On the Complexity of Coordinated Display of Multimedia Objects*

Martha L. Escobar–Molano	Shahram Ghandeharizadeh
Computer Science and Engineering	Computer Science Department
University of South Florida	University of Southern California
Tampa, FL 33620, U.S.A.	Los Angeles, CA 90089, U.S.A.
mescobar@csee.usf.edu	*shahram@cs.usc.edu*

Abstract

A multimedia presentation can be represented as a collection of objects with temporal constraints that define when the objects are rendered. The display of a presentation is termed *coordinated* when the display of its objects respects the pre-specified temporal constraints. Otherwise, the display might suffer from failures that translate into meaningless scenarios. For example, a chase scene between a dinosaur and a jeep becomes meaningless if the system fails to render the dinosaur when displaying the scene.

A previous study [EMGI96] introduced a resource scheduling technique that guarantees a coordinated display of a presentation for single-disk architectures. This technique minimizes both latency and memory requirements and has a worst case time complexity $\mathcal{O}(n \lg n)$. This paper extends the previous study to multi-disk architectures and demonstrates the following: (1) the computation of a resource schedule that supports a coordinated display and yields the minimum latency is NP-Hard, and (2) the computation of the minimum memory requirements to support a coordinated display within a pre-specified latency is NP-Hard.

Keywords: Multimedia, Storage Management, Resource Scheduling, Non-textual Data Types, Complexity.

*This research was supported in part by NSF grants IRI-9222926, IRI-9258362 (NYI award), and CDA-9216321, and a unrestricted cash/equipment gift from Hewlett-Packard.

To appear in Theoretical Computer Science.

Figure 1: System Architecture

1 Introduction

Multiple media types are becoming widely used in a variety of fields such as medicine, education, science, and entertainment. A challenging task is the storage management of non-textual media types (video, audio, animations, graphics, images). These media types might require high volumes of storage, high bandwidth and continuous display. Furthermore, media objects might be synchronized and shared by several presentations. For example, consider a presentation where the display of a computer generated animation is overlaid on a video clip. This animation consists of 3D representations of two dinosaurs, each dinosaur's positions on the screen and the times of their appearances, and rendering features such as light intensity and viewpoint. The video clip consists of a sequence of frames that must be displayed at a pre-specified rate (30 frames per second). The display of the video clip must be synchronized with that of the animation, which in turn must satisfy the temporal constraints specified by each dinosaur's times of appearance. Also, the video clip and the 3D representations of a dinosaur might be shared independently by several presentations.

We assume a hardware architecture (Figure 1) consisting of D disks, and a limited amount of memory. All multimedia objects reside on disk. When a user requests a presentation, the storage manager retrieves participating objects from disk into memory so that they can be transmitted over the network and presented to the user according to their temporal constraints (e.g., times of a dinosaur's appearance, first frame's time of display). To illustrate, suppose that both transmission time over the network and time to render a 3D object on the screen are negligible. Then the storage manager must have in memory frame 1 during the first $\frac{1}{30}$ seconds, frame 2 during the second $\frac{1}{30}$ seconds, and so on. The 3D representation of a dinosaur must also be in memory during its times of appearance.

This paper studies the complexity of scheduling the delivery of objects from disk to memory according to their temporal constraints. This problem (termed *resource scheduling*) is fundamental for a storage management system that supports multimedia applications such as computer animation and coordinated use of multiple streams (e.g., video, audio) in multimedia documents.

An unbalanced placement of objects across the disks with respect to a presentation might cause a subset of disks to become bottlenecks. The objects referenced by the presentation during a period might reside in a subset of disks. These disks might become bottlenecks during the display of the presentation. The system can detect these bottlenecks in advance because when a request arrives, the identity of objects participating in the presentation, their temporal constraints, and their data placement on disk are pre-specified. A resource schedule might manipulate the data placement to avoid these bottlenecks. To change the data placement, the scheduler can replicate or migrate objects from one disk to another.

A resource schedule that supports a coordinated display of a presentation consists of retrievals, migrations, and replications. Once the display starts, the schedule must render objects memory resident according to their temporal constraints. To satisfy these constraints, the system can either retrieve the objects immediately before their display, pre-fetch them at an earlier time and have them memory resident until they are displayed, or manipulate the data placement to retrieve them from disks different from their original location.

In Section 2, we present the formal statement of the problem. Section 3 describes related studies. Section 4 shows that deciding whether there is a schedule to change the placement of objects across disk drives based on a system load is NP-Complete. Section 5 proves that the computation of a resource schedule that supports a coordinated display and yields the minimum latency is NP-Hard; and the computation of the minimum memory requirements to display a presentation within a pre-specified latency is NP-Hard. In Section 6, we present conclusions and future research directions.

2 Statement of the Problem

To simplify the discussion, assume that the unit of transfer from each disk is fixed-sized and termed a page. This assumption will be removed later on to generalize the results to variable-sized units of transfer between memory and disk. An object might be either smaller or larger than a page. When an object x is larger than a disk page, it is represented as a collection of pages. It is first partitioned into $k = \lceil \frac{size\ of\ x}{size\ of\ a\ page} \rceil$ disk pages. Subsequently, these pages are assigned to the D disks. The system may cluster several small objects in a single disk page. Each disk can perform independent reads/writes. The D disk drives may retrieve D pages into D different memory frames at the same time. Another assumption is that each disk has enough space to accommodate pages migrated/replicated to the disk during the display of a presentation in addition to those pages residing in the disk before the display.

We discretize time into fixed-sized units, termed *time intervals*. The duration of each time interval is denoted as t. The beginning of a time interval i is termed *time instant* i (Figure 2). When a user requests a presentation, the system has advance knowledge of the identity of pages that should be memory resident at specific times to support its display. This schedule is termed a *display schedule*:

Figure 2: Time interval and time instant.

Definition: A *display schedule* is a sequence $\{P_0, \ldots, P_{m-1}\}$ of disk page sets. Where m is the duration of the presentation in time intervals, and P_i is the set of pages displayed during interval i.

To minimize the observed startup latency and the required amount of memory, a resource schedule overlaps the display and the retrieval of disk pages and manipulates the placement of data on disks. To illustrate, suppose that the display schedule consists of disjoint set of pages (i.e., $P_i \cap P_j = \emptyset$, for $i \neq j$). Ideally, the collection of pages that constitute P_i should be retrieved into memory during time interval $i-1$. This would minimize the amount of required memory. However, this ideal situation might be infeasible at times because the pages that constitute P_i might be unevenly dispersed across the disks, exhausting the bandwidth of one or more disks (while other disks are idle) such that they fail to retrieve the set P_i during a time interval. Note that in this scenario the total bandwidth of the disks is sufficient, the primary limitation is the placement of data in combination with the display schedule that results in formation of bottleneck disks. The system may pursue two alternative solutions to resolve bottlenecks: (1) retrieve some pages of P_i during earlier time intervals, $i-2$, $i-3$, ..., etc., (these pages are termed *pre-fetched* pages), or (2) manipulate the placement of data prior to time interval i so that P_i is more evenly distributed across disks. A resource schedule to support a coordinated display of a structured presentation might have three components: (1) page retrievals from disks to memory, (2) page writes to change the data placement, and (3) page discards (from memory) to accommodate new retrievals. If the system resolves bottlenecks with pre-fetches only, the second component becomes unnecessary. Otherwise, reads from the first component and writes from the second component specify the migrations/replications to manipulate the *placement of data*. Formally, the placement of data is defined as a mapping from a page identifier and a time interval into one or more disk drives.

The set of pages occupying memory frames at instant i (S_i) is defined based on the set of pages that occupy memory at instant $i-1$ (S_{i-1}), those discarded from memory (K_i), those flushed to disks (U_i^d), and those retrieved from different disks (F_i^d). The number of pages in S_i should be lower than or equal to the C buffers that constitute the memory.

Definition: Given a system with D disks, the state of memory at each instant i is defined as:

$$S_i = (S_{i-1} - K_{i-1} - (U_{i-1}^0 \cup \ldots \cup U_{i-1}^{D-1})) \cup (F_{i-1}^0 \cup \ldots \cup F_{i-1}^{D-1})$$

Formally, a resource scheduler consumes a display schedule $\{P_0, \ldots, P_{m-1}\}$, a system configuration (B, C, D), and a placement of data, $\mathcal{P}$, to compute a schedule of page discards, writes, and retrievals that satisfy the temporal constraints dictated by the display schedule.

Definition: Given a system with C memory buffers, D drives each with sufficient disk bandwidth to retrieve/write B pages during a time interval, a display schedule $\{P_0, \ldots, P_{m-1}\}$, an initial state of memory S_{-p}, and an initial placement of data $\mathcal{P}$, a *resource schedule* consists of $p+m$ time intervals: m of these overlap with the display, and p of them either pre-fetch pages into memory or modify the placement of data across the disks in preparation for the display. In essence, p denotes the incurred startup latency. Associated with each time interval i are:

(1) a collection of pages retrieved from each of the D disks during time interval i, denoted as $F_i^0, \ldots, F_i^{D-1}$,

(2) a collection of pages written to each of the D disks during time interval i, denoted as $U_i^0, \ldots, U_i^{D-1}$,

(3) a collection of pages discarded from memory to accommodate these retrievals, denoted as K_i.

