J Supercomput (2012) 59:1431–1454
DOI 10.1007/s11227-011-0555-y

Scheduling and planning job execution of loosely coupled applications

Enis Afgan · Purushotham Bangalore · Tibor Skala

Published online: 2 February 2011

Abstract Growth in availability of data collection devices has allowed individual researchers to gain access to large quantities of data that needs to be analyzed. As a result, many labs and departments have acquired considerable compute resources. However, effective and efficient utilization of those resources remains a barrier for the individual researchers because the distributed computing environments are difficult to understand and control. We introduce a methodology and a tool that automatically manipulates and understands job submission parameters to realize a range of job execution alternatives across a distributed compute infrastructure. Generated alternatives are presented to a user at the time of job submission in the form of tradeoffs mapped onto two conflicting objectives, namely job cost and runtime. Such presentation of job execution alternatives allows a user to immediately and quantitatively observe viable options regarding their job execution, and thus allows the user to interact with the environment at a true service level. Generated job execution alternatives have been tested through simulation and on real-world resources and, in both cases, the average accuracy of the runtime of the generated and perceived job alternatives is within 5%.

Keywords Distributed architectures · Load balancing and task assignment · Scheduling and task partitioning · User interfaces

E. Afgan (✉)
Department of Biology, Emory University, Atlanta, GA 30322, USA
e-mail: eafgan@emory.edu

P. Bangalore
Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL 35294, USA
e-mail: puri@cis.uab.edu

T. Skala
University of Zagreb, Getaldiceva 2, 10 000, Zagreb, Croatia
e-mail: tibor.skala@grf.hr

1 Introduction

Continuous advancement and acceptance of data collection devices such as sequencing instruments, observation networks, and even the social media has enabled an enormous amount of information to be generated or collected on a daily basis [1]. Availability of such volume of data is desirable because advances in research that were unthinkable just a few years ago are made possible. However, with the rise of the data volume, the demand for computational resources that process the collected data is also rising. Results that used to be collected and analyzed on researcher's laptop now require access to a dedicated workstation, a cluster, or even a collection of clusters.

At the same time, much of the data being collected comes from within domains that were traditionally not computationally intensive, such as biology, psychology, or sociology. Because of the fast pace of data collection and corresponding tool development, many of the scientists being forced to deal with the data deluge lack sufficient training in how to process given data and how to operate available compute resources. This is in contrast to the traditionally computational domains such as physics or mathematics where researchers have been developing their own tools and using computational resources for generations. Due to the mentioned lack of computational culture among domain scientists in the new computational domains, simply running a data analysis program represents a barrier, let alone managing and dealing with compute infrastructure capable of handling the available data.

Because of the described demand for computational resources, many departments and research labs have acquired considerable computational resources. This has led to dispersion of computational resources where many small pockets or hotspots with considerable computational capacity exist. Access to these resources is typically granted through command line tools, custom developed portals, or dedicated access points. Depending on the given institution's policies, access may be granted directly to individual machines or from a single location that distributes jobs across all of the available resources.

If direct access to individual resources is granted to an applied scientist, all of the available resources and choices regarding job submission might appear equivalent and inherent differences would not be recognized. User's selection of which resource to run the job on might be random or based on previous experience. In addition, once a routine has been established, even though the resource availability, input data, algorithms, or even the applications change, the user may always use the same resource(s) and/or job parameters. The resulting observation is that the available infrastructure does not meet users' expectations and results in underutilized resources in terms of both, cost and time [2].

Alternatively, distribution of user jobs can be realized automatically from a single access point through job scheduling. Job scheduling can be defined as a selection of appropriate mappings between resources and an application. The act of job scheduling is fundamental to the success of a distributed computational infrastructure and the computational process imposed on researchers because it alleviates users from many low-level technical details [3]. However, this scheduling process is complicated due to complexities introduced by resource availability, application and resource dependencies [4], as well as user goals and requirements. In order to deliver desired Quality of Service (QoS) to users and make efficient use of available resources, job scheduling abstractions need to be raised to the level where they can fulfill user's aspirations. The user experience has to be tailored to support individual users and provide them with the options they need and desire.

From a job scheduler's perspective, job scheduling can be viewed as a function of resource heterogeneity, resource and application availability, and application options [5]. In addition to these influencing factors, multiple, conflicting objectives can be incorporated into a scheduling decision. Examples of such objectives include timeliness, cost, reliability, security, and accuracy. Such scheduling approaches are termed multiobjective scheduling [6]. Due to the multidimensional exploration space, it is not possible to find a single, optimal solution that maximizes/minimizes all of the included factors. Current schedulers that implement multiobjective scheduling [7], automatically trade off individual objectives on users' behalf to choose one solution that strikes a balance between the objectives. Although such solutions hold significant value, they fall short of focusing on an individual user's requirements, which differ from other users and also evolve as a function of time.

For the sake of illustrating the potential and significance of this reasoning, we draw an analogy to an action that many people can easily relate to: the purchase of an airline ticket. When a customer wants to purchase a flight ticket, a popular option is to visit one of the online flight search engines and provide simple trip details (i.e., departure/destination and travel dates). The system then analyzes many possible routes, companies, layovers, etc. resulting in a self-contained list of travel alternatives. Such alternatives offer the user a choice regarding ticket cost, departure time, flight duration, layover location, and so on. In the end, the user can make informed decisions on customized choices that meet their current needs. For example, a person traveling on business with a tight schedule and set meeting times may choose the shortest flight or one that lands at a particular time, irrespective of the cost. On the other hand, if a person is traveling for leisure with their family, they may choose the most economical alternative and not be so concerned with other travel circumstances.

The solution presented in this paper realizes the same objectives for job scheduling and user job submission in a distributed compute environment; it incorporates notions behind multiobjective scheduling, but rather than automatically selecting one job execution option and acting on it, it presents a range of job execution options directly to the user before job submission. This approach *focuses on individual users and their current needs* instead of treating all users the same and continuously making the same decisions. Rather than being bogged down with the low-level operating details, the user is provided with high-level information of a job's execution options, allowing one to embrace the technology. The proposed approach represents an exciting solution to an end-user because the user is abstracted from architectural details of the infrastructure while enjoying maximum system flexibility and support.

In order to achieve such a solution, knowledge about a particular application and possible mappings and dependencies to underlying resources is needed. Such knowledge needs to be interpreted and the process automated, yielding a set of job execution alternatives. Beyond automating and hiding the underlying technological details, the job scheduling process needs to be presented to a user through a job submission interface that immediately and quantitatively influences the user. This is achieved by presenting the user with concrete values regarding their job, in metrics relevant to them directly (e.g., cost vs. runtime or accuracy vs. runtime) as opposed to technical terms (e.g., number of CPUs, amount of required memory).

In this paper, these goals are realized by developing a new job scheduler that operates in two steps:

1. At the low level, through automatic yet application-specific exploration and generation of possible job execution alternatives prior to a job's execution.
2. Through the user interaction module, which interprets derived alternatives and presents them to the user in a concise and clear manner, allowing the user to select the most appropriate option for executing the job.

