job and informs the user that data are faulty and that he/she should review them before launching a new job.

3.4. Usage and Customisation of SCBI_MapReduce. Although the code can be downloaded from http://www.scbi.uma.es/downloads/, its easiest installation is as any other Ruby gem [32] using the single command *sudo gem install scbi_mapreduce*. An on-line help can be obtained using the *scbi_mapreduce -h* command. Skeleton customisation only requires modifying parts of the initial configuration parameters, data chunk sizes, worker processes, and work dispatcher to achieve a fully distributed system (see details at the

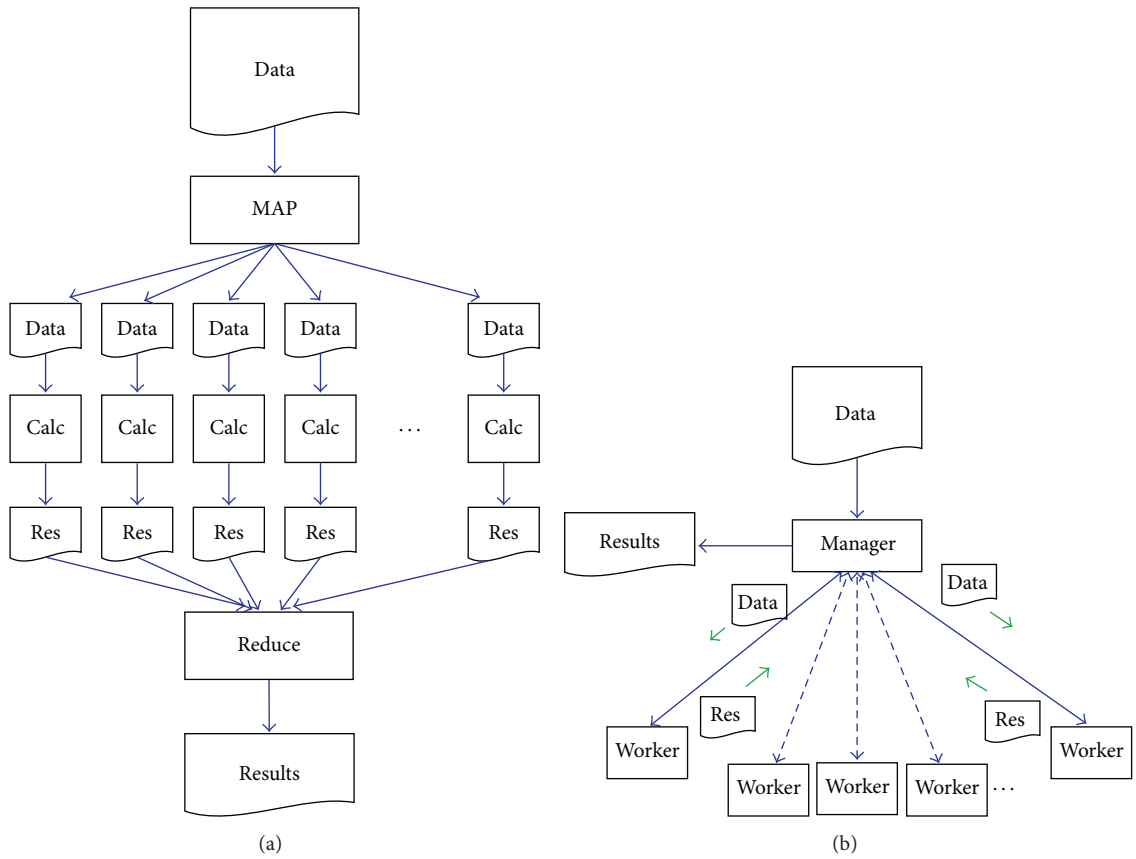

Figure 1: Comparison of parallelisation flowgrams for a single "job". (a) The classic MapReduce view in which the input data of a job are split into smaller data chunks by a "mapper" process and executed by separated parallel "tasks"; once all tasks have been finished, the results are combined by a "reduce" process. (b) SCBI_MapReduce task-farm flowgram, in which a job is handled by a single "manager", which is controlling the start and end of each distributed "task" executed by every "worker".

appendix). This requires from the user some knowledge for calling external code and I/O operations.

Three different templates (one for string capitalisation, a second for the simulated calculations on integers as shown in Table 1, and a third for calculations as shown in Table 1 using the datasets of artificial and real-world sequences, which finds and removes barcodes from sequences) are provided as a customisable startup point for any project using the command *scbi_mapreduce my_project template_name*.

3.5. SCBI_Distributed_Blast Gem. Basic Local Alignment Search Tool (Blast) [4] is the tool most frequently used for calculating sequence similarity, usually being the computationally intensive part of most genomic analyses. Its popularity is based on its heuristics that enable it to perform the search faster. This drives us to choose it as a proof-of-concept for distribution of legacy software. The binary release v. 2.2.24 of Blast+ [4] was elected to demonstrate that SCBI_MapReduce can be used as a distribution wrapper for legacy algorithms. Since a main drawback of Blast parallelisations is that the entire sequence database must be copied on each node, SCBI_Distributed_Blast takes advantage of shared storage to avoid the waste of disk space and the potential scalability impairment for large databases.

