2010

Resource Characteristic Based Optimization for Grid Scheduling

Peng Du
University of Windsor

Resource Characteristic Based Optimization for Grid Scheduling

by

Peng Du

A Thesis

Submitted to the Faculty of Graduate Studies

through the Computer Science

in Partial Fulfillment of the Requirements for

the Degree of Master of Applied Science at the

University of Windsor

Windsor, Ontario, Canada

2010

RESOURCE CHARACTERISTIC BASED OPTIMIZATION FOR GRID SCHEDULING

by

Peng Du

APPROVED BY:

September 20, 2010

Author’s Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone’s copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Scheduling is an active research area in the Computational Grid environment. The objective of grid scheduling is to deliver both the Quality of Service (QoS) requirement of the grid users, as well as high utilization of the resources. To obtain optimal scheduling in the generalized grid environment is an NP-complete problem. A large number of researchers have presented heuristic algorithms to find a near-global optimum for the static scheduling model of the grid. Relatively a smaller number of researchers have worked on the scheduling problem for the dynamic scheduling model.

This thesis proposes a new resource characteristic based optimization method, which may be combined with Earlier Gap, Earliest Deadline First (EG-EDF) policy to schedule jobs in a dynamic environment. The proposed algorithm generates near-optimal solutions, which are better than those reported in the literature for a specific range of datasets. Extensive experimentation has proved the efficacy of our method.

DEDICATION

This thesis is dedicated to my parents for their endless support.

ACKNOWLEDGEMENTS

I wish to express my deep sense of gratitude to Dr. Akshaikumar Aggarwal and Dr. Robert D. Kent, my advisors, for their inspiring ideas, their patient guidance and enthusiasm they invested in this whole research. Their encouragement has assured the quality of work and light up the road of my study.

I would like to express my appreciation to Dr. Huapeng Wu, Dr. Asish Mukhopadhyay, and Dr. Jianguo Lu for being on my committee and offering me valuable advice and comments concerning this thesis.

I would also grateful to all of my friends who have given me useful suggestions and supports during the research.

Contents

List of Figures

List of Tables

1. Introduction

Grid is a distributed computing environment that connects resource providers to users [1]. Grid users submit their jobs into the system. The Grid system is aware of the resources available at any point of time. It uses scheduling algorithms for allocating resources to different applications, while guaranteeing to the users the Quality of Service (QoS) requirements. The resource providers expect the scheduling process to maximize the use of resources.

Quality of Service (QoS) from a user's perspective includes the most popular criteria of makespan. Makespan is the time in which a set of jobs can be completed by using a set of resources available on the grid. Other criteria are the number of missed deadlines and tardiness, where tardiness is the sum of delays from deadlines for all the jobs. From the resource provider's perspective, Resource Utilization must be maximized. It may not be possible to schedule the jobs on the available resources in such a way that all the different criteria can be optimized. Thus maximizing the resource utilization may not lead to minimization of makespan.

The problem of allocation of jobs to resources in such a way that all the criteria of interest are optimized is called the problem of optimizing the grid scheduling. Since the problem is known to be NP-complete [2], researchers have developed heuristic algorithms for obtaining near-optimal schedules. Optimizing grid scheduling for a static environment, where one has complete information about all the jobs is an easier problem than the problem of obtaining an optimum schedule for a dynamic environment, where jobs are arriving even as these are being scheduled.

This thesis proposes a new resource characteristic based optimization method, combined with the Earlier Gap, Earliest Deadline First (EG-EDF) policy, to schedule jobs in a dynamic grid environment. The model of the environment assumes that new jobs are arriving even as the jobs are being scheduled and allocated to resources. However the resources are assumed to form a static set.

The rest of this thesis is organized as follows. In section 2, we describe some related works. Section 3 states the model of the problem, which is being studied in this thesis. In section 4, we present our scheduling method. Our testing methodologies, as well as the experimental results are shown in section 5. In section 6, we analyze the test results. Section 7 presents the conclusions.

2. Related Works

Scheduling algorithms are applied in a distributed system to satisfy multi-objectives for both the users and the resource-providers. Scheduling problems can be classified into static and dynamic scheduling problems. In static scheduling problems, the resources and the tasks in each job are known to the scheduling system in advance. On the other hand, a dynamic scheduling system does not have full information about the resources and the jobs, when the execution starts. Additional jobs continue to arrive dynamically, even when some of the jobs have been scheduled and are getting executed [3]. Some local search based scheduling algorithms are computationally costly and are usually applied to static problems, while schedule based scheduling algorithms are mostly used in dynamic environments [4]. Most of the scheduling algorithms, used in today's grids, are queue based scheduling algorithms. Klusacek and Rudova [4] state that queue-based scheduling can handle single objectives. However, complex objectives such as deadlines, resource utilization, response time, flow time, or slow down are hard to achieve by queue-based solutions.

2.1 Queue Based Scheduling

Sun Grid Engine [5], Condor [6] together with Grid management system GridWay [7] are well-known systems that use queue based scheduling policies. All the scheduling progress is managed based on a single queue or multi-queues.

2.1.1 Sun Grid Engine

Sun Grid Engine (SGE) [5] is a Sun open source resource management software. It has been selected as the scheduler for the world's largest operating grid infrastructure: the

Enabling Grids for E-ScienceE (EGEE) [8]. When a job is submitted by the user, it is sent to the scheduler. The user requests certain execution features when they submit a job, and SGE would allocate that job to the queue of a system, which can provide the features [8]. Sun Grid Engine uses the notion of queues to distinguish between different types of jobs and different components of the cluster. Grid Engine queues can allow execution of many jobs concurrently, and Grid Engine tries to start new jobs in the cluster, that is most suitable and least loaded. Sun Grid Engine improves the average resource usage. It has been claimed that SGE was able to obtain a utilization of as much as 98% [5].

2.1.2 Condor-G

The Condor project [6] has been used since 1984. Condor-G agent is one of the products of the Condor project.

2.1.2.1 Condor in the Grid

The Grid Resource Access and Management (GRAM) protocol [9], designed by the Globus project [10], provides an abstraction for remote process queuing and execution. Condor-G represents the marriage of technologies from the Condor project and Globus project [6]. Figure 2-1shows the architecture of the Condor and Condor-G in the Grid environment.

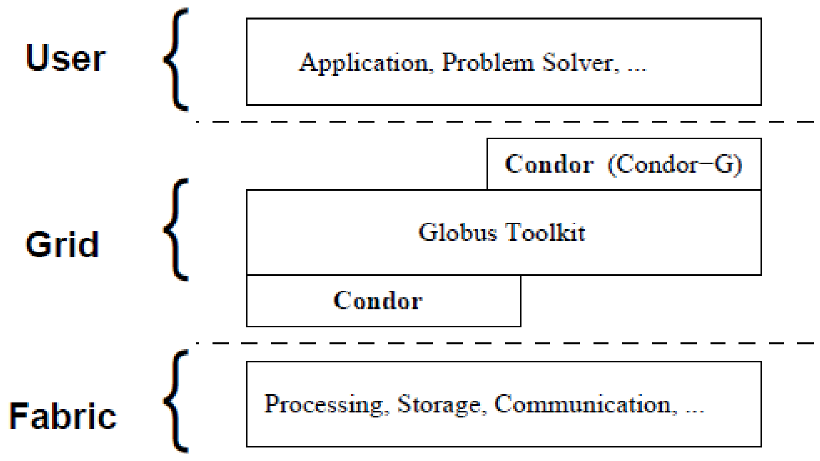
User
Application, Problem Solver, ...
Condor (Condor–G)
Grid
Globus Toolkit
Condor
Fabric
Processing, Storage, Communication, ...

Figure 2-1 Condor in the Grid (Figure 1 from [6])

Condor-G does not support all the features of GRAM, otherwise it would have become complex and unusable [6].

2.1.2.2 Kernel and Condor Pool

Condor-G can be used in Grids for providing both the reliable submission and job management service. The kernel of Condor-G performs the fundamental operations of scheduling. The whole process is shown below:

When a job is sent by a user to an agent, the agent stores the job information in the shadow.

1. Both the agent (A) and the resource (R) are in contact with the matchmaker (M).
2. The matchmaker matches the requirement of the job and the compatible resource.

3. The agent contacts the resource and validates the information about the resource, and the agent sends the job to the resource for execution.

Figure 2-2 shows the whole scheduling process. In the figure, the sandbox provides the resource to prevent mischief by other jobs. The process is executed in the Condor pool, shown in Figure 2-3.

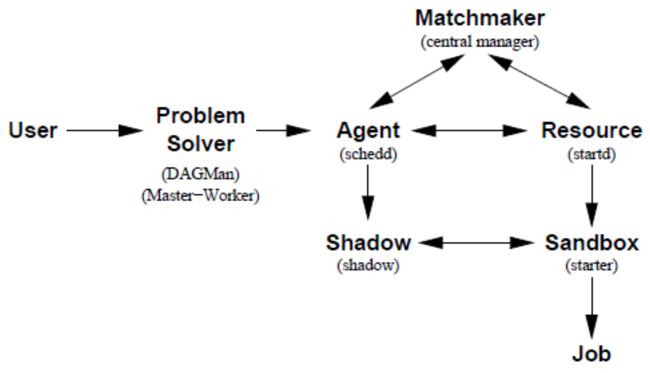

Figure 2-2 Scheduling Process (Figure 2 from [6])

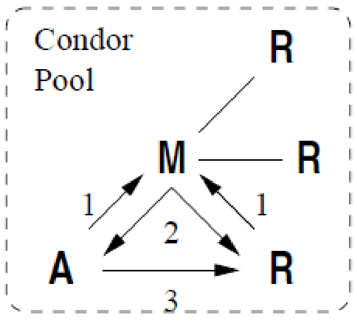

Figure 2-3 Condor Pool (Figure 3 from [6])

2.1.2.3 Gateway Flocking

Each condor pool may have a Gateway (G) for communication with other condor pools. The gateways may be connected through a network. If the local condor pool A does not have a resource, which matches the requirements of the job, the Matchmaker (M) of the condor pool A can go to a resource in another condor pool though the gateway of pool A. This process is called “Gateway Flocking”. Figure 2-4 shows the case of two condor pools A and B. The matchmaker in pool A finds that none of the resources in pool A matches the requirements of the job, submitted by agent A. So the matchmaker sends the job requirements out through the gateway. The matchmaker in condor pool B finds that a Resource R in condor pool B matches the requirements of the job, submitted by the agent A in condor pool A.

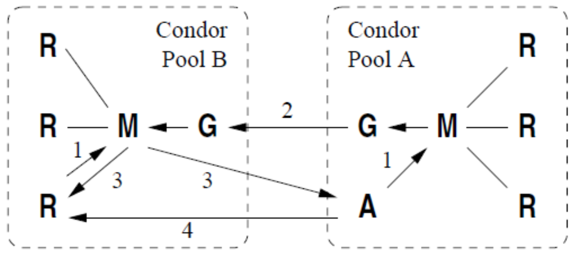

Figure 2-4 Gateway Flocking (Figure 4 from [6])

2.1.2.4 Direct Flocking

Gateway Flocking supports communicating at the organizational level, while direct flocking permits an individual user to join multiple communities. Instead of communicating through the gateways, in direct flocking, an agent (A) reports itself to multiple matchmakers (M) in different condor pools, as shown in Figure 2-5:

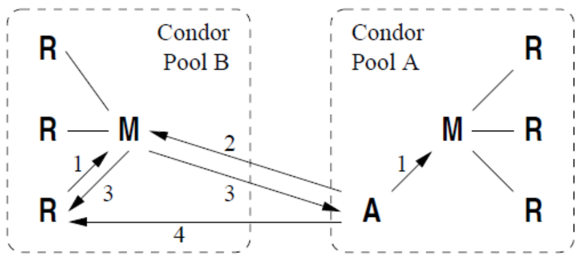

Figure 2-5 Direct Flocking (Figure 6 from [6])

2.1.2.5 Scheduling through Foreign Batch Queues

When an agent receives a large number of jobs, it may execute the jobs through foreign batch queues. This service is provided by Condor-G.

Figure 2-6 shows an example of scheduling of two jobs through foreign batch queues in two different condor pools:

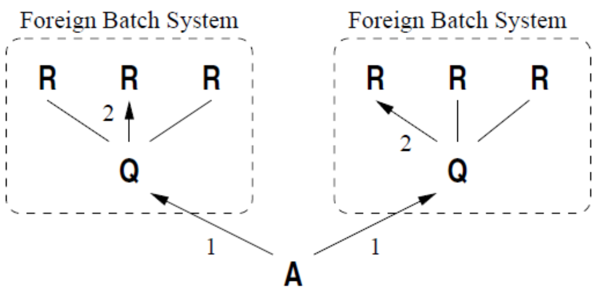

Figure 2-6 Scheduling through Foreign Batch Queues (Figure 7 of [6])

2.1.2.6 Gliding in

Thain et al. [6] state the following disadvantage of the Condor-G system:

- GRAM couples resource allocation and job execution, so that the agent must direct a particular job to a particular queue. Furthermore, the queue-based system must submit jobs to multiple queues or potentially long queues.

