MANAGEMENT OF RESOURCES TO SUPPORT COORDINATED DISPLAY OF STRUCTURED PRESENTATIONS

by

Martha Lucía Escobar-Molano

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Science)

August 1996

Copyright 1996 Martha Lucía Escobar-Molano
Acknowledgments

I gratefully acknowledge the help of so many people whom have made this dissertation possible. I thank my advisor, Shahram Ghandeharizadeh, whose encouragement to develop my own ideas was valuable for the completion of this dissertation. His feedback contributed greatly to the presentation of this material. His continuing technical and economic support have been helpful for the conclusion of this work. Next, I would like to thank David Barrett, Chao-Kuei Hung, and Douglas Ierardi who took time from their busy schedule to read this work and suggest necessary improvements. I also thank the other members of my thesis committee, Tomlinson Holman and Dennis McLeod, for providing additional comments and suggestions for improvements to this work.
Contents

Acknowledgments ii
List Of Tables v
List Of Figures vi
Abstract viii
1 Introduction 1
2 Related Work 10
3 Conceptual Model of the Object Space 14
  3.1 Atomic Objects ........................................ 17
  3.2 Composed Objects ....................................... 19
  3.3 Rendering Features ..................................... 20
4 Complexity of Resource Scheduling 21
  4.1 Statement of the Problem ................................. 21
  4.2 Complexity of Resource Scheduling for Single-Disk Architectures 26
  4.3 Complexity of Resource Scheduling for Multi-Disk Architectures 31
    4.3.1 Replications Scheduling ............................ 34
    4.3.2 Resource Scheduling ............................... 38
5 Scheduling Approaches 44
  5.1 Memory-Based Resource Schedule ......................... 46
  5.2 Replication-Based Resource Schedule .................... 46
  5.3 A Hybrid of Memory and Replication-Based Resource Schedule 49
  5.4 Evaluation ............................................. 50
    5.4.1 Simulation Model .................................. 50
    5.4.2 Performance Results ............................... 52
      5.4.2.1 Seven-Disk System ............................ 53
      5.4.2.2 Trends ....................................... 59
5.4.2.3 Summary ................................................. 61

6 Data Layout and Availability .................................. 63
   6.1 Initial Placement of Data ................................... 63
   6.1.1 Statement of the Problem ................................ 64
      6.1.1.1 Single-Copy Placement .............................. 64
      6.1.1.2 Multi-Copy Placement ............................... 66
   6.1.2 Placement Approaches .................................... 68
   6.2 Fault Tolerance ........................................... 69

7 Conclusions and Future Research Directions ................ 73

Reference List ................................................... 74

Appendix A
   Reduction from SAT to Replications Scheduling ................ 80

Appendix B
   Reduction from SAT to Resource Scheduling .................... 84
List Of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>List of terms used in this paper and their definitions</td>
<td>23</td>
</tr>
<tr>
<td>4.2</td>
<td>Display schedule ( {P_0, \ldots, P_{m-1}} ) and schedule of retrievals and discards ( {\hat{S}_0, \ldots, \hat{S}_m} )</td>
<td>41</td>
</tr>
<tr>
<td>5.1</td>
<td>Performance of alternative heuristics for Presentation 1 using 7 disks</td>
<td>54</td>
</tr>
<tr>
<td>5.2</td>
<td>Performance of alternative heuristics for Presentation 2 using 7 disks</td>
<td>54</td>
</tr>
<tr>
<td>5.3</td>
<td>Measurements obtained for Presentation 1 using 8 disks</td>
<td>59</td>
</tr>
<tr>
<td>5.4</td>
<td>Measurements obtained by memory-based and hybrid techniques for Presentation 1 in a system with 10 and 12 drives. Both techniques yielded identical measurements.</td>
<td>60</td>
</tr>
<tr>
<td>5.5</td>
<td>Measurements obtained for Presentation 1 using 14 disks</td>
<td>60</td>
</tr>
<tr>
<td>5.6</td>
<td>Measurements obtained for Presentation 1 in a system with 16 drives</td>
<td>60</td>
</tr>
<tr>
<td>5.7</td>
<td>Performance comparison assuming that the threshold ( \frac{L_C}{100} ) is smaller than the memory requirements of memory-based scheduling.</td>
<td>62</td>
</tr>
</tbody>
</table>
# List Of Figures

1.1 Conceptual Model ........................................... 2
1.2 Alternatives to display a structured presentation .......... 4
1.3 System Architecture ......................................... 5

2.1 Spatial Constructs. (a) object \( o \) and vector \( v_o \). (b) rendering space and vector \( v_r \). (c) the placement of \( o \) in the rendering space after applying the spatial construct: \( v_o \rightarrow v_r \). ........................................... 15

3.2 Levels of abstraction in the object space. (a) Description of each level (b) Scene of Simba walking. ........................................... 16
3.3 Atomic object schema ....................................... 18
3.4 Composed Object schema .................................... 19
3.5 The rendering features schema ................................ 20

4.1 Time interval and time instant ................................ 22
4.2 Resource schedules (a) without manipulation of the data placement. (b) with manipulation of the data placement. .................. 25
4.3 The greedy algorithm ......................................... 28
4.4 Optimal resource schedule ................................... 28
4.5 Schedule of a replication ..................................... 33
4.6 System load yielded by SAT2RepSc. .......................... 36
4.7 Example of reduction of a SAT instance into a replications scheduling instance. (a) SAT instance, (b) Corresponding set \( R \) of replications, (c) Patterns to denote alternative schedules of replications associated to variables and clauses, (d) Corresponding system load \( A \). ............ 37

5.1 Taxonomy of possible approaches to resource scheduling .... 45
5.2 Simulation Model ............................................. 51
5.3 Number of pages referenced during each time interval by Presentation 1 52
5.4 Coordinated display of Presentation 1 using memory-based with a 7-disk system ............................................. 55
5.5 Theoretical minimum for Presentation 1 in a system with 7 disks ... 56
5.6 Three consecutive displays of Presentation 1 in a system with 7 disks 58

6.1 Alternative placement of pages referenced by a presentation ..... 65
6.2 Degrees of parallelism. (a) based on the multi-copy placement, (b) based on the single-copy placement with $d$ on disk 1, and (c) based on the single-copy placement with $d$ on disk 0 . . . . . . . . . . . . . . . . 67

6.3 Placement of pages referenced by display schedules $\{a, b, c, d\}, \{e, f\}$, $\{e, g, h, i\}$ and $\{e, j, k, l\}, \{e, m, l\}, \{h, i, m, n\}$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
Abstract

With the structured approach to represent video clips, a presentation consists of a collection of background objects and actors (3-D representations) constrained using spatial and temporal constructs along with rendering features such as shading and the audiences’ viewpoint. While the spatial constructs define the position of the objects on the screen, the temporal constructs describe when the objects are displayed. As compared with an alternative approach (termed stream-based) that conceptualizes a video clip as a sequence of frames, the structured approach provides for both re-usability of objects in other presentations and effective query processing techniques for retrieval of relevant data. The display of a structured presentation is termed coordinated when the rendering of its objects respects the pre-specified temporal and spatial 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.

This dissertation proposes the structured approach to conceptualize video in a database. We partition a video clip into an Object Space and a Name Space. The object space represents the rendering aspect of a presentation, e.g., 3D representations of characters, positions of characters in a scene, audience’s viewpoint. The name space represents the user’s interpretation of video, e.g., name of characters, actions such as running, walking, etc. We organize the object space into three layers: (1) components, (2) spatial and temporal constructs, and (3) rendering features such as light sources, viewpoint, etc. The object space provides a framework to study resource scheduling that supports the display of coordinated presentations.

Assuming a hardware platform configured with $D$ disks and a fixed amount of memory, this dissertation studied the complexity of resource schedulers that support the display of structured presentations. The obtained results are as follows:
(1) For a single-disk architecture \((D = 1)\), an optimal resource scheduler exists with worst case time complexity of \(O(n \lg n)\). This scheduler minimizes both the incurred startup latency and the amount of memory required by a display.

(2) For a multi-disk architecture \((D > 1)\), a resource scheduler that minimizes the startup latency is NP-Hard.

This study provides a taxonomy of resource scheduling heuristics that guarantee a coordinated display of structured presentations for a multi-disk hardware platform \((D > 1)\). A subset of these techniques manipulate the placement of data across the available disks. We employ a simulation study to quantify the trade-off associated with the alternative heuristics. The results are compared with a theoretical minimum, demonstrating that one of the proposed techniques provides a performance almost identical to this minimum.
Chapter 1

Introduction

One may represent video using two alternative approaches [19]: stream-based and structured [14]. With the stream-based approach, a video clip consists of a sequence of frames that are displayed at a pre-specified rate (e.g., 30 frames per second). With the structured approach, a presentation consists of a collection of objects (e.g., 3-D representations of a dinosaur) that are constrained using spatial and temporal constructs (e.g., positions of the dinosaur and the time of their appearances) along with their rendering features (e.g., light intensity, view point). Presently, the structured approach is used to produce animated sequences. For example, “Toy Story” [4] and “Reboot” [5] are animations generated using the structured approach.

The structured approach provides for both re-usability of information and development of effective query processing techniques [15]. They enable a user to extract a character (e.g., a dinosaur) or a motion path (e.g., the trajectory and timing of the dinosaur’s motion) from one presentation and re-use it in another. In addition, one can devise algorithms to support processing of queries that reason about the temporal and spatial information of a structured presentation. To illustrate, consider the animation “The Lion King”. Assuming it was represented using the structured approach, to retrieve the scene where Simba finds his father (Mufasa) dead, a user can pose the following query: select those scenes that contain both Simba and Mufasa such that Mufasa is static while Simba is moving. The system locates the relevant data by analyzing the temporal and spatial constraints imposed on the 3-D representations of Mufasa and Simba.

This dissertation proposes a conceptual model (Figure 1.1) of structured presentations that separates its object space from its name space. The object space
Show me the scenes with Simba running in the jungle

Figure 1.1: Conceptual Model.
represents the rendering aspect of presentations, e.g., a collection of 3-D representations of characters and backgrounds associated using temporal and spatial constructs along with rendering characteristics. This information is already present in animations, therefore it is straightforward to populate the object space for animated sequences. The name space represents the user’s interpretation of objects and their relationships, e.g., name of characters, actions such as chasing, running, walking, etc. To populate the name space, the system might consider the input given by the author in the script (e.g., names of characters, actions, etc.), the input given by the viewer (e.g., Siskel and Ebert rating of the presentation), and the results of some inference rules (e.g., the description of the motion of a character who “runs” at the beginning of the scene and “walks” at the end of the same scene can be generalized to “moves” by an inference rule).

When a user poses a query against this name space, the system identifies the relevant data. If the output of the query is a scene, then it also delivers the scene specification to the rendering package according to the temporal constraints. For example, suppose that the name space of a presentation contains a list of background names, and descriptions of the motions in the scenes (Figure 1.1). Each description consists of the name of the character and the type of motion (e.g., running, walking, etc.). To process the query *Show me the scenes with Simba running in the jungle*, the system selects those scenes that have a jungle as their background and Simba running as one of their motions. The terms “jungle” and “running” are user’s interpretation of data. Another user might have named them “forest” and “jogging”, respectively. Once the system retrieves a scene from the name space, it retrieves the corresponding rendering description from the object space and delivers it to the rendering package.

There are alternative ways of displaying structured video (Figure 1.2). A compiler might transform a structured presentation into a stream-based one to complement the existing infrastructure. For example, Toy Story was generated using the structured approach. Subsequently, it was compiled and released to the movie theaters for mass previewing. An interpreted display renders a scene and its objects in real-time [13]. The display of a presentation can be either static or dynamic. A dynamic display (e.g., video games, flight simulators) accepts input from a user to guide the presentation. On the other hand, a static display renders a presentation based on its specification defined by a producer. This dissertation studies resource scheduling
Figure 1.2: Alternatives to display a structured presentation

to guarantee coordinated display of static interpreted structured presentations. The system renders a presentation in real time by retrieving and displaying objects at the appropriate time and location on the computer screen. A database of structured presentations might be shared by a variety of users who author presentations, e.g., producers, animation artists, etc. Once a structured presentation is defined, its author may want to display it in order to refine the presentation. (This process is almost identical to a programmer authoring a program and desiring to execute it in order to evaluate and refine its functionality.) A challenging task for the system is to ensure a coordinated display where the system satisfies the author’s pre-specified temporal and spatial constraints imposed on the presentation of objects. If these constraints are not satisfied, then the presentation might suffer from errors. To illustrate, consider the sequence of postures $p_1, \ldots, p_n$ that provide the illusion that a dinosaur is walking. These postures might be a collection of persistent delta updates on the dinosaur’s 3-D representation changing its facial expressions (morphing) and moving its body along a curve as a function of time $[40, 41, 25]$. If the system fails to render the original 3-D object, the deltas will yield a partially missing character with a changing face and moving body parts.

Our target hardware platform (Figure 1.3) consists of a fixed amount of memory (DRAM) and $D$ homogeneous magnetic disk drives where objects participating in
a structured presentation reside. We assume that the unit of transfer from each disk is fix-sized and termed a page. An object might be either smaller or larger than a disk page. When an object is larger than a disk page, it is represented as a collection of pages. The system may cluster several small objects in a single disk page. To display a structured presentation in real-time, the system must render those objects that constitute a scene memory resident according to the pre-specified temporal constraints. These objects are contained in a set of pages. A display schedule defines which pages must be in memory at what time to satisfy the pre-specified temporal constraints. The storage manager processes the display schedule to generate a resource schedule that satisfies the temporal constraints. Based on the resource schedule, the system delivers data from disk to memory. The rendering package processes the pages in memory to display the presentation. We assume that the bottleneck resource is the disk bandwidth (i.e., the CPU is fast enough to process the data as long as the required data is in memory) which is in accordance with the current technological trends. (The speed of CPUs is growing much faster than the
disk bandwidth.) This dissertation studies the problem of computing a resource schedule that satisfies the temporal constraints imposed by the display schedule.

One approach to resolve the resource scheduling problem is to render an entire structured presentation memory resident prior to initiating its display. This approach suffers from the following limitations. First, it wastes memory because it requires objects that are displayed at the end of the presentation to be rendered memory resident for the entire duration of the display. Second, it fails to display those presentations whose total object sizes exceed the amount of available memory (some systems use swap memory and employ the operating system to swap pages in and out of memory). Third, it incurs a high startup latency\(^1\) because the display is delayed until all referenced objects are rendered memory resident. This dissertation studies resource scheduling techniques that overlap the display of objects with their retrieval from disk into memory, such that (1) the incurred startup latency and the amount of memory required for a display are minimized, and (2) the temporal constraints imposed on the presentation are satisfied.

For single-disk architectures \((D = 1)\), this dissertation presents an algorithm that employs the information provided by the display schedule to construct an optimal resource schedule for staging objects from disk onto memory. This optimal schedule supports a coordinated display while minimizing both the startup latency and the amount of memory required for the display.

For multi-disk architectures \((D > 1)\), there are two alternatives to retrieve data from disk [32]: (1) grouped reads and (2) individual reads. A grouped read transfers a disk page that is declustered across a group of \(D_g\) disks into memory by activating all \(D_g\) disks simultaneously. An individual read transfers a disk page from a single disk into memory. For the case of individual reads, parallel retrievals from several disks improves the system’s performance. The impact of seek and rotation latency times on the system’s performance depends on the size of the disk page. For individual reads, if the size of a disk page is \(P\) then the percentage of wasted disk bandwidth attributed to seek and latency times is \(\frac{\text{seek} + \text{latency} \times \frac{1}{P}}{\text{seek} + \text{latency}}\), where \(\text{seek}\) and \(\text{latency}\) are the worst case seek\(^2\) and latency times. For grouped reads, if the size of a disk page is

\(^1\)The time elapsed from the arrival of a request to the onset of its display.

\(^2\)The seek time could be reduced when the system reads batches of \(NR\) pages following the elevator algorithm[38]. In this case, the seek time becomes \(\frac{1.5 \times \text{seek}}{NR}\).
$G$ then the percentage of waste is $\frac{\text{seek + latency}}{\text{seek + latency} + \frac{G}{D}}$. The waste would be identical for both cases when $G = D \cdot P$, higher for grouped reads when $G < D \cdot P$, and higher for individual reads when $G > D \cdot P$. In general, the disk page for grouped reads should be larger than for individual reads to diminish the impact of seek and latency times on the disk bandwidth.

