A Teradata Content-Based Multimedia Object Manager for Massively Parallel Architectures

W. O'Connell,[2] I.T. Ieong,[1] D. Schrader,[1] C. Watson,[1] G. Au,[1] A. Biliris,[3] S. Choo,[1] P. Colin,[1] G. Linderman,[1] E. Panagos,[3] J. Wang,[1] T. Walter[1]

[1]NCR Corporation, 100 N. Sepulveda Blvd., El Segundo, CA 90245 U.S.A.
[2]Lucent Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974 U.S.A.
[3]AT&T Research, 600 Mountain Ave., Murray Hill, NJ 07974 U.S.A.

Abstract

The Teradata Multimedia Object Manager is a general-purpose content analysis multimedia server designed for symmetric multiprocessing and massively parallel processing environments. The Multimedia Object Manager defines and manipulates user-defined functions (UDFs), which are invoked in parallel to analyze or manipulate the contents of multimedia objects. Several computationally intensive applications of this technology, which use large persistent datasets, include fingerprint matching, signature verification, face recognition, and speech recognition/translation.

Index Terms - Parallel Multimedia Database, Teradata, User-Defined Functions, Content-Based Analysis.

1 Introduction

Multimedia applications have become affordable and more widespread as computer technology advances. Performance acceleration of these applications requires moving computation into servers, especially when there is a large amount of shared data or a large amount of computing. There are two types of multimedia applications that require server support:

- Storage/retrieval
- Content-based analysis

The first kind of applications store and retrieve multimedia objects and perhaps perform a few simple transformations on the objects such as compression and decompression. Content analysis applications require far more processing on the multimedia objects. Examples include tumor recognition in a set of magnetic resonant images, face recognition in a set of images, speech recognition in a stored audio clip, and keyword searching of a document.

To support both kinds of applications, NCR is building a powerful multimedia database back-end server to store and share multimedia objects and perform computationally demanding operations on them. Symmetric multiprocessing (SMP) and massively parallel processing (MPP) hardware architectures provide such capabilities. The computer resources on these machines can be configured as a shared-nothing set of virtual processors (vprocs), each with its own CPU, memory, and disk space. These vprocs operate in a shared-nothing database environment [1]. Each vproc operates independently and in parallel with other vprocs. The number of vprocs on these machines is configurable and not necessarily equal to the number of physical CPUs. By using vprocs as addressable logical processors, system reconfiguration and fault resilience operations become more manageable. This flexibility allows addressable vprocs to be moved between physical nodes

The goal of the Multimedia Object Manager project at NCR is to provide content analysis capability on multimedia data on massively parallel platforms. The Multimedia Object Manager team is building a highly reusable technology platform that leverages expertise in the Teradata parallel database system and core Bell Labs competencies in multimedia. It is used as a parallel object engine that extends the Teradata database with SQL3 multimedia capabilities on both SMP and MPP platforms [2][3].

The system's infrastructure of programmable agents (tasks) allows system developers to load and execute user-defined functions (UDFs). UDFs are algorithms for content-based analysis of the objects. Agents are programmed with the sequence of vprocs to visit and the operations they will perform on each vproc. Agents may be dispatched to all or a subset of the vprocs. Each agent attempts to invoke the UDFs as close to the multimedia objects as possible to prevent large object movement on the interconnect; the idea is to move computation instead

SIGMOD '96 6/96 Montreal, Canada
© 1996 ACM 0-89791-794-4/96/0006...$3.50

of data. The Multimedia Object Manager is a generic extendable platform that incorporates transactional semantics for easily executing UDFs in parallel. Objects in the system are maintained in a parallel persistent storage system. Object location is hidden from the external user. Furthermore, as the system is reconfigured, objects are transferred transparently from one vproc to another.

The remainder of the paper is organized as follows: Section 2 highlights the key contributions. Section 3 provides a system overview. Section 4 discusses programmable agents. Section 5 covers parallel persistent storage. Section 6 discusses the User Defined Function library software. Section 7 describes feature extractions. Section 8 describes some preliminary performance evaluations. Section 9 relates experience with terabyte relational databases and reflects on their use with large multimedia datasets. Section 10 summarizes our work.

2 Key Contributions

Experience with the Teradata Database System has shown that performance is important as larger amounts of data are required and the complexity of computation increases. These demands require hardware/software designs that support effective scale-up and speed-up.[1] The Multimedia Object Manager project at NCR provides content analysis capability on multimedia data on massively parallel platforms with excellent scalability. Its parallel UDF invocation mechanisms for massively parallel platforms allow large data sets to be manipulated/analyzed. The system incorporates the following:

- A generic platform that is application-independent and extensible.
- Programmable agents to meet the needs of the application.
- An open interface and content-based object analysis capabilities through system-supplied multimedia functions (UDFs).
- Memory-mapping techniques that provide near-memory-speed pointer dereferencing (for memory-resident objects in the cache).
- Extensive support for large object data placement, including capabilities for striping across disk volumes.
- A UNIX-like file system application programming interface (API) for accessing very large objects.

1. NCR builds the largest commercial database with over 11 terabytes of on-line relational data on NCR MPP platforms using the Teradata database.

3 The Multimedia Object Manager System

As shown in Figure 1, the Multimedia Object Manager's software architecture consists of the following:

- An Interface Manager
- Object managers
- Multimedia infrastructure

The Interface Manager is the entry point into the Multimedia Object Manager. It comprises a set of global API services accessible from any vproc. These services are global transaction management, agent/event dispatching, and parallel file management. The dispatching service is used to parallelize a user request using agents spanning the appropriate vprocs for processing.

