International Journal of Parallel Programming (2019) 47:317–342
https://doi.org/10.1007/s10766-018-0600-z

A History-Based Resource Manager for Genome Analysis Workflows Applications on Clusters with Heterogeneous Nodes

Ferran Badosa[1] · Antonio Espinosa[1] · Cesar Acevedo[1] · Gonzalo Vera[2] · Ana Ripoll[1]

Received: 13 November 2017 / Accepted: 7 September 2018 / Published online: 22 September 2018

Abstract
Bioinformatics workflows require large amounts of resources and are commonly executed in clusters. Determining the adequate amount of resources for bioinformatics applications is a tricky matter, since the resource usage of a single application might vary substantially from one execution to the next. Resource management systems in clusters don't consider these variations and subsequent needs. As a result, the computing power offered by clusters is not harnessed properly, compromising both application performance and resource efficiency. To tackle these issues, we propose a History-Based Resource Manager for bioinformatics workflows applications running on clusters with heterogeneous nodes. The proposed resource manager features a prediction model that generates multiple performance predictions for each job under different combinations of cluster resources. Furthermore, the proposed resource manager includes a scheduling algorithm that considers the degree of multiprogramming of the nodes, scheduling combinations of applications for simultaneous same-node execution upon their compatibility. To test the proposed resource manager, we process two workloads formed by different amounts of workflows made up by common bioinformatics applications. Results prove that for the given cases, the proposed resource manager improves the performance obtained with SLURM, using First Come First Served policy. The proposal shows an average workflow makespan improvement range between 28 and 35%, averaging 32%, an average workflow efficiency improvement range between 75 and 83%, averaging 79%, and an average resource usage improvement range between 96 and 101%, averaging 99%. Furthermore, the pro-

Funded by the Spanish Economy ministry. Project Number: TIN2014-53234-C2-1-R.

✉ Ferran Badosa
ferran.badosa@uab.cat

[1] Computer Architecture and Operative Systems Department, Universitat Autònoma de Barcelona, Bellaterra, Spain

[2] Centre for Research in Agricultural Genomics, Bellaterra, Spain

posed scheduling algorithm can improve the average workflow makespan by a range of values between 26 and 36%, averaging 31%, compared to Max–Min and Min–Min algorithms.

Keywords Resource manager · Bioinformatics workflows · Multivariate regression prediction · Scheduling algorithm · Resource sharing · Slowdown · Makespan

1 Introduction

Biologists and data analysts mainly employ bioinformatics applications to analyze genomic data. Many kinds of bioinformatics applications exist, performing different tasks involved in genome analysis, such as genome alignment, variant calling or annotation, among others. Genome aligners find the location of a sample sequence in the whole reference genome, by aligning the former to the latter. Variant callers look for differences between the aligned sample genome and the reference genome. Annotation applications review the differences or variants presented by the sample with respect to the reference in order to determine its biological significance. Some variants may convey valuable information about the analyzed organism, such as the development of certain diseases or response to drugs.

Users may choose among different applications conduct each of the multiple stages of genome analysis. Depending on the case of study, some applications provide better outcomes than others. To determine the most suitable ones for each case, past experiments performed by the bioinformatic community may be reviewed. Users rarely intervene in the development of bioinformatics applications. However, they may integrate multiple applications to carry out data analysis, building bioinformatics workflows. Galaxy [1] and Taverna [2] are two of the most popular frameworks for building and executing workflows [3]. Users may design workflows as they see fit for their analyses, combining multiple applications in parallel or sequentially. An example of a short bioinformatics workflow [4], with a single input dataset file, is depicted in Fig. 1. When workflows applications are executed in cluster nodes, they generate intermediate data files which may become the input of the next workflow application. Eventually, the output file containing the analyses results is generated.

In the present work, a list of popular Bioinformatics applications commonly found within workflows has been characterized. Many of these applications use complex algorithms that perform memory-intensive operations, such as memory requests. Usually, the amount of memory requests grows as larger amounts of execution threads are selected. For a relatively big number of threads, memory bandwidth may become a performance-limiting factor, for large amounts of memory accesses must be dealt with, while also keeping latencies low. Among the numerous genomics applications characterized in this work, focus has been cast on sequence mappers, which have been categorized as memory-bound.

One example of a memory-bound application is Blast, which compares pairs of sequences by resorting to the Needleman-Wunsch algorithm [5]. Blast compares each sequence in the reads file (containing fragments of the genome to be analyzed), with each sequence in the reference genome file of the same species (used as a template). For

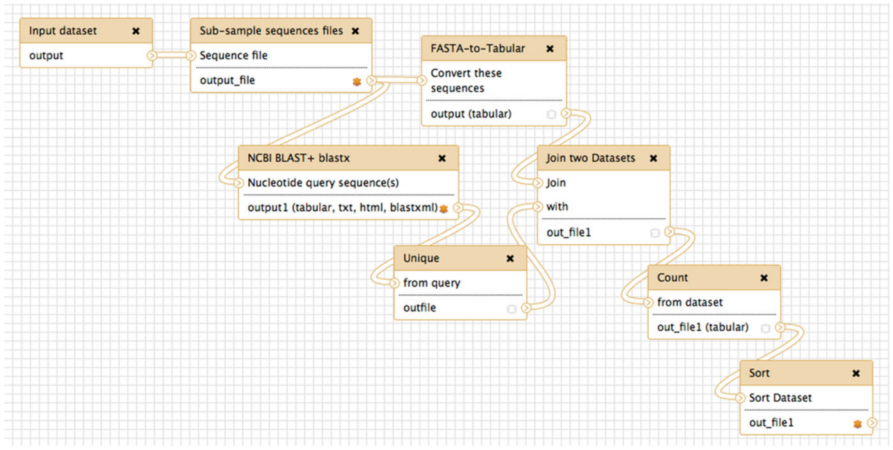

Fig. 1 Representation of a bioinformatics workflow used for genomic data analysis, with applications arranged sequentially and in parallel

each single pair of sequences, the algorithm must generate and fill a matrix of as many rows as the length of read sequence, and as many columns as the length of the reference sequence. Due to the reading and storing of matrices generated when comparing every reference-read pair of sequences, Needleman-Wunsch algorithm shows both time and space quadratic complexity [6].

From the point of view of the resources, multi-core CPU node clusters, have become popular platforms to execute bioinformatics workflows applications and perform genomic data analysis. When executing workflows, different performance goals may be sought by multiple parties, such as applications users and platform administrators. Users' performance criteria generally involves minimizing the execution makespan or cost, with the purpose of meeting deadlines or budget constraints. Conversely, platform administrators usually focus on a much different criteria, such as maximizing resource efficiency and usage. Users and platform criteria are detrimental to one another and usually conflict. To meet users' and platform's conditions in so as far as possible, cluster resources must be properly allocated, accounting for the characteristics and resource usage of applications. Allocation of applications to resources is a NP-hard problem, which has been long-analyzed in previous research [7,8]. Prior insight on the resources to be used by applications may become crucial to maximize performance.

The vast majority of HPC applications running in clusters are programmed in a distributed-memory paradigm, and executed in multiple processors of different nodes. To facilitate the communication between processes, external libraries such as MPI, are employed. In these cases, users are prompted to provide explicit description of the resources needed by the application. Thus, it is implied that before submission, previous knowledge of the resources to be used exists, and has been acquired by users. Hence, applications usually come along with at least an estimation of the amount of resources needed by applications.

Conversely, most bioinformatics applications are programmed in a shared-memory paradigm. Execution is carried out in a single node at a time, exploiting thread-level

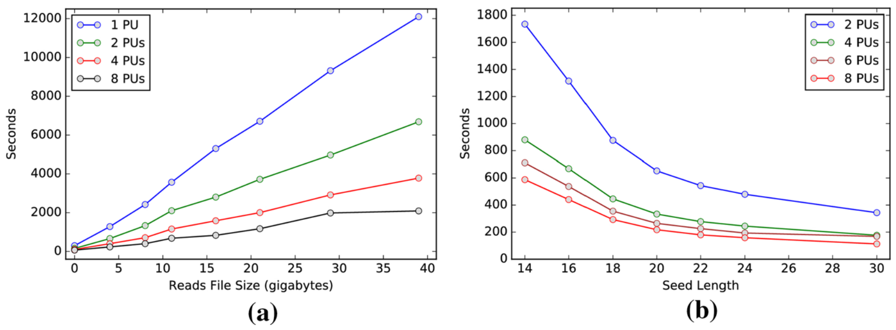

Fig. 2 Execution time variation of bioinformatics applications experienced when parameter values or data characteristics are modified. **a** Execution time variation of Hisat with different reads sizes. **b** Execution time variation of Bowtie with different seed length parameter values

parallelism. In these cases, before execution, it is convenient but not strictly necessary to provide description of the resources needed by the application. Entailing that, in most scenarios, little or no prior insight on the resources required by the application is available before execution. Furthermore, biologists or genomic data analysts, tend to choose resources based on the quality of the output or the kind of complexity of data analysis. As a result, resources are rarely selected based on applications' resource needs.

