Objective-Oriented Algorithm for Job Scheduling in Parallel Heterogeneous Systems

Pham Hong Hanh * and Valery Simonenko **

Department of Computer Science
National Technical University of Ukraine
KPI-20-20, Pr. Pobedy 37, Kiev-252056,Ukraine.

* Hanh@pham.kiev.ua
** V.Simonenko@p47.f360.n463.z2.fidonet.org

Abstract. This paper presents a new approach to solve the problem of job scheduling for parallel processing in *heterogeneous* systems. The optimization goals are: (i) minimum total execution time including communication costs and (ii) shortest response time for all jobs. We introduce a classification for the given scheduling problem by the heterogeneity of the systems, from the view of the schedulers' eyes. Then, according to this analysis, a new scheduling strategy for so-called "Strictly-Heterogeneous" systems is proposed. The key idea of the new approach is the use of the Hungarian method, which provides a quick and objective-oriented search for the best schedule by the given optimization criteria. In addition, by modifying this method into so-called Objective-Oriented Algorithm (OOA), the time complexity for scheduling is decreased to O(n(E+nlogn)). The simulation results show us that OOA provides better solution quality while scheduling time is less than the existing methods.

1 Introduction

For the last few years, a number of job scheduling algorithms for uniform systems have been published. However, in heterogeneous systems, where not only jobs (tasks) belong to different classes but also, resources (computing nodes) can be heterogeneous, scheduling problems are *more complex* than the ones in uniform systems. Besides, a lot of algorithms for uniform systems are specialized on a particular uniform type of the system architecture [4],[15],[2]. Therefore, applications of these algorithms for heterogeneous systems are limited.

In the uniform systems, because of the homogeneity of the jobs and the resources, the optimization for job scheduling is carried out at high level where the jobs (and the resources) are considered not alone but in some groups (batches or gangs). Meanwhile, in the heterogeneous systems, the optimization for job scheduling must be carried out at *both* high and low levels. At the low level, where every single job is assigned to a single resource, because of the *heterogeneity* of the jobs and the resources, there is a *problem of choosing the best job-resource assignment* among the different and possible ones. The solution of this problem has big influence on the

utilization of the hardware: not only on the fraction of the used resources to the available ones, but also on the efficiency with which these resources are used.

Therefore, while keeping on solving other common problems as in uniform systems, job scheduling strategies in heterogeneous systems must *focus* on the problem which is mentioned above and which derives from the *heterogeneity* of the computing systems.

In this paper we study the job scheduling problem for parallel processing in heterogeneous systems by this way. We provide a heuristic algorithm (named Objective-Oriented Algorithm) based on a strategy that has not yet been used before in job scheduling.

The paper is organized as follows: In section 2 we provide a classification for the given job scheduling problem by the heterogeneity of the computing systems; Then, in order to make the scheduling algorithm comparable with the others and available for its application in the real systems, we give section 3 for a detailed description of the computing model and the problem statement; In section 4, a quick review of related work and the key ideas of the new strategy are provided; The algorithm itself is described in section 5; Simulation results are analyzed in section 6; And finally, the conclusions are stated in section 7.

2 Influence of Systems' Heterogeneity on Job Scheduling

Before moving on to describing the computing model, we would like to make a preliminary classification of the types of the systems' heterogeneity in parallel processing (from the view of schedulers' eyes).

Suppose that at a moment in time, in a parallel heterogeneous system, there are M jobs ready to be executed and there are N resources available. The requirement of the job scheduling problem is to assign these M jobs to N resources so that the received schedule is satisfied by the following: First, the *processing requirements* of the system (e.g. the resource must have enough memory and communication capacity, which are required by the job that is assigned to this resource etc.); Second, the *optimization requirements* (e.g. minimum parallel executing time, high resource utilization etc.).

We assume that the scheduling process can be divided into two smaller ones:

- *Checking* processing requirements: from *Ri* all possible and different variants of schedules, we have to pick out the set of *Rp* variants which are satisfied by the processing requirements, but not yet the optimization requirements (OR);
- *Optimizing* by satisfying OR: from *Rp* variants we have to choose *Ro* optimum schedules (in most cases *Ro=1*) which are satisfied by the optimization requirements.

Obviously, the numbers *Ri* and *Rp* determine the scale and the complexity of the Checking and the Optimizing steps**.**

In the case, when there are two OR: (i) shortest response time for all jobs and (ii) minimum total executing time, one of the ways to achieve these two goals at the same time is to distribute the jobs to the maximal number of resources (suppose that there are *Rmm* such variants) and then choose from *Rmm* variants the best one with

minimum total execution time. Now, we see how the heterogeneity of a system (how different the jobs and the resources are) can have influence on *Ri* and *Rp*, which show us the *scale* and the *complexity* of job scheduling problem.

- In so-called "*Half-Heterogeneous*" systems, where there is no processing requirement (i.e. any assignment of a job to a resource is possible), the resources are the same (uniform) but the jobs are different [3] or vice versa. In this case, we have *Ri* = *Rp*, which means that the job scheduling problem leads straightforward to the second step: the optimization step without dealing with processing requirements. The scale of the problem is $Rp = \frac{M!}{(M-N)!}$,
- In so-called "*Completely-Heterogeneous*" systems, there is no processing requirement but either the resources and the jobs are different [8,16]. In this case, the complexity of the job scheduling problem is the same as the previous one, but the scale is different with $Ri = Rp = \frac{M!}{(M-N)!} \times N!$,
- In so-called " *Strictly-Heterogeneous* " systems, the resources and also the jobs can be different. Moreover, in this case there are processing requirements (i.e. not all assignments of jobs to resources are possible) [15]. Therefore, job scheduling problem is more complicated. It now contains two steps: checking processing requirements and optimizing the schedule. The scale is almost the same as in the previous case with $Ri = \frac{M!}{(M-N)!} \times N!$ and *Rp* =*Rmm*, where *Rmm* is the number of the variants of maximum matching for the bipartite graph of M and N nodes. Theoretically, $Rmm \in [0, Ri]$ but usually in practice $Rmm << Ri$**.** Besides, it is possible that *Rmm=0***,** which means that there is no schedule with N size (i.e. for N resources). This also means that some resources are strongly unrelated to such a kind of M given jobs.

Note that the scheduling strategies, which are for commonly-called uniform systems, belong to the first class: "*Half-Heterogeneous*" systems, where the resources are the same (uniform) but the jobs are different (even just by the amount of execution time for each job). The scheduling strategies, which are for commonly-called heterogeneous systems, actually belong to the second class: "*Completely-Heterogeneous*" systems. The third class: " *Strictly-Heterogeneous* " systems, has received the least attention because of the following reason: The parallel computing systems in this class are characterized with having not only *jobs and resources of different classes* but also some strict *processing requirements*. This kind of parallel systems (e.g. PVM) became available for common use in the real world only several years ago. One more fact is that any algorithm that works for the systems of the third class will also work for the systems of the second one, although with a little less efficiency than in its own class.

In this paper we will focus on the third class of scheduling, for " *Strictly-Heterogeneous* " systems.

3 Heterogeneous Computing Model

3. 1 Description of System's Model

The computing model for a "*Strictly-Heterogeneous*" systems, in which our scheduling algorithm works, has derived from a large parallel and distributed system (Fig. 1.) and has been studied before in [13].

In the real world, this system consists of : First, *different resources*, they are heterogeneous nodes-users U_i (e.g. different computers or processors) and the common resources CRj (e.g. the severs) ; Second, *tasks of different kinds* T_i, which come from users-nodes or from outside (e.g. from other systems). As in any parallel systems, these so-called mother-tasks have to be maximally parallelized. They are divided into son-tasks (small computing modules) which we will call jobs. These *jobs are different* as well because they come from different tasks.