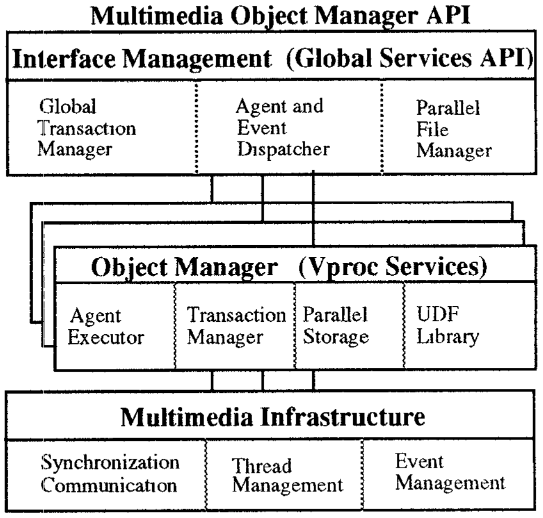
Multimedia Object Manager API
Interface Management (Global Services API)
Global Transaction Manager
Agent and Event Dispatcher
Parallel File Manager
Object Manager (Vproc Services)
Agent Executor
Transaction Manager
Parallel Storage
UDF Library
Multimedia Infrastructure
Synchronization Communication
Thread Management
Event Management

Figure 1 Multimedia Object Manager Software Architecture

The Object Managers are local to each vproc and provide access to the persistent storage and UDF library. Once the Interface Manager has dispatched the agents, one or more Object Managers may be involved in servicing the agents in parallel. The Multimedia Infrastructure provides communication and system resource management across the vprocs.

Three Multimedia Object Manager software components are discussed in this paper:

- Programmable agents [5]
- A parallel storage system
- A library of User-Defined Functions (UDFs) [4]

Each user request results in agents being dispatched, in parallel, to the appropriate vprocs. Each request is transformed into one or more agents by the Multimedia

Object Manager's agent API. The agents are programmable through scripts and are executed by an executor, one on each vproc. Agents may be dispersed over all vprocs in parallel and/or may migrate between vprocs. Each agent applies UDFs on a set of objects. As a result, computation is based on agents' execution of tasks, which is triggered by the receipt of events.

The Multimedia Object Manager's parallel storage system provides storage/retrieval capabilities with transactional semantics on all vprocs. All vprocs on one node share one instance of the parallel storage system. Since the Multimedia Object Manager operates in a shared-nothing architecture, each vproc supports computation, storage, and retrieval. All objects in the Multimedia Object Manager are maintained in the parallel persistent storage system. The location of each object is hidden from the external user. The Multimedia Object Manager provides an object identifier (OID) for each object. The parallel file system determines the location of an object, given its OID. Furthermore, as the system is reconfigured, objects are transferable from one vproc to another without changing the OID.

The UDF library provides an extensible set of abstract data type (ADT) algorithms to do content-based analysis on multimedia objects. These algorithms may be applied in parallel by agents on multiple vprocs. Each parallel agent applies the UDFs over a subset of the objects in a vproc. The system supplies a standard UDF library that contains simple building blocks, such as compression and decompression algorithms. The system also allows applications to load, store, and execute application-specific UDFs – examples include color histograms or texture mapping for image ADTs.

The following sections describe each of these three software components in more detail. Section 4 describes the programmable agents. Section 5 discusses the parallel storage system. Section 6 discusses the UDF library.

4 Programmable Agents

Computation in the system is based on the execution of tasks (agents). A computation is triggered by the receipt of a message (event). First, a user request invokes the dispatcher API. The dispatcher installs agents at appropriate vprocs in the system., depending on the location of data to be accessed. After these agents are installed, a start event is broadcast to initiate computation.[2] When, in the course of executing its script, an agent discovers an operation that has a set operand, and the objects in the set reside on other vprocs, it clones itself on the appropriate vprocs. The instructions depend on the user's request.

2. To reduce communication overhead, the broadcasts of the start event and the installation of agents can be piggybacked.

The agents remain dormant until events trigger their execution; the agents then execute on vprocs. During an agent's execution, it may do any of the following:

- Trigger other events
- Access the Multimedia Object Manager's persistent storage
- Execute UDFs

Agents may also migrate between vprocs. Agent migration is useful when a task requires one or more objects not residing on the same vproc. If the objects are not needed concurrently, the agent can migrate to the appropriate vprocs sequentially to access the objects. An agent is moved with its current state information; this is less costly than moving large multimedia objects over the interconnect. The migrating agent's context is piggybacked along with the event that is sent to its new location (vproc). When an agent needs to access two non-co-resident objects concurrently, remote object access (ROA) is used for the object residing on the another vproc. In this case, an object (usually the smaller) is moved over the interconnect. ROA over the interconnect is not done when objects reside on another vproc but on the same physical node (for example, an SMP node).

After execution completes, the agent waits until another event triggers its execution. An agent may remove itself from the system when it receives an appropriate event type.

4.1 Agent Model

The Multimedia Object Manager's agent model consists of the following four components:

- A script[3]
- Context
- Events (messages)
- A mailbox (to receive events) [6]

Agents are programmed through a scripting language, which defines the control flow and invocation of UDFs. Since the agent executor must understand the semantics of the script, the operations that an agent can perform range from very simple to very complex. Agent scripts are executed by an executor, one on each vproc. The agents' initial context is set via its initial event.