To solve the problem, Thain et al. [6] used “gliding in” technique. The 3-step process is explained through Figures 2-7, 2-8 and 2-9:

- Step 1: A Condor-G agent (A) submits the jobs, received by it, to two foreign batch queues via GRAM, shown in Figure 2-7.

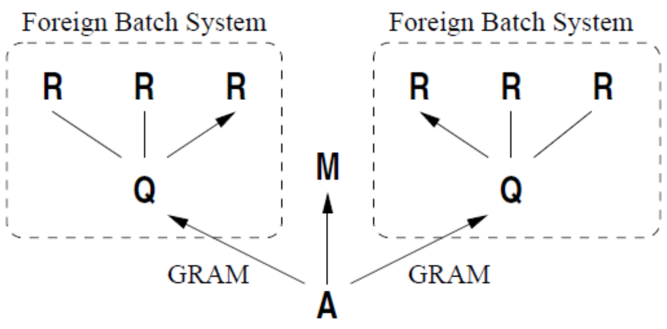

Figure 2-7 (Figure 8a from [6])

- Step 2: The resources form a personal condor pool with the user's personal matchmaker, shown in Figure 2-8.

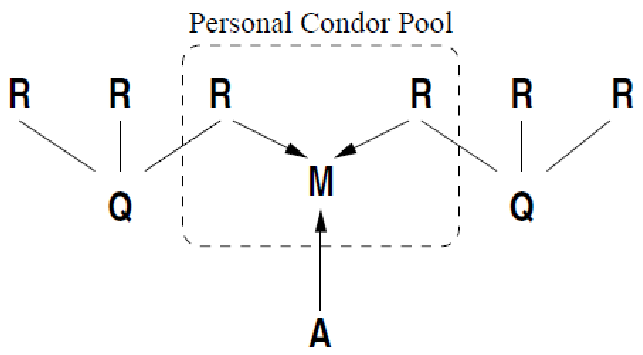

Figure 2-8 (Figure 8b from [6])

- Step 3: The agent gets the jobs executed through the resources in the personal condor pool as shown in Fig 2-9.

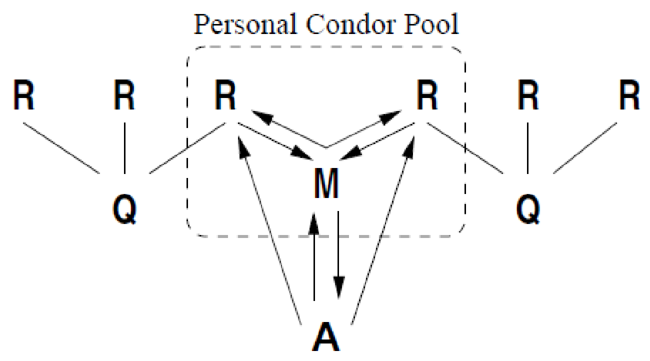

Figure 2-9 (Figure 8c from [6])

2.1.3 GridWay

Huedo et al. in [7] state that GridWay Framework is a tool that hides the complexity and dynamicity of Grid from developers and users, allowing the solution of large computational problems in a Grid environment by adapting the scheduling and execution of jobs to the changes in Grid conditions and in the demands from application. The Local Resource Management (LRM) system is the generic denomination for the cluster component which manages the execution of user applications [8]. GridWay is a Local Resource Management (LRM) like environment for submitting, monitoring, and controlling jobs [11]. In GridWay Framework, a transfer queue is used for communicating jobs from the Local Resource Management (LRM) system to the gridway meta-schedulers [11]. When jobs are submitted to the system, they are put into the transfer queue, and all the operations, such as start executing a job, terminating or suspending a running job, or resuming a job, are all executed through the queue, shown in Figure 2-10. It is concluded in [7] that GridWay provides adaptive scheduling and execution on Grids in an efficient way.

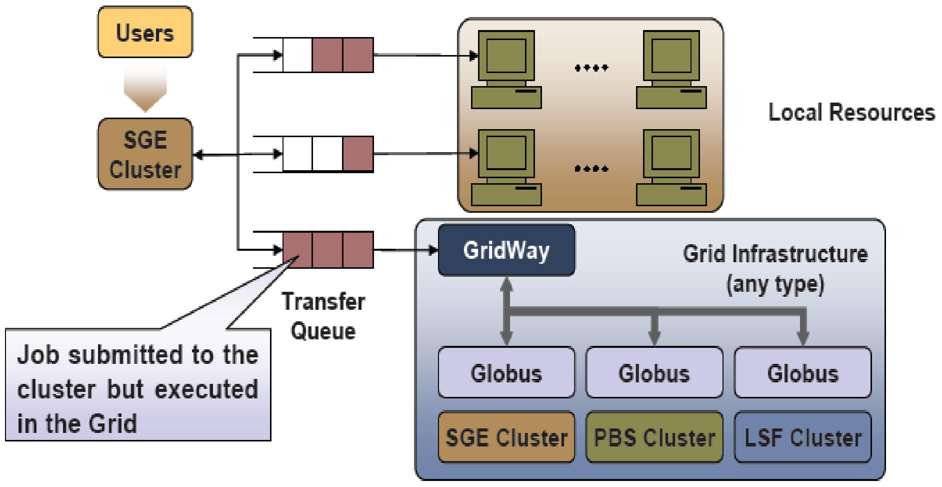

Figure 2-10 Transfer Queue in Gridway

2.2 Local Search Based Scheduling

The scheduling problem is recognized as a NP-complete problem [12], [13]. Various local search procedures have been developed for solving computational optimization. Local search can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions. Local search algorithms move from solution to solution in the space of candidate solutions (the *search space*) until a solution deemed optimal is found or a time bound is elapsed.

Local search algorithms can be used to solve the distributed scheduling problems: Given a set of distributed resources and jobs, scheduling is the process of allocating the jobs to the compatible resources. The objective of scheduling is to satisfy both the QoS of the users and the usage of the resources, and they are also used as the evaluation of the performance of scheduling. The relations between general local search algorithm and the application of the local search algorithm to the distributed scheduling are shown in Table 2-1:

	Local Search Algorithm	**Distributed Scheduling**
Objective	Finding a solution maximizing a criterion among a number of candidate solutions.	Finding a schedule minimizing the makespan, number of delayed jobs, total tardiness, etc.
Solution	A new solution is generated for each move.	A new schedule is generated for each one or several jobs arrival.
Termination	A solution deemed optimal is found; A time bound is elapsed; The best solution found by the algorithm has not been improved in a given number of steps, etc.	A schedule deemed optimal is found; A time bound is elapsed; The best solution found by the algorithm has not been improved in a given number of steps, etc
Feature	A deemed optimal solution (not global optimal solution) can be found	A deemed optimal schedule (not global optimal schedule) can be found

Table 2-1 How to Apply Local Search Algorithm in Scheduling Problems

Most local search based scheduling algorithms use heuristics [4]. The heuristic algorithms, such as Tabu Search (TS), Genetic Algorithm (GA), Simulated Annealing (SA), and Ant Colony Optimization (ACO), are considered to be a good way to find a local optimum solution in the search space [14].

Kousalya.K and Balasubramanie.P [14] showed the main strategy of Ant Colony Optimization (ACO), while Abraham et al. [15] give a brief introduction Tabu Search (TS), Genetic Algorithm (GA) and Simulated Annealing (SA).

2.2.1 Ant Colony Optimization

The Ant Colony Optimization (ACO) is a probabilistic technique for solving computational problems. In Ant Colony Optimization (ACO), the ants try to build a feasible solution to apply the stochastic decision policy repeatedly [14]. If one ant finds a good (i.e., short) path from the colony to a food source, it leaves a pheromone trail. Other ants are more likely to follow that path, and positive feedback eventually leads all the ants following a single path.

Kousalya.K and Balasubramanie.P [14], [16] use ACO to allocate a set of independent jobs to a number of distributed resources. The objective is to obtain a schedule with a minimum value of makespan. Three important values: pheromone (τ_{ij}), the attractive of the move as computed by some heuristic information (η_{ij}), and the completion time of ith job on the jth machine (CT_{ij}) are defined. The value of τ_{ij} indicates a prior desirability of the current move, while η_{ij} indicates how profiTable it has been in the past to make that particular move [14]. The value P_{ij} decides which job to be run, and which machine is used to execute that job. P_{ij} represents the probrability to move from a state i to a state j, and is calculated by the formula $P_{ij} = \frac{\tau ij \cdot \eta ij (\frac{1}{CTij})}{\sum \tau ij \cdot \eta ij (\frac{1}{CTij})}$. The proposed algorithm starts only if there are some tasks which are not scheduled. The authors of [14] propose the use of two algorithms. The value of pheromone evaporation ρ initialized to 0.05, while the pheromone deposit τ_0 is set to 0.01. The number of ants is set to 2.

Algorithm 1 aims to schedule the jobs which are in the set of unscheduled job list, while the local search algorithm is shown in Algorithm 2 for further optimization. The pseudo code of Algorithm 1 is shown in Table 2-2:

```
Algorithm 1 Algorithmic frame for a Ant Algorithm:

For each Ant do
    Randomly select $\mathbf{Task_i}$ and $\mathbf{resource_j}$
    Add ($\mathbf{Task_i}$, $\mathbf{resource_j}$, $\mathbf{free[j], free[j]+ET_{ij}}$) to the output list.
    Remove the $\mathbf{Task_i}$ from the unscheduled list to scheduled list
    For each $\mathbf{Task_i}$ in the unscheduled list do
        Calculate the heuristic information ($\eta_{ij}$), where
        A minimization function F and the heuristic information $\eta_{ij}$ is used to find out the best resource:
            $F = \max(free(j))$, and $\eta_{ij} = 1/free(j)$
        Find out the current pheromone trail value ($\tau_{ij}$)
        Update the pheromone trail matrix where,
            $\tau_{ij} = \rho\tau_{ij} + \Delta\tau_{ij}$
        Calculate the Probability matrix where,
            $P_{ij} = \frac{\tau ij \cdot \eta ij (\frac{1}{CTij})}{\sum_{j=1}^{m}\sum_{i=0}^{n} \tau ij \cdot \eta ij (\frac{1}{CTij})}$
        Find out the highest value of $\mathbf{P_{ij}}$ and add ($\mathbf{Task_i}$, $\mathbf{resource_j}$, $\mathbf{free[j]}$,
                                        $\mathbf{free[j]+ET_{ij}}$) to the output list.
        Remove the $\mathbf{Task_i}$ from the unscheduled list
        Modify the resource free time
            $\mathbf{free[j] = free[j] + ET_{ij}}$
    done
    Find out the best feasible solution by analyzing of all the ants scheduling list
done
```

Table 2-2 Algorithmic Frame for an Ant Algorithm (Algorithm 1 from [14])

Algorithm 1 can be concluded as the following 5 basic steps shown in Table 2-3, where the first three steps are done by each ant, and the 4th and the 5th step are performed by the whole system:

Step 1: Each ant randomly select job i from the set of tasks and resource j from the set of resources.

Step 2: Once a job is selected and scheduled, Task i is removed from the unscheduled list to the scheduled list.

Step 3: For each ant, in each iteration of Task i in the unscheduled list, calculate and find out the highest value (best solution) of P_{ij}.

Here, each ant generates a list of solutions, we store the best solution from each list into set S_l, *and then we select the best solution from* S_l*:*

Step 4: Find out the best feasible solution from the scheduling list of all the ants.

Step 5: Outputs ($Task_i$, $resource_j$, free [j], free[j]+ET_{ij}) to Algorithm 2.

Table 2-3 Ant Algorithm Applied in Scheduling

After the solutions generated from Algorithm 1, there may have some 'problem' resources that performs bad. The authors in [14] try to reduce the makespan using local search techniques. The neighborhood is a solution of single transfer of a job from the problem resource to any other resource [14], so that a new better solution may be generated. This is shown in Algorithm 2, and some values are defined and initialized as:

- S:Current solution, initialized as (Task$_i$, resource$_j$, free[j], free[j]+ET$_{ij}$) which is generated from Algorithm 1.
- s': New generated solution (initialized to NULL).

The pseudo code of Algorithm 2 is shown in Table 2-4:

Algorithm 2 Algorithmic frame for a local search algorithm:

```
Repeat until s' <> S
  Find out the problem resource's and problem resource's problem job
  Create neighbor of S(s') to transfer the problem job to some other resource
    If s' is better quality than S then
         S = s'
End repeat
The output is in S
```

Table 2-4 Algorithmic Frame for a Local Search Algorithm (Algorithm 2 from [14])

Algorithm 2 can be concluded as the following 2 basic steps shown in Table 2-5: create a set of neighbors of S, and find out the better solutions from the set of neighbors:

Step 1: Create a set of neighbors of S, calculate the value of neighbor s'. (The value of S and s' is evaluated by P$_{ij}$.)

Step 2: If s' is better than S, then S=s'. (A better solution is found)

Table 2-5 Local Search Algorithm

As a result, ACO is used in scheduling problem, and the experiment results in [14] shows that using ACO algorithm can achieve better resource utilization and better scheduling.

