J Indian Inst Sci, July–Aug. 1990, 70, 305–315
© Indian Institute of Science.

Static allocation of communicating processes for distributed computing systems with resource heterogeneity

R. SATYANARAYANAN AND C. R. MUTHUKRISHNAN
Department of Computer Science and Engineering, Indian Institute of Technology, Madras 600036, India

Received on October 11, 1989; Revised on April 19, 1990.

Abstract

Allocation of processes to the processing nodes in such a way as to minimise interprocessor communication and also ensure load balancing is a major problem in distributed computing systems The problem is even more complicated if the system is heterogeneous In this paper we have presented the affinity graph model for performing process allocation for distributed computing systems and augment the model to cover distributed systems with heterogeneity of resources The main advantages of the affinity graph model are its ability to reflect both the load balancing as well as the minimisation of interprocessor communication criteria in a single representation and the facility for varying the weightage given to the above two criteria We have presented an algorithm for performing allocation for a two-processor system and also briefly illustrate how allocation can be done for systems with a larger number of processors by using a binary-tree-structured system as an example

Key words: Augmented affinity graph, distributed computing systems, heterogeneity, load balancing, static allocation

1. Introduction

Distributed computing systems are coming into use in large numbers because of their manifold advantages of reliability, availability, fault tolerance, and increased throughput and performance due to the exploitation of parallelism. But they suffer from a major disadvantage, *viz.*, the *saturation effect*[1]. The saturation effect arises because of excessive interprocessor communication. Interprocessor communication is due to the communication requests made by processes residing on processing nodes for communicating with processes residing on other processing nodes. Interprocessor communication is a function of allocation of processes to processors and the communication requests made by the processes. So, to minimise communication among processors, we have to allocate the processes as close as possible. Ideally, we have to allocate all the processes to a single processor so that interprocessor communication is nil. But such an allocation is the worst possible from the load-balancing point of view (by load balancing we mean providing as equal a load as is possible to all the processors); hence, one has to allocate the processes as evenly as possible to all the processors. One can visualise that the load-balancing criterion and the

minimisation of interprocessor communication criterion are two competing factors. A good allocation scheme is one which balances these two competing factors in an optimal manner.

A number of suggestions have been made for solving the allocation problem. The solutions can be classified into three main categories, *viz.*, heuristic methods, integer programming approach, and graph-theoretic methods[1]. Ma *et al*[2] have proposed an allocation scheme based on the branch-and-bound method. In this scheme, starting from process 1, each process is allocated one of the processors subject to the constraints imposed on the relations between processes and processors. Shen and Tsai[3] have proposed an allocation model based on graph-matching approach where each graph match corresponds to a specific allocation. Minimax criterion is used to minimise the cost function which is based on a single unit, *viz.*, time. A state-space-search method is employed to find an optimal allocation corresponding to minimum cost matching. Chou and Abraham[4] have proposed an algorithm based on results in Markov decision theory for optimal allocation. Stone[5] has proposed a graph-theoretic model and has studied the problem of optimally partitioning a modular program over a dual-processor system so as to minimise the total running cost of the program. Extension of Stone's model for a large number of processors results in an intractable allocation algorithm[1]. Bokhari[6] has extended Stone's results for dynamic allocation on a two-processor system.

The problem of process allocation to heterogeneous systems is much tougher compared to allocation to homogeneous systems. In the case of heterogeneous systems, one has to take into account not only the load balancing and minimisation of interprocessor communication criteria, but also of the fact that different nodes have different facilities and resources. A given process may require any subset of the resources distributed throughout the system. Allocation has to take into account this important fact and try to minimise the overhead due to the usage of remote resources by the processes. Most of the allocation schemes available are meant for homogeneous systems only.

In this paper, we have presented a model for process allocation called the *affinity graph model*. We have also presented ways of augmenting the affinity graph model taking into account the heterogeneous nature of the distributed computing system arising out of the distribution of resources in the system. We have presented an algorithm which takes the affinity graph model of the processes to be allocated and finds an allocation for a two-processor heterogeneous system. We have also shown how the allocation can be done for systems with a larger number of processors by using a binary-tree-structured distributed computing system as an example. The advantages of the affinity graph model include its ability to express both the load balancing and minimisation of interprocessor communication criteria in a single representation and the facility for varying the weightage given to these two criteria. The affinity graph model has, as its vertices, the processes to be allocated. The weight of the edges connecting the vertices represent the affinity the processes have for each other. The affinity function has been defined in such a way that both the competing demands of load balancing and minimisation of interprocessor communication are taken care of. The affinity graph can even be used for allocation for systems with a large number of processors like hypercube systems of large dimensions[7].

