**Task Allocation in heterogeneous Distributed Real Time System for Optimal Utilization of Processor’s Capacity**

Shipra Singh* and M. L. Garg  
Department of Computer Science, DIT University, Dehradun, India

**Abstract:** In Distributed Real Time System (DRTS), systematic allocation of the tasks among processors is one of the important parameter, which determine the optimal utilization of available resources. If this step is not performed properly, an increase in the number of processing nodes results in decreasing the total system throughput. The Inter-Task Communication (ITC) is always the most costly and the least reliable parameter in the loosely coupled DRTS. In this paper an efficient task allocation algorithm is discussed, which performs a static allocation of a set of \( m \) tasks \( T = \{t_1,t_2,\ldots,t_m\} \) of a program to a set of \( n \) processors \( P = \{p_1,p_2,\ldots,p_n\} \) (where, \( m >> n \)) to minimize the application program’s Parallel Processing Cost (PPC) with the goal to maximize the overall throughput of the system through and allocated load on all the processors should be approximately balanced. While designing the algorithm the Execution Cost (EC) and Inter Task Communication Cost (ITCC) have been taken into consideration.  

**Keywords:** Distributed Real time System, Execution Cost, Inter Task Communication Cost, Task Allocation, Load Balancing

---

**I. Introduction**

The increasing complexity of various real life problems results in greater demand for faster computer components. One of the approaches to meet this growing demand is the use of parallel processing. An alternative and closely related to parallel computers is the concept of DRTS. Distributed real time system is a computer system in which multiple processors connected together through a high-bandwidth communication link. These links provides a medium for each processor to access data and programs on remote processors.

A user-oriented definition of distributed computing is reported by [1,2] that " The Multiple Computers utilized cooperatively to solve problems i.e. to process and maintained the large scale database of the programs which are to be executed on these type of computing environment". The assignment of task to processors is an essential step in exploiting the capabilities of a DRTS and may be done in a variety of ways (i) Static Allocation and (ii) Dynamic 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. These problems may be categorized in static [3 -10]. In order to make the best use of resources in a distributed real time computing environment, it is essential to reassign the tasks dynamically during program execution, so as to the benefit of changes in the local reference patterns of the program [11-18]. Although the dynamic allocation has potential performance advantages, Static allocation is easier to realize and less complex to operate.

Several other methods have been reported in the literature, such as, Integer programming [19, 21], Branch and bound technique [22-23], Matrix reduction technique [7], and reliability evaluation to deal with various design and allocation issues in a DRTS by [24-30]. In this paper we introduce an algorithm which performs static allocation of such program tasks in a heterogeneous DRTS to minimize the application program’s Parallel Processing Cost with the goal to maximize the overall system throughput and allocated load on all the processors should be approximately balanced. Because strictly balanced load distribution may not be possible to achieve, a system is considered to be balanced if the load on each processor is equal to the average load, within a reasonable tolerance. A tolerance of 10-15% of average load is generally chosen. We assume that the number of program modules is much larger than the number of processors, so that no processor remains idle. Several sets of input data are considered to test the efficiency and complexity of the algorithm. It is found that algorithm is suitable for arbitrary number of processors with the random program structure and is workable in all the cases.

**II. Definitions**

**Execution Cost:** The e\(_{ij}\), is the amount of the work to be performed by the executing task t\(_i\) on the processor p\(_j\) where \( 1 \leq i \leq m, 1 \leq j \leq n \). If a task is not executable on any of the processor due to absence of some resources, execution cost of same task on that processor is taken to be \( (\infty) \) infinite. The process of allocation of the problem can be formulated by a function Aalloc, for the task assign to processors j. Aalloc: T→P, where Aalloc (i) = j, if the task t\(_i\) is assigned to processor p\(_j\), the overall EC of a given assignment Aalloc is then computed by equation (1).
Task Allocation in heterogeneous Distributed Real Time System for Optimal ....

