Sharing of Cluster Resources among Multiple Workflow Applications

Uma Boregowda[1] and Venugopal Chakravarthy[2]

[1]Department of Computer Science and Engineering, Malnad College of Engineering, Hassan, India
[2]Department of Electronics and Engineering, Sri Jayachamarajendra College of Engineering, Mysore, India

ABSTRACT

Many computational solutions can be expressed as workflows. A Cluster of processors is a shared resource among several users and hence the need for a scheduler which deals with multi-user jobs presented as workflows. The scheduler must find the number of processors to be allotted for each workflow and schedule tasks on allotted processors. In this work, a new method to find optimal and maximum number of processors that can be allotted for a workflow is proposed. Regression analysis is used to find the best possible way to share available processors, among suitable number of submitted workflows. An instance of a scheduler is created for each workflow, which schedules tasks on the allotted processors. Towards this end, a new framework to receive online submission of workflows, to allot processors to each workflow and schedule tasks, is proposed and experimented using a discrete-event based simulator. This space-sharing of processors among multiple workflows shows better performance than the other methods found in literature. Because of space-sharing, an instance of a scheduler must be used for each workflow within the allotted processors. Since the number of processors for each workflow is known only during runtime, a static schedule can not be used. Hence a hybrid scheduler which tries to combine the advantages of static and dynamic scheduler is proposed. Thus the proposed framework is a promising solution to multiple workflows scheduling on cluster.

KEYWORDS

Task scheduling, workflow, DAG*, PTG.*

1. INTRODUCTION

Many business, industrial and scientific areas, such as high-energy physics, bioinformatics, astronomy, epigenomics, stock market and others involve applications consisting of numerous components(tasks) that process data sets and perform scientific computations. These tasks communicate and interact with each other. The tasks are often precedence-related. The problem of scheduling jobs with precedence constraints is an important problem in scheduling theory and has been shown to be NP-hard [1]. Data files generated by one task are needed by other tasks. The requirement of large amount of computations and data storage of these applications can be provided by a cluster. Because of huge technological changes in the area of parallel and distributed computing, powerful machines are now available at low prices. This is visible in large spreading of cluster with hundreds of homogeneous/heterogeneous processors connected by high speed interconnection network [2]. This democratization of cluster calls for new practical administration tools.

The task scheduling problem is to allocate resources (processors) to the tasks and to establish an order for the tasks to be executed by resources. There are two different types of task scheduling: static and dynamic. Static strategies define a schedule at compile time based on estimated time required to execute tasks and to communicate data. Static schedule can be generated only when the application behaviour is fully deterministic and this has the advantage of being more efficient and a small overhead during runtime. The full global knowledge of application in the form of DAG will help to generate a better schedule. Dynamic strategies, on the other hand are applied when tasks are generated during runtime. Tasks are assumed to be non-preemptive.

Workflows have recently emerged as a paradigm for representing complex scientific computations [26]. Few widely used example workflows are Montage (Fig. 2), cybershake, LIGO, SIPHT. Workflows represented by one of many workflow programming languages can be translated into DAG, in general. Thus workflow scheduling is essentially a problem of scheduling DAG. Although much work has been done in scheduling single workflow [3], multiple workflow scheduling is not receiving deserved attention. Few initial studies are found in the literature [4, 5]. Because of huge computing power of a cluster and the inability of a single DAG to utilize all processors on cluster, multiple DAG applications need to be executed concurrently. Thus a scheduler to deal with multi-user jobs with the objectives of maximizing resource utilization and minimizing overall DAG completion time is essential. The contributions of this paper are 1) a new method to find minimum, optimal and maximum number of processors that can be allotted for a DAG and this information is used to find one best way to share available processors among multiple DAGs 2) a framework to receive submission of DAGs, find the allotment for each submitted DAG and schedule tasks on allotted processors, with the objectives of maximizing resource utilization and minimizing overall completion time.

1.1. Application Model

The data flow model is gaining popularity as a programming paradigm for parallel processors. When the characteristics of an application is fully deterministic, including task's execution time, size of data communication between tasks, and task dependencies, the application can be represented by directed acyclic graph (DAG) as shown in Fig.1. Each node in DAG represents a task to be performed and the edges indicate inter-task dependencies. Node weight stands for the computation cost of the corresponding task and the edge cost represents the volume of data to be communicated between the corresponding nodes. The node and edge weights are usually obtained by estimation or profiling. Communication-to-Computation (CCR) is the ratio of average communication cost to the average computation cost of a DAG. This characterizes the nature of DAG. The objective of scheduling is to map tasks onto processors and order their execution so that task dependencies are satisfied and minimum overall completion time is achieved. Makespan is the total time required to complete a DAG.

