A PROCEDURE OF EFFICIENTLY OBTAIN FAULT-RATIOS
IN DELAYED-STAGING STORAGE HIERARCHIES

by

Gabriel M. Silberman

Technical Report No. 191

January 1981
A PROCEDURE TO EFFICIENTLY OBTAIN FAULT-RATIOS IN DELAYED-STAGING STORAGE HIERARCHIES

GABRIEL M. SILBERMAN

Technion, Israel Institute of Technology, Haifa, Israel

ABSTRACT: A procedure to efficiently obtain fault-ratios in n-level Delayed-Staging (D-S) storage hierarchies, without recurring to simulation runs for each configuration desired, is presented.

The procedure is based on the applicability of Stack Processing techniques to D-S configurations. These hierarchies (D-S) are a special case of non-linear storage hierarchies, of which linear configurations represent a particular instance.

Keywords: Storage Hierarchies, Stack Algorithms, Fault-Ratios.

1. INTRODUCTION

A procedure to obtain Fault-Ratios for every level in an n-level, demand-driven, Delayed-Staging (D-S) storage hierarchy, given as data the fault-ratios for suitable two-level D-S configurations, is presented. This allows the efficient evaluation of the D-S type of storage hierarchy, without resorting to explicit simulation for each configuration being considered.

Delayed-Staging storage hierarchies 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 Figure 1). This generalization of linear hierarchies [1,2,3] enables some parallelism in the flow of data through the hierarchy.
Stack Processing techniques have been shown in [4] to apply to D-S hierarchies. This provides the basis for extending Slutz and Traiger's [5] results for linear hierarchies, to the (more general) D-S case.

Section 2 reviews the D-S concept and presents the relevant configuration parameters. The procedure to calculate the fault-ratios is presented in Section 3, and a numerical example follows in Section 4.

Comments on the accuracy limitations of the procedure and the range of values for some configuration parameters are given in Section 5. Section 6 points out some possibilities for further research in the area of D-S storage hierarchies.

2. DELAYED-STAGING STORAGE HIERARCHIES

The existence of parallel paths for data migration within a D-S storage hierarchy present the potential for more sophisticated managing schemes than are possible with linear hierarchies. Nevertheless, we shall examine a management scheme based on demand, both because of its simplicity and its similarity to equivalent schemes in linear hierarchies.

D-S hierarchies managed by demand allow a hit within the $k$ fastest levels to result in a direct access by the CPU and, in addition, the scheduling (by the hierarchy management facility) of the page containing the information being sought, for staging (hence the name, Delayed-Staging) towards the fastest level (if not already there).

A hit below level $k$ (refer to Figure 1) will cause staging in the usual manner, and by the time the requested information arrives at level $k$, a decision can be made whether to use the "short-cut" to the CPU.
Notice that the page size may very well differ between different levels, reflecting hardware transfer efficiency as well as program and/or data locality sizes.

Consider now, from the CPU's standpoint, the access speed of a D-S hierarchy as opposed to the regular (linear) case. If the hit occurs within the k fastest levels of a D-S hierarchy, the access time is that of the level of the hit alone, whereas in the linear case it is made up of the access times for levels 1 through the level of the hit.

The "penalty" (to performance) of D-S is the absence from the fastest level, at least for a while, of a page hit at one of the k levels. This results in slower subsequent accesses to the same page than would have been possible with linear staging, until the delayed staging is completed.

D-S needs processing capabilities and hardware support to implement the required algorithms, queues and buffers. The costs of these factors might very well outweigh any benefit resulting from the use of D-S, if the same capabilities are not used for, or are a part of, other improvements which might then justify their presence (more on this in Section 6).

The configuration parameters for D-S hierarchies are:

- $B_i$ - page size for information movement between levels $i+1$ and $i$,
- $C_i$ - capacity of level $i$ ($C_i/B_i$ = number of pages in level $i$; $1 < i < n-1$; $C_n/B_{n-1}$ = number of pages in the "backing store")
- $D_i$ - staging delay to move a page (of size $B_i$) from level $i+1$ to level $i$, $1 < i < k-1$ (notice that $D_1 = 0$, $k < i < n-1$).