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

and the per processor EC for processor \( p_j \) is defined to be

\[ EC(\text{Alloc})_j = \sum_{\substack{1 \leq i \leq m \in TS_j}} e_{i,\text{Alloc} (i)} \quad (2) \]

where \( TS_j \) is the set of tasks allocated to processor \( p_j \)

\[ TS_j = \{ i : \text{Alloc} (i) = j, \quad j=1, 2…n \} \]

**Inter Task Communication Cost:** The ITCC \( c_{ij} \) is incurred between tasks, where \( c_{ij} = g > 0 \) if tasks \( t_i \) communicates with task \( t_j \) for some cost \( g \) when \( \text{Alloc} (i) \neq \text{Alloc} (j) \). Whenever a group of tasks is assigned to the same processor, the \( c_{ij} = 0 \). The overall ITCC of a given assignment \( \text{Alloc} \) can be expressed by equation (3).

\[ \text{ITCC}(\text{Alloc}) = \sum_{\substack{1 \leq i \leq m \leq k \leq m \atop \text{Alloc} (i) \neq \text{Alloc} (k)}} c_{\text{Alloc} (i),\text{Alloc} (k)} \quad (3) \]

and the per processor ITCC is given by equation (4)

\[ \text{ITCC}(\text{Alloc})_j = \sum_{\substack{1 \leq i \leq m \leq k \leq m \atop \text{Alloc} (i) \neq \text{Alloc} (k) \quad \text{Alloc} (i) \in TS_j}} c_{\text{Alloc} (i),\text{Alloc} (k)} \quad (4) \]

**III. Mathematical Model**

Considered an application program that consists of “m” communicating tasks, \( t_1, t_2,….t_m \) and a heterogeneous distributed real time system with “n” processors, \( p_1, p_2,….p_n \), unified by communication relations. It has processors as nodes and there is a weighted edge between two nodes if the corresponding processors can communicate with each other. The weight \( w_{ij} \) on the adjoin between processors \( p_i \) and \( p_j \) represent the time lag involved in sending or receiving the message of unit length from one processor to another. In order to have an approximate estimate of this time lag, irrespective of the two processors, we use the average of the weights on all the edges in the processor graph. This is called the average unit time lag. The load balancing, which involves sending load from over utilized processors to underutilized processors, should be carried out with due regard for communication overhead so that it is completed as speedily as possible. It becomes essential to optimize the overall throughput of the processors by allocating the tasks in such a way that the allocated load on all the processors should be approximately balanced. Therefore, the systematic allocation of tasks in a DRTS is the fundamental requirement for optimal utilization of processor’s capacity.

While developing the algorithm, it is assumed that the processing cost of these tasks on all the processors is given in the form of Execution Cost Matrix \( [ECM] \) of order \( m \times n \) and the ITCC incurring between two communicating when they are assigned to two distinct processors is taken in the form of a symmetric matrix named as Inter Task Communication Cost Matrix \( [\text{ITCCM}] \), which is of order \( m \).

The proposed methodology will work as follow:

- **Computation of Average Load must be assigned to each Processor**

  Select ECM (,) and compute the average load must be assigned to each processor \( p_j \) by using the equation 5 and total load to be assigned on the system by equation 6.

  \[ L_{\text{avg}}(p_j) = \frac{W_j}{n}, \quad j = 1, 2, ..., n \quad (5) \]

  where \( W_j = \sum_{1 \leq i \leq m} e_{ij} \)

www.iosrjournals.org 11 | Page
\[ T_{\text{Ind}} = \sum_{i=1}^{n} \text{Lavg}(p_j) \] (6)

**Determination of Average Minimum Link**
First concentrate on those “n” tasks which have the average minimum link between the tasks using equation (7). The average minimum link is stored in a two dimensional array AML (,) the first column of the array represents the task number and second column represents the average minimum link between the tasks. The array is sorted in ascending order by assuming second column as sorting key.

\[ \text{ITCC}_{\text{avg}}(t_i) = \frac{\text{CC}_i}{m}, \text{where } \text{CC}_i = \sum_{1 \leq j \leq m} c_{ij} \] (7)

**Determination of Initial Assignment**
Select first “n” task from AML (,) and apply the Yadav et al Algorithm [28] on these “n” tasks in ECM (,). Store these assignment in \( T_{\text{ass}} \{ \} \) and also store the processors position in Aalloc{ }. The total number of task allocated to the processor is than stored in TTASK (j) which can be computed by adding the values of Aalloc (j) if a task \( t_i \) is assigned to processor \( p_j \) otherwise continue. Remaining unassigned (m-n) task are then store in \( T_{\text{non-ass}} \{ \} \).

