DELAYED-STAGING HIERARCHY OPTIMIZATION

by

Gabriel M. Silberman

Technical Report No. 192

January, 1981
DELAYED-STAGING HIERARCHY OPTIMIZATION

GABRIEL M. SILBERMAN

Technion, Israel Institute of Technology, Haifa, Israel

ABSTRACT: An optimization procedure is developed for Delayed-Staging hierarchies using Geometric Programming techniques, based on power forms for both the technology-cost (access time) and fault-ratio functions.

The criterion for optimization is the minimization of average hierarchy access time subject to a cost constraint (i.e. a maximum allowed price), given the cache access time and the backing-store capacity.

Keywords: Storage Hierarchies, Non-linear Programming, Geometric Programming.

1. INTRODUCTION

Delayed-Staging (D-S) storage hierarchies [1,2] are a special case of non-linear configurations, in which the presence of extra paths between the k fastest levels and the CPU is allowed (see Fig.1). This generalization of linear hierarchies [3] enables some parallelism in the flow of data through the hierarchy.

This paper presents a procedure whereby the parameters (i.e. access time and capacity for each level) for an optimal D-S hierarchy configuration may be chosen. The criterion for optimization is the minimization of average hierarchy access time, subject to a cost constraint, given the cache (fastest, smallest level) access time, the backing-store (slowest, largest level) capacity and n, the total number of levels.

In order to solve the resulting non-linear optimization problem,
we use power function forms for the fault-ratios and (technology) cost functions. This enables the application of Geometric Programming techniques [4,5] to obtain the solution in a closed form. Section 2 presents the problem statement and its assumptions, especially those associated with the usage of power functions. Section 3 then develops the solution for the general case, i.e., finding the optimum average access time. In Section 4, a particular instance of the general case (only two levels connected directly to the CPU, see Figure 2) is fully derived. Once the optimum average access time is determined, Section 5 shows how to derive the matching configuration, i.e. capacity and access time for each level in the D-S hierarchy.

Some numerical examples of the optimization procedure follow in Section 6, the results being compared to equivalent cases of linear hierarchies, using the results derived by Chow in [6]. Section 7 summarizes and points to further research in the area.

2. STATEMENT OF THE PROBLEM AND ASSUMPTIONS

We denote the access time to level \( i \) by \( t_i \); then a hit at level \( i \) is serviced in time \( T_i \) given by:

\[
T_i = \begin{cases} 
  t_i, & 1 \leq i \leq k \\
  \frac{1}{\Sigma_{j=k}^{i} t_j}, & k + 1 \leq i \leq n
\end{cases}
\]

(1)

The average hierarchy access time is then given by

\[
T = \sum_{i=1}^{n} H_i T_i
\]

(2)

where \( H_i \) is the probability of a hit at level \( i \). The \( H_i \) can be
expressed in terms of $F_i$, the probability of a fault at level $i$, using

$$
H_i = \begin{cases} 
1-F_i & i = 1 \\
F_{i-1} + F_i & 2 \leq i \leq n-1 \\
F_{n-1} & i = n,
\end{cases}
$$

(3)

Thus, (2) reduces to

$$
T = t_1 + \sum_{i=2}^{n} (F_{i-1} t_1 - \sum_{i=1}^{k-1} F_i t_4).
$$

(4)

We assume that the fault-ratio probability is a function of the capacity only, i.e., $F = F(C)$ (later we will see where other factors, such as page size could come into play).

The cost of a unit of storage at level $i$ is assumed to be a function of its access time, i.e., $S_i = g(t_i) C_i$, therefore the total cost is given by:

$$
S = \sum_{i=1}^{n} g(t_i) C_i.
$$

(5)

We can now state the optimization problem as follows:

**Given:**
- $t_1$ - cache access time
- $C_n$ - backing store capacity
- $S_0$ - total hierarchy cost (constraint)
- $F(C)$ - fault-ratio function
- $g(t)$ - technology cost function
- $n$ - number of storage levels in the hierarchy
- $k$ - levels directly connected to CPU.

