Optimal Execution Cost of Distributed System: Through Clustering

Anju Khandelwal

#Dept. of Basic Engg. Sciences, S.R.M.S. Women’s College of Engineering and Technology, Bareilly (India)

Abstract- “Distributed Systems” is used to describe when ever there are several computers interconnected in some fashion so that program or procedure running on the system with multiple processors. However, it has different meanings to different systems because processors can be interconnected in many ways for various reasons. The task allocation in a Distributed Processing System finds extensive applications in the facilities, where large amount of data is to be processed in relatively short period of time, or where real-time computations are required. Also the purpose of task assignment in distributed computing systems it to reduce the job turnaround time and increase the throughput. The main objective of this paper is to minimize the total program execution period by allocating the tasks optimally.

Key Words- Distributed Computing Systems, Task Allocation, Cluster, Inter task Communication Cost, Inter processor Distance.

I. INTRODUCTION

Distributed computing systems [DCSs] offer the potential for improved performance and resource sharing. To make the best use of the computational power available, 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.

Distributed computing systems have become more and more attractive and important in recent year due to the advancement of VLSI and computer network technologies. Distributed computing systems not only provide the facility for utilizing remote computer resources or data not existing in local computer systems but also increase the throughput by providing facilities for parallel processing. Rapid advances in communication technology and proliferation of inexpensive PCs and workstations have created a wide avenue for Distributed computing system to move into mainstream computing. A distributed computing system (DCS) consists of a number of PCs or workstation interconnected through PPP, LAN or WAN. These systems provide a higher performance, better reliability and throughput over centralized mainframe systems.

The main incentives for choosing DCS are higher throughput, improved availability, and better access to a widely communicated web of information. The increased commercialization of communication system means that ensuring system reliability is of critical importance. Inherently, distributed system is more complex; therefore, it is very difficult to predict the performance of DCS. Mathematical modeling is the tool which can plays an important role to predict the performance of DCS. Therefore, there is an urgent need to develop a method for it. Allocation of tasks in a DCS may be done in verity of ways (i) Static Allocation: In static allocation when a task is assigned to processor, it remains there while the characteristic of the computation change and a new assignment must be computed. (ii) Dynamic Allocation: In order to make the best use of resources in a distributed system, it is essential to reassign modules or tasks dynamically during program execution, so as to the advantage of changes in the local reference patterns of the program.

II. PROBLEM STATEMENT

The main objective of this problem is to minimize the total program execution period by allocating the tasks in such a way that the allocated load on each processor should be balanced. The model utilized the mathematical programming technique for execution of the module considering that each module to be executed through all the processors. The Execution Cost and Inter Tasks Communication Cost are considered for developing the algorithm. The impact of Inter Processor Distance also considered and it is mentioned in the Inter Processor Distance Matrix [IPDM (,)] of the order n. The EC and ITCC are presented by arrays in the form Execution Cost Matrix [ECM (,)] of order m x n and Inter Tasks Communication Cost Matrix [ITCCM (,)] of order m.

Since the number of task are more than the number of processors, so that it is required to form the clusters of tasks. To forming the cluster of tasks arrange the upper diagonal value of ITCCM (,) in descending order and store the result in a linear array maxcc() also store their respective positions in two dimensional array pos(.). The maximum number of tasks in a cluster shell be governed by the formula nt = [m/n], where nt is the
number of tasks in a cluster. Select the first value of maxcc() and its corresponding positions from pos(,) say, (t_i, tk) and store the cl(i,j) where i= 1,2,..no. of cluster and j=1,2,..nt). If j<nt then pickup the next value from maxcc() and its corresponding positions from pos(,) say, (t_l,tk) select the task form this combination which not already exit in cl(i,j) say, t_l and store the same in cl(i,j). This process continues until the entire cluster to be formed. Some of the tasks, which are not involved in any cluster, are known as isolated tasks. Assign these clusters and isolated tasks by using the KSY Algorithm [4]. Calculate the execution cost and inter tasks communication cost with processor distance of each processor and store the result in a linear array pec(j) and pitccpd(j) respectively where j= 1,2,…n.

