An Efficient Tasks Scheduling Model in Distributed Processing Systems Using ANN

Pradeep Kumar Yadav  
Central Building Research Institute, Roorkee (UK) (India),  
E-mail: Prd_yadav@rediffmail.com.

M.P. Singh  
Professor, Gurukula Kangri Vishwavidyalaya Hardwar (UK) (India).

Avanish Kumar  
Bundelkhand University, Jhansi-284128, (UP) (India)  
E-mail: dravanishkumar@indiatimes.com.

Babita Agarwal  
Research Scholar, Gurukula Kangri Vishwavidyalaya,  
Hardwar (UK) (India).

ABSTRACT: The single communication channel is shared by all the processors for Inter-Processor Communication (IPC) in a distributed system. A program whose execution is distributed among several processors in a distributed system has the total processing cost equal to the sum of Execution Costs (EC) and Inter-Processor Communication, which are functions of the amount of data transmitted. An optimal assignment is a distribution of tasks that has lowest total execution cost. The neural network architecture is used to form tasks clusters between maximally linked tasks and to restrict the clusters size, the average load to be assigned to each processor. The Artificial Neural Network (ANN) based model has been discussed in this paper. The model provides near to optimal solution for assigning a set of “m” tasks of a program to set of “n” processors where “m >> n” in such a way that allocated load on all processors is balanced according to the average load. For testing the efficiency of the algorithm, several sets of input data have been considered for all program
categories, and it is found that our algorithm resulted optimal allocation in most of the cases.

**Keywords:** Distributed system, inter-processor communication, execution cost, artificial neural network, optimization.

1. INTRODUCTION

Distributed system is a computer system in which multiple processors connected together through the high band-width communication links. These links provide a medium for each processor to access data and programs on remote processors. Distribution of resource in distributed system is seen as a way to improve system throughput and availability. An important resource for distributed system is user’s program that consists of a set of tasks.

Response time is the important parameter for measuring performance of distributed system. A distributed system is designed for solving some specific real time applications. The system is required to efficient scheduling of the task in computer communication network and finish a certain task within a specific time limit [15]. The excessive Inter-Processor Communication is always most costly and least reliable factor in the loosely coupled distributed system. Therefore, an efficient strategy is required for optimal utilization of processor’s capacity in distributed system. For optimal utilization of processor’s capacity it is essential to minimize IPC that arises when the interacting tasks reside on different processors. Only minimizing the IPC alone may not provide a good result. A processor’s load indicates the sum of execution times of the tasks residing on that processor. Balancing the load on processors and minimizing the IPC can minimize the response time of the system.

Several approaches have been reported for solving the task allocation problem in distributed system. Program partitioning into the small tasks and their allocations on the distributed computing system for processing play the important role in utilizing the distributed system power. The task partitioning, allocation and granularity [1] activities influence the distributed software properties such as Inter Processors Communication (IPC) cost and potential for parallelism. The manner in which this partitioning is done determines the efficiency of a given application when it is executed on distributed computing systems.
Therefore an efficient task allocation strategy is required for the proper utilization of computational resources and minimization of IPC. Several approaches to solve the problem of task assignment in distributed systems have been developed but most of these methods deal with homogeneous systems and the complexity of these algorithms increases rapidly as the problem size increased. Some of the heuristic models for task allocation in distributed computer systems have been discussed by [2-4]. The task allocation model based on graph-theoretic approach has been developed by Hesham H. Ali and Hasham E. El-Rewini [5] for NP-hard problem. A model was discussed by Chu, T., and Abraham J [6] considering the load balancing through systematic scheduling of tasks in distributed computing environments. Chu, W., Holloway, L., Lan, M., and Efe, K., [7] discussed a task allocation model for distributed data processing. A task allocation model developed by Ma, P., Lee, E. and Tsuchiya, M [8] using branch & bound methods. An algorithm for solving the Unbalanced Assignment Problems has been developed by [9]. Yadav, P.K., Kumar, Avanish and Gupta, A. R [10] has provided the solution for task allocation in distributed systems. Cost Based Static Approach for Performance Enhancement of Distributed Networks has been discussed by [11]. Improved Reliability of Distributed Computing Systems using Mathematical Programming has been discussed by [12]. Tasks scheduling algorithm for Multiple Processors system with Dynamic Re-assignment has been recently developed by [13].

