Optimal sizing, loading and re-loading in a multi-level memory hierarchy system

by S. R. ARORA

University of Minnesota and UNIVAC Division, Sperry Rand Corporation
Roseville, Minnesota

and

A. GALLO

UNIVAC Division, Sperry Rand Corporation
Roseville, Minnesota

Starting with the appearance of the third generation computers the demand for memory storage devices has increased. This increased demand has been for both the main storage and auxiliary storage devices. The major reasons for this increase were the problems of larger data processing tasks and the introduction of multiprogramming and time-sharing.

The increase of memory storage space cannot be accomplished economically without reducing the speeds of the memories. System designers have been trying to exploit the heterogeneous nature of programs and data files so that files that are least frequently accessed are stored in slower memory modules and files with high activity are loaded in faster memory modules. This has been done more or less on heuristic basis.

This problem became more acute with the introduction of the directly addressable bulk. In this system organization the processor executes instructions and data fetches directly from the bulk core, which operates at a lower speed than the main memory. The proper allocation of the data and program segments in different memory modules became very critical in the overall performance.

This paper studies the effects of loading and re-loading of program and data segments in various directly addressable memory modules. It seeks optimal rules of loading and re-loading. It also studies the problem of optimal sizing of different memory modules.

We first select the criterion of effectiveness with respect to the problem of memory allocation and memory sizing decisions. We define the central processor unit and the directly addressable memory levels as a sub-system of the whole computer system and take response time as the performance measure of this sub-system. Considering the central processor as a single server, the response time of the sub-system is the serial sum of the instruction execution times of the various program modules making up a benchmark. We will call the individual programs and data segments “objects.” The collection of all objects is our benchmark. We characterize an object through the following parameters.

\[ j = \text{number identifying an object} \]

\[ K = \text{total number of objects in the benchmark} \]

\[ j = 1, 2, \ldots, K \]

\[ v_j = \text{size of the } j\text{th object} \]

\[ p_j = \text{total number of references made to object } j \]

\[ \text{during the total benchmark run time} \]

\[ i_j = \text{average number of instructions or data fetches} \]

\[ \text{per reference to object } j \]

\[ R = \text{total run time of the selected benchmark} \]

As stated earlier, an object can be a set of instructions or a set of data fetches. It can belong to the operating system or to the user’s library. Some of the objects will show high level of activity, which means that the processor will loop through a small set of instructions many times during one reference. Typically, loop bound programs are of this type. In other cases out of a large quantity of instructions or data only a few will be executed during one reference. Data banks and decision bound programs typically belong to this category.
OPTIMAL LOADING RULE IN VARIOUS DIRECTLY ADDRESSABLE MEMORY LEVELS

As stated earlier, our first objective is to determine how to load the given $K$ objects in memory levels with different cycle times so that the total processing time of these objects is minimized.

Define

$$h_j = \frac{p_j}{v_j}; \quad j=1, 2, \ldots, K$$ (1)

Then the optimal loading rule is that the objects are loaded, starting from the fastest memory level to the slowest in descending order of the index $h_j$.  

**Theorem**

The total processing time $R$ for processing $K$ objects in an $N$ level memory system with sizes $S_n$ is minimized if objects are loaded within these levels, starting with the fastest memory in descending order of the quantity $h_j$.

**Proof**

Consider two objects $j$ and $j'$ with sizes $v_j$ and $v_{j'}$ and indexes $h_j$ and $h_{j'}$ such that $h_j > h_{j'}$. Consider two adjacent memory levels $n$ and $n+1$, with speeds $t_n$ and $t_{n+1}$. Let object $j$ be currently loaded in level $n$ and object $j'$ in level $n+1$, which is contrary to the optimal loading rule. Let $R'$ be the total response time with the current loading and $R$ be the total response time if objects $j$ and $j'$ were to interchange their levels over the space $v$, where $v = \min(v_j, v_{j'})$. Values of $R$ and $R'$ are given by

$$R' = L + t_n p_{j'} \cdot \frac{v}{v_j} + t_{n+1} p_{j'} \cdot \frac{v}{v_{j'}}$$ (2)

$$R = L + t_{n+1} p_j \cdot \frac{v}{v_j} + t_n p_{j} \cdot \frac{v}{v_{j'}}$$ (3)

where $L$ is the response time of objects other than $j$ and $j'$. The ratios $v/v_j$, $v/v_{j'}$, denote the fractions of the objects $j$ and $j'$ which are affected by the above interchange over the space $v$.

