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 two 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 [**?**]. To minimize the impact of such overheads, task clustering techniques [2, 3, 4, 5, 6, 7, 8, 9**?**] 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 [**?**]. 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 [10]. A common technique to handle load imbalance is overdecomposition [11]. 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.

In this work, 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 as our first contribution. A performance evaluation study shows that the methods can significantly reduce the imbalance problem. However, there is a tradeoff between runtime and data dependency balancing. For instance, balancing runtime may aggravate the Dependency Imbalance problem, and vice versa. A quantitative measurement of workflow characteristics is required to serve as a criterion to select and balance these solutions. To achieve this goal, we propose a series of metrics that reflect the internal structure (in terms of task runtimes and dependencies) of the workflow as our second contribution.

In particular, we provide a novel approach to capture these metrics. Traditionally, there are two approaches to improve the

*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)

performance of task clustering. The first one is a top-down approach [12] 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 an optimization problem does not scale well since most methods use genetic algorithms. The second one is a bottom-up approach [2, 8] 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.

Our third contribution is an analysis of the quantitative metrics and balancing methods. These metrics characterize the workflow imbalance problem. A balancing method, or a combination of those, is selected through the comparison of the relative values of these metrics.

To the best of our knowledge, this study is the first example of task granularity control that considers runtime variance and data dependency. The next section gives an overview of the related work. Section 3 presents our workflow and execution environment models, Section **??** 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. [2] 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 [3] to determine task granularity based on task file size, CPU time, and resource constraints. Recently, they proposed an online scheduling algorithm [4, 5] that groups tasks based on resource network utilization, user's budget, and application deadline. Ng et al. [6] and Ang et al. [7] 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 [8] 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. [9] 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. [**?**] 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. [11] proposed to use work stealing and a hierarchical persistence-based rebalancing algorithm to address the imbalance problem in scheduling. Zheng et al. [13] 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 [14] (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. Fig. 1 (left) shows an illustration of a DAG composed by four tasks. This model fits several workflow management systems such as Pegasus [15], Askalon [16], and Taverna [17].

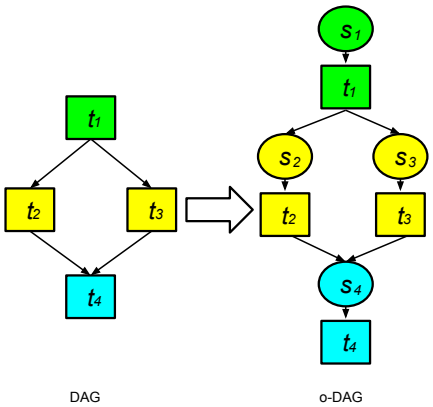

Figure 1: Extending DAG to o-DAG.

Fig. 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:

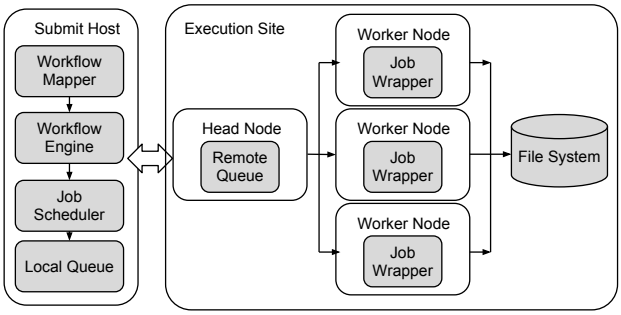

Figure 2: A workflow system model.

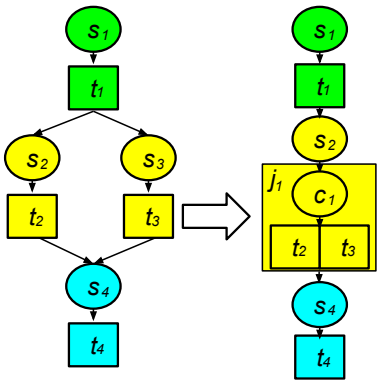

Figure 3: An Example of Horizontal Clustering. (Color indicates the horizontal level of a task)

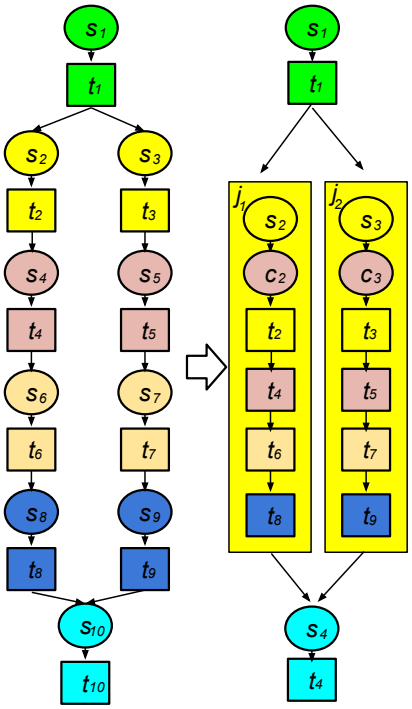

Figure 4: An Example of Vertical Clustering.

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. This is called **Task Clustering**. A job is a single execution unit in the workflow execution systems and may contain 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 the necessary actions. The time period when a job is free (all of its parents have completed successfully) to when it is submitted to the job scheduler is denoted the workflow engine delay. The workflow engine delay is usually configured by users to assure that the entire workflow scheduling and execution system is not overloaded.

