Journal of Intelligent Information Systems

Volume , No. , Summer 1994

Journal of Intelligent Information Systems, , 1–31 (1994)

Multimedia Object Modeling and Storage Allocation Strategies

KINGSLEY C. NWOSU NWOSUCK@DONALD.AIX.KINGSTON.IBM.COM
IBM POWER Parallel Systems Div.,
MS/992, Kingston, NY 12401

C. Y. ROGER CHEN RCHEN@CAT.SYR.EDU
ECE Dept., Syracuse University, Syracuse, NY 13244

P. BRUCE BERRA BERRA@CAT.SYR.EDU
CASE Center & ECE Dept., Syracuse University, Syracuse, NY 13244

Editor: Ramesh Jain and Anca Ralescu

Abstract. The improvements in disk speeds have not kept up with improvements in processor and memory speeds. Many techniques have been proposed and utilized to maximize the bandwidths of storage devices. These techniques have proven useful for conventional data, but when applied to multimedia data, they tend to be insufficient or inefficient due to the diversified data types, bandwidth requirements, file sizes and structures of complex objects of multimedia data. In this paper, we discuss the design of an efficient multimedia object allocation strategy that strives to achieve the expected retrieval rates and I/O computational requirements of objects; and also effectively balances the loads on the storage devices. We define a multimedia object model, describe the multimedia object and storage device characteristics, the classification of the multimedia objects according to their I/O requirements, and the fragmentation strategies. We use a bipartite graph model for mapping of fragments to storage devices. A cost function based on a disk utilization per allocated space, the amount of free space, and the bandwidth of a storage device is used to determine the optimal allocation for an object's data.

Keywords: *bipartite matching, efficient allocation, fragmentation, multimedia object modeling, storage allocation*

1. Introduction

Over the years, different areas of computing technology have developed, each in its own respect and most often ignoring other concurrent advancements in related areas. Most of these advances, in any given area, have been primarily to solve specific problems related to a given application area. The speed and bandwidth of computer networks have increased substantially so that in the near future we will have gigabit per second networks. The speed of the CPU has increased significantly as a result of advances in VLSI technology to the point that today we have tens of hundreds of MIPS processors and we expect even faster ones in the near future. The capacity of mass storage devices has increased enormously and at the same time their physical sizes have shrunk. The presence of powerful personal computers and work-stations with very high resolution monitors and low prices have become

pervasive. The price of memory has also decreased considerably. The availability of multicomputers with high processing and storage capabilities has changed our vision of computers. Today, we have multicomputers with several hundreds of processors and enormously large disk space and the future looks brighter for multicomputers with thousands of processors and larger disk space. In spite of the heterogeneity of computing systems, it has become increasingly easy to interconnect different systems across the networks. There have been considerable improvements in digital compression algorithms and the development of specialized processors for digitalization and processing of video and related media information.

Since most of the technological advances progressed in parallel, it has become obvious that the integration and exploitation of these advances will bring the computing technology and information processing to a different dimension. This dimension is the capability to process those media information that we, over the years, thought too large to store, process, and transport and too diverse to integrate; that of *multimedia*.

The term *multimedia* has different meanings depending on the operating environment. To some people it simply means the integration of text and graphics on a stand-alone system like a personal computer. To others it means the integration of text, graphics, and audio. In general, multimedia is a combination of motion and still video, special effects, synthetic video, graphics, text, voice, audio, and images. Multimedia information processing encompasses the integrated generation, representation, processing, storage, and dissemination of independent machine processable information expressed in multifarious time dependent and independent media.

A unique feature of multimedia is the highly diversified media types and file sizes. In order to avoid dealing with the heterogeneity of multimedia data, multimedia applications are usually developed using an object-oriented approach. By using the object-oriented approach, multimedia data can be processed and manipulated by users in a universal way, regardless of the media types and sizes of objects. However, from a system's point of view, many problems arise in supporting such an object-oriented multimedia system. Among the problems of processing multimedia information, the most serious one is related to the storage. This is due to the fact that processor speed, memory speed, and memory size have grown exponentially over the past few years [Bell, 1984; Myers, 1986], while disk speeds have improved at a far slower rate. Consequently, the speed of the disk rather than the speed of the CPU's is the limiting factor in many applications. For real-time information retrieval and presentation, it is imperative that data, for a given medium, be retrievable at some given rate. The rates for some media are very high for current storage devices. The most conspicuous of these is in the area of digital video. For example, the video data object based on the *NTSC* standard requires that video data be retrievable at a rate of 45 Mbits/sec. However, the peak speed of a magnetic disk drive is about 10 Mbits/sec. and CD-ROMs operate at 1.2 Mbits/sec. To meet the bandwidth requirement of a full-motion video file, it is clear that the file has to be spilt into multiple sub-files, stored in different disks; when needed, an

interleaving technique will be performed to combine the multiple data streams into a single data stream and then present it to the user.

1.1. Related Work

Different storage techniques and file system types have been proposed and utilized to maximize the bandwidth of data retrieval in a given computing system. A storage technique normally falls into one of the following groups: (1) Data stripping/de-clustering, (2) Data compression, and (3) Data contiguity/clustering.

Data stripping has been used in many variations based on the stripping unit for a disk array. It operates by allocating a specified number of data blocks of an object (file) across different storage devices that can be accessed in parallel. In [Kim, 1986] the author studies synchronized disk interleaving as a high performance mass storage system architecture. She proposes a stripping unit of one byte (byte interleaving) and using queuing models, she finds that under light loads, byte interleaving yields higher throughput than a collection of non-cooperating disks. In [Livny, et al., 1987] the authors propose a de-clustering scheme where the stripping unit is 1 track and they compare the performance with an infinitely large stripping unit (clustering) and conclude that de-clustering consistently yields higher throughput than clustering. This is attributed to two factors: 1) de-clustering allows increased parallelism and 2) de-clustering load-balances the disks by spreading each file across multiple disks. Reference [Chen & Patterson, 1990] addresses how to stripe data to get maximum performance from disks. It examines how to choose the stripping unit and also synthesize the rules for determining the best stripping unit for a given range of workloads. It shows how the choice of stripe unit depends on the number of outstanding requests in the disk system at any given time and the *average positioning* $\times$ *data transfer rate* of the disks. The primary objective of disk striping or disk arrays is to maximize a system's throughput. The allocation unit is usually uniformly defined among a configuration of disk arrays. In a multimedia environment where different objects have different real-time requirements, the same allocation units configured for a disk array are applied to all the objects. The way an object is stored (number of blocks per disk) is determined by the configuration of the disk array. This is unacceptable and inadequate for a real-time multimedia information processing.

It is undoubtably obvious that reducing the storage size (*data compression*) of an object reduces the amount of that object's data that should be retrieved from storage and also enhances the chance of achieving the required display data availability rate of the object. Numerous compression techniques [Held, 1987; Storer, 1988; LeGall, 1991] have been proposed and utilized for data compression, but with the advent of multimedia, the importance and necessity for data compression have become even more crucial and essential. Today, there are numerous very efficient compression algorithms and specialized compression processors. References [Held, 1987; Storer, 1988] discuss many of the data compression algorithms and tech-

niques; the necessary data analysis for near-optimal data compression technique for a given problem; the memory requirements, and timing considerations.

The concept of *data contiguity* (e.g., [Bell, et al. 1988; Zupan, 1987; Deogun, et al. 1984]) tries to maximize bandwidth of devices by organizing related objects contiguously in a storage device thereby reducing the seek time. This relationship is normally based on data that are usually accessed or needed together. The fact that one organizes related data together does not imply that the real-time bandwidth will be achieved especially when the storage device bandwidth is much smaller than the required object retrieval rate as is pervasive in multimedia environments.

Several file system level approaches (e.g., [Anderson & Osawa, 1992; Rangan & Vin, 1991; Weikum, et al. 1991]) have been utilized for efficient management of data for local and remote users. For example, [Weikum, et al. 1991] deals with the dynamic allocation in disk arrays for complex objects but does not address the diverse characteristics of multimedia objects and associated problems with storage device configurations. In [Anderson & Osawa, 1992; Rangan & Vin, 1991], the authors have proposed some storage allocation techniques for multimedia objects based on a model that relates disk and device characteristics to the playback rates for sequences of continuously recorded audio samples or video frames (*strands*). These techniques, although innovating, do not go far enough in capturing other important properties of multimedia objects and the model does not provide a reliable cover for different I/O requirements for multimedia objects.

Here, we discuss the design of efficient multimedia object allocation strategies that strive to achieve the expected retrieval rates, I/O computational requirements, and balanced distribution of loads among the storage devices. We define storage device configuration and multimedia object models, describe the multimedia object and storage device characteristics, the classification of multimedia objects with respect to their I/O requirements, and the decomposition techniques. A bipartite graph model is presented which is used for mapping multimedia object fragments to storage devices. A cost function based on the cost induced by expected disk traffic per unit of allocated space, cost induced by free space with respect to the cumulative traffic requirement, and cost induced by the bandwidth factor is used to determine an efficient allocation of an object and to balance the loads on the storage devices.