The outcome of the proposed approach to job scheduling is a set of *job execution options* mapped onto conflicting objectives that are presented to a user for final selection. A single job execution option represents decomposition of a job into a set of tasks and a mapping of this set of tasks to a set of execution resources. Each option furthermore represents a distribution or allocation of job input requirements (e.g., input data, number of iterations), representing the job's workload, to selected resources in a manner that best meets selected resources' capabilities and minimizes load imbalance across employed tasks. Repeated generation of such job execution options leads to a *job execution space*. Effective presentation of this job execution space offers deep insight into job execution tradeoffs to an individual user.

Currently, the proposed method works with embarrassingly parallel applications due to their wide-spread use [8] as well as their overall suitability for distributed environments [9]. However, the same approach can be quite easily extended to other application types (see Sect. 6.3 for further discussion). Presented solution focuses on small, relatively static, and distributed environments where the given job scheduler is used as a primary method for accessing underlying compute resources. As a result, contention between multiple such job schedulers is alleviated and rapid changes in resource availability are minimized. The described usage scenario and applicable infrastructures are well suited for departments and research labs that have access to a pool of distributed compute resources. In particular, it is suitable for the quickly emerging domain-specific labs. The proposed method further contributes to the desire of individual labs to aggregate their resources into a common resource pool that may be shared between multiple labs while not imposing resource management responsibilities onto their users. Through the proposed method, although the underlying compute infrastructure is composed of multiple, distributed resources, users of the system do not need to concern themselves with which resource is currently available, which one has desired application available, or how is one resource different than the other—the proposed method alleviates users from this effort.

Overall, this paper realizes the following three general contributions, which are realized through a tool called OptionView:

1. *Two-way live communication between a user and the scheduler* is introduced, which fundamentally changes how job are submitted and how a user interacts with the infrastructure.
2. Adopted model allows *focus on individual users and individual jobs*, rather than a one-solution-fits-all approach that is currently prominent.

3. *Effective mapping of a job onto a set of resources*, which is achieved through understanding of resource capabilities in terms of selected application and provided data, followed by appropriate data distribution.

The rest of this paper is organized as follows. Section 2 provides further motivation for the problem and associates proposed work with other respective projects. Considered scheduling problem is presented in Sect. 3, while Sect. 4 presents current implementation and solutions. Section 5 shows results of proposed work in simulation as well as real world distributed environments. Section 6 presents some discussion about the results and the approach while Sect. 7 concludes the paper.

2 Motivation and related work

With the advent of distributed computing, a need for meta-level scheduling systems quickly became apparent [10]. In addition, because of the potential for high degree of system heterogeneity across encompassing systems, it also became evident that there is a need for application-specific scheduling approaches [4]. Such application-specific scheduling systems were able to target resource selection not only to meet, but also to maximize application's requirements and resource capabilities. The prime example of such an approach is the Application Level Scheduler (AppLeS) [11]. AppLeS, and the later developed AppLeS Parameter Sweep Templates (APST) [12], focused on scheduling parameter sweep applications with the goal of minimizing job's makespan. This was achieved by taking into account the resource suitability from the perspective of given application followed by generation of a job execution plan. However, one of the main difficulties that arose from experiences with the AppLeS scheduler and its application-specific structure was adaptability to new applications. The AppLeS scheduler was tightly integrated with the particular application, so adopting the scheduler to alternate applications was not trivial.

As part of the Grid Application Development Software (GRADS) project [13], the AppLeS scheduler was decoupled from the underlying application [14]. This enabled development of application-specific and resource-specific performance models that could be applied to the scheduling process, irrespective of the application itself. Another early project in this context was Nimrod-G [15]. Nimrod-G focused on scheduling parameter sweep applications. Similar to AppLeS, Nimrod-G performed application specific scheduling but the process was based on deadlines and an economy model.

Subsequent projects developed by the community took a different approach to job scheduling where by focusing on ease of use rather than strictly job performance [16–18]. Such schedulers focus on wide-spread applicability and simplicity of use by supporting the "submit and forget" methodology [19]. Although available schedulers provided a comprehensive job submission environment with a focus on runtime minimization, they require users to decide on job parallelization method prior to job's execution (e.g., data vs. task parallel), divide the input data as deems appropriate, and then submit necessary number of tasks, specifying each task's execution parameters.

Condor's classads [20] represent an approach to involve the user into the job submission and the scheduling process. Through classads, users can "communicate" with

Table 1 Categorized summary and a comparison of projects related and/or extended with work presented in this paper. (Symbols used: ✓ full support or intended usage; ≈ partial support or requires manual configuration to support given feature; ✗ no support)

	Application-oriented scheduling	Nimrod-G	Condor classads	Usability approaches	Workflow approaches	OptionView
Projects?	[11, 12, 14]	[15]	[20]	[16, 17, 18]	[21, 22, 23]	
Application-specific	✓	≈	✗	✗	✗	✓
Economy	✗	✓	≈	✗	≈	✓
User customization	✗	✓	✓	≈	✗	≈
Simplicity of use	✗	≈	✗	✓	✗	✓
Single objective optimization	≈	✓	✗	✗	✓	✗
Multiple objective optimization	✗	✗	✗	✗	✓	✓
Works for workflows	✗	≈	✗	✗	✓	✗

a scheduler and, in abstract terms, provide requirements for their jobs. However, use of classads assumes users are familiar with the demand their application impose on underlying infrastructure, correlations between multiple job submission requirements (e.g., input data size vs. number of CPUs vs. amount of memory required), and current resource availability.

Several workflow scheduling and execution projects have adopted a different approach and explore the notion of multicriteria scheduling [21–23]. These projects aim at providing a user with tradeoffs, including runtime vs. cost or runtime vs. accuracy. Such approach allows presentation of a range of QoS levels regarding job execution as opposed to uniformly minimizing job turnaround time. Work performed in [7] incorporates the multiobjective approach to job scheduling but focuses on parameter sweep applications exclusively. Although the execution of user tasks is fully automated, task submission is performed only for the number of tasks the user provides during job submission. Table 1 summarizes relevant projects and compares them to the work presented in this paper in an easy-to-compare presentation.

The work presented in this paper builds on the outcomes of the application-specific scheduling and adopts benefits of multiobjective scheduling. The application-specific scheduling is adopted to allow the necessary level of detail and customization for each individual job. In order to exploit derived benefits of application-specific scheduling, the multiobjective ranking of job options is presented to a user. This has the effect of transferring the impact of job parameters directly to the user but also not requiring the user to deal with such low-level details. As a result, the proposed approach differs from the original works [7, 12, 14, 15] greatly by providing a user with a set of execution alternatives as opposed to any single solution. The more recent works [21–23] are all workflow oriented as opposed to providing solutions for standalone jobs. In addition, the user-scheduler interaction presented in proposed work is not available in any other work. Lastly, a significant contribution of this work is the simplicity of use from the user's perspective; the user only needs to select an application to run and the job input data without needing to specify any other details required by the job parameterization process or the infrastructure.

3 Job execution planning problem

Because of the overall suitability for distributed computing environments [9], supported by a recent study showing wide spread use across national grids [8, 9], execution properties of the Embarrassingly Parallel (EP) class of applications deserve considerable attention and have thus been chosen as a focus of this work. The EP class of applications is composed of a class of applications whose computational load consists of executing the same application multiple times, with each instance operating on varying input properties and possibly varying input parameters [24]. In this context, input properties represent the workload, for example input data or number of iterations, and input parameters represent instance configuration options, for example, number of threads employed by the instance (e.g., [2]). The most significant characteristic of EP applications is that, once started, there is no communication between individual tasks.