1.2. Platform

A cluster with ‘P’ homogeneous processors, each of which is a schedulable resource is considered. Processors are interconnected by a high speed and low latency network. A processor can communicate with several other processors simultaneously with multi-port model.

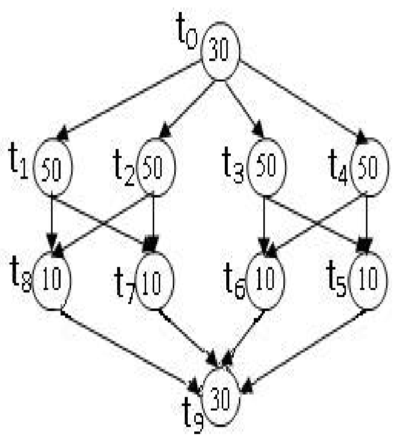


Figure 1. A Typical DAG

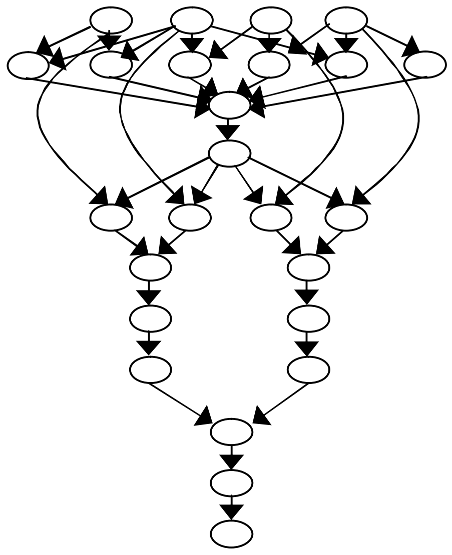


Figure 2. Montage – a Workflow

2. RELATED WORK

Extensive work has been done on scheduling a single DAG [6, 7, 8]. Zhao et al.[4] have proposed few methods to schedule multiple DAGs. One approach is to combine several DAGs into one by making the entry nodes of all DAGs, immediate successors of new entry node and then use standard methods to schedule the combined DAG. Another way is to consider tasks from each DAG in round robin manner for scheduling. They have proposed other policies to optimize both makespan and fairness. The key idea is to evaluate, after scheduling a task, the slowdown value of each DAG against other DAGs and make a decision on which DAG must be considered next for scheduling.

A list scheduling method to schedule multi-user jobs is developed by Barbosa et al. [9] with an aim to maximize the resource usage by allowing a floating mapping of processors to a given job, instead of the common mapping approach that assigns a fixed set of processors to a user job for a period of time. A master DAG where each node is a user job and each edge representing a priority of one job over another is constructed using all submitted DAGs. A list scheduling algorithm [6] is used to schedule all tasks of Master DAG. The master DAG is created based on job priorities and deadlines

Bittencourt et al. [10] have used Path Clustering Heuristic (PCH) to cluster tasks and the entire cluster is assigned to a single machine. They have proposed four heuristics which differ in the order tasks of multiple DAGs are considered for scheduling. The methods are sequential, Gap search method, Interleave algorithm and Group DAGs method. A meta-scheduler for multiple DAGs [11] merges multiple DAGs into one to improve the overall parallelism and optimize idle time of resources. The efforts are limited to the static case and they do not deal with dynamic workloads.

Duan et al. [12] have proposed a scheduling algorithm based on the adoption of game theory and idea of sequential cooperative game. They provide two novel algorithms to schedule multiple DAGs which work properly for applications that can be formulated as a typical solvable game. Zhifeng et al. [13] addresses the problem of dynamic scheduling multiple DAGs from different users. They expose a similar approach from Zhao et al. [4] without merging DAGs. Their algorithm is similar to G-heft algorithm.

An application which can exploit both task and data parallelism can be structured as Parallel Data Graph (PTG) in which task can be either sequential or data parallel. Data parallelism means parallel execution of the same code segment but on different sections of data, distributed over several processors in a network. A DAG is a special case of PTG where task can only be sequential task. Thus PTG scheduling is quite similar to DAG scheduling. Not much work is carried out in Multiple PTG scheduling. Tapke et al. [5] have proposed an approach where each PTG is given a maximum constraint on number of processors it can use and tasks are scheduled using a known PTG scheduling algorithm. The size of each processor subset is determined statically according to various criteria pertaining to the characteristics of PTG like maximum width, total absolute work to be done and proportional work to be carried out.