Objects in a structured presentation vary in size (e.g., from 2 KBytes to 30 MBytes). Objects larger than a disk page are partitioned into blocks smaller than or equal to a page. When the size of an object is not a multiple of a page size, the disk page containing its tail end might remain partially empty. The remaining space can be utilized by storing smaller objects in the partially empty page. In general, objects and blocks smaller than a disk page can be clustered in a single disk page. Clustering objects/blocks that are referenced by a presentation at one point in time in a single disk page enhances the buffer pool hit ratio to free disk bandwidth. However, clustering objects/blocks that are not referenced by a presentation simultaneously in a single disk page wastes both disk bandwidth and memory. Each time an object/block in the disk page is referenced, the system retrieves the entire disk page utilizing both disk bandwidth and memory to retrieve non-referenced objects/blocks. The ideal clustering is to have only objects/blocks referenced simultaneously in a disk page. Unfortunately, the reference pattern of objects in a database might be very complex. Objects might be referenced both by several presentations and by the same presentation at different times. Furthermore, the reference pattern changes with additions of new presentations to the database. In sum, clustering objects so that the waste of disk bandwidth and memory is minimized is a hard problem. An alternative to minimize this waste is to avoid clustering by assuming a small disk page. This reduces the number of objects that can fit in a disk page. Then, individual reads would reduce this waste as opposed to grouped reads. We chose individual reads as the alternative to retrieve data from a multi-disk architecture.

The placement of pages on disks impacts both the incurred startup latency and the amount of required memory. This is because it might yield a reference pattern that forces a subset of disks to become temporary bottlenecks during the display. For example, consider a system with 4 disk drives and a combination of data placement with a display schedule that causes all the referenced pages to be retrieved from a
single disk. During the display, this disk becomes a bottleneck while the other three disks are idle. To support a coordinated display, a resource scheduler detects these bottlenecks in advance. This can be done because both the schedule of page references and the placement of data are pre-specified. The scheduler can resolve these bottlenecks by either (1) pre-fetching some pages at an earlier time and maintaining them in memory until they are referenced, or (2) replicating or migrating in advance those pages that result in bottlenecks onto the idle disk drives. Therefore, a resource schedule for a multi-disk architecture consists of page retrievals, page migrations, and page replications.

For multi-disk architectures ($D > 1$), this dissertation demonstrates the following: (1) the computation of a resource schedule that satisfies a pre-specified display schedule and minimizes the startup latency is NP-Hard, (2) the computation of the minimum memory requirement to display a coordinated presentation within a pre-specified latency is NP-Hard, and (3) deciding whether there is a schedule to change the placement of data across disk drives based on a pre-specified system load is NP-Complete.

Based on these results, we introduce a taxonomy of heuristic-based resource scheduling techniques. These techniques can be classified based on the method used to resolve bottlenecks. There are three alternatives to resolve temporary bottlenecks: memory-based, replication-based, and migration-based. While the memory-based techniques pre-fetch pages and keep them in memory until they are referenced, both the replication and migration-based techniques manipulate the placement of pages prior to their reference to dissolve the bottlenecks. Techniques that manipulate the placement of data can make the changes either persistent or transient. Persistent manipulation of data placement impacts future displays of the same schedule while transient ones do not.

This dissertation introduces three heuristics to compute resource schedules that support coordinated display of structured presentations. One heuristic is memory-based, the other is replication-based, and the third is a hybrid of memory and replication-based techniques. We evaluated these heuristics using a simulation model and compare them with a theoretical minimum based on the following metrics: startup latency, memory requirements, disk bandwidth and disk space requirements.
The obtained results demonstrate that the persistent replication-based heuristic approximates the theoretical minimum as the number of displays increases.

The initial placement of pages across disk drives impacts the performance of the system. Unbalanced placement\(^3\) impedes the system to maximize parallel retrieval from disks. The objective of an initial data placement technique should be to assign pages to disks so that parallel retrieval is maximized. The complexity of the computation of a data placement that maximizes parallelism is an open question that needs further investigation. This dissertation proposes a data placement technique that employs the temporal information\(^4\) in a display schedule to assign pages to disks aiming for maximum parallelism. The evaluation of this technique and development of additional techniques constitute a future research direction of this dissertation.

The resource scheduling techniques proposed in this dissertation assume that all disks are operational. However, the mean time to failure of some disk decreases as the number of disks \((D)\) increases [32]. Therefore, fault tolerance is an important issue for multi-disk architectures. This dissertation presents a framework for fault tolerance in the context of structured presentations. Fault tolerance schemes for structured presentations need further investigation.

The rest of this dissertation is organized as follows. Chapter 2 provides a survey of previous work related to this dissertation. Chapter 3 presents a layered representation of the object space. Chapter 4 presents the complexity of computing a resource schedule that supports a coordinated display of structured presentations. Chapter 5 presents a taxonomy of resource scheduling heuristics, three resource scheduling techniques, and a simulation-based evaluation of these techniques. Chapter 6 introduces the problems of initial data placement and fault tolerance for structured presentations. Chapter 7 presents the conclusions and future research directions of this dissertation.

\(^3\)Pages referenced simultaneously are not spread evenly across all disk drives.

\(^4\)Pages referenced simultaneously.
Chapter 2

Related Work

Temporal representation and its manipulation were studied prior to the advent of multimedia. Allen [1] introduced an interval-based representation of time and a reasoning algorithm to maintain it. With the introduction of continuous media types such as video, researchers have studied temporal modeling in the context of databases. Oomoto and Tanaka [31] proposed an object-based data model with interval-inclusion based inheritance. Gibbs et al. [22] proposed a timed stream as the basic element of time-based media. Their study introduced a basic structuring mechanisms for timed streams. Little and Ghafoor [26] presented an interval-based representation of temporal relationships between multimedia streams as part of a database. Their study introduced algorithms for forward, reverse and partial play-out of the streams based on the introduced representation. Schloss and Wynblatt [36] presented a conceptual object-oriented database to represent multimedia data. It divides the multimedia database in layers for modularity and re-usability purposes. It considers a multimedia database as a collection of streams (e.g., audio, video) with temporal relationships, and synchronization policies (i.e., strictness and resynchronization after failure). In sum, these studies focused on the organization of the frames of a stream-based representation and did not consider the spatial and temporal relationships between the objects that participate in individual frames of a video.

Spatial and temporal relationships of objects that constitute a multimedia document were investigated by Weitzman and Wittenburg [44]. Their study defines a presentation as a collection of objects and relations (e.g., author of, title of), a relational grammar, and a collection of constraints (spatial, temporal, graphics
attributes). This abstraction has similarities with our abstraction of structured presentations. The main focus of their study is on the rendering aspect of multimedia documents. It does not address issues such as data organization and retrieval from secondary storage.

Several researchers have studied the complexity of scheduling problems [17, 16, 18]. Their studies assume a pre-defined number of jobs with specific resource requirements and duration. Conversely, in a resource schedule that supports a coordinated display, the duration and resource requirements of the jobs are not pre-defined. 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 either migrate the object directly from $d_1$ to $d_3$ (one step), or migrate the object from disk $d_1$ to $d_2$, and then from $d_2$ to its final destination $d_3$ (two steps). 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 the disk drives used by the migration or replication process are not pre-defined. Therefore, the duration and resource requirements of the jobs in a resource schedule are not pre-defined.

This dissertation studies resource schedules that ensure a coordinated display of structured presentations while minimizing the latency incurred by a display and its total memory requirements. 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 fix-sized cache. Page replacement algorithms based on heuristics (e.g., LRU, MRU, LRU-K [30] and FIFO) are widely used in applications where future page references are unknown. Optimal strategies have been presented for the case where the entire schedule of page references is known [10]; 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 [37] 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 [28].

However, neither heuristic based nor optimal strategies have 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, it strives to assure a coordinated display while minimizing the latency observed by the user and the memory requirement.

A number of studies have investigated techniques to ensure a continuous display of stream-based presentations [9, 29, 3, 2, 34, 42, 20, 35, 6, 23, 27, 12]. These studies conceptualize a presentation as a file that is read sequentially at a pre-specified rate. They also assume a data layout so that the disk reference pattern is regular, e.g., read the first block of a presentation from disks 0, 1, and 2 during the first time cycle, read the second block from disks 3, 4, and 5 during the second time cycle, so on and so forth. Alternatively, a structured presentation consists of a non-linear collection of object references based on a pre-specified display schedule. The disk reference pattern of these presentations is non-regular. A single object might be referenced at different times by the schedule. Moreover, it might be shared by several presentations. These conceptual differences render all the stream-based studies irrelevant. A stream-based study would relate to this dissertation if it assumed that: (1) the frames of a movie are randomly dispersed across the available disks and (2) a frame might be used at different times during a regular display.

A number of studies have analyzed techniques to schedule tasks with real-time deadlines [24, 8, 33]. These studies schedule the CPU with the objective to minimize the total cost to the system (the number of tasks that miss their real-time deadline and their associated cost). This study is novel because we focus on I/O scheduling, data placement, and caching that are key processing components of a data intensive application. There are no deadlines on when a display should start. Instead, once a display starts, the computed resource schedule should support the constraints defined on when objects should be rendered memory resident in preparation for their display. In addition, a presentation should employ the disks containing relevant data (with CPU scheduling literature, a task may access any CPU as there is no dependency on data).
Placement of pages on disks have been studied before. Copeland et al. [11] and Weikum et al. [43] introduced techniques to assign pages to disks aiming to balance the load. These techniques were based on the frequency of access to pages. However, these techniques do not consider temporal relationships between pages (e.g., pages referenced simultaneously by a presentation). Load balancing techniques do not prevent the system from assigning simultaneously retrieved disk pages to the same disk drive.

The technique proposed in [21] places objects across the processors (nodes) in parallel systems based on a shared-nothing architecture. This technique aims to minimize the number of inter-node references and to balance the load. In addition to the heat of objects, it considers the frequency of traversals of inter-object references in an object oriented database. However, this technique would place related objects (e.g., objects referenced simultaneously) in the same node to minimize communication overheads between nodes. On the other hand, placement techniques for structured presentations should strive to distribute pages referenced simultaneously across all disk drives.

Previous studies [39, 7] introduced data layouts and scheduling techniques to support the continuous display of multimedia streams in the presence of disk failure. These studies assume a regular disk reference pattern (e.g., the retrieval of a stream follows this pattern: read the first block from disks 0, 1, and 2 during the first time cycle, read the second block from disks 3, 4, and 5 during the second cycle, so on and so forth). With the structured approach to represent video, the disk reference pattern is non-regular. This pattern depends on the display schedule. Other studies [32] assume unknown disk reference pattern. However, the display of a structured presentation has pre-defined disk reference pattern.
Chapter 3

Conceptual Model of the Object Space

The object space of structured video consists of a collection of objects, spatial and temporal constructs, and rendering features. The spatial and temporal constructs define where in the rendering space and when in the temporal space the component objects are displayed. The rendering features define how the objects are displayed.

A Rendering Space is a coordinate system defined by \( n \) orthogonal vectors, where \( n \) is the number of dimensions (i.e., \( n = 3 \) for 3-D, \( n = 2 \) for 2-D). A spatial construct specifies the placement of a component in the rendering space. For example, the representation of a dining room (4 walls, a table and 8 chairs) consists of a 3-D coordinate system (rendering space) and 13 spatial constructs, one for each object. These spatial constructs define unambiguously the location of the walls, the table and the chairs in the rendering space. They also implicitly define spatial relationships such as the chairs are around the table.

Formally, a Spatial Construct of an object \( o \) maps a vector \( v_o \) in \( o \) into a vector \( v_r \) in the rendering space. The placement of object \( o \) in the rendering space is the rigid translation of \( o \) into the rendering space so that \( v_o \) and \( v_r \) coincide (Figure 3.1).

Analogously, different components are rendered within a time interval, termed Temporal Space. A temporal construct specifies the subinterval in the temporal space when the component object is rendered. For example, consider a presentation that consists of five scenes of one minute each. Suppose that Simba appears in the first three scenes and Mufasa during the last three scenes. The temporal space of the presentation is the time interval \([0, 5]\). The temporal constructs for the five scenes are \([0, 1], [1, 2], [2, 3], [3, 4], \) and \([4, 5]\), respectively. These constructs implicitly define
temporal relationships such as Simba and Mufasa appearing simultaneously during the third scene.

Motion representations have both temporal and spatial constructs. A motion consists of a timed path and a sequence of postures $p_1, \ldots, p_n$ that provide the illusion that the object is moving. These postures must be displayed at the appropriate times and follow the path to provide the continuation of the movement. A motion is represented by a collection of postures, each posture may employ a temporal and a spatial construct. The spatial constructs determine the path followed by the postures and the temporal constructs the timing of their appearances. For example, suppose that there is a scene where Mufasa is chasing Simba. It contains two motions, one for each character. Each motion consists of a sequence of postures with their positioning and timing of appearances. This information implicitly defines relationships such as chasing. This implicit relationship is named “chasing” in the name space.

There are three layers to represent the object space:

(i) Atomic objects that consist of objects participating in the presentation. These objects are indivisible (i.e., they are displayed in their entirety).

(ii) Composed objects that consist of objects constrained using temporal and spatial constructs.

(iii) Rendering features that consist of rendering attributes such as camera position, light sources, etc.
<table>
<thead>
<tr>
<th>RENDRING FEATURES</th>
<th>View point, light sources, etc. of each scene in the movie.</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPOSED OBJECTS</td>
<td>Temporal and spatial association to objects</td>
</tr>
<tr>
<td>ATOMIC OBJECTS</td>
<td>Different postures of Simba: p1, p2, ... A jungle: sc1.</td>
</tr>
</tbody>
</table>

(a) (b)

Figure 3.2: Levels of abstraction in the object space. (a) Description of each level (b) Scene of Simba walking.
Example 1: Suppose Simba is walking along a path in a scene (Figure 3.2). Then, we have the 3-D representations of different postures of Simba in the atomic objects layer. For example, his posture when he starts walking, his posture one second later, etc. They are denoted by p1, p2, etc. These postures might have been originals composed by an artist or generated using interpolation. We also include the 3-D representation of the background (denoted by sc1) in the atomic objects layer.

To represent the motion, we have to specify spatial and temporal constructs in the composed objects layer. The spatial constructs for the motion representation are specified by the curve labeled c1. The curve illustrates the path followed by Simba (i.e., the different positions reached). The vectors control the placement of his postures in the rendering space. The temporal constructs are specified by the triplets: component, starting and ending times, representing the posture of the character and its timing. For example, the point labeled (p1, 0, 1) indicates that the posture p1 appears at time 0 and lasts for 1 second in a scene.

To associate the motion of Simba with the background, we have the spatial and temporal constructs in the composed objects layer represented by c2. The spatial constructs define where in the rendering space the motion and the background are placed. The rendering space is represented by the 3-D coordinate system and the placement of the motion and background by the vectors. The motion is mapped to the vector labeled (c1, 5, 25) and the background to the other vector. The temporal constructs define the timing of the appearances of the background and the motion. For instance, the background sc1 appears at the beginning of the scene while Simba’s motion (c1) starts at the 5th second.

Finally, the rendering features are assigned by specifying the view point, the light sources, etc., for the time interval at which the scene is rendered.

3.1 Atomic Objects

Video applications may represent atomic objects in alternative ways at the physical level, e.g., wire-frame, surface, solid representation, etc. From a conceptual perspective, these physical representations are considered as an unstructured unit.
(i.e., a BLOB). Atomic objects can also be represented at the conceptual level as either:

(i) A procedure that takes some parameters as inputs and constructs a BLOB that represents the object. For example, a geometric figure can be represented by the parameters of its parts, i.e., the radius of a sphere, the length of a side of a square, etc., and a procedure that consumes those parameters and produces a bitmap that represents the object. This type of representation is termed **Parametric**.

(ii) An interpolation of two other atomic objects. For example in animation, the motion of a character can be represented as postures at selected times and the postures in between can be obtained by interpolation. As in animation, this representation is termed **In-Between**.

(iii) A transformation applied to another atomic object. For example the representation of a posture of Mickey Mouse can be obtained by applying some transformation to a master representation. This representation is termed **Transform**.
In Figure 3.3, we present the schema of the type Atomic that describes these alternative representations. The conventions employed in this schema representation as well as others presented in this dissertation are as follows. The names of built-in types (i.e., strings, integers, etc.) are all in capital letters as opposed to defined types that use lower case letters. ANYTYPE refers to strings, integers, characters and complex data structures. A type is represented by its name surrounded by an oval. The attributes of a type are denoted by arrows with single line tails. The name of the attribute labels the arrow and the type is given at the head of the arrow. Multi-valued attributes are denoted by arrows with two heads and single value attributes by arrows with a single head. The type/subtype relationship is denoted by arrows with double line tails. The type at the tail is the subtype and the type at the head is the supertype.

