THE COMPETITIVENESS OF ON-LINE DEMAND PAGING
ALGORITHMS FOR GENERALIZED TWO LEVEL MEMORIES
(Preliminary Version)

by

G. Matzliach and O. Shmueli

Technical Report #619
March 1990
This research was supported by the Fund for the Promotion of Research at the Technion.
paging algorithm works as follows (once the primary memory is full); when a page $p_i$ needs to be fetched, the primary memory is searched. If $p_i$ is in primary memory then no paging is performed. If $p_i$ is not in primary memory (i.e. a page-fault occurs), then $p_i$ is paged-in (namely, brought from the secondary memory to the primary memory), while a page which is in primary memory (and which is chosen by the demand paging algorithm) is paged-out (to the secondary memory) to free a page frame for $p_i$. No additional paging may be performed.

Given a sequence of fetches, general paging algorithms and demand paging algorithms may be either on-line or off-line. An on-line algorithm is an algorithm that bases its decisions only on the past, as if the fetch requests arrive and are being processed one at a time, without knowing what are the future requests. (So, if a page-fault occurs due to the fetch of $e_i$, an on-line paging algorithm bases its decision concerning this page-fault only on fetches that have been already performed, i.e. $e_1 \ldots e_{i-1}$). An off-line algorithm is an algorithm that has the whole sequence of fetches as input, and thus may use this information to obtain better performance. Needless to say that in practice, most paging algorithms are on-line.

For a given sequence of fetches, the cost of an on-line/off-line demand/general paging algorithm for a two level memory is the number of accesses made to the secondary memory while performing the sequence of fetches. It is known [ST] that premature paging can not improve performance (i.e. can not reduce the total number of accesses to secondary memory) when two level memories are considered. Thus, with respect to two level memories, it is sufficient to deal with demand paging algorithms.

It is well known that the optimal off-line paging algorithm is [B] [MS]:

*When a page-fault occurs, among all pages in the first level of the memory, page-out the page whose next fetch is the latest in the sequence.*

The above optimal off-line paging algorithm is called MIN. Some known on-line demand paging algorithms [PS] are LRU ("when a page-fault occurs, page-out the least recently used page in primary memory"), LFU ("when a page-fault occurs, page-out the page in primary memory that has been accessed with the least frequency"), FIFO ("when a page-fault occurs, page-out the page in primary memory which was the first to be paged in"), and LIFO ("when a page-fault occurs, page-out the page in primary memory which was the last to be paged in"). Note that in practice, LRU is the most commonly used algorithm.

A common method for evaluating the performance of an on-line algorithm is using the performance of the optimal off-line algorithm as a yardstick [BLS] [MMS] [RS] (and many others). An on-line demand paging algorithm $A$ is said to be $c$-competitive if there exists a constant $a$ so that for all sequences of fetches $\text{COST}(A) \leq a + c \text{COST(OPT)}$, when OPT is the optimal off-line demand paging algorithm. $c$ is called the competitive factor. Algorithm $A$ is said to be strongly $c$-competitive [MMS] [MS] if in addition, there is no on-line demand paging algorithm that has a competitive factor which is smaller than $c$.

The competitiveness of on-line paging algorithms for two level memories was investigated by Sleator and Tarjan [ST]. They proved that $M$, the size of primary memory (in terms of number of page frames), is a lower bound for the performance of any on-line al-
algorithm, in comparison with the optimal off-line algorithm MIN. In other words, for each on-line paging algorithm, one can construct a sequence of fetches, so that when the on-line algorithm is used, the number of page-faults that occur is $M$ times the respective number for MIN. Moreover, [ST] proved that this is also the upper bound for LRU and FIFO (i.e. FIFO and LRU have at most $M$ times more page-faults than MIN, for any sequence of fetches), while both LIFO and LFU do not have an upper bound, as we can make the ratio between their costs and the cost of MIN as large as desired. Thus [ST] proved that LRU and FIFO are strongly $M$-competitive algorithms for the two level paging problem, while LFU and LIFO are not competitive at all.

A task system for processing sequences of tasks [BLS] consists of a set of possible system states, a set of possible tasks, and two cost matrices; the state transition cost matrix which indicates the cost of changing the current system state (this matrix must obey the triangle inequality), and the task processing matrix which indicates the cost of processing each task in each state (as the processing cost of a task depends on the system state). There is more than one way of executing a sequence of tasks because after each execution of a task one may change the current system state. The cost of a specific execution of a sequence of tasks is the sum of the costs of all state transitions performed during execution, and the total cost of tasks processing. An instance of a task system is the $k$-server problem [MMS] which consists of a weighted directed graph (the weights obey the triangle inequality) and $k$ mobile servers. (The problem is termed symmetric if the graph is undirected). A task here specifies a vertex to be covered by a server. Thus, the cost of a specific execution of a sequence of tasks in the $k$-server problem is the cost induced by moving the servers among vertices.

The general paging problem of two level memories is an instance of a task system whose set of states is all the possible partitions of the set of pages between the primary memory and the secondary memory, each task specifies a page to be fetched, and the cost of a state transition from state $S_1$ to another state $S_2$ is the number of pages which are stored in $S_2$ in the primary memory, but stored in $S_1$ in the secondary memory (namely, the number of accesses to the secondary memory which are required to change the state). Moreover, this problem is a special case of the $M$-server problem [MMS] in which all weights (also called distances) are identical. [MMS] conjectured, that for any symmetric $k$-server problem (for all $k > 1$) there exists a strongly $k$-competitive algorithm. So far, their conjecture was shown to hold for resistive graphs [CDRS]. (Resistive graphs are weighted graphs that can be viewed as electrical networks. Examples to such graphs are trees, and all graphs having the same weight on all edges). Note that there exists an on-line algorithm for the $k$-server problem whose competitive factor is independent of the distances between nodes [BLS]. Thus, any on-line algorithm that has a lower bound on its competitive factor, which depends on the distances, is not strongly competitive.

If we represent the demand paging problem of two level memories as a task system (similarly as done for the general paging problem), then most of the state transitions are forbidden; a transition from state $S_1$ to state $S_2$ is allowed to a demand paging algorithm, with respect to a sequence of fetches, only if $M - 1$ of the pages that are in $S_1$ in primary

\[3\] Their proof is with respect to demand paging algorithms for two level memories. However, this result holds for general paging algorithms because premature paging can not improve performance when two level memories are considered.
memory, are also in \( S_2 \) in primary memory, and the remaining page in the primary memory of \( S_2 \) need be fetched (now). Thus the decision whether a state transition is allowed depends on the next page which need be fetched. However, this problem can still be regarded as a task system (and as a \( M \)-server problem) because premature paging (i.e. forbidden state transitions) can not improve performance and thus, these forbidden state transitions are not needed to obtain minimal cost.

[RS] suggested a generalization of the demand paging problem. In their problem (called the generalized cache problem) a fault on page \( x \) costs \( w(x) \) (rather than a constant cost as in the "ordinary" problem). In this generalized problem as well, premature paging can not improve performance. They discuss this generalized paging problem in the context of randomized demand paging algorithms.

We generalize the demand paging problem with respect to deterministic algorithms, and consider a generalized two level memory, which is a two level memory whose second level is composed of several different speed memories. The first level is called the primary memory, and the second level is a collection of secondary memories. (Thus the cost of fetching a page which is stored in the second level is not constant; it depends on the secondary memory in which the page is stored). We model aspects of practical systems in which a generalized two level memory can be used:

1. A generalized two level main memory. For example, the first level of the main memory are its registers, and the second level is a collection of different speed fast memories.

2. A distributed two level memory. The secondary memories reside in different sites and are connected to the primary memory via communication links. The costs of accessing the secondary memories within the sites are negligible in comparison with the cost of transmitting a page (on the network). The cost of page transmission may change from one secondary memory to another (as they are in different sites). We consider two variants (pinned pages and non-pinned pages) of the distributed two level memory.

3. A two level memory whose second level is a collection of different speed disks. Here the cost of fetching a page from the second level is dominated by the cost of accessing the appropriate disk.

4. A complete action two level memory, which is a two level memory in which an operation can be performed only once all the disk accesses due to the previous operation are complete.

We present models for these environments, and we mainly discuss the competitiveness of LRU and FIFO with respect to these models. LRU and FIFO were chosen due to their wide use. (Similar to [ST], LIFO and LFU are not competitive with respect to all models considered in this paper). The models in which premature paging can not improve performance can be regarded as \( M \)-server problems (and thus as task systems). However, in some of the models, premature paging can improve performance and therefore, the demand paging problem is not a task system (this is proved) and thus not a \( M \)-server problem.

We do not have polynomial-time optimal off-line demand paging algorithms for these models (finding such algorithms is an open problem); this does not affect the discussion.
Naturally, the analysis is made under the assumption that both the on-line algorithms and the optimal off-line algorithms operate on the same generalized two level memory. Then, we extend the discussion by permitting the on-line algorithms to use \( \Delta \) extra primary memory page frames at the expense of \( \Delta \) page frames of the second level. Thus, the relationship between extra primary memory space and the ability to have the whole sequence of fetches as input is determined.