The units for the staging delay $D_i$ are the average number of CPU references generated in the interval it takes to effect the staging of
a page from level \( i+1 \) to level \( i \), i.e., virtual time units measured in memory references by the CPU.*

Clearly, there is no staging delay between the CPU and the fastest level since no virtual (CPU) time elapses from request to access. Therefore, the linear case is obtained if we set \( k=1 \) or, equivalently, \( D_i = 0 \) for all \( 1 \leq i \leq k-1 \).

3. THE PROCEDURE

The general approach is to evaluate, from the slowest level to the fastest level, the fault ratio of level \( i \) by assuming it to be the fastest level, and merging all slower levels \( j \) \( (i < j \leq n) \) (refer to Fig. 1) into one, with physical parameters which reflect the various levels merged.

From the resulting procedure, an efficient way to obtain fault-ratios in \( n \)-Level D-S storage hierarchies with varying parameters (page sizes, level capacities and staging delays) is obtained. These fault-ratios can then be plugged into models for performance evaluation to obtain results which will aid with hierarchy design and connection topography decisions.

We define \( F(B,C,D) \), the fault-ratio of a cache with page size \( B \), capacity \( C \) and transfer delay \( D \), as the ratio of the number of misses at the cache to the total number of references.

To clarify the concept of \( D \), consider the following example:

- Transfer of a "word" (width is CPU dependent)
  - a) from cache to CPU = 1 time unit
  - b) from main memory to CPU = 11 time units.

* For example, if a transfer initiated by reference \( x_t \) would be completed by the time reference \( x_{t+2} \) is generated, then \( D=2 \) (\( x_t \) is the page referenced at time \( t \)).
Transfer of a page (of size B) from main memory to cache = 40 time units

- The fault-ratio \( F(B, C, D) = 10\% \).

The resulting average reference time, if we assume no delay at the CPU, is \( 1 \times 0.9 + 11 \times 0.1 = 2.0 \) units of time (or .5 references per unit of time); \( \bar{D} \) is then obtained by dividing the actual page transfer time by the average units of time it takes to generate a single CPU reference, i.e., \( D = 40/2.0 = 20 \) references (average).

Proposition: The fault-ratio * for each level \( i, \bar{F}_i \), in an \( n \)-level D-S storage hierarchy, with the \( k > 1 \) fastest levels having direct paths to the CPU, are given by:

\[
\bar{F}_i = \begin{cases} 
0 & \text{if } i = n; \\
F(B_i, C_i, 0) & \text{if } k \leq i < n-1; \\
F(B_i, C_i, D_i) & \text{if } i = k-1; \\
F(B_i, C_i, \bar{D}_i) & \text{if } 1 \leq i < k-2.
\end{cases}
\]

(1a) 

where

\[
\bar{D}_i = D_i + \frac{1}{\bar{F}_i} \sum_{j=i+1}^{k-1} \bar{F}_j.
\]

(2)

Notice that the fault-ratios for the slower \( n-k \) levels (those not directly connected to the CPU) are identical to those for equivalent levels in a regular aging hierarchy. Furthermore, if we set \( k = 1 \) (equations (1c-d) drop) we obtain an algorithm which is identical to that presented by Slutz and Traiger (making the substitution

\[
\bar{F}_i = \sum_{j=i+1}^{n} \bar{H}_j, 1 \leq i \leq n-1;
\]

* The ratios are to the total number of references.
Since equations (1a) and (1b) are equivalent to those used by Slutsky and Traiger, the proof of their validity will not be repeated here. Rather, we shall concentrate on equations (1c) and (1d).

The applicability of Slutsky and Traiger's results to stack algorithms is based on two features of these algorithms, the so-called "Two-level" and "Nesting" properties.

"Two-level" refers to the fact that each level behaves, from the replacement algorithm standpoint, as if it were the top level in a two-level setup. Therefore, each level can be managed independently of the others (although using the same replacement algorithm).

"Nesting" means that whenever a page \( x \) is present at storage level \( i \), its parent page (the one which contains \( x \)) is present at all levels \( i+1, i+2, \ldots, n-1, n \).

Equation (1c) involves the next-to-fast level \( k-1 \) which is connected directly to the CPU. Since we can treat all levels \( k \) through \( n \) as a single one having a staging delay of \( D_{k-1} \) (all levels \( k \leq i \leq n-1 \) have \( D_i = \emptyset \)), it is clear that the two-level property holds for level \( k-1 \). As far as the nesting property is concerned, its applicability in this case follows from the two-level property quite trivially.