Determining the adequate resources for bioinformatics applications is a hard task, since the resource usage of a single application might vary substantially from one execution to the next. Variation depends on two factors: the values given to the configuration parameters of each application and the input dataset characteristics. The parameters determine the specifics of each experiment. Some analyses are intended to solely report high-quality similarities between the reference and the observed samples or fragments. These analyses call for deeper, more-thorough observations of the genomes than those others seeking lower-quality similarities. Finding high-quality similarities requires more exhaustive searches and more-complex algorithmic heuristics. Usually, these cases are much more-resource consuming, prompting execution times to lengthen substantially. The characteristics of the input datasets also have a great impact on resource usage and execution time, such as the files sizes or the number or length of the sequences.

Figure 2 shows the extent of execution time variation of two read mappers, Hisat and Bowtie, when modifying a single parameter or data characteristics. In Fig. 2a, Hisat is executed with different reads file sizes, ranging from 1 GB to 40 GB, each of which is mapped against the same human reference genome. In Fig. 2b, Bowtie is executed with identical input files, but different values of one of its parameters, the seed length. That is, the length of the subsequence within the target sequence. The seed length can be used to balance the commitment between sensitivity and speed. Shorter seed lengths yield high-quality alignments, but also require large execution times. As the seed length increases, lower-quality alignments are found, resulting in shorter execution times. Modifying the seed length parameter causes non-linear execution time variations.

Most clusters are shared among many jobs, which may have to wait in admission queues before being granted resources. As jobs run, chances are their performance is affected by a dynamic, varying-over-time workload. Unlike in the exclusive case, applications don't have full-node resources available, which generally ensure fast execution. Instead, they are likely to be allocated logical processing units of a node, referred to in this work as processing units (PUs), i.e. a node containing 8 cores with hyper-threading accounts for 16 PUs. In these cases, the number of applications simultaneously running in a node, referred to as Degree of Multiprogramming (DP), is greater than one. Sharing resources may significantly improve the efficiency and resource usage of applications. Nonetheless, it may have consequences on applications' makespans.

Applications running in the same node compete for the same resources and interfere on each other's execution, slowing execution times down compared with the exclusive cases. Applications' slowdown can be measured by comparing applications' makespan times when using node resources exclusively with those obtained when node resources are shared. The extent of slowdown depends not only on the DP of each node, but also on resource usage and characteristics of applications sharing the node. Applications bound by different resources that are running in the same node, such as memory-bound and CPU-bound, are more likely yield lower slowdown than applications bounded by the same resource.

When users of bioinformatics workflows submit their jobs to the cluster, face the challenge of determining which combination resources is more suitable for application performance (turnaround, cost), given the parameter values, dataset characteristics, and time-varying resource availability. Finding the proper combination of resources for each case is a hard and critical task, since it strongly affects the resulting performance.

The main goal of this work is to maximize the performance of bioinformatics workflow applications running on clusters with heterogeneous nodes. Given a series of submitted workflows, the objective is to determine the resources needed by each workflow application, so that average workflow makespan or computational cost are minimized, and average efficiency and resource usage maximized. To do so, focus is cast on two major lines. On one hand, exclusive-mode predictions of the resources needed to minimize makespan or cost while also maximizing efficiency and resource usage. On the other hand, minimization of slowdown spawned when multiple applications share the same node resources.

Current Resource Management Systems (RMS) in clusters, discussed in Sect. 2 don't account for the particularities of bioinformatics applications. These particularities cause them to have significantly different execution times or resource consumptions depending on the combination of parameter values and data characteristics. As a result, resource allocation doesn't adjust to the actual requirements of the applications, compromising application performance, resource efficiency, and the capacity of the cluster to analyze data. To adapt resource allocation of RMS to the particularities of bioinformatics applications, we introduce in this work a History-Based Resource Manager (RM) for bioinformatics workflows applications on clusters.

The proposed RM can be applied to many other kinds of applications with similar resource requirements. However, for the present study we have chosen to focus on the well-known context of bioinformatics applications with large datasets volumes, which represent a compelling example of a real data-processing domain.

The several steps followed to build the proposed RM are explained in Sect. 3. First, we characterized a set of relevant applications and analyzed its performance in a cluster, formed by 4 nodes described in Sect. 4. Second, based on performance information, we developed a Multivariate Regression Predictor. Given parameters and data of applications, the predictor generates multiple performance predictions with different resources. Third, we developed scheduling algorithm which is fed with performance predictions and slowdown information. The algorithm determines how to allocate resources maximizing average workflow makespan or cost, efficiency and resource usage. Moreover, the algorithm considers the DP of the nodes, determining which combinations of applications make best-possible candidates for same-node execution so that slowdown is minimized. In Sect. 4, the proposed RM is tested by processing two queues of workflows. In Sect. 4.1, testing and validation are carried out in a real-environment scenario. In Sect. 4.2, further testing is conducted in a simulated-environment scenario. Finally, the improvements achieved with the proposed RM are discussed.

2 Related Work

Bioinformatics workflows applications usually require large amounts of resources in order to perform complex and multi-step data analysis. Users must build sometimes-complex pipelines, which may include the processing of large datasets. To make the most of the resources and successfully run their experiments, users must properly configure the working environment. E.g., arranging necessary data or installing analysis tools. Several tools have been developed in order to facilitate users' creation and execution of workflows in large environments. Among the most relevant ones, web-based Galaxy, or GenePattern [9] can be found. Galaxy provides a framework for assisting users through the analysis steps that must be followed, and guidance to process the necessary data. Similarly, GenePattern provides access to over 150 tools for genomic analysis, emphasizing on reproducibility. In turn, OnlineHPC [10], provides access to HPC resources and uses Taverna as workflow engine, carrying along its wide range of functionalities. Although these tools are designed to run jobs on local systems by default, they can also be configured to run jobs on clusters nodes.

RMS allocate jobs to cluster resources, monitor the status of the system, and manage the submission queues, among other things. MAUI, Sun Grid Engine or SLURM, which features in approximately 60% of top 500 supercomputers, are among the most popular RMS. SLURM is composed multiple daemons such as slurmctld or slurmd. Slurmctld runs on the management node, acting as a central entity that monitors system status. It has numerous commands: squeue, sinfo or sacct, among others. Slurmd daemons run on computing nodes and carry out slurmd commands, such as launching, killing or monitoring jobs. SLURM also provides multiple plug-ins. Some of them are general-purpose and allow for basic functionalities, whereas others are optional, and provide extra features that adapt to platforms' needs. Examples are node selection plug-ins, used to determine the resources used for a job, or scheduler plug-ins, which decide how and when SLURM schedules jobs [11]. Among the optional features, one may find the SLURM-SPANK package for the development of such plug-ins.

However, despite all the numerous functionalities included in SLURM, we haven't found any accounting for the particular needs of bioinformatics applications. That is, their variable behavior, which yields highly different resource consumptions or execution times depending on parameter values and data characteristics.

Sharing resources may have consequences on jobs' performance, prompting them to wait in admission queues and subjecting them to a dynamic workload. Chances are multiple applications attempt to use the same resources, whose availability varies over time. Using prior knowledge of bioinformatics applications behavior with different parameters, data and resources, may help predict which resources best suit applications' performance and efficiency under different scenarios of cluster availability. With that previous knowledge, RMS may adjust scheduling decisions to both applications' needs and current resource status. The History-Based RM proposed in this work is aware of the resource needs of bioinformatics applications, and uses such awareness to improve their overall performance and efficiency. The RM developed in this work is able to operate alongside SLURM, and can act as an auxiliary block of it. The proposed RM assists SLURM in taking scheduling decisions, which are based on a performance Prediction Model and a scheduling algorithm.

Different prediction approaches exist in order to estimate the performance of jobs running in clusters. Previous research analyses have focused on predicting execution time of jobs [12,13], waiting time [14,15] and slowdown time of applications spawned when resources are shared [16].