2. Multimedia Object Model and Classification Criteria

In this section, a multimedia data storage model is proposed for multimedia information processing. Multimedia data are usually organized in multi-level complex objects, each represented by an object tree. An object tree for a *composite object* represents a conglomeration of related multimedia objects. An internal node of a composite object is called a *complex object*. Each leaf node in the composite object tree represents a storable unit. We refer to a leaf node as a *Data Element* (DE). A composite object and the associated DEs are dynamically created and stored in the system. To a user, an object tree represents an instance of a multimedia session.

Example: In Figure 1, $o8$ and $o9$ form the complex object $o6$ while $o6$ and $o7$ form the complex object $o3$. □

It is imperative that in the presence of a network connecting a number of storage devices, we must strive to exploit I/O parallelism. In order to achieve some degree of I/O parallelism, an object should be decomposed to allow for parallel retrieval or storage of the data.

Figure 1. An example of a composite multimedia object tree.

2.1. Effects of Storage Device Bandwidths

There are two motivating factors for striving for parallel I/O. In the first case, the bandwidth requirement (the amount of data expected per unit of time) exceeds the I/O transfer rate of a disk. Some objects require that specified amounts of their data be retrievable per unit time in order to satisfy the display data requirements. Due to the size and type of these objects, those availability rates are very high for the bandwidth of a storage device. For example, the video data object based on the *NTSC* standard requires that video data be retrievable at a rate of 45 Mbits/sec. In a magnetic disk storage device environment, we cannot achieve the expected retrieval rate since the peak speed of a magnetic disk is about 10 Mbits/sec. Therefore, we need to decompose an object into sub-objects capable of being stored in different disks, which, when needed, can be retrieved in parallel and combined into a single data stream. In the rest of the discussion, we refer to this type of problem as case 1.

2.2. Necessity for Parallel Access

The second case occurs when parallel access of an object is helpful in computations for either multiprocessors or vector processors, and the disk I/O transfer rate is not a limiting factor. In this case, there is no specified bandwidth requirement for an object's retrieval, but there is an underlying assumption that the bandwidths of the storage devices are sufficient to satisfy the objects data availability rate. That is, the data rate covered by each parallel access is expected to be less than the bandwidth of any of the storage devices. For example, considering an object which is basically an array of a large number of images, if the object is decomposed into multiple sub-objects stored in different disks, then multiple images can be retrieved

and processed in parallel. In the rest of the discussion, we refer to this requirement as case 2.

2.3. Necessity for Dual or None I/O Requirements

A third case arises as a derivative of the preceding two conditions. An object, under the third case, requires that a specific amount of data be retrievable by a specific number of parallel accesses. Unlike case 1 objects where the bandwidths of the storage devices pose some problems, case 3 objects assume that there exist some storage devices whose bandwidths satisfy the data requirement. Therefore, besides specifying the number of parallel accesses, it becomes necessary to stipulate the size of each parallel access so that it can be used to determine the storage devices that satisfy the requirement. In order to fully understand the differences between case 3 objects and others, assuming that $\{bw_1, bw_2, \ldots\}, \{sp_1, sp_2, \ldots\}$ are the sets of the storage device bandwidths and sizes of parallel access, respectively, we derive two interpretations:

1. The bandwidths of the storage devices satisfy the size of each parallel access. For example, given that the size of the parallel access of an object is sp_j, then

$$\forall i \; \exists bw_i \text{ such that } sp_j \leq bw_i.$$

   (At this time, we are not concerned with the number of storage devices that satisfy the requirement.)

2. There exists an object whose size of parallel access exceeds the bandwidth of any of the storage devices. For example, given the set of the sizes of the parallel accesses as above, then

$$\forall j \; \exists sp_j \text{ such that } \forall i \quad \nexists bw_i \text{ such that } sp_j < bw_i.$$

It is obvious that interpretation (2) incorporates cases 1 and 2 into one object, thereby, complicating its decomposition objectives. We cannot satisfy cases 1 and 2 in one object since those requirements are supposed to be mutually exclusive. As a result, an object that requires those conditions is classified as a case (1) object. Specifically, an object that requires that x bytes of its data be retrievable in y parallel accesses (where x is as discussed in interpretation (2)) is similar to a bandwidth of $x \times y$ as in case 1. On the other hand, interpretation (1) is consistent in that it does not incorporate two conflicting requirements. Therefore, in subsequent discussions, we refer to objects that satisfy interpretation (2) as case 3 objects.

A fourth case arises when an object's data availability is not affected by the bandwidths of the storage devices and does not benefit from any parallel access for either multi-processing or vector-processing. Most conventional text files fall into this case. In subsequent discussions, we refer to this requirement as case 4.

2.4. About the Object Model

The multimedia object model described here is a conceptual storage model. Its primary objective is to depict and represent, in one general storage framework, the different objects' I/O requirements that may exist in a multimedia information processing system. We do not, in any way, claim that the model is a multimedia database model. The properties and characteristics of multimedia object database model and storage model are different. The former are usually designed to capture the operational and manipulative requirements of multimedia data for a multimedia database system, and, therefore, incorporates the different data attributes and relationships. For example, the authors in [Woelk, et al. 1986; Masunga, 1989; Bancilhon, 1988] discuss several multimedia database object models which address the essential features of a multimedia database data model such as inheritance, encapsulation, class, etc. On the other hand, in spite of the differing properties of both models, the model presented here can handle and satisfy the I/O requirements of the objects of multimedia database systems. In other words, the data retrieval requirements of any of the other models can easily be embedded, and subsequently achieved, in our model.

2.5. Decomposition Analysis and Classifications

Having detailed the I/O and display requirements of various objects, we now present some analytical implications for object decomposition, storage allocations, and object classifications. In case 1, it is required that an object's data be distributed among the number of storage devices required to meet its I/O bandwidth requirement, i.e., the amount of data required per unit time. Furthermore, it requires that some minimum number of storage devices be available to achieve the parallelism. In case 2, only the amount of an object's data covered by the specified number of parallel accesses are retrievable per unit time. Consequently, the object is decomposed into a number of sub-objects equal to the specified number of parallel accesses where the size of each sub-object depends on the bandwidth of its target storage device. In case 3, the total amount of data retrievable per unit time is the product of the specified number of parallel accesses and the unit of each parallel access. Therefore, the object is decomposed into a number of sub-objects equal to the specified number of parallel accesses where the size of each sub-object is equal to the specified data availability rate.

Example: Figure 2 shows a composite object $O1$ which contains four DEs (i.e., $DE1, DE2, DE3$, and $DE4$), where $DE1$ is an object of case 1 and is decomposed into 4 fragments (i.e., $f1, f2, f3$, and $f4$), $DE2$ is an object of case 2 and is decomposed into 5 fragments (i.e., $f5, f6, f7, f8$, and $f9$), $DE3$ is an object of case 3 and it is decomposed into 3 fragments (i.e., $f10, f11$, and $f12$), and $DE4$ is an object of case 4 and it is not decomposed. □

Due to the fact that objects of case 1 require I/O bandwidths that, in most cases, exceed the bandwidths of the storage devices, and objects of case 3 require I/O bandwidths, that in some cases, exceed the bandwidths of the storage devices, their fragments usually will occupy most of the disk I/O bandwidth. On the other hand, fragments of cases 2 and 4 usually will only occupy partial disk I/O bandwidth. Note that in Figure 2, $DE4$ is not decomposed because parallel retrievals of $DE4$ do not help any computation or display.

Since the whole tree representing a composite object may be retrieved together, in order to benefit most from parallel disk retrievals, it is better to store all the fragments of decomposed DEs and DEs that are not decomposed in different disks. For example, in Figure 2, an efficient allocation will store $f1$, $f2$, $f3$, $f4$, $f5$, $f6$, $f7$, $f8$, $f9$, $f10$, $f11$, $f12$, and $DE4$ in different disks.

Figure 2. An example of composite object decomposition.

Reiterating, we refer to DEs of type $DE1$ as *class-one* DEs; objects of type $DE2$ as *class-two* DEs, DEs of type $DE3$ as *class-three* DEs, and DEs of type $DE4$ as *class-four* DEs. Any of the singly storable units, $f1$, $f2$, $f3$, $f4$, $f5$, $f6$, $f7$, $f8$, $f9$, $f10$, $f11$, $f12$, and $DE4$, will henceforth, also be referred to as an Allocation Unit (AU). An AU is a storable unit of data that must be contiguously allocated to a storage device.

2.6. Object Storage Allocation Problem Formulation

Based on the requirements discussed and the model presented, the multimedia object allocation problem then becomes:

1. Given a composite multimedia object, how can one decompose the DEs to build the AUs? What decomposition or fragmentation strategies are appropriate to generate the AUs of the DEs of a composite object such that the allocation of the AUs meets the expected objectives?