Sueter et al. [14] have focused on developing strategies that provide a fair distribution of resources among Parallel Task Graphs (PTG), with the objectives of achieving fairness and makespan minimization. Constraints are defined according to four general resource sharing policies: unbounded Share(S), Equal Share (ES), Proportional Share (PS), and Weighted Proportional Share (WPS). S policy uses all available resources. ES policy uses equal resources for each PTG. PS and WPS use resources proportional to the work of each PTG, where the work is considered as critical path cost by width of PTG.

A study of algorithms to schedule multiple PTGs on a single homogeneous cluster is carried out by Casanova et al. [15]. Therein it is shown that best algorithms in terms of performance and fairness all use the same principle of allocating a subset of processors to each PTG and that this subset remains fixed throughout the execution of the whole batch of PTGs. The basic idea in job schedulers [19] is to queue jobs and to schedule them one after the other using some simple rules like FCFS (First Come First Served) with priorities. Jackson et al. [20] extended this model with additional features like fairness and backfilling.

Online scheduling of multiple DAGs is addressed in [16]. Authors have proposed two strategies based on aggregating DAGs into a single DAG. A modified FCFS and Service-On-Time (SOT) scheduling are applied. FCFS appends arriving DAGs to an exit node of the single DAG, while SOT appends arriving DAGs to a task whose predecessors have not completed execution. Once the single DAG has been built, scheduling is carried out by HEFT.

An Online Workflow Management (OWM) strategy [18] for scheduling multiple mix-parallel workflows is proposed. DAG tasks are labelled, sorted, and stored into independent buffers. Labelling is based on the upward rank strategy. The sorting arranges tasks in descendent order of the task rank. Task scheduling referred to as a rank hybrid phase determines the task execution order. Tasks are sorted in descending order when all tasks in the queue belong to the same workflow. Otherwise, they are sorted in ascending order. Allocation assigns idle processors to tasks from the waiting queue.

Raphael et al. [23] have addressed online scheduling of several applications modelled as work-flows. They have extended a well-known list scheduling heuristic (HEFT) and adapted it to the multi-workflow context. Six different heuristics based on HEFT key ideas are proposed. These heuristics have been designed to improve the slowdown of different applications sent from multiple users.

Much work has not been done on scheduling multiple DAG applications. A common approach is to schedule a single DAG on fixed number of processors [6] but methods to find the number of processors to be used for a DAG, are not found in literature. Tapke et al. [5] have proposed methods to find maximum resource constraint for each PTG, while scheduling multiple PTGs. But they have not restricted scheduling of a PTG to the fixed set of processors. A method proposed by Barbosa et al. allows floating number of processors to a given job, instead of fixed number of processors. Many existing workflow scheduling methods do not use fixed set of processors for each DAG. Instead a task based on some heuristic is picked among tasks of all DAGs and is scheduled on a processor where it can start earliest based on some heuristic. The work proposed in this paper is similar to Barbosa [9] method, in the sense that a fixed set of processor is allocated for each DAG which later can be varied during runtime with the objective of maximizing resource usage. Their work does not address several issues like - how initial processor allotment for each DAG is made, a method to decide number of DAGs to be scheduled concurrently among several submitted DAGs, to deal with online submission of DAGs. This work addresses all the above mentioned issues.

3. PROCESSOR ALLOTMENT FOR A DAG

A schedule for a DAG can be obtained with varied number of processors. By increasing the number of processors allotted for a DAG, its makespan decreases. The gain in terms of reduction in makespan, reduces as more number of processors are allotted to a DAG. This is due to communication overhead and limited parallelism present in DAG. The optimal and maximum number of processors for a DAG will help in finding processor allotment while scheduling multiple DAGs concurrently on a cluster.

3.1. Maximum Number of Processor for a DAG

The maximum number of processors a DAG can utilize depends on its nature and degree of parallelism present in it. The number of allotted processors, beyond which DAG's makespan does not decrease with any more additional processors, is the maximum number of processors that can be utilized by a DAG. A brute force method can be used to find this, by making several calls to scheduling method and recording the makespan for each case. But an efficient binary search based method [24] is used in this work and its time complexity is O(log n) against O(n) of the brute force method.

3.2. Optimal Number of Processor for a DAG

Optimal number of processors for a DAG is that number up to which every added processor is utilized well and beyond it, they are underutilized. With increase in number of allotted processors, DAG's makespan decreases and average processor utilization decreases due to communication overhead and limited parallelism. Average processor utilization can best be measured using computing area, which is the product of makespan and the number of processors used. In this work, computing area is used to find the optimal number of processors for a DAG. As processor allotment to a DAG is increased, makespan decreases and computing area increases. Initially decrease in makespan is more than increase in computing area, justifying the worthiness of increase in processor allotment. After the processor allotment reaches a certain value, the increase in computing area is more than the decrease in makespan for every added processor, indicating that any further increase in processors allotment is not of significant use.

