Tool Support for Distributed Workflow Management with Task Clustering

Ayesh Weerasinghe[1]*, Kalana Wijethunga[2]*, Randika Jayasekara[#3]*, Indika Perera[4]*, Anuradha Wickramarachchi[5†]

***Abstract*— When in need for executing complex sets of interrelated calculations on High-Performance Computing (HPC) environments the obvious choice is to use scientific workflows. As workload management software do not support the execution of interrelated tasks, workflow management systems have been introduced to execute workflows on HPC environments. Recently, a new distributed architectural model that offers dynamic workflow execution capabilities to workflow management systems is introduced. It executes workflows on a per-task basis. While this approach facilitates dynamic workflows, it adds a considerable overhead to workflows substantially increasing their makespans. As most workflows are static, task-wise execution of workflows degrades the performance of most workflows. In this paper, we introduce a distributed workflow management system, SwarmForm that introduces task clustering to the new architectural model. SwarmForm is open source and offers better performance than existing distributed workflow management systems by clustering workflow tasks to reduce overheads while allowing the users to choose between task-wise and cluster-wise execution of workflows depending on the workflow nature. The paper proves that SwarmForm enables the use of all the features introduced with the new architectural model while providing better makespans for scientific workflows.**

Keywords*— *Task Clustering, Workflow Management Systems, Scientific Workflows.

I. Introduction

Almost every scientific domain such as Astrophysics, Bio and health informatics, Physics, and Bio-Sciences use workflows to express complex sets of tasks that are dependent on one another using Scientific workflows. These workflows are executed in High-Performance Computing (HPC) environments as they need a lot of computing power to execute. Workload management software like PBS Pro [1], SLURM [2], TORQUE [3] are installed on these HPC environments to manage the computing resources of the environment. However, they do not support workflow scheduling but only support the execution of independent jobs. Given the complexity of real-world workflows, the execution becomes cumbersome as the users have to manage a large number of individual job execution files. Therefore, Workflow Management Systems (WMS) have been introduced to execute scientific workflows on HPC environments.

A workflow management system is able to get a workflow consisting of a series of interrelated tasks as input and submit them as separate jobs to a workload management software while maintaining the dependencies between the tasks in order to be executed in an HPC environment. WMSs executes workflows either by executing each task as a job and passing its results to other tasks (Chained Jobs) or by executing the whole workflow as a single job (Pilot Job). Running a workflow as a pilot job results in better makespan with poor resource utilization of the execution environment whereas running a workflow as chained jobs results in better resource utilization with poor makespan.

Distributed WMSs execute workflows as chained jobs with a separate job for each task whereas centralized WMSs execute workflows as pilot jobs. Therefore, centralized WMSs have better makespan with poor resource utilization while distributed WMSs have better resource utilization with poor makespan. Although centralized WMSs minimize this issue by clustering the tasks in the workflow and submitting them as few chained jobs, they fail to provide many features available in distributed WMSs like dynamic workflows, concurrent execution of multiple workflows, failure detection and correction, etc. Therefore, it is observed that distributed WMSs offer much more important functions than centralized WMSs. Scheduling a job on an HPC environment consists of a considerable overhead [4]. Thus, scheduling of jobs using a distributed WMS causes a significant increase in the makespan of the workflow as they execute each task as a separate job. The advantages offered by distributed WMSs can be retained while reducing the makespan of workflows by introducing task clustering to distributed WMSs. This is clearly demonstrated in Fig. 1(a) where the jobs are scheduled in the chained fashion resulting in a longer makespan and Fig. 1(b) where the jobs are executed within a much shorter makespan with somewhat of a compromise on the resource utilization. However, there is a significant potential to improve the resource utilization while

Correspondence: R. Jayasekara[#3] (E-mail: rpjayasekara.16@cse.mrt.ac.lk)
Received: 20-12-2020 Revised: 14-02-2021 Accepted: 10-03-2021

This paper is an extended version of the paper "SwarmForm: A Distributed Workflow Management System with Task Clustering" presented at the ICTer 2020 conference.

A. Weerasinghe[1*,] K. Wijethunga[2*,] R. Jayasekara[#3*,] I. Perera[4*]are from the Department of Computer Science & Engineering, University of Moratuwa, Sri Lanka. {ayeshweerasinghe.16, kalana.16, rpjayasekara.16, indika}@cse.mrt.ac.lk

Anuradha Wickramarachchi[5†] is from the Australian National University, Australia. (anuradha.wickramarachchi@anu.edu.au)

DOI: http://doi.org/10.4038/icter.v14i2.7223

having pilot jobs in a chained fashion to make a near optimal balance of trades.