Job Scheduler and Local Queue. manage individual workflow jobs and supervise their execution on local and remote resources. The time period 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.

More specifically, in this paper we focus on two types of task clustering: horizontal clustering and vertical clustering. **Horizontal Clustering** (HC) merges multiple tasks that are at the same horizontal level of the workflow, in which horizontal level of a task is defined as the furthest distance from the root task to this task. **Vertical Clustering** (VC) merges tasks at the same pipeline of the workflow. Tasks at the same pipeline share a single-parent-single-child relationship, which means one task (t_a) is the only parent of the other task (t_b) and t_b is the only child of t_a.

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 [18].. Fig. 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 our work, 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 [18].

With an o-DAG model, we can explicitly express the process of task clustering. Fig. 3 shows a simple example of how we perform horizontal clustering, in which two tasks t_1 and t_2, 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

Table 1: Overhead (in seconds) and Runtime Information

Workflow	Venue	Nodes	Tasks	Workflow Engine Delay	Queue Delay	Task Runtime
SIPHT	UW Madison	8	33	17	69	20
Broadband	Amazon EC2	8	770	17	945	308
Epigenomics	Amazon EC2	8	83	6	311	158
CyberShake	Skynet	5	24142	12	188	5
Periodogram	OSG	39	235300	464	2230	22
Montage	USC	20	10427	182	26136	523

by the job scheduler. After horizontal clustering, t_1 and t_2 in j_1 can be executed in sequence or in parallel, if supported. In this paper, we consider sequential executions only. Given a single resource, the overall runtime for the workflow in Fig. 3 (left) is $runtime_1 = s_1 + t_1 + s_2 + t_2$, and the overall runtime for the clustered workflow in Fig. 3 (right) is $runtime_2 = s_1 + c_1 + t_1 + t_2$. $runtime_1 > runtime_2$ as long as $c_1 < s_2$, 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. Fig. 4 illustrates an example of vertical clustering, in which t_2, t_4, t_6 and t_8 are merged into j_1 while t_3, t_5, t_7 and t_9 are merged into j_2. Similarly, clustering delay 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. Task clustering has been applied to many scenarios where resources are much less than tasks, which is true for many scientific workflows [19, 20, 21], and has achieved significant improvement (i.e., 97% as reported by [19]) over the case without clustering. Table 1 shows the statistical workflow information (average task runtime etc.) of six widely used scientific applications and their runtime information (number of nodes, average queue delay, etc.) [18]. For all of these workflows, there are a lot more tasks than available nodes and the average task runtime is shorted than system overheads. Therefore, task clustering can achieve significant improvement over no clustering. Besides the benefits of runtime improvement, task clustering also allows us to run on some federated distributed environment. For example, FutureGrid [22] allows a user to use up to 20 VMs at a time. We will introduce the details of these workflows and distributed platforms in Session 5

4. Balanced Clustering

Task clustering has been widely used to group fined-grained tasks into coarse-grained jobs [2, 3, 4, 5, 6, 7, 8, 23] . However, the challenge of balancing the runtime imbalance and dependency imbalance is still not yet 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 [10].

In another way, 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.

However, what makes it even difficult is there is a tradeoff between runtime and data dependency balancing. For instance, balancing runtime may aggravate the dependency imbalance problem, and vice versa. A quantitative measurement of workflow characteristics is required to serve as a criterion to select and balance these solutions. To achieve this goal, we propose a series of metrics that reflect the internal structure (in terms of task runtimes and dependencies) of the workflow and introduce balancing methods based on these metrics to address these challenges.

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 [12] 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 an optimization problem does not scale well since most methods use genetic algorithms. The second one is a bottom-up approach [2, 8] 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.

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 ratio of the standard deviation in task runtime to the average runtime of tasks/jobs 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. Fig. 5 shows an example of four independent tasks t_1, t_2, t_3 and t_4 where task runtime of t_1 and t_2 is half of that of t_3 and t_4. 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., $HRV > 0$ (Fig. 5-top). In contrast, a balanced clustering strategy should try its best to evenly distribute task runtime among jobs as shown in Fig. 5 (bottom). Generally speaking, a smaller *HRV* means that the runtime of tasks at the same 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 the tasks.

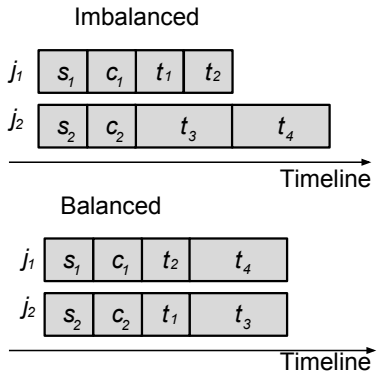

Figure 5: An Example of Runtime Variance.

