Process Reassignment with Reduced Migration Cost in Grid Load Rebalancing*

Lin Chen Cho-Li Wang Francis. C.M. Lau
Department of Computer Science,
The University of Hong Kong,
Pokfulam Road, Hong Kong
{lchen2, clwang, fcmlau}@cs.hku.hk

Abstract

We study the load rebalancing problem in a heterogeneous grid environment that supports process migration. Given an initial assignment of tasks to machines, the problem consists of finding a process reassignment that achieves a desired better level of load balance with minimum reassignment (process migration) cost. Most previous algorithms for related problems aim mainly at improving the balance level (or makespan) with no explicit concern for the reassignment cost. We propose a heuristic which is based on local search and several optimizing techniques which include the guided local search strategy and the multi-level local search. The searching integrates both the change of workload and the migration cost introduced by a process movement into the movement selection, and enables a good tradeoff between low-cost movements and the improving balance level. Evaluations show that the proposed heuristic can find a solution with much lower migration cost for achieving the same balance level than previous greedy or local search algorithms for a range of problem cases.

1 Introduction

In this paper, we consider the problem of dynamic load balancing through process migration in a heterogeneous grid environment. Given an initial assignment, the problem is to find a process reassignment which achieves a desirable level of load balance with minimum migration cost. This is called the *load rebalancing problem.* The minimum migration cost requirement suggests that this happens during the running of an application and not initially at the beginning.

The system conditions of a dynamic grid evolve quickly, and to obtain optimal load balance is not that feasible. It is more practical to maintain the load balance level of the machines within an acceptable range [4]. By achieving some reasonable target load balance level, our solution ensures that a low degree of resource wastage and a high service rate.

Both the grid resources and the applications can be highly heterogeneous. Multiple solutions of process reassignment can achieve the same balance level, but possibly with diverse migration costs. We set out to find the solution that can deliver the desired load balance level and its cost is acceptable. We aim at avoiding redundant process migrations and high-cost migrations when trying to improve the load balance level. In the process, the scheduler or job manager is provided with information on the lower bounds on the migration overhead.

The requirement on minimizing the rebalancing cost deviates from from the traditional load balancing approaches. With the double objectives of achieving a target load balance level and migration cost minimization, our problem is generally harder than the traditional Multiprocessor Scheduling Problem (MSP) which only aims at optimizing the makespan [7][10][13]. Since the conditions for optimality are more stringent, the number of available optimal solutions is smaller.

Our problem is a variant of the load rebalancing problem in [1] which is to find a process reassignment achieving optimal load balance with the constraint that at most a given number of processes are reassigned. That previous problem assumes that each process migration is of same cost. We minimize instead the actual migration costs which may vary from process to process. Existing proposed algorithms are based on greedy heuristics and can only loosely bound the number of migrations [25][1].

Our problem is also different from the remapping problem [26][16] where a data domain is re-partitioned to achieve a balanced workload. In data migration schemes, the data (workload) can be divided into arbitrarily small pieces for distribution among machines (i.e., divisible) [17][19] [20]. In our problem, the workload associated with a process must be moved as a whole (i.e., non-divisible). Process (workload) exchanges are needed in order to reduce

*This research is supported by a Hong Kong RGC grant (HKU 7176/06E) and a China 863 grant (2006AA01A111).

Table 1. Notations

Notation	Description
l	the total number of processes
q_i $(1 \leq i \leq l)$	a process
w_i $(1 \leq i \leq l)$	the workload of process q_i
m	number of machines
g_j $(1 \leq j \leq m)$	a machine
c_j $(1 \leq j \leq m)$	the computational capacity of machine g_j (measured by the amount of workload computed per second.)
E_I	an initial process assignment
d_j $(1 \leq j \leq m)$	the total workload in machine g_j, $d_j = \sum_{q_i \ in \ g_j} w_i$
dc_j $(1 \leq j \leq m)$	the load level in g_j, measured by the total workload in g_j normalized by its capacity, $dc_j = d_j / c_j$
DC	the maximum load level among all machines, $DC = max(dc_1, dc_2, \ldots, dc_m)$
DC_I	the DC of the initial process assignment E_I
DC_T	a target DC value specified in the problem
E_N	a new process assignment
DC_N	the DC of the new process assignment E_N
mig_cost	the migration cost caused by the new process assignment E_N (measured by the time spent in migration.)

the load differences among machines. Existing algorithms for divisible load problems generally do not consider workload exchange, and so cannot be directly applied here.

We refer to the main heuristic algorithm proposed in this paper as the Iterative Multi-Level PAIRWISE (IMLP) algorithm. The heuristic is based on local search and some special strategies for finding a low-cost solution. The algorithm focuses the search in the solution space surrounding the initial assignment, which has a strong likelihood of containing the desired low-cost solution. By a guided local search technique [18][15][23], the algorithm is able to generate diverse searching trajectories leading to different candidate reassignments, thus increasing the chance to find a reassignment with lower cost. A multi-level local search is proposed, where different restrictions on acceptable migration costs are defined to control the sub-steps of each local search step. The migration cost is a function of two practical parameters: process size and communication bandwidth. With such refined searching, our local search procedure decides on a practical tradeoff between low-cost process movements and target balance level. The evaluations show that our heuristic can find a reassignment which delivers a good level of load balance with a reasonably low migration cost.

The paper is organized as follows. In Section II, the load rebalancing problem is defined and discussed. The IMLP algorithm is presented in Section III. They are evaluated against other algorithms for various test cases in Section IV. We briefly discuss related work in Section V and conclude the paper in Section VI.

2 Problem Definition

The load rebalancing (LRB) problem is to find a process reassignment achieving a target load balancing level with minimum migration cost. A formal description of the problem is presented in the following. The notations used in the formulation are listed in Table 1.

- **Input**
  - The process set Q with l processes ($Q = \{q_1, \ldots, q_l\}$ where q_i has workload w_i).
  - The grid platform G with m machines ($G = \{g_1, \ldots, g_m\}$ where g_j has computational capacity c_j).
  - The initial assignment E_I with DC_I.
  - A predefined target DC_T ($DC_T < DC_I$).
- **Output**
  - A new process assignment E_N, satisfying: (1) $DC_N \leq DC_T$; (2) mig_cost is minimized.

DC is a metric measuring the degree of workload balance among the machines. The lower the value of DC the better the load balance of the system. DC can be easily mapped to other metrics (such as makespan, completion time, execution time, etc.) used by job managers or resource schedulers of typical parallel systems. With load rebalancing, the actual execution time is the execution time under the assignment with DC plus the migration time. So the assignment with lowest DC does not necessarily result in the lowest actual execution time, since it is possible for an assignment with a low DC to incur a large migration cost.

We can use the following method to decide on an appropriate DC_T. We first get an ideal value DC_o, assuming the workload can be perfectly balanced among machines. Then multiple values of DC_t between DC_o and the current DC_i are selected by say a binary search and tried. An appropriate DC_t is the one yielding the best tradeoff between improvement in the performance (in terms of reduction of completion time) and additional migration time.

In this paper we assume that DC_T is an empirical parameter specified by the system administrator. We will only focus on the algorithm that finds assignment solutions with one arbitrary DC_T value.

The cost of a process migration act is the time spent in the migration, which depends on several factors. The content to be moved during process migration includes the execution state (the information of Java frames) and data (objects, classes and local variables) in the process. According to our evaluation in a practical environment, "G-JavaMPI" [6], data transferring dominates the migration time; and so the migration time is very sensitive to data size. The time also depends on the CPU capacity and availability in the source and destination machines and the network bandwidth between these two machines. The source and destination machines are usually defined inherently in the procedure of load rebalancing, whereby over-loaded machines would serve be as sources of load transferring, and underloaded machines as destinations. Therefore, it is important to identify suitable processes with appropriate workloads and small data sizes, as well as to choose network links with higher bandwidths. The formula for a process migration cost is $cost = \frac{c \cdot size}{bandwidth}$ (c is a constant). For evaluation purposes, the *total* migration cost of a reassignment is the summation of all the individual process migration costs.