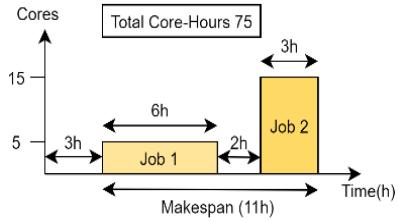

Fig. 1(a) Running a workflow as a set of Chained jobs

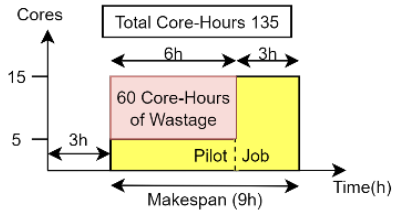

Fig. 1(b) Running a workflow as a Pilot job

The paper presents the following contributions to the domain of workflow scheduling:

- The research introduces SwarmForm [5] a new open source distributed workflow management system with task clustering capabilities.
- The research introduces an extension to the existing Workflow and Platform Aware Clustering algorithm [6] to improve its performance
- The research implements the Resource Aware Clustering (RAC) algorithm [7] in SwarmForm to maximize the resource utilization of the clustered workflows that are executed through SwarmForm

The rest of the paper is arranged as follows. Section II presents a review of the existing literature and background of the study. Section III presents the work proposed in the study. Section IV evaluates the performance improvement introduced by the proposed work and Section V concludes the paper with an overview on the future work in Section VI.

II. RELATED WORK

Liu et al. [8] show that the functional architecture of a WMS consists of 5 layers - Workflow Execution Plan (WEP) generation, WEP execution, Presentation, User services, and Infrastructure. According to how these layers are managed, existing WMSs can be categorized as centralized WMSs and distributed WMSs. In WMSs like Pegasus [9], Taverna [10], etc. all these functional layers are managed by a single program and all the computing nodes in the HPC environment are managed by a single or a few instances of the WMS which makes them centralized WMSs. WMSs like eHive [11], FireWorks [12] consist of a set of programs that manages different functional layers and they have independent instances of the WMS per each computing node of the HPC environment which makes them distributed WMSs. Each instance of the distributed WMSs can be associated with a different database which eliminates the need to have a single central queue for all the jobs that are expecting to be run in the HPC environment.

Centralized WMSs such as Pegasus [9], Taverna [10], Kepler [13] have been used for over a decade for executing workflows in many high-performance computing environments all around the world. They include features such as workflow submission, special CLI tools for workflow design and management, ability to store provenance data, etc. that are key requirements when executing a scientific workflow. Taverna and Kepler include a versatile workbench that allows fully graphical workflow design, which is extremely helpful in designing new scientific workflows. These systems are much more effective in executing scientific workflows than using workload management software for scientific workflow execution. The inability to execute dynamic workflows can be seen as the major drawback of the centralized WMSs. All these systems need the workflows to be defined at the beginning of the workflow and they do not allow modifying the workflow while it is being executed. In addition to that, most of them run workflows as a single pilot job. As explained by Rodrigo et al. [14] executing a workflow as a single pilot job causes a huge resource wastage as many of the resources of the HPC environment are idle most of the time. Furthermore, they do not support concurrent execution of workflows as they are submitted as pilot jobs.

The above issues have been addressed in eHive [11] and FireWorks [12] using a new architectural model. They follow a blackboard-based architecture with 3 main components: a central database that holds details of each workflow submitted by the users, a set of clients that pull tasks from the database and execute them on the backend and a client manager that handles spawning, killing, and managing of the clients. All three components work as independent programs and that provides a distributed architectural model to these systems. The distributed architecture resolves the single point of failure in the existing centralized systems by having different programs control different layers in workflow management. In addition to that, these systems support concurrent execution of multiple workflows and the system manages the scheduling of tasks across workflows. Distributed WMSs submit workflows as chained jobs with each task packed as a single job. This allows the workflows to change its structure at the runtime while resulting in a substantial increase in resource utilization of the execution environment as only the required resources are obtained per each job. It also makes sure that the failure of one job does not affect the execution of other jobs.

Three major overheads are present when executing a job on an HPC environment: i.e., Scheduling overhead (Time taken to schedule the job on a specific node), Queue Delay (Time a job must wait in the queue until it gets the opportunity to be executed) and Communication Overhead (Time taken to transfer the results of parent job to its children). Therefore,

executing each task of a workflow as separately chained jobs will cause a substantial increase in the makespan [14] as executing each job adds a considerable overhead to the total runtime of the workflow.

While distributed WMSs offer a lot of features that are not available in centralized WMSs, the increased makespan of workflows due to the execution of each task as a chained job raises a major concern. Even though this adds support for dynamic workflows, executing both static and dynamic workflows as individually chained jobs cause an unnecessary overhead. A better approach to this problem would be to introduce task clustering to distributed WMSs and allow the user to decide whether he needs clustering or not depending upon the application. This will reduce the makespan of workflows in distributed WMSs while ensuring that all the advantages offered by distributed WMSs are preserved.

To address this issue, we introduce a new distributed workflow management system SwarmForm which includes task clustering to reduce the makespan of workflows. Using SwarmForm, we intend to deliver all the advantages of a distributed WMS to users while maintaining the optimum balance between the makespan of workflows and the resource utilization of the environments.

Task clustering is already implemented in some of the centralized WMSs like Pegasus [9] and in some Grid middleware management systems like Xavantes [15]; we intend to use those techniques to provide better makespans for workflows executed using distributed WMSs.