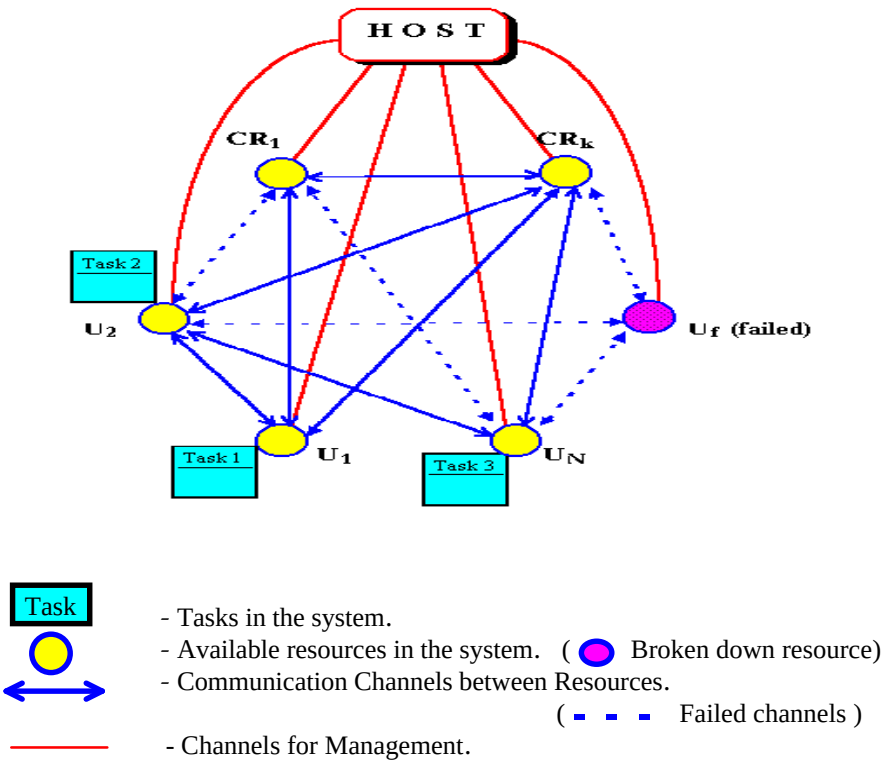

Fig. 1. *Tasks and Resources in a Parallel Heterogeneous System*.

Therefore, the given system can be represented by:

- a data set about Q *tasks* ST={T_1, ...,T_Q} with their heterogeneous processing *requirements*.

- a data set about N *resources* SR={R_1, ...,R_N} with their heterogeneous processing *capacities*;
- a data matrix about the *communication channels between the resources* MCR[1..N,1..N], where MCR[i,j] is the rate of the cost for communicating between R_i and R_j , and MCR[i,j] $\in \Re^+$. We say there is "no connection" between R_i and R_j when MCR[i,j] > Ω_0 (some given number)**.**

3. 2 General Scheduling Scheme

In order to make explicit how much and what kind of work the job scheduling problem does and where it takes place in the general scheduling process, we provide here a quick review of the general scheduling scheme. Usually, it contains the following steps:

(1) Input Tasks: First, accept tasks T_1, ...,T_Q from the users or from the outside of the system; Second, analyze tasks and prepare beforehand data for the next step.

(2) Parallelize Tasks into Jobs: Each task T_i is maximally parallelized (without the resource constraint) into jobs J^i_k , k=1,..,Ki . After parallelizing, each task can be represented by a DAG of job-nodes (Fig.2.a). Thus, we have Q graphs: VT_1, ...,VT_Q**,** where VT_1={J^1_1, .., J^1_{K1}} , ..., VT_Q={J^Q_1 , .., J^Q_{KQ}}. Then, all jobs of different tasks (i.e. of different classes) are grouped (and renamed) into S common clusters B_1, B_2, .., B_S (Fig.2.b) considering their precedence, where B_1={$J^{1,1}$, ..,$J^{1,M1}$}, .. , B_S={$J^{S,1}$,..,$J^{S,MS}$}.

(3) Prepare and send Ready Jobs to Buffer-in: A Filter, whose work is based on the rule of job precedence, chooses ready jobs from the nearest cluster B_1 and sends it into the Buffer-in (Fig.2.b). The rule of job relationship is that a job $J^{i,t} \in B_i$ is ready only when all its predecessor in cluster B_{i-1} have been executed. Besides, the jobs in all clusters B_1,B_2, .., B_S are also moved into the next cluster all the time by this rule.

(4) Schedule Jobs to Resources: At a given moment in time, there are M ready and independent jobs in the Buffer-in and N available resources in the system. The scheduler has to assign these jobs onto the resources so that the received schedule is optimum by one or some given optimization criteria.

(5) Reschedule Failed Jobs: In the case, when a resource fails during executing the job that has been assigned on it, the ID of this job has to appear in the Buffer-in again, as a ready job for the next scheduling cycle (this step may be executed by the system monitor but not by the scheduler).

(6) Output Executed Tasks: This is the reverse process of step (2). After the execution, each job is put to Buffer-out (Fig. 2.b). Then, they are collected back to their mother-tasks. A task is completely executed and removed from the scheduling system when all its son-jobs have been executed.

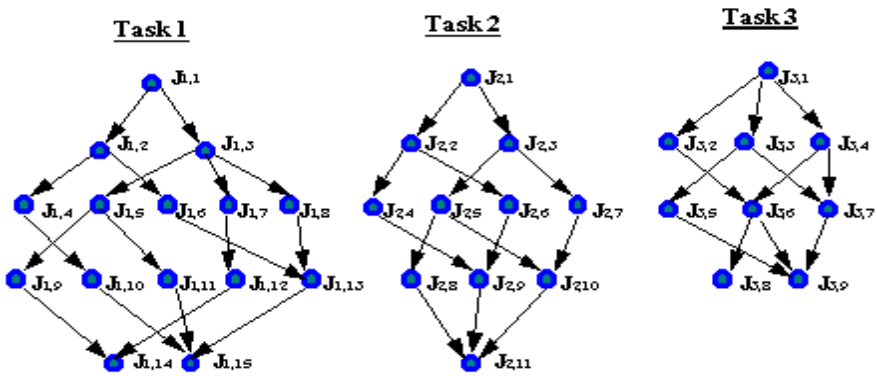

Fig. 2. a *Parallelizing Tasks into Jobs.*

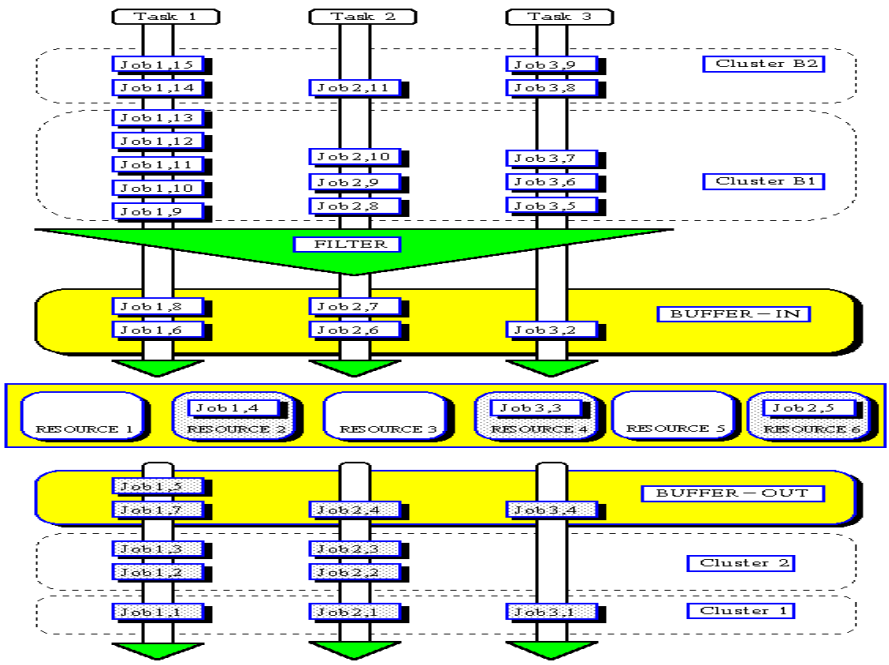

Fig. 2.b *Jobs in clusters during the scheduling process.*

3. 3 Statement for Job Scheduling Problem

As is discussed above, the scheduling process is a complex of procedure-steps. In this paper we will focus on the most important step (4) - step of scheduling jobs to resources. In more detail, it can be stated as follows:

