# 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 5.3 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

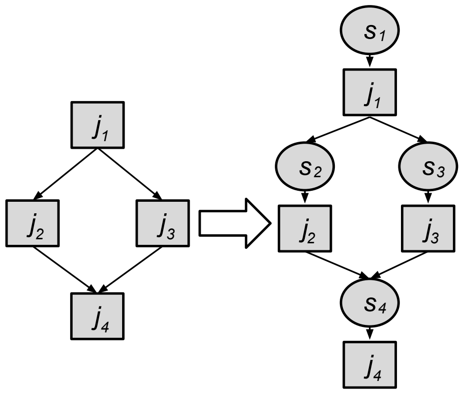

Figure 1: Extending DAG to o-DAG.

A workflow is modeled as a Directed Acyclic Graph (DAG) as shown in Fig. 1 (Left). Each node in the DAG often represents a workflow job ($j$), and the edges represent dependencies between the jobs that constrain the order in which the jobs are executed. Dependencies typically represent data-flow dependencies in the application, where the output files produced by one job are used as inputs of another job. Each job is a single execution unit and it may contains one or multiple tasks, which 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 jobs. This model fits several workflow management systems such as Pegasus [15], Askalon [16], and Taverna [17].

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:

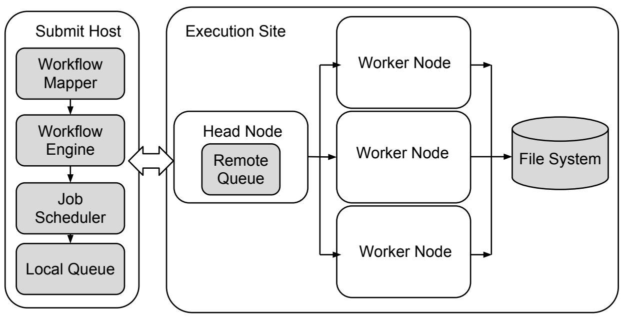

Figure 2: A workflow system model.

*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, workflow mapper is particularly used to merge small tasks together into a job such that system overheads are reduced, which is called Task Clustering. A job is a single execution unit in the workflow execution systems and it 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. 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 Wrapper.* extracts tasks from clustered jobs and executes them at the worker nodes. The clustering delay is the elapsed time on the extraction process.

use pegasus as an example but not a framework

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

The execution of a job is comprised of a series of events as shown in Fig. 3 and they are defined as:

1. Job Release is defined as the time when the workflow engine identifies that a job is ready to be submitted (when its parents have successfully completed).
2. Job Submit is defined as the time when the workflow engine submits a job to the local queue.

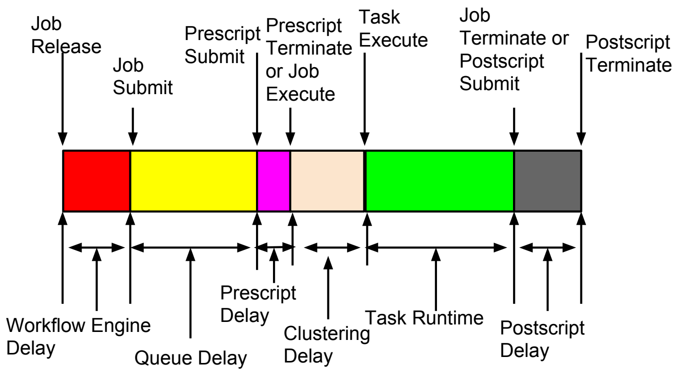

Figure 3: Workflow Events

3. Job Execute is defined as the time when the workflow engine sees a job is being executed.
4. Task Execute is defined as the time when the job wrapper sees a task is being executed.
5. Pre/Postscript Start is defined as the time when the workflow engine starts to execute a pre/postscript.
6. Pre/Postscript Terminate is defined as the time when the pre/postscript returns a status code (success or failure).

Fig. 3 shows a typical timeline of overheads and runtime in a compute job. We do not specify the data transfer delay in this timeline because data transfer is handled by data transfer jobs (stage-in and stage-out jobs).

As shown in our prior work [18], we have classified workflow overheads into four categories as follows.

1. Workflow Engine Delay measures the time between when the last parent job of a job completes and the time when the job gets submitted to the local queue. The completion time of the last parent job means this job is released to the ready queue and is waiting for resources to be assigned to it. The workflow engine delay reflects the efficiency of a workflow engine (i.e., DAGMan [19]).
2. Queue Delay is defined as the time between the submission of a job by the workflow engine to the local queue and the time the local scheduler sees the job running. This overhead reflects the efficiency of the local workflow scheduler (e.g. Condor [20]) to execute a job and the availability of resources for the execution of this job.
3. Pre/Postscript Delay is the time taken to execute a lightweight script under some execution systems before/after the execution of a job. For example, prescripts prepare working environment before the execution of a job starts and postscripts examine the status code of a job after the computational part of this job is done.
4. Clustering Delay measures the difference between the sum of the actual task runtime and the job runtime seen by the job wrapper. The cause of Clustering Delay is usually because we use a job wrapper in worker nodes to execute a clustered job that requires some delay to extract the list of tasks.

The overhead aware DAG model (o-DAG) we use in this work is an extension of the traditional DAG model. System overheads play an important role in workflow execution and constitute a major part of the overall runtime when jobs are poorly mapped to resources. 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 delay and queue delay. In summary, an o-DAG representation allows the specification of high level system overhead details, which is more suitable for the study of overhead aware scheduling.