3 IMLP Algorithm

The IMLP heuristic considers the actual migration cost and aims to find a reassignment with reduced migration cost. It can also be used to find a reassignment with a small number of process migrations, for the problem assuming uniform cost for all process migrations.

3.1 Algorithm Skeleton

The IMLP heuristic adopts the guided local search (GLS) which is a meta-heuristic strategy based local search. The general working principle of GLS is to dynamically change the objective function used in local search, so that multiple local minima can be reached in the solution space [15][23]. The heuristic includes multiple iterations of local search. Each local search starts from the initial assignment and reaches a different new assignment (a local minimum). Among all the assignments found, the one achieving the target load balance level with the minimum migration cost is selected as the final solution.

Algorithm 1 Iterative Multi-Level PAIRWISE (IMLP)

```
 1: $E_I \leftarrow$ initial assignment, $DC_I \leftarrow DC$ in initial assignment, $DC_T \leftarrow$ target
 2: $E_N \leftarrow E_I, mig\_cost_N \leftarrow \infty$
 3: if $DC_T \geq DC_I$ then return $E_I$ endif
 4: for $i = 1$ to $num\_iter$ do
 5:     $E^{'} \leftarrow$ Multi-level_LOCAL_SEARCH($E_I$, $DC_I$, $DC_T$) (The maximum
        load level among machines in $E^{'}$ is $DC^{'}$. The migration cost of trans-
        forming to $E^{'}$ is $mig\_cost^{'}$.)
 6:     if $DC^{'} \leq DC_T$ && $mig\_cost^{'} < mig\_cost_N$ then
 7:         $E_N \leftarrow E^{'}, mig\_cost_N \leftarrow mig\_cost^{'}$
 8:     end if
 9:     PENALIZATION (update penalization status based on $E^{'}$)
10: end for
11: return $E_N$
```

Algorithm 1 shows the basic structure of the IMLP algorithm. The algorithm executes two procedures—the multi-level local search procedure (*Multi-level_LOCAL_SEARCH*) and the penalization procedure (*PENALIZATION*) for a number of iterations ("for" loop in lines 4–10). In each iteration, the multi-level local search starts from the initial assignment E_I, and searches for a new assignment $E^{'}$ through successive steps of process movements (line 5). As an essential part of guided local search strategy, the penalization procedure executed afterwards is used to modify the objective function used in the local search of the next iteration, so that a different new assignment can be reached (line 9). The objective function is modified to incorporate the factor of how many times each process movement is selected in the previously found solutions. The iterations of local search use different modified objective functions, so that they can find different new assignments.

The multi-level local search and penalization are executed for num_iter iterations. num_iter can be used to terminate the algorithm. The appropriate num_iter varies from problem case to problem case, and also depends on the time limit set on executing the algorithm. There is no clear theoretical analysis yet for the exact formula of num_iter. Instead, in our evaluation (see Section IV), we terminate the searching when no new and feasible solution (with DC satisfying $DC <= DC_T$) can be found after many iterations. The actual number of iterations varies from case to case.

In summary, three strategies adopted in the algorithm help achieve a target assignment with low migration cost. First, multiple candidate assignments are produced from the iterations, of which one assignment with the lowest cost will be selected. Second, the local search in each iteration starts from the initial assignment; that is, the algorithm searches more intensively in the solution space surrounding the initial assignment. Third, the refined local search step gives priority to process movements with lower costs. In the rest of paper, we use "step" to mean one iteration inside the local search procedure, and "iteration" to mean one iteration of the whole algorithm (which includes a local search procedure and a penalization procedure).

3.2 Multi-level Local Search Procedure

Local search starts with an initial assignment, and searches for a better assignment through a sequence of small steps. A small step refers to one or two process movements between a pair of machines. We use 1-move to denote moving a process from one machine to another; 1-swap to denote the exchange of two processes between two machines. Instead of considering movements between all machine pairs, we permit only movements between the machine with the maximum load level and another machine.

The core of multi-level local search is the local search procedure (Algorithm 2). In each step of the local search (the "while" loop in lines 2–17), L_1 always represents the machine with the maximum load level. Other machines (from L_m to L_2) are selected one after another for finding an appropriate 1-move or 1-swap between them and L_1 in the procedure F (lines 6–12). Suppose L_1 and L_j ($m \leq j \leq 2$) are considered, and (dc_1,dc_j) and ($dc_1^{'}$,$dc_j^{'}$) are their workloads respectively before and after selected movements are performed. F is successful only

Algorithm 2 Multi-level_LOCAL_SEARCH(E_I, DC_I, DC_T)

```
1: $E \leftarrow E_I, DC \leftarrow DC_I$
2: while 1 do
3:     Sort machines in descending order of their load levels, in a list
       $L \leftarrow \{L_1, L_2, \ldots, L_m\}$.
4:     $updated \leftarrow 0$
5:     for $i$ = 1 to $num\_level$ do
6:         for $j = m$ to 2 do
7:             if $F(L_1, L_j)$ with the level-$i$ threshold succeeds then
8:                 perform selected movements, update $E$, $dc_1$, $dc_j$, $DC$
9:                 $updated \leftarrow 1$
10:                BREAK
11:            end if
12:        end for
13:        if $updated = 1$ then BREAK endif
14:    end for
15:    if $updated = 0$ then BREAK endif
16:    if $DC \leq DC_T$ then return $E$ endif
17: end while
18: return $E$
```

if $max(dc_1^{'}, dc_j^{'}) < max(dc_1, dc_j)$ is true. If F succeeds, DC and E are updated (line 7) and a new step starts. If F fails for all machine pairs ($update = 0$ in line 15), the procedure terminates. When DC_T is achieved, the procedure also terminates immediately (line 16).

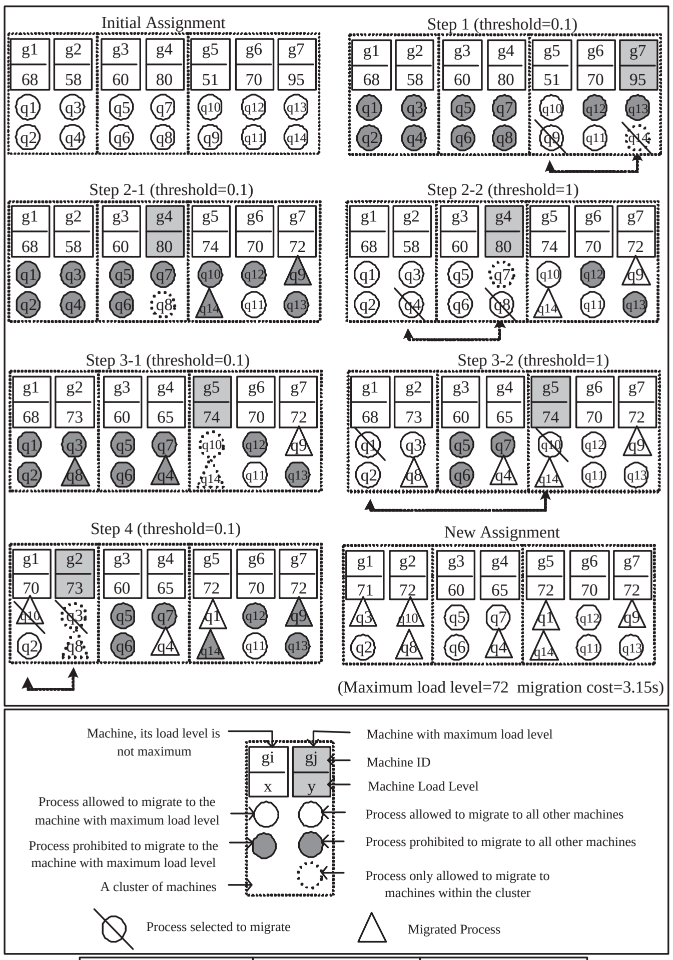
The preferred 1-move or 1-swap in F is the one which can reduce the maximum of the load levels of two machines to the largest extent. Suppose machine L_1 has $dc_1 = \frac{d_1}{c_1}$ and L_j has $dc_j = \frac{d_j}{c_j}$. The cost function is $CF = \max(dc_1, dc_j)$. Suppose after a 1-move or 1-swap, the workload of L_1 is reduced by w and the workload of L_j is increased by w. We then have $dc_1^{'} = (d_1 - w_x)/c_1$ and $dc_j^{'} = (d_j + w_x)/c_j$ for two machines. To minimize $CF^{'} = \max(dc_1^{'}, dc_j^{'})$, we need to find w satisfying $CF^{'} < CF$ and minimizing $|(dc_1^{'} - dc_j^{'})|$. The most preferred value of w is $\frac{d_1 c_j - c_1 d_j}{c_1 + c_j}$, because it makes L_1 and L_j perfectly balanced, i.e., $|dc_1^{'} - dc_j^{'}| = 0$. Otherwise, w should be less than $\frac{d_1 c_j - c_1 d_j}{c_1}$ and be as close to $\frac{d_1 c_j - c_1 d_j}{c_1 + c_j}$ as possible. For 1-move, we select a process with such a workload w. For 1-swap, we select two processes (one process with larger workload from L_1, and another with smaller workload from L_j) whose workload difference obeys the above rules regarding w.