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 Fig. 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 Fig. 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 intuition behind the Impact Factor is that we aim to capture the similarity of tasks/jobs in a graph by measuring their relative impact factor or importance to the entire graph. Intuitively speaking, tasks with similar impact factors should be merged together compared to tasks with different impact factors. Also, if all the tasks have similar impact factors, 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,

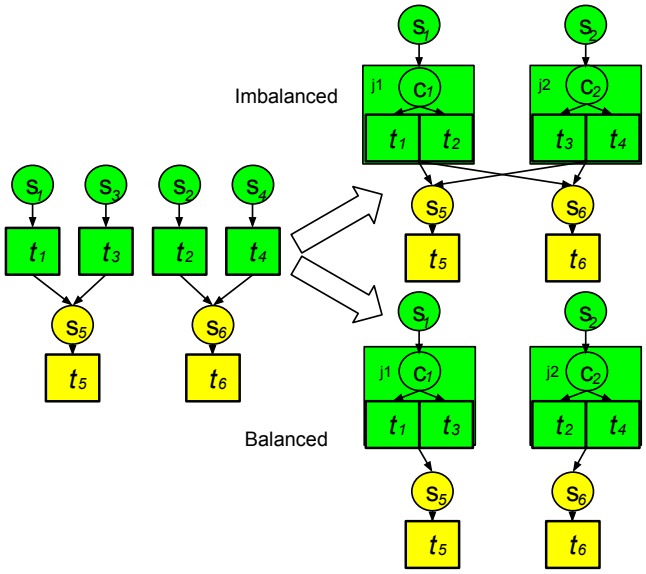

Figure 6: An Example of Dependency Variance.

we assume the *IF* of a workflow exit task (e.g. t_5 in Fig. 6) as 1.0. For instance, consider the two workflows presented in Fig. 7. *IF* for t_1, t_2, t_3, and t_4 are computed as follows:

$$IF(t_7) = 1.0, IF(t_6) = IF(t_5) = IF(t_7)/2 = 0.5$$
$$IF(t_1) = IF(t_2) = IF(t_5)/2 = 0.25$$
$$IF(t_3) = IF(t_4) = IF(t_6)/2 = 0.25$$

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 Fig. 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 $O(n)$.

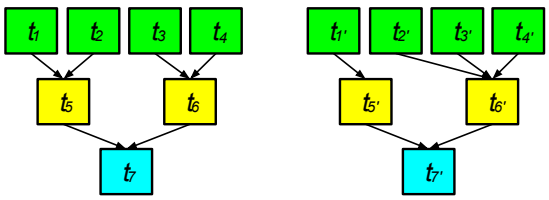

Figure 7: Example of workflows with different data dependencies.

Distance Variance (*DV*) describes how 'closely' tasks are to 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 2 shows the distance matrices of tasks from the first level for both workflows of Fig. 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. Intuitively speaking, a smaller *HDV* means the tasks at the same horizontal level are more equally 'distant' to each other and thus the workflow structure tends to be more 'evenly' and 'regular'.

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

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 2: Distance matrices of tasks from the first level of workflows in Fig. 7.

4.2. Balanced Clustering Methods

In this subsection, we introduce our balanced clustering methods used to improve the runtime balance and dependency balance in task clustering. We first introduce the basic runtime-based clustering method and then two other balancing methods that address the dependency imbalance problem. We use the metrics presented in the previous subsection to evaluate a given workflow to decide which balancing method(s) is(are) more appropriate.

Algorithm 1 shows the pseudocode of our balanced clustering algorithm that uses a combination of these balancing methods and metrics. The maximum number of clustered jobs (size of *CL*) is equal to the number of available resources multiplied by a *clustering factor*.

We examine tasks in a level-by-level approach starting from the level with the largest width (number of tasks at the same level, `line 2`). The intuition behind this breadth favored approach is that we believe it should improve the performance most. Then, we determine which type of imbalance problem a workflow experiences based on the balanced clustering metrics presented previously (*HRV*, *HIFV*, and *HDV*), and accordingly, we select a combination of balancing methods. GetCandidateJob selects a job (`line 12`) from a list of potential candidate jobs (*CL*) to be merged with the targeting task (t).

Algorithm 1 Balanced Clustering algorithm

Require: W: workflow; CL: list of clustered jobs; C: the required size of CL;
Ensure: The job runtime of CL are as even as possible

```
1: procedure Clustering(W, D, C)
2:     Sort W in decreasing order of the size of each level
3:     for level <the depth of W do
4:         TL ← GetTasksAtLevel(w, level)      ▷ Partition W based on depth
5:         CL ← Merge(TL, C)                   ▷ Form a list of clustered jobs
6:         W ← W − TL + CL                     ▷ Merge dependencies as well
7:     end for
8: end procedure
9: procedure Merge(TL, C)
10:    Sort TL in decreasing order of task runtime
11:    for t in TL do
12:        J ← GetCandidateJob(CL, t)          ▷ Get a candidate task
13:        J ← J + t                           ▷ Merge it with the clustered job
14:    end for
15:    return CL
16: end procedure
17: procedure GetCandidateJob(CL, t)
18:    Selects a job based on balanced clustering methods
19: end procedure
```

Below we introduce the three balancing methods proposed in this work.

Horizontal Runtime Balancing (HRB) aims to evenly distribute task runtime among 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. Fig. 8 shows how HRB works in an example of four jobs with different job runtime (assuming the height of a job is its runtime). For the given task (t_0), HRB sorts the potential jobs (j_1, j_2, j_3, and j_4) based on their runtime and selects the shortest job (in this case j_1 or j_2).

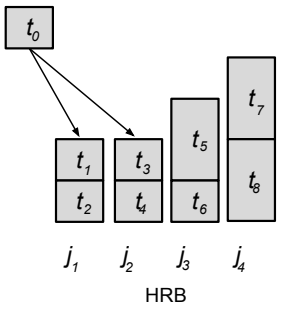

Figure 8: An example of HRB.