Different researches have introduced different clustering techniques. In the related literature, Horizontal Runtime Balancing, Horizontal Impact Factor Balancing and Horizontal Distance Balancing algorithms introduced by Chen et al. [16] are being used as the baseline for workflow task clustering. Kaur et al. [17] has introduced a new clustering technique called Hybrid Balanced Task Clustering Algorithm that clusters tasks both vertically and horizontally. Chen et al. [16] has introduced a Balanced clustering technique for horizontal clustering and Sahni et al. [6] has introduced the Workflow and Platform Aware task clustering (WPA) Algorithm. Zhang et al. [18] has introduced a new metric called Dependency Correlation to cluster tasks in their Dependency Balance Clustering Algorithm. Dependency Balancing Clustering Algorithm cluster tasks based on the similarity of their dependencies. WPA Algorithm uses the knowledge about the structure of the workflow and the execution environment to cluster tasks such that there is the least possible ineffective parallelism as possible. Hybrid Balanced Task Clustering Algorithm combines all three baseline algorithms to present a novel approach to cluster tasks both vertically and horizontally. A novel approach to cluster tasks considering both execution time of tasks and resource requirements has been introduced by RAC algorithm [7]. This algorithm tries to cluster the tasks that are most similar in the resource requirements while trying to ensure that all the created clusters have near similar runtimes. It makes sure that the resource wastage is minimized, and all the tasks in clustered jobs are released nearly at the same time when tasks in a workflow are clustered together to reduce the makespan. None of the existing task clustering algorithms except the RAC algorithm take resource requirements of tasks into account when clustering. Considering the pros and cons of each algorithm, we implemented an extended version of the WPA algorithm and RAC algorithm to carry out task clustering in SwarmForm while giving the privilege to the user to choose which algorithm he wants to use.

III. Methodology

A distributed WMS called SwarmForm has been developed, which offers task clustering, to address the drawbacks in existing WMSs explained above. Task clustering will play a key role in reducing the makespan of a workflow in the new WMS.

A. WPA Algorithm

A workflow is represented as a Directed Acyclic Graph (DAG) with nodes of the graph representing tasks and edges between the nodes representing the dependencies between the tasks. The WPA algorithm only clusters the tasks at the same level of the workflow DAG where the level of a task is defined as the longest distance from root node task(s) to task node. While this improves the makespan of a workflow, it causes a dependency imbalance in the workflow. It also does not reduce the communication overhead caused when transferring the output of the parent task to the child tasks as children and parents are not clustered together. To address these issues, we introduce a new technique to cluster the workflows both horizontally and vertically as follows.

Algorithm 1 Extended Workflow-and-platform aware clustering algorithm

```
1: procedure WPA(Workflow w)
2:     Begin
3:     w ← ClusterSingleParentSingleChild(w)
4:     for level = depth(w) to 2 step − 1 do
5:         cLatlevel ← {}
6:         taskList ← getTasksAtLevel(w, level)
7:         taskList ← SortTasksByIncreasingOrderOfTheLongestParent(taskList)
8:         for each task t in taskList do
9:             cls ← AssignParentsToClusters(t)
10:            Add cls to cLatlevel
11:        end for
12:        taskListParents ← getParentNodesOfTaskList(taskList)
13:        w ← w − taskListParents + cLatlevel
14:    end for
15: end procedure
```

Under the proposed technique, first, the tasks with single-child single-parent relationships are clustered together and then the resulting tasks are clustered horizontally using the WPA algorithm. The WPA algorithm takes the available number of computing nodes as an input. Since in most of the HPC environments we cannot get the exact number of resources available at the time of execution, we have proposed a slight modification to the WPA algorithm along with the addition of our vertical clustering approach. The modified pseudocode of the WPA algorithm is given in algorithm 1.

Fig. 2 illustrates the significance of this task clustering approach. Fig. 2(a) depicts an example workflow with 5 levels and the number on each node states the execution time of the task. First, the tasks are being clustered vertically considering their single-parent single-child relationships (Fig. 2(b)). Fig. 2(c) shows the result of the proposed vertical clustering technique on the example workflow of Fig. 2(a). Then the resulting workflow tasks are clustered horizontally using the WPA algorithm (Fig. 2(c)). Fig. 2(d) shows the result of our proposed extended WPA clustering algorithm.

