Imbalance Optimization and Task Clustering in Scientific Worfklows

Weiwei Chen[a,*], Rafael Ferreira da Silva[a], Ewa Deelman[a], Rizos Sakellariou[b]

[a] *University of Southern California, Information Sciences Institute, Marina del Rey, CA, USA*
[b] *University of Manchester, School of Computer Science, Manchester, U.K.*

Abstract

Scientific workflows can be composed of many fine computational granularity tasks. The runtime of these tasks may be shorter than the duration of system overheads, for example, when using multiple resources of a cloud infrastructure. Task clustering is a runtime optimization technique that merges multiple short tasks into a single job such that the scheduling overhead is reduced and the overall runtime performance is improved. However, existing task clustering strategies only provide a coarse-grained approach that relies on an over-simplified workflow model. In our work, we examine the reasons that cause Runtime Imbalance and Dependency Imbalance in task clustering. Next, we propose quantitative metrics to evaluate the severity of the two imbalance problems respectively. Furthermore, we propose a series of task balancing methods to address these imbalance problems. Finally, we analyze their relationship with the performance of these task balancing methods. A trace-based simulation shows our methods can significantly improve the runtime performance of five widely used workflows compared to the actual implementation of task clustering.

Keywords: Scientific workflows, Performance analysis, Scheduling, Workflow simulation, Task clustering, Load balancing

1. Introduction

Many computational scientists develop and use large-scale, loosely-coupled applications that are often structured as scientific workflows, which consist of many computational tasks with data dependencies between them. Although the majority of the tasks within these applications are often relatively short running (from a few seconds to a few minutes), in aggregate they represent a significant amount of computation and data [1]. When executing these applications on a multi-machine distributed environment, such as the Grid or the Cloud, significant system overheads may exist and may adversely slowdown the application performance [2]. To minimize the impact of such overheads, task clustering techniques [3, 4, 5, 6, 7, 8, 9, 10, 11] have been developed to group *fine-grained* tasks into *coarse-grained* tasks so that the number of computational activities is reduced and their computational granularity is increased thereby reducing the (mostly scheduling related) system overheads [2]. However, there are several challenges that have not yet been addressed.

In a scientific workflow, tasks within a level (or depth within a workflow directed acyclic graph) may have different runtimes. Merging tasks within a level without considering the runtime variance may cause load imbalance, i.e., some clustered jobs may be composed of short running tasks while others of long running tasks. This imbalance delays the release of tasks from the next level of the workflow, penalizing the workflow execution with an overhead produced by the use of inappropriate task clustering strategies [12]. A common technique to handle load imbalance is overdecomposition [13]. This method decomposes computational work into medium-grained balanced tasks. Each task is coarse-grained enough to enable efficient execution and reduce scheduling overheads, while being fine-grained enough to expose significantly higher application-level parallelism than that is offered by the hardware.

Data dependencies between workflow tasks play an important role when clustering tasks within a level. A data dependency means that there is a data transfer between two tasks (output data for one and input data for the other). Grouping tasks without considering these dependencies may lead to data locality problems where output data produced by parent tasks are poorly distributed. Thus, data transfer times and failures probability increase. Therefore, we claim that data dependencies of subsequent tasks should be considered.

We generalize these two challenges (Runtime Imbalance and Dependency Imbalance) to the generalized load balance problem. We introduce a series of balancing methods to address these challenges. However, there is a tradeoff between runtime and data dependency balancing. For instance, balancing runtime may aggravate the Dependency Imbalance problem, and vice versa. Therefore, we propose a series of quantitative metrics that reflect the internal structure (in terms of task runtimes and dependencies) of the workflow to serve as a criterion to select and balance these solutions.

In particular, we provide a novel approach to capture these metrics. Traditionally, there are two approaches to improve the performance of task clustering. The first one is a top-down approach [14] that represents the clustering problem as a global optimization problem and aims to minimize the overall workflow execution time. However, the complexity of solving such

*Corresponding address: USC Information Sciences Institute, 4676 Admiralty Way Ste 1001, Marina del Rey, CA, USA, 90292, Tel: +1 310 448-8408
Email addresses: weiweich@acm.org (Weiwei Chen), rafsilva@isi.edu (Rafael Ferreira da Silva), deelman@isi.edu (Ewa Deelman), rizos@cs.man.ac.uk (Rizos Sakellariou)

an optimization problem does not scale well since most methods use genetic algorithms. The second one is a bottom-up approach [3, 9] that only examines free tasks to be merged and optimizes the clustering results locally. In contrast, our work extends these approaches to consider the neighboring tasks including siblings, parents, and children because such a family of tasks has strong connections between them.

The quantitative metrics and balancing methods were introduced and evaluated in [15] on two workflows. In this work, we complement this previous presentation by studying (*i*) the performance gain of using our balancing methods over a control execution on a larger set of workflows; (*ii*) the performance gain over two additional legacy task clustering methods from the literature; (*iii*) the performance impact of the variation of the average data size and number of resources; and (*iv*) the performance impact of combining our balancing methods with vertical clustering.

The rest of this paper is organized as follows. Section 2 gives an overview of the related work. Section 3 presents our workflow and execution environment models. Section 4 details our heuristics and algorithms, Section 5 reports experiments and results, and the paper closes with a discussion and conclusions.

2. Related Work

The low performance of *fine-grained* tasks is a common problem in widely distributed platforms where the scheduling overhead and queuing times at resources are high, such as Grid and Cloud systems. Several works have addressed the control of task granularity of bag of tasks. For instance, Muthuvelu et al. [3] proposed a clustering algorithm that groups bag of tasks based on their runtime—tasks are grouped up to the resource capacity. Later, they extended their work [4] to determine task granularity based on task file size, CPU time, and resource constraints. Recently, they proposed an online scheduling algorithm [5, 6] that groups tasks based on resource network utilization, user's budget, and application deadline. Ng et al. [7] and Ang et al. [8] introduced bandwidth in the scheduling framework to enhance the performance of task scheduling. Longer tasks are assigned to resources with better bandwidth. Liu and Liao [9] proposed an adaptive fine-grained job scheduling algorithm to group fine-grained tasks according to processing capacity and bandwidth of the current available resources. Although these techniques significantly reduce the impact of scheduling and queuing time overhead, they are not applicable to scientific workflows, since data dependencies are not considered.

Task granularity control has also been addressed in scientific workflows. For instance, Singh et al. [10] proposed a level- and label-based clustering. In level-based clustering, tasks at the same level can be clustered together. The number of clusters or tasks per cluster are specified by the user. In the label-based clustering, the user labels tasks that should be clustered together. Although their work considers data dependency between workflow levels, it is done manually by the users, which is prone to errors. Recently, Ferreira da Silva et al. [11] proposed task grouping and ungrouping algorithms to control workflow task granularity in a non-clairvoyant and online context, where none or few characteristics about the application or resources are known in advance. Their work significantly reduced scheduling and queuing time overheads, but did not consider data dependencies.

A plethora of balanced scheduling algorithms have been developed in the networking and operating system domains. Many of these schedulers have been extended to the hierarchical setting. Lifflander et al. [13] proposed to use work stealing and a hierarchical persistence-based rebalancing algorithm to address the imbalance problem in scheduling. Zheng et al. [16] presented an automatic hierarchical load balancing method that overcomes the scalability challenges of centralized schemes and poor solutions of traditional distributed schemes. There are other scheduling algorithms [17] (e.g. list scheduling) that indirectly achieve load balancing of workflows through makespan minimization. However, the benefit that can be achieved through traditional scheduling optimization is limited by its complexity. The performance gain of task clustering is primarily determined by the ratio between system overheads and task runtime, which is more substantial in modern distributed systems such as Clouds and Grids.

3. Model and Design

A workflow is modeled as a Directed Acyclic Graph (DAG). Each node in the DAG often represents a workflow task (*t*), and the edges represent dependencies between the tasks that constrain the order in which tasks are executed. Dependencies typically represent data-flow dependencies in the application, where the output files produced by one task are used as inputs of another task. Each task is a program and a set of parameters that need to be executed. Figure 1 (left) shows an illustration of a DAG composed by four tasks. This model fits several workflow management systems such as Pegasus [18], Askalon [19], and Taverna [20].

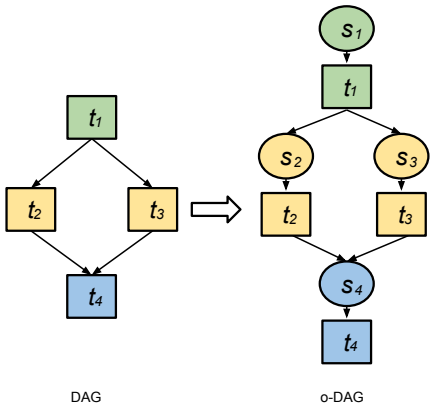

Figure 1: Extending DAG to o-DAG.