At a moment in time, after the step 3 in the scheme above, there are:

■ N *heterogeneous resources* of the system, which are represented by a graph $G_R=(V_R,E_R,W_{VR},W_{ER})$, where:

- $V_R=\{R_1, R_2, \ldots, R_N\}$is the set of N resources-nodes (the ID), $R_i \in N|$, $i=1..N$.
- $E_R=\{E_1, E_2, \ldots, E_d\}$is the set of edges, which represents the *physical communication link between resources* $E_i = \{R_i,R_j\}$, where $R_i, R_j \in V_R$ and $0 \le d \le N^2$.
- $W_{VR}=\{ WVR_1, WVR_2, \ldots, WVR_N \}$is the set of nodes' weights, where $WVR_i =\{RE_i, RT_i\}$. For $\forall$ $i=1..N$:
  - (i) $RE_i \in \Re^+$ is the ratio that characterizes the *capacity* (e.g. the speedup, local memory) of the given resource R_i ;
  - (ii) $RT_i \in \{0,1\}$ is the *state* of the given resource (free or occupied).
- $W_{ER}=\{ WER_1, WER_2, \ldots, WER_p \}$ is the set of edges' weights and it can be represented by a matrix $RC=RC[i,j] \in \Re^+$, where $i=1..N$, $j=1..N$, and $0 \le p \le N*N$.

■ M different and independent jobs, which can be executed in parallel. They are represented by a set $V_J =\{J_1, J_2, \ldots, J_M\}$. The *heterogeneity* of each job is characterized by the data set $J_i =\{JN_i, JE_i, JL_i\}$, $i=1..M$, where:

- $JN_i \in N|$ is the ID of the job (e.g. the number).
- $JE_i \in \Re^+$ is the *work amount* for executing the given job.
- $JL_i =\{(R^1, \varphi_1), \ldots, (R^q, \varphi_q)\}$ represents the *logical communication link* of the given job J_i to the resources $\{R^1, \ldots, R^q\}$, where:

  (i) $R^t \in V_R$ $(t=1..q , q \in N|)$ is the resource which the given job need to communicate with. In the given scheduling system, this is the resource on which the predecessor-job of the given job has been executed before;

  (ii) $\varphi_t \in \Re^+$ is the data amount that is needed to transfer in communicating with R^t.

  For example, if the job $J_{1,2}$ has been executed on the resource $R^1 = R_1$ and $J_{1,6}$ on $R^2 = R_3$ then the job $J_{1,6}$ (Figure 2.a and 2.b) will have logical communication link $JL= \{(R^1, \varphi_1), (R^2, \varphi_2)\}$.
- $JP_i \in \Re^+$ is the *priority* of the job J_i if the priority system for jobs exists.

■ The *requirement* of the given problem is to find out a schedule for assigning M jobs onto N resources so that we can achieve two following optimization goals:

- minimum total actual execution time (and)
- shortest response time for all jobs.

4. Solution Basis

4. 1 Related work

It has been shown in [10] that the given problem (with the name "Assignment Problem") comes from the "Traveling Salesman Problem", which is NP-complete [9]. Moreover, the given problem, indeed, is listed in the form of the problem N43 in section A2.5. of the List of NP-complete problems in [9]. Therefore, most algorithms for solving it use heuristic or genetic approaches [7],[16],[5],[3],[19].

In solving any scheduling problem, there are two important issues that we should consider. They are: *solution quality* (how the received schedule is near to the optimum one) and *solving time* (for how long it takes to find the schedule).

In dealing with solution quality: Solution quality depends on the optimization criteria and the scale of the optimization area. Usually, there are three main kinds of optimization criteria :

- Focus on the executing time of jobs and not consider communication costs (as in most of balancing algorithms) [8].
- Focus on the minimization of communication costs [7] (e.g. using "critical path" in clustering algorithms[20]).
- Focus on other parameters (e.g. response time or other time constraints in the Real-time systems[15]).

After choosing criteria, the optimization can be carried out in two ways:

- Through local minimization [15],[6]. This is simpler than the next way and it requires less information (in local scale). However, because it is local, it has to be carried out several times during a scheduling cycle.
- Through global minimization [7],[8]. This is more complex than the first way and it requires more information about jobs, resources, system performance (in global scale). Therefore, in practice, the algorithms usually are simplified in order to decrease the scheduling time.

To achieve the desired results with more than one optimization criterion, the scheduling is sometimes carried out by a combination of the above discussed ways.

In dealing with solving time: The requirement of the solving time depends on the scheduling type. Static scheduling algorithms can have a long solving time while the Dynamic ones always have a short solving time.

As is said above, because of the complexity of the problem, most of the scheduling algorithms are heuristic or genetic. In the algorithms that have been published recently, a genetic method called Simulated Annealing (SA) is used very popularly [7],[16],[11],[18]. This method is good because of its flexibility. By resetting the values of the parameters for the simulation (the initial and freezing temperatures, the ratio for decreasing temperatures) we can have many kinds of schedulers, which are different by the correlation between solution time and solution quality as follows:

- From the fastest scheduler, which gives a schedule with a random quality for the minimum solution time;

- To the slowest one, which checks all possible variants of schedules and gives us the exact solution (real optimum schedule).

Usually, the simulation parameters are set so that the result is in the middle between these two extremists. However, the solution time for achieving an acceptable-optimum schedule is too long, especially when the problem size (M,N) is large, and also at the same time, the solution quality is unpredictable.

4. 2 New Approach with Hungarian Method

As it has been reviewed above, in order to achieve two conflicting goals: short scheduling time and good quality of the schedule, we always have a trade-off between solving time and solution quality. The point is how to achieve the "golden mean".

Scheduling strategy for dealing with solving time: For the system that is described above, our aim is to develop a balanced algorithm, which gives us a schedule of good quality for an acceptable solving time.

In order to see explicitly the difference of our approach from the existing ones, let us analyze again (but now from the view of the strategists' eyes) the above approach of scheduling by using simulated annealing (SA). Suppose that all possible variants (*Ri*) of the schedules can be set as the "balls" in a "box". The variants-balls are located in chaos. Among them, there is a real optimum one that has to be found. Now, see how it is found by SA and by our algorithm using Hungarian method, which is described in [10],[12],[1].

In SA, all the variants are put in the Markovian chain as if all the balls are connected each to another with a visual "thread" (this is not shown in Fig.3.a). The search is started with a random variant-ball Vs (which is put in the "pocket") and is guided by this thread. Continuing the search, the next ball is compared with the one in the pocket. If the next one is better than the one in the pocket then it will occupy the pocket, and so on. After a given number of steps with the last-checked ball Vf, the variant-ball in the pocket is regarded as the optimum one of all the balls in the box. Usually, the Markovian chain is built so that the "thread" is spread over all the box in order to avoid local minimum. However, still, if the real optimum variant-ball was not in the searched area (the red line in Fig. 3.a) then the received variant, obviously, is not optimum.

