Balanced Task Clustering in Scientific Workflows

Weiwei Chen*, Rafael Ferreira da Silva‡, Ewa Deelman*, Rizos Sakellariou§
*University of Southern California, Information Sciences Institute, Marina Del Rey, CA, USA
{wchen,deelman}@isi.edu
‡University of Lyon, CNRS, INSERM, CREATIS, Villeurbanne, France
rafael.silva@creatis.insa-lyon.fr
§University of Manchester, School of Computer Science, Manchester, U.K.
rizos@cs.man.ac.uk

***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 workflow, data locality, load balance, task clustering

I. Introduction

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

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

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

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 performance of task clustering. The first one is a top-down approach [14] that represents the clustering problem as a global optimization problem and aims to minimize the overall workflow execution time. However, the complexity of solving such an optimization problem does not scale well since most methods use genetic algorithms. The second one is a bottom-up approach [3], [9] that only examines free tasks to be merged

and optimizes the clustering results locally. In contrast, our work extends these approaches to consider the neighboring tasks including siblings, parents, and children because such a family of tasks has strong connections between them.

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 III presents our workflow and execution environment models, Section IV details our heuristics and algorithms, Section V reports experiments and results, and the paper closes with a discussion and conclusions.

II. Related Work

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

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

A plethora of balanced scheduling algorithms have been developed in the networking and operating system domains. Many of these schedulers have been extended to the hierarchical setting. Lifflander et al. [13] proposed to use work stealing and a hierarchical persistence-based rebalancing algorithm to address the imbalance problem in scheduling. Zheng et al. [15] 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 [16] (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.

III. 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 [17], Askalon [18], and Taverna [19].

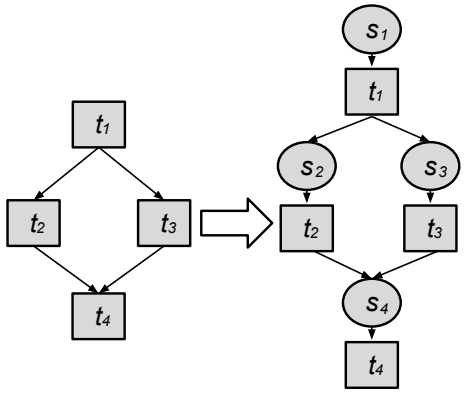

Fig. 1: Extending DAG to o-DAG.

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:

a) 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.

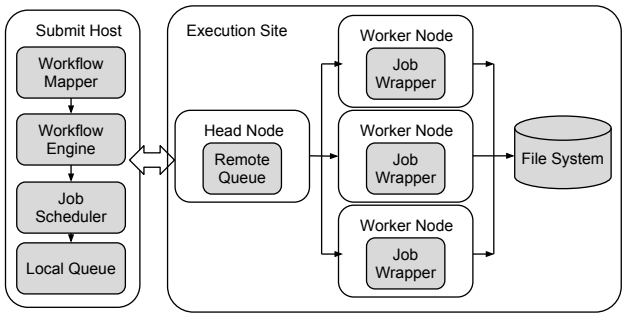

Fig. 2: A workflow system model.

b) 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.

c) 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.

d) 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.

In this work, 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. Fig. 1 shows how we augment a DAG to be an o-DAG with the capability to represent scheduling overheads (s) such as workflow engine and queue delays. This classification of scheduling overheads is based on our prior study on workflow analysis [2].

With an o-DAG model, we can explicitly express the process of task clustering. For instance, in Fig. 3, 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 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.

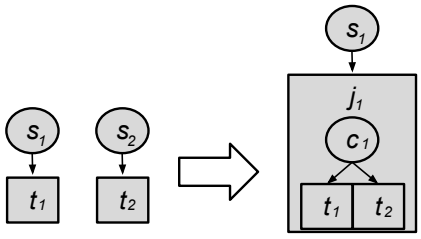

Fig. 3: An Example of Task Clustering.

Fig 3 shows a typical example of a Horizontal Clustering (HC) technique that groups tasks at the same workflow level. In our work, we define the level of a task as the longest depth from the root task to this task because the longest depth controls the final release of this task.

In summary, an o-DAG representation allows the specification of high level system overhead details, which is more suitable than DAG models when clustering tasks.

IV. Balanced Clustering

In this section we introduce the imbalance problem we are going to address and we present metrics to quantitatively capture workflow characteristics, and methods to handle runtime and dependency imbalances.

A. Balanced Clustering 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. 4 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. 4-top). In contrast, a balanced clustering strategy should try its best to evenly distribute task runtime among jobs as shown in Fig. 4 (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.