2.2.2 Tabu Search

Tabu Search (TA) is a meta-strategy to solve local optimality and has become an optimization approach that is used in many fields [15]. Tabu Search explores a set of problem solutions, repeatedly moves from one solution S to another solution S' in the neighborhood N(S) of S, to find a local optimal evaluated by some objective functions [15].

A template for simple Tabu Search [17] is shown in Table 2-6:

Notation:

- S: the current solution
- S*: the best-known solution
- f*: values of S*
- N(S): the neighborhood of S
- N'(S): the "admissible" subset of N(S) (i.e. non-tabu or allowed by aspiration)
- T: tabu list

```
Initialization:
    Choose (construct) an initial solution $S_0$.
    Set $S := S_0$, $f^* := f(S_0)$, $S^* := S_0$, $T := \emptyset$
Search:
    While termination criterion not satisfied do

    • Select S in argmin[(f(S'));

S'∈N'(S)

    • If $f(S) < f^*$, then set $f^* := f(S)$, $S^* := S$;
    • Record tabu for the current move in T (delete oldest entry if
      necessary);

Endwhile
```

In this algorithm, argmin returns the subset of solutions in N'(S) that minimizes f.

Table 2-6 Template for Simple Tabu Search (Simple Tabu Search in [17])

Klusacek et al. [18] applied Tabu Search (TS) for dynamic arrived distributed job scheduling problem. Rasooli [19] used Tabu Search based algorithm to allocate the independent jobs to the distributed resources.

The Tabu Search optimization stated in [19] aims to minimize both the flowtime (the total running time of all the jobs) and makespan (the maximum time that a job used) of the whole process. According to the algorithm of scheduling in [19], the distributed jobs are first, based on their deadline in ascending order, allocated to the resources, and waiting for their execution in a set of queues. Before executing, Tabu Search optimization is applied, that the jobs may be allocated to another position of the queues. Once a job is moved to another position in the queue, the flowtime and the total makespan are changed, and can be calculated by the scheduler in the system.

The Tabu Search Optimization applied in [19] can be described as the following 6 steps:

1. Select the resource with the highest expected flowtime/makespan.
2. Transfer the job which has the maximum completion time to other resources.
3. If a job is moved onto machines with smaller flowtime/makespan, the move is accepted.
4. Once the job is moved, it is placed into the Tabu-list to prevent cyclic moves.
5. Once the machine is selected, it is placed into the Tabu-list to prevent cyclic selection.
6. Repeat step 1 to 5, until a terminate condition is reached.

The experimental results in [19] show that using Tabu Search optimization can improve the flowtime and makespan of the distributed job scheduling problems.

A scheduling problem solved using Tabu Search is shown in Section 2.3.3.

2.2.3 Simulated Annealing

Simulated Annealing (SA) is another heuristic that searches through the neighborhood of an initial state to find a local optimum solution [15]. SA avoids getting trapped within a local minimum [15]. At the beginning of the algorithm, the control parameters, T (temperature), p (probability), and f (objective function) are set to an initial state. The value T is reduced by a specific rate in each move, and once it is reduced to a specific small value (set by the user), SA is terminated, and the solution is found.

Fidanova in [20] applied a SA-based algorithm to solve the scheduling problem. The objective is to schedule all the incoming applications (jobs) to the available distributed resources [20], and minimize the makespan for the complete schedule. The jobs are distributed and independent, while the resources may be either homogeneous or heterogeneous.

The author in [20] introduced three important variables: The completion time of job i on machine j (CT_{ij}), the function free(j), and the starting time of job i (b_i). The value of bi indicates the starting time of job i on machine j, where machine j would have finished the previously assigned jobs. The value of CT_{ij} is defined as the wall-clock time at which machine j completes job i, and it is calculated according to the formula $\mathbf{CT_{ij} = b_i + ET_{ij}}$, where ET_{ij} represents the expected execution time of task i on machine j. If machine j has no load before the allocation of job i, $b_i = 0$. Funtion free(j) indicates the time that machine j is free, that is, the value of b_i is decided by free(j).

The whole algorithm can be described as the following 5 steps:

- Generate the initial feasible solution S

As we mentioned, the objective is to schedule all the incoming applications (jobs) to the available distributed resources, so the solution of each allocation is represented by the triples (job, machine, starting time). For example, if one solution is written as (j_i, m_j, b_i), then it means job i is executed on machine j starting at time b_i. The set of solutions can be represented as a matrix M with three columns, the first column represents the jobs, the second represents the machines to execute the job in the same row, and the third represents the corresponding starting times. Each row of matrix M represents a solution. S represents the set of the solutions (rows) of M. The author in [20] used a greedy heuristic to create the initial solution: the first job executed on the first free machine j with the minimal value of free(j), and the same method is used for the second job in the set and so on.

- Initialize the Cooling Parameters

The author [20] has chosen the initial value of the temperature parameter T as 8 and the cooling rate F as 0.9.

- Generate the neighbors and select the solutions from S

The set of initial solutions S is generated and represented as a matrix with three columns. To create the neighbors (S') of S, we swap two of the tasks, so that the value of starting times and the free(j) functions are changed. If a better solution is generated from S', we replace this new solution as the current best solution. This swapping step is repeated according to the termination criterion, which is shown in the next two steps.

- Update the Temperature Parameters

The temperature parameter T is updated according to the formula $T_k = F*T_k$, where k = 1, 2, …

- Termination of the Solution (Termination Criterion)

The whole process is terminated if the value of T is updated less than a specific small value Θ, (where $0< \Theta <1$).

The experimental results in [20] show that using SA algorithm can obtain good load balancing of the machines and achieve a good performance

2.2.4 Genetic Algorithm

Genetic Algorithm (GA) can be used to solve optimization problems based on the genetic process of biological organisms [15].

GA has four basic processes:

- Initialization: Randomly generate the initial population.
- Selection: A proportion of existing population is selected to breed a new generation.
- Reproduction: Generate a second generation population through genetic operators: crossover and/or mutation.

- Termination: Terminate the process when the condition for termination is reached.

Aggarwal.M et al. [13] applied Genetic Algorithm to "obtain the best schedule for mapping of tasks to compute-nodes". In this paper, the assumption is that the grid workload may consist of multiple jobs, and each job is represented by a Directed Acyclic Graph (DAG). In [21], Martino and Mililotti applied Genetic Algorithm to solve the problem of "allocating a set of distributed jobs to a set of interconnected computing nodes". Fissgus [22] applied Genetic Algorithm to solve the problem of "scheduling of mixed task and data parallel modules comprising computation and communication operations" [22].

Abraham et al. [15] use Genetic Algorithm to allocate J_n (n=1, 2… N) independent user jobs to R_m (m=1, 2, … M) heterogeneous resources, and to minimize the makespan, and flowtime. The authors in [13] introduced an important variable: The completion time. It is the time, when the last job *j* finishes processing (C_j). The makespan in [15] is defined as C_{max} = max {Cj, j = 1, …, N}, and flowtime as $\sum_{i=1}^{N}$ Cj. To formulate the algorithm, the authors of [15] also proposed 3 job lists and 3 resource lists:

JList1 = Job list maintaining the list of all the jobs to be processed.
Jlist2 = Job list maintaining only the list of jobs being scheduled.
Jlist3 = Job list maintaining only the list of jobs already allocated.
Rlist1 = List of available resources (including time frame).
Rlist2 = List of resources already allocated to jobs.
Rlist3 =List of free resources = (Rlist1-Rlist2).

Table 2-7 shows the pseudo code of GA approach for job scheduling applied in [15]:

```
1. If the grid is active and (Jlist1=0) and no new jobs have been submitted, wait for new jobs to be submitted. Update Rlist1 and Jlist1.

2. If (Rlist1=0), wait until resources are available. If Jlist1>0, update Jlist2. If Jlist2 < Rlist1 (available resources) allocate the jobs on a first-come-first-serve basis and if possible allocate the longest job J on the fastest resource M according to the LJFR heuristic. If Jlist1 > Rlist1, job allocation is to be made by following the heuristic algorithm detailed below. Take jobs and available resources from Jlist2 and Rlist3.

3. At t =0; generate an initial population with P chromosomes $Pop_i(t)$, encoding the schedules. Feasibility of each chromosome is to be checked and makespan for each schedule represented by the chromosome is to be calculated. In certain cases, illegal offspring's (duplicated jobs, missing jobs, jobs outside the list) are generated due to genetic operators that require to be repaired immediately to ensure that each job appears only once in the sequence.

4. Begin GA loop
While (until the specified fitness value is achieved) do;

  a. For each chromosome (i=1 to P), first allocate the jobs to the available resources based on the LJFM heuristic and once a resource is free (due to job completion), a job is allocated based on the SJFM heuristic. There after LJFR – SJFR heuristic is applied alternatively after completion of every job.
  Calculate the make span and total flowtime for the generated schedule. Also calculate fitness value of each chromosome, $Fitness_i=FitPop_i(t)$;

  b. For (i=1 to P), Create new population NewPopi(t+1) which is choosen randomly based on the fitness value of each chromosome $Pop_j(t)$ in the current population Pop(t). The probability for selection a chromosome for the next generation may be defined as $p_j = \frac{\mathbf{Fitness}j}{\sum_{k=1}^{P} \mathbf{Fitness}k}$ or according to the selection strategy adopted;

  c. Apply crossover operator on the population according to the probability selected, Crospop(t+1) = recombined chromosomes of the population $NewPop_i(t+1)$;

  d. Apply mutation operator on the population according to the probability selected,
  MutPop(t+1)=mutated population CrosPop(t+1).

5. Evaluate fitness of each individual and when the $Fitness_i$ has reached the required value end loop.

6. Check the feasibility of the generated schedule with respect to resource availability and user specified requirements. Then allocate the jobs to the resources and update Jlist2, JList3, RList2 and Rlist3. Un-allocated jobs (infeasible schedules or resource non-availability) shall be transferred to JList1 for re-scheduling or dealt with separately.

7. Repeat steps 1-6 as long as the grid is active.
```

Table 2-7 GA Approach for Job Scheduling on the Grid (GA Approach in [15])

The basic steps of Genetic Algorithm: Initialization, Selection, Reproduction, and Termination applied in the GA approach for job scheduling on the Grid in [15] are shown in Table 2-8:

Initialization: Randomly generate the initial population

Selection: Select the new population according to $p_j = \frac{\textbf{Fitness}j}{\sum_{k=1}^{P} \textbf{Fitness}k}$ or the selection strategy adopted;

Reproduction:

- Crossover: Apply crossover operator on the population according to the probability selected, Crossop(t+1) = recombined chromosomes of the population *NewPopi(t+1);*
- Mutation: Apply mutation operator on the population according to the probability selected,
  MutPop(t+1)=mutated population *CrosPop(t+1).*

Termination: Evaluate fitness of each individual and when the *Fitness$_i$* has reached the required value *end loop.*

Table 2-8 Basic Steps of GA Approach

2.2.5 Hybrid Heuristic Algorithms

Some researchers have proposed hybrid heuristic algorithms. In [23], a hybridization of Genetic Algorithms (GAs) and Tabu Search (TS) for scheduling independent tasks in computational grids has been presented. In [24], an evolutionary hybrid scheduling algorithm is proposed. Benedict et al. [24] state that Niched Pareto Genetic Algorithm (NPGA) with Simulated Annealing (SA) works better than other heuristics, which they have tested.

Abraham et al. [15] have evaluated scheduling systems based on GA, TS and SA. The authors [15] claim that the hybridization of GA and SA algorithm provides a better convergence, while a hybrid GA and TS algorithm improves the search efficiency as compared to a scheduler, which uses GA only.

Abraham et al. [15] proposed hybrids GA-SA approach for job scheduling in Grids. Some steps of the hybrid GA-SA approach are the same as the GA approach stated in section 2.2.4.

Table 2-9 shows the pseudo code of hybrid GA-SA approach in [15]:

```
1 and 2 are the same as used in GA.

3 Generate an initial population of P schedule vectors and for i =1 to P, initialize the i_th threshold,
Th(i), with the energy of the i_th configuration. For each schedule (i=1 to P), first allocate the jobs to
the available resources based on the LJFM heuristic and once a resource is free (due to job
completion), a job is allocated based on the SJFM heuristic. There after LJFR – SJFR heuristic is
applied alternatively after completion of every job.

4 Begin the cooling loop
• Energy bank (EB) is set to zero and for i = 1 to N randomly mutate the i_th schedule vector.
• Compute the Energy (E) of the resulting mutant schedule vector.
• If $E > T_h(i)$ , then the old configuration is restored.
• If $E \le T_h(i)$ , then the energy difference ($T_h(i) – E$) is incremented to the Energy Bank (EB) =
EB+$T_h(i)$ –E. Replace old configuration with the successful mutant
End cooling loop.

5 Begin reheating loop.

    • Compute reheating increment $eb=\frac{EB*Ti(k)}{N}$, for i = 1 to N. (Ti(k) = cooling constant).
    • Add the computed increment to each threshold of the schedule vector.

End reheating loop.

6 Go to step 4 and continue the annealing and reheating process until an optimum schedule vector is
found.

7 Same as step 6 and 7 as mentioned in GA approach.
```

Table 2-9 GA-SA Approach for Job Scheduling in Grid (GA-SA Approach in [15])

The GA-SA approach is different from GA approach applied for job scheduling on the Grid in [15]. Instead of the reproduction step used in GA approach, GA-SA applies two basic steps of SA: cooling loop and reheating loop, which help to find a better schedule.