Furthermore, the retrieved, written, and discarded pages must satisfy the following constraints:

(i) Once the display starts, the set of pages in memory at each instant i is a superset of those required by the display schedule: For each $i \in [0, m-1]$, $P_i \subseteq S_i$ and $P_i \subseteq S_{i+1}$.

(ii) The number of pages retrieved and written to a disk during a time interval does not exceed B: For each $i \in [-p, m-1]$ and each $d \in [0, D-1]$, $|F_i^d| + |U_i^d| \leq B$.

(iii) The number of memory resident pages at each time instant is lower than the number of available memory frames: For each $i \in [-p, m]$, $|S_i| \leq C$.

(iv) The retrievals respect the placement of data: For each page a, interval i, and disk d: $a \in F_i^d$ implies that $d \in \mathcal{P}'(a, i)$, where $\mathcal{P}'$ is the placement of data resulting from updating $\mathcal{P}$ with migrations and replications scheduled prior to interval i.

If there is no manipulation of the data placement (i.e., for each interval i and drive d, $U_i^d = \emptyset$), then we have a *retrieval schedule*.

To illustrate these concepts, consider a display schedule for three time intervals: $P_0=\{a,b\}$, $P_1=\{c,d\}$, and $P_2=\{e,f\}$. Assume that the system consists of two disks (D=2), each with the bandwidth to retrieve one disk page during a time interval (B=1). Assuming that all referenced pages reside on disk one, Figure 3(a) shows a retrieval schedule that supports a coordinated display. In this figure, a negative time instant corresponds to page retrievals performed prior to the display. A page might either be retrieved during the time interval prior to its display (e.g., f is retrieved at interval 1 and displayed at interval 2) or pre-fetched at an earlier time interval (e.g., a is retrieved at interval -4 and displayed at interval 0). Pre-fetching increases the memory requirements of the system. For example, 5 frames of memory are allocated at instant one (a, b, c, d, e) while the display schedule dictates that only four should be allocated (a, b, c, d). The other page, e is pre-fetched for later use and increase the memory requirements of the system.

As illustrated by this example, an unbalanced schedule of references to disks might result in formation of bottleneck disks that requires the system to pre-fetch pages while other disks remain idle. In our example, while the bandwidth of two disks could accommodate the retrieval of two pages, the system was forced to pre-fetch pages because they all reside on disk one. The scheduler may construct resource schedules that utilize the idle disk bandwidth in order to minimize the number of pre-fetched pages. Figure 3(b) shows one such schedule. With this schedule, the system reads page e from disk one during time interval -4 and replicates or migrates it to disk zero during time interval -3 ($U_{-3}^0 = \{e\}$). This allows the system to free the memory frame occupied by e at time instant -2 and, utilize disk zero to retrieve e during time interval one to satisfy the display schedule. With this schedule, only 4 memory frames are required at instant one (a, b, c, d).

The schedule for migrations and replications depends on the available disk bandwidth and memory during each time interval, which in turn depends on the system load.

Figure 3: (a) Retrieval Schedule. (b) Resource Schedule. Pages in memory inside an oval are those required by the display schedule at that instant.

Definition: The *system load* for a period $[0, N]$ is defined as a sequence $[a_0, \ldots, a_{N-1}]$ of records consisting of $(D+1)$ attributes that specifies the availability of system resources during each time interval. Each record a_i correspond to time interval i, $0 \leq i \leq N-1$, and consists of the following attributes $< B_i^0, \ldots, B_i^{D-1}, M_i >$; where B_i^d is the disk bandwidth available at drive d during interval i and M_i is the number of page frames available at instant i.

A request to change the placement of a page is denoted as $(a, source \rightarrow \{target_1, \ldots, target_n\})$, where a is the disk page to migrate/replicate, $source$ is a disk drive that contains a, and $\{target_1, \ldots, target_n\}$ are alternative drives to contain a. To migrate/replicate a page, the system can utilize intermediate disk drives. For example, consider the migration $(a, 5 \rightarrow \{2, 4, 6\})$ to be scheduled in a system with 10 disk drives ($D = 10$). One possible schedule (Figure 4) reads a from disk 5 during interval 1 and writes it to disk 7 during interval 2. Next, it reads a from disk 7 during interval 4 and writes it to disk 8 during interval 6. Subsequently, it reads a from disk 8 during interval 8 and writes it to disk 6 during interval 9. The advantage of using intermediate disks (e.g., disks 7 and 8) is that it reduces the memory requirements when there is insufficient bandwidth to accommodate the writing of a on disks $2, 4$, or 6. Using disks 7 and 8 as intermediate disks prevented the system from staging a in memory during intervals $[3, 4]$ and $[7, 8]$.

To simplify the discussion, assume that the system does not allow to have replicas of a page on disk. This limitation forces the scheduler to consider migrations as the only alternative to manipulate the data placement. We will remove this assumption later on to generalize the results to systems that allow both migrations and replications as alternatives to change the placement.

Definition: Given a collection M $(\{m_1, \ldots, m_r\})$ of requests to change the data placement on a system load A for a period $[0, N]$, a schedule of migrations for M on A maps each request $m_i = (a, source \rightarrow$

Figure 5: Schedule of migration m_i.

$\{target^i_1, \ldots, target^i_{n_i}\}$) into a sequence (Figure 5) $\{(source, t^i_0)$, $(d^i_1, [t^i_1, t^i_2])$, …, $(d^i_{k_i}, [t^i_{2k_i-1}, t^i_{2k_i}])$, $(target^i_p, t^i_{2k_i+1})\}$ where k_i is the number of intermediate disks. Such that:

(i) There is disk bandwidth available in *source* during interval t^i_0 to read a.

(ii) For each $j \in [1, k_i]$, disk d^i_j has sufficient bandwidth to write a during interval t^i_{2j-1} and to read a during interval t^i_{2j}.

(iii) There is disk bandwidth available to write a on disk $target^i_p$ during interval $t^i_{2k_i+1}$.

(iv) The migrations are scheduled within the period $[0, N]$ and the reads and writes are scheduled in the right order: $0 \leq t^i_0 < t^i_{2k_i+1} \leq N$ and: (a) for each $j \in [1, 2k_i]$, $0 \leq t^i_0 < t^i_j < t^i_{2k_i+1} \leq N$, and (b) $j < l$, for $l \in [1, 2k_i]$, implies $t^i_j < t^i_l$.

(v) Page a is migrated to one of the target disks: $p \in [1, n_i]$.

(vi) There is memory available to have the page memory resident during the following periods: $(t^i_0, t^i_1]$, $(t^i_2, t^i_3]$, …, $(t^i_{2k_i}, t^i_{2k_i+1}]$.

(vii) The intermediate drives are different from the source and target drives, otherwise the system would be performing wasteful work: for each $j \in [1, k_i]$, $d^i_j \notin \{source, target^i_1, \ldots, target^i_{n_i}\}$

3 Related Work

Several researchers have studied the complexity of scheduling problems [GJ75, GJS76, BGJ77]. Their studies assume a pre-defined number of jobs and tasks with specific resource requirements and duration. In contrast, the resource requirements and duration of the jobs and tasks are not pre-defined for a resource schedule that supports a coordinated display. To render an object memory resident, the system might either retrieve the object directly from the disk containing it or manipulate the placement of data so that the object is retrieved from another disk. The placement of data can be manipulated with either replications or migrations of disk pages. A replication or migration of a page might incur several I/Os before reaching its destination. For instance, consider the migration of an object from disk d_1 to disk d_3. The system might migrate the object directly from d_1 to d_3 (one step), or migrate the object from d_1 to d_2, and then from d_2 to its final destination d_3 (two steps). In the latter case, disk d_2 is used as an intermediate disk. Furthermore, the object must be memory resident between consecutive reads and writes during either a migration or a replication step (e.g., between *read from* d_1 and *write to* d_2). The number of steps, the time elapsed between the read and the write of each step, and which disk drives are used at each step are not pre-defined. Whether we consider all steps required to bring a page into memory as a job and a each step as a task or we consider each step as a job, the resource requirements and duration of the jobs and tasks are not pre-defined. Moreover, the scheduling problems in [GJ75, GJS76] consider an overall deadline as opposed to individual deadlines as in the case of our resource scheduling problem. Also, the scheduling problems in [GJS76, BGJ77] are not resource constrained, while our schedules are constrained by the amount of memory.

This paper studies resource schedules that ensure a coordinated display of a presentation while minimizing the latency. One resource managed by this scheduler is memory. Several caching studies attempt to minimize the number of misses scored by references to pages that are not resident in a fixed-sized cache. Optimal strategies have been presented for the case where the entire schedule of page references is known [CR93]; and competitive online algorithms have been studied for online variants of this problem in which an unknown sequence of references is generated by an adversary [ST85] or by a Markov process. In addition, these results have been generalized and elaborated to deal with data placement in distributed systems and file migration [MS91]. However, these studies have not considered temporal constraints such as the one that supports a coordinated display. The resource schedulers described in this study do not aim to minimize the number of page faults. Instead, they strive to assure a coordinated display while minimizing the latency observed by the user.