Although executions of individual instances of EP application jobs are considered independent, systematic execution of those invites considerable technical difficulties when executed in distributed environments [5], including following considerations: resource heterogeneity, dynamic resource availability, input data distribution, task load balancing, and failure handling. In this context, we broadly define the EP application job scheduling as a problem of (1) selecting available resources for execution, (2) distributing the input data to meet resources' capabilities, and (3) assigning generated task instances to selected resources.

An instance of an EP application is referred to as an EP job J and it is represented by a set of tasks t_i that work toward a common goal: $J = \{t_1, t_2, \dots, t_n\}$. Because of the heterogeneity of distributed resources, individual tasks t_i comprising the job J are likely to exhibit heterogeneous runtime characteristics [5]. In the EP application execution model, a job is considered complete only after the longest running task has completed. Therefore, in order to achieve maximum performance for a job, load imbalance across tasks needs to be minimized while resource utilization is maximized. To achieve such job execution characteristics, factors that affect runtime characteristics of a task need to be understood. The following are factors affecting runtime characteristics of a task t_i [5]:

- d task input data
- r task execution resource
- p task invocation parameters

As a result, the runtime characteristics C of a task t_i are a function of the three factors:

$$C(t_i) = f(d, r, p) \tag{1}$$

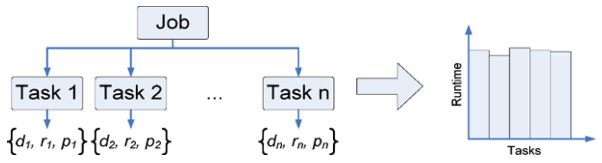
Understanding and controlling how these factors cumulatively affect task runtime characteristics is an example of task parameterization. *Task parameterization* is defined as understanding and selecting the task parameters (i.e., user controllable and application dependent options that can be changed when submitting a task, such as the number of threads employed or algorithm used) that are algorithm, input data, and resource dependent. Controlling individual tasks that comprise a job leads to a fine level of control of a job and thus the ability to realize desired objective from the perspective of a job. In other words, task parameterization leads to job parameterization. *Job parameterization* is then defined as coordination and control of individual tasks (and relevant factors) in such a fashion that a desired objective is realized (e.g., minimize runtime, maximize accuracy).

In order to achieve maximum job utilization given selected set of resources, the aim of job parameterization is to minimize the load imbalance across tasks comprising the job. For the case of static data distribution required by the proposed approach, achieving such a goal requires a scheduler to coordinate resource capabilities, match those to application's observed potential, realize appropriate data distribution, and finalize the process through individual task parameterization. The results of a job scheduling action is a job plan comprising of a set of heterogeneous tasks whose interactions and execution characteristics are simultaneously understood, balanced, and coordinated (please see Fig. 1).

The process of job parameterization can be formalized as follows: given a job J with a single input D of size $size(D)$, we may create n tasks $t_i \in J$ and $J = \{t_1, t_2, \ldots, t_n\}$ such that $size(D) = \sum_{i=1}^{n} size(t_i)$ holds ($size(t_i)$ is defined as size of the input assigned to a task t_i). Let R be the set of available resources. Each resource r_j, $1 \leq j \leq |R|$ has capacity $c(r_j)$. Although capacity of individual resources is considered application, data, and parameter dependent, in this discussion it is considered constant. Let E (where $E \subseteq R$) be a set of resources selected for executing job J. Function $resourceUtilization(J)$ then defines a policy stating that full capacity of each resource in E is consumed by the job J (e.g., if x CPUs are available on r_j, all x CPUs are consumed by the task assigned to the given resource). Furthermore, two functions are assumed existent, $estTime(t, r)$, which provides an estimate of runtime for a task t on a resource r, and $estCost(t, r)$, which provides an estimate of cost for executing task t on resource r. $loadImbalance(J)$ is defined as $stdDev(estTime(t_1, r_1), estTime(t_2, r_2), \ldots, estTime(t_n, r_n))$ is minimized, where r_i indicates a resource to which tasks t_1 through t_n are assigned.

A function $plan_J$ is then defined, which generates the set of tasks $\{t_1, t_2, \ldots, t_n\}$ and parameterizes each task t_i according to (1) so that $loadImbalance(J)$ is small

Fig. 1 Illustration of the job parameterization process aiming at minimizing load imbalance



and $resourceUtilization(J)$ is satisfied. Single execution of the function $plan_J$ provides a single job execution option o_l, $1 \leq l \leq m$ for executing given job J. $O = \{o_1, o_2, \ldots, o_n\}$ defines the job option space as a set of all execution options generated for the job J.

4 OptionView tool

In this section, we take a close look at the approach and the implementation details of work presented in this paper. The next four subsections, in order, describe the overall framework of OptionView tool, the job execution option selection process, the scheduling algorithm derived for generating selected options, and the subsequent mapping of derived options to absolute values that are presented to a user.

4.1 OptionView architecture

As illustrated in Fig. 2, a user initiates a job submission through a job submission interface (step 1); OptionView then analyzes job properties and resource availability leading to a job execution space (steps 2 through 7) that is presented back to the user (step 8). The user considers presented options and makes a selection as to which option to execute (step 9). The control is then transferred to a job submission engine for execution on appropriate resources (step 10). *Controller, scheduler, mapper* and *GUI generator* are the four major components comprising OptionView.

More specifically, the *controller* accepts a job submission request from the job submission interface and implements the logic controlling option generation. It acquires information from the Resource Information Services (RIS) such as Ganglia[1]

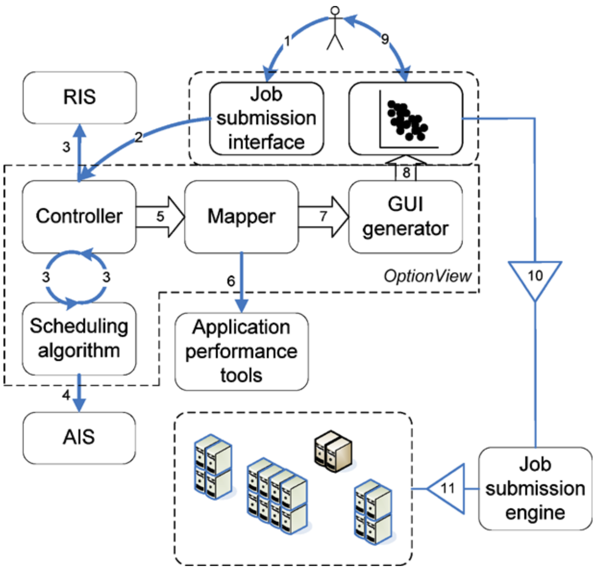


Fig. 2 High level OptionView architecture with numbers indicating general information progress flow

[1]http://ganglia.sourceforge.net/.

or Nagios[2] regarding resource state and availability. For each option the controller selects to generate, the *scheduler* is invoked. The scheduler acquires application-specific information required for effective scheduling from the Application Information Service (AIS) [25] and generates tasks comprising the given option. Upon all of the selected options have been generated, the *mapper* is called, which maps options' relative values to the absolute ones based on information from available tools (e.g., performance prediction tools [26], AIS [27], AppDB [28]). Lastly, the *GUI generator* presents the derived options to the user by mapping them onto conflicting objectives.