**Variables:**
- $t_i$, $2 \leq i \leq n$ - access times
- $C_i$, $1 \leq i \leq n-1$ - capacities.

* Remember we require $t_1 < t_2 < \ldots < t_n$; this fact will come into play later on.
Objective function:

\[
\text{Minimize } T = t_1 + \sum_{i=2}^{n} F(C_{i-1}) t_i - \sum_{i=1}^{k-1} F(C_i) t_i, \quad (6)
\]

Constraints:

\[
S = \sum_{i=1}^{n} g(t_i) C_i \leq S_0 \quad (7)
\]

\[
t_1 > 0, \quad 2 \leq i \leq n
\]

\[
c_i > 0, \quad 1 \leq i \leq n-1.
\]

Notice that since \( t_1 \) is given, the function to be minimized is actually \( T - t_1 \).

Power functions:

We assume for both the technology cost and fault-ratio functions, a power function form, i.e.,

\[
F(C) = F_0 C^{-a}, \quad \text{for } F_0^{1/a} \leq C \leq C_p
\]

\[
F_0, \quad a > 0.
\]

Without loss of generality we take \( F_0 = 1 \), i.e., use \( F_0^{1/a} \) as the unit of storage capacity,

\[
F(C_i) = C_i^{-a}, \quad 1 \leq i \leq n-1.
\]

For the cost function:

\[
g(t) = g_0 t^{-b}, \quad \text{for } t, g_0, b > 0.
\]

Again, we take \( g_0 = 1 \), i.e.

\[
g(t_1) = t_1^{-b}, \quad 2 \leq i \leq n.
\]

It is worth mentioning that \( a \) and \( b \) can be replaced by different \( a_i \) and \( b_i \) for each level without invalidating the solution procedure. However, we use single values because it is adequate to illustrate the method and serves to obtain a simpler form in the final solution.
Empirical evidence seems to support the power functions assumption and their usage greatly simplifies the problem statement. Moreover, they enable the application of Geometric Programming to obtain the solution.

Several papers ([6, 7, 8, 9, 10]) use this assumption and their results provide useful insight into the hierarchy optimization problems they tackle (e.g. average access time, task-switching policy, reliability). These papers though, treat only linear (we call them regular) storage hierarchies, which yield a simpler form for the Geometric Programming problem (no degrees of freedom, see below). They do not handle non-linear cases such as Delayed-Staging hierarchies for which a method is developed here.

By using (8) and (9), the optimization given by (6) and (7) transforms to:

\[
\begin{align*}
\text{Minimize } T_{-v_1} &= \sum_{i=2}^{n} C_{i-1}^{-a} \cdot t_i - \sum_{i=1}^{k-1} C_i^{-a} \cdot t_i \\
\text{Subject to: } S/S_0 &= 1/S_0 \sum_{i=1}^{n} t_i^{-b} \cdot C_i \leq 1 \\
t_i &> 0, \quad 2 \leq i \leq n \\
C_i &> 0, \quad 1 \leq i \leq n-1.
\end{align*}
\]

This optimization problem can be tackled using Geometric Programming as extended by Beightler [11]. The resulting non-linear programming problem has \(k-1\) degrees of freedom, where \(k\) is the number of levels making up the D-S sub-hierarchy (see Figure 1).

The procedure to be used applies to other hierarchies with topologies similar to D-S hierarchies but using different staging algorithms. The difference would be implicit in the fault-ratio function or, in the case of power function, in the exponent related to the fault-ratio function.
3. FINDING THE OPTIMUM AVERAGE ACCESS TIME

The General Case

The problem as stated now is solvable using Geometric Programming [4] with the following adjustments:

a) Since some terms are negative, i.e. violate the "posynomial" (polynomial with all positive coefficients) assumption, a more general orthogonality and normality conditions (see below) must be used in the solution (see [11] Chapter 5 for details).

b) The resulting Geometric program has k-1 "degrees of difficulty" (degrees of freedom), therefore a further optimization step with k-1 free variables is required in addition to the application of Geometric Programming (we will show an example for k=2, i.e. one degree of difficulty).

It is interesting to observe the economic interpretation of negative terms in the objective function. If this problem would have represented some kind of manufacturing or production operation, the positive terms would be costs for the several steps in the production process, whereas the negative terms represent "profits", e.g. the selling of possible by-products.

In the context of D-S hierarchies, the negative terms are the (time) "gains" obtained by having the direct connections to/from the CPU present in a D-S hierarchy. However, in order to be real "profits", these terms must outweigh the difference in fault-ratios (more on this later).