## 4. Task Clustering

The realistic characteristics of workflows are critical to optimal workflow orchestration and profiling is an effective approach to investigate the behaviors of many complex applications. In this paper, we particularly focus on the overlapped and cumulative overheads of workflow events because this approach represents a promising trend of optimizing workflows and there is a lack of supporting analysis tools.

Workflow is typically a graph of computational activities, data transfer and system overheads as shown in our o-DAG model. Different branches of these activities may overlap with each other in the timeline and our major concern is their cumulative projection on the timeline or makespan. People have been exploring on new approaches to overlap these overheads without the necessity of reducing them. However, the challenge is how these overlapped activities contribute to the eventual makespan and how to reveal their overlapping effects in timeline from different aspects.

Fig. 4 shows the timeline of the workflow in Fig. 1. In this simple example, we are interested in questions such as: (1) whether the workflow engine delay has a good overlapping within itself? (2) wether the queue delay has a good overlapping with other types of overheads? (3) whether those 'bottleneck' overheads have overlapped with others? In the rest of this section, we will use this workflow as an example to show how we implement these metrics.

### *4.1. Metrics to Evaluate Cumulative Overheads and Runtimes*

In this section, we propose four metrics to calculate cumulative overheads of workflows, which are $Sum$, $Projection(PJ)$, $Exclusive\ Projection(EP)$ and $Impact\ Factor(IF)$. $Sum$ simply adds up the overheads of all jobs without considering their overlap. $PJ$ subtracts from $Sum$ all overlaps of the same type of overhead. It is equal to the projection of all overheads to the timeline. $EP$ subtracts the overlap of all types of overheads from $PJ$. It is equal to the projection of overheads of a particular type excluding all other types of overheads to the timeline. $IF$ uses a reverse ranking algorithm to index overheads and then calculates the cumulative overhead weighted by the ranks (impact factors). The idea is brought by web page indexing algorithms such as PageRank [21]. The calculation of $Sum$, $PJ$ and $EP$ is straightforward and Fig. 5 shows how to calculate the impact factor ($IF$) of the same workflow graph in Fig. 4. Both $PJ$ and $EP$ are proposed to answer Questions (1) and (2) and $IF$ answers Question (3), while $Sum$ serves as a comparison for them.

$$IF(j_u) = d + (1-d) \times \sum_{j_v \in Child(j_u)} \frac{IF(j_v)}{\|Parent(j_v)\|} \tag{1}$$

Equation 1 means that the $IF$ of a node (overhead or job runtime) is determined by the $IF$ of its child nodes. $d$ is the damping factor, which controls the speed of the decrease of the $IF$. $\|Parent(j_v)\|$ is the number of parents that node $j_v$ has. Intuitively speaking, a node is more important if it has more children and/or its children are more important. In terms of workflows, it means an overhead has more power to control the release of other overheads and computational activities. There are two differences compared to the original PageRank:

1. We use output link pointing to child nodes while PageRank uses input link from parent nodes, which is why we call it reverse ranking algorithm.
2. Since a workflow is a DAG, we do not need to calculate $IF$ iteratively. For simplicity, we assign the $IF$ of the root node to be 1. And then we calculate the $IF$ of a workflow ($G$) based on the equation below, while $\phi_{j_u}$ indicates the duration of node $j_u$.

$$IF(G) = \sum IF(j_u) \times \phi_{j_u} \tag{2}$$

$IF$ has two important features: (1) $IF$ of a node indicates the importance of this node in the whole workflow graph and its likelihood to be a bottleneck, the greater the more important; (2) $IF$ decreases from the top of the workflow to down, the speed of which is controlled by the damping factor $d$. It reflects the intuition that the overhead occurred at the beginning of the workflow execution is likely to be more important than that at the end. By analyzing these four types of cumulative overheads, researchers have a clearer view of whether their optimization methods have overlapped the overheads of a same type (if $PJ < Sum$) or other types (if $EP < PJ$). Besides importance, $IF$ is also able to show the connectivity within the workflow, the larger the denser. We use a simple example workflow with four jobs to show how to calculate the overlap and cumulative overheads. Fig. 4 shows the timeline of our example workflow. $j_1$ and $j_4$ are the parent and children of $j_2$ and $j_3$ respectively.

At $t$=0, $j_1$ is first released and once it is done at $t$=40, $j_3$ and $j_2$ are released. Finally, at $t$=140, $j_4$ is released. Table. 1 shows their overhead and runtime duration (assuming). For example, $PJ$ of queue delay is 40 since the queue delay of $j_2$ and $j_3$ overlaps from $t$=50 to 60. $EP$ of queue delay is 10 less than $PJ$ of queue delay since the queue delay of $j_2$ overlaps with the runtime of $j_3$ from $t$=60 to 70.

For simplicity, we do not include the data transfer delay, prescript delay and others. The overall makespan for this example workflow is 180. Table 2 shows the percentage of overheads and job runtime over makespan.