To use SCBI_Distributed_Blast, it is only needed to wrap any generic Blast or Blast+ command as follows:

```
scbi_distributed_blast -w 8
'any_blast_command'
```

which distributes Blast between 8 cores/workers ($-w$ option) with chunks of 100 sequences (default value of the $-g$ option). An example can be as follows:

```
scbi_distributed_blast -w 8
'blastn -task blastn-short -db
myDB.fna -query inputfile.fna
-out outputfile.fna',
```

where `blastn` is executed using 8 cores with `inputfile.fna` as a Fasta query file and `myDB.fna` as a customised database, `outputfile.fna` being the name of the output file.

4. Results

4.1. Scalability Studies. SCBI_MapReduce performance was tested in first instance using a dataset of 1000 objects (integers), as shown in Table 1, column "Integer dataset". Jobs were

Table 1: SCBI_MapReduce performance tests using three different datasets on the "x86 upgraded" cluster. Execution times are expressed in seconds. The number immediately before X indicates the number of reads grouped in a chunk for every parallel task.

Cores	Integer dataset[a]	Real-world sequences[b]				Artificial sequences[c]			
		1X	100X	250X	2000X	1X	100X	250X	2000X
1[d]	1264	13608	13849	13424	19328	23264	22124	24185	33393
2	635	8824	7903	7584	10251	11462	11554	11776	15302
4	322	4363	4507	4167	5890	6776	6507	5881	7503
8	164	2182	2194	2231	3132	3403	3337	3371	4874
16	81	1097	1098	1121	1633	1901	1797	1817	2602
32	41	568	549	569	899	921	888	915	1339
64	21	293	282	295	532	506	449	466	755
128	12	173	153	179	352	268	233	245	464

[a]Integers were subjected to futile, intensive calculations that took at least 1 s on every object.
[b]The dataset of real-world sequences consisted of 261 304 sequence reads (mean: 276 nt; mode: 263 nt; coefficient of variation: 11%) obtained from a 454/FLX sequencer downloaded from the SRA database (AC# SRR069473).
[c]The dataset of artificial sequences consisted of 425 438 sequences obtained using the software ART with a 2X coverage simulating a 454/FLX sequencing from the *Danio rerio* chromosome 1 (AC# NC_007112.5).
[d]Using one core is equivalent to a linear job without any parallelisation or distribution; it acts as control reference.

Table 2: Percent of time spent by the manager on every sequence-based job similary as detailed in Table 1.

Cores	Real-world sequences				Artificial sequences			
	1X	100X	250X	2000X	1X	100X	250X	2000X
2	0.92	0.48	0.49	0.41	0.84	0.44	0.43	0.36
4	0.71	0.41	0.45	0.35	0.68	0.37	0.39	0.32
8	0.58	0.40	0.41	0.34	0.55	0.35	0.34	0.29
16	0.52	0.39	0.39	0.34	0.58	0.41	0.37	0.28
32	0.52	0.50	0.51	0.37	0.47	0.45	0.44	0.33
64	0.47	0.56	0.55	0.40	0.37	0.45	0.48	0.34
128	0.61	0.57	0.62	0.37	0.54	0.48	0.49	0.40

launched using 1 (as control reference) to 128 cores on the "x86 upgraded" cluster. Since the speed-up achieved is close the maximal theoretical one (Figure 2(a), compare dotted line and solid lines), it can be suggested that SCBI_MapReduce scales well with simple objects such as integers. In fact, it is able to manage up to 18000 tasks of 1 kB each per second with a single core manager on the "x86" cluster (results are not shown).

SCBI_MapReduce can find a use beyond integers, as is demonstrated by the testing of two sequence datasets, in which each object is a sequence in Fasta format. Real-world sequences correspond to a true 454/FLX sequencing and the artificial sequences correspond to a simulation of 454/FLX sequencing using ART (http://www.niehs.nih.gov/research/resources/software/biostatistics/art/). Barcodes were localised on both sequence datasets varying sequence chunk sizes. Table 1 shows that the *a priori* most affordable parallelisation of sequence-by-sequence (1X chunk) did not provide the best speed-up (Figure 2(a), dark and open triangles). This could be explained in part by the fact that the manager is spending a little more time building a lot of small data chunks (Table 2). Higher chunks (100X and 250X) provided shorter, similar execution times (Figure 2(a), dark and open squares and circles). Regarding execution time (Table 1) and speed-up (Figure 2(a), dark and open diamonds), the hugest chunk (2000X) is not a good election. Since the manager is not taking more time during this job (Table 2), the reason for this speed-up impairment is that using higher chunks, the computational resources are not optimally used during the last calculation cycle, where most workers are idle and the manager is waiting for a few workers to finish a long task. This issue is also observed with other chunks, but it becomes apparent only with chunks of 2000X sequences since workers spend more time on every data chunk. In conclusion, SCBI_MapReduce was scaling almost linearly, and the optimal number of reads in a chunk is dependent on the number of workers and chunks used for the parallelisation. When the number of sequences (objects) to process is unknown, chunks of 2000X and 1X provided the lowest speed-up, while small chunks ranging from 100X to 250X sequences are preferable.

4.2. Compression and Encryption Overhead is Acceptable. One of the main features of SCBI_MapReduce is the capability of data compression and encryption on-the-fly. Since these capabilities rely on the manager, data security and privacy will be maintained by an increase in execution time. This overhead was tested using real-world jobs of Table 1 launched on the "x86 upgraded" cluster with the encryption and compression capabilities being enabled (Table 3). Distribution of 1X chunks was dramatically affected (Figure 2(b), open

Table 3: Analysis of compression and encryption of the same real-world sequence jobs in Table 1. Both the execution times (in seconds) and the percent of this time used by the manager are provided.