Notice that if we assume equal fault-ratios for the regular and D-S hierarchies (see [12]), their access times differ by

\[ \sum_{i=1}^{k-1} C_i^{a_k t_i} \]
the D-S time being smaller since all the variables involved (c_i-s and t_i-s) are positive. This implies that if we optimize a regular hierarchy and then add to it direct paths to the CPU making the hierarchy a D-S one, the average access time must improve.

The solution to the Geometric Programming problem given by (10)-(13) translates into solving the following system of linear equations (see [11]):

\[ d_i - e_i = b_s_i, \quad 2 \leq i \leq k-1 \]  
\[ d_i = b_s_i, \quad k \leq i \leq n \]  
\[ s_i = a d_{i+1}, \quad l \leq i \leq k-1 \]  
\[ s_i = a d_{i+1}, \quad k \leq i \leq n-1 \]  
\[ \Sigma d_i + \Sigma e_i = 1 \]  
\[ i=2 \quad i=1 \]

where the solution vector for the "dual" problem is given by:

\[ (d_2, d_3, \ldots, d_n, e_1, e_2, \ldots, e_{k-1}, s_1, s_2, \ldots, s_n) \]

and has \( n-1+k-1+n = 2n+k-2 \) non-negative elements.

Equations (14)-(17) are called the "orthogonality conditions", equation (18) is called the "generalised normality condition".

These "dual variables" are related to the terms of the primal problem (given by (10) and (11)) at optimum by:

\[ d_i = \frac{c_i-a t_1}{T - t_1}, \quad 2 \leq i \leq n \]  
\[ e_i = \frac{c_i^a t_1}{T - t_1}, \quad i = 1 \leq i \leq k-1 \]  
\[ s_i = \frac{L C_i}{S_0 t_1}, \quad 1 \leq i \leq n \]  
\[ n \leq \Sigma e_i \]  
\[ i=1 \]

with \( p = \Sigma e_i \)
The dual function (which has an optimum equal to that of the original, or primal problem) is given by:

\[ T^* - t_1 = L \prod_{i=2}^{n} \left( \frac{K_i}{d_i} \right) \prod_{i=1}^{k-1} \left( \frac{K_{n+i}}{e_i} \right) \prod_{i=1}^{n} \left( \frac{K_{n+k-1+i}}{s_i} \right) \]  

(23)

with

\[ K_i = \begin{cases} 
1 & , 2 \leq i \leq n+k-2 \\
t_1 & , i=n+k-1 \\
(t_1^{-1} S_0)^{-1} & , i=n+k \\
1/S_0 & , n+k+1 \leq i \leq 2n+k-2 \\
C_i/S_0 & , i=2n+k-1 
\end{cases} \]  

(24)

The system of equations given by (14)-(18) is made up of 2n-1 equations on 2n+k-2 variables, therefore we have k-1 degrees of freedom in the system. A possible way of finding the optimal solution in this case is given by [5] and it is based on the fact that maximizing (23) when 2n-1 variables are expressed in terms of the other k-1 ("free") variables yields the same optimal solution as the original dual function.

Since the objective function (10) contains negative terms (i.e. it is a "signomial" and not a "posynomial" - see [11]), each stationary point (i.e. where the first derivative vanishes, indicating either a minimum, maximum, or an inflection point) obtained from the dual function given in terms of the k-1 free variables (the so called "substituted dual function") must be further tested for optimality. This is due to the fact that the objective function is not necessarily convex (and the dual concave).

Checking for optimality can be done by simultaneously solving (19)-(21) for the \( C_i \)'s and \( t_1 \)'s (the "primal" variables). Since
we have $2n+k-2$ equations on $2n-2$ variables, we may obtain $k$ variables each from two different equations, supplying an easy way to check for optimality. Also, substituting the so obtained primal variables into (10) should yield the same result as the one obtained through maximization of the dual.

The above procedure will be illustrated for the $k=2$ case using some numerical examples.

Alternatively, more sophisticated techniques such as "condensation" [11] can be applied to solve the dual problem provided the particular instance of the optimization problem lends itself to such a manipulation.