2. Definitions and assumptions

Partitioning a graph is the division of the nodes of the graph into disjoint subsets called *blocks*. If we represent a graph by a matrix A with $A(i,j)$ being the weight of the edge linking vertex i and vertex j, and partition the vertices into two blocks, $S1$ and $S2$, then the *internal* and *external costs* of an element $x \in S1$ are $\Sigma_{y \in S1} A(x,y)$, and $\Sigma_{y \in S2} A(x,y)$, respectively.

The following assumptions have been made in this paper. There are no precedence relations among the processes to be allocated. A measure of the processing cost of the processes, the amount of communication expected to take place among the processes, and the usage of the resources in the distributed computing system by the processes are assumed to be known. The heterogeneity among the processing nodes arises because of the non-uniform set of resources available with the various processing elements.

3. The augmented affinity graph model for process allocation

If it is possible to find out for each pair of processes a measure of the *affinity* the processes have for each other (*affinity* between two processes is the amount by which allocation of the two processes together contributes to both balancing the load and minimising interprocessor communication), then by allocating a process depending on the affinity of the process for other processes it is possible to get an allocation satisfying the load-balancing and minimisation of interprocessor communication criteria. Processes with less affinity are allocated less close compared to processes with large affinity for each other which are allocated as close as possible. This is the basic idea behind the formation of the affinity graph.

Let there be p processes to be allocated on n processors. Let P_c be a row matrix of p elements such that $P_c(i)$ is the processing cost of process p_i. Let $DP_{c_{pxp}}$ be a matrix of processing cost difference such that $DP_c(i,j) = |P_c(i) - P_c(j)|$. We can form a graph of processing cost difference with p vertices. The weight associated with an edge (i,j) is the value of $DP_c(i,j)$. The vertices i and j represent the processes p_i and p_j, respectively. In the graph of processing cost difference, the weight of an edge connecting a process requiring high processing cost and another process requiring low processing cost is high whereas the weight of an edge between two processes requiring large processing costs is low. In other words, processes with large processing cost have more affinity for processes with low processing cost, with the amount of affinity being in direct proportion to the difference of processing costs, whereas processes requiring large processing costs have less affinity for each other. The graph of processing cost difference reflects the load-balancing criterion.

We can form a communication matrix C_{pxp} such that $C(i,j)$ is the amount of communication expected to take place between processes p_i and p_j. In the corresponding communication graph, we represent the processes by vertices with the weight associated with an edge (i,j) being the measure of communication expected to take place between processes p_i and

p_j. It is clear that in the communication graph, the processes which communicate heavily have larger affinity for each other compared to processes with less communication between them. By allocating together processes with large affinity for each other in the communication graph, we can reduce interprocessor communication. Thus the communication graph reflects the minimisation of interprocessor-communication criterion.

We can form an *affinity graph* from the graph of processing cost difference and the communication graph as follows. We define an affinity matrix A_{pxp} such that

$$A(i,j) = \alpha \ DP_c(i,j) + \beta \ C(i,j).$$

Here, the constants α and β serve as both normalisation constants (since processing cost and interprocess communication are measured in different units) and also as scale factors enabling us to give different weightages to the load balancing and minimisation of interprocessor communication criteria. We form the affinity graph with p vertices representing processes $p_1, p_2, p_3, \ldots, p_p$ with the weight of the edge linking the vertices representing p_i and p_j being $A(i,j)$. This graph has been termed as *affinity graph* because the amount of affinity two processes have for each other in the affinity graph is directly related to the amount the allocation of these two processes together contributes towards both load balancing as well as minimisation of interprocessor communication. The more the affinity the two processes have for each other, the better it is to allocate the two processes together. The first major advantage of the affinity graph representation is that it is able to express both the load-balancing criterion as well as minimisation of interprocessor-communication criterion in a single representation. The second advantage is the facility for varying the weightage given to the above two criteria. The constants α and β serve this purpose. The process allocation algorithms should exploit the affinity information available in the affinity graph model of the processes. This can be done by partitioning the affinity graph. The actual allocation depends heavily on how well the affinity information is exploited. Now, we suggest ways of augmenting the affinity graph to take into account the non-uniformity of resources among the processing nodes of the distributed computing system. We will explain here the methodology with respect to a two-processor distributed computing system. Extension of the model for a system with more number of processors is also similar as can be seen from the discussion of the allocation algorithm for a binary-tree-structured distributed computing system in the next section.