Figure 2 shows a typical workflow execution environment. The submit host prepares a workflow for execution (clustering, mapping, etc.), and worker nodes, at an execution site, execute jobs individually. The main components are introduced below:

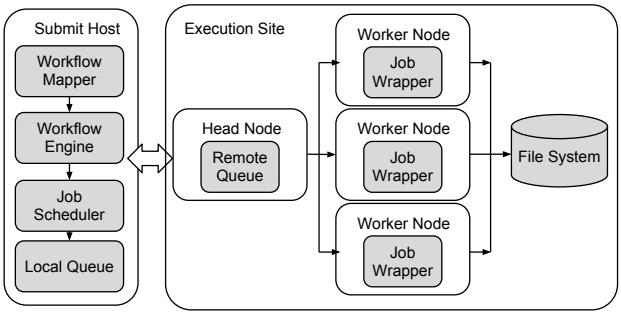

Figure 2: A workflow system model.

Workflow Mapper. Generates an executable workflow based on an abstract workflow provided by the user or workflow composition system. It also restructures the workflow to optimize performance and adds tasks for data management and provenance information generation. In this work, the workflow mapper is used to merge small tasks together into a job such that system overheads are reduced (**task clustering**). A job is a single execution unit in the workflow execution systems and is composed by one or more tasks.

Workflow Engine. Executes jobs defined by the workflow in order of their dependencies. Only jobs that have all their parent jobs completed are submitted to the Job Scheduler. The Workflow Engine relies on the resources (compute, storage, and network) defined in the executable workflow to perform computations. The elapsed time from when a job is released (all of its parents have completed successfully) to when it is submitted to the job scheduler is denoted the workflow engine delay.

Job Scheduler and Local Queue. Manage individual workflow jobs and supervise their execution on local and remote resources. The elapsed time from when a task is submitted to the job scheduler to when it starts its execution in a worker node is denoted as the queue delay. It reflects both the efficiency of the job scheduler and the resource availability.

Job Wrapper. Extracts tasks from clustered jobs and executes them at the worker nodes. The clustering delay is the elapsed time of the extraction process.

We extend the DAG model to be overhead aware (o-DAG). System overheads play an important role in workflow execution and constitute a major part of the overall runtime when tasks are poorly clustered [2]. Figure 1 shows how we augment a DAG to be an o-DAG with the capability to represent system overheads (s) such as workflow engine and queue delays. In addition, system overheads also include data transfer delay caused by staging-in and staging-out data. This classification of system overheads is based on our prior study on workflow analysis [2].

With an o-DAG model, we can explicitly express the process of task clustering. In this work, we address task clustering horizontally and vertically. **Horizontal Clustering** (HC) merges multiple tasks that are at the same horizontal level of the workflow, in which task horizontal level is defined as the furthest distance from the root task to this task. **Vertical Clustering** (VC) merges tasks within a pipeline of the workflow. Tasks at the same pipeline share a single-parent-single-child relationship, which means a task t_a is the unique parent of a task t_b, which is the unique child of t_a.

Figure 3 shows a simple example of how to perform HC, in which two tasks t_2 and t_3, without data dependency between them, are merged into a clustered job j_1. A job j is a single execution unit composed by one or multiple task(s). Job wrappers are commonly used to execute clustered jobs, but they add an overhead denoted the clustering delay c. The clustering delay measures the difference between the sum of the actual task runtimes and the job runtime seen by the job scheduler. After horizontal clustering, t_2 and t_3 in j_1 can be executed in sequence or in parallel, if supported. In this work, we consider sequential executions only. Given a single resource, the overall runtime for the workflow in Figure 3 (left) is $runtime_l = s_1 + t_1 + s_2 + t_2 + s_3 + t_3 + s_4 + t_4$, and the overall runtime for the clustered workflow in Figure 3 (right) is $runtime_r = s_1+t_1+s_2+c_1+t_2+t_3+s_4+t_4$. $runtime_l > runtime_r$ as long as $c_1 < s_3$, which is the case of many distributed systems since the clustering delay within an execution node is usually shorter than the scheduling overhead across different execution nodes.

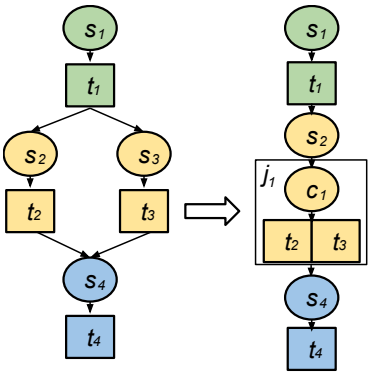

Figure 3: An example of horizontal clustering (color indicates the horizontal level of a task).

Figure 4 illustrates an example of vertical clustering, in which the pipeline containing tasks t_2, t_4, t_6 and t_8 is merged into j_1, while the pipeline containing tasks t_3, t_5, t_7 and t_9 is merged into j_2. Similarly, clustering delays c_2 and c_3 are added to j_1 and j_2 respectively, but system overheads s_4, s_5, s_6, s_7, s_8 and s_9 are removed.

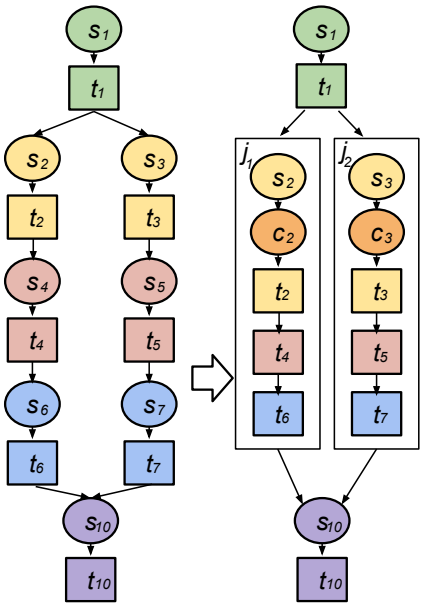

Figure 4: An Example of vertical clustering.

4. Balanced Clustering

Task clustering has been widely used to address the low performance of very short tasks on platforms where the system overhead is high, such as distributed computing infrastructures. However, techniques do not consider the load balance problem. In particular, merging tasks within a level without considering the runtime variance may cause load imbalance (Runtime Imbalance), or merging without considering data dependency may lead to data locality problems (Dependency Imbalance). In this section, we introduce metrics that quantitatively capture workflow characteristics to measure runtime and dependence imbalances. We then present methods to handle the load balance problem.

4.1. Imbalance Metrics

Runtime Imbalance describes the difference of the task/job runtime of a group of tasks/jobs. In this work, we denote the **Horizontal Runtime Variance** (*HRV*) as the standard deviation in task runtime at the same horizontal level of a workflow. At the same horizontal level, the job with the longest runtime often controls the release of the next level jobs. A high *HRV* value means that the release of next level jobs has been delayed. Therefore, to improve runtime performance, it is meaningful to reduce the standard deviation of job runtime. Figure 5 shows an example of four independent tasks t_1, t_2, t_3 and t_4 where the task runtime of t_1 and t_2 is 10 seconds and the task runtime of t_3 and t_4 is 30 seconds. In the Horizontal Clustering (HC) approach, a possible clustering result could be merging t_1 and t_2 into a clustered job and t_3 and t_4 into another. This approach results in imbalanced runtime, i.e., $HRV > 0$ (Figure 5-top). In contrast, a balanced clustering strategy should try its best to evenly distribute task runtime among jobs as shown in Figure 5 (bottom).

A smaller *HRV* means that the runtime of tasks within a horizontal level is more evenly distributed and therefore it is less necessary to balance the runtime distribution. However, runtime variance is not able to describe how regular is the structure of the dependencies between tasks.

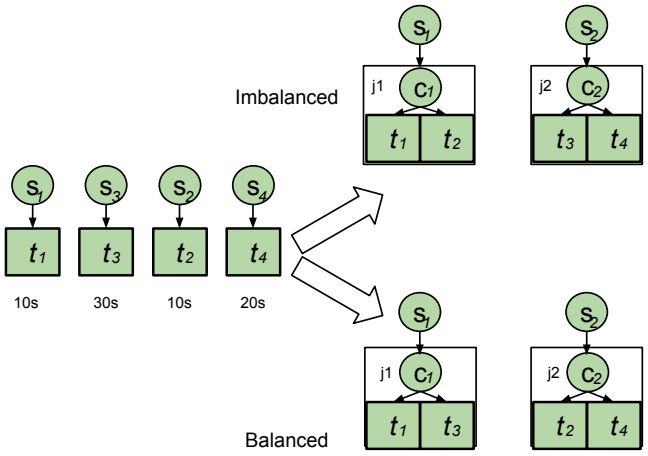

Figure 5: An example of runtime variance.