By successively increasing number of processors allotted for a DAG, makespan and computing area are recorded. The number of processors for which decrease in makespan becomes less than the increase in computing area, fixes the optimal number of processors for a DAG.

4. MULTIPLE DAGS SHARING CLUSTER

It is advantageous to schedule multiple DAGs simultaneously on a cluster instead of dedicating the entire cluster to a single DAG, due to communication overhead. Furthermore, it is beneficial to schedule more number of DAGs each with relatively less number of processors than scheduling less number of DAGs each with large number of processors, because of communication overhead. The returns, in terms of decrease in makespan, for each additional processor differs for each DAG depending on its nature and number of processors already allotted to it. Hence additional processors must be allotted to those DAGs which will be benefitted most by means of reduction in makespan. To find the most benefitting DAGs, reduction in makespan for the next added processor must be known for each DAG. Reduction in makespan for every added processor can be best captured as an equation using regression analysis and are provided to scheduler along with DAG. During regression analysis of large number of DAGs, it is observed that for any DAG, makespan reduction follows either exponential or power curve. Thus for each DAG, makespan reduction for each added processor is recorded and curve fitting is done. The type of equation and its constants are stored along with each DAG, which then is used by the scheduler while finding processor allotment for each DAG, while scheduling multiple DAGs. The scheduler is invoked when a DAG arrives or a DAG completes execution. The minimum number of processors to be allotted for each DAG is assumed to be four, by conducting the experiments large number of times. The algorithm for the proposed scheduler is given below.

```
Algorithm multi_dag_scheduler()
// information submitted along each DAG – opt_proc, max_proc, eqn_type, eqn_const
//avail_proc – currently available number of free processors
// let min_core = 4
Input : submitted DAGs
Output : processor allotment and calling an instance of scheduler for each DAG

Step 1:  if (arrival)  then     // DAG has arrived
Step 2:       if (avail_proc < min_core ) then
Step 3:           append to waiting queue
Step 4:       else
Step 5 :          allot (min_proc or max_proc or opt_proc) whichever best fits avail_proc
Step 6 :          create an instance of scheduler for a DAG on allotted processors
Step 7 :      endif
Step 8:   else        //DAG has completed
Step 9:       if ( waiting queue is not empty) then
Step 10 :         do_allot()
Step 11 :     endif
Step 12 : end_algorithm
```

```
Algorithm do_allot()
Step 1:  remove those many number of DAGs from queue beginning, whose sum of their
           min_proc is less than avail_proc
Step 2 : if (sum of opt_proc of all removed DAGs is less than available processors) then
Step 3 :      allot opt_proc to each removed DAG
Step 4 :  else
Step 5 :      for each removed DAG allot their min_proc number of processors
Step 6 : endif
Step 7 :  if  (free processors are left)   then
Step 8 :    distribute those free processors among DAGs, in such a way that each processor is
             added to that DAG for which it  yields maximum reduction in makespan, using
             equations types and their constants
Step 9 :  endif
Step 10 : end_algorithm
```