The model's basis is the Actors programming model. [6] The

3. A script is a set of instructions telling the agent what to do.

Multimedia Object Manager team has extended the model for appropriate step-execution of UDFs by the agent scripts [3][7]. The model provides a powerful transportation mechanism for data as well as an execution mechanism for control structures. An agent is a script along with its context and a mailbox. The arrival of an event in its mailbox triggers its execution [6]. This triggering causes the agent to execute the agent's script.

The degree of concurrency in the system can be controlled by adding or reducing the number of agents in the Multimedia Object Manager. Agents may be initially placed on all or a subset of the parallel vprocs [6].

4.2 Agent Flexibility and Scalability

Using an asynchronous light-weight agent model as the basic form of computation offers advantages over process-oriented models. In addition to easier dataflow modeling, [7] the agent infrastructure provides flexibility. Encapsulating the UDF execution into agents provides additional flexibility in optimizing system performance: Agents can be dispatched to the vprocs so the amount of remote object access required in UDF execution is minimized. Furthermore, agents fit nicely into a massively parallel environment. This is due to the asynchronous nature of agents concurrently overlapping communication and computation [7]. The computation scales easily by increasing the number of cloned agents on the system. These cloned agents operate in parallel with the only control synchronization at the point of merging answer sets.

4.3 Installing Agents

Agents can be sent/installed from any application to any vproc (through the Multimedia Object Manager's API) in one of the following three modes:

- Monocast
- Multicast
- Broadcast

These operations are very efficient on networks such as the BYNET interconnect in the NCR 5100M MPP machine [9]. In the multicast mode, the application can select a predefined subset of vprocs. Broadcast mode involves all vprocs. Sending events to agents can also be done in any mode.

The cloning of agents is a powerful model of computation. An agent makes a copy of itself on one or more vprocs; this is a scatter operation. The results of all the cloned agent computations are returned to the cloning agent; this is a gather operation. These scatter/gather operations may be done recursively by one or more of the cloned agents which forms a tree structure of computation. In the case of a tree structure, results propagate back to the root agent. The completion of the gathering operation returns the results to the requesting application.

4.4 Executing Events

Events received by an agent are processed in the order of arrival. Depending on the application's requirements, after an agent's execution finishes, it either returns the event message with the results to the sender, or it forwards the event to another vproc. By forwarding the event, control flow migrates through the various vprocs. This is not agent migration in which we piggyback the agent's context along with the event, but it is the ability to control the flow of computation by conditionally sending events to other agents based on the current agent's state information. This is a form of data flow [7].

This provides a very powerful computation paradigm in which a multi-step agent execution plan can be accomplished by a number of vprocs. More important, this occurs in a massively parallel environment where many agents execute simultaneously.

4.5 Agents and UDF Invocation

Each agent's script is a set of instructions indicating what operations (UDFs) should be applied to which object(s). Since the code for each UDF is replicated on all nodes, a UDF operation can be performed in parallel on all or a subset of vprocs.

UDFs can be treated as building blocks so that agents can channel the output of one UDF to the input of another; for example, a decompression UDF followed by a feature extraction UDF. This allows many simple transformations to be sequenced as a pipeline to accomplish complex operations. The system supplies a core set of UDFs that are available to applications. These core UDFs can be augmented by user-provided, application-specific UDFs.

For security or asset management reasons, each UDF invocation requires the requesting application to have access to all the UDF(s) involved in its operation(s). Each agent's context contains information about the originating user request. This allows the agent executor to verify that an application has security access to the desired UDF(s).

5 Parallel Persistent Storage System

The Multimedia Object Manager exploits parallelism, data availability, and partitioning strategies to obtain high performance and reliability on the system. Stringent data availability requirements dictate that application data must stay available despite hardware component failures. This is done by allowing objects to be accessed by adjacent physical nodes during node faults. Several partitioning strategies are used to distribute objects across the shared-nothing vprocs on the MPP to improve performance on various types of access patterns and to increase total storage capacity. Parallelism is achieved through data distribution, allowing UDF invocation to be done on each vproc in parallel.

The Multimedia Object Manager utilizes BeSS (Bell Labs Storage System) to provide persistent storage and allocation, concurrency control, and recovery [8]. BeSS is a high-performance configurable database storage manager that provides Two-Phase Commit and Two-Phase Locking protocols to the Multimedia Object Manager [10]. All operations in BeSS are performed under database transactional semantics. Additionally, BeSS uses memory mapping techniques that provide near-memory-speed pointer dereferences (for memory-resident objects). BeSS also offers extensive support for large objects, including capabilities for striping across disk volumes. The BeSS server running on each physical node has its own cache to maintain cache coherency and manage the shared virtual memory addresses for the cache frames. An interesting aspect of BeSS used in the Multimedia Object Manager is that it allows processes to access data cached on memory shared by other processes.

The system stores similarly typed objects in directories. Each directory is defined with one of numerous object distribution strategies over the vprocs. The system does not impose limits on the size of the objects. Single objects may span multiple disks on a vproc.