It has been assumed that a measure of the resource usage by the processes is known. Some of the resources may be available on both the processing nodes while some may be available on only one of the two nodes in the system. The latter situation introduces heterogeneity into the system. Let DR be the set of resources present in either of the two processing nodes but not on both. If R_1 is the set of resources on processing node 1 and R_2 is the set of resources on node 2, then $DR = R_1 \cup R_2 - (R_1 \cap R_2)$. A process p_i may access only resources in $R_1 \cap R_2$. The allocation of these processes does not depend directly on the heterogeneity of the system. Some processes may access the resources in DR. It is the allocation of these processes which depend directly on the heterogeneity of the system. Depending on the usage of the resources in DR by these processes, the allocation has to

be performed so as to minimise the overhead associated with the usage of remote resources by the processes. Let the information regarding the usage of the resources by the processes be in a matrix U_{pxm} where $m = |R_1 \cup R_2|$, such that $U(i, j)$ is a measure of usage of resource r_j by process p_i. If a process p_i makes heavy use of resources in $R_1 - R_2$ compared to the resources in $R_2 - R_1$, then to minimise the overhead due to separation of resources from processes, it is better to allocate process p_i on processing node 1. So, we can augment the affinity graph as follows.

Add all elements of DR as vertices in the affinity graph. For all $r_i, r_j \in R_1 - R_2$, connect the vertex representing r_i and that representing r_j by an edge with weight ∞. Similarly, for all $r_i, r_j \in R_2 - R_1$, connect the vertex representing r_i and that representing r_j by an edge with weight ∞. For all processes p_i using resource $r_j \in DR$, connect the vertex representing p_i with the vertex representing resource r_j with an edge of weight $\gamma\ U(i, j)$. The constant γ serves the same function as the constants α and β. The resources in $R_1 - R_2$ are not linked to the resources in $R_2 - R_1$ and *vice versa*. The purpose of linking the resources within $R_1 - R_2$ and similarly that within $R_2 - R_1$ by edges with weight ∞ is to ensure that these edges are not cut while obtaining the partitioning of the augmented affinity graph. If a process p_i is linked to elements of $R_1 - R_2$ with edges of more weight compared to the edges linking the process p_i with the elements of $R_2 - R_1$, then allocating p_i to processor 1 will result in less overhead because of the utilization of the resources. We can impose the condition that a given process must be processed on the processing node having a particular resource by making the weight of the edge linking the resource and the process as ∞. In the next section we present allocation algorithms making use of the augmented affinity graph for a two-processor distributed computing system and a binary-tree-structured system.

Example 3.1: Let us consider a collection of six processes, *viz.*, $p_1, p_2, \ldots, p_6$. Let the P_c vector and communication matrix C be as given below. Since C is a symmetric matrix, only the elements along the main diagonal and those below it are given.

$$P_c = \begin{array}{c} \\ [\end{array} \begin{array}{cccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \\ 100 & 110 & 50 & 20 & 60 & 15 \end{array} \begin{array}{c} \\ ]; \end{array}$$

$$C = \begin{array}{c|cccccc} & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \\ \hline p_1 & 0 & & & & & \\ p_2 & 10 & 0 & & & & \\ p_3 & 100 & 30 & 0 & & & \\ p_4 & 150 & 25 & 160 & 0 & & \\ p_5 & 50 & 70 & 30 & 20 & 0 & \\ p_6 & 45 & 85 & 20 & 10 & 75 & 0 \end{array}.$$

Let the resources r_1 and r_2 be in processing node 1 and let r_3 and r_4 be in node 2.