In Table 2, we can conclude that the sum of $Sum$ is larger than makespan and smaller than makespan×(number of resources) because it does not count the overlap at all. $PJ$ is larger

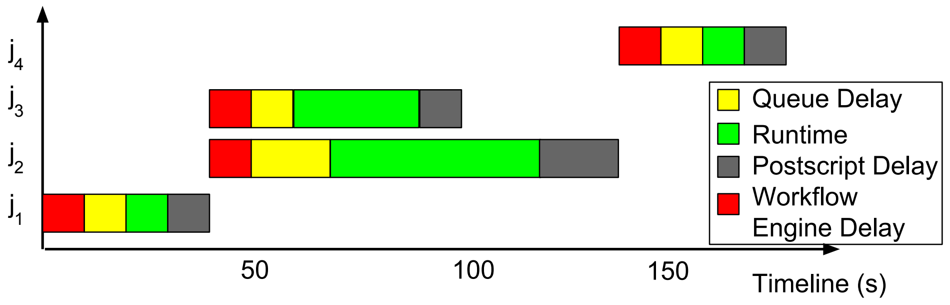

Figure 4: Workflow Timeline

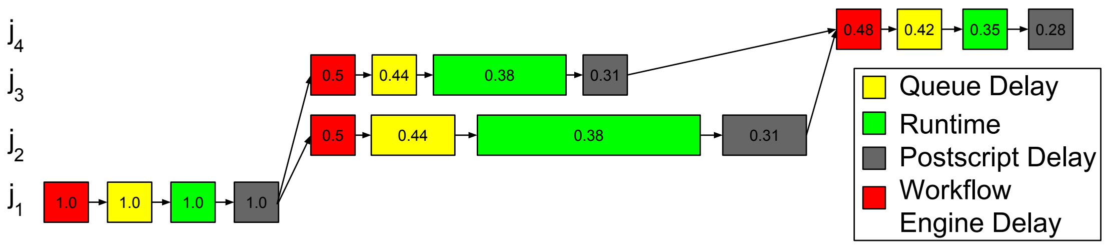

Figure 5: Impact Factor

Table 1: Overhead and Runtime Information

| Job | Release Time | Queue Delay | Workflow Engine Delay | Runtime | Postscript Delay |
|---|---|---|---|---|---|
| $j_1$ | 0 | 10 | 10 | 10 | 10 |
| $j_2$ | 40 | 10 | 20 | 50 | 20 |
| $j_3$ | 40 | 10 | 10 | 30 | 10 |
| $j_4$ | 150 | 10 | 10 | 10 | 10 |

Table 2: Percentage of Overheads and Runtime

| Contribution | Sum | PJ | EP | IF |
|---|---|---|---|---|
| runtime(%) | 57.1 | 42.8 | 28.5 | 17.7 |
| queue delay(%) | 28.5 | 21.4 | 14.2 | 13.0 |
| workflow engine delay(%) | 21.4 | 14.2 | 14.2 | 14.2 |
| postscript delay(%) | 28.5 | 28.5 | 21.4 | 11.2 |

than makespan since the overlap between more than two types of overheads may be counted twice or more. *EP* is smaller than makespan since some overlap between more than two types of overheads may not be counted. *RR* shows how intensively these overheads and computational activities are connected to each other.

### *4.2. Experiments and Evaluations*

We applied these four metrics to a widely used workflow in our experiments. These workflows were run on distributed platforms including clouds, grids and dedicated clusters. On clouds, virtual machines were provisioned and then the required services (such as file transfer services) were deployed. We examined two clouds: Amazon EC2 [22] and FutureGrid [23]. Amazon EC2 is a commercial, public cloud that is been widely used in distributed computing. We examined the overhead distributions of a widely used astronomy workflow called Montage [24] that is used to construct large image mosaics of the sky. Montage was run on FutureGrid [23]. FutureGrid is a distributed, high-performance testbed that provides scientists with a set of computing resources to develop parallel, grid, and cloud applications.

### *4.3. Relationship between Overhead Metrics and Overall Performance*

In this section, we aim to investigate the relationship between the overhead metrics that we proposed and the overall performance of popular workflow restructuring techniques. Among them, task clustering [25] is a technique that increases the computational granularity of tasks by merging small tasks together into a clustered job, reducing the impact of the queue wait time and also the makespan of the workflow. Data or job throttling [26] limits the amount of parallel data transfer to avoid overloading supporting services such as data servers. Throttling is especially useful for unbalanced workflows in which one task might be idle while waiting for data to arrive. The aim of throttling is to appropriately regulate the rate of data transfers between the workflow tasks via data transfer servers by ways of restricting the data connections, data threads or data transfer jobs. Provisioning tools often deploy pilot jobs as placeholders for the execution of application jobs. Since a placeholder can allow multiple application jobs to execute during its lifetime, some job scheduling overheads can be reduced.

*4.3.1. How Task Clustering Reduces Overheads*

| Metric | Workflow Engine Delay | Queue Delay | Runtime |
|---|---|---|---|
| SUM | $1.5 \times 10^5$ | $1.9 \times 10^7$ | $1.5 \times 10^6$ |
| SUM/n | 54(0.16%) | $6.8 \times 10^3$(20%) | 515(1.6%) |
| PJ | $3.2 \times 10^3$(9.3%) | $2.9 \times 10^4$(85%) | $3.3 \times 10^4$(102%) |
| EP | 70(0.2%) | 665(1.9%) | $4.5 \times 10^3$(14%) |
| IF | 310(0.9%) | $1.6 \times 10^4$(45%) | $3.1 \times 10^3$(9.4%) |

Table 3: Metrics Without Task Clustering

| Metric | Workflow Engine Delay | Queue Delay | Runtime |
|---|---|---|---|
| SUM | 681(74%) | $1.0 \times 10^3$(112%) | $4.4 \times 10^3$(475%) |
| SUM/n | 6.6(0.72%) | 10(1.1%) | 43(4.6%) |
| PJ | 116(13%) | 197(21%) | 660(72%) |
| EP | 72(7.8%) | 90(9.8%) | 420(46%) |
| IF | 117(13%) | 165(18%) | 442(48%) |