Prediction techniques can be further classified into several categories: benchmarks, application code analysis techniques, parameter prediction techniques or simulation, among others. Benchmarks are employed to estimate the performance of a system, given a parameter of reference. Application code analysis techniques are used to predict the performance of applications. They require deep knowledge of the code of the application. Acquiring it becomes inefficient when dealing with various applications due to cost-effort issues. Moreover, they fail to consider the system's characteristics. Finally, parameter prediction techniques, predict the execution times of applications by using estimation models. These techniques account for the characteristics of applications, such as the parameters that have an impact on performance. Parameter prediction techniques are adequate for bioinformatics applications, since they are able to capture their variable behavior, highly influenced by parameters and datasets.

Parameter prediction techniques can apply analytical or statistical models. Analytical models [17,18] provide performance metrics like execution time. However, they predict these metrics by assuming the cluster has a determined, non-dynamic status. Statistical models employ time-series methods. They resort to the high time-correlation presented usually by systems' loads [19,20], and can be suitable to model workloads and waiting times of shared systems [21,22]. History-based learning approaches may also be used by statistical methods. These approaches consider both applications' characteristics and current status of the system [23] to predict the execution time. Statistical models can be further divided into two techniques. Categorization techniques, which account for the characteristics of applications, and instant-based learning techniques (IBL), which also take into account the status of resources. On one hand, IBL methods can be divided into meta-heuristics, such as data mining techniques or genetic algorithms. On the other hand, IBL methods can be divided into correlation models,

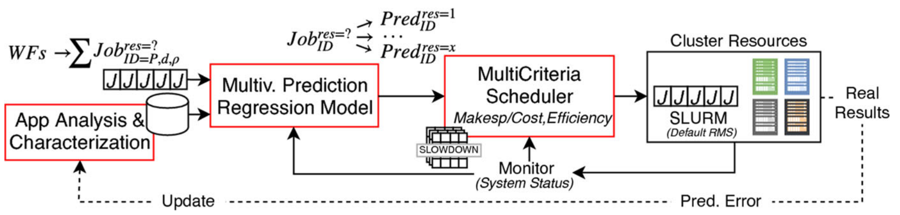

Fig. 3 Proposed RM (red), and its disposition with respect to the cluster's default RMS and nodes (Color figure online)

such as univariate or multivariate regression models. The RM developed in this work is based in a prediction model that generates predictions accounting for the dynamic characteristics of bioinformatics applications (parameters, data), and also resources. To do so, we considered appropriate to employ Multivariate Regression.

The proposed RM features a scheduling algorithm that schedules jobs based on performance predictions generated by a predictor. Moreover, it takes into consideration the fact that clusters have their own RMS, and is able to operate in compliance the cluster's default RMS, SLURM. In other words, the RM can be thought of as an extra module assisting SLURM to take improved scheduling decisions when dealing with bioinformatics workflow jobs.

3 Proposed Resource Manager

To improve the performance of bioinformatics applications, we developed a resource manager compatible with the cluster's default RMS, SLURM. The proposed RM is composed of three blocks or phases: Application Analysis and Characterization, Multivariate Regression Prediction, and Scheduling Multicriteria. The main blocks of the proposed RM can be seen in Fig. 3, as well as disposition with respect to the cluster's nodes and default RMS. Description of the mentioned blocks is provided in this section.

3.1 Application Performance Analysis

The purpose is for the resource manager to work with a wide range of bioinformatics applications, commonly featuring in workflows running in large platforms such as clusters. The first step to develop the resource manager consisted in building a representative set of bioinformatics applications to characterize and analyze. In order to pick the applications, several research articles focused on comparing multiple applications were reviewed [24–29]. As a result, a set of well-known applications with outstanding performance and different resource usage, shown in Table 1 was obtained.

Next, the main performance-determining characteristics of bioinformatics applications were listed, and broadly classified in four categories. The first category is formed by the configuration parameters of each application ($P = P_{1,\ldots,x}$). One example is Blast's hits parameter, which adjusts the desired quality of similarities to be found between the reference and the reads sequences. Another example is Blast's word size

Table 1 Set of representative workflows applications, built upon relevance criteria

Memory-bound							CPU-bound			
Blast	Bwa-mem	Bwa-aling	Bowtie	Hisat	Soap	Star	Phyml	Mrbayes	Fasttree	Raxml
2.6.0	0.7.5a	0.7.5a	2.2.6	2.0.5	2.21	2.4.2a	2.4.5par	3.1.2h	2.1.3.c	8.2.9

parameter, which states the length of the region of a sequence from which an exact match is searched (and later on extended). The second category is formed by the characteristics of the input datasets ($d = d_{1,\ldots,y}$) to be analyzed. The third one is the performance criteria of the application user (ρ: makespan, cost). The fourth and last one, are the resources used (res). The execution of bioinformatics applications on clusters is depicted in Eq. 1.

$$Job_{ID} = App^{res}_{P,d,\rho} = App^{node,PU,mem}_{P_{1,\ldots,x},d_{1,\ldots,y},\rho} \tag{1}$$

Once the set of applications and its main characteristics were defined, the performance analysis experiments were designed. With these experiments, performance information of applications under different conditions P, d, res, was obtained, and subsequently used to generate the prediction model.

Bioinformatics applications may be executed in up to millions of different combinations of P, d, res values. Applications may have tens of parameters with a range of values each, and can be fed with datasets of many different characteristics. The number of possible combinations of P, d, res values form a huge parametric space that is unfeasible to analyze or include in the experiments. However, the scientific community mainly focuses on a relevant subset of parameter values. Due to that, our approach doesn't evaluate all combinations, but follows a look-up table strategy instead. The results of previous executions are stored, since similar ones with resembling input parameter values and data are very likely to be submitted again in future workflow instances. When a new submission is received, similar past executions are looked up and used to predict the result of the new one. To deal with the immensity of the parametric space, we focused on the relevant parameters of the bioinformatics workflows applications. That is, parameters commonly used for the bioinformatics experiments published by the scientific community [30–32].

The performance experiments were carried out as follows: each application of the set was given ranges of P,d values. Next, for each combination of values P,d, applications were executed in all cluster nodes, with a range of PUs. As each experiment ran, the performance was monitored and stored in a historical database. The following metrics were tracked: makespan, cost, efficiency, memory (peak, average, accesses, page faults), CPU (instructions, cycles, user and kernel time), disk (data written/read per unit of time), and cache (last-level and all-level misses, and hit-rates).

3.2 Multivariate Regression Prediction Model

Even when repeating the same exact job execution with identical parameters, data characteristics, and resources, performance data such as makespan or cost may vary.

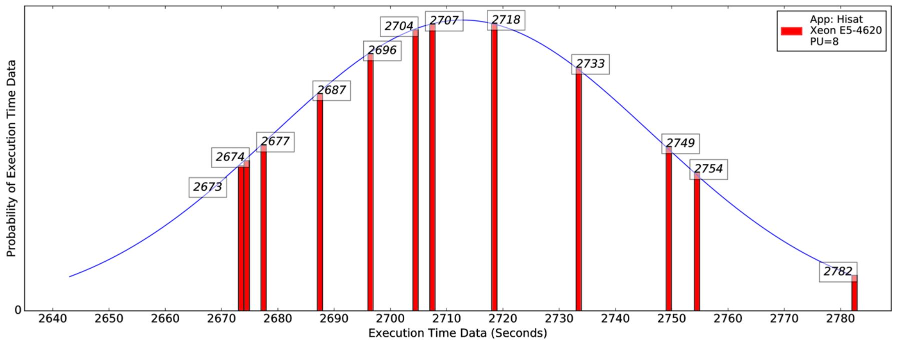

Fig. 4 Squared sample values (s), and resulting Probability Density Function

Prediction is only viable if performance data is inferential. Before building the prediction model, makespan data variation was analyzed to determine whether it is inferential or not. To do so, multiple makespan data samples were generated. Since the same conclusions were extracted from all the samples, and the same explanation applies for all of them, only one sample case is shown. Sample data was generated by executing 12 identical instances of Hisat application, with identical parameter values, data characteristics and resources. Sample values obtained are represented within squares in Fig. 4, ranging between 2673 and 2782, and averaging 2712.9.

The first analysis step consisted in determining whether makespan data is parametric. That is, whether it follows a normal distribution. To that end, the Shapiro–Wilk Normality Test was applied. A p value of 0.037, greater than 0.05, was obtained, implying that data is indeed Normally Distributed. Once proven so, the confidence interval was calculated with the One-Sample T-Test. For the given sample, lower and upper limits equaling 2691 and 2734 s were obtained, respectively. The width of the confidence interval (43 s), represents a tiny percentage (1.65%) of the sample mean, thus proving that makespan data is inferential, and the prediction model viable.