Substituting (1) in (2) and (3), we obtain

$$R' = L + t_n h_{j'} \cdot v + t_{n+1} h_j \cdot v$$ (4)

$$R = L + t_{n+1} h_{j} \cdot v + t_n h_{j'} \cdot v$$ (5)

Subtracting (4) from (5) we obtain

$$R - R' = (t_{n+1} - t_n) (h_{j'} - h_j) \cdot v$$

Since $(t_{n+1} - t_n)$ is positive and $(h_{j'} - h_j)$ is negative by assumptions, $R - R'$ is negative. In other words the total response time has been reduced by the above interchange of objects $j$ and $j'$. Continuing similar interchange over other objects, we will converge to the optimal loading rule, which will minimize the total response time. Arranging all the objects in the descending order of index $h_j$ makes the loading rule optimal for any $N$ level memory system of any sizes, including the particular case when the $\{S_n\} = \{v_j\}$.

The heterogeneity of the objects makes it economical to execute them from memory levels with different speeds. Let us define the following parameters for our memory hierarchy system.

$$t_n = \text{instruction execution time per instruction from the } n\text{th memory level}$$

$$c_n = \text{cost per unit space per unit time for the } n\text{th memory level}$$

Assume that memories are numbered according to their speeds, the fastest is assigned number 1 and the slowest is assigned number $N$. It is clear that a memory hierarchy with different speeds and different costs for various memory levels is meaningful only if they satisfy the following relationships:

$$t_1 < t_2 < \cdots < t_N$$

$$c_1 > c_2 > \cdots > c_N$$ (7)

**ACTIVITY PROFILE CURVE**

We define the activity level of program $j$ to be equal to $p_{j}$, which is also equal to $v_j h_j$ according to equation (1). The activity profile curve establishes a relationship between the cumulative activity level over the collection of objects and the cumulative memory space demand for these objects, when objects are assumed to be numbered according to the index $h_j$. The object with the highest value of the index is numbered 1 and the one with the lowest value is numbered $K$. Let $F(s)$ in Figure 1 denote the activity profile curve. $F(s)$ and $s$ are defined as below:

$$F(s) = \sum_{j=1}^{k} p_{j} h_j$$

$$s = \sum_{j=1}^{k} v_j$$ (8)
where $k$ is a subset of the objects $K$. In particular point $A$ on the $s$-axis and point $B$ on the $F(s)$ axis are given by

$$A = \sum_{j=1}^{K} v_j$$

$$H = \sum_{j=1}^{K} p_j$$

(9)

If we normalize $\sum_{i=1}^{K} p_{j,i}/\sum_{i=1}^{K} p_{j,i}$, then we have a cumulative distribution curve. Let $F'(s)$ denote the normalized function. The shape of the curve $F'(s)$ can be obtained by curve fitting techniques from the actual file data. Exponential curve given by $F(s) = 1 - e^{-as}$ was observed to fit several benchmarks, where the parameter characterizes the particular shape of the function for the given benchmark.

OPTIMAL SIZING OF VARIOUS ADDRESSABLE MEMORY LEVELS

The response time $R$ for the given benchmark in the specified sub-system of the processor and directly addressable memories is given by

$$R = \sum_{n=1}^{N} \sum_{j=1}^{K} t_{n,j}x_{n,j}$$

(10)

where $0 \leq x_{n,j} \leq 1$. It takes the value $0$, if the object is not loaded in the $n$th level, a value of 1, if it is completely loaded and a value $0 < x_{n,j} < 1$, if it is fractionally loaded in the $n$th level.