Because OptionView's focus is on individual users and their individual jobs, it represents user's workspace for access to the distributed resources. The workspace targets single user use and the implementation can either reside on user's local machine or, in the future, in a personal account within a portal. Also, note that with adopted approach, there is no need for job queues within OptionView as jobs are mapped only to available resources. For the case where no resources are available or where resource availability changes rapidly, proposed approach can be integrated with solutions for advance reservation (e.g., [29, 30]).

4.2 The controller

The *controller* is responsible for administering generation of job execution options. A job option represents a single parameterization of the job. Correspondingly, a single job option corresponds to a single execution of the function $plan_J$ defined in Sect. 3. Descriptively, a single job option is comprised of a set of resources and a CPU assignment across those resources as they are chosen among all of the available resources and resources' maximum processing core capacity. Furthermore, within each option, each selected resource is assigned a task that meets resource's capability and all tasks across selected resources have their workload distributed in such fashion that the overall load imbalance within a job option is minimized.

In order to initiate generation of job execution options, the controller's two main functions are: determining the range of possible parameters for options, followed by selection of which options to generate. Acceptable range of parameters for options is governed by the resource availability. Because individual options are primarily distinguished by the different selection of resources and assignment of CPUs across those resources, the total number of possible options is the product of idle CPUs across available resources. Note that here we make a simplification and use the number of CPUs on a resource as a measure of resource's size. For the case of computationally intensive applications of particular interest to us, as well as the application benchmarks used by the scheduling algorithm, we feel this is an acceptable simplification. For example, if three resources are available with 10, 15, and 8 CPUs available, the total number of options would be $(10 + 1) * (15 + 1) * (8 + 1) = 1584$ (note that 1 is added to the total number of CPUs on each resource to account for zero CPUs on each resource, i.e., not using the resource). From this simple example, it is obvious that the total number of possible options for all but nontrivial resource availability

[2] http://www.nagios.org/.

will be very large. However, because the total number of possible job options represents an exhaustive list of resource and CPU assignments, it is clear that such an approach is unnecessary and even unwanted. Rather, a subset of all possible options should be selected, processed and presented to the user. Nonetheless, such a subset needs to be representative of the overall set of options to allow for a comprehensive overview of the possible job execution options to the user.

Selection of the subset of options can be performed using either of two main statistical sampling principles: random or targeted [31]. A form of targeted search includes selection of options based on their quality with the solution yielding a Pareto line [6] of job execution options. However, because the details of the sample space (i.e., options) are unknown prior to generating the data points (i.e., options), the quality of individual data points cannot be calculated. Such an approach would thus require generation of all the options by invoking the scheduling algorithm followed by option ranking and selection. Due to the associated computational cost, this is not a feasible method. An alternate targeted search is systematic sampling [31], where resource- and application-specific knowledge is exploited to drive option exploration. In such a case, certain predetermined resource configurations could be explored, for example, use all idle CPUs on a resource, use half of idle CPUs on a resource, etc. Furthermore, application-specific information about scalability of the application (e.g., number of CPUs must be a power of 2, or maximum number of CPUs this application scales to is n) can be used to select option configurations. The benefit of such an approach is that these option configurations can be systematically chosen, requiring generation of only a small subset of all possible options. Furthermore, application-specific knowledge enables such targeted option selection to maximize job's performance. However, if needed application-specific information is not available, systematic sampling resorts to resource-based sample selection and can thus omit interesting data points.

In cases where application-specific information is lacking, simple random sampling method [31] can be used to perform option selection. From our observations (data now shown), the exhaustive list of job execution options is characterized by the normal distribution. Well established statistical methods [31] can thus be used to obtain high level of confidence about truthful representation of a sample compared to the overall sample space. To obtain the desired level of accuracy, a confidence level of 95% is used with a 5% confidence interval. These values can easily be adjusted by the user, but the presented values offer a suitable balance between required and presented number of sample data points. The calculated number of job execution options is randomly selected by the controller without replacement from the exhaustive list of job execution alternatives. Use of the sampling method results in a number of options selected to less than 400 regardless of resource availability and the total possible number of job execution options. This represents a manageable number of data points to be calculated by the scheduler.

4.3 Scheduling algorithm

The focus of the presented scheduling algorithm is to provide a job plan for an individual job execution option under the constraints of the function $plan_J$ described in

Sect. 3. Implementing the described function results in generation of a set of tasks that are mapped to selected resources, each directly corresponding to capabilities of assigned resource. Because the overall approach described in this paper requires generation of job options prior to job submission, given function $plan_J$ and the implementing algorithm focus on static scheduling and static data distribution. By definition of the $plan_J$ function, minimizing load imbalance among tasks to be created maximizes option performance and is thus the single most important goal of the scheduling algorithm. In the context of presented algorithm and selected application type, this is realized through relative comparison of available resources and assignment of appropriate data allocations to each resource. This results in cumulative minimization of load imbalance at the level of an individual job option.

Overall, when a job execution option data is presented to the metascheduling algorithm, the input data is decomposed by applying a static scheduling scheme for the specific application and matched to selected resources. As a result, a fraction λ_i of entire data is allocated to task t_i. The data decomposition is based on the performance information of current application on selected resources. The objective of the scheduling action is determining the workload distribution $\{\lambda_1, \lambda_2, \ldots, \lambda_i\}$ corresponding to a set of tasks $\{t_1, t_2, \ldots, t_i\}$.

Algorithm 1—Job option generation algorithm

```
1: Receive resource info $R$, job input size $D$, application info $A$
2: for $i = 1$ to $|R|$ do {$|R|$—number of resources}
3: $optionPerf[i]$ = adjustResourcePerfToApp ($R_i$, $A$);
4: $dData$ = divideData ($R$, $D$, $optionPerf$);
5: for $i = 0$ to $|R|$ do
6:     $task$ = new Task ($i$, $anOption$, $dData$, $resourceID$);
7:     $option$.add ($task$);
8: return $option$;
```

The pseudocode for the scheduling algorithm is provided in Algorithm 1 and it proceeds as follows: after receiving initialization data that consists of resource configurations, input data size and application name, the performance of resources in terms of current application is computed (lines 2 and 3). This strategy accounts for heterogeneity of individual resources, so that the amount of data λ_i assigned to resource R_i is proportional to the resource capacity. Equation (2) is used to perform the resource performance calculation:

$$optionPerf_i = n_i * a_i \tag{2}$$

where n_i is the number of processing elements on resource R_i and a_i is application performance metric, or weight, for the same resource. Calculation of such application performance across selected resources is implemented in a separate module, *adjustResourcePerfToApp* on line 3. This module interacts with AIS to obtain needed application- and resource-specific information. Alternatively, if needed information is not available within AIS, desired performance data can be obtained by directly running benchmarks on given resource, through a study of historical performance of the application and resource (e.g., [25, 28]), or by using the generic SPEC benchmark for the particular resource [32]. Such application-specific approach yields the devised algorithm (and the overall solution) an *application-specific solution*. On line 4, the data allocations $dData = \{\lambda_1, \lambda_2, \ldots, \lambda_i\}$ for corresponding resources are calculated. In the most general format, λ_i is computed as (n_i is the number of processing elements on resource R_i):

$$\lambda_i = optionPerf_i * \frac{D}{\sum_{i=1}^{n} n_i} \tag{3}$$

Depending on the application though, workload allocation process may differ from the simple one just shown and can thus be implemented as a plug-in to OptionView's scheduling algorithm by implementing *divideData* module. By extracting implementations of resource weight calculation and data distribution into separate modules, presented algorithm is made more versatile while providing support for application-specific scheduling [14]. With all of the information calculated and available, the second *for* loop iterates over the number of available resources, assembles information into a single task, and adds the task to the current option list. The generated option is then returned to the controller.