Dependency Imbalance means that the task clustering at one horizontal level forces the tasks at the next level (or even subsequent levels) to have severe data locality problem and thus loss of parallelism. For example, in Figure 6, we show a two-level workflow composed of four tasks in the first level and two in the second. Merging t_1 with t_2 and t_3 with t_4 (imbalanced workflow in Figure 6) forces t_5 and t_6 to transfer files from two locations and wait for the completion of t_1, t_2, t_3, and t_4. A balanced clustering strategy groups tasks that have the maximum number of child tasks in common. Thus, t_5 can start to execute as soon as t_1 and t_3 are completed, and so can t_6. To measure and quantitatively demonstrate the Dependency Imbalance of a workflow, we propose two metrics: (*i*) Impact Factor Variance, and (*ii*) Distance Variance.

We define the **Impact Factor Variance** (*IFV*) of tasks as the standard deviation of their impact factor. The Impact Factor aims at capturing the similarity of tasks/jobs in a graph by measuring their relative impact factor or importance to the entire graph. Tasks with similar impact factors are merged together, so that the workflow structure tends to be more 'even' or 'regular'. The **Impact Factor** (*IF*) of a task t_u is defined as follows:

$$IF(t_u) = \sum_{t_v \in Child(t_u)} \frac{IF(t_v)}{\|Parent(t_v)\|} \quad (1)$$

where $Child(t_u)$ denotes the set of child tasks of t_u, and $\|Parent(t_v)\|$ the number of parent tasks of t_v. For simplicity, we assume the *IF* of a workflow exit task (e.g. t_5 in Figure 6) as 1.0. For instance, consider the two workflows presented in Figure 7. *IF* for t_1, t_2, t_3, and t_4 are computed as follows:

$$IF(t_7) = 1.0, IF(t_6) = IF(t_5) = IF(t_7)/2 = 0.5$$

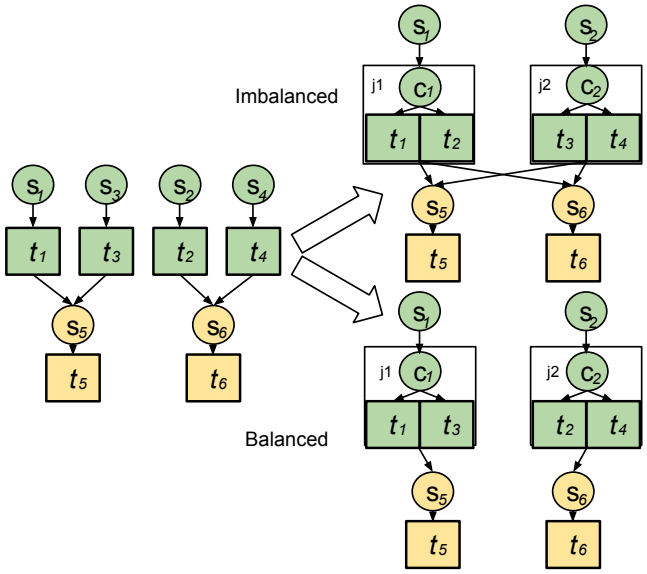

Figure 6: An example of dependency variance.

$$IF(t_1) = IF(t_2) = IF(t_5)/2 = 0.25$$
$$IF(t_3) = IF(t_4) = IF(t_6)/2 = 0.25$$

Thus, IFV(t_1, t_2, t_3, t_4) = 0. In contrast, IF for $t_{1'}$, $t_{2'}$, $t_{3'}$, and $t_{4'}$ are:

$$IF(t_{7'}) = 1.0, IF(t_{6'}) = IF(t_{5'}) = IF(t_{1'}) = IF(t_{7'})/2 = 0.5$$
$$IF(t_{2'}) = IF(t_{3'}) = IF(t_{4'}) = IF(t_{6'})/3 = 0.17$$

Therefore, the IFV value for $t_{1'}$, $t_{2'}$, $t_{3'}$, $t_{4'}$ is 0.17, which means it is less regular than the workflow in Figure 7 (left). In this work, we use **HIFV** (Horizontal IFV) to indicate the IFV of tasks at the same horizontal level. The time complexity of calculating all the IF of a workflow with n tasks is $O(n)$.

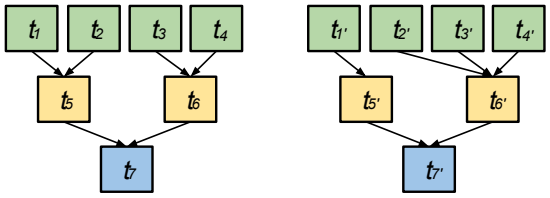

Figure 7: Example of workflows with different data dependencies (For better visualization, we do not show system overheads in these figures in the rest of the paper).

Distance Variance (DV) describes how 'closely' tasks are from each other. The distance between two tasks/jobs is defined as the cumulative length of the path to their closest common successor. If they do not have a common successor, the distance is set to infinity. For a group of n tasks/jobs, the distance between them is represented by a $n \times n$ matrix D, where an element $D(u, v)$ denotes the distance between a pair of tasks/jobs u and v. For any workflow structure, $D(u, v) = D(v, u)$ and $D(u, u) = 0$, thus we ignore the cases when $u \geq v$. Distance Variance is then defined as the standard deviation of all the elements $D(u, v)$ for $u < v$. The time complexity of calculating all the D of a workflow with n tasks is $O(n^2)$.

Similarly, HDV indicates the DV of a group of tasks/jobs at the same horizontal level. For example, Table 1 shows the distance matrices of tasks from the first level for both workflows of Figure 7 (D_1 for the workflow in the left and D_2 for the workflow in the right). HDV for t_1, t_2, t_3, and t_4 is 1.03, and for $t_{1'}, t_{2'}, t_{3'}$, and $t_{4'}$ is 1.10. In terms of distance variance, D_1 is more 'even' than D_2. A smaller HDV means the tasks at the same horizontal level are more equally 'distant' from each other and thus the workflow structure tends to be more 'evenly' and 'regular'.

D_1	t_1	t_2	t_3	t_4
t_1	0	2	4	4
t_2	2	0	4	4
t_3	4	4	0	2
t_4	4	4	2	0

D_2	t'_1	t'_2	t'_3	t'_4
t'_1	0	4	4	4
t'_2	4	0	2	2
t'_3	4	2	0	2
t'_4	4	2	2	0

Table 1: Distance matrices of tasks from the first level of workflows in Figure 7.

In conclusion, runtime variance and dependency variance offer a quantitative and comparable tool to measure and evaluate the internal structure of a workflow.

4.2. Balanced Clustering Methods

In this subsection, we introduce our balanced clustering methods used to improve the runtime and dependency balances in task clustering. We first introduce the basic runtime-based clustering method and then two other balancing methods that address the dependency imbalance problem.

Horizontal Runtime Balancing (HRB) aims to evenly distribute task runtime among clustered jobs. Tasks with the longest runtime are added to the job with the shortest runtime. This greedy method is used to address the imbalance problem caused by runtime variance at the same horizontal level. Figure 8 shows an example of HRB where tasks in the first level have different runtimes and should be grouped into two jobs. HRB sorts tasks in decreasing order of runtime, and then adds the task with the highest runtime to the group with the shortest aggregated runtime. Thus, t_1 and t_3, as well as t_2 and t_4 are merged together. For simplicity, system overheads are not displayed.

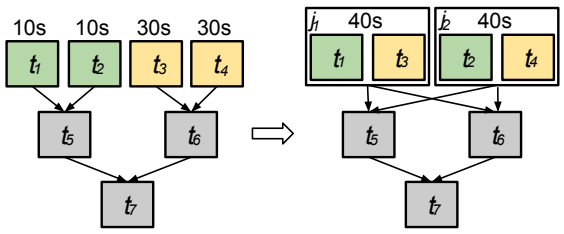

Figure 8: An example of HRB.

However, HRB may cause a dependency imbalance problem since the clustering does not take data dependency into consideration. To address this problem, we propose the **Horizontal Impact Factor Balancing** (HIFB) and the **Horizontal Distance Balancing** (HDB) methods.

In HRB, candidate jobs are sorted by their runtime, while in HIFB jobs are first sorted based on their similarity of IF, then on runtime. For example, in Figure 9, t_1 and t_2 have $IF = 0.25$, while t_3, t_4, and t_5 have $IF = 0.16$. HIFB selects a list of candidate jobs with the same IF value. Then, HRB is performed to select the shortest job. Thus, HIFB merges t_1 and t_2 together, as well as t_3 and t_4.

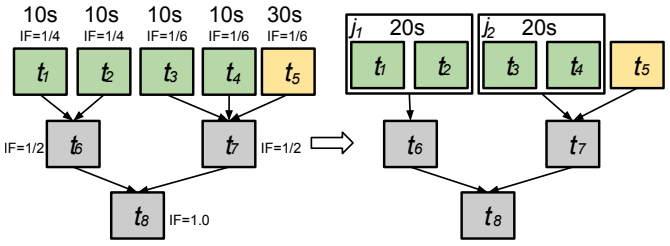

Figure 9: An example of HIFB.