All the results reported in this paper (except for those of section 5) are also valid for a two level memory whose second level is composed of only two different speed secondary memories. We denote by \( M \) the size of the primary memory, by \( L/2 \) the time needed to fetch a page from the slowest secondary memory, and by \( I/2 \) the time needed to fetch a page from the fastest secondary memory. For simplicity, we assume that all algorithms initially store the same set of pages in primary memory and in each of the secondary memories.

A very interesting phenomena is observed. In all considered models for generalized two level memories in which premature paging can improve performance, LRU and FIFO are shown to be strongly competitive (with respect to demand paging algorithms). All other models considered (namely, the models in which premature paging can not improve performance), can be regarded as symmetric \( M \)-server problems. We prove lower bounds on the competitive factor of LRU and FIFO in these models, which depend on the costs of accessing the secondary memories. Thus, LRU and FIFO are not strongly competitive in these models. The (ordinary) two level demand paging problem (discussed in [ST]) is in this sense unique; although premature paging can not improve performance, LRU and FIFO are strongly competitive.

In section 2 we consider a generalized two level main memory. In section 3 we consider two variants of the distributed two level memory. In the first variant (pinned pages) each page has a fixed location in one of the secondary memories (this problem is identical to the generalized cache problem of [RS]). In the second variant (non-pinned pages), pages may change their location while performing the sequence of operations. In section 4 we consider a tightly coupled two level memory, namely, a two level memory whose second level is composed of a collection of different speed disks. In section 5 we consider a complete action two level memory, in which an operation can be executed only once all the accesses to secondary memories due to the previous operation are complete. In section 6 we extend all results to the case of additional primary memory page frames for the on-line algorithms.

2 A generalized two level main memory

We consider a two level main memory which is composed of a very fast memory (as the first level of memory), and a collection of fast, different speed, memories (as the second level of memory). The elements that are paged between the first level and the second level are called pages. The model for the generalized two level main memory is as follows:

- This is done as follows. Each primary memory page-frame is considered as a server. The graph is a weighted clique in which each node represents a page.
- A page resides in a page frame. When it may contain useful data it is called a data page and otherwise the page consists of a free ("empty") page frame.
1. There are \( z \) fast memories, \( U_1 \ldots U_z \), whose capacities (in number of pages) are \( C_1 \ldots C_z \), which constitute the second level of the generalized two level main memory. The capacity of the first level (i.e. primary memory) is \( M \) pages.

2. The cost of fetching a page from the primary memory is 0. The cost of fetching (writing) a page from (to) \( U_1 \) is \( l_i/2 \) (without loss of generality, \( i < j \Rightarrow l_i < l_j \)). We denote \( l_i \) by \( l \) and \( l_z \) by \( L \).

3. The total number of data pages in the two level memory is \( M + \sum_{i=1}^{z} C_i \) pages. Thus, each page is stored either in the primary memory or in one of the second level fast memories, and there are no free ("empty") page frames.

4. If a page \( p \) is fetched from memory \( U_i \) (i.e. a page-fault occurs), then one page must be paged out from primary memory (to free a page frame for \( p \)) to \( U_i \). (We assume that there is a buffer that enables this exchange of pages). The cost of such a page-fault is \( l_i \) (as one page is fetched from \( U_i \) at a cost \( l_i/2 \) and one page is written to \( U_i \) at a cost \( l_i/2 \)).

Note that MIN is no longer an optimal off-line algorithm when a generalized two level main memory is considered (see case 2.1 of theorem 2.1). We prove later that this problem is not a task system (and thus, not a \( M \)-server problem). It follows that premature paging can improve performance (because otherwise, this problem would have been a task system).

We start in proving a lower bound for an arbitrary on-line demand paging algorithm \( A \). An optimal on-line demand paging algorithm for the generalized two level main memory problem is denoted by OPT. Given a sequence of fetches, we denote by \( COST(A) \) the cost of performing the sequence by algorithm \( A \).

**Theorem 2.1:** Let \( A \) be any on-line demand paging algorithm. If both \( A \) and OPT initially store the same set of pages in primary memory and in each memory of the second level, and if \( M > 1 \), then there exists a sequence of fetches of length \( 1 + M \times X \) (for all \( X \geq 1 \)) for which
\[
COST(A) \geq \frac{2^*l - L + L \times M \times X}{2^*l + l \times X} \times COST(OPT)
\]

Example: Consider a two level main memory whose second level is composed of two memories, \( U_1 \) and \( U_2 \) (\( l_1 = l_2 = L \)). Let \( C_1 = 1 \), \( C_2 = 1 \), and \( M = 3 \). Let \( S = \{s_1, s_2, s_3\} \) be the initial content of primary memory, let \( a \) be the single page in \( U_1 \), and let \( b \) be the single page in \( U_2 \). The following table demonstrates the basic idea behind the construction of the sequence (for which the theorem holds). In this example (in which \( X = 1 \)), LRU is the on-line algorithm and MIN is the off-line algorithm.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>LRU's primary memory</th>
<th>( U_1 )</th>
<th>( U_2 )</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( s_1, s_2, s_3 )</td>
<td>( a )</td>
<td>( b )</td>
<td></td>
</tr>
<tr>
<td>( b )</td>
<td>( s_2, s_3, a )</td>
<td>( a )</td>
<td>( s_1 )</td>
<td>( L )</td>
</tr>
<tr>
<td>( c_2 = a )</td>
<td>( s_3, b, a )</td>
<td>( s_2 )</td>
<td>( s_3 )</td>
<td>( L )</td>
</tr>
<tr>
<td>( c_3 = s_1 )</td>
<td>( b, a, s_1 )</td>
<td>( s_2 )</td>
<td>( s_3 )</td>
<td>( L )</td>
</tr>
<tr>
<td>( c_4 = s_3 )</td>
<td>( a, s_1, s_2 )</td>
<td>( s_2 )</td>
<td>( b )</td>
<td>( L )</td>
</tr>
</tbody>
</table>

For \( i > 3 \), \( c_i \) is the page that was paged-out by LRU due to the fetch of \( c_{i-1} \). \( s_2 \) never
appears in the sequence. Thus LRU faults on all $e_i$, $i > 2$, at a cost $L$. The following table describes the execution of MIN on the same sequence.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>MIN's primary memory</th>
<th>$U_1$</th>
<th>$U_2$</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1 = b$</td>
<td>$s_1, s_2, s_3$</td>
<td>$a$</td>
<td>$b$</td>
<td></td>
</tr>
<tr>
<td>$e_2 = a$</td>
<td>$s_1, s_2, b$</td>
<td>$a$</td>
<td>$s_2$</td>
<td>$L$</td>
</tr>
<tr>
<td>$e_3 = s_1$</td>
<td>$s_1, s_2, a$</td>
<td>$b$</td>
<td>$s_2$</td>
<td>$l$</td>
</tr>
<tr>
<td>$e_4 = s_3$</td>
<td>$s_1, s_3, a$</td>
<td>$b$</td>
<td>$s_2$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

After fetching $e_2$, MIN faults once for each $M$ faults of LRU, at a cost $l$. It is easy to see that the ratio of costs in this example is larger than the ratio of theorem 2.1 and thus, theorem 2.1 holds for this example.

Proof sketch: It is sufficient to prove the theorem when $A$ is compared with an off-line demand paging algorithm $B$ instead of OPT, because $COST(OPT) \leq COST(B)$. Except for a single case (discussed below), algorithm $B$ acts like MIN, namely, pages out the page whose next fetch is the latest in the sequence (in comparison with all other pages in primary memory).

The basic idea of the proof is the following. We consider only $M+2$ pages; the set of $M$ pages which is initially stored in primary memory, one page which is initially stored in $U_1$, and one page which is initially stored in $U_2$. The first element ($e_1$) is the page of $U_2$, and the second element ($e_2$) is the page of $U_1$. The page that is paged-out by $A$ to $U_1$ due to the fetch of $e_2$, never appears in the sequence that we construct. This page is paged-out by the off-line algorithm $B$ to $U_2$ due to the fetch of $e_1$. Thus, once $e_2$ is fetched, each fault costs $L$ for $A$, but only $l$ for $B$. Moreover, in the sequence that we construct, $B$ faults only once for each $M$ faults of $A$. Thus an asymptotic lower bound of $ML/l$ is obtained.

Denote by $S = \{s_1, \ldots, s_M\}$ the set of $M$ pages that are initially stored by both $A$ and $B$ in primary memory. We choose one page from $U_1$ and denote it by $a$, and one page from $U_2$ and denote it by $b$. We will construct a sequence $e_1, \ldots, e_n$, of length $n = 1 + M \ast X$ for which the theorem holds. This sequence will only include pages from the set $S \cup \{a\} \cup \{b\}$. The first two elements of the sequence are, $e_1 = b$ and $e_2 = a$. Based on the paging decisions of $A$ on $e_1$ and $e_2$, we prescribe $e_3$. Based on the paging decision of $A$ on $e_3$, we prescribe the rest of the sequence. Thus, end up with one of three sequences depending on $A$'s early decisions (see below), we show that for each such sequence there is an off-line paging strategy, called "$B"$, that can accomplish the required bound. Since an optimal off-line algorithm will do at least as good as these strategies, the theorem is established.