**Clustering of remaining unassigned task**
Remaining (m-n) tasks store in \( T_{\text{non-ass}} \{ \} \) are clustered based on their communication requirement. Highly communicating tasks are clustered together to reduce communication delays. Usually number of tasks clusters should be equal to the number of processor so that one to one mapping may result.

These clusters will be fixed throughout their execution. Since we have ‘n’ number of processors in DRTS, therefore we will make ‘n’ number of tasks clusters. A cluster may contain up to maximum number \( C = \left\lfloor \frac{(m-n)^3}{n} \right\rfloor \) tasks. Store the ITCCM (,) in NITCCM (,) and reduce NITCCM (,) by removing the Tasks Store in \( T_{\text{ass}} \{ \} \) and the upper diagonal \( k=\left\lfloor \frac{(m-n)^3((m-n)-1)/2}{3} \right\rfloor \) of values of NITCCM (,) are stored in a array CCMAX (,) of order \( k \times 3 \) the first column represents first task (say \( r^\text{th} \) task), second column represent the second task (say \( s^\text{th} \) task) and third column represent the ITCC (crs ) between these \( r^\text{th} \) and \( s^\text{th} \) tasks. The CCMAX (,) is sorted in descending order by assuming third column as sorting key. Initially each task is treated like a cluster \( C_i = \{ t_i \} \) for \( i=1 \) to \( m-n \). Store these clusters in a linear array CLS= \{ \( C_i, 1 \leq i \leq m-n \} \). Select the first tasks pair say ( \( t_i, t_j \) ) from CCMAX (,). If the sum of number of tasks for clusters \( C_i \) and \( C_j \) is \( \leq C \), then fuse the clusters \( C_i \) with \( C_j \) otherwise select the next tasks pair from CCMAX (,). Modify CLS= \{ \} by replacing the cluster \( C_i \) as \( C_i \leftarrow C_i \cup C_j = \{ t_i, t_j \} \) and deleting the cluster \( C_j \). Modify the CCMAX (,) by deleting this tasks pair ( \( t_i, t_j \) ) and replace the value between \( t_i \) and \( t_j \) to zero in NITCCM (,) also reduce the matrix by add the \( r^\text{th} \) row with \( s^\text{th} \) and \( r^\text{th} \) and \( s^\text{th} \) column with \( s^\text{th} \). Some of the tasks may not involve in any cluster may be treated as independent task. The above procedure is repeated until and unless we do not get number of tasks clusters equal to number of processors.

**Identification of Final Assignment**
The ECM (,) is also radiuses by summing the corresponding row and apply the Yadav et al Algorithm [28] on these “n” cluster for their allocation.

**Computation of overall EC, ITCC and per processor EC, ITCC**
The overall EC, ITCC and per processor EC, ITCC for processor \( p_j \) of a given assignment Aalloc is then computed by equation (1), equation (2) equation (3) and equation (4) respectively.

**Identification the Service Rate and Throughput of each processor**
The Service Rate (SR) and Throughput (TRP) of the processors are calculated by using the equation (8) and (9) respectively.

\[ \text{SR}_j = \frac{1}{\text{EC}(\text{Aalloc}_j)} \] (8)

\[ \text{TRP}_j = \frac{\text{TTASK}(j)}{\text{EC}(\text{Aalloc}_j)} \] (9)
• **Identification of Parallel Processing Cost (PPC)**

The PPC is a function of the amount of computation to be performed by each processor and the communication load. This function is defined by considering the processor with the heaviest aggregate computation and communication loads. PPC for a given assignment Aalloc is defined guardedly by assuming that computation cannot be overlapped with communication) calculated by using the equation (10)

\[ PPC(Aalloc) = \max_{1 \leq j \leq n} \{EC(Aalloc)_j + ITCC(Aalloc)_j\} \] (10)