Let the U matrix be

$$U = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{array} \begin{array}{c} \begin{array}{cccc} r_1 & r_2 & r_3 & r_4 \end{array} \\ \begin{bmatrix} 10 & 0 & 30 & 40 \\ 45 & 50 & 10 & 15 \\ 0 & 50 & 50 & 65 \\ 10 & 10 & 20 & 20 \\ 30 & 0 & 0 & 0 \\ 40 & 60 & 10 & 20 \end{bmatrix} \end{array}.$$

The augmented affinity matrix A with $\alpha=1$, $\beta=2$, and $\gamma=1$ is given below. Only the elements along the main diagonal and those below it are given.

$$A = \begin{array}{c|cccccccccc} & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & r_1 & r_2 & r_3 & r_4 \\ \hline p_1 & 0 \\ p_2 & 30 & 0 \\ p_3 & 250 & 120 & 0 \\ p_4 & 380 & 140 & 350 & 0 \\ p_5 & 140 & 190 & 70 & 80 & 0 \\ p_6 & 175 & 265 & 75 & 25 & 195 & 0 \\ r_1 & 10 & 45 & 0 & 10 & 30 & 40 & 0 \\ r_2 & 0 & 50 & 50 & 10 & 0 & 60 & \infty & 0 \\ r_3 & 30 & 10 & 50 & 20 & 0 & 10 & 0 & 0 & 0 \\ r_4 & 40 & 15 & 65 & 20 & 0 & 20 & 0 & 0 & \infty & 0 \end{array}.$$

4. Allocation using the augmented affinity graph

The allocation of the processes for the processing nodes of a distributed computing system can be performed by making use of the affinity information present in the augmented affinity graph. In the next two sub-sections, we present an algorithm for allocation on a two-processor distributed computing system and an informal discussion of extending the allocation scheme for systems with a larger number of processors by using a binary-tree-structured system as an example.

4.1. *Allocation algorithm for a two-processor distributed computing system*

Having formed the augmented affinity graph of the processes to be allocated, we have to partition the augmented affinity graph into two, assigning one block for each of the two processing nodes. The partitioning has to be performed in such a way as to minimise the cost of the edges cut during partitioning. All the members of R_1-R_2 will be in one block while those of R_2-R_1 will be in the other. This is because the weight of the edges linking the members of R_1-R_2, as also of R_2-R_1, is ∞. Since the partitioning is performed so as to minimise the weight of the edges cut, the edges with weight ∞ are never cut. The block

containing the members of R_1-R_2 is assigned to processing node 1, while the block containing the members of R_2-R_1 is assigned to processing node 2. The processes in each block are to be processed on the processor to which the block has been assigned. So, the allocation is basically a partitioning process on the augmented affinity graph. The allocation algorithm based on partitioning exploits the information contained in the affinity graph. The nature of the actual allocation heavily depends on the partitioning algorithm used. The algorithm presented here is based on Kernighan and Lin's graph-partitioning algorithm[8] along with a method of obtaining a starting solution. The allocation algorithm makes use of the P_c vector and the augmented affinity graph in the form of an augmented affinity matrix A with $(p+r)\times(p+r)$ elements where the r additional elements are the resources in DR. In the augmented affinity matrix, $A(i,p+j)=A(p+j,i)=\gamma U(i,j)$ where $A(i,p+j)$ is the measure of the affinity between process p_i and resource r_j. The augmented affinity matrix entries corresponding to the weight of the edges between resources in R_1-R_2 are set to ∞. Similarly, the weight of the edges between the resources in R_2-R_1 are set to ∞. The algorithm for allocation is as below.

STEP 1: In this step we form a starting solution for Kernighan and Lin's graph-partitioning algorithm.

(a) Form two sets S_1 and S_2 such that S_1 has all the members of R_1-R_2 and S_2 has all the members of R_2-R_1.

(b) Select the process p_i requiring the maximum processing cost, *i.e.*, $P_c(i)=\text{MAX}_{1\leqslant j\leqslant p}\, P_c(j)$. Allocate this process p_i to the set which has the maximum affinity for it. That is p_i is allocated to set S_1 if $\Sigma_{x\in S_1} A(i,x)>\Sigma_{y\in S_2} A(i,y)$. Otherwise p_i is allocated to S_2.

(c) Select the process $p_j (p_i\neq p_j)$ for which p_i has the least affinity, *i.e.*, $A(i,j)=\text{MIN}_{1\leqslant k\leqslant p}\, A(i,k)$. Allocate p_j to the set other than the one to which p_i has been allocated. $S=S-\{p_i,p_j\}$.

(d) The turn for selecting the next member is that of the set, the sum of processing cost of the processes in which it is less than the sum of processing cost of the processes in the other.