For a single-disk architecture, there is an optimal resource schedule that supports the coordinated display of a presentation [EMGI96]. This optimal schedule minimizes both the memory requirement at each instant and the latency and can be computed in time $O(n \lg n)$.

The complexity of transferring data from one site to another has been studied before in the context of networks. Several researchers [Whi90, RVVN92, JGJL85] have studied the problem of scheduling file transfers between nodes in a network that minimizes overall finishing time. We studied the problem of scheduling transfers (migrations/replications) between disks. In [JGJL85], forwarding is not allowed; each file is transferred directly from the source node to the target node. On the other hand, we allow a migration schedule to utilize intermediate disks. In [Whi90, RVVN92], a transfer might utilize intermediate nodes. However, these studies assume that every file takes constant time to move directly (without intermediate nodes) from one node to another; while, in our case, the time to transfer a page directly from one disk to another may vary. The time elapsed between two consecutive read and write operations in a migration schedule is not constant. Moreover, unlike the studies in [Whi90, RVVN92, JGJL85], the migration scheduling problem is constrained by a central resource: memory.

Figure 6: Alternative schedules for the migration associated with v_i and schedules associated with clauses $C_1, \ldots, C_n$ that conflict with those for v_i.

4 Data Placement Manipulation

To show that deciding whether there is a schedule of migrations for a set M of requests based on a system load A defined over a period $[0, N]$ is NP-Complete, we reduce SAT to this decision problem. An instance of SAT is defined as a collection $\{C_1, \ldots, C_n\}$ of n clauses over a set $\{v_1, \ldots, v_k\}$ of k variables. The SAT problem is deciding whether there is a variable assignment that makes all clauses true. Without loss of generality assume that there is not a clause in the SAT instance with both v_i and $\neg v_i$ as its disjuncts (if this is the case, remove such clauses because they are true for any truth assignment).

We first introduce a polynomial algorithm *SAT2MigSc* that transforms any instance $C_1, \ldots, C_n, v_1, \ldots, v_k$ of SAT into an instance M, A, N of the migrations scheduling problem, i.e., is there a schedule of migrations for requests M on a system load A defined over $[0, N]$? This algorithm associates one migration request with each variable and each clause in the instance of SAT, resulting in a set M of $k+n$ requests. In addition, this algorithm computes a system load A such that the migration request associated with variable v_i has only two alternative schedules on A and the migration request associated with clause C_j has exactly l alternative schedules on A, where l is the number of disjuncts in C_j. Furthermore, the system load A is defined over the period $[0, N]$ which is divided into k sub-periods of $4 \cdot (n+1)$ time intervals. Each sub-period corresponds to the time when the replication associated with a variable can be scheduled.

Figure 6 shows the sub-period associated with variable v_i. Thick lines represent time instants when the memory capacity is exhausted (0 memory frames available at that instant). While thin lines represent time instants when the memory has 1 memory frame available at that instant. The migration for v_i can be scheduled either during the first $2 \cdot (n+1)$ time intervals or during the last $2 \cdot (n+1)$ intervals. The first alternative corresponds to assigning *false* to v_i in SAT and the second to assigning *true* to v_i. If v_i is a disjunct in C_j, then one possible schedule for C_j would be during two consecutive time intervals $(q + 2 \cdot (j-1)$ and $q + 2 \cdot (j-1) + 1)$ in the first half of the sub-period. This schedule for C_j conflicts with the schedule for v_i during the first half because there is only one memory frame available at the time instant $(q + 2 \cdot (j-1) + 1)$ between the two consecutive time intervals of the schedule for C_j and two memory frames are required (one for v_i and the other for C_j). This conflict corresponds to assigning false to v_i, which would not make C_j true. Symmetrically, if $\neg v_i$ is a disjunct in C_j, then one possible schedule for C_j would be during two consecutive time intervals in the second half; which conflicts with the schedule for v_i during the second half

Figure 7: Example of reduction of SAT instance: $C_1 = v_1 \vee \neg v_2 \vee v_3$ and $C_2 = v_1 \vee v_2 \vee \neg v_3$, into a migration scheduling instance. Dashed time instant 0 represents a memory with one or more available frames.

of the sub-period.

To illustrate, consider the example shown in Figure 7. *SAT2MigSc* will output five migration requests (one for each variable and one for each clause). Each migration request has more than one alternative schedule on A. There are two alternatives to schedule a migration for v_1: the first one spans time intervals 0 to 5 and the second one time intervals 6 to 11. Similarly, alternative schedules for v_2 and v_3 span intervals 12 to 17 and 18 to 23, and intervals 24 to 29 and 30 to 35, respectively. There are three alternatives to schedule a migration for C_1: the first one spans time intervals 0 and 1, the second one spans 18 and 19, and the third one 24 and 25. Similarly, alternative schedules for C_2 span intervals 2 and 3, 14 and 15, and 32 and 33.

A possible schedule for all the requests is to migrate v_1 during intervals 6 to 11, v_2 during intervals 12 to 17 (or 18 to 23), v_3 during intervals 24 to 29 (or 30 to 35), C_1 during intervals 0 and 1, and C_2 during intervals 2 and 3. Another possible schedule is to migrate v_1 during intervals 0 to 5, v_2 during intervals 12 to 17, v_3 during intervals 24 to 29, C_1 during intervals 18 and 19, and C_2 during intervals 32 and 33. Notice that the spans of the schedules are disjoint.

Formally, the algorithm SAT2MigSc is defined as follows:

SAT2MigSc

Input: $C_1, \ldots, C_n, v_1, \ldots, v_k$

Output: M, A, N

Method: $N = (2 \cdot n + 2) \cdot 2 \cdot k$

Let A be as described below.

Create one migration request for each variable and each clause:

$$M = \{(v_i, s_i \rightarrow \{t_i, u_i\}) \mid 1 \leq i \leq k\} \cup \{(C_j, d_j \rightarrow (\{d_{ji} \mid v_i \in C_j\} \cup \{e_{ji} \mid \neg v_i \in C_j\})) \mid 1 \leq j \leq n\},$$

where pages v_i and C_j are distinct and different from those in memory at instants $0, 1, \ldots, N$

The system load A computed by $SAT2MigSc$ is defined in Figure 8. Thick time instants denote that

Figure 8: System load yielded by SAT2MigSc.

the system has 0 memory frames available at that instant:

$$\text{For } i \in \{2, 4, 6, \ldots, N\},\ M_i = 0$$

Dashed time instants denote that the system has at least one memory frame available at that instant (instant 0 in Figure 8):

$$M_0 > 0$$

Thin time instants (Figure 8) denote that the system has only 1 memory frame available at that instant:

$$\text{For } i \in \{1, 3, 5, \ldots, N-1\},\ M_i = 1$$

Labels $w_i, s_i, d_i, t_i, u_i, d_{ij}, e_{ij}$ on time intervals (Figure 8) denote disk drives with available bandwidth during the interval. A superscript $+$ denotes that the disk has available bandwidth to retrieve/write at least one page. A label without a superscript denotes bandwidth availability for exactly one page retrieval/write. A superscript c indicates that the bandwidth availability depends on a condition of the form $v_i \in C_j$ or $\neg v_i \in C_j$, which denotes that v_i and $\neg v_i$ are disjuncts of C_j respectively. If the condition is satisfied, then the disk has sufficient bandwidth for one retrieval/write; otherwise, the disk does not have bandwidth available during the interval. To illustrate consider interval 2, there is disk bandwidth available in drives w_3 and d_{11}. If $v_1 \in C_2$, then disk d_2 also has bandwidth available during this interval. The other disk drives do not have bandwidth available during this interval. The system load for interval 2 is as follows: $M_2 = 0$; for each $j \notin \{w_3, d_{11}, d_2\}$, $B_2^j = 0$; $B_2^{w_3} > 0$; $B_2^{d_{11}} = 1$; if v_1 is a disjunct in C_2 then $B_2^{d_2} = 1$, otherwise $B_2^{d_2} = 0$.

The set of pages to migrate (in M) and the set of pages in memory that yield the system load A are disjoint. Also, all requests in M are for different pages. Therefore, the identity of the disk page in a migration schedule is irrelevant. We thus omit it to simplify the notation.

□

To illustrate the transformation consider the example in Figure 9. Note that the drives w_j cannot participate in any schedule for a migration because they are neither a source nor a target of a migration. Moreover, they cannot be intermediate drives because they have bandwidth available only during one time interval. Schedules of migrations compete with each other for disk bandwidth and memory. For example, $\{(d_1, 0), (d_{11}, 1)\}$ (an alternative migration schedule for the request associated with C_1) competes with $\{(s_1, 0), (d_{11}, [1, 2]), (d_{21}, [3, 4]), (t_1, 5)\}$ (an alternative migration schedule for the request associated with v_1) for bandwidth of d_{11} at interval 1 and for memory at instant 1. Intuitively, a variable assignment that makes v_1 true is equivalent to schedule the migration associated with v_1 during intervals 6 to 11. This assignment also makes C_1 true. Moreover, the migration associated with C_1 can be scheduled during the intervals 0 and 1 because it would neither compete for disk bandwidth nor for memory with the schedule for v_1.