However, HIFB is suitable for workflows with asymmetric structure. For symmetric workflows, such as the one shown in Figure 8, the IF value for all tasks from the first level will be the same (e.g. $IF = 0.25$), thus the method may also cause dependency imbalance. HDB jobs are sorted based on the distance between them and the targeted task t, then on their runtimes. For instance, in Figure 11, the distances between tasks $D(t_1, t_2) = D(t_3, t_4) = 2$, while $D(t_1, t_3) = D(t_1, t_4) = D(t_2, t_3) = D(t_2, t_4) = 4$. Thus, HDB merges a list of candidate tasks with the minimal distance (t_1 and t_2, and t_3 and t_4). Note that even if the workflow is asymmetric (Figure 9), HDB would obtain the same result as in HIFB.

In some rare cases, HDB would perform worst than HIFB. Figure 10 shows a case when merging t_1, t_2, t_3, t_4, t_5, HDB does not tell the difference between t_3, t_4, t_5 since the distances between each pair of them are all 2. However, HIFB does distinguish these tasks since their impact factors are quite different. This example exists in some of the LIGO workflows, which we will discuss its details in Section. 5.4.

Table 2 summarizes the imbalance metrics and balancing methods presented in this work.

4.3. Combining Vertical Clustering Methods

In this section, we discuss how we combine the horizontal clustering methods with vertical clustering. For one example workflow shown in Figure 12, we may simply merge tasks at the fourth level and tasks at the fifth level vertically. Vertical clustering can always guarantee a non-negative improvement over the case without clustering since it reduces the system overheads and data transfer between the tasks in the same pipeline. Horizontal clustering does not have the same guarantee since its performance depends on the comparison of system overheads

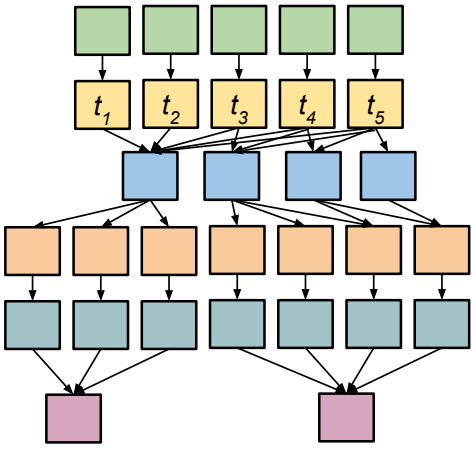

Figure 10: An example when HDB does not distinguish tasks while HIFB does

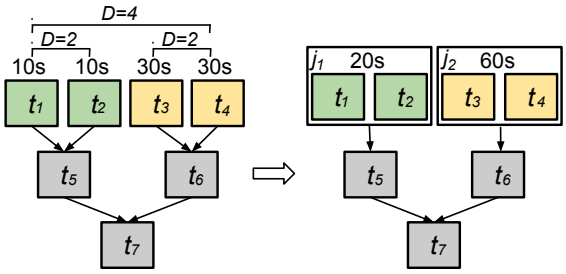

Figure 11: An example of HDB.

and task runtime. However, vertical clustering has limited performance improvement if we do not have many pipelines to perform it. Therefore, we are interested to understand how we should combine them together in an efficient way. Here we indicate two approaches to combine vertical clustering and all the horizontal clustering methods: **Vertical Clustering Prior** means we perform vertical clustering first and then horizontal clustering methods. Figure 13 shows that after performing vertical clustering as shown in Figure 12, we can further merge clustered jobs together based on the different criteria (HIFB, HDB, HRB or HC).

Vertical Clustering Posterior means we perform horizontal clustering methods first and then vertical clustering. For the same workflow, assuming we merge tasks horizontally as shown in Figure 14, we can see that we cannot perform ver-

Imbalance Metrics	*abbr.*
Horizontal Runtime Variance	*HRV*
Horizontal Impact Factor Variance	*HIFV*
Horizontal Distance Variance	*HDV*
Balancing Methods	***abbr.***
Horizontal Runtime Balancing	HRB
Horizontal Impact Factor Balancing	HIFB
Horizontal Distance Balancing	HDB

Table 2: Imbalance metrics and balancing methods.

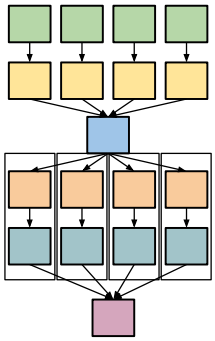

Figure 12: An example of Vertical Clustering.

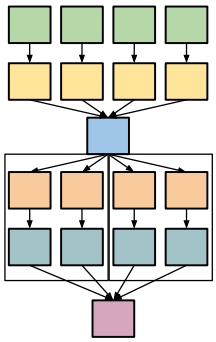

Figure 13: Vertical Clustering Prior.

tical clustering to clustered jobs at the fourth level and the fifth level since the original pipeline structures have been destroyed by horizontal clustering. This phenomenon suggests us VC-posterior may work better compared to VC-prior, generally speaking. However, some opposite cases do exist. We will verify our hypothesis in Section 5.4. We will also compared the two combining approaches with **VC-only**, which means we perform vertical clustering only and **No-VC**, which means we just perform horizontal clustering methods without vertical clustering.

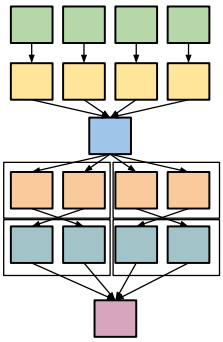

Figure 14: Vertical Clustering Posterior.

5. Evaluation

The experiments presented hereafter evaluate the performance of our balancing methods when compared to an existing and effective task clustering strategy named Horizontal Clustering (HC) [10], which is widely used by workflow management systems such as Pegasus. We also compare our methods with two legacy heuristics from the literature: DFJS [3], and AFJS [9]. DFJS groups bag of tasks based on the task durations up to the resource capacity. AFJS is an extended version of DFJS that is an adaptive fine-grained job scheduling algorithm to group fine-grained tasks according to processing capacity and bandwidth of the current available resources.

5.1. Scientific workflow applications

Five real scientific workflow applications are used in the experiments: LIGO Inspiral analysis, Montage, CyberShake, Epigenomics, and SIPHT. In this subsection, we describe each workflow application and present their main characteristics and structures.

LIGO. The Laser Interferometer Gravitational Wave Observatory (LIGO) [21] workflows are used to search for gravitational wave signatures in data collected by large-scale interferometers. The observatories' mission is to detect and measure gravitational waves predicted by general relativity Einstein's theory of gravity in which gravity is described as due to the curvature of the fabric of time and space. LIGO Inspiral workflow is a data intensive workflow. Figure 15 shows a simplified version of the workflow.

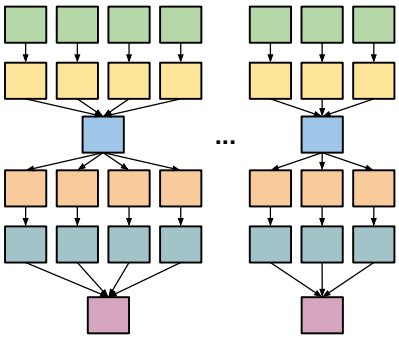

Figure 15: A simplified visualization of the LIGO Inspiral workflow.

Montage. Montage [22] is an astronomy application that is used to construct large image mosaics of the sky. Input images are reprojected onto a sphere and overlap is calculated for each input image. The application re-projects input images to the correct orientation while keeping background emission level constant in all images. The images are added by rectifying them into a common flux scale and background level. Finally the reprojected images are co- added into a final mosaic. The resulting mosaic image can provide a much deeper and detailed understanding of the portion of the sky in question. Figure 16 illustrates a small Montage workflow. The size of the workflow

depends on the number of images used in constructing the desired mosaic of the sky. The structure of the workflow changes to accommodate increases in the number of inputs, which corresponds to an increase in the number of computational tasks.

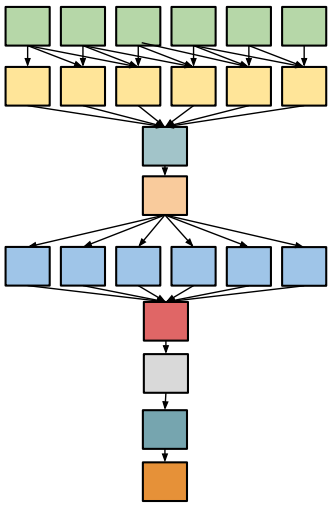

Figure 16: A simplified visualization of the Montage workflow.

Cybershake. CyberShake [23] is a seismology application that calculates Probabilistic Seismic Hazard curves for geographic sites in the Southern California region. It identifies all ruptures within 200km of the site of interest and convert rupture definition into multiple rupture variations with differing hypocenter locations and slip distributions. It then calculates synthetic seismograms for each rupture variance and peak intensity measures are then extracted from these synthetics and combined with the original rupture probabilities to produce probabilistic seismic hazard curves for the site. Figure 17 shows an illustration of the Cybershake workflow.