2. Given a number of AUs produced from the fragmentation strategy, how does one define an allocation process or technique? How does one determine the allocatability of an AU to a storage device? What mapping techniques, new or existing, should be used effectively between AUs and storage devices?

3. Having determined the storage devices to which an AU is allocatable, what criteria are necessary and sufficient in determining the most efficient storage device to store an AU? How are these factors computed and what are their cumulative effects on the storage devices?

4. Given that one has selected an allocation strategy, how can one demonstrate the effects on the throughput of a computing system? That is, what are the impacts on the throughput of different considerations of the allocation factors?

5. Given an allocation strategy, how can one demonstrate that it fairly and sufficiently balances the loads among the storage devices?

3. Multimedia Object and Storage Device Characteristics

The composite multimedia objects described here and their associated DEs have certain properties that are necessary for the storage allocation strategies to be proposed. Each of the attributes of a multimedia object or storage device plays an important role in determining an efficient allocation of a multimedia object.

3.1. Multimedia Object Characteristics

We now describe the various properties and characteristics of the different classes of multimedia objects. Due to the fact that the allocation of multimedia objects are performed per composite object, each composite multimedia object is denoted as O, each DE in a composite object is denoted as DE_i, and the number of DEs in O is denoted as n. The size of DE_i is denoted as DE_i^{size}.

Each class-one DE, DE_i, is associated with an *access frequency*, fd_i. The access frequency represents the relative probability that the DE will be requested for retrieval. It indicates how frequently the DE is used. The frequencies of class-one DEs are either staticly defined according to media types or dynamicly determined by the system at a DE's creation. For any DE_i, clearly, we expect that $0 < fd_i \leq 1$. For class-one DEs, since they have the inherent problem with I/O bandwidth, each class-one DE, DE_i, has an *expected retrieval rate*, err_i. This rate represents the minimum amount of the DE's data that should be retrieved per unit time in order to achieve some display or real-time requirements. The expected retrieval rate may depend on data format or compression technique as well.

Each class-two DE, DE_j, is associated with a *degree of parallelism*, dp_j, which indicates the degree of concurrent accesses to the DE that are required for computations in either multi-processors or vector processors. A DE's degree of parallelism indicates the number of parallel accesses to storage devices that are necessary to efficiently support the computing environment. Each class-two DE's access distribution and size are similarly defined and denoted as those of class-one DEs.

Each class-three DE has both an expected retrieval rate and a degree of parallelism. A class-three DE's expected retrieval rate is the minimum amount of data that should be retrieved from each of the number of storage devices specified by the degree of parallelism. When necessary, we refer to both the expected retrieval rate and degree of parallelism of a class-three DE as the *data-parallel rate*. Each class-three DE also has an access frequency.

3.2. Storage Device Characteristics and Grouping

Like the multimedia objects, the storage devices have certain properties and characteristics that are important for our storage strategies. The kth storage device is denoted as S_k. For each storage device, S_k, the maximum number of bytes retrievable per unit time (i.e., bandwidth) from the storage device is denoted by $BW(S_k)$. Furthermore, the total amount of space already allocated in the storage device, S_k, is denoted as S_k^{alloc} and the amount of free space is denoted as S_k^{free}. Since we may be dealing with a heterogeneous environment where the computing environment comprises different types of storage devices with different characteristics, the differing characteristics that are of paramount importance to us are the bandwidths of the storage devices. As a result of the heterogeneity of the storage devices, we group related devices together based on their bandwidths into storage device groups $SDG_1, SDG_2, \ldots, SDG_w$, where w is the number of different bandwidths in the system and the bandwidth of each storage device in SDG_i is $BW(SDG_i)$.

4. Problem and Allocation Analysis

It is important that we discuss the vital decisions that must be made to address the facets of the allocation process. Each allocation process comprises a composite object with its associated DEs. The storable elements are the DEs. A composite object is dynamically created and stored in the system with respect to the current status of the storage devices. For an allocation process, all the DEs of a composite object are considered simultaneously. The most common operation during an allocation process is the determination of the mappings of AUs to storage devices. Below, we describe, in a step-wise process, the allocation policies that are applied to different groups of the constituents of a composite object in determining the final policies for the allocation of the DEs.

4.1. Intra DE Allocation

Since a DE may be decomposed into a number of AUs, we must define an allocation policy that governs the allocation of those AUs. The *intra DE allocation* stipulates the allocation policy that must exist when allocating the AUs of a given DE with respect to other AUs of the same DE. For a class-one DE, the expected retrieval rate stipulates the amount of data that must be available per unit time. Likewise, for a class-two DE, its degree of parallelism indicates the number of parallel accesses that are necessary. An AU, as we stated, is a unit of allocation that must be stored in a storage device. Consequently, in order to guarantee that the expected retrieval rate, degree of parallelism, or data-parallel rate of a DE is achieved, we must be able to retrieve different AUs concurrently. This can only be accomplished if each of the AUs of a DE is stored in a different storage device. Figure 3 shows a possible

allocation of the AUs of a DE using the intra DE allocation policy. It shows that each AU of the DE is distinctly allocated to a different storage device.

Example: Given that $\{AU_{i,1}, \ldots, AU_{i,DE_i^{\#aus}}\}$ are the AUs of DE_i, if there exists $AU_{i,k}$ $(1 \leq k \leq DE_i^{\#aus})$ such that $AU_{i,k}$ is stored in S_h $(1 \leq h \leq m)$ then there should not exist $AU_{i,g}$ $(1 \leq g \leq DE_i^{\#aus}$ and $k \neq g)$ such that $AU_{i,g}$ is also stored in S_h. □

Figure 3. Allocating a DE with intra DE allocation policy.

The intra DE allocation also makes it possible for one to retrieve the entire data of a DE in a unit time. The class-four DEs are not governed by this allocation policy since each class-four DE comprises one AU.

4.2. Intra Complex Object Allocation

Above the level of the DEs, it is necessary that we specify the allocation rules for a conglomeration of DEs, i.e., a complex object. As is prevalent in most complex object oriented systems, users or applications may need to access all the data associated with a complex object concurrently. In order to achieve that, we must allocate each AU of a complex object to a different storage device. When that is successfully achieved, it is possible then to access all the applicable storage devices concurrently to retrieve all the data associated with a complex object. For example, Figure 4 shows the allocations of the AUs of the complex objects of a composite object. It indicates that an AU of a complex object is distinctly allocated to a storage device exclusively with respect to other AUs of the same complex object.

Figure 4. An example allocation for intra complex object allocation policy.

Example: Consider, for example, that $DE_1, \ldots, DE_j$ are DEs of a complex object and $\{AU_{x1,1}, AU_{x1,2}, \ldots, a_{x1,DE_{x1}^{\#aus}}\}$, $\{AU_{x2,1}, AU_{x2,2}, \ldots, AU_{x2,DE_{x2}^{\#aus}}\}$ are the AUs of DE_{x1} and DE_{x2}, respectively, where $1 \leq x1, x2 \leq j$. If there exists $z1$ $(1 \leq z1 \leq DE_{x1}^{\#aus})$ such that $AU_{x1,z1}$ is stored in S_h $(1 \leq h \leq m)$ then there

should not exist $z2$ ($1 \leq z2 \leq DE_{x2}^{\#aus}$) such that $AU_{x2,z2}$ is also stored in S_h. □

4.3. Inter Composite Object Allocation

Another access situation that we must consider is when an access crosses multiple composite objects. In an environment with a limitless number of storage devices, we can afford to store every AU in a different storage device. However, that situation is unrealistic. We, sometimes, expect the situation depicted in Figure 5 to occur where composite objects $O1$ and $O2$ share $o2$. For allocation purposes, we logically think of shared DEs as, physically, belonging to each of the composite objects. For example, using Figure 5, if $O1$ were allocated first, then none of $O2$'s AUs should be allocated in the same device with $o2$.

Figure 5. An example allocation when a DE is shared.

5. Fragmentation Strategies

Having established that we must decompose the objects of some of our object classes, we now describe the decomposition techniques necessary to achieve the I/O and display requirements of these objects. For class-one DEs, we have to decompose their data into fragments that foster parallel reads to achieve their expected retrieval rates. In order to obtain these fragments, we have to determine the degree of decomposition of a given DE. We need to compute the number of storage devices that can be accessed in parallel to satisfy the retrievability requirement. A DE's *Storage Set* is the set of the numbers of storage devices needed to achieve its expected retrieval rate based on the amount of data that can be retrieved from each storage device per unit time (i.e., bandwidth) in parallel.

