A Tasks Allocation Algorithm for Optimum Utilization of Processor’s in Heterogeneous Distributing Computing Systems

*P.K. Yadav, **Preet Pal Singh, ***P. Pradhan

*Central Building Research Institute, Roorkee, U.K., India, E-mail:pktyadav@cbri.res.in
**Ph.D. Scholar, Dept. of Maths & Stats., Gurukula Kangri Vishwavidyalaya, Hardwar, U.K., India
***Dept. of Maths & Stats., Gurukula Kangri Vishwavidyalaya, Hardwar, U.K., India

Abstract -- In Distributed Processing System (DPS), partitioning of the application software into small tasks and the proper mapping of these tasks among processors are one of the important parameter which determine the efficient utilization of available Processor’s Capacity. The model discussed here performs static task mapping/ allocation of a set of ‘m’ tasks of a program to a set of ‘n’ processors (where m > n) with the constraints of minimizing Inter Task communication (ITC) cost and maximize the overall throughput of the system in such a way that allocated load on all the processors is balanced. While designing the model Per Bit Processor Service Rate PSR(,)and Inter Task Communication Cost ITCC(,) and Task Size TS (,) have taken into consideration.

1. INTRODUCTION

Distributed processing applications range from large data base installations where processing load is distributed for organizational efficiency to high-speed signal processing systems, where extremely fast processing must be performed in a real-time environment. The distributed real time environment in which, the services provided for the network reside at multiple sites. Therefore, Systematic scheduling in distributed processing environment is an important issue to improve performance of the system. Partitioning of the application software into small module of tasks and the proper allocation of these modules among processors are one of the important parameter which determines the efficient utilization of available resources [1, 2]. If this step is not done systematically it may cause in degradation in throughput. The allocation policy could be either static or dynamic. The problem of finding an optimal dynamic assignment of a modular program for a two-processor system has been analyzed by [3].

One measure usefulness of a general-purpose distributed computing system is the system’s ability to provide a level of performance commensurate to the degree of multiplicity of resources present in the system. Taxonomy of approaches to the resource management problem is reported [4]. The taxonomy, presented and discussed in terms of distributed scheduling, is also applicable to most types of resource management. A model for allocating information files has been reported by [5]. This model considers storage cost, transmission cost, file lengths, and request rates, as well as updating rates of files, the maximum allowable expected access times to files at each computer, and the storage capacity of each computer. The model is formulated into a nonlinear integer zero-one programming problem, which may be reduced to a linear zero-one programming problem. Known solutions of a large number of important (and difficult) computational problems called NP-complete problems depend on enumeration techniques which examine all feasible alternatives. The design of enumeration schemes in a distributed environment have been reported by [6].

A task allocation model that allocates application tasks among processors in distributed computing systems satisfying: 1) minimum inter-processor communication cost, 2) balanced utilization of each processor, and 3) all engineering application requirements has been reported by Perng-Yi Richard Ma et.al [7]. This problem of task allocation in heterogeneous distributed systems with the goal of maximizing the system reliability has been addressed [8]. The model is based on the well-known simulated annealing (SA) technique. Yadav et al have reported an algorithm for reliability evaluation of distributed system based on failure data analysis [9]. An efficient algorithm for optimal tasks allocation through optimizing reliability index in heterogeneous distributed processing system has been discussed by [10]. J. B. Sinclair [11] considered the problem of finding an optimal assignment of the modules of a program to processors in a distributed system. A module incurs an execution cost that may be different for each processor assignment, and the modules which are not assigned to the same processor but communicate with one another incur a communication cost. An optimal solution to the problem of allocating communicating periodic tasks
to heterogeneous processing nodes (PNs) in a distributed real-time system has been reported [12]. Matrix reduction technique has been used by Sagar et al, according to the criteria given therein a task is selected randomly to start, with and then assigned to a processor [13]. A fast algorithm for allocation task in distributed processing system has been reported by Kumar et al [14, 16]. In this method the author tried to cluster heavily communicated tasks and allocated them to same processor. An efficient Algorithm for allocating tasks to processors in a distributed system has also been reported by Kumar et al. [15].