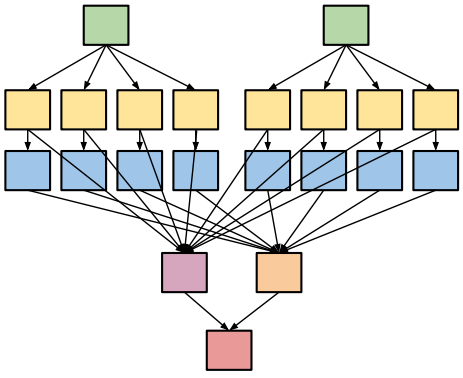

Figure 17: A simplified visualization of the CyberShake workflow.

Epigenomics. The Epigenomics workflow [24] is a pipeline workflow. Initial data are acquired from the Illumina-Solexa Genetic Analyzer in the form of DNA sequence lanes. Each Solexa machine can generate multiple lanes of DNA sequences. These data are converted into a format that can be used by sequence mapping software. The mapping software can do one of two major tasks. It either maps short DNA reads from the sequence data onto a reference genome, or it takes all the short reads, treats them as small pieces in a puzzle and then tries to assemble an entire genome. In our experiments, the workflow maps DNA sequences to the correct locations in a reference Genome. This generates a map that displays the sequence density showing how many times a certain sequence expresses itself on a particular location on the reference genome. Scientists draw conclusions from the density of the acquired sequences on the reference genome. Epigenome is a CPU-intensive application and its simplified structure is shown on Figure 18.

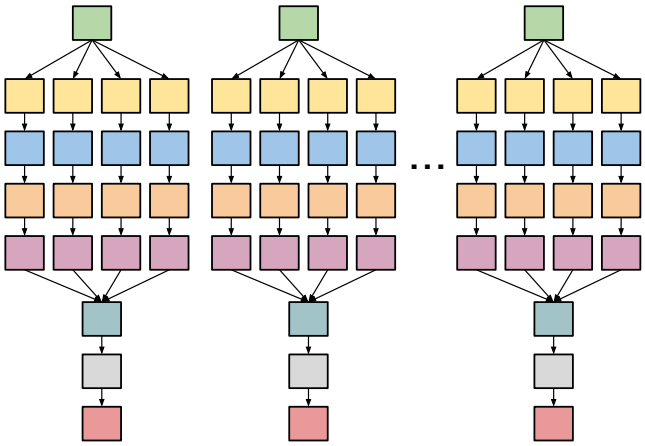

Figure 18: A simplified visualization of the Epigenomics workflow.

SIPHT. The SIPHT workflow [25] conducts a wide search for small untranslated RNAs (sRNAs) that regulates several processes such as secretion or virulence in bacteria. The kingdom-wide prediction and annotation of sRNA encoding genes involves a variety of individual programs that are executed in the proper order using Pegasus WMS. These involve the prediction of ρ-independent transcriptional terminators, BLAST (Basic Local Alignment Search Tools) comparisons of the inter genetic regions of different replicons and the annotations of any sRNAs that are found. A simplified structure of the SIPHT workflow is shown on Figure 19.

Table 3 shows the summary of the main workflows characteristics: number of tasks, average data size, and average task runtimes.

5.2. Task clustering techniques

In the experiments, we compare the performance of our balancing methods to the Horizontal Clustering (HC) [10] technique, and with two methods well known from the literature, DFJS [3] and AFJS [9]. In this subsection, we briefly describe each of these algorithms.

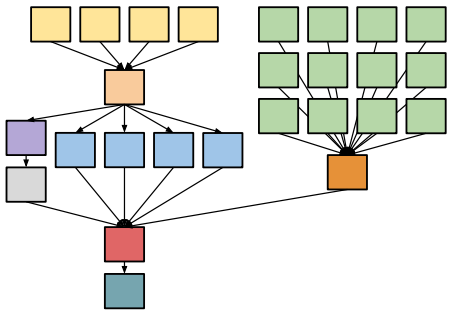

Figure 19: A simplified visualization of the SIPHT workflow.

Workflow	Number of Tasks	Average Data Size	Average Task Runtime
LIGO	800	5 MB	228s
Montage	300	3 MB	11s
CyberShake	700	148 MB	23s
Epigenomics	165	355 MB	2952s
SIPHT	1000	360 KB	180s

Table 3: Summary of the scientific workflows characteristics.

HC. Horizontal Clustering (HC) merges multiple tasks that are at the same horizontal level of the workflow. The clustering granularity (number of tasks within a cluster) of a clustered job is controlled by the user, who defines either the number of tasks per clustered job (*clusters.size*), or the number of clustered jobs per horizontal level of the workflow (*clusters.num*). This algorithm has been implemented and used in Pegasus WMS [10]. For simplicity, we define *clusters.num* as the number of available resources. In our prior work [15], we have compared the runtime performance of defining different clustering granularity. The pseudo-code of the HC technique is shown in Algorithm 1.

Algorithm 1 Horizontal Clustering algorithm.

Require: W: workflow; C: max number of tasks per job defined by *clusters.size* or *clusters.num*
1: **procedure** CLUSTERING(W, C)
2: **for** *level* < $depth(W)$ **do**
3: $TL \leftarrow$ GETTASKSATLEVEL(W, *level*) ▷ Partition W based on depth
4: $CL \leftarrow$ MERGE(TL, C) ▷ Returns a list of clustered jobs
5: $W \leftarrow W - TL + CL$ ▷ Merge dependencies as well
6: **end for**
7: **end procedure**
8: **procedure** MERGE(TL, C)
9: $J \leftarrow \{\}$ ▷ An empty job
10: $CL \leftarrow \{\}$ ▷ An empty list of clustered jobs
11: **while** TL is not empty **do**
12: J.add (TL.pop(C) ▷ Pops C tasks that are not merged
13: CL.add(J)
14: **end while**
15: **return** CL
16: **end procedure**

DFJS. The dynamic fine-grained job scheduler (DFJS) was proposed by Muthuvelu et al. [3]. The algorithm groups bag of tasks based on their granularity size—defined as the processing time of the task on the resource. Resources are ordered by their decreasing values of capacity (in MIPS), and tasks are grouped up to the resource capacity. This process continues until all tasks are grouped and assigned to resources. Algorithm 2 shows the pseudo-code of the heuristic.

Algorithm 2 DFJS algorithm.

Require: W: workflow; *max.runtime*: max runtime of clustered jobs
1: **procedure** CLUSTERING(W, *max.runtime*)
2: **for** *level* <the depth of W **do**
3: $TL \leftarrow$ GETTASKSATLEVEL(W, *level*) ▷ Partition W based on depth
4: $CL \leftarrow$ MERGE(TL, *max.runtime*) ▷ Returns a list of clustered jobs
5: $W \leftarrow W - TL + CL$ ▷ Merge dependencies as well
6: **end for**
7: **end procedure**
8: **procedure** MERGE(TL, *max.runtime*)
9: $J \leftarrow \{\}$ ▷ An empty job
10: $CL \leftarrow \{\}$ ▷ An empty list of clustered jobs
11: **while** TL is not empty **do**
12: $t \leftarrow TC$.pop() ▷ Get a task that is not mereged
13: **if** J.runtime + t.runtime > *max.runtime* **then**
14: CL.add(J)
15: $J \leftarrow \{\}$
16: **end if**
17: J.add(t)
18: **end while**
19: **return** CL
20: **end procedure**

AFJS. The adaptive fine-grained job scheduler (AFJS) [9] is an extension of DFJS. It groups tasks not only based on the maximum runtime defined per cluster job, but also on the cumulated data size per clustered job. The algorithm adds tasks to a clustered job until the job's runtime does not trespass the cumulated runtime or the cumulated data size thresholds. The AFJS heuristic pseudo-code is shown in Algorithm 3.

Algorithm 3 AFJS algorithm.

Require: W: workflow; *max.runtime*: the maximum runtime for a clustered jobs; *max.datasize*: the maximum data size for a clustered job
1: **procedure** CLUSTERING(W, *max.runtime*)
2: **for** *level* <the depth of W **do**
3: $TL \leftarrow$ GETTASKSATLEVEL(W, *level*) ▷ Partition W based on depth
4: $CL \leftarrow$ MERGE(TL, *max.runtime*, *max.datasize*) ▷ Returns a list of clustered jobs
5: $W \leftarrow W - TL + CL$ ▷ Merge dependencies as well
6: **end for**
7: **end procedure**
8: **procedure** MERGE(TL, *max.runtime*, *max.datasize*)
9: $J \leftarrow \{\}$ ▷ An empty job
10: $CL \leftarrow \{\}$ ▷ An empty list of clustered jobs
11: **while** TL is not empty **do**
12: $t \leftarrow TC$.pop() ▷ Get a task that is not mereged
13: **if** J.runtime + t.runtime > *max.runtime* OR J.datasize + t.datasize > *max.datasize* **then**
14: CL.add(J)
15: $J \leftarrow \{\}$
16: **end if**
17: J.add(t)
18: **end while**
19: **return** CL
20: **end procedure**

DFJS and AFJS require parameter tuning (e.g. maximum cumulated runtime per clustered job) to efficiently cluster tasks

into coarse-grained jobs. For instance, if the maximum runtime is too high, all tasks may be grouped into a single job, leading to loss of parallelism. In contrast, if the runtime threshold is too low, the algorithms do not group tasks, leading to no improvement over a control execution.