For example, in Figure 3.3 Parametric is a subtype of Atomic, and it has two attributes: Parameters and Generator. Parameters is a set of elements of any type and Generator is a function that maps a set of elements of any type (i.e., Parameters) into a BLOB.

3.2 Composed Objects

Composed objects are collections of objects with spatial and/or temporal constructs.
Definition: A Composed object $C$ is represented by the set:

$$\{(e_i, p_i, s_i, d_i) \mid e_i \text{ is a component of } C, \ p_i \text{ is a vector in the rendering space that defines a spatial construct on } e_i, \ \text{and } [s_i, d_i] \text{ is a subinterval that defines a temporal construct on } e_i\}$$

A composed object may have more than one occurrence of the same component. For example, a character may appear and disappear in a scene. Then, the description of the scene includes one 4-tuple for each appearance of the character. Each tuple specifies the character’s position in the scene and a subinterval when the character appears.

Figure 3.4 illustrates the schema associated to the representation of spatial and temporal constructs in the composed objects layer.

### 3.3 Rendering Features

Rendering features are associated to intervals in the temporal space of a composed object. They are a collection of 2-tuples, $(descriptor, value)$, to represent the description and value of a feature. Figure 3.5 represents the schema associated to the rendering features layer.
Chapter 4

Complexity of Resource Scheduling

Interpreted display of structured presentations defined statically imposes temporal constraints on the retrieval of data from disks. Atomic objects must be retrieved from disks according to the temporal constructs defined in the composed object layer. The retrieval of composed objects and rendering features is negligible as compared with the retrieval of atomic objects. Therefore, we focus on resource schedules that satisfy the temporal constraints imposed on atomic objects. This chapter defines such resource schedules and presents time complexity of computing a resource schedule for both single and multi-disk architectures.

4.1 Statement of the Problem

Interpreted display of structured presentations defined statically imposes temporal constraints in the retrieval atomic objects and rendering features from disks.

We discretize time into fix-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 4.1). A collection of objects, represented as a set of disk pages, are displayed during each time interval. We represent a display schedule as a sequence of time intervals. Each time interval $i$ is tagged with a collection of disk pages, termed $P_i$, displayed during interval $i$. We assume that the system can support a coordinated display if the set $P_i$ of pages displayed during interval $i$ is memory resident at instants $i$ and $i + 1$, $0 \leq i < m$. 
Figure 4.1: Time interval and time instant

**Definition:** A *display schedule* is a sequence \( \{P_0, \ldots, P_{m-1}\} \) of disk pages 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. 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, \( i - 2, i - 3, \ldots, \) 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
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>Number of time intervals in the presentation</td>
</tr>
<tr>
<td>$t$</td>
<td>Duration of a time interval</td>
</tr>
<tr>
<td>Time interval $i$</td>
<td>$[i, i + 1]$</td>
</tr>
<tr>
<td>Instant $i$</td>
<td>The beginning of time interval $i$</td>
</tr>
<tr>
<td>$S_i$</td>
<td>Pages in memory at instant $i$</td>
</tr>
<tr>
<td>$P_i$</td>
<td>Pages containing objects displayed during interval $i$</td>
</tr>
<tr>
<td>$P_i^d$</td>
<td>Pages in $P_i$ that reside in $d$</td>
</tr>
<tr>
<td>$F_i$</td>
<td>Pages retrieved from disk onto memory during time interval $i$</td>
</tr>
<tr>
<td>$F_i^d$</td>
<td>Pages retrieved from disk $d$ onto memory during time interval $i$</td>
</tr>
<tr>
<td>$U_i^d$</td>
<td>Pages written to disk $d$ during time interval $i$</td>
</tr>
<tr>
<td>$K_i$</td>
<td>Pages discarded from memory during time interval $i$</td>
</tr>
<tr>
<td>$R_i^d$</td>
<td>Read page $a$ from disk $j$</td>
</tr>
<tr>
<td>$W_{j}^{a}$</td>
<td>Write page $a$ to disk $j$</td>
</tr>
<tr>
<td>$D$</td>
<td>Disk drives in the system</td>
</tr>
<tr>
<td>$B$</td>
<td>Maximum number of pages read/written by a drive during an interval</td>
</tr>
<tr>
<td>$B_i^d$</td>
<td>Disk bandwidth available at drive $d$ during interval $i$</td>
</tr>
<tr>
<td>$C$</td>
<td>Number of memory buffers in the system</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of clauses</td>
</tr>
<tr>
<td>$k$</td>
<td>Number of variables</td>
</tr>
<tr>
<td>$R$</td>
<td>Set of replications to be scheduled</td>
</tr>
<tr>
<td>$[0, N]$</td>
<td>Period when $R$ must be scheduled</td>
</tr>
<tr>
<td>$A$</td>
<td>System load during $[0, N]$</td>
</tr>
<tr>
<td>$p$</td>
<td>Start up latency in time intervals</td>
</tr>
</tbody>
</table>

Table 4.1: List of terms used in this paper and their definitions

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\(^1\) \(\mathcal{P}\) to compute a schedule of page retrievals, writes, and discards 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 subset 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 \(i \in [-p, m]\), \(|S_i| \leq C\).

---

\(^1\)The placement of data maps a page identifier and a time interval into one or more disk drives.
(iv) The retrievals respect the placement of data. More formally, 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 $i$.

To illustrate these concepts, consider a display schedule that consists of three time intervals: $P_0=\{a, b\}$, $P_1=\{c, d\}$, and $P_2=\{e, f\}$. Assume that the system consists of four disks ($D=4$), each with the bandwidth to retrieve one disk page during a time interval ($B=1$). Assuming that all the referenced pages reside on disk one, Figure 4.2 (a) shows a resource schedule that supports a coordinated display without changing the data placement (i.e., only retrievals and discards). $R_1^a$ denotes that page $a$ is retrieved from disk one. 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., $f$) or pre-fetched at an earlier time interval (e.g., $a$). 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 increases 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 four disks could accommodate the retrieval of four 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 4.2 (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 two (denoted as \( W_2^e \)) during time interval -3. This allows the system to free the memory frame occupied by \( e \) at time instant -2 and, utilize disk number two to retrieve \( e \) during time interval one to satisfy the display schedule. With this schedule, only 4 memory frames are required at instant one \( (a, b, c, d) \). If a page is replicated, its original copy is termed primary while its other copies are termed secondary. A disadvantage of constructing secondary copies of a page is that it wastes both disk bandwidth and disk space. Once the display is completed, the system may either (1) allow both the primary and secondary copies of \( e \) to continue to exist or (2) maintain one copy of \( e \). In the first case (and the second case when \( e \) is migrated or the primary copy of \( e \) on disk 1 is deleted to make its secondary copy on disk 2 as its primary), a subsequent display of the schedule would incur a three time interval startup latency because there is no need to replicate/migrate \( e \).

### 4.2 Complexity of Resource Scheduling for Single-Disk Architectures

This section presents an optimal algorithm that computes a resource schedule for a single-disk architecture \( (D = 1) \). This optimal algorithm has a worst case time complexity \( O(n \lg n) \) and minimizes both the startup latency and the memory requirement. The resulting resource schedule consists of retrievals and discards (i.e., there is no writes). Therefore, it can be expressed as a sequence of memory states \( \{S_0, \ldots, S_m\} \). The pages retrieved from disk to memory and the pages discarded from memory can be derived from the memory states as follows: \( F_i = S_{i+1} - S_i \) and \( K_i = S_i - S_{i+1} \).
The optimal algorithm (Figure 4.3) consumes a display schedule \(\{P_0, \ldots, P_{m-1}\}\) and the number of pages the target disk can retrieve during a time interval \(B\) as input to produce a sequence \(\{S_0, \ldots, S_m\}\) of memory states. The latency is \(|S_0|/B\) time intervals. The memory requirement of the resource schedule is \(\max_{0 \leq i \leq m} |S_i|\).

The request is rejected if this quantity exceeds \(C\), the maximum number of memory pages available. It traverses the display schedule backwards to determine the disk pages to retrieve during each time interval. It assumes that only pages referenced during the final time interval are in memory at the end of the display \((S_m = P_{m-1})\).

To minimize pre-fetching, it strives to retrieve during interval \(i\) all pages in memory at instant \(i + 1\) \((S_{i+1})\) that are not referenced during intervals \(i\) and \(i - 1\) \((P_i \cup P_{i-1})\).

If all pages in \(S_{i+1} - (P_i \cup P_{i-1})\) cannot be retrieved during interval \(i\), then it pushes the retrieval of some pages to an earlier time interval. Pages in \(S_{i+1} - (P_i \cup P_{i-1})\) with the lowest most recent instant at which they are referenced are retrieved during interval \(i\), the other retrievals are pushed to earlier time intervals. For each time interval \(i\) and page \(a\), we define \(\text{last}(a, i)\) to be the most recent instant \(j\) \((j \leq i)\) at which page \(a\) was referenced. More formally, let \(\text{last}(a, 0) = -\infty\) and for each \(i > 0\) define

\[
\text{last}(a, i) = \begin{cases} 
  i & \text{if } a \in P_{i-1} \\
  \text{last}(a, i-1) & \text{otherwise}
\end{cases}
\]

We demonstrate the scheduler using an example. Subsequently, we prove its optimality.

**Example 2:** Consider a display schedule that consists of eight time intervals:

\[
\begin{align*}
  P_0 &= \{a, b, c\} & P_1 &= \{a, d\} & P_2 &= \{a, e\} & P_3 &= \{a, c, f\} \\
  P_4 &= \{a, b\} & P_5 &= \{a, e\} & P_6 &= \{a, f, g\} & P_7 &= \{a\}
\end{align*}
\]

Figure 4.4 illustrates the resource schedule computed by greedy for this presentation when \(B = 1\). The memory required to display the presentation is 4 page frames (i.e., \(\max_{0 \leq i \leq 8} |S_i|\)). And the latency is 3 time intervals.

---

\(^2\)The retrievals before the display are not included because they are determined by the state of memory at the beginning of the display \((S_0)\).
**Greedy Scheduler** \((\langle P_i, \ldots, P_{m-1} \rangle, B)\)

\[ S_m = P_{m-1} \]

\[
\textbf{for} \ i = m - 1 \ \textbf{to} \ 0 \ \textbf{do} \\
\quad \textbf{if} \ i = 0 \ \textbf{then} \quad RS_i = P_i \\
\quad \textbf{else} \quad RS_i = P_i \cup P_{i-1} \\
\quad \text{Let } N_i \text{ be the sequence of pages in } S_{i+1} - RS_i, \\
\quad \text{sorted by } \text{last}(\cdot, i). \\
\quad \textbf{if} \ |N_i| \leq B \ \textbf{then} \quad F_i = N_i \\
\quad \textbf{else} \quad F_i = \{\text{the first } B \text{ pages in } N_i\} \\
\quad S_i = RS_i \cup (N_i - F_i) \\
\textbf{end for} \\
\]

Figure 4.3: The greedy algorithm

---

![Diagram](image)

\( S_m = P_{m-1} \)

\[
\textbf{for} \ i = m - 1 \ \textbf{to} \ 0 \ \textbf{do} \\
\quad \textbf{if} \ i = 0 \ \textbf{then} \quad RS_i = P_i \\
\quad \textbf{else} \quad RS_i = P_i \cup P_{i-1} \\
\quad \text{Let } N_i \text{ be the sequence of pages in } S_{i+1} - RS_i, \\
\quad \text{sorted by } \text{last}(\cdot, i). \\
\quad \textbf{if} \ |N_i| \leq B \ \textbf{then} \quad F_i = N_i \\
\quad \textbf{else} \quad F_i = \{\text{the first } B \text{ pages in } N_i\} \\
\quad S_i = RS_i \cup (N_i - F_i) \\
\textbf{end for} \\
\]

Figure 4.4: Optimal resource schedule
Consider the loop iteration when \( i = 5 \) (i.e., the schedule of retrievals for time interval 5). The state of the variables is as follows: \( S_5 = \{a, e, f, g\} \), \( RS_5 = \{a, b, e\} \), \( N_5 = \{f, g\} \). The system can only retrieve either \( f \) or \( g \), but not both because \( B = 1 \). The scheduler retrieved \( g \) because \( f \) must be retrieved prior to time interval 3 (\( f \in P_3 \)), while \( g \not\in P_j \) for \( j < 5 \). In other words, \(-\infty = \text{last}(g, 5) < \text{last}(f, 5) = 4\). Therefore, \( f \) remains memory resident during three time intervals (from interval 3 to 6). If the scheduler retrieves \( f \) instead of \( g \) then the system must allocate some bandwidth to retrieve \( g \) at an earlier time interval. There are two alternative approaches: First, increase the latency and retrieve \( g \) prior to time interval 0. Second, retrieve \( g \) instead of \( b \) (e) during time interval 3 (4) and render \( b \) (e) memory resident from time interval 1 to 4 (3 to 5). The first alternative increases latency to 4 time intervals while the second increases the memory requirement to 5 page frames because \( S_2 \) (\( S_4 \)) now consists of \( \{a, b, c, d, e, f\} \).

**Theorem 4.1** The resource schedule produced by the greedy scheduler is optimal.

**Proof:** Consider a request \( R_q = (\langle P_0, \ldots, P_{m-1} \rangle, B) \). Let \( \mathcal{G} = S_0', \ldots, S_m' \) denote the resource schedule produced by the greedy scheduler on \( R_q \). It suffices to prove that for any resource schedule \( \mathcal{S} = S_0, \ldots, S_m \) for \( R_q \), \( |S'_i| \leq |S_i| \) for all \( i = 0, \ldots, m \). To prove this, we transform \( \mathcal{S} \) into \( \mathcal{G} \), state by state, starting with \( S_m \) and ending with \( S_0 \). Specifically, we show inductively that for each \( i \), there is a way of transforming states \( S_0, \ldots, S_m \) into a sequence of states \( S'_0, \ldots, S'_m \) so that these transformations do not require extra memory, nor do they ever exceed the available bandwidth. More formally, for all \( j = 0, \ldots, m \), \( |S'_j| \leq |S_j| \), and for all \( j \geq i \), \( S'_j = S'_j \). It will follow that the memory requirement of \( \mathcal{G} \) at each instant is optimal. Therefore the total memory requirement (\( \max_{0 \leq i \leq m} |S'_i| \)) is minimized. Moreover, the latency is also minimized because it is determined by \( |S'_0| \).

For the basis of the induction consider \( S_m \), then either \( S_m = S'_m = RS_m \) or \( S'_m \subseteq S_m \). We can eliminate all pages in \( S_m - S'_m \) (i.e., \( S'_m = S'_m \)) because they are not displayed during time interval \( m - 1 \). For the induction step, suppose that we have transformed \( \mathcal{S} \) so that \( S'_j = S'_j \) for all \( j = i + 1, \ldots, m \). (To simplify notation, from now on we denote transformed sequences by \( \mathcal{S} \) instead of \( S'_i \).) Consider the possible differences between \( S_i \) and \( S'_i \). If \( S_i = S'_i \), then there is nothing to do.
Suppose that $S_i - S^g_i \neq \emptyset$. Hence there are pages $a \in S_i - S^g_i$ that $S$ has resident, and $G$ does not. Since $a \not\in S^g_i$, then $a \not\in RS_i$. So these sets can be equated by removing $a$ from $S_i$, and propagating this change backwards. There are several cases needed to realize this backwards propagation. If $a \not\in S_{i+1}$ then eliminate it from $S_i$. If $a \in S_{i+1} = S^g_{i+1}$, then $a$ can be retrieved during time interval $i$ in $S$ as long as there is sufficient bandwidth available to accommodate this retrieval. In this case $a$ is eliminated from $S_i$. Assume that there is insufficient bandwidth to support the retrieval of $a$. Hence $S$ is retrieving $B$ pages during $i$. It follows that $G$ retrieves $a$ during $i$ because $a \in S^g_{i+1}$ and $a \not\in S^g_i$. Moreover, it must be the case that there is some other page $b \neq a$ that is retrieved by $S$ during $i$, but not retrieved by $G$. Because $G$ retrieves at most $B$ pages during every time interval. However, by the inductive hypothesis, $S_{i+1} = S^g_{i+1}$, so $b \in S^g_i$, thus $b \not\in RS_i$. Therefore, \( \text{last}(a, i) \leq \text{last}(b, i). \) To equate $S_i$ and $S^g_i$, $S$ will now exchange the intervals during which $a$ and $b$ were retrieved. There are two cases. First, if $a$ is retrieved after \( \text{last}(b, i) \) then let $j$ to be the interval when $a$ was retrieved. In this case, one swaps the retrieval of $a$ with that of $b$, so that

$$
\text{for } k = j + 1, \ldots, i, \ S_k = (S_k - \{a\}) \cup \{b\}
$$

(4.1)

If $b$ was already present in memory when $a$ was retrieved in $S$, then no retrieval is actually needed; the modifications maintain $b$ memory resident until interval $i$. Second, if $a$ is retrieved previous to or at \( \text{last}(b, i) \) then we set $j = \text{last}(b, i)$. Since $b$ is in $S_j$, maintain $b$ in memory until $i$ as in Equation 4.1. These transformations increase neither the memory nor the bandwidth requirement of an interval.