We now present the algorithm to calculate the fault ratios \( F_i \), \( 1 \leq i \leq k-1 \). The validity of (1d) will be tackled after we gain some insight into what \( D_i \) means in (2).

Algorithm 1:

[1] \( F_{k-1} = F(B_{k-1}, C_{k-1}, D_{k-1}) \)

[2] for \( i = k-2 \) down to 1 do
solves
\[
\begin{cases}
\bar{P}_i - P(B_i, C_i, D_i) \\
\sum_{j=1}^{k-1} \bar{P}_j \cdot D_j \\
\bar{P}_i = P_i + \frac{i+1}{\bar{P}_i}
\end{cases}
\]

For each iteration (value for i) of Step [2] above, we have to find \( \bar{D}_i \) such that
\[
P(B_i, C_i, \bar{D}_i) = \bar{P}_i - \frac{i+1}{(\bar{D}_i - D_i)}
\tag{3}
\]
(notice that \( \bar{D}_i > D_i \), \( 1 \leq i \leq k-2 \), since \( \bar{D}_i \) is monotonically increasing as \( i \) decreases).

Equation (2) simply expresses the average staging delay \( \bar{D}_i \) for level \( i \). This average depends only on levels \( j > i \), its value averaging their physical (hardware) staging delays \( D_j \), and the averaging weights being their fault ratios \( \bar{P}_j \). Therefore, equation (1d) simply says that we "merge" all levels \( j > i \) into a single, equivalent level having capacity \( C_n \) (backing store) and (average) staging delay \( \bar{D}_i \).

The right hand side of equation (3) is clearly inversely proportional to \( \bar{D}_i \). Its left hand side must be determined empirically; in the case of stack algorithms, this can be done quite efficiently using the results described in [4].

Also, from measurements carried out using several values of \( B_i \), \( C_i \) and \( D_i \) (see Section 5 for range limitations), it can be concluded that \( P(B_i, C_i, D_i) \) increases as \( D_i \) increases for fixed values of \( B_i \) and \( C_i \) (see Figure 2).
Because of the above observations, equation (3) is solvable.

A possible procedure to obtain the solution (i.e., determine \( \tilde{D}_i \) and \( \tilde{F}_i \)) is to iteratively guess a value for \( \tilde{D}_i \) and insert it in (3); three cases are then possible:

1. left-hand of (3) = righthand of (3); in which case we have obtained the correct \( \tilde{F}_i \) and \( \tilde{D}_i \);

2. left-hand of (3) > righthand of (3); use a smaller value of \( \tilde{D}_i \) as next guess **;

3. left-hand of (3) < righthand of (3); use a larger value of \( \tilde{D}_i \) as next guess **.

In the process of determining \( \tilde{F}_i \), there will be several values of \( \tilde{D}_i \) for which \( F(B_1, C_i, D_i) \) has to be determined (empirically). In order to cut down on the number of measurements needed, \( F(B_1, C_i, D_i) \) is obtained for integer values of \( D_i \), and then some interpolation technique may be used to determine \( F(B_1, C_i, D_i) \) for non-integer \( D_i \) (see example below).

After the \( \tilde{F}_i \), \( 1 \leq i \leq n-1 \), have been determined, the hit ratios \( \tilde{H}_i \) can be derived simply by:

\[
\tilde{H}_i = \begin{cases} 
1 - \tilde{F}_i, & \text{if } i = 1; \\
\frac{\tilde{F}_{i-1} - \tilde{F}_i}{\tilde{F}_{i-1}}, & \text{if } 2 \leq i \leq n-1; \\
\tilde{F}_{n-1}, & \text{if } i = n.
\end{cases}
\]  

(4)

* Equality here means to desired accuracy.

** How larger or smaller the next value of \( \tilde{D}_i \) is can be determined in a manner similar to the bisection method of numerical approximation.
This completes the description of a procedure to efficiently derive the hit (and/or fault) ratios for all levels in a D-S hierarchy, given the hit (or fault) ratio for two-level D-S hierarchies with suitable parameters (B, C and D). This procedure is based upon the usage of stack algorithms as extended to D-S hierarchies in the way described in the last section.

A numerical example of the procedure follows in the next section.

