International Review on Computers and Software (I.RE.CO.S.), Vol. 8, N. 5
ISSN 1828-6003 May 2013

An Efficient Iterative Modulo Scheduling Approach for Improved Resource Allocation for Effective Multimedia Communication on Grid Computing Environment

G. Saravanan[1], V. Gopalakrishnan[2]

Abstract – *This paper discusses Iterative modulo scheduling techniques on heterogeneous resource for multimedia services to deal with the demands of next-generation multimedia applications on grid computing environment. The proposed scheduling algorithm has the subsequent features, which can be make use of on grid computing background. Initially the algorithm assistance with the resource practice constrained scheduling. The grid mainly consists of the resources that are possessed by decentralized society. Second, the algorithm performs the optimization-based scheduling. It gives an optimal solution to the grid resource allocation problem. Third, the algorithm takes for granted that a set of resources is dispersed geographically and varied in natural world. Fourth, the scheduling method dynamically adjusts to the grid status. It tracks the present workload of the various resources. The proposed algorithm performance is estimated with a set of predefined metrics. In addition to that the simulation results show the outperformance of the Iterative modulo scheduling algorithm.*

Keywords*: Resource Allocation, Scheduling, Grid Computing, Mixed Ancestral Graph*

I. Introduction

In science and commerce they are proposed two methods to resolve the major risk. They are Computational Grids and peer-to-peer (P2P) computing systems [1], [2]. It allows establishing of virtual enterprises (VEs) for allocation and combination of resources [3] to spread around the companies and organizational domains. These networks include numerous resources like workstations, clusters and supercomputers, fabric managing systems and applications like scientific, engineering, commerce and profitable with many essentials. The producers and consumers have many aims, methodology and supply-and-demand designs. To deal with some critical situations established methods to resource management that effort to enhance system-wide performance measure cannot be engaged. Established methods to use centralized policies to want fulfilled state report and a popular fabric management policy, or a reorganized consensus-based policy. Because of the difficult in developing successful Grid situations, it is not possible to derive a suitable system-wide performance matrix and common fabric management policy [4]. In most of the interconnected work in Grid computing Committed to resource management and scheduling problems to implement a conventional style where a scheduling factor determines which jobs are to be executed at which site created certain cost functions (Legion [5], Condor [6], AppLeS [7], Netsolve [8], Punch [9]).

Some cost functions are always driven by system-centric parameters that develop system quantity and utilization rather than increasing the utility of application processing. All resources are treating as the same cost and all the results of all applications are the same value.

But it is not possible in the reality. The consumer does not need to pay the highest price, but they need to pay a particular price based on the demand, value, priority and available budget.

At the different times, the different applications contain different values. In an economics method, the scheduling decision is not done constantly by a single scheduling entity but directed to the end user's obligation.

The conventional cost model frequently deals with software and hardware costs for running applications, more often than not, the economic model charges the consumer for services that they consume based on the value they derive from it. Driver in the ready for action, economic market model provides the Pricing based on the demand of users and the supply of resources. Thus, users have a struggle with other users and a resource owner with other resource owners.

The active and different nature of the grid coupled with critical resource usage policy issues poses exiting challenges to connect the resources in an effective manner. In this paper, the novel optimization scheduling techniques and their performance on open science grid (OSG), a worldwide consortium of university resources consisting of 2000+ CPUs. The recreation results show

Manuscript received and revised April 2013, accepted May 2013

that the proposed algorithm can effectively:

1) To assign grid resources to a set of applications beneath the constraints offered with resource usage policies;
2) To carry out optimized scheduling on heterogeneous resource by means of iterative approach and binary integer programming (BIP);
3) Increase the completion time of workflows in integration with job execution tracking modules of GRIDWAY scheduling middleware [10].

Verboven et al. [11] provides a Grid Information Prediction System (GIPSy) structure which creates sweep prediction based on previous runtime data and there are various parameters used to organize every job. They are providing the prediction based grid scheduling technique which is shared with GIPSy to acquire a real-world grid execution. Because of huge problems of job scheduling and resource management in different systems in grid computing environment, not many established scheduling methods through the heterogeneous processors environment.

Consequently, Hsu and Chen [12] intended performance-oriented and economization-oriented scheduling techniques for handling applications with quality of service (QoS) demands in grid and optimization algorithms which are based on QoS min-min algorithm. Hsu et al. [13] established a two-level scheduling method separates local messages from inter-processor messages and schedules both kinds of messages in separated steps to efficiently avoid synchronization delay.

The intended technique has been developed with the established scheduling method gives to improve schedules for different processors environment.

II. Related Work

The related algorithms work as better than the traditional high performance-computing environment; they do not achieve the acceptable manner with the features of grid discussed in the last section.

A. Iterative List Scheduling [14]

An iterative list-scheduling algorithm, it constructs with scheduling on different computing systems. From the previous results, to enhance the excellence of the schedule in the iterative manner is the main concept of the iterative scheduling algorithm. Whereas the algorithm can possibly give the shorter length it does not support the resource usage policies. It is a constant scheduling algorithm, which simulates the static or stable computing situations. In the non-static and policy constrained grid environment the algorithm may not executed the similar results are shown in this paper.

B. Dynamic Critical Path Scheduling [15]

In this the author tells a static scheduling algorithm for allocating task graphs to fully linked multiprocessors. It reduces the make-span subject to priority constraint, which describes by the significant path of the task graph.

The similar CPU-based scheduling algorithm undertakes that the scheduler could manage the scheduling priority of jobs in a processor. In the grid environment it is not true that the resources have reorganized ownership and different local scheduling policies reliant on their VO.

C. Reliability Cost Driven Scheduling [16]

A two-phase scheme is used here to describe the priority constraints of scheduling of tasks that provides a reliability measure is one of the goals in a real-time and different distributed system. To obtain the exploit reliability from the static algorithm schedules real-time tasks. In the algorithm the utility function get with the initial time of jobs in an application. The algorithm may not be capable to get the suitable resource allocation to the application in the existence of the policy environment.