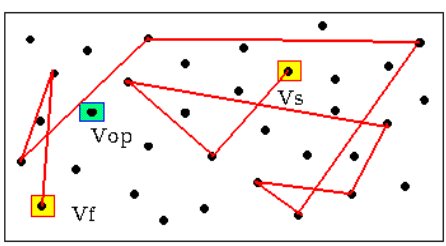

Fig. 3.a *Scheduling strategy in SA*

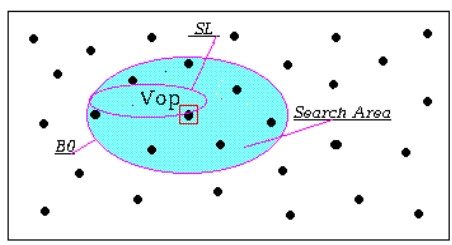

Fig. 3.b *New scheduling strategy.*

The key idea of the new approach is to detach from all possible variants (the box) a zone, which contains exactly the real optimum variant *Vop* (for the given optimization criteria) and is much smaller than the box (Fig.3.b). Then, the search is carried out *only* in this area until it reaches the real optimal variant *Vop*. Therefore, we will receive the exact solution while the solving time is much less than it is in SA for finding a solution of the same quality.

Moreover, using the Hungarian method (which will be described in detail later) for such an approach allows us to carry out not a random search (as it is in SA) but a so-called "*objective-oriented search"* in the chosen area, where the objective is the real optimum variant *Vop*. That is why we name the algorithm "Objective-Oriented" (OOA). The selection of the Hungarian method is not by chance but is based on a careful investigation which is studied in [14].

Scheduling Strategy for dealing with solution quality: Now we have to determine the optimization criterion and the scale of optimization area.

- Heterogeneous systems usually are distributed. Therefore, we think that it is necessary to consider communication costs in scheduling for such systems. To escape the conflicting goals: (i) minimizing execution time (by choosing separately the most fitting job-resource pairs); (ii) minimizing response time of jobs (by maximally parallelizing them on the maximum number of resources); (iii) minimizing communication costs (by decreasing the number of the used resources), we use the following balanced optimization scheme. The optimization is carried out by two steps:
  (1) maximize the number of resources for executing the given jobs as far as possible (for achieving minimum response time);
  (2) choose the best schedule with the minimum total weight that is determined by an estimating function (which characterizes the execution time of jobs including communication costs).
- Because the Hungarian method works very efficiently when the problem size is 20-100, we propose two variants of optimization:
  - First variant; when M>100 , the jobs are randomly gathered into a group of N jobs. Then the local optimization is carried out for N jobs and N resources. The size of the optimization zone n = N.
  - Second variant; when M<100, all the jobs are in a global optimization with K groups of N resources, where $K \in N|$ and K>[M/N]. The size of the optimization area n = M.

4. 3 Algorithm Basis

With the chosen strategies for scheduling, which are mentioned above, there are three important steps which now will be studied in detail:

- Forming the "box".
- Determining the searching zone.
- Building a rule-guide for the objective-oriented search.

A. Box Forming

This means the way to represent the data for scheduling. For the initial data (which is described in 3.3.), we have: a set of M independent jobs with their heterogeneous processing requirements; a graph of N resource-nodes with their heterogeneous processing capacities.

Now, in order to form the box, we have to reform these two separate data sets into a form that represents directly the relationship between jobs and resources: a matrix of size M*N and which we will call Job-Resource matrix (JR). Each element JR[i,j] of this matrix is the weight of the assignment of the job *i* onto the resource *j*. These weights are determined by an optimization function Δ(i,j). The formation of this function is based on the optimization criteria that is determined in 3.3.

Suppose that the jobs and the resources are identified respectively by two sets $V_J=\{J_1, J_2, \ldots, J_M\}$, $V_R=\{R_1, R_2, \ldots, R_N\}$. In general, the optimization function can be determined as the following:

$$\Delta(J_i, R_j) = \delta_{i,j} = \prod_{k=1}^{K} P_k^{i,j} \times \prod_{x=1}^{H} C_x^{i,j} \times \sum_{y=1}^{G} L_y O_y^{i,j} \qquad (1)$$

Where,

- $\prod_{k=1}^{K} P_k^{i,j}$ is called absolute *priority* of assignment (J_i,R_j). It is calculated by multiplication all K relative priorities $P_k^{i,j} \in \Re^+$ which are derived from different factors (e.g. the permitted waiting time of the given job, the desirable usage frequency of the given resource).
- $\prod_{x=1}^{H} C_x^{i,j}$ is the final result of analyzing all H *processing requirements* (e.g. the requirements about memory, speedup...), where $C_x^{i,j}$ is the degree of satisfying the processing requirement x (x = 1..H) for the assignment (J_i,R_j), $C_x^{i,j} \in \{0,1\}: C_x^{i,j} = 1$ if the resource R_j is satisfied by a requirement x of the job J_i, otherwise, $C_x^{i,j} = 0$.
- $\sum_{y=1}^{G} L_y O_y^{i,j}$ is the final result of analyzing all G *optimization requirements*, where $O_y^{i,j} \in \Re^+$ is the degree of satisfying the optimization requirement y for the assignment (J_i,R_j); $L_y \in \Re^+$ is the weights assigned to the optimization criterion y.

For the given computing system with the given optimization criteria and according to the scheduling strategy for dealing with solution quality in section 4.2., the estimating function (which calculates the weight of the assignment of a job J_i onto a resource R_j, $R_j \in V_R$ and $J_i \in V_J$) should be determined as follows:

- To reduce the complexity of the problem, we suppose that $\prod_{k=1}^{K} P_k^{i,j} = \rho_i$ (2), where $\rho_i = JP_i$ is the priority of the job J_i (e.g. it can be formed by the time Tw_i, for which the job J_i has been waited in the system).
- For simplicity, we suppose that there is only one *processing requirement* (H=1): $\prod_{x=1}^{H} C_x^{i,j} = C^{i,,j}$ which is determined by comparing the physical communication links of the resource R_j: $E_R=\{E_1,E_2,..,E_d\}$in the resource graph with the logical communication links of the job J_i: $JL_i=\{(R^1,\varphi_1),..,(R^q,\varphi_q)\}$. Recall that $R^t \in V_R$ (t=1..q) is the resource which the given job needs to communicate with.

  The rule for determining them is: $C^{i,j} = \prod_{t=1}^{q} CC_t^{i,j}$ (3)

  where, $\forall$ t=1..q : $CC_l^{i,j} = 1$, if $(R_j, R^t) \in E_R$;

  $CC_l^{i,j} = 0$, if $(R_j, R^t) \notin E_R$;

- The third part of (1) is determined as the follows:

  $\sum_{y=1}^{G} L_y O_y^{i,j} = 1/Te_{i,j} + 1/Tc_{i,j}$, where

- $Te_{i,j}$ is the *executing time* for the job Ji on the resource Rj. If the *amount of work* for executing the given job $JE_i = \varepsilon_i$; the *realization ratio* of the resource $RE_j = k_j$, then it can be determined by the following: $Te_{i,j} = \varepsilon_i * k_j$;
- $Tc_{i,j}$ is the communication costs for the assignment (J_i,R_j). If the list of the resources which the job requires to communicate with is $JL_i = \{(R^1, \varphi_1), \ldots, (R^q, \varphi_q)\}$, and the communication ratio between the resources R_j and R_l (l=1..N) is $RC[j,l]=\beta_{j,l}$; then we have $Tc_{i,j} = \sum_{l=1}^{q} (\varphi_l * \beta_{j,l})$

  Therefore, $\sum_{y=1}^{G} L_y O_y^{i,j} = 1 \div (\varepsilon_i \times k_j) + 1 \div \sum_{l=1}^{q} (\varphi_l * \beta_{j,l})$ (4)

From (1),(2),(3), and (4) we have:

$$\Delta(J_i,R_j) = \delta_{i,j} = \rho_i \times C^{i,j} \times [1 \div (\varepsilon_i \times k_j) + 1 \div \sum_{l=1}^{q} (\varphi_l * \beta_{j,l})] \quad (5)$$

Obviously, $\delta_{i,j} \geq 0 \ \forall$ i=1..N , j=1..M . Therefore:

$inf(\Delta(J_i,R_j)) = 0$.