4. A NUMERICAL EXAMPLE

We now show a numerical example to illustrate the use of the procedure described in the last section. The storage hierarchy to be used for the example (D₄ - 8 do not necessarily reflect physical properties) consists of five levels (n=5), four of which have direct connections to/from the CPU (k=4), leaving only the last level (backing store) connected solely to the level next to it. The physical parameters of the hierarchy are given in Table 1.

We now use equations (1) and Algorithm 1 to calculate the fault ratios $\bar{P}_i$, $1 \leq i \leq n$. Remember $n=5$, $k=4$.

a) $i=5$; this case is handled by equation (1a), i.e. $\bar{P}_5 = 0$

b) $i=4$; $i=k$ and (1b) applies here, i.e. $\bar{P}_4 = P(16K, 4M, 0) = 0.000059$.

Notice that all values of $P(B, C, D)$ are empirically determined through simulation runs in two-level hierarchies with parameters B, C, D (see the Appendix for address reference string description).

c) $1 \leq i \leq 3$; here we apply Algorithm 1 which makes use of (1c-d) and (2) as described before.
[1] \[ F_3 = f(8K, 1M, 75) = 0.00042 \]

[2] We repeat this step for \( i = 2 \) and then \( i = 1 \).

\( i = 2 \):

Assume \( D_2 = 68 \)

First iteration:

\[ F_2 = f(2K, 256K, 68) = 0.0013 \]

\[ D_2 = 74 \]

Second iteration:

\[ F_2 = f(2K, 256K, 74) = 0.00138 \]

\[ D_2 = 73 \]

Third iteration:

\[ F_2 = f(2K, 256K, 73) = 0.00137 \]

\[ D_2 = 73 \]

After this iteration \( D_2 \) remains the same (after rounding), therefore, we stop with the value of \( F_2 \) obtained.

\( i = 1 \):

Assume \( D_1 = 3 \)

First iteration:

\[ F_1 = f(64, 2K, 3) = 0.092 \]

\[ D_1 = 2 \]

Second iteration:

\[ F_1 = f(64, 2K, 2) = 0.07 \]

\[ D_1 = 2 \]

Here we stop after two iterations.

In order to check the accuracy of the values obtained by using Algorithm 1, we run a simulation on the complete hierarchy. These results are compared in Table 2.
Notice that the hit ratios are relative to the total number of references generated by the CPU.

The hit ratios of each level, relative now to the number of references directed to that level are obtainable by:

$$
\bar{H}_i = \begin{cases} 
H_i, & 1 \leq i < n-1; \\
\frac{H_i}{H_{i-1}}, & 2 \leq i \leq n-1; \\
1, & i = n.
\end{cases}
$$

Table 3 compares these "relative" hit-ratios obtained by direct simulation and by using the procedure described in last section.

It is worth recalling that the advantage of using Algorithm 1 depends on the efficiency with which \( P(B,C,D) \), the fault ratio in a two-level setup, can be calculated (empirically) for various values of \( B, C \) and \( D \). However, it has already been shown that \( P(B,C,D) \) can be efficiently determined for several values of \( C \) on one pass through an address reference string, if the underlying replacement algorithm is a stack algorithm (see [4]).

The accuracy of the results is typical of the several examples that were run to check the algorithm.

5. ACCURACY LIMITATIONS AND PARAMETER RANGES

The procedure described in Section 3 for fault-ratio computation in a D-S hierarchy is limited in its accuracy, for levels 1 through \( k-1 \) (refer to Figure 1). The main reasons for this "inaccuracy" are summarized below.

**Two-Level Property:** It does not hold generally, because \( D_1 \) is only an average based on fault ratios to slower levels (refer to equation (2)).
Nevertheless, on a long reference string (in the order of $10^{15}$ references) it will tend to be quite close (see example in last section).

**Nesting Property:** If special care is not taken in the implementation of the replacement algorithm at each level, this property may be violated.

For example, consider a hit at level $i+1$ ($1 < i < k$) at time $t$; this causes a page (say $x$) to start migrating towards level $i$. At time $t+D_i$, $x$ is already at level $i$, but if $D_{i-1} \neq \emptyset$ it is still not present at level $i-1$. This means a reference to page $y$, the page to be replaced by $x$ at level $i-1$ will result in a hit at level $i-1$, but will cause a fault at level $i$.