The devised algorithm presents an approach for minimizing load imbalance across tasks that is application-specific. Nonetheless, a generic version of the given algorithm could be used instead. It is left as part of future work to see what the outcome of such an effort would be.

4.4 User interaction module

The user interaction module consists of the mapper and the GUI generator and the approach implemented through the user interaction module aims at fundamentally changing the interaction mode between a job scheduler and a user. Most of the previous work in grid job scheduling area aims at automatically optimizing execution of user's jobs in terms of execution time, or cost, or both, e.g., [7]. Here, proposed work focuses on exploring and presenting a set of tradeoffs and concrete values regarding one's job before job execution begins. Through this model, the user is exposed to and is interacting at the true service level. Rather than being concerned with how many resources to select for execution, how many processors are being employed on each resource, or how much data is being transferred between hosts, a user is presented with discrete, quantitative metrics, for example, job execution time and associated cost. This represents a much more friendlier environment for non-computational savvy users because they do not have to specify low-level infrastructure details that may all appear equivalent where subtle differences are not recognized. To accommodate such interaction mode, the user interaction module maps generated job execution options onto concrete metrics of direct benefit to the user.

The *mapper* is concerned with taking the relative values of job execution options generated by the scheduling algorithm and replacing those with the absolute ones.

During execution, the scheduling algorithm normalizes performance of individual resources and thus the options it returns are relative to each other on the normalized scale. In order to map those onto absolute values, whether it is cost and time or accuracy and time, the mapper must analyze each option's parameters and adjust the values. To achieve so, the mapping component interacts with the available services to obtain actual resource cost and application's absolute performance values (i.e., base execution time). This is achieved through calls to $estTime(t_i, r_j)$ and $estCost(t_i, r_j)$ methods defined in Sect. 3. Overall, the scheduling algorithm focuses on abstract job parameterization (by focusing on minimizing load imbalance for a given job execution option). The mapper then maps generated options onto specific job option attributes such as runtime, cost, or accuracy. This process translates the internal, low-level description of a job option into a high-level, meaningful option a user can easily evaluate.

Note should be taken that derived job options have the property of *relative comparison,* meaning that one option is directly comparable to any other option. This property follows from the uniformity of method through which the options are generated (i.e., a single scheduling algorithm). The benefit is that runtime and cost estimation for all the options can be derived by scaling such information from any single option. This is more so important because proposed method operates on an application-specific basis and, as such, options that it generates are very unique and specific leading to hardship in accurate runtime/cost estimation by generic methods [26]. Therefore, mapping of options to desired objectives is performed by obtaining relevant information for a single option (e.g., from historical data or performance prediction tool) and then scaling the remainder of options accordingly. Even though this approach is dependent on underlying tool for its initial accuracy, relational accuracy of individual options is increased by such an approach. Furthermore, advances in runtime estimation and development of application-specific repositories and services will automatically have an impact on the correctness of described solution.

The *GUI component* represents the user-scheduler interaction module. All the complexities of resource selection, task parameterization, and data distribution are abstracted and hidden from the user who is presented with a clean interface enumerating available options and visualizing them in terms of selected tradeoffs. Such an approach to user-scheduler interaction enables a two-way interaction model where the user is offered insight into their current job's execution properties before committing to job submission. Through this model, the user becomes aware of job's execution space and can choose job execution option that they might not have known even existed. This is in direct contrast to solutions such as Condor classads or a cluster manager (e.g., SGE, PBS) where a user must provide low level details regarding their current job, such as maximum time they believe a job will run, number of processors, minimum amount of memory, and so on. To a non-computational user providing such values is a barrier to system usability because they are not interested in drawing necessary correlations. Therefore, such values are always set arbitrarily high and often results in poor resource utilization [3]. Moreover, once a user has an established a routine, they are unlikely to change it when input data and even applications change.

An example of the user interface displaying a set of possible job execution options is provided in Fig. 3. As can be seen, the interface provides a mapping of available

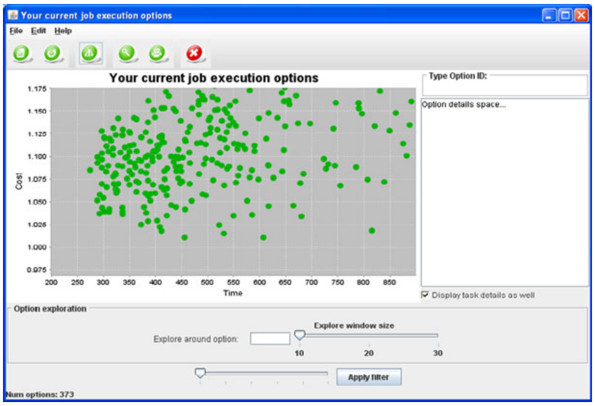


Fig. 3 A snapshot of the OptionView GUI module presenting a job execution space to the user

and generated job options onto the two objectives, namely time and cost. The user can easily interpret available options, consider tradeoffs, and select an option for execution. Such a presentation of a comprehensive spectrum of available job execution options allows the user great flexibility in terms of meeting their current needs. The interface furthermore allows the user to see details about any one option and apply a filter that will reduce the number of elements displayed on the screen. If user has narrowed down the desired area, local exploration functionality has also been implemented. Based on user input, the controller can be invoked with a specific job execution option around which the user wishes to explore for additional job execution options. The controller initiates exhaustive computation of a set of job execution options within a predefined window that are neighboring the one specified by the user (neighboring based on CPU assignment). The aim of this functionality is to provide the user with deeper insight regarding job execution options within a targeted range.

5 Experimental studies

Validation of the presented approach has been done in two stages. In the first stage, the entire set of job execution options, as generated by OptionView, is validated for the accuracy through simulation. In the second stage, accuracy of a representative subset of the generated options is validated on real-world resources, further demonstrating the validity of job option generation mechanism.

5.1 Environment setup

With the focus on generating job execution space that is applicable to an individual application and an available set of resources, validation methodology employed focuses on one application, namely Basic Local Alignment Search Tool (BLAST). BLAST is a popular sequence alignment tool used to perform similarity searches between an input query and database of infrequently changing sequences. For the performed tests, the *nr* database was used to execute the searches. *nr* database is a non-redundant protein database with entries from GenPept, Swissprot, PIR, PDF,

Table 2 Resource details used during experiments. PS refers to processing slot or a node. PE refers to a processing element or a core. MIPS stands for Millions Instructions Per Second of a single PE and is a resource performance metric employed by GridSim toolkit. PE MIPS were derived based form normalized application-specific performance benchmarks for given resource

Resource ID	# PSs	# PEs	PE MIPS rating	Cost/time unit
R1 (F)	5	40	100	$0.10
R2 (E)	10	20	52	$0.10
R3 (C)	15	15	57	$0.10

PDB, and RefSeq. Used version was 1.6 GB in size and available from NCBI.[3] Reasonably so, BLAST application and needed search database are assumed available on given resources. The input query files used for given job originated from 1,024 protein queries/sequences in FASTA format that were randomly selected from Viral Bioinformatics Resource Center (VBRC)[4] database.