Let $s_1, \ldots, s_n, \ldots, s_N$ denote the sizes for the $N$ memory levels. Assuming that loading of the objects follows the optimal loading rule, the response time $R$

may be written as

$$R = t_1 F(s_1) + t_2 [F(s_1 + s_2) - F(s_1)] + \cdots$$

$$+ t_N \left[ F\left(\sum_{n=1}^{N} s_n\right) - F\left(\sum_{n=1}^{N-1} s_n\right)\right]$$

(11)

Equation (11) is derived from equation (10) by substituting $F(s)$ for $\sum_{i=1}^{K} p_{j,i}$ as defined in equation (8).

The cost of processing the benchmark is given by the following equation.

$$Z = R \left( \sum_{n=1}^{N} c_n s_n + G \right)$$

(12)

where $G$ is the unit time rental cost of the processor. $Z$ measures the total cost incurred by the sub-system for running the benchmark. The quantity

$$\left( \sum_{n=1}^{N} c_n s_n + G \right)$$

denotes the unit time rental cost of the sub-system and this rental cost is paid for the total response time $R$.

The objective is to minimize $Z$ subject to the constraint that the response time has to be smaller than a certain fixed number. Mathematically, the problem of optimal sizing may be stated as

Find $s_n \geq 0$ for $n = 1, \ldots, N$

which minimize

$$Z = R \left( \sum_{n=1}^{N} c_n s_n + G \right)$$

subject to

$$R \leq R_0$$

(14)

Minimization may be achieved by defining the Lagrange Function $Z'$ and applying Kuhn Tucker conditions. We will simplify our problem by initially ignoring the non-negativity constraint on $s_n$ and also by treating (14) as an equality constraint. If on solving the first order conditions, it is found that some of the $s_n$ values
are negative, such solutions will be unfeasible. The boundary solutions, where some of the \( s_n \) values were zero, will be examined systematically according to the algorithm described below. The Lagrange function \( Z' \) is given by

\[
Z' = R \left( \sum_{n=1}^{N} c_n s_n + G \right) - \lambda (R - R_0) \tag{15}
\]

If \( Z' \) is minimized at \( (s_1^*, \ldots, s_n^*, \ldots, s_N^*) \) the following conditions must be satisfied at the optimal point.

\[
\frac{\partial Z'}{\partial s_n} = \frac{\partial R}{\partial s_n} \left( \sum_{n=1}^{N} c_n s_n + G \right) - \lambda \frac{\partial R}{\partial s_n} + R c_n = 0
\]

\[
= \frac{\partial R}{\partial s_n} \left( \sum_{n=1}^{N} c_n s_n + G - \lambda \right) + R c_n = 0 \tag{16}
\]

for \( n = 1, 2, \ldots, N \)

\( \partial R / \partial s_n \) defines the rate of change of the total response time \( R \) with respect to a change in \( s_n \), keeping the size of other levels unchanged. Subtracting equation (16) with index \( (n-1) \) from the same equation with index \( (n) \), we obtain

\[
\left( \frac{\partial R}{\partial s_n} - \frac{\partial R}{\partial s_{n-1}} \right) \left( \sum_{n=1}^{N} c_n s_n + G - \lambda \right) + R (c_n - c_{n-1}) = 0 \tag{17}
\]

for \( n = 2, 3, \ldots, N \)

On re-arranging we obtain

\[
\frac{\frac{\partial R}{\partial s_n} - \frac{\partial R}{\partial s_{n-1}}}{c_{n-1} - c_n} = \frac{R}{\sum_{n=1}^{N} c_n s_n + G - \lambda} \tag{18}
\]

for \( n = 2, 3, \ldots, N \)

By differentiating \( R \) from equation (11) with respect to \( s_n \) and \( s_{n-1} \) and subtracting the two derivatives we obtain

\[
\frac{\partial R}{\partial s_n} - \frac{\partial R}{\partial s_{n-1}} = (t_n - t_{n-1}) f(s_1 + s_2 + \cdots + s_{n-1}) \tag{19}
\]

where \( f(s) \) denotes the derivative of the function \( F(s) \).