D. Heterogeneous Earliest Finish Time Scheduling [17]

The main aim of the algorithm is to choose the weight for the nodes and edges of a directed acyclic graph (DAG), and it tests with a number of different methods for computing these weights. The established system uses the mean value approach to get the length of the iterative method to the different resources. The test results evaluate the two schemes. The offline and priority-based scheduling may not be possible to the grid-computing environment

E. Dynamic Level Scheduling [18]

To get a pair of job and a processor in an complete way from the scheduling algorithm. In DAG, the job is on the serious path and it begins on the processor in the initial time. The algorithm is used in the mean value method on the different CPU resource environment. In a policy-based grid computing, the static and mean value-based scheduling does not provide a better scheduling result.

F. Optimal Assignment with Sequential Search [19]

Based on the *A** technique the author introduces two more algorithms, they are:

- sequential algorithm,
- assignment algorithm.

The sequential algorithm decreases the search space. The assignment algorithm offers a lower time complexity, by running the assignment algorithm in parallel and attains considerable speedup.

The modified algorithm generates random solution and shortens the tree, when the exhaustive and sequential search for the optimal assignment is not possible to a large tree search space.

The recommended algorithm executes the optimal assignment in a heterogeneous method.

On the other hand, we use a sub tree and iterative methods for whole tree and various resources.

III. Proposed Iterative Scheduling Algorithm

Optimal scheduling algorithm is used in the dynamic programming to improve the utility function of the entire system. The proposed algorithm is iteratively get improved task agents and resource agent's effectiveness functions as sub-problems of the Qos for grid resource scheduling optimization. To satisfy the user requests, the created grid QoS scheduling algorithm gets a multiple quality of service solution optimal for grid users. When the task agent in each cycle evaluates separately, to change its optimal payment for grid resource agents regulate its computation resource demand and network resources demand and notifies the grid. The grid prices are updated and converse the new prices to the grid task, when the new computation resource and network resource demand are observed by the computation resource agent and network resource agent respectively and the same cycle is repeated.

A. Iterative Modulo Scheduling and GridWay

To get a good DAG completion time and to improve a resource allocation decision, the iterative modulo scheduling method is used in the algorithm.

To examine a number of iterative algorithms and priority functions, the near-best in schedule quality and near-best in computational complexity are verified in the extensions of the acyclic list scheduling algorithm and the commonly used height-based priority function.

The iterative algorithm and the intuition underlying the choice in heuristics are required and it is explained in [20]. Two pseudo-operations are implicit. In dependence graph, the START and STOP are added. All the other operations in the graph, START and STOP are done to be the predecessor and successor, correspondingly.

Iterative schedule calls in the Procedure Modulo Schedule which in turn larger values of II, to initiate with an earlier value is equal to the MII up to the loop has been scheduled.

Iterative Schedule appears the most conservative acyclic list scheduling algorithm. The various points are shown as below:

The detailed operation can be unplanned the schedule and again plan the schedule, operation scheduling, quite than instruction scheduling, is utilized. An operation is going to start, when the acyclic list scheduling notion and it is scheduled only after its predecessors is scheduled, the minimum value in iterative modulo scheduling whereas it is probable for a predecessor operation to be unscheduled after its successor has been scheduled:

- The function Highest Priority Operation returns the unscheduled operation is the highest priority in agreement with the priority system in use. It may return the same operation multiple times if that operation has been unscheduled in temporarily. It is not occur in the acyclic list scheduling.
- The calculation of Estart, the earliest start time for an operation is restricted by its dependences on its predecessors and it is affected when the operations can be unscheduled. In one or more of the predecessors is no longer scheduled, when an operation is selected and scheduled next. Likewise, when scheduling the first operation in a SCC, at least one of its predecessors is should not schedule.
- Observance of the modulo constraint is assisted by the use of a special version of the schedule reservation table [21]. To schedule an operation in a particular time occupies the use of resource R at time T, and then the location $((T \bmod II), R)$ of the table is used to record it. As a result, the schedule reservation table require only be as long as the II. Consequently, in a reservation table, a modulo reservation table (MRT) [22] is named.
- While resource reservations are made on a MRT, differences in time T indicate the difference at all times $T \pm k * II$. So, it is enough to think about an adjacent set of candidate times that duration in an interval of II time slots. Then the MaxTime is considered as the largest time slot, is set in to $MinTime + II - 1$, whereas in acyclic list scheduling is effectively set to infinity.
- The currently listed operation is selected by the FindTimeSlot. Suppose the MaxTime is vast, it will be an acyclic scheduling, the functioning of FindTimeSlot is as list scheduling; the while-loop always exits to found a legal. Conflict-free time slot, as the MRT is used with modulo scheduling, the MaxTime is as $(MinTime + II - 1)$. It is reason for the while-loop to end without found any conflict-free time slot. Lacking in unscheduling which leads to one or more operations is not feasible to schedule the current operation.

B. The function of Iterative Modulo Scheduling

```
Function Iterative Schedule (II, Budget: integer): boolean;
{Budget is the maximum number of operations scheduled}
{before giving up and trying a larger initiation }
{interval. II is the current value of the initiation }
{interval for which modulo scheduling is being }
{attempted.
}
var
Operation, Estart: integer;
MinTime,MaxTime ,TimeSlot: integer;
Begin
{compute height-based priorities }
HeightR;
{schedule START operation at time 0 }
Schedule(START, 0 );
Budget: = Budget-1;
{Mark all other operations as }
```

```
{having never been scheduled }
For operation:= 2 to NumberOfOperations do
Neverscheduled[Operation]:= true;
{ continue iterative scheduling until either all }
{ operation have been scheduled, or the budget is }
{ exhausted. }
While(the list of unscheduled operations is not empty)
And (Budget >0) do
Begin
{ Pick the highest priority operation }
{ from the prioritized list }
Operation : = HighestPriorityOperation();
{ Estart is the earliest start time for }
{ Operation as constrained by currently }
{ scheduled predecessors }
Estart: = CalculateEarlyStart (Operation);
MinTime := Estart;
MaxTime :=MinTime + II -1;
{ Select time at which Operation }
{ is to be scheduled }
TimeSlot : = FindTimeSlot (Operation,MinTime,MaxTime);
        { The procedure Schedule schedules Operation at }
        { time TimeSlot. In so doing, it displaces all }
        { previously scheduled nodes that conflict with }
        { it either due to resource conflicts or }
        { dependence constraints. It also sets }
        { NeverScheduled[Operation] equal to false. }
Scheduled(Operation,TimeSlot);
Budget := Budget – 1;
End; { while}
IterativeSchedule := (the list of unscheduled operations is empty);
End;{ IterativeSchedule }
```

III.1. Computation of the Scheduling Priority

In acyclic list scheduling, there are a boundless number of priority tasks are developed for modulo scheduling.