Cores	Execution time[a] (s)				Manager time (%)			
	1X	100X	250X	2000X	1X	100X	250X	2000X
2[b]	9482	8178	7270	10279	4.73	0.56	0.58	0.44
4	4619	4307	3814	5234	2.76	0.47	0.49	0.39
8	2359	2156	2165	3145	1.76	0.43	0.43	0.35
16	1274	1085	1142	1692	1.54	0.42	0.40	0.33
32	913	553	571	905	2.40	0.51	0.52	0.37
64	821	282	294	540	3.54	0.57	0.55	0.43
128	709	163	173	346	3.87	0.62	0.61	0.39

[a]Compression was performed with ZLib and encrypted with AES-256; any other method installed on computers can be used.
[b]There is no need of compression or encryption using one single core.

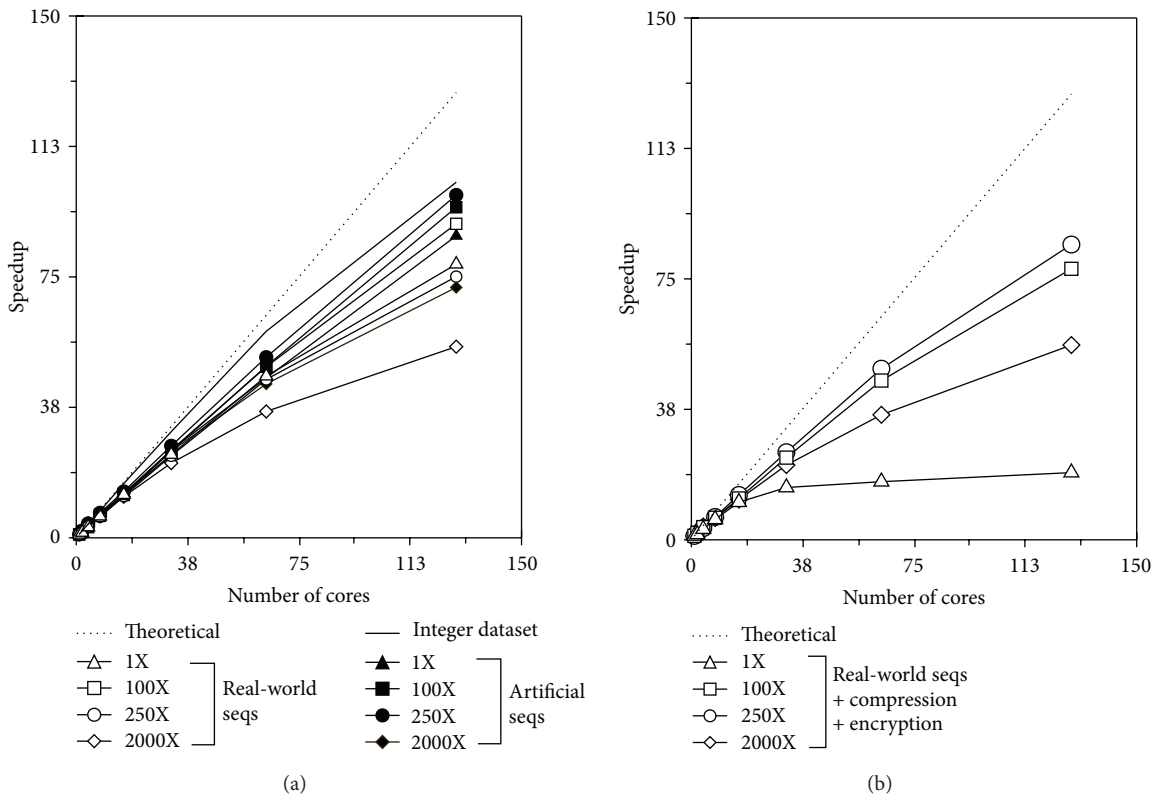

Figure 2: Speed-up achieved by SCBI_MapReduce implementations. (a) Speed-up based on Table 1 data was calculated dividing the time taken with 1 core by the time taken by each number of cores. (b) Performance using compressed and encrypted real-world sequences based on execution times in Table 3. The speed-up was calculated dividing the time that real-world sequences took with 1 core in Table 1 by their corresponding times in Table 3. In both plots, theoretical values correspond to a speed-up that equals the number of cores used.

triangles), and this may be due to the important increase of the time spent by the manager on every job (from 0.91%–0.47% in Table 2 to 4.75%–1.54% in Table 3). But overhead became bearable when using any other chunk size, since (i) the execution time in Table 3 for 100X–2000X chunks is close to the presented in Table 1, (ii) the speed-up can recover the previously observed values (Figure 2(b)), and (iii) the manager spends nearly the same percent of time. These results suggest that overhead introduced by encryption and compression can be alleviated using chunks, providing a significant speed-up, and that distribution of sequence-by-sequence (1X chunks) was the worst available approach. In conclusion, compression and encryption capabilities can be regularly used for distributed calculations when encrypted connections are not available but desirable, without the dramatic increase of execution time.