• **Computation of Overall System Cost (OSC)**

Overall OSC of the DRTS for the given assignment Aalloc is calculated by using the equation (11)

\[ OSC(Aalloc) = EC(Aalloc) + ITCC(Aalloc) \] (11)

**Procedure**

**Step-1** Input m, n, ECM (,) and ITCCM (,)

**Step-2** AVERAGE_LOAD(:) // Select ECM (,) and Compute the average load must be assigned to each processor p_j also determine the total load can be allocated to the system

**Step-3** AVERAGE_MINIMAL_LINKED(:) Select ITCCM (,) and Determine the average minimally linked between the tasks and store these link in a two dimensional array AML (,) the first column of the array represents the task number and second column represents the average minimum link between the tasks. The array is sorted in ascending order by assuming second column as sorting key.

**Step-4** TASK_MAPPING (): // Select ECM (,) and apply Yadav et al Algorithm [28] in respect of first “n” tasks of ALM (,)

Store these assignment in T_{nm}{} } and also store the processors position in Aalloc{ }

Store in TTASK (j) which can be computed by adding the values of Aalloc (j)

**Step-4.1** Remaining unassigned (m-n) task are then store in T\_non\_ass{}

**Step-5** TASK_CLUSTERS (:)// Select ITCCM (,)and store NITCCM (,)

**Step-5.1** Reduce NITCCM (,) by removing the Tasks Store in Tass{}

Prepared “n” Cluster of the remaining (m-n) tasks

Compute maximum number of tasks in cluster C = \( \left \lfloor \frac{(m-n)}{n} \right \rfloor \)

**Step-5.2** k = \( \left \lfloor \frac{(m-n)((m-n)-1)}{2} \right \rfloor -1 \) Upper diagonal values of NITCCM (,) are stored in a array CCMAX (,)

of order k x 3 the first column represents first task (say r-th task), second column represent the second task (say s-th task) and third column represent the ITCC (crs)

**Step-5.3** The CCMAX (,) is sorted in descending order by assuming third column as sorting key

**Step-5.4** Initially each task is treated like a cluster Ci = {ti} for i=1 to m-n.

Store these clusters in a linear array CLS= {Ci, 1 \leq i \leq m-n}

Select the first tasks pair say ( tr , ts) (say tr \in Cr and ts \in Cs ) from CCMAX (,)

**Step-5.5** If number of tasks for clusters Cr and Cs is \leq C, than fuse the clusters Cr with Cs otherwise select the next tasks pair from CCMAX (,)

**Step-5.6** Modify CLS= {} by replacing the cluster Cr as Cr \leftarrow Cr \cup Cs= [tr , ts] and deleting the cluster Cs.

Modify the CCMAX (,) by deleting this tasks pair (tr , ts) and replace the value between t_r and t_s to zero in NITCCM (,) also reduce the matrix by add the r^{th} row with s^{th} and r^{th} column with s^{th}

**Step-6** Modify the ECM (,) by summing the r^{th} row with s^{th}

**Step-7** If Ci\#n Then Go to step 5.4 Otherwise
Step-8  FINAL_TASK_MAPPPING():// Yadav et al Algorithm [28]
Store these assignment in T_{\text{task}} and also store the processors position in A_{\text{alloc}}
Stored in TTASK (j) which can be computed by adding the values of A_{\text{alloc}} (j)

Step-9  COST_COMPUTATION()://Compute overall EC , per processor EC for processor pj of a given
assignment A_{\text{alloc}}
Compute overall ITCC, per processor ITCC for processor pj of a given assignment A_{\text{alloc}}
Compute overall OSC of the DRTS for the given assignment A_{\text{alloc}}
Compute P_{\text{PC}} of a given assignment A_{\text{alloc}}
Compute Service Rate (SR) and Throughout TRP of the processors

Step-10 Stop

IV. Result & Discussions
To justify the application and usefulness of the present algorithm, a DTRS have been taken which
consist of a set of “n = 3” processors P = \{p_1, p_2, p_3\} connected by an arbitrary network. A typical program graph of a set of “m = 9” tasks T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9\} has been taken from the literature as considered by Yadav et al [31], Younes [32] and Elsadek [3].