Mostly used one is getting the priority, in a single or many way, the operations are return to the circuit over that are not [23], [22], [24]. It is reflecting to schedule such operations are more difficult, while the first one scheduled in a SCC is subject to a deadline. Alternatively, a priority function is used to a direct conservatory of the height-based priority [25], [26] that is popular in acyclic list scheduling [27].

For expanding the height-based priority function is used in iterative modulo scheduling needs to take into account inter-iteration dependences. To consider a successor Q of operation P with a dependence edge from P to Q having a distance of D. Imagine the operation Q that is in the same iteration as P has a height based priority of H. Then the P's successor Q is actually D iterations later, and the STOP pseudo-operation D iterations later is II*D cycles later than the STOP pseudo-operation is in the same iteration.

The priority function used in iterative modulo scheduling, HeightR ( ), is attained by resolving the system of implicit equations in Fig. 5(a).

HeightR (P) is directly available as $MinDist[P.STOP]$, when the MinDist matrix to complete dependence graph is evaluated. The above implicit set of equations for HeightR () is iteratively solve by using a less costly procedure. For identifying the SCCS of a graph during a depth-first traversal of the graph [28] is utilized based on the algorithm. HeightR () has a couple of good properties are described in this algorithm somewhere [33]. In their structure, a large fraction of the loops are rather simple. By using the topological sort order, HeightR () is ensures whether the operations are scheduling in one pass. This is a better chance for some loops. Second, HeightR() provides higher priority to operations in some SCCs which have less slack.

The Function FindTimeSlot:

```
Function FindTimeSlot (Operation. MinTime, MaxTime : integer): integer;
  var
   CurrTime,SchedSlot : integer;
  Begin
    CurrTime : = MinTime;
    SchedSlot := null;
    While (SchedSlot = null) and (CurrTime <= MaxTime) do
        If ResourceConflict (Operation,CurrTime) then
        { There is a resource conflict at }
        { CurrTime. Try the next time slot. }
          CurrTime : = CurrTime + 1;
   Else
    { There is no resource conflict at CurrTime }
    { Select this time slot. Note that dependence }
    { conflicts with successor operations are }
    { ignored. Dependence constraints due to }
    { predecessor operations were honored in }
    { the computation of MinTime. }
      SchedSlot :=CurrTime;
  { If a legal slot was not found, then pick (in }
  { decreasing order of priority ) the first available }
  { option from the following : }
  { }
  { -MinTime,either if this is the first time that }
  { operation is being scheduled, or if MinTime is }
  { greater than PrevScheduleTime[Operation], (where}
  { PrevScheduleTime[Operation] is the time at which
```

```
}
    {operation was last scheduled)
}
    {-PrevScheduleTime[operation]+1
}
            If SchedSlot = null then
               If  (neverScheduled [operation] or
                (MinTime > PrevScheduleTime[Operation]
then
                        SchedSlot := MinTime
   Else
    SchedSlot := PrevScheduleTime[operation]+1;
   FindTimeSlot:=SchedSlot;
   End;{ FindTimeSlot}
```

III.2. Calculation of the Range of Candidate Time Slots

The MRT requires accurate schedules from a source usage viewpoint. Suitably, from the viewpoint of dependence constraints are required by antecedents, is taken by computing and using Estart, the most early time the operation is scheduled, if respect the dependences on its predecessors. In this context of reappearance and iterative modulo scheduling, it is not possible to assure that all of an operations, predecessors have been scheduled, and remaining are scheduled, when the time comes to schedule the operation in question. The Estart is calculated, when consider these immediate predecessors are presently scheduled. The early start time for operation P is provide the equation where Pred(P) is to set the immediate predecessors of P and SchedTime(Q) is the time to Q is scheduled.

An operation is not scheduled by its Estart are privileged to dependences with predecessor operations. Dependences with successors operations are credited by virtue of the fact, when an operation is scheduled; all operations are conflict with it, either because of resource usage or due to dependence conflicts, are unscheduled. Consequently the operations are scheduled, the Estart is computed by them, and the dependence restraint is monitored. In any time, the incomplete schedule for the presently scheduled operations entirely credits all the constraints between the scheduled operations.

Considering more than II adjacent time slots are starting with Estart, is meaningless and unwanted. Because of resource conflicts, a legal time slot is not found in this range, it is also not found in outside of the range. Therefore, MaxTime is set equal to Estart +II – l.

III.3. Selection of Operations to be Unscheduled

Consider the time slot is created among MinTime and MaXTime, it does not result in a resource variance with any currently scheduled operation [29]. The operations which are unscheduled, those direct successors with whom there is a dependence conflict. The operations are not unscheduled, because of a resource conflict.

At the same time, in every time slot for MinTime to MaxTime results in a resource is conflict. The two decisions are formed which is given below

1. Should select a time slot in which to schedule the current operation.
2. Should select in which currently scheduled operations are replace from the schedule.

The first decision is made to make sure that the forward growth by this event the current operation is previously scheduled; it can't be rescheduled at the same time. It prevents an environment where two operations to keep continuously replace each other from the schedule.

The operation is scheduled, when the Estart is less than the previous schedule time. If Estart is greater than the previous schedule time, it is scheduled one cycle later than it was scheduled previously.

Inspite of, to schedule the operation in the particular time slot, one or more operations are unscheduled for the reason of resource conflicts. When the multiple alternatives for scheduling an operation, the choice of alternative decides to which operations are unscheduled.

Periodically, we choose an alternative which replaces the lowest priority operations. In place of attempting to make this examination straightly, all the operations are unscheduled which conflict with the use of any alternatives.

By using one of the alternatives the current operation is scheduled. Rescheduled the replaced operations, at the same time, may be the priority function order is specified.