Let $x$ be the page that is paged-out by $A$ due to the fetch of $e_1 = b$, and let $y$ be the page that is paged-out by $A$ due to the fetch of $e_3 = a$. By the definition of demand paging, $x \neq y$. Thus, after fetching $a$ and $b$, $x$ is stored by $A$ in $U_2$, and $y$ is stored by $A$ in $U_1$. We consider two cases: $y \neq b$, and $y = b$.

**THE FIRST CASE:** $y \neq b$

In this case, we use MIN as the off-line algorithm $B$. In the sequence that we construct, $y$ will never appear. (Thus $y$ is paged-out by $B$ to $U_2$ due to its fault on $e_1$). We first

---

We just note that if LRU (FIFO) is the algorithm considered, then $y \neq b$.  

7
define the next $M - 1$ elements of the sequence. Let $e_3 = x$ (thus $A$ faults on $e_3$), and for all $i$, $3 < i \leq M + 1$, let $e_i$ be the page that was paged-out by $A$ due to the fetch of $e_{i-1}$. The first group of additional $M$ pages, so that $B$ faults at most once (at a cost of 1) while $A$ faults all the time (at a cost $L$), is constructed as follows: for all $i$, $M + 1 < i \leq 2M + 1$, let $e_i$ be the page that was paged-out by $A$ due to the fetch of $e_{i-1}$. The subsequent groups are constructed identically. This yields a ratio of costs which is larger than the ratio in (1).

**The second case:** $y = b$.

Define the third element of the sequence to be $e_3 = b$. Thus, $e_3$ causes $A$ a page-fault. Let $p$ ($p \neq b$) be the page that is paged-out by $A$ (to $U_1$) due to the fetch of $e_3 = b$ (from $U_1$). There are two cases:

1. $p \neq a$. In this case as well, MIN is considered as the off-line algorithm $B$. The page $p$ will not appear in the sequence that we construct. (Thus $p$ is paged-out by $B$ to $U_1$ due to its fault on $e_1$). We first define additional $M - 2$ elements of the sequence. Let $e_3 = x$ (thus $A$ faults on $e_3$), and for all $i$, $4 < i \leq M + 1$, let $e_i$ be the page that was paged-out by $A$ due to the fetch of $e_{i-1}$. Each group of additional $M$ elements is defined as in the first case above. This yields a ratio of costs which is larger than the ratio in (1).

2. $p = a$. In this case, $a$ will not appear in the rest of the sequence that we construct (thus, its only appearance is as $e_2$). The elements $e_4 \ldots e_{M+1}$ are defined as in case 2.1 above, and so are the subsequent groups (of $M$ elements each). Unlike case 2.1, after fetching $e_1 = b$, algorithm $B$ can not page-out $p = a$ (to $U_2$) because $a$ is in $U_1$. In this special case, $B$ is defined to act as follows; let $F$ be the set of pages that do not appear in the first $M + 1$ elements of the sequence, and that are initially stored in primary memory. $|F| \geq 2$ because $|S| = M$, $e_1 = b$, $e_2 = a$ and $e_3 = b$). Unlike MIN, $B$ pages out to $U_4$ (due to the fault on $e_1$), the page in $F$ which is the first to be fetched after the first $M + 1$ elements of the sequence. With respect to all elements except for $e_1$, $B$ acts like MIN. This refinement of $B$ (in comparison with the previous cases) is sufficient to show that the theorem holds for this case as well. □

**Corollary 2.1:** As we can choose the length of the sequence to be as large as we want, the lower bound for any on-line demand paging algorithm is asymptotically $M \cdot L/1$.

**Theorem 2.2:** The demand paging problem of the generalized two level main memory is not a task system.

**Proof:** [BLS] describes $a_2$ on-line paging algorithm which is $S^2$-competitive for every task system having $S$ states. Thus the performance of this on-line algorithm (relative to the optimal off-line algorithm) is independent of the state transition cost matrix. Assume by a way of contradiction, that the demand paging problem under this model is a task system having $S$ states. Choose $L$ to be $1 + S^2$. Theorem 2.1 implies that the lower bound on the performance of any on-line algorithm is $M \cdot S^2$. This contradicts the result of [BLS]. □

**Corollary 2.2:** Premature paging can improve performance under this model.

We now prove that the upper bound for LRU is equal to the lower bound just proven. The theorem holds for FIFO as well, and the proof is identical.
Theorem 2.3: For $M > 1$, if both LRU and OPT initially store the same set of pages in primary memory and in each memory of the second level, then for any sequence, of any length,

$$\text{COST(LRU)} \leq \frac{L \cdot M}{l} \cdot \text{COST(OPT)}$$

Proof: We first prove that OPT must fault at least once for every $M$ faults of LRU. As the proof is essentially identical to the proof of theorem 6 in [ST], we outline it in the appendix. In the worst case (for LRU), each fault costs $L$ for LRU, while each fault only costs $l$ for OPT. The theorem follows. □

Theorems 2.1 and 2.3 imply that both LRU and FIFO are strongly $\frac{M^2 L}{l^2}$-competitive algorithms for the generalized two level main memory demand paging problem. The following table summarizes the results with respect to the generalized two level main memory.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Asymptotic Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any on-line demand paging alg.</td>
<td>$ML/l$</td>
<td>algorithm specific</td>
</tr>
<tr>
<td>LRU &amp; FIFO</td>
<td>$ML/l$</td>
<td>$ML/l$</td>
</tr>
<tr>
<td>LIFO &amp; LFU</td>
<td>--</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

3 A Distributed Two Level Memory

In the previous section, we have considered a main memory which is composed of a very fast memory and a collection of fast memories. In this section, we consider a collection of secondary memories, residing in different sites, which are connected to the primary memory via communication links. The cost of fetching a page from a secondary memory is dominated by the cost of sending a message (requesting the page) to the secondary memory, and the cost of sending the page to the primary memory (the cost of accessing a secondary memory is relatively negligible). This configuration is called a Distributed Two Level Memory (DTLM for short). While in the first model both levels together were viewed as the main memory, in this model, the second level is considered as the secondary memory, namely, the major storage device of the distributed system. Thus, some of the page frames in the secondary memories are probably free ("empty"). Assume messages arrive in sending order on links.

Denote the cost of sending a message to a secondary memory $U$, by $li/2$. A page-fault, in which page $p_i$ is fetched from $U_i$ and page $p_j$ is paged-out to $U_j$ (i may be equal to $j$), always costs $li$; first, a message is sent to $U_i$ (at a cost $li/2$) requesting $p_i$. Then, $p_i$ is sent to the primary memory at a cost $li/2$. Now, in order to write $p_j$ to $U_j$, $p_i$ is sent to $U_j$, but we can carry on in executing the sequence of fetches (immediately after sending $p_i$), as we do not have to wait until $p_i$ reaches $U_j$. (This is so even if the next fetch is of $p_i$, because the new request message for $p_i$, reaches $U_j$ after the message that writes $p_i$).

We consider two variants of this problem. In subsection 3.1, we consider applications for which each page must have a fixed locations in one of the secondary memories.\(^5\) Thus

\(^5\)Recall that this variant is identical to the generalized cache problem of [RS]. However, they discussed it with respect to randomized on-line demand paging algorithms.
we say that the pages are **pinned**. Examples of such applications are data structures (such as B-trees or hash tables) that use pointers which are actual addresses. In subsection 3.2, we consider applications for which the pages need not be pinned. This is usually the case when indirect addressing, which is implemented either in software or in hardware, is used.

### 3.1 The pinned pages variant of the DTLM

As all pages are pinned, the free ("empty") page frames in the secondary memories may be ignored; these applications never use them. Therefore, we assume (without loss of generality) that there are no free page frames. The optimal off-line algorithm for this variant is denoted by OPT'. The model for the pinned pages variant of the DTLM follows:

1. There are \( x \) secondary memories, \( U_1 \ldots U_x \), whose capacities are \( C_1 \ldots C_x \), which reside in different sites and which are connected (via communication links) to the primary memory whose capacity is \( M \).
2. The cost of fetching a page from the primary memory is 0. The cost of sending (receiving) a message to (from) \( U_i \) is \( l_i/2 \). (Without loss of generality, \( i < j \Rightarrow l_i < l_j \). We denote \( l_i \) by \( l \) and \( l_j \) by \( L \).
3. The pinned pages variant of the DTLM contains \( \sum_i C_i \) distinct data pages. Each data page has a fixed location in one of the secondary memories. replicas of \( M \) data pages are stored in the primary memory at any point in time.
4. A page-fault, in which page \( p_1 \) is fetched from the secondary memory \( U_i \) and page \( p_2 \) is paged out from the primary memory to \( U_j \) (\( i \) may be equal to \( j \)), costs \( l_i \).

At first glance, the pinned pages variant of the DTLM seems as an asymmetric \( M \)-server problem; each page is a node, and the length of the directed arc from \( p_i \) to \( p_j \) is equal to the cost of fetching \( p_j \) from its fixed location in one of the secondary memories. However, it can be viewed as a symmetric \( M \)-server problem [RS] (by charging in point 4 above \( (l_i + l_j)/2 \) rather than \( l_i \), namely, charging half cost when being paged-in and half cost when being paged-out). It follows that premature panning can not improve performance when the pinned pages variant of the DTLM is considered.