Symmetrically, it may be the case that $S^g_i - S_i \neq \emptyset$. Thus $G$ has made some page $b$ memory resident that is not resident in $S$ at $i$. However, since $S$ is a resource schedule that satisfies the display schedule, it must be the case that each such $b$ is not in $RS_i$. This implies that $b \in S^g_{i+1} = S_{i+1}$ as well and $G$ retrieved $B$ pages during $i$. Thus, there must be some page $a$ that $G$ retrieves during $i$ that $S$ does not retrieve. The nature of the greedy algorithm implies that \( \text{last}(a, i) \leq \text{last}(b, i). \) But by the inductive hypothesis, $S$ has $a \in S_{i+1}$. Since $S$ does not retrieve $a$ during time $i$, it must have $a \in S_i$. To equate $S_i$ and $S^g_i$, modify $S$ so that $a$ is retrieved instead of $b$ during $i$, and propagate the change through previous intervals as above.
Note that, there are transformations for each possible difference between $S_i$ and $S^g_i$. Therefore, the greedy algorithm produces an optimal resource schedule. □

Since the greedy schedule minimizes the memory requirement at each interval, it must be the case that whenever it is possible to schedule the request using memory $C$, such a schedule will be found. It is not difficult to see that an optimal resource schedule can be computed in time $O(n \log n)$ where $n = \sum_{i=0}^{m} |P_i|$.

### 4.3 Complexity of Resource Scheduling for Multi-Disk Architectures

This section demonstrates the following: (1) deciding whether there is a schedule to change the placement of data across disk drives based on a given system load is NP-Complete, (2) the computation of a resource schedule that supports a coordinated display and yields the minimum latency is NP-Hard, and (3) the computation of the minimum memory requirements to display a coordinated presentation within a pre-specified latency is NP-Hard. We first introduce some simplifications in the notation used in this section, then introduce new concepts, and finally demonstrate the three complexity results in the order above.

A distinction between replication and migration is the availability of a page once the process completes. Once a secondary copy of page $a$ is constructed on disk $d_2$ (with its primary copy on disk $d_1$), $a$ can be retrieved from either $d_1$ or $d_2$. On the other hand, if the system migrates $a$ from $d_1$ to $d_2$, then $a$ is on disk $d_2$ and no longer can be retrieved from $d_1$. For the purpose of the proofs this distinction is redundant, because the display schedules reference page $a$ only once after the migration or replication completes. In either case (migration or replication), the system would retrieve $a$ from $d_2$. Retrieving $a$ from $d_1$ after constructing a secondary copy of $a$ on $d_2$ implies that the system has performed wasteful work (the replication was

---

3 Another distinction is disk space. However, this dissertation assumes that disk space is not a constraint (cost of a megabyte of disk is dropping rapidly).

4 The display schedules considered in the presented proofs reference a page only during one continuous period. Therefore, if a page is replicated or migrated, then it is retrieved only once after the migration or replication.
unnecessary). Then, the distinction between replications and migrations vanishes and we refer to both as replications.

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

**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 \(i\), \(0 \leq i \leq N-1\), consisting 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 replication is denoted as \((a, \text{source}, \rightarrow \{\text{target}_1, \ldots, \text{target}_n\})\), where \(a\) is the disk page to be copied, \(\text{source}\) is the disk drive that contains the primary copy of \(a\), and \(\{\text{target}_1, \ldots, \text{target}_n\}\) are alternative drives to contain the secondary copy of \(a\). There is only one drive (\(\text{source}\)) that contains the primary copy of \(a\), because the data placement defined in the proofs assigns each page to only one drive. To construct a secondary copy, the system can utilize intermediate disk drives. For example, consider the replication \((a, 5 \rightarrow \{2, 4, 6\})\) to be scheduled in a system with 10 disk drives \((D = 10)\). Assume disk 6 is to contain the secondary copy of \(a\). One possible schedule (Figure 4.3) 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 disk 6. Using disks 7 and 8 as intermediate disks prevented the system from staging \(a\) in memory during intervals \([3, 4]\) and \([7, 8]\). This process constructs three secondary copies of \(a\) residing on disks 6, 7, and 8.

**Definition:** Assume a collection \(R (\{r_1, \ldots, r_r\})\) of replications and a system load \(A\) for a period \([0, N]\). A schedule for \(R\) on \(A\) maps each replication \(r_i = (a, \text{source} \rightarrow \{\text{target}_1^i, \ldots, \text{target}_n^i\})\) into a sequence \((\text{source}, t_i^1), (d_i^1, [t_i^1, t_i^2]), \ldots,)\).
Figure 4.5: Schedule of a replication.

\[(d_k, [t_{2k-1}, t_{2k}], \{target_{p}, t_j\})\] where \(k\) is the number of intermediate disks. Such that:

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

(ii) For each \(j \in [1, k]\), 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 construct a secondary copy of \(a\) on disk \(target^i_p\) during interval \(t^i_t\).

(iv) The replications are scheduled within the period \([0, N]\) and the reads and writes are scheduled in the right order: \(0 \leq t^i_s < t^i_t \leq N\) and for each \(j \in [1, 2k]\) and \(l \in [1, 2k]\), \(0 \leq t^i_s < t^i_j < t^i_l \leq N\), and \(j < l\) implies \(t^i_j < t^i_l\).

(v) A secondary copy is created on 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: \((t^i_s, t^i_t], (t^i_1, t^i_3], \ldots, (t^i_{2k-1}, t^i_t]\).

(vii) The intermediate drives are different from the source and target drives, otherwise the system would be performing wasteful work: for each \(j \in [1, k]\), 
\[d^i_j \not\in \{source, target^i_1, \ldots, target^i_{n_i}\}\]

We first demonstrate that deciding whether there is a schedule for a set \(R\) of replications based on a pre-specified system load \(A\) is NP-Complete. Then, we show that the computation of a resource schedule that supports a coordinated display and yields the minimum latency is NP-Hard. The identity of the disk page in a replication is irrelevant for the proofs and is therefore omitted.
4.3.1 Replications Scheduling

To show that deciding whether there is a schedule for replications $R$ on a system load $A$ defined over a period $[0, N]$ is NP-Complete, we reduce the SAT problem into 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 disjuncts $v_i$ and $\neg v_i$ for some variable $v_i$ (if this is the case, remove such clauses because they are true for any truth assignment).

We first introduce a polynomial algorithm $SAT2RepSc$ that transforms any instance $C_1, \ldots, C_n, v_1, \ldots, v_k$ of SAT into an instance $R, A, N$ of the replications scheduling problem, i.e., is there a schedule for replications $R$ on a system load $A$ defined over $[0, N]$?

The definition of $SAT2RepSc$ is as follows. Let $N$ be $(2n + 2) \cdot 2k$ and $A$ be defined as in Figure 4.6. Thick time instants denote that 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 4.6):

$$M_0 > 0$$

Thin time instants (Figure 4.6) 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 $w_i, s_i, d_i, t_i, u_i, d_{ij}, c_{ij}$ on time intervals (Figure 4.6) denote disk drives with available bandwidth during the interval. If a label has a superscript $+$ then the disk has bandwidth available for the retrieval/write of at least one page during the interval. Otherwise, the disk has bandwidth available for only one page retrieval/write.
during the interval. The expressions \( v_i \in C_j \) and \( \neg v_i \in C_j \) denote that \( v_i \) and \( \neg v_i \) are disjuncts of \( C_j \) respectively. These expressions specify whether a disk has sufficient bandwidth for one retrieval/write. To illustrate, consider time 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 interval 2. The other disk drives do not have bandwidth available during interval 2. The system load for interval 2 is as follows: \( M_2 = 0; \) for each \( j \not\in \{ w_3, d_{11}, d_2 \}, B_j^i = 0; B_2^w > 0; B_2^{d_{11}} = 1; \) if \( v_1 \) is a disjunct in \( C_2 \) then \( B_2^{d_2} = 1, \) otherwise \( B_2^{d_2} = 0. \)

Let \( R \) be defined as follows: For each variable \( v_i, s_i \rightarrow \{ t_i, u_i \} \) is a replication in \( R. \) For each clause \( C_j, d_j \rightarrow \{ d_{ji} \mid v_i \in C_j \} \cup \{ e_{ji} \mid \neg v_i \in C_j \} \) is a replication in \( R. \)

\( \square \)

To illustrate the transformation consider the example in Figure 4.7. The patterns in Figure 4.7 (c) are used to denote the alternatives to schedule replications associated to variables and clauses. Note that the drives \( w_j \) cannot participate in any schedule because they are neither a source nor a target of a replication. Moreover, they cannot be intermediate drives because they have bandwidth available only during one time interval. Schedules of replications compete with each other for disk bandwidth and memory. For example, \( \{ 0, (1, d_{11}) \} \) (an alternative to schedule the replication associated to \( C_1 \)) compete with \( \{ 0, (d_{11}, [1, 2]), (d_{21}, [3, 4]), (t_1, 5) \} \) (an alternative to schedule the replication associated to \( v_1 \)) 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 replication associated to \( v_1 \) during the interval \([6, 12] \) denoted by the line besides \( v_1 \) (Figure 4.7 (d)) with the pattern of \( v_1 \) (Figure 4.7 (c)). Then, the replication associated to \( C_1 \) can be scheduled during the interval \([0, 2] \) denoted by the line besides \( \pi_T \) (Figure 4.7 (d)) with the pattern of \( C_1 \) (Figure 4.7 (c)). These lines do not cover overlapping intervals, then they would not compete for disk bandwidth nor for memory.

To prove that the reduction from SAT to the replications scheduling problem is correct, we start by showing that if an instance of SAT has a solution then the corresponding replications scheduling instance has a solution.
Figure 4.6: System load yielded by SAT2RepSc.
Figure 4.7: Example of reduction of a SAT instance into a replications scheduling instance. (a) SAT instance, (b) Corresponding set $R$ of replications, (c) Patterns to denote alternative schedules of replications associated to variables and clauses, (d) Corresponding system load $A$. 
Lemma 1 Let $R, A, N$ be the output of $\text{SAT2RepSc}(C_1, \ldots, C_n, v_1, \ldots, v_k)$. 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 for replications $R$ on $A$ during interval $[0, N]$.

Proof: See Appendix A. □

We now prove the other direction, if the replications scheduling instance yielded by $\text{SAT2RepSc}$ has a solution then the input SAT instance has a solution.

Lemma 2 Let $R, A, N$ be the output of $\text{SAT2RepSc}(C_1, \ldots, C_n, v_1, \ldots, v_k)$. If there is a schedule for replications $R$ 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. □

With Lemmas 1 and 2 in combination with the fact that the transformation $\text{SAT2RepSc}$ is a polynomial time algorithm, we conclude the following.

Theorem 4.2 Deciding whether there is a schedule for a set $R$ of replications on a system load $A$ over a period $[0, N]$ is NP-Complete.

4.3.2 Resource Scheduling

This section demonstrates that: (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 display a structured 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 $\text{SAT2ResSc}$ that transforms any instance $C_1, \ldots, C_n, v_1, \ldots, v_k$ of SAT into an instance $\{P_0, \ldots, P_{m-1}\}, P, B, C, D$ 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 placement of data $P$? We then show that given an instance $C_1, \ldots, C_n, v_1, \ldots, v_k$ of SAT, $\text{SAT2ResSc}(C_1, \ldots, C_n, v_1, \ldots, v_k)$ has a solution if and only if $\text{SAT2RepSc}(C_1,$
\[ C_n, v_1, \ldots, v_k \] has a solution. Then, because of Lemmas 1 and 2, 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.

The definition of \( SAT2ResSc \) is as follows. It first computes a replication scheduling instance \((R, A, N)\). Next, based on this instance, it computes the resource scheduling instance \((D, B, C, \{P_0, \ldots, P_{m-1}\}, \mathcal{P})\). Its details are as follows, let \( R, A, N \) be \( SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k) \). Let \( m \) be \( N+n+k \), \( D \) be \( m+1+2\cdot k+n+2n\cdot k \), \( MaxCard \) be \( \max\{ \text{cardinality of } \text{target}_i \mid i \in [1,n] \} \) and \( \text{source}_i \rightarrow \text{target}_i \) is the replication associated to \( C_i \), \( B \) be \( MaxCard + 1 \), \( q \) be \( B \cdot (D - 1) \), and \( C \) be \( 2 \cdot q \). Let \( \{P_0, \ldots, P_{m-1}\} \) be the display schedule in column \( P_i \) of Table 4.2. Let the placement of pages \( \mathcal{P} \) on the disk drives be so that: (1) any resource schedule must include a replication schedule for \( R \), and (2) the system load after applying resource schedule \( greedy(\{P_0, \ldots, P_{m-1}\}, B \cdot D) \) (Column \( \hat{S}_i \) in Table 4.2) would be identical to \( A \) during the period \([0, N]\). To achieve these two goals, the placement of pages referenced by the display schedule is as follows:

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

**Placement of pages retrieved during interval \(-1 \ (F_{-1})\):**

\( a_1, \ldots, a_q \) would be placed on 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 \( w_0 \).

**Placement of pages retrieved during even intervals in \([0, N-1] \ (F_i, \text{for } i \in \{0, 2, \ldots\})\):**

For the assignment of the \( q - 1 \) pages retrieved during even time intervals before instant \( N \), we have two cases:

1. There are three disks \((x, y, w_i)\) with available bandwidth during the interval in \( A \): assign the first \((D - 3) \cdot B \) pages to drives different from \( x, w_i \) \( B \) pages to each drive), assign the next \( B - 1 \) to drive \( x \), the next \( B - 1 \) to drive \( y \), and the last page to \( w_i \). Then drives \( x \) and \( y \) would have disk bandwidth available for one page retrieval/write each. And, drive \( w_i \) would have disk bandwidth available for \( B - 1 \) retrievals/writes.
(2) There are two disks \((x, w_i)\) with available bandwidth during the interval in \(A\): assign the first \((D - 2) \cdot B\) pages to drives different from \(x, w_i\) (\(B\) 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 \(w_i\) for \(B\) retrievals/writes.

**Placement of pages retrieved during odd intervals in** \([0, N - 1]\) \((F_i, \text{ for } i \in \{1, 3, \ldots\})\):

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

**Placement of pages retrieved during** \([N, N + k - 1]\) \((F_i, \text{ for } i \in [N, N + k - 1])\):

The assignment of the \(q\) pages retrieved during each interval \(i\) is as follows: Let \(t_{i-N+1}, u_{i-N+1}\) be targets of the replication 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 \(s_i\) would have exceeded the disk bandwidth requirement by one page, and drive \(w_j\) would have bandwidth available for \(B - 1\) retrievals/writes.

**Placement of pages retrieved during** \([N + k, N + k + n - 1]\) \((F_i, \text{ for } i \in [N + k, N + k + n - 1])\):

The assignment of the \(q\) pages retrieved during each interval \(i\) is as follows: Let \(x_1, \ldots, x_j\) be the target disk drives of replication 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.
Table 4.2: Display schedule \( \{P_0, \ldots, P_{m-1}\} \) and schedule of retrievals and discards \( \{\hat{S}_0, \ldots, \hat{S}_m\} \)

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.

\[ \square \]

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

Let \( R, A, N = SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k) \). 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:

1. There is a resource schedule for \( \{P_0, \ldots, P_{m-1}\} \) consisting of a schedule \( Ret = \{\hat{S}_0, \ldots, \hat{S}_m\} \) of retrievals and discards\(^5\) and a schedule \( Rep \) for replications \( R \) over \([0, N]\) on the system load resulting from applying \( Ret \), such that: the system load during period \([0, N]\) resulting from applying \( Ret \) is identical to \( A \).