Distributed computing systems offer the potential for improved performance and resource sharing. To make the best use of the computational power available in multi processing system, it is essential to assign the tasks dynamically to that processor whose characteristics are most appropriate for the execution of the tasks in distributed processing system [17, 18]. The new methodology augments the maximally linked module concept by using stochastic techniques and by adding constructs which take into account the limited and uneven distribution of hardware resources (often associated with heterogeneous systems) has been discussed by Elsade et al. Authors used pure simulated annealing and the randomized algorithm for randomly-generated systems and synthetic structures which were derived from real-world problems [19]. Communication technology has also opened many avenues for processing, sharing and transferring the data. Yadav, et al [20] reported task scheduling in computer communication network.

An Artificial Neural Network (AANN) based task scheduling model has been discussed by Yadav et al [21]. For developing the model authors used feedback neural network architecture. Singh et al reported an ANN based model for Load Distribution in fully connected client server network [22]. Analysis of load distribution in distributed processing systems through systematic allocation task and an exhaustive approach of performance analysis to the distributed systems based on cost assignments have been reported by Kumar et al [23, 24].

The main objective of this paper is to minimize the total system cost as well as the total program execution cost. The model discussed here performs the static task mapping/ allocation of a set of ‘m’ tasks of a program to a set of “n” processors (where, m > n) with the constraints of minimizing inter task communication (ITC) cost and maximize the overall throughput of the system in such a way that allocated load on all the processors should be balanced.

2. MATHEMATICAL MODELLING

The specific task allocation problem being addressed as follows:

Considered an application program consisting of a set of “m” tasks \( T = \{ t_1, t_2, \ldots, t_m \} \) and a heterogeneous distributed processing system consisting a set of “n” processors \( P = \{ p_1, p_2, \ldots, p_n \} \), where it is assumed that \( m > n \), and allocated each of the \( m \) tasks to one of the \( n \) processors in such a manner that total system time is minimized and processing load on the processors is balanced. While developing the model following inputs have been taken into consideration.

- Per Bit Processor Service Rate (PSR),
- Task Size (TS),
- Inter Task Communication Cost (ITCC)
- Execution Cost (EC)

2.1 Processor’s Execution Rate (PER): Per Bit Processor’s service rate which is the execution rate \( er_j \, (1 \leq j \leq n) \) of each processor is the speed (bytes/second) of the processor at which they execute the tasks given in the form \( PSR(j) \) (where \( j=1,2,3,\ldots,n \)).

\[
PSR(j)=
\begin{bmatrix}
er_1 \\
er_2 \\
er_3 \\
\cdot \\
\cdot \\
er_n
\end{bmatrix}
\]

2.2 Task Size (TS): A task is a sequential program, which performs some predefined action and possibly communicates with other tasks in a system. The task size \( ts_i \, (1 \leq i \leq m) \) of each task depends on the length of tasks and generally counted in bytes. The task size is taken in the form of a linear array \( TS(i) \) (where \( i=1,2,3,\ldots,m \))

\[
TS(i)=
\begin{bmatrix}
ts_1 \\
ts_2 \\
ts_3 \\
\cdot \\
\cdot \\
ts_m
\end{bmatrix}
\]

2.3 Inter Task Communication Cost (ITCC): The inter task communication cost \( cc_{ik} \) of the interacting tasks \( t_i \) and \( t_k \) is incurred due to the exchange of data units between them, during the process of execution. The ITCC is taken in the form of a symmetric matrix
named as Inter Task Communication Cost Matrix (ITCCM), which is of order m.

2.4 Execution Cost (EC): The execution cost $e_{ij}$ Where $1 \leq i \leq m, 1 \leq j \leq n$ of each task $t_i$ depends on the processor $p_j$, to which it is assigned and the work to be performed by each of tasks of that processor $p_j$. To determine EC, initially we have taken the product of transpose of the PSR (j) and TS (i) and stored the result in Execution Cost Matrix (ECM(i,j)) of order m x n.