If an acceptable optimum schedule vector is found in the cooling and reheating process, the whole process is terminated.

Heuristic	Literature
Tabu Search	Klusacek et al. [18], Rasooli et al. [19]
Genetic Algorithm	Aggarwal et al. [13], Abraham et al. [15], Martino and Mililotti [21], Fissgus [22],
Simulated Annealing	Fidanova [20]
Ant Colony Optimization	Kousalya.K and Balasubramanie.P [14] and [16]
Hybrid	Abraham et al. [15], Xhafa et al. [23], Benedict et al. [24]

Table 2-10 Literature on the Use of Local Search Algorithms to Develop Grid Scheduler

2.3 Schedule Based Scheduling

Schedule based scheduling methods allow precise mapping of jobs onto resources in a dynamic environment [4]. The advanced Grid resource management system GORBA [25], as well as CCS [26] use schedule based policies to schedule jobs or workflows. Furthermore, in [6] and [25], heuristic algorithms are also applied for their further optimization.

2.3.1 GORBA

GORBA (Global Optimizing Resource Broker and Allocator) uses simple polices to create the schedule, and applies a heuristic algorithm as its optimization method. Suß et al. [25] state that the applications in GORBA are represented as workflows, while the resource broker is the component to form and calculate the schedule as well as planning for the future resource deployment. In GORBA, each work node is managed by the resource management system (RMS). A hybrid algorithm, HyGLEAM, is used in

GORBA for optimizing the schedules. HyGLEAM is a combination of the Evolutionary Algorithm GLEAM (General Learning Evolutionary Algorithm and Method) and local search algorithms [27]. Jakob et al. [27] state that GLEAM algorithm consists of the Evolution Strategy and real-coded Genetic Algorithms with data structuring concepts. The local search algorithms used by HyGLEAM are the Rosenbrock algorithm [28] and the Complex method [29].

Stucky et al. [30] state that if time constraint exists, only one heuristic is deployed. In addition, due to the re-schedule strategy, it is quite time consuming for the large number of jobs [30]. In GORBA, an application and resource management system is embedded between two interfaces:

- The Grid Application Interface (GAI) to users and to administrators [30]
- Interfaces to third-party grid middleware, such as Globus service [30]

Grid Application Interface (GAI) mediates between the application and resource management system. GORBA uses a third-party middleware layer to get up-to-date information of the resources [30]. Currently, GORBA uses Globus service as its third-party middleware, but it is planned to be flexible in deploying third-party software and utilize UNICORE alternatively [25].

Figure 2-11 shows the architecture of the GORBA system in Grid, and the coarse architecture of GORBA.

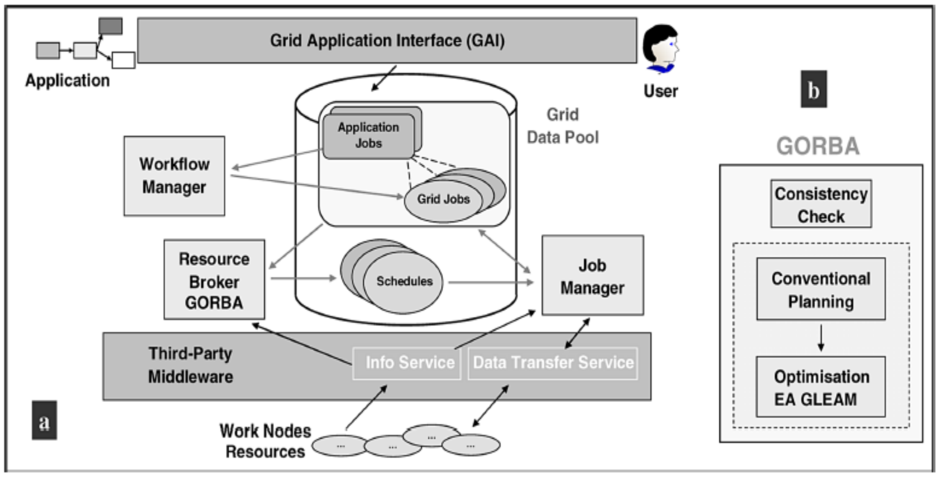

Figure 2-11 Architecture of GORBA in Grid (a) and the coarse architecture of GORBA (b) (Figure 1 from [30])

2.3.2 CCS

Hovestadt et al. [26] state that advanced reservation and quality of service are needed to be solved by co-allocation. It is hard for queuing approach based resource management system to solve this problem. Instead, advanced reservations can be solved by planning approach: assigning start times to each resource request, so that a complete schedule is planned. CCS (Computing Center Software) [26] has been designed to provide “a uniform access interface” [31] for HPC users, and provide “a means for describing, organizing, and managing high performance computing (HPC) systems” [26] for system administrators.

A complete schedule about the future resource usage is computed and made available to the users. First Come First Serve (FCFS), Shortest/Longest Job First (SJF/LJF) policies are used to assign the jobs into the schedule, while backfilling is applied to fill in the gaps of the schedule. CCS re-computes the schedule from scratch when a dynamic change such as job arrival or machine failure appears [4].

Fig 2-12 shows the 5 components of (Computing Center Software) CCS

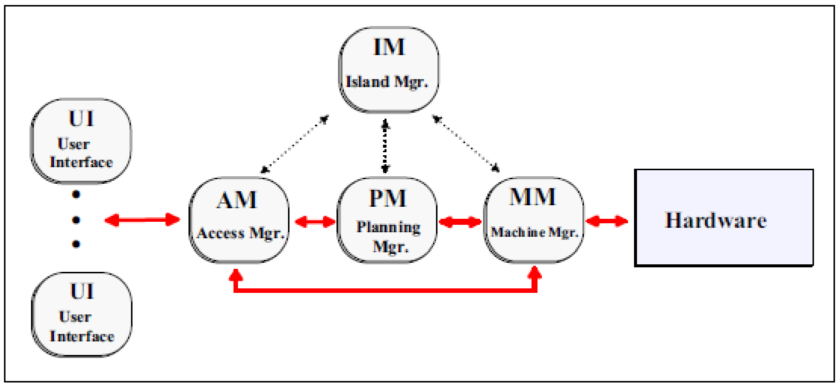

Figure 2-12 Architecture of CCS (Figure 3 from [26])

- The User Interface provides a single access point to one or more system via an X-window or ASCII interface.
- The Access Manager (AM) manages the user interfaces and is responsible for authentication, authorization, and accounting.
- The Planning Manager (PM) plans the user requests onto the machine.
- The Machine Manager (MM) provides an interface to machine specific features like system partitioning, job controlling, etc.
- The Island Manager (IM) / Domain Manager (OM) provides name service and watchdog facilities to keep the island/domain in a sTable condition.

Computing Center Software (CCS) provides mechanisms for the user-friendly system access and management of clusters [31]. It is defined as a planning system in [26], and as a schedule based system in [4]. CCS is able to handle both advanced reservation and quality of service [26], [31].

2.3.3 Earlier Gap, Earliest Deadline First and Tabu Search Optimization

Klusacek and Matyska [2] applied a schedule based method to handle the scheduling in dynamic environment. Dynamic jobs arrival is simulated, and a Tabu Search optimization is used for further optimizing the schedule.

Earlier Gap, Earliest Deadline First (EG-EDF), combined with Tabu Search Optimization is a solution designed and applied by Klusacek and Matyska [2] to simulate and solve "the distributed large scale dynamic arrived jobs scheduling" problem. Earlier Gap, Earliest Deadline First (EG-EDF) and Tabu Search Optimization are the two main parts of the scheduling process. Earlier Gap, Earliest Deadline First (EG-EDF) add newly arrived jobs to the schedule, while Tabu Search Optimization moves the jobs already allocated in the schedule and make further optimization for the schedule. Earlier Gap, Earliest Deadline First (EG-EDF) is applied for any time a new job arrives, while Tabu Search Optimization is applied after every 10 jobs have arrived.

The authors [2] attempt to satisfy three objectives: Minimizing the total tardiness (the total delayed time) of the jobs, the number of the delayed jobs, and the makespan (the processing time from the first start of the first job execution to the end time of the last execution) of the whole scheduling process.

2.3.3.1 Earlier Gap, Earliest Deadline First

Earlier Gap, Earliest Deadline First (EG-EDF) algorithm is applied for each newly arrived job into the existing schedule of each resource. Once a new job arrives, the best position for the job according to the current schedule is selected.

Gaps may be generated during the execution. A gap is considered to be the CPU idle (no job is executing on some CPUs of some machines) of the machines during the execution. A suitable gap for a job means the gap contains more CPUs than the job required. Figure 2-13 shows an example of a gap with 2 CPUs generated at time t_0.

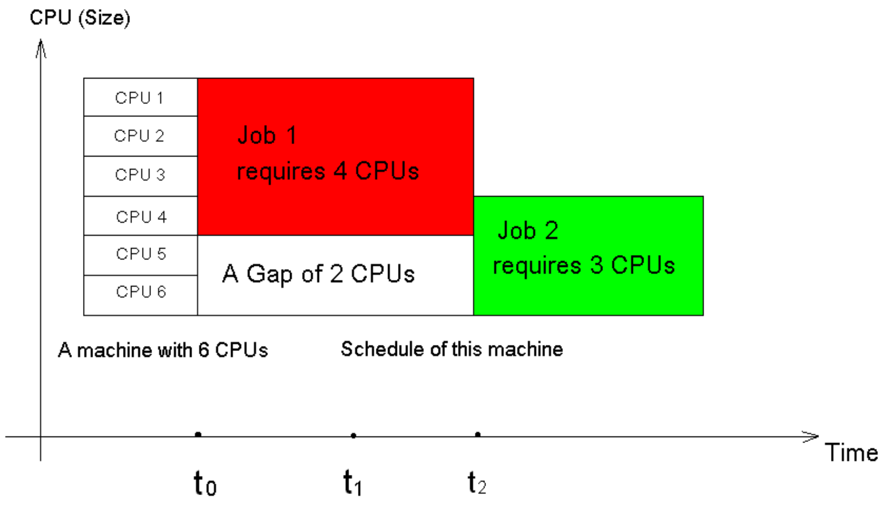

Figure 2-13 A Gap with 2 CPUs

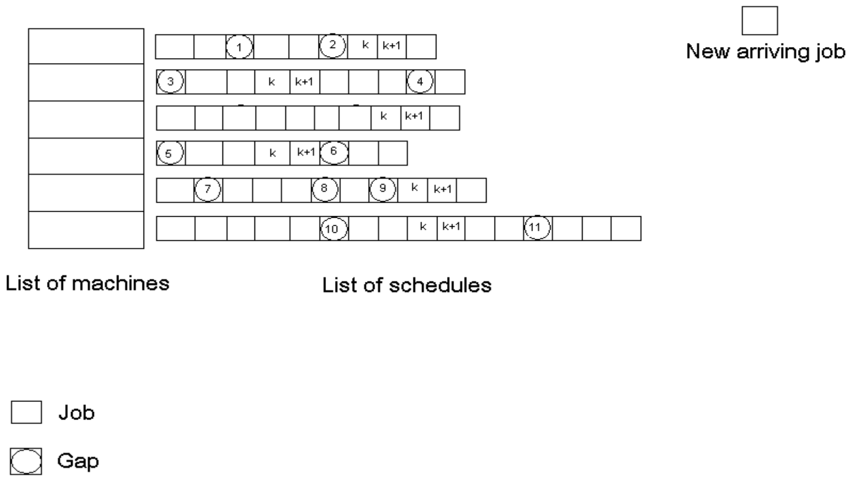

Figure 2-14 Earlier Gap, Earliest Deadline First

Figure 2-14 shows the main strategy of Earlier Gap, Earliest Deadline First (EG-EDF) Algorithm. When a new job arrives, it follows the following policies:

1. Gap 1 to gap 11 represent the suitable gaps for new arriving job, and all of them are candidate positions.

2. Generally, jobs on schedules of each machine are sorted according to their deadline. The deadline of the new arrived job is assumed between the *k*th and the *k+1*th job on each schedule. So, positions between k to k+1 of each schedule of machine are candidate positions.

3. EG-EDF tries on all the candidate positions, evaluates and figures out which candidate performs the best.

4. The scheduler search for all the candidate positions from the first machine to the last. For each machine, it is searched from the earliest candidate position to the latest one.

5. Once the job is allocated, it cannot be moved by EG-EDF.

6. The scheduler has all the information of the machines, schedules, and the jobs arrived.

7. The performance are evaluated by the three objectives: makespan, tardiness, and number of delayed jobs by using AcceptanceCriterion().