Application performance information, obtained as explained in Sect. 3.1, and stored in the historical database, was used to develop the prediction model. Thanks to prediction, when users submit news job to the cluster, represented as in Eq. 1, don't need to provide description of the resources $node = ?$, $mem = ?$, $PU = ?$ to be used. Instead, the proposed RM searches through the historical database for identical executions (same $App_{P,d,\rho}$ values) that may have been previously executed in the cluster. In case no matches are found, which is likely since up to millions of combinations of $App_{P,d,\rho}$ exist, similar executions are searched for. The predictor generates performance predictions (makespan, cost, efficiency, memory) and estimates the best combinations of resources to maximize application performance.

To develop the prediction model, the historical data stored in the database is statistically analyzed. Historical data includes many input variables. That is, different parameters and data characteristics. Some of these input variables have a stronger influence on the predicted variables. To determine which of the input variables have to be included in the predicted model, the Pearson Correlation Coefficients, stating the amount of linear correlation between variables, were calculated.

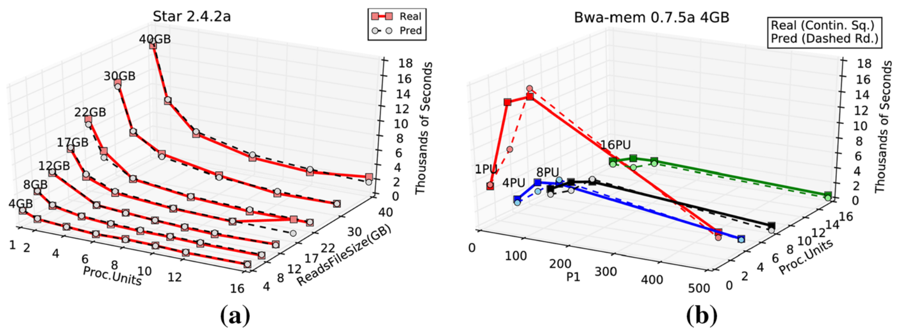

Fig. 5 Real and predicted makespans of Star and Bwa-mem applications, each in function of two predictor variables. **a** Real and predicted makespans of Star, in function of PUs and reads file size. **b** Real and predicted makespans of Bwa-mem, in function of P1 (Parameter 1, seed length), and PUs

When building the prediction equations, little prediction error is obviously sought. However, obtaining lesser prediction error when increasing the number of variables of a predictor doesn't always imply the prediction is better. Sometimes this phenomenon is due to the over-fitting of the model, which is the undesired modeling of the noise of data samples. Although little prediction error may be obtained on a sample, chances are the behavior of the sample doesn't represent that of the overall population of real observations. To prevent over-fitting, and evaluate the quality of the prediction model, the Adjusted R^2 coefficient has been calculated, as well as the relative prediction error.

Based on makespan predictions, cost predictions and resource efficiency predictions can be calculated. The execution cost was calculated by assigning fees to both the CPU model and peak memory consumed, by the hour of usage. Fares were estimated following the scale of Amazon's public EC2 instance pricing.

Figure 5, provides graphical representation of real and multivariate regression prediction results obtained for two execution scenarios. Figure 5a, shows the real and the predicted makespans of Star application in function of both the PUs and the reads file size, generating a mean relative error of 8.5%, and an Adjusted R^2 = 0.92. Figure 5b, shows the real and the predicted makespans of Bwa-mem application, in function of both the Parameter 1 (seed length) and the PUs. A mean relative error of 12%, and an Adjusted R^2 = 0.9, were obtained. As it can be observed from both graphical representations and assessment results, the prediction results remain close to the real results. For Star's case, the prediction Eq. 2 was generated, where: $\alpha = 3.6*10^{-7}$, β=1.03*10^{-8}, and γ=219.4. For Bwa-mem's case, the prediction Eq. 3 was generated, where: $\alpha = -0.88$, $\beta = 0.016$, $\gamma = -3.3 * 10^{-5}$, and $\delta = 8.6$.

$$PredTime = \alpha * (Size/PUs) + \beta * Size + \gamma \tag{2}$$

$$PredTime = e^{\alpha * ln(PUs) + \beta * P_1 + \gamma * P_1^2 + \delta} \tag{3}$$

In shared environments, multiple applications simultaneously attempt to use the same resources. Some of them may be allocated resources, as others wait for them to be released. Cluster efficiency has to be taken into account to prevent, among other things, over-allocation of resources, which would cause them go wasted as other jobs

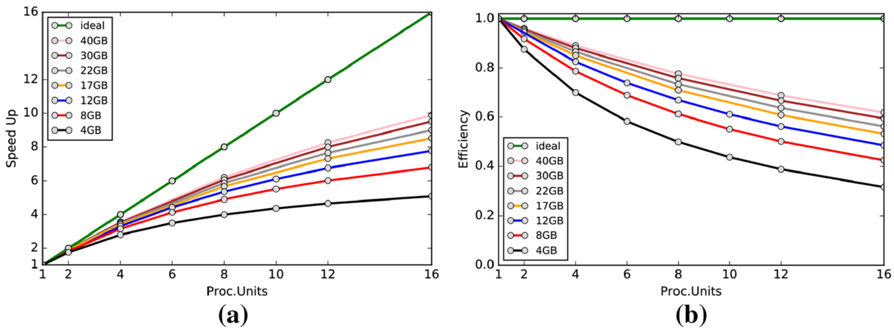

Fig. 6 Predicted speed ups and efficiencies for Star application. **a** Star's predicted speed ups. **b** Star's predicted efficiencies

wait. Furthermore, chances are jobs aren't able to reserve all the desired resources, such as the whole node. On top of that, jobs may not even get to run with as many PUs as the scalability threshold marks, $PU_{MaxSpeed}$, as they would in exclusive mode. Instead, they may have to be executed with less PU, i.e. $PU_{LowSpeed}$, yielding longer makespans (makespan penalty).

To properly deal with the trade-off between performance and the amount of resources used, speed up predictions and efficiency predictions are calculated. The speed ups and the efficiencies can be obtained from time predictions generated with multiple resources, such as a wide range of PUs in each node. Figure 6, shows the speed up and the efficiency predictions of Star, obtained from Fig. 5a. The scalability thresholds for these two applications with the given conditions and sets of reads file sizes is around 16 PUs. A range of efficiencies between 0.31 and 0.62, averaging 0.5, is obtained. Nonetheless, results show that by allocating them 12 PUs, little makespan penalty is inflicted and efficiency increased, whereas also leaving more resources available for other applications.

3.3 Scheduling Multicriteria

Read mappers like the ones of Table 1 manage huge amounts of genomic file sizes, calling for large memory requirements. Mappers mainly have two kinds of input files: the indexed genome reference files, usually of a size that fits into memory and the genomic sequence reads files, generally of a much greater size. Most read mappers follow the principles of Bwa's algorithm. That is: dividing reads files into chunks, and implementing a double buffer scheme. The scheme is implemented to hide chunk read latencies while distributing sequences within a chunk among threads for analysis (mapping) [33]. After studying genome mapping applications, it has been observed that the main bottleneck of their performance is caused by the saturation of the memory bandwidth, which becomes more and more noticeable as the number of threads is increased [34]. Read mapping applications are memory-bound, and performance limitations shown when increasing the number of concurrent execution threads cause them to have a low scalability [35–37]. As a result of the memory bandwidth saturation, mappers may be prompted to wait for memory requests to be solved. While waiting for memory, processing units remain low utilized, whereas other jobs can't access them. Thus, there is margin to improve application performance and resource efficiency by increasing the DP of the nodes. That is, increasing CPU usage. Due to that, we have focused on improving the resource management of these applications' main bottlenecks.

Multiple applications can be executed simultaneously in the same node (DP > 1) in order to increase resource efficiency. However, node sharing alters job performance, slowing makespans down compared with the exclusive-mode case. In this work, slowdown is quantified as the percentage of makespan increase of applications sharing the node, compared with their respective exclusive-mode makespans.

Slowdown depends not only on the node specifications, but also on the characteristics and resource usage carried out by applications. Applications with the same limiting resources are much more likely to increase resource competition and interfere on each other's performance than applications limited by different resources (i.e. memory-bound and CPU-bound). Different applications produce much lesser slowdown and are more suitable to share the same nodes.

In order to determine the amount of slowdown, a series of experiments were designed and conducted. Multiple combinations of applications were executed in the same node. The makespan of each application was tracked and compared with the exclusive makespan of the same application (with same parameters and data). Table 2, shows a sample of the slowdown information obtained among pairs of applications. To simplify, applications are executed with an amount of PUs matching their maximum speed up, numPU$_{MaxSpeed}$.

When increasing the DP of the nodes, the average slowdown increases too. Augmenting the DP is beneficial as long as the slowdown increase remains low, since it improves overall performance of applications, resource efficiency, and usage. However, nodes have a maximum DP, a point from which augmenting the number of applications is no longer efficient or reasonable, bringing about an abrupt slowdown increase.

