Compromise Scheme for Assigning Tasks on a Homogeneous Distributed System

Joo-Man Kim, *Member, KIMICS*

***Abstract*—We consider the problem of assigning tasks to homogeneous nodes in the distributed system, so as to minimize the amount of communication, while balancing the processors' loads. This issue can be posed as the graph partitioning problem. Given an undirected graph *G*=(*nodes*, *edges*), where nodes represent task modules and edges represent communication, the goal is to divide *n*, the number of processors, as to balance the processors' loads, while minimizing the capacity of edges cut. Since these two optimization criteria conflict each other, one has to make a compromise between them according to the given task type. We propose a new cost function to evaluate static task assignments and a heuristic algorithm to solve the transformed problem, explicitly describing the tradeoff between the two goals. Simulation results show that our approach outperforms an existing representative approach for a variety of task and processing systems.**

***Index Terms*—task assignment problem, homogeneous system, load balancing, graph partitioning.**

I.INTRODUCTION

THE cluster systems are not only provide facilities for utilizing resources and/or data at remote sites, but also enhance system performance and reliability with the multiplicity of processors and communication paths. Distributed applications range from large data base installations where the computing load is distributed for organizational efficiency, to small signal-processing systems where computation must be done very fast in a real-time environment.

There are, however, many problems to be resolved before realizing the potential of a distributed system, such as file and task assignment [6]. To alleviate part of these problems, we consider the problem of assigning tasks in cluster computing systems. Specifically, we will deal with static assignment of a given task to the processors in a distributed system which neither requires nor precludes the subsequent dynamic migration of the task. In particular, we are interested in developing centralized task assignment algorithms using global knowledge of the task and system characteristics. These task assignment algorithms attempt to assign each task to the processors so as to achieve such goals as the minimization of *inter-processor communication*, good *load balancing* among the processors, a small response time for the task, and efficient utilization of system resources in general. This statement of the static task assignment problem is also useful for assignment of tasks to fewer processors than the distributed algorithm was initially designed for, at the time of initial assignment or dynamically at execution time due to node failure, node withdrawal, or phase transitions in the distributed computation [7].

In this paper, we propose a new cost function for making and evaluating *static task* assignments, which describes the inherent conflict between the goals of minimizing communication and balancing loads by combining the two criteria into a single objective function. Also, the system designer scanmake an appropriate compromise between the two conflicting goals by systematically adjusting the weight in this cost function according to the underlying task type. We show that the task assignment problem can be modeled as the problem of minimizing the cost of an *n*-cut of a graph (the *minimum n-cut* problem) instead of the minimum balanced *n-cut* problem.

Note that while the minimum balanced *n-cut* problem has two objectives, the minimum *n-cut* problem has only one objective to optimize. The new problem, however, systematically deals with both communication cost minimization and load balancing. The main result of this paper is the development of a heuristic algorithm that allows the system designers to systematically study the effects of relaxing the load balancing constraint on the total communication cost.

The remainder of this paper is organized as follows. The next section presents background information on the task assignment problem. Section IIIdescribes the system model and problem statement for the task assignment problem. It is shown in Section IV that the task assignment problem can be modeled as the problem of finding a minimum *n-cut* in a graph using a graph-modification technique. In Section V, we present an iterative algorithm to solve this problem. Some experimental results and concluding remarks are made in the last two sections.

II.BACKGROUND

The task assignment problem was first introduced by Stone[2]. Stone's original work lays down the TIG model

Manuscript received February 18, 2011; revised April 1, 2011; accepted April 15, 2011.

Joo-Man Kim is with the Department of Applied IT and Engineering, Pusan National University, Pusan609-735, Korea (Email: joomkim@pnu.edu)

to represent sequentially executing tasks. Bokhari[3][4] conducted a number of task-assignment studies without any inter-task constraints by minimizing total execution and communication costs. Lee[6][8] proposed an exact algorithm that map TIGs to processors in the array networks for minimizing the sum of total execution and communication costs. Lo[7] proposed some heuristic algorithms to improve the degree of concurrency in task assignment by extending Stone's model. Salman[10] proposed a particle swarm optimization for the static task assignment problem to effectively exploit the capabilities of distributed or parallel computing systems. Other researchers[11][12] investigated the task assignment problem by minimizing communication subject to certain constraints on the degree of load balancing, or the minimization of task completion time.

There are numerous studies addressing the task assignment problem under various characterizations. For some later works on mapping TIGs to processors in order to minimize turnaround time see [14]for exact algorithms under processor heterogeneity and network homogeneity; [15] for exact algorithms under processor and network heterogeneity and [13] for heuristics that map TIGs to processors in order to minimize total communication time in a heterogeneous network.

The task assignment problem can be modeled as the problem of partitioning the nodes of a graph into *n*subsets so as to minimize the cost of the *n-cut* (i.e., *the communication cost*) and balance the subset size (i.e., *load balancing*). We will call this problem the *minimum balancedn−cut* problem which is known to be NP-complete [6][9].Kernighan and Lin[1] proposeda highly-efficient heuristic for the case of n= 2. The heuristic can be applied to solve theminimum balanced *n*-cut problem when$n$>2. However, improvements in these heuristics were made for the case of n= 2, and hence, they have littleuse for the case of n>2, because their efficiency decreases significantly as *n*gets larger [6].