If we now choose the $k-1$ variables $e_1, \ldots, e_{k-1}$ to be the free variables, (any other, more convenient choice is also possible), we can use (14)-(18) to express all other variables in terms of these. After considerable (but simple) algebraic manipulations, we obtain:

$$d_i = \begin{cases} 
(ab)^{n-1}d_n + \frac{1-ab}{(ab)^i} \sum_{j=1}^{k-1} (ab)^j e_j, & 2 \leq i \leq k-1 \\
(ab)^{n-1}d_n, & k \leq i \leq n-1 
\end{cases} \quad (25)$$

$$d_n = \frac{1-ab}{1-(ab)^{n-1}} \left[ 1 + \sum_{i=1}^{k-1} (ab)^{i-1} \right] \quad (26)$$

$$s_i = \begin{cases} 
a(d_2 - e_1), & i = 1 \\
b^{-1}(d_i - e_i), & 2 \leq i \leq k-1 \\
b^{-1}d_i, & k \leq i \leq n. 
\end{cases} \quad (27)$$

Notice that (25)-(27) reduce to the equivalent equations in [6] for the regular $(k=1)$ hierarchy case.
If we now solve (22) in terms of the \( s_i, 1 \leq i \leq k-1 \), we can express (23) in terms of those variables only. This latter step is solvable for the general case (it will be illustrated by the \( k=2 \) case later) but it does not yield results which add further insight to the nature of the solution.

Now, \( L \) (from (22)) is given by:

\[
L = \frac{1-(ab)^n}{b[1-(ab)^{n-1}]} + \frac{1-ab}{b[1-(ab)^{n-1}]} \sum_{i=1}^{k-1} e_i (ab)^{i-1} \tag{28}
\]

The next step is to use (25)-(27) to obtain (23) in terms of the \( e_i \) only. In this manner, and using (28), we get the substituted dual function which, when optimized, will yield the optimum for the original problem.

The next section illustrates this last step using a configuration with only the two fastest levels connected directly to the CPU (i.e. \( k=2 \)).

4. MINIMIZING AVERAGE ACCESS TIME \( (k=2) \)

We now turn our attention to the special case where \( k=2 \), i.e., only the two fastest levels of the D-S hierarchy have direct connections to the CPU (see Figure 2). The equations equivalent to (25)-(28) are:

\[
d_i = (ab)^{n-i} d_n, \quad 1 \leq i \leq n-1 \tag{29}
\]

\[
d_n = \frac{1-ab}{1-(ab)^{n-1}} (1+ e_1) \tag{30}
\]

\[
e_i = \begin{cases} 
0 & , i = 1 \\
\frac{a d_{i-1} - e_1}{b^{-1} d_i} & , 2 \leq i \leq n
\end{cases} \tag{31}
\]

\[
L = Q_1 \left[ 1 + \frac{1-ab}{1-(ab)^{n-1}} e_1 \right] \tag{32}
\]
where

\[ Q_1 = \frac{1-(ab)^n}{b[1-(ab)^{n-1}]} \quad (33) \]

The dual function is given now by:

\[ T^* - t_1 = \left( \frac{L}{S_0} \right) e_{1-2} - e_{1} \prod_{i=2}^{n} d_i \prod_{i=1}^{n} s_{i-1} \quad (34) \]

Using (29)-(32) and \( \Sigma d_i - e_1 = 1 \) (the normality condition), we get:

\[ T^* - t_1 = d(e_1) = t_1 \left( \frac{Q_2 e_{1-2} - e_{1}}{ab} \right) *_{C_n} \left( l + \frac{1-ab}{1-(ab)^n} e_1 \right) \]

\[ \left( l + e_1 \right) (1 + e_1) (1 + b^{-1}) \quad (35) \]

where

\[ Q_2 = \frac{1-(ab)^n}{(ab)^{n-2}(1-ab)} \quad (36) \]

\[ Q_3 = \frac{1-ab}{1-(ab)^{n-1}} \quad (37) \]

\[ Q_4 = \frac{1-ab}{b[1-(ab)^{n-1}]} \quad (38) \]

\[ Q_5 = \frac{ab}{ab-1} - \frac{(n-1)(ab)^{n-1}}{(ab)^{n-1}-1} \quad (39) \]

\[ Q_6 = \frac{1-ab}{1-(ab)^{n-1}} = Q_4 * b \quad (40) \]
All it remains to be done is to perform a one-dimensional search to locate stationary points of (35) as a function of the value of \( e_1 \). Since we are dealing with a signomial objective function, these points may result in dual variables which violate the non-negativity constraint, this in turn would imply a negative optimal solution.