4.3. Fault-Tolerance Testing. The fault-tolerance of SCBI_MapReduce was tested using the complete dataset of real-world sequences as input, and the same analysis was

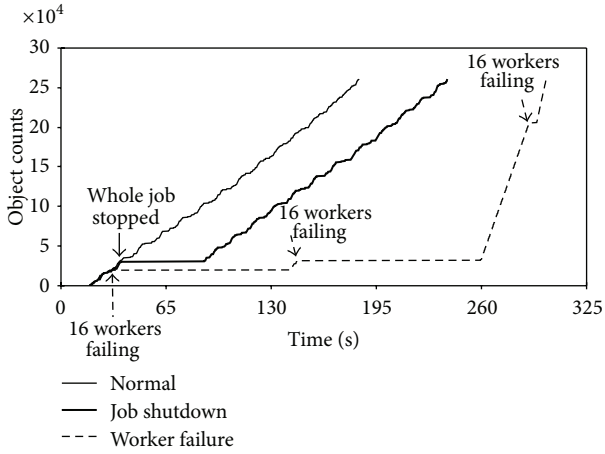

Figure 3: Fault tolerance testing using the real-world sequences in 100X chunks on the "x86 upgraded" cluster using 128 cores. The "Normal" execution occurred without errors. The "Job shutdown" included a complete job shutdown (indicated by a solid arrow) and then a manual restart. The "Worker failure" execution included three shutdowns (indicated by dashed arrows) of 16 workers each during the job.

performed on Table 1 on the "x86 upgraded" cluster using 128 cores. Sequences were used in 100X chunks, and the job was executed three times. The first execution occurred without errors and took 184 s in finishing the analysis of 261 304 reads (objects). Figure 3 shows that this "Normal" execution presents the expected constant slope. The second execution was to test the unexpected "Job shutdown", that was simulated with a manual interruption of the whole job, with the job being then manually relaunched. It can be seen in Figure 3 that the interruption adds a lag time to the job, increasing to 239 s which is the time required to finish the analysis of all sequences. The sequence counts were the same than in "Normal", indicating that no sequence and no chunk were reanalysed twice. Finally, the test of recovery after a "Worker failure" was performed stopping 16 workers at three different time points of the job execution (a total of 48 different workers were affected). In this case, the manager handles automatically the reanalysis of the unfinished chunks and the job took 300 s. Again, no sequence was saved twice in the output file. As expected, the output result of the "Normal" execution and the interrupted executions was exactly the same (results not shown). In conclusion, the fault-tolerance implementation of SCBI_MapReduce is able to handle execution exceptions, and broken workers and stopped jobs can be restarted without the reanalysis of finished tasks.

4.4. Distributed Blast+ Using Chunks of Sequences. The generic *blastn* command *blastn -task blastn-short -db myDB .fna -query inputfile.fna -out outputfile.fna* was launched with real-world sequences as input, and a customised database of 240 MB containing complete bacterial genomes. The speed-up achieved by SCBI_Distributed_Blast compared to the nondistributed execution (using 1 single core) is presented in Figure 4(a). Outputs of all executions were exactly the same in all cases, and also they were identical to the output of the original binary Blast+ (results not shown). Moreover, SCBI_Distributed_Blast was able to cope, without modification, with Blast+ versions 2.2.23 to 2.2.27.

Blast+ is described to have threading capabilities [4]. Therefore, 10 000 reads of AC# SRR069473 were launched with native `blastn` and with SCBI_Distributed_Blast, both configured to use the 8 cores of a single blade at the "x86" cluster. Figure 4(b) shows that Blast+ did not appear to efficiently parallelise since it started using only 5 cores and rapidly decreased to only one single core, taking 224 min to finish the task. Similar behaviour was confirmed in other computers, indicating that it seems to be an inherent feature of Blast+ releases. In contrast, SCBI_Distributed_Blast used 8 cores all the time and finished in 24 min (that is the reason why 0 CPU are "used" since then). Therefore, the speed-up introduced by SCBI_Distributed_Blast is 9.4, demonstrating that it performs much better in exploiting multicore capabilities than the threaded implementation included in Blast+.

5. Discussion

5.1. SCBI_MapReduce is an Efficient Task-Farm Skeleton. SCBI_MapReduce customisation is simpler than that customisation of other frameworks such as FastFlow [27], and it does not need compilation of the final code, making it portable to most computers "as is". Its flexibility allows to include SCBI_MapReduce as a coding part of any new algorithms as well as a wrapper for already existing functions, scripts, or compiled software. Promising and efficient results were provided when used within several of our algorithms (e.g., SeqTrimNext [http://www.scbi.uma.es/seqtrimnext] and Full-LengtherNext [http://www.scbi.uma.es/fulllengthernext], both are specifically designed to manage sequences obtained from next-generation sequencing), as well as a wrapper for Blast+ in the SCBI_Distributed_Blast gem (Figure 4). Therefore, SCBI_MapReduce seems to be sufficiently powerful for most of parallelisation and distribution needs concerning new algorithms, legacy software, and existing scripts in most bioinformatics contexts.