The multi-level local search considers both process workloads and process migration costs. Each local search step (the "while" loop in lines 2–17) is refined into multiple sub-steps (the "for" loop in lines 5–14). The different restrictions on acceptable migration cost of a process movement (thresholds on acceptable migration cost) are defined in the sub-steps. Only process movements with costs lower than the threshold will be considered. The sub-step with the lowest level (level-1) threshold is first performed. If an appropriate movement can be found, the movement is performed, other sub-steps are skipped, and a new step starts (the BREAK in line 13). Otherwise, the sub-step with a higher level (more relaxed) threshold is performed. Thus the restriction level on migration cost is relaxed gradually from sub-step to sub-step. The sub-step with the highest level threshold is the last sub-step, where the threshold is high enough so that no movements will be prohibited. If no appropriate movement can be found in all the sub-steps ($updated = 0$), the local search terminates (line 15 and line 18).

The selection of the thresholds is crucial for the effectiveness of the multi-level local search in finding a low-cost solution. The thresholds should reflect the range and magnitude of the costs of all possible movements globally, so that they will not be too restrictive nor too loose. In the experiments, we evaluate three alternative ways of determining the thresholds.

Figure 1 illustrates the steps of the 3-level local search for an example problem. For demonstration's sake, the machine capacities are homogeneous, so the machine load level is represented by its workload. Intra-cluster communication bandwidth is 100MB/s and inter-cluster bandwidth is 10MB/s. The processes have various workloads and sizes (in MB). The cost of a single migration is the ratio of process size to bandwidth. Based on three thresholds, 0.1, 1.0 and 10, three levels of sub-steps are considered. The machine with a dotted rectangle is with the maximum load level. In this machine, processes with dotted circle is only allowed to migrate to machines within the cluster. Processes with shadowed circles are prohibited to migrate to all other machines. Processes with solid circles are allowed to migrate to any machine. In machines with solid rectangles, processes with shadowed circles are prohibited to migrate to the machine with the maximum load level, since their migration costs exceed the threshold. All processes with solid circles can be considered for migration.

In step 1, processes q_9 and q_{14} are exchanged, so that the load level of g_7 is decreased from 95 to 72. In step 2, only process q_8 is allowed to migrate to g_3, and process q_{11} is allowed to migrate to g_4. However, both migrations cannot decrease the load level of g_4. Therefore the higher-level subspace with threshold 1 is considered in step 3. The same situation happens in steps 4 and 5. In step 3-2, g_3 with lowest DC is not considered for movement, because both processes q and q will cause migration cost larger than the threshold and thus are prohibited for migration. Regardless of the thresholds, any process is allowed to migrate back to its initial location, so that the cost caused by its previous movement is deleted from the total migration cost. Such a movement is encouraged in the algorithm as long as it can also bring good improvement in reducing the maximum load level DC. For instance, in step 6, process q_4 in a black circle is allowed to migrate to g_2 (its initial location). The maximum load level in the final result is 72 and the migra-


Initial Assignment
Step 1 (threshold=0.1)
Step 2-1 (threshold=0.1)
Step 2-2 (threshold=1)
Step 3-1 (threshold=0.1)
Step 3-2 (threshold=1)
Step 4 (threshold=0.1)
New Assignment
(Maximum load level=72 migration cost=3.15s)
Machine, its load level is not maximum
Machine with maximum load level
Machine ID
Machine Load Level
Process allowed to migrate to the machine with maximum load level
Process allowed to migrate to all other machines
Process prohibited to migrate to the machine with maximum load level
Process prohibited to migrate to all other machines
A cluster of machines
Process only allowed to migrate to machines within the cluster
Process selected to migrate
Migrated Process

process	workload	size (MB)	process	workload	size (MB)	process	workload	size (MB)
q1	22	9	q6	33	31	q11	39	1
q2	46	5	q7	32	43	q12	31	25
q3	25	8	q8	48	5	q13	45	50
q4	33	9	q9	27	8	q14	50	9
q5	27	44	q10	24	6			

Figure 1. Illustration of Multi-level Local Search (Intra-cluster bandwidth 100MB/s, inter-cluster bandwidth 10MB/s, low level threshold=0.1, middle level threshold=1)

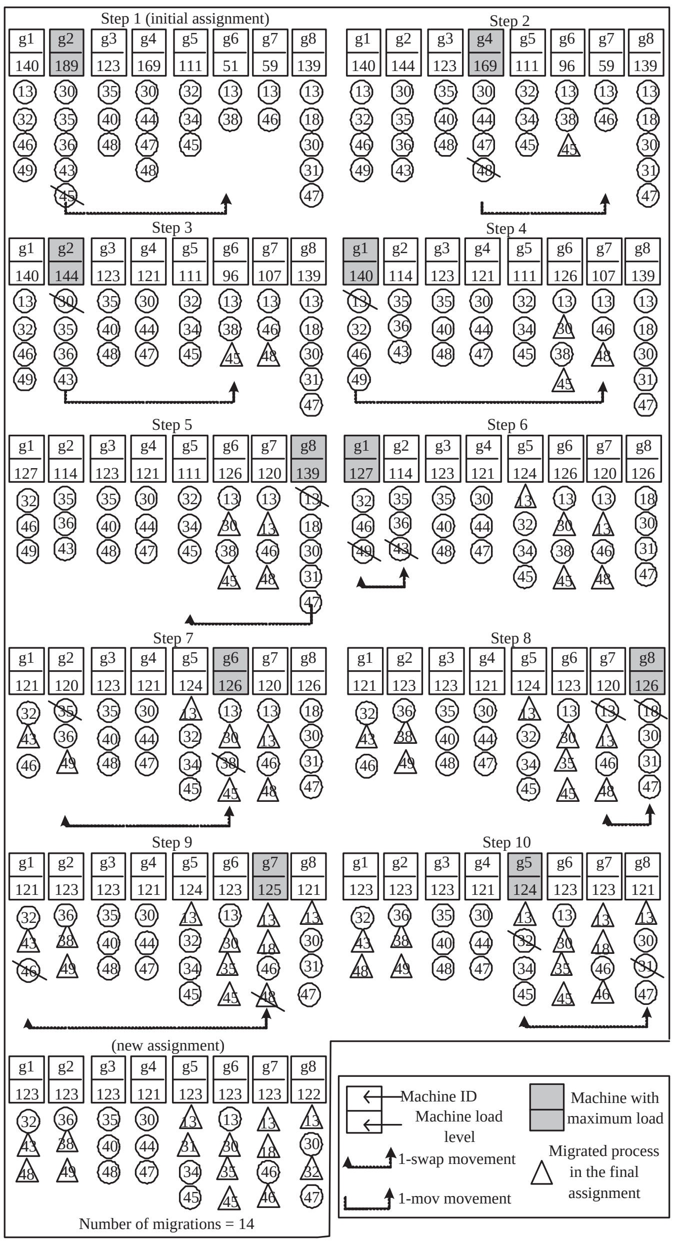


Figure 2. Illustration of Basic Local Search

tion cost is 3.15 (seconds).