To prove that the reduction from SAT to the migrations scheduling problem is correct, we start by showing that if an instance of SAT has a solution then the corresponding migrations scheduling instance has a solution.

Lemma 4.1: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. If there is a truth assignment for variables $\{v_1, \ldots, v_k\}$ that makes all clauses $C_1, \ldots, C_n$ true, then there is a schedule of migrations for M on A during interval $[0, N]$.

Figure 9: Example of reduction of a SAT instance into a migrations scheduling instance. (a) SAT instance, (b) Corresponding set M of migrations requests, (c) Corresponding system load A.

Proof: See Appendix A. $\square$

We now prove the other direction, if the migrations scheduling instance yielded by $SAT2MigSc$ has a solution then the input SAT instance has a solution.

Lemma 4.2: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. If there is a schedule of migrations for M on A, then there is a truth assignment for $\{v_1, \ldots, v_k\}$ that makes all clauses $\{C_1, \ldots, C_n\}$ true.

Proof: See Appendix A $\square$

We now generalize the results to have both migrations and replications as alternatives to manipulate the data placement.

Lemma 4.3: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. There is a schedule of migrations for M on A if and only if there is a schedule of replications for M on A.

Proof: The distinction between a migration and a replication is the availability of the page after it is written to an intermediate disk. For a replication, the system can retrieve the page from the source disk or from any of the previous intermediate disks. For a migration, the system must retrieve the page from the preceding intermediate disk. To prove the if direction is straightforward, we thus omit it.

For the other direction, if there is a replication schedule that retrieves a page a during interval t from a disk d different from the preceding intermediate disk, then d must be either the source or another previous intermediate disk. In that case, the I/O operations of this schedule follows this sequence[1]: $R^a_{source}, W^a_{d_1}, \ldots, R^a_{d_{i-1}}, W^a_{d_i}, R^a_{d_i} \ldots, R^a_{d_{j-1}}, W^a_{d_j}, R^a_d, W^a_{d_{j+1}}, \ldots$; where $d = source$ or $d = d_i$. Then, the system could have avoided the I/O operations after the previous write to d (or from the beginning, if d is the source) and before t. The schedule resulting after removing these superfluous I/O operations would have the form: $R^a_{source}, W^a_{d_1}, \ldots, R^a_{d_{i-1}}, W^a_{d_i}, R^a_{d_i}, W^a_{d_{j+1}}, \ldots$; or $R^a_{source}, W^a_{d_{j+1}}, \ldots$. Given a replication schedule with superfluous I/O operations, consider the schedule resulting from removing the superfluous I/O operations as described above. It is easy to see that the schedule without superfluous operations retrieves a page from the preceding intermediate disk (except for the first retrieval).

If after writing a page to an intermediate disk, the replication schedule always retrieves the page from this disk. Then, the system could have scheduled a migration instead. Therefore, if there is a schedule of replications for M on A then there is a schedule of migrations for M on A. $\square$

With Lemmas 4.1, 4.2, and 4.3, we conclude the following.

Lemma 4.4: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. There is a truth assignment for variables $\{v_1, \ldots, v_k\}$ that makes all clauses $C_1, \ldots, C_n$ true if and only if there is a schedule of migrations/replications for M on A during interval $[0, N]$.

It is easy to see that the time complexity of transformation $SAT2MigSc$ is polynomial. This transformation is also a valid reduction from SAT to the problem of deciding whether is possible to change the data placement on a system load (Lemma 4.4). Therefore:

[1] R^a_j denotes that disk page a is read from disk j and W^a_j that page a is written to disk j.

Theorem 4.5: Deciding whether there is a schedule to replicate/migrate fixed-sized units on a system load A over a period $[0, N]$ is NP-Complete.

The problem of scheduling replications/migrations of variable-sized units includes all instances of this scheduling problem for fixed-sized units. Since the latter problem is NP-Complete, then the first one is also NP-Complete.

Corollary 1 *Deciding whether there is a schedule to replicate/migrate objects on a system load A over a period $[0, N]$ is NP-Complete.*

5 Resource Scheduling

This section demonstrates that: (1) the computation of a resource schedule that supports a coordinated display of a presentation and yields the minimum latency is NP-Hard, and (2) the computation of the minimum memory requirements to display a presentation within a pre-specified latency is NP-Hard. It suffices to show that deciding whether there is a resource schedule for a given display schedule that yields a one-time-interval latency is NP-Complete. We reduce SAT into this decision problem. We first introduce a polynomial algorithm *SAT2ResSc* that transforms any instance $C_1, \ldots, C_n, v_1, \ldots, v_k$ of SAT into an instance $\{P_0, \ldots, P_{m-1}\}, \mathcal{P}, B, C, D$ of the resource scheduling problem, i.e., is there a one-time-interval-latency resource schedule that satisfies the display schedule $\{P_0, \ldots, P_{m-1}\}$ on a system with D disk drives each with bandwidth B and memory capacity C, assuming an initial data placement $\mathcal{P}$? We then show that given an instance $C_1, \ldots, C_n, v_1, \ldots, v_k$ of SAT, $SAT2ResSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$ has a solution if and only if $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$ has a solution. Then, because of Lemma 4.4, an instance $C_1, \ldots, C_n, v_1, \ldots, v_k$ of SAT has a solution if and only if $SAT2ResSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$ has a solution.

We now introduce the algorithm *SATResSc*. This algorithm first computes the instance $M, A, N =$ SAT2MigSc$(C_1, \ldots, C_n, v_1, \ldots, v_k)$. Based on this instance, it computes the system configuration: Number of disks (D), maximum number of pages retrieved/written into a disk during a time interval (B), and memory capacity in pages (C). It sets D to the number of disks with available bandwidth in A plus one (w_0, the only drive with available bandwidth during interval -1), B to the maximum number of disjuncts in a clause plus one, and C to $2 \cdot B \cdot (D-1)$. Then, it computes the display schedule as follows: It sets the duration (m) of the display to $N + n + k$ and the number (q) of pages to be displayed during each time interval to $\frac{C}{2}$. It sets the pages displayed at each time interval (Column P_i in Table 2) so that the pages that must be in memory during the display (Column $\hat{S}_i$ in Table 2) would exhaust the memory capacity at even instants between 2 and N and at all instants after N. Also, these pages ($\hat{S}_i$) would require $C - 1$ memory frames at odd instants between 1 and N, leaving only one memory frame available for pre-fetches or migrations. For example, pages displayed during intervals 1 and 2 must be in memory at instant 2 to satisfy the display schedule ($\hat{S}_2 = a_q \ldots a_{3q-1}$). Therefore, the display schedule demands $2 \cdot q = C$ memory frames at instant 2 exhausting the memory capacity.

Next, *SAT2ResSc* computes the placement $\mathcal{P}$ on disk of all pages referenced by the display schedule such that the system neither has to pre-fetch nor migrate pages to satisfy the display schedule from interval 0 to $N-1$. For each $i \in [0, N-1]$, all pages in $\hat{S}_{i+1} - \hat{S}_i$ can be retrieved during interval i. Also, the system load resulting from retrieving all pages in $\hat{S}_{i+1} - \hat{S}_i$ during interval i, for $i \in [0, N-1]$, is identical to A. Moreover, the placement of pages referenced by the display schedule from interval N to $m-1$ forces the system to schedule the migrations in M before instant N. For each $i \in [N, m-1]$, the retrieval of pages in $\hat{S}_{i+1} - \hat{S}_i$ must be done during interval i because the memory capacity is exhausted from instant N to $m-1$, making the system unable to pre-fetch. The pages that must be in memory ($\hat{S}_i$) from instant N to $m-1$ to satisfy the display schedule exhaust the memory capacity. Therefore, if there is not enough bandwidth to retrieve a page after instant N, the system is forced to schedule a migration to be able to retrieve the page from other disk drive. *Sat2ResSc* sets $\mathcal{P}$ so that the system must schedule the migrations in M associated with variables to satisfy the display schedule during intervals N to $N+k-1$. For each $i \in [N, N+k-1]$, only one page in $\hat{S}_{i+1} - \hat{S}_i$ cannot be retrieved during interval i; because the bandwidth of the disk where it resides is exhausted with the retrieval of other pages in the same disk. Furthermore, the disk where this page resides and the disks with bandwidth available during interval i are the source and the targets, respectively, of the migration associated with variable v_{i-N+1}. Similarly, *Sat2ResSc* sets $\mathcal{P}$ so that the system must schedule the migrations in M associated with clauses to satisfy the display schedule during intervals $N+k$ to $N+k+n-1 = m-1$.

Formally, the algorithm SAT2ResSc is defined as follows:

SAT2ResSc

Input: $C_1, \ldots, C_n$, $v_1, \ldots, v_k$

Output: $D, B, C, \{P_0, \ldots, P_{m-1}\}, \mathcal{P}$

Method: $(M, A, N) = \text{SAT2MigSc}(C_1, \ldots, C_n, v_1, \ldots, v_k)$

$D = N + 3 \cdot k + n + 2 \cdot n \cdot k + 1$

$B = max\{\text{ number of disjuncts in } C_j \mid j \in [1, n]\} + 1$

$C = 2 \cdot B \cdot (D-1)$

$m = N + n + k$

$\{P_0, \ldots, P_{m-1}\}$ be the display schedule in column P_i of Table 2

$\mathcal{P}$ be the placement of pages on the disk drives as defined below

The placement of pages referenced by the display schedule is defined as follows:

- Every page resides in one disk and has no replicas.
- Set the placement of disk pages in the display schedule as follows:

  Pages to be retrieved by interval -1 ($\hat{S}_0$):

  Assign $a_1, \ldots, a_q$ to disk drives different from w_0 (B pages on each drive). Because $D \cdot B = q + B$, the only drive with available bandwidth during interval -1 is w_0.