It has been described that GPU-based and some MPI-based parallelisations lack of good scalability when dealing with rapidly growing sequence data, while MapReduce seems to perform better in those settings [9]. That could explain why SCBI_MapReduce skeleton shows a speed-up of 31-fold for 32 cores and 59-fold for 64 cores, even with sequence data (Figure 2(a)). This performance is better than the one displayed by the R package `pR`, where 32 cores provide speedups of 20–27-fold, depending on the process [25]. Several design reasons can also be invoked to explain such an efficiency [34]: (i) disk I/O operations are reduced to minimum (data are read only at the beginning and results are saved only at the end); (ii) absence of asymmetry impact (Figure 1(b)); (iii) the manager overhead is limited when using more than 2 cores and chunks of sequences (Tables 2 and 3); and (iv) longer tasks increased the efficiency because the manager is on standby most of the time, while waiting

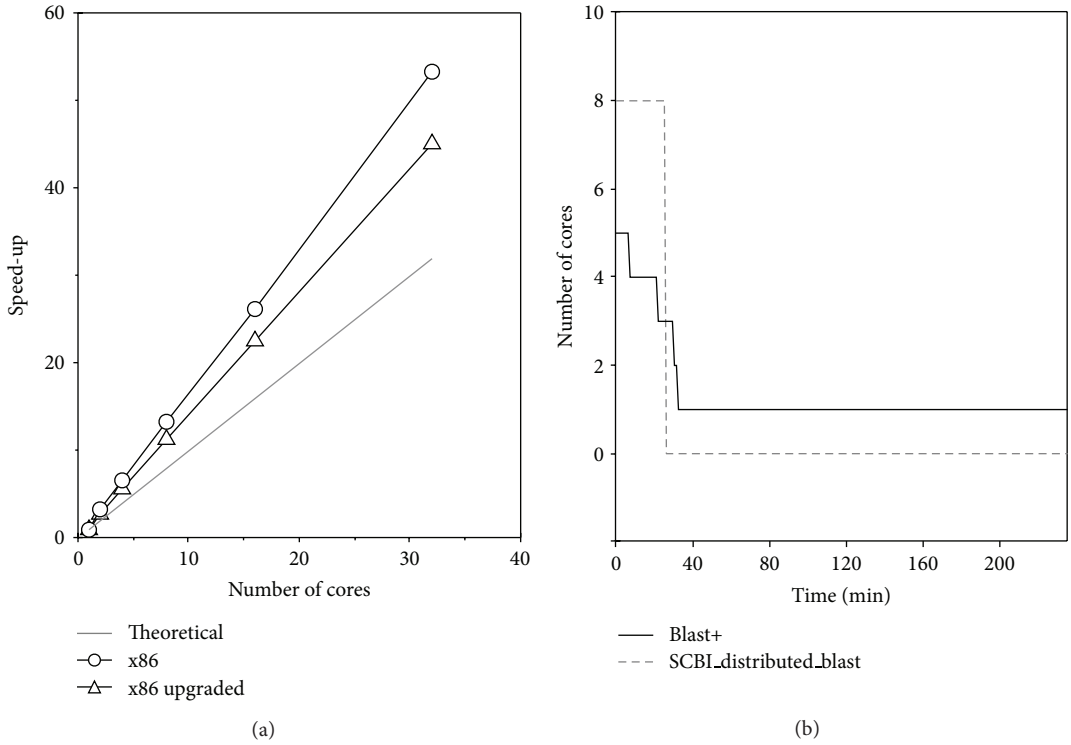

Figure 4: Behaviour of SCBI_Distributed_Blast. (a) Blast+ speed-up in chunks of 100X in two different clusters using both different network protocols and queue systems. Theoretical speed-up corresponds to the one that equals the number of cores used. Speed-up was calculated dividing the time spent using 1 core by the time of the corresponding number of cores. The following execution times were used: for 50 000 reads from AC# SRR069473 in the "x86" cluster, 25.8 h (92 880 s; 1 core), 27 600 s (2 cores), 13 980 s (4 cores), 6960 s (8 cores), 3540 s (16 cores), and 1740 s (32 cores); for the 261 304 reads of AC# SRR069473 in the "x86 upgraded" cluster, 88.6 h (318 960 s; 1 core), 115 161 s (2 cores), 56 385 s (4 cores), 28 180 s (8 cores), 14 123 s (16 cores), and 7068 s (32 cores). (b) Threaded Blast+ and SCBI_Distributed_Blast use differently the 8 cores available in the same computer. Blast+ was executed with the *-num_threads 8* option, and SCBI_Distributed_Blast was executed with the $-w$ 8 option using chunks of 100X by default in the "x86" cluster.

for the workers to finish, avoiding relaunching of internal or external programs for brief executions.

SCBI_MapReduce includes implementation of error handling and job checkpointing methods. It has been demonstrated (Figure 3) that data chunks from a broken worker, or even a job shutdown, can be relaunched in a running worker. This provides robustness and fault-tolerance, guarantees safe, long-lasting executions, and provides for preserving computational resources since it avoids processing objects that have already been processed. Such properties will serve to save time and make the job execution traceable at any time. Therefore, SCBI_MapReduce represents another step in the direction of programming environments with the task-farm skeleton concept.

5.2. Distribution in Chunks is More Efficient. SCBI_MapReduce was intended to deal with problems that involve processing a huge number of small sequences (as in high-throughput sequencing or RNA-Seq experiments). Results showed that splitting datasets into small chunks yields a better speed-up than sending sequences one by one or in big chunks (Figure 2(a)). An analogous idea has already been reported in mpiBLAST [35], but for database segmentation instead of sequence grouping. Therefore, grouping reads in chunks appear to be another way to provide speed-up, always taking into account that big chunks could be detrimental when the number of chunks produced is not divisible by the number of workers used (see 2000X in Figure 2).

Since chunk sizes of 100X and 250X perform similarly (Figure 2), a chunk size of 100X can suit well as default value, even if the optimal chunk size has not been assessed taking into account the number of cores and the number of objects to split in chunks. It could be then hypothesised that the use of chunks may reduce the manager surcharge (Tables 2 and 3). Figure 4(a) shows that speed-up could achieve superscalar behaviour using chunks combined with distribution, although this is dependent on the task performed by the worker (Blast+ in this instance) and not on the capabilities of SCBI_MapReduce. In conclusion, the use of chunks provides an improved overall performance.