Substituting (19) in (18), we obtain

\[
\frac{(t_n - t_{n-1})}{(c_{n-1} - c_n)} f(s_1 + s_2 + \cdots + s_{n-1}) = \frac{R}{\sum_{n=1}^{N} c_n s_n + G - \lambda} = \text{constant} \tag{20}
\]

for \( n = 2, 3, \ldots, N \)

From equation (20) we obtain \((N-2)\) equations of the form

\[
\frac{(t_n - t_{n-1})}{(c_{n-1} - c_n)} f(s_1 + s_2 + \cdots + s_{n-1}) = \frac{(t_{n+1} - t_n)}{(c_n - c_{n+1})} f(s_1 + s_2 + \cdots + s_n) \tag{21}
\]

We also have to satisfy the following two conditions, which provide us with the additional two equations needed to solve the optimal values of \( s_n \).

\[
s_1 + s_2 + \cdots + s_N = \sum_{j=1}^{K} v_j \tag{22}
\]

\[
R = R_0 \tag{23}
\]

SEQUENCE OF ADMISSIBLE MEMORIES

It may be recalled that while defining the activity profile curve, objects were numbered in descending order of the index \( h_j \). Consequently the derivative \( f(s) \) of the activity profile curve \( F(s) \) is a monotonic non-increasing function of \( s \). From equation (21) we conclude that the sequence \( \{(t_n - t_{n-1}) / (c_{n-1} - c_n)\} \) for all \( n \) is monotonic non-decreasing. Any memory level which does not comply with this rule is a non-admissible memory level. Memory levels of this type can be eliminated from search for optimal solutions.

ALGORITHM FOR SOLVING OPTIMAL VALUES OF MEMORY SIZES

Optimal solution of the memory sizes has to satisfy equations (21) through (23). After the non-admissible memories have been eliminated, the sequence \( \{(t_n - t_{n-1}) / (c_{n-1} - c_n)\} \) for the admissible memories is monotonic non-decreasing. Also if objects are arranged in the descending order of the index \( h_j \), the derivative \( f(s) \) of the activity profile curve \( F(s) \) is a monotonic non-increasing function of \( s \).

The following algorithm will systematically watch for the compliance of equations (21) through (23). Initially load all the objects in the slowest memory level \( N \), and test whether the response time computed with equation (11) satisfies the specified response time constraint given in equation (23). If this constraint is satisfied, then the optimal solution has already been found, otherwise include the next faster memory level \( N-1 \).

We have two unknown values which we can solve by applying equations (22) and (23). If equation (23) indicates that the response time constraint cannot be
TABLE I—Airline Reservation Benchmark