  Pages to be retrieved by even intervals in $[0, N-1]$ ($\hat{S}_{i+1} - \hat{S}_i$, for $i \in \{0, 2, \ldots\}$):

  For the assignment of the $q - 1 = B \cdot (D-1) - 1$ pages to be retrieved by even time intervals before instant N, we have two cases:

  (1) There are three disks (x, y, w_{i+1}) with available bandwidth during interval i in A: assign the first $(D-3) \cdot B$ pages to drives different from x, y, w_{i+1} (B pages to each drive), assign the next $B-1$ to drive x, the next $B-1$ to drive y, and the last page to w_{i+1}. Then drives x and y would

i	P_i	$\hat{S}_i$	$\hat{S}_{i+1} - \hat{S}_i$
-1			$a_1 \ldots a_q$
0	$a_1 \ldots a_q$	$a_1 \ldots a_q$	$a_{q+1} \ldots a_{2q-1}$
1	$a_q \ldots a_{2q-1}$	$a_1 \ldots a_{2q-1}$	$a_{2q} \ldots a_{3q-1}$
2	$a_{2q} \ldots a_{3q-1}$	$\mathbf{a_q \ldots a_{3q-1}}$	$a_{3q} \ldots a_{4q-2}$
3	$a_{3q-1} \ldots a_{4q-2}$	$a_{2q} \ldots a_{4q-2}$	$a_{4q-1} \ldots a_{5q-2}$
4	$a_{4q-1} \ldots a_{5q-2}$	$\mathbf{a_{3q-1} \ldots a_{5q-2}}$	.
.	.	.	.
.	.	.	.
N	$b_{q+1} \ldots b_{2q}$	$\mathbf{b_1 \ldots b_{2q}}$	$b_{2q+1} \ldots b_{3q}$
$N+1$	$b_{2q+1} \ldots b_{3q}$	$\mathbf{b_{q+1} \ldots b_{3q}}$	$b_{3q+1} \ldots b_{4q}$
$N+2$	$b_{3q+1} \ldots b_{4q}$	$\mathbf{b_{2q+1} \ldots b_{4q}}$	.
.	.	.	.
.	.	.	.
$m-1$	.	.	.

Table 2: Display schedule ($\{P_0, \ldots, P_{m-1}\}$), pages required to satisfy the display schedule ($\hat{S}_i$), and pages that must be retrieved by interval i to satisfy the display schedule ($\hat{S}_{i+1} - \hat{S}_i$). The third column is computed as follows: $\hat{S}_i = P_i \cup P_{i-1}$, for $0 < i < m$, and $\hat{S}_0 = P_0$.

have disk bandwidth available for one page retrieval/write each. And, drive w_{i+1} would have disk bandwidth available for $B-1$ retrievals/writes.

(2) There are two disks (x, w_{i+1}) with available bandwidth during interval i in A: assign the first $(D-2) \cdot B$ pages to drives different from x, w_{i+1} (B pages to each drive), assign the last $(B-1)$ pages to drive x. Then drive x would have disk bandwidth available for one page retrieval/write and drive w_{i+1} for B retrievals/writes.

Pages to be retrieved by odd intervals in $[0, N-1]$ ($\hat{S}_{i+1} - \hat{S}_i$, for $i \in \{1, 3, \ldots\}$):

The assignment of the $q = B \cdot (D-1)$ pages to be retrieved by odd time intervals before instant N and after instant 0 is as follows: Let x, w_{i+1} be the disk drives with available bandwidth during interval i in A. Assign the first $(D-2) \cdot B$ pages to drives different from x, w_{i+1} (B pages to each drive), assign the next $(B-1)$ pages to drive x, and the last page to w_{i+1}. Then drive x would have available disk bandwidth for one page retrieval/write and drive w_{i+1} for $B-1$ retrievals/writes.

Pages to be retrieved during $[N, N+k-1]$ ($\hat{S}_{i+1} - \hat{S}_i$, for $i \in [N, N+k-1]$):

The assignment of the q pages to be retrieved during each interval i is as follows: Let t_{i-N+1}, u_{i-N+1} be targets of the migration request associated to v_{i-N+1}. Assign the first $(D-3) \cdot B$ pages to drives different from $\{t_{i-N+1}, u_{i-N+1}, w_{i+1}\}$ (B pages to each drive), assign the next page to s_{i-N+1}, the next $B-1$ to t_{i-N+1}, the next $B-1$ to u_{i-N+1}, and the last one to w_{i+1}. Then drives t_{i-N+1} and u_{i-N+1} would have disk bandwidth available for one page retrieval/write each, drive s_{i-N+1} would have exceeded the disk bandwidth requirement by one page, and drive w_{i+1} would have bandwidth available for $B-1$ retrievals/writes.

Pages retrieved during $[N+k, N+k+n-1]$ ($\hat{S}_{i+1} - \hat{S}_i$, for $i \in [N+k, N+k+n-1]$):

The assignment of the q pages to be retrieved during each interval i is as follows: Let $x_1, \ldots, x_j$ be the target disk drives of the migration request associated to $C_{i-N-k+1}$. Assign the first $(D-j-1) \cdot B$ pages to drives different from $x_1, \ldots, x_j, w_{i+1}$ (B pages to each drive), the next page to $d_{i-N-k+1}$, the next $B-1$ to x_1, the next $B-1$ to x_2, and so forth. Finally, assign the last $j-1$ pages to w_{i+1}. Then, drives $x_1, \ldots, x_j$ would have disk bandwidth available for one page retrieval/write each, drive $d_{i-N-k+1}$ would have exceeded the disk bandwidth requirement by one page, and drive w_{i+1} would have bandwidth available for $B-j+1$ retrievals/writes.

□

Observation 1 *From the transformation $SAT2ResSc$, we can observe the following:*

Let $M, A, N = SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. The transformation $SAT2ResSc$ produces a display schedule $\{P_0, \ldots, P_{m-1}\}$, a system configuration (B, C, D), and an initial data placement $\mathcal{P}$ such that:

(1) There is a resource schedule for $\{P_0, \ldots, P_{m-1}\}$ consisting of a schedule $Ret = \{\hat{S}_0, \ldots, \hat{S}_m\}$ (Table 2) of retrievals and discards[2] and a schedule of migrations for M over $[-p, N]$. Moreover, the system load resulting from applying Ret is identical to A during $[0, N]$.

(2) Ret does not pre-fetch pages. Therefore for any resource schedule, for each instant i, $0 \leq i \leq m-1$, $\hat{S}_i \subseteq S_i$.

(3) For any resource schedule that supports $\{P_0, \ldots, P_{m-1}\}$, there is no memory available at instants $N, \ldots, m-1$ for either pre-fetching or migrating disk pages. Because, for each $i, N \leq i \leq m-1$, $|\hat{S}_i| = C$.

We now show that given an instance $C_1, \ldots, C_n, v_1, \ldots, v_k$ of SAT, there is a one-time-interval-latency resource schedule for $SAT2ResSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$ if and only if $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$ has a solution.

Lemma 5.1: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. Let $\{P_0, \ldots, P_{m-1}\}$, $\mathcal{P}$, B, C, D be the output of $SAT2ResSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. If there is a schedule of migrations for M on A during $[0, N]$, then there is a resource schedule that yields a one-time-interval latency and supports a coordinated display of $\{P_0, \ldots, P_{m-1}\}$ on a system configuration (B, C, D) and an initial data placement $\mathcal{P}$.

Proof: See Appendix B. □

We now prove the other direction of the equivalence.

[2]The retrievals (F_i) and discards (K_i) in *Ret* can be derived from the memory states: $F_i = \hat{S}_{i+1} - \hat{S}_i$ and $K_i = \hat{S}_i - \hat{S}_{i+1}$.

Lemma 5.2: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. Let $\{P_0, \ldots, P_{m-1}\}$, $\mathcal{P}$, B, C, D be the output of $SAT2ResSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. If there is a resource schedule that yields a one-time-interval latency and supports a coordinated display of $\{P_0, \ldots, P_{m-1}\}$ on a system configuration (B, C, D) and an initial data placement $\mathcal{P}$, then there is a schedule of migrations for M on A during $[0, N]$.

Proof: See Appendix B. $\square$

Because of Lemma 4.3, we can generalize Lemmas 5.1 and 5.2 to include replications in the schedules for M on A during $[0, N]$.

Lemma 5.3: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. Let $\{P_0, \ldots, P_{m-1}\}$, $\mathcal{P}$, B, C, D be the output of $SAT2ResSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. There is a schedule of migrations/replications for M on A during $[0, N]$ if and only if there is a resource schedule that yields a one-time-interval latency and supports a coordinated display of $\{P_0, \ldots, P_{m-1}\}$ on a system configuration (B, C, D) and an initial data placement $\mathcal{P}$.