We first show that MIN is not an optimal off-line algorithm for this variant. Let \( S = \{s_1 \ldots s_M\} \) be the set of \( M \) pages initially stored in MIN's primary memory. Let \( a \notin S \) be a page which is stored in \( U_i \), and let \( b \notin S \) be a page which is stored in \( U_j \), so that \( l_i < l_j \), and let \( c, c \notin S, c \neq a, c \neq b \), be a page which is stored in \( U_k \). (\( k \) may be equal to \( i \) or \( j \)). Now consider the following sequence of \( M + 3 \) fetches: \( e_1 = a, e_2 = b, e_3 = c \), for all \( i, 1 \leq i \leq M - 2, e_{i+3} = s_i, e_{M+2} = a, \) and \( e_{M+3} = b \). MIN faults on \( c \) and \( e_2 \). As \( s_{M-1} \) and \( s_M \) never appear in the sequence, MIN pages out these two pages, due to the faults on \( e_1 \) and \( e_2 \). MIN also faults on \( e_3 = c \) and thus it pages out \( b \), as all other pages in primary memory appear before \( b \) in the rest of the sequence. Therefore, MIN faults on \( e_{M+3} = b \), and thus this sequence costs MIN \( l_i + 2l_j + l_k \). But an algorithm that pages out \( a \), instead of \( b \), due to the fetch of \( e_3 = c \) (and thus faults on \( e_{M+2} \) instead of \( e_{M+3} \)), obtains a cost which is only \( 2l_i + l_j + l_k \). Thus MIN is not optimal.
We do not have a lower bound for an arbitrary on-line algorithm $A$ (with respect to arbitrary long sequences) which is larger than $M$.\footnote{Recall that \cite{MMS} conjectured that $M$ is the lower bound.} But using the following theorems, we prove tighter lower bounds for LRU and FIFO. The following theorems hold when the capacities of both the fastest and the slowest secondary memories are larger than $M$ pages (i.e. $C_1>M$ and $C_s>M$).

**Theorem 3.1.1:** If both LRU and OPT' initially store the same set of pages in primary memory and in each secondary memory, and if $M > 1$, then there exists a sequence of fetches of length $M + 2\cdot(M+1)\cdot X$ (for all $X \geq 1$) for which

$$\text{COST(LRU)} \geq \frac{(M-1)L + I + 2X\cdot((M-1)L + I)}{(M-1)L + I + 2X\cdot I} \cdot \text{COST(OPT')}. $$

*Proof:* It is sufficient to prove the theorem when LRU is compared with an off-line algorithm $B$. Let $C = \{c_1, c_2\}$ be a set of two distinct pages of $U_1$, and let $D = \{D_1 \ldots D_{M-1}\}$ be a set of $M - 1$ distinct pages of $U_2$. We construct a sequence of fetches, that only contains pages from the set $C \cup D$. The first $M$ elements of the sequence are: $c_1 = c_1$, and for all $i$, $2 \leq i \leq M$, $e_i = D_i-1$. Each additional group of $2\cdot(M+1)$ elements is as follows; $c_1$, $c_2$, $D_1 \ldots D_{M-1}$, $c_2$, $c_1$, $D_1 \ldots D_{M-1}$. Note that after fetching the first $M$ elements, LRU faults on $M$ out of $M+1$ elements.

The off-line algorithm $B$ constantly stores the pages of the set $D$ in primary memory (after fetching the first $M$ elements of the sequence). The theorem follows. \hfill $\Box$

**Corollary 3.1.1:** As we can choose the length of the sequence to be as large as we want, the lower bound obtained from theorem 3.1.1 for LRU is asymptotically:

$$1 + \frac{(M-1)L}{I}.$$  

In the appendix we prove that OPT' faults at least once for every $M$ faults of LRU. Thus we can obtain an upper bound $M \cdot L/I$ for LRU. (It follows that LRU is \frac{M}{M'-I} competitive). But we conjecture (and have not been able to prove) that the lower bound just obtained for LRU is also the upper bound. In any case, the lower and upper bounds obtained for LRU are very close.

In theorem 3.1.4 we present a group of $M$ lower bounds for FIFO (no explicit proof is provided). Depending on parameters' values, one of these bounds may be larger than the other. For clarity, we first present two propositions in which we prove two of these bounds.

**Proposition 3.1.2:** If both FIFO and OPT' initially store the same set of pages in primary memory and in each secondary memory, and if $M > 1$, then there exists a sequence of fetches of length $M + (M+1)\cdot X$ (for all $X \geq 1$) for which

$$\text{COST(FIFO)} \geq \frac{(M-1)L + I + X\cdot((M-1)L + 2I)}{(M-1)L + I + 2X\cdot I} \cdot \text{COST(OPT')}$$

*Proof:* It is sufficient to prove the theorem when FIFO is compared with an off-line algorithm $B$. Let $C = \{c_1, c_2\}$ be a set of two distinct pages of $U_1$, so that $c_1$ is not initially in primary memory. Let $D = \{D_1 \ldots D_{M-1}\}$ be a set of $M-1$ distinct pages of $U_2$.
which are not initially in primary memory. We construct a sequence of fetches, that only contains pages from the set \( C \cup D \). The first \( M \) elements of the sequence are: \( e_1 = c_1 \), and for all \( i, 2 \leq i \leq M \), let \( e_i = D_{i-1} \). Each additional group of \( M + 1 \) elements is as follows; \( c_2, c_1, D_1 \ldots D_{M-1} \).

The off-line algorithm \( B \) constantly stores the pages of the set \( D \) in primary memory (after fetching the first \( M \) elements of the sequence). The theorem follows. \( \square \)

Corollary 3.1.2: As we can choose the length of the sequence to be as large as we want, the lower bound obtained from proposition 3.1.2 for FIFO is asymptotically:

\[
1 + \frac{(M-1)L}{2L} = \Theta(M) \tag{2}
\]

Proposition 3.1.3: If both FIFO and OPT’ initially store the same set of pages in primary memory and in each secondary memory, and if \( M > 1 \), then there exists a sequence of fetches of length \( M + (M^2 - 1) \times X \) (for all \( X \geq 1 \)) for which

\[
COST(FIFO) \geq \frac{(M-1)l + L + X(M-1)l + (M^2 - 1)L}{(M-1)l + L + XM^2l} \times COST(OPT')
\]

Proof: It is sufficient to prove the theorem when comparing FIFO with an off-line algorithm \( B \). Let \( d \) be a page of \( U_d \), which is not in primary memory. Let \( C = \{c_1 \ldots c_M\} \) be a set of \( M \) distinct pages of \( U_1 \), such that at least one of them, denoted by \( c_1 \), is not initially in primary memory. We construct a sequence of fetches that only contains pages from the set \( C \cup \{d\} \), and on which FIFO always faults. The first \( M \) elements of the sequence are: for all \( i, 1 \leq i \leq M-1 \), \( e_i = c_{i+1} \), and \( e_M = d \). Each additional group of \( (M-1) \times (M+1) \) elements is as follows (we describe the first group): let \( e_{M+1} = c_1 \) and for all \( j \) and \( k \), \( 0 \leq j \leq M \) and \( 1 \leq k < M \) (except for the pair \( j = 0 \) and \( k = 1 \)), let \( e_k(M+1+j) \) be the page that was paged-out by FIFO due to the fetch of \( e_k(M+1+j-1) \). In other words, the subsequent \( M^2 - 1 \) elements of the sequence are constructed by repeating \( M - 1 \) times the following; fetch all \( M \) pages of the set \( C \) and the page \( d \) (each page once) so that FIFO faults on all fetches.

The off-line algorithm \( B \) constantly stores the page \( d \) in primary memory (after fetching the first \( M \) elements of the sequence), and except for that, acts like MIN (i.e. \( B \) pages out the primary memory page whose next fetch is the latest in the sequence in comparison with all other pages in primary memory which are different than \( d \)). Thus \( B \) never faults on \( d \). Consider now a page-fault of \( B \) on a page \( c_i \in C \). All the pages of the set \( C - \{c_i\} \) are in primary memory. Thus there is exactly one page in \( C - \{c_i\} \), denoted by \( c_p \), which does not appear in the subsequent \( M^2 - 2 \) elements \( (M-1) \) elements, if one of the subsequent \( M^2 - 2 \) elements is \( d \) of the sequence. \( B \) pages out \( c_p \), and thus \( B \) does not fault on the subsequent \( M^2 - 2 \) elements \( (M-1) \) elements, if one of the subsequent \( M^2 - 2 \) elements is \( d \) of the sequence. We conclude that \( M^2 - 1 \) elements were added to the sequence. On the \( M-1 \) elements which are \( d \), \( B \) never faults. \( B \) faults on the first element \( (e_{M+1} = c_1) \). When \( B \) faults on an element, it does not fault on the subsequent \( M^2 - 2 \) elements \( (M-1) \) elements if one of the subsequent \( M^2 - 2 \) elements is \( d \). It follows that \( B \) faults exactly \( M \) times (because, \( M \) faults, plus \( M^2(M-2) \) fetches of pages from the set \( C \) on which \( B \) does