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

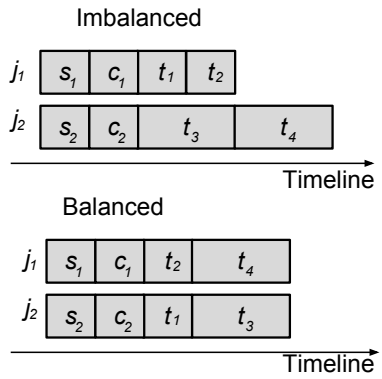

Fig. 4: An Example of Runtime Variance.

thus loss of parallelism. For example, in Fig. 5, 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. 5) 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.

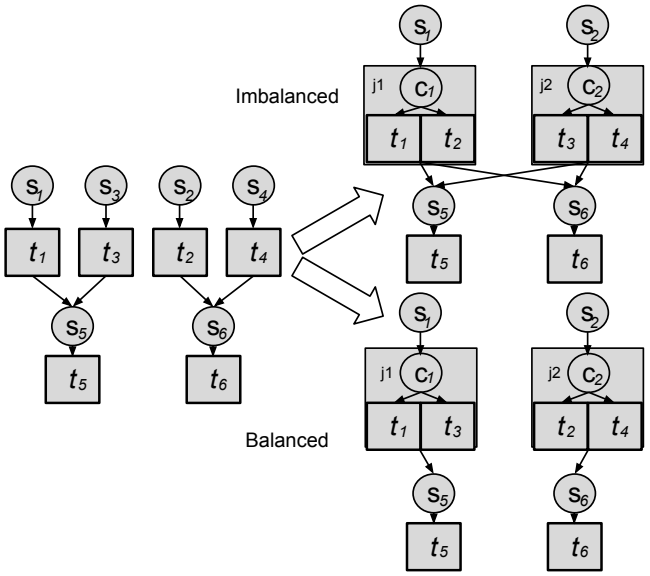

Fig. 5: An Example of Dependency Variance.

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:

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

where $Child(t_u)$ denotes the set of child tasks of t_u, and $L(t_v)$ the number of parent tasks of t_v. For simplicity, we assume the IF of a workflow exit task (e.g. t_5 in Fig. 5) as 1.0. For instance, consider the two workflows presented in Fig. 6. 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. 6 (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 $O(n)$.

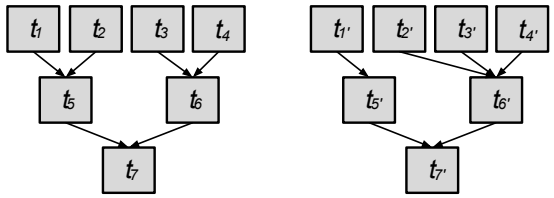

Fig. 6: Example of workflows with different data dependencies.

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 I shows the distance matrices of tasks from the first level for both workflows of Fig. 6 (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 I: Distance matrices of tasks from the first level of workflows in Fig. 6.

B. 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 compare the performance of using different *clustering factor* in Section 5.

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: $TL \leftarrow$ GETTASKSATLEVEL($w, level$) ▷ Partition W based on depth
5: $CL \leftarrow$ MERGE(TL, C) ▷ Form a list of clustered jobs
6: $W \leftarrow W - TL + CL$ ▷ 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: $J \leftarrow$ GETCANDIDATEJOB(CL, t) ▷ Get a candidate task
13: $J \leftarrow J + t$ ▷ 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. 7 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).

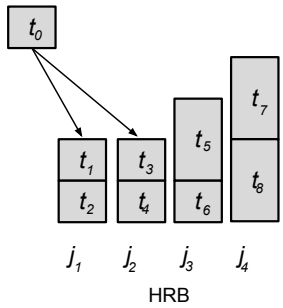

Fig. 7: An example of HRB.

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. 8, 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).

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. 9, 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. Table II summarizes the imbalance metrics and

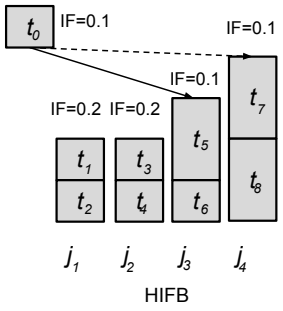

Fig. 8: An example of HIFB.

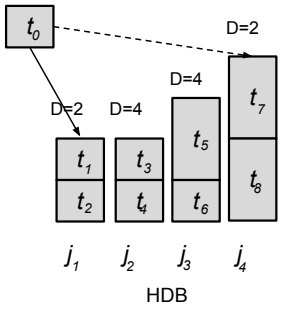

Fig. 9: An example of HDB.

balancing methods presented in this work.

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

TABLE II: Imbalance metrics and balancing methods.

V. Experiment and Evaluation

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) [10], which is widely used by workflow management systems such as Pegasus.