The maximum DP supported by a node depends on the applications, the PUs selected, and node characteristics. Finding the maximum DP for each execution case is a complex issue beyond the scope of this work. For the present study, a DP $= 2$ has been chosen, since it allows us to show the improvements of the proposal. In order to reflect the complexity of calculating the maximum DP, a summarized example, which can be extrapolated to any other case, is provided below. The example has been conducted by selecting 8 applications of Table 1, a specific number of PUs for each application (numPU$_{MaxSpeed}$), and a cluster node. The first step consists in simultaneously executing all possible combinations of 2 applications (DP $= 2$) and calculating the average slowdowns. Next, the same average slowdown calculation is conducted with all combinations of some other amounts of applications. E.g.: 3 applications (DP $= 3$), 4 applications (DP $= 4$), and all 8 applications (DP $= 8$). Figure 7 shows the slowdown evolution as the number of applications (DP = 2, DP = 3, DP = 4 and DP = 8) increased. For the given example ($Apps_{P,d}$ and resources), increasing the DP from 2 to 3, and 3 to 4, augments the slowdown by 1.4% and 4%, respectively. Results indicate that for this particular example, 4 applications can be simultaneously

Table 2 Slowdowns obtained when running top-row alongside left-column applications

	Blast	BwaM	Bowtie	BwaA	Hisat	Star	Soap	Phyml	Mrbayes	Fasttree	Raxml
Blast	8%	%	%	%	%	%	%	%	%	%	%
BwaM	18%	2%	%	%	%	%	%	%	%	%	%
Bowtie	6%	12%	10%	%	%	%	%	%	%	%	%
BwaA	16%	23%	21%	3%	%	%	%	%	%	%	%
Hisat	18%	2%	16%	18%	8%	%	%	%	%	%	%
Star	21%	19%	30%	6%	18%	31%	%	%	%	%	%
Soap	6%	3%	15%	2%	6%	13%	10%	%	%	%	%
Phyml	2%	6%	7%	11%	3%	1%	0.2%	12%	%	%	%
Mrbayes	3%	6%	2%	3%	3%	5%	4%	27%	6%	%	%
Fasttree	5%	1%	2%	0.5%	7%	0%	0.5%	21%	4%	6%	%
Raxml	3%	3%	4%	2%	4%	7%	1%	16%	4%	6%	8%

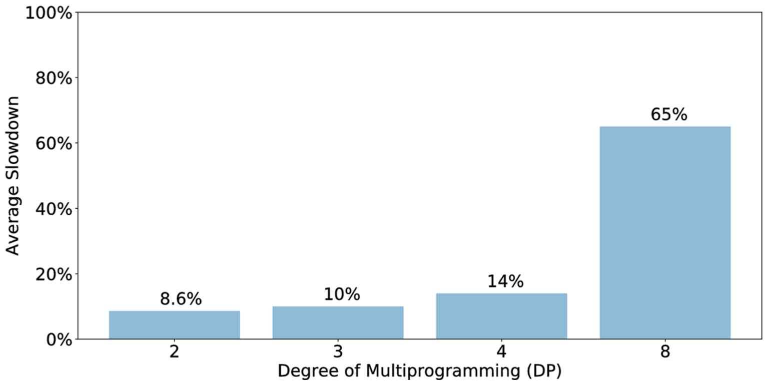

Fig. 7 Average slowdowns when increasing the DP, for a given set of applications and resources

executed in the same node with little consequences on performance. Conversely, when increasing the DP from 4 to 8, slowdown sharply rises up to 65%.

As previously mentioned, for the experiments of Sect. 4, involving more applications and nodes, DP = 2 has been chosen. This way, a clearer explanation on how the DP affects slowdown and overall performance can be provided.

To maximize application performance and resource efficiency, a scheduling algorithm, shown in Algorithm 1, has been developed. It is important to point out that the algorithm doesn't dismiss the dependencies of applications but receives ready ones. That is, applications whose dependencies have been previously analyzed and solved.

When submitting a list of applications to a cluster featuring the proposed RM, applications go through the prediction and scheduling steps. Detailed explanation of these steps is provided with an example. To generate the example, a sample of 6 jobs from Table 3 has been extracted and scheduled in a cluster, using a single node to simplify. Graphical representation of the explanation is provided in Fig. 8, and followed from left to right. The submitted list of applications to schedule (LApps=$J_{ID=1,\ldots,6}^{res=?}$, where each $J_{ID}^{res=?} = App_{P,d,\rho}$) is sent to the predictor, which takes the following inputs: the application (App), relevant parameter values (P), data characteristics (d), and user criteria (ρ, time). E.g.: *App=Blast*, $P =<$ *Hits*, *Word Size*, *Allow Gaps*, *MisMatch Penalty*, *Match Reward*, ... $>$, $d =<$ *File Size*, *Num Sequences*, *Length*, *format*, ... $>$, $\rho = time$. The predictor goes through the historical database, looking for past executions with similar inputs. Based on the similar past executions, the predictor generates time predictions with different combinations of resources, $res = 1, \ldots, 3$. Resource usage predictions, stemming from the classification of applications into CPU-bound and memory-bound, are also generated to consider the slowdown when the DP>1. As a result, a list of time predictions (LPred) and list of compatibility slowdowns (LCompat), are obtained. LPred and LCompat are in turn the input data of the scheduling Algorithm 1. With these inputs, the algorithm sorts jobs by descending time (top black row). In turn, for each time-sorted job, the algorithm sorts other jobs by ascending slowdown (gray rows), generating a list of priority-sorted jobs (LPrio). Once LPrio is generated, jobs are allocated. At the right hand side of Fig. 8, the algorithm steps are provided. At the

Table 3 Detailed information on the list of workflow applications processed in the cluster to test the proposed RM: average predicted makespans (in seconds) with $numPU_{MaxSpeed}$ in each node, and mean relative prediction errors

		Adm	Blast	BwaM	Bowt.	BwaA	Hisat	Star	Soap	Phyml	Mrbay.	Fasttr.	Raxml
Case A	Case B	WF_1	3640	3313	–	4136	–	–	2921	1622	3079	3000	2767
		WF_2	–	–	2934	5688	864	–	4021	3068	3510	1985	–
		WF_3	–	4576	3681	–	1220	1207	5362	–	–	3265	3443
		WF_4	7021	–	5420	8346	1966	1572	–	3753	–	6508	4099
		WF_5	4787	4015	4255	5102	2418	6791	4534	3970	3391	3056	3197
		WF_6	3040	6254	4500	4167	3567	5615	4129	3826	2988	3719	2699
		MRE	8.1%	8.6%	9.2%	10.7%	8.4%	10%	9%	9.3%	10.5%	9.4%	9.6%

Case A processes 4 workflows (WF_1, WF_2, WF_3WF_4), and case B processes 6 workflows (WF_1, WF_2, WF_3WF_4, WF_5, WF_6)

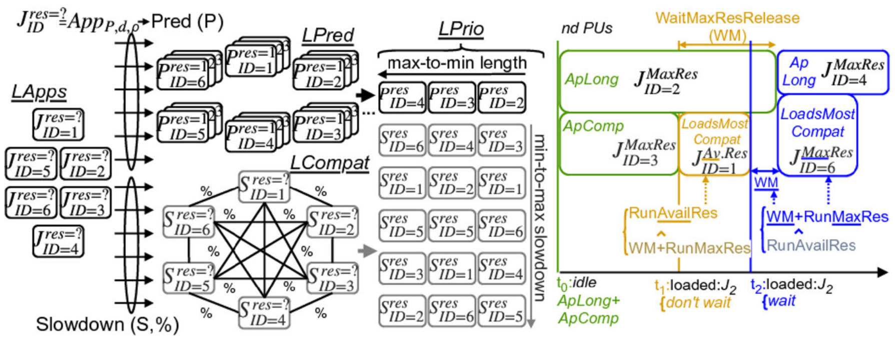

Fig. 8 Steps followed by a sample queue of jobs processed by the proposed RM, from submission to scheduling, focusing in one node

beginning of the simulation, t_0, the node is assumed to be idle. Thus, as the algorithm dictates, the longest application in Lapps (ApLong), plus ApLong's most compatible application (ApComp), are scheduled together. At t_1, the node is not idle but loaded, since J_2 is running. At this point, the algorithm selects the load's most compatible application (LoadsMostCompat, J_1), and compares RunAvailRes (line 27 of Algorithm 1), with Wait&RunMaxRes (line 26 of Algorithm 1). Where RunAvailRes is J_1's predicted time running at once with the resources currently available (AvailRes). And Wait&RunMaxRes is the sum of J_1's predicted waiting time in order for the J_1's maximum resources (that is, J_1's $numPU_{MaxSpeed}$) to be released (WaitMaxRes-Release), plus J_1's predicted running time with the so-called maximum resources (MaxRes). For the given example, the fastest option is assumed to be running with at once with AvailRes. At t_2, the node is also loaded with J_2, and the same comparison is conducted. Conversely, in this case, waiting for MaxRes release plus waiting with MaxRes (Wait&RunMaxRes) is assumed to be faster and thus the preferred option.