---

\(^7\)The constraint on the pages of the set \( D \) can be relaxed, i.e. the theorem holds even if there are less than \( M - 1 \) pages of \( U_1 \) which are not initially in primary memory. We add this constraint for simplicity.
not fault, plus $M-1$ fetches of the page $d$, equals $M^2 - 1$). Thus the cost of $B$ is $M \ast l$. The theorem follows for $B$ and thus for OPT'. □

Example: Consider the following example for $M = 3$ and $X = 1$:

<table>
<thead>
<tr>
<th>Sequence</th>
<th>FIFO's primary memory</th>
<th>Paged-out</th>
<th>Cost</th>
<th>B's primary memory</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$s_1, s_2, s_3$</td>
<td></td>
<td></td>
<td>$s_1, s_2, s_3$</td>
<td></td>
</tr>
<tr>
<td>$c_2$</td>
<td>$s_2, s_3, c_2$</td>
<td>$s_1$</td>
<td>1</td>
<td>$s_2, s_3, c_2$</td>
<td>1</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$s_3, c_2, c_3$</td>
<td>$s_2$</td>
<td>1</td>
<td>$s_3, s_1, c_3$</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>$c_2, c_3, d$</td>
<td>$s_3$</td>
<td>$L$</td>
<td>$c_2, c_3, d$</td>
<td>$L$</td>
</tr>
<tr>
<td>$c_1$</td>
<td>$c_1, d, c_1$</td>
<td>$s_2$</td>
<td>1</td>
<td>$c_1, c_1, d$</td>
<td>1</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$d, c_1, c_2$</td>
<td>$s_3$</td>
<td>1</td>
<td>$c_1, c_1, d$</td>
<td>1</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$c_1, c_2, c_3$</td>
<td>$s_2$</td>
<td>1</td>
<td>$c_1, c_2, c_3$</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>$c_2, c_3, c_1$</td>
<td>$d$</td>
<td>1</td>
<td>$c_2, c_3, c_1$</td>
<td>1</td>
</tr>
<tr>
<td>$c_1$</td>
<td>$c_1, d, c_1$</td>
<td>$c_2$</td>
<td>$L$</td>
<td>$c_1, c_1, d$</td>
<td>1</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$d, c_1, c_2$</td>
<td>$c_3$</td>
<td>1</td>
<td>$c_1, c_3, c_3$</td>
<td>1</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$c_1, c_2, c_3$</td>
<td>$d$</td>
<td>1</td>
<td>$c_1, c_2, c_3$</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>$c_2, c_3, d$</td>
<td>$c_1$</td>
<td>$L$</td>
<td>$c_1, c_2, c_3$</td>
<td>1</td>
</tr>
</tbody>
</table>

Corollary 3.1.3: As we can choose the length of the sequence to be as large as we want, the lower bound obtained from proposition 3.1.3 for FIFO is asymptotically:

$$\frac{(M-1) + (M-1)*L}{M*l}.$$  \hspace{1cm} (3)

Theorem 3.1.4: If both FIFO and OPT' initially store the same set of pages in primary memory and in each secondary memory, and if $M > 1$, then there exists sequences that yield $M-1$ lower bounds for the performance of FIFO (relative to OPT'). The $i$-th lower bound, $1 \leq i < M$, is asymptotically:

$$\frac{(M-i) + i*(M-i)*L}{(M-i+1)*l}.$$ \hspace{1cm} (4)

Proof: First note that the lower bound (2) (corollary 3.1.2) is equal to (4) for $i = M-1$, and the lower bound (3) (corollary 3.1.3) is equal to (4) for $i = 1$.

In order to prove (4), we construct sequences of length $M + (M-i)*(M+1)*X$ (for all $X \geq 1$). Let $C = \{c_1, \ldots, c_{M-i+1}\}$ be a set of distinct pages of $U_1$, so that $c_1$ is not initially in primary memory. Let $D = \{D_1, \ldots, D_i\}$ be a set of $M-1$ distinct pages of $U_2$, which are not initially in primary memory (this restriction can be relaxed; we add it for simplicity). The sequences that we construct contain pages only from the set $C \cup D$. Similar to the proof of proposition 3.1.3, each subsequent group of $(M-i)*(M+1)$ elements of the sequence is constructed by repeating $M-i$ times the following; fetch all pages of the set $C$ and all pages of the set $D$ (each page once) so that FIFO faults on all fetches. □

The following corollary conclude the discussion concerning the lower bound of FIFO.

Corollary 3.1.4:
The tightest lower bound we know for FIFO, depending on the values of $l$, $L$, and $M$, is:

$$\text{MAX}_{0 \leq i \leq M} \left\{ \frac{(M-i) + i*(M-i)*L}{(M-i+1)*l} \right\}$$
We now prove an upper bound for FIFO which is smaller than the upper bound obtained for LRU (i.e. \( M \cdot L/l \)). We believe that this upper bound is not tight.

**Theorem 3.1.5:** If both FIFO and OPT' initially store the same set of pages in primary memory and in each secondary memory, and if \( M > 1 \), then for any sequence, of any length,

\[
COST(FIFO) \leq 1 + \frac{L(M-1)}{l} \cdot COST(OPT)
\]

**Proof:** Consider a sequence \( \sigma \) of fetches. We execute FIFO on \( \sigma \) and we mark the elements on which FIFO faults. We construct a new sequence from \( \sigma \), denoted by \( \sigma_1 \), by omitting the unmarked element of \( \sigma \). The cost of FIFO on \( \sigma_1 \), denoted by \( COST_{FIFO}(\sigma_1) \), is equal to its cost on \( \sigma_1 \), as the fetches of pages which are in primary memory cost 0, and do not change the order among the pages in FIFO’s primary memory\(^8\) (this order is defined as follows; \( p_1 < p_2 \) iff \( p_1 \) was last pagéd-in before \( p_2 \). Thus \( p_1 \) will be pagéd-out before \( p_2 \). On the other hand, it can be shown that the cost of OPT' on \( \sigma_1 \) is smaller or equal to its cost on \( \sigma \).

In the appendix we prove that OPT' faults at least once for each group of \( M \) faults of FIFO. FIFO faults on all elements of \( \sigma_1 \). Consider the first \( M \) elements of \( \sigma_1 \). The best scenario for OPT' is that it faults only once on these elements, at a cost \( l_i \). In this case, the maximum cost for FIFO is \( l_i + L(M-1) \). The ratio of costs is maximum for \( l_i = l \). This is the case for each subsequent group of \( M \) elements of \( \sigma_1 \), \( e_{i+1} \) \( \cdots \) \( e_{i+M} \) (for every \( i \geq 0 \)). Thus:

\[
\frac{COST_{FIFO}(\sigma_1)}{COST_{OPT}(\sigma)} \leq \frac{COST_{FIFO}(\sigma_1)}{COST_{OPT}(\sigma_1)} \leq 1 + \frac{L(M-1)}{l} \quad \square
\]

Theorem 3.1.5 implies that FIFO is \((1 + \frac{LM-1}{l})\)-competitive. The following table summarizes the results obtained with respect to the pinned pages variant of the Distributed Two Level Memory.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Asymptotic Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any on-line alg.</td>
<td>( M )</td>
<td></td>
</tr>
<tr>
<td>LRU</td>
<td>( 1 + \frac{(M-1)L}{l} )</td>
<td>( ML/l )</td>
</tr>
<tr>
<td>FIFO</td>
<td>[ MAX_{0 \leq k &lt; M} { (M-i) + \frac{M(M-1)L}{l(M+i)} } ]</td>
<td>( 1 + \frac{M-1}{l} )</td>
</tr>
<tr>
<td>LIFO &amp; LFU</td>
<td>(-)</td>
<td>( \infty )</td>
</tr>
</tbody>
</table>

As the lower bounds on the competitive factors of LRU and FIFO depend on the costs of accessing the secondary memories, LRU and FIFO are not strongly competitive paging algorithms for the pinned pages variant of the DTLM.

### 3.2 The non-pinned Distributed Two Level Memory

Unlike the pinned pages variant of the DTLM, we now assume that the distributed system contains an address translation mechanism, and that all applications use indirect addressing\(^8\) Note that this is not the case for LRU.
(i.e. pointers are not actual addresses). Thus, the pages need not be pinned, and therefore, a page that was.paged-in from \( U_i \), may later be paged-out to any free page frame in one of the secondary memories. The model for the non-pinned DTLM (np-DTLM for short) follows:

1. There are \( z \) secondary memories, \( U_1 \ldots U_z \), whose capacities are \( C_1 \ldots C_z \), which reside in different sites and which are connected (via communication links) to the primary memory which contain \( M \) data pages.

2. The cost of fetching a page from the primary memory is 0. The cost of sending (receiving) a message to (from) \( U_i \) is \( i_i/2 \). (Without loss of generality, \( i < j \Rightarrow i_i < i_j \)). We denote \( i_i \) by \( i \) and \( i_j \) by \( j \).

3. Each secondary memory \( U_i \) initially contains \( F_i \) free page frames (0 \( \leq F_i \leq C_i \)).

4. The np-DTLM contains \( M + \sum_{i=1}^{z} (C_i - F_i) \) distinct data pages.

5. A page in primary memory may be paged-out to any free page frame in one of the secondary memories. A page-fault, in which page \( p_i \) is fetched from the secondary memory \( U_i \) and page \( p_j \) is paged out from primary memory to \( U_j \) (\( i \) may be equal to \( j \)), costs \( i_i \) (two messages).