<table>
<thead>
<tr>
<th>j</th>
<th>id</th>
<th>h_i</th>
<th>p_ji</th>
<th>( \sum p_ji )</th>
<th>v_i</th>
<th>( \sum v_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P</td>
<td>8.66</td>
<td>18300</td>
<td>18300</td>
<td>2112</td>
<td>2112</td>
</tr>
<tr>
<td>2</td>
<td>P</td>
<td>6.14</td>
<td>19860</td>
<td>21270</td>
<td>254</td>
<td>2366</td>
</tr>
<tr>
<td>3</td>
<td>P</td>
<td>.8125</td>
<td>21420</td>
<td>25320</td>
<td>6000</td>
<td>10670</td>
</tr>
<tr>
<td>4</td>
<td>P</td>
<td>6.0</td>
<td>1920</td>
<td>27240</td>
<td>3200</td>
<td>13870</td>
</tr>
<tr>
<td>5</td>
<td>P</td>
<td>.558</td>
<td>1536</td>
<td>28776</td>
<td>2752</td>
<td>16822</td>
</tr>
<tr>
<td>6</td>
<td>P</td>
<td>.441</td>
<td>28888</td>
<td>2752</td>
<td>254</td>
<td>16876</td>
</tr>
<tr>
<td>7</td>
<td>P</td>
<td>.361</td>
<td>30188</td>
<td>832</td>
<td>17798</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>P</td>
<td>.320</td>
<td>32188</td>
<td>1800</td>
<td>36120</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>P</td>
<td>.291</td>
<td>375</td>
<td>29985</td>
<td>1500</td>
<td>19208</td>
</tr>
<tr>
<td>10</td>
<td>P</td>
<td>.260</td>
<td>31454</td>
<td>6000</td>
<td>26296</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>P</td>
<td>.226</td>
<td>780</td>
<td>32325</td>
<td>384</td>
<td>26680</td>
</tr>
<tr>
<td>12</td>
<td>P</td>
<td>.223</td>
<td>780</td>
<td>33105</td>
<td>3500</td>
<td>30180</td>
</tr>
<tr>
<td>13</td>
<td>P</td>
<td>.148</td>
<td>780</td>
<td>33980</td>
<td>3500</td>
<td>33680</td>
</tr>
<tr>
<td>14</td>
<td>P</td>
<td>.140</td>
<td>2327</td>
<td>34232</td>
<td>1800</td>
<td>36120</td>
</tr>
<tr>
<td>15</td>
<td>P</td>
<td>.118</td>
<td>375</td>
<td>34607</td>
<td>3168</td>
<td>39288</td>
</tr>
<tr>
<td>16</td>
<td>P</td>
<td>.118</td>
<td>375</td>
<td>35117</td>
<td>34000</td>
<td>379288</td>
</tr>
<tr>
<td>17</td>
<td>D</td>
<td>15.0×10^-4</td>
<td>510</td>
<td>35548.7</td>
<td>516800</td>
<td>896088</td>
</tr>
<tr>
<td>18</td>
<td>D</td>
<td>8.35×10^-4</td>
<td>431.7</td>
<td>35548.9</td>
<td>330</td>
<td>896418</td>
</tr>
<tr>
<td>19</td>
<td>D</td>
<td>6.66×10^-4</td>
<td>.2</td>
<td>35549.46</td>
<td>2112</td>
<td>89530</td>
</tr>
<tr>
<td>20</td>
<td>D</td>
<td>2.65×10^-4</td>
<td>.56</td>
<td>35549.46</td>
<td>2112</td>
<td>89530</td>
</tr>
<tr>
<td>21</td>
<td>D</td>
<td>2.22×10^-4</td>
<td>28</td>
<td>35577.46</td>
<td>126000</td>
<td>1024530</td>
</tr>
<tr>
<td>22</td>
<td>D</td>
<td>1.47×10^-4</td>
<td>41.97</td>
<td>35610.43</td>
<td>285360</td>
<td>1308890</td>
</tr>
<tr>
<td>23</td>
<td>D</td>
<td>1.3×10^-4</td>
<td>3.0</td>
<td>35622.43</td>
<td>215000</td>
<td>1331900</td>
</tr>
<tr>
<td>24</td>
<td>D</td>
<td>.972×10^-4</td>
<td>5.25</td>
<td>35627.68</td>
<td>54012</td>
<td>1385402</td>
</tr>
<tr>
<td>25</td>
<td>D</td>
<td>.62×10^-4</td>
<td>3.0</td>
<td>35630.68</td>
<td>45000</td>
<td>1433402</td>
</tr>
<tr>
<td>26</td>
<td>D</td>
<td>.53×10^-4</td>
<td>.07</td>
<td>35630.75</td>
<td>1320</td>
<td>1434722</td>
</tr>
<tr>
<td>27</td>
<td>D</td>
<td>.456×10^-4</td>
<td>27.36</td>
<td>35658.11</td>
<td>60000</td>
<td>2034722</td>
</tr>
<tr>
<td>28</td>
<td>D</td>
<td>.433×10^-4</td>
<td>4.9</td>
<td>35663.01</td>
<td>113154</td>
<td>2147876</td>
</tr>
<tr>
<td>29</td>
<td>D</td>
<td>.38×10^-4</td>
<td>5.4</td>
<td>35668.41</td>
<td>143160</td>
<td>2291236</td>
</tr>
<tr>
<td>30</td>
<td>D</td>
<td>.20×10^-4</td>
<td>.02</td>
<td>35668.43</td>
<td>1000</td>
<td>2292236</td>
</tr>
<tr>
<td>31</td>
<td>D</td>
<td>.185×10^-4</td>
<td>7.99</td>
<td>35676.42</td>
<td>432000</td>
<td>2724336</td>
</tr>
<tr>
<td>32</td>
<td>D</td>
<td>.175×10^-4</td>
<td>105.00</td>
<td>35781.42</td>
<td>600000</td>
<td>8724236</td>
</tr>
<tr>
<td>33</td>
<td>D</td>
<td>.167×10^-4</td>
<td>1.5</td>
<td>35782.92</td>
<td>90024</td>
<td>8814260</td>
</tr>
<tr>
<td>34</td>
<td>D</td>
<td>.159×10^-4</td>
<td>.07</td>
<td>35782.99</td>
<td>4400</td>
<td>1818600</td>
</tr>
<tr>
<td>35</td>
<td>D</td>
<td>.09×10^-4</td>
<td>45.20</td>
<td>35828.19</td>
<td>763000</td>
<td>1644860</td>
</tr>
<tr>
<td>36</td>
<td>D</td>
<td>.0415×10^-4</td>
<td>.49</td>
<td>35828.68</td>
<td>118060</td>
<td>16566720</td>
</tr>
<tr>
<td>37</td>
<td>D</td>
<td>.033×10^-4</td>
<td>.1</td>
<td>35828.78</td>
<td>30030</td>
<td>16596750</td>
</tr>
<tr>
<td>38</td>
<td>D</td>
<td>.030×10^-4</td>
<td>4.04</td>
<td>35832.82</td>
<td>1333322</td>
<td>17830102</td>
</tr>
<tr>
<td>39</td>
<td>D</td>
<td>.0249×10^-4</td>
<td>.05</td>
<td>35832.87</td>
<td>20064</td>
<td>17950066</td>
</tr>
<tr>
<td>40</td>
<td>D</td>
<td>.0207×10^-4</td>
<td>1.0</td>
<td>35833.87</td>
<td>482600</td>
<td>18429666</td>
</tr>
<tr>
<td>41</td>
<td>D</td>
<td>.016×10^-4</td>
<td>.8</td>
<td>35834.67</td>
<td>264000</td>
<td>18696666</td>
</tr>
<tr>
<td>42</td>
<td>D</td>
<td>.00413×10^-4</td>
<td>.01</td>
<td>35834.68</td>
<td>21102</td>
<td>18797768</td>
</tr>
<tr>
<td>43</td>
<td>D</td>
<td>.0016×10^-4</td>
<td>.01</td>
<td>35834.69</td>
<td>60060</td>
<td>18777828</td>
</tr>
<tr>
<td>44</td>
<td>D</td>
<td>.0014×10^-4</td>
<td>.02</td>
<td>35834.71</td>
<td>151922</td>
<td>18928860</td>
</tr>
<tr>
<td>45</td>
<td>D</td>
<td>.00012×10^-4</td>
<td>.0003</td>
<td>35834.713</td>
<td>240042</td>
<td>19168902</td>
</tr>
</tbody>
</table>