Moreover, they all minimize the cutset cost and consider load balancing only as a constraint. It is very difficult for them to make a tradeoff between load balancing and communication minimization for the following two reasons. First, they do not take into account the degree of load balancing as long as they meet the load balancing constraints. Thus, in case a loose constraint is used, they may yield unbalanced assignments because their communication costs are slightly lower than the costs of other well-balanced assignments. Second, they do not consider any assignment which does not satisfy the load balancing constraint. Therefore, when a strict load balancing constraint is to be met, they may select assignments with much higher communication costs than others due to the constraint. However, for the task assignment problem, both load balancing and communication minimization should be achieved and must be considered at the same time. Also, the system designers should be able to make an appropriate compromise between the two criteria according to the given task type. Thus, the existing heuristics are not very useful for the above task assignment problem.

III. TASK SYSTEM AND PROBLEM STATEMENT

A. Task System Model

We assume that a given distributed system consists of *n*nodes and an interconnection network that provides full connectivity among the nodes. Formally, we define a task force as a set of m tasks $T = \{t_1, t_2, \cdots, t_m\}$ which are to be assigned to n nodes $P = \{p_1, p_2, \cdots, p_n\}$ in the homogeneous distributed system. Let x_i be the execution cost of task*i* and let the interaction among the tasks in T be represented by a TIG, in which nodes correspond to the tasks in T and there exists an edge between two nodes if and only if the corresponding tasks interact. Task assignment to processors, is given by a matrix a, where a_{ip} is 1, if task i is assigned to processor p, and 0 otherwise. Let c_{ij} be a binary matrix such that denote the communication cost between two tasks t_i and t_j if they are assigned to different processors. Obviously, if no edge exists between t_i and t_j in the graph, then $c_{ij} = 0$. Both the execution and communication costs are derived from the types of the tasks. A task set T can be one of two types: *computation module* or *communication module*. They may be specified explicitly by the programmer, or deduced automatically by the compiler, or refined by monitoring the previous executions of the task. In this paper, we presume that the data about the execution and communication costs are available and they are positive integer values.

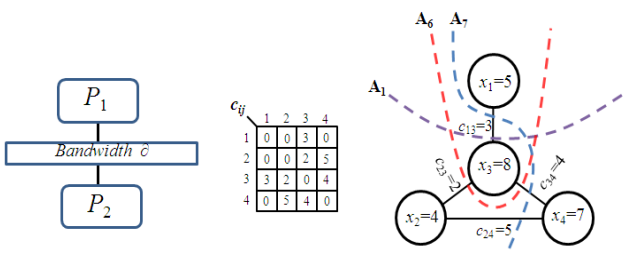

Fig. 1. TIG for a 2-processor system.

For tractability we ignore other attributes of the task-processor system such as timing and precedence constraints, and we assume that the communication costs are independent of the communication link upon which they occur. We will assume that the communication cost between two tasks executed on the same processor is negligible, since such communication is accomplished by

accessing local shared memory, as opposed to using inter-processor communication links. An assignment of tasks to processors can formally be described by a function from the set of tasks to the set of processors, $A: T \rightarrow P$. In a system of ***m*** tasks and ***n*** homogeneous processors there are $1/2 \cdot n^m$ possible assignments of tasks to processors. For example, consider the TIG for two-processor system depicted in Figure 1.

To minimize the communication overhead, assignment A_1 with the communication cost3is the best solution where t_1 is assigned to P_1 and all the other tasks assigned to P_2. Assignment A_1 however, induces the processor 'load6 and 22 on P_1 and P_2, respectively, making the system highly-unbalanced. To achieve load balancing, we can choose assignment A_7 that is perfectly load-balanced. However, the communication cost of A_7 is 12 which is much higher than that of A_1 . If t_4 assigned to P_1 is exchanged t_3 assigned to P_2under A_7.i.e., assignment A_6,the Comm. cost decreases to 6 at a small loss of load balancing. Assignment A_6 may be chosen if no strict condition is imposed on the load balancing objective. Therefore, the system designer should be able to make an appropriate compromise between the two conflicting objectives according to the task type.

B. Problem Statement

The communication cost of an assignment *A*, denoted as *Comm*(*A*), is then calculated as :

$$Comm\ (A) = \sum_{i<j} \sum_{p=1}^{n} a_{ip}(1 - a_{jp})\ c_{ij}$$

The load L_p of processor p is the total cost of running the tasks assigned to processor p under this assignment:

$$L_p = \sum_{i=1}^{m} a_{ip} x_i$$

$$L_{tot} = \sum_{p=1}^{n} L_p$$

The standard deviations is essential for assessing the degree of dispersion of the loads around its mean load. If this load could be uniformly distributed among the n processors, each processor would be assigned a share of mean $\bar{L}$. For a given task, L_{tot} and $\bar{L}$ are uniquely determined irrespective of the assignment of the task. $\sigma^2(A)$, the variance, for each assignment *A*,

$\sigma^2(A) = \sum_{p=1}^{n} \frac{(L_p - \bar{L})^2}{n} = \frac{n-1}{n^2} {L_{tot}}^2 - \frac{2}{n} \sum_{p<q} L_p \cdot L_q$. is the variance of load distribution under the assignment. The first term, $(n-1){L_{tot}}^2 / n^2$, is constant for a given task, and only the second term, $(2/n)\sum_{p<q} L_p \cdot L_q$, varies between 0 and $(n-1){L_{tot}}^2 / n^2$, depending on the assignment of the task. Then, the following inequality holds :

$$0 \le \sigma^2(A) \le \frac{(n-1)}{n^2} {L_{tot}}^2, \quad for\ all\ assignments\ A.$$