A negative optimal solution implies an average access time smaller than that of the cache \((t^d_1)\), which is given usually by CPU technological constraints. Looking back at (10) for \( k=2 \), the implication is \( t_2 < t_1 \) resulting in a main memory faster than the given cache. The optimal configuration in this case has no cache.

The above case can occur when a high fault-ratio (small \( a \)) is used, and it could be easily identified since the stationary points obtained for the dual substituted function occur for values of \( e_1 \) lying outside the space defined by the non-negativity constraints and the relations (29)-(31), (in particular \( e_1 \) from (31)).

After obtaining \( e_1 \) (if at all possible) from optimizing the substituted dual function, its value (say \( e^* \)) and \( T^* - T_1 \) (which is the value assumed by the substituted dual function at \( e^* \)) can be used to determine the corresponding values for \( t_1, 2 \leq i \leq n, \) and \( C_i, 1 \leq i \leq n-1 \). The next section shows the derivation of these values.

5. DERIVING OPTIMAL CAPACITIES AND ACCESS TIME \((k=2)\)

We assume the value of \( e_1 \) has been determined as described in the last section so that the substituted dual function is maximized. This value we have called \( e^* \) and the optimal values of \( t_1, 2 \leq i \leq n, \) and \( C_i, 1 \leq i \leq n-1, \) will be derived as a function of it.

We recall that at optimality, equations (19)-(21) give the
relations between the primal and dual variables. Since 2n-1 dual
variables have been expressed as a function of the free dual variable
\# (equations (29)-(31)), we can now obtain their values and those of
the primal variables by using \( e^\#: 

The capacity of the cache, \( C_1 \) can be immediately determined from
(21), its value would be given by:

\[
C_1 = \sum_0^b \left( \frac{ab}{1-(ab)n-1} \right)^n \frac{1-(ab)^{n-2}}{1-(ab)^{n-1}} \]  

In order to obtain the values for the \( C_i, 2 \leq i \leq n-1 \), we form a
recurrence relation on the capacities only. This will yield a differ-
ence equation for which \( C_n \) and \( C_1 \) (as determined in (41) above) will
provide boundary conditions.

Forming the ratios of adjacent \( d_i \)'s and \( a_i \)'s we obtain:

\[
\frac{1}{ab} = \frac{C_{i+1}}{C_i} \left( \frac{t_i}{t_{i+1}} \right)^b, \quad 2 \leq i \leq n-1 
\]

\[
\frac{1}{ab} = \left( \frac{C_{i-1}}{C_i} \right)^a \frac{t_{i+1}}{t_i}, \quad 2 \leq i \leq n-1 
\]

By multiplying both sides of (42) by the corresponding sides of
(43) raised to the \( b \) power we get:

\[
\left( \frac{1}{ab} \right)^{1+b} = \frac{C_{i+1}}{C_i} \left( \frac{d_{i-1}}{d_i} \right)^b 
\]

This equation can be transformed into a linear recurrence relation
by taking logarithms of both sides. The resulting expression is:

\[
\ln \frac{C_i}{C_1} = \frac{1-(ab)^{i-1}}{1-(ab)^{n-1}} \left[ \ln \frac{C_n}{C_i} - Q_b(n-1) \right] + Q_b(i-1) 
\]
with
\[ Q_8 = \frac{(1+b)\ln ab}{ab-1} \]  (46)

or, solving for \( C_4 \) explicitly:
\[ C_4 = \exp \left\{ \frac{1-(ab)^{n-1}}{1-(ab)^{n-1}} \left[ \ln \frac{C_4}{C_1} - Q_8 (n-1) \right] + Q_8 (n-1) + \ln C_1 \right\} \]  (47)

where \( \exp(y) \) is the \( y \)-th power of the natural base \( "e" \).