5.1. Decompositions for Homogeneous Storage Devices

As a result of the homogeneity of the storage devices in a homogeneous configuration of storage devices, the fragmentation process and requirements are characteristically different and simpler than its counterpart. The computation of the number of storage devices needed is straightforward since all of the storage devices have the same bandwidth. Therefore, each DE can be associated with, at most, one storage set. The number of storage devices required for each storage set is called its *storage*

set size. That is the number of storage devices needed to achieve a DE's expected retrieval rate or data-parallel rate based on the amount of data that can be retrieved from each storage device per unit time in parallel. For homogeneous storage devices, given a class-one DE, DE_i, its storage set size, denoted by ss_i^{size}, for storage devices whose bandwidths are σ is computed as:

$$ss_i^{size} = \lceil \frac{err_i}{\sigma} \rceil. \quad (1)$$

Example: In a computing environment where the bandwidth of each storage device is $10MB/sec.$, if a class-one DE has an expected retrieval rate of $45MB/sec.$, then its storage set size is 5. That is, in order to satisfy the amount of data that must be available per unit time, it is necessary that a parallel access to 5 storage devices be performed per I/O request. □

For a class-two DE, DE_i, the storage set size is implicitly defined by the degree of parallelism; therefore, $ss_i^{size} = dp_i$. On the other hand, for a class-three DE, DE_i, each expected retrieval rate is satisfied by a single storage device; however, its storage set size depends on its degree of parallelism; therefore, $ss_i^{size} = dp_i$. Having computed the storage set size, one of these two situations arises:

1. $ss_i^{size} > m$, or
2. $ss_i^{size} \leq m$

where m is the number of storage devices in the computing environment. We expect the first situation to rarely or never happen. If the first situation occurs, then one cannot achieve err_i or dp_i because we need to access in parallel, at a minimum, ss_i^{size} storage devices.

The value of the storage set size of a DE also represents the number of AUs to be generated from the DE. For class-one and class-two DEs, each AU generated for the storage set size can be targeted to any of the storage devices. However, for class-three DEs, an AU can only be targeted to a storage device if that storage device's bandwidth is greater than or equal to the expected retrieval rate.

5.2. Decompositions for Heterogeneous Storage Devices

Due to the ramifications of the differences in storage devices in a heterogeneous configuration of storage devices, the fragmentation process and requirements are more cumbersome than for homogeneous devices. We must consider combinations of different storage devices with different bandwidths. Consequently, a DE can have multiple storage sets. An element of a storage set indicates the number of storage devices of the corresponding group of devices that are necessary to achieve the real time requirement of a DE. It is obvious that the number of possible different combinations for storage sets could become very large. As a result, some constraints,

as described below, are utilized to minimize the number of storage sets. To that end, therefore, the number of sets in a DE's storage set is reduced to at most $2^w - 1$ where w is the number of different bandwidth values for the storage devices in a system. Let $ss_k = \{y_1, y_2, \ldots\}$ be the kth storage set of a DE where y_i is the number of storage devices from SDG_i needed to achieve the expected retrieval rate.

5.2.1. Determining Valid Storage Sets for Class-one DEs

There are some requirements that must be met by each storage set before it is considered valid for a DE. For each combination of device groups that forms a storage set, each group must be represented by at least one storage device. We determine the number of storage devices from each group with the aim of minimizing the amount of data retrievable per unit time from those storage devices with respect to the expected retrieval rate of a DE. The amount of data retrievable from the storage devices of a storage set must not be less than the expected retrieval rate and should not exceed the expected retrieval rate by a value greater than the bandwidth of any one of the device groups. The storage device groups are arranged in order of decreasing bandwidths. The number of storage devices per device group in a storage set decreases with increasing bandwidth, when applicable. That is, for a given storage set, the number of storage devices for a device group of lower bandwidth is always at least as many as that of the next group with a higher bandwidth. A storage set is valid if reducing the number of storage devices of a device group violates the minimization rule described. Stated formally, therefore, given err_j and $ss_k = \{y_1, y_2, y_3\}$ then the following conditions must hold:

1. $BW(SDG_1) > BW(SDG_2) > BW(SDG_3)$,
2. $[y_1 BW(SDG_1) + y_2 BW(SDG_2) + y_3 BW(SDG_3)] \geq err_j$, and
   - (A) $[(y_1 - 1)BW(SDG_1) + y_2 BW(SDG_2) + y_3 BW(SDG_3)] < err_j$,
   - (B) $[y_1 BW(SDG_1) + (y_2 - 1)BW(SDG_2) + y_3 BW(SDG_3)] < err_j$,
   - (C) $[y_1 BW(SDG_1) + y_2 BW(SDG_2) + (y_3 - 1)BW(SDG_3)] < err_j$.
3. $y_1 \leq y_2 \leq y_3$.

If any of the conditions above is violated, then the corresponding storage set is invalid. The above conditions are equivalent to solving the integer linear programming problem:

$$y_1 BW(SDG_1) + y_2 BW(SDG_2) + y_3 BW(SDG_3) \geq err_j,$$
$$y_1 \leq y_2, \;\; y_2 \leq y_3, \;\; y_3 > 0.$$

An ss_k with $\mid ss_k \mid = g$ is *acceptable* if
(1) $\sum_{i=1}^{g} y_i \leq m$, and
(2) $\forall j \; [j = 1 : g] \; y_j \leq \mid SDG_j \mid$.

Example: Consider a class-one DE of size $120KB$ and bandwidth requirement of $60KB/s$, given that $SDG_1 = \{S_1, S_2, S_3\}$, $SDG_2 = \{S_4, S_5\}$, $SDG_3 = \{S_6, S_7, S_8\}$, $BW(SDG_1) = 30KB/s$, $BW(SDG_2) = 20KB/s$, and $BW(SDG_3) = 10KB/s$. The sets of the combinations of groups of storage devices are $\{SDG_1\}$, $\{SDG_2\}$, $\{SDG_3\}$, $\{SDG_1, SDG_2\}$, $\{SDG_1, SDG_2, SDG_3\}$, $\{SDG_1, SDG_3\}$, and $\{SDG_2, SDG_3\}$. The valid storage sets are $ss_1 = \{2\}$, $ss_2 = \{3\}$, $ss_3 = \{6\}$, $ss_4 = \{1, 2\}$, $ss_5 = \{1, 3\}$, $ss_6 = \{2, 2\}$, and $ss_7 = \{1, 1, 1\}$. Obviously, ss_2, and ss_3 are not possible. Furthermore, without the constraints discussed above, it is evident that given $\{SDG_2, SDG_3\}$, the storage sets $\{3, 0\}$, $\{1, 4\}$, $\{0, 6\}$, $\{2, 3\}$, and $\{1, 5\}$ can achieve the I/O or display requirements. However, applying the constraints limits the option to $\{2, 2\}$. □

If none of the storage sets of a DE is acceptable, then we cannot allocate the DE. When that happens, a message may be sent to the user suggesting a higher degree of data compression on the class-one DEs or a lower degree of parallelism for class-two or class-three DEs. Otherwise, we know that given DE_j^{size} and $\sum_{h=1}^{g} y_h$ storage devices, we can retrieve $\sum_{i=1}^{g} y_i BW(SDG_i)$ bytes of data from each of the applicable storage devices in parallel and, consequently, achieve err_j.

5.2.2. Determining Valid Storage Sets for Other Classes

The above discussion on fragmentation strategy is in the context of class-one DEs. In the case of a class-two DE, the degree of parallelism represents the expected number of AUs. Therefore, for a class-two DE, a storage set is valid if its storage set size is equal to the DE's degree of parallelism. Furthermore, conditions (1) and (2) and the acceptability requirement discussed above must hold. That is, a storage set ss_k with $\mid ss_k \mid = g$ for a class-two DE, DE_j, is acceptable if

$$\sum_{i=1}^{g} y_i = dp_j \tag{2}$$

If the applications of the above rules yield no storage set, then a storage set whose storage set size minimally exceeds the degree of parallelism is selected, i.e., storage set sizes $ss_j^{size} + \epsilon$ for $(\epsilon = 1, 2, 3, \ldots, m)$ are successively considered with each unacceptable storage set computed. Also, in the case of class-three DEs, the degree of parallelism represents the expected number of AUs to be generated. This number of AUs can be generated in a number of ways also limited by $2^w - 1$. In addition to satisfying the conditions for class-two DEs, as discussed above, the condition below must also be satisfied:

$$\forall i \ [i = 1 : g] \ BW(SDG_i) \geq err_j$$

The above condition also requires that the chosen storage devices have bandwidths that are closest to the expected retrieval rate. When no valid storage set is obtained

with the strict application of the storage set size, then the same process for class-two DEs is applied. In the case of class-four DEs, each DE is made up of one storage set consisting of one AU.

