ISSN 2047-3338

Dynamic Clustering for Scientific Workflows with Load Balancing in Resource

Roya Bagheri[1] and Abolfazel Toroghi Haghighat[2]

[1,2]Department of Computer Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran

[1]royabagheri_eng@yahoo.com, [2]at_haghighat@yahoo.com

***Abstract*— Task clustering is a method that merges multiple short tasks into a single job so that the job runtime is enhanced and the total system overhead is reduced also is efficient way to decrease a communication delay in DAGs by classification heavily communicating tasks to the similar labeled clusters and then assigning tasks in a cluster to the similar resource. A workflow usage is usually containing of various tasks with the requirement for different resource types to complement which we recall it heterogeneity in workflow. The important idea in this research is based upon the idea of defining the set of tasks that could run concurrently and distribute them into various sub-workflows and then assigned each sub-workflow in resource cluster instead of dedicating individual tasks. This can decrease inter-task communication cost and attain high parallelism the received DAG partition into several sub-workflows that is discovered by Dynamic Clustering Workflows (DCW) algorithm and thus progress workflow execution accomplishment.**

***Index Terms*— Scientific Workflows, Task Clustering, Communication Delay and Load Balancing**

I. INTRODUCTION

MANY calculation scientists improve and use large-scale, loosely-coupled usages that are often organized as scientific workflows. Although the most of the tasks within these usage are often relatively brief running (from a little seconds to a little minutes), in congregate they represent a significant value of calculation and data [1], [2]. When carrying out these usages in a multi-machine, distributed environment, such as the Cloud or the Grid, important system overheads may exist and may decelerate the application execution [3]. To decrease the impact of suchlike overheads, task clustering methods [4]–[12] have been improved. The clustering is another impressive way to decrease a communication delay in DAGs by regimentation heavily communicating tasks to the similar labeled clusters. Clustering algorithms have two phases: the task grouping phase that divides the original task graph into clusters for designation instead of designating individual tasks. This can decrease inter-task communication cost and thus develop workflow execution efficacy and post-clustering phases which can regenerate the clusters produced in the prior phase and achieve the final task-to-resource map.

Data dependencies between workflow tasks perform an important role when grouping tasks within a level. The concept of data dependency is a data transfer between two tasks (output data for one and input data for the other). Grouping tasks without investigating these dependencies may cause to data locality problems, where the output data constructed by parent tasks are few distributed. Therefore, the times of data transfer and failure probabilities increase. As result, we contend that the data dependencies of subsequent tasks should be investigated.

Practically, there are two approaches to develop the efficacy of task clustering. The first one is a top-down method [15] that demonstrates the clustering problem as a global optimization matter and aims to minimize the overall workflow performance time. However, the complication of solving such an optimization matter does not scale well since solutions are based on genetic algorithms. The second approach is a bottom-up method [4], [10] that only investigates free tasks to be merged and optimizes the grouping results locally.

The low efficacy of fine-grained tasks is a general matter in widely distributed programs where the scheduling overhead and alignment times at resources are excellent, such as Cloud and Grid systems.

The rest of the article is structured as follows: Section II presents literatures review. Fuzzy logic is explained in Section III. The offered routing protocol is given in section IV. The simulation outcomes have been demonstrated in Section V. The final section consists of conclusions and future works.

II. REALATED WORKS

Several researches have addressed the control of task granularity of tasks. For example, Muthuvelu et al. [4] proposed a grouping algorithm that classifies bag of tasks based on runtime, and later based on CPU time, task file size and resource constraints [5] .Recently, they planed an online scheduling algorithm [6,7] that classifies tasks based on resource network utilization, application deadline and user's budget. Ng et al. [8] and Ang et al. [9] displayed bandwidth in the scheduling framework to increase the efficacy of task scheduling. Longer tasks are specified to resources with well bandwidth. Liu and Liao [10] offered an adaptive fine-grained job scheduling algorithm to relegate fine-grained tasks according to the bandwidth processing capacity of the current available resources. Although these methods significantly decrease the impact of scheduling and alignment time overhead, they do not investigate data dependencies. Task granularity control has also been addressed in theoretical workflows. For example, Singh et al. [11] displayed level- and label based clustering. In level-based grouping, tasks at the same level of the workflow can be grouped together.