Once the optimal capacities have been determined using (42), the optimal access times can be derived from (45), i.e.,
\[ t_4 = t_2 \left[ \frac{(ab)^{i-2} C_4^{1/b}}{C_2} \right] ^{1/b}, \quad 3 \leq i \leq n \]  (48)

with \( t_2 \) given by substituting (29), (30) and (31) into the ratio \( s_1/s_2 \) as given by (21), i.e.,
\[ t_2 = t_1 \left\{ \frac{ab C_2}{C_1} \left[ \frac{1-e^*}{(ab)^{n-2}(1-ab)(1+e^*)} \right] \right\} ^{1/b} \]  (49)

Some numerical examples of the results obtained in Sections 4 and 5 follow.

6. NUMERICAL EXAMPLES

We now consider some examples in order to compare optimal configurations for both Regular and Delayed-Staging hierarchies, under the same constraints of cost, capacity and cache access time. The examples are chosen to illustrate three possible forms for the substituted dual function.

The fault-ratio exponent \( a \) is assumed to be uniform throughout.
the hierarchy, even though in practice it is affected by factors such as page size (e.g. 0.2 for page size 64, 0.7 for size 4096) which usually varies from level to level. Also, the technology-cost exponent b is assumed constant while in reality it varies from technology to technology (e.g. 0.2 for MOS or Core, 2.0 for Disk).

A further simplification is achieved by using the same value of a for both Regular and Delayed-Staging hierarchies. This is backed-up by empirical evidence for small values of the staging delay D (see [12]).

As we have already mentioned, the assumptions on a and b are not needed for solving the optimization problem, they just make the solution easier. The same method described here can be used to numerically solve any specific configuration using different values for a and b at each storage level.

For all the examples, the following parameters are kept constant:

* Maximal Cost:
  \[ S_0 = 1000000 \] (same units as the technology-cost function).

* Backing store capacity:
  \[ C_n = 16M \] (same units as the fault-ratio function).

* Cache access time:
  \[ T_1 = 1 \] (the time unit).

The optimal access times, capacities and average hierarchy access time are calculated using (I) Chow's [6] results for Regular hierarchies, and (II) the results obtained in this paper for Delayed-Staging hierarchies.

Example 1:

\[ a = 0.3, \quad b = 0.5, \quad n = 4. \]

This case results in a substituted dual function of the form (not to scale) depicted in Figure 3. Obviously, the only stationary point
lies outside the non-negative constraints for $e_1$, thus yielding a negative objective function value for $T^* - t_1$.

As we saw in past sections, this result violates our assumption of $t_1 < t_2$ and therefore we cannot accept the optimal solution obtained by our method.

The results for a regular hierarchy are:

<table>
<thead>
<tr>
<th>Level(i)</th>
<th>$t_i$</th>
<th>$c_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2870</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>25372</td>
</tr>
<tr>
<td>3</td>
<td>23</td>
<td>605595</td>
</tr>
<tr>
<td>4</td>
<td>389</td>
<td>16777216</td>
</tr>
</tbody>
</table>

These values yield an average access time of 9.41 time units. If we use this same hierarchy configuration, but now calculate the average access time using (10) for the D-S case (i.e. we add the path connecting level 2 and the CPU), the result is 9.36 time units. The cache fault ratio is $2870^{0.3} = 9\%$.

If we now examine the boundaries for $e_1$, we see that point P in Figure 3 yields an average access time (and associated hierarchy) identical to that of the regular case.

As we move inside the boundaries towards $Q$, both the capacities and access times decrease, until $t_2$ reaches the value of $t_1 (1.0)$ at $Q$ (notice that we cannot use equation (21) here because this boundary implies $s_1 = 0$).

The average access time at point $Q$ is 9.288 time units, the resulting D-S hierarchy configuration (with two "caches") is:
with fault-ratios of 16% and 5% for levels 1 and 2 respectively.

Example 2:

\[ a = 0.75, \quad b = 0.5, \quad n = 4. \]

The substituted dual function in terms of \( e_i \) is of the form shown in Figure 4. By further analysis of the two stationary points \( P \) and \( Q \) (as described before), \( P \) is selected as the optimum one. The resulting configuration is:

<table>
<thead>
<tr>
<th>Level(i)</th>
<th>( t_i )</th>
<th>( C_i )</th>
<th>( t_i )</th>
<th>( C_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>430</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( \sim 1 )</td>
<td>19189</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>583931</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>387</td>
<td>16777216</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The average access times for the (optimal) Regular and D-S hierarchy configurations are 1.0220 and 1.0216 time units respectively.