5.3. Generating Atomic Units' Constituents

Previously, we have only identified an AU as a non-decomposable unit that must be contiguously stored in a storage device. We now describe the integral morphology of an AU with respect to the target storage device. Since the size of each data retrieved per unit time from each storage device is equal to its bandwidth, each AU stored in a storage device comprises a number of chunks of data whose size is equal to the bandwidth of the storage device. We call each of these chunks of data a *Storage Element* (SE). An AU then consists of one or more SEs arranged in such a way that guarantees parallel retrieval of contiguous data. Remember that ss_k^{size} is the *storage set size* of a storage set for a DE, DE_k. Consequently, each ss_k comprises ss_k^{size} AUs where each AU is denoted as $AU_{k,l},\ 1 \leq l \leq ss_k^{size}$. We represent the number of SEs in $AU_{k,l}$ as $AU_{k,l}^{\#ses}$. The sum of all the sizes of all the SEs of all the AUs of a DE must be at least as large as the size of the DE. Furthermore, reducing any AU of any storage set of a DE by one SE must violate the preceding condition. Therefore, if $AU_{i,j}^{sesize}$ is the size of each SE in $AU_{i,j}$, then

1. $\sum_{l=1}^{ss_k^{size}} (AU_{k,l}^{\#ses} \times AU_{k,l}^{sesize}) \geq DE_k^{size}$ and

   (A) reducing any of the numbers of SEs by one or more values violates the preceding condition. For example, given DE_k, $((AU_{k,q}^{\#ses} - 1) \times AU_{k,q}^{sesize}) + (\forall t\ [t = 1 : ss_k^{size}, t \neq q] \sum AU_{k,t}^{\#ses} \times AU_{k,t}^{sesize}) < DE_k^{size}$.

When we generate the SEs, we sequentially index them and assign them to the applicable AUs. We denote as $f_q \Rightarrow AU_{k,l}$ the fact that SE f_q belongs to $AU_{k,l}$. Therefore, for ss_k,

$$\forall q\ [q = 1 : AU_{k,l}^{\#ses}] \text{ if } f_q \Rightarrow AU_{k,l}\ \ (1 \leq l \leq DE_k^{\#aus}) \text{ then } \forall h\ [[h = 1 : AU_{k,l}^{\#ses}] \text{ and } (h\ mod\ ss_k^{size} = q\ mod\ ss_k^{size}) \text{ and } h \neq q]\ f_h \Rightarrow AU_{k,l}.$$

As is evident from building the AUs, the data represented by each AU do not constitute contiguous data in a DE. The physical addresses of the SEs in an AU differ by some factors of the bandwidths of the storage devices. This is a consequence of data interleaving which is essential for achieving parallel I/O for a stream of data.

Example: A class-one DE of size $120KB$ and expected retrieval rate of $40KB$ per unit time, in an environment with storage devices of $20KB$ bandwidth, can be split into 6 SEs, $\{f_1, f_2, \ldots, f_6\}$, and then grouped into 2 AUs, $\{AU_1, AU_2\}$. as shown in Figure 6. □

Figure 7 shows the SEs and AUs from the preceding example on page 14 for a heterogeneous configuration of storage devices. The numbers beside the boxes represent the physical addresses of the SEs in a DE.

Figure 6. The SEs and AUs of a 120KB DE.

Figure 7. A sample generation of storage sets.

6. The Proposed Mapping Techniques

Assuming that the fragmentation strategies have decomposed the DEs of a composite object into a number of AUs accordingly, then given a number of storage devices, we have to specify the conditions for *allocatability* of an AU to a storage device. In other words, what conditions must be satisfied by both an AU and a storage device that make the AU allocatable to the storage device? Obviously, the paramount condition is the amount of free space in the storage device. For an AU to fit in some free space in a storage device, that space must be contiguous and large enough. As we discussed in the fragmentation strategies, each AU comprises a number of SEs where each SE is the size of the bandwidth of one of the storage device groups, SDG_i. The sizes of the SEs play a paramount role in achieving the expected retrieval rates or degree of parallelism of objects. Therefore, an AU could only be allocatable to a storage device if the storage device's bandwidth is equal to the size of the AU's SE. Remember that in a homogeneous storage device configuration, all the AUs consist of SEs of same size and every AU is considered for allocation to every storage device if the free space requirement is met.

Given an AU, we have a list of storage devices to which it is allocatable. We must select one of those storage devices as the most efficient storage device for the AU. To accomplish this, one must consider the effects of allocating a given AU to all the possible storage devices to which it is allocatable. If the AUs and storage devices represent nodes in a graph, then we can construct an edge from an AU to a storage device to which that AU is allocatable. If we assign a weight to each of these nodes, then one can, using some criteria, determine the best allocation for a given AU. In order to fairly balance the loads, we need to specify some factors that will help to determine an efficient allocation of an AU. Prominent among these factors are the

current status of a storage device with respect to the AUs already allocated, the effect of the free space and bandwidth of the storage device.

6.1. Effects of Access Frequency and Size

Two of the important characteristics of a multimedia object are the access frequency and size. Although they convey different properties of an object, in order to establish a reliable means of efficient allocation of objects, we must develop a function to qualitatively quantify their integrated functionalities. We define the current status function to capture those objectives. The current status function is defined in terms of an AU's size and access frequency and we call it the *expected disk traffic requirement* (*edtr*), and represent the *edtr* of AU_i as AU_i^{edtr}. The *edtr* must always be defined such that a DE's access frequency is emphasized and certain characteristics of the multimedia environment also taken into consideration. A large AU with high access frequency will have a higher traffic requirement than an AU of equal size but smaller access frequency. We denote the effect of the access frequency of AU_i on AU_i^{edtr} as a function $F(AU_i^{freq})$, where AU_i^{freq} is the access frequency of AU_i. The primary objective is that the effective value of an access frequency should relatively increase with its real value. A number of functions can be developed on the access frequencies and sizes of objects that exhibit the required behavior. If AU_i^{size} is the size of AU_i, then one such function is

$$AU_i^{edtr} = AU_i^{size} \times F(AU_i^{freq}) \tag{3a}$$

where

$$F(AU_i^{freq}) = \frac{1}{1 - AU_i^{freq}} \tag{3b}$$

We use the function F to represent the storage effects of the access frequency and size of an AU in a given storage device. Over a period of time, a number of AUs are stored in a storage device. We need to capture the total utilization effects of those AUs with respect to their *edtr*'s. The current *cumulative traffic requirement* (*ctr*) of S_k, represents the cumulative effect of the *edtr*'s of the AUs already stored on S_k, and we denote that as S_k^{ctr}. Assuming that a total of h AUs have already been stored in S_k, then

$$S_k^{ctr} = \sum_{i=1}^{h} AU_i^{edtr} \tag{4}$$

6.2. Effects of Utilization per Unit of Allocated Space

It is often necessary to reduce the granularity of parameters to capture their essential implications. The cumulative traffic requirement of a storage device is an

indication of the expected access to the storage device with respect to the AUs allocated to it. Consequently, a reasonable motivation is to allocate the next AU to the storage device with lowest cumulative traffic requirement. However, that factor alone does not determine an efficient storage device to allocate an AU. In order to get a more vivid understanding of the effect of the cumulative traffic requirement, we need to determine the *expected disk traffic per unit of allocated space* in a storage device. It is obtained by dividing the current cumulative traffic requirement of a storage device by the total amount of space already allocated in that device. That value indicates the disk traffic exerted per unit of allocated space in a given storage device. We extend the expected disk traffic per unit of allocated space and determine the *induced expected disk traffic per unit of allocated space* (*iedt*). That is the expected disk traffic per unit of allocated space if the AU under consideration is allocated to a given storage device. We denote as S_k^{iedt} the induced expected disk traffic per unit of allocated space by an AU on S_k. Consequently, for a given AU_i,

$$S_k^{iedt} = \frac{S_k^{ctr} + AU_i^{edtr}}{S_k^{alloc} + AU_i^{size}} \tag{5}$$

After a successful allocation of an AU to S_k, S_k^{iedt} is used to compute the new S_k^{ctr}; that is, S_k^{ctr} becomes $S_k^{iedt} \times (S_k^{alloc} + AU_i^{size})$.

6.3. Effects of Storage Device Free Space

It is undoubtably obvious that the amount of free space in a storage device plays a role in determining the current and future utilization of a storage device. The fact that a storage device has a low cumulative traffic requirement relative to another storage device does not convincingly indicate that it is under-utilized relatively. If the storage device with higher cumulative traffic requirement has considerably larger free space, then it is imperative that relative to their available spaces, it is under-utilized.