The pseudo code of EG-EDF is listed in Table 2-11 and Table 2-12:

```
Algorithm 1 Earliest Gap – Earlier Deadline First (job)

1: $schedule_{initial}$ := [$mach\ sched_1, .., mach\ sched_m$]; $schedule_{new}$ := ∅;
$schedule_{best}$ := ∅; gap_found := false; k := 0;
2: for i := 0 to m do
3:   if $machine_i$ is suiTable to perform job then
4:     if suiTable gap for job was found in $schedule_{new}[i]$ then
5:       gap_found := true;
6:       $schedule_{new}$ := $schedule_{initial}$;
7:       $schedule_{new}[i]$ := place job into found gap in $schedule_{new}[i]$; (EG strategy)
8:     else if gap_found = false then
9:       $schedule_{new}$ := $schedule_{initial}$;
10:      k := index of the first $job_k \in schedule_{new}[i]$ whose $d_{jobk} > d_{job}$; (k is the
index of the first job with later deadline)
11:      $schedule_{new}[i]$ := insert job into $schedule_{new}[i]$ between $job_{k-1}$ and $job_k$;
(EDF strategy)
12:    end if
13:    if AcceptanceCriterion($schedule_{best}$, $schedule_{new}$) = true then
14:      $schedule_{best}$ := $schedule_{new}$;
15:    end if
16:  end if
17: end for
18: return $schedule_{best}$
```

Table 2-11 Earliest Gap, Earlier Deadline First (EG-EDF from [2])

```
Algorithm 2 AcceptanceCriterion($schedule_{best}$, $schedule_{new}$)

1:  if $schedule_{best}$ = ∅; then
2:    return true;
3:  end if
4:  compute $makespan_{best}$ and $nondelayed_{best}$ according to $schedule_{best}$;
5:  compute $makespan_{new}$ and $nondelayed_{new}$ according to $schedule_{new}$;
6:  $weight_{makespan}$ := ($makespan_{best} - makespan_{new}$)/($makespan_{best}$);
7:  $weight_{deadline}$ := ($nondelayed_{new} - nondelayed_{best}$)/($nondelayed_{best}$);
8:  weight := $weight_{makespan} + weight_{deadline}$;
9:  if weight > 0.0 then
10:   return true;
11: else
12:   return false;
13: end if;
```

Table 2-12 Method AcceptanceCriterion() (AcceptanceCriterion() from [2])

2.3.3.2 Tabu Search Optimization

Unlike Earliest Gap, Earlier Deadline First (EG-EDF), Tabu Search is applied to make some further optimization for the jobs which are already allocated to the schedule of each machine. Tabu Search Optimization is necessary, even if the jobs are currently allocated to the best position by Earliest Gap, Earlier Deadline First (EG-EDF). EG_EDF algorithm does not guarantee to generate the best solution, when more new jobs arrive. Furthermore, once a job is allocated, even if a better solution were possible, EG_EDF will not allow the jobs to be moved.

As we mentioned, EG-EDF algorithm is applied for every newly arrived job, while Tabu Search algorithm is used for every 10 jobs arrived. Tabu Search is a local search based heuristic algorithm. The maximum number of iterations of the Tabu Search manipulation in [2] is set to 1000. During each iteration, only one job in the schedule is to be moved.

Before showing the Tabu Search Optimization, there are two concepts to introduce:

- Tabu Job

Tabu-Job is generated during the Tabu Search Optimization. TabuList is a list with specific size. In the process of Tabu Search Optimization, once a job is moved, or put back to the original position, it is put into the TabuList, and once a job is put into the TabuList, it becomes a tabu-job. Once a job becomes a tabu-job, it is no longer a candidate job that considered to be moved. Furthermore, sometimes the TabuList may be full, and a new job is put into the TabuList. In this case, the job first put into the TabuList is no longer a tabu-job, and that job becomes a candidate job that considered to be moved.

For example, if there the size of TabuList is 5 and now we put the 6th job into the TabuList. In this case, the 1st job in the TabuList is no longer a Tabu-Job.

The number of Tabu-Job is initialized as 0.

- Machine Used

The jobs have allocated by EG-EDF are stored in the schedules of machines, waiting for their executions or further moving by Tabu Search Optimization. In each iteration of Tabu Search Optimization, only one job is considered to be moved. Before selecting the jobs, the machine is selected in advance. If in some iteration, a machine may contain only tabu-jobs in its schedule. In that case, this machine becomes a “Machine Used”, or a used machine. Used machines are not considered to be a source to select.

Figure 2-15 and 2-16 show the main strategy of Tabu Search Optimization. When every 10 jobs have arrived and when the jobs have been allocated to suitable machines by EG-EDF, Tabu Search Optimization is applied and follows the following policies:

1. Before Tabu Search Optimization, all the machines and the jobs allocated in the schedules by EG-EDF are not in the “Tabu Job” and “Machine_used” List.

1. The Scheduler knows which jobs have been delayed.

2. In each iteration, the job selected to be moved is the last “non-tabu job” in the schedule of the “not used machine” with the highest number of delayed job.

3. Assume all gaps in Figure 2-15 are the suitable gaps for the job to be moved by Tabu Search Optimization, and are considered to be the candidate positions.

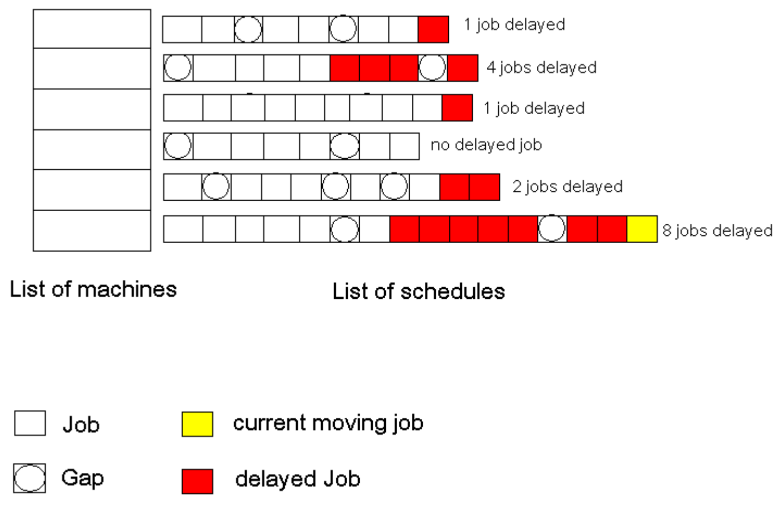

Figure 2-15 Tabu Search Optimization – Before Moving a Job

4. Tabu Search Optimization tries to move the job to the candidate positions, once a better performance occurs, the current iteration terminates, and the job is moved. A new iteration is started, shown in Figure 2-16.

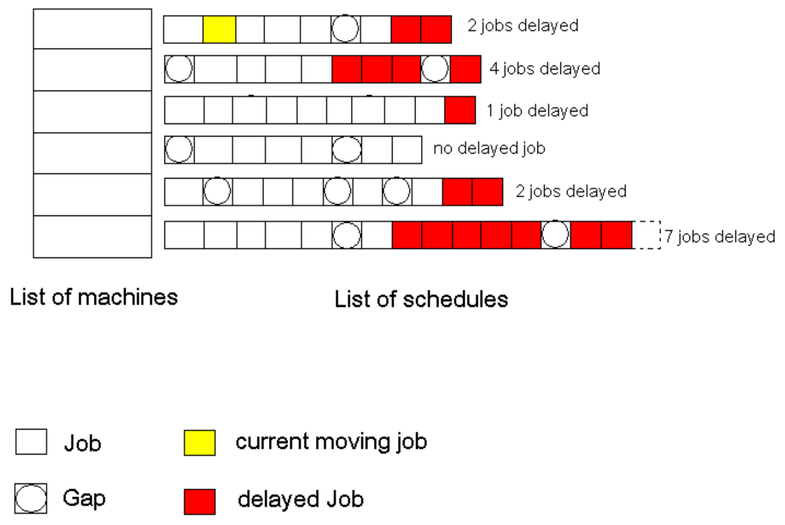

Figure 2-16 Tabu Search Optimization – After Moving a Job

5. If no better performance is found, the job is moved back to the original position and a new iteration is started.

6. Once a new iteration starts, the job tested in the last iteration is set as a "tabu-job", shown in Figure 2-17.

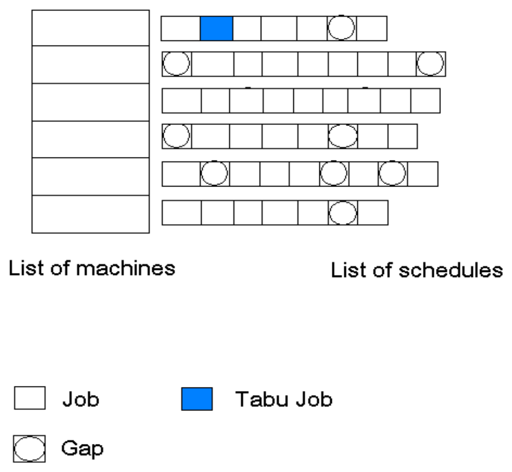

Figure 2-17 Tabu Job

7. Once a machine contains only "tabu jobs", it is set to a used machine, shown in Figure 2-18. And once all the machines are used machines, a new iteration is started, and all the machines are no longer used machine, shown in Figure 2-19.

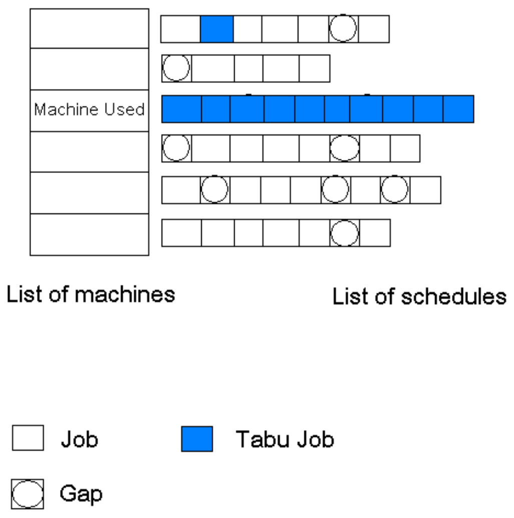

Figure 2-18 Machine Used

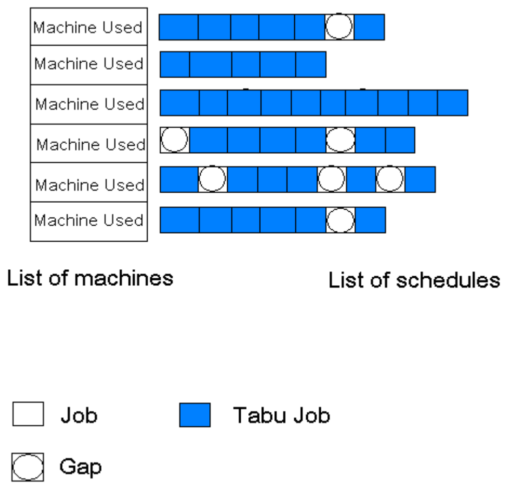

Figure 2-19 All Machines are Used Machines

The pseudo code of Tabu Search Optimization is listed in Table 2-13 and Table 2-14:

```
Algorithm 3 Tabu Search (iterations)

1:  $schedule_{best}$ := [$mach\ sched_1$, .., $mach\ sched_m$]; $schedule_{new}$ := $schedule_{best}$; $tabu_{jobs}$ := ∅; $machines_{used}$ := ∅;
2:  for $i$ := 0 to $iterations$ do
3:    $source$ := $k$ such that: $k \in (1..m)$, $machine_k \notin machines_{used}$, $schedule_{new}[k]$ has highest number of delayed jobs;
4:    if $source$ = $null$ then
5:      $machines_{used}$ := ∅; (All machines were used – start a new round)
6:      continue with new iteration;
7:    end if
8:    $job$ := last job from $schedule_{new}[source]$ such that: $job \notin tabu_{jobs}$;
9:    if $job$ = $null$ then
10:     $machines_{used}$ := $machines_{used} \cup machine_{source}$; (No non-tabu job is available in $schedule_{new}[source]$)
11:     continue with new iteration;
12:   end if
13:   remove $job$ from $schedule_{new}[source]$;
14:   if MoveJob($job$, $schedule_{best}$, $schedule_{new}$) = false then
15:     $schedule_{new}$ := $schedule_{best}$; (returns job to the original position);
16:   else
17:     $schedule_{best}$ := $schedule_{new}$; (updates the best so far found solution)
18:   end if
19:   $tabu_{jobs}$ := $tabu_{jobs} \cup$ job; (and remove oldest item if $tabu_{jobs}$ is full)
20: end for
21: return $schedule_{best}$
```

Table 2-13 Tabu Search Optimization (Tabu Search from [2])

```
Algorithm 4 MoveJob($job$, $schedule_{best}$, $schedule_{new}$)

1:  Sort the list of machines with their number of CPUs, if two machines have the same number CPUs, sort them according to
    their speed;
2:  for $j$ := 0 to $m$ do
3:    if $machine_j$ is suitable to perform job and suiTable gap for $job$ was found in $schedule_{new}[j]$ then
4:      $schedule_{new}[j]$ := place $job$ into found gap in $schedule_{new}[j]$;
5:      if AcceptanceCriterion($schedule_{best}$, $schedule_{new}$) = true then
6:        return true;
7:      else
8:      $schedule_{new}[j]$ := $schedule_{best}[j]$ (removes the proposed move);}
9:      end if
10:   end if
11: end for
12: return false;
```

Table 2-14 Method MoveJob() (MoveJob() from [2])

3. Motivation and Model Formalization

3.1 Motivation

In Chapter 2, we mentioned the schedule based scheduling method EG-EDF and Tabu Search Optimization. EG-EDF policy applied in [2] make use of resources by the utilization of gap, and the centralized scheduler is responsible for the evaluation of the performance of the whole system, such as makespan, tardiness, number of delayed jobs, and resource utilization.