The variance of load distribution indicates how evenly the load is distributed among the processors, i.e., the lower the variance the better balanced the load distribution. The maximum value of $\sigma^2(\cdot)$ is achieved when all the tasks are assigned to a single processor. $\sigma^2(\cdot)$ reaches its minimum value when all the processors are most evenly loaded. In such a case, the system is said to be *best load-balanced*. Thus, one may use $\sigma^2(A)$ as a yardstick to measure *the degree of load balancing* of each assignment *A*. For simplicity, we define *thedegree of load balancing* of *A*, denoted as *LB*(*A*), as :

$$LB(A) = 1 - \frac{\sigma^2(A)}{n - 1/n^2 \cdot {L_{tot}}^2} = \frac{2n}{(n-1){L_{tot}}^2} \sum_{p<q} L_p \cdot L_q .$$

Note that *LB*(*A*) always satisfies the inequality $0 \le LB(A) \le 1$ .Using $Comm(\cdot)$ and $\sigma^2(\cdot)$, we propose the following cost function :

$$COST(A) = Comm(A) + \alpha \cdot \sigma^2(A)$$

$$= Comm\ (A) + \alpha \cdot (1 - LB(A)) \frac{n-1}{n^2} {L_{tot}}^2,$$

TABLE I
ASSIGNMENT RESULT FOR TIG OF FIGURE 1

Assign	Assign. to Proc.		Load		$\sigma^2(A)$	$LB(A)$	$Comm(A)$	$COST(A)$
	P_1	P_2	L_p	L_p				
A_1	t_1	$t_2\ t_3\ t_4$	**5**	**19**	**49**	**0.66**	**3**	**3+α·48.96**
A_2	t_2	$t_1\ t_3\ t_4$	4	20	64	0.56	7	7+α·63.36
A_3	t_3	$t_1\ t_2\ t_4$	8	16	16	0.89	9	9+α·15.84
A_4	t_4	$t_1\ t_2\ t_3$	7	17	25	0.83	9	9+α·24.48
A_5	$t_1\ t_2$	$t_3\ t_4$	9	15	9	0.94	10	10+α·8.64
A_6	$t_1\ t_3$	$t_2\ t_4$	**13**	**11**	**1**	**0.99**	**6**	**6+α·1.44**
A_7	$t_1\ t_4$	$t_2\ t_3$	**12**	**12**	**0**	**1.00**	**12**	**12+α·0**
A_8	$t_1\ t_2\ t_3\ t_4$	ϕ	28	0	144	∞	0	0+α·∞

The result given in Table 1 are 8-assignments of 4 tasks to 2 processors. where the *weighting factor* α is some nonnegative constant to be chosen by the system designer. A good compromise between the two criteria can be made by setting α to an appropriate value. For example, if we want to put more emphasis on load balancing, we may set α to a higher value. By setting α to a lower value, one can put more emphasis on communication cost than on load balancing.

B. The Weighting Factor

Based on the proposed cost function one can represent the tradeoff between load balancing a communication cost by setting α to an appropriate value. There are two extreme cases in the task assignment problem. The one is to assign tasks with the objective of minimizing the communication cost only, without considering load balancing at all. The other is to assign tasks so as to achieve the maximum degree of load balancing. These two extreme cases may occur when one needs to assign highly communication-bound tasks and

highly computation-bound tasks, respectively. The proposed cost function can represent the objective of the first case by setting α to zero. In the second case, however, we have to estimate a certain value of α at which the maximum degree of load balancing (or best load-balancing) is achieved.

As α increases, the load-balancing degree of an optimal assignment increases until it becomes best load-balanced. Let $\hat{\alpha}$ be the smallest weighting factor with which optimal assignment are best load-balanced. We call $\hat{\alpha}$ the "*upper-bound*" of the weighting factor. Then, let $A_o^{\hat{\alpha}}$ be an optimal assignment with the weighting factor $\hat{\alpha}$

Definition 1: The *gain* in communication cost of an assignment A, denoted by $Comm_{gain}(A)$, is defined as

$$Comm_{gain}(A) = 1 - \frac{Comm(A)}{Comm(A_o^{\hat{\alpha}})}.$$

Definition 2: The *loss* in load balancing of an assignment A, denoted by $LB_{loss}(A)$, is defined as

$$LB_{loss}(A) = 1 - LB(A).$$

By setting α lower than $\hat{\alpha}$, we gain in communication cost but we lose in load balanced. To make a compromise between the two, we have to determine the weight on each of them according to the given task type. This compromise factor should be reflected in the cost function.

Definition3 :A compromise factor δ is valid if it satisfies the following two conditions :

$$if\ \ Comm_{gain}(A_1) - \delta \cdot LB_{loss}(A_1) > Comm_{gain}(A_2) - \delta \cdot LB_{loss}(A_2),$$
$$then\ \ COST(A_1) < COST(A_2)$$

$$if\ \ Comm_{gain}(A_1) - \delta \cdot LB_{loss}(A_1) = Comm_{gain}(A_2) - \delta \cdot LB_{loss}(A_2),$$
$$then\ COST(A_1) = COST(A_2)\ .$$

Theorem 1 : The cost function proposed in this paper is a δ-*policy* if

$$\alpha = \delta \cdot \frac{Comm(A_o^{\hat{\alpha}})}{\frac{N-1}{N^2} L^2}.$$

Proof : For any assignments A_1 and A_2, suppose

$$Comm_{gain}(A_1) - \delta \cdot LB_{loss}(A_1) > Comm_{gain}(A_2) - \delta \cdot LB_{loss}(A_2).$$

Then,

$$\begin{aligned} &COST(A_1) - COST(A_2) \\ &= Comm(A_1) - Comm(A_2) + \delta \cdot Comm(A_o^{\hat{\alpha}}) \cdot (LB(A_2) - LB(A_1)) \\ &= Comm(A_o^{\hat{\alpha}})((Comm_{gain}(A_2) - Comm_{gain}(A_1)) \\ &\qquad + \delta \cdot (LB_{loss}(A_1) - LB_{loss}(A_2))) \\ &< 0. \end{aligned}$$

Obviously, $COST(A_1) = COST(A_2)$ from *definition* 3. Therefore, it is a valid *weighting factor*.