For the purpose of comparison, we present the steps of the basic local search for the same example problem in Figure 2. The basic local search selects the processes for migration based on their workloads only. Thus the 3-level local search finds more migrations but with lower migration costs than the basic local search, since processes with smaller process sizes are considered with higher priorities.

3.3 Penalization Procedure

Penalization is a kind of memorization strategy. The technique records the count (or "frequency") that a component of the solution has been visited in the previous iterations. The higher frequency the component has, the more penalty it will receive, and the greater its likelihood to be unselected in the following iterations. This leads to a diversification guiding the search towards components that are not incorporated frequently enough in the past solutions. Such diversification enables more distinct solutions scattered in the solution space to be visited, and increases the chance of coming across a better solution. Some similar memorization strategies have been used in existing local search-based heuristics, such as the long-term memory strategies for diversification in Tabu search [12].

The basic component in an assignment solution is the status that a process is assigned to a machine. The implementation of the penalization scheme is as follows. A frequency matrix B with l rows and m columns is defined, where $B(x, j)$ and $B(y, i)$ represent the frequencies of process q_x being assigned to machine g_j and process q_y being assigned to g_i in all the solutions found so far. The cost function CF in the local search procedure is augmented to include the factors of frequencies. Suppose q_x is selected to migrate from g_i to g_j, and q_y is selected to migrate from g_j to g_i. The augmented cost function is $CF^{''} = \max(dc_i^{'}, dc_j^{'}) + \gamma \cdot (B(x,j) + B(y,i))$. γ is a constant called the penalty coefficient. $\gamma \cdot (B(x,j) + B(y,i))$ is the augmentation to the original cost function and is called the "penalty" on the corresponding movements.

The processes which minimize $\max(dc_i^{'}, dc_j^{'})$ might not deliver the minimum $CF^{''}$, if $\gamma \cdot (B(x,j) + B(y,i))$ is large. Therefore the 1-move or 1-swap with higher frequencies are more penalized and discouraged. If the movement(s) can achieve a small enough value of $\max(dc_i^{'}, dc_j^{'})$, it might still be selected in spite of its non-zero penalty. However, in the forthcoming iterations when its frequency (so is the penalty) is increased to a certain value, it will not be selected. As a result, chances are given to other movements that are less selected before.

Figure 3 demonstrates the second iteration of local search based on the penalization status.

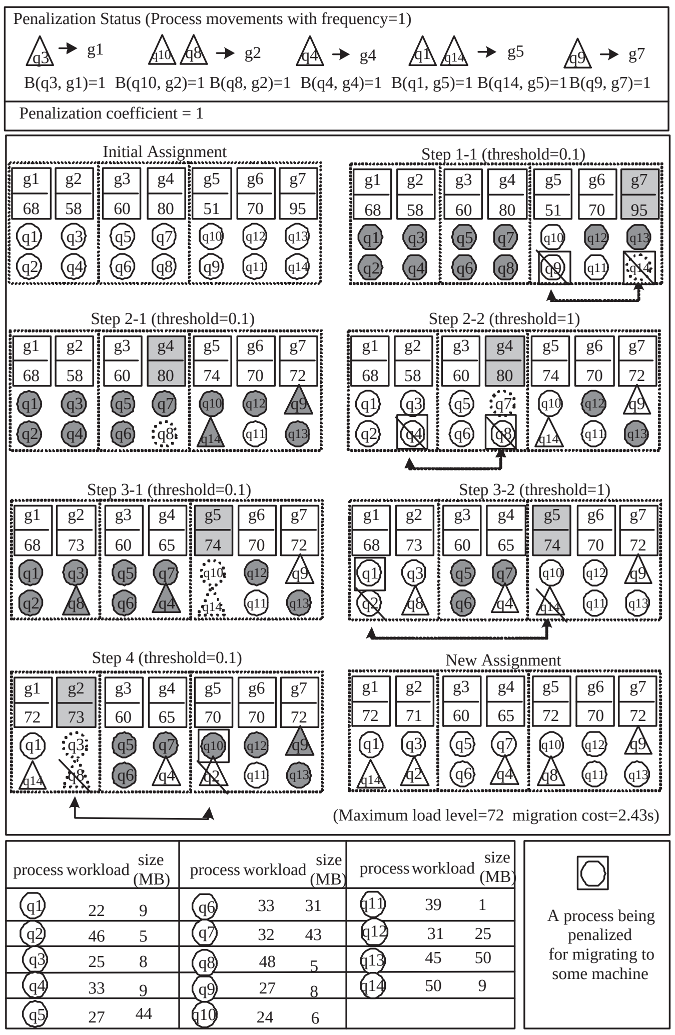
In the above example, all process movements are penalized. Other than the full penalization scope, alternatively we may only penalize part of the process movements. γ is another adjustable parameter. We evaluate the algorithms with different choices of penalization scope and γ values in the experiment section.

3.4 Algorithm Complexity

The algorithm's complexity mainly depends on the number of machine pairs considered (i.e., the number of steps in Algorithm 2) and the time spent in each execution of the procedure F. The number of levels (i.e., the number of sub-steps) in the multi-level local search is a small constant number. Therefore the complexity of IMLP is $O(l^2)$ (l is the number of process).

4 Performance Evaluation

The algorithms are tested for four groups of instances as in Table 2. The number of processes is from 40 to 640, and



Penalization Status (Process movements with frequency=1)
B(q3, g1)=1 B(q10, g2)=1 B(q8, g2)=1 B(q4, g4)=1 B(q1, g5)=1 B(q14, g5)=1 B(q9, g7)=1
Penalization coefficient = 1
Initial Assignment
Step 1-1 (threshold=0.1)
Step 2-1 (threshold=0.1)
Step 2-2 (threshold=1)
Step 3-1 (threshold=0.1)
Step 3-2 (threshold=1)
Step 4 (threshold=0.1)
New Assignment
(Maximum load level=72 migration cost=2.43s)
A process being penalized for migrating to some machine

process	workload	size (MB)	process	workload	size (MB)	process	workload	size (MB)
q1	22	9	q6	33	31	q11	39	1
q2	46	5	q7	32	43	q12	31	25
q3	25	8	q8	48	5	q13	45	50
q4	33	9	q9	27	8	q14	50	9
q5	27	44	q10	24	6			

Figure 3. Illustration of the Second Iteration of Multi-level Local Search

the system size from 16 to 256. All performance results are the average over 100 instances. In addition to the attributes in Table 1, several more attributes are used here. k denotes the number of processes in one machine in the initial assignment. c describes the heterogeneity of machine capacities (the range of the ratio of machine capacity to the average capacity). DC_O refers to the lower bound of the maximum load level (the ideal optimal DC, calculated as the ratio of total workload to total capacity). $w = 20\%/80\%$ means that 20% processes have workloads in $(20, (200 - 20) \cdot 80\%)$, and 80% processes have workloads in $((200 - 20) \cdot 20\%, 200)$. $c = .2/.3/.5$ means that 20% machines have capacity $\frac{0.5 \cdot total\_cap}{0.2 \cdot p_1}$, 30% machines have capacity $\frac{0.3 \cdot total\_cap}{0.3 \cdot p_1}$ and 50% machines have capacity $\frac{0.2 \cdot total\_cap}{0.5 \cdot p_1}$. $c = (0.6, 1.4)$ means that the machine capacities are dis-