5.3. The Added Value of Compression and Encryption Capability. In distributed grids or the cloud, encrypted connections cannot be always established for data privacy, and data compression can accelerate any transfer, particularly in low bandwidth connections. The overhead introduced by encryption and compression is particularly evident when data are processed one-by-one (Figure 2(b), open triangles), since the

```
class MyWorkerManager < WorkManager
 def self.init_work_manager
  # open input fastq file, and results as output
  @@fastq_file=FastqFile.new(fastq_file_path)
  @@results=FastqFile.new('./results.fastq','w+')
 end
 def self.end_work_manager
  #close files on finish
  @@fastq_file.close
  @@results.close
 end
 # this method is called every time a worker
 # needs a new work
 def next_work
  # get next sequence or nil from file
  name,fasta,qual,comments=@@fastq_file.next_seq
  if !name.nil?
   return name,fasta,qual,comments
  else
   return nil
  end
 end
 def work_received(results)
  # write results to disk
  results.each do |name,fasta,qual,comments|
   @@results.write_seq(name,fasta,qual,comments)
  end
 end
end
```

Algorithm 1

use of more and more cores did not significantly speed up the process. But compression and encryption overhead become acceptable when the dataset is split into chunks (compare slopes in Figure 2, and execution times in Tables 1 and 3). Encryption capability per chunks should be enabled only when untrusted networks are involved in distributed jobs. Compression per chunks could be envisaged when using low bandwidth networks (e.g., in some grids [2]), provided that compressed data transfer is faster than the time spent in compressing data. As a result, SCBI_MapReduce can be used on grids with confidential data when encrypted connections cannot be established.

5.4. SCBI_MapReduce is Ready for Grid Computing. It has been shown (Figure 4(a)) that SCBI_MapReduce, and therefore SCBI_Distributed_Blast, could work with homogeneous clusters (the "x86" and "x86 upgraded" clusters) consisting of different types of CPUs. It has been tested that SCBI_MapReduce was also able to deal with one heterogeneous grid consisting of one x86 computer using OSX, one x86 computer Linux, 24 cores of the "x86" cluster, and 32 cores of the "Superdome" (results are not shown). Hence, SCBI_MapReduce can cope with different queue systems (PBS, Slurm) and networks and can distribute in symmetric multiprocessing machines ("Superdome"), clusters (Figure 4(a)) and heterogeneous Unix-based grids (above). Other features that enable SCBI_MapReduce, at least theoretically [2, 36, 37], to be used in nearly any type of computer grid are (i) the above described encryption and compression capabilities; (ii) lack of administrator privileges; (iii) the case that running is on-demand only; and (iv) minimal requirement of hard disk since it takes advantage of shared storage only when necessary, making it highly portable to other computer systems. Testing SCBI_MapReduce in "cloud computing" services remains a pending task; however, it is expected that it should work and provide benefits related to cost-effectiveness.

5.5. SCBI_Distributed_Blast is a Boosted Version of Blast+. Previous improvements of Blast were performed by skilled programmers and provide parallelised versions tightly bonded to one released version. The development of SCBI_Distributed_Blast based on the SCBI_MapReduce task-farm skeleton comes to remove version bonding and coding challenges, since it can boost in a core-dependent way (Figure 4(a)) any Blast+ release installed on the scientist computer, not only the version tested in this study, enabling the update of the Blast+ release while maintaining the same SCBI_Distributed_Blast code.

In contrast to other Blast parallelisations, including mpiBLAST and MapReduce Blast [9], SCBI_Distributed_Blast distributed tasks are seeded with

sequence chunks while maintaining intact the database. This is because it does not need to copy the sequence database on each worker since it takes advantage of shared storage. This is also the reason why it provides exactly the same results as the original Blast+ in less time (Figure 4). Another MapReduce approach for Blast, CloudBlast [9], has very poor scalability since it is optimised for short reads mapping and needs to copy the database on each node, while the speed-up observed with SCBI_Distributed_Blast was linear and appeared to be superscalar in tested clusters (Figure 4(a)). However, superscalar behaviour was exclusively observed for 2 cores (speed-ups of 3.3 in the "x86" cluster and 2.8 in the "x86 upgraded" cluster), since taking two cores as reference, the speed-up slope was close to the theoretical speed-up (4.0, 8.1, 16.3, 32.6, 65.0, and 127.6).

Comparing the speed-up of our "boosted" Blast+ and a threaded execution of Blast+ (Figure 4(b)), it can be seen that SCBI_Distributed_Blast can take advantage of all computing capabilities and scales linearly, in contrast to native `Blast+` (Figure 4(b)) and the older NCBI-Blast [30]. In conclusion, SCBI_Distributed_Blast illustrates the ease-of-use and performance of SCBI_MapReduce, opening the way for code modifications that can easily produce scalable, balanced, fault-tolerant, and distributed versions of other Blast-related programs, like PSI-Blast, WU-Blast/AB-Blast, NCBI-Blast, and the like. Furthermore, Blast-based genome annotation processes can take advantage of SCBI_Distributed_Blast with minor changes in the code.