To further investigate the effects of *compromise factor* δ on both load balancing and communication cost, 20 experiments were performed. In these experiments, 30 $\leq m \leq$50 (the number of tasks), and2 $\leq n \leq$4 (the number of processors) were picked randomly. An edge was added between any two nodes with the probability p_d. The weight of each edge and the size of each node were picked randomly between 0 &r_w and 0 &r_s, respectively. Also, in each experiment, p_d, r_w, and $r_s$were chosen randomly between 20 & 40, 1 & 10, and 5 & 15, respectively.

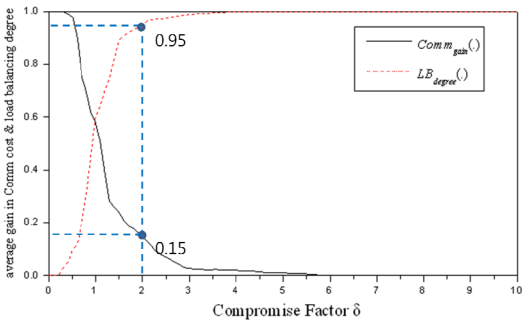

Fig. 2. Average $Comm_{gain}(\cdot)$ and $LB(\cdot)$ by compromise factor δ.

Figure.2 shows the average gain in communication cost, $Comm_{gain}(\cdot)$, and the average load-balancing degree, $LB(\cdot)$, of optimalassignment, as the value of δincreases from 0 to 10. As δincreases, $Comm_{gain}(\cdot)$ decreases while $LB(\cdot)$ increases, as expected. As shown in this figure, when δ= 2 we can obtain, on average, the gain of 0.15 in communication with the loss of 0.05 in load balancing.

IV.PROBLEM TRANSFORMATION

An *n*-cut of a graph *G* is a set of edges that partitions the nodes of *G* into *n* disjoint subsets $P_1, P_2, \cdots, P_n$. The weight of an *n*-cut is the sum of the weights of the edges in the *n*-cut. The size of each subset P_k is the sum of the sizes of nodes in P_k. There is a one-to-one correspondence between *n*-cuts of the TIG of a given task and assignments of the task to *n* processors. Then, the weight of an *n*-cut is equal to the communication cost and the load-balancing degree of size distribution among the *n* disjoint subsets is equal to the load-balancing degree of the corresponding assignment. Therefore, the task assignment problem considered here is equivalent to the problem of finding an *n*-cut of a graph with the objective of minimizing the *n*-cut weight as well as balancing the size of subset. In the original TIG, each cutest has no information on the size of the resulting subsets. Thus, it is difficult to develop efficient heuristics on the original graph, since one must consider the load-balancing degree of an existing partition as well as the cutest weight of the partition. Kernighan-Lin's(K&L) algorithm, for example, keeps an existing partition balanced by using pairwise-exchange of nodes of the same size given an initial balanced partition and tries to minimize the cutest weight. Thus, we can see that, in K&L algorithm, there exist two separate parts which consider communication and load balancing, respectively. If we modify the TIG such that the weight of each edge

contains the information on the size (i.e., running costs) of the corresponding nodes as well as the original weight of the edge, it gets easy for us to develop an efficient method for finding a tradeoff between the two objectives because both pieces of information are included in the weights of the edges in an *n*-cut after modification. We propose a modification technique for the TIG of a given task such that any cutest on the modified graph, say *G*, has the information on the load-balancing degree of the corresponding assignment as well as the original weights of the edges. For any two nodes m_i and m_j of the TIG :

1. if there is an edge between them then modify the weight of the edge to be $W_{ij} - x_i \cdot x_j \cdot (2\alpha / n)$
2. otherwise, make an edge between them which has the weight of - $-x_i \cdot x_j \cdot (2\alpha / n)$,

where $\alpha = \delta \cdot Comm\ (A_o^{\hat{\alpha}}) / (\frac{N-1}{N^2} L_{tot}{}^2)$.

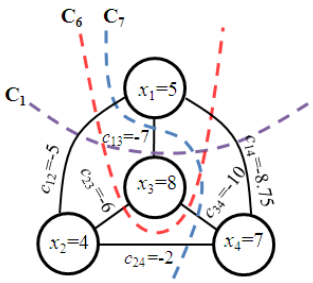

Fig. 3. An example of graph modification.

For example, for the TIG in Figure 1, the modified graph *G*with a compromise factor $\delta = 3$ is shown in Figure 3. Note that, in this example, $Comm(A_o^{\hat{\alpha}}) = 12$, i.e.,

$$\begin{aligned}\alpha &= \delta \cdot Comm\ (A_o^{\hat{\alpha}}) / (\tfrac{N-1}{N^2} L_{tot}{}^2) \\ &= 3 \times 12 / (\frac{2-1}{2^2} 24^2) \\ &= 0.25\end{aligned}$$

The 2-cuts C_1, C_6, C_7 in Figure 3 correspond to the assignments A_1, A_6, A_7 in Figure 1, respectively. An algorithm based on this transformation requires more storage since the initial potentially-sparse graph is transformed into an almost complete graph before the algorithm begins. However, with the resulting modified graph *G*, it becomes much easier to devise efficient heuristics for the task assignment problem since each edge has the information on both the execution and the communication costs. Since each *n*-cut on the modified graph *G* has the complete information on the total cost of its corresponding assignment, the task assignment problem considered here can be transformed into the problem of finding a minimum-weight *n*-cut in a graph which is called the *minimumn*-cut problem. This is proved in Theorem 4.