Table 4: Metrics with Task Clustering

In the following sections, we use a Montage workflow to show how different optimization methods improve overall performance. Many workflows are composed of thousands of fine computational granularity tasks. Task clustering is a technique that increases the computational granularity of tasks by merging small jobs together into a clustered job, reducing the impact of the queue wait time and minimizing the makespan of the workflow. Table. 4 and Table. 3 compare the overheads and runtime of the Montage workflow. $SUM/n$ indicates the average overhead or runtime per job.The percentage is calculated by the cumulative overhead ($PJ$, $EP$ or $IF$) divided by the makespan of workflows. In this example, we can see that $PJ$, $EJ$ and $IF$ consistently show that the portion of runtime is increased compared to overheads with task clustering. For example, the $IF$ of queue delay without clustering (45%) is much larger than the $IF$ of runtime (9.4%). In the case of task clustering, the $IF$ of runtime (48%) is much larger than the $IF$ of queue delay (18%). We can conclude that task clustering improves the runtime performance by reducing the influence of queue delay (more specifically, overlapping). The metric $SUM$ and $SUM/n$, which are usually used in other work, do not provide useful insight for us.

## 5. Imbalance Metrics

Horizontal Clustering (HC) is a technique that groups tasks at the same horizontal level. In our work, we define the level of a task as the longest depth from the root task to this task (Depth First Search) because the longest depth controls the final release of this task. With an o-DAG model, we can explicitly express the process of task clustering. For instance, in Fig. 6, 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 a overhead denoted the clustering delay $c$. 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, $t_1$ 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. 6 (left) is runtime1 $= s_1 + t_1 + s_2 + t_2$ , and the overall runtime for the clustered workflow in Fig. 6 (right) is runtime2 $= s_1 + c_1 + t_1 + t_2$. runtime1 ¿ runtime2 as long as $c1 < s2$, 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.

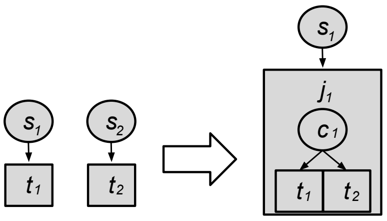

Figure 6: An Example of Horizontal Clustering.

Task clustering has been widely used to group fined-grained tasks into coarse-grained jobs [2, 3, 4, 5, 6, 7, 8, 27] . 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.

### *5.1. 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. 7 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. 7-top). In contrast, a balanced clustering strategy should try its best to evenly distribute task runtime among jobs as shown in Fig. 7 (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.

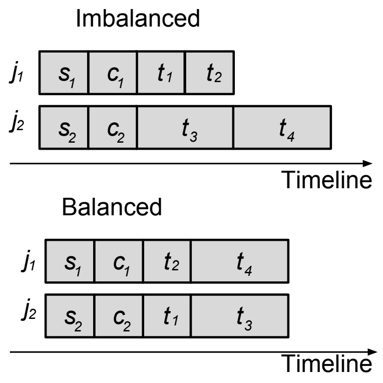

Figure 7: An Example of Runtime Variance.

**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. 8, 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. 8) 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.

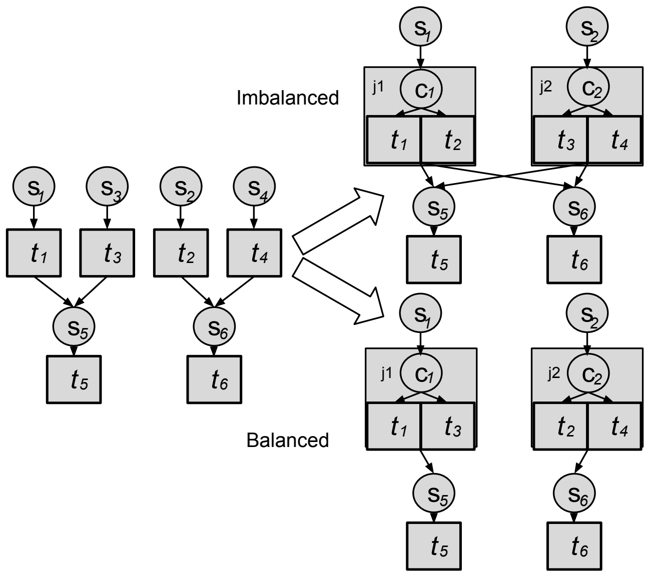

Figure 8: An Example of Dependency Variance.

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:

$$IF(t_u) = \sum_{t_v \in Child(t_u)} \frac{IF(t_v)}{\|Parent(t_v)\|} \tag{3}$$

where $Child(t_u)$ denotes the set of child tasks of $t_u$, and $\|Parent(t_v)\|$ the number of parent tasks of $t_v$. Compared to Eq. 1, the difference is that we set the damping factor to be 0 since we are doing horizontal clustering and thus there is no need to differentiate the level of a task. For simplicity, we assume the $IF$ of a workflow exit task (e.g. $t_5$ in Fig. 8) as 1.0. For instance, consider the two workflows presented in Fig. 9. $IF$ 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. 9 (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 $O(n)$.

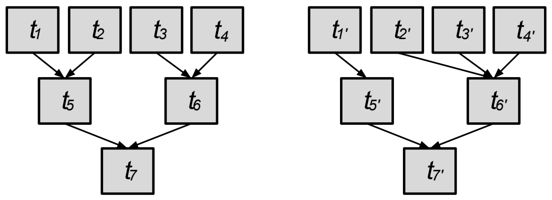

Figure 9: Example of workflows with different data dependencies.

**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 5 shows the distance matrices of tasks from the first level for both workflows of Fig. 9 ($D_1$ 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.

## *5.2. 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.

| $D_1$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ |
|---|---|---|---|---|
| $t_1$ | 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 5: Distance matrices of tasks from the first level of workflows in Fig. 9.