As we mentioned, Tabu Search Optimization [2] method is applied to further optimize the performance after applying EG-EDF policy. However, it may generate some shortcomings that influent the whole performance. As we mentioned in chapter 2, the schedules are changed only according to the characteristics of the jobs waiting in the schedule. Every 10 jobs arrive, the scheduler would make alternations on the schedules according to the characteristics of those 10 jobs as well as the jobs former arrived but not starting their executions. This leads to the randomization of the alternation of the schedule, as well as the whole performance. Furthermore, it is impossible to guarantee the completion time of each machine at the same time.

Figure 3-1 and 3-2 shows the schedule forms after applying EG-EDF and Tabu Search Optimization:

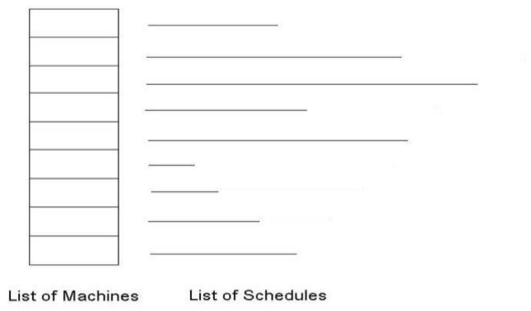

Figure 3-1 List of Schedules

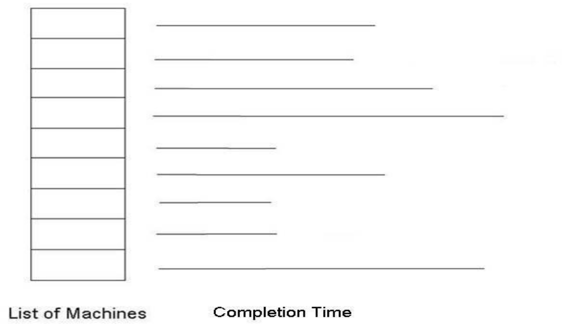

Figure 3-2 List of Completion Time

In Figure 3-1, the machines are sorted according to their speed of CPUs in descending order. The lengths of jobs that each machine handles are random, so that the completion time (shown in Figure 3-2) of each machine would also random, and it is impossible to complete the executions of each machine at the same time.

To fix this shortcoming, we aim to figure out a new optimization method to complete the execution of each machine at the same time. We allocate more lengths of jobs to the machines which have higher speeds, and fewer lengths of jobs to the machines which have slower speeds. The ideal schedules and completion times we aim to form are shown in Figure 3-3 and 3-4:

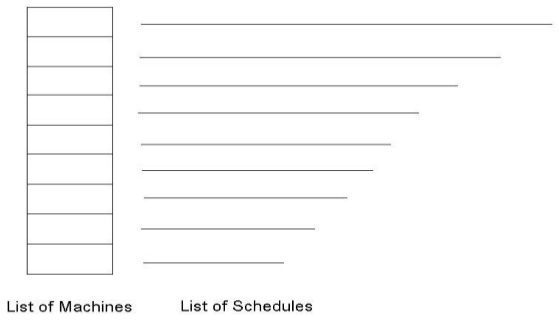

Figure 3-3 Objective List of Schedules

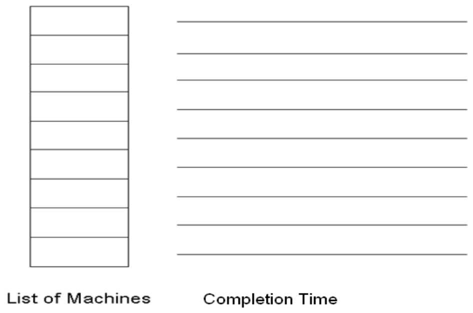

Figure 3-4 Objective List of Completion Times

3.2 Model Formalization

In this thesis, we attempt to solve the optimal scheduling problem in a dynamic model. However we assume that the computational resources are a set of machines, which are waiting for the arrival of a dynamic set of independent jobs. During the execution, once a job arrives, it is allocated to an appropriate machine by using a well-defined policy.

Before introducing the algorithm, we describe some basic definitions.

- *Release date*, of a job represents the time of arrival of that job.
- *Deadline* of a job is the desired completion time of that job.
- If job *j* fails to complete its execution before its deadline, it is *delayed*.

- The *tardiness* of job *j* is defined as the time interval between the actual completion time and the deadline of the job *j*.

- Job *j* may require one or more CPUs for its execution. Let m be the number of CPUs required by the job j during a particular interval of time after its release. During this interval of time, let n be the number of free CPUs available with resource i. Then job *j* can be executed on machine *i* provided n is greater than or equal to m.

- The interval of time is time required to execute job *j*. It will depend upon the speed of the machine *i*, and the length of job *j*.

Each machine in our work represents one resource, and contains multiple processors (CPUs). All the CPUs within a machine are assumed to be identical. The machines have the same allocation policy: It is not allowed to execute the same job on two or more machines. Moreover, once a job is being executed, it is not moveable until its completion.

3.3 Evaluation Criteria

We use 4 criteria for evaluating the quality of the schedule:

- Makespan

- Number of delayed jobs

- Tardiness

- Machine Utilization

3.4 Objectives

The two objectives of this thesis are:

1) Minimize the criteria: makespan, number of delayed job, tardiness, and machine utilization;

2) Further characterize the datasets (jobs) for which our approach would provide better.

4. Proposed Algorithm

4.1 Earliest Gap, Earlier Deadline First (EG-EDF)

The proposed algorithm is a modified version of the Klusacek and Rudova algorithm [2]. Our algorithm is also consists within two parts: when a new job arrives, we apply the Earliest Gap, Earlier Deadline First (EG-EDF) to generate the initial schedules, and then after the arrival of 10 jobs, and scheduled by using EG-EDF, the schedule is refined by using the Resource Characteristic Based Optimization (RCBO).

In Chapter 2, we have described EG-EDF. It is applied for every newly arrived job. It determines the ‘tentative best’ position for the newly arrived job in the schedule.

4.2 Resource Characteristic Based Optimization

To further optimize the schedule formed by EG-EDF, we apply a new Resource Characteristic Based Optimization (RCBO) every time after the number of new jobs, after the last application of RCBO, has reached the value of ten. RCBO is applied to change the positions of some jobs that have already arrived and are waiting in the schedules of some machines. RCBO may move the jobs to a better evaluated position.

As we mentioned, in our simulation, the jobs are dynamic, that is, the meta-data about the jobs are not known. The data about the machines is assumed to be static. Instead of creating the strategy of the Tabu Search optimization based on the information of the arrived jobs, as was attempted by Klusacek and Rudova [2], we use RCBO policy for optimization of the schedule.

The pseudo code of Resource Characteristic Based Optimization (RCBO) is shown in Figure 4-1 and 4-2:

```
1. Sort the machines in descending order according to their speeds
2. Select the last job from the slowest machine, and find the candidate positions of this job from
   the fastest machine to the slowest machine. Once a position is found, evaluate the
   performance according to the Weight_Function(). If the Weight_Function() return true,
   move that job to the position. Else, try the next position. If all the positions are tested and no
   better performance is found, return the job to the original position.
3. Once a job is selected, it is marked as job_selected, and not considered to be a source in the
   next iteration.
4. Once a machine contains only job_selected, all the jobs are no longer job_selected. This
   machine is marked as machine_selected.
5. The length of job_selected and machine_selected are set to 100. If more than 100 jobs are
   marked as a job_selected, the jobs marked earlier are no longer in the list of job_selected.
6. Iteration is set to 1000.
```

Figure 4-1 RCBO

```
1. Evaluate the value of current makespan $makespan_{cur}$ and num of delayed jobs $delayed_{cur}$;
2. Evaluate the value of new makespan $makespan_{new}$ and num of delayed jobs $delayed_{new}$;
3. $weight_{makespan}=(makespan_{cur} - makespan_{new})/makespan_{cur}$
4. $weight_{delayed}=(delayed_{cur} - delayed_{new})/delayed_{cur}$
5. $Weight\_Function() = weight_{makespan} + weight_{delayed}$
6. If Weight_Function() > 0, return true;
   Else return false;
```

Figure 4-2 Weight_Function()

The main strategy of RCBO is that, as shown in chapter 3, we try to allocate more lengths of jobs to the machines with a higher CPU speed, while allocating less to the machines with a lower CPU speed, while maintaining the constraints on the type of machines required for the job.

5. Experiment Evaluation

5.1 Testing for Validation

In this section, we present the test results for the proposed algorithm. The performance of our new approaches is tested using ALEA simulator [1]. We have compared our results with the results for the Klusacek and Rudova (KR) algorithm as reported in [2]. We have performed extensive tests by using 8 sets of jobs with the same 150 machines, as reported in [2]. The number of the 8 job sets are set to 3000, 12000, 30000, and 60000. The jobs are generated by a fuzzy generator after the range of parameters is set.

We assume that the resources as specified in the machine description file do not change. The jobs in each job description file, and the machines in the machine description file are represented by different parameters, and each of the parameters is generated randomly, according to a uniform distribution, within a reasonable range. For example, the number of CPU requirement of each job ranges from 1 to 8, while the machine speed ranges from 200 to 600.

Table 5-1 shows the range of each parameter of both the jobs and the resources. The performance of RCBO, compared with the algorithm applied in [2], is shown in Table 5-2, 5-3, and 5-4.

Job execution time	500–3000
Jobs with deadlines	70%
Number of CPUs required by job	1 – 8
Number of CPUs per machine	1 – 16
Machine speed	200 – 600

Table 5-1 Range of Parameters for the jobs

The performance of RCBO, compared with the algorithm applied in [2], is shown in Table 5-2, 5-3, 5-4 and 5-5.

Num of Job	KR	RCBO
3000	6846	6877
3000	6875	6877
12000	14803	14793
12000	17460	17468
30000	17460	17468
30000	14514	14457
60000	18462	18240
60000	12776	12652

Table 5-2 Comparison of Makespan

Num of Job	KR	RCBO
3000	2578	2570
3000	2578	2570
12000	11304	11304
12000	11736	11740
30000	29340	29370
30000	24520	24540
60000	49040	56520
60000	51820	51840

Table 5-3 Comparison of the Number of Nondelayed Jobs

Num of Job	KR	RCBO
3000	243108	246316
3000	239655	246316
12000	125624	128657
12000	34185	34137
30000	34185	34137
30000	1484505	1531140
60000	128657	125242
60000	435844	433932

Table 5-4 Comparison of the Total Tardiness

Num of Job	KR	RCBO
3000	92.05%	91.98%
3000	89.495%	90.705%
12000	90.535%	90.88%
12000	88.75%	88.995%
30000	88.94%	89.28%
30000	91.17%	91.49%
60000	92.985%	94.90%
60000	92.985%	93.92%

Table 5-5 Comparison of the Machine Utilization

We found that as compared with the KR algorithm makespans are improved in 4 of the 8 data sets by using RCBO algorithm. The number of non-delayed jobs is type better for RCBO algorithm for 5 of the datasets. As compared to the KR algorithm, the total tardiness is reduced by RCBO algorithm for 4 of the 8 datasets.

5.2 Analysis of Tests

The algorithms applied in [3] take processing time, which is highly uncertain. For some of the schedules, the optimization schedule may take a short time, while for others it may be required to spend a much longer time. The end-time for processing in different machines is also likely to differ a great deal. We sort the list of machines according to their CPU speed. Figure 5-1 shows the length of the schedules. The processing time of each machine is shown in Figure 5-2.

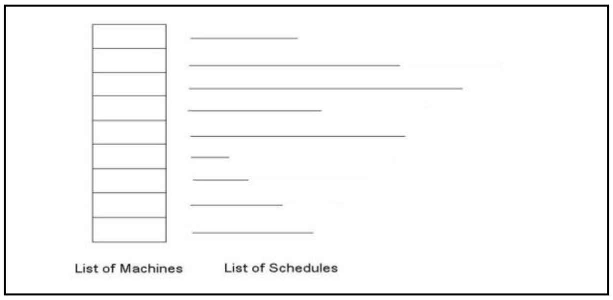

Figure 5-1. The Length of the Schedule in [3]

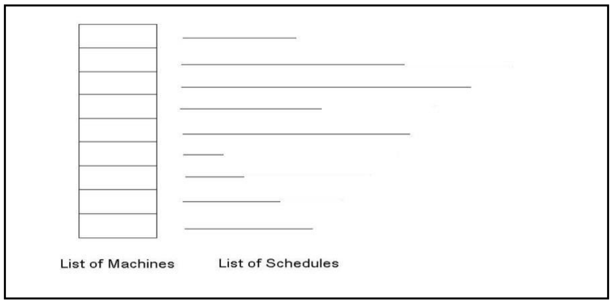

Figure 5-2. The Processing Time in [3]

5.2.1 Objective of RCBO

Figure 5-3 shows the schedules that RCBO algorithm aims to form.