It is easy to see that the time complexity of transformation $SAT2ResSc$ is polynomial. Also, because of Lemmas 4.4 and 5.3, this transformation is a valid reduction from SAT to the resource scheduling problem assuming fix-sized units of transfer. Therefore, this scheduling problem is NP-Complete. Presentations with variable-sized objects include those with fix-sized objects (pages). Therefore, the resource scheduling problem for variable-sized units of transfer is also NP-Complete.

Theorem 5.4: Deciding whether there is a resource schedule that yields the latency of one time interval for a given display schedule is NP-Complete.

Corollary 2 *Computing a resource schedule that yields the minimum latency for a given display schedule is NP-hard.*

The computation of a resource schedule is constrained by the memory capacity of the system. An increase in memory might lead to a decrease in latency. One question that arises is what the minimum memory requirement is to render a resource schedule with a pre-specified latency. However, deciding whether there is a resource schedule that yields a latency of one time interval on a system with memory capacity C is NP-Complete. Therefore, computing the minimum memory capacity is NP-hard.

Corollary 3 *Computing the minimum memory requirements to display a presentation within a pre-specified latency is NP-hard.*

6 Conclusions and Future Research

A coordinated display of a presentation must satisfy the temporal constraints associated with each object. Once the display starts, objects must be rendered at pre-specified times defined by the temporal constraints. We studied the complexity of resource scheduling that supports coordinated display of presentations. As demonstrated in [EMGI96], there is a polynomial time algorithm (*greedy*) to compute a resource schedule that minimizes both memory and latency for single-disk architectures. As demonstrated in this paper, resource scheduling that minimizes latency becomes NP-Hard for the case of multi-disk architectures. Also, computing the minimum memory requirement to support a coordinated display within a pre-specified latency is NP-Hard.

Constraining the resource schedules might lead to polynomial time solutions. For example, the computation of retrieval schedules[3] that minimize latency can be done in polynomial time, for the case of fix-sized units of transfer. An extension of *greedy* computes retrieval schedules that yield the minimum latency as follows: Given a display schedule and a data placement across the D disks, this extension extracts the display schedule for each disk based on the pages that reside on that disk. It invokes the greedy scheduler [EMGI96] using the display schedule of each disk to compute a retrieval schedule for that disk. The union of these D retrieval schedules yield a final retrieval schedule for the display. The disk with the longest startup latency (p) determines the overall latency incurred by the display. For the given data placement, this retrieval schedule minimizes the amount of memory required because the greedy scheduler minimizes the memory requirement at each instant i for a single disk [EMGI96]. By minimizing the number of pages that constitute S_0, this extension minimizes the incurred latency. One question that arises is whether the resource scheduling problem is still NP-Hard when migrations/replications schedules do not include intermediate disks. Another question is whether this problem is still NP-Hard for special cases of multi-disk architectures such as a system with 2 disks.

7 Acknowledgments

We would like to thank David A. Barrett and the anonymous reviewers for their helpful comments.

References

[BGJ77] P. Brucker, M. R. Garey, and D. S. Johnson. Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness. *Mathematics Operations Research*, 2:275–284, 1977.

[CR93] A. Choi and M. Ruschitzka. Managing locality sets: The model and fixed–size buffers. *IEEE Transactions on Computers*, 42(2):190–204, February 1993.

[EMGI96] M. L. Escobar-Molano, S. Ghandeharizadeh, and D. Ierardi. An Optimal Resource Scheduler for Continuous Display of Structured Video Objects. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):508–511, June 1996.

[GJ75] M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, December 1975.

[GJS76] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics Operations Research*, 1(2):117–129, May 1976.

[JGJL85] E. G. Coffman Jr, M. R. Garey, D. S. Johnson, and A. S. Lapaugh. Scheduling file transfers. *SIAM Journal on Computing*, 14(3):744–780, August 1985.

[MS91] L. McGeoch and D. Sleator, editors. *On-Line Algorithms*, number 7. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, AMS, February 1991.

[3] Resource schedules without data placement manipulation

[RVVN92] P. I. Rivera-Vega, R. Varadarajan, and S. Navathe. Scheduling file transfers in fully connected networks. *Networks*, 22:563–588, 1992.

[ST85] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

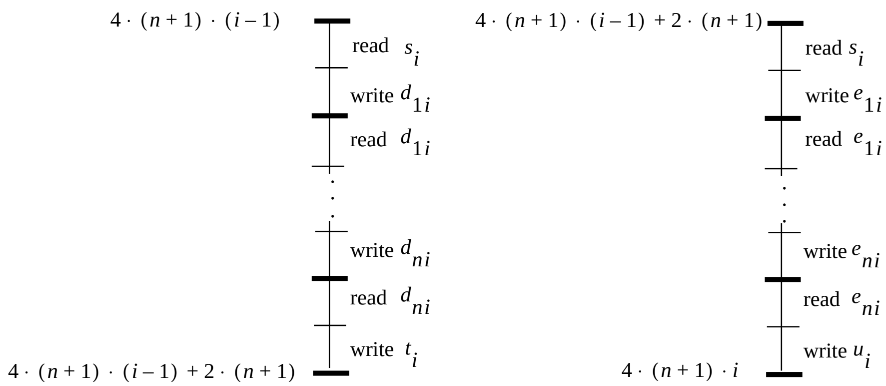
[Whi90] J. Whitehead. The complexity of file transfer scheduling with forwarding. *SIAM Journal on Computing*, 19(2):222–245, April 1990.

A Reduction from SAT to Migrations Scheduling

This section shows that a SAT instance $C_1, \ldots, C_n, v_1, \ldots, v_k$ has a solution if and only if the migrations scheduling instance $M, A, N = SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$ has a solution.

SAT2MigSc defines a system load A and a collection M of migrations requests so that possible schedules of migrations for M follow a specific pattern. For the case of requests associated with variables, there are only two alternatives to schedule them.

Lemma A.1: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$, and $r = s_i \to \{t_i, u_i\}$ be the migration request in M associated with variable v_i. There are only two alternatives to schedule r on A:



(a) (b)

Proof: There are only two time intervals with bandwidth available at drive s_i. Consider the case when the schedule starts with read the page from s_i at interval $4 \cdot (n+1) \cdot (i-1)$. Because there is not memory available at instant $4 \cdot (n+1) \cdot (i-1) + 2$ then the next step must be to write the page to drive d_{1i} during interval $4 \cdot (n+1) \cdot (i-1) + 1$. The next operation to schedule must be to read the page from d_{1i} during interval $4 \cdot (n+1) \cdot (i-1) + 2$, because there will not be other interval with bandwidth available for drive d_{1i} afterwards. A similar argument can be applied to conclude that the subsequent steps in the schedule are to write the page from d_{2i} during interval $4 \cdot (n+1) \cdot (i-1) + 3$, and then read it from d_{2i} during interval $4 \cdot (n+1) \cdot (i-1) + 4$, so on and so forth. The final step in the schedule must be to write the page to drive t_i at interval $4 \cdot (n+1) \cdot (i-1) + 2 \cdot (n+1) - 1$, because there will not be memory available to hold the page at instant $4 \cdot (n+1) \cdot (i-1) + 2 \cdot (n+1)$. In sum, one alternative to schedule the migration associated with v_i is the sequence in (a). Similarly, we can show that the other alternative is the sequence in (b). □

For the migrations requests associated with clauses, there are only c possible schedules on A for a clause with c disjuncts.

Lemma A.2: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$, and $d_j \to \{d_{ji} \mid v_i \in C_j\} \cup \{e_{ji} \mid \neg v_i \in C_j\}$ be the migration request associated with clause C_j. Let $l = 4 \cdot (n+1) \cdot (i-1)$ and $s = 2 \cdot (j-1)$. There are only c (c = number of disjuncts in C_j) alternative schedules for the request associated with C_j:

$$\begin{array}{l} \{\{(d_j, l+s), (d_{ji}, l+s+1)\} \mid v_i \in C_j\} \cup \\ \{\{(d_j, l+2 \cdot (n+1)+s), (e_{ji}, l+2 \cdot (n+1)+s+1)\} \mid \\ \neg v_i \in C_j\} \end{array}$$

Proof: The schedule for the request associated with C_j must start with a read from d_j. From *SAT2MigSc*, we conclude that there are exactly c time intervals in A with bandwidth available in disk d_j. Moreover, the time intervals with bandwidth available for disk d_j are $l+s$, if $v_i \in C_j$, or $l + 2 \cdot (n+1) + s$, if $\neg v_i \in C_j$. Let r be the time interval when the read is scheduled. There are two cases: (1) $r = l+s$ for some i, and $v_i \in C_j$; or (2) $r = l + 2 \cdot (n+1) + s$ for some i, and $\neg v_i \in C_j$. Consider case (1): From the construction of A (Transformation *SAT2MigSc*) we conclude that there will be bandwidth available at drive d_{ji} during interval $r+1$ and there will not be memory available at instant $r+2$. Moreover, d_{ji} is an alternative target for the migration. Therefore, the schedule must finish with a write to disk d_{ji} at interval $r+1$. Similar argument can be applied to case (2). In conclusion, the possible migration schedules for the request associated with C_j are the c alternatives described above. □

Lemma 4.1: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. If there is a truth assignment for variables $\{v_1, \ldots, v_k\}$ that makes all clauses $C_1, \ldots, C_n$ true, then there is a migrations schedule for M on A during interval $[0, N]$.