By balancing load on the processors, it is possible to get efficient utilization of available computational power. In this paper, authors present a task allocation algorithm with the help of ANN working in union to solve specific problems.

Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements (neurons) working in parallel to solve a specific problem.

An artificial neuron is a device with several inputs and one output. The neuron works in two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using
mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.

Feed-forward ANNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straightforward networks that associate inputs with outputs. In this paper we use feed-forward ANN to assign tasks to the processor and find the near optimal solution. Neural networks do not perform miracles. But if used sensibly they can produce some amazing results.

The allocation is done before the actual run of the application program. It is assumed that the execution cost of the subtasks and the IPC cost, which arise due to the interacting subtasks residing on different processors are known. It allocates the modules or subtasks of an application problem as evenly as possible among the processors and tries to minimize the IPC cost.

2. TASK ALLOCATION PROBLEM

We conceive of a distributed system as composed as a set of “n” processing nodes \( P = \{p_1, p_2, \ldots, p_n\} \) and an interconnection structure providing full connectivity between the nodes and an application program consists a set of “m” tasks \( T = \{t_1, t_2, \ldots, t_m\} \) to be executed on these processing nodes. An allocation of tasks to processors is defined by a function \( Aalloc \), from the set of tasks to the set of processors.
An Efficient Tasks Scheduling Model in Distributed Processing Systems...

\[ \text{Alloc} : T \to P, \text{ where } \text{Alloc} (i) = j \] 
If the task \( t_i \) is assigned to processor \( P_j, 1 \leq i \leq m, 1 \leq j \leq n \).

### 3. DEFINITIONS

#### 3.1. Execution Cost

The execution cost \( e_{ij} \) is the amount of the work to be performed by the executing task \( t_i \) on the processor \( p_j \). The overall execution cost (EC) of a given allocation (Alloc) and execution time for each processor (PEC) are calculated by using equation (1) and (2) respectively.

\[
\text{EC (Alloc)} = \sum_{1 \leq i \leq m} e_{i, \text{Alloc}(i)} \quad \text{...(1)}
\]

\[
\text{PEC (Alloc)_j} = \sum_{1 \leq i \leq m} e_{i, \text{Alloc}(i)} \quad \text{...(2)}
\]

Where \( TS_j = \{i : \text{Alloc} (i) = j, \ j = 1, 2 ... n\} \)

#### 3.2. Inter Processor Communication Cost

The inter processor communication cost \( c_{ik} \) is incurred due the data limits exchanged between the executing task \( t_i \) and residing task \( t_k \) if they are on different processors. The overall inter-processor communication (IPC) cost of a given allocation Alloc is then calculated by equation (3) and for each Processor Inter-Processor Communication (PIPC) cost is then given by equation (4).

\[
\text{IPC (Alloc)} = \sum_{1 \leq i \leq m \atop 1 \leq j \leq m} c_{\text{Alloc}(i), \text{Alloc}(j)} \quad \text{...(3)}
\]

\[
\text{PIPC (Alloc)_j} = \sum_{1 \leq i \leq m \atop 1 \leq k \leq m \atop \text{Alloc}(i) \neq \text{Alloc}(k)} c_{\text{Alloc}(i), \text{Alloc}(k)} \quad \text{...(4)}
\]

(Where \( j = 1, 2, ... n \))

#### 3.3. Response Time (RT) of the System

Response time of the system is a function of amount computation to be performed by each processor and the computation time. This function is defined by considering the processor with the heaviest
aggregate computation and communication load. Response time of the system for a given assignment is defined by

$$RT(\text{Alloc}) = \max_{1 \leq j \leq n} \{\text{EC}(\text{Alloc})_j + \text{PIPC}(\text{Alloc})_j\}$$  ... (5)

3.4. Maximally Linked Task

A task $t_k$ in a program graph is called a maximally linked task if the sum of IPC costs of $t_k$ with other tasks is larger than the sum of IPC costs of any neighboring task i.e.

$$\sum_i c_{ik} > \sum_j c_{kj} \text{ for all } h \neq k \text{ such that } c_{kh} \neq 0$$

Where $c_{ij}$ is the IPC cost between the tasks $t_i$ and $t_j$, when $t_i$ and $t_j$ are assigned to different processors.