\(^5\)The retrievals \( (F_i) \) and discards \( (K_i) \) in \( Ret \) can be derived from the memory states: \( F_i = S_{i+1} - S_i \) and \( K_i = S_i - S_{i+1} \).

\(^6\)The system has total availability of resources during interval \([-p, 0]\). The system load for this interval is \([<B, \ldots, B, C>, \ldots, <B, \ldots, B, C>]\).
(2) Ret does not pre-fetch pages. Therefore for any resource schedule, for each instant \(i, 0 \leq i \leq m - 1\), \(\hat{S}_i \subseteq S_i\) (Table 4.2).

(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 replication. 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 \(SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)\) has a solution.

**Lemma 3** Let \(R, A, N\) be the output of \(SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)\). 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 \(RS\) for replications \(R\) 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:** See Appendix B. □

We now prove the other direction of the equivalence.

**Lemma 4** Let \(R, A, N\) be the output of \(SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)\). 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 replications \(R\) on \(A\) during \([0, N]\).

**Proof:** See Appendix B. □

It is easy to see that the computation of \(SAT3ResSc\) is polynomial time. Therefore, because of Lemmas 1, 2, 3, and 4, we conclude:

**Theorem 4.3** Deciding whether there is a resource schedule that yields the latency of one time interval for a given display schedule is NP-Complete.
**Corollary 1**  *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 of the memory capacity 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 2**  *Computing the minimum memory requirements to display a structured presentation within a pre-specified latency is NP-hard.*
Chapter 5

Scheduling Approaches

In this chapter, we devise heuristics to compute resource schedules that support the coordinated display of structured presentations. Our use of heuristics is justified with the NP-Hard results of Chapter 4. This chapter introduces a taxonomy of possible heuristics for resource scheduling. Then, it presents three heuristics and evaluates their performance using a simulation study. As a yard stick, we employ a theoretical minimum to identify the best heuristic.

Figure 5.1 shows a taxonomy of the possible approaches. They are categorized into memory-based, replication-based, and migration-based. Memory-based employs pre-fetching to solve the bottlenecks caused by unbalanced placement of data across the disk drives. It has no impact on the placement of data. Example of Figure 4.2(a) is an illustration of this approach. Replication-based detects bottlenecks and identifies those pages that cause the bottlenecks. It resolves bottlenecks by constructing secondary copies of these pages during earlier time intervals. Example of Figure 4.2(b) is an illustration of this technique. With a transient replication, the system deletes the newly constructed secondary copies at the end of a display. In the example of Figure 4.2(b), transient replication would delete the secondary copy of $e$ assigned to disk 2 at the end of the display.

With persistent replication, the display of a presentation might produce a new organization of data. This technique is categorized into single-copy, and multi-copy. With single-copy persistent replication, disk space is a scarce resource and the system maintains only one copy of the page at the end of display. In the example of Figure 4.2(b), this technique might delete the primary copy of $e$ from disk number one, making its secondary copy on disk 2 as its primary copy. One approach to decide
which copy to maintain as the primary is as follows. The system monitors the number of references to different copies during the display and maintains the most frequently referenced as the primary copy by deleting the rest. With multi-copy persistent replication, the system maintains the secondary copies of a page on disk. The expectation with both single-copy and multi-copy persistent replication is that the new placement (with possibly additional secondary copies) minimizes both the incurred startup latency and the amount of required memory when the presentation is displayed a second time. In the example of Figure 4.2(b), with a new copy of \( e \) on disk two, the subsequent display of the schedule incurs a three time interval startup latency and requires 4 memory frames.

With migration, when the system migrates \( e \) to disk two during interval -3, it deletes the primary copy of \( e \) from disk one. This is useful when disk space is scarce and cannot accommodate secondary copies during the display. With persistent migration, the new assignment of \( e \) persists at the end of display. With transient migration, the system remembers the original placement of data across the disks and restore it at the end of the display. This would require additional disk operations at the end of a display, potentially delaying the display of other presentations. A replication-based schedule has lower latency and memory requirements than the corresponding migration-based schedule. The reason for this is as follows.
Assume the primary copy of page $a$ resides on disk $d_1$. With a secondary copy of this page on $d_2$, the probability of a subsequent display encountering bottlenecks on both $d_1$ and $d_2$ is lower than encountering a bottleneck on $d_2$, if the primary copy of $a$ was migrated to $d_2$. Our main objective in this study is to minimize latency and memory requirements, therefore we focus on memory-based and replication-based resource scheduling and leave migration-based scheduling as a future research direction (Chapter 7).

5.1 Memory-Based Resource Schedule

This approach employs memory to pre-fetch pages in order to resolve bottlenecks. It has no impact on the placement of data across the disks. Given a display schedule and the current placement of data across the $D$ disks, this technique extracts the display schedule for each disk based on the pages that reside on that disk. It invokes the greedy scheduler (Figure 4.3) using the display schedule of each disk to compute a resource schedule\(^1\) for that disk. The union of these $D$ resource schedules yield a final resource schedule for the display. The disk with the longest startup latency ($\sigma$) determines the overall latency incurred by the display. For the given placement of data, this resource schedule minimizes the amount of memory required because the greedy scheduler minimizes the memory requirement at each instant $i$ for a single disk (Theorem 4.1). By minimizing the number of pages that constitute $S_0$, the greedy scheduler minimizes the incurred latency. This technique fails to display an object when the number of pages that constitute $S_i$ exceeds $C$. In this case, the system must employ the replication technique of Section 5.2.

5.2 Replication-Based Resource Schedule

This technique solves the bottlenecks created by unbalanced data placement using replication. It first detects the bottlenecks and then schedule replications to resolve them. It consists of three stages. During the first stage, it employs the greedy

\(^1\) A resource schedule that consists of retrievals and discards. i.e., for each $i$, $U_i^0 = \ldots = U_i^{D-1} = \emptyset$. 

46
method of Section 5.1 to construct a resource schedule for each disk. It constructs $D$
resource schedules (one for each disk) for instant $m - 1$. Disk $d_i$ is a bottleneck if the
display schedule requires more than $B$ pages from this disk during time interval $m - 1$.
$|N_{m-1}| - B$ pages out of the $|N_{m-1}|$ referenced pages (see Figure 4.3 for definition)
should be replicated. The disks that can serve as target candidates to hold these
pages are those with idle disk bandwidth during interval $m - 1$. In the presence of
candidate disks, the following record is appended to a list $L$ of replications: {page-
id, time interval $m - 1$, bottleneck disk $d_i$, a list of target candidate disks}. If no
target candidate disk exists then the system must pre-fetch these $|N_{m-1}| - B$ pages.
This process is repeated for each of the remaining intervals. At the end of this
process, the scheduler has: 1) a schedule of retrievals and discards for each disk that
is independent of the replication requests, and 2) a list $L$ of replications describing
the pages to be replicated, their current location and possible target disks, and the
time interval prior to which the replication must be done in order to resolve the
bottleneck. Stage one is now complete.

The second stage selects a target from the candidate disks for each element $e$ in
$L$. We start with a simple description of this stage. During stage two, this heuristic
computes a demand and supply value for each disk drive. While the demand for a
disk quantifies how frequently that disk might be used as the target for containing
the secondary copy of a page, its supply strives to measure the available bandwidth
of the disk. Assuming that $e(\text{targets})$ denotes the set of candidate disks that can
serve as target for $e$ and $|e(\text{targets})|$ denotes the number of disks in that set, the
demand for disk $d_i$ is:

$$
demand(d_i) = \sum_{e \in L \text{ and } d_i \in e(\text{targets})} \frac{1}{|e(\text{targets})|}
$$

The reciprocal weights the demand of a candidate disk based on its degree of freedom,
how many other disks can substitute for $d_i$. For example, an element of $L$ with one
target candidate ($d_i$) makes the demand for $d_i$ higher than if it could choose between
four possible target disks. Assuming that $|F^d_j|$ denotes the number of pages retrieved from $d_i$ during interval $j$, the supply of $d_i$ is estimated as:

$$supply(d_i) = \sum_{-p \leq i \leq m-1} (B - |F^d_i|)$$

A disk that is almost completely utilized during each time interval will have a low supply value. A disk that participates as a possible candidate for many page replications might have a high demand value depending on the number of disks that compete with it to serve as a potential candidate. Next, replication computes the demand to supply ratio of each disk.

A disk with a high demand to supply ratio has the maximum number of constraints. As a general rule of thumb, the bandwidth of these disks should be employed with care because they could potentially be used to resolve several bottlenecks. The heuristic to accomplish this is as follows. It picks the disk with the lowest demand to supply ratio (least constrained disk), termed $d_{free}$. It identifies those elements of list $L$ that contain $d_{free}$ as a possible candidate to resolve bottlenecks. From these elements, it chooses the element with the fewest target candidate disks. Still several elements may qualify because their number of target candidate disks is identical and this is the smallest value. From these, it chooses the element whose other candidate disks are highly constrained (i.e., have the highest demand to supply ratio). The satisfying element is modified to have $d_{free}$ as the only alternative to contain the secondary copy. The supply value for $d_{free}$ is adjusted and this process is repeated until each disk is left with one candidate target disk.

In the final stage, this heuristic extends the schedule of retrievals and discards of stage one with appropriate read and write operations to construct secondary copies. At this time, each page to be replicated has a source containing its primary copy and a target disk to contain its secondary copy, termed $d_{source}$ and $d_{target}$ respectively. Moreover, this page must be replicated prior to the time interval that the bottleneck was encountered, termed $i_{bottleneck}$. (This information is maintained for each element in list $L$, see stage one.) We strive to construct secondary copies during the display (between time interval 0 and the one corresponding to the bottleneck) in order to minimize the startup latency. Moreover, the number of intervals elapsed from when a replicated page is read from $d_{source}$ to the time that this page is written to $d_{target}$
must be minimized in order to minimize the amount of memory required from the system. Based on these two objectives, the system employs the following heuristic to replicate a page. It locates two adjacent positive time intervals \((j, j+1)\) prior to the occurrence of the bottleneck \((j+1 < i_{\text{bottleneck}})\) such that \(d_{\text{source}}\) and \(d_{\text{target}}\) have available disk bandwidth during intervals \(j\) and \(j+1\) respectively. If this step is successful, intervals \(j\) and \(j+1\) of the schedule of retrievals and discards is extended in order to replicate the target page. Otherwise, it performs the replication as close as possible to \(i_{\text{bottleneck}}\) in order to avoid using a negative time intervals. The read and write of this page are performed as close as possible to minimize the memory requirement.

5.3 A Hybrid of Memory and Replication-Based Resource Schedule

With a hybrid approach, the system uses either memory or replication to resolve a bottleneck. Hybrid employs replication as soon as the display reaches a pre-specified threshold on how much memory is allowed to utilize. Memory requirements below the threshold implies that solving a bottleneck with replications might impose unnecessary demands for both disk space and disk bandwidth, therefore it resolves the bottleneck using memory. On the other hand, memory requirements above the threshold invoke the algorithm to replicate in order to resolve bottlenecks to reduce the demand for excessive memory that might increase the latency. Therefore it would be better to solve the bottlenecks with replications. This technique is very similar to the heuristic of Section 5.2. The only difference is that during stage one, it employs pre-fetching if the amount of utilized memory is less than or equal to the threshold. Otherwise, it employs replication. When constructing the resource schedule, hybrid might toggle between replication and memory-based technique multiple times depending on when the utilized memory reaches the specified threshold.
5.4 Evaluation

We quantified the trade off associated with memory-based, replication-based, and hybrid scheduling techniques using a simulation study. The performance of each technique depends on both the placement of data across the disks and the display schedule. As a yard-stick to evaluate the alternative techniques, we compute a theoretical minimum on the amount of memory and latency required to support a coordinated display. This theoretical minimum conceptualizes $D$ disks as a single disk that can support the retrieval of $B \times D$ pages during each time interval. Thus, it renders the placement of data across the disks irrelevant. It employs the greedy algorithm (Figure 4.3) to compute both the minimum amount of required memory and incurred latency. This technique is theoretical because it might be impossible to employ the aggregate bandwidth of $D$ disks with arbitrary schedules and data layouts.

This section starts with a description of our simulation model and experimental design. Next, we present and discuss the obtained results.

5.4.1 Simulation Model

The simulation model consists of four modules (Figure 5.2): a display schedule generator that produces a disk page reference pattern for a presentation; a greedy scheduler that computes the theoretical minimum; a data placement generator that assigns the pages that constitute the database to the disk drives; and a multi-disk resource scheduler that generates a resource schedule that satisfies the temporal constraints imposed by the display schedule. This component implements the memory-based, replication-based, and hybrid techniques along with their persistent (single and multi-copy) and transient modes of operation.

The display schedule generator assumes that a presentation consists of a sequence of scenes. Each scene consists of a collection of background objects, and a collection of motions. A motion is represented as a sequence of deltas to an object (the first posture), where a delta is implemented either as a computation or a page retrieval that describes the required changes. Once applied to an object, they provide the illusion of a movement. The duration of each scene ($dur_{scene}$) is selected randomly
Figure 5.2: Simulation Model
Figure 5.3: Number of pages referenced during each time interval by Presentation 1 from a collection of possible values. These values were generated by analyzing the duration of several scenes for two different animations “Lion King” and “Reboot”. The number of pages that constitute the first interval of a scene (i.e., the pages that contain background objects and the first postures of the motions) are generated randomly using a value in the range of $[1, 60]$. The number of motions ($n_{motions}$) is a fraction of the number of pages referenced during the first interval. This fraction is a randomly chosen real value between 0 and 1 to represent scenes that range from static background to those that change rapidly. Deltas can translate into either disk retrievals or computations. The display schedule generator selects a percentage ($p$) of the total number of deltas as retrievals assuming a delta per motion per time interval is required. It assumes that this percentage is distributed uniformly. The number of deltas that translate into disk retrievals are computed as follows: $p \times n_{motions} \times dur_{scene}$. Finally, it assigns page identifiers to all pages in the first interval and all pages corresponding to deltas.

The data placement module assigns pages to $D$ disks with the objective to distribute them evenly.

5.4.2 Performance Results

For the purposes of this evaluation, we generated a single presentation using our display schedule generator. This presentation is 24 minutes long and references

---

2This range was based on the number of characters appearing simultaneously in these two animations. We assumed that the size of an object is smaller than a disk page. The specification of a large object in the background in the display schedule is equivalent to the specification of several small objects in the background. Similarly, a motion of a large object is equivalent to several small object motions.
65,002 pages. The total number of pages in the database is 131,072. The display schedule consists of 5400 time intervals, each \( \frac{8}{37} \) seconds long. Figure 5.3 shows the number of page references per time interval for this presentation. With this presentation, most of page references occurred during the first few scenes of the movie. We analyzed an alternative presentation (termed Presentation 2) that was the reflection of the one shown in Figure 5.3. With Presentation 2, most of page references occurred during its last few scenes.

Our target system was configured with a 32-Kilobyte disk page size. Its buffer pool was 32 Megabytes in size, partitioned into 1,024 disk page frames \((C=1,024)\). The number of disks in the system varies from 1 to 16 \((1 \leq D \leq 16)\). Each disk supports a 68 mbps transfer rate, 17 millisecond seek times, and 8.33 millisecond latency. Thus, a disk can read nine random pages during a time interval \((B=9)\). In the following, we compare the alternative techniques (memory-based, replication-based and hybrid) based on their startup latency, memory requirement, disk bandwidth and disk space requirements. We start with an analysis of a system configured with 7 disks. Subsequently, we describe the observed trends as a function of the number of disks. Finally, we establish the trade-off associated with alternative techniques.

### 5.4.2.1 Seven-Disk System

Tables 5.1 and 5.2 show the results obtained for the two alternative presentations using a seven-disk system. These tables quantify the latency, the amount of memory and disk bandwidth required to support a coordinated display with alternative techniques. In addition, it reports on the overhead due to replication (percentage of disk bandwidth, disk space, and memory used by replications). The theoretical minimum and memory-based schedules are independent of the amount of available memory \((C)\). Indeed, these two techniques dictate a value for \(C\). The theoretical minimum defines a lower bound of memory required to support a coordinated display and the memory-based defines the minimum memory required to display a presentation without modifying the placement of data.