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*.

**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
```

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 ($t$). 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. 10 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 $j_2$).

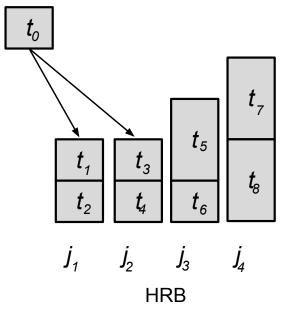

Figure 10: An example of HRB.

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. 11, 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 ($j_3$).

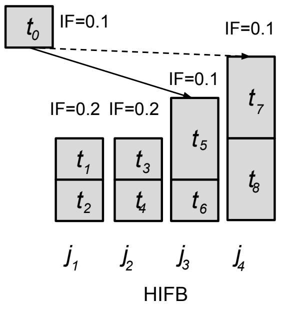

Figure 11: An example of HIFB.

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. 12, 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 ($j_1$).

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.

### *5.3. Experiment Conditions*

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.

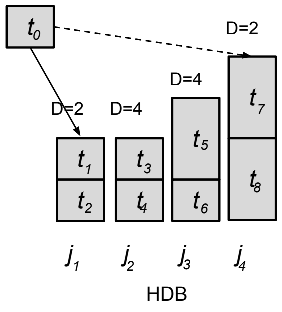

Figure 12: An example of HDB.

We extended the WorkflowSim [28] 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 [29] and FutureGrid [23]. Each machine has 512MB of memory and the capacity to process 1,000 million instructions per second. Task scheduling is data-aware, i.e. tasks are scheduled to resources which have the most input data available.

Two workflows are used in the experiments: LIGO [1] inspiral analysis, and Epigenomics [30]. Both workflows are generated and varied using the WorkflowGenerator[1]. LIGO is composed by 400 tasks and its workflow structure is presented in Fig. 13 (top); Epigenomics has about 500 tasks and is structured as showed in Fig. 13 (bottom). Runtime (average and task runtime distribution) and overhead (workflow engine delay, queue delay, and network bandwidth) information were collected from real traces production environments [18, 31], then used as input parameters for the simulations. We randomly select 20% from LIGO workflow tasks and increase their task runtime by a factor of *Ratio* to simulate the system variation in a production environment. As we have mentioned in Subsection 5.2 , the maximum number of clustered jobs is equal to the number of available resources multiplied by a *clustering factor*. Similar to [32], we use $clustering\ factor = 2$ in the experiments conducted in this work.

Two sets of experiments are conducted.

Experiment 1 evaluates the reliability and the influence of average data size in our balancing methods, since data has becoming more and more intensive in scientific workflows [31]. In this experiment set, there is no runtime variance ($HRV = 0$). The original average data size (both input and output data) of the LIGO workflow is about 5MB, and of the Epigenomics workflows is about 45MB. We increase the average data size up to 5GB.