Table 2. Test Cases

Case Group	m	k	w	c	DC_I	DC_O	l
$I_{1,1}$	16	(2,3)	(20,200)	1.0	124	74	40
$I_{1,2}$	16	(8,12)	(20,200)	1.0	101	74	160
$I_{1,3}$	64	(2,3)	(20,200)	1.0	136	74	160
$I_{1,4}$	64	(8,12)	(20,200)	1.0	109	74	640
$I_{1,5}$	256	(2,3)	(20,200)	1.0	144	74	640
$I_{2,1}$	16	(2,3)	(20,200)	(0.6,1.4)	151	74	40
$I_{2,2}$	16	(8,12)	(20,200)	(0.6,1.4)	130	74	160
$I_{2,3}$	64	(2,3)	(20,200)	(0.6,1.4)	182	74	160
$I_{2,4}$	64	(8,12)	(20,200)	(0.6,1.4)	149	74	640
$I_{2,5}$	256	(2,3)	(20,200)	(0.6,1.4)	204	74	640
$I_{3,1}$	16	(2,3)	(20,200)	.2/.3/.5	328	74	40
$I_{3,2}$	16	(8,12)	(20,200)	.2/.3/.5	273	74	160
$I_{3,3}$	64	(2,3)	(20,200)	.2/.3/.5	333	74	160
$I_{3,4}$	64	(8,12)	(20,200)	.2/.3/.5	270	74	640
$I_{3,5}$	256	(2,3)	(20,200)	.2/.3/.5	356	74	640
$I_{4,1}$	16	(2,3)	.2/.8	(0.6,1.4)	153	74	40
$I_{4,2}$	16	(8,12)	.2/.8	(0.6,1.4)	140	74	160
$I_{4,3}$	64	(2,3)	.2/.8	(0.6,1.4)	173	74	160
$I_{4,4}$	64	(8,12)	.2/.8	(0.6,1.4)	162	74	640
$I_{4,5}$	256	(2,3)	.2/.8	(0.6,1.4)	181	74	640

tributed uniformly in $(0.6 \cdot avg\_cap, 1.4 \cdot avg\_cap)$. All the above ranges and distribution properties of parameters are selected by following the traditional settings used in many previous process scheduling papers such as [3][2][9]. The case groups I_1/I_2 are derived from the test problems called UNIFORM in [3]. The case groups I_3/I_4/I_5 are derived from the test problems called TRIPLETS in [3]. All these case settings are based on the classical bin-packing instances available at the OR-Library of J.E. Beasley (available at http://people.brunel.ac.uk/ mastjjb/jeb/orlib/binpackinfo.html).

4.1 Effects of Algorithm Parameters

We compare several variations of the IMLP algorithm, which use different values for the algorithm's parameters. The algorithm parameters and their alternative values are listed below.

- The number of levels ($nolevels$). Two alternative values we use are 3 and 6.

- The cost thresholds in the multi-level local search. (1) IMLP-count selects the thresholds which divides the sorted list of all single process migration costs into $nolevels$ equal-length sublists. (2) IMLP-AS selects the thresholds which constitute an arithmetic sequence. (3) IMLP-CRS selects the thresholds which constitute a common ratio sequence. The selection of the common ratio depends on the range of single migration costs. For example, IMLP-CRS2 means that the common ratio of threshold sequence is 2. In all the above methods, the maximum threshold is selected as the largest single migration cost.

- The scope of penalization. (1) IMLP-full penalizes all process movements in the solution found in each iteration. (2) IMLP-limit penalizes only the process movements in the max-loaded machine(s) of the solution found in each iteration.

- The value of penalization coefficient (γ). The alternative values for γ are 0.25, 1, 4 and 8.

In this set of evaluations, each process is assigned a distinct process size in addition to the attributes in Table 2. Process sizes are randomly generated in the range of 1MB to 100MB. Processes with sizes smaller than 10MB occupy 2/3 of the whole process population. We simulate an aggregation of three clusters of equal sizes in a grid environment. The inter-cluster communication bandwidth is 10MB/s, and the intra-cluster bandwidth is 100MB/s. The distribution of the single migration costs is that 2/9 migration costs are lower than 0.1 second, 5/9 costs are between 0.1–1 second, and 2/9 costs are between 1–10 seconds.

Figure 4 compares the various IMLP algorithms with different $nolevels$ and threshold selection methods. The x-axis indicates different DC_T values specified for the algorithms, and the y-axis indicates the migration costs for achieving those DC_T's. The IMLP-AS algorithm finds solutions with higher costs than IMLP-count and IMLP-CRS. The algorithms with $nolevels = 6$ can usually find lower-cost solutions than the algorithm with $nolevels = 3$. Comparing the algorithms with $nolevels = 6$ and those with $nolevels = 9$, we find that the improvements by $nolevels = 9$ are not so significant. The six thresholds (average values) defined in IMLP-CRS2-level6 are 0.312, 0.625, 1.25, 2.5, 5 and 10. The six thresholds in IMLP-count-level6 are 0.082, 0.292, 0.555, 0.814, 3.873 and 10. The six thresholds in IMLP-AS-level6 are 1.666, 3.333, 5, 6.666, 8.333 and 10. In most cases, IMLP-CRS2-level6 achieves the lowest costs and IMLP-count-level6 also achieves migration costs close to the lowest ones. In the following sections, we focus on the experiments for the algorithm IMLP-CRS2-level6.

Figure 5 compares the IMLP algorithms with different γ values and penalization scopes. The IMLP-full algorithm can find solutions with lower costs than the IMLP-limit algorithm. The most suitable γ for IMLP-full is 1 and the most suitable γ for IMLP-limit is 4. Values that are higher or lower than the suitable one might deteriorate the performance. In the following sections, we look at the experiments for the algorithm IMLP-CRS2-level6-full-1.

4.2 Evaluation on Nonuniform Migration Cost Problems

IMLP is evaluated against several related algorithms, including ISLP, MLP and TABU. ISLP (Iterative Single-