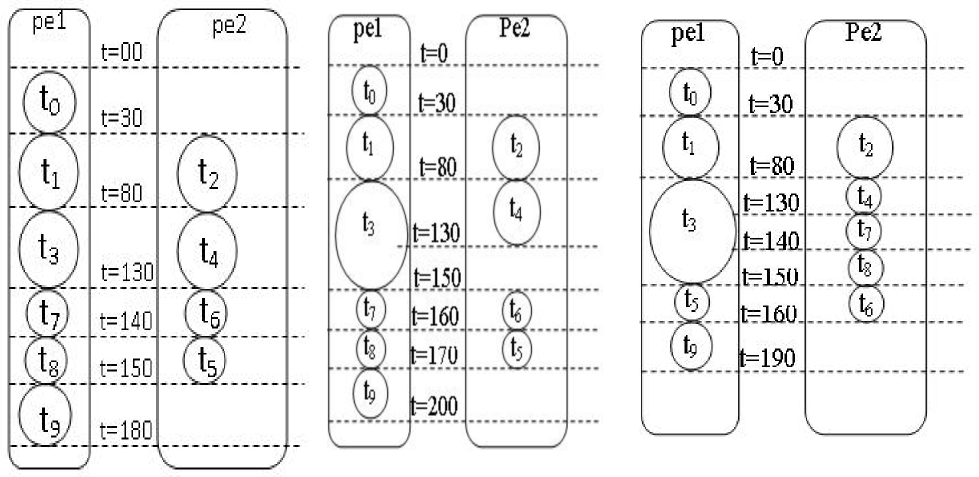
5. A HYBRID SCHEDULER

Two important factors that affect task schedule are - balancing the load across all processors and minimizing communication time. Towards these end, novel hybrid scheduling methods are proposed to map a task to the right processors and to order their execution on the allotted processor, using the static information of DAG. A dynamic scheduler tries to balance the load on all processors and the only information that it can get is the number of tasks yet to be completed one ach processor. Since the scheduler does not know the execution time of tasks, it can not find the exact load on each processor, thus the schedule suffers. Static scheduler is generally preferred when the nature of application is deterministic because it generates schedule close to optimal and incurs low runtime overhead. Also, task execution and communication very often differs from the estimated values. In all such cases, use of static schedule to run the application suffers. Thus the static scheduler run into the problem of pre-committing the schedule to a processor, even when better scheduling is possible on another processor. Also, it is not possible to predict execution time of tasks, as it depends on several factors like actual input, memory and bandwidth available to it, other co-running processes.

5.1. Motivational Example

Static scheduler is generally preferred when the nature of application is deterministic because it generates schedule close to optimal and incurs low runtime overhead. An example is used to illustrate the motivation for this work. This shows the drawback/behavior of static schedule when execution time of tasks is different from their predicted values. A DAG of 10 tasks, along with execution time of each task is shown in Fig. ?? . In order to understand the real spirit behind this work, communication time is not considered. The static schedule for the DAG in Fig ?? using ETF method is given below.

Processor PE1 – t0, t1, t3, t7, t8, t9
Processor PE2 – t2, t4, t5, t6



The execution map of DAG is given in Fig. ??, using the above static schedule, when all tasks take exactly the predicted time for execution. The DAG completion time is 180 units. Due to runtime dynamics, if task t3 gets delayed, taking 70 units instead of 50 units, then the execution map is shown in Fig ??. It is evident from the figure Fig ??, that processor P2 is idle waiting for task t6 to become ready, which in turn is waiting for the completion of task t3. But a better schedule would have been to start executing the already task t8 on PE1 instead of waiting for task t6. With this delay in task t3 time, DAG completion time is increased from 180 to 200 units. The DAG execution by the proposed hybrid scheduler is given in Fig. ??. Here tasks are maps are mapped to the processors when they become ready for execution, with the objective of balancing the load across all processors. Thus when task t3 gets delayed, other ready tasks t7 and t8 are scheduled on PE2, thus eliminating idle time. The improved schedule completes DAG in 190 units, an improvement over the use of static schedule.

6. EXPERIMENTAL SETUP AND RESULTS

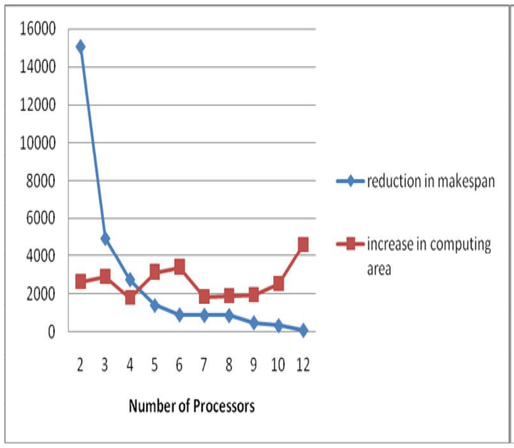
5.1. Experiment Setup

A discrete-event based simulator is developed to simulate the arrival, scheduling, execution and completion of DAGs. Simulation allows performing statistically significant number of experiments for a wide range of application configurations. Poisson distribution is used to simulate the arrival time of DAGs. Several kinds of benchmark DAGs from several sources are used to experiment the proposed scheduler for different types of DAGs. Series-parallel DAGs from Task Graphs For Free [22], random DAGs from Standard Task Graph Set [21], DAGs of linear algebra applications like FFT, LU decomposition, Gauss-elimination, Laplace transform and workflows like LIGO, cybershake, Montage, SIPHT[25]. DAGs with CCR values of 0.1, 0.4, 1 and 5 are used in experiments.

5.2. Results and Analysis

5.2.1. Optimal and Maximum Number of Processors for a DAG

An efficient binary search based method [24] with time complexity of O(log(n)) is used to find the maximum number of processors a DAG can utilize. The decrease in makespan and increase in computing area (decrease in average processor utilization) for every added processor is used to fix the optimal number of processors for a DAG. The plot of decrease in makespan and increase in computing area for different number of processors, for a DAG is given in Fig.3. The crossover point gives the optimal number of processors for that DAG. The method can be used for any kind of DAG.