$$pec(i) = \sum_{j=1}^{m} c_{ij} x_{ij}, i = 1,2,\ldots,n$$

Where $$x_{ij} = \begin{cases} 0, & \text{if } t_i \text{ and } t_j \text{ are on the same processor.} \\ 1, & \text{otherwise} \end{cases}$$

and

$$pitccpd(j) = \sum_{i=1}^{m} c_{ij} * d_{ij} x_{ij}, j = 1,2,\ldots,n$$

Where $$x_{ij} = \begin{cases} 1, & \text{if } t_i \text{ is on the } t_j \text{th processor.} \\ 0, & \text{otherwise} \end{cases}$$

Finally, sum up the value of pec(j) and pitccpd(j), (j=1,…..n) and store the result in tcbp(j) and pickup the maximum value of tcbp(j) i.e. toscbp called as total optimal system cost for busy period.

A. Computational Algorithm

The method discussed in this paper is to determine the tasks allocations in distributed processing environment based on the following components.

1. Determine the initial allocation
2. Determine the cluster of m-n tasks
3. Determine the final allocation
4. Computation the total system cost and throughput of the processors.

B. Algorithm

To given an algorithmic representation to the technique mentioned in the problem statement, let us consider a system in which a set T= {t1, t2, t3… tm} of “m” tasks is to be executed on a set P= {p1, p2, p3… pn} of “n” available processors.

Step-1:

Input: m, n;//m is the number of modules of a task, n is the number of processor//
Input: etm(,);//matrix to hold the execution time of each task to each processor//
Input: itctm(,);//matrix to hold the Communication time amongst the tasks//

Step-2:

maxct()←0;//linear array to store the inter task communication time//
itask()←0;//linear array to store the tasks for initial assignment//
Itask()←0;//linear array to store the remaining tasks for initial assignment//
bmin←0;//Variable for selecting the best minimum//
Tass()←0;//linear array to hold the tasks in order to assignment made//
Tnon-ass()←0;//linear array to store the non assigned tasks//
nomade←0;//variable for counting the number of the assignment made//
alloc()←0;//linear array to hold processor’s position in order of assignment//
msr()←0;//linear array to hold processor mean service rate//
trp()←0;//linear array to hold the throughput of the processors//
mst()←0;// linear array to hold processor mean service time //
pos(,)=>to dimensional array to hold to the corresponding position of maxct()//
cpos(,)=0;// to dimensional array to hold to the position of common element //
netm(,)=0;//operational matrix for execution time//
nictm(,)=0;// operational matrix for inter task communication time //
task() ← 0; //linear array to store the total number of task to the processors/

**Step-3:**
for i ← 1 to n do
    for j ← 1 to n do
        store the etm(,) in netm(,) as
        netm(,) ← etm(,)
    repeat
for i ← 1 to n do
    for j ← 1 to n do
        store the itctm(,) in nitctm(,) as
        nitctm(,) ← itctm(,)
    repeat

**Step-4:** set k ← m (m-1)/2

**Step-5:**
for i ← 1 to m do
    for j ← 1 to k do
        arrange the upper diagonal values of nitctm(,) in non–ascending order and store the result in maxct(,)
        until j = k
    repeat

**Step-5.1:** for i ← 1 to k do
    for j ← 1 to 2 do
        store the combinations of tasks of maxct(,) in pos(,)
    repeat
    repeat

**Step-5.1.1:**
set nt ← 0
    set count ← m/n
**Step-5.1.2:**
set nt ← count
    for i ← 1 to m do
        for j ← 1 to k do
            pick-up the maximum value of maxct(,) and check the corresponding combination in pos(,), say(tj,tk)
            if nt=count
                store the cluster in a linear array cli()
            else
                if pos(i,j)<pos(i,k)
                    nmax ← pos(i,j), until j=k
                    repeat for i
                    nmax ← pos(i,k), until j=k
                endif
                check the corresponding combination of nmax in pos(,),say(tj,tk), store the cluster in a linear array cli()
            repeat
        repeat
    set k ← (m-nt(m-nt)-1)/2
**Step-5.1.4:** modify the pos(,) by deleting the combination and reduce maxct() by eliminating the corresponding values.
    modify the nectm(,) by adding the ith and kth rows together, also modify the nitctm(,) by adding the ith and kth rows and then column, remove  kth rows from nectm(,) and kth column from nitctm(,)
goto Step-5.1.2.