6. Conclusions

This work does not aim at advancing parallelisation technology, but it aims to apply distribution advantages in the use of bioinformatic tools that are useful, for example, for genomics, giving attractive speedups. In a context of continuous development of parallel software, SCBI_MapReduce provides a task-farm skeleton for parallelisation/distribution with features such as fault-tolerance, encryption and compression on-the-fly, data distribution in chunks, grid-readiness, and flexibility for integration of new and existing code without being a skilled programmer. In fact, SCBI_MapReduce was designed for researchers with a biological background that consider complicated MPI, Hadoop, or Erlang solutions for parallelisation/distribution. That is why Ruby was selected since it has a shallow learning curve, even for biologists, and easily manages the programming necessities. In the context of genomic studies, one significant advantage is that SCBI_MapReduce enables to reuse, in a commodity parallel/distributed computing environment, existing sequential code with little or no code changes. SCBI_Distributed_Blast can illustrate this.

Results indicate that SCBI_MapReduce scales well, is fault-tolerant, can be used on multicore workstations, clusters, and heterogeneous grids, even where secured connections cannot be established, can use several interconnection networks, and does not need special hardware or virtual machine support. It is also highly portable and shall diminish the disk space costs in "cloud computing". In conclusion, SCBI_MapReduce and hence SCBI_Distributed_Blast are ready, among other uses, for intensive genome analyses and annotations.

```
class MyWorker < Worker
 # process each obj in received objs
 def process_object(objs)
  # find barcodes
  find_mids(objs)
  return objs
 end
end
```

Algorithm 2

```
# get custom worker file path
custom_worker_file = 'my_worker.rb'

# init worker manager
MyWorkerManager.init_work_manager

# use any available ip and first empty port
ip='0.0.0.0'; port=0; workers = 4

# launch Manager and start it
manager = Manager.new(ip,port, workers,\...
  MyWorkerManager,custom_worker_file)
manager.start_server
```

Algorithm 3

Appendix

Customisation of the Three Files That Govern SCBI_MapReduce

SCBI_MapReduce consists of a number of files, but in order to be customised for particular needs, users only need to modify the I/O data management methods at the manager file (`my_worker_manager.rb`), the computation to be distributed at the worker file (`my_worker.rb`), and the main file (`main.rb`).

The methods to redefine in `my_worker_manager.rb` are (i) *next_work*, that provides new data for workers or *nil* if there is no more data available (in the following code, it simply reads one sequence at a time from a `fastq` file on disk); (ii) *self.init_work_manager*, that opens I/O data files; (iii) *self.end_work_manager*, that closes files when finished; and (iv) *work_received*, that writes results on disk as they are generated. The relevant code for my_worker_manager.rb is (see Algorithm 1).

Customisation of the worker file (`my_worker.rb`) includes redefinition of the *process_object* method that contains the function call to *find_mids*. The function *find_mids* can be defined by the user in his/her own source code or a compiled algorithm or an existing code. The relevant code for `my_worker.rb` is (see Algorithm 2).

The main program file (`main.rb`) has to be invoked to launch the distributed job. It can be used as it is from the

command line as a common Ruby script (`ruby main.rb`) or as a part of a more complex code. Skilled users can also modify its code and/or name to enable special features or even receive user parameters, which is the case when using SCBI_MapReduce for distribution of an internal part of an algorithm. The number of workers is defined here, at least one for the manager and one for one worker. The relevant code for `main.rb` is (see Algorithm 3).

Conflict of Interests

The authors declare that they have no conflict of interests.

Acknowledgments

The authors gratefully acknowledge Rafael Larrosa and Rocío Bautista for the helpful discussions and the computer resources of the Plataforma Andaluza de Bioinformática of the University of Málaga, Spain. This study was supported by Grants from the Spanish MICINN (BIO2009-07490) and Junta de Andalucía (P10-CVI-6075), as well as institutional funding to the research group BIO-114.

References

[1] C. Huttenhower and O. Hofmann, "A quick guide to large-scale genomic data mining," *PLoS Computational Biology*, vol. 6, no. 5, Article ID e1000779, 2010.

[2] M. C. Schatz, B. Langmead, and S. L. Salzberg, "Cloud computing and the DNA data race," *Nature Biotechnology*, vol. 28, no. 7, pp. 691–693, 2010.

[3] D. Patterson, "The trouble with multi-core," *IEEE Spectrum*, vol. 47, no. 7, pp. 28–53, 2010.

[4] C. Camacho, G. Coulouris, V. Avagyan et al., "BLAST+: architecture and applications," *BMC Bioinformatics*, vol. 10, article 421, 2009.

[5] S. Gálvez, D. Díaz, P. Hernández, F. J. Esteban, J. A. Caballero, and G. Dorado, "Next-generation bioinformatics: using many-core processor architecture to develop a web service for sequence alignment," *Bioinformatics*, vol. 26, no. 5, pp. 683–686, 2010.

[6] H. Lin, X. Ma, W. Feng, and N. F. Samatova, "Coordinating computation and I/O in massively parallel sequence search," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 4, pp. 529–543, 2011.

[7] T. Nguyen, W. Shi, and D. Ruden, "CloudAligner: a fast and full-featured MapReduce based tool for sequence mapping," *BMC Research Notes*, vol. 4, article 171, 2011.

[8] T. Rognes, "Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation," *BMC Bioinformatics*, vol. 12, article 221, 2011.