To avoid this, an algorithm such as Gegei's Least Recently Freed [3], or LRF, can be used. The idea behind this algorithm is to allow a page at level $i$ ($2 \leq i \leq n-1$) to be replaced only if none of its descendants are present at any level $k$ ($1 < k < i-1$). Furthermore, it can be shown that LRF is equivalent to LRU (see [3]).

As far as the valid ranges for physical parameter values, i.e. $B_i$, $C_i$, $D_i$ and $k$, the procedure described in Section 3 applies to all values. Nevertheless, technological limitations such as transfer efficiency, must be taken into account when assigning values to these parameters.

Since $B_i$ (page size) and $C_i$ (level capacity) are parameters already present in linear hierarchies (and their ranges are well defined by technology), we direct our attention to the ranges for $D_i$ (staging delay) and $k$ (number of levels with direct paths to the CPU).

Common sense and present technology combine to limit these two design parameters. On one hand, a $D_i$ which is comparable in number of page references to the size (in pages) of the faster of two storage
levels will nullify any advantage of Delayed-Staging, since the page
being staged stands a good chance of being discarded immediately upon
arrival to the faster storage level.

On the other hand, k (the number of levels with direct paths to/
from the CPU) is limited to include only those storage technologies
for which single word (or information unit) access is reasonably
efficient, therefore justifying the "slowing-down" of the CPU to adjust
to the level from which the information comes (on a READ), or the
usage of a small buffer (on a WRITE). For example, a magnetic bubble
device may be considered within the first k levels, whereas a disk,
tape or mass storage device may not be so.

6. SUMMARY

We have shown a procedure to obtain fault-ratios in n-level D-S
storage hierarchies, parallel to, and actually a generalization of,

This procedure requires only the knowledge of fault-ratios for
suitable two-level D-S hierarchies, in order to produce the data for
the more general, n-level case. Stack replacement algorithms as
extended to D-S hierarchies (see [4]) are assumed throughout.

A step-by-step application of the procedure yielded results which,
when compared with those obtained by direct simulation, showed a
remarkable accuracy.

As directions for further research in the area of D-S hierarchies,
we point to the critical area of the architectural environment. This
aspect is of utmost importance, since D-S hierarchies by themselves
seem to imply more costs (e.g. in hardware) than possible advantages,
if the required capabilities are not shared with other facilities in the computer system (see [6] for details).

In the context of adequate environments for D-S hierarchies, the Active Memory Unit concept (units with sophisticated processing capabilities) should play a major role (see [6, 7]).

Furthermore, hierarchy management techniques other than demand (e.g. prefetching) may also present added advantages for the D-S approach. Also in this context, task-switching policies should be adapted to optimize multi-programming in a D-S environment.

7. ACKNOWLEDGEMENTS

I would like to thank Prof. Gideon Frieder of SUNY/Buffalo for his valuable advice during the course of this research.

8. REFERENCES


APPENDIX - ADDRESS REFERENCE STRING USED IN SIMULATIONS

The address reference string used in the simulation runs was obtained as follows:

1. A Nanodata MM-I running an IBM 360 emulator produced a complete instruction trace, containing for each instruction its operation code, operand(s) length (if variable), and the operand(s) effective addresses. Also, non-sequential changes to the program counter (i.e. branch instructions) are recorded as "operands" to an 'opcode X'00' (non-existent) instruction.

2. The instruction trace was then used to generate an address trace which reflected all memory accesses, both for instructions and operands. This involved processing instructions by groups, depending on whether their operands were being used in memory accesses (e.g. ST-store-instruction) or as immediate operands (e.g. LA-load address-instruction), and also taking into account multiple accesses resulting from a variable length operand (e.g. MVC-move character-instruction).

Each address reference consisted of 32 bits, 24 used for the memory address itself and 8 containing flags describing the instruction which generated the address and the nature of the memory access. This code is described in Figure 3.

The instruction trace contained both user and supervisor code. It consisted of over 3.5 million instructions which generated over 15 million address references. The program being traced was a COBOL execution which processed data residing both in memory and on disk.