The number of groups or tasks per cluster is defined by the user. In the label-based clustering approach, the user labels tasks that should be grouped together. Although their work investigates data dependencies between the workflow levels, it is done handle by the users, which is inclined to errors and it is not scalable. Lately, Ferreira da Silva et al. [12], [20] propounded task clustering and ungrouping algorithms to control workflow task granularity in an online context and non-clairvoyant, where none or little characteristics about the usage or resources are known in advance. Their research significantly decrease scheduling and alignment time overheads, but did not investigate data dependencies.

III. DESIGN AND MODELS

A. *Workflow model*

We model workflows as Directed Acyclic Graphs (DAGs), G = (V,E,q, w), in which V = {t_i |i =1, 2, . . . , m} be the finite set of tasks t_i and E be the set of directed arcs of the form e(t_i, t_j), An edge e (t_i,t_j) represents data or control flow dependencies from t_i to t_j where t_i is called a prior of t_j , and t_j is a successor of t_i and q_i represents the computational cost of task t_i. The weight w of e (t_i,t_j) represents the communication cost from task t_i to t_j Grouping tasks without considering these dependencies may lead to data locality problems,. An unclustered job contains only one task that has one process or computation. A clustered job contains multiple tasks to be executed in a sequence or in parallel. In our models and experiments, tasks within a job are executed in a sequential order.

B. *Resources model*

When a workflow receives, it will break into subgraphs, according to knowledge about available resources, by executing the Dynamic Clustering Workflow (DCW) algorithm and then scheduler will run the local level scheduling and map tasks in subgraph to local computational node. A computational resource is denoted as R_i where i is the resource id. Let P_i is the expected performance of R_i so The average performance of all available Computational resources *P* is given by:

$$\bar{P} = \frac{1}{m}\sum_{1<i<m} \bar{P}_{.i} \qquad (1)$$

Where m is the number of resource. We assumed that the computation power of resource effect on the time required completing a task and the time to finish a data transfer is commensurate with the communication cost of the link.

C. *DCW: Scheduling Algorithm*

High parallelism means to dispatch more tasks simultaneously to different resources. To achieve this, the main task graph needs to be partitioned into subgraphs and each subgraph has to be assigned to a resource**.** The main parameter must be determined in partitioning of graph, is the number of partitions should be made (N). To determine N, CTC parameter is used. *CTC* is the ratio of communication-to-computation. A high *CTC* value means a task graph is computation intensive. Formally, *CTC* is defined as:

$$CTC = \frac{\bar{P}}{\bar{q}} \qquad (2)$$

Where $\bar{q}$ is the average processing requirement of all tasks. As the *CTC* increases, high parallelism is preferred because more computational power is required. DCW determine the number of graph partitions to be created, *N*, according to different workflow patterns and communication costs:

$$N = \min\left(m, \left\lceil \frac{CTC}{ComCost\_C_i} \times \beta \right\rceil\right) \qquad (3)$$

$$\beta = \Delta p/\bar{P} \qquad (4)$$

$$\Delta p = \sqrt{\frac{1}{m}\sum_{1<i<m}(\bar{P} - P_i)^2} \qquad (5)$$

Here β is the accelerating factor and Δ*p* is the standard deviation on the computational power of different resource. It is clear that *N* is always no greater than the number of available resource.