The simulation part of validating proposed scheduling approach is performed through the GridSim toolkit [33]. GridSim is a distributed infrastructure simulation package that allows for creation and customization of individual resources as well as creation of heterogeneous jobs. Jobs created are packaged as Gridlets, which are specified in terms of job length in Millions of Instructions Per Second (MIPS), the size of job input, and the size of job output in bytes. Processing times of jobs within GridSim are proportional to the predefined speed of resources and the size of the job with a random variation of 0–10% to account for heterogeneity present in real-world environments.

Real-world validation has been performed on a set of resources available at UAB. Individual tasks were parameterized as per results of OptionView and then passed on to GridWay [34], which was used as the job submission engine. Relevant hardware characteristics of employed resources are shown in Table 2. In the table, PSs refers to the maximum number of Processing Slots available for use. A PS is single scheduling unit on a given resource and it depends on scheduling policy employed on the resource. Examples of PS include a single processing core or a single compute node with multiple, multicore CPUs on it. Processing Element (PE) refers to the smallest computational unit on given resource (i.e., a CPU in case of a single, one-core CPU resource or a core in case of a multi-CPU, multicore resource). With the available resources, the exhaustive number of job execution options is 1,056, and thus n, or the selected sample generated by OptionView (as described in Sect. 4.2), is 282. Therefore, 282 job execution options were simulated in GridSim and analyzed.

5.2 Simulated results

For the simulation experiments, a set of resources was created within GridSim that represent available real-world resources in terms of their configuration as well as

[3] http://www.ncbi.nlm.nih.gov/.

[4] http://www.biovirus.org/.

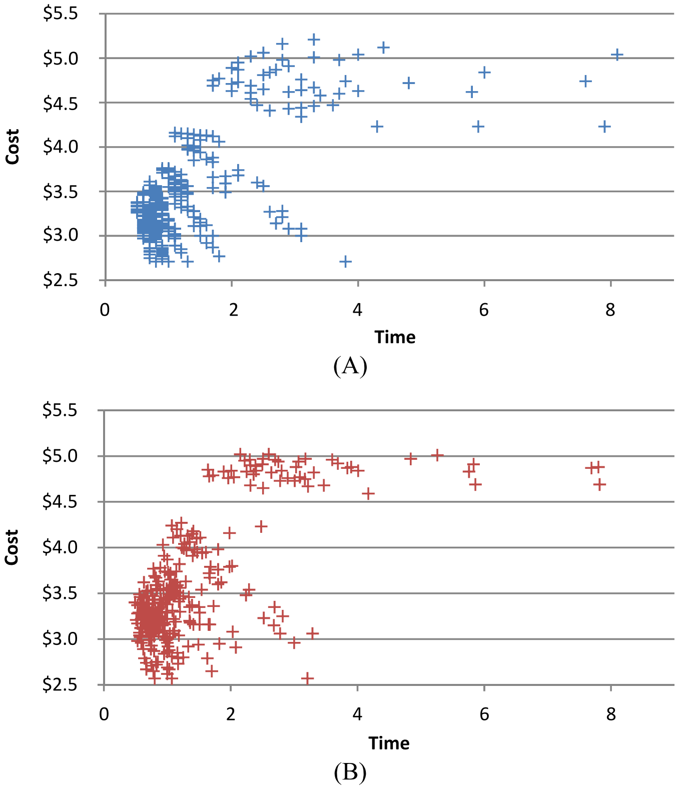


Fig. 4 Job option execution space as (**A**) generated by OptionView, and (**B**) simulated through GridSim. Each individual point shown represents a single job execution option, namely all the details required to submit a job in an application-oriented fashion (i.e., resource(s) selected for execution, data distribution under resource capability constraints, and individual task parameterizations)

relative performance. Relative performance of the resources was assigned based on application-specific resource benchmarks; BLAST was executed with the same input data across all available resources and obtained runtime values were normalized to the fastest resource. Derived values were used for the performance rate (i.e., Millions of Instructions Per Second—MIPS) of individual processing elements of a resource within GridSim (see Table 2). Cost associated with consuming resources was uniformly assigned to $0.10 per unit of execution. OptionView was used to generate a set of job execution options, each of which consisted of resource assignments and appropriate data distributions as per implementation of function $plan_J$. The derived options were simulated through GridSim by setting tasks' sizes as derived by OptionView and submitting the jobs to respective resources.

Experiment results are shown in Fig. 4(A) and 4(B) for generated and simulated data, respectively. From the shown figures, it is apparent that the job execution space generated by OptionView is somewhat more structured and regular than the simulated counterpart. However, the overall shape of the job execution space is main-

Table 3 Statistics of differences between generated and simulated job execution options. Numbers indicate difference in respective units and the corresponding percentage of simulated results when compared to the estimated values

	Runtime analysis		Cost analysis	
Avg Δ	0.01	4.08%	\$0.04	0.93%
Mean Δ	−0.01	−0.94%	\$0.03	0.98%
Std dev Δ	0.21	15.00%	\$0.12	3.18%
Max Δ	0.93	84.55%	\$0.46	10.87%

tained across generated and simulated data indicating global accuracy achieved by OptionView.

Statistical results of the analysis of simulated data at the individual option level are presented in Table 3. In the table, delta (Δ) is calculated by noting the difference between each of the generated and simulated data points and summarizing those across the sample space. From this data, it can be observed that, on average, the system achieves a very acceptable level of accuracy (approx. within 4%). Furthermore, value for the median statistic indicates that the system is very evenly generating data points on the positive as well as negative side. As a result, over time, a user would experience a high level of overall system accuracy. However, noteworthy values for the standard deviation and maximum delta indicate that many data points are likely to be incorrectly generated. Based on values of standard deviation measurements the error is largely contained within 15% of the observed results. In the area of application runtime prediction, this is considered an acceptable result (e.g., [35]).

The cost analysis presented shows that the cost component of the generated job options is at least as accurate as the runtime component. Overall, with the GridSim's built-in randomized variability of each task's runtime, these results present a very accurate solution.

5.3 Experimental results

Because comprehensive validation of the generated options would require executing n parameterizations of the same BLAST job, for the validation purposes, a smaller, representative subset of job execution options was selected. This subset was selected in the same fashion as the selection process of the sample space generated by OptionView. When using OptionView for meaningful computations, the user would obviously select only one such option to execute, namely the one that meets their demand most closely as opposed to executing the same job numerous times.