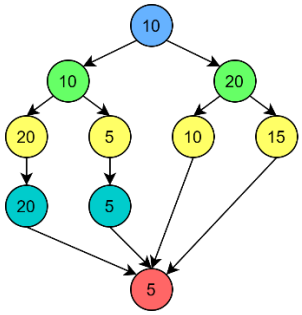

Fig. 2(a) Initial workflow

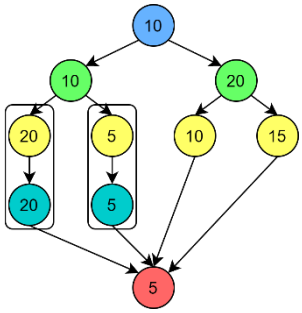

Fig. 2(b) Cluster nodes with single-parent single-child relationships

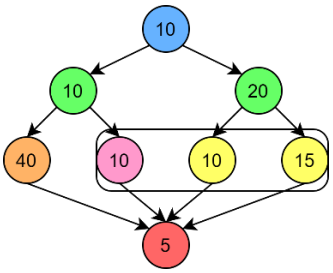

Fig. 2(c) Horizontally cluster the resultant workflow

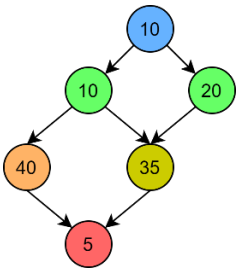

Fig. 2(d) Clustered workflow

Algorithm 2 Cluster Single-Parent Single-Child

```
1:  procedure ClusterSingleParentSingleChild(Workflow w)
2:      Begin
3:      for level = 1 to depth(w) do
4:          taskList ← getTasksAtLevel(w, level)
5:          for each task t in taskList do
6:              cluster ← {}
7:              while t has single child and t.child has single parent do
8:                  Add t to cluster
9:                  t ← t.child
10:             end while
11:         end for
12:         clusteredTasks ← getClusteredTaskList()
13:         w ← w − clusteredTasks + cluster
14:     end for
15:     Return w
16: end procedure
```

The algorithm 2 explains the pseudocode of the proposed vertical clustering technique. The algorithm takes a workflow as the input. It begins with the first level of the workflow and iterates to the depth of the workflow (Line 3). It selects the tasks at each level (Line 4) and iterates the tasks, one by one (Line 5). If the task only has a single child and that child task has no other parents (single-parent single-child relationship), both the task and its child task are grouped into a cluster. This process is repeated in a depth-first manner until there are no more single-parent single-child relationships for the selected task (Line 7-10). Finally, the workflow is updated if a selected task is clustered with its children (Line 13).

B. SwarmForm Workflow Management System

1) SwarmForm Architecture: SwarmForm distributed WMS is developed on top of FireWorks [12] distributed WMS which is the state-of-the-art system in the domain of distributed WMSs. FireWorks is used as an open source library in the implementation of SwarmForm. SwarmForm ensures that all the functionalities of FireWorks are available to the user while offering additional functionalities for workflow management. SwarmForm is highly decoupled from FireWorks and this approach provides the ability to develop FireWorks and SwarmForm independently ensuring fast and easy adaptations to any update to FireWorks.

The architecture of SwarmForm bears a close resemblance to FireWorks with some additional improvements. In SwarmForm, a workflow is referred to as a SwarmFlow. A SwarmFlow can be represented as a Directly Acyclic Graph (DAG) and these SwarmFlows can be defined by the Python interface, command-line interface or by directly loading a JSON or YAML SwarmFlow definition. SwarmForm adapts the workflow definition format introduced by FireWorks for defining SwarmFlows as this format helps to define workflows in a more easy and readable way in contrast to the existing DAX format.

A SwarmFlow consists of one or more individual tasks that are called Fireworks (FWs). These FWs represent the nodes in the SwarmFlow definition DAG whereas the edges of the DAG represent dependencies between FWs. A Firework can have a sequence of one or more atomic tasks that are called FireTasks. These FireTasks are separate Python functions that can call shell scripts, transfer files, read/write files or call other Python functions. FireTasks can return FWActions that can modify the SwarmFlow dynamically at runtime based on the computational conditions which give the dynamic behaviour to the system. SwarmPad is another key part of the SwarmForm WMS that is used to store all the details of SwarmFlows, FWs, provenance data and other data related to execution of SwarmFlows. SwarmPad is a NoSQL database which is built using MongoDB. FireWorkers are the clients who pull FWs from the SwarmPad and execute. It launches unique agents called Rockets to pull and execute each FW. Workflow management is handled by the SwarmPad and workflow execution is handled by Rockets and FireWorkers which provides the distributed behaviour to the SwarmForm WMS.

Fig. 3 shows the architecture of the SwarmForm WMS. The FlowParser takes the input workflow and passes it to the SwarmFormer. SwarmFormer clusters the SwarmFlow and adds it to the database. Optionally, the FlowParser can save the SwarmFlows directly to the database without clustering, based on the user requirement. The SwarmFormer takes a SwarmFlow as the input and clusters the tasks in the SwarmFlow and saves the clustered SwarmFlow in the database. Later, the FireWorkers can pull tasks using Rockets and execute clustered Fireworks in HPC environments as shown in Fig. 3.

2) SwarmForm Features: As we have described above, the overhead in executing a job is a critical factor which results in increasing the makespan of a workflow. Even the state-of-the-art distributed workflow management system does not address this issue as it executes each task in a workflow as a separately chained job. As a solution to the aforementioned problem, we introduce task clustering to SwarmForm which reduces the makespan of the workflows by minimizing the overheads in the execution of a workflow. In section IV, we have proven that SwarmForm outperforms the state-of-the-art distributed WMS FireWorks [12] when task clustering is enabled.