(e) The set which has the turn, selects from the set of unselected processes S a process p_i as its next member process, such that the (internal cost − external cost) of process p_i is the maximum among the processes in S, if p_i is made a member of the set with the turn for selection. That is, if S_1 has the turn to select the next process for it, then it will select p_i if $(\Sigma_{x\in S_1} A(i,x)-\Sigma_{y\in S_2} A(i,y))=\text{MAX}_{a\in S}\,(\Sigma_{x\in S_1} A(a,x)-\Sigma_{y\in S_2} A(a,y))$.

(f) The process p_i selected in STEP 1 (e) is removed from the set S of unselected processes, *i.e.*, $S=S-\{p_i\}$. The process p_i is added to the set which selected it.

(g) If there are no more processes to be selected, that is when S is empty, then go to STEP 2 else go to STEP 1 (d).

STEP 2: For all elements $a\in S_1$, calculate $E_a=\Sigma_{y\in S_2} A(a,y)$ and $I_a=\Sigma_{x\in S_1} A(a,x)$. Also calculate $D_a=E_a-I_a$. Similarly for all elements $b\in S_2$, calculate $E_b=\Sigma_{y\in S_1} A(b,y)$, $I_b=\Sigma_{x\in S_2} A(b,x)$, and $D_b=E_b-I_b$.

STEP 3: $m=1$; $S_1^m=S_1$; $S_2^m=S_2$.

STEP 4: Select $a_i\in S_1^m$ and $b_j\in S_2^m$ such that $g_m=D_{a_i}+D_{b_j}-2A(a_i,b_j)$ is maximum.

STEP 5: $a'_m=a_i$; $b'_m=b_j$; $S_1^{m+1}=S_1^m-a_i$; $S_2^{m+1}=S_2^m-b_j$.

STEP 6: If $m < (\text{MIN}\ (|S_1|, |S_2|))$ then $\{m = m+1$; update D values for S_1^m and S_2^m, go to STEP 4$\}$

STEP 7: Choose k to maximise $G = \Sigma_{i=1}^{k} g_i$.

STEP 8: If $G > 0$ then move $a'_1, \ldots, a'_k$ to S_2 and $b'_1, \ldots, b'_k$ to S_1 and go to STEP 2.

STEP 9: Allocate all processes in block S_1 to processing node 1 and all processes in block S_2 to processing node 2.

In this algorithm, STEP 1 finds a starting solution for the remaining steps in the algorithm. Steps 2 to 9 try to perturb the starting solution to find if there is any improvement possible in minimising the weight of the edges connecting the elements of block S_1 with those of S_2. The perturbation is carried on until no further improvement in reducing the weight of the edges connecting block S_1 to block S_2 is possible. At this point, the algorithm halts and the processes in block S_1 which contain the members of $R_1 - R_2$ are allocated to processing node 1 while the processes in block S_2 which contain the members of $R_2 - R_1$ are allocated to processing node 2. If the two processors have different processing powers, then we can remove processes with large values of (external cost − internal cost) from the block corresponding to the processor with less powerful processor and add them to the block corresponding to the more powerful processor, with the amount of such overloading being directly proportional to the excess of processing power.

In the above algorithm, STEP 1 (e) has a time complexity $O(p^2)$. Since it is repeated $O(p)$ times, the time complexity of STEP 1 is $O(p^3)$. Steps 2 to 9 are derived from Kernighan and Lin's algorithm and could be performed with a complexity of $O(p^2 \log p^8)$. So, the time complexity of the whole algorithm is $O(p^3)$.

Example 4.1.1: Consider the set of processes given in example 3.1. If we apply the algorithm presented in this section, we get an allocation of p_2, p_5, and p_6 to processing node 1 and p_1, p_3, and p_4 to processing node 2.