---

\(^3\)One time interval is the duration of the display of 8 frames in the stream-based approach assuming a 30 frame per second display rate.
<table>
<thead>
<tr>
<th>Technique</th>
<th>Latency</th>
<th>Resource Requirements</th>
<th>Memory</th>
<th>Bandwidth</th>
<th>% Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Display</td>
<td>Max</td>
<td>Avg</td>
<td>Max</td>
<td>Avg</td>
</tr>
<tr>
<td>theoretical minimum</td>
<td>N/A</td>
<td>1</td>
<td>117</td>
<td>78.79</td>
<td>59</td>
</tr>
<tr>
<td>memory-based</td>
<td>N/A</td>
<td>12</td>
<td>3,155</td>
<td>445.41</td>
<td>63</td>
</tr>
<tr>
<td>replication-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi-copy</td>
<td>1st</td>
<td>94</td>
<td>703</td>
<td>175.79</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>2nd</td>
<td>1</td>
<td>126</td>
<td>79.19</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>3rd</td>
<td>1</td>
<td>118</td>
<td>78.79</td>
<td>59</td>
</tr>
<tr>
<td>single-copy</td>
<td>1st</td>
<td>94</td>
<td>703</td>
<td>175.79</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>2nd</td>
<td>85</td>
<td>1,024</td>
<td>187.36</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>4th</td>
<td>75</td>
<td>1,024</td>
<td>187.40</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>5th</td>
<td>70</td>
<td>1,024</td>
<td>181.53</td>
<td>63</td>
</tr>
<tr>
<td>hybrid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi-copy</td>
<td>1st</td>
<td>12</td>
<td>1,024</td>
<td>263.58</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>2nd</td>
<td>1</td>
<td>461</td>
<td>155.07</td>
<td>63</td>
</tr>
<tr>
<td>single-copy</td>
<td>1st</td>
<td>12</td>
<td>1,024</td>
<td>263.58</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>2nd</td>
<td>2</td>
<td>1,024</td>
<td>259.96</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>3rd</td>
<td>2</td>
<td>1,024</td>
<td>258.85</td>
<td>63</td>
</tr>
</tbody>
</table>

Table 5.1: Performance of alternative heuristics for Presentations 1 using 7 disks

<table>
<thead>
<tr>
<th>Technique</th>
<th>Latency</th>
<th>Resource Requirements</th>
<th>Memory</th>
<th>Bandwidth</th>
<th>% Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Display</td>
<td>Max</td>
<td>Avg</td>
<td>Max</td>
<td>Avg</td>
</tr>
<tr>
<td>theoretical minimum</td>
<td>N/A</td>
<td>1</td>
<td>117</td>
<td>78.79</td>
<td>59</td>
</tr>
<tr>
<td>memory-based</td>
<td>N/A</td>
<td>2</td>
<td>3,155</td>
<td>445.43</td>
<td>63</td>
</tr>
<tr>
<td>replication-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi-copy</td>
<td>1st</td>
<td>2</td>
<td>1,024</td>
<td>205.15</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>2nd</td>
<td>2</td>
<td>1,024</td>
<td>80.26</td>
<td>63</td>
</tr>
<tr>
<td>single-copy</td>
<td>1st</td>
<td>2</td>
<td>1,024</td>
<td>205.15</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>2nd</td>
<td>2</td>
<td>1,024</td>
<td>171.99</td>
<td>63</td>
</tr>
</tbody>
</table>

Table 5.2: Performance of alternative heuristics for Presentation 2 using 7 disks
Both replication-based and hybrid are dependent on $C$. To schedule a replication, the system verifies that there is available bandwidth to read and write the page and available memory to stage the page between the read and the write. All measurements for replication-based and hybrid techniques quantify the performance of the techniques in either transient or persistent mode. The measurements for both are identical (Tables 5.1 and 5.2) except for the last column (percentage of extra disk space) which is always 0 for the transient mode.

The memory-based technique might have excessive memory requirements to support a coordinated display. Unbalanced placement of pages referenced by a presentation during time interval $i$ increases the memory requirement during previous intervals $j$ ($j < i$). Similarly, unbalanced placement during interval $i - 1$ increases the memory requirement during previous intervals $j$ ($j < i - 1$). Therefore, the increase in memory requirement might be cumulative if consecutive intervals result in formation of bottlenecks. Figure 5.4(a) shows the total number of disk pages retrieved from the disks by the memory-based resource schedule. Figure 5.4(b) shows the number of pages rendered memory resident with this technique. Presentations 1 and 2 require 3,155 memory pages, thus a system with fewer memory frames cannot support a coordinated display of either presentation.

The startup latency observed by both replication-based and hybrid techniques depends on the time intervals when the bottlenecks occur. Presentations whose reference pattern results in the formation of bottlenecks at the beginning of the

Figure 5.4: Coordinated display of Presentation 1 using memory-based with a 7-disk system
Figure 5.5: Theoretical minimum for Presentation 1 in a system with 7 disks
display might incur a higher latency than those whose schedule results in a bottleneck
at the end of their display. Presentation 2 has a low disk bandwidth requirement at
the beginning of its display. This enables the system to replicate pages during the
display of this presentation, resulting in a lower startup latency (2 for Presentation
2 and 12 for Presentation 1 with memory-based, compare Table 5.1 with 5.2).

If a system configuration can support the coordinated display of a structured pre-
sentation using memory-based, then there is a replication-based resource schedule
that supports a coordinated display of the presentation as well. The replication-
based scheduling technique resolves all bottlenecks by constructing secondary copies
and scheduling their construction such that the memory requirement does not ex-
ceed the memory available to the system \(C\). Assuming that disk space is not a
limitation, if the minimum memory requirement of a coordinated display (theoretical
minimum) is lower than the system’s memory \(C\) then there is a replication-based
schedule for the presentation. The memory available to the system (1,024) is greater
than the theoretical minimum (117) for both Presentations 1 and 2 (Tables 5.1 and
5.2), enabling replication-based to support both. The replication-based technique
requires 703 and 1,024 memory frames (Tables 5.1 and 5.2) to display Presentation
1 and 2, respectively.

Figures 5.5 shows the number of disk pages retrieved and the number of memory
frames required per time interval for the theoretical minimum. If the seven disks were
replaced by one disk whose bandwidth was equivalent to the aggregate bandwidth
of seven disks (rendering the placement of data as irrelevant), a coordinated display
of Presentation 1 would have required 117 memory frames and a startup latency equivalent to one time interval (Table 5.1).

In a system that allows multiple copies of a page, the replication-based technique in persistent mode approximates the theoretical minimum as a presentation is displayed repeatedly. In the absence of disk space limitations, the performance of this technique will be identical to the theoretical minimum after a finite number of presentations, e.g., three consecutive displays of Presentation 1 (see Figure 5.6). This is because the secondary copies eliminate some bottlenecks in subsequent displays, reducing both the memory and disk bandwidth requirement of a display along with its incurred startup latency. Figure 5.6 shows the bandwidth and memory requirement of replication-based schedules for each of three consecutive displays of Presentation 1. Note that the maximum amount of memory required for the first display (703 frames) decreased by a factor of six for the second display of presentation 1 (Table 5.1). The last profile (Figure 5.6(e) and (f)) is almost identical to that of the theoretical minimum (compare with Figure 5.5), enabling the scheduler to harness the bandwidth of $D$ disks. The resulting replicated data requires an additional 23.77% of the disk space.

With the single-copy replication-based technique in persistent mode, the disk bandwidth overhead also decreases as a function of subsequent displays. This technique maintains the copy of a page that was most frequently referenced by the display and deletes the rest. This change in data placement might decrease the number of bottlenecks in subsequent displays, reducing the number of page replications from 15,348 for the first display to 10,637 for the second display and 9,429 for the eight display (this is reflected as a decrease in the percentage overhead in disk bandwidth requirement, see Table 5.1). Note that with this technique both the memory requirements and the memory overhead due to replications in subsequent displays might increase (Table 5.1). The data placement changes induced by this technique might cause a bottleneck at different time, forcing the replications to be scheduled during a period of scarce resources. Moreover, the heuristic might schedule the replications so that the length of time a page remains memory resident prior to be written is made longer because it strives to minimize startup latency. This increases the memory requirements of the schedule. For example, in Table 5.1, the memory requirement
Persistent replication, first display

Persistent replication, second display

Persistent replication, third display

Figure 5.6: Three consecutive displays of Presentation 1 in a system with 7 disks
<table>
<thead>
<tr>
<th>Technique</th>
<th>Display</th>
<th>Latency</th>
<th>Resource Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Max</td>
</tr>
<tr>
<td>theoretical minimum</td>
<td>N/A</td>
<td>1</td>
<td>117</td>
</tr>
<tr>
<td>memory-based</td>
<td>N/A</td>
<td>1</td>
<td>164</td>
</tr>
<tr>
<td>replication-based</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi-copy</td>
<td></td>
<td>1</td>
<td>187</td>
</tr>
<tr>
<td>2nd</td>
<td></td>
<td>1</td>
<td>122</td>
</tr>
<tr>
<td>3rd</td>
<td></td>
<td>1</td>
<td>117</td>
</tr>
<tr>
<td>single-copy</td>
<td></td>
<td>1</td>
<td>187</td>
</tr>
<tr>
<td>2nd</td>
<td></td>
<td>1</td>
<td>199</td>
</tr>
<tr>
<td>3rd</td>
<td></td>
<td>1</td>
<td>155</td>
</tr>
</tbody>
</table>

Table 5.3: Measurements obtained for Presentation 1 using 8 disks

increased from 703 in the first display (replication-based) to 1,024 in the second display (replication-based, single-copy, second display).

The hybrid technique requires less disk bandwidth (columns Bandwidth-Avg. and Overhead-% Band. in Table 5.1) and disk space than the replication-based technique. Instead of solving all bottlenecks with replications that increase both disk bandwidth and space requirements, it pre-fetches pages until a threshold (50% of $C$ in our experiments) is reached. The percentage of extra disk space attributed to replication is 4.92% for the first display of hybrid while 22.66% for the first display of replication-based. Hybrid reaches a stable state sooner than the replication-based technique. The second display using the hybrid technique constructs no additional secondary copies, while the third display using the replication-based technique continues to construct secondary copies.

### 5.4.2.2 Trends

With fewer than seven disks, none of the proposed techniques can support a co-ordinated display of either Presentation 1 or 2 because there is insufficient disk bandwidth and memory. With more than 12 disks, the available disk bandwidth is abundant, rendering the difference between the alternative techniques negligible (Tables 5.5 and 5.6).

An increase in the number of disks has the following effects: (1) decreases the memory requirements of the memory-based technique (columns Memory in Tables 5.3, 5.4, 5.5, and 5.6), (2) decreases the disk bandwidth requirements of the replication-based technique (columns Bandwidth and Overhead-% Band. in Tables 5.5 and 5.6).
Resource Requirements

<table>
<thead>
<tr>
<th>Drives</th>
<th>Lat.</th>
<th>Memory</th>
<th>Bandwidth</th>
<th>% Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>129</td>
<td>79.55</td>
<td>65</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>125</td>
<td>79.02</td>
<td>65</td>
</tr>
</tbody>
</table>

Table 5.4: Measurements obtained by memory-based and hybrid techniques for Presentation 1 in a system with 10 and 12 drives. Both techniques yielded identical measurements.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Display</th>
<th>Latency</th>
<th>Resource Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>theoretical minimum</td>
<td>N/A</td>
<td>1</td>
<td>117</td>
</tr>
<tr>
<td>memory-based</td>
<td>N/A</td>
<td>1</td>
<td>124</td>
</tr>
<tr>
<td>multi-copy</td>
<td>N/A</td>
<td>1</td>
<td>121</td>
</tr>
<tr>
<td>single-copy</td>
<td>1st</td>
<td>1</td>
<td>121</td>
</tr>
</tbody>
</table>

Table 5.5: Measurements obtained for Presentation 1 using 14 disks

5.3, 5.5, and 5.6), and (3) might decrease the memory requirements of the replication-based technique (columns Memory in Tables 5.3, 5.5, and 5.6). This is because additional disks reduce the possibility of bottleneck disks. For the replication-based schedules, the memory requirement depends not only on the number of pre-fetches but also on the scheduling of the replications. The heuristic presented in this study strives to reduce the startup latency of a display, resulting in a possible increase in the memory requirements of the display.

A hybrid of replication and memory-based behaves the same as the memory-based resource schedule if the memory requirement is below the threshold. In our

<table>
<thead>
<tr>
<th>Technique</th>
<th>Display</th>
<th>Latency</th>
<th>Resource Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>theoretical minimum</td>
<td>N/A</td>
<td>1</td>
<td>117</td>
</tr>
<tr>
<td>memory-based</td>
<td>N/A</td>
<td>1</td>
<td>124</td>
</tr>
<tr>
<td>replication-based</td>
<td>1st</td>
<td>1</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td>2nd</td>
<td>1</td>
<td>117</td>
</tr>
<tr>
<td>single-copy</td>
<td>1st</td>
<td>1</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td>2nd</td>
<td>1</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td>3rd</td>
<td>1</td>
<td>121</td>
</tr>
</tbody>
</table>

Table 5.6: Measurements obtained for Presentation 1 in a system with 16 drives
experiments, the memory requirement of the memory-based technique for a system with at least 8 disk drives is lower than the threshold (512 memory frames). Then this hybrid technique behaves like memory-based for a system with 8-16 drives (Figures 5.3, 5.4, 5.5 and 5.6).

5.4.2.3 Summary

Table 5.7 shows a performance comparison of the different scheduling techniques. This table only considers the case when the memory threshold used by hybrid is lower than the memory requirement of the memory-based technique. If the threshold is higher than this memory requirement then hybrid scheduling is identical to memory-based scheduling (i.e., the last column can be ignored). The first criteria determines whether a technique supports the display of a presentation. If the memory requirement of the theoretical minimum is less than or equal to \( C \), then the replication-based and hybrid techniques would be able to support a coordinated display at all times. On the other hand, the memory-based technique might be unable to display it because of excessive (> \( C \)) memory requirement. The other criteria are: (i) whether the technique approximates to the theoretical minimum as the number of displays of the same presentation increases, (ii) latency of the first display and n-th display of the presentation where \( n \) is the number of displays required by multi-copy persistent replication-based technique to reach the theoretical minimum, (iii) disk bandwidth required to display the presentation the first and n-th time, (iv) disk space requirement between displays, and (v) memory required to display the presentation the first and n-th time. Criteria (i) to (v) apply only to presentations that can be displayed with any of the scheduling techniques (including memory-based).

Integers (i.e., 0, 1, 2, and 3) specify an order on the techniques using a performance criteria. For the first display of a presentation, the latency of the memory-based (0) is smaller than hybrid’s (1). This in turn is smaller than both single and multi-copy replication approach (2). The superscript ’*’ on two techniques indicates that one technique might be smaller than or equal to the other and vice-versa. For the n-th display of a presentation, the latency of memory-based might be smaller than, greater than, or equal to the latency of the single-copy technique. Also for
<table>
<thead>
<tr>
<th>Criteria</th>
<th>Memory-based</th>
<th>Persistent replication-based</th>
<th>Hybrid T% Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Single-copy</td>
<td>Multi-copy</td>
</tr>
<tr>
<td>Supports Coordinated Display</td>
<td>not always</td>
<td>always</td>
<td>always</td>
</tr>
<tr>
<td>Approach, Theoretical Min.</td>
<td>not always</td>
<td>always</td>
<td>always</td>
</tr>
<tr>
<td>Latency (1st disp.)</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Latency (n-th disp.)</td>
<td>$2^*$</td>
<td>$2^*$</td>
<td>0</td>
</tr>
</tbody>
</table>

**Resource Requirements**

<table>
<thead>
<tr>
<th></th>
<th>Disk Bandwidth (1st disp.)</th>
<th>Disk Bandwidth (n-th disp.)</th>
<th>Disk Space</th>
<th>Memory (1st disp.)</th>
<th>Memory (n-th disp.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>$0^*$</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>$2^*$</td>
<td>0</td>
<td>$0^*$</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.7: Performance comparison assuming that the threshold ($\frac{T \cdot C}{100}$) is smaller than the memory requirements of memory-based scheduling.

The n-th display of a presentation, both memory-based and single-copy techniques yield higher latency than hybrid which has a higher latency than multi-copy. Bold-face highlights the best technique using a criteria (e.g., replication-based and hybrid for supporting coordinated display, multi-copy-replication-based for approaching the theoretical minimum).