3.5. Average Load

Average load on a processor is the sum of EC of the tasks assigned to it then the average load that must be assigned to each processor $p_j$ is calculated as

$$L_{avg}(P_j) = \frac{W_j}{m}, j = 1, 2, \ldots, n.$$  

where

$$W_j = \sum_{1 \leq i \leq m} c_{ij}, j = 1, 2, \ldots, n.$$  

$$T_{load} = \sum_{1}^{n} L_{avg}(P_j)$$

4. ASSUMPTIONS

To keep the algorithm reasonable in size, several assumptions have been made while designing the algorithm. A program is assumed to be collection of “$m$” tasks to be executed on a set of “$n$” processors, which have different processing capabilities. A task may be portion of an executable code or a data file and size of all the tasks are equal. The number of tasks to be allocated is more than the number of processors ($m >> n$), as normally is the case in the real life. It is assumed that the execution time of a task on each processor is known, if a task is not executable on any of the processor due to absence of some resources. The execution time of that task on that processor is
taken to be (∞) infinite. It is also assumed that once a task has completed its execution on a processor, the processor stores the output data of the task in its local memory. If the data is needed by some another task being computed on the same processor, it reads the data from the local memory. Using this fact, the algorithm tries to assign maximally linked tasks to the same processor. Whenever groups of tasks or cluster are assigned to the same processor, the data transfer between them is zero. Completion of a program from computational point of view means that all related tasks have got executed.

5. TASK ALLOCATION MODEL

In many heuristic approaches, a search is made for a pair of module with maximum communication cost between them. Such a module pair is assigned to a processor to minimize the IPC cost. But, it increases when a module is interacting with many other modules. In this paper, authors do not search for such pairs of modules, but search maximally linked modules. The flowchart of whole program is shown in Figure 1.

![Flow Chart of the Program](image-url)
5.1. Maximum Linked Task

It is three layer feed forward network using neuron having the unipolar activation function given in figure 2. Each of the network layers is described by the formula \( O = \Gamma(WX) \). At first layer, the input vector contains the elements of the upper triangular matrix of the IPC matrix \( T \) and output contains the \( m \) elements. The value of each weight for the first layer is 1. At the second layer the output of the first layer work as an input vector and the output vector has \( m(m - 1) \) elements. In this layer, each element is compared with others \((m - 1)\) elements to search the maximum linked module. When \( i^{th} \) element is compared with \( j^{th} \) element then the weight \( W_{ij} = \text{Sgn}(t_{ij}) - 0.5 \) and for others it is equal to one. At third layer each weight has value one. The output will be one for maximum linked module and zero for others. If maximum linked module is not equal to the number of processors then we select remaining modules according to maximum inter processing communication cost. To obtain them, the max comparator has been used. Thus the module clusters are formed around the maximally linked module and these \( n \) modules are assigned on each processor.

![Figure 2: Network for Maximally Linked Task.](image_url)
**Processing loss function:** This function is trying to assign $i^{th}$ task $t_i$ to $j^{th}$ processor $p_j$ in memory the average load of each processor is stored. Now processing loss function find the difference of the average load and total execution cost of tasks which are assigned on processor $p_j$. This difference is called processing loss.

**IPC Loss Function:** When a $i^{th}$ task is assigned at $j^{th}$ processor, then IPC cost of the $i^{th}$ task with the task which are already assigned at $j^{th}$ processor will becomes zero and it will be added with another task which are not assigned at $j^{th}$ processor. Memory contains the previous IPC cost of assigned tasks at each processor. The difference of these IPC cost is called IPC loss. Neural network of IPC loss and processing loss of both is shown in figure 3. If the sum of the processing loss and IPC loss is positive then the $i^{th}$ task can be assigned to $j^{th}$ processor. This process is continued until the processing loss approaches zero/less than zero.

![Figure 3: (A)-IPC Loss Task, (B)-Processing Loss Task.](image-url)
6. EXAMPLE

Consider a distributed program consisting of a set $T = \{t_1, t_2, t_3, ..., t_9\}$ of 9 tasks to be allocated to a set $P = \{p_1, p_2, p_3\}$ of 3 processors. The objective of task assignment is to minimize the completion cost of a distributed program by properly mapping the tasks to the processors. Authors of [14] have devised a problem and developed a solution for it. We have tested our solution on that problem. To illustrate the algorithm, a program graph has been constructed and is shown in figure 4 with the execution time matrix $E$ being shown in Table 1 for three processors. Average load on each processor is shown in Table 2.