To resolve the scheduling problem modeled in BIP, the proposed algorithm prepares an optimal scheduling decision. The proposed algorithm is used to the mean value method to make an initial scheduling decision on different. The implementation time of a job is modified with a specific value on a decided processor as the iterative modulo scheduling continues. When there is no enhancement in dag completion time, the iteration is finished. The above mentioned algorithm is described as detailed. The algorithm develops an iterative modulo scheduling scheme to treat with different resources. By solving the policy-based scheduling problem, that is modeled in BIP is also executes an optimized scheduling.

GridWay

GridWay is one of the open source meta-scheduling knowledge that provides large-scale, protected, reliable and well-organized sharing of computing resources (clusters, computing farms, servers, supercomputers...), managed by dissimilar Distributed Resource Management Systems (DRMS), such as SGE, Condor, PBS or LSF, within a single organization or scattered across several administrative domains. At this end, GridWay maintains several Grid middleware.

IV. Notation and Variable Definition

The proposed scheduling algorithm the notations and variables that are referred to be defined are shown in this

section.

In different resource situation, the computation or execution time of a job on a processor (com_{pij}) is not equal along with a set of processors. The algorithm puts an initial execution time of a job with the mean value of the different time on a set of accessible processors. Based on the total workflow completion time, the algorithm modifies the scheduling decision; the time is renewed with an execution time on a specific processor. Between any two processors, the data transfer or communication time are also various in the environment:

com_{pij}: computation time of job i on processor j
$comm_{pj}$: communication time from processor p to j

An application or workflow is in the format of DAG.

Each job i has a set of precedent ($prec_i$) and succeeding ($comm_i$) jobs in a DAG. The dependency is represented by the input/output file relationship:

$prec_i$: a set of the precedent jobs of job i

$succ_i$: a set of the succeeding jobs of job i

To execute a job maintains track of the availability of a processor ($Availij$) by this algorithm. Before a new job gets started, a processing model in a way that all the jobs in a processor queue should be completed are considered. consider a nonpreemptive model.

GridWay attains the information from a grid monitoring system such as MonALISA or GEMS, in the grid scheduling middleware. the earliest start time of a job on each processor ($ESTij$) is work outs by this algorithm. A job can establish its execution on a processor only after assuring the two conditions; first one is a processor must be obtainable ($Availij$) to execute the job. Second, all the precedent jobs be finished:

$$(Max_{k\in prec_i}\{EFT_{kp} + comm_{pj}\})$$

On the same processor or the others. The earliest start time ($ESTij$) and the job completion or execution time on the processor ($compij$) is defined by the earliest finish time of a job on a processor ($EFTij$). The workflow completion time from a job to the end of a DAG ($compLeni$) is defined recursively from the bottom to the job *i*. The significant path in the workflow is decided by value. To make width of the significant path is used as a reason to finish the algorithm. The algorithm ends the scheduling when there is not a series of enhancement in the DAG completion time:

$Avail_{ij}$: the available time of processor j for job i

EST_{ij}: earliest start time of job i on processor j

$$EST_{ij} = MAX\ \{Avail_{ij}, Max_{k\in prec_i}\{EFT_{kp} + comm_{pj}\}\}$$

EFT_{ij}: earliest finish time of job i on processor j

$$EFT_{ij} = EST_{ij} + comp_{pj}$$

$compLen_i$: workflow completion length from job i

$$compLen_i = comp_{iP_i} + Max_{k\in succ_i}\begin{pmatrix} comm_{p_ip_k} + \\ +compLen_k \end{pmatrix}$$

V. Optimization Model

In the proposed algorithm, to find an optimal solution to the scheduling problem must create a BIP model.

Here, we describe a scheduling profit function, which uses the workflow completion time and the initial end time of a job on an each different processor. Then discuss about the optimization model for the scheduling problem.

Scheduling profit (p_{ij}): the profit when job (i) is assigned to processor (j):

$$p_{ij} = \frac{compLen_i}{EFT_{ij}}, \text{where } EFT_{ij} > 0$$

when a job on a considerable path is scheduled with another job on a non considerable path, the profit function to be described to generate higher profit. The job ends soon than the other processor, the profit value is also higher with a processor for a job *i*. A scheduling algorithm providing the function tries to give a higher priority to the job whose completion time is longer than the others' completion time. Which job has been completed earlier than others the algorithm allocates the job with higher priority to the processor.

Fig. 1 is based on the scheduling function (P_{ij}), Table I shows the example that gives a procedure to prioritize a set of jobs on the DAG. Based on the earliest finish time of a job on a processor(EFT_{ij}), also shows the processor assignment to the jobs. An example of a workflow in DAG and a process to significant a set of jobs on the DAG and to allocate the jobs onto a set of processors, respectively are shown in Fig. 1 and Table I. Considering two processors, P1 and P2 is shown in Table I, illustrates the different execution time of the jobs on each of the processors (execution time). The execution policy of the jobs are also shown in the table. The processor permits a job to be executed on a corresponding processor is indicated by the mark *. On the basis of the average execution time (avg.exec. time) of the job over the processors, the workflow completion time from a job to the end of a DAG (*compLen*) is calculated [30]. The jobs are signified and allotted onto the processors.

The main goal of the optimization model uses the profit function to choose a processor for a job from the scheduled job list. The scheduling problem in the optimization model is resolved by the scheduling algorithm. It tries to allot the jobs in a significant path onto the processors, which gives the job with the earliest end time. The task is restricted by a set of constraints.

By using the BIP optimization model gives the higher profit values to a set of jobs and processors subject to the

several restrictions. The resource usage constraint is detailed with the two values quota (qij) and necessity (bij) for a job i and a processor j. The quantitative model is easily to convey the procedure in different resource types.

The assignment constraint creates, it sure that a job is not divided or assigned onto more than two processors.

In every processor, asset of assigned job above the predefined quota (tj) should not be loaded. Now the load is defined with the number of assigned jobs. By using the available BIP solvers, the BIP model $(xij = 0\ or\ 1)$ is implemented. We use the GNU linear programming kit in the implementation:

$$Max \sum_i \sum_j p_{ij}.x_{ij}$$

s.t.