With the number of subgraphs, to be created, DCW is to specify how tasks in the main graph should be assigned. To achieve high parallelism and avoid inessential external communication, the size of a subgraph assigned to a resource should be as large as possible under a certain threshold value. The weights of edges connecting different subgraphs should be as small as possible to minimize communication cost. According to purpose, we need to detect the set of tasks that could run concurrently and distribute them into different subgraphs and then specify the maximum number of nodes that could run concurrently when assigned to the same resource that call *Maximum Concurrent Node* (*MCN*).To get *MCN*, two parameters are defined: For a node t_i in a DAG, its *Earliest Start Time (EST)* is defined as follows:

$$EST(t_i) = \max_{t_j \in Pred(t_i)} (ET(t_j) + e(t_j, t_i)) \qquad (6)$$

Where *pred* (t_i) is the set of immediate predecessors of t_i and Execution Time (ET) of t_j is defined as:

$$ET(t_j) = \frac{q_j}{p} \qquad (7)$$

The Earliest Finish Time (EFT) of t_i is defined as follows:

$$EFT(t_i) = EST(t_i) + ET(t_i) \qquad (8)$$

Now, we can specify which nodes could run concurrently. We call t_i and t_j parallel peers, if following equation is satisfied to one of them: if the *EST*(t_i)after *EST* (t_j)and before the EFT (t_j) or the EFT (t_i) after *EST* (t_j)and before the EFT (t_j).By checking parallel peers of every node, we can find the largest set of concurrent nodes in task graph *G*, whose size is the value of *MCN*. The size of a partition is also related with the computational power of resource clusters.

To be adaptive to the dynamic and heterogeneous nature of the computational resource on the one hand and on the other, Having Load balancing on resources, DCW introduces two parameters to describe the related properties of a resource, namely the resource *Rank* (R_i) and *Parallel Threshold* (T_i).

Thus,

$$R_i = 1 - \frac{1}{2W_i}\left(d_i + n_i\right) \qquad (9)$$

Where d_i is the standard deviation of the performance fluctuation of P_i in various time slots, and n_i is The number of refers to the source per unit of time and $_{Wi}$ is weight of resource i, that is calculated based on various parameters such as failure rate or computation power, etc. a larger d_i means the performance of p_i is more unstable and a larger n_i means High resource load. Then the initial threshold value T_i' of i is defined as:

$$t_i = R_i \times \frac{w_i}{\sum_{i=1}^{m} w_i \ /m} \qquad (10)$$

The above equation, the parallel computation power of each cluster is also taken into account. Then the parallel threshold value T_i of each subgraph to be created is given by:

$$T_i = \frac{t_i}{\sum_{i=1}^{N} t_i} MCN(G) \qquad (11)$$

The pseudo-code for the DCW is given in Algorithm1. DCW is described as follows. When a scheduler receives a job, it first traverses the job's DAG *G* to compute its *CTC*, the number of partitions to create *N*, and the level of each task node. Then, the scheduler selects *N* resources whose ranks are the highest (*N* out of *m)*, according to its knowledge. A graph partition iteration checks every remaining nodes in *G* to determine whether the node can be put into a subgraph *G'*.

Algorithm1 DCW

```
Input: A task DAG G (V, E) and available resources.
Output: A subgraph of G and assigned to resource.
1. Compute CTC, N and EFT and EST of each node in G, and resource ranks R and threshold values T;
2. Mark all nodes unassigned;
3. Select the resource with the highest threshold value Ti;
4. Find the largest edge e (a, b) in which a, b have not been checked;
5. IF $Br(G' + a) \le Ti$ and $Br(G' + b) \le Ti$ {
6. Add a and b to G' and mark a and b as checked;
7.}
8. $G = G - G'$;
9. Put ($G'$, $R_i$) in the output set.
```

VI. DISCUSSION

In this section, we will present our simulation of the Dynamic Clustering Workflow algorithm.

A. Simulation model

We evaluate the performance of DCW using different task graph. In terms of input task graphs, We used random graph generation with the ability of generating a variety of task graphs according to different configuration parameters, such as average number of task nodes of each graph, average outgoing and incoming degrees for each node in a graph, and computational and communication cost for each type of task nodes and edges. Each graph has a single entry and a single exit node. We used the random graph generator discussed in [13]. This random graph generator requires following input parameters:

- *V*: The number of task in the DAG.
- Out degree: The ratio of maximum out edges of a node to total nodes of the DAG.
- Communication to Computation Ratio (CCR). It is the ratio of the average communication cost to the average computation cost.
- *B*: The computational heterogeneity factor of resources.
- *α*: The depth parameter of the DAG. This parameter indicates the depth of a DAG by using the uniform distribution with the mean value equal to $\frac{\sqrt{V}}{\alpha}$.