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 Fig. 9, assuming 0.2, 0.2, 0.1, and 0.1 IF values of j_1, j_2, j_3, and j_4 respectively, HIFB selects a list of candidate jobs with the same IF value, i.e. j_3 and j_4.

Then, HRB is performed to select the shortest job (j_3).

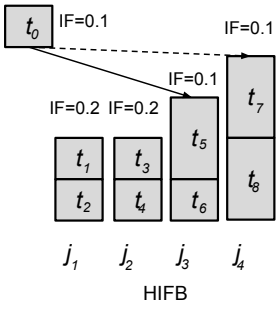

Figure 9: An example of HIFB.

Similarly, in HDB jobs are sorted based on the distance between them and the targeted task t_0, then on their runtimes. For instance, in Fig. 10, assuming 2, 4, 4, and 2 the distances to j_1, j_2, j_3, and j_4 respectively, HDB selects a list of candidate jobs with the minimal distance (j_1 and j_4). Then, HRB is performed to select the shortest job (j_1).

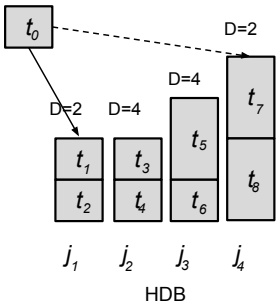

Figure 10: An example of HDB.

In conclusion, these balancing methods have different preference on the selection of a candidate job to be merged with the targeting task. HIFB tends to group tasks that share similar position/importance to the workflow structure. HDB tends to group tasks that are closed to each other to reduce data transfers.

5. Experiments

We applied these imbalance metrics and balanced clustering methods to four widely used workflows in our experiments. We adopt a traced based simulation to evaluate our these metrics and methods. We first extended the WorkflowSim [24] simulator with the balanced clustering methods and imbalance metrics to simulate a distributed environment where we could evaluate the performance of our methods when varying the average data size and task runtime. 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 [25] and FutureGrid [22]. Amazon EC2 is a commercial, public cloud that is been widely used in distributed computing. FutureGrid is a distributed, high-performance testbed that provides scientists with a set of computing resources to develop parallel, grid, and cloud applications. We then collected traces of these workflows (including overhead information and task runtime information) from real runs of workflows that were executed on FutureGrid and Amazon EC2. With these traces based information, we use WorkflowSim to vary algorithms used, resources, workflow size etc. to illustrate the different runtime performance of these methods.

Each simulated VM has 512MB of memory and the capacity to process 1,000 million instructions per second. Task scheduling algorithm is data-aware, i.e. tasks are scheduled to resources which have the most input data available.

The experiments presented hereafter evaluate the performance of our balancing methods in comparison with an existing and effective task clustering strategy named Horizontal Clustering (HC) [9], which is widely used by workflow management systems such as Pegasus. We also compare our methods with DFJS proposed by Muthuvelu et al. [2] that groups bag of tasks based on their runtimetasks are grouped up to the resource capacity and its extended version AFJS proposed by Liu and Liao [8], which is an adaptive fine-grained job scheduling algorithm to group fine-grained tasks according to processing capacity and bandwidth of the current available resources. However, DFJS and AFJS requires parameter tuning (resource cap, i.e.) and therefore in this work we use their best performance that is already tuned. In practice, these algorithms are not feasible to use since parameter tuning requires a lot of prior knowledge and it takes much time to search the parameter space without proper heuristics. For example, if the resource capacity is too high, all of the tasks may be grouped together and there is loss of parallelism. If the resource capacity is too low, the algorithms do not group tasks at all, which leads to no improvement over no clustering. For simplicity, we use DFJS* and AFJS* to indicate the best performance of them in the rest of our paper.

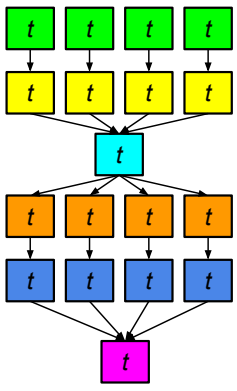

Figure 11: A simplified visualization of the LIGO Inspiral workflow

Four workflows are used in the experiments: LIGO [1] Inspiral analysis, Montage [26], CyberShake [27] and Epigenomics [28].

Table 3: Workflow Information

Workflow	Tasks	Avg. Data Size (MB)	Avg. Task Runtime (secs)
Epigenomics	165	355	2952
CyberShake	700	148	23
LIGO	800	5	228
Montage	300	3	11

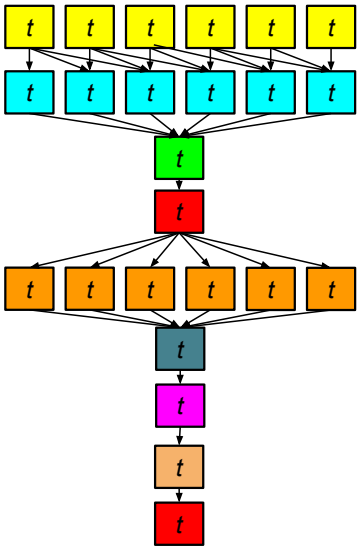

Figure 12: A simplified visualization of the Montage workflow

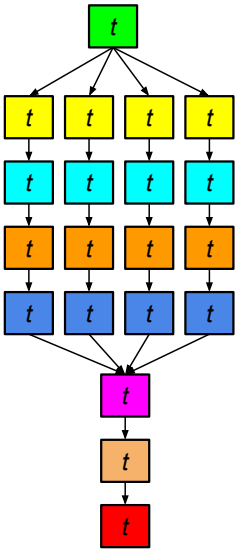

Figure 13: A simplified visualization of the Epigenomics workflow

LIGO (Laser Interferometer Gravitational Wave Observatory) 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 relativityEinstein's theory of gravityin which gravity is described as due to the curvature of the fabric of time and space. LIGO Inspiral workflow is a data intensive workflow.