$bijxij \leq qij$ for each job i and processor j (policy)

$\sum_j x_{ij} = 1$for each job i (assignment)

$\sum_i x_{ij} \leq t_j$for each processor j (load)

$xij = 0\ or\ 1$ (binary)

where:

bij : resource usage requirement of job i on processor j;

qij : resource usage quota of job i on processor j;

tj : the limit of assigned jobs on processor j.

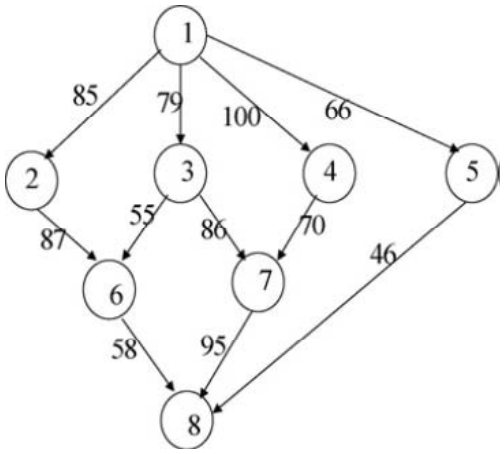


Fig. 1. Example workflow in DAG

TABLE I
EXAMPLE FOR JOB PRIORITIZATION AND PROCESSOR ASSIGNMENT

JOB#	J1	J2	J3	J4	J5	J6	J7	J8
Execution time								
P1	62	60	75	84	25	60	19	90
P2	80	45	90	20	80	82	15	30
policy								
P1	*	*	*			*		*
P2	*		*	*	*		*	
Avg.Exec Time	71	52.5	82.5	52	52.5	71	17	60
compLen	500	317	325	315	180	185	170	60
Prioritization	1	3	2	4	6	5	7	8
Assignment	P1	P2	P1	P1	P1	P2	P1	P1

VI. Experiment and Simulation Results

The performance is calculate with the simulation results by using the projected iterative modulo scheduling algorithm and the test application is executed on the OSG.

Here we discussed about the simulated and experimental performance to evaluate the algorithm with the list scheduling which uses the mean value method to the different resource atmosphere.

A. Network Configuration and Test Application

The scientific computing support OSG is a grid-computing infrastructure. It consists of more than 25 sites, and together provides more than 2000 CPUs. Seven different scientific applications, including three high energy physics simulations and four data analyses in high-energy physics, biochemistry, astrophysics, and astronomy are used by the resources.

Consequently, the performances of the algorithms are compared, to generate a set of test workflows in a DAG format. The Workflow reproduces a simple application that gets input files, and generates an output file. In each job the size of the output file is differed and by default, in the execution site the file is placed. DAG structure is as the depth and width is set with different values are arranging to the experiment and simulation parameters, are discussed in the following sections.

a. List Scheduling with the Mean Value Approach

By the performance of DAG completion time, the list-scheduling algorithm with the iterative modulo scheduling algorithm and policy scheduling algorithm is compared in this experiment. To decide the execution time of a job on different resources the list scheduling uses the mean value approach.

A set of jobs in a workflow is sorted by the list-scheduling algorithm by using the non descending order of the workflow completion time. For each job, up to the end of the workflow, the workflow completion time is calculated. On the same time, the workflow completion time is affected in a significant manner, when the algorithm gives the high scheduling priority to the job.

One by one the job is sorted by using the algorithm and allots a job onto the processor on which the job can conclude as soon as feasible. The assignments are hold back by the resource usage policies.

Sometimes, the mean value for the job execution time is not reflect the actual job execution on the different resources. It shows the result in a non-optimal scheduling for a workflow in the list-scheduling method.

An unreasonable scheduling decision, the policy constraint might drive the resource assignment in the list scheduling. The main reason is to schedule the jobs in the sorted list one by one, it does not have a chance to think about the policy constraints between multiple jobs. The method shows the result in a long DAG completion time to allocate a job to a false processor in terms of the earliest finish time of a job $(EFTij)$.

b. Simulated Performance Evaluation

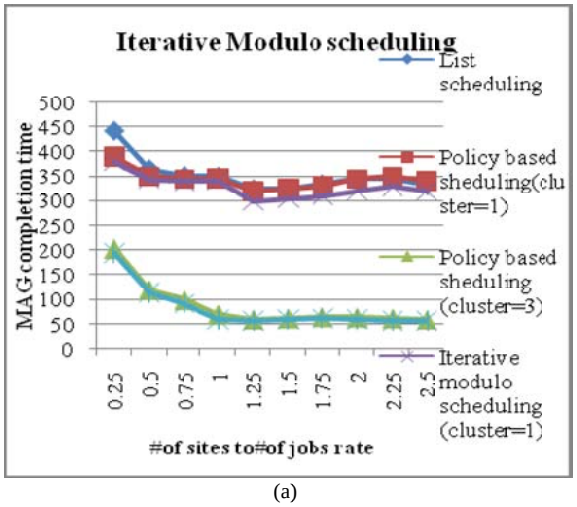
When the ratio of the processors to the jobs is different, the first model provides the performance of the algorithms in terms of DAG completion time and scheduling improvement.

In graph Figs. 2 show the performances when the ratio is changed from 0.25 to 2.5 increased by 0.25. In the first graph, the iterative modulo scheduling gives the best list scheduling by 68% to 82% the longest completion time of a job in the DAG gives the best DAG completion time. The scheduling improvement signifies the ratio of the scheduling time reduction from the initial scheduling time.

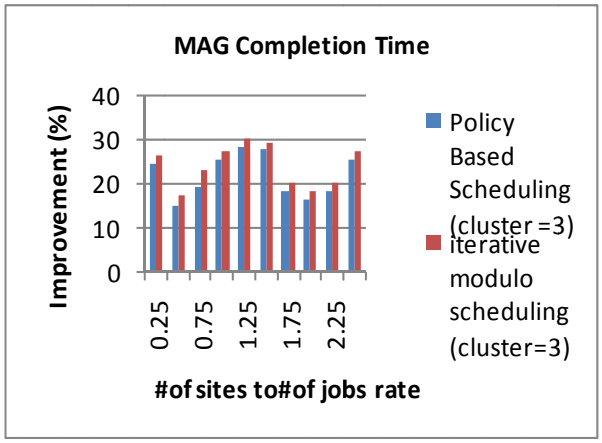
It is defined with:

$$\left(\frac{(Init\_Sched\_Time \; - \; Best\_Sched\_time\;)}{Init\_Sched\_time}\right) \times 100$$

When the iterative algorithm constraint is different, the performances of the algorithms are evaluated by the next simulation. The constraint describes the ratios of the available processors to the total processors.