Hybrid uses memory and replications to resolve bottlenecks in a controlled fashion, so that resources are utilized effectively. As summarized in Table 5.7, hybrid compromises memory and disk resources (i.e., bandwidth and space). It requires less disk resources than replication-based and less memory than memory-based. However, replication-based requires less memory and yields lower latency than hybrid. While memory-based requires less disk resources than hybrid. The threshold determines how much memory to trade for disk resources. As the threshold approaches to 0%, hybrid starts to approximate replication-based. (Similarly, as it approaches to 100%, it approximates memory-based.) Therefore, the threshold should be tuned based on the availability of system resources.
Chapter 6

Data Layout and Availability

This chapter introduces a framework for initial placement of data and fault tolerance. Data layout plays an important role in resource scheduling. Placement of pages on disks impacts the performance of the system. The resource scheduling techniques presented in Chapter 5 manipulate this placement during the display to dissolve bottlenecks that degrade system’s performance. However, the changes in data placement for one presentation might cause bottlenecks for the display of other presentations. This chapter introduces the problem of initial placement of data so that the formation of bottlenecks during the display of all presentations is minimized. Data availability is another key issue in multi-disk architectures. The chances of some disk failure increases as the number of disks in the system increases. This chapter presents issues to address in a system that supports the display of structured presentations in presence of disk failure.

6.1 Initial Placement of Data

Placement of pages on disks impacts the performance of the system. Latency, disk bandwidth, disk space and memory requirements of a display depend on the placement of pages referenced by the presentation. Unbalanced placement\(^1\) might increase the memory requirements and the latency of a presentation displayed following a memory-based resource schedule. Unbalanced placement might also increase the

\(^1\)Pages referenced during a time interval are not spread evenly across all disk drives.
latency, memory, disk bandwidth and disk space requirements of presentations displayed following either a replication-based or a hybrid resource schedule. On the other hand, balanced placement maximizes parallel retrieval from disks reducing the resource requirements and latency of the presentation. Pages referenced by the presentation during a time interval should be spread evenly across all disk drives to achieve the maximum degree of parallelism.

6.1.1 Statement of the Problem

This section states formally the problem of initial data placement on disks for structured presentations. It first introduces the case when a single copy of a page is allowed in the system (i.e., replication is not permitted). Then, it defines the problem for the multi-copy case.

6.1.1.1 Single-Copy Placement

A data placement \( (\mathcal{P}) \) is a partition \( (\mathcal{P}_0, \ldots, \mathcal{P}_{D-1}) \) of a set of disk pages (e.g., pages in the database) into \( D \) subsets. Each partition \( \mathcal{P}_i \) represents the pages on disk \( i \) and should not exceed its storage capacity. The degree of parallelism achieved by two disks during the retrieval of a set of pages is the number of times two pages are retrieved in parallel. For example, consider a data placement that assigns pages \( a, c, e, g, i \) to disk \( d_1 \) and pages \( b, d, f, h, j \) to \( d_2 \). One schedule to retrieve \( a, b, c, d, e, q \) is as follows: retrieve \( a \) and \( b \) in parallel, then \( c \) and \( d \) in parallel, and finally \( e \). In this case, the system maximized parallelism two times in a row. In fact, the maximum level of parallelism to retrieve this set is 2. Therefore, the degree of parallelism during the retrieval of pages \( a, b, c, d, e, q \) from disks \( d_1 \) and \( d_2 \) is 2. The degree of parallelism of two disks with respect to display schedule \( \{P_0, \ldots, P_{m-1}\} \) is the aggregate degree of parallelism during the retrieval of two disks with respect to display schedule \( \{P_0, P_1 - P_0, \ldots, P_{m-1} - P_{m-2}\} \).

The placement of pages across the disk drives and the degree of parallelism with respect to a display schedule is represented by a fully connected undirected graph with weighted edges. Each node represents a disk in the system and the weight associated to an edge represents the degree of parallelism of the two linked disks\(^2\) \( \mathcal{P}_i \) for the retrieval of \( \mathcal{P}_{i+1} \) because they are already in memory.

---

\(^2\)The system does not have to retrieve pages in \( P_i \) for the retrieval of \( P_{i+1} \) because they are already in memory.
with respect to the display schedule. Each node is tagged with the set of pages in
the corresponding disk. For the case of multiple display schedules, the graph is the
same except for the weights of the edges. These weights are the aggregate degrees
of parallelism with respect to each display schedule.

Example 3: Consider the display schedule \{\{a, b, c, d, e\}, \{f, g, h\}, \{a, b\}\}. Sup-
pose that pages a and b reside on disk 0, pages c, d, and e reside on disk 1, and pages
f, g, and h reside on disk 2 (Figure 6.1(a)). The degree of parallelism of disks 0 and
1 during the retrieval of \{a, b, c, d, e\} is 2. The retrieval of a and b from disk 0 can
be performed in parallel with the retrieval of two pages on disk 1 (e.g., c and d).

Suppose that pages a and h reside on disk 0, pages b, d, and f on disk 1, and
pages c, e, and g on disk 2, Figure 6.1(b). In this case, the degree of parallelism of
disks 0 and 1 during the retrieval of each set (\{a, b, c, d, e\}, \{f, g, h\}, and \{a, b\}\) is
1. Therefore, the degree of parallelism of disks 0 and 1 with respect to this display
schedule is 3.

Formally, the degree of parallelism of two disks with respect to a set of pages is
defined as follows.

Definition: Given a placement of data \(\mathcal{P}\) and a set of pages to be retrieved
simultaneously (denoted \(S\)), the degree of parallelism of disks \(d_1\) and \(d_2\) with respect
to \(S\) (denoted as \(\text{deg}(\mathcal{P}, S, d_1, d_2)\)) is:

\[
\text{Max}\{\text{Min}(\|S_1\|, \|S_2\|) \mid S_1 \cup S_2 \subseteq S \text{ and } S_1 \subseteq \mathcal{P}_{d_1} \text{ and } S_2 \subseteq \mathcal{P}_{d_2}\}
\]
where $P_{d_1}$ and $P_{d_2}$ are the pages in disks $d_1$ and $d_2$, respectively.

Given a placement of data $P$, the degree of parallelism for disks $d_1$ and $d_2$ with respect to display schedule $DS = \{P_0, \ldots, P_{m-1}\}$ (denoted as $\text{DEG}(P, DS, d_1, d_2)$) is

$$\text{deg}(P, P_0, d_1, d_2) + \sum_{i=1}^{m-1} \text{deg}(P, P_i - P_{i-1}, d_1, d_2)$$

The objective of a placement technique is to assign all pages in the database to the disks in the system such that the total degree of parallelism is minimized. The total degree of parallelism is the sum of the degrees of parallelism of each pair of disks with respect to each display schedule. Formally, the data placement problem is stated as follows. Given a set of pages, a set $\{0, \ldots, D-1\}$ of disks, and a set $S$ of display schedules, find a placement $P$ such that $\sum_{DS \in S} \sum_{i=0}^{D-1} \sum_{j=i+1}^{D-1} \text{DEG}(P, DS, i, j)$ is minimized.

### 6.1.1.2 Multi-Copy Placement

In a system that has secondary copies of pages, a data placement $(P^r)$ on disks $\{0, \ldots, D-1\}$ is a collection $(P^r_0, \ldots, P^r_{D-1})$ of subsets of disk pages in the database such that each page in the database is in at least one subset. Each subset $P^r_i$ represents the pages on disk $i$ and should not exceed the capacity of the disk drive. As opposed to the single-copy case, these subsets are not necessarily disjoint. As in Section 6.1.1.1, the objective is to maximize parallel retrievals.

The computation of the degree of parallelism during the retrieval of a set of pages is as follows. The multi-copy placement must be restricted to a partition before computing the degree of parallelism. For example, consider a data placement that assigns pages $a$, $d$, and $h$ to disk 0, pages $b$, $d$, and $f$ to disk 1, and pages $c$, $e$, and $g$ to disk 2 (Figure 6.2). The degree of parallelism of this placement with respect to display schedule $\{\{a, b, c, d, e\}, \{f, g, h\}, \{a, b\}\}$ based on the definition of Section 6.1.1.1 is 10 (Figure 6.2(a)) while for the placement in Figure 6.1(b) is 8. However, as compared with Figure 6.1(b), the extra replica of $d$ does not increase the parallelism during the retrieval of $\{\{a, b, c, d, e\}, \{f, g, h\}, \{a, b\}\}$. In both cases (Figures 6.1(b) and 6.2(a)), the retrieval of $\{a, b, c, d, e\}$ requires two times the retrieval of a disk page, while the retrieval of $\{f, g, h\}$ and $\{a, b\}$ the same as one page each. The
Figure 6.2: Degrees of parallelism. (a) based on the multi-copy placement, (b) based on the single-copy placement with d on disk 1, and (c) based on the single-copy placement with d on disk 0.

Problem is that the degree of parallelism in Figure 6.2(a) considers the retrieval of d from disks 0 and 1 during the retrieval of \{a, b, c, d, e\}. The degree of parallelism of drives 0 and 2 with respect to \{a, b, c, d, e\} is 2 (a, c and d, e are retrieved in parallel). Similarly, the degree of parallelism of drives 1 and 2 with respect to \{a, b, c, d, e\} is 2 (b, c and d, e are retrieved in parallel). In the first case d is retrieved from disk 0, while in the second case d is retrieved from disk 1. Note that retrieving d twice during the same time interval would be a waste of resources. To avoid counting a page retrieval more than once in the computation of the degree of parallelism, the placement of data on disks must be restricted to a partition. For example, the degree of parallelism with respect to our example display schedule assuming a partition with the primary copy of d on disk 1 (i.e., ignoring the secondary copy of d on 0) is shown in Figure 6.2(b). Similarly, the degree of parallelism assuming that the primary copy of d resides on disk 0 is presented in Figure 6.2(c). Notice that both cases (Figures 6.2(b) and (c)) yield a degree of parallelism identical to that of Figure 6.1(b).

Formally, the degree of parallelism of a data placement with respect to a set of pages is defined as follows.

**Definition:** Given a placement of data \( \mathcal{P}^r \) and a set of pages \( S \), the degree of parallelism of \( \mathcal{P}^r \) with respect to \( S \) (denoted as \( \widehat{\text{deg}}(\mathcal{P}^r, S) \)) is:
\[
\begin{align*}
\text{Max} \left\{ \left( \sum_{d_1=0}^{D-1} \sum_{d_2=d_1}^{D-1} \deg(\mathcal{P}, S, d_1, d_2) \right) \left| \mathcal{P} \text{ is a single-copy placement of } \mathcal{P}_0^r \cup \ldots \cup \mathcal{P}_{D-1}^r \text{ and for each } 0 \leq i \leq D-1, \mathcal{P}_i \subseteq \mathcal{P}_i^r \right\} \right.
\end{align*}
\]

Given a placement of data \( \mathcal{P}^r \), the degree of parallelism with respect to display schedule \( DS = \{P_0, \ldots, P_{m-1}\} \) (denoted as \( \widehat{DEG}(\mathcal{P}, DS) \)) is

\[
\widehat{deg}(\mathcal{P}^r, P_0) + \sum_{i=1}^{m-1} \widehat{deg}(\mathcal{P}^r, P_i - P_{i-1})
\]

Similar to the single-copy placement, the objective of a multi-copy placement is to assign all pages in the database to the disks in the system such that the total degree of parallelism is minimized. Formally, the multi-copy data placement problem is stated as follows. Given a set of pages, a set \( \{0, \ldots, D-1\} \) of disks, and a set \( \mathcal{S} \) of display schedules, find a placement \( \mathcal{P} \) such that \( \sum_{\forall DS \in \mathcal{S}} \widehat{DEG}(\mathcal{P}, DS) \) is minimized.

### 6.1.2 Placement Approaches

One question that arises is whether the single-copy data placement problem is NP-Hard. If this problem is NP-Hard, then the multi-copy data placement problem is also NP-Hard. This is an open question that needs further investigation (Chapter 7). The nature of this problem leads us to formulate the hypothesis that this is an NP-Hard problem. If the hypothesis is true, then heuristic-based approaches must be devised to solve the problem.

A single-copy heuristic to assign pages in a database traverses all display schedules and assigns the pages to disks in a round robin fashion. It avoids assigning two pages referenced simultaneously by a display to the same disk. It first assigns pages in \( P_0 \) that are not on disk, then pages in \( P_1 - P_0 \) that are not on disk, next pages in \( P_2 - P_1 - P_0 \) that are not on disk, and so forth. If the round robin assignment places two pages referenced simultaneously by the current display during the current time interval\(^3\), then it skips disks until either there is no conflict or it already skipped all

---

\(^3\)Pages in \( P_i \).
Figure 6.3: Placement of pages referenced by display schedules \(\{a, b, c, d\}, \{e, f\}, \{e, g, h, i\}\) and \(\{c, j, k, l\}, \{e, m, l\}, \{h, i, m, n\}\) disks. Then, it assigns the page to the first non-conflicting disk in the round robin sequence, or to the next disk if all disks conflict.

For example, consider a database with two display schedules \(\{a, b, c, d\}, \{e, f\}, \{e, g, h, i\}\) and \(\{c, j, k, l\}, \{e, m, l\}, \{h, i, m, n\}\). The heuristic first assigns \(a, b, c, d\) to disks 0, 1, 2, 3 respectively. Then, it assigns \(e, f\) to 0, 1. Next, it assigns \(g, h, i\) to 2, 3, 1. It skips disk 0 in the round robin assignment because \(e\) is referenced simultaneously and is on 0. Similarly, it assigns the pages in the other display schedule to disks (Figure 6.3).

The heuristics for data placement can be evaluated using the simulation model of Section 5.4.1. The module Data placement generator (Figure 5.2) is replaced by the heuristic and the performance results (i.e., latency and resource requirements) can be compared with those of random placement of Section 5.4.1. As in Section 5.4, the performance of the system under each scheduling technique (memory-based, replication-based, hybrid, and theoretical minimum) can be compared. Then based on these results, we can characterize the heuristic. This remains as a future extension.

### 6.2 Fault Tolerance

Technological trends show that CPU speed (MIPS), memory chip capacity, magnetic disks density, and speed of delivery from memory to a CPU are growing much faster than speed of delivery from a single disk to main memory [32]. An alternative to diminish this unbalance is to have a multi-disk architecture. The speed of delivery from disk to memory increases as the number of disks increases. However, the mean
time to failure of some disk decreases as the number of disks increases [32]. Therefore, fault tolerance is an important issue for multi-disk architectures.

The scheduling techniques proposed in this dissertation assume that all disks on the system are operational. An important extension of this work is to study fault tolerance schemes: (1) data layout to support a display in the presence of disk failures, and (2) scheduling techniques that allows the system to display a presentation in the presence of disk failures.

To support fault tolerance, the system has to store redundant parity information. Disk pages from different disks are grouped and its parity (exclusive or of pages in the group) is stored in a disk different from the disks where the pages reside. When a disk fails the system can re-construct the missing data from the other pages in the group and its parity. An example of a data layout would be to have three data disk pages \((d_1, d_2, d_3)\) on disks 1, 2, 3, respectively, and the parity page \(p = d_1 \text{xor} d_2 \text{xor} d_3\) on disk 4. Then, if disk 2 fails, \(d_2\) can be re-constructed as \(d_1 \text{xor} d_3 \text{xor} p\).

The objective of dividing the database into parity groups is to increase the availability of data in case of disk failure. However, the retrieval of pages (data and parity) needed to re-construct missing data imposes additional demands on the disks where the other data and parity pages reside. These additional demands might interfere with the retrieval of other units to satisfy the display schedule. Therefore, the placement of disk pages in a group (data and parity) impacts the performance of the system in the presence of disk failure.

One research direction is how to divide the database into groups of pages and where to store the parity page for each group, i.e., what is a good data layout for fault tolerance. This data layout can be either independent or dependent from the display schedules. A data layout independent from the display schedules is a mapping from a collection of data pages into a collection of parity pages. It maps a pair \(<pgid, dsk>\) (data page identifier and disk where it resides) into a parity page \(<pgid, dsk>\). All data pages mapped to the same parity page belong to the same parity group and reside in different disks. Examples of this type of grouping are RAID levels 4 and 5 [32]. In the presence of disk failure, the re-construction of missing data might be expensive for this type of layout. The system might have to retrieve all other pages.

\[\text{4A group size of one disk drive is equivalent to the mirroring scheme.}\]
in the group (data and parity) each time the display references a page in the faulty disk. The disk page reference pattern defined by display schedules is useful for fault tolerance. They define pages that must be in memory simultaneously. If these pages belong to one parity group, then the re-construction of one of them (when a disk fails) requires the retrieval of the parity page only. The other data pages do not need to be retrieved because they are already in memory. Data layout schemes to support the display of structured presentations in the presence of disk failure is an open issue (Chapter 7).