We will assume that there is no physical communication link between resources R_j and R_i when $Tc_{i,,j} = \sum_{l=p}^{q} (\phi_l * \beta_{j,l}) > \lambda_o$, where λ_o is some given number. This is the threshold for determining if the communication between two resources is possible or

not. According to this value, we have some value δ_{exist} which determines if the discussed assignment is possible or not.

The executing time $Te_{i,j}$ must have some lower limit T^o, then the highest limit for $\Delta(J_i,R_j)=\delta_{i,j}$ can be determined as the following:

$$sup\ (\Delta(J_i, R_j)) = (\mu_{0j} \times \rho_{0j}) \times [1/T^0 + 1/\lambda_0] = \delta_{max}.$$

For applying the Hungarian method in the next step, we have to rebuild a new optimization function:

$$\Psi(J_i, R_j) = \delta_{max}. - \Delta(J_i, R_j) = \psi_{i,j} \qquad (6)$$

Therefore, finally, for all elements of the matrix JR[i,j]=$\psi_{i,j}$, i=1..M, j=1..N, we have : $0 \le \psi_{i,j} \le \delta_{max}$ and the value $\psi_{exist} = \delta_{max} - \delta_{exist}$ as the threshold for determining if the assignment(Ji,Rj) is possible or not. The size of the box *n* is determined as in section 2 (based on the value of M and N) and according to the optimization area, which is determined in section 4.2.

For the example that is given in Fig. 2.b., the "box" or the Job-Resource matrix JR will have the size M=5 and N=3 as in Figure 4. The values of matrix elements are supposed to be the results of the calculation by the formulas (5) and (6) with some data which are given in section 3.3 for the jobs $J_{1,6}$, $J_{1,8}$, $J_{2,7}$, $J_{2,8}$, $J_{3,2}$ and for the resources R_1,R_3,R_5 .

		RESOURCES				
J		$J_{1,6}$	$J_{1,8}$	$J_{2,7}$	$J_{2,6}$	$J_{3,2}$
O	R_1	**18**	**45**	**7**	**47**	**6**
B	R_3	**6**	**3**	**39**	**61**	**4**
S	R_5	**75**	**9**	**3**	**6**	**58**

Fig. 4. *BOX with Matrix JR (δ_{max}.=80; ψ_{exist} =20;)*

However, for a good explanation in the next sections, we suggest to study the searching technique with such a "box" (the Job-Resource matrix JR) as in Fig.5.a, where M=N=n=6.

B. Determining searching zone

After determining the job-resource matrix JR as the "box" (suppose that it is in Fig. 5.a), the next and important step is to determine the zone for searching the real optimum variant of possible schedules. For further study, we provide some definitions about the assignments and the schedules as follows:

Definition 1: There are two given sets: $V_J=\{J_1, J_2, \ldots, J_n\}$ and $V_R=\{R_1, R_2, \ldots, R_n\}$. The pair $a=(J_i,R_j)$ is called *Assignment* of the job $Ji \in V_J$ onto the resource $Rj \in V_R$. Each assignment has its *weight*, which is determined as $\Psi(a)=\Psi(Ji,Rj)$, where Ψ is the function determined by (6).

Definition 2: For two given sets: $V_J=\{J_1, J_2, \ldots, J_n\}$ and $V_R=\{R_1, R_2, \ldots, R_n\}$, a set $A=\{a_1,a_2, \ldots, a_{n*}\}= \{(J^1, R^1), (J^2, R^2), \ldots, (J^{n*}, R^{n*})\}$ is called *Maximum matching* for such jobs and resources if:

- $\forall$ i=1.. n*, $\Psi(a_i)=\Psi(J^i,R^i)<\psi_{exist}$.
- $\forall$ i=1..n*, $R^i \notin AR \setminus R^i$, $J^i \in AJ \setminus J^i$, where $AR=\{R^1, R^2, \ldots, R^{n^*}\}$, $AJ=\{J^1, J^2, \ldots, J^{n^*}\}$.
- n* is maximized as far as it can be.

Definition 3: A maximum matching A* with the size n* for two given sets: $V_J=\{J_1, J_2,\ldots, J_n\}$ and $V_R=\{R_1, R_2, \ldots, R_n\}$is a possible *Schedule* for such jobs and resources if n* = n. The weight of the schedule then is determined by: $D(A^*) = \sum_{i=1}^{n} \Psi(a_i^*)$.

Definition 4: Suppose that $Xp=\{A_1, A_2, \ldots, A_p\}$, $p \in N|$ is the set of all possible schedules. A schedule A* is the *real optimum variant* of possible schedules if :

$$D(A^*) = \sum_{i=1}^{n} \Psi(a_i^*) = min\{D(A_1), D(A_2), \ldots, D(A_p)\} = \min_{j=1}^{p}\{D(A_j)\}$$

According to the Hungarian method, the searching zone SZ is limited by so-called minimum assignments a*=(J*,R*) which have the minimum weights Ψ(J*,R*) in comparison with other assignments in the same rows or in the same columns of the matrix JR with size n. These minimum assignments create the first bound for SZ (we call it B0), which can be found in the following way:

For i=1.. n, j=1..n, *if* :

- $(\Psi(a^*)=\Psi(J^*,R^*)<\psi_{exist})$ and
- $(\Psi(a^*)=\min_{i=1}^{n}\Psi(J_i,R_j)$ or $\Psi(a^*)=\min_{j=1}^{n}\Psi(J_i,R_j))$

then a*= (J*,R*) ∈ B0.

Therefore, the bound B0 of the searching zone SZ is determined by a set of minimum assignments $A_b = \{a_1^*, a_2^*, ..a_{b0}^*\}$, $b0 \in N|$.

In the Hungarian method, the bound is marked by making the minimum assignments A_{b0}, which are on it, become the so-called zero assignments. It is obvious that if we subtract the same number S from all elements of a row (or of a column) in the matrix JR then the minimum assignments still remain as the minimum ones. In other words, if we subtract the weight of the minimum assignment from all elements, for each row and each column, we will still have the same bound B0 with the same minimum assignments $A_b = \{a_1^*, a_2^*, ..a_{b0}^*\}$. The only difference is that these assignments will have zero weights.

In summery, the first bound B0 of the searching zone is a set of the zero assignments $A_b=\{a_1^*, a_2^*, ..a_{b0}^*\}$ as in Fig.5.b. (for the "box" in Fig.5.a).

		RESOURCES					
		1	*2*	*3*	*4*	*5*	*6*
JOBS	*1*	**8**	**5**	**71**	**7**	**6**	**9**
	2	**6**	**3**	**9**	**1**	**47**	**7**
	3	**5**	**59**	**3**	**6**	**8**	**8**
	4	**9**	**1**	**8**	**95**	**5**	**4**
	5	**1**	**2**	**1**	**80**	**2**	**3**
	6	**69**	**70**	**3**	**5**	**3**	**10**

Fig.5.a *BOX with Matrix JR*

		RESOURCES					
		1	*2*	*3*	*4*	*5*	*6*
JOBS	*1*	**3**	**0**		**2**	**0**	**2**
	2	**5**	**2**	**8**	**0**		**4**
	3	**2**		**0**	**3**	**4**	**3**
	4	**8**	**0**	**7**		**3**	**1**
	5	**0**	**1**	**0**		**0**	**0**
	6			**1**	**3**	**0**	**6**

Fig.5.b *B0 of "0"s in Matrix JR* *(n=6; δ_{max}.=100; ψ_{exist} =30;)*

C. Objective-Oriented Search

After determining the bound of the searching zone, we have to carry out the last and the most important step - to search the optimum variant of the schedules in the determined searching zone. The search starts from the found bound B0, which is marked by zero assignments, and goes to the so-called current "Searching Line" (SL) by the following steps:

(1) First, check if the optimum variant Vop is on SL or not (for the first time, SL is the bound B0). That means to check if there is a schedule among the minimum assignments $A_{b0}=\{a_1^*,a_2^*,..a_{b0}^*\}$ on B0. If so, go to (2). If not, go to (3).