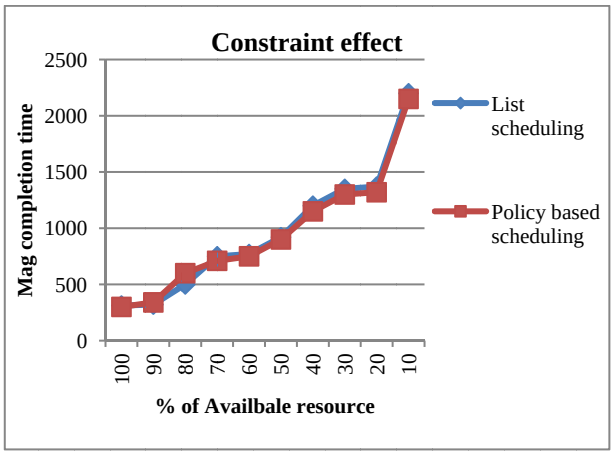
(a)



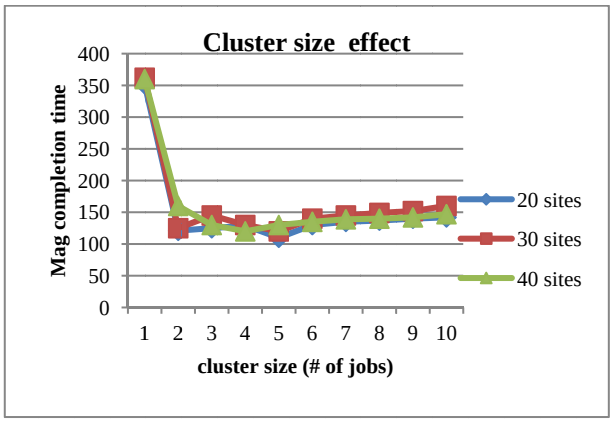
(b)

Figs. 2. DAG completion time when the rate of processors to job is different

The maximum job execution in the iterative constraint is only available on the set of processors. The DAG completion time is shown in the Figs. 3 in the first graph, when the policy constraint modifies from 100% to 10% decreased by 10%.



(a)



(b)

Figs. 3. Constraint and clustering effect DAG completion time

A job is able to run on all the processors, when the available resource indicates as 100%, whereas the 10% of processors are available to run a job means it indicates as 10%. The policy-based scheduling algorithm illustrates the good acceptance to the constraints.

The better constraint performance is given by iterative scheduling algorithm compared than policy-based scheduling. When the available processor ratio from 100% to 60%, the DAG completion time is stable.

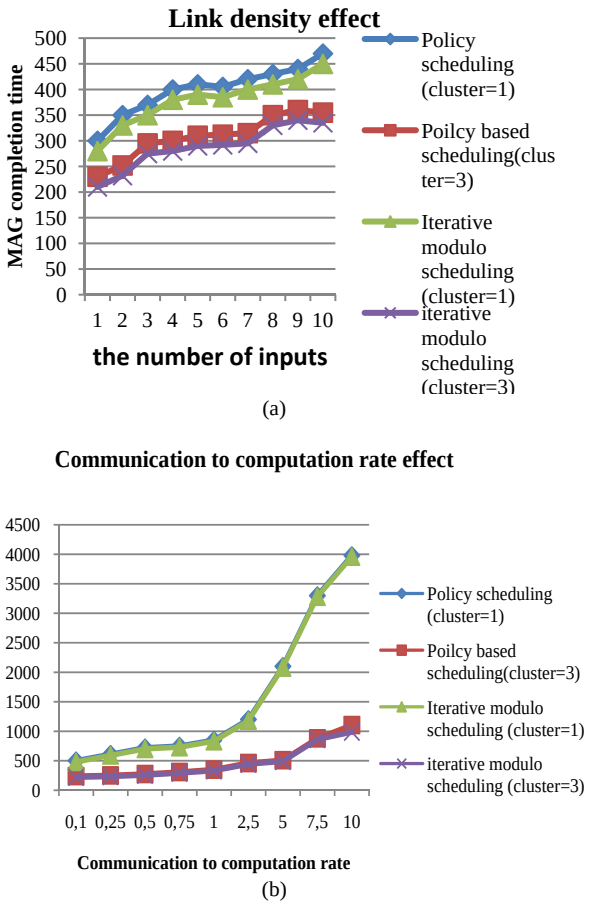
The scheduling time is shown in the second graph in Figs. 3 when the cluster size is different. In the optimization scheduling, the number of jobs are scheduled together is called the cluster size. The performance with the different size of the cluster is shown by the policy-based scheduling algorithm but when it is compared with the iterative scheduling algorithm the scheduling presents the constant performance. The scheduling is performed per job is done by its size. It means, in the optimization procedure, the scheduling not able to make use of the policy

constraints of multiple jobs. The scheduling algorithm considers the constraints of multiple jobs, when the cluster size is more than two and low DAG completion time is provided by the better scheduling decision. The simulation illustrates the performance of the algorithms with different types of workflows. The workflow types are described with two features. One is the communication to computation rate (CCR). By using the CCR the communication time is described by:

$$Communication\ Time = \frac{(Computation\ Time \times CCR)}{Communication\ Rate}$$

By using this experiment, the computation time and communication rate are selected randomly with the subsequent range. Communication rate is 1, 2, 3 and 4 and the computation time is from 10 to 100 increased by 10. The constant performance with different cluster size is described in the iterative modulo scheduling.

That means, in different the different types of workflows such as the communication oriented and the computation-oriented, the performance of the scheduling algorithm is constant. When the link density between jobs is different, the performances of the algorithms are shown in the second graph in Figs. 4.