Example: Let S_1^{ctr}, S_2^{ctr} be the current cumulative traffic requirements and S_1^{free}, S_2^{free} the amount of free space in S_1 and S_2, respectively. If $S_1^{ctr} < S_2^{ctr}$, one's first inclination is to select S_1 for allocation. However, if we know that $S_2^{free} >> S_1^{free}$, then it becomes obvious that S_1 has a higher utilization relative to its size; therefore, it may be more efficient to allocate to S_2 whose utilization is less relative to its size. □

In order to meaningfully apply the effect of the size of free space in a storage device, we should express the free space relative to the cumulative traffic requirement of the storage device; and so we denote that as S_k^{efree}, for S_k. We compute the value as

$$S_k^{efree} = \frac{S_k^{ctr}}{S_k^{free}} \tag{6}$$

6.4. Effects of Storage Device Bandwidth

The fact that a storage device has a high cumulative traffic requirement relative to another storage device should not imply an automatic rejection of that storage device. If a storage device has a high cumulative traffic requirement but a high bandwidth, then the resultant effect of the cumulative traffic requirement is reduced by the fact that a large chunk of data is retrievable per unit time. This condition is analogous to the preceding example if $S_1^{free} = BW(S_1)$ and $S_2^{free} = BW(S_2)$. Consequently, we must account for the impacts of the bandwidths of the storage devices, in a heterogeneous environment, when determining efficient allocation of AUs.

It is obvious that, in terms of magnitude, the bandwidth of a storage device is comparatively smaller than its total allocated space and free space (in most cases). Therefore, expressing the impacts of allocated space, free space, and bandwidth with respect to the cumulative traffic requirement require that the impact from the bandwidth be expressed in such a way that it does not obscure its counterparts. The impact from the bandwidth should be related to the disparity between the bandwidths, i.e., if there is a considerable gap between the smallest and largest bandwidths of the storage devices under consideration, then the bandwidth factor should also reflect that situation. Therefore, the bandwidth factor can be expressed as the ratio of the difference between the maximum and minimum bandwidths of the storage devices with respect to the maximum bandwidth. However, we know that given that MAX and MIN are the maximum and minimum values of a set of positive integer numbers, then $0 \leq \frac{MAX-MIN}{MAX} < 1$. Furthermore, any number between the MAX and MIN also falls within that range if MIN is substituted with the number. For example, if $MAX = 100$ and $MIN = 30$, therefore, $\frac{MAX-MIN}{MAX} = 0.7$. For $30 \leq NUM \leq 100$, $0 \leq \frac{MAX-NUM}{MAX} \leq 0.7$. Obviously, using this condition for the bandwidth factor produces a decreasing effect on the allocation factors. As a result, this factor should be normalized to a value greater than 1, but not too large to re-introduce the magnitude problem. One of the best ways to achieve the objective is to express the above situation as $1 \leq \frac{MAX-MIN}{MAX} < 2$. In this way, we retain the factorial implications of the difference in values without adversely affecting the utilization factors. This range can be modified to use different degrees to emphasize the bandwidth factor, as our example below shows. We denote the bandwidth factor as bwf.

6.5. Developing the Allocation Cost Value

Having discussed the pertinent factors and functions for determining an efficient allocation of an AU, we now develop the unified and integrated value for object allocation based on the factors described. From the preceding discussions on the allocation factors, it becomes evident that an optimal allocation of an AU depends on

1. induced expected disk traffic per unit of allocated space ($S_k^{iedt},\ 1 \leq k \leq m$),
2. free space with respect to the cumulative traffic requirement ($S_k^{efree},\ 1 \leq k \leq m$), and
3. bandwidth factor (bwf).

These three values incorporate all the allocation issues discussed.

It is obvious that the effects of the first two factors can either be increased or decreased, in a given computing environment, to emphasize or de-emphasize its relative importance. It may be required that the real utilization effect of the induced expected disk traffic per unit of allocated space of an AU be accentuated to obtain the actual value. Likewise, the effect of free space may also be accentuated. Consequently, we introduce two coefficients, e_1 and e_2 which are used to accentuate induced expected disk traffic per unit of allocated space and effects of free space, respectively. The values of the coefficients, relative to each other, indicates which one is emphasized or de-emphasized. For example, a system implementation may require that the effects of free space in a storage device be emphasized vis-á-vis the disk utilization per unit of allocated space. In that case $e_2 > e_1$. In order to control the resultant effects of the coefficients, we recommend that the coefficients (e_1, e_2) be in the range of zero and 1. If no accentuation is required for either factor then $e_1 = e_2 = 1$.

In the discussion about the bandwidth effects, we explained that we need to keep the bandwidth factor within a given range. In order to accomplish that, we need to utilize a control value. The coefficient e_3 is used to control the effects of the bandwidth factor and it is determined from the maximum and minimum bandwidths in the system (as described above). As an example, e_3 can be selected such that the bandwidth factor is in the range 1.5 to 1.99. That is, $1.5 \leq 1 + \frac{e_3 BW_{max} - BW_{min}}{e_3 BW_{max}} < 1.0$, where BW_{max} is the maximum bandwidth of the storage devices to which an AU is allocatable and BW_{min} is the minimum bandwidth of the storage devices to which an AU is allocatable.

Example: Given that $BW_{min} = 10MB$ and $BW_{max} = 50MB$, if $e_3 = 2$, then the bandwidth factor is in the range of 1.5 to 1.9. On the other hand, if $e_3 = 3$, then the bandwidth factor is in the range of 1.67 to 1.93. The latter range relatively has a larger impact on the utilization factor than the former range. □

Therefore, we compute the bandwidth factor as

$$bwf = 1 + \left[\frac{(e_3 BW_{max} - BW_{S_k})}{e_3 BW_{max}}\right] \tag{7}$$

where BW_{S_k} is the bandwidth of the storage device currently under consideration from the set of storage devices allocatable to the AU.

Consequently, we develop an allocation cost value, $\mathcal{C}$, based on the effect of bandwidths on the accentuated values of the utilization per unit of allocated space and

free space. We denote the allocation cost value of AU_i to S_k as $\mathcal{C}_{i,k}$ and compute it as

$$\mathcal{C}_{i,k} = (e_1 S_k^{iedt} + e_2 S_k^{efree}) \times bwf \tag{8}$$

6.6. Using the Allocation Cost Value

There are certain rules that are necessary for the utilization of the cost value during the determination of optimal allocations of objects. Given an AU and the $\mathcal{C}$ costs of allocating it to different storage devices, we select the storage device with minimum cost. We denote the fact that AU_i is allocatable to the storage device, S_k, as $AU_i \perp S_k$. Therefore, $AU_i \perp S_k$ if

1. AU_i belongs to AUs of class-one or class-three DEs, and
   (A) $AU_i^{sesize} = BW(S_k)$,
   (B) $S_k^{free} \geq AU_i^{size}$.
2. AU_i belongs to AUs of class-two or class-four DEs, and
   (A) $S_k^{free} \geq AU_i^{size}$.

Our minimization goal is such that given g AUs of a composite multimedia object and $\mathcal{C}_{i,k}$ as the cost of allocating AU_i to the storage device S_k, therefore,

$$\text{minimize } \sum_{i=1}^{g} \mathcal{C}_{i,k} \quad \text{where } \exists S_k \ (1 \leq k \leq m) \text{ such that } AU_i \perp S_k.$$

Example: Table 1 shows the current values for 6 storage devices, and Table 2 shows the sizes and frequencies of 5 AUs of a composite multimedia object. After determining allocatabilities of the AUs and computing the $\mathcal{C}$ costs, Table 3 shows the $\mathcal{C}$ cost of each AU to the storage device to which it is allocatable. In this example, $e_1 = e_2 = 1$ and $e_3 = 2$. Applying the allocation and minimization rules, we have

$$AU_1 \perp S_5,$$
$$AU_2 \perp S_1,$$
$$AU_3 \perp S_6,$$
$$AU_4 \perp S_4,$$
$$AU_5 \perp S_3,$$

with a total cost of 210.54. □

[h]

Table 1. A sample of storage devices' current values used for allocation decision.

	S_1	S_2	S_3	S_4	S_5	S_6
Free space	10MB	100KB	500MB	100MB	1MB	15MB
Allocated space	5MB	120MB	10KB	500KB	50MB	70MB
Current cumulative traffic requirements	15728640	262144000	102400	1048576	125829120	104857600
Bandwidths	1MB	1MB	10MB	15MB	15MB	10MB

[h]

Table 2. A sample of sizes and frequencies of some AUs.

	AU_1	AU_2	AU_3	AU_4	AU_5
Size	50KB	75KB	1.5MB	100KB	3MB
Frequency	0.4	0.19	0.10	0.3	0.01

[h]

Table 3. $\mathcal{C}$ costs from Tables 1 and 2.

	S_1	S_2	S_3	S_4	S_5	S_6
AU_1	8.82	5038.76	-	-	183.60	-
AU_2	8.80	5038.76	-	2.93	-	-
AU_3	-	-	1.95	2.03	-	13.48
AU_4	-	-	3.68	2.93	183.60	-
AU_5	7.38	-	1.73	-	-	13.46

Obviously, our allocation problem is the selection of the optimal storage devices for AUs of a composite multimedia object with the aim of minimizing the cumulative cost value. This selection process is similar to the classical problem of the bipartite matching problem which has been applied to numerous problems such as the *max-flow problem* [Cormen, et al. 1990] and *bipartite weighted matching problem* [Papadimitriou, et al. 1982], also known as the *assignment problem*. Several algorithms have been developed to solve these problems; however, the one that best fits our problem is the *Hungarian Method* [Papadimitriou, et al. 1982]. Before we discuss how to achieve the optimal mapping with the Hungarian Method, we will describe the bipartite graph model on which the mapping algorithm depends.