CyberShake 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.

Montage 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.

The Epigenomics workflow 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.

Fig. 11, 12, 13 and 14 show their simplified workflow structures respectively. For simplicity, we only show one of their branch. All of these workflows are generated and varied using the Workflow Generator [29]. Runtime (average and task runtime distribution) and overhead (workflow engine delay, queue delay, and network bandwidth) information were collected from

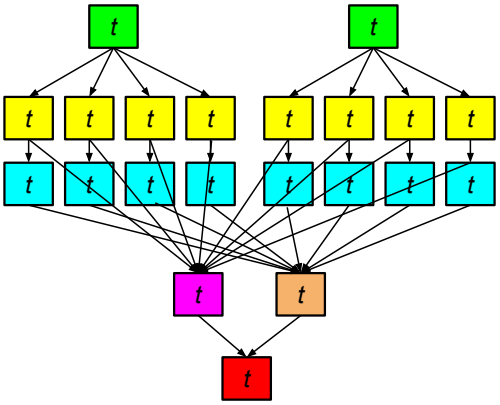

Figure 14: A simplified visualization of the CyberShake workflow

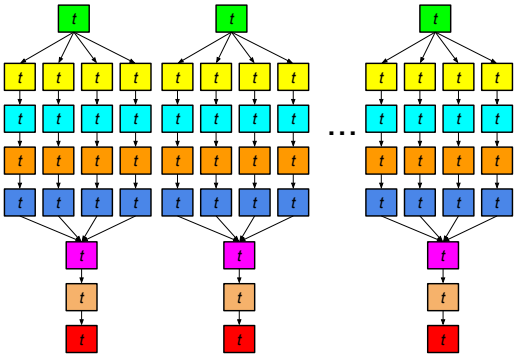

Figure 15: A visualization of the Epigenomics workflow

real traces production environments [18, 30], then used as input parameters for the simulations.

Table 3 show their statistical information.

Three sets of experiments are conducted.

Experiment 1 evaluates the performance gain over No Clustering (μ) with our balancing methods (HIFB, HDB, HRB) and existing HC, DFJS* and AFJS* algorithms. Since DFJS requires one parameter (resource capacity) and AFJS requires two parameters (resource capacity and bandwidth capacity), we search all possible parameters to use their optimal performance gain. In practice, their performance should be equal to or lower than the optimal values.

Experiment 2 evaluates the influence of average data size, number of VMs and workflow size (the number of tasks of a workflow) in our balancing methods for one workflow (LIGO). The original average data size (both input and output data) of the LIGO workflow is about 5MB as shown in Table 3, and we increase it to 500MB to simulate the case with data intensive workflows.

Experiment 3 evaluates the influence of combining our horizontal clustering methods with vertical clustering. We compare the performance gain under four scenario, VC-prior: we perform VC first and then HIFB, HDB, HDB or HDC; VC-posterior: we perform horizontal methods and then VC; no VC: horizontal methods only; and VC only. The motivation behind this experiment is that we believe VC will change imbalance metrics (HIFV, HDV and HRV) and we aim to show how these metrics can help us understand the performance of VC better.

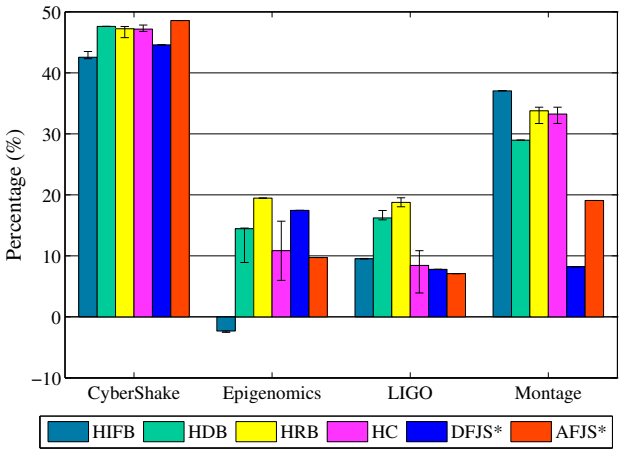

Figure 16: Performance Gain over No Clustering of Six Algorithms (* indicates the best performance of DFJS and AFJS)

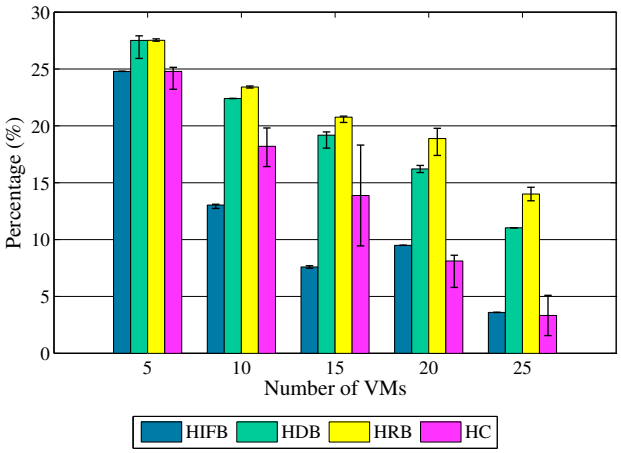

Figure 18: Performance Gain over No Clustering with Different Number of Resources (Average data size is 5MB).