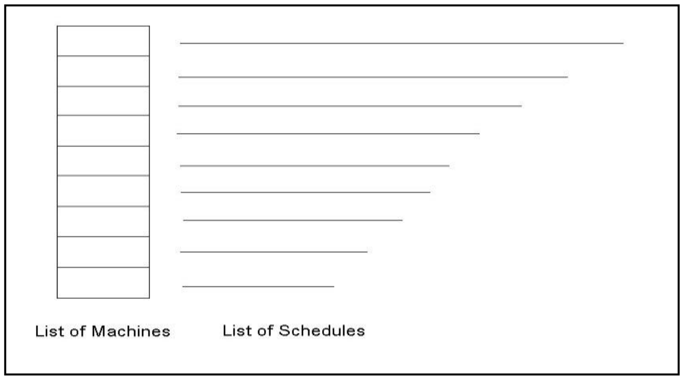

Figure 5-3. Length of Schedules using RCBO

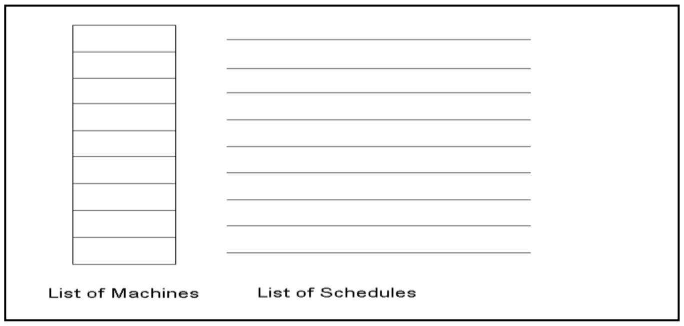

Figure 5-4. The Processing Time using RCBO

If we can make the time of each schedule of machines equivalent, shown in Figure 5-4, the processing time of the total schedule would be shortened. Since the requirements for each job may be different and the machines in a grid are assumed to be heterogeneous, it may not be possible to achieve the ideal, as shown in Figures 5-3 and 5-4. However since the algorithm expressly aims at the objective, the expectation is that the results obtained by RCBO algorithm would be better than those from the KR algorithm.

5.2.2 Analysis of the total processing time of the proposed new approach

Before the analysis of the new approach, we state some variables in Table 5-5:

VARIABLE	REPRESENTATION
S_i	The CPU speed of the *i*th machine
N_i	Number of CPU on the *i*th machine
R_j	Number of CPU required by job *j*
$T_{j,i}$	Processing time of job *j* on machine *i*
$Length_j$	The length of job *j*
L_i	Total job length (Schedule length) on machine *i*

Table 5-6 Variables

The objective is to optimize the total schedule, and the performance of total schedule is decided by the slowest schedule[k] (if the *k*th machine performs slowest) of the set of *m* machines. So, we aim to minimize schedule[k] = $schedule[i]_{max}$, where $i \in [1, m]$.

We have $\boldsymbol{T_{j,i} = Length_j \ / \ S_i}$, (each machine) and $\boldsymbol{L_i = \sum_{x=1}^{n} Lengthx, i}$ (Assume schedule[i] contains *n* jobs).

So, processing time of schedule[i] can be represented as:

$\boldsymbol{T_i = L_i/S_i}$, with the constraint that $N_i \geq R_j$.

Our objective can be represented as:

Minimize $\boldsymbol{(L_i/S_i)_{max}}$

To analysis the two forms shown in Figure 5-3 and Figure 5-4, we research on one **same set of jobs**. Given a set of jobs, we have the corresponding total Length of the jobs, which can be represented by:$\sum_{i=1}^{m} \boldsymbol{Li}$.

Fig 5-4 shows the machines finish execution of the schedule[i] at the same time, say T.

If some machines finish their executions faster than T, others must finish slower than T. (As shown in Fig 5-2.) As a result, T is the smallest value of execution time.

Fig. 5-4 is showing a good form that minimizing the schedule, which is minimizing the value of schedule[i]$_{\text{max}}$

5.2.3 Distribution of the parameters of the jobs

The number of CPU required by each job is between 1 and 8. Figure 5-5 shows the CPU distribution of each data set that we used to test:

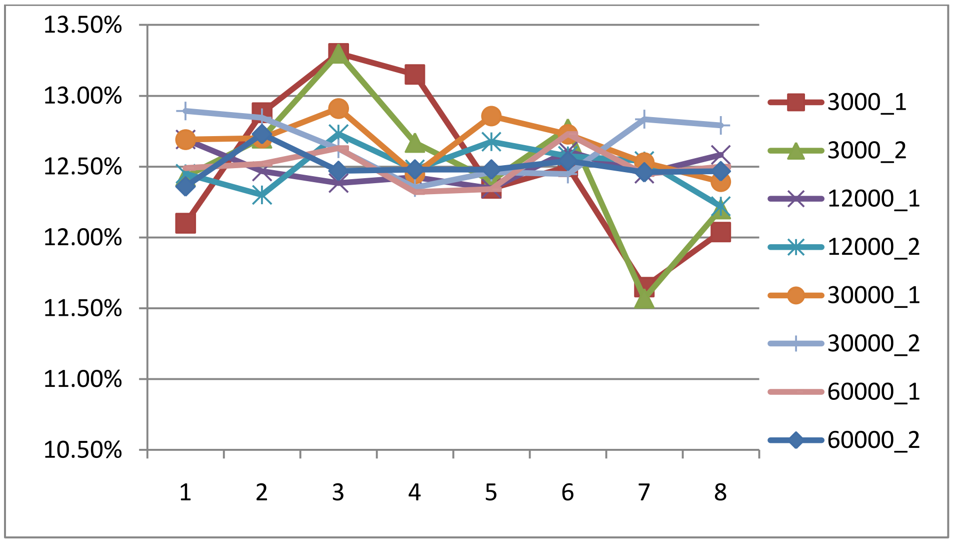

Figure 5-5 CPU Distribution of Each Date Set

In Fig.5-5, x-axis represents the number of CPUs a job required, while y-axis shows the percentage of the quantity the job requires the specific number of CPU. For example, a point P (2, 12.5%) represents there are 12.5% jobs of the total jobs require 2 CPUs, and if the total number of jobs of a data set is 60000, then there are 60000×12.5% = 7500 jobs require 2 CPUs.

We randomly select 4 data sets with the number of jobs 3000, 12000, 30000, and 60000 respectively. The CPU distribution is shown in Fig 5-6:

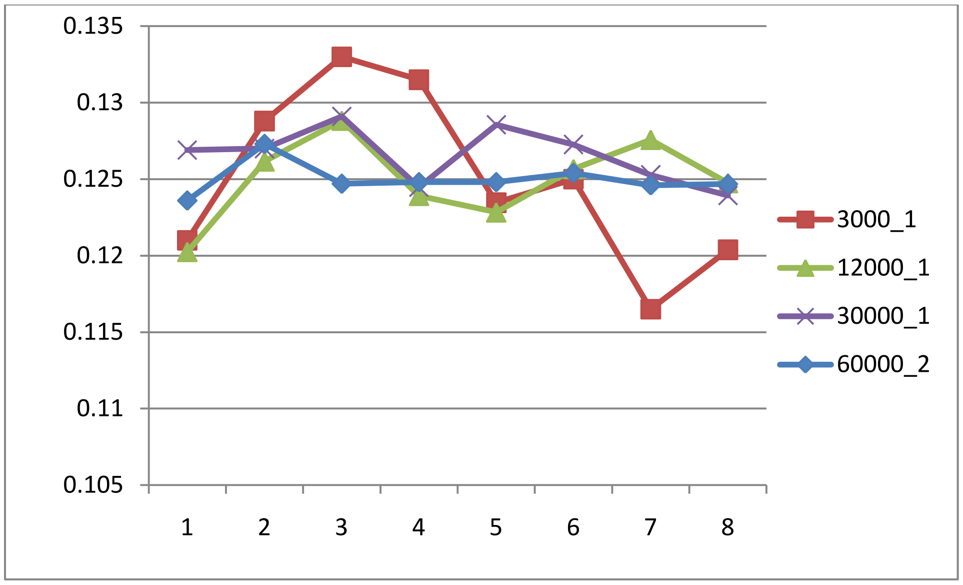

Figure 5-6 CPU Distribution of 4 Data Sets

In Fig. 5-6, the CPU distribution is shown. To represent the property of each distribution, we use the definition of “standard deviation” and “variance” to represent how much variation there is from the "average" (mean).

5.2.4 Standard Deviation

In probability theory and statistics, the standard deviation of a statistical population, a data set, or a probability distribution is the square root of its variance. Standard deviation is a widely used measure of the variability. The variance of a random variable or distribution is the expected, or mean, or the deviation squared of that variable from its expected value or mean. Thus the variance is a measure of the amount of variation within the values of that variable.

Table 5-6 shows the value of standard deviation and the variance of the CPU distribution of the 8 data sets.

Number of Job	Standard Deviation of CPU Distribution	Variance of CPU Distribution	Number of Criteria Improved	Average Number of Criteria Improved
3000_1	0.539%	0.290%	0	2
3000_2	0.469%	0.219%	4	
12000_1	0.254%	0.065%	2	2.5
12000_2	0.162%	0.026%	3	
30000_1	0.174%	0.030%	3	3
30000_2	0.198%	0.039%	3	
60000_1	0.128%	0.0163%	4	4
60000_2	0.099%	0.0098%	4	

Table 5-7 Standard Deviation and Variance of CPU Distributions

5.2.5 Analysis of the probability of two different distributions

To find the relations between the number of jobs, the value of standard deviation and the variance, and the performance, we generate Fig 5-7, and Fig 5-8:

Figure 5-7 shows the relations between the number of jobs and the number of criteria (performance) improved:

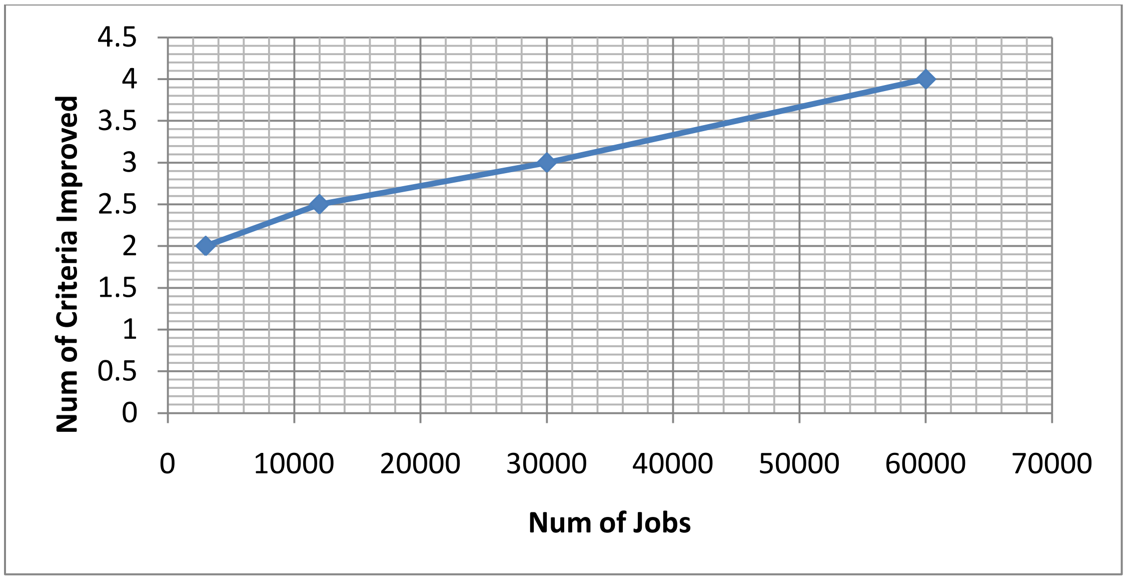

Figure 5-5 Relations between the Number of Jobs and the Average Number of Criteria Improved

In Figure 5-7, x-axis represents the number of jobs, while y-axis shows the number of the criteria improved. As it is shown, the more the jobs, the more criteria improved. This is analyzed and proved in part a) of this section.