**Step-6:**
for k ← 1 to n do
    for j ← 1 to n do
        find out minimum of kth rows, say mrkj, of netm(i,j) lying in jth column and subtract mrkj from all the values of kth rows
    repeat
    repeat
Step-6.1: for j ← 1 to n do for k ← 1 to n do
find out minimum of jth column, say mckj, of netm(i,j) lying in kth row and subtract mckj from all the values of jth column
repeat
repeat

Step-7: for k ← 1 to n do for j ← 1 to n do
row in netm(,) has only one zero at position (1,2)
assign task t1 to p2;
alloc(k) ← j; alloc(k(2))
nomade ← nomade + 1; nomade=1
tass ← tass \ {tk}; \{t1\}
repeat

Step-7.1: for j ← 1 to n do for k ← 1 to n do
search for a column in netm(,), which has only one zero, say at position (k,j); assign task(s) corresponding to kth, say, tk, row to pth processor, say pj,
alloc(k) ← j
nomade ← nomade + 1
tass ← tass \ {tk}
repeat

Step-8: if nomade ≠ n then pick-up an arbitrary zero,
go to Step–7
else
go to Step-7.1
endif

Step-8.1: check column(s) position of zero(s) in unassigned row(s) check the row (s) any previous assignment in the corresponding column(s) store the positions of the common elements in cpos(,), say i,j), find the minimum element of all the elements of the remaining rows, say minij, subtract minij, from these elements add minij, at the common positions and then go to Step-6.

Step-9: for k ← 1 to m do for j ← 1 to n do
compute the etij by summing-up th value of etij for each processors and store the result in a linear array pet(j).
compute the mean service rate of the pth processor, say pj, stored in alloc(k) for the assigned tasks corresponding to kth, say tk, in tass()
msr(j) ← 1/pet(j)
repeat
repeat

Step-9.1: for j ← 1 to n do
Compute the maen services time of the processors and store the result in a linear array mst(j)
msr(j) ← 1/msr(j)
repeat

Step-9.2: for i ← 1 to n do
pcount ← 0
for j ← 1 to m do
compute the processor’s throughput and store the result in a linear array trp(i) as,
if i = alloc(j)
then
pcount ← pcount + 1
else
next j
endif
trp(i) ← pcount
repeat
ttask(i) ← pcount
repeat
trp(i) ← ttask/mst(i)
repeat
Step-10:  
tcc←0
for i ← 1 to m do
for j ← 1 to n do
compute the itct for each processor as,
pcc()←tcc+itctm(I,j)
repeat
repeat
Step-11:  
for j ← 1 to n do
compute the total busy time for each processor
tpbt()←pet()+pcc()
repeat
and select the maximum value from tbpt()
tost←max{tpbt()//tost is the total system time
Step-12:Stop.

C. Implementation

Consider a distributed computing system which is consisting of a set P = \{p_1, p_2, p_3\} of “n = 3” processors connected by an arbitrary network. The processors only have local memory and do not share any global memory. The processor connections graph is depicted in figure-1 and tasks execution graph also pictorially depicted in figure-2. A set T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\} of “m = 8” executable tasks which may be portion of an executable code or a data file. The Inter tasks communication graph is depicted in figure-3.
Input: \( m = 8, n = 3 \)