5.1 Data Organization

Data organization is key to the Multimedia Object Manager's shared-nothing massively parallel architecture. Data location can have a dramatic impact on performance, due to the expense of copying large multimedia objects between physical nodes and/or to the client. Object identifiers (OIDs) provide physical location independence allowing data objects to migrate when the machine scales (to balance the load). In addition, logical OIDs allow objects to be accessed through adjacent nodes during node faults.

Due to the stringent data availability requirement on high-end systems (where individual hardware faults have a higher probability of occurrence), multiple hardware data paths are provided to each physical storage device. This technique is used by the Teradata database running on the NCR System 5100M MPP machine [9]. This technique increases the availability of data in case of path failures (for example, a lost physical node). In cases of storage device failures (for example, a lost disk), data redundancy is used.

In the Multimedia Object Manager, the concept of a datapath is defined with two end points: an application process and the storage device. The datapath includes the interconnect network and a node connected to the storage device. In the case of the failure of a component in the datapath, the system does fast failover. Availability is accomplished by supplying multiple shared SCSI adapters on the same I/O bus (disk array). As a result, more than one node has access to any one physical disk.

The system is configured to perform best when using a particular subset of all paths (primary data paths). However, when failure occurs, nodes connected to alternate datapaths may become overloaded. One solution for an alternate path in the system is to provide a backup for each node. In the case of a path failure, each backup node will have twice its normal load, affecting overall performance. The minimal theoretical performance reduction in the system would then be $1/N$ for an N-node system. In reality, experience with the Teradata DBS indicates that this results in dreadful interconnection problems on a large system.

5.1.1 Cliques

The Multimedia Object Manager uses the Teradata database technique called cliques [3]. Cliques are groups of nodes of size C (where C is configurable, typically as four nodes). By grouping nodes in cliques, instead of by twos, the work of a failed node is split evenly across $C-1$ nodes. To achieve this, the disks on each node are split into $C-1$ sets; also, in addition to each disk being connected to a node by a primary path, an alternate path exists to each of the other $C-1$ nodes in the clique (see Figure 2).

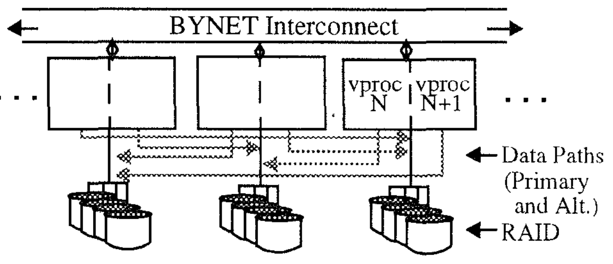

Figure 2 Clique Where Size C=3

However, cliques introduce a software problem. Because disks are divided into $C-1$ partitions, a clique of size C must be treated as $C*(C-1)$ independent subsets. For cliques of size $C=3$, we get six subsets. For $C=4$, we get 12 subsets. Large values of C exacerbate this problem and provide diminishing returns. That is why Teradata typically configures clique sizes of 4 (depending on size of machine). Each of the $C*(C-1)$ subsets are known as a virtual processor (vproc). When a node fails, each of the $C-1$ vprocs on that node move to a predetermined alternate node in the clique.

5.1.2 Data Redundancy

Figure 2 illustrates a clique configuration where $C=3$. In this case there are six vprocs (two per node). Each node has two alternate paths, one to each of the other node's storage devices. In case of a node failure, the computation from each of its vprocs goes to a preassigned node in its clique.

In case of storage device failure, data availability is maintained through data redundancy. Two types of redundancy are available to recover from failed storage devices: data duplication and redundant array of inexpensive disks (RAID). The Multimedia Object Manager uses RAID technology. Failed devices are hot swapped and rebuilt when on-line.

5.1.3 Data Distribution

Similarly typed objects are grouped logically into directories. Each directory is defined with one of numerous distribution strategies over the vprocs. The system does not impose limits on object size.[4] Single objects may span multiple disks on a vproc.

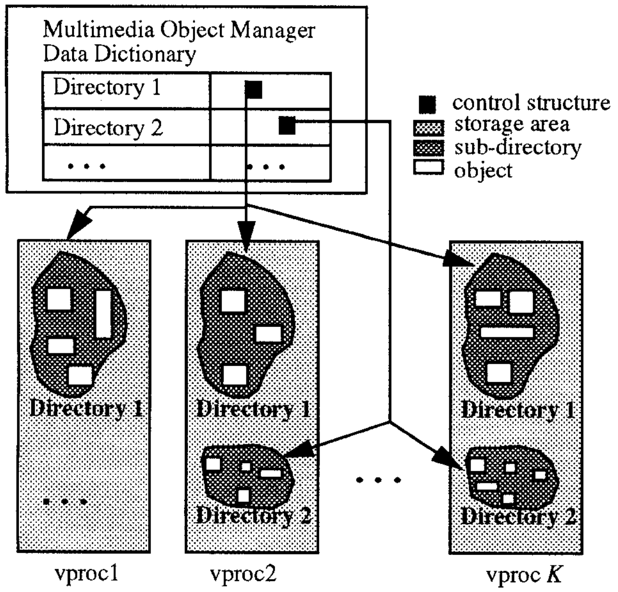

Figure 3 Directory Organization

While directories are distributed over multiple vprocs, each individual vproc manages one BeSS database, which consists of one or more physical storage areas. Directories spanning multiple vprocs have a directory entry in each of their vprocs' databases. Each storage area in a database is a character device raw disk partition. In this case, we bypass all buffering done by the operating system; the storage manager on each node encapsulates its own cache and buffer management facilities [8][11].