According to definition 3, we have a schedule on SL if there are such n minimum assignments $A^*=\{a_1^*,a_2^*,..a_n^*\} \subseteq A_{b0}$ that A* is the maximum matching for the given jobs $V_J=\{J_1, J_2, \ldots, J_n\}$ and the given resources $V_R=\{R_1, R_2, \ldots, R_n\}$.

Suppose that (for these two sets), we have $A^*=\{(J^1, R^1), (J^2, R^2), \ldots, (J^n, R^n)\}$ and $AR=\{R^1, R^2, \ldots, R^n\}$, $AJ=\{J^1, J^2, \ldots, J^n\}$. By definition 2, A* is the maximum matching if:

- $\forall$ i=1.. n, $\Psi(a_i^*)=\Psi(J^i,R^i) < \psi_{exist}$.
- $\forall$ i=1..n, $R^i \notin AR \setminus R^i$, $J^i \notin AJ \setminus J^i$.

In the given example in Fig.5.b, the maximum matching for the minimum assignments (the zeros) on B0 is {(5,1),(1,2),(3,3),(2,4),(6,5)} in Fig.6.a. It has the size 5 which is less than the matrix size n=6. Therefore, in this case there is not any schedule in the bound B0. We go to (3).

(2) If any schedule is found, it is the real optimum variant Vop. The search has to be stopped.

(3) If we can not find any schedule from the first-minimum assignments $A_{b0}=\{a_1^*,a_2^*,..,a_{b0}^*\}$, this means there is no real optimum variant Vop on SL. Therefore, the search must be continued in the searching zone. The new SL inside the search zone SZ is found from the current SL by the procedure of Making New Zeros in the Hungarian method [10] as the set of second-minimum assignments (Fig.7.a). Then, go to (1) and repeat the same procedures for the new SL as for the current ones.

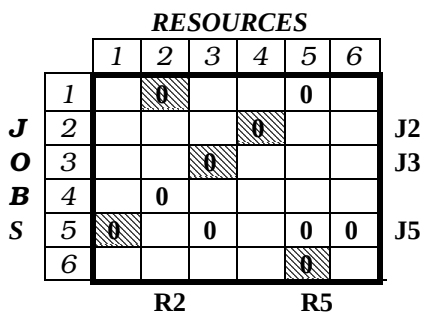

Fig. 6.a *Maximum matching and Zero Lines for B0 of "0"s*

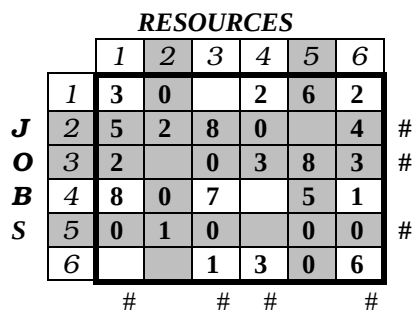

Fig.6.b *Marked Columns and Rows in JR*

The procedure of Making New Zeros contains the following steps:

1. Find the Zero Lines. This is the minimum set of lines that cover all the current zeros. For the given example, the Zeros Lines are J2,J3,J5, R2, and R5 (Fig. 6.a).
2. Find the minimum value of the elements which are not on the Zeros Lines. For the given example, these elements are the clear cells in Fig. 6.b: (1,1), (1,3), (1,4), (1,6), (4,1), (4,3), (4,4), (4,6), (6,1), (6,3), (6,4), (6,6), and the minimum value of them is 1.
3. Subtract this minimum value from all the elements of the columns that are not the Zeros Lines (R1,R3,R4,R6). Then add this minimum value to all the elements of the rows that are the Zeros Lines (J2,J3,J5). In the given example, these columns and rows are marked with # (Fig.6.b).

After doing so we receive the new set of zero elements (Fig.7.a), which indeed is the set of second-minimum assignments. In this way, the new SL for further searching is found. In the given example, the new SL are the set of the new zero elements in Fig.7.b. According to the procedure of searching, which is described above, we go to (1). In checking SL, we find the maximum matching A*={(1,2), (2,4), (3,3), (4,6), (5,1), (6,5)}with size 6 (equal to the size of the matrix JR) for the new zero assignments in Fig.7.b. Therefore, we go to (2) and finish the search.

RESOURCES

		1	2	3	4	5	6
	1	**2**	**0**		**1**	**0**	**1**
J	2	**5**	**3**	**8**	**0**		**4**
O	3	**2**		**0**	**3**	**5**	**3**
B	4	**7**	**0**	**6**		**3**	**0**
S	5	**0**	**2**	**0**		**1**	**0**
	6			**0**	**2**	**0**	**6**

RESOURCES

		1	2	3	4	5	6
	1		**0**			**0**	
J	2				**0**		
O	3			**0**			
B	4		**0**				**0**
S	5	**0**		**0**			**0**
	6			**0**		**0**	

Fig.7.a *New SL of "0"s in Matrix JR* **Fig.7.b** *Vop on SL is found in Matrix JR*

In this way, by applying the Hungarian method, we receive the schedule A*. This is the optimum schedule for the given jobs V_J={1,2,3,4,5,6} and given resources V_R = {1,2,3,4,5,6}, whose relationship has been estimated by the formulas (5) and (6), and is given in the matrix form in Fig.5.a.

The found schedule A* is optimum because: (i) the response time is 0 for each job (only one job is assigned to one resource and therefore no job has to wait); (ii) the total processing time (including execution time and communication time) for all jobs is minimum: D(A*) = Ψ(1,2)+ Ψ(2,4)+ Ψ(3,3)+ Ψ(4,6)+ Ψ(5,1)+ Ψ(6,5) = 5+1+3+4+1+3 =17 (from Fig.5.a)

Note that there are two conditions for finding Vop: First, there is a set of the minimum assignments; Second, there is a schedule (a maximum matching) in this set. In the steps described above, the search is continued not randomly inside the whole area of SZ but only with the assignments on the line SL which characterizes the first

condition for having Vop. Therefore, *the search is oriented to Vop* all the time, unlike the random search in the simulated annealing method. And that is why we call the algorithm objective-oriented (OOA).

5 Objective-Oriented Algorithm

5. 1 Description of Data Input and Data Output

The input data for OOA is the needed data for forming matrix JR. The input data comes from:

- ***Buffer-in:*** For characterizing the *states of M jobs*, which are in the buffer-in at the given moment in time, there are 3 data sets. According to sections 3.3. and 4.3., they are:
  - BIN.E[i]= ε_i (execution time),
  - BIN.LR[i]={Rp, .. , Rq}, BIN.LC[i]={ϕ_p , .. , ϕ_q} (communication requirements about the communicating resources and data transfer amounts),
  - BIN.P[i]=ρ_i (priority).

  For all: i=1..M.
- ***Monitor:*** For characterizing the *states of R resources* in the system, there are 2 data sets (according to sections 3.3. and 4.3.) :
  - SIR.TR[i]=0(if resource is free) or 1(if otherwise),
  - SIR.J[i]=J_i (the ID number of the job that is executing on i resource),

  (Suppose that there are N free resources at the given moment in time).
  For all: i=1..R.
- ***System archive:*** For characterizing the *capacity of the resources* and the *physical links* between them, according to sections 3.3. and 4.3., there are 2 data sets:
  - RC[j,h]=$\beta_{j,h}$, where j=1..R, h=1..R (ratios of resource connection),
  - RE[j]=k_j , where j=1..R (ratio of resource performance for job execution).