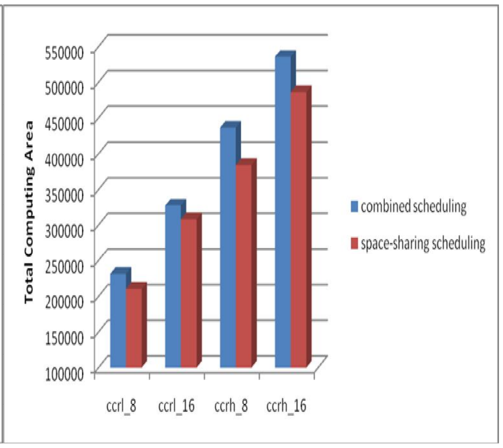


Figure 3. To find Optimal Number of Processors for a DAG

5.2.1. Multiple DAGs Scheduling

Recent works on multiple DAG scheduling [4, 5, 9, 10, 11, 12, 14] have not considered allotment of fixed set of processors to a DAG. Instead, tasks from all DAGs are scheduled on any processor on which they can start earliest, using some heuristic. Hence initially, it is proved experimentally that space partitioning of processors among multiple DAGs, delivers improvement in performance compared to combined DAGs scheduling. To experiment this, a set of DAGs of all kinds, were scheduled on a cluster with 100 numbers of processors. The metric used is the sum of computing area of all scheduled DAGs. To study the effect on both computation intensive and communication intensive applications, DAGs with both low and high CCR are considered. Two sets of DAGs each with 8 and 16 number of DAGs, under each category are considered. Thus the four categories of DAGs are labelled as ccrl_8, ccrl_16, ccrh_8 and ccrh_16. Since the behaviour depends on the nature of DAG, 50 sets of DAGs are considered for each category. Care is taken to consider all different types of DAGs in the sets of DAGs. The results obtained from 50 sets are averaged and the same is shown in Fig. 4. The performance of the proposed method is better than combined DAGs scheduling for all four categories of DAGs. For the category ccrh_8, proposed method shows maximum improvement of 12%, since DAGs are communication intensive and thus scheduling tasks on fixed set of processors reduces time to complete the DAG. Performance improvement is only 9% for the category ccrh_18, as there is less scope for further improvement due to large number of DAGs being scheduled together.

Figure 4. Combined DAGs scheduling vs Proposed Space-sharing Schedule

The benefits of space partitioning processors which cannot be measured for DAGs with dummy tasks are 1) as tasks of a DAG are scheduled on the same set of processors, they will be benefitted from cache-warm and secondary memory warm. 2) an online scheduler can be used for each DAG, after allotting a set of processors to it. 3) processor allotment for a DAG can be varied depending on availability of processor, with the objectives of maximizing resource utilization.

A highlight of this work is to find one best way to share available processors among multiple DAGs, using regression analysis. The proposed work is compared against policies proposed by Tapke et al. [14] - unbounded Share(S), Equal Share (ES), Proportional Share (PS), and Weighted Proportional Share (WPS). The strategy S which is a selfish allocations and tasks of different DAGs are not differentiated is used as a baseline performer for other strategies as it gives an indication of performance of heuristics originally designed for single DAG. Values obtained are normalized with the value of S strategy, to help in comparison. Performance metric used is average makespan and resource utilization which is measured as the sum of computing area of all DAGs scheduled together. Five categories of DAGs each with 4, 8, 12, 16 and 20 number of DAGs are considered. Random, series-parallel, linear algebra DAGs and various workflows like montage, SIPHT, epigenemics, LIGO are considered. 100 sets of DAGs are considered for each category and the results obtained are averaged. The result is shown in Fig. 5 and Fig. 6. The proposed method is better than all policies found in literature.

For less number of DAGs, performance of all methods is almost the same, as there will not be much conflict for resources. With more number of DAGs, resource conflicts increase and the proposed method shows considerable good performance over previous methods.