Figs. 4. Algorithm sensitivity: CCR effect and link density effect

The link density is defined by the number of inputs of each job. The number of outputs is set to one in this experiment. The DAG completion time is increased, because of the number of inputs increases. The main reason is the job gets larger time to be ready to run on a processor with multiple inputs than with a small number of inputs.

c. Performance Evaluation on OSG

The performance evaluation on OSG is performed by the performance simulation method. The performance of iterative modulo scheduling algorithm, policy-based scheduling algorithm, and it compares with the list-scheduling algorithm is shows in this experiment. Also, a resource stands for a grid site on OSG. Especially, the experiment is used to the CPU resource to run a set of workflow.

As a result, a resource shows the CPU resource in a grid site in this section. DAG completion time when the ratio of available resources to the total resources in OSG is different is shown in the first graph in Figs. 5. In the proposed algorithm the resource usage policy is used to choose the scheduling based on the accessibility of the resource.

A job is allowed to run the policy constraint to identify a set of available processors and the available set of processors maximizes the job execution. The list scheduling by 40% to 47% is executed by the iterative modulo scheduling. There are two ways of the out-performance.

The iterative scheduling which approximates the impact of the current decision to the DAG completion is explained in the first one and to modify the scheduling to improve the completion time.

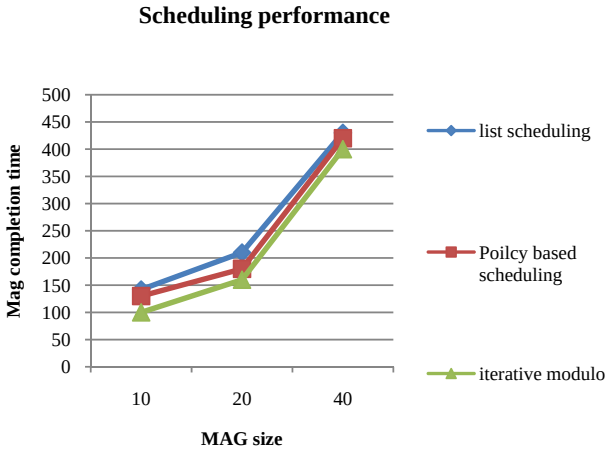
The list scheduling is not working in this way, it also creates the scheduling decision of each execution is ready job in only one time and the target resource is present.

The optimization with a cluster of jobs is shown in the second reason. As an alternative, to make the scheduling decision of execution-ready jobs in one by one fashion (list scheduling), schedule by clustering the jobs are executed in the proposed algorithm and the scheduling problem is solved in BIP format.

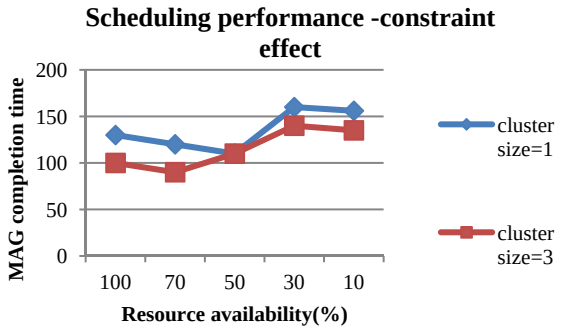
When the size of DAG is different with 10, 20, and 40 jobs, the performance of the algorithms are illustrated in the second graph in Figs. 5. By this experiment, the policy constraint in the resource availability is 60%.

The list scheduling algorithm and policy based scheduling algorithm is compared with iterative modulo scheduling algorithm. According to the number of jobs scheduled in a DAG varies 35 % to 70% is executed in the proposed algorithm. When the resource availability is different, the average job execution time is shown in Fig. 6. The availability is getting low and the execution time is increased, because of the limited amount of resources is able to execute jobs. Compared with the policy scheduling algorithm and list scheduling algorithm, the proposed iterative modulo scheduling algorithm gives the better because the limited amount of resources is able to

execute jobs, as the availability is getting low the execution time is increased.



(a)



(b)

Figs. 5. Test DAG completion time on OSG with various constraints and DAG size

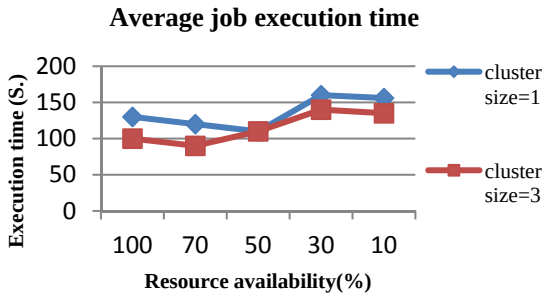


Fig. 6. Average job execution time on OSG with different policy constraints

VII. Conclusion

In this paper a novel Iterative modulo scheduling algorithm is proposed. By using the resource usage policies under the constraints are presented, grid resources are allocated to an application. By using iterative method and BIP, it executes optimized scheduling on different resource. The completion time of an application is integrated with job execution tracking and history modules of GRIDWAY scheduling middleware are developed in this algorithm.

Now able to make resource allocation satisfying QoS from multiple jobs are expand in this algorithm, an optimal scheduling decision subject to the policy constraint and QOS is developed in this algorithm. It possible to schedule jobs onto OSG according to the constraints are made by GRIDWAY is implemented in the algorithm. A set of practical application from high-energy physics experiments such as CMS based on the scheduling decision is scheduled, and to study the performance execution.

References

[1] Abramson D, Giddy J, Kotler L. High performance parametric modeling with Nimrod/G: Killer application for the Global Grid? Proceedings International Parallel and Distributed Processing Symposium (IPDPS 2000), Cancun, Mexico, 1–5 May 2000. IEEE Computer Society Press: Los Alamitos, CA, 2000.

[2] Buyya R, Abramson D, Giddy J. A case for economy Grid architecture for service-oriented Grid computing. Proceedings of the International Parallel and Distributed Processing Symposium: 10th IEEE International Heterogeneous Computing Workshop (HCW 2001), 23 April 2001, San Francisco, CA. IEEE Computer Society Press: Los Alamitos, CA, 2001.