satisfied, the next faster memory has to be included and now three equations with three unknowns have to be solved; equations (23), (22) and out of the set of equations given in (21) we include the following:

\[
\frac{t_{N-1} - t_{N-2}}{c_{N-1} - c_{N}} f(s_1, s_2, \ldots, s_{N-2}) = \frac{t_N - t_{N-1}}{c_N} f(s_1, s_2, \ldots, s_{N-1})
\]

If the response time constraint cannot be satisfied, we include the next faster memory and bring in one more equation from the set of equations given in (21). The first time a feasible solution exists, we try to obtain the optimal solution. If we want to carry on the optimization with the technical constraints of the smallest size of the memory modules, an adjustment can be made so that \( R \leq R_\text{s} \) should be observed. If the response time constraint cannot be satisfied, including the last
available memory level, then no feasible solution exists.

If the solution to the involved equations violates the non-negativity constraint on some \( s_n \) in the set of included memory levels, then all combinations of various \( s_n \) being equal to zero have to be tested for the optimal solution.

**Example**

In this example we are taking an airline reservation system. The memory allocation for the objects in an airline reservation system is reasonably static. The granularity of the program and data files recognizes 47 objects with a total memory requirement of 19.2M words. Suppose we decided to buy 2M words of directly addressable memory levels. We have to make the selection out of three available memory levels. The speeds of these memory levels are

\[
l_1 = .5 \mu s \quad l_2 = .75 \mu s \quad l_3 = 1.5 \mu s
\]

The proportional costs of the respective memory levels are

\[
ce_1 = 7 \quad c_2 = 5 \quad c_3 = 1
\]

Table I contains the file elements arranged in the descending order of the index \( h_j \), along with their parameters \( h_j, p_j, \sum p_j, v_j \), and \( \sum v_j \). The second column of the table identifies the objects; \( P \) stands for programs and \( D \) stands for data.