![Image](Figure 4: Example Program Graph.)

Table 1

<table>
<thead>
<tr>
<th>Task number</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>174</td>
<td>156</td>
<td>110</td>
</tr>
<tr>
<td>$t_2$</td>
<td>95</td>
<td>15</td>
<td>134</td>
</tr>
<tr>
<td>$t_3$</td>
<td>196</td>
<td>79</td>
<td>156</td>
</tr>
<tr>
<td>$t_4$</td>
<td>148</td>
<td>215</td>
<td>143</td>
</tr>
<tr>
<td>$t_5$</td>
<td>44</td>
<td>234</td>
<td>122</td>
</tr>
<tr>
<td>$t_6$</td>
<td>241</td>
<td>225</td>
<td>27</td>
</tr>
<tr>
<td>$t_7$</td>
<td>12</td>
<td>28</td>
<td>192</td>
</tr>
<tr>
<td>$t_8$</td>
<td>215</td>
<td>13</td>
<td>122</td>
</tr>
<tr>
<td>$t_9$</td>
<td>211</td>
<td>11</td>
<td>208</td>
</tr>
</tbody>
</table>
An Efficient Tasks Scheduling Model in Distributed Processing Systems...

Table 2
Average Load on Each Processor.

<table>
<thead>
<tr>
<th>Average Load</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>Total Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>445.333</td>
<td>325.333</td>
<td>404.667</td>
<td>1175.333</td>
<td></td>
</tr>
</tbody>
</table>

In starting first three maximally linked tasks are selected. Which are $t_1$, $t_8$, $t_9$ assigned to their best processor according to minimum execution cost on processor? In this case task $t_1$ is assigned to $p_3$, task $t_8$ to $p_2$ and task $t_9$ to $p_1$ which is the remaining unallocated processor. After this the algorithm proceeds as described in section 4.2 and 4.3. It should be noted that after each assignment of a task to the processor the total IPC time associated with that processor decreases. Results are shown in section 6.

7. RESULTS & CONCLUSION

The static task allocation is that when a task is assigned to processor, it remains there while the characteristic of the computation change and a new assignment is to be computed. The suggested piece of research is in finding the assignment pattern that holds for the life time of a task, and result in the optimum value of the measure of effectiveness. An artificial neural network based solution of static allocation of tasks to processors is used. In distributed computing environment the problem of distributing tasks to process is discussed and the results obtain are given in Table 3 shows the result obtained by the model.

Table 3
Result

<table>
<thead>
<tr>
<th>Task/Order</th>
<th>Cumulative Execution Cost</th>
<th>Aggregate IPC</th>
<th>Processing Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor-$p_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_9/3$</td>
<td>211</td>
<td>27</td>
<td>234.333</td>
</tr>
<tr>
<td>$t_8/6$</td>
<td>452</td>
<td>18</td>
<td>-6.667</td>
</tr>
<tr>
<td>$t_7/9$</td>
<td>464</td>
<td>15</td>
<td>-18.667</td>
</tr>
<tr>
<td><strong>Response time of Processor $p_1$ is = 479</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Processor-$p_2$ |                          |              |                 |
| $t_8/1$        | 13                        | 31           | 312.333         |
| $t_7/4$        | 247                       | 25           | 78.333          |
| $t_4/7$        | 462                       | 16           | -136.667        |

As it is not possible to achieve, a distributed system is considered to be balanced if the load on each processor is equal to the average load, within a reasonable tolerance that is of 20-30% of average load is generally chosen. Figure 5 show the maximum busy time of the system is 479 which relates to processor $p_2$ after getting the assignment from the algorithm is concluded that the total assigned load on the system is 1372 with the tolerance of 20%. This is very near to the calculated load given in Table 2. The result obtained by our algorithm is similar to [14] but in present algorithm the solution proposed is highly parallel and can be implemented on chip level.

As we have assumed the size of all the tasks are equal and from the result it is concluded that all the processors are executing three tasks form the figure 6, the through put of the $p_3$ processor is maximum because in the execution cost of processor $p_3$ is minimum.
The run time complexity of the algorithm suggested by Elsade [14] is $O(n^2 + m^2 + m^2 n)$ and the run complexity of the algorithm presented in this paper is $O(m^2 n + mn + n^2)$. Figure 6 shows the comparison between both the method and it concluded that present method is much better than Elsade [14].
REFERENCES