Figure 5-8 shows the relations between the number of the criteria that improved and the average standard deviation of the CPU distribution:

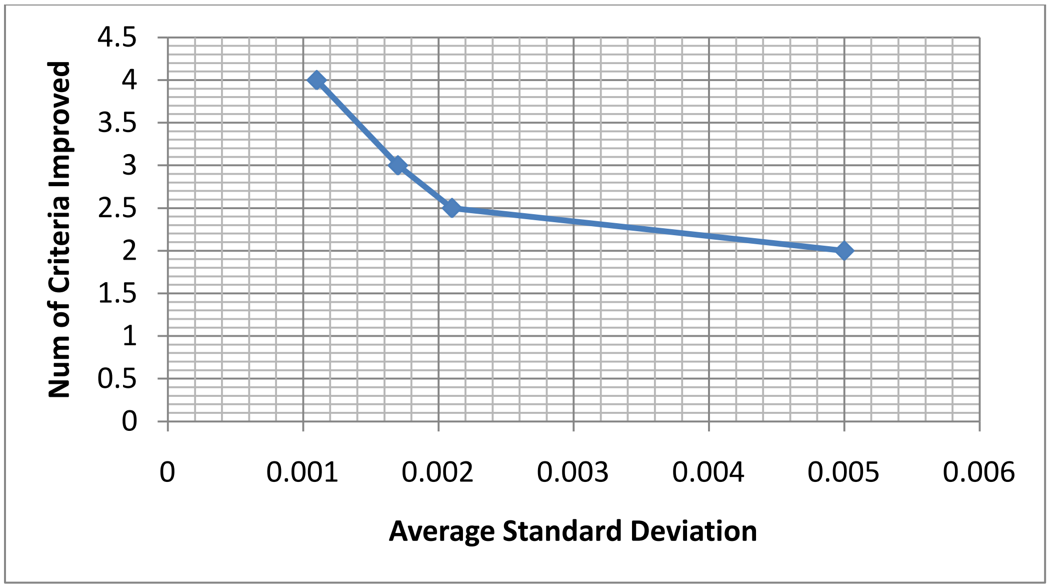

Figure 5-6 Relations of the Average Number of the Criteria Improved and the Average Standard Deviation of the CPU Distribution

In Figure 5-8, x-axis represents the value of the average standard deviation of the data sets with the same number of jobs, while y-axis shows the number of the criteria improved. As it is shown, the lower the value of the standard deviation it is, the more criteria improved. This is analyzed and proved in part a) of this section.

From both Figure 5-7 and 5-8, we can indicate that the more the number of jobs it has, the lower the value of the standard variation it would be. This indication can be proved by the property of the randomization: If the number of job is large enough, the distribution of their CPU required is more likely to be equivalent, that is, the value of standard deviation and variance would more likely to be smaller. While, if the number of job is small, the distribution would be less equivalent, that is, the value of standard deviation would more likely to be larger.

According to Figure 5-7, and 5-8, we generate Figure 5-9, which shows the relations between the value of standard variation and the number of jobs:

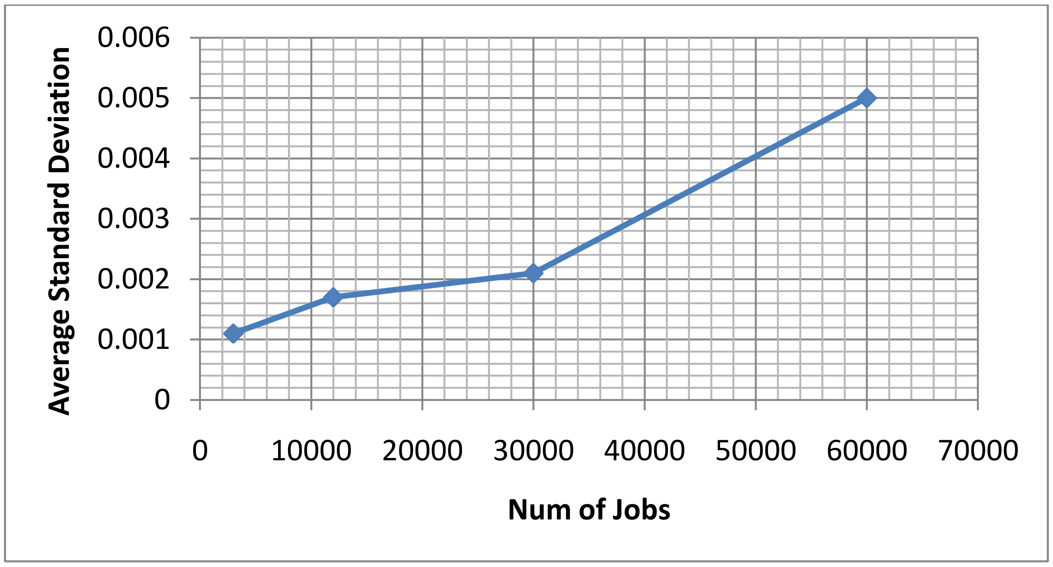

Figure 5-7 Relations between the Average the Standard Variation and the Number of Jobs

In Figure 5-9, x-axis represents the number of the jobs, while y-axis shows the value of the correspond value of standard deviation.

As we have indicated and analyzed, the larger the number of jobs it has, the lower the value of the standard variation it would be.

Therefore, if the quantity of job is limited in a small amount (for example, 10 jobs), even if the parameters varied in a specific range, it is hard to predict the parameters of each job. It is even harder to predict the parameter of the next job arrives. However, if we focused on a large number of jobs (infinite number of job if possible), and each parameter of jobs are varied in a same specific range, it is reasonable to predict the distribution of the jobs, and the parameter of the next job.

6. Conclusion and Future Works

In this thesis, we have proposed a new resource characteristic based optimization algorithm. The algorithm first uses EG-EDF to put every newly arrived job in the existing schedule. After the schedule has been thus modified ten times by ten newly arrived jobs, it is refined by using RCBO method.

Compared with the KR algorithm, the RCBO algorithm generates solutions, which in many cases, are better than those generated by using the KR algorithm. We are working on modifying various parameters of the RCBO algorithm to improve its working. We are also conducting further studies to determine whether a meaningful correlation between the standard deviation in the characteristics of the data sets with the improvement in the performance by use of the RCBO algorithm can be established.

In this thesis, resource characteristic based optimization implemented using a static set of resources and dynamic jobs environment, where the dataset of the jobs and the resources are generated synthetically. In the future, we intend to incorporate additional factors, such as the preemptive of the job, the failure tolerance, and the cost of the process. We are investigating algorithm’s performance using larger and real workloads. Furthermore, new proposed algorithms can be implemented into a more complex model with both dynamic resources and jobs.

Bibliography

[1] Klusacek, D., Matyska, L. and Rudova, H. Alea – Grid Scheduling Simulation Environment. In Parallel Processing and Applied Mathematics. Heidelberg, Germany: Springer-Verlag, Lecture Notes in Computer Science 4967. (2008)

[2] Klusacek, D. and Rudova, H. Local Search for Grid Scheduling. In Doctoral Consortium at the International Conference on Automated Planning and Scheduling (ICAPS'07), Providence, RI, USA. (2007)

[3] F. Dong, S.K. Akl, Scheduling algorithms for grid computing: State of the art and open problems, Technical Report No. 2006-504, School of Computing, Queen's University, Kingston, Ontario, Canada, January. (2006)

[4] Klusacek, D. and Rudova, H, Improving QoS in computational Grids through schedule-based approach. In Scheduling and Planning Applications Workshop at the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS'08), Sydney, Australia. (2008)

[5] Gentzsch, W. Sun Grid Engine: towards creating a compute power grid. In Proceedings of the first IEEE/ACM International Symposium on Cluster Computing and the Grid, 35–36. (2001)

[6] Thain, D., Tannenbaum, T., and Livny, M. Distributed computing in practice: the Condor experience. Concurrency - Practice and Experience 17(2-4):323–356. (2005)

[7] Huedo, E., Montero, R., and Llorente, I. The GridWay framework for adaptive scheduling and execution on Grids. Scalable Computing: Practice and Experience 6(3):1–8. (2005)

[8] Borges, G., David, M., Gomes, J., Fernandez, C., Lopez Cacheiro, J., Rey Mayo,P., Simon Garcia, A., Kant, D., & Sephton, K. Sun Grid Engine, a new scheduler for EGEE middleware. In IBERGRID–Iberian Grid Infrastructure Conference. (2007)

[9] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In Proceedings of the IPPS/SPDP Workshop on Job Scheduling Strategies for Parallel Processing, pages 62-82. (1988)

[10] Ian Foster and Carl Kesselman. Globus: A metacomputing intrastructure toolkit. International Journal of Supercomputer Applications, 11(2):115-128. (1997)

[11] http://www.ogf.org/OGF20/materials/789/GridWay_Update_vOGF.pdf as of 2nd March 2010

[12] Papadimitriou, C. and Steilglitz, K. Combinatorial Optimization: Algoritms and Complexity, Dover Publications, 1998.

[13] Mona Aggarwal, Robert Kent and Alioune Ngom, "Genetic Algorithm based Scheduler for Grid Applications", accepted at HPCS' 05, Guelph

[14] Kousalya.K and Balasubramanie.P. Ant Algorithm for Grid Scheduling Powered by Local Search. Int. J. Open Problems Compt. Math., Vol. 1, No. 3, December. (2008)

[15] Abraham, R. Buyya, B. Nath, Nature's heuristics for scheduling jobs on Computational Grids, International Conference on Advanced Computing and Communications (2000).

[16] K. Kousalya. To Improve Ant Algorithm's Grid Scheduling Using Local Search. International Journal of Intelligent Information Technology Application, 2(2):71-79. (2009)

[17] http://www.ifi.uio.no/infheur/Bakgrunn/Intro_to_TS_Gendreau.htm as of 27th March 2010

[18] Klusacek, D., Matyska, L., and Rudova, H. Local Search for Deadline Driven Grid Scheduling. In: Third Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2007), pp. 74–81 (2007)

[19] Aysan Rasooli, Mohammad Mirza-Aghatabar, Siavash Khorsandi. Introduction of Novel Dispatching Rules for Grid Scheduling Algorithms. In Proceedings of the International Conference on Computer and Communication Engineering, Kuala Lumpur, Malaysia, pp. 13-15 (2008)

[20] Fidanova, S.: Simulated Annealing for Grid Scheduling Problem. In: IEEE John Vincent Atanasoft International Symposium on Modern Computing(JVA' 06), pp. 41-45 (2006)

[21] Di Martino, V. and Mililotti, M. Sub optimal scheduling in a grid using genetic algorithms. Parallel Computing, Vol 30, p. 553.565. (2004)

[22] U. Fissgus. Scheduling Using Genetic Algorithms. In Proceedings of the20th International Conference on Distributed Computing Systems. IEEE Computer Society Press, pp:662-669, (2000)

[23] Xhafa, F., Gonzalez, J.A., Dahal, K.P., Abraham, A.: A GA(TS) Hybrid Algorithm for Scheduling in Computational Grids. In: Corchado, E., Wu, X., Oja, E., Herrero, A ., Baruque, B. (eds.) HAIS 2009. LNCS (LNAI), vol. 5572, pp. 285–292. Springer, Heidelberg (2009)

[24] Shajulin Benedict, Rejitha R. S, and V. Vasudevan. An Evolutionary Hybrid Scheduling Algorithm for Computational Grids. Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol.12, No.5 pp. 479-484. (2008)

[25] Suß, W., Jakob, W., Quinte, A., and Stucky, K.-U. GORBA: A global optimising resource broker embedded in a Grid resource management system. In Zheng, S. Q., ed., International Conference on Parallel and Distributed Computing Systems, PDCS 2005, 19–24. IASTED/ACTA Press. (2005)

[26] Hovestadt, M., Kao, O., Keller, A., and Streit, A. 2003. Scheduling in HPC resource management systems: Queuing vs. planning. In Feitelson, D. G.; Rudolph, L.; and Schwiegelshohn, U., eds., 9th International Workshop, JSSPP, volume 2862 of LNCS, 1–20. Springer. (2003)

[27] Jakob, W., Quinte, A., Süß, W., Stucky, K.-U.: Optimised Scheduling of Grid Resources Using Hybrid Evolutionary Algorithms. Proc. 6th Int. Conf. on Parallel Processing and Applied Mathematics, Poznan, PL, Springer, LNCS 3911. (2005)

[28] ROSENBROCK, H. H. "An automatic method for finding the greatest or least value of a function," The Computer Journal, Vol. 3, p. 175. (1960).

[29] M. J. Box, A new method of constrained optimization and a comparison with other methods,
Compute. Journal., 8, pp. 42–52. (1965)

[30] Stucky, K.-U., Jakob, W., Quinte, A., and Suß, W. Solving scheduling problems in Grid resource management using an evolutionary algorithm. In Meersman, R., and Tari, Z., eds., OTM Conferences (2), volume 4276 of LNCS, 1252–1262. Springer. (2006)

[31] A. Keller and A. Reinefeld. Anatomy of a Resource Management System for HPC Clusters. In Annual Review of Scalable Computing, vol. 3, Singapore University Press, pages 1–31, (2001)

VITA AUCTORIS

Name:	Peng Du
PLACE OF BIRTH:	Beijing, China
YEAR OF BIRTH:	1985
EDUCATION:	Wuhan University of Technology, Wuhan, China
	2003- 2007 BEng
	University of Windsor, Windsor, Ontario, Canada
	2008-2010 M.Sc