Proof: Let a be a truth assignment that makes all clauses $C_1, \ldots, C_n$ true. Consider the following migrations schedule for M:

(i) For each variable v_i: if $a(v_i)$ is true, then consider the schedule in Lemma A.1 (b) for the request associated with v_i. Otherwise, consider the schedule in Lemma A.1 (a).

(ii) For each clause C_j: let v_i be the variable such that either $a(v_i)$ is true and $v_i \in C_j$ or $a(v_i)$ is false and $\neg v_i \in C_j$. Let $l = 4 \cdot (n+1) \cdot (i-1)$ and $s = 2 \cdot (j-1)$. If $a(v_i)$ is true and $v_i \in C_j$, then consider the schedule $\{(d_j, l+s), (d_{ji}, l+s+1)\}$ for the request associated with C_j. If $a(v_i)$ is false and $\neg v_i \in C_j$, then consider the schedule $\{(d_j, l+2 \cdot (n+1)+s), (e_{ji}, l+2 \cdot (n+1)+s+1)\}$ for the request associated with C_j.

To prove that the above is a schedule of migrations for M on A, it suffices to show that the schedules for each migration do not overlap each other (i.e., they do not compete for neither disk bandwidth nor memory). The schedules of migrations associated with variables span disjoint periods of time: For each i and j such that $i \neq j$, the following time intervals are disjoint:

$(4 \cdot (n+1) \cdot (i-1), 4 \cdot (n+1) \cdot (i-1) + 2 \cdot (n+1))$
$(4 \cdot (n+1) \cdot (i-1) + 2 \cdot (n+1), 4 \cdot (n+1) \cdot i)$
$(4 \cdot (n+1) \cdot (j-1), 4 \cdot (n+1) \cdot (j-1) + 2 \cdot (n+1))$
$(4 \cdot (n+1) \cdot (j-1) + 2 \cdot (n+1), 4 \cdot (n+1) \cdot j)$

Similarly, the schedules of migrations associated with clauses span disjoint periods of time.

Suppose that the schedule for a variable v_i overlaps the schedule for a clause C_j. Therefore, either v_i or $\neg v_i$ makes C_j true. If $a(v_i)$ is true, then the schedule for v_i spans period $(4 \cdot (n+1) \cdot (i-1) + 2 \cdot (n+1), 4 \cdot (n+1) \cdot i)$ and the schedule for C_j spans period $(4 \cdot (n+1) \cdot (i-1) + 2 \cdot (j-1), 4 \cdot (n+1) \cdot (i-1) + 2 \cdot (j-1) + 1)$. However, these two periods are disjoint. Hence, it contradicts the assumption that the schedules for v_i and C_j overlap. Similarly for the case where $a(v_i)$ is false, we can conclude that the schedules would not overlap.

Therefore, if there is a truth assignment for variables $\{v_1, \ldots, v_k\}$ that makes all clauses $C_1, \ldots, C_n$ true, then there is a schedule of migrations for M on A during period $[0, N]$. □

We now prove the other direction, if the migrations scheduling instance yielded by $SAT2MigSc$ has a solution then the input SAT instance has a solution.

Lemma 4.2: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. If there is a schedule of migrations for M on A, then there is a truth assignment for $\{v_1, \ldots, v_k\}$ that makes all clauses $\{C_1, \ldots, C_n\}$ true.

Proof: The schedules of migrations associated with variables in $SATRepSc$ follow either pattern of Lemma A.1. Therefore, a valid truth assignment a is as follows: $a(v_i)$ is true if the migration schedule for the request associated with v_i follows the pattern in Lemma A.1 (b), and is false if it follows the pattern in Lemma A.1 (a).

We now show that a makes all clauses $\{C_1, \ldots, C_n\}$ true. Suppose that there exists a clause C_j such that all its disjuncts are false. The migration schedule associated with C_j must be either (Lemma A.2): (a) $\{(d_j, 4 \cdot (n+1) \cdot (i-1) + 2 \cdot (j-1)), (d_{ji}, 4 \cdot (n+1) \cdot (i-1) + 2 \cdot (j-1) + 1)\}$, if $v_i \in C_j$; or (b) $\{(d_j, 4 \cdot (n+1) \cdot (i-1) + 2 \cdot (n+1) + 2 \cdot (j-1)), (e_{ji}, 4 \cdot (n+1) \cdot (i-1) + 2 \cdot (n+1) + 2 \cdot (j-1) + 1)\}$, if $\neg v_i \in C_j$. Suppose that the schedule of C_j is as described in (a). Then the migration schedule associated with v_i must follow the pattern in Lemma A.1 (b). Otherwise, there would be a conflict, for the disk bandwidth of d_{ji} and the memory frame available at instant $4 \cdot (n+1) \cdot (i-1) + 2 \cdot (j-1) + 1$, between the schedules for C_j and v_i. Therefore $a(v_i)$ is true, according to the definition of a described above. However as stated in (a), $v_i \in C_j$ then C_j is true. This contradicts the assumption that all disjuncts in C_j are false. Similarly, we can reach a contradiction when the schedule for C_j is as described in (b).

Therefore, a makes all clauses $\{C_1, \ldots, C_n\}$ true. □

B Reduction from SAT to Resource Scheduling

This section shows that given an instance $C_1, \ldots, C_n, v_1, \ldots, v_k$ of SAT, there is a one-time-interval-latency resource schedule for $SAT2ResSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$ if and only if $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$ has a solution.

Lemma B.1: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. The transformation $SAT2ResSc$ produces a display schedule $\{P_0, \ldots, P_{m-1}\}$, a system configuration (B, C, D) and an initial placement of data $\mathcal{P}$ such that any resource schedule for $\{P_0, \ldots, P_{m-1}\}$ that yields a one-time interval latency must schedule migrations for M during time interval $[0, N]$.

Proof: The system must migrate the pages that cannot retrieve during intervals $N, \ldots, m-1$ (Observation 1 (3)) before instant N. Hence, for each $i \in [1, k]$ the system must migrate a page from drive s_i to either t_i, u_i, or w_{N+i} before interval N. And, for each $i \in [1, n]$ the system must migrate a page from drive d_i to either drive in the target set of the migration request associated with C_i or to w_{N+k+i}, before interval N.

The schedule *Ret* in Observation 1 retrieves each page in the display schedule only once (Column $\hat{S}_{i+1} - \hat{S}_i$ in Table 2) and does not pre-fetch pages (Observation 1 (2)). Therefore, any resource schedule would require at least the disk bandwidth required by *Ret* before interval N to satisfy the display schedule during $[0, N]$. Then the source, target and intermediate drives in the schedule of a migration must have disk bandwidth available in A during $[0, N]$. Otherwise, the bandwidth requirements of a disk drive would exceed the disk bandwidth availability during $[0, N]$. Thus, disk drives w_i for $i \in [N+1, m]$ cannot be a target drive of a migration. Therefore, the system must schedule migrations for M before instant N.

Migrations requests in M must be scheduled after instant 0 otherwise the latency would be higher than one time interval. Starting the migration schedule at interval -1 would increase the latency because: (1) the retrieval of all pages in P_0 would require the disk bandwidth of all disks except w_0 and (2) w_0 is not a source drive for any migration request. In sum, the system must schedule migrations for M during $[0, N]$. □

Lemma 5.1: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. Let $\{P_0, \ldots, P_{m-1}\}$, $\mathcal{P}$, B, C, D be the output of $SAT2ResSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. If there is a schedule of migrations for M on A during $[0, N]$, then there is a resource schedule that yields a one-time-interval latency and supports a coordinated display of $\{P_0, \ldots, P_{m-1}\}$ on a system configuration (B, C, D) and an initial placement of data $\mathcal{P}$.

Proof: Let S be a schedule of migrations for M on A during $[0, N]$. Construct a resource schedule as follows:

Step 1: Include retrieval schedule *Ret* in Observation 1.

Step 2: Change the retrievals in *Ret* of pages in M to be retrieved from their target drives in RS.

Step 3: Include the schedule of migrations S.

This resource schedule supports a coordinated display of $\{P_0, \ldots, P_{m-1}\}$ that yields a one-time interval latency. □

To prove the other direction, we show that scheduling a migration for $r \in M$ as part of a resource schedule for $SAT2ResSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$ requires at least the memory required by the migration schedule for r on A during $[0, N]$. Where M, A, N is the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$.

Given a migration schedule, the time intervals when the reads and writes are scheduled determines the memory requirements. The memory requirements of two migrations that coincide in the time interval when a read and the next write is scheduled are identical.