4. The maximum length in bytes is limited by physical storage.

5.1.4 Distribution Scope

Directories can be distributed over all or a subset of all vprocs. Each directory can select a system pre-defined distribution scope, which can be one, all, or a fraction of all vprocs in the system, for example: 1/2, 1/3, 1/5, 1/10, and so forth. Distribution scope allows flexibility in data distribution and helps balance the load on each vproc across the system. Objects that belong to the same directory are grouped together in each vproc. A Multimedia Object Manager directory is composed of its corresponding constituent directories in the vprocs. The data dictionary stores the directory organization information.

As illustrated in Figure 3, Directory 1 and Directory 2 have different distribution strategies.

5.1.5 Data Partitioning Strategies

Objects in the Multimedia Object Storage Engine are grouped into directories. Each directory specifies a physical set of vprocs to contain the objects in that directory. Objects in a directory share a Placement Policy, which specifies a function to map any object onto one vproc. In particular, each object is mapped first into an Object Key, then the Object Key is used to calculate a Placement Key, and finally a Placement Key is used to identify a vproc.

Users may specify their own functions for computing Object Keys. The type of Placement Policy used for a set of objects in a directory is specified by the creator of the table that uses those objects. Several Placement Policies are supported:

- Random. An object is placed on a randomly-selected vproc within the directory. No ObjectKey is required.
- Hashing. An ObjectKey is computed for the object. The ObjectKey is hashed using a system or user-defined hashing function to calculate a vproc.
- Value Partitioning. An ObjectKey is computed for the object. The space of ObjectKeys is partitioned into disjoint subsets. All ObjectKeys in a partition map to the same PlacementKey, which may be further mapped to one or a set of vprocs.

Not all requests are best served by the same placement strategy. Both Random and Hashing placement policies result in roughly uniform distributions of objects across vprocs. Both are advantageous for queries that access sets of objects. For example, a UDF to compare a target fingerprint against a set of fingerprints might benefit from random placement of the fingerprint set across all vprocs, to maximize parallelism.

Value Partitioning is useful for UDFs with non-uniform object accesses. In this case, remote object references can be avoided by ensuring that subsets of objects used together are co-located in the same vproc. Value Partitioning is well suited for range queries, where many parallel subqueries operate on subsets of the directory over different sub-ranges. These sub-ranges span a portion of the directory and can contain similar (or dissimilar) objects grouped near each other. Value Partitioning is also good for multi-dimensional spatial objects (also referred to as Spatial Partitioning), in which feature extracts are used as ObjectKeys. Similar objects, i.e., those that are "close" to one another in 1-D, 2-D, or in general N-dimensional space, can be physically placed in storage according to their physical spatial representations.

Both the directory identifier and the object's Placement Key are part of the logical OID. The directory number and the Placement Key are logical, not physical. The system computes vprocs from Placement Keys dynamically, which handles the case of reconfiguration of vprocs. The remainder of the logical OID is used by the node's object storage manager to locate the object within the vproc. This scheme allows objects to be relocated. When a system scales, the system migrates objects physically to keep the load balanced without affecting existing OIDs.

5.2 Object Storage Manager

BeSS is the Multimedia Object Manager's storage manager. Each node's storage manager operates in a peer-to-peer server relationship, while client processes (with respect to the storage manager) operate in a client-server mode [14][8].

Each storage manager manages data access for a single node; each node contains numerous vprocs. Each vproc on a node manages a single BeSS database, consisting of one or more storage areas. Each storage area is partitioned into fixed-sized extents. Allocation from an extent is based on the binary buddy system [15]. All control information (storage manager's metadata, such as control segments) is kept in a root (primary) area, while data segments can be dispersed throughout all areas in the database. Since control and data information have been separated, data segments can be moved freely within the database for balancing the load between the raw disk partitions. As a result of this organization, movement of data segments does not affect current object references and/or OIDs.

Figure 4 illustrates the logical storage representation on one vproc. Objects are shown distributed over multiple storage areas. Any one object may be striped over all or a subset of them. There is no software limit on the number of storage areas, just on the number of physical disks and the partitioning done on them.

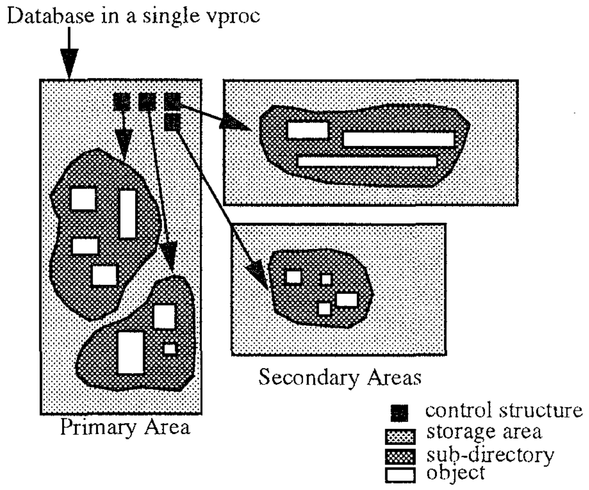

Figure 4 Logical Vproc Storage Representation

5.3 Object Access Methods

The Multimedia Object Manager is extensible. The Multimedia Object Manager software and trusted UDF code is linked directly to the BeSS process on each node, which allows the Multimedia Object Manager to access and manipulate regular objects, indices, and control structures directly on the segments for which they reside in the cache, without incurring in-memory copying costs. In addition, internal pointers within regular objects are swizzled when brought into the cache after page faults.