After multiplication, the computed final ECM (i, j) is as follow:

$$
ECM(i,j) = \begin{bmatrix}
    t_1 & t_2 & \cdots & t_m \\
    t_1 & e_{11} & e_{12} & \cdots & e_{1m} \\
    t_2 & e_{21} & e_{22} & \cdots & e_{2m} \\
    \vdots & \vdots & \ddots & \vdots & \vdots \\
    t_m & e_{m1} & e_{m2} & \cdots & e_{mm}
\end{bmatrix}
$$

An allocation of tasks to processors is defined by a function $A_{alloc}$ from the set T of tasks to the set P of processors such that:

$A_{alloc}: T \rightarrow P$, where $A_{alloc}(i) = j$ if task $t_i$ is assigned to processor $p_j$, $1 \leq i \leq m, 1 \leq j \leq n$.

For an assignment, the task set (TAS) of a processor can now be defined as the set of tasks allocated to the processor

$$
TAS_j = \{i: A_{alloc}(i) = j, \quad j = 1, 2, \ldots, n\}
$$

Initially, ‘n’ Minimally Linked Task (MLT) has been determined by using equation (1) and stored the result in an array MLT(i) (where $i = 1, 2, \ldots, m$) and also stored their corresponding task in an another array TLM(k) (where $k = 1, 2, \ldots, m$) also arranging the MLT () in ascending order.

Then re-arranged the ECM (i, j) accordingly and mapped the first ‘n’ tasks to each ‘n’ processors by applying the algorithm developed by Yadav et al. [24]. Stored the initial assignment in an linear array $T_{ass}(j)$ where $j = 1, 2, \ldots, n$ and the processor positions in a linear array $A_{alloc}(j)$. The value of $T_{TASK}(j)$ is also computed by adding the values of $A_{alloc}(j)$, if a task $t_i$ is assigned to processor $p_j$ otherwise continue. The remaining (m-n) tasks are then stored in a linear array $T_{non-ass}()$.

Tasks assigned to processors $p_j$ or stored in $T_{non-ass}()$ which are obviously

$$
T = T_{ass} \cup T_{non-ass}
$$

All the tasks stored in $T_{non-ass}()$ are then fused with those assigned tasks stored in $T_{ass}()$ on the bases of minimum addition of EC and ITCC. The Fused Execution Cost (FEC) of a task $t_a \in T_{non-ass}()$ with some other task $t_i \in T_{ass}()$ on processor $p_j$ is obtained by using equation (2).

$$
FEC(j)_{ai} = \left[ e_{ai} + \sum_{t \in T_{ass}} c_{ai} \right], \quad (1 \leq a \leq m, 1 \leq i \leq m, 1 \leq j \leq n; i \neq a)
$$

Let $cc_{ai}$ be the ITCC between $t_a \in T_{non-ass}()$ and $t_i \in T_{ass}()$. Then Fused Communication Cost (FCC) for $t_a$ with $t_i$ is calculated by using equations (3).

$$
FCC(j)_{ai} = \sum_{t \in T_{ass}} c_{ai}
$$

Here, $c_{ai} = 0$ if fused with $t_i$ or $a = 1$ and remaining $c_{ai}$ values are Additive Fused Cost (AFC) and determined by the equation (4).
This process will be continued until all the tasks stored in \( T_{\text{non-ass}}() \) are fused. After completing allocation, calculate \( PEC(j), \) \( \text{PITCC}(j) \) from the equations (5), (6) respectively.

\[
P_{\text{EC}}(A_{\text{alloc}}) = \sum_{1 \leq i \leq m} e_{i,k} A_{\text{alloc}}(i)
\]

(5)

\[
\text{ITCC}(A_{\text{alloc}}) = \sum_{1 \leq i \leq m} c_{i,k} A_{\text{alloc}}(i)
\]

(6)

Finally, we calculated the TOC by summing up the values of \( PEC(j) \) and \( \text{ITCC}(j) \) by using equation (7). This function is defined by considering the processor with the heaviest aggregate computation and communication loads. The computed results are then stored in a linear array \( \text{TOC}(j) \) where \( j = 1, 2, \ldots , n \).

\[
\text{TOC}(j) = \max \{ PEC(j) + \text{ITCC}(j) \}
\]

(7)

The overall throughput and the mean service rate (MSR) of each processor has been calculated by using equation 8 and 9 as and stored the results of throughput in the linear arrays \( \text{TRP}(j) \), where \( j = 1, 2, \ldots , n \) and MSR in MSR\((j)\) respectively.

\[
T_{\text{RP}}(j) = \frac{T_{\text{TASK}}(j)}{P_{\text{EC}}(A_{\text{alloc}})} = 1, 2, 3, \ldots , n
\]