7. The Bipartite Graph Model

One of the widely studied graphs is the *bipartite graph* [Cormen, et al. 1990]. It has been used to solve numerous combinatorial problems. A bipartite graph is an undirected graph $G = (V, E)$ in which V is partitioned into two sets V_1 and V_2 such that $(u, v) \in E$ implies that either $u \in V_1$ and $v \in V_2$ or $u \in V_2$ and $v \in V_1$. In other words, all edges go between the two sets V_1 and V_2. The bipartite graph used here consists of two sets of vertices, V_a and V_s. These two sets of vertices are referred to as *vertex partitions*. The V_a vertex partition is made up of the AUs of a composite multimedia object while the V_s vertices are made up of the storage

devices. An edge can only exist between two vertices if they do not belong to the same vertex partition. In our bipartite model, every AU of class-one, class-two, class-three, or class-four DE belongs to V_a every storage device belongs to V_s. In order to build a bipartite graph, we need the list of AUs and storage devices. The list of AUs make up the V_a vertex partition while the list of storage devices make up the V_s vertex partition. Using our earlier definitions of allocatability, an edge can only exist between two nodes from different vertex partitions if the allocatability requirements are met. An AU has $m^{'}$ edges where $m^{'}$ is the number of storage devices to which it is allocatable. We label each edge with the $\mathcal{C}$ cost of allocating an AU to a storage device.

7.1. Complete and Incomplete Bipartite Graphs

The fact that we have constructed a bipartite graph does not imply that we can confidently apply the matching process. There are some bipartite graphs that do not allow complete mapping of a composite object's AUs. Our bipartite graph is *complete* if every class-four or class-three AU in V_a vertex partition has an edge to a node in V_s vertex partition and every AU of at least one storage set of every class-one or class-two DE has an edge to a node in V_s vertex partition. Otherwise, the graph is *incomplete*. Certain characteristics of a bipartite graph make it susceptible to *incompleteness*. Obviously, a bipartite graph, G, is incomplete if

1. there is an AU that does not have an edge to any storage device,
2. there are two or more AUs that have edges to only one storage device,
3. any number of AUs have edges to a number of storage devices, but the number of storage devices is less than the number of edges.

Consequently, given the AUs of some valid storage sets of the DEs of a composite object, we must preprocess the expected bipartite graph for completeness. An incomplete bipartite graph can only be made complete by coalescing holes in the storage devices or reducing the number of AUs of some DEs, if possible. We consider reducing the number of AUs if the total number of AUs of a composite object exceeds the number of storage devices. As an effective way of reducing the number of AUs, we remove the storage sets of a DE when multiple storage sets exist. The storage sets of a DE are removed in order of decreasing storage length. The intention is to remove those storage sets that require, comparatively, more storage devices. However, each DE must be represented by at least one storage set. We reduce the storage sets of class-two DEs, class-three, and class-one DEs in that order. If necessary, AUs of class-four DEs are coalesced. When we cannot form a complete bipartite graph for a composite object, the user may be have to modify certain characteristics of some of the DEs, such as the bandwidth requirements or higher degree of data compression. For example, Figure 8 shows the bipartite graph built from Tables 1, 2, and 3. Using the Hungarian Method, the bipartite

matching yields the mapping indicated by the bold faced edges. However, during preprocessing, the bipartite graph in Figure 9 is determined to be incomplete and so we do apply the Hungarian Method to it. One can easily notice that a DE in a complete bipartite graph may have more than one storage set; however, only one storage set is necessary for the proper allocation of the DE. In that case, we select the storage set whose total $\mathcal{C}$ cost is minimal.

Figure 8. An example of matching using the Hungarian Method.

Figure 9. An example of an unallocatable bipartite graph.

8. Simulation Model

Our primary objective is to efficiently allocate multimedia objects to storage devices. In achieving that efficiency, we need to satisfy the real time requirements of the individual DE of a composite multimedia object and also fairly distribute the loads among the devices. Satisfying the real time requirement is an inherent part of the fragmentation strategies and, therefore, is achieved by a successful mapping of a DE's AUs to some storage devices. In order to get a better understanding of the extent of data distribution from the allocation techniques and criteria, we need to analyze some simulation results. We use the final cumulative cost value of the storage devices to determine the extent of the distributions. We generated 3 groups of devices where each group has common characteristics such as the bandwidth and size. Each group comprises 10 storage devices. For multimedia DEs, the sizes, frequencies, expected retrieval rates, and degree of parallelisms, where applicable, were uniformly and randomly generated. The classification of a DE was also randomly determined. We ran three sets of composite objects; the first comprised a fair mixture of DEs of different sizes; the second set comprised a large number of DEs with very small sizes, while the third comprised a large number of DEs with very large sizes.

[h]

Table 4. Data distribution for simulations

Sample 1		Sample 2		Sample 3	
Size	fd	Size	fd	Size	fd
< 2KB	.160	< 2KB	.340	< 2KB	.010
< 10KB	.147	< 10KB	.170	< 10KB	.020
< 100KB	.130	< 100KB	.120	< 100KB	.040
< 500KB	.120	< 500KB	.100	< 500KB	.050
< 1MB	.110	< 1MB	.080	< 1MB	.070
< 10MB	.093	< 10MB	.070	< 10MB	.080
< 50MB	.080	< 50MB	.050	< 50MB	.100
< 200MB	.067	< 200MB	.040	< 200MB	.120
< 400MB	.053	< 400MB	.020	< 400MB	.170
< 500MB	.040	< 500MB	.010	< 500MB	.340

8.1. Simulation Results

We show below the results of applying the allocation strategies described in this research. In the results shown below, 750 multimedia DEs were utilized where each composite multimedia object consisted of a number of these DEs. Each DE is of size between 1 byte and 500MB. The expected retrieval rates for the objects ranged from .125KB to 30MB. Every composite object consisted of a number of DEs that was randomly generated. Group 1 of the devices has size of 100MB and bandwidth of 1MB; group 2 has size of 75MB and bandwidth of 750KB while group 3 has size of 50MB and bandwidth of 500KB. These values were arbitrary, but reasonably, chosen. Table 4 shows the distribution, according to frequencies, of the total number of DEs generated for each experiment (Samples 1, 2, and 3). In order to emphasize on the frequency and at the same time reduce the range of the cumulative traffic requirements of the storage devices the expected disk traffic function used for AU_i is

$$AU_i^{edtr} = AU_i^{size} \times [\frac{1}{1 - AU_i^{freq}}] \tag{9}$$

Different values of e_1 and e_2, that emphasized different induced costs were tested. The results shown below are for $e_1 = e_2 = 1$. Irrespective of the relationships between the accentuating values, the cumulative traffic requirements of the storage devices within a group are fairly balanced as indicated by the relative uniformity of the final cumulative cost values of the storage devices within a device group. Figure 10 shows the final cumulative traffic requirements of the storage devices when a fair mixture of DEs of different sizes were used. Figure 11 shows the results when more DEs of very small sizes were used, while Figure 12 shows the results from using more DEs with large sizes. Devices 1 to 10 are group 3 devices, devices 11 to 20 are group 2 devices, while devices 21 to 30 are group 3 devices. The increase or decrease in the final cumulative cost of each storage device at each experiment is indicative of the sizes of the objects. As the graphs show, within a group of storage

devices, the final cumulative allocation costs of the storage devices are relatively uniform. The relative uniformity of the final cumulative allocation costs indicate that the utilizations of the storage devices, with respect to the allocation criteria defined, are fairly and evenly distributed. Therefore, the expected frequency of accesses to related storage devices do not differ substantially.

Figure 10. A sample load distribution (fairly mixed sizes of DEs, i.e., Sample 1).

Figure 11. A sample load distribution (more DEs of very small sizes i.e., Sample2).

Figure 12. A sample load distribution (more DEs of very large sizes, i.e., Sample 3).

9. Conclusions

We have presented a multimedia object model and described an allocation strategy for multimedia objects based on that model. We described different ways of splitting multimedia objects and discussed the pertinent characteristics of the multimedia objects and storage devices that are necessary for our allocation strategy. We classified the objects of composite multimedia objects into three classes based on their I/O requirements, presented some fragmentation strategies and discussed other factors that strive to achieve our objectives. We have associated an expected access frequency with each multimedia object. This frequencies are either staticly defined for each media type or dynamicly determined by the user at the creation of the multimedia object. A bipartite graph model is presented and its characteristics

discussed. The bipartite graph presented forms the basis for multimedia object allocation to storage devices. A matching process between the AUs and storage devices using a cost function determines the efficient allocation of an AU to a storage device via the Hungarian Method for bipartite matching. We have also shown, through a simulation model, that our strategies achieve the expected objectives.