4.2. *Allocation of processes for a binary-tree-structured distributed computing system*

In this section, we informally describe an allocation algorithm using the augmented affinity graph model for binary-tree-structured distributed computing systems. The binary-tree-structured system we consider here consists of processing nodes at the leaves of a full binary tree. The non-leaf nodes of the tree are communication processors. Such a binary-tree-structured architecture for distributed computing systems is highly useful when a distributed system is to be constructed from a number of existing single processing nodes. Figure 1 shows such a tree-structured system with four processing nodes, *viz*., PROC_1, PROC_2, PROC_3, and PROC_4 and three communication nodes, *viz*., COMM_P_1, COMM_P_2, and COMM_P_3. Each processing node has a set of resources. It is easy to visualise that in the tree-structured system, the cost of communicating with processing nodes within a given subtree is always less than the cost of communicating with a node outside that subtree. For example, the cost of communication between PROC_1 and PROC_2 is less than the cost of communication from PROC_1/PROC_2 to PROC_3/PROC_4. The allocation for

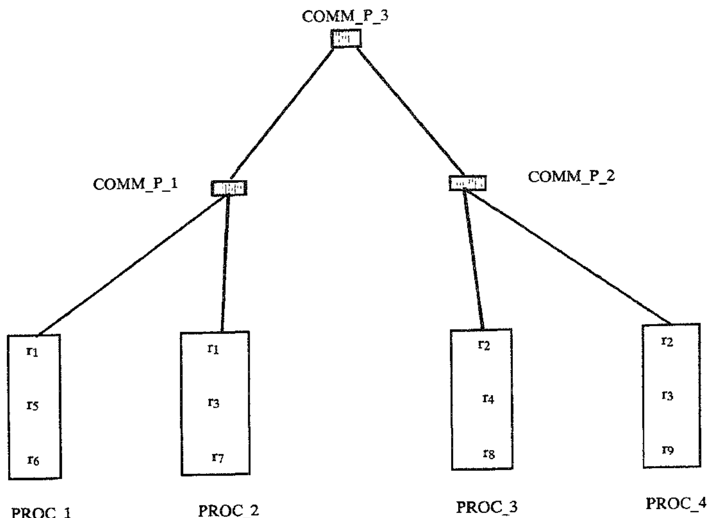

FIG. 1. A binary-tree-structured distributed computing system.

such a tree-structured system is a partitioning of the augmented affinity graph. For a tree-structured system of height n, the augmented affinity graph is partitioned into two, assigning one block to the left subtree of height (n-1) and the other to the right subtree of height (n-1). Two augmented affinity graphs are formed from the two blocks obtained as above. These two augmented affinity graphs are again partitioned assigning one block each for each of the four subtrees of height (n-2). This process is repeated until we get one block each for each of the trees of height 0, *i.e.*, the individual processing nodes. The processes in each block are allocated to the processing node corresponding to the partition. An important point one has to note is that the augmented affinity graph and the set DR change after each level of allocation. Resources linked by edges with weight ∞ may not be linked together in a lower level augmented affinity graph. Each allocation consists of the following basic stages.

Stage 1: Form the augmented affinity graph for the processes. The links among the resource nodes are formed as follows. Let R_{ls} be the set of resources in the left subtree of the current tree for which allocation is to be performed. Similarly, let R_{rs} be the set of resources in the right subtree of the current tree for which allocation is to be performed. The set DR for this allocation is given by $DR=(R_{rs} \cup R_{ls})-(R_{rs} \cap R_{ls})$. Form the augmented affinity graph

for the processes assigned to this tree. In this augmented affinity graph, for all r_i, $r_j \in R_{ls}$ link r_i, r_j with edges of weight ∞. Similarly for all r_i, $r_j \in R_{rs}$ link r_i, r_j with edges of weight ∞.

Stage 2: Perform a two-way partitioning of the augmented affinity graph in the same way as in section 4.1, where all operations are performed with respect to the processes assigned to the current tree. Let P_{ls} be the block containing the elements of $R_{ls} - R_{rs}$ and let P_{rs} be the block containing the elements of $R_{rs} - R_{ls}$. Assign the processes in P_{ls} to the left subtree of the current tree. Assign the processes in P_{rs} to the right subtree of the current tree.