Once a data layout is defined, the next question is how to schedule resources to support the display of a structured presentation in the presence of a disk failure. Reconstructing missing data due to a disk failure imposes additional disk bandwidth and memory requirements. The other data pages and the parity page of the group must be retrieved from disks into memory if they are not already in memory. If a disk fails before the display starts, it suffices to consider a resource schedule that satisfies a new display schedule. This new display schedule is obtained from the presentation’s display schedule as follows. Instead of a page in the failed disk, the new display schedule has the other pages in the group (data and parity). This resource schedule can be computed using the techniques presented in this dissertation. If a disk fails during a display, the display schedule changes after the failure (i.e., the display schedule is generated dynamically.) This change might require resources that are not available making impossible to support a coordinated display after the failure. For example, consider a presentation that references a disk page \(a\) on disk 2. Assume disk 2 fails in the middle of the display and the parity page associated to \(a\) is on disk 5. To re-construct \(a\), the system has to retrieve the parity page from 5. If disk 5 has insufficient bandwidth after the failure and prior to \(a\) being referenced, then the system fails to support a coordinated display of the presentation. A resource schedule for a dynamically generated display schedule is an open issue (Chapter 7).

Metrics to evaluate a data layout and a scheduling technique to support the display in the presence of disk failure include: (1) their success/failure to support coordinated displays when a disk fails during and before the display, (2) their impact on latency when a disk fails before the display, (3) their impact on disk bandwidth and memory requirements when a disk fails during and before the display, and (4)
their cost in terms of disk space, disk bandwidth, and memory required when the database is updated.
Chapter 7

Conclusions and Future Research Directions

This dissertation introduces a new conceptual data model, termed structured, to represent a full-motion presentation. With the structured approach, a presentation consists of a collection of objects with spatial and temporal constraints along with their rendering features. This new approach facilitates both re-usability of information and effective query processing techniques.

A coordinated display of a structured presentation must satisfy the temporal and spatial 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 a resource scheduler that supports a coordinated display of structured presentations for both single and multi-disk architectures. For a single-disk architecture, we presented a polynomial time algorithm to compute a resource schedule that supports a coordinated display of a given structured presentation. This algorithm minimizes both the startup latency and the memory requirements of a display. For a multi-disk architecture, it showed the following: (1) the computation of a resource schedule that supports a coordinated display and yields the minimum latency is an NP-Hard problem, and (2) given a system load, the computation of a schedule to change the placement of data across disk drives in minimum time is an NP-Hard problem.

This study introduced a taxonomy of resource scheduling techniques that support a coordinated display of structured presentations. These techniques are categorized into memory-based, replication-based, and migration-based. The last two categories manipulate the placement of data across the available disks. We proposed three
techniques that compute a memory-based, a replication-based and a hybrid of memory and replication-based resource schedules. The memory-based technique employs pre-fetching to resolve bottlenecks. The replication-based approach employs idle disk bandwidth to resolve bottlenecks. The hybrid technique uses either memory or replication depending on the availability of system resources when the bottleneck occurs.

We evaluated the proposed techniques using a simulation study. We compared these techniques based on their startup latency, memory requirement, disk bandwidth and disk space requirements. We also compared them with a theoretical minimum that ignores the placement of data and its impact on a coordinated display, i.e., a single-disk with the aggregate bandwidth of $D$ disks. The replication-based technique proved to support coordinated displays that the memory-based technique is unable to support (because they require excessive memory). In a system that employs multiple copies, the replication-based technique in persistent mode approximates the theoretical minimum on memory requirement and latency as a function of the number of times a presentation is displayed. In a system with scarce disk space, the single copy replication-based technique in persistent mode reduces the disk bandwidth overhead caused by replications in subsequent displays. The hybrid technique reduces the disk bandwidth and disk space requirement of the replications by resolving some of the bottlenecks with pre-fetching and the rest with replications.

At the conceptual level, future research directions include: (1) a conceptual model for the name space, and (2) data manipulation languages for structured presentations. At the theoretical level, future research directions include: (1) expressiveness of the data manipulation languages at the conceptual level, (2) complexity of the resource scheduler for special cases of multi-disk architectures such as a system with 2 disk drives, and (3) complexity of the initial placement of structured presentations on disks. At the systems level, future research directions include: (1) evaluation of the data placement technique in Section 6.1.2, (2) development and evaluation of fault tolerance schemes, (3) development and evaluation of migration-based techniques, (4) resource scheduling techniques for multi-user environments where several users display different presentations, and (5) indexing techniques to access data efficiently.
Reference List


Appendix A

Reduction from SAT to Replications 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 replications scheduling instance $R, A, N = SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$ has a solution.

SAT2RepSc defines a system load $A$ and a collection $R$ of replications so that alternative schedules for replications in $R$ on $A$ follow a specific pattern. For the case of replications associated to variables, there are only two alternatives to schedule these replications.

**Lemma 5** Let $R, A, N$ be the output of $SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$, and $r = s_i \rightarrow \{t_i, u_i\}$ be the replication in $R$ associated to variable $v_i$. There are only two alternatives to schedule $r$ on $A$:

1. **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

\[ 4 \cdot (n+1) \cdot (i-1) \]

\[ \text{read } s_i \]

\[ \text{write } d_{1i} \]

\[ \text{read } d_{1i} \]

\[ \text{write } d_{ni} \]

\[ \text{read } d_{ni} \]

\[ \text{write } t_i \]

\[ 4 \cdot (n+1) \cdot (i-1) + 2 \cdot (n+1) \]

\[ \text{read } s_i \]

\[ \text{write } e_{1i} \]

\[ \text{read } e_{1i} \]

\[ \text{write } e_{ni} \]

\[ \text{read } e_{ni} \]

\[ \text{write } u_i \]

\[ 4 \cdot (n+1) \cdot i \]

\[ \text{write } u_i \]
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 replication associated to \( v_i \) is the sequence in (a). Similarly, we can show that the other alternative is the sequence in (b). \( \square \)

For the replications associated to clauses, there are only \( c \) possible schedules on \( A \) for a clause with \( c \) disjuncts.

**Lemma 6** Let \( R, A, N \) be the output of \( \text{SAT2RepSc}(C_1, \ldots, C_n, v_1, \ldots, v_n) \), and \( r = \{d_j \mid v_i \in C_j\} \cup \{e_{ji} \mid \neg v_i \in C_j\} \) be the replication associated to clause \( C_j \). Let \( l = 4 \cdot (n + 1) \cdot (i - 1) \) and \( s = 2 \cdot (j - 1) \). There are only \( c \) (\( c = \text{number of disjuncts in } C_j \)) alternative schedules for the replication associated to \( C_j \):

\[
\begin{align*}
\{ & l + s, (d_{ji}, l + s + 1) \} \mid v_i \in C_j \} \\
\{ & l + 2 \cdot (n + 1) + s, (e_{ji}, l + 2 \cdot (n + 1) + s + 1) \} \\
& \neg v_i \in C_j \}
\end{align*}
\]

**Proof:** The schedule for the replication associated to \( C_j \) must start with a read from \( d_j \). From \( \text{SAT2RepSc} \), 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: \( \{l + s \mid v_i \in C_j\} \cup \{l + 2 \cdot (n + 1) + s \mid \neg v_i \in C_j\} \). 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 \( \text{SAT2RepSc} \)) 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 replication. 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 schedules for the replication associated to $C_j$ are the $c$ alternatives described above. □

**Lemma 1:** Let $R, A, N$ be the output of $SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. 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 for replications $R$ 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 schedule for the replications in $R$:

(i) For each variable $v_i$: if $a(v_i)$ is true, then consider the schedule in Lemma 5 (b) for the replication associated to $v_i$. Otherwise, consider the schedule in Lemma 5 (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 $\{l+s, (d_{ji}, l+s+1)\}$ for the replication associated to $C_j$. If $a(v_i)$ is false and $\neg v_i \in C_j$, then consider the schedule $\{l+2 \cdot (n+1) + s, (e_{ji}, l+2 \cdot (n+1) + s+1)\}$ for the replication associated to $C_j$.

To prove that the above is a replication schedule for $R$ on $A$, it suffices to show that the schedules for each replication do not overlap each other (i.e., they do not compete for neither disk bandwidth nor memory). The schedules for replications associated to 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 for replications associated to 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 the 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 the 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 replication schedule \(R\) on \(A\) during period \([0, N]\). \(\square\)

We now prove the other direction, if the replications scheduling instance yielded by \(SAT2RepSc\) has a solution then the input SAT instance has a solution.

**Lemma 2:** Let \(R, A, N\) be the output of \(SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)\). If there is a schedule for replications \(R\) 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 for the replications associated to variables in \(SATRepSc\) follow either pattern of Lemma 5. Therefore, a valid truth assignment \(a\) is as follows: \(a(v_i)\) is true if the execution of the replication associated to \(v_i\) follows the pattern in Lemma 5 (b), and is false if it follows the pattern in Lemma 5 (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 schedule of the replication associated to \(C_j\) must be either (Lemma 6): (a) \(\{4 \cdot (n + 1) \cdot (i - 1) + 2 \cdot (j - 1), (d_{ji}, 4 \cdot (n + 1) \cdot (i - 1) + 2 \cdot (j - 1) + 1)\}\), if \(v_i \in C_j\); or (b) \(\{4 \cdot (n + 1) \cdot (i - 1) + 2 \cdot (n + 1) + 2 \cdot (j - 1), (e_{ji}, 4 \cdot (n + 1) \cdot (i - 1) + 2 \cdot (n + 1) + 2 \cdot (j - 1) + 1)\}\), if \(\neg v_i \in C_j\). Suppose that the schedule of \(C_j\) is as described in (a). Then the schedule of the replication associated to \(v_i\) must follow the pattern in Lemma 5 (b). Otherwise, there would be a conflict for the disk bandwidth of \(d_{ji}\) 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 that \(C_j\) 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. \(\square\)
Appendix 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 $SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$ has a solution.

**Lemma 7** Let $R, A, N$ be the output of $SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. 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 $P$ such that any resource schedule for $\{P_0, \ldots, P_{m-1}\}$ that yields a one-time interval latency must schedule replications $R$ during time interval $[0, N]$.

**Proof:** The system must replicate 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 replicate 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 replicate a page from drive $d_i$ to either drive in the target set of the replication associated to $C_i$ or to $w_{N+k+i}$, before interval $N$.

The system must schedule replications $R$ before instant $N$. $Ret$ retrieves each referenced page during $[0, N]$ only once and does not pre-fetch pages (Observation 1 (2)). Therefore, the disk bandwidth requirement from a drive during $[0, N]$ is at least the disk bandwidth required by $Ret$ during the same period. Then the source, target and intermediate drives in the schedule of a replication 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 replication.
Replications $R$ must be scheduled after instant 0 otherwise the latency would be higher than one time interval. Starting the schedule of a replication 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 replication. In sum, the system must schedule replications $R$ during $[0, N]$.

**Lemma 3:** Let $R, A, N$ be the output of $SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. 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 $RS$ for replications $R$ 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:** Construct a resource schedule as follows:

**Step 1:** Include retrieval schedule $Ret$ in Observation 1 (1).

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

**Step 3:** Include the schedule of replications $RS$.

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 replication $r \in R$ 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 scheduling of $r$ on $A$ during $[0, N]$. Where $R, A, N$ is the output of $SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$.

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

**Definition:** Given a replication schedule

$$\{t_1^i, (d_1^i, [t_2^i, t_3^i]), \ldots, (d_{k_i-1}^i, [t_{2k_i-2}^i, t_{2k_i-1}^i]), (target^i_p, t^i_{2k_i})\}$$
the memory requirements of the replication schedule during $[-p, N]$ is defined as the sequence:

\[
\begin{array}{cccccccc}
0, \ldots, 0 & 1, \ldots, 1 & 0, \ldots, 0 & \ldots & 1, \ldots, 1 & 0, \ldots, 0 \\
p+\ell'_1+1 & \ell'_2-\ell'_1 & \ell'_2-\ell'_1 & \ell'_{2k_t}-\ell'_{2k_t-1} & \ell'_{2k_t}-\ell'_{2k_t-1} & \ell'_{2k_t}-\ell'_{2k_t-1} & \ell'_{2k_t}-\ell'_{2k_t-1} & \ell'_{2k_t}-\ell'_{2k_t-1}
\end{array}
\]

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

Let $R, A, N$ be the output of $SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. For any resource schedule for $SAT2ResSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$, there are two alternatives to schedule a replication $r$ in $R$: (1) schedule $r$ based on the system load $A$, or (2) modify the retrieval schedule $Ret$ (See observation 1) to accommodate the replication $r$. For the second alternative, the system might schedule additional replications. For example, to schedule a replication of page $a$ from drive $s$ to drive $t$. The system might utilize the disk bandwidth used to retrieve a page $b$ from $s$ in $Ret$ to read the page $a$ from $s$ at interval $t_i$. Then, page $b$ can either be pre-fetched at an earlier time interval, or be replicated from $s$ to a disk $u$ with available bandwidth at $t_i$ so that $b$ can be retrieved from $u$ at $t_i$. If there is not memory to pre-fetch $b$, then the system is forced to replicate $b$. Therefore the schedule of the replication (from $s$ to $t$) includes the schedule of a new replication (from $s$ to $u$). The additional replications 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 $R, A, N$ be the output of $SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. If the scheduling of a replication $r_0$ in $R$ modifies the retrieval schedule $Ret$ in such a way that additional replications $r_1, \ldots, r_n$, must be scheduled. Then the extension of $r_0$ is the set of replications $\{r_0, \ldots, r_n\}$.

For example, suppose that the system schedules replications $r_1$ and $r_2$ to accommodate replication $r_0$. Suppose that the memory requirements of the replications schedules are as follows: The timing for replication $r_0$ is $(0, 0, 0, 0, 0, 1, 0, 1, 0)$, for $r_1$ is $(0, 0, 0, 1, 0, 0, 0, 0, 0)$, and for $r_2$ is $(0, 0, 1, 0, 0, 0, 0, 0, 0)$. 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 replications schedules define a partial order on the schedules.

**Definition:** A replication schedule $sr_1$ is greater (smaller) than a replication 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 8** Let $\{P_0, \ldots, P_{m-1}\}, P, B, C, D$ be the output of $SAT2ResSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. Let $R, A, N$ be the output of $SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k)$. 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 a replication associated to clause $C_j$. Then, there exists some 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 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 retrieval from a target drive that was scheduled during $l$ in $Ret$, or (2) to replicate a page $a$ retrieved from a target drive during $l$ in $Ret$ 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., $v_i, t_i, e_{ji}, d_{ji}$) do not have disk bandwidth available at an earlier time interval. Therefore, the replication 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, an alternative $sr'$ to schedule the replication associated with $C_j$ in Lemma 6 is smaller than $sr$. □
Lemma 9 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 R, A, N be the output of SAT2RepSc(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 a replication associated to variable \(v_i\). Then, there exists some 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 8, 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) replicate \(a\) from \(s_i\) to a disk with available bandwidth during \(l - 1\) so that \(a\) can be retrieved from the other disk, or (2) replicate 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 replicate a page \((a\ or\ b)\) from \(s_i\) to \(d_{ni}\). To schedule this replication, the write a page 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 an alternative \(sr'\) to schedule the replication associated with \(v_i\) in Lemma 5 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 replicate a page from \(s_i\) to either \(d_n\) (if there is available bandwidth in \(d_n\) or \(d_{(n-1)i}\). Because there is not available bandwidth for drive \(d_n\) during an odd time interval, the system has to schedule the replication from \(s_i\) to \(d_{(n-1)i}\). Similar reasoning can be applied iteratively to conclude that an alternative \(sr'\) to schedule the replication associated with \(v_i\) in Lemma 5 is smaller than \(sr\). □

We now conclude the proof of the other direction of the equivalence of instances.
Lemma 4: Let \( R, A, N \) be the output of \( SAT2RepSc(C_1, \ldots, C_n, v_1, \ldots, v_k) \). 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 replications \( R \) 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 replication schedule for \( R \) on \( A \) during \([0, N]\). Consider the following schedule for replications \( R \) on \( A \) \((Sa)\): For each replication \( r \in R \), consider a schedule \( sr' \) in Lemmas 8 and 9 such that \( sr' < sr \), where \( sr \) is the schedule of \( r \)'s extension in \( Sc \).

Because there is not a replication schedule for \( R \) on \( A \), there must be two replications \( r_1 \) 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 to a variable \( v_i \) and \( r_2 \) to 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 \( Sc \). \( \square \)