The values for the input parameters are shown in Table 1. The following metrics are used to evaluate the performance of the proposed algorithm:

TABLE I
THE VALUES FOR THE INPUT PARAMETERS

PARAMETER	Value
V	20, 40, 60, 80, 100
Out degree	0.1, 0.2, 0.3, 0.4, 0.5
CCR	0.1, 0.5, 1.0, 5.0, 10.0
α	0.5, 1.0, 2.0
B	0.1, 0.25, 0.5, 0.75, 1.0

The simulated computing platform is composed of 20 single homogeneous core virtual machines (worker nodes), which is the quota per user of some typical distributed environments.

B. Simulation results

To evaluate the performance of the proposed algorithm we compared it with the Selective Reclustering (SR), Dynamic Reclustering (DR), Vertical Reclustering (VR) [14], and Horizontal Distance Balancing (HDB) [15] methods. The performance metric used in the simulation is the average makespan. To test our proposed algorithm, the following parameters are considered in the experiment: (1) The average number of task nodes in a graph *v* (2) The ratio of the average degree of a task node to the total number of tasks in a graph (Edge density in a graph) (3) *CCR*. Fig. 1 presents the average makespan of our approach over the others approach with respect to different number of task.

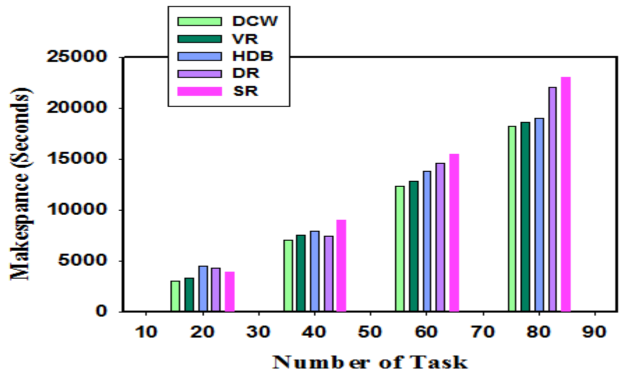

Fig. 1. Avg Makespane with different number of task

Fig. 2 presents the average makespan of our approach over the others approach with respect to different degree.

Fig. 3 presents the average makespan of our approach over the others approach with respect to different CCR.

Experiment 1: Fig. 1 shows the performance of the Horizontal Clustering (HC), Selective Reclustering (SR), Dynamic Reclustering (DR), and Vertical Reclustering (VR) and propose algorithm (DCW). DCW and VR significantly improve the makespan when compared in a large scale.