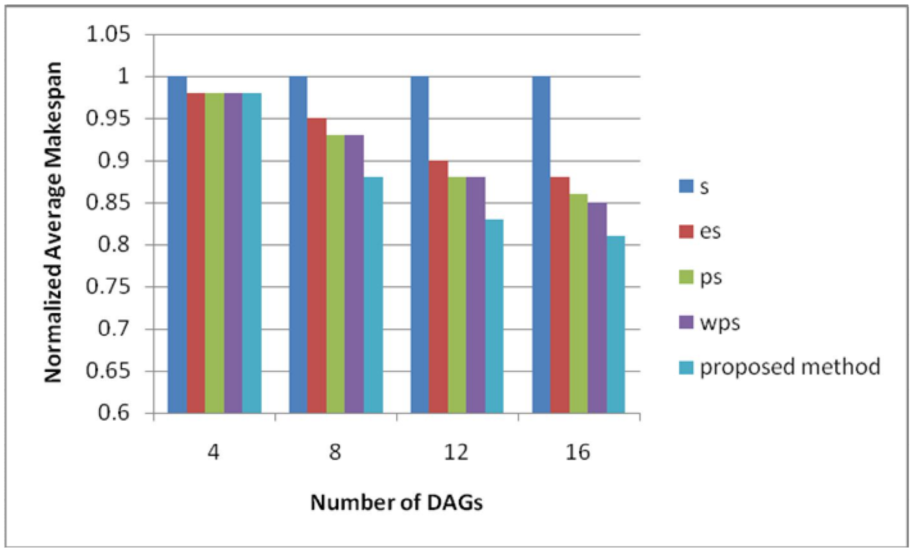


Figure 5. Normalized Average Makespan of Set of DAGs

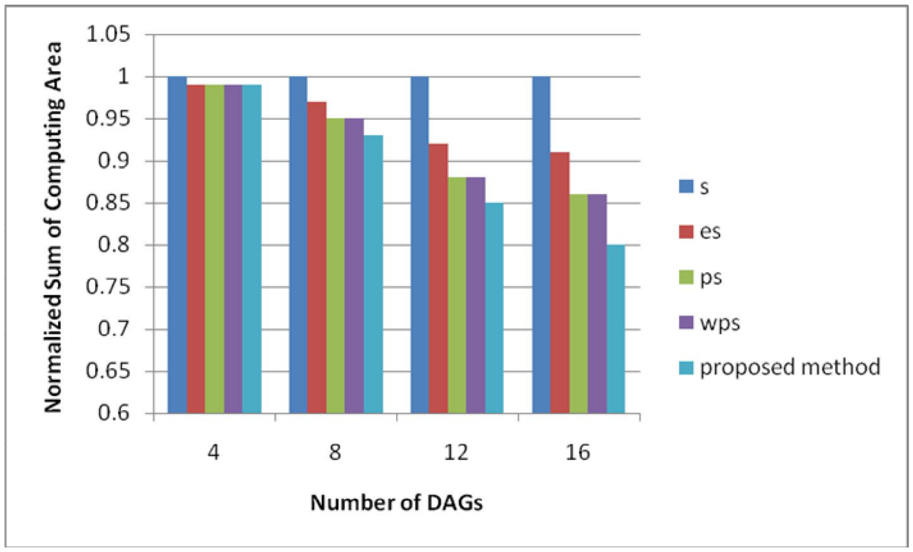


Figure 6. Normalized Sum of Computing Area of Set of DAGs

7. CONCLUSIONS

Multiple DAGs scheduling on a cluster is not receiving the deserved attention. Few methods found in literature performing combined DAGs scheduling. But in this work, it is proposed to allot a fixed number of processors to each DAG and an instance of local DAG scheduler to schedule DAG's tasks only on the allotted fixed set of processors. A method to find the maximum and optimal number of processors that can be allotted to a DAG is given, which will be used to find the processor allotment for each DAG while scheduling multiple DAGs. A new framework to schedule multiple DAGs with the objectives of maximizing resource utilization and minimizing DAGs completion time is proposed. Regression analysis is used to find the number of processors to be allotted to each DAG while scheduling multiple DAGs. This method is proved to outperform other methods found in literature by around 10-15%.

The other big advantage of the proposed approach is that instead of static schedule, an online scheduler for each DAG can be used to schedule tasks, as they are generated, onto the allotted processor. An Hybrid scheduler overcomes drawbacks of both static schedule and dynamic schedule. Also static DAG information is used to further improve performance. Because of space sharing of processors, the number of processors allotted to each DAG can be varied during runtime, depending on the availability of free processors.

//This will improve resource utilization, hence performance of the scheduler. In future work, the idea of online scheduler and varied processor allotment for each DAG will be experimented.

REFERENCES

[1] Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3rd edition. Springer (2008)

[2] D.E. Culler, J. P. Singh, and A. Gupta. Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann Publishers, inc., SanFrancisco, CA, 1999..