5.1. Results and Discussion

Experiment 1: Fig. 16 shows the performance gain over No Clustering (NC) μ of the balancing methods for the four workflows. We have a few conclusions from Fig. 16: (1). In general, all of these horizontal clustering methods improve the runtime performance significantly (up to 48%) except for the case using HIFB in Epigenomics. The reason is that each branch of Epigenomics happen to have the same number of pipelines as Fig. 15 shows. In such case, the IFs of each task at the same horizontal level is the same and thus the HIFB cannot distinguish tasks and their dependency variance as shown in Table. 5. (2). Among

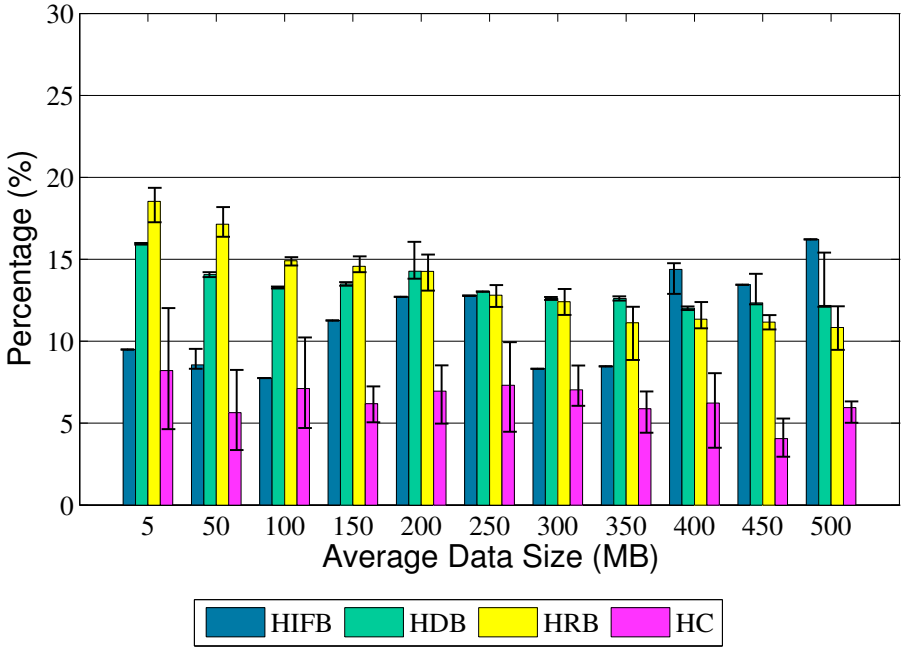

Figure 17: Performance Gain over No Clustering with Different Data Size

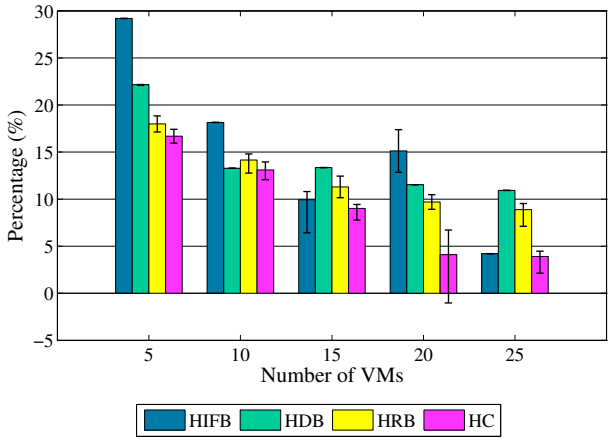

Figure 19: Performance Gain over No Clustering with Different Number of Resources (Average data size is 500MB)).

the four workflows, we see more significant improvement for the CyberShake workflow and the Montage workflow compared to Epigenomics and LIGO. The reason is Epigenomics and LIGO have a relatively short task runtime compared to the system overheads and task clustering can significantly improve the overall runtime. (3). Among the five methods, in most of the cases, it is clear that our methods (HIFB, HDB and HRB) perform better than HC, DFJS* and AFJS*. The reason is that they don't take the runtime variance and dependency variance into consideration. As to the comparison among HIFB, HDB and HRB, we will show in Experiment 2.

Experiment 2: Fig. 17 shows the performance gain of HIFB, HDB, HRB and HC for the LIGO Inspiral workflow. The reason we chose the LIGO Inspiral workflow is that it has most significant difference among these methods. It is clear that with the increase of average data size, HIFB performs better and better while HRB performs worse. We don't see much difference of HC and HDB while varying data size. The reason is HIFB is able to capture the structural and runtime information, reducing data transfers between tasks, while HRB focuses on runtime distribution only. Therefore, while the workflow is more data intensive, we observe better performance of HIFB and a worse performance of HRB. HC does not change much since it is randomly merges tasks at the same horizontal level without information about runtime or data dependency. HDB does not change much since its performance gain is already good enough.