A. Experiment Conditions

We extended the WorkflowSim [20] 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 [21] and FutureGrid [22]. 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 [23]. Both workflows are generated and varied using the WorkflowGenerator[1]. LIGO is composed by 400 tasks and its workflow structure is presented in Fig. 10 (top); Epigenomics has about 500 tasks and is structured as showed in Fig. 10 (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 [2], [24], then used as input parameters for the simulations.

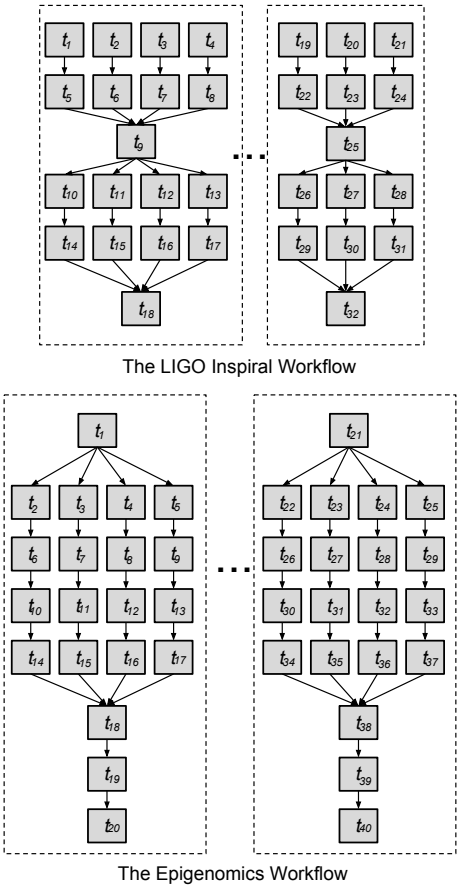

Fig. 10: A simplified visualization of the LIGO Inspiral workflow (top) and Epigenomics workflow (bottom).

Three sets of experiments are conducted. Experiment 1 aims at determining an appropriate *clustering factor* such that both the workflow runtime performance and the reliability over the dynamic system variation are improved. We randomly select 20% from LIGO workflow tasks and increase their task runtime by a factor of *Ratio* to simulate the system variation

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

in a production environment.

Experiment 2 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 [24]. 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.

Experiment 3 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 (μ) 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.

B. Results and Discussion

As we have mentioned in Subsection IV-B , the maximum number of clustered jobs is equal to the number of available resources multiplied by a *clustering factor*. Experiment 1: Fig. 11 shows the speedup of Horizontal Clustering (HC) for different *Ratio* and *clustering factors*. We use the speedup of HC in overall runtime compared to the original overall runtime without clustering. The speedup decreases with the increase of the *clustering factor*. However, a smaller *clustering factor* performs worse when the *Ratio* is high. For simplicity, we use $clustering\ factor = 2$ in the experiments conducted in this work.

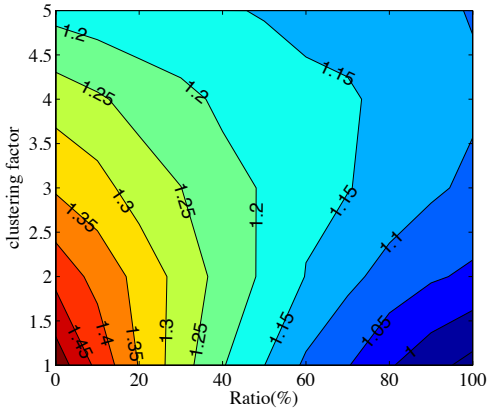

Fig. 11: Experiment 1: Speedup of Horizontal Clustering (HC).

Experiment 2: Fig. 12 (top) shows the performance gain over HC μ 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. 12 (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. 10-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. 10-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.

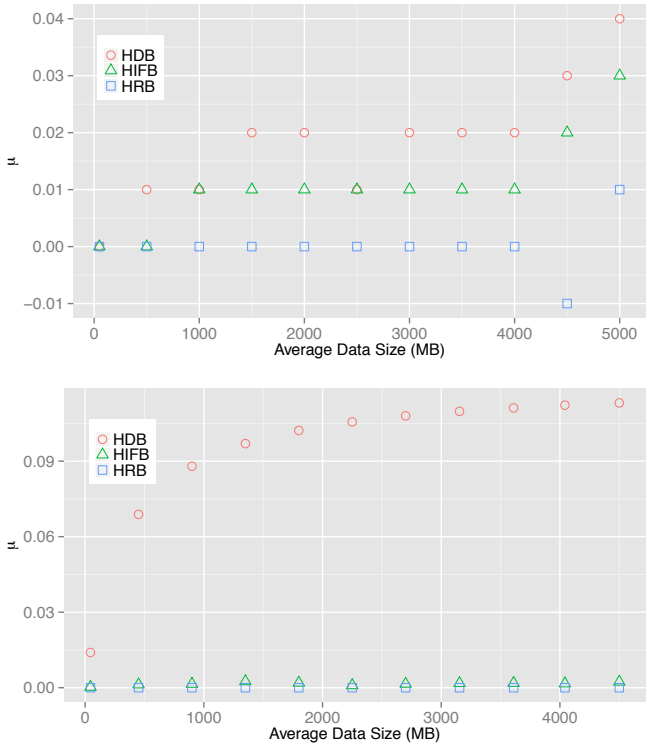

Fig. 12: Experiment 2: Performance of the LIGO workflow (top) and the Epigenomics workflow (bottom).