For the comparison purposes of this work, we perform a parameter study in order to tune the algorithms for each workflow application described in Section 5.1. Explore all possible parameter combinations is a cumbersome and exhaustive task. In the original DFJS and AFJS works, these parameters are empirically chosen, however this approach requires deep knowledge about the workflow applications. Instead, we performed a parameter tuning study, where we first estimate the upper bound of *max.runtime* (n) as the sum of all task runtimes, and the lower bound of *max.runtime* (m) as 1 second for simplicity. Data points are divided into ten chunks and sample one data point from each chunk. We then select the chunk that has the lowest makespan and set n and m as the upper and lower bounds of the selected chunk, respectively. These steps are repeated until n and m have converged into a data point.

We do not provide a mathematical proof of the correctness of our method, since we are not focused on demonstrate the optimal tuning for DFJS and AFJS algorithms, but a rough estimation of the minimal makespan. Instead, we show the relationship between the makespan and the *max.runtime* for an example Montage workflow application (Figure 20)—experiment conditions are presented in Section 5.3. Data points are divided into 10 chunks of 250s each. As the lower makespan values belongs to the first chunk, n is updated to 250, and m to 0. The process repeats until the convergence around 240s. Even though there are multiple local minimal makespan values, these data points are nearly close to each other, and the difference between their values (about seconds) is negligible.

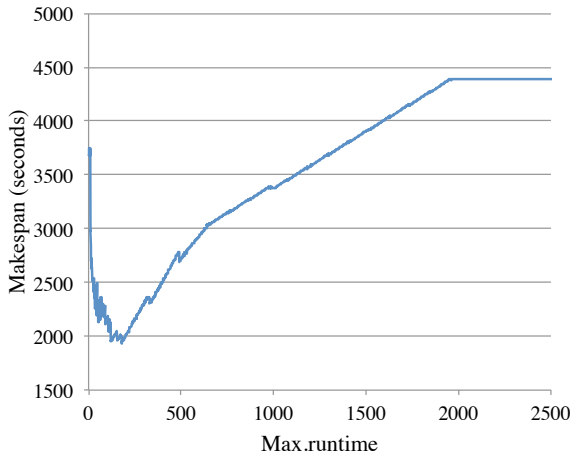

Figure 20: Relationship between the makespan of workflow and the specified maximum runtime in DFJS (Montage).

For simplicity, in the rest of this work we use DFJS* and AFJS* to indicate the best estimated performance of DFJS and AFJS respectively.

5.3. Experiment conditions

We adopt a trace-based simulation approach, where we extended the WorkflowSim [26] simulator with the balanced clustering methods and imbalance metrics to simulate a controlled distributed environment. WorkflowSim is a workflow simulator that extends from CloudSim [27] by providing support of task clustering, task scheduling and resource provisioning at the workflow level. It has been recently used in multiple workflow study areas [15, 28, 29] and its correctness has been verified in [26].

The simulated computing platform is composed by 20 single homogeneous core virtual machines (worker nodes), which is the quota per user of some typical distributed environments such as Amazon EC2 [30] and FutureGrid [31]. Amazon EC2 is a commercial, public cloud that is been widely used in distributed computing, in particular for scientific workflows [32]. FutureGrid is a distributed, high-performance testbed that provides scientists with a set of computing resources to develop parallel, grid, and cloud applications. Each simulated virtual machine (VM) has 512MB of memory and the capacity to process 1,000 million instructions per second. The default network bandwidth is 15MB according to the real environment in FutureGrid where our traces were collected. Task scheduling algorithm is data-aware, i.e. tasks are scheduled to resources which have the most input data available.

We collected workflow execution traces [2, 33] (including overhead and task runtime information) from real runs (executed on FutureGrid and Amazon EC2) of the scientific workflow applications described in Section 5.1. The traces are used to fill the Workflow Generator toolkit [34] to generate synthetic workflows to perform simulations with several different configurations under controlled conditions. The toolkit uses the information gathered from actual scientific workflow executions to generate synthetic workflows resembling those used by real world scientific applications. The number of inputs to be processed, the number of tasks in the workflow, and their composition determine the structure of the generated workflow.

Three sets of experiments are conducted. Experiment 1 evaluates the performance gain (μ) of our balancing methods (HRB, HIFB, and HDB) over a control execution (No Clustering). The goal of the experiment is to identify conditions where each method work best and worst. In addition, we also evaluate the performance gain of using workflow structure metrics (HRV, HIFV, and HDV), which require less *a-priori* knowledge from task and resource characteristics, over task clustering techniques from the literature (HC, DFJS*, and AFJS*).

Experiment 2 evaluates the performance impact of the variation of average data size and the number of resources available in our balancing methods for one scientific workflow application (LIGO). The original average data size (both input and output data) of the LIGO workflow is about 5MB as shown in Table 3. In this experiment, we increase the average data size up to 500MB to study the behavior of data intensive workflows. We control resource contingence by varying the number of available resources (VMs). High resource contingence is achieved by setting the number of available VMs to 5, which represent

less than 10% of the required resources to compute all tasks in parallel. On the other hand, low contingence is achieved when the number of available VMs is increased to 25, which represents about 50% of the required resources.

Experiment 3 evaluates the influence of combining our horizontal clustering methods with vertical clustering (VC). We compare the performance gain under four scenarios: (*i*) *VC-prior*, VC is first performed and then HRB, HIFB, or HDB; (*ii*) *VC-posterior*, horizontal methods are performed first and then VC; (*iii*) *No-VC*, horizontal methods only; and (*iv*) *VC-only*, no horizontal methods.

We define the performance gain over a control execution (μ) as the performance of the balancing methods related to the performance of an execution without clustering. Thus, for values of $\mu > 0$ our balancing methods perform better than the control execution. Otherwise, the balancing methods perform poorer.

5.4. Results and discussion

Experiment 1. Figure 21 shows the performance gain μ of the balancing methods for the five workflow applications over a control execution. All clustering techniques significantly improve (up to 48%) the runtime performance of all workflow applications, except HC for SIPHT. Cybershake and Montage workflows have the highest gain and nearly the same performance independent of the algorithm. This is due to their symmetric structure and low values for the imbalance metrics as shown in Table 4. Epigenomics and LIGO have higher variance of runtime and distance, thus the lower performance gain. In particular, each branch of the Epigenomics workflow (Figure 18) have the same number of pipelines, consequently *IF* values of tasks in the same horizontal level are the same. Therefore, HIFB cannot distinguish tasks from different branches, which leads the system to a dependency imbalance problem. In such cases, HDB captures the dependency between tasks and yields better performance. Furthermore, Epigenomics and LIGO workflows have high runtime variance, which has higher impact on the performance than data dependency. Last, the performance gain of our balancing methods is comparable (and in some cases better) to the tuned algorithms DFJS* and AFJS*.

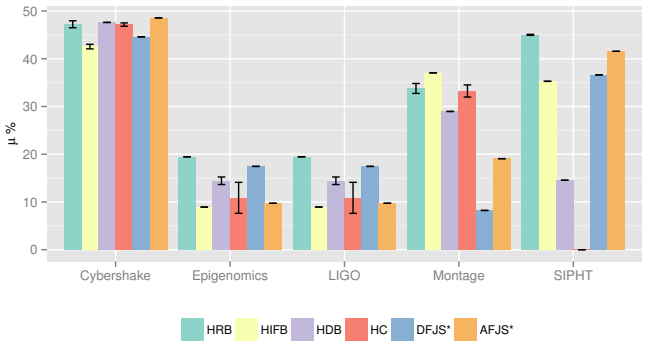

Figure 21: Experiment 1: performance gain (μ) over a control execution for six algorithms (* indicates the tuned performance of DFJS and AFJS).

Experiment 2. Figure 22 shows the performance gain μ of HRB, HIFB, HDB, and HC over a control execution for the LIGO Inspiral workflow. We chose LIGO due to its significant performance difference among these methods as shown in Figure 21. For small data sizes (up to 100 MB), the application is CPU-intensive and runtime variations have higher impact on the performance of the application. Thus, HRB performs better than any other balancing method. When increasing the data average size, the application turns into a data-intensive application, i.e. data dependencies have higher impact on the application's performance. HIFB captures both the workflow structure and task runtime information, which reduces data transfers between tasks and consequently yields better performance gain over the control execution. HDB captures the strong connections between tasks (data dependencies), while HIFB captures the weak connections (similarity in terms of structure). In some cases, HIFV is zero while HDV is less unlikely to be zero.