6. Let:

\[
K = \text{Max} \{ i \mid \sum_{j=i+1}^{z} (C_j - F_j) < \sum_{j=1}^{i} F_j \} \\
T = \text{Min} \{ i \mid F_i > 0 \}
\]

First note that if for all \( i \), \( F_i = 0 \) (i.e. \( T \) is not defined), then the model coincides with that of section 2, and thus, the problem modeled in section 2 is a special case. We have presented the first problem separately because it also has a practical use as a generalized two level main memory. From now on, assume that \( T \) is well defined.

The definition of \( K \) means that if \( K = k_0 \), then all the data pages can be stored in \( U_1 \ldots U_{k_0+1} \), but not in \( U_1 \ldots U_{k_0} \). In the appendix we prove that the optimal off-line demand paging algorithm faults at least once for each group of \( M \) faults of LRU (FIFO). Therefore, the upper bound for any LRU-like on-line algorithm (i.e. an algorithm that chooses the page to be paged-out like LRU), that pages-out only to \( U_1 \ldots U_{k+1} \), is asymptotically \( (M + l_{K+1})/l_1 \). The upper bound for any LRU-like on-line algorithm is naturally \( M*L/l \) (i.e. \( M*l_1/l_1 \)). These results hold for FIFO-like on-line algorithms as well. Note that if \( K \) is undefined (i.e. \( \sum_{i=1}^{z}(C_i - F_i) \leq F_1 \)), then the lower bound for any on-line algorithm \( A \) is asymptotically only \( M \) as \( A \) may page-out only to \( U_1 \). From now on, we assume that \( K \) is well defined. An optimal off-line demand paging algorithm for the np-DTLM is denoted by OPT'.

**Lemma 3.2.1:** Let \( A \) be any on-line demand paging algorithm. If \( T = 1 \) and \( K \) is defined, and if both \( A \) and OPT' store the same set of pages in primary memory and in each of the secondary memories, then an asymptotic lower bound for the performance of \( A \) is

\[
\frac{M* l_{K+1}}{l_1}
\]
Proof: We first present the main idea. We construct a generator $G$ that gets as input the on-line demand paging algorithm $A$ and a parameter $n$, and constructs a sequence of length $n$ ($G$ operates in such a way that for all $t$, the sequence of length $t$ is a prefix of the sequence of length $t+1$). In the sequences that $G$ generates, all elements are taken from $U_{K+1} \ldots U_n$ of $A$ and thus, a page that is paged-out by $A$ to a free page frame in $U_1 \ldots U_K$, will not appear in the rest of the sequence. There are initially $\sum_{i=1}^{K} F_i$ free page frames in $U_1 \ldots U_K$, but $\sum_{i=K+1}^{n} (C_i-F_i) > \sum_{i=1}^{K} F_i$. Thus, there exists a value $n_0 \geq 0$ such that on all sequences of length $n > n_0$ that are generated by $G$, the on-line algorithm $A$ pages out only to $U_{K+1} \ldots U_n$ due to the fetch of each sequence element $e_i$, $i > n_0$.

All sequences of length $n > n_0$ that are generated by $G$, are constructed such that each element $e_i$, $i > n_0$, is a member of a set $Q$ that contains $M + 1$ distinct pages, and such that $A$ always faults. The optimal off-line algorithm $OPT''$ stores $M$ pages of the set $Q$ in primary memory, and the remaining page is stored by $OPT''$ in $U_1$. Thus $G$ generates arbitrary long sequences (of length greater than $n_0$) such that with respect to all elements beside the first $n_0$ elements, $A$ faults on each element at a cost $l_{K+1}$, and $OPT''$ faults once for each $M$ faults of $A$, at a cost $t$. Thus, the asymptotic lower bound follows.

We formally define the generator $G$. Let $S$ be the set of $M$ pages which are initially stored in primary memory of both $A$ and $OPT''$. For a given $n$, $G$ constructs the following sequence of fetches, on which $A$ always fault. The first element, $e_1$, is an arbitrary page from one of the secondary memories $U_{K+1} \ldots U_n$. $e_i$, $i \leq n$, is defined as follows: let $p$ be the page that was paged-out by $A$ to the secondary memory $U_j$, due to the fetch of $e_{i-1}$.

1. If $j > K$, then let $e_i = p$.
2. If $j \leq K$, then let $e_i$ be an arbitrary page $q$, $q \notin S$, that has not appeared in the sequence so far, and which is stored by $A$ (after fetching $e_{i-1}$) in one of the secondary memories $U_{K+1} \ldots U_n$. Such a page $q$ exists, see below.

The following are a few observations regarding the above sequence:

1. $A$ faults on all the sequence elements. In each page-fault of $A$, $A$ pages-in from one of the secondary memories $U_{K+1} \ldots U_n$, and thus each fault costs at least $l_{K+1}$.
2. If a page $r$ is paged-out by $A$ to one of the secondary memories $U_1 \ldots U_K$ (i.e. case (2) above), then it will never appear again in the rest of the sequence.
3. While constructing the sequence, case (2) can occur at most $\sum_{i=1}^{K} F_i$ times. Therefore, when case (2) occurs due to the fetch of $e_i$, we can always find a page $q$ such that $q \notin S$, $q$ never appeared in the sequence so far, and $q$ is stored by $A$ (after fetching $e_{i-1}$) in one of the secondary memories $U_{K+1} \ldots U_n$, because $\sum_{i=K+1}^{n} (C_i-F_i) > \sum_{i=1}^{K} F_i$.

As $F_i > 0$, there is at least one page frame in $U_1$ which is initially free ("empty"). We choose one such page frame and call it the guarded place. It is sufficient to prove lemma 3.2.1 with respect to some off-line algorithm $B$, which is defined as follows.

When $B$ faults on $e_{i-1}$:

- If there exists a page $w$ in the primary memory of $B$, which never appears in the rest of the sequence (i.e. either $w$ was initially in primary memory and it never appears in the sequence, or $w$ was paged-out by $A$ to one of the secondary memories $U_1 \ldots U_K$, due to a fetch of a sequence element $e_j$, $j < i$), then $w$ is the one to be paged-out to
a free page frame in one of the secondary memories $U_{K+1} \ldots U_z$. (If there are two or
more such pages, then one of them is arbitrarily chosen and paged-out).

- Otherwise, the page whose next fetch is the furthest down the sequence is paged-out
to the guarded place (in $U_i$), if the guarded place is free, and otherwise, to a free
page frame in one of the secondary memories $U_{K+1} \ldots U_z$.

It can be shown that for arbitrary long sequences of length $s > n_0$ that are generated
by $G$, the ratio of cost (between $A$ and $B$) is asymptotically $M*1_{K+1}/l$. □

Remarks: The requirement that both $A$ and OPT" store the same set of pages in
each secondary memory, can be relaxed. It is sufficient that OPT" initially has one free
page frame in $U_1$, and that both $A$ and OPT" initially store the same set of pages (denoted
by $R$) in $U_1 \ldots U_K$, but not necessarily the same subset of $R$ in each $U_i$ ($i \leq K$). The proof
is almost identical.

**Theorem 3.2.1:** Let $A$ be any on-line demand paging algorithm. If $K$ and $T$ are
defined, and if both $A$ and OPT" store the same set of pages in primary memory and in
each of the secondary memories, then an asymptotic lower bound for the performance of $A$ is

$$M*1_{K+1}/l.$$ 

Proof sketch: We consider two cases:
If $K < T$, then it is obvious that $U_1 \ldots U_{T-1}$ are full (i.e. they have no free page frames),
and that either $K = T - 1$ (i.e. there are data pages somewhere in $U_T \ldots U_z$) or $K = T - 1$
(otherwise). Using the idea of the proof of theorem 2.1, one can construct a sequence for
which this theorem holds.
If $K > T > 1$, then we first access an element from $U_1$ (to enable OPT" to free a page frame in
$U_1$). If $A$ pages out to $U_j$, $j \leq K$, then OPT" pages out (due to this first fetch) to one
of $U_2 \ldots U_K$, and otherwise, OPT" pages to the same page frame that $A$ pages to. Now
lemma 3.2.1 (see remark above) holds. □

**Corollary 3.2.1:** The demand paging problem under np-DTLM is not a task system. 10

**Corollary 3.2.2:** Premature paging can improve performance under np-DTLM.

The following table summarizes the results obtained with respect to the np-DTLM,
when $K$ is defined. In the table, $S_{LRU}$ ($S_{FIFO}$) denotes the family of on-line algorithms that
choose the page to be paged-out like LRU (FIFO). $C_{LRU}$ ($C_{FIFO}$) is a sub-family of
$S_{LRU}$ ($S_{FIFO}$) that contains the algorithms that never page-out to the secondary memories
$U_{K+2} \ldots U_z$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Asymptotic Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any on-line demand alg.</td>
<td>$M*1_{K+1}/l$</td>
<td>$M*1_{K+1}/l$</td>
</tr>
<tr>
<td>$C_{LRU}$ &amp; $C_{FIFO}$</td>
<td>$M*1_{K+1}/l$</td>
<td>$M*1_{K+1}/l$</td>
</tr>
<tr>
<td>$S_{LRU}$ &amp; $S_{FIFO}$</td>
<td>$M*1_{K+1}/l$</td>
<td>$M*1_{K+1}/l$</td>
</tr>
<tr>
<td>LIFO &amp; LFU</td>
<td>-</td>
<td>$M*1_{K+1}/l$</td>
</tr>
</tbody>
</table>

10 Theorem 3.2.1 extends lemma 3.2.1 for values of $T$ which are greater than 1.

11 Similar to theorem 2.2., the lower bound depends on the paging costs.
It follows that all algorithms in $C_{LRU}$ and $C_{FIFO}$ are strongly competitive on-line demand paging algorithms under np-DTLM.