For our purposes we fitted an exponential curve over the first 29 objects for which the directly addressable memory is being considered. The equation of the exponential curve is given by

\[
F(s) = 1 - e^{-(a_0 + a_1 s)}
\]

The least square estimates for \( a_0 \) and \( a_1 \) are

\[
\hat{a}_0 = 1.858 \\
\hat{a}_1 = 2.4166 \times 10^{-4}
\]

It may be verified that all three memory levels are admissible.

Consider the three cases when \( R \leq 30 \) msec., \( R \leq 20 \) msec. and \( R \leq 10 \) msec. respectively.

For \( R = 30 \) assume we have \( s_3 \) memory level only. Multiplying the total activity with the instruction execution time of the 3rd memory level we obtain 53.5 msec. which does not satisfy the response time constraint.

Including \( s_2 \) we solve equations (22) and (23) and obtain

\[
s_2 = 1973704 \\
s_3 = 26296
\]

For \( R = 20 \) the response time constraint cannot be satisfied with memory levels \( s_2 \) and \( s_3 \) only. \( s_1 \) also has to be included. Solving equations (21), (22) and (23) we obtain

\[
s_1 = 265660 \\
s_2 = 168250 \\
s_3 = 1566090
\]

For \( R = 10 \) no feasible solution exists. This can be seen also when we multiply the total activity level by the instruction execution time obtained with the fastest memory level only. This is 17.83 msec. This is the best response time which can be obtained by buying the fastest memory level only, which is the most expensive system.

**DYNAMIC ALLOCATION**

In the real life one benchmark may not be representative of the working conditions of a computer system. It might be desirable to observe several benchmarks. The collection of objects in different benchmarks might overlap. This is particularly evident for the operating systems, which will be accessed by all the benchmarks. Under these conditions it may be meaningful to re-locate some of the objects. The criterion of re-location is that if the improvement in response time is greater than the re-location overhead, including the work required to do time-to-time checking for carrying on necessary re-location, then re-location is desirable.

Define the following notations.

\( K_{r,n} \) = set of objects accepted in memory level \( n \) at decision time \( r \) for the time period \((r, r+1)\) (Figure 3).

\[
K_{r,n} = \{K_{r,n}^* \mid \text{when no re-allocation is decided at time } r\}
\]

\( K_{r,n}^* \) = set of objects that should be loaded in level \( n \) at time period \( r \), if optimal loading rule was followed.

\[
K_{r,n} = \begin{cases} \left(K_{r-1,n} \right) & \text{when no re-allocation is decided at time } r \\ \left(K_{r,n}^* \right) & \text{when re-allocation is decided at time } r \end{cases}
\]

\( J_{n,m} \) = set of objects that have to be transferred from level \( n \) to level \( m \), if re-allocation is decided at time \( r \).
only the ranking of one index has to be observed from time to time.

REFERENCES

1 Y C CHEN  S C HSIEH
Selective transfer analysis
IBM Research

2 A L VEREHA  R M RUTLEDGE  M M GOLD
Strategies for structuring two level memories in a paging environment
Carnegie-Mellon University Research Paper

3 C V RAMAMOORTHY  K M CHANDY
Optimization of memory hierarchies in multiprogrammed systems
ACM Journal July 1970

4 F BASKETT  J C BROWNE  W M RAIKE
The management of a multi-level non-paged memory system
AFIPS Spring Joint Computer Conference 1970

5 S R ARORA  A GALLO
Optimal loading, reloading, memory sizing in a system with multi-level memory hierarchy
UNIVAC Tech Memo 01EN 108

6 P DENNING
Thrashing: Its causes and prevention
AFIPS March 1969

7 D N FREEMAN
A storage hierarchy system for batch processing
AFIPS March 1968

8 L A BELADY
A study of replacement algorithms for a virtual storage computer
IBM Systems Journal Vol 5 No 2 1966