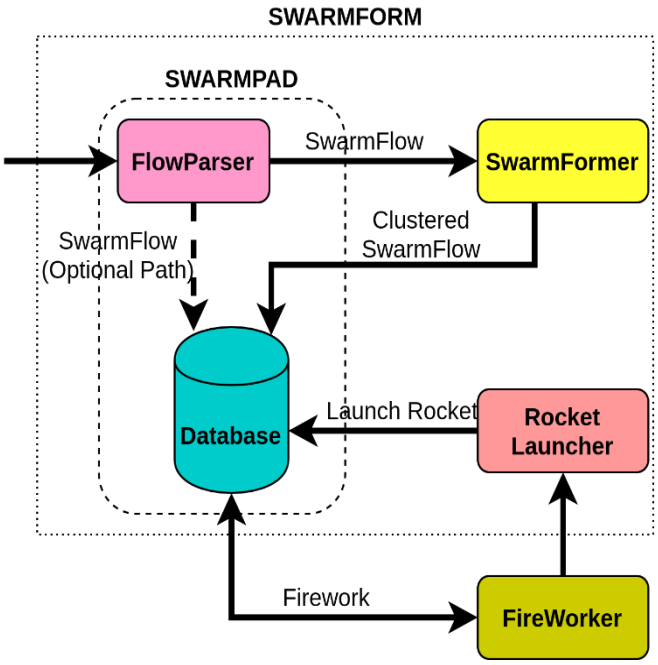

Fig. 3 SwarmForm Architecture

In SwarmForm, workflows which are referred to as SwarmFlows are treated as primary entities and Fireworks are considered as secondary entities. This considerably eases the process of managing workflows when executing workflow operations like task clustering. In addition to that, SwarmFlow can accept and process multiple task parameters like cost, execution time, resource requirements of the task etc. These parameters can be used for making better scheduling decisions and workflow management decisions like how the tasks will be clustered which increases the performance of the system. The support to these parameters is added in such a way that a user can easily extend the parameter set by easily adding new parameters. The WMS takes cost parameters like execution time, required number of cores per task as inputs through *_queueadapter* identifier in the workflow definition. Therefore, users will be able to define new parameters like memory required, wall time etc. which can be used in further workflow management decisions.

Initially, SwarmForm was only equipped with the WPA clustering algorithm, which did not consider the resource requirements when making task clustering decisions. Later, we implemented the RAC algorithm [7] which takes both execution time and resource requirements into consideration when making task clustering decisions. We integrated the RAC algorithm [7] to the system in such a way that SwarmForm WMS could use any task clustering algorithm based on the user requirement, without limiting to a single task clustering algorithm. With this modification, a developer can easily implement new task clustering algorithms and use them without modifying the core components of the WMS.

With the integration of these task clustering algorithms, we introduce a new feature to express the estimated resource wastage due to task clustering. Because of this feature, users can see the resource wastage that could be occurred due to clustering of the workflow before executing workflows in HPC environments. This can be used to make decisions for selecting suitable task clustering algorithms without executing workflows in resource intensive environments. Resource wastage of a single cluster containing l tasks (W_j) and total resource wastage of the workflow containing k clusters (W_t) can be calculated as in (1) and (2) respectively.

$$W_j = \left(\sum_{i=0}^{l}\left(\max\left(\bigcup_{i=0}^{l} r_{t_i}\right) * m_{t_i}\right) - \sum_{i=0}^{l}\left(r_{t_i} * m_{t_i}\right)\right) \quad (1)$$

$$W_t = \sum_{j=0}^{k}\left(\sum_{i=0}^{l}\left(\max\left(\bigcup_{i=0}^{l} r_{t_i}\right) * m_{t_i}\right) - \sum_{i=0}^{l}\left(r_{t_i} * m_{t_i}\right)\right) \quad (2)$$

where:

r_{t_i} = Cores(resources) required by the i^{th} task

m_{t_i} = Execution time of the i^{th} task

As the workflow definition format used in FireWorks and SwarmForm is a novel format, the users have to put an extra effort to convert their existing workflows to the new format. To ease out this process we introduce a Workflow Generator which can be easily used to generate workflows by inputting the minimum parameters possible. In addition to that, it supports converting DAX files directly to the new workflow definition format with no user intervention at all.

C. RAC Algorithm

RAC algorithm [7] is chosen for this due to its ability to minimize resource wastage in the execution environment. It uses a novel metric called Resource Aware Clustering coefficient to identify the most suitable tasks that should be assigned to the same cluster. Although the RAC algorithm does not always outperform the existing task clustering algorithms in makespan reduction of workflows, it outperforms all the existing task clustering algorithms in maximizing resource utilization while providing competitive makespan reductions in workflows. Therefore, RAC algorithm [7] is implemented in SwarmForm to maximize the resource utilization of the execution environment while minimizing the makespan of the workflow.