Example 3:

\[ a = 0.5, \quad b = 0.7, \quad n = 3. \]

Here, the substituted dual function presents a minimum within the boundaries forced on \( e_i \) by the non-negativity constraints on the dual variables (see Figure 5). By examining this point we determine it to represent the optimal solution. The resulting Regular and D-S configurations are:
If we now observe the results of this section, it is evident that they do not indicate a dramatic advantage for D-S over Regular hierarchies (remember that these results are only for \( k=2 \)). Although the D-S case is better in all instances, it fails to be so by a large margin (all values are the same order of magnitude).

Therefore, if we take into account the additional cost of the path between the CPU and the second fastest storage level, a \( k=2 \) D-S hierarchy may not be justifiable in terms of cost/performance. Nevertheless, if the above path and its supporting hardware/firmware are already built into the memory unit, as it might be the case in an Active Memory Unit (memory units with sophisticated processing capabilities - see [12,13]), then the usage of D-S is clearly advantageous.

Regarding Example 1, we notice that the resulting D-S configuration has a much smaller cache than the Regular setup, but compensates for it by having a faster second and third levels. Also, even though our solution method did not yield an optimal result, using the configuration obtained in the Regular case showed an improvement in average access time for the D-S case, and so did the solution obtained at the boundary imposed by the non-negativity constraints (refer to the analysis of Example 1).
7. SUMMARY

This paper has developed an analytical tool to evaluate Delayed-Staging hierarchies. Although the formulae and derivations get quite involved in the general case, a tool such as a computerized algebraic manipulation system (e.g., REDUCE [14, 15]) could be used to derive the necessary formulae, thus yielding a tractable problem.

The power function assumption results in an oversimplification of the real problem, although the usage of different exponents to represent the physical characteristics of the technologies involved could very well make the model very close to reality (again, this would complicate the formulae involved, requiring the use of an algebraic manipulation system).

Notice that the cost of having the additional paths to/from the CPU in the D-S case is not taken into account in the optimization problem. This factor can be introduced in the cost constraint as additional terms depending only on the access times of those levels (1 through k) directly connected to the CPU.

This aspect of the cost can be incorporated into the solution as presented here by using a power function on the access time for the cost of the direct paths (we have no evidence to support such an assumption). The resulting Geometric program will have then 2*(k-1) degrees of freedom.

Again we must view the results obtained in light of the assumptions made to simplify the problem. Some of these assumptions can be trivially removed without affecting the method of solution (they make it a little more algebraically complex), e.g., various $a_i$'s and $b_j$'s instead of single values. Other assumptions would void the applicability of
Geometric Programming, as would be the case of non-power functions for the fault-ratio and/or technology-cost function, and therefore require a different approach altogether.

It is worthwhile recapining that the examples in the "2-out-of-n" D-S case showed only a marginal improvement (not exceeding 8%) over the equivalent linear configuration. This would not justify by itself the added costs in a D-S configuration.

Therefore, it is necessary to consider the D-S approach only within an environment which lends itself easily to support the functions involved in the D-S implementation. Such an environment is supplied under the Active Memory Unit approach mentioned in Section 6.

An Active Memory Unit [12] is a computer module which provides, independent from the CPU, a one-level memory space [16], subsuming in it all file I/O operations. This is done by providing very large (in the order of millions of Mega-Bytes) addressing space coupled with processing capabilities which enable the memory unit to absorb all memory management functions, including virtual memory mapping and control, file management, etc.

This new concept, and the more general k-out-of-n D-S case within it, together with its potential to supply the tools to more efficient large computer systems (as noted by Stein in [17]) warrant further research in the area.
REFERENCES

REFERENCES (cont'd)


Figure 1 - A "k-out-of-n" Delayed-Staging Storage Hierarchy
Figure 2 - A "2-out-of-n" Delayed-Staging Storage Hierarchy
Figure 3 - Substituted Dual Function vs. Free Variable $e_1$ (Example I)
Figure 4 - Substituted Dual Function vs. Free Variable $e_1$ (Example 2).
Figure 5 - Substituted Dual Function vs. Free Variable $e_1$ (Example 3).