[1]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

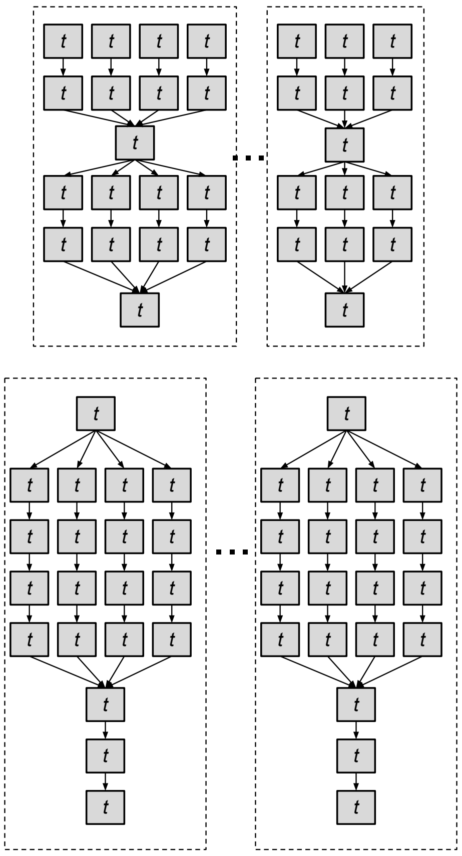

Figure 13: A simplified visualization of the LIGO Inspiral workflow (top) and Epigenomics workflow (bottom).

Experiment 2 evaluates the influence of the runtime variation ($HRV$) in our balancing methods. We assume a normal distribution to vary task runtimes based on average and standard deviation. In this experiment set, there is no variation on the data size.

Simulation results present a confidence level of 95%. We define the performance gain over HC ($\mu$) as the performance of the balancing methods related to the performance of Horizontal Clustering (HC). Thus, for values of $\mu > 0$ our balancing methods perform better than the HC method. Otherwise, the balancing methods perform poorer.

### *5.4. Results and Discussion*

Experiment 1: Fig. 14 (top) shows the performance gain over HC $\mu$ of the balancing methods compared to the HC method for the LIGO workflow. HIFB and HDB significantly increase the performance of the workflow execution. Both strategies capture the structural and runtime information, reducing data transfers between tasks, while HRB focuses on runtime distribution, which in this case is none. Fig. 14 (bottom) shows the performance of the balancing methods for the Epigenomics workflow. When increasing the average data size, only HDB demonstrates significantly improvement related to HC. Investigating the structure of the Epigenomics workflow (Fig. 13-bottom), we can see that all tasks at the same horizontal level share the same IFs ($HIFV = 0$), because each branch (surrounded by dash lines) happen to have the same amount of pipelines. Thus, HIFB has no performance improvement when compared to HC. However, for LIGO (Fig. 13-top), $HIFV \neq 0$, thus HIFB improves the workflow runtime performance. HDB captures the strong connections between tasks (data dependencies) and HIFB captures the weak connections (similarity in terms of structure). In both workflows, $HDV$ is not zero thus HDB performs better than HC.

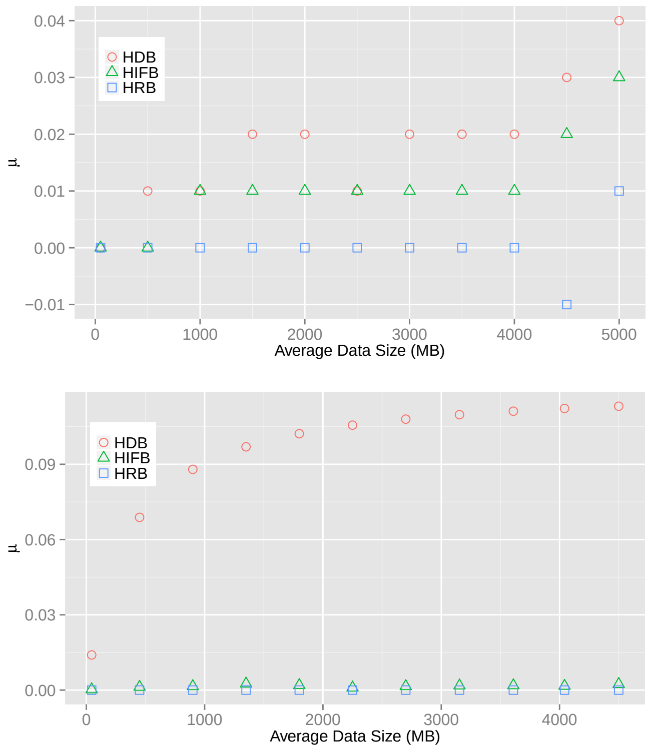

Figure 14: Experiment 1: Performance of the LIGO workflow (top) and the Epigenomics workflow (bottom).