The Multimedia Object Manager allows two modes of persistent memory access:

- Shared memory access (direct access into the cache)
- Copy on access

In shared memory mode, processes have memory-speed access into cache segments based on memory mapping techniques; processes access and manipulate objects directly in the cache frames. The system prevents database corruption from bad pointers by separating data and control segments/structures. All control structures are protected by ordinary mechanisms provided by the virtual memory management hardware [8][16].

Since users may load their own UDFs, the system must treat these loadable software executables as untrusted software components, mainly due to data security and malicious corruption reasons. In this case, processes operate in copy on access mode. Only data (non-control) portions of objects are passed into a process's virtual address space. Object boundaries, not the data segment page boundaries, are copied to prevent malicious access to inappropriate data. In this mode, processes operate on objects in their own private buffer pool.

5.3.1 Transactional Semantics

The system provides the traditional ACID transaction properties (Atomicity, Concurrency, Isolation, and Durability). Concurrency control is provided through two-phase locking (2PL), while recovery is provided through logging [10]. Since moving computation (versus data) is not always possible, each storage manager supports remote object access. Thus, in the case of both locally and remotely accessed data, all pages and locks accessed by processes remain cached on the node where the transaction is executing. Cache consistency is guaranteed by using the callback algorithm [8][16]. The two-phase commit (2PC) protocol is employed for distributed commits, and time-outs are used for distributed deadlock detection.

5.4 Large Object Management

Stringent requirements are placed on the storage manager to effectively handle large multimedia objects [15][17]. First, objects are unlimited in size, limited by physical storage only. Second, large objects must support operations that deal with specific byte ranges within the object: for example, read or replace a random byte range within the object, insert or delete arbitrary byte ranges, and append bytes to the end of the object. The last three operations may cause the object to grow or shrink. Finally, objects must be protected from transaction and system failures.

Byte range operations require good random access (to locate arbitrary byte ranges) as well as good sequential access performance. Random access performance requires that the cost of locating byte ranges be independent of the object's size. Sequential access performance requires that reading/writing large chunks of objects must be close to the transfer rate [15].

The Multimedia Object Manager makes effective use of asynchronous I/O parallelism, data striping, and disk allocation for high-performance storage and retrieval [8]. Large objects spanning multiple pages can be accessed and updated as if they were a stream of bytes, even though internally they may not be contiguous and may possibly be striped over many disks. Efficient byte-range operations, such as insert, append, and truncate are provided, among others.

In addition, since the Multimedia Object Manager is managing all its disk allocation, the storage manager must store objects so that utilization of large objects is close to 100%. This requires that objects be stored with minimal internal fragmentation [15].

Each large object is stored in a set of variable-sized segments, each consisting of adjacent disk pages. As insertions, deletions, and appends are done, the segments may be grown or broken up. New segments may be added as the object grows. Each set of large object segments may contain segments that vary drastically in size. A B-Tree-like structure ties the segments together. Only the root index may not be relocated. The remaining index pages and leaf nodes may be moved among local storage areas to balance storage media. Finally, leaf nodes are striped across multiple storage areas on the node to increase parallelization due to asynchronous pre-fetching.

6 User-Defined Function Library

In the Multimedia Object Manager, applications can load, store, and execute UDFs. This allows developers to build a library of UDF building blocks for their use. Security permissions are given to each UDF to specify its accessibility. The permissions allow UDFs to be shared between all, a subset, or only one application. Permissions also allow sharing to be denied in the case of proprietary code.

The Multimedia Object Manager supplies a standard UDF library for basic algorithmic building blocks, such as compression/decompression and text manipulation. Additionally, a mechanism is provided for application clients to load their own UDF object code onto the Multimedia Object Manager.

7 Feature Extraction UDFs

If object attributes are defined on ADTs, then it is desirable to pre-generate feature extractions (through feature-extracting UDFs). A feature extract is a feature vector that represents an attribute of a the raw object, possibly in a spatial representation. Such vectors can be searched to locate similar objects. Sample feature extraction algorithms are face recognition, fingerprint matching, signature verification, and time-series analysis.

Comparison of two feature extracts requires a distance-computing UDF on feature vectors. A typical distance function is Euclidean distance.

To accomplish fast querying, objects are preprocessed (through UDFs). This step generates additional metadata on the objects, which must be updated when the raw objects are modified.

The Multimedia Object Manager will evolve to provide a generic framework for spatial indexing on feature vector keys [13]. This framework allows faster access to objects based on their attributes. Each ADT feature attribute may require its own index structure.

When an agent is accessing objects, it first checks to see if the attribute of interest in the ADT is indexed. If so, it accesses the index. Otherwise, the agent is cloned over the appropriate vprocs and each scans all feature extract objects.

8 Performance Evaluation

One of the main goals of the Multimedia Object Manager is the storage/retrieval of large multimedia objects in parallel with transactional semantics. To measure the overhead of the agent and storage management subsystems, the following performance experiment was performed using the Multimedia Object Manager executing on one vproc (Sparc10).[5] This experiment evaluates the Object Manager's ability to create, insert, and append to large objects when accessed by agents. These numbers represent a cold cache, no parts of any object's B-Tree-like index were initially in the cache.