The output data from OOA is a schedule for M jobs and N resources, which has the following form: SCH.J[i]=J_i and SCH.R[j]=R_j where i=1..M, j=1..N.

The input data above is considered for the general case, when the schedulers try to use all *possible knowledge* about the system. They are the knowledge about the jobs (work amount), the resources (capacities and resources' network), and also the historical profile (with which resources the latest jobs have been executed).

5. 2 Algorithm Description

The steps of OOA are executed in the following order (Fig.8): The needed data for OOA is inputted from the buffer-in (Fig.2.b), the monitor and the system archive as is described above. The first step of OOA is to determine the number N of the free resources and the number M of the ready jobs. Then, the optimization scale *n* is

determined by the rule in 3.3. The second step is to form the matrix JR of size *n* by the formulas (5) and (6) in 4.3.A. The third step is to determine the bound B0 of the searching zone SZ as is described in 4.3.B, the bound B0 is marked as the current searching line SL. Then, in the fourth step, the search for the real optimal variant of schedules Vop is carried out on this SL. If Vop is found, the search is stopped and we output the found optimal schedule Vop. Otherwise, do the fifth step, which determines a new searching line SL (as is described in 4.3.C) and then go back to the fourth step and so on, until Vop is reached. Finally, when the optimal schedule Vop is found the search is finished.

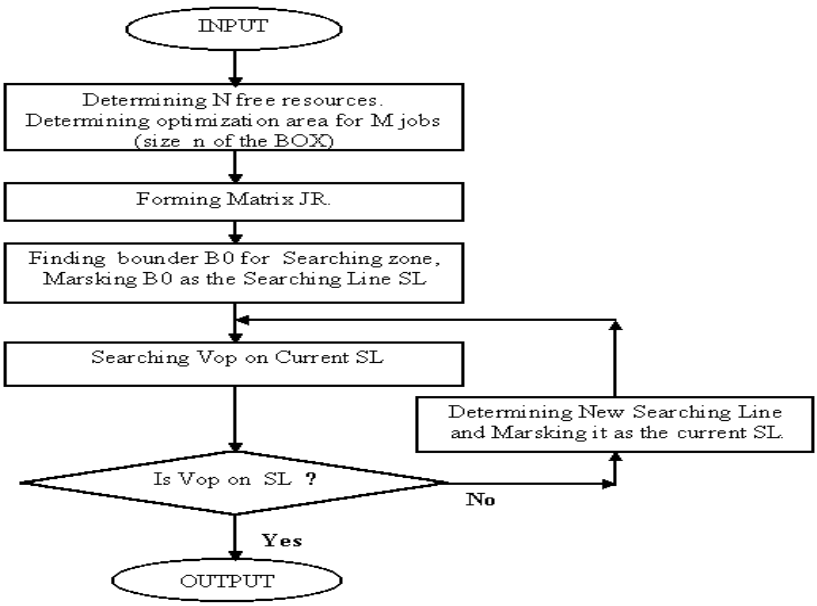

Fig.8. *Objective-Oriented Algorithm.*

The time complexity for the Hungarian method is $O(n^3)$, plus the time complexity for forming matrix JR, which is n^2. Therefore the time complexity for OOA is $O(n^3+n^2)$. However, if we use algorithm AMA which is described in [14] for searching Vop on SL the time complexity for OOA will be $O(n(E+n\log n)+n^2)$, where E is the maximum number of zero assignments on SL in the matrix JR, therefore $n \le E \le n^2$. Since the time complexity for forming the initial data usually is not taken into account, and the matrix JR is only the initial data for the job scheduling problem. Therefore, actually the time complexity of OOA is $O(n(E+n\log n))$.

In addition to this theoretical estimation that is considered for the *worst case*. We note that the scheduling algorithm OOA is dealing with the data in the matrix form all the time. Therefore this algorithm can be *parallelized* as well. The time complexity of the parallelized OOA then might be around $O(E+n\log n)$.

6 Analysis of Simulation Results

Besides the theoretical analysis, the practical analysis that is based on the simulation results is another way to show the advantage of a new algorithm and to examine its performance in comparison with the existing ones.

6. 1 Simulation Issues

An algorithm with the random scheduling and an algorithm using SA technique [16] are executed together with OOA to compare their performance. The criteria of the comparison are: (i) the solving times (the time for finding the schedule) and (ii) solution quality of the received schedules (the processing time for executing M jobs with N resources by the found schedule in the given system, which is called the length of the schedule).

The simulated computing model is as in 3.3.. The input data is formed of the so-called basic data sets (BDS) of M jobs and N resources according to 5.1. There are two factors that have influence on the solving time and the solution quality:

- The size *n* of the system which is characterized by M and N.
- The heterogeneity H_{et} of the system which is characterized by how different the elements in the matrix JR can be. H_{et} is the ratio (in %):

$$H_{et} = Nd / Na$$

where, *Nd* is the number of the different elements in JR; *Na* is the number of all elements in JR (n^2).

To illustrate the efficiency of OOA, we use 50 random BDS, where the number of jobs $M \in [20,120]$; the number of resources $N \in [3,30]$. The algorithm using SA technique is characterized by the following parameters: Initial temperature Ti=1000; Freezing temperature Tfr=0.01; and the rate of the temperature decreasing λ=0.999.

6. 2 Experiments and Results

First, the problem size n is changed from 3 to 30 while the heterogeneity is fixed. For each given value n, we investigate the solving times and the solution quality of Random, SA, and OOA. The simulation results are shown in Fig. 9.a and Fig. 9.b. They show us that the quality of the received schedule by OOA is much better than the ones by SA and, of course, by Random scheduling (Fig. 9.b.). Although the solving time of OOA is more than the one of Random scheduling, it is much less than the solving time of SA (Fig. 9.a.). We will have similar graphics if the problem size is increasing to more than 30. We do not provide the graphics of comparing with larger size than 30 because of the long solving time of SA. Indeed, the larger the problem size (n) is, the better OOA is, in both scheduling issues: solution quality and solving time.

Second, while the problem size is fixed, in our case *n* = 30, but the heterogeneity is changed and $H_{et} \in [10\%,90\%]$, we study the solving times and the solution quality of Random, SA, and OOA. The simulation results show us that the solving time of all algorithms (Random,SA, and OOA) do not depend on system heterogeneity.

Therefore we do not provide this graphic in the paper. However, the heterogeneity of the systems has influence on the solution quality. The graphics in Fig. 10. shows us that the more heterogeneous the systems are, the more efficiently OOA works, compared with SA and Random.