For our calculation of the validation sample space, the confidence level was set to 95% and confidence interval to 15%. Selected confidence interval value was selected in accordance to observed accuracy levels of application runtime prediction tools (e.g., [35]). As a result, from the 282 job execution options generated by OptionView for given resource availability, 37 represent the desired sample space. These 37 options were randomly selected for execution on real-world resources. Each of the jobs was executed based on the parameterization calculated by OptionView tool and

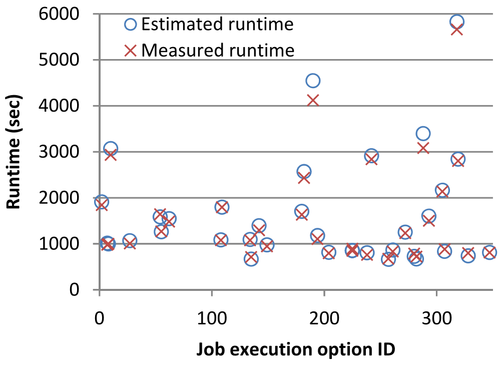


Fig. 5 Experimental runtime data for real-world resources for selected job execution options. Circles represent job execution option runtime estimates generated by OptionView while x's represent observed runtime characteristics after job's execution on real-world resources as per instructions of the job plan generated by OptionView

Table 4 Statistical analysis of runtime accuracy across all executed job execution options

Average error	Median error	Max error
−1.43%	−2.13%	8.21%

runtime results were recorded. Because in the utilized model, the cost is a function of task runtime, experiments focused on collection and accuracy of runtime data only.

Figure 5 is showing the runtime results of executing selected job execution options on resources specified in Table 2. Because of the individualized parameterization of any one option, the behavior of the scheduling system can be verified by considering accuracy of runtime estimation for a given option and then generalizing it by cumulative accuracy of all executed options. Due to the high level of detail, parameterization particulars on individual options that were executed are not provided. Analysis of details at such low level is not necessary because, from the user's perspective, the interaction with OptionView takes place at the job option level anyway. When interacting with OptionView, the user only sees available job execution options without regard for the details as to how each option execution is implemented. That is one of the internal tasks of OptionView as described throughout Sect. 4. Therefore, analyzing accuracy of the individual options provides insight into overall accuracy of the system.

Based on the data shown in Fig. 5, overall accuracy of individual job options in real-world setting is well within expected and acceptable outcomes. Data in Table 4 shows results of a statistical analysis of runtime accuracy across all executed options. This data is showing average accuracy of the system within 2% with maximum error being within 10%. These results furthermore coincide with the results obtained through simulation as described in the previous section.

6 Discussion

This section captures reflective discussion regarding limitations and extensions of the presented approach.

6.1 Result accuracy

Based on the experimental data, it can be concluded that job execution options automatically calculated and effectively presented through OptionView experience a high level of accuracy. This high level of accuracy is a result of the application-specific orientation and available application benchmarks. By understanding the dependencies that exist between an application and resource, resource capabilities can be more adequately met resulting in high confidence regarding runtime estimation. Such approach also presents a downside to the described approach because, in order to attain such high level of accuracy, an application and resource relationship needs to be understood and an application specific module needs to be built. Nevertheless, irrespective of the overall level of accuracy, OptionView performs metascheduling actions across resources in a fully automated fashion. It performs such actions in application- and resource-specific manner, and thus represents a best-effort type of a solution that will, in general, perform at least as good as an otherwise generic metascheduler.

6.2 Resource availability

As noted at the end of Sect. 4.1, OptionView operates best in an environment where resource availability does not rapidly change and where it is the only access point to the given set of resources (thus, no contention arises between multiple schedulers submitting jobs to the same set of resources). As a result, OptionView is seen as an ideal solution where a small number of users interact with the given compute infrastructure. Although this represents a niche area, it is a very important area because users in such environments typically lack time, knowledge, and desire to tweak their job submission process. Therefore, for users that do not posses needed informatics expertise, OptionView provides an awaited solution.

Because of the two-way communication between a user and the scheduler developed as part of OptionView, the span of time from the start of job submission, as initiated by a user, to submission of the actual tasks by a job manager may take considerably longer than a simple and immediate job submission performed by a fully automated scheduler. This implies that OptionView assumes non-rapidly changing resource availability. Going back to the airline analogy from the Introduction, the same model is adopted and it is working by flight reservation companies. Nonetheless, if the underlying environment resource availability is changing rapidly or it often happens that no available resources exist, OptionView can be implemented as an interface to an advance reservation system allowing users to realize the full potential for their jobs as well to observe viable job tradeoffs. However, this work and analysis are left as part of future work.

6.3 Extensions beyond EP applications

Because of lack of communication among individual tasks, a common assumption is that execution of an EP application in distributed environments is easy or at least significantly easier than execution of tightly coupled applications (e.g., MPI). However, based on an earlier study [5], execution of EP applications in distributed environments is confined by resource availability, optimized through task parameterization, hindered by simultaneous use and management of heterogeneous resources belonging to different administrative domains, and dependent on user requirements. Consequently, an act of effective application execution inherently includes an act of application scheduling. Because of the multiple influencing components, scheduling becomes a major component for effective execution of EP applications and should be handled comprehensively with respect to application execution environment variables and user desires.

The model of the EP class of applications possesses a versatility that can be used to model other classes of applications. This can be achieved by encapsulating those application classes within the EP class itself. A benefit of this feature of the EP class of applications is that it allows the same scheduling principles and techniques to be applied to other application classes.

For example, a sequential application can be represented by a single task that would also define the entire job within the EP application model. At that point, the scheduling model is simplified because the requirement to minimize load imbalance is removed and only a direct comparison of individual resources in terms of the given application needs to be performed. Similarly, because of the general inability of MPI applications to cross individual cluster boundaries, a tightly coupled MPI application can be encapsulated within a single task and scheduled as such. As the case is with the sequential applications, such an application instance can be scheduled irrespective of other such instances. Therefore, scheduling one such application job reduces to the ability to understand and leverage capabilities of individual resources from the perspective of given application, and then selecting the one resource that is the most likely to realize desired objective.

7 Conclusions and future work

Existing scheduling applications and approaches for scheduling loosely coupled applications across distributed environments focus on minimizing job turnaround time, cost, or ease of use. In such works, the user is assumed to know specifics about the given infrastructure and selected application and must provide initial job properties (e.g., number of tasks or number of CPUs to use). Then the scheduler must operate within those constraints. In addition, prior to job's completion, the user has no or minimal insight into concrete job metrics, such as the actual job execution time or cost. Furthermore, the user cannot observe differences among multiple execution alternatives and use the scheduler in an 'exploratory' mode.

The scheduler presented in this paper focuses on generating, providing, and presenting a set of job execution alternatives to a user so that the user can make targeted and timely decisions. Such presentation of job execution options allows for clear understanding of tradeoffs between the conflicting objectives. Adopting presented flexibility allows users to merge their requirements with the available job execution options and thus achieve higher level of QoS. The user becomes abstracted from the technology to the point where the technology presents its full potential to the user—automatically, without requiring specific knowledge form the user. Adoption of presented work allows delivery of focused solutions to domain scientists and enables them to concentrate on their work without requiring them to learn and understand how to use underlying technology operates. Specifically, the OptionView tool presents a fresh and novel approach to distributed application scheduling that has also shown a high level of accuracy for application scheduling in simulated as well as real-world environments.

In summary, we have developed a solution that analyzes application requirements and resource capabilities to generate a set of job execution options. A scheduling algorithm has been developed that effectively parameterizes execution of each generated option and, in turn, allows mapping of the options onto the two conflicting objectives. On top of the scheduling algorithm, a novel user interaction module has been developed that interprets job options and maps those to absolute values corresponding to chosen application and selected resources. Available options are presented to the user in terms of job execution tradeoffs allowing the user to better meet their needs. The system was validated through simulation as well as across real-world resources and has shown a high level of accuracy in terms of data presented and runtime results.