The first graph (Figure 5) illustrates object creation times. Pieces of a large object are inserted sequentially to create a 10-Mbyte large object (versus inserting the whole object at once). Each piece is called a chunk. The second graph (Figure 6) illustrates the reading of a 10-Mbyte object in chunks.

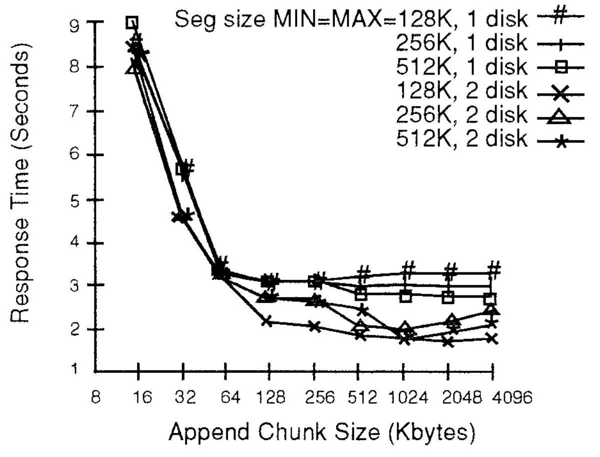

Figure 5 10-Mbyte Object Build Time

Since large objects can be striped over many disks, the performance numbers are taken using both one and two raw disks. The system can control the maximum and minimum sizes of the striped segments. In this example, the minimum/maximum sizes are made equal where the striped segments were 128K, 256K, and 512K, respectively. Typically, the striped segment size is a multiple of the disk drive block size.

The two raw disks are Seagate/Barracuda drives with external transfer rates of 10 Mbytes per second. Figure 5 illustrates build times. For one disk (no striping), appending chunks greater than 128K produces almost the same results as 128K chunks (approx. 3 seconds), with larger segment values producing slightly better results. For two disks (minimum striping), and appends greater than 256K, build time is approximately 2 seconds. These results are not conclusive; larger segments may slow down build time. The key issue is to build an object so read time is faster, not build time.

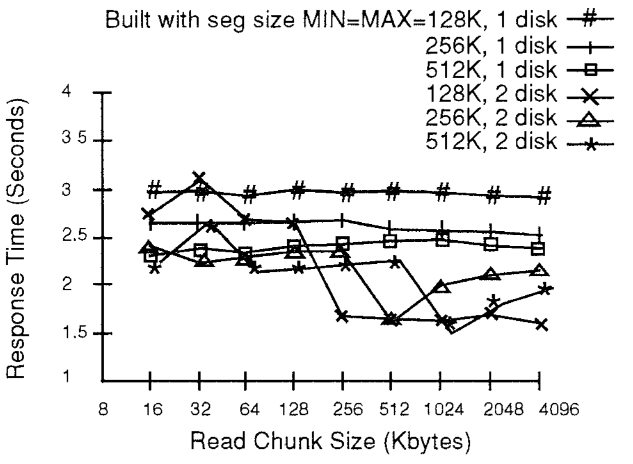

Figure 6 10-Mbyte Object Read Time

When reading objects, response times are fairly constant, with a significant drop when the read chunk is twice the storage chunk when striping over two disks. This is due to a fully balanced parallel read of two disks.

9 Terabyte Database Experience

NCR has years of experience in the retail, communications, and financial markets. Teradata customers are requesting multimedia extensions to their existing systems. Examples include fraud prevention (face/fingerprint/voice matching), market trend analysis (time series), and geographic information systems (maps). These examples go beyond the traditional support for the storage/retrieval of binary large-objects (BLOBs), and require the ability to support non-traditional content analysis of user-defined abstract data types.

Most commercial database vendors are augmenting their systems with object-oriented extensions that allow semantics to be defined on blobs [19][20][21][22][23].

5. A distributed SPARC SunOS environment is being used as a UDF development platform.

These extended relational systems are potentially capable of complex queries that analyze the content of objects, such as an image or audio stream. Additional database servers are also being introduced that specialize in selected ADTs, such as image matching [22] and geographic information systems [25].

NCR's experience in configuring and managing the largest commercial decision support database system, which has over eleven terabytes of on-line relational data, has given us insight into the need of high end applications with multimedia capabilities. As the MPP architecture scales, the notion of a logical vproc becomes increasingly important for load balancing and availability reasons.

The BYNET interconnect on the NCR System 5100M MPP machine supports data merging/sorting in hardware. In addition, the interconnect supports dynamic communication groups allowing efficient multicasts on the network.

Efficient step-execution plans over many MPP vprocs manage the parallelization of database queries. We are using the scatter/gather capabilities of the cloning agents to manage the steps. The flexibility of agent technology also allows agents to migrate across vprocs.

As Teradata customers migrate their systems over time to handle multimedia objects, we expect the following three data management scenarios:

- Large amounts of relational data, small amount of multimedia data
- Small amounts of relational data, large amounts of multimedia data
- Even distribution of relational and multimedia data.

The architecture handles these cases by providing flexible options for coded data and multimedia data placement across nodes of the system.

10 Conclusions

The Multimedia Object Manager system provides an extensible platform for users to define and manipulate UDFs on persistent multimedia objects in a general-purpose massively parallel environment. Users who understand the semantics of the data can load into the system the UDF modules for their selected applications, such as fingerprint matching, signature verification, face recognition, and speech recognition/translation. The Multimedia Object Manager is designed to meet the processing challenges of the emerging generation of multimedia content-based applications.