4 Experiments

The performance of the proposed RM is evaluated by conducting a series of experiments. In Sect. 4.1, the experiments are carried out in a real environment and compared with SLURM's FCFS (First Come First Served). The steps followed by the workflows applications under the proposed RM are covered in depth. Once the performance of the RM is validated in a real environment, simulated-environment experiments follow in Sect. 4.2. In the simulator, the proposed RM is tested once more, and its performance is compared with those of Max–Min and Min–Min scheduling algorithms [38].

4.1 Real-Environment Experiments

The proposed RM is tested by simultaneously processing two different sets of bioinformatics workflows applications in a cluster featuring SLURM as a default RMS. Each set of workflows (WFs$_A$ for case A, and WFs$_B$ for case B) is processed in two different

Algorithm 1: Scheduling Algorithm

```
Inputs : LApps: List of Applications to Schedule;
         LPred: List of Time Predictions of Applications in Exclusive Mode, for PUs in node, for app in LApps;
         LCompat: List of Compatibility Slowdowns of Applications, for app in LApps;
Output: LPrio: List of Applications Sorted by Priority;
1  Read LApps;    // Read predictions of apps in LApps with different PUs, nodes,
   from LPred
2  while apps in LApps do
3      Sort List of Applications to Schedule (LApps) by Time Descending;      // Longest to Shortest
4      ApLong = First in LApps;
5      Sort List of Compatibility Slowdowns of Applications (LCompat) for ApLong by Slowdown Descending;
       // Compat apps of each app
6      ApComp = First in LCompat for ApLong ;                    // ApLong's most compat app
7      SelectedApps = ApLong & ApComp;
8      MaxRes = PUs for Max SpeedUp for app in SelectedApps ;          // Obtained from LPred
9      Read Resource Status;
10     if IdleNodes in cluster then
11         if AvailRes > MaxRes then
12             SelectResources = MaxRes for SelectedApps;
13         end
14         else
15             SelectResources = Combination of PUs for SelectedApps minimizing sum of LPred times;
16         end
17         UpdateLists(SelectedApps);
18     end
19     if LoadedNodes in cluster then
20         SelectedApps = LoadsMostCompatApp (for app in LApps) ;     // Get app in LApps most
           Compat with node's Load
21         if AvailRes > MaxRes(SelectedApps) then
22             SelectResources = MaxRes for SelectedApps;
23         end
24         else
25             WaitMaxResRelease = (PredTimeApp – CurrentTimeApp) for App in Load;
26             Wait&RunMaxRes = WaitMaxResRelease + PredTime running with MaxRes;        // Wait
               MaxResRelease, run with MaxRes
27             RunAvailRes = PredTime running with AvailRes;     // Run at once with AvailRes
28             if RunAvailRes < Wait&RunMaxRes then
29                 SelectResources = AvailRes for SelectedApps;
30             end
31             else
32                 SelectResources = MaxRes for SelectedApps;
33             end
34         end
35         UpdateLists(SelectedApps);
36     end
37 end
38 return List of Applications sorted by Priority (LPrio);
39 Function UpdateLists (SelectedApps)
40     Remove SelectedApps from LApps;
41     Add SelectedApps to LPrio;
42     Go to Next App;
```

ways. First, by using both the proposed RM and SLURM. That is, with applications going through the prediction and scheduling phases described in Sect. 3. Second, by using only SLURM with FCFS policy. At the end, the makespan, the efficiency and the resource usage of the two different methods are reviewed and compared.

The performance results obtained when scheduling applications with the proposed RM vary upon the workload processed. The number of applications, their characteristics, and their compatibilities, play a major role in the ensuing makespan, efficiency, and resource usage. To consider that fact, two different workloads (WFs$_A$ and WFs$_B$)

are processed. The main purposes of this section are to assess how the proposed RM processes jobs, as well as to prove that it can improve SLURM's average workflow makespan, efficiency, and resource usage, under different workloads.

The efficiency of applications in exclusive mode has been calculated as the speed up divided by the number of processors, as in Eq. 4. Similarly, the efficiency of multiple applications in shared mode, i.e. App$_A$ and App$_B$ executed simultaneously, has been calculated as in Eq. 5. The resource usage has been calculated as the ratio between: the number of node PUs used by an application (or aggregated number of PUs used by multiple applications, if in shared mode) and the number of PUs of the node.

$$Efficiency(App)^{nPU}_{Exclusive} = \frac{Makespan(App^{1PU})}{Makespan(App^{nPU}) * n} \quad (4)$$

$$Efficiency\left(App^{nPU}_{A}, App^{mPU}_{B}\right)_{Shared} = \frac{Makespan(App^{1PU}_{A})_{Excl.} + Makespan(App^{1PU}_{B})_{Excl.}}{Makespan(App^{nPU}_{A}, App^{mPU}_{B})_{Shared} * (n+m)} \quad (5)$$

Clusters may be shared among many kinds of applications, such as MPI, MapReduce, or multithread. Thus, it is convenient to define partitions upon applications' resource needs. For data-intensive, shared-memory bioinformatics workflows applications, a partition of 4 heterogeneous nodes with large memory capacities has been defined: AMD IO-6376 (2.3GHz, 64PU, 128GB), Intel Xeon E5-4620 (2.2GHz, 64PU, 128GB), and 2 Intel Xeon E5-2620 (2.1GHz, 24PU, 64GB).

Each case-experiment begins by submitting multiple workflows to the cluster partition. For each workflow application, the predictor generates multiple performance predictions, such as makespans and costs, in all nodes with a wide range of PUs. Based on this information multiple metrics can be calculated: the fastest resources for each application (regardless of the resources available), the scalability threshold ($numPU_{MaxSpeed}$), the time penalties obtained when using less PU ($numPU_{LowSpeed}$), or the resource efficiency. With this information, the makespan-efficiency trade-off may be balanced.

Workflows may be submitted dynamically to the cluster. The submission rate and computing requirements of the jobs may exceed the capacity of the cluster, saturating it. In some of these cases, the cluster load may surpass by far the cluster capacity, prompting waiting times to grow unreasonably long. To prevent that, the cluster submission queue is divided into processing batches, and some of them are granted admission. To define the batch size, the aggregated capacities of the main resources of the cluster used by applications, memory and PUs, are taken as reference. To determine how many applications fit in each batch, performance predictions are consulted. Thus, a batch is going to be formed by as many applications whose predicted aggregated memory and $PU_{MaxSpeed}$ can be attended to with the cluster's: PUs ($\sum_{Node=1}^{n} numPUs_{Node}$) and memory ($\sum_{Node=1}^{n} Mem_{Node}$), given a cluster of n nodes. Next, a certain amount of batches are selected for admission. The present experiments involve two different workloads which are processed independently. For cases A and B, 2 and 4 batches have been selected, respectively. These amounts have been chosen since they contain