Most of the LIGO branches are like Figure 15, however, as mentioned in Section. 4.2, the LIGO workflow has a few branches that interwind with each other as shown in Figure 10. Since most branches are isolated from each other, HDB, initially performs well compared to HIFB. However, with the increase of average data size, the performance of HDB is more and more constrained by this phenomenon, which is shown in Figure 22.

HC has nearly constant performance despite of the average data size, due to its randomly fashion to merge tasks at the same horizontal level regardless of the runtime and data dependency information.

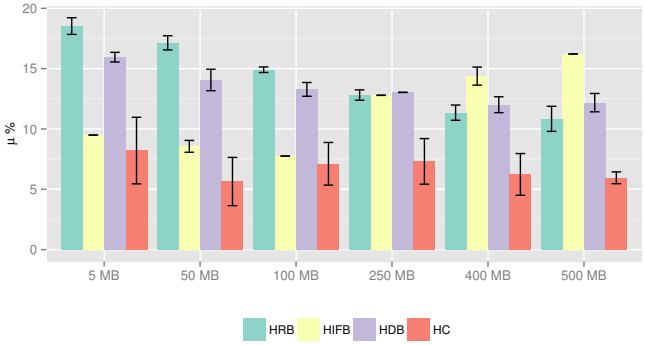

Figure 22: Experiment 2: performance gain (μ) over a control execution with different average data sizes for the LIGO workflow.

Figures 23 and 24 show the performance gain μ when varying the number of available VMs for the LIGO workflows with an average data size of 5MB (CPU-intensive) and 500MB (data-intensive) respectively. In high contention scenarios (small number of available VMs), all methods perform similar when the application is CPU-intensive (Figure 23), i.e., runtime variance and data dependency have smaller impact than the system overhead (e.g. queuing time). As the number of available resources increases, and the data is too small, runtime variance has more impact on the application's performance, thus HRB

performs better than the others. Note that as HDB captures strong connections between tasks, it is less sensitive to the runtime variations than HIFB, thus it yields better performance. For the data-intensive case (Figure 24), data dependency has more impact on the performance than the runtime variation. In particular, in the high contention scenario HRB performs poor clustering leading the system to data locality problems. However, the method still improves the execution due to the high system overhead. Similarly to the CPU-intensive case, under low contention, runtime variance increases its importance and then HRB performs better.

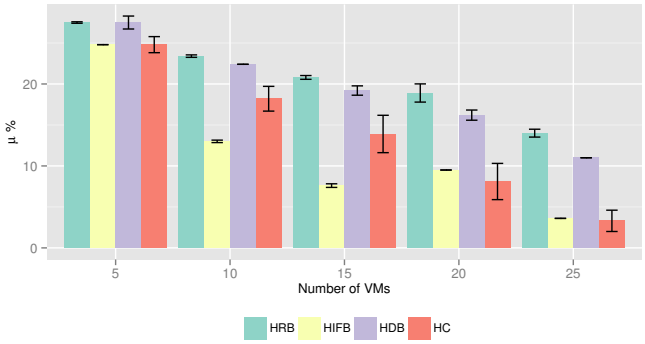

Figure 23: Experiment 2: performance gain (μ) over control execution with different number of resources for the LIGO workflow (average data size is 5MB).

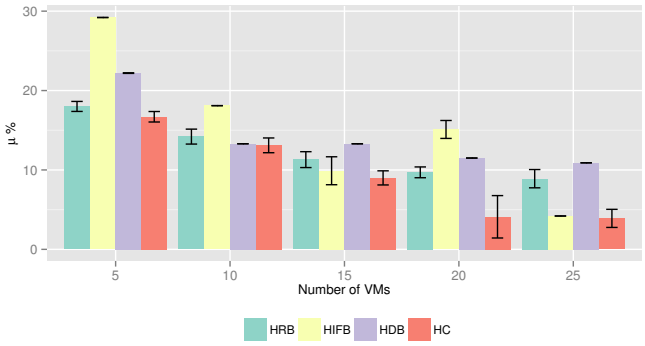

Figure 24: Experiment 2: performance gain (μ) over control execution with different number of resources for the LIGO workflow (average data size is 500MB).

Experiment 3. Figure 25 shows the performance gain μ for the Cybershake workflow over a control execution when using vertical clustering (VC) combined to our balancing methods. Vertical clustering does not aggregate any improvement to the Cybershake workflow (μ(*VC-only*) $\approx$ 0.2%), because the workflow structure has no explicit pipeline (see Figure 17). Similarly, VC does not improve the SIPHT workflow due to the lack of pipelines on its structure (Figure 19). Thus, results for this workflow are omitted.

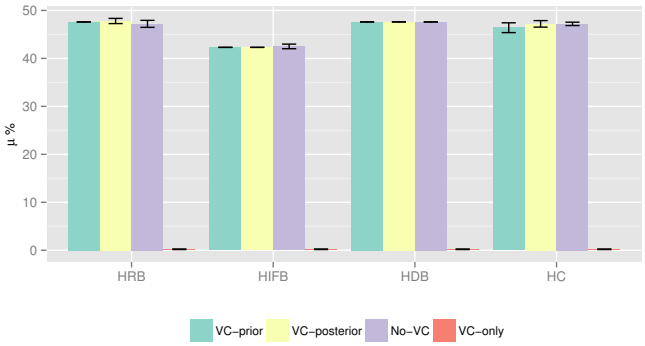

Figure 25: Experiment 3: performance gain (μ) for the Cybershake workflow over control execution when using vertical clustering (VC).

Figure 26 shows the performance gain μ for the Montage workflow. In this workflow, vertical clustering is often performed on the two pipelines (Figure 16). These pipelines are commonly single-task levels, thereby no horizontal clustering is performed on the pipelines. As a result, whether performing vertical clustering prior or after horizontal clustering, the result is about the same.

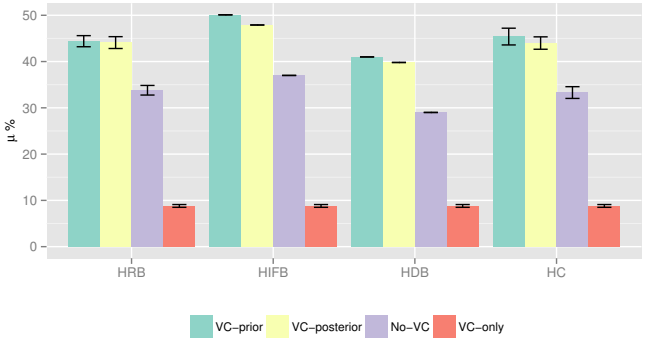

Figure 26: Experiment 3: performance gain (μ) for the Montage workflow over control execution when using vertical clustering (VC).

The performance gain μ for the LIGO workflow is shown in Figure 27. Vertical clustering yields better performance gain when it is performed prior to horizontal clustering (*VC-prior*). The LIGO workflow structure (Figure 15) has several pipelines that when primarily clustered vertically reduce system overheads (e.g. queuing and scheduling times). Furthermore, the runtime variance (HRV) of the clustered pipelines increases, thus the balancing methods, in particular HRB, evenly distribute task runtime among clustered jobs. When vertical clustering is performed *a posteriori*, pipelines are broken due to the horizontally merging of tasks between pipelines neutralizing vertical clustering improvements.

Similarly to the LIGO workflow, the performance gain μ values for the Epigenomics workflow (see Figure 28) are better

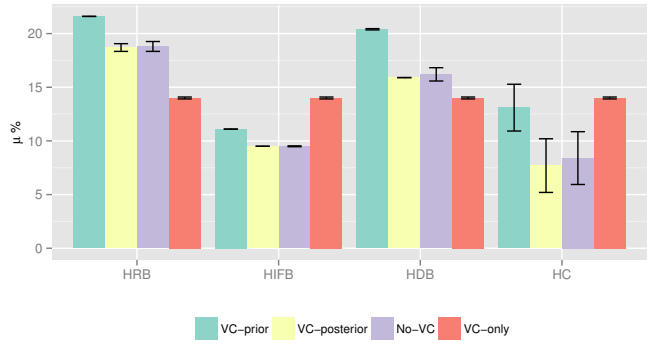

Figure 27: Experiment 3: performance gain (μ) for the LIGO workflow over control execution when using vertical clustering (VC).

when VC is performed *a priori*. This is due to several pipelines inherent to the workflow structure (Figure 18). However, vertical clustering has poorer performance if it is performed prior to the HDB algorithm. The reason is the average task runtime of Epigenomics is much larger than other workflows as shown in Table. 3. Therefore, VC-prior generates very large clustered jobs vertically and makes it difficult to horizontal methods to improve further.

not sure whether it is right

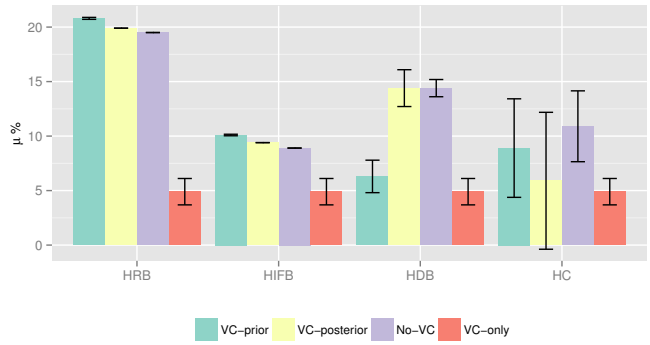

Figure 28: Experiment 3: performance gain (μ) for the Epigenomics workflow over control execution when using vertical clustering (VC).