11 Acknowledgments

The authors gratefully acknowledge the Teradata SQL3 project team. Each team member participated greatly in the Multimedia Object Manager's design. We also would like to acknowledge the management support of our coaches, Alan Chow, Wayne Boyle, and Narain Gehani.

In addition, Marilyn Folster of AT&T Network Systems and George Thiruvathukal of R.R. Donnelly & Sons' MetroMail division provided valuable comments on the paper.

12 References

[1] M. Stonebraker, "The Case for Shared Nothing," *IEEE Data Engineering Bulletin*, Vol. 9, No. 2, pp. 4-9.

[2] W. O'Connell, D. Schrader, H. Chen, "*Teradata SQL3 Multimedia Database Server*," Technology For Multimedia, IEEE Press, Book Chapter, To appear late 1996.

[3] F. Carino, W. Sterling, I.T. Ieong, "Moonbase—A Complete Multimedia Database Solution," *Proc. of the ACM Multimedia Conf. Wkshp on Multimedia Database Mgmt Sys.*, San Francisco, CA, Oct. 1994.

[4] M. Olson, "Cover You Assets," Illustra Information Technologies, Inc., *SIGMOD*, Vol. 24, No. 2, San Jose, CA, 1995, p. 453.

[5] M. Geneserethm, S. Ketchpel, "Software Agents," *Comm. of ACM, Special Issue Intelligent Agents*, Vol. 37, No. 7, July 1994, pp. 48-53.

[6] G. Agha, "*Actors: A Model of Concurrent Computing in Distributed Systems*," MIT Press, 1986.

[7] G. Thiruvathukal, W. O'Connell, T. Christopher, "Toward Scalable Parallel Software: Interfacing to Non-von Neumann Programming Environments," *Proc. of the SIAM '95*, San Francisco, CA, Feb. 1995.

[8] A. Biliris, E. Panagos, "A High-Performance Configurable Storage Manager," *Proc. of the IEEE 9th Int'l Conf. on Data Engineering*, Taipei, Taiwan, March 1995, pp. 35 - 43.

[9] F. Carino, W. Sterling, P. Kostamaa, "*Industrial Database Supercomputer Exegesis--DBC/1012, the NCR 3700, The Ynet and the BYNET*," Emerging Trends in Knowledge and Database Systems, IEEE Adv. Computer Science Book Chapter, 1994.

[10] J. Gray, A. Reuter, "*Transaction Processing: Concepts and Techniques*," Morgan Kaufmann Publishers, Inc., 1993.

[11] N. Gehani, H. V. Jagadish, W. Roome, "OdeFS: A File System Interface to an Object-Oriented Database," *Proc. of the 20th VLDB Conf.*, Santiago, Chile, 1994.

[12] S. Ghandehardizadeh, D. DeWitt, "Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines," *Proc. of the 16th VLDB*, Brisbane, Australia, 1990, pp. 481-492.

[13] C. Faloutsos, N. Koudas, I. Kamel, "Declustering Spatial Databases on a Multi-Computer Architecture,"

Technical Report TM 950720-07, AT&T Bell Laboratories, Sept. 1995; *to appear in the Proc. Extending Database Tech.*, France, March 1996.

[14] A. Biliris, W. O'Connell, E. Panagos, "BeSS Reference Guide," Release 0.8, Technical Report, AT&T Bell Laboratories, Sept. 1995.

[15] A. Biliris, "An Efficient Database Storage Structure for Large Dynamic Objects," *Proc. of the IEEE 8th International Conference on Data Engineering*, Phoenix, AZ, Feb. 1992, pp. 301-308.

[16] C. Lamb, G. Landis, J. Orenstein, D. Weinreb, "The ObjectStore Database System," *Comm. of the ACM*, 34(10):pp. 51-63, Oct. 1991.

[17] A. Biliris, E. Panagos, "EOS: An Extensible Object Store," *Proc. of ACM-SIGMOD 1994 Int'l Conference on Management of Data,"* Minneapolis, MN, May 1994, p 517.

[18] W. Sterling, F. Carino, C. Boss, "Multimedia Databases and Servers," *AT&T Technical Journal*, Sept./Oct. 1995, pp. 54-67.

[19] W. Kim, "UniSQL/X Unified Relational and Object-Oriented Database System," *Proc. of the ACM SIGMOD*, Vol. 23, No. 2, p 481, June 1994.

[20] M. Ubell, "The Montage Extensible Datablade Architecture," *Proc. of the ACM SIGMOD*, Vol. 23, No. 2, p 482, June 1994.

[21] H. Pirahesh, "Object-Oriented Features of DB2 Client/Server," *Proc. of the ACM SIGMOD*, Vol. 23, No. 2, p 483, June 1994.

[22] Oracle Corp., "Oracle Media Server--Data Sheet," in http://www.oracle.com/info/products/newMedia/oms.html

[23] Sybase, Inc., "Command Reference Manual," Release 4.9, 1992.

[24] Virage Technology, Inc., "Visual Information Retrieval White Paper," available in http://www.virage.com/.

[25] D. Dewitt, N. Kabro, J. Luo, J. Patel, J.Yu, "Client-Server Paradise," *Proc. of the 20th VLDB Conference*, Santiago, Chile, Sept. 1994.