Experiment 3: Fig. 13 shows the performance gain μ 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.

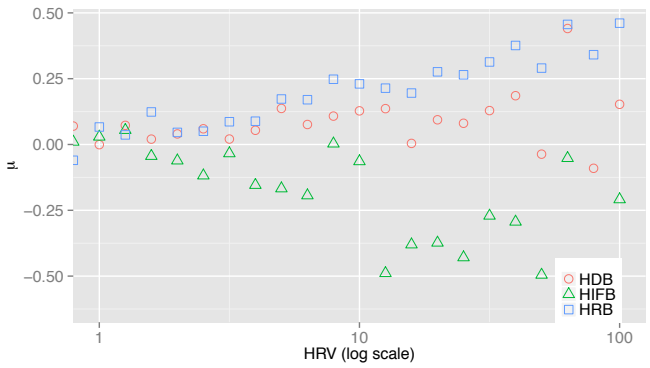

Fig. 13: Experiment 3: Influence of HRV (LIGO workflow).

VI. Conclusion

We presented three balancing methods to address the load balance problem when clustering workflow tasks. We defined three imbalance metrics to quantitatively measure workflow characteristics based on task runtime variation (HRV), task impact factor (HIFV), and task distance variance (HDV).

The balanced clustering methods were implemented in the WorkflowSim simulator. Three experiments were conducted using two real workflows. The first experiment showed the gain of task clustering by using a naive horizontal clustering technique. Results showed that our balancing methods can significantly reduce the runtime and data dependency imbalance. For high HRV values, a runtime variance based approach (HRB) performs best over a naive horizontal clustering algorithm. When data dependency is more important HIFB and particularly HDB methods perform better than the naive approach, while HRB performs similarly. In our future work, we will explore the influence of different overheads, and how the HDB method can be extended to perform vertical clustering, i.e. multiple levels clustering.

References

[1] B. P. Abbott et al., "Ligo: the laser interferometer gravitational-wave observatory," *Reports on Progress in Physics*, vol. 72, no. 7, p. 076901, 2009.

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

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

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

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

[6] N. Muthuvelu, C. Vecchiolab, I. Chaia, E. Chikkannana, and R. Buyyab, "Task granularity policies for deploying bag-of-task applications on global grids," *FGCS*, vol. 29, no. 1, pp. 170 – 181, 2012.

[7] W. K. Ng, T. F. Ang, T. C. Ling, and C. S. Liew, "Scheduling framework for bandwidth-aware job grouping-based scheduling in grid computing," *Malaysian Journal of Computer Science*, vol. 19, 2006.

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

[9] Q. Liu and Y. Liao, "Grouping-based fine-grained job scheduling in grid computing," in *ETCS '09*, vol. 1, 2009, pp. 556 –559.

[10] G. Singh, M.-H. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D. S. Katz, and G. Mehta, "Workflow task clustering for best effort systems with pegasus," in *Mardi Gras'08*, 2008, pp. 9:1–9:8.

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

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

[13] J. Lifflander, S. Krishnamoorthy, and 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*, ser. HPDC '12, New York, NY, USA, 2012, pp. 137–148.

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

[15] G. Zheng, A. Bhatelé, E. Meneses, and L. V. Kalé, "Periodic hierarchical load balancing for large supercomputers," *Int. J. High Perform. Comput. Appl.*, vol. 25, no. 4, pp. 371–385, Nov. 2011.

[16] 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, and 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*, vol. 61, no. 6, pp. 810–837, 2001.

[17] 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, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Sci. Program.*, vol. 13, no. 3, pp. 219–237, Jul. 2005.

[18] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, Jr., and H.-L. Truong, "Askalon: a tool set for cluster and grid computing: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 17, no. 2-4, pp. 143–169, Feb. 2005.

[19] 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, and C. Wroe, "Taverna: lessons in creating a workflow environment for the life sciences: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1067–1100, Aug. 2006.

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

[21] Amazon.com, Inc., "Amazon Web Services." [Online]. Available: http://aws.amazon.com

[22] "FutureGrid." [Online]. Available: http://futuregrid.org/

[23] "USC Epigenome Center." [Online]. Available: http://epigenome.usc.edu

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