Theorem 2 : The task assignment problem can be transformed into the minimum *n*-cut problem.

Proof : Let an *n*-cut *C* of a modified graph *G* correspond to an assignment *A*. Suppose *C* partitions the nodes of *G* into *n* subsets $P_1, P_2, \cdots, P_n$. Let each subset P_k be $\{m_{k_i} : 1 \le i \le p_k\}$. Then the weight of *C*, $W(C)$, is as follows.

$$\begin{aligned}W(C) &= \sum_{k<l} \left(\sum_{\substack{1 \le i \le p_k \\ 1 \le j \le p_l}} \left(W_{k_i l_j} - x_{k_i} \cdot x_{l_j} \cdot \frac{2\alpha}{n} \right) \right) \\ &= \sum_{k<l} \sum_{\substack{1 \le i \le p_k \\ 1 \le j \le p_l}} W_{k_i l_j} - (\sum_{k<l} L_k \cdot L_l) \cdot \frac{2\alpha}{n} \\ &= Comm\ (A) - (\sum_{k<l} L_k \cdot L_l) \cdot \frac{2}{n} \cdot \alpha \\ &= COST\ (A) - \frac{n-1}{n^2} L_{tot}{}^2 \cdot \alpha .\end{aligned}$$

The second term of the last equation is constant for each task. Therefore, the task assignment problem is equivalent to the problem of finding a minimum-cost *n*-cut in a graph generated by the proposed graph modification technique. For example, the weights of C_1, C_6, C_7, in Figure 3 are -20.75, -29.75, and -12.0, respectively. $C_6$is the minimum weight 2-cut. Then the costs of their corresponding assignments are :

$$COST(A_1) = W(C_1) + \frac{n-1}{n^2} L_{tot}{}^2 \cdot \alpha = -20.75 + 144 \times 0.25 = 15.25,$$

$$COST(A_6) = W(C_6) + \frac{n-1}{n^2} L_{tot}{}^2 \cdot \alpha = -29.75 + 144 \times 0.25 = 6.25,$$

$$COST(A_7) = W(C_7) + \frac{n-1}{n^2} L_{tot}{}^2 \cdot \alpha = -24.0 + 144 \times 0.25 = 12.$$

Therefore, as shown in Figure 1, when $\delta = 3$, the best assignment is A_6 which corresponds to the minimum weight 2-cut C_6.

V. ASSIGNMENT ALGORITHM AND ANALYSIS

A. The minimum n-cut problem

When the value of δ is given, we have to find $A_o^{\hat{\alpha}}$ so as to determine the weighting factor α. Since it is computationally intractable to find such an optimal best load-balanced assignment, we have to resort to heuristics. This problem can be transformed to the minimum *n*-cut problem using the proposed graph modification technique by setting α to its upper bound $\hat{\alpha} = \max_i (\sum_{j=1}^{m} n \cdot W_{i,j} / 4x_i)$ .We first present a heuristic algorithm for the minimum *n*-cut problem. We wish to find a minimum weight *n*-cut C_o which assigns the nodes in *V* to *n* subsets (i.e., processors) $P_1, P_2, \cdots, P_n$.

Here, $W(C_o) = \min_C W(C) = \min_C \sum_{i,j} c_{i,j}$, where t_i and t_j are assigned different subsets. This is called the *minimum n-cut problem*.

The basic approach for the *minimum n-cut problem* is to start with an arbitrary assignment and to improve it by iteratively choosing one node in a processor and moving it to another processor. The node to be moved is chosen such that a maximum decrease in the cutest weight may be obtained (or minimum increase if no decrease is feasible). The algorithm consists of a series of passes (iterations): in each pass, every node should be moved only once. In each pass, the modes to be moved are chosen among those that have not yet been moved during the pass. The *m* assignments produced during the pass are examined and the one with the smallest cutest is chosen as the starting assignment for the next pass. Passes continue until no further improvement in the cutest weight can be made.

Definition 4: We define the *gain* $g(t_a, P_j)$ of each node t_a in P_i over $P_j (i \neq j)$ as the amount by which the cutset weight would decrease if t_a is moved from P_i to P_j ,i.e.,a node with the largest gain is s elected as the node to be moved.

For example, if the gain of t_a in P_i over P_j $g(t_a, P_j)$, is maximum, t_a will be moved from P_i to P_j, one-move operation. It will often be the case that the gain is non-positive. In that case, we still move the node with the expectation that the move will allow the algorithm to "climb out of local minima" as was done by Kernighan and Lin [1]. After all nodes have been forced to move, the best partition encountered during the pass is taken as the output of the pass.