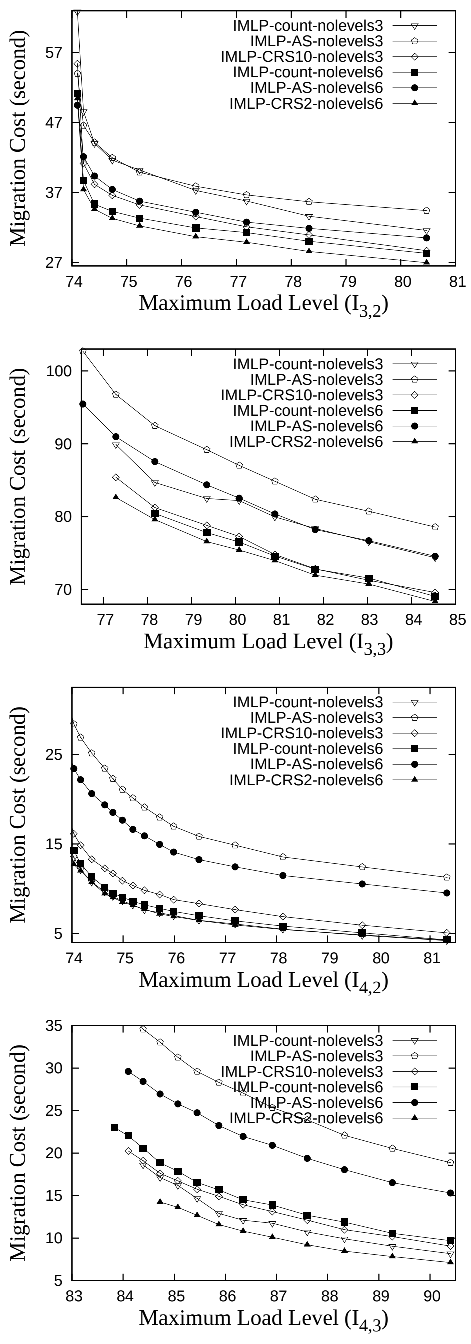


Figure 4. Performance of IMLP Algorithms with Diverse Multi-Level Parameters

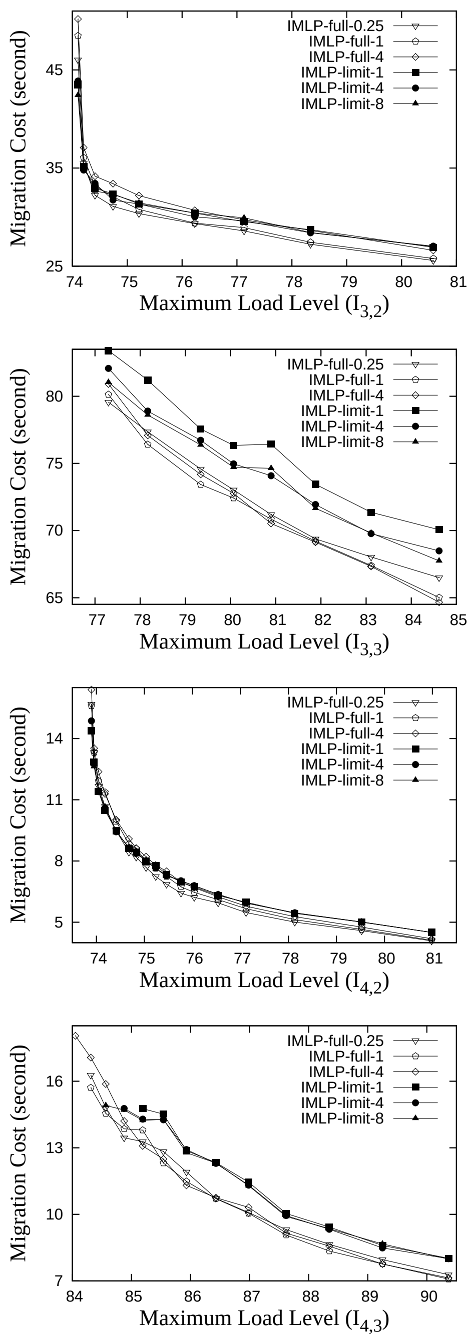


Figure 5. Performance of IMLP Algorithms with Diverse Penalization Parameters

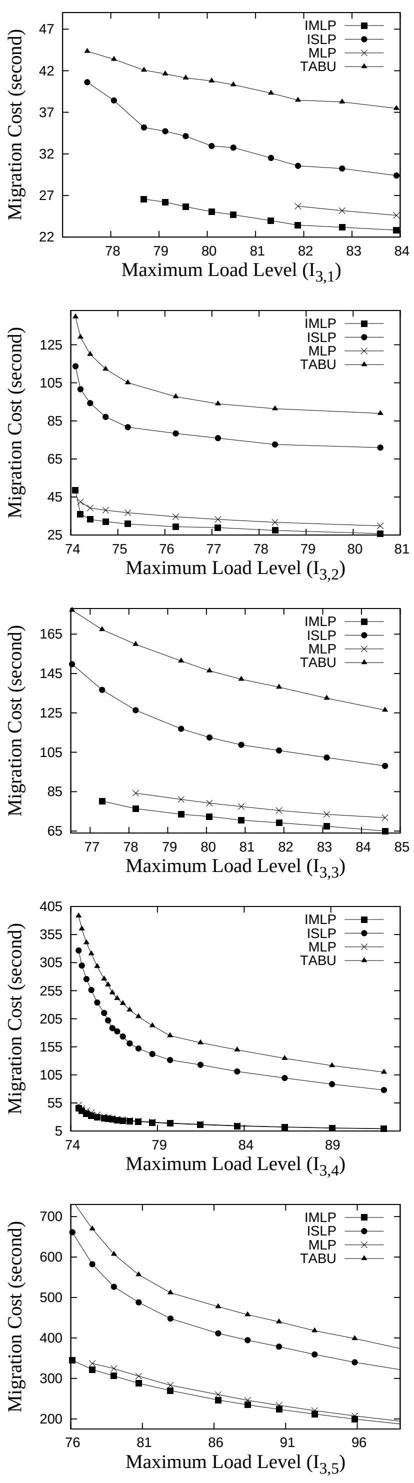


Figure 6. Performance of IMLP on Instances $I_{3,1}$-$I_{3,5}$

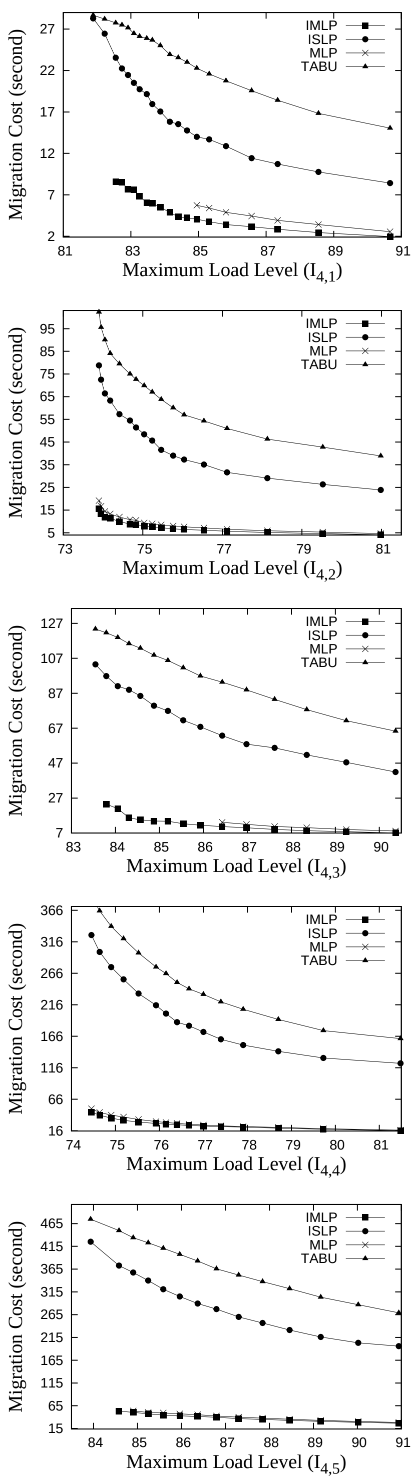


Figure 7. Performance of IMLP on Instances $I_{4,1}$-$I_{4,5}$

Level PAIRWISE) is the IMLP algorithm with $nolevels = 1$. MLP (Multi-Level PAIRWISE) is the first iteration of IMLP. TABU is the classical search method [5] [11] [22] for the independent task scheduling problem. For fairness of comparison, we implement the same basic local search procedure in TABU as that in IMLP. TABU never goes back to the initial solution to restart the searching, but always jumps from the current solution to a new solution according to the selected movement. The introduction of the tabu list avoids the search trapping in a local minimum. The best solution is selected among all solutions found during the searching. All algorithms are executed for the same total searching time.