The above two stages are repeated until the left and the right subtrees are individual processing elements. As an example, consider the tree-structured system in fig. 1. When the allocation is started, in the initial augmented affinity graph, resources r_1, r_5, r_6, and r_7 are linked with edges of weight ∞. Similarly, resources r_2, r_4, r_8, and r_9 are linked with the edges of weight ∞. After the first level of allocation, we get two blocks, one each for each of the subtrees with roots COMM_P_1 and COMM_P_2. When the allocation for the tree with COMM_P_1 at the root is performed, the edge connecting resources r_5 and r_6 has a weight of ∞ in the augmented affinity graph for the processes in the block assigned to this tree. Similarly, resources r_3 and r_7 are linked by an edge of weight ∞. Resource r_1 does not figure in the augmented affinity graph corresponding to the allocation for the tree with COMM_P_1 as the root since it is present in both the left and the right subtrees, namely, PROC_1 and PROC_2. Similarly, the augmented affinity graph for the tree with COMM_P_2 as the root is formed. Two partitions are performed on each of the two augmented affinity graphs formed as above. Out of the four blocks obtained, the block with resources r_5 and r_6 is assigned to PROC_1. Similarly, the other blocks are also assigned to their respective processing nodes. The processes in each block are processed on the processor to which the block containing them is assigned.

An example of a binary-tree-structured system was given to show the importance of the architecture of the system in designing the allocation algorithm. The binary tree structure enabled the use of divide and conquer strategy in allocation. Similar techniques could be applied to a certain class of hypercube systems also[7]. For a general distributed system of arbitrary interconnection structure the following method could be followed. The affinity graph is formed as described in Section 3. To augment the affinity graph all resources except those present in all nodes are added as vertices in the graph. All the resources present in a given processor are interlinked by edges of weight ∞. Each process p_i is linked by an edge of weight $\gamma U(i,j)$ to the resource r_j present in the graph. After the augmented affinity graph is formed, a multiple-way partitioning[8] is performed on the graph. The processes which are in the block containing a given set of resources are allocated to the processor with which the resources are associated.

5. Conclusions

In this paper, we have presented a graph-based model for process allocation on distributed computing systems. In this model, called the affinity graph model, the vertices represent the processes to be allocated. The vertices are linked by edges with weight equal to the affinity the vertices have for each other. The affinity has been defined in such a way that

allocation of processes with more affinity as close as possible contributes towards the goal of load balancing as well as minimising interprocessor communication. In the affinity graph, both the load balancing and the minimisation of interprocessor communication criteria are reflected in a single representation. It is also possible to vary the weightage given to these two criteria. We have presented a way of augmenting the affinity graph so as to take into account the heterogeneity of the system arising out of the distribution of the resources in the system over the processing nodes. By the use of the augmented affinity graph, it is also possible to minimise the overhead due to usage of remote resources in addition to load balancing and minimising interprocessor communication. An algorithm for allocation of processes for a two-processor system uses the augmented affinity graph. We have also briefly outlined how the allocation could be done for systems with a larger number of processors by using a binary-tree-structured system as an example. Simulation studies have shown that the proposed method has good load-balancing characteristics and is also responsive to changes in weightage given to different criteria[9].

Acknowledgements

The authors sincerely thank the referees for their very careful review, encouraging comments, and helpful suggestions. Their suggestions played a crucial role in moulding the paper to its present form.

References

1. CHU, W. W. *et al* — Task allocation in distributed data processing, *IEEE Computer*, 1980, **13**, 57–69.

2. MA, P. R. *et al* — A task allocation model for distributed computing systems, *IEEE Trans.*, 1982, **C-31**, 41–47.

3. SHEN, C. AND TSAI, W. — A graph matching approach to optimal task assignment in distributed computing systems using minimax criterion, *IEEE Trans.*, 1985, **C-34**, 197–203.

4. CHOU, T. C. K. AND ABRAHAM, J. A. — Load balancing in distributed systems, *IEEE Trans.*, 1982, **SE-8**, 401–412.

5. STONE, H. S. — Multiprocessor scheduling with the aid of network flow algorithm, *IEEE Trans*, 1977, **SE-3**, 85–93.

6. BOKHARI, S. H. — Dual processor scheduling with dynamic reassignment, *IEEE Trans.*, 1979, **SE-5**, 341–349.

7. SATYANARAYANAN, R. AND MUTHUKRISHNAN, C. R. — A task allocation scheme for hypercube distributed computing systems using the affinity graph model, *Proc. IEEE Region* 10 *Conf.*, Bombay, 1989.

8. KERNIGHAN, B. W. AND LIN, S. — An efficient heuristic procedure for partitioning graphs, *Bell System Tech. J.*, 1970, 291–307.

9. SATYANARAYANAN, R. AND MUTHUKRISHNAN, C. R. — A static scheduling scheme for tree-structured parallel computing systems, communicated to *Microprocessors and Microsystems*.