\[
\begin{array}{cccccccc}
 & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \\
 t_1 & 0 & 3 & 4 & 2 & 6 & 8 & 1 & 0 \\
t_2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\
t_3 & 4 & 0 & 0 & 4 & 3 & 2 & 0 & 0 \\
\text{itccm}(,) = & t_4 & 2 & 0 & 4 & 0 & 5 & 3 & 2 & 5 \\
t_5 & 6 & 0 & 3 & 5 & 0 & 0 & 0 & 0 \\
t_6 & 8 & 0 & 2 & 3 & 0 & 0 & 6 & 8 \\
t_7 & 1 & 0 & 0 & 2 & 0 & 6 & 0 & 5 \\
t_8 & 0 & 5 & 0 & 5 & 0 & 8 & 5 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
P_1 & P_2 & P_3 \\
t_1 & 6 & 3 & 5 \\
t_2 & 4 & 2 & 3 \\
t_3 & 3 & 1 & 2 \\
\text{etm}(,) = & t_4 & 5 & 2 & \infty \\
t_5 & 3 & 4 & 2 \\
t_6 & 6 & \infty & 6 \\
t_7 & 5 & 6 & 7 \\
t_8 & \infty & 2 & 5 \\
\end{array}
\]
tasks $t_2$ and $t_7$ are not involved in any cluster known as isolated tasks. After applying the KSY algorithm\cite{4} the final allocations are:

\[
\begin{align*}
\text{Cluster} &= cl (i, j) = \left[ 1 : t_1, t_6, t_8 \right] \\
&\quad \quad \quad 2 : t_3, t_4, t_5
\end{align*}
\]

### III. CONCLUSIONS

The problem discussed in this paper provides an optimal solution in a Distributed System to maximize the overall throughput and the balanced load of all allocated tasks on each processor. This approach forms the clusters before making the assignments in the example mentioned in the implementation section. Two clusters have been formed of containing three tasks. There is two tasks are not involved in any of the cluster treated as isolated tasks. The Inter Task Communication Cost with Inter-processor distance of each processor pitccpd() = (48, 74, 106)

Total cost of busy period of each processor tcbpt() = (57, 81, 122)

Maximum value of tcbpt() = 122

Total optimal system cost for busy period = 122

### Execution cost of each processor

pec() = (9, 7, 16)

### Inter tasks communication cost with inter-processor distance of each processor

pitccpd() = (48, 74, 106)

### Total cost of busy period of each processor

\[ \text{tcbpt()} = (57, 81, 122) \]

### Maximum value of tcbpt() = 122

### Total optimal system cost for busy period = 122

### III. CONCLUSIONS

The problem discussed in this paper provides an optimal solution in a Distributed System to maximize the overall throughput and the balanced load of all allocated tasks on each processor. This approach forms the clusters before making the assignments in the example mentioned in the implementation section. Two clusters have been formed of containing three tasks. There is two tasks are not involved in any of the cluster treated as isolated tasks. The Inter Task Communication Cost with Inter-processor Distance and Execution Cost on different processors has been obtained. The total optimal system cost for busy period in the example mentioned is found 122 units. The final results are shown in the following table:

<table>
<thead>
<tr>
<th>Processor $\rightarrow$ Tasks</th>
<th>Pec()</th>
<th>Pitccpd()</th>
<th>tcbpt()</th>
<th>Tosecbp</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1 \rightarrow t_2, t_7$</td>
<td>9</td>
<td>48</td>
<td>57</td>
<td>122</td>
</tr>
<tr>
<td>$P_2 \rightarrow t_3, t_4, t_5$</td>
<td>7</td>
<td>74</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>$P_3 \rightarrow t_1, t_6, t_8$</td>
<td>16</td>
<td>106</td>
<td>122</td>
<td></td>
</tr>
</tbody>
</table>
The graphical representation of the optimal assignment has been shown in the figure-4. Out of two clusters one goes to processor $p_2$ while the other to processor $p_3$. The isolated tasks are executed on processor $p_1$. The total optimal system cost for busy period is 122 units which include the impact of the processor distance and inter tasks communication cost. The developed problem is programmed in Visual Basic 6.0 and implemented on the several sets of input data are used to test the effectiveness and efficiency of the algorithm. It is found that the model is suitable for arbitrary number of processors with the random program structure.

IV. REFERENCES