Because of limited space, we only present the results for the third and fourth case groups in Figure 6 and Figure 7. These cases are harder since they have a higher degree of heterogeneity in their attributes. IMLP achieves the lowest migration cost than all other algorithms in all these cases. MLP still achieves much lower migration cost than ISLP which performs multiple iterations of the basic local search. This indicates that multi-level local search is essential in the search for a solution with low cost, and it contributes much to the winning algorithm IMLP. A serious drawback in using solely the multi-level local search is that low DC values cannot be achieved (the points on the leftmost in the curves indicate the best DC achieved). In such scenarios, guided local search strategy helps to find solutions with lower DC by intensifying the searching. Therefore IMLP is usually the best choice with consideration of both metrics, i.e., achievable DC and migration cost. ISLP still gets the chance to reduce the cost to a great extent. Without restarting the search from the initial assignment, TABU searching tends mostly to walk in the solution space far away from the initial assignment. This is why ISLP can achieve lower migration cost than TABU.

From the figures, we obtain a sense on the increasing rate of migration cost as the load balance level gets better. In the x-axis, from the right points to left points, the load balance level is getting better (i.e., the maximum load level is dropping). The migration cost increases in a mild rate in most of the range. After some point, the cost increases very rapidly, especially when the load balance level is getting close to the best level. The migration cost might go to too high a level which offsets the benefit. Therefore it is desirable to choose a modestly satisfactory load balance level but not an optimal or near-optimal level as the target. We have also found that instances $I_{3,2}$, $I_{3,4}$, $I_{4,2}$ and $I_{4,4}$ are much easier than $I_{3,3}$, $I_{3,5}$, $I_{4,3}$ and $I_{4,5}$. The average number of processes in each machine is higher in these instances, so the granularity of process workloads for the machine capacities is finer, which enables a more even distribution of the workloads. As a result, our algorithms tend to find much lower DC (very close to the optimal) for these cases.

Table 3. Execution Times of IMLP and Other Algorithms

Case Group	Total time (s)	IMLP	ISLP	MLP	TABU
$I_{3,1}$	0.037	0.009	0.01	0.0024	0.0083
$I_{3,2}$	0.63	0.14	0.017	0.046	0.02
$I_{3,3}$	1.26	0.43	0.22	0.075	0.037
$I_{3,4}$	11.37	4.24	0.35	0.74	0.32
$I_{3,5}$	66.9	29.4	12.2	4.06	3.36
$I_{4,1}$	0.023	0.0045	0.003	0.0014	0.008
$I_{4,2}$	0.34	0.05	0.014	0.019	0.018
$I_{4,3}$	0.646	0.157	0.091	0.04	0.123
$I_{4,4}$	11.36	4.236	0.353	0.739	0.316
$I_{4,5}$	28.1	10.32	3.54	1.43	0.24

Table 3 lists the times spent in finding the best solutions in all the algorithms. All the algorithms are given the same total execution time, as listed in the second column. IMLP spends more time in finding the best solutions than MLP and TABU. This is because it needs to perform multiple iterations of searching before their best solutions are found. IMLP needs more time than ISLP in larger-size problems because of additional overhead in the multi-level local search.

4.3 Evaluation on Uniform Migration Cost Problems

In this section, we test the algorithms on the cases where all process migrations have the same uniform cost. The migration cost is reduced to the number of migrations. IMLP algorithm is simply reduced to ISLP with $nolevels = 1$. We compare the performance of ISLP with the greedy heuristic and the TABU heuristic.

M-PARTITIONV (MPARV for short, Algorithm 3) is a variant from M-PARTITION heuristic proposed in [1] for migration-constraint makespan minimization. It includes multiple iterations of PARTITION procedure (refer to the pseudo-code presented in [1]). Each PARTITION is given with an estimated OPT value on DC. Based on the OPT, it identifies the processes with large and small workloads. Smallest number of additional large and small processes are removed so that DC of all machines are lower than OPT. It then reassigns the removed processes with larger processes considered first. OPT in the first iteration is set as DC_T and then gradually decreased in the following iterations.

We use different DC_T values to test the ISLP, MPARV and TABU algorithms. Table 4 presents the success rate, the number of migrations and searching time of these three algorithms for two different DC_T values. MPARV, which is a greedy heuristic, cannot find very low DC_T values. Therefore its success rates for some cases with low DC_T are not 100%. The number of migrations found by MPARV is also much higher than those in ISLP and TABU. ISLP can find smaller number of migrations than TABU in all cases, al-

Algorithm 3 M-PARTITIONV (MPARV)

```
1: Use $DC_T$ as the starting value for $OPT$.
2: PARTITION(OPT), $DC_N$ ← achieved $DC$ in new assignment, $mc$ ← the
   migration cost
3: while $OPT \geq theminimumprocessworkload$ do
4:     Decrease the value of $OPT$ to the next lower threshold value
5:     PARTITION(OPT), $DC_N$ ← achieved $DC$ in new assignment, $mc$ ←
       the migration cost
6: end while
7: return the new assignment satisfying $DC_N \leq DC_T$ and achieving the mini-
   mum $mc$ among all new assignments.
```

Table 4. Comparison of ISLP, MPARV and TABU

Case	Alg	success (%)	mig num	time (ms)	success (%)	mig num	time
		DC_T=76			DC_T=79.3		
$I_{1,1}$	ISLP	93	18.5	1.2	100	11.8	0.66
	MPARV	8	26.8	2.2	79	21.9	2.26
	TABU	100	21.4	0.45	100	13.6	0.11
		DC_T=74.4			DC_T=76.9		
$I_{1,2}$	ISLP	100	18.6	1.27	100	19.8	0.17
	MPARV	60	111.9	35	100	45.9	34.7
	TABU	100	21.2	0.23	100	9.9	0.09
		DC_T=76.4			DC_T=80		
$I_{1,3}$	ISLP	100	70.8	11.3	100	45.9	4.6
	MPARV	10	102	137	90	85.3	59
	TABU	100	74.7	1.9	100	48	0.94
		DC_T=74.4			DC_T=77.5		
$I_{1,4}$	ISLP	100	78.6	13.8	100	36.8	0.92
	MPARV	50	572.6	636.9	100	325.7	636.7
	TABU	100	82.6	2.14	100	36.8	0.86
		DC_T=78			DC_T=82.9		
$I_{1,5}$	ISLP	100	235.9	44.2	100	143.6	28.4
	MPARV	60	389.5	1307	100	323	1351
	TABU	100	240	17.6	100	145.3	9.4
		DC_T=76			DC_T=79.3		
$I_{4,1}$	ISLP	96	18.2	1.36	100	11.1	0.83
	MPARV	0	-	2.07	10	29.6	2.07
	TABU	100	20.1	0.56	100	13.2	0.14
		DC_T=74.7			DC_T=78		
$I_{4,2}$	ISLP	100	43.3	3.9	100	26	1.3
	MPARV	10	131.6	14.1	80	88.6	14
	TABU	100	47.4	0.68	100	27.5	0.26
		DC_T=83.5			DC_T=94.4		
$I_{4,3}$	ISLP	92	75.3	27.8	100	29	3.9
	MPARV	0	-	34.9	4	125	34.8
	TABU	100	79.5	10.34	100	31.4	0.5
		DC_T=74.6			DC_T=79.7		
$I_{4,4}$	ISLP	100	216.4	89.1	100	97.5	11.4
	MPARV	0	-	195.9	76.7	468.6	195.2
	TABU	100	232	12	100	98.6	2.25
		DC_T=84			DC_T=101		
$I_{4,5}$	ISLP	87	295	635.5	100	86	18.4
	MPARV	0	-	570	16.7	582	567
	TABU	100	298	149	100	87	5.3

though it is slightly weaker in achieving lower DC_T than TABU.

5 Related Work

The classical Multiprocessor Scheduling Problem (MSP) denoted as $R||Cmax$ [7][10][13] focuses on finding optimal makespan. Most local search-based heuristics in the literature use 1-move or 1-swap or their variations or combinations as the neighborhood structure [11][21][8][9]. But they use local search solely without any meta-heuristic strategy (such as the guided local search in our algorithm). A solution with target DC can be found, but the solution might not have a small migration cost.