To prevent the one-move process from "thrashing" or going into an infinite loop, we immediately "lock" each selected node in its new set for the remaining part of a pass. Thus, only "free" nodes are actually allowed to make one-move during the pass until all nodes are locked. After each pass, all nodes are released and hence become "free" again. After moving the selected node, we recalculate the gains of all the other free nodes. For example, if t_r in P_i is moved to P_j, the new gains of free nodes are as follows:

$$\hat{g}(t_a, P_j) = g(t_a, P_j) + 2c_{ar}, \quad for \ \ all \ \ t_a \in P_i$$

$$\hat{g}(t_a, P_l) = g(t_a, P_l) + c_{ar}, for \ \ all \ \ t_a \in P_i, \ 1 \le l(\neq i, j) \le n,$$

$$\hat{g}(t_b, P_i) = g(t_b, P_i) - 2c_{br}, for \ \ all \ \ t_b \in P_j,$$

$$\hat{g}(t_b, P_l) = g(t_b, P_l) - c_{br}, for \ \ all \ \ t_b \in P_j, 1 \le l(\neq i, j) \le n,$$

$$\hat{g}(t_c, P_i) = g(t_c, P_i) - c_{cr}, for \ \ all \ \ t_c \in P_l, 1 \le l(\neq i, j) \le n,$$

$$\hat{g}(t_c, P_j) = g(t_c, P_j) + c_{cr}, for \ \ all \ \ t_c \in P_l, 1 \le l(\neq i, j) \le n.$$

The correctness of these expressions is easy to verify.

B. The minimum n – cut algorithm

In the first step, compute the gains for all nodes of vertices for an initial assignment. This assignment may be obtained simply by assigning all the nodes randomly. Set all the nodes "free". In the second step, choose a free t_a in P_i such that

$$g_1 = g(t_a, P_j) = \sum_{t_b \in P_j} c_{ab} - \sum_{t_c \in P_i} c_{ac}$$

is maximum; and move t_a from P_i to P_j and lock it. For notational convenience, mark (t_a, P_i) as (t_1', P_1'). In the third step, recalculate the gains for all free nodes.

```
*Assign MinCut(graph *G) {

   Assign[M] : integer;   //t_i is assigned to P_Assign[i]
   gain[M][N] : real;   //gain[i][j] = g(t_i,P_j)
   state[M] : boolean;   //0=free, 1=locked state
   history[M][2] : integer;   //information on previous move operation
   temp[M] : integer;   //temporary gain for previous move operation

   construct an initial admissible assignment A;
   calculate gain[i][j] and set state[i] to 0 for all i, j {
      until all i-th nodes are moved {
         select free t_d in P_x with the maximum gain g(t_d, P_l);
         Assign[d]:=l, state[d]:=1; //move and lock t_d
          history[i][1]:=d. history[i][2]:=x, temp[i]:= gain[d][l];
          recalculate j-th gains of all unused elements {
             if state[j] = 1 then continue;
             if Assign[j] = x then gain[j][l]:= gain[j][l] + 2c_dj;
             else if Assign[j] = l then gain[j][x]:= gain[j][x] - 2c_dj;
             else gain[j][x]:= gain[j][x] - c_dj, gain[j][l]:= gain[j][l] + c_dj;
          }
      }
      choose k to maximum  T = sum_{i=1}^{k} temp[j]; // select the best assignment A'
      if an improvement is obtained, then {   //if T > 0 then
         remove each j-th element from (k+1)-th to make A'
            Assign[history[j][1]]:=history[j][2];
      else do   // if no improvement, then A is locally optimal
         remove each j-th element for all to make A {
            Assign[history[j][1]]:=history[j][2];
            Break;
         }
      //start next pass with the partition A'
   }
   //exit with the resulting assignment : Assign[M]
}
```

Fig. 4. The minimum *n*-cut algorithm:*MinCut.*

Now, repeat the second step, choosing a free node t_2' in P_2' with the maximum gain of g_2 .Thus, g_2 is the additional gain when node t_2' (as well as t_1') is moved, and this additional gain is maximum, given the previous choices. Move t_2' to the corresponding set and lock it. Continue this process until all free nodes have been exhausted, thus identifying (t_3', P_3'), …, (t_m', P_m'), and the corresponding maximum gains g_3, … , g_m .If $K = \{t_1', t_2', \cdots, t_k'\}$, then the decrease in weight when the nodes in the set K are moved is precisely $g_1 + g_2 + ... + g_k$. Note that some of the g_i 's may be negative.

Choose k that maximizes the partial sum $\sum_{i=1}^{k} g_i = T$. Now, if $T > 0$, a reduction in T can be made by moving the nodes in K. After this is done, the whole procedure is repeated from the first step with the resulting

assignment as the initial assignment. If $T \leq 0$, we have arrived at a local optimum assignment. We now have the choice of repeating with another starting assignment. The algorithm described thus far will be called *MinCut*, which is formally described in a Pascal-like algorithm in Fig. 4. Especially, an M-tuple implemented in the form of an array Assign[M] is used to describe the current assignment. The k-thelement of the array gives the processor number to which task t_k is assigned under the current assignment. History [M][2] is used to save the history of one-move operations. For example, when one-move on t_k is made from P_i to P_j history [k][1] = *j* and history [k][2] = *i*.

C. Time-Complexity Analysis

For time complexity analysis, we define a pass to be the operation involved in making one cycle from step 2 to step 5 of *MinCut*. The computing time needed for step 2 is $O(NM^2)$ since we need an $O(M)$ time to compute the gain of each node over each processor. Each iteration of step 3 needs an $O(NM)$ computing time due mainly to the selection of a node with the largest gain. Thus, the total time needed for step 3 is $O(NM^2)$. The computing time of $O(M)$ is sufficient for step 4 and 5. Therefore, the total computing time for a pass is $O(NM^2)$.

The number of passes required for *MinCut* to terminate is small. In our experiments on graphs with up to 200 nodes and with various values of N (up to 20), it has almost always been between 3 and 6. From these experiments, we can see that the number of passes does not strongly depend on the value of *M*.