	Tasks	HRV	HIFV	HDV
Level	(a) **CyberShake**			
1	4	0.309	0.03	1.22
2	347	0.282	0.00	0.00
3	348	0.397	0.00	26.20
4	1	0.000	0.00	0.00
Level	(b) **Epigenomics**			
1	3	0.327	0.00	0.00
2	39	0.393	0.00	578
3	39	0.328	0.00	421
4	39	0.358	0.00	264
5	39	0.290	0.00	107
6	3	0.247	0.00	0.00
7	1	0.000	0.00	0.00
8	1	0.000	0.00	0.00
9	1	0.000	0.00	0.00
Level	(c) **LIGO**			
1	191	0.024	0.01	10097
2	191	0.279	0.01	8264
3	18	0.054	0.00	174
4	191	0.066	0.01	5138
5	191	0.271	0.01	3306
6	18	0.040	0.00	43.70
Level	(d) **Montage**			
1	49	0.022	0.01	189.17
2	196	0.010	0.00	0.00
3	1	0.000	0.00	0.00
4	1	0.000	0.00	0.00
5	49	0.017	0.00	0.00
6	1	0.000	0.00	0.00
7	1	0.000	0.00	0.00
8	1	0.000	0.00	0.00
9	1	0.000	0.00	0.00
Level	(e) **SIPHT**			
1	712	3.356	0.01	53199
2	64	1.078	0.01	1196
3	128	1.719	0.00	3013
4	32	0.000	0.00	342
5	32	0.210	0.00	228
6	32	0.000	0.00	114

Table 4: Experiment 1: average number of tasks, and average values of imbalance metrics (HRV, HIFV, and HDV) for the 5 workflow applications.

6. Future Work

References

[1] R. Ferreira da Silva, G. Juve, E. Deelman, T. Glatard, F. Desprez, D. Thain, B. Tovar, M. Livny, Toward fine-grained online task characteristics estimation in scientific workflows, in: Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science, WORKS '13, ACM, 2013, pp. 58–67. doi:10.1145/2534248.2534254.

[2] W. Chen, E. Deelman, Workflow overhead analysis and optimizations, in: Proceedings of the 6th workshop on Workflows in support of large-scale science, WORKS '11, 2011, pp. 11–20.

[3] N. Muthuvelu, J. Liu, N. L. Soe, S. Venugopal, A. Sulistio, R. Buyya, A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids, in: Proceedings of the 2005 Australasian workshop on Grid computing and e-research - Volume 44, 2005, pp. 41–48.

[4] N. Muthuvelu, I. Chai, C. Eswaran, An adaptive and parameterized job grouping algorithm for scheduling grid jobs, in: Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on, Vol. 2, 2008, pp. 975 –980.

[5] N. Muthuvelu, I. Chai, E. Chikkannan, R. Buyya, On-line task granularity adaptation for dynamic grid applications, in: Algorithms and Architectures for Parallel Processing, Vol. 6081 of Lecture Notes in Computer Science, 2010, pp. 266–277.

[6] N. Muthuvelu, C. Vecchiolab, I. Chaia, E. Chikkannana, R. Buyyab, Task granularity policies for deploying bag-of-task applications on global grids, Future Generation Computer Systems 29 (1) (2012) 170 – 181.

[7] W. K. Ng, T. Ang, T. Ling, C. Liew, Scheduling framework for bandwidth-aware job grouping-based scheduling in grid computing, Malaysian Journal of Computer Science 19 (2) (2006) 117–126.

[8] T. Ang, W. Ng, T. Ling, L. Por, C. Lieu, A bandwidth-aware job grouping-based scheduling on grid environment, Information Technology Journal 8 (2009) 372–377.

[9] Q. Liu, Y. Liao, Grouping-based fine-grained job scheduling in grid computing, in: First International Workshop on Education Technology and Computer Science, Vol. 1, 2009, pp. 556 –559.

[10] G. Singh, M.-H. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D. S. Katz, G. Mehta, Workflow task clustering for best effort systems with pegasus, in: 15th ACM Mardi Gras Conference, 2008, pp. 9:1–9:8.

[11] R. Ferreira da Silva, T. Glatard, F. Desprez, On-line, non-clairvoyant optimization of workflow activity granularity on grids, in: F. Wolf, B. Mohr, D. Mey (Eds.), Euro-Par 2013 Parallel Processing, Vol. 8097 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 255–266. doi:10.1007/978-3-642-40047-6_28.

[12] W. Chen, E. Deelman, R. Sakellariou, Imbalance optimization in scientific workflows, in: Proceedings of the 27th international ACM conference on International conference on supercomputing, ICS '13, 2013, pp. 461–462.

[13] J. Lifflander, S. Krishnamoorthy, L. V. Kale, Work stealing and persistence-based load balancers for iterative overdecomposed applications, in: Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing, HPDC '12, 2012, pp. 137–148.

[14] W. Chen, E. Deelman, Integration of workflow partitioning and resource provisioning, in: Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on, 2012, pp. 764–768.

[15] W. Chen, R. Ferreira da Silva, E. Deelman, R. Sakellariou, Balanced task clustering in scientific workflows, in: eScience (eScience), 2013 IEEE 9th International Conference on, 2013, pp. 188–195. doi:10.1109/eScience.2013.40.

[16] G. Zheng, A. Bhatelé, E. Meneses, L. V. Kalé, Periodic hierarchical load balancing for large supercomputers, Int. J. High Perform. Comput. Appl. 25 (4) (2011) 371–385.

[17] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. A. Hensgen, R. F. Freund, A comparison of eleven static heuristic for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed Computing 61 (6) (2001) 810–837.

[18] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, D. S. Katz, Pegasus: A framework for mapping complex scientific workflows onto distributed systems, Sci. Program. 13 (3) (2005) 219–237.

[19] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, Jr., H.-L. Truong, Askalon: a tool set for cluster and grid computing: Research articles, Concurr. Comput. : Pract. Exper. 17 (2-4) (2005) 143–169.

[20] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, C. Wroe, Taverna: lessons in creating a workflow environment for the life sciences: Research articles, Concurr. Comput. : Pract. Exper. 18 (10) (2006) 1067–1100.

[21] D. Brown, P. Brady, A. Dietz, J. Cao, B. Johnson, J. McNabb, A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows for e-Science, Springer London, 2007, pp. 39–59. doi:10.1007/978-1-84628-757-2_4.

[22] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, M. Su, Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand, in: SPIE Conference on Astronomical Telescopes and Instrumentation, Vol. 5493, 2004, pp. 221–232. doi:10.1117/12.550551.

[23] R. Graves, T. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, D. Okaya, P. Small, K. Vahi, CyberShake: A Physics-Based Seismic Hazard Model for Southern California, Pure and Applied Geophysics 168 (3-4) (2011) 367–381. doi:10.1007/s00024-010-0161-6.

[24] USC Epigenome Center, `http://epigenome.usc.edu`.

[25] SIPHT, `http://pegasus.isi.edu/applications/sipht`.

[26] W. Chen, E. Deelman, Workflowsim: A toolkit for simulating scientific workflows in distributed environments, in: E-Science (e-Science), 2012 IEEE 8th International Conference on, 2012, pp. 1–8. doi:10.1109/eScience.2012.6404430.

[27] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.

[28] W. Chen, E. Deelman, Fault tolerant clustering in scientific workflows, in: Services (SERVICES), 2012 IEEE Eighth World Congress on, 2012, pp. 9–16.

[29] F. Jrad, J. Tao, A. Streit, A broker-based framework for multi-cloud workflows, in: Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds, ACM, 2013, pp. 61–68.

[30] Amazon.com, Inc., Amazon Web Services, `http://aws.amazon.com`. URL `http://aws.amazon.com`

[31] FutureGrid, `http://futuregrid.org/`.

[32] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, Scientific workflow applications on amazon ec2, in: In Cloud Computing Workshop in Conjunction with e-Science, IEEE, 2009.

[33] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, Vol. 29, 2013, pp. 682 – 692, special Section: Recent Developments in High Performance Computing and Security. doi:http://dx.doi.org/10.1016/j.future.2012.08.015.

[34] Workflow Generator, `https://confluence.pegasus.isi.edu/display/pegas`