The resulting address reference string was partitioned into segments which were then used to drive several simulation runs, some of them with the same parameters, to check the effect of different trace segments on the results obtained.
**Figure 1** - A "k-out-of-n" Delayed-Staging Storage Hierarchy
Figure 2 - Fault-Ratio $F(B,C,D)$ vs. Staging Delay $D$ in a Two-Level Hierarchy
Figure 3 - Address Reference Code

<table>
<thead>
<tr>
<th>Address Reference Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000-0001</td>
<td>Reserved</td>
</tr>
<tr>
<td>0000-0010</td>
<td>Reserved</td>
</tr>
<tr>
<td>0010-1010</td>
<td>IM/TOX XXXX</td>
</tr>
<tr>
<td>0100-1000</td>
<td>SW/XXX</td>
</tr>
<tr>
<td>0001-0010</td>
<td>BR/XX</td>
</tr>
<tr>
<td>0011-0010</td>
<td>BALA R1, R2 WHERE R2, R0</td>
</tr>
<tr>
<td>0011-0010</td>
<td>BALA R1, R2 WHERE R2, R0</td>
</tr>
<tr>
<td>0101-0100</td>
<td>CONDITIONAL BRANCH (R1 or R2, R0, R0, R0, R0, R0)</td>
</tr>
<tr>
<td>0101-0100</td>
<td>CONDITIONAL BRANCH (R1 or R2, R0, R0, R0, R0, R0)</td>
</tr>
<tr>
<td>1110-1100</td>
<td>Multiply Instruction</td>
</tr>
<tr>
<td>1110-1100</td>
<td>Multiply Instruction</td>
</tr>
<tr>
<td>1110-1100</td>
<td>Write to Address, Multiple Times</td>
</tr>
<tr>
<td>1110-1100</td>
<td>Write to Address, Single Time</td>
</tr>
<tr>
<td>1111-1100</td>
<td>Read from Address, Multiple Times</td>
</tr>
<tr>
<td>1111-1100</td>
<td>Read from Address, Single Time</td>
</tr>
<tr>
<td>1111-1100</td>
<td>Instruction Prefix After Branch</td>
</tr>
<tr>
<td>1111-1100</td>
<td>Instruction Prefix After Branch</td>
</tr>
<tr>
<td>1111-1100</td>
<td>Instruction Prefix After Branch</td>
</tr>
<tr>
<td>1111-1100</td>
<td>Instruction Prefix After Branch</td>
</tr>
<tr>
<td>1111-1100</td>
<td>Instruction Prefix After Branch</td>
</tr>
<tr>
<td>1111-1100</td>
<td>Instruction Prefix After Branch</td>
</tr>
<tr>
<td>Level (i)</td>
<td>Page Size (B_i)</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>2</td>
<td>2048 (2K)</td>
</tr>
<tr>
<td>3</td>
<td>8192 (8K)</td>
</tr>
<tr>
<td>4</td>
<td>16384 (16K)</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1 - Physical Hierarchy Parameters**

<table>
<thead>
<tr>
<th>Level (i)</th>
<th>( \bar{F}_1 )</th>
<th>( H_1 )</th>
<th>( \bar{F}_1 )</th>
<th>( H_1 )</th>
<th>( \bar{D}_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.06</td>
<td>0.94</td>
<td>0.07</td>
<td>0.93</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0.0013</td>
<td>0.059</td>
<td>0.00137</td>
<td>0.069</td>
<td>73</td>
</tr>
<tr>
<td>3</td>
<td>0.00042</td>
<td>0.90088</td>
<td>0.00042</td>
<td>0.00095</td>
<td>75</td>
</tr>
<tr>
<td>4</td>
<td>0.000059</td>
<td>0.00036</td>
<td>0.000059</td>
<td>0.00036</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.000059</td>
<td>0</td>
<td>0.000059</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 2 - Comparison of Empirical and Analytical Results.** \( H_1 \) - obtained from \( \bar{F}_1 \) 's using (4)

<table>
<thead>
<tr>
<th>Level (i)</th>
<th>( RH_1 )</th>
<th>( RH_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.94</td>
<td>0.93</td>
</tr>
<tr>
<td>2</td>
<td>0.98</td>
<td>0.99</td>
</tr>
<tr>
<td>3</td>
<td>0.68</td>
<td>0.69</td>
</tr>
<tr>
<td>4</td>
<td>0.86</td>
<td>0.86</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 3 - "Relative" Hit-Ratios - Comparison of Empirical and Analytical Results.**