4 A tightly coupled two level memory

In this section, unlike section 3, total cost is dominated by the cost of accessing the secondary memories (and not by communication costs). In fact, we try to model a two level memory that is composed of a primary memory and $x$ different speed disks. This is done with respect to the model of the pinned pages variant of the DTLM (subsection 3.1). The only difference between this model and the model of subsection 3.1 is the way of estimating the cost of a page-fault.

Consider a page-fault in which a sequence element $e_i = p_1$ is paged-in from $U_i$, and page $p_2$ is paged-out to $U_j$ (i may be equal to j). We have to read $p_1$ from the disk $U_i$ at a cost $l/2$ at most. In modern disks, one can write $p_2$ to the buffer of the disk $U_j$, and then to carry $e_i$ in fetching the next element of the sequence (i.e. $e_{i+1}$), knowing that $p_2$ will be eventually written. But if $e_{i+1} = p_0$ then the cost of fetching $p_2$ from $U_j$ may be much smaller than $l/2$ as the page may still be in the buffer of disk $U_j$, and otherwise, if it was just written to the disk $U_j$, then the disk head might already be at the right position for fetching $p_2$. On the other hand, if $e_{i+1} \neq p_2$, and $e_{i+1}$ is fetched from $U_i$ (i may be equal to j), then $l/2$ is a reasonable approximation for the cost of fetching $e_{i+1}$.

To avoid this non-uniformity of costs, we now prove lower bounds for the tightly coupled two level memory, using sequences that obey the following disk restriction:

if $p$ was page-out due to the fetch of a sequence element $e_i$

then the next element $e_{i+1}$ must be a page different than $p$.

We now prove a lower bound for arbitrary on-line algorithms which is equal to $(M+1)/2$, using a sequence that obeys the disk restriction (with respect to both the on-line algorithm $A$ and the optimal off-line algorithm). Denote by $S$ the set that contains the $M$ pages that are initially stored in primary memory, and two additional pages of $U_1$, $p_1$ and $p_2$. Let the first element of the sequence be $e_1 = p_1$. (Thus $A$ faults on $e_1$). The element $e_{i+1}$ is defined to be the page of $S$ which is not in primary memory (after fetching $e_i$) and then was not paged out due to the fetch of $e_i$. Thus $A$ faults on all elements while the optimal off-line algorithm may fault twice for each $M+1$ faults of $A$.

Similar to the proof of theorem 3.1.1, let $C = \{c_1, c_2, c_3\}$ be a set of three pages of $U_1$, and let $D = \{D_1 \ldots D_{M-1}\}$ be a set of $M-1$ pages of $U_2$. The following sequence proves a lower bound for LRU, which is asymptotically

$$1 + \frac{(M-1)+L}{2},$$

for sequences that obey the disk restriction; the first $M$ elements are $c_1, D_1 \ldots D_{M-1}$. Then we repeatedly add $M+2$ elements to the sequence; the first element is the only page of the set $C$ which is in primary memory, the second and the third are the other two pages of the set $C$, and the rest $M-1$ elements are $D_1 \ldots D_{M-1}$ in that order.
Similar to the proof of theorem 3.1.4, let \( C = \{\epsilon_1 \ldots \epsilon_{M-i+2}\} \) be a set of \( M-i+2 \)
pages of \( U_1 \), and let \( D = \{D_1 \ldots D_i\} \) be a set of \( i \) pages of \( U_2 \). Assume that \( D_1 \ldots D_i \) are
not initially in primary memory (this restriction can be relaxed; we add it for simplicity). We can construct sequences of length \( M + (M-i) \cdot (M+2) \cdot X \) (for all \( X \geq 1 \)) that obey
the disk restriction which yield a lower bound for FIFO which is asymptotically

\[
\text{MAX}_{1 \leq i < M} \left\{ \frac{(M-i) + i \cdot (M-i) \cdot L}{(M-i+2) \cdot i} \right\}
\]

The following table summarizes the lower bounds obtained for the tightly coupled two
level memory. For some parameters' values, the expressions in the table for the lower
bounds of LRU and FIFO may be smaller than \( (M+1)/2 \); in these cases, the lower bounds
for LRU and FIFO are naturally \((M+1)/2\). Note that the upper bounds of subsection 3.1
hold here as well.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Asymptotic Lower Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any on-line algorithm</td>
<td>((M+1)/2)</td>
</tr>
<tr>
<td>LRU</td>
<td>1 + (\frac{(M-1)L}{M})</td>
</tr>
<tr>
<td>FIFO</td>
<td>\text{MAX}_{1 \leq i &lt; M} \left{ \frac{(M-i) + i \cdot (M-i) \cdot L}{(M-i+2) \cdot i} \right}</td>
</tr>
<tr>
<td>LIFO &amp; LFU</td>
<td>-</td>
</tr>
</tbody>
</table>

5 A complete action two level memory

In this model, similar to subsection 3.1, the pages are pinned. When a page-fault, in which
page \( p_1 \) is paged-in from \( U_1 \) and page \( p_2 \) is paged-out to its fixed location in \( U_2 \), occurs, \( p_1 \) must be read and \( p_2 \) must be written before fetching the next element of the sequence.
The model for the complete action two level memory follows:

1. The two level memory is composed of \( z \) secondary memories, \( U_1 \ldots U_z \), whose capacities are \( C_1 \ldots C_z \), and a primary memory whose capacity is \( M \).
2. The cost of fetching a page from the primary memory is 0. The cost of fetching (writing) a page from (to) \( U_i \) is \( I_i/2 \). (\( i < j \Rightarrow I_i < I_j \)). We denote \( I_i \) by \( I \) and \( I_j \) by \( L \).
3. The two level memory contains \( \sum_{i=1}^{z} C_i \) distinct data pages. Each page has a fixed location in one of the secondary memories. \( M \) data pages are replicated and stored in the primary memory as well.
4. A page-fault, in which page \( p_1 \) is fetched from the secondary memory \( U_i \) and page \( p_2 \) is paged out from primary memory to its fixed location in \( U_j \), costs \( I_i \), if \( i = j \), and \( \text{MAX}\{I_i/2, I_j/2\} \), if \( i \neq j \), as we model the wait period until both the page-read and the page-write operations are complete.

This problem is, in fact, a symmetric \( M \)-server problem; each page is a node, and the length of the (undirected) edge between \( p_1 \) and \( p_2 \) is equal to the cost of paging \( p_1 \) instead of \( p_2 \) (or vice versa). Once again, premature paging can not improve performance.

The sequences used to prove the theorems of subsection 3.1, also prove some lower
bounds on the performance of LRU and FIFO under this model.
The lower bounds obtained for LRU and FIFO under this model are very close to the bounds obtained under the pinned pages variant of the DTLM, while the upper bounds are identical. We note that if there are more than two secondary memories and we can enlarge the lower bounds of LRU and FIFO. For example, the pages $c_1$, $c_2$ that we use in the proof of theorem 3.1.1 could be chosen out of $U_1$, $U_2$ respectively, so the cost of paging in $c_1$ instead of $c_2$ (or vice versa) is $l_2/2$ instead of $l$. Once again, LRU and FIFO are not strongly competitive paging algorithms under the complete action memory, as the lower bounds on their competitive factors depend on the cost of accessing the secondary memories.

6 Additional page frames for the on-line algorithms

An off-line algorithm has the whole fetch sequence in advance and thus can obtain good results in comparison with on-line algorithms. It is interesting to examine the effect of allowing on-line algorithms to use additional primary memory page frames at the expense of secondary memory page frames. More formally, the on-line algorithms use a primary memory of size $M + \Delta$ page frames (while the optimal off-line algorithm uses a primary memory of size $M$). When the generalized two level main memory problem and the optimal off-line demand paging algorithm) is:

$$M_A * L$$

Under the assumption that at least one page is left in $U_a$ under the new scenario, the new lower bound for the performance of an arbitrary on-line demand paging algorithm $A$ (in comparison with the optimal off-line demand paging algorithm) is:

$$\frac{(M - i)^2 * L + i * (M - i) * L}{(M - i + 1) * i}$$

The lower bounds obtained for LRU and FIFO under this model are very close to the bounds obtained under the pinned pages variant of the DTLM, while the upper bounds are identical. We note that if there are more than two secondary memories and we can enlarge the lower bounds of LRU and FIFO. For example, the pages $c_1$, $c_2$ that we use in the proof of theorem 3.1.1 could be chosen out of $U_1$, $U_2$ respectively, so the cost of paging in $c_1$ instead of $c_2$ (or vice versa) is $l_2/2$ instead of $l$.