Table 4: Imbalance Metrics (Montage)

Level	Tasks	HRV	HIFV	HDV
1	49	0.022	0.007	189.170
2	196	0.010	0	0
3	1	0	0	0
4	1	0	0	0
5	49	0.017	0	0
6	1	0	0	0
7	1	0	0	0
8	1	0	0	0
9	1	0	0	0

Fig. 18 and Fig. 19 vary the number of VMs with an average data size of 5MB and 500MB respectively. We can see that with the increase of VMs, the performance gain of all of these

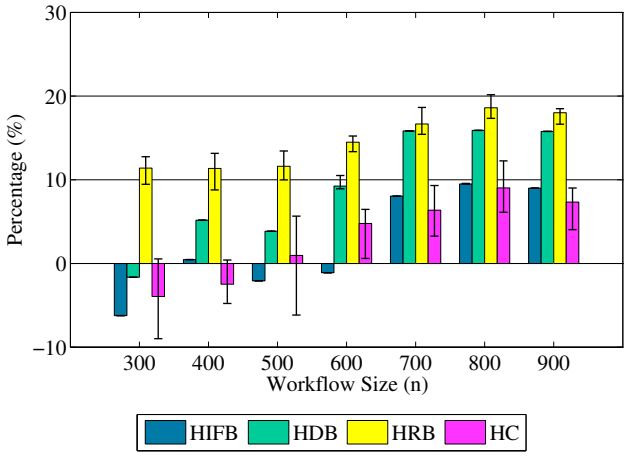

Figure 20: Performance Gain over No Clustering with different Workflow Sizes

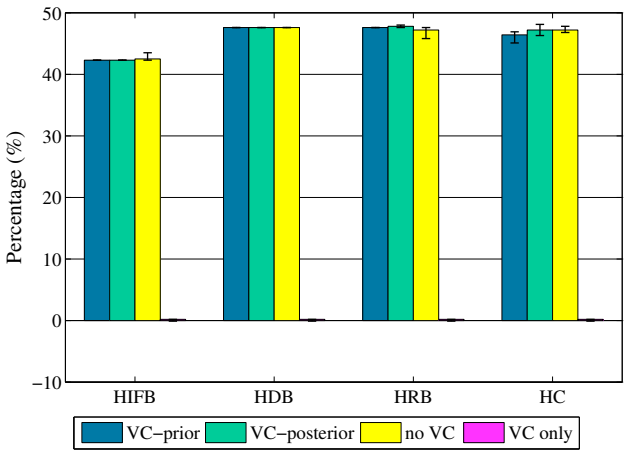

Figure 21: Performance Gain over No Clustering for Vertical Methods (CyberShake)

Table 5: Imbalance Metrics (Epigenomics)

Level	Tasks	HRV	HIFV	HDV
1	3	0.327	0	0
2	39	0.393	0	578
3	39	0.328	0	421
4	39	0.358	0	264
5	39	0.290	0	107
6	3	0.247	0	0
7	1	0	0	0
8	1	0	0	0
9	1	0	0	0

methods decrease while HIFB and HDB decrease more significantly than HRB and HC. The reason is similar to the case of Fig. 17 since HIFB and HDB are data dependency aware. The difference between HDB and HIFB is HDB captures the strong connections between tasks (data dependencies) and HIFB cap-

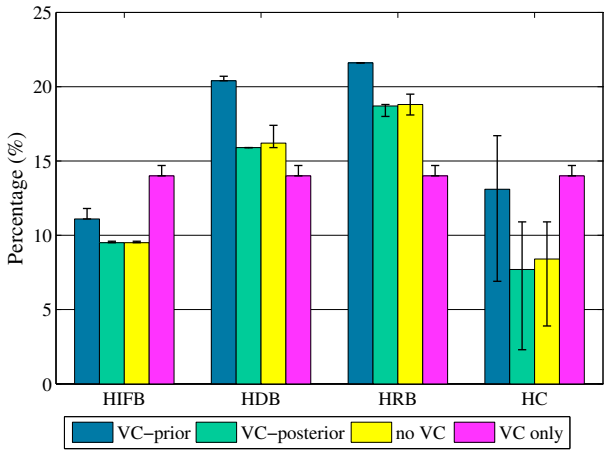

Figure 22: Performance Gain over No Clustering for Vertical Methods (LIGO)

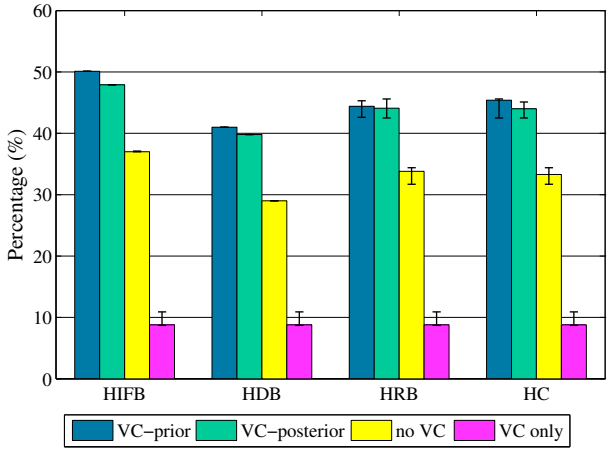

Figure 23: Performance Gain over No Clustering for Vertical Methods (Montage)

Table 6: Imbalance Metrics (CyberShake)

Level	Tasks	HRV	HIFV	HDV
1	4	0.309	0.031	1.225
2	347	0.282	0	0
3	348	0.397	0	26.2
4	1	0	0	0