Example: Input m = 9, n = 3 ECM (.), and ITCCM (.)

\begin{align*}
\text{ECM}(,) & = \begin{bmatrix}
174 & 156 & 110 \\
95 & 15 & 134 \\
196 & 79 & 156 \\
148 & 215 & 143 \\
44 & 234 & 122 \\
241 & 225 & 27 \\
12 & 28 & 192 \\
215 & 13 & 122 \\
211 & 11 & 208
\end{bmatrix} \\
\text{ITCCM}(,) & = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{align*}

Average and total load to be assigned on each processor is shown in table 1 after calculating by using the
equation 5 and 6.

Table-1: Average and total load to be assigned on each processor

<table>
<thead>
<tr>
<th>Load</th>
<th>p_1</th>
<th>p_2</th>
<th>p_3</th>
<th>Total Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Load</td>
<td>445</td>
<td>325</td>
<td>405</td>
<td>1175</td>
</tr>
<tr>
<td>Tolerance of 10</td>
<td>45</td>
<td>33</td>
<td>40</td>
<td>118</td>
</tr>
<tr>
<td>Total</td>
<td>490</td>
<td>358</td>
<td>445</td>
<td>118</td>
</tr>
</tbody>
</table>

Compute the Average Minimally Linked between the Tasks using equation (7)

\begin{align*}
\text{ALM}(,) & = \begin{bmatrix}
3.22 \\
2.00 \\
2.00 \\
2.11 \\
0.67 \\
1.67 \\
1.56 \\
1.78 \\
3.00
\end{bmatrix} \\
\text{ALM}(,) & = \begin{bmatrix}
0.67 \\
1.56 \\
1.67 \\
2.00 \\
2.11 \\
3.00 \\
2.00 \\
3.22 \\
3.00
\end{bmatrix}
\end{align*}

Initial Allocation obtained by applying Yadav et al [28] Algorithm are given in Table-2
T_{ass} = \{ t_5, t_7, t_9 \}, Aalloc(j) = (p_1, p_2, p_3) and TTASK (j) = (1,1,1)

T_{non-ass} = \{ t_8, t_2, t_3, t_4, t_9, t_1 \}

### Clustering of remaining unassigned task

Maximum number of tasks in cluster c = 2

Store the ITCCM (.) in NITCCM (.) and reduce NITCCM (.) by removing the Tasks Store in T_{ass} = \{ t_5, t_7, t_9 \}

\[
\begin{align*}
\text{TCCMAX}(.) & = \frac{1}{2} t_1 t_2 t_3 t_4 t_5 t_6 \\
& = \begin{bmatrix}
0 & 8 & 10 & 4 & 0 & 0 \\
8 & 0 & 7 & 0 & 3 & 0 \\
10 & 7 & 0 & 1 & 0 & 0 \\
4 & 0 & 1 & 0 & 8 & 0 \\
0 & 3 & 0 & 8 & 0 & 5 \\
0 & 0 & 0 & 0 & 5 & 0 \\
\end{bmatrix}
\end{align*}
\]

Upper diagonal k = [(m-n)*((m-n)-1)/2]-1 = 14 values of NITCCM (.) are stored in an array CMAX (.) of order k x 3