[3] Yu, J., Buyya, R.: A Taxonomy of Scientific Workflow Systems for Grid computing. SIGMOD Records 34(3),44–49 (2005)

[4] Zhao, H., Sakellariou, R.: Scheduling Multiple DAGs onto Heterogeneous systems. In: Parallel and Distributed Processing Symposium, 2006. (IPDPS 2006). 20th International, pp. 14–pp. IEEE (2006)

[5] N'takp´e, T., Suter, F., Casanova, H.: A Comparison of Scheduling Approaches for Mixed-Parallel Applications on Heterogeneous Platforms. In: Parallel and Distributed Computing, 2007. ISPDC'07. Sixth International Symposium on, pp. 35–35. IEEE (2007)

[6] Y.-K. Kwok and I. Ahmad. Static Scheduling Algorithms for Allocating Directed Task graphs to Multiprocessors. ACM Computing Surveys, 31(4):406–471, 1999

[7] E. Ilavarasan and P. Thambidurai ,.Low Complexity Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Environments , Journal of Computer Sciences 3 (2): 94-103, 2007

[8] T. Hagras, J. Janeček . Static vs. Dynamic List-Scheduling Performance Comparison, Acta Polytechnica Vol. 43 No. 6/2003

[9] J. Barbosa and A. P. Monteiro, A List Scheduling Algorithm for Scheduling Multi-user Jobs on Clusters, High Performance Computing for Computational science- VECPA8 2008, Lecture Notes in Computer Science volume 5336, 2008, pp123-136

[10] Bittencourt L.F., Madeira: Towards the Scheduling of Multiple Workflows on Computational Grids. J. Grid Computing 8, 419–441 (2010)

[11] U. H. Nig and W. Schiffmann. A Meta-algorithm for Scheduling Multiple DAGs in Homogeneous System Environments. In Proceedings of the 18th International Conference on Parallel and Distributed Computing and Systems (PDCS'06), 2006

[12] R. Duan, R. Prodan, and T. Fahringer. Performance and Cost optimization for Multiple large-scale Grid Workflow Applications. In Proceedings of the 2007 ACM/IEEE conference on Supercomputing, pages 1–12, New York, NY, USA, 2007.

[13] Zhifeng Yu and Weisong Shi. A Planner-guided Scheduling Strategy for Multiple work-flow Applications. Parallel Processing Workshops, International Conference on, 0:1–8,2008.
[14] N'takpé, T., Suter, F.: Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations. In: International Parallel and Distributed Processing Symposium/International Parallel Processing Symposium, pp. 1–8 (2009)
[15] Casanova, H., Desprez, F., Suter, F.: On cluster Resource Allocation for Multiple Parallel Task Graphs. Journal of Parallel and Distributed Computing 70(12), 1193–1203 (2010)
[16] Zhu, L., Sun, Z., Guo, W., Jin, Y., Sun, W., Hu, W.: Dynamic Multi DAG Scheduling Algorithm for Optical Grid Environment. New Architecture Management Applications. V 6784(1), 1122 (2007)
[17] H Topcuoglu, S. Hariri and M. Y. Yu, Performance Effective and Low-complexity Task Scheduling for Heterogene Computing, IEEE TPDS, 13(3):260-274, 2002
[18] Hsu, C.-C., Huang, K.-C., Wang, F.-J.: Online scheduling of workflow applications. In Grid environments. Future Gen. Comput. Syst. 27, 860–870 (2011)
[19] R. L. Henderson, Job Scheduling under the Portable Batch System, Job Scheduling Strategies for Parallel Processing, volume 949 of LNCS, pages 279–294, 1995.
[20] D. Jackson, Q. Snell, and M. J. Clement. Core algorithms of Job Schedule, Job Scheduling Strategies for Parallel Processing, volume 2221 of LNCS, pages 87–102, 2001.
[21] Standard Task Graph Set http://www.kasahara.elec.waseda.ac.jp/schedule/.
[22] Task Graphs for Free http://ziyang.eecs.umich.edu/~dickrp/tgff
[23] Raphael Bolze, Frederic Desprez and Benjamin Insard, Evaluation of Online Multi-workflow Heuristics based on List Scheduling Methods, Gwendia ANR-06-MDCA-009
[24] Uma B , C R Venugopal, A Novel Binary Search based method to find minimal number of cores required to obtain minimal makespan, given makespan and given utilization of cores for a DAG application, 4th International Conference on Computer and Automation Engineering, (ICCAE 2012) January 14–15, 2012, Mumbai, India.
[25] https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator
[26] Taylor, I.J., Deelman, E., Gannon, D.B., Shields, Workflows for e-Science - Scientific Workflows for Grids. Springer, New York (2007).

Authors

C R Venugopal received his Ph. D from IIT, Bombay. Presently serving as a Professor in Sri Jayachamarajendra College of Engineering, Mysore, India. His main research interests are Cloud computing, High Performance computing, VLSI Technology and File System development. Has authored more than 50 international conference and journal papers.

Uma B completed M. Tech. in Computer Science and Engineering from IIT, Delhi. Currently working as a Associate Professor in Malnad College of engineering, Hassan, India. Her research interests are Parallel Programming, High Performance Computing and Task scheduling.