Since the scientific workflow is represented as a directed acyclic graph (DAG), we have defined our data structure to model the workflow in SwarmForm which we referred to as DAG model. That DAG model is used to implement the WPA algorithm. The same approach is followed when implementing

the RAC algorithm. The algorithm takes the workflow represented using the DAG object and number of clusters per horizontal level(R) as the inputs and returns a DAG object which represents the workflow with clustered tasks. The algorithm traverses the DAG following a level-by-level approach, starting from level one. It takes the tasks at each level and clusters the tasks at level only if the number of tasks at level is greater than the number of clusters per level. In each level, first it creates R number of empty clusters and iterates the tasks in the level task by task. In each iteration in the inner loop the resource aware clustering factor is calculated for the task respective to the clusters created for that level. Then it selects the cluster with the minimum factor value since resource-aware clustering factor gives the smallest value with the cluster that the considering task fits best and checks whether the cluster has not exceeded the number of tasks that it can hold. If it does not exceed, the task is put into that cluster. This process is repeated for each task in each level. After populating the clusters by tasks for each layer workflow DAG is updated as it needs to preserve the dependencies. First, it removes the task in the considering level from the workflow DAG and adds the new clusters to the workflow. Then updates the parent-child relationships appropriately as the updated workflow DAG needs to preserve the dependencies between tasks.

IV. Results

In this section, we evaluate the performance of SwarmForm WMS. As FireWorks is the state-of-the-art in distributed workflow management systems, WPA task clustering enabled SwarmForm WMS is compared and evaluated against the FireWorks WMS. To have the same evaluation setup for both systems, we have evaluated both WMSs on standard benchmark workflows CyberShake (Fig. 4), LIGO (Fig. 5) and SIPHT (Fig. 6) presented by Bharathi et al. [19].

The workflow definitions of CyberShake 100 job workflow, LIGO 100 job workflow, and SIPHT 97 job workflow provided by Pegasus workflow generator are used for the evaluation [20]. We use a workflow simulation setup for evaluating the performance of the systems. This is a widely used approach since reserving an HPC environment for evaluation purposes is highly costly. The simulation setup consists of 5 rockets with each rocket acting as a computing node with a single core. Each rocket pulls a job from the database and executes it. We have added a constant delay after completion of each job to represent the communication overhead incurred when transferring the output of a parent job to its children jobs. Only the considered workflow is present in the database throughout the evaluation.

Initially, two sets of the same workflows in DAX format are taken and converted into SwarmForm/Firework readable format using the SwarmForm workflow generator. Then, a set of workflows are clustered and executed using the SwarmForm WMS and the other set of workflows are directly executed using the FireWorks WMS. Makespan of each workflow is measured in both systems and the Performance Gain (3) is calculated.

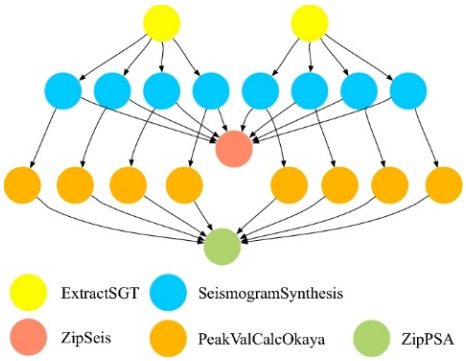

Fig. 4 CyberShake workflow structure

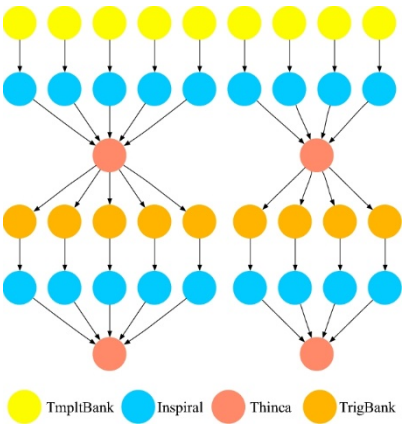

Fig. 5 LIGO workflow structure

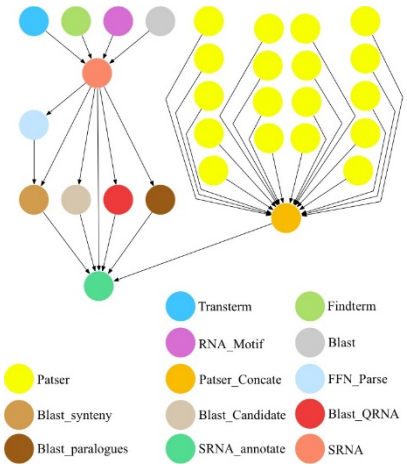

Fig. 6 SIPHT workflow structure

$$Performance\ Gain = \frac{\left(\begin{matrix}Makespan\ of\ executing\ the\\ workflow\ in\ FireWorks\end{matrix}\right) - \left(\begin{matrix}Makespan\ of\ executing\ the\\ workflow\ in\ SwarmForm\end{matrix}\right)}{\left(\begin{matrix}Makespan\ of\ executing\ the\\ workflow\ in\ FireWorks\end{matrix}\right)} \quad (3)$$

From Fig. 7, it can be observed that the makespan of each workflow has been reduced when executed using SwarmForm than with FireWorks. This proves that executing workflows with task clustering enabled in SwarmForm reduces the makespan of each workflow considerably than executing it in FireWorks.