As part of future work, a generalized approach will be devised for incorporating any application into the scheduling framework. Moreover, adding support for specifying and executing an entire workflow without intermittent user intervention is envisioned. In order to do so effectively and enable easy addition of new applications, integration with the Application Information Service [27] framework is under development allowing generalization of applications performance characteristics as well as description of an application through standardized means. Lastly, cloud computing offers a quickly emerging domain that fits well into the functionality supported by OptionView (i.e., on-demand provisioning of distributed, heterogeneous resources to individuals). Although the resource acquisition model is different in cloud computing when compared to the traditional distributed resources (i.e., unlimited vs. limited resource pool), the solution offered by OptionView is highly applicable and it is our intent to adopt OptionView to execute within the cloud computing environments as well.

Acknowledgements This work was made possible in part by funding from the Department of Computer and Information Sciences at the University of Alabama at Birmingham, a grant of high performance computing resources from the CIS at UAB, the NSF Award CNS-0420614, and NIH/NIAID Contract No. HHSN266200400036C—Viral Bioinformatics Resource Center.

References

1. Segaran T, Hammerbacher J (eds) (2009) Beautiful data: the stories behind elegant data solutions. O'Reilly Media, Sebastopol
2. Afgan E, Bangalore P (2007) Performance characterization of BLAST for the grid. In: IEEE 7th international symposium on bioinformatics & bioengineering (IEEE BIBE 2007). Boston, MA, pp 1394–1398
3. Lee CB, Snavely A (2006) On the user–scheduler dialogue: studies of user-provided runtime estimates and utility functions. Int J High Perform Comput Appl 20:495–506
4. Berman F (1998) High-performance schedulers. In: Foster I, Kesselman C (eds) The grid: blueprint for a new computing infrastructure. Morgan Kaufmann, San Francisco, pp 279–309
5. Afgan E, Bangalore P (2008) Embarrassingly parallel jobs are not embarrassingly easy to schedule on the grid. Presented at the SC08 international conference for high performance, networking, storage and analysis—workshop on many-task computing on grids and supercomputers, Austin, TX
6. Leung JY-T (ed) (2004) Handbook of scheduling: algorithms, models, and performance analysis. CRC Press, Boca Raton

7. Buyya R, Murshed M, Abramson D, Venugopa S (2005) Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost-time optimization algorithm. Softw Pract Exp 35:491–512
8. Iosup A, Dumitrescu C, Epema DH, Li H, Wolters L (2006) How are real grids used? The analysis of four grid traces and its implications. In: International conference on grid computing 2006, Barcelona, Spain, pp 262–269
9. Abramson D, Giddy J, Kotler L (2000) High performance parametric modeling with Nimrod/G: killer application for the global grid. In: International parallel and distributed processing symposium (IPDPS), Cancun, Mexico, pp 520–528
10. Foster I, Kesselman C (eds) (1999) The grid: blueprint for a new computing infrastructure. Morgan Kaufmann, San Mateo
11. Berman FD, Wolski R, Figueira S, Schopf J, Shao G (1996) Application-level scheduling on distributed heterogeneous networks. In: Supercomputing '96, Pittsburgh, PA, p 28
12. Casanova H, Obertelli G, Berman F, Wolski R (2000) The AppLeS parameter sweep template: user-level middleware for the grid. In: Supercomputing 2000. Dallas, TX
13. Berman F, Chien A et al (2001) The GrADS project: software support for high-level grid application development. Int J High Perform Comput Appl 15:327–344
14. Dail H, Berman F, Casanova H (2003) A decoupled scheduling approach for grid application development environments. J Parallel Distrib Comput 63:505–524
15. Buyya R, Abramson D, Giddy J (2000) Nimrod-G: an architecture for a resource management and scheduling in a global computational grid. In: 4th international conference and exhibition on high performance computing in Asia-Pacific region (HPC ASIA 2000), Beijing, China, pp 283–289
16. Smith C (2003) Open source metascheduling for virtual organizations with the community scheduler framework (CSF). Platform Comput Whitepaper, August
17. Dvořák F, Kouř D et al (2006) gLite job provenance. In: Provenance and annotation of data. Springer, Berlin, pp 246–253
18. Montero RS, Huedo E, Llorente IM (2006) Grid scheduling infrastructures based on the GridWay meta-scheduler. IEEE Technical Committee on Scalable Computing (TCSC) Newsletter, vol 8
19. Kertesz A, Kacsuk P (2007) A taxonomy of grid resource brokers. In: Kacsuk P et al (eds) Distributed and parallel systems from cluster to grid computing. Springer, Berlin, pp 201–210
20. Solomon M (2004) The classad language reference manual, May. Available: http://www.cs.wisc.edu/condor/classad/refman/
21. Wieczorek M, Podlipnig S, Prodan R, Fahringer T (2008) Bi-criteria scheduling of scientific workflows for the grid. In: 2008 eighth IEEE international symposium on cluster computing and the grid (ccGrid), Lyon, France, pp 9–16
22. Yu J, Kirley M, Buyya R (2007) Multi-objective planning for workflow execution on grids. In: Grid 2007, Austin, TX, pp 10–17
23. Li C, Li L (2007) Utility-based QoS optimisation strategy for multi-criteria scheduling on the grid. J Parallel Distrib Comput 67:142–153
24. Kumar V (2002) Introduction to parallel computing. Addison-Wesley, Longman, Boston
25. Afgan E, Bangalore P (2007) Application specification language (ASL)—a language for describing applications in grid computing. In: The 4th international conference on grid services engineering and management—GSEM 2007, Leipzig, Germany, pp 24–38
26. Elmroth E, Tordsson J, Fahringer T, Nadeem F, Gruber R, Keller V (2008) Three complementary performance prediction methods for grid applications. In: CoreGRID integration workshop 2008, Heraklion, Greece
27. Afgan E, Bangalore P, Duncan D (2009) GridAtlas—a grid application and resource configuration repository and discovery service. In: IEEE Cluster 2009, New Orleans, LA, p 10
28. Afgan E, Bangalore P, Mukkai S, Yammanuru S (2008) Design and implementation of a readily available historical application performance database (AppDB) for grid. University of Alabama at Birmingham (UAB), Birmingham, AL UABCIS-TR-2008-0506-1, May 6
29. Siddiqui M, Villazón A, Fahringer T (2006) Grid allocation and reservation—grid capacity planning with negotiation-based advance reservation for optimized QoS. In: 2006 ACM/IEEE conference on supercomputing, Tampa, FL, pp 21–35
30. Sotomayor B, Keahey K, Foster I (2008) Combining batch execution and leasing using virtual machines. In: ACM/IEEE international symposium on high performance distributed computing 2008 (HPDC 2008), Boston, MA, pp 87–96
31. Olofsson P (2005) Probability, statistics, and stochastic processes, 1st edn. Wiley-Interscience, New York

32. Standard Performance Evaluation Corporation, July 24. Available: http://www.spec.org/
33. Buyya R, Murshed M (2002) GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. J Concurr Comput Pract Exp (CCPE) 14:1175–1220
34. GridWay (2009) Job template options, Feb 16. Available: http://www.gridway.org/documentation/stable5.4/user/gridway-user-functionality.html#id2578278
35. Pfeiffer W, Wright NJ (2008) Modeling and predicting application performance on parallel computers using HPC challenge benchmarks. In: IEEE symposium on parallel and distributed processing (IPDPS), Miami, FL, pp 1–12