Definition: Given a migration schedule

$$\{(source^i, t_1^i), (d_1^i, [t_2^i, t_3^i]), \ldots, (d_{k_i-1}^i, [t_{2k_i-2}^i, t_{2k_i-1}^i]), (target_p^i, t_{2k_i}^i)\}$$

the *memory requirements* of the migration schedule during $[-p, N]$ is defined as the sequence:

$$(\underbrace{0,\ldots,0}_{p+t_1^i+1 \text{ times}}, \underbrace{1,\ldots,1}_{t_2^i-t_1^i \text{ times}}, \underbrace{0,\ldots,0}_{t_3^i-t_2^i \text{ times}}, \ldots, \underbrace{1,\ldots,1}_{t_{2k_i}^i-t_{2k_i-1}^i \text{ times}}, \underbrace{0,\ldots,0}_{N-t_{2k_i}^i \text{ times}})$$

that represents the number of memory frames required by the schedule at each instant i, $i \in [-p, N]$

Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. For any resource schedule for $SAT2ResSc($ C_1, $\ldots$, C_n, v_1, $\ldots$, $v_k)$, there are two alternatives to schedule a migration for r in M: (1) schedule a migration for r based on the system load A, or (2) modify the retrieval schedule *Ret* (Observation 1) to accommodate the migration for r. For the second alternative, the system might schedule additional migrations. For example, to schedule a migration of page a from drive s to drive t. The system might utilize the disk bandwidth used at interval ti to retrieve a page b from s in *Ret* to read the page a from s. Then, page b can either be pre-fetched at an earlier time interval, or be migrated from s to a disk u with available bandwidth at ti so that b can be retrieved from u at ti. If there is not memory to pre-fetch b, then the system is forced to migrate b. Therefore the migration schedule (from s to t) includes the schedule of a new migration (from s to u). The additional migrations also increase the memory requirements. Therefore their memory requirements should also be considered to obtain the memory requirements of the schedule.

Definition: Let $\{P_0, \ldots, P_{m-1}\}, \mathcal{P}, B, C, D$ be the output of $SAT2ResSc(C_1,\ \ldots,\ C_n,\ v_1,\ \ldots,\ v_k)$. Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. If scheduling a migration for r_0 in M modifies the retrieval schedule *Ret* in such a way that additional migration requests $r_1, \ldots, r_{n_r}$ must be scheduled. Then the *extension* of r_0 is the set of migrations requests $\{r_0, \ldots, r_{n_r}\}$.

For example, suppose that to schedule a migration for r_0 the system requires two additional migrations (r_1 and r_2). Suppose that the memory requirements of the schedules for these migrations are as follows: $(0,0,0,0,0,0,1,0,1,0)$ for r_0, $(0,0,0,0,1,0,0,0,0,0)$ for r_1, and $(0,0,1,0,0,0,0,0,0,0)$ for r_2. Then the extension of r_0 is $\{r_0, r_1, r_2\}$ and its memory requirements is $(0,0,1,0,1,0,1,0,1,0)$.

The memory requirements of schedules of migrations define a partial order on the schedules.

Definition: A migration schedule sr_1 is *greater* (*smaller*) than a migration schedule sr_2 if and only if for each instant $i \in [-p, N]$ the memory requirement of sr_1 at i is greater (smaller) than or equal to the memory requirement of sr_2 at i

Lemma B.2: Let $\{P_0, \ldots, P_{m-1}\}, \mathcal{P}, B, C, D$ be the output of $SAT2ResSc(C_1,\ \ldots,\ C_n,\ v_1,\ \ldots,\ v_k)$. Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. Let sr be the schedule of the extension of r in a one-time-interval-latency resource schedule for $\{P_0, \ldots, P_{m-1}\}$, where r is the migration request associated with clause C_j. Then, there exists some migration schedule sr' for r on A such that sr' is smaller than sr.

Proof: It suffices to consider the case when the system changes Ret to accommodate the scheduling of a migration for r. sr must end with a write page on drive d_{ji} or drive e_{ji}. This write page must be scheduled at an interval l such that there is memory available at instant l. Therefore, the write page must be scheduled during an odd time interval (i.e., $1, 3, 5$ etc.). Suppose that the write page is scheduled during an interval l that does not have bandwidth available for any drive in the target set. Then, there are two alternatives: (1) to pre-fetch a page retrieved during l in Ret from a target drive at an earlier time interval, or (2) to migrate a page a retrieved during l in Ret from a target drive to a drive with available bandwidth in l, so that a can be retrieved from another drive during l. The first alternative is not possible, because the write a page operation requires an additional memory frame at instant l to hold the page. The memory is thus exhausted at instant l, then there is not memory available to hold the pre-fetched page. The second alternative is not possible either, because the disk drives with available bandwidth during odd time intervals (e.g., u_i, t_i, e_{ji}, d_{ji}) do not have disk bandwidth available at an earlier time interval. Therefore, the migration of a would increase the disk bandwidth requirement of such drives to more than what is available during the period $[0, l]$. In sum, the write page operation must be scheduled during an odd time interval l that has bandwidth available for a drive in the target set. Because there is not memory available at instant $l-1$, the page must be read during interval $l-1$. In conclusion, one of the migration schedules in Lemma A.2 is smaller than sr. □

Lemma B.3: Let $\{P_0, \ldots, P_{m-1}\}, \mathcal{P}, B, C, D$ be the output of $SAT2ResSc(C_1, \ldots, C_n,\ v_1,\ \ldots,\ v_k)$. Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. Let sr be the schedule of the extension of r in a one-time-interval-latency resource schedule for $\{P_0, \ldots, P_{m-1}\}$, where r is the migration request associated with variable v_i. Then, there exists some migration schedule sr' for r on A such that sr' is smaller than sr.

Proof: It suffices to consider the case when the system changes Ret to accommodate the scheduling of r. sr must end with a write page a to either drive t_i or drive u_i. As for the case of write page on a target drive in proof of Lemma B.2, the write page on either drive t_i or u_i must be scheduled during an odd time interval l that has bandwidth available for either drive t_i or u_i. Without loss of generality, suppose that it writes the page on drive t_i. Because there is not memory available at instant $l-1$, the page must be read during interval $l-1$. However, there is not available disk bandwidth for drive s_i during interval $l-1$. Then the system must either (1) migrate a from s_i to a disk with available bandwidth during $l-1$ so that a can be retrieved from this disk, or (2) migrate a page b retrieved from s_i during $l-1$ in Ret to a drive with available bandwidth during $l-1$ so that a is retrieved from s_i and b from the new location during $l-1$. Then the system must migrate a page (a or b) from s_i to d_{ni}. To schedule this migration, the write operation on drive d_{ni} must be scheduled at interval $l-2$ because it is the only odd time interval before $l-1$ with available disk bandwidth for d_{ni}. Then the system has to schedule a read from s_i at interval $l-3$ because there is not memory available at instant $l-3$. If there is only one clause in the SAT instance, then one of the migration schedules in Lemma A.1 is smaller than sr. If there is more than one clauses in the SAT instance, then there is not available disk bandwidth for s_i at interval $l-3$. Therefore, as before the system has to migrate a page from s_i to either d_n (if there is available bandwidth in d_n) or $d_{(n-1)i}$. Because a write must be scheduled during an interval ti with memory available at instant ti and there is not available bandwidth for drive d_n during an odd time interval, the system has to schedule the migration from s_i to $d_{(n-1)i}$. Similar reasoning can be applied iteratively to conclude that one of the migration schedules in Lemma A.1 is smaller than sr.

□

We now conclude the proof of the other direction of the equivalence of instances.

Lemma 5.2: Let M, A, N be the output of $SAT2MigSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. Let $\{P_0, \ldots, P_{m-1}\}$, $\mathcal{P}$, B, C, D be the output of $SAT2ResSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. If there is a resource schedule that yields a one-time-interval latency and supports a coordinated display of $\{P_0, \ldots, P_{m-1}\}$ on a system configuration (B, C, D) and an initial placement of data $\mathcal{P}$, then there is a schedule for migrations M on A during $[0, N]$.

Proof: Suppose that there is a resource schedule Sc for $\{P_0, \ldots, P_{m-1}\}$ that yields a latency of one interval and there does not exist a schedule of migrations for M on A during $[0, N]$. Consider the following schedule, Sa, for requests in M on A: For each request $r \in M$, consider a migration schedule sr' in Lemmas B.2 and B.3 such that $sr' < sr$, where sr is the schedule of r's extension in Sc.

Because there is not a schedule of migrations for M on A, there must be two requests in M, r_1 and r_2, such that their corresponding schedules in Sa conflict. The only possibility of conflict between the schedules for r_1 and r_2 in Sa is if r_1 is associated with a variable v_i and r_2 with a clause C_j. Because the other combinations do not have overlapping periods. Without loss of generality suppose that the schedule of r_1 in Sa span the period $(x, x + 2 \cdot (n + 1))$ where $x = 4 \cdot (n + 1) \cdot (i - 1)$ and the schedule of r_2 the period $(x + 2 \cdot (j - 1), x + 2 \cdot (j - 1) + 1)$. Both schedules require a memory frame at instant $x + 2 \cdot (j - 1) + 1$. Then, the schedules of the extensions of r_1 and r_2 in Sc would also require two memory frames at instant $x + 2 \cdot (j - 1) + 1$. However, there is only one memory frame available at this instant. Therefore Sc is not a resource schedule for $\{P_0, \ldots, P_{m-1}\}$ that yields a latency of one interval, which contradicts our assumption about Sc. □