The Performance Gain shows the percentage improvement in the makespan of each workflow executed in SwarmForm compared to FireWorks. From the results of the experiments

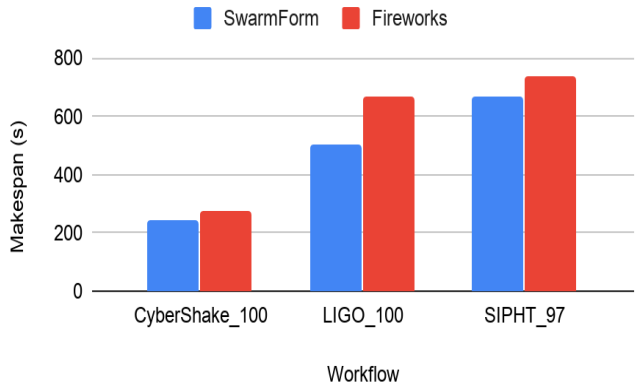

Fig. 7 Comparison of the makespan of each workflow executed using FireWorks and SwarmFlow with task clustering enabled.

(Fig. 8), it can be observed that SwarmForm shows a 10.19% improvement in the makespan of CyberShake, 24.36% improvement in the makespan of LIGO and 9.41% improvement in the makespan of SIPHT workflows. Further, it should be noted that the performance gain of each workflow is positive which shows that SwarmForm outperforms FireWorks when task clustering is enabled.

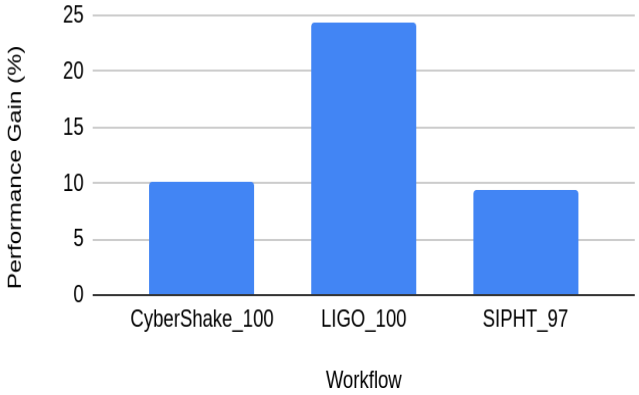

Fig. 8 Comparison of the average performance gain in executing each workflow in SwarmForm and Fireworks

In this evaluation, we have considered only the communication delay between tasks and the queue delay among jobs in the same workflow as the overhead. Clustering related tasks together eliminate the communication overhead between those tasks as they are executed in the same node under the same job. The improvement shown in the evaluation mainly results from the reduction of communication overhead between the tasks. However, in real environments, there are many more overheads like scheduling overhead and queue delays due to the competition for limited resources by a large number of jobs from multiple workflows. Among them, queue delay can increase the makespan by a substantial amount as the delay increases considerably with the increase of the job submissions. These overheads are reduced when tasks are clustered. Therefore, we expect that SwarmForm will perform even better when used with real workflows in HPC environments.

V. Discussions

This paper presents SwarmForm, a new distributed workflow management system with task clustering capabilities. SwarmForm is built using FireWorks which is an open source library and offers useful features such as support for dynamic workflows, concurrent workflow execution, and failure detection and correction that are not available in the existing centralized WMSs. SwarmForm introduces task clustering to increase the performance of existing distributed WMSs, a DAX workflow importer and a workflow generator that can be used for workflow simulation purposes.

As another contribution, the research has introduced an extension to the WPA algorithm which improves its performance. The extension of the WPA algorithm is to introduce a hybrid clustering approach, which clusters the tasks both vertically and horizontally. We implement the updated clustering algorithm in SwarmForm and evaluate SwarmForm with FireWorks and prove that execution of workflows in SwarmForm yields better makespans than executing them in the existing state-of-the-art distributed WMS due to the introduction of task clustering.

Finally, we implement the RAC algorithm as the primary clustering algorithm in SwarmForm to introduce resource management capabilities to SwarmForm. None of the existing WMSs consider minimizing resource wastage when clustering tasks. Therefore, executing workflows using SwarmForm by clustering their tasks with RAC algorithm significantly reduces the resource wastage of the execution environment while providing a considerable improvement in the makespan of the workflow. Further, the users are given the opportunity to choose any of the task clustering algorithms depending on the requirement for clustering their workflows while providing the developers with the ability to implement any required task clustering algorithm and use them without having to change any core components of SwarmForm. The estimated resource wastage after clustering of workflows with each clustering algorithm is also shown to the users which allows them to choose the algorithm that gives them the best resource utilization and the makespan.