(8)

\[
\text{MSR}(j) = \frac{1}{P_{\text{RC}}(A_{\text{alloc}})} = 1, 2, 3, \ldots , n
\]

(9)

**Algorithm**

**Step 1:** Input: \( m, n, \text{PSR}(\cdot), \text{TS}(\cdot), \text{ITCC}(\cdot) \)

**Step 2:** Calculate ECM by \( \text{TS}(\cdot) \times \text{PSR}(\cdot) \).

**Step 3:** Compute MLT(i) for all tasks.

**Step 4:** Sort the tasks in a linear array \( T_{\text{non-ass}}() \) by increasing order of their MLT(i) values.

**Step 5:** While there are tasks in \( T_{\text{non-ass}}() \) do

**Step 5.1:** Select the first \( m \) task from the \( T_{\text{non-ass}}() \).

**Step 5.2:** Assign them tasks to the \( n \) processor respectively

**Step 5.3:** Now assign \( (m+1) \)th tasks from \( T_{\text{non-ass}}() \) and assign to the processor \( p_i \) with

\[
\text{AFC}(j)_{ai} = \min \left\{ (\text{FEC}_{a1} + \text{FCC}_{a1}), (\text{FEC}_{a2} + \text{FCC}_{a2}), \ldots , (\text{FEC}_{an} + \text{FCC}_{an}) \right\}
\]

(4)

**Step 5.4:** now delete that (\( m+1 \))th task from \( T_{\text{non-ass}}() \) and add it in \( T_{\text{ass}}() \)

**Step 5.5:** repeat this until \( T_{\text{non-ass}} \) is empty

**Step 5.6:** end while.

**Step 6:** Compute \( P_{\text{EC}}(A_{\text{alloc}}), \text{PITCC}(A_{\text{alloc}}), \text{RT}(A_{\text{alloc}}), \text{MSR}(j), \text{TRP}(j) \).

**Step 7:** End.

3. RESULTS & DISCUSSIONS

To check the usefulness of the present method, an example of a DCS is considered. This example consists of a set of three processors connected by an arbitrary network and a set of nine executable tasks. We have taken processors service rate (PSR), Tasks size (TS) and inter tasks communication cost (ITCM) randomly as below:

**Input m=9, n=3**

\[
\begin{array}{cccc}
\text{PSR}(\cdot) &=& 0.585 & 0.756 & 0.679 \\
\text{TS}(\cdot) &=& 258 & 320 & 185 & 200 & 270 & 200 & 230 & 295 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{ITCC}(\cdot) &=& 0.585 & 0.756 & 0.679 \\
\end{array}
\]

**RESULTS & DISCUSSIONS**
Compute MLT () for all tasks by using equation 2

<table>
<thead>
<tr>
<th>Task</th>
<th>MLT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>150.93</td>
</tr>
<tr>
<td>t2</td>
<td>187.20</td>
</tr>
<tr>
<td>t3</td>
<td>106.23</td>
</tr>
<tr>
<td>t4</td>
<td>175.59</td>
</tr>
<tr>
<td>t5</td>
<td>122.85</td>
</tr>
<tr>
<td>t6</td>
<td>117.00</td>
</tr>
<tr>
<td>t7</td>
<td>134.55</td>
</tr>
<tr>
<td>t8</td>
<td>172.59</td>
</tr>
</tbody>
</table>

Arrange the MLT () in the increasing order.

<table>
<thead>
<tr>
<th>Task</th>
<th>MLT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>284</td>
</tr>
<tr>
<td>t3</td>
<td>288</td>
</tr>
<tr>
<td>t1</td>
<td>295</td>
</tr>
<tr>
<td>t4</td>
<td>297</td>
</tr>
<tr>
<td>t7</td>
<td>300</td>
</tr>
<tr>
<td>t6</td>
<td>301</td>
</tr>
<tr>
<td>t5</td>
<td>303</td>
</tr>
<tr>
<td>t8</td>
<td>340</td>
</tr>
</tbody>
</table>

So we have Tnon_ass= {t7, t5, t9, t6, t8, t4, t2, t3, t1} and Tass= {} initially because there is no allocation to any processor. Now first of all, take first three tasks (equal to the number of processors) from Tnon_ass i.e. t7, t5 and t9 and assign them to each processor respectively i.e. t7 to p1; t5 to p2 and t9 to p3.