Experiment 2: Fig. 15 shows the performance gain $\mu$ when varying task runtimes for the LIGO workflow. As expected, when $HRV$ increases HRB over performs HC. However, HDB and HIFB demonstrate poor performance because they merge tasks based on data dependencies first, and then, they balance the runtime distribution. For high values of $HRV$, we just simply need to use HRB. Otherwise, we can use either HDB or HIFB while in some cases HIFB fails to capture the structural information.

## 6. Conclusion, Discussion and Future Work

In this work, we introduced a series of quantitative and structural metrics in revealing workflow structure information and we further applied them to three usage scenario to show their influence with traced based analysis and simulations. The experiments show that these metrics can significantly improve the performance of task clustering strategies and task scheduling strategies.

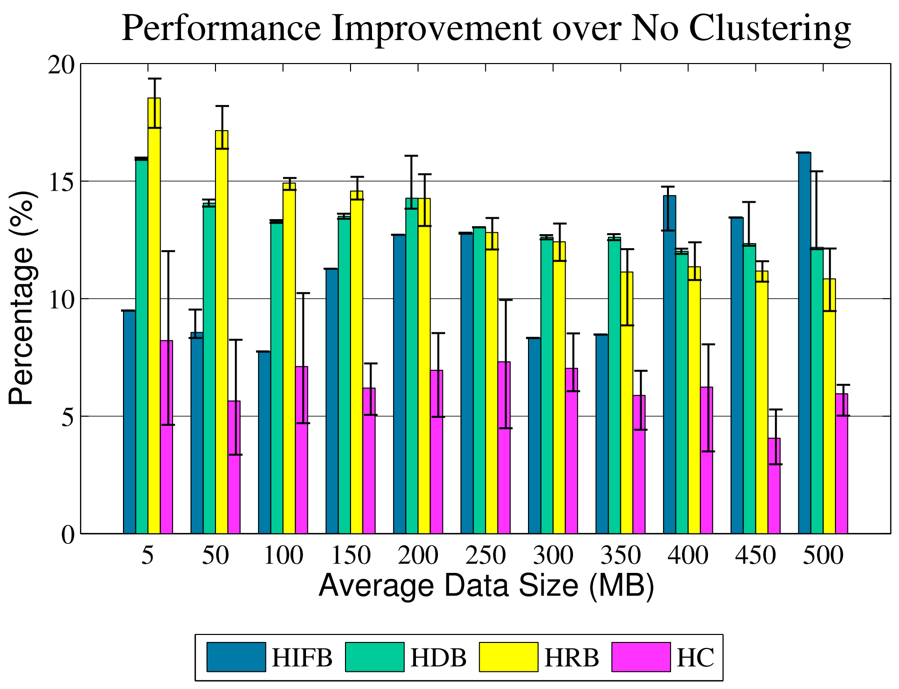

Figure 16: Runtime Improvement over No Clustering with Different Data Size

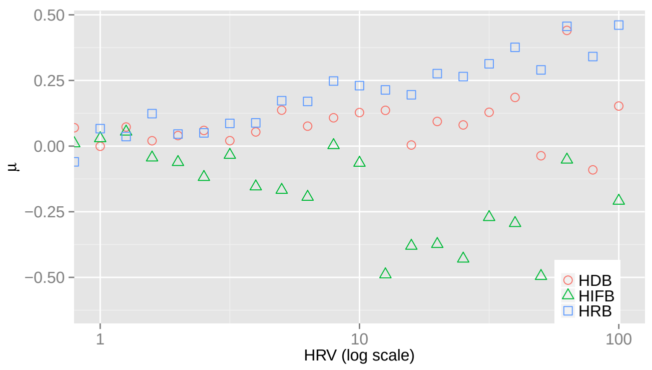

Figure 15: Experiment 2: Influence of *HRV* (LIGO workflow).

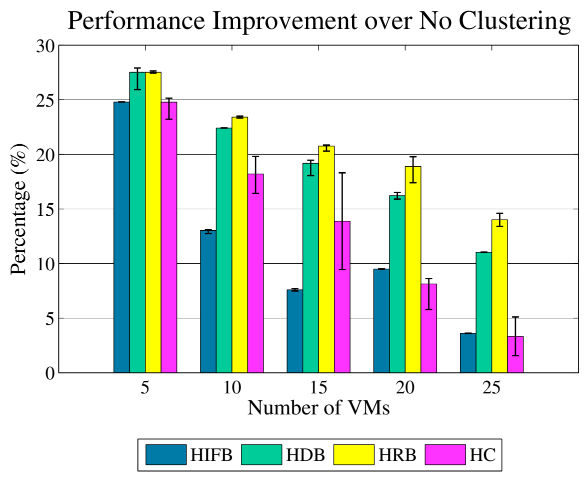

Figure 17: Runtime Provement with Different Number of Resources (Average data size is equal to 5MB).

In our work, we have shown that these metrics have strong connection with the performance of different optimization methods. However, not all of them perform well under all of the circumstances. Also, the origin of these metrics are heavily depending on human intuition. Still, there is a lack of understanding of whether we can have more and better metrics in revealing how workflow structures interact with different optimization methods. As an initial effort, we will continue to use pattern discovery techniques to collect as many metrics as possible and performance experiments on them.

With enough metrics, one important step we are going to take is to aggregate multiple metrics and use machine learning techniques to reveal their correlation. However, combining the metrics we have proposed above is challenging. First, these metrics are quantitative and can be measured, however, on their own they are hard to compare against each other and determine whether one metric is more significant than others. Second, having multiple metrics together increase the complexity and this is against our initial goal, which is reduce the complexity in analyzing workflows. A more sophisticated alternative, however, is to conduct a statistical analysis based on Principal Component Analysis (PCA). This method determines in a more precise way the existing correlation between the collected metrics and their impact on workflow performance.