[9] X.-L. Yang, Y.-L. Liu, C.-F. Yuan, and Y.-H. Huang, "Parallelization of BLAST with MapReduce for long sequence alignment," in *Proceedings of the 4th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP '11)*, pp. 241–246, IEEE Computer Society, December 2011.

[10] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg, "Searching for SNPs with cloud computing," *Genome Biology*, vol. 10, no. 11, article R134, 2009.

[11] M. Needham, R. Hu, S. Dwarkadas, and X. Qiu, "Hierarchical parallelization of gene differential association analysis," *BMC Bioinformatics*, vol. 12, article 374, 2011.

[12] M. K. Gardner, W.-C. Feng, J. Archuleta, H. Lin, and X. Mal, "Parallel genomic sequence-searching on an ad-hoc grid: experiences, lessons learned, and implications," in *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing*, vol. 1, pp. 1–14, 2006.

[13] L. Yu, C. Moretti, A. Thrasher, S. Emrich, K. Judd, and D. Thain, "Harnessing parallelism in multicore clusters with the All-Pairs, Wavefront, and Makeflow abstractions," *Cluster Computing*, vol. 13, no. 3, pp. 243–256, 2010.

[14] M. K. Chen and K. Olukotun, "The Jrpm system for dynamically parallelizing Java programs," in *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA '03)*, pp. 434–445, San Diego, Calif, USA, June 2003.

[15] P. Haller and M. Odersky, "Scala Actors: unifying thread-based and event-based programming," *Theoretical Computer Science*, vol. 410, no. 2-3, pp. 202–220, 2009.

[16] J. Armstrong, R. Virding, C. Wikström, and M. Williams, *Concurrent Programming in ERLANG*, Prentice Hall, 2nd edition, 1996.

[17] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, Mass, USA, 2nd edition, 1999.

[18] L. Dagum and R. Menon, "Openmp: an industry-standard api for shared-memory programming," *IEEEComputational Science & Engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[19] Q. Zou, X.-B. Li, W.-R. Jiang, Z.-Y. Lin, G.-L. Li, and K. Chen, "Survey of mapreduce frame operation inbioinformatics," *Briefings in Bioinformatics*. In press.

[20] R. C. Taylor, "An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics," *BMC Bioinformatics*, vol. 11, supplement 12, p. S1, 2010.

[21] J. Lin, "Mapreduce is good enough?" *Big Data*, vol. 1, no. 1, pp. 28–37, 2013.

[22] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience," *Concurrency Computation Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[23] S. Pellicer, G. Chen, K. C. C. Chan, and Y. Pan, "Distributed sequence alignment applications for the public computing architecture," *IEEE Transactions on Nanobioscience*, vol. 7, no. 1, pp. 35–43, 2008.

[24] J. Hill, M. Hambley, T. Forster et al., "SPRINT: a new parallel framework for R," *BMC Bioinformatics*, vol. 9, article 558, 2008.

[25] J. Li, X. Ma, S. Yoginath, G. Kora, and N. F. Samatova, "Transparent runtime parallelization of the R scripting language," *Journal of Parallel and Distributed Computing*, vol. 71, no. 2, pp. 157–168, 2011.

[26] F. Berenger, C. Coti, and K. Y. J. Zhang, "PAR: a PARallel and distributed job crusher," *Bioinformatics*, vol. 26, no. 22, pp. 2918–2919, 2010.

[27] M. Aldinucci, M. Torquati, C. Spampinato et al., "Parallel stochastic systems biology in the cloud," *Briefings in Bioinformatics*. In press.

[28] A. Matsunaga, M. Tsugawa, and J. Fortes, "CloudBLAST: combining MapReduce and virtualization on distributed resources for bioinformatics applications," in *Proceedings of the 4th IEEE International Conference on eScience (eScience '08)*, pp. 222–229, IEEE Computer Society, Washington, DC, USA, December 2008.

[29] W. Lu, J. Jackson, and R. Barga, "AzureBlast: a case study of developing science applications on the cloud," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10)*, pp. 413–420, ACM, Chicago, Ill, USA, June 2010.

[30] P. D. Vouzis and N. V. Sahinidis, "GPU-BLAST: using graphics processors to accelerate protein sequence alignment," *Bioinformatics*, vol. 27, no. 2, pp. 182–188, 2011.

[31] C. S. Oehmen and D. J. Baxter, "Scalablast 2.0: rapid and robust blast calculations on multiprocessor systems," *Bioinformatics*, vol. 29, no. 6, pp. 797–798, 2013.

[32] J. Aerts and A. Law, "An introduction to scripting in Ruby for biologists," *BMC Bioinformatics*, vol. 10, article 221, 2009.

[33] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai, "The impact of performance asymmetry in emerging multicore architectures.," *SIGARCH Computer Architecture News*, vol. 33, no. 2, pp. 506–517, 2005.

[34] L. Jostins and J. Jaeger, "Reverse engineering a gene network using an asynchronous parallel evolution strategy," *BMC Systems Biology*, vol. 4, article 17, 2010.

[35] O. Thorsen, B. Smith, C. P. Sosa et al., "Parallel genomic sequence-search on a massively parallel system," in *Proceedings of the 4th Conference on Computing Frontiers (CF '07)*, pp. 59–68, Ischia, Italy, May 2007.

[36] M. Armbrust, A. Fox, R. Griffith et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[37] C.-L. Hung and Y.-L. Lin, "Implementation of a parallel protein structure alignment service on cloud," *International Journal of Genomics*, vol. 2013, Article ID 439681, 8 pages, 2013.