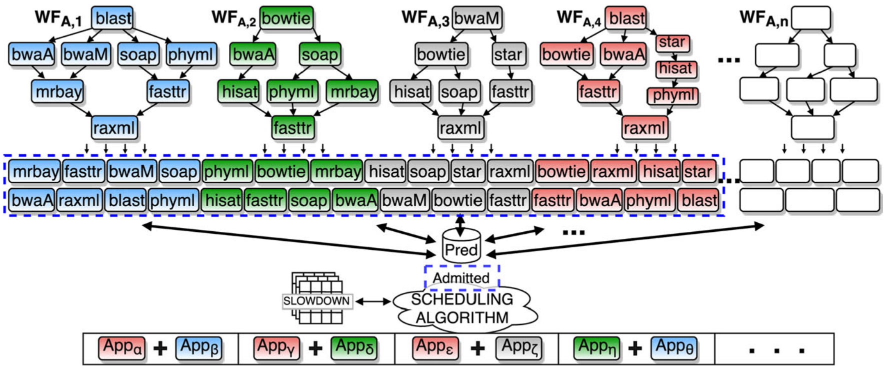

Fig. 9 From top to bottom (case A): (i) dynamically-submitted workflows, (ii) list of ready applications (admitted ones within blue-dashed rectangle), (iii) applying the scheduling algorithm that mixes up applications based on predictions and slowdowns, (iv) list of scheduled applications, sorted by priority (Color figure online)

enough load to explain how the proposal works, recreating different, real-like scenarios where load's computational requirements surpass the capacity of the cluster. Different sets of bioinformatics workflows are extracted from the admitted batches. For case A, the WFs$_A$ set is obtained. For case B, the WFs$_B$ set is obtained. In turn, from each workflow, a list of applications is obtained. Figure 9, upper part, shows case-A, admitted workflows (WF$_{A,1}$, WF$_{A,2}$, WF$_{A,3}$ and WF$_{A,4}$), and the resulting set of admitted applications (within blue-dashed rectangle). For the present work, the dependencies have been ruled out. The proposed RM comes into action assuming applications to be in ready state. So has been done to focus on the main point of this section, which is to expose the details on how scheduling based on predictions and compatibilities has been carried out, and how performance has been improved. Furthermore, this stand has been taken since SLURM, the RMS compatible with the proposed RM, already includes a dependency-solving flag *–dependency, -d*. This flag delays dependent jobs' execution start until their dependencies are solved, based on different criteria. Dependent jobs can start execution after specific jobs have begun *-d after*, terminated *-d afterany* or terminated in a failed state *-d afternotok*, among other options [11].

Once predictions are generated and the list of admitted applications is defined, the processing of applications continues, as shown in the lower part of Fig. 9, for case A. The performance predictions and slowdown information of each application in the list are sent to the Scheduler Multicriteria, and a list of priority-sorted applications is obtained: (App$_\alpha$ + App$_\beta$), (App$_\gamma$ + App$_\delta$), and so on.

The prediction information generated from the lists of applications (cases A and B), is provided in Table 3. To simplify the explanation, only the average predicted makespan (in seconds) of each application in all cluster nodes with $PU_{MaxSpeed}$, is shown.

For cases A and B, the scheduling algorithm receives makespans predictions, summarized in Table 3 and slowdown information, summarized in Table 2. Predictions reveal the amount of resources for both fast and efficient execution. Compatibility

Table 4 Workflows makespans (in seconds), efficiencies and resource usages obtained when processing the 4 workflows of case A, with the proposed RM alongside SLURM, and only SLURM

	Makespans case A			Efficiencies case A			Resource usages A		
	RM	SLURM	Improv.	RM	SLURM	Improv. (%)	RM (%)	SLURM (%)	Improv. (%)
WF_1	17,959	23,057	22	0.43	0.25	73	99	49	102
WF_2	18,243	23,589	23	0.52	0.30	75	88	46	91
WF_3	18,175	25,963	30	0.52	0.28	85	80	44	82
WF_4	17,959	27,958	36	0.52	0.31	68	87	36	138
Av.	18,084	25,141	28	0.50	0.29	75	88	43	101

slowdown information is employed to manage the DP of the nodes. This way, applications can be combined for same-node execution in such a way that slowdown is reduced as much as possible.

As mentioned before, cases A and B are processed in the cluster in two different ways: by using the proposed RM alongside SLURM, and by using only SLURM. The experiments are performed in the following order. First, case A, which contains 4 workflows or 30 applications, is processed in both ways. Once case A finishes, case B, involving 6 workflows or 52 applications, is processed in both ways too.

Based on applications' performance information in both shared and exclusive mode, the proposed RM outputs a list of applications and resources sorted by priority, which is sent to the cluster's default RMS, SLURM. The proposed RM acts then as an auxiliary module of SLURM, providing resource scheduling information that is based on a thorough study and knowledge of applications' characteristics. Assisted by the information sent by the proposed RM, and based on its own implemented policies, SLURM eventually schedules applications.

Results obtained after processing the 4 workflows of case A, in both ways described, are provided in Table 4. Three performance metrics: makespan, efficiency and resource usage, have been calculated. For the given case, employing the proposed RM (prediction, scheduling), improves average workflow makespan by up to 28%, average workflow efficiency by up to 75%, and average resource usage of workflows by up to 101%, compared to solely using SLURM.

Once case A finishes, the 6 workflows of case B are processed in the same both ways. Case-B results are provided in Table 5. For the given case, the proposed RM improves average workflow makespan by 35%, average workflow efficiency by 83%, and average resource usage of workflows by 95%, compared with SLURM.

The average makespan of the proposed RM increases from case A (18084 seconds) to case B (27,752 s) due to the saturation of the resources. However, RM's efficiency improves (from 0.5 in case A to 0.68 in case B), and so does RM's resource usage (from 88% in case A to 93% in case B). As it can be seen, the resource usage starts approaching its maximum as more and more applications are scheduled on the same amount of resources. This situation may overwhelm the resource capacity, increasing makespan. However, new applications included for the experiments of case B may generate new combinations of applications leading to improved slowdown and efficiency.

Table 5 Workflows makespans (in seconds), efficiencies and resource usages obtained when processing the 6 workflows of case B, with the proposed RM alongside SLURM, and only SLURM

	Makespans case B			Efficiencies case B			Resource usages B		
	RM	SLURM	Improv.	RM	SLURM	Improv. (%)	RM (%)	SLURM (%)	Improv. (%)
WF_1	24,762	39,005	37	0.64	0.31	108	99.5	52	91
WF_2	26,401	37,589	30	0.83	0.45	84	92	50	84
WF_3	27,125	40,212	33	0.61	0.33	87	82	47	76
WF_4	27,603	39,616	30	0.47	0.33	41	89	40	126
WF_5	31,011	52,546	41	0.78	0.43	81	95	49	94
WF_6	29,611	50,223	41	0.77	0.39	94	99.8	49	102
Av.	27,752	43,198	35	0.68	0.37	83	93	48	95.7

4.2 Simulated-Environment Experiments

The performance of the proposed RM is tested in a simulated environment and compared with that of Min–Min and Max–Min algorithms. To do so, the same sets workflows of the real-environment experiments of Sect. 4.1 are processed in a cluster. That is, WFs_A for case A, and WFs_B for case B. The simulation is conducted with WorkflowSim [39] simulator, an open-source tool widely accepted by the scientific community. WorkflowSim processes workflows defined within XML files and includes multiple scheduling policies, out of which relevant Max–Min and Min–Min, are chosen.

The mentioned sets of workflows are processed under three different scheduling algorithms (proposed RM, Max–Min and Min–Min). Since the three scheduling approaches require time predictions, they all have been equipped with the prediction model of the proposed RM. However, after going through the prediction phase, workflows are scheduled in different ways. First, with the proposed RM, that is, with applications going through the prediction and scheduling phases described in Sect. 3. Second, with the Min–Min algorithm, which calculates the overall completion time of all tasks and prioritizes the shortest one by assigning it to the node yielding minimum completion time. Third, with the Max–Min algorithm, which proceeds similarly to Min–Min, but sorting the tasks from maximum to minimum overall time. At the end, the average workflows makespans obtained with the three different methods are reviewed and compared.

The resource specifications of the simulated cluster designed for the experiments have been adjusted to be identical to those of the real cluster used for the previous experiments.

Similarly to the real-environment case exposed in Sect. 4.1, the simulation of the four workflows of the case A was firstly conducted. Table 6 shows the workflows' makespans obtained in seconds with the three methods once the simulation was terminated. Results show that applying the proposed scheduling algorithm to a list of workflows applications, considering the predicted makespans and the slowdown when the $DP>1$, can reduce by 26% the average workflow makespan obtained with both Min–Min and Min–Min algorithms.

Table 6 Workflows makespans (in seconds) obtained when simulating the processing the 4 workflows of case A, with the proposed scheduling algorithm, Max–Min algorithm and Min–Min algorithm

	Makespans case A			
	RM	Max–Min	Min–Min	Mean impr. (%)
WF_1	19,754	23,113	23,182	15
WF_2	18,790	23,342	29,356	29
WF_3	19,083	23,884	29,146	28
WF_4	19,395	26,850	30,853	33
Av.	19,256	24,297	28,134	26

Table 7 Workflows makespans (in seconds) obtained when simulating the processing the 6 workflows of case B, with the proposed scheduling algorithm, Max–Min algorithm, and Min–Min algorithm

	Makespans case B			
	RM	Max–Min	Min–Min	Mean impr. (%)
WF_1	26,991	41,740	20,933	14
WF_2	27,985	43,855	41,209	34
WF_3	29,837	46,196	48,071	37
WF_4	29,673	41,258	50,922	36
WF_5	33,147	41,348	51,684	29
WF_6	32,139	39,230	53,288	31
Av.	29,963	42,271	44,351	36

Finally, the execution of the 6 workflows of the case B was simulated too. The workflow makespans obtained can be seen in Table 7. For this second case, with a nearly-doubled amount of applications (from 30 in case A to 52 in case B), the proposed scheduling algorithm achieved an average workflow makespan reduction of 36% compared to Min–Min and Max–Min algorithms. The makespan improvement achieved in case B is 10% greater than that of case A.