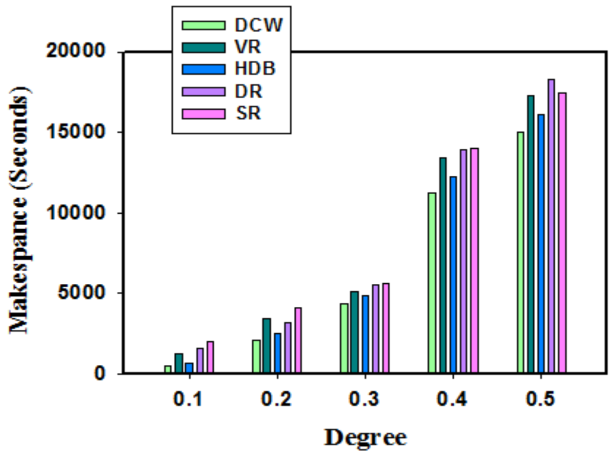

Fig. 2. Avg Makespane with different degree

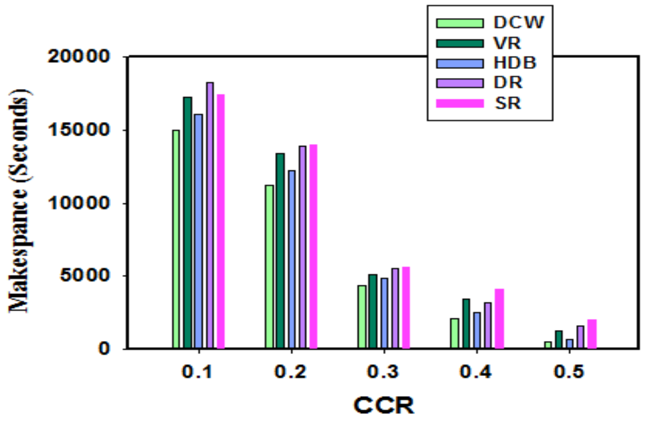

Fig. 3. Avg Makespane with different CCR

Experiment 2: Fig. 2 and Fig. 3 show the performance of the task clustering methods, according to the ratio of maximum out edges of a node to total nodes of the DAG. DCW and HDB improve the makespan because they merge tasks based on data dependencies.

V. CONCLUSION

We proposed task clustering methods that try to balance the workload across clusters and improve the makespan. In addition, we compared our method with four algorithms. Experimental results showed that the proposed method significantly improve the workflow's makespan when compared to an existing task clustering method used in workflow management systems.

REFERENCES

[1]. R. Ferreira da Silva, G. Juve, E. Deelman, T. Glatard, F. Desprez, D. Thain, B Tovar, M. Livny, "Toward fine-grained online task characteristics estimation in scientific workflows", in: Proceedings of the 8th Workshop on

Workflows in Support of Large-Scale Science, WORKS'13, ACM, 2013, pp. 58–67.

[2]. G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, "Characterizing and profiling scientific workflows", 29 (2013) 682–692. Special Section: Recent Developments in High Performance Computing and Security.

[3]. 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.

[4]. 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, vol. 44, 2005, pp. 41–48.

[5]. 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.

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

[7]. N. Muthuvelu, C. Vecchiolab, I. Chai, E. Chikkannana, R. Buyya, "Task granularity policies for deploying bag-of-task applications on global grids", Future Gener. Comput. Syst. 29 (1) (2012), pp. 170–181.

[8]. W.K. Ng, T. Ang, T. Ling, C. Liew, "Scheduling framework for bandwidth-aware job grouping-based scheduling in grid computing", Malays. J. Comput. Sci. 19 (2) (2006), pp. 117–126.

[9]. T. Ang, W. Ng, T. Ling, L. Por, C. Lieu, "A bandwidth-aware job grouping-based scheduling on grid environment", Inf. Technol. J. 8 (2009), pp. 372–377.

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

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

[12]. R. Ferreira da Silva, T. Glatard, F. Desprez, "On-line, non-clairvoyant optimization of workflow activity granularity on grids", in: F. Wolf, B. Mohr, D. Mey (Eds.), Euro-Par 2013 Parallel Processing, in: Lecture Notes in Computer Science, vol. 8097, Springer, Berlin, Heidelberg, 2013, pp. 255–266.

[13]. Topcuoglu H, Hariri S, Wu M-y, "Performance-effective and low-complexity task scheduling for heterogeneous computing", Parallel and Distributed Systems, IEEE Transactions on. 2002; 13(3): 260-74.

[14]. Weiwei Chena, Rafael Ferreira da Silva a, Ewa Deelmana, Rizos Sakellariou, "Using imbalance metrics to optimize task clustering in scientific workflow executions", 2015, Future Generation Computer Systems.

[15]. Weiwei Chen, Rafael Ferreira da Silva, Ewa Deelman, Rizos Sakellariou, "Balanced Task Clustering in Scientific Workflows", 2013 IEEE 9th International Conference on e-Science.