## References

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

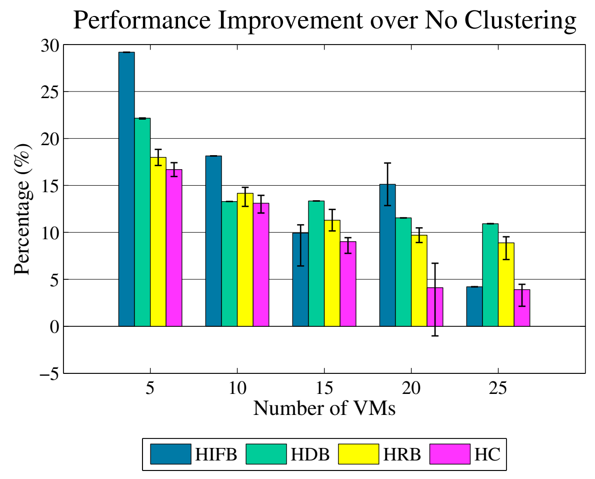

Figure 18: Runtime Provement with Different Number of Resources (Average data size is equal to 500MB)).

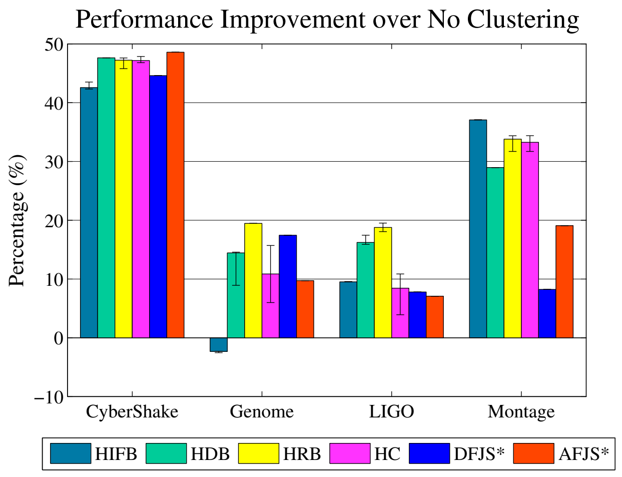

Figure 19: Runtime Provement of all the Algorithms

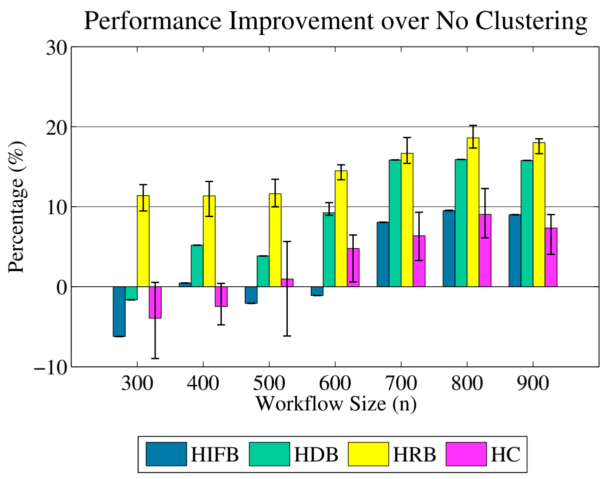

Figure 20: Runtime Provement with different Workflow Sizes

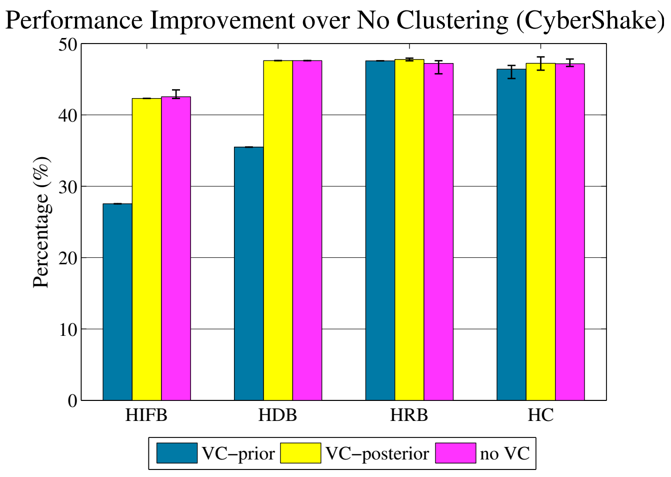

Figure 21: Runtime Provement over No Clustering for Vertical Methods

[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. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta,

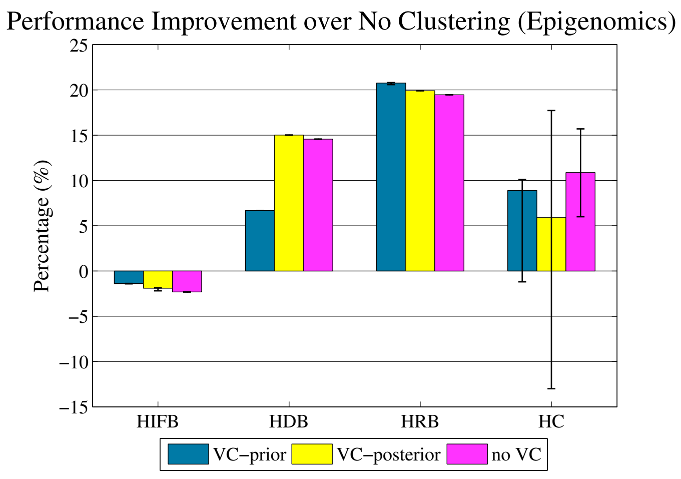

Figure 22: Runtime Provement over No Clustering for Vertical Methods

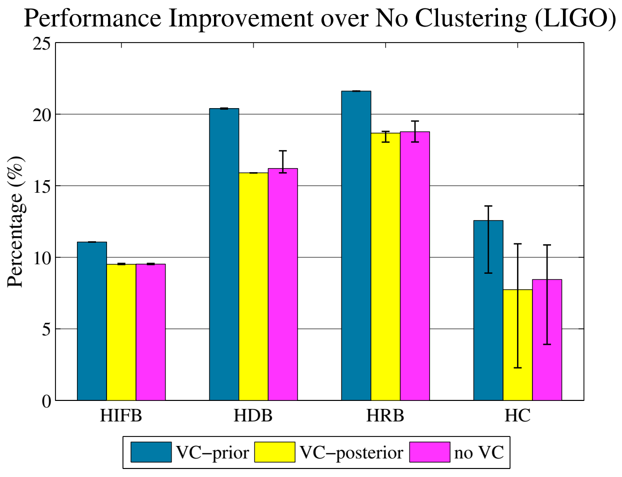

Figure 23: Runtime Improvement over No Clustering for Vertical Methods

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, Scientific Programming 13 (3) (2005) 219–237.

[16] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, H. Truong, ASKALON: a tool set for cluster and grid computing, Concurrency and Computation: Practice & Experience 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] DAGMan: Directed Acyclic Graph Manager, `http://cs.wisc.edu/condor/dagman`.

[20] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-G: a computation management agent for Multi-Institutional grids, Cluster Computing 5 (3) (2002) 237–246.

[21] L. Page, S. Brin, R. M. andTerry Winograd, The pagerank citation ranking: bringing order to the web, 1999.

[22] Amazon.com, Inc., Elastic Compute Cloud (EC2), `http://aws.amazon.com/ec2`.

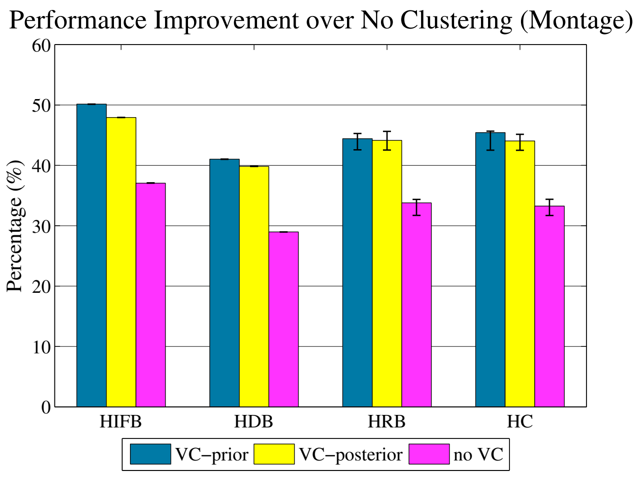

Figure 24: Runtime Provement over No Clustering for Vertical Methods

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

[24] 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.

[25] 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.

[26] S.-M. Park, M. Humphrey, Data throttling for data-intensive workflows, in: IEEE Intl. Symposium on Parallel and Distributed Processing, 2008.

[27] 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.

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

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

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

[31] 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.

[32] W. Chen, R. F. da Silva, E. Deelman, R. Sakellariou, Balanced task clustering in scientific workflows, in: The 9th IEEE International Conference on e-Science, IEEE, 2013.