Now delete them from Tnon_ass() and add them in Tass() so now Tnon_ass= {t6, t8, t4, t2, t3, t1}; and Tass= {t7, t5, t6, t8}; we continue this process until Tnon_ass=Ø after executing this process up to the end we get:

**Table-1**

<table>
<thead>
<tr>
<th>Processor</th>
<th>Allocated tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>t7, t5, t9</td>
</tr>
<tr>
<td>p2</td>
<td>t8, t4, t2, t3, t1</td>
</tr>
<tr>
<td>p3</td>
<td>t6, t8, t3</td>
</tr>
</tbody>
</table>

Because AFC(1)_{6,7} = min{ AFC(j)ai} so t6 is allocated to p1.

Now Tnon_ass= { t8, t4, t2, t3, t1}; and Tass= {t7, t5, t6, t8};

Similarly now we take t8 to allocate and find:

<table>
<thead>
<tr>
<th>Processor</th>
<th>Allocated tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>p3</td>
<td>t6, t8, t3, t1</td>
</tr>
</tbody>
</table>

Because AFC(3)_{8,9} = min{ AFC(j)ai} so t8 is allocated to p3.

Now Tnon_ass= { t4, t3, t1}; and Tass= {t7, t5, t6, t8, t9}; we continue this process until Tnon_ass=Ø after executing this process up to the end we get:

**Table-2**

<table>
<thead>
<tr>
<th>Processor</th>
<th>EC</th>
<th>ITCC</th>
<th>MSR</th>
<th>TRP</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>462.150</td>
<td>698</td>
<td>0.00216</td>
<td>0.00649</td>
<td>1150.150</td>
</tr>
<tr>
<td>p2</td>
<td>580.608</td>
<td>692</td>
<td>0.00172</td>
<td>0.00517</td>
<td>1272.608</td>
</tr>
<tr>
<td>p3</td>
<td>482.090</td>
<td>676</td>
<td>0.00207</td>
<td>0.00622</td>
<td>1158.090</td>
</tr>
</tbody>
</table>

Now calculated EC, ITCC, MSR and TRP for this allocation are:

**Fig-1a**: Mean service rate and Throughput of the processors
It should be noted that after each assignment of a task to a processor the aggregate ITCT associated with that processor usually decreases. Table 2 shows the resulting allocation where it indicates the order in which each task is assigned to a processor. Fig 1a show that the through put and services rate, of the processor from the Fig. It is concluded that both are the ideally linked. Fig 1b shows that total allocated to processors from the figure it is concluded that the load assigned to the processor almost balance and the maximum busy time of the system 1s 1272.608 which is associated with the processor p2.

4. CONCLUSION

The present paper deals with a simple, yet efficient mathematical and computational algorithm to identify the optimal allocation method to use the processors capacity and evaluation performance of distributed systems. Table 1 shows that three tasks are executing on processor p1, three tasks are executing on p2 and three tasks are executing on p3. From Table 2 we conclude that the maximum busy time of the system is 1272.608 time units, which is related to processor p2. Throughput of the processors p1, p2 and p3 are 0.00649, 0.00517 and 0.00622 times units respectively and the average through put of the system is 0.00596 time units. The performance of the algorithm is compared with the algorithm reported by Singh et al [25]. The example reported in this paper has also been solved by the algorithm reported in this paper and it is found that the present algorithm is giving the better results.

Example:

![Figure 2](image-url)

Figure 2 shows that the comparisons between the optimal busy time of the system the optimal busy time of the system obtained by the present is 39 on processor’s p3 and optimal busy time of the system is reported by the Singh et al [25] is 47 on processor’s p1.
Fig 2: Comparisons optimal busy time of the systems

Figure 3a and 3b shows the comparisons between mean service rate and throughput of the processors respectively. From the figure 3a it is concluded that the mean service rate of the processor’s are reciprocal and through put of the present model is better than the Singh et al [25].

Fig 3a Comparisons processor’s Mean Service Rate

Fig 3b Comparisons processor’s Throughput

5. REFERENCES