[3] Fanxin Kong, Hongyan Hao, Jianmin Zuo, Classification and Dynamic Fuzzy Clustering of Mold Resources Based on Mold Manufacturing Grid Platform, (2012) *International Review on Computers and Software (IRECOS)*, 7 (5), pp. 2447-2452.

[4] Ferguson D, Nikolaou C, Sairamesh J, Yemini Y. Economic models for allocating resources in computer systems. Market-Based Control: A Paradigm for Distributed Resource Allocation. World Scientific Press: Singapore, 1996.

[5] Chapin S, Karpovich J, Grimshaw A. The Legion resource management system. Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, San Juan, Puerto Rico, 16 April 1999. Springer: Berlin, 1999.

[6] Litzkow M, Livny M, Mutka M. Condor—a hunter of idle workstations. Proceedings 8th International Conference of Distributed Computing Systems (ICDCS 1988), San Jose, CA, January 1988. IEEE Computer Society Press: Los Alamitos, CA, 1988.

[7] Berman F, Wolski R. The AppLeS Project: A status report. Proceedings of the 8th NEC Research Symposium, Berlin,Germany, May 1997.

[8] Casanova H, Dongarra J. NetSolve: A network server for solving computational science problems. International Journal of Supercomputing Applications and High Performance Computing 1997; 11(3):212–223.

[9] Kapadia N, Fortes J. PUNCH: An architecture for Web-enabled wide-area network-computing. Cluster Computing:

[10] The Journal of Networks, Software Tools and Applications 1999; 2(2):153–164.

[11] S. Verboven, P. Hellinckx, F. Arickx, and J. Broeckhove, “Runtime prediction based grid scheduling of parameter sweep jobs,” J. Internet Technol., vol. 11, no. 1, pp. 47–53, 2010.

[12] C.-H. Hsu and S.-C. Chen, “A two-level scheduling strategy for optimizing communications of data parallel programs in clusters,” Int. J. Ad Hoc Ubiquitous Comput., vol. 6, no. 4, pp. 263–269, 2010.

[13] C.-H. Hsu and T.-L. Chen, “Performance and economization-oriented scheduling techniques for managing applications with QoS demands in grids,” Int. J. Ad Hoc Ubiquitous Comput., vol. 5, no. 4, pp. 219–226, 2010.

[14] G. Q. Liu, K. L. Poh, and M. Xie, “Iterative list scheduling for heterogeneous computing,” J. Parallel Distrib. Comput., vol. 65, no. 5, pp. 654–665, 2005.

[15] Y. K. Kwok and I. Ahmad, “Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors,”

IEEE Trans. Parallel Distrib. Syst., vol. 7, no. 5, pp. 506–521, May 1996.
[16] X. Qin and H. Jiang, “Reliability-driven scheduling for real-time tasks with precedence constraints in heterogeneous distributed systems,” in Proc. Parallel Distrib. Comput. Syst. Conf., Nov. 2000, pp. 617–623.
[17] H. Zhao and R. Sakellariou, “An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm,” in Proc. Int. Conf. Euro-Par, LNCS 2790. 2003, pp. 189–194.
[18] G. C. Sih and E. A. Lee, “Dynamic-level scheduling for heterogeneous processor networks,” in Proc. IEEE Int. Conf. Parallel Distrib. Syst.Process., Dec. 1990, pp. 42–49.
[19] M. Kafil and I. Ahmad, “Optimal task assignment in heterogeneous distributed computing systems,” IEEE Concurrency, vol. 6, no. 3, pp. 42–50, Jul.–Sep. 1998.
[20] Rau,B.R. Iterative Modulo scheduling HPL technical report Hewlett-Packard Laboratories,1994.
[21] Rau B R and Glaeser C D some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing. In proc Fourteenth annual workshop on microprogramming,(October 1981) 183-198.
[22] Lam.M Software pipelining : an effective scheduling technique for VLIW machines. In proc ACM SIGPLAN ’91 conference on programming language design and implementation (june 1991) 219-228.
[23] Hsu P.Y.T Highly concurrent scalar processing PhD thesis university of Illinois Urbana-Champaign 1986
[24] Dehnert, J.C., and Towle,R.A Compiling for the Cydra 5. The Journal of Supercomputing 7,1/2(May 1993),181-228.
[25] Hu.T.C Parallel sequencing and assembly line problems. Operations Research 9, 6 (1961), 841-848.
[26] Ramamoorythy,C.V., Chandy,K.M and Gonzalez, M.J Optimal scheduling strategies in a multiprocessor system. IEEE transactions on computers C-21.2 (February 1972), 137-146.
[27] B.Ramakrishna Rao, “iterative modulo scheduling: an algorithm for software pipelining loops” Hewlett-Packard Laboratories, 1501 Page Mill Road, Bldg.3L, Palo Alto, CA 94304.
[28] Adam,T.L., Chandy,K.M and Dickson J.R A comparison of list schedules for parallel processing systems . Communication of the ACM 17, 12(December 1974) 685-690.
[29] T. Guesmi, S. Hasnaoui, H. Rezig, Network Priority Mapping Using Dynamic RT-CORBA Scheduling Service, (2006) *International Review on Computers and Software (IRECOS)*, 1 (2), pp. 124-131.
[30] E. Ilavarasan, P. Thambidurai, Genetic Algorithm for Task Scheduling on Distributed Heterogeneous Computing System, (2006) *International Review on Computers and Software (IRECOS)*, 1 (3), pp. 233-242.

Authors’ information

[1]Assistant Professor, P. A. College of Engineering and Technology.
E-mail: pacet.saravanan@gmail.com

[2]Associate Professor, Government College of Technology
E-mail: gopalgct@hotmail.com

G. Saravanan received his M.E. degree in Computer Science and Engineering from Government College of Technology, Coimbatore, in 2008. Currently he is working as an Assistant Professor in P. A. College of Engineering and Technology, Coimbatore. His research areas are on Grid Computing system, Scheduling algorithms and simulation model.

Dr. **V. Gopalakrishanan** received his Ph.D degree in Computer Science and Engineering from Anna University Chennai at 2009. Currently he is working as an Associate professor in Government College of Technology, Coimbatore. His research areas are on Networking and power systems.