<table>
<thead>
<tr>
<th>Original CMAX (.)</th>
<th>Sorted CMAX (.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1 t_2 t_3 t_4 t_5 t_6</td>
<td>t_1 t_2 t_3 t_4 t_5 t_6</td>
</tr>
<tr>
<td>t_1 t_2 10</td>
<td>t_1 t_2 8</td>
</tr>
<tr>
<td>t_1 t_3 4</td>
<td>t_4 t_6 8</td>
</tr>
<tr>
<td>t_1 t_4 0</td>
<td>t_2 t_5 7</td>
</tr>
<tr>
<td>t_1 t_5 0</td>
<td>t_4 t_6 5</td>
</tr>
<tr>
<td>t_2 t_3 7</td>
<td>t_1 t_4 4</td>
</tr>
<tr>
<td>t_2 t_4 0</td>
<td>t_2 t_5 3</td>
</tr>
<tr>
<td>t_2 t_5 3</td>
<td>t_3 t_4 1</td>
</tr>
<tr>
<td>t_2 t_6 0</td>
<td>t_1 t_8 0</td>
</tr>
<tr>
<td>t_3 t_4 1</td>
<td>t_1 t_8 0</td>
</tr>
<tr>
<td>t_3 t_5 0</td>
<td>t_2 t_4 0</td>
</tr>
<tr>
<td>t_3 t_6 8</td>
<td>t_3 t_6 0</td>
</tr>
<tr>
<td>t_4 t_6 5</td>
<td>t_3 t_6 0</td>
</tr>
</tbody>
</table>

Flowing three clusters are form and shown in Table-3

<table>
<thead>
<tr>
<th>Table-3 Final Clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_1</td>
</tr>
<tr>
<td>C_2</td>
</tr>
<tr>
<td>C_3</td>
</tr>
</tbody>
</table>

After implementation of full procedure the final assignments and EC, ITCC, PPC, Service Rate and Throughput of different processors achieved by the model are shown in Table 4.

<table>
<thead>
<tr>
<th>Table-4 Final Results Obtained by the Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
</tr>
<tr>
<td>(1)</td>
</tr>
<tr>
<td>t_1+t_3</td>
</tr>
<tr>
<td>t_1+t_4</td>
</tr>
<tr>
<td>t_1+t_5</td>
</tr>
<tr>
<td>Total allocated Load</td>
</tr>
</tbody>
</table>

Table 4 and Figure 1 shows the results of the proposed model from the table and figure it is concluded that the Parallel Processing Cost of the system is 401 which is related to processor p_1. Figure 2 shows the through put and services rate of the processor. From the figure it is concluded that both are the ideally linked. The Figure 3 depicted the comparisons between calculated load and allocated load form the figure concluded that the allocated load assigned to the processors is much less than the calculated load.
The present paper deals with a simple, yet efficient mathematical and computational algorithm to identify the optimal busy cost of the system. The performance of the algorithm is compared with the algorithm reported by Yadav et al [31], Elsadek et al [3] and Younes [32]. The figure 4 and table 5 shows the comparisons of optimal cost of the system reported by the [31, 32] and the present method. From the figure it is concluded that the PPC of the system is much less obtained by the present method.

<table>
<thead>
<tr>
<th>Table -5: Comparisons of PPC of the system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yadav et al [31]</td>
</tr>
<tr>
<td>Elsadek [3]</td>
</tr>
<tr>
<td>Younes [32]</td>
</tr>
<tr>
<td>Present Model</td>
</tr>
</tbody>
</table>

It can also be perceived from the example presented here that wherever the algorithm of better complexity is encountered Kumar et al [6] present technique gets an upper hand by producing better optimal results with slight enhancement in the cost due to minor in complexity factor. The worst case run time complexity of the algorithm suggested by Kumar et al [6] is \( O(m^2n+n^2 + 2mn) \), Elsade et a [3] is \( O(n^2 + m^2 + m^2n+2mn) \) and the run complexity of the algorithm presented in this paper is \( O(m^2n+mn+n^2) \). Table 6 and Figure 4 shows the run time complexity of Kumar et al [6], Elsade et al [3] and Present Method considering different cases of tasks and processors.
Table-6: Run time complexity compression

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>O (m*n)</td>
<td>O (m^n + 2mn + n^2)</td>
<td>O (m*(mn+m+n))</td>
</tr>
<tr>
<td>n'</td>
<td>m'</td>
<td>m'n'</td>
<td>m'n'</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>

Fig. 4: Run time complexity compression

References

Task Allocation in heterogeneous Distributed Real Time System for Optimal ....