The average workflow makespans obtained with the proposed RM in both real and simulated environments have been compared in order to verify that it is indeed feasible to convey the proposal into a simulated environment for further testing. By comparing the results, the time spent by the proposed RM in the pre-processing phase can also be determined. For case A, the pre-processing time accounts for 1172 s, which represents a 6% of the total processing time. For case B, involving more applications, the pre-processing time accounts for 2210 s. That is, a 7.4% of the processing time. As it can be observed, the amount of time spent in this phase depends on the number and complexity of the input applications, and the amount of available resources. In a context with many applications and limited amount of resources, finding a scheduling solution becomes more complex and thus more time-consuming than in contexts with many applications but scarce resources.

5 Conclusions and Future Work

In this work we developed a History-Based Resource Manager for bioinformatics workflows applications running in clusters with heterogeneous nodes. The RM is made up of three blocks: application performance analysis, prediction and scheduling.

The proposed RM becomes aware of the particular characteristics of bioinformatics workflows applications, which cause them to show variable performance depending on parameter values and data characteristics. By considering the characteristics of bioinformatics applications, we managed to adjust resource allocation to the actual needs of these applications. Also, we tested the performance of the proposed RM, by processing two sets of workflows applications, referred to as case A (4 workflows, 30 applications) and case B (6 workflows, 52 applications). First, we developed the experiments in a real cluster, and concluded that for the two workloads processed, attaching the proposed RM to SLURM improved makespan, efficiency and resource usage. Namely, in a real environment, average workflow makespan was improved by up to 36%, average workflow efficiency was improved by up to 83%, and average resource usage was improved by up to 101%, compared with solely using SLURM. Second, we tested the proposed RM in a simulated environment with WorkflowSim simulator. The obtained results showed that for both cases, the proposed scheduling algorithm achieved an average workflow makespan of 26% (case A), and 36% (case B), compared to Min–Min and Max–Min algorithms.

As future steps we consider conducting further tests with WorkflowSim simulator in order to enhance the functionalities of the proposed RM. Testing is to be carried out in larger environments and with workloads, which could open the door for the algorithm to be tested under greater DP and workloads. These testing conditions could help us extend or add new functionalities to the developed scheduling algorithm. Other future actions may involve adapting the current proposal so that it can be adapted to popular workflows managers such Galaxy or Taverna. Hence assisting towards to the resource managing stage, usually performed by SLURM, alongside which the proposed RM has proven beneficial.

References

1. Goecks, J., et al.: Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol. **11**(8), R86 (2010)
2. Wolstencroft, K., et al.: The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic Acids Res. **41.W1**, W557–W561 (2013)
3. Leipzig, J.: A review of bioinformatic pipeline frameworks. Brief. Bioinform. **18**(3), 530–536 (2017)
4. Cock, P.J.A., et al.: Galaxy tools and workflows for sequence analysis with applications in molecular plant pathology. PeerJ **1**, e167 (2013)
5. Needleman, S.B., et al.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. **48**(3), 443–453 (1970)
6. Feng, B., et al.: Distributed parallel Needleman–Wunsch algorithm on heterogeneous cluster system. In: International Conference on Network and Information Systems for Computers (ICNISC), pp. 358–361. IEEE (2015)
7. Calinescu, G., et al.: Improved approximation algorithms for resource allocation. In: International Conference on Integer Programming and Combinatorial Optimization, pp. 401–414. Springer (2002)

8. Chang, F., et al.: Optimal resource allocation in clouds. In: International Conference on Cloud Computing (CLOUD), pp. 418–425. IEEE (2010)
9. Reich, M., et al.: GenePattern 2.0. Nat. Genet. **38**(5), 500–501 (2006)
10. OnlineHPC Workflow Designer. http://www.onlinehpc.com (2012)
11. SLURM Workload Manager, Version 17.02. https://slurm.schedmd.com/job_array.html (2017)
12. Shanthini, J.: Anatomy study of execution time predictions in heterogeneous systems. Int. J. Comput. Appl. **45**(7), 39–43 (2012)
13. Seneviratne, S., et al.: A taxonomy of performance prediction systems in the parallel and distributed computing grids. arXiv preprint arXiv:1307.2380 (2013)
14. Murali, P., et al.: Qespera: an adaptive framework for prediction of queue waiting times in supercomputer systems. Concurr. Comput. Pract. Exp. **28**(9), 2685–2710 (2016)
15. Prodan, R.: Specification and runtime workflow support in the ASKALON Grid environment. Sci. Program. **15**(4), 193–211 (2007)
16. Figueira, S.M., et al.: A slowdown model for applications executing on time-shared clusters of workstations. IEEE Trans. Parallel Distrib. Syst. **12**(6), 653–670 (2001)
17. Seneviratne, S., Levy, D.: Enhanced host load prediction by division of user load signal for grid computing. J. Clust. Comput. (2005) **(submitted to)**
18. Seneviratne, S., et al.: Task profiling model for load profile prediction. Future Gener. Comput. Syst. **27**(3), 245–255 (2011)
19. Iosup, A., et al.: The grid workloads archive. Future Gener. Comput. Syst. **24**(7), 672–686 (2008)
20. Yang, L., et al.: Conservative scheduling: using predicted variance to improve scheduling decisions in dynamic environments. In: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, p. 31. ACM (2003)
21. Downey, A.B.: Predicting queue times on space-sharing parallel computers. In: 11th International Symposium on Parallel Processing, pp. 209–218. IEEE (1997)
22. Song, B., et al.: Parallel computer workload modeling with markov chains. In: Workshop on Job Scheduling Strategies for Parallel Processing, pp. 47–62. Springer (2004)
23. Christopher, A.: Locally weighted learning. Artif. Intell. Rev. **11**(1–5), 11–73 (1997)
24. Liu, K., et al.: RAxML and FastTree: comparing two methods for large-scale maximum likelihood phylogeny estimation. PLoS ONE **6**(11), e27731 (2011)
25. Borozan, I., et al.: Evaluation of alignment algorithms for discovery and identification of pathogens using RNA-Seq. PLoS ONE **8**(10), e76935 (2013)
26. Otto, C., et al.: Lacking alignments? The next-generation sequencing mapper segemehl revisited. Bioinformatics **30**(13), 1837–1843 (2014)
27. Rahman, F., et al.: benchNGS: an approach to benchmark short reads alignment tools. arXiv preprint arXiv:1504.06659 (2015)
28. Baruzzo, G., et al.: Simulation-based comprehensive benchmarking of RNA-seq aligners. Nat. Methods **14**(2), 135–139 (2017)
29. Warnow, T.: Large-scale multiple sequence alignment and phylogeny estimation. In: Chauve, C., El-Mabrouk, N., Tannier, E. (eds.) Models and Algorithms for Genome Evolution, pp. 85–146. Springer, London (2013)
30. Langmead, B.: Aligning short sequencing reads with Bowtie. Curr. Protoc. Bioinformatics. **32**(1), 11.7.1–11.7.14 (2010)
31. Li, H.: Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. arXiv preprint arXiv:1303.3997 (2013)
32. Ladunga, I.S.: Finding similar nucleotide sequences using network BLAST searches. Curr. Protoc. Bioinform. **26**, 3.3.1–3.3.26 (2009)
33. Herzeel, C., et al.: Resolving load balancing issues in BWA on NUMA multicore architectures. In: International Conference on Parallel Processing and Applied Mathematics, pp. 227–236. Springer (2013)
34. Herzeel, C., et al.: Performance analysis of BWA alignment. Technical Report Exascience Life Lab (2013)
35. Nelson, C., et al.: Shepard: a fast exact match short read aligner. In: 10th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE), pp. 91–94. IEEE (2012)
36. Olson, C.B., et al.: Hardware acceleration of short read mapping. In: 20th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 161–168. IEEE (2012)

37. Decap, D., et al.: Performance analysis of a parallel, multi-node pipeline for DNA sequencing. In: Parallel Processing and Applied Mathematics. Springer, pp. 233–242 (2016)
38. Maheswaran, M., et al.: Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: Eighth Heterogeneous Computing Workshop (HCW), pp. 30–44. IEEE (1999)
39. Chen, W., et al.: Workflowsim: a toolkit for simulating scientific workflows in distributed environments. In: 8th International Conference on E-science (e-science), pp. 1–8. IEEE (2012)