There was traditional research on dynamic remapping for data-parallel applications (such as those in molecular dynamics (MD) and computational fluid dynamics (CFD)), where the computational requirements associated with different parts of the data domain may change as the computation proceeds. The nearest-neighbor algorithms used by Xu and Lau [26] reduce the domain remapping operation to adjustment of sub-domain borders, so that data redistribution cost can be kept low. In addition, the cost can be reduced through limiting the number of redistribution operations across remote machines [16].

In other dynamic data remapping works, the data and the associated workload can be divided into arbitrary small pieces and distributed among machines [17][19] [20][24][14]. Since workload can be arbitrarily divided, it is easy to know how much workload should be migrated out/in the over-loaded/under-loaded machines. This information is used as constraint conditions in the problem formulation. While in our problem, the whole workload associated with a process will be moved when the process is migrated. So existing algorithms for divisible load problems cannot be applied to our problem.

A few existing projects consider the number of migrations caused in task reassignment. Westbrook [25] considers an online load balancing problem, and proposes a 5.83-competitive algorithm with bounded number of migrations for identical machines, and another 8-competitive algorithm with bounded number of migrations for related machines. Aggarwal, Motwani and Zhu [1] consider the LRB problem in placing websites onto a set of web servers. They aims to maximize the makespan with a predefined constraint on the number of migrations. A heuristic with a 1.5 approximation ratio on the makespan is proposed. These heuristics are variants of the greedy heuristic. Although providing a guarantee on the worst case performance by bounds, they do not explicitly minimize the migration cost.

6 Conclusion

In this paper we study the important problem of *load rebalancing*. The problem has an emphasis on the concern of the reassignment cost since a good tradeoff between system load balance level and its associated cost should be achieved in practice. With the functionality to find an reassignment with lower cost, system administrators could compare the costs for achieving various load balanced levels. This information is helpful for the administrators to decide the most appropriate reassignment and load balance level for maximizing their ultimate performance objectives. Our proposed algorithm, IMLP, incorporates some special strategies to reduce the reassignment cost. These strategies (including concentration of the searching effort in the solution space surrounding the initial assignment, guided local search, and multi-level local search) are proved to be able to achieve lower migration costs when compared to existing heuristics. Our proposed strategy can be easily incorporated in many existing LB algorithms for cost minimization.

References

[1] G. Aggarwal, R. Motwani, and A. Zhu. The Load Rebalancing Problem. *Journal of Algorithms*, 60(1):42–59, 2006.

[2] A.C.F. Alvim and C.C. Ribeiro. A Hybrid Bin-Packing Heuristic to Multiprocessor Scheduling. In *Proceedings of the Third International Workshop on Efficient and Experimental Algorithms (WEA'2004)*, 2004.

[3] A.C.F. Alvim, C.C. Ribeiro, F. Glover, and D.J. Aloise. A Hybrid Improvement Heuristic for the One-Dimensional Bin Packing Problem. *Journal of Heuristics*, 10(2):205–229, 2004.

[4] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-Based Load Management in Federated Distributed Systems. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI2004)*, 2004.

[5] T.C. Braun, H.J. Siegel, and N. Beck. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61:810–837, 2001.

[6] L. Chen, T. Ma, C.L. Wang, F.C.M. Lau, and S. Li. G-JavaMPI: A Grid Middleware for transparent MPI task migration. In B.D. Martino, J. Dongarra, A. Hoisie, L.T. Yang, and H. Zima, editors, *Engineering the Grid: Status and Perspective*. Nova Science Publisher, 2006.

[7] T. Cheng and C. Sin. A State-of-the-Art Review of Parallel-machine Scheduling Research. *Eropean Journal of Operational Research*, 47(2):271–292, 1990.

[8] P.M. Franca, M. Gendreau, G. Laporte, and F.M. Muller. A Composite Heuristic For The Identical Parallel Machine Scheduling Problem With Minimum Makespan Objective . *Computers Operations Research*, 21(2):205–210, 1994.

[9] A. Frangioni, E. Necciari, and M.G. Scutella. A Multi-Exchange Neighborhood for Minimum Makespan Machine Scheduling Problems. *Journal of Combinatorial Optimization*, 8:195–220, 2004.

[10] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[11] C.A. Glass, C.N. Potts, and P. Shade. Unrelated Parallel Machine Scheduling Using Local Search. *Mathematical Computing and Modelling*, 20(2):41–52, 1994.

[12] F. Glover. *Tabu Search*. Kluwer Academic Publishers, 1997.

[13] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.J.G. Rinnooy Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[14] E. Jeannot and F. Vernier. A Practical Approach of Diffusion Load Balancing Algorithm. In *Europar*, pages 211–221, 2006.

[15] P. Kilby, P. Prosser, and P. Shaw. Guided Local Search for the Vehicle Routing Problem with Time Windows. In S. Vob, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics: Advances and trends in local search paradigms for optimization*, pages 473–486. Kluwer Academic, 1999.

[16] Z. Lan, V.E. Taylor, and Y. Li. DistDLB: Improving Cosmology SAMR Simulations on Distributed Computing Systems Through Hierarchical Load Balancing. *Journal of Parallel and Distributed Computing*, 66(5):716–731, 2006.

[17] Y. Li and Z. Lan. A Novel Workload Migration Scheme for Heterogeneous Distributed Computing. In *Proceedings of the 5nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CC-GRID2005)*, 2005.

[18] P. Mills and E. Tsang. Guided Local Search for Solving SAT and weighted MAX-SAT Problems. In *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT2000)*, pages 89–106, 2000.

[19] L. Oliker, R. Biswas, and H.N. Gabow. Performance Analysis and Portability of the PLUM Load Balancing System. In *Proceedings of Euro-Par'98 Parallel Processing*, pages 307–317, 1998.

[20] I. Pardines and F.F. Rivera. Minimizing the Load Redistribution Cost in Cluster Architectures. In *Proceedings of Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2004.

[21] N. Piersma and W. Van Dijk. A Local Search Heuristic for Unrelated Parallel Machine Scheduling with Efficient Neighborhood Search. *Mathematics and Computer Modeling*, 24(9):11–19, 1996.

[22] B. Srivastava. An Effective Heuristic for Minimising Makespan on Unrelated Parallel Machines. *The STOR Journal of the Operational Research Society*, 49(8):886–894, 1998.

[23] C. Voudouris and E. Tsang. Guided Local Search and Its Applications to the Travelling Saleman Problem. *European Journal of Operational Research*, 113:469–499, 1999.

[24] J. Watts and S. Taylor. A Practical Approach to Dynamic Load Balancing. *IEEE Transactions on Parallel and Distributed Systems*, 9(3):235–248, 1998.

[25] J. Westbrook. Load Balancing for Response Time. *Journal of Algorithms*, 35(1):1–16, 2000.

[26] C.-Z. Xu, F.C.M. Lau, and R. Diekmann. Decentralized Remapping of Data Parallel Applications in Distributed Memory Multiprocessors. *Concurrency: Practice and Experience*, 9(12):1351–1376, 1997.

Biographies

Lin Chen received her PhD degree in Computer Engineering from The University of Hong Kong (HKU) in Jan. 2007. Since Jun. 2006, she worked as an assistant researcher in the University of Hong Kong and the Shenzhen Institute of Advanced Technology, Chinese Academy of Science for one year and half a year respectively. Her research interests include load balancing, grid scheduling and grid-based applications.

Cho-Li Wang is an associate professor in the Department of Computer Science in the University of Hong Kong. His research interests include middleware design for Cluster and Grid Computing. He received his Ph.D. degree in Computer Engineering from University of Southern California in 1995.

Francis C.M. Lau is a professor in computer science in the University of Hong Kong. His research interests include operating systems, parallel and distributed systems, pervasive and mobile computing, and wireless sensor networks. He received a PhD in computer science from the University of Waterloo in 1986.