D. The Assignment Algorithm

To determine the weighting factor a, we have to find $A_o^{\hat{\alpha}}$ which corresponds to a minimum *n*-cut of the graph G' resulting from the modification of the original TIG with $\alpha = \hat{\alpha}$. After determining α, the final assignment can be found on the graph *G* modified from TIG with the value of α determined. Therefore, we have to run *MinCut*exactly twice to find the final assignment. The assignment algorithm *TaskAssign* is shown in Fig. 7.

```
*Assign  TaskAssign(TIG, δ) {
   modify graph TIG to G' with  α̂ = max_i(Σ_{j=1}^{m} n·W_{i,j} / 4x_i);
   call MinCut(G') to find the optimal assignment  A_o^α̂ ;
   calculate Comm(A_o^α̂) through Comm= Σ_{i=1}^{m-1} Σ_{j=i+1}^{m} c_ij ;
   modify graph TIG to G with  α = δ · Comm /((N-1)/N^2 · L_tot^2);
   call MinCut(G) to find the final assignment;
   // exit with the final assignment : Assign[]
}
```

Fig. 5. The optimal task assignment : *TaskAssign.*

To determine the weighting factor α, we have to find $A_o^{\hat{\alpha}}$ which corresponds to a minimum *n*-cut of the graph G' resulting from the modification of the original TIG with $\alpha = \hat{\alpha}$. After determining α, the final assignment can be found on the graph *G* modified from TIG with the value of α determined. Therefore, we have to run *MinCut*exactly twice to find the final assignment. The assignment algorithm ***TaskAssign*** is shown in Figure 5.

VI. EXPERIMENTAL RESULT AND ANALYSIS

A number of experiments were performed to evaluate the performance of *MinCut*. The experiments were performed for the task assignment problem with the constraint of perfectload balancing, i.e., to assign tasks so as to minimize communication cost while maintaining perfect loadbalancing. This problem is equivalent to the problem of perfectly balanced *N*-way partitioning of graphs [5]. On the basis of the experimental evidence as previously described, $\alpha$is set to its approximate upper bound.

We have performed a number of experiments for regular and random graphs. For regular, we performed two experiments for the cases of 10 × 10 two-dimensional meshes and full binary trees of depth 7, respectively, to compare our algorithm with K&L for the balanced two-cut problem. For each experiment,20 graphs were tested. Also, the experiments in case of $N = 4$ were performed for random graphs to compare our heuristic algorithm with K&L algorithm adapted to the general balanced *n*-cut problem.

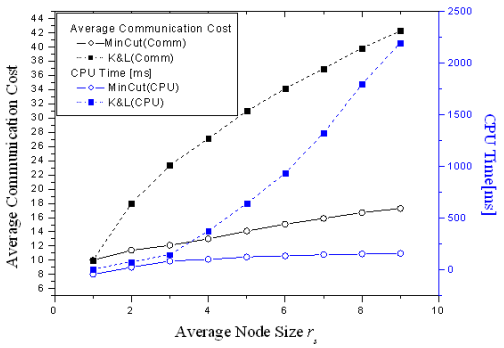

Fig. 6.10 × 10 2DMesh: Regular TIG.

The size of each node was picked randomly between0 and r_s. Each experiment is divided into nine cases: r_s = 1, 2,…,9. For all generated graphs, the weight of each edge was set to one in order or evaluate the performance in terms of the number of edges in the final cutest.

On all graphs tested, both K&L and our algorithms always found perfectly balanced partitions. The main performance measure is therefore the cost of the resulting cutsets, i.e., the communication cost.

The experimental results for meshes and trees are summarized in Figure 6 and Figure 7. For each experiment, both K&L and our algorithm always found optimal solutions when $r_s = 1$.However, as can be seen in the figure, our algorithm significantly outperforms K&L as the size difference among the nodes increases.

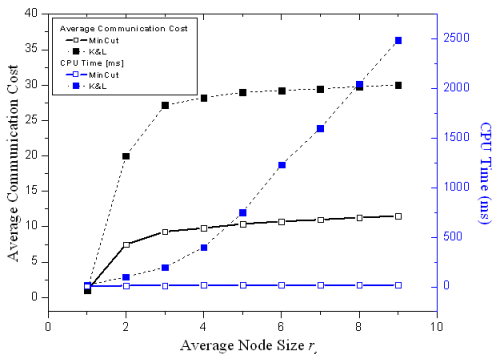

Fig. 7. depth-7 Tree Regular TIG.

For random, our experiment was performed for the task assignment problem with two conflicting objectives, i.e., minimizing communication cost and balancing loads. To compare our "integrated" approach with an existing approach in which minimizing communication cost is the only optimization criterion and a balancing condition is given as a constraint, we modified the "pair wise-exchange" scheme of K&L algorithm as: any two different-size nodes can be candidates to be exchanged if the balancing constraint is satisfied after exchanging them.

The experiments for N= 4 were performed for random graphs. We compared the solution quality of our *MinCut* with that of modified K&L for compromise factor δ(0 to 10). For the experiment, 100 random graphs were generated with 100 nodes and about 1500edges where the weight of each edge was set to one.

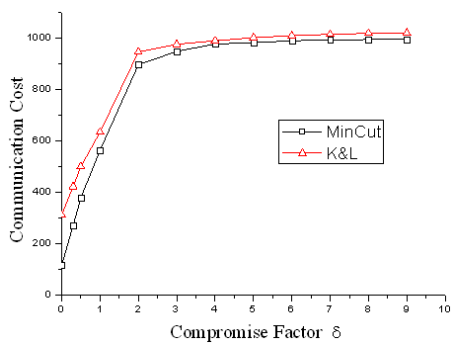

Fig. 8. Comm. Cost Comparison by ***δ.***

The experimental results for random graphs are summarized in Figure 8 and Figure 9. Our algorithm is shown to outperform K&L in solution quality even when $r_s = 3$. Also, the time consumed by our algorithm is less than that in K&L.