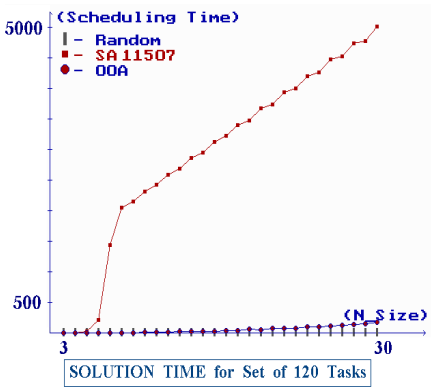

Fig.9.a *Investigation of Solving Time*

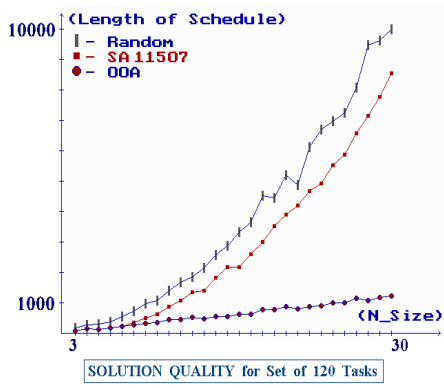

Fig.9.b *Investigation of Solution Quality*

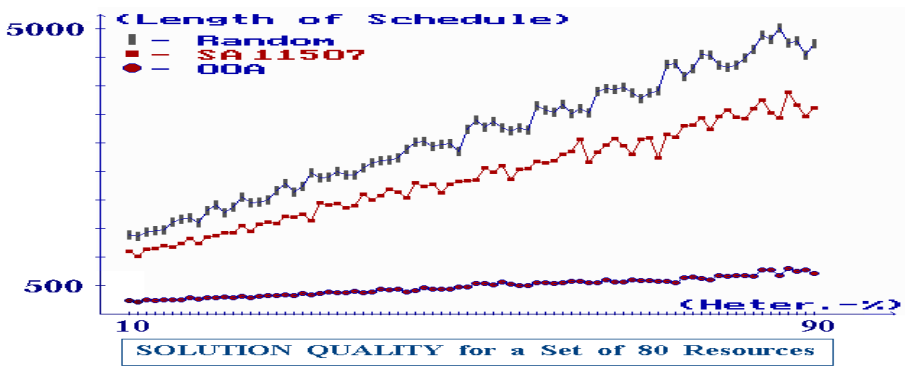

Fig.10. *Investigating the Dependency of Solution Quality on the Heterogeneity of the System.*

6. 3 Analysis

The simulation results have proved our theoretical analysis which is discussed in the sections 4 and 5. In comparing with the existing methods, the scheduling quality of OOA is much better, and the scheduling time of OOA is decreased extensively. That is because the search area in OOA is reduced substantially. This efficiency of OOA is shown while both factors: the problem size and the heterogeneous of the systems are changing.

In Fig.9.b, the results show us that SA does not do much better than the random one in terms of schedule quality. That is because of the chosen parameters for the simulated annealing. With the given parameters above, we have a SA scheduler that

gives us the "best" schedule after $11507 \approx 10^4$ times of changing the temperature (Tct). In the meantime, the real best schedule can be found, theoretically, only after 120!*30!/(120-30)! Tct, which is more than 10^{120} Tct (!). That is why the result of SA does not look very good.

The quality of SA scheduler can easily be improved by changing the parameters for the simulated annealing so that the number of Tct is increased, for example SA with 10^{10}. However, the scheduling time of SA then is increased too. And when the scheduling time of SA is too long it makes the scheduling times of Random and OOA (which are much less than the SA's one as in Fig.9.a.) become closer each to other and also closer to 0. Since such a picture of relationship between scheduling times of Random and OOA are not real, we think that it is better to show the results of SA with 11507 Tct than the results of SA with more Tct .

7. Conclusions

We have presented a new approach to solve the problem of job scheduling for parallel processing in *heterogeneous* systems. The new approach, using the Hungarian method, provides a quick and objective-oriented search for the best schedule by two optimization goals: (i) minimum total execution time including communication costs and (ii) shortest response time for all jobs. In addition, using the modified algorithm (which has been provided in [14]) for this method, the solving time is decreased to O(n(E+nlogn)).

Note that this time complexity is for finding the *real* optimum schedule by the given optimization criteria and can be *reduced* by parallelizing OOA. Besides, as is shown in the simulation results, the actual scheduling time in practice is almost linear. The explanation of this fact is that the modified algorithm [14] which is applied for searching Vop on current SL (in 4.3.) is a very good fit for this kind of searching.

The advantages of the proposed algorithm OOA are: (i) achieving a good balance of several conflicting optimization goals: minimum execution time, minimum communication costs, shortest response time; (ii) scheduling for a relatively short time; (iii) flexibility in application because scheduling is carried out in a general computing model, where there is no requirement for the system architecture; and finally, (iv) especially efficient for so-called Strictly-Heterogeneous systems, where either the jobs and the resources can be very different, even unrelated.

In future work, there are two ways by which we can decrease the solving time or increase the solution quality of the proposed algorithm OOA: (i) combine OOA with SA technique for creating a new algorithm that has less solving time while keeping the control of the solution quality so that it is acceptable; (ii) simplify the local optimization in OOA and add a simple optimization function for global optimization (e.g. by considering workload balance at the high level).

Acknowledgment

We would like to thank the anonymous reviewers for their very helpful comments that improved both the content and the presentation of this paper.

References

[1] C. Berge, *Theorie des graphes et ses application*, Dunod, Paris, 1958.

[2] J. Blazevicz, M. Drozdowski, G. Schmidt, and D. De Werra, "Scheduling independent multiprocessor tasks on a uniform k-processor system", *Journal of Parallel Computer* 20, pp. 15-28, 1994.

[3] T. Bultan and C. Aykanat, "A new mapping heuristic based on mean field annealing", *Journal of Parallel and Distributed Computing*, Vol. 16, N4, December 1992.

[4] T.L. Casavant and J.G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems", *IEEE Trans. Softw.Eng.14* , pp. 141-154, 1988.

[5] K. Efe, "Heuristic models for task assignment scheduling in distributed systems", *IEEE Computer*, June 1982.

[6] H. El-Rewini and T.G. Lewis, "Scheduling Parallel tasks onto Arbitrary Target Machines", *Journal of Par. and Distr. Com.*,Vol.9, pp. 138-153, 1990.

[7] A. A. Elsadek and B.E Wells, "Heuristic model for task allocation in a heterogeneous distributed systems", *Proceeding of PDPTA'96*, California USA, Vol.2, pp. 659-671, August 1996.

[8] R. F Freund, B.R.Carter, Daniel Watson, et al., "Generational Scheduling for Heterogeneous Computing Systems", *Proceeding of PDPTA'96*, California-USA, Vol.2, pp769-778, August 1996.

[9] M.R. Garey and D.S. Johnson, *Computer and Intractability-A guide to the Theory of NP- completeness*, Freeman New York , 1979.

[10] A. Kaufmann, *Introduction a la combinatorique en vue des aplications*, Dunod, Paris, 1968.

[11] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by simulated annealing", *Journal of Science*, Vol.220, N.4589, May 1983.

[12] X. Papadimitry, K. Stayglitsh, *Combinatory optimization, algorithm and complexity*, Moscow- Mir, 1985.

[13] Hanh H. Pham and Valery Simonenko, "A new algorithm and simulation for task assignment in parallel distributed systems", *Proceeding of the 11th European Simulation Multiconference '96*, Budapest- Hungary, pp. 95-99, June 1996.

[14] Hanh H. Pham and Valery Simonenko, "Adaptation of algorithms for Job-Resource Assignment in Heterogeneous Distributed Systems", *Proceeding of PDPTA'96*, California-USA, Vol.2, pp. 835-845, August 1996.

[15] Riedl Reinhard and Richter Lutz, "Classification of Load Distribution Algorithms", *Proceeding of IEEE PDP'96*, pp. 404-413, 1996.

[16] P.Shroff, D.W Watson, N.F. Flann, and R.F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments", *Proceeding of Heterogeneous Computing Workshop '96*, pp.98-104, April 1996.

[17] M. Tan, J.K Antonio, et. al. , "Scheduling and data relocation for sequentially executed subtasks in a heterogeneous computing system", *Proceeding of Heterogeneous Computing Workshop '95*, pp109-120, 1995.

[18] Salleh Shaharuddin et. al., "A Mean-field Annealing Model For Task Scheduling in Multi-processor Systems", *Proceedings of PDPTA'96*, California-USA, Vol.2, pp. 189-198, August 1996.

[19] Shen S. Wu and David Sweeting, "Heuristic algorithms for task assignment and scheduling in a processor network", *Journal of Parallel Computing* 20, pp. 1-14, 1994.

[20] Honbo Zhou, Scheduling DAGs on a Bounded number of Processors, *Proceedings of PDPTA'96*, Vol.2, pp. 823-834, August 1996.