Table 7: Imbalance Metrics (LIGO)

Level	Tasks	HRV	HIFV	HDV
1	191	0.024	0.01	10097
2	191	0.279	0.01	8264
3	18	0.054	0	174
4	191	0.066	0.01	5138
5	191	0.271	0.01	3306
6	18	0.04	0	43.7

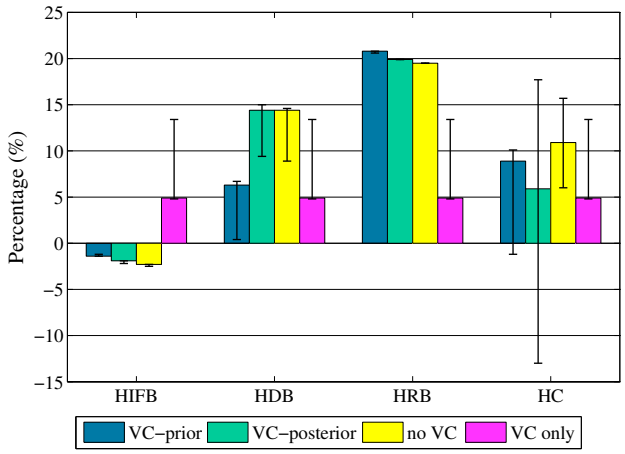

Figure 24: Performance Gain over No Clustering for Vertical Methods (Genome)

Table 8: Imbalance Metrics (LIGO after performing VC)

Level	Tasks	HRV
1,2	191	0.271
3	18	0.054
4,5	191	0.268
6	18	0.04

tures the weak connections (similarity in terms of structure). In some cases, HIFV is zero (i.e, Epigenomics) while HDV is less unlikely to be zero. Fig. 20 compares the performance gain of these algorithms by varying the workflow sizes (number of tasks in a workflow). By default, the LIGO Inspiral workflow in our paper has 800 tasks. With the decrease of workflow size, we can see that all of these methods perform worse. The reason is there is less resource contention with the decrease of workflow size and therefore task clustering does not perform well. The performance of HRB is more stable since the increase (or decrease) of workflow size has more significant impact on data transfer (roughly linear increase in runtime and square increase in data transfer).

Experiment 3: Fig. 21, 22, 23 and 24 show the performance gain of combining VC along with our horizontal methods with four workflows respectively. For the CyberShake workflow as shown in Fig. 21, we do not observe any significant change with VC since CyberShake does not have an explicit pipeline that we can perform vertical clustering (the performance gain of VC only is almost zero).

For the LIGO Inspiral workflow as shown in Fig. 22, we can see that VC-prior perform better than other combination approaches. The reason is VC-prior increases HRV and allows horizontal methods to improve the performance further while VC-posterior does not have many tasks to merge since horizontal methods change the pipeline structures. Table. 8 shows the HRV after we perform VC on the LIGO Inspiral workflow.

For the Montage workflow as shown in Fig. 23, we do see significant improvement with VC (about 15% more than the case without VC). However, the performance improvement does not distinguish these horizontal methods. The reason is tasks merges by VC (the middle and the tail levels of Montage as shown in Fig.12)are different from the one merged by horizontal methods (the first two levels in Fig. 12). Therefore, the performance of VC-posterior and VC-prior does not differ from each other.

For the Epigenomics workflow as shown in Fig. 24, we observe similar phenomenon in HRB compared to the LIGO Inspiral workflow. As to the performance of HDB, VC-prior merges tasks at the second, third, fourth and fifth level together and thus HDB has less space to further improve the overall runtime.

6. Future Work

References

[1] Laser Interferometer Gravitational Wave Observatory (LIGO), `http://www.ligo.caltech.edu`.

[2] 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.

[3] 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.

[4] 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.

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

[6] W. K. Ng, T. F. Ang, T. C. Ling, C. S. Liew, Scheduling framework for bandwidth-aware job grouping-based scheduling in grid computing, Malaysian Journal of Computer Science 19.

[7] 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.

[8] Q. Liu, Y. Liao, Grouping-based fine-grained job scheduling in grid computing, in: ETCS '09, Vol. 1, 2009, pp. 556 –559.

[9] 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: Mardi Gras'08, 2008, pp. 9:1–9:8.

[10] 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.

[11] 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, New York, NY, USA, 2012, pp. 137–148.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] G. Singh, M. 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.

[20] H. Ying, G. Mingqiang, L. Xiangang, L. Yong, A webgis load-balancing algorithm based on collaborative task clustering, Environmental Science and Information Application Technology, International Conference on 3 (2009) 736–739.

[21] A. Zomaya, G. Chan, Efficient clustering for parallel tasks execution in distributed systems, in: Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, 2004, pp. 167–.

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

[23] R. Ferreira da Silva, T. Glatard, F. Desprez, On-line, non-clairvoyant optimization of workflow activity granularity on grids, in: Euro-Par 2013 Parallel Processing, Vol. 8097 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 255–266.

[24] W. Chen, E. Deelman, Workflowsim: A toolkit for simulating scientific workflows in distributed environments, in: The 8th IEEE International Conference on eScience, 2012.

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

[26] 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, 2004.

[27] 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) (2010) 367–381.

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

[29] Workflow Generator, `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator`.

[30] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, in: Future Generation Computer Systems, 2013, p. 682692.