As shown in these figures, the average communication cost for MinCut is much less than that for K&L. The performancegap becomes large as the balancing constraint becomes loose, i.e., as δdecreases.

From these results of the experiment, we can see that our integrated approach is better than the existing approach for the task assignment problem considered in this paper.

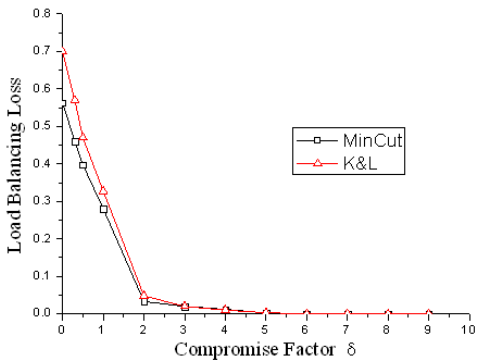

Fig. 9. Load Balancing Loss by ***δ.***

VII. CONCLUSION

Our investigation of the static task assignment problem has resulted in the development of an "integrated" approach which systematically resolves the two conflicting goals, i.e., minimizing communication costs and balancing loads. This was done by using the variance statistics of load distribution to represent the degree of load balancing among processors. We proposed a new cost function which represents the inherent tradeoff between the two goals by combining them into a single objective function. We also proposed the weighting and compromise factors with which one can systematically make a compromise between the two conflicting goals according to the underlying task type. The task assignment problem has been shown to be transformed into the minimum n-cut problem which has only one objective, whereas the original problem has two objectives to optimize. Also, we have developed a heuristic algorithm for the transformed problem. Experimental results indicate that the algorithm performs quite well on a variety of task-processor systems.

Task assignment still poses to offer a variety of challenging problems such as assignment with resource relocation, real-time constraints, communication link loads, heterogeneous multi-processors, not to mention the precedence relationships and load balancing. While much work has been done dealing with each of these problems, it is interesting to extend our results by increasing the complexity of the model to include such factors

REFERENCES

[1] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, pp. 291-307, Feb. 1970.

[2] H. S. Stone, "Critical load factors in distributed computer systems," *IEEE Trans. Software Eng*., vol. SE-4, pp. 254-258, May 1978.

[3] S. H. Bokhari, "A shortest tree algorithm for optimal assignments across space and time in a distributed processor systems," *IEEE Trans. Software Eng.*, vol. SE-7, pp. 583-589, Nov. 1981.

[4] S. H. Bokhari, "Assignment Problems in Parallel and Distributed Computing," *Kluwer Academic Publishers*, Boston, 1987.

[5] S. Pulidas, D. Towsley, and J. A. Stankovic, "Embedding gradient estimators in loadbalancing algorithms," *Proc. 8th Int. Conf. Distributed Comput. Syst.*, pp. 482-490,1988.

[6] C. -H. Lee, C. -I. Park, and M. Kim, "Efficient algorithm for graph-partitioning problem using a problem transformation method," *Computer-Aided Design*, vol. 21, pp. 611-618, Dec. 1989.

[7] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Trans. Comput*., vol. C-7-37, pp. 1384-1397, Nov. 1988.

[8] C. -H. Lee, K. G. Shin, "Optimal task assignment in Homogeneous Networks," *IEEE Trans. on Parallel and Distributed Systems*., vol.8, No.2, pp. 119-129, Feb 1997.

[9] J. -M. Kim, C. -H, Lee, "A Repeated Mapping Scheme of Task Modules with Minimum Communication Cost in Hypercube Multicomputers," *ETRI Journal.*, vol.20, No.4, pp.327-245, Dec, 1998.

[10] A. Salman, I. Ahmad, S. Al-Madani, "Particle swarm optimization for task assignment problem," Microprocessors and Microsystems 26, pp.363-371, 2002.

[11] J. -M.Kim, "Task assignment to consider the communication costs and the load balancing in Distributed and Parallel Systems," *PhD thesis.*, Chungnam National University, Aug. 2003.

[12] B. Ucar, C. Aykanat, K. Kaya and M. Ikinci, "Task assignment in heterogeneous computing systems," j. Parallel Distrib. Comput. 66, pp.32-46, 2006.

[13] J. –M. Orduna, F. Silla and J. Duato, "On the development of a communication-aware task mapping technique," J. Systems Archit. 50(4), pp.207-220, 2004.

[14] M. Kafil, I. Ahmad, "Optimal Task Assignment in Heterogeneous Distributed Computing Systems," IEEE Concurrency, vol. 6 no. 3, pp.42-51, Jully-Sept. 1998.

[15] A.P. Tom, C.S.R. Murthy, "Optimal task allocation in distributed systems by graph matching and state space search," J. Systems and Software 46 (1) pp.59–75, 1999.

Joo-Man Kim received the B.S. degree in computer science from Soongsil University, Seoul, Korea in 1984 and the M.S. and Ph.D. degrees in computer engineering from Chungnam National University, Daejeon, Korea in 1998 and 2003, respectively. From 1985 to 2000, he worked for ETRI in Daejeon, Korea as principal member for research staff at the OS research team. From 2000 to 2005, he was an assistant professor in the Dept. of Informations and Communications Engineering, Miryang National University. He is currently an associate professor in the Dept. of Applied IT and Engineering, Pusan National University, Pusan, Korea. His research and teaching interests are in embedded system, real-time and ubiquitous computing.