As was alluded to in the introduction, many techniques and methods have been utilized over the years to improve the retrieval rates of data from secondary storage devices. These techniques have proven very useful until the advent of multimedia. The primary problem in a multimedia environment is the variable message sizes and retrieval rate requirements of the media types. Due to these variations in object characteristics, properties, and requirements, it has become inevitable that more intelligent methodologies and techniques be developed to handle these ramifications. The proposed allocation strategy has some advantages over some of the conventional approaches.

One of the primary advantages of the proposed allocation strategy lies in its model. The multimedia object model described here surpasses other known models that have been proposed for efficient storage of multimedia objects due to the fact that it is more exhaustive in covering different objects with respect to their I/O and display requirements than its counterparts. Furthermore, the allocation process lends itself to dynamic configurability. It is easily extendible to incorporate any storage device configuration. Any number of devices can be added to the configuration without affecting future object mapping or previously mapped objects.

The primary goal of conventional storage allocation techniques is to maximize the bandwidth of the storage devices. The expected retrieval rate of an object is not part of the equation. In a multimedia environment, it is necessary that the expected retrieval rates be taken into account when storing these objects, especially class-one objects, in secondary devices. The fact that one has maximized the bandwidth of some storage devices does not imply that the required data will be retrieved at the expected rate. Merely clustering data to reduce seek and rotational overheads does not usually solve the problem. If the bandwidth of the storage devices is much lower compared with the expected retrieval rates of an object, then clustering in conjunction with an intelligent allocation strategy may be necessary.

As we stated earlier, a high compression ratio has the tendency to be lossy. There are some media types in a multimedia computing environment that we cannot afford to compress lossyly. In our allocation strategy, we specify an expected retrieval rate for a data element. This retrieval rate could be a function of the compression ratio. Consequently, one does not have to compress high message size objects with high compression ratios. Specifically speaking, one can compress the objects with very low compression ratios and then factor them into the expected retrieval rates. Since there is a limit beyond which one cannot compress data without loss of information and there are some data types that we cannot afford to compress lossyly, then knowing fully well that the primary objective of compression is to

reduce the amount of data to be retrieved to achieve some bandwidth, it becomes imperative that strategies like ours are of paramount importance.

Since our strategy is based on one of the most widely studied graph types, it lends itself to easy utilization by any of the applicable graph-based algorithms.

Acknowledgements

We would like to thank Mrs. Robin Kaplan, my manager at IBM. I can safely say that without her, this and other related works towards my academic program would have been a dream deferred. It is refreshing to know that there are still people like her in this world.

Appendix

We now present the formal mathematical formulations for some of the important conditions and definitions presented.

Intra DE Allocation Policy

Given a class-one, class-two, or class-three DE, DE_i, the intra DE allocation policy states that

$$\forall k\ [k = 1 : DE_i^{\#aus}] \text{ if } AU_{i,k} \Rightarrow S_h \text{ then } \forall g\ [g = 1 : DE_i^{\#aus}]$$
$$\nexists AU_{i,g} \text{ such that } AU_{i,g} \Rightarrow S_h \quad (k \neq g) \text{ and } (1 \leq h \leq m). \tag{A.1}$$

Intra Complex Object Allocation

If $DE_1, \ldots, DE_j$ are DEs of a complex object, then

$$\forall x_1\ [x_1 = 1 : j]\ \forall x_2\ [x_2 = 1 : DE_{x_1}^{\#aus}] \text{ if } AU_{x_1,x_2} \Rightarrow S_h \text{ then}$$
$$\forall x_3\ [x_3 = 1 : j]\ \forall x_4\ [x_4 = 1 : DE_{x_3}^{\#aus}]$$
$$\nexists AU_{x_3,x_4} \text{ such that } AU_{x_3,x_4} \Rightarrow S_h, \quad (x_3 \neq x_1, 1 \leq h \leq m). \tag{A.2}$$

Inter Composite Object Allocation

$$\text{if } \exists DE_g \in O_i,\ DE_h \in O_k,\ DE_g = DE_h, \text{ and } \exists k_2 \text{ such that } AU_{h,k_2} \Rightarrow S_z \text{ then}$$
$$\nexists i_1, i_2 \text{ such that } AU_{i_1,i_2} \Rightarrow S_z,$$
$$(1 \leq k_2 \leq DE_h^{\#aus}, 1 \leq i_2 \leq DE_g^{\#aus}, 1 \leq z \leq m). \tag{A.3}$$

Conditions for a Bipartite Graph Incompleteness

A bipartite graph where DE_j has m_j storage sets is incomplete if any of the following occurs:

1. $\forall k\ [k = 1 : m_j]\ \forall l\ [l = 1 : DE_k^{\#aus}]\ \exists AU_{k,l}$ such that
$\forall h\ [h = 1 : m]\quad \nexists S_h\ \wedge\ AU_{k,l} \perp S_h$

2. $\forall k\ [k = 1 : m_j]\ \exists AU_{k,x_1}, AU_{k,x_2}, \ldots, AU_{k,x_n}$ such that
$AU_{k,x_1} \perp S_h, AU_{k,x_2} \perp S_h, \ldots, AU_{k,x_n} \perp S_h$
and
$\nexists S_g$ such that $AU_{k,x_1} \perp S_g \vee \ldots \vee AU_{k,x_n} \perp S_g$,
$(1 \leq h \leq m, 1 \leq g \leq m, h \neq g, 1 \leq x_z \leq DE_k^{\#aus})$.

3. $\exists A, G$ such that A is the set of AUs that have edges to the set of storage devices G and that $\mid A \mid > \mid G \mid$.

References

Anderson, D. P., and Osawa, Y. (1992). "A File for Continuous Media," *ACM Trans. on Computers*, Vol. 10-4, pp. 311-337.

Bancilhon, F. (1988). "Object-Oriented Database Systems," in *Proc. of the PODS* pp. 152-162.

Bell, C. G. (1984). "The mini and macro industries," *IEEE Computer*, Vol. 17-10, pp. 14-30.

Bell, A. D., McErlean, F, J., and Stewart, P. M. (1988). "Clustering Related Rules in Databases," *The Computer Journal*, Vol. 31-6, pp. 253-257.

Chen, P., and Patterson, D.(1990). "Maximizing Performance in a Striped Disk Array," in *Proc. ACM SIGARCH 17th Intern. Symp. on Comp. Arch.*, Seattle, WA, pp. 322-331.

Cormen, T.H., Leiserson, C.E., and Rivest, R.L. (1990). *Introduction to Algorithms*, The MIT Press.

Deogun, J. S., Raghava, V.V., and Tsou, T.K.W. (1984). "Organization of Clustered Files for Consecutive Retrieval," *ACM Trans. on Database Systems*, Vol. 9-12, pp. 646-671.

Held, G. (1987). *Data compression: techniques and applications, hardware and software considerations*, Wiley, NY.

Kim, M. Y.(1986). "Synchronized Disk Interleaving," *IEEE Trans. on Computers*, Vol. 35-11, pp. 978-988.

LeGall, D. J. (1991). "MPEG: a video compression standard for multimedia applications," *Communications of ACM*, Vol. 34-4.

Livny, M., Khoshafian, S., and Boral, H. (1987). " Multi-Disk Management Algorithms," in *Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Compter Systems*, pp. 67-77.

Masunga, Y. (1989). "An Object-Oriented Approach to Multimedia Database Organization and Management," in *Proc. International Symposium on Database Systems for Advanced Applications*, Seol, Korea, pp. 190-200.

Myers, W. (1986). "The Competitiveness of U.S.A. Disk Industry," *IEEE Computer*, Vol. 19-11, pp. 85-90.

Papadimitriou, C. H., and Steiglitz, K. (1982). *Combinatorial Optimization*, Prentice-Hall.

Rangan, P. V., and Vin, H. M. (1991). "Designing File Systems for Digital Video and Audio," in *Proc. 13th ACM Symp. on Operating Systems Principles*, Vol. 25-5, pp. 81-94.

Storer,J. (1988). *Data compression: methods and theory*, Computer Science Press, Rockville, MD.

Weikum, G., Zabback, P., and Scheuermann, P. (1991). "Dynamic File Allocation in Disk Arrays," in *Proc. 1991 ACM SIGMOD Intern. Conf. om Management of Data*, Denver, Colorado, pp. 406-415.

Woelk, D., Kim, W., and Luther, W. (1986). "An Object-Oriented Approach to Multimedia Databases," in *Proc. of 1986 ACM-SIGMOD Conf.* pp. 311-325.
Zupan, J. (1987). "Clustering of Large Data Sets," *Technometrics*, Vol. 29-11, pp. 497.