Once again, LRU and FIFO are not strongly competitive paging algorithms under the complete action memory, as the lower bounds on their competitive factors depend on the cost of accessing the secondary memories.

The generalized two level main memory:

Denote $M_A = M + \Delta$. Under the assumption that at least one page is left in $U_a$ under the new scenario, the new lower bound for the performance of an arbitrary on-line demand paging algorithm $A$ (in comparison with the optimal off-line demand paging algorithm) is:

$$\frac{M_A * L}{(M_A - M + 1) * L}$$

This is also the upper bound for LRU and FIFO.

The pinned pages variant of the DTLM:

The lower bound for any on-line algorithm $A$ is:

$$\frac{2M}{M_{LRU} - M + 1}\,[ST]$$

If LRU may use $\Delta$ extra primary memory page frames (denote $M_{LRU} = M + \Delta$), then the lower bound for LRU is asymptotically:

$$1 + \frac{(M - 1) * L}{(M_{LRU} - M + 1) * L}$$
This is proven similar to the proof of theorem 3.1.1 by constructing a sequence of elements from the set \( \{D_1, \ldots, D_{M-1}, c_2, \ldots, c_{M_{LRU} - M + 2}\} \). The upper bound for LRU in this case is:

\[
M \cdot L \frac{1}{(M_{LRU} - M + 1) \cdot l}.
\]

Similarly, if FIFO may use \( \Delta \) extra primary memory page frames (denote \( M_{FIFO} = M + \Delta \)), then the lower bound for FIFO (see theorem 3.1.4) becomes:

\[
\text{MAX}_{0 \leq i < M} \left\{ \frac{(M_{FIFO} - i) \cdot (M_{FIFO} - i + 1) \cdot l + i \cdot (M_{FIFO} - i) \cdot L}{(M_{FIFO} - i + 1) \cdot (M_{FIFO} - M + 1) \cdot l} \right\}
\]

This is proven similar to the proof of theorem 3.1.4 by constructing for each \( i \), a sequence of elements from the set \( \{D_1, \ldots, D_i, c_1, \ldots, c_{M_{FIFO} - i + 1}\} \). Similar to the proof of theorem 3.1.5, we can prove that the upper bound for FIFO in this case is:

\[
1 + \frac{(M - 1) \cdot L}{(M_{FIFO} - M + 1) \cdot l}.
\]

The np-DTLM:

If we permit the on-line algorithm \( A \) to have \( \Delta \) extra primary memory page frames, at the expense of \( \Delta \) secondary memories' page frames, then the lower bound becomes:

\[
\frac{M_A \cdot l_{K+1}}{(M_A - M + 1) \cdot l}.
\]

This is also the asymptotic upper bound for on-line algorithms of the family \( C_{LRU} (C_{FIFO}) \). Similarly, the upper bound for on-line algorithms of the family \( S_{LRU} (S_{FIFO}) \) is:

\[
\frac{M_A \cdot L}{(M_A - M + 1) \cdot l}.
\]

The tightly coupled two level memory:

The extended lower bounds for LRU is:

\[
1 + \frac{(M - 1) \cdot L}{(M_{LRU} - M + 2) \cdot l}
\]

and the extended lower bounds for FIFO is:

\[
\text{MAX}_{1 \leq i < M} \left\{ \frac{(M_{FIFO} - i) \cdot (M_{FIFO} - i + 2) \cdot l + i \cdot (M_{FIFO} - i) \cdot L}{(M_{FIFO} - i + 2) \cdot (M_{FIFO} - M + 1) \cdot l} \right\}
\]

The complete action memory:

The extended lower bound for LRU is:

\[
\frac{(M_{LRU} - M) \cdot l + (M - 1) \cdot L}{(M_{LRU} - M + 1) \cdot l}
\]

and the extended lower bound for FIFO is:

\[
\text{MAX}_{1 \leq i < M} \left\{ \frac{(M_{FIFO} - i)^2 \cdot l + i \cdot (M_{FIFO} - i) \cdot L}{(M_{FIFO} - i + 1) \cdot (M_{FIFO} - M + 1) \cdot l} \right\}
\]

\( ^{11} X \) is determined according to \( A \)'s second level.
7 Conclusions

We have generalized the paging problem of two level memories [ST], which is a special case of the symmetric server problem [MMS]. In the generalized problems, the first level (called primary memory) contains $M$ page frames, and the second level is composed of a collection of different speed secondary memories. We have presented several versions for the generalized problem; each models aspects of a practical system, for example, a generalized two level main memory, a Distributed Two Level Memory (DTLM for short)\textsuperscript{12}, a tightly coupled two level memory, and a complete action two level memory. We have investigated the competitiveness of deterministic on-line demand paging algorithms for these generalized paging problems. Then we extended the results for the case in which the on-line algorithms may use additional primary memory page frames.

The demand paging problem in a generalized two level main memory and in the np-DTL, are not task systems, and thus premature paging can improve performance. For these two problems we have obtained tight lower bounds for the performance of any on-line demand paging algorithm and showed that these bounds are also the upper bounds for LRU and FIFO.\textsuperscript{13} Thus, LRU and FIFO are strongly competitive on-line demand paging algorithms under these models.

The paging problems of the pinned pages variant of the DTLM and of the complete action memory, can be both regarded as symmetric server problems [MMS]. In these problems, premature paging can not improve performance. We proved lower bounds and upper bounds (which are close) for LRU and FIFO. The lower bounds depend on the costs of accessing the secondary memories. Thus LRU and FIFO are not strongly competitive with respect to these models.\textsuperscript{14}

Thus for two models, LRU and FIFO are the optimal on-line algorithms. An open question is that of finding deterministic polynomial time optimal off-line algorithms for these generalized paging problems.

References


\textsuperscript{12}We have considered two variants of the DTLM. In the first variant, called the pinned pages variant of the DTLM, each page has a fixed location in one of the secondary memories. In the second variant, denoted by np-DTLM, a page that is paged out of primary memory may be stored in any free page frame in one of the secondary memory.

\textsuperscript{13}With respect to the np-DTLM, this statement holds for LRU-like (FIFO-like) on-line algorithms which always page-out to the "cheapest" secondary memory having free page frames.

\textsuperscript{14}We note that [RS] considered a generalized cache problem which is identical to the pinned pages variant of the DTLM problem. They found a randomized on-line algorithm which is $M$-competitive (in a "weak randomized sense") for this problem.
Appendix

We prove that LRU (FIFO) faults at most $M$ times more than each optimal off-line demand paging algorithm, with respect to all the models considered in this paper. The proof is essentially identical to the proof of theorem 6 in [ST].

Theorem: Consider one of the paging problems considered in this paper, and let BEST be an optimal off-line demand paging algorithm for it. For $M > 1$, if both LRU (FIFO) and BEST initially store the same set of pages in primary memory and in each secondary memory, then for any sequence of fetches, of any length, LRU (FIFO) faults at most $M$ times more than BEST.

Proof: Given a sequence of fetches, we execute LRU (FIFO) on it, and we mark each sequence-element that caused a page-fault. We then partition the sequence to sub-sequences, starting at the end of the sequence, so that each sub-sequence (besides the sub-sequence that includes the first element of the sequence, to which we refer as the "first sub-sequence") contains exactly $M$ marked sequence-elements (i.e. elements that caused page-faults for LRU (FIFO)), and such that the element just before the beginning of the sub-sequence, is also marked. Thus the first sub-sequence may contain less than $M$ marked elements, but no more than $M$.

Consider the first sub-sequence. As both LRU (FIFO) and BEST initially store the same set of pages in primary memory, the first element that causes LRU (FIFO) a page-fault, also causes BEST a page-fault.

Consider now the $i$-th sub-sequence. Let $p_k$ be the element just before the first element of the $i$-th sub-sequence. Thus $p_k$ is in primary memory of both LRU (FIFO) and BEST at the beginning of the $i$-th sub-sequence. If $p_k$ appears in the $i$-th sub-sequence as a marked element, or if there exists a page $p_l$ that appears twice in the $i$-th sub-sequence as a marked element, then the definition of LRU implies that there are at least $M$ additional distinct pages (which are different than $p_k$), that appear in the $i$-th sub-sequence (before the appearance of $p_k$ or between the two appearances of $p_l$). If neither of the two cases above occur, then all the marked elements are distinct pages which are not $p_k$. (Note that the first two cases are possible only for LRU, because in FIFO, a page that was just paged-in will be paged-out only after $M$ page-faults, and because each sub-sequence contains only
Thus in all three cases, \( M \) distinct pages which are not \( p_k \), appear in the \( i \)-th sub-sequence. Therefore, BEST must perform at least one page-fault on an element of the \( i \)-th sub-sequence, as it cannot store in its memory more than \( M - 1 \) distinct pages, each different than \( p_k \), at the beginning of the \( i \)-th sub-sequence. Therefore, BEST makes at least one page-fault for each \( M \) page-faults of LRU (FIFO). \( \square \)