VI. Future Work

Task clustering is done to achieve different objectives along with reducing makespan like minimizing resource wastage, minimizing dependency imbalance, achieving QoS requirements etc. Currently, SwarmForm contains only two task clustering algorithms which are capable of solving resource imbalance and runtime imbalance problems. We plan to implement a few more task clustering algorithms in SwarmForm thus allowing the user to choose the suitable

algorithm depending on the use case from a variety of task clustering algorithms.

We plan to improve our workflow generator to generate actual workflows and to import actual workflows defined in DAX format into SwarmForm workflow definition format. It will later be extended to support Common Workflow Language [21] as well. Further, we plan to introduce a GUI to SwarmForm to easily define new workflows graphically as the existing distributed WMSs consist of Graphical User Interfaces (GUI) only for reporting.

REFERENCES

[1] J. Nabrzyski, J. M. Schopf and J. and Węglarz, "PBS Pro: Grid Computing and Scheduling Attributes," in *Grid resource management*, Boston, Kluwer Academic Publishers, 2004, pp. 183-190.

[2] A. B. Yoo, M. A. Jette and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job scheduling strategies for parallel processing*, Berlin, Springer, 2003, pp. 44-60.

[3] D. Klusáček, V. Chlumský and H. Rudová, "Planning and Optimization in TORQUE Resource Manager", *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 203-206, 2015.

[4] W. Chen and E. Deelman, "Workflow overhead analysis and optimizations", *Proceedings of the 6th workshop on Workflows in support of large-scale science - WORKS '11*, pp. 11-20, 2011. Available: 10.1145/2110497.2110500.

[5] "SwarmForm/SwarmForm", *GitHub*, 2020. [Online]. Available: https://github.com/SwarmForm/SwarmForm.

[6] J. Sahni and D. P. Vidyarthi, "Workflow-and-Platform Aware task clustering for scientific workflow execution in Cloud environment," *Futur. Gener. Comput. Syst.*, vol. 64, pp. 61–74, 2016, doi: 10.1016/j.future.2016.05.008

[7] A. Weerasinghe, K. Wijethunga, R. Jayasekara, I. Perera and A. Wickramarachchi, "Resource Aware Task Clustering for Scientific Workflow Execution in High Performance Computing Environments", in *22nd IEEE International Conference on High Performance Computing and Communication*, Fiji, 2020.

[8] J. Liu, E. Pacitti, P. Valduriez and M. Mattoso, "A Survey of Data-Intensive Scientific Workflow Management", *Journal of Grid Computing*, vol. 13, no. 4, pp. 457-493, 2015. Available: 10.1007/s10723-015-9329-8.

[9] E. Deelman et al., "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems", *Scientific Programming*, vol. 13, no. 3, pp. 219-237, 2005. Available: 10.1155/2005/128026.

[10] D. Turi, P. Missier, C. Goble, D. D. Roure and T. Oinn, "Taverna Workflows: Syntax and Semantics," in *Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007)*, Bangalore, 2007, pp. 441-448.

[11] J. Severin et al., "eHive: An Artificial Intelligence workflow system for genomic analysis", *BMC Bioinformatics*, vol. 11, no. 1, p. 240, 2010. Available: 10.1186/1471-2105-11-240.

[12] A. Jain et al., "FireWorks: a dynamic workflow system designed for high-throughput applications", *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 5037-5059, 2015. Available: 10.1002/cpe.3505.

[13] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows," *Proceedings. 16th International Conference on Scientific and Statistical Database Management*, 2004., Santorini Island, Greece, 2004, pp. 423-424.

[14] E. Elmroth, P. Östberg, L. Ramakrishnan and G. P. Rodrigo, "Enabling Workflow-Aware Scheduling on HPC Systems", in *HPDC '17: The 26th International Symposium on High-Performance Parallel and Distributed Computing*, Washington DC USA, 2017, pp. 3 - 14.

[15] L. Bittencourt and E. Madeira, "A dynamic approach for scheduling dependent tasks on the Xavantes grid middleware", in *Middleware06: 7th International Middleware Conference*, Melbourne, Australia, 2006.

[16] W. Chen, R. F. Da Silva, E. Deelman, and R. Sakellariou, "Balanced task clustering in scientific workflows," *Proc. - IEEE 9th Int. Conf. e-Science, e-Science 2013*, pp. 188–195, 2013, doi: 10.1109/eScience.2013.40.

[17] A. Kaur, P. Gupta, and M. Singh, "Hybrid balanced task clustering algorithm for scientific workflows in cloud computing," *Scalable Comput.*, vol. 20, no. 2, pp. 237–258, 2019.

[18] L. Zhang, D. Yu, and H. Zheng, "Optimization of cloud workflow scheduling based on balanced clustering," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10581 LNCS, pp. 352–366, 2017.

[19] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. Su and K. Vahi, "Characterization of scientific workflows," 2008 Third Workshop on Workflows in Support of Large-Scale Science, Austin, TX, 2008, pp. 1-10.

[20] "WorkflowGenerator - Pegasus - Pegasus Workflow Management System", *Confluence.pegasus.isi.edu*, 2014. [Online]. Available: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowHub.

[21] B. Chapman et al., *Common Workflow Language, v1.0*. United States: figshare, 2016.