Object-Based Storage Model for Object-Oriented Database

Zhongmin Li[1], Zhanwu Yu[1][1]

[1]State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan, Hubei, China, 430079
{zhongmli, yzw2008}@gmail.com

Abstract. The current storage models for Object-Oriented DataBase (OODB), which organize data as objects according to the Object-Oriented Data Model (OODM), are mainly established on the block storage devices. In this way, the storage manager does not have detailed knowledge of the characteristics of the underlying storage devices, and the storage subsystem does not have the semantic knowledge of the data stored in the block storage devices, so it is very difficult to implement some workload-dependent tasks such as data layout and caching. Furthermore, the storage subsystem of OODB has to organize the objects in the pages, which is not well-suited to the objects storage. In this paper, we present an Object-Based Storage Model (OBSM) for OODB by using the recently-standardized Object-based Storage Device (OSD) interface. OSD offloads the storage management into the storage device itself and provides an object interface to data, which brings more intelligence into the storage subsystem. In the first glance at using OBSD in OODB, we explore the methods to map OODM to OBSM including direct mapping, mapping clustering to collection and declustering a large object to a series of sub-objects, and analyze the benefits to the storage subsystem of OODB by using OBSM such as providing storage functionalities offloading, providing objects sharing pool, providing integrative object persistence.

Keywords: object-oriented database, object-based storage, storage model, object-oriented data model

1 Introduction

The storage subsystem is one of the kernel modules in database system, and its storage performance is of utmost importance for database applications. The standard interfaces to storage subsystem virtualize storage as a simple linear array of fixed-size logical blocks.

[1] Supported by the National Key Basic Research and Development Program of China (No.2004CB318206).

Corresponding author: Zhanwu Yu, Professor, Ph.D. Supervisor. Major Research Interests: Multimedia Communication, Massive Information Storage. yzw2008@gmail.com

Most database systems use the N-ary Storage Model (NSM) as their low-level data layout. In the NSM, all attributes of a conceptual schema record are stored together. NSM organizes he table into fixed-size pages (e.g., 8KB) each containing a short range of full records, stores records contiguously starting from the beginning of each disk page, and uses an offset table at the end of the page to locate the beginning of each record, and the pages are stored sequentially on disk [1]. NSM is well-suited to workloads that access full records, which are always fetched into memory regardless of whether the query actually touches the data. In this way, it wastes memory capacity and, more importantly, disk bandwidth, so it is not well-suited to workloads that access partial records.

The Decomposition Storage Model (DSM) organizes a table in column major order by storing individual fields sequentially on the disk [2]. DSM gives good performance when accessing just one or a few fields from a relation, but suffers when reconstructing full records. In order to solve the problem, Ramamurthy et al suggested DSM stores two copies of each attribute relation: one copy is clustered on the value while the other is clustered on the surrogate [3]. To support the relational model, intermediate and final results need an N-ary representation. However, this technique needs double storage space and requires updating both copies of the relation.

To address the issue of low cache utilization in NSM, Ailamaki et al. introduce Partition Attributes Across (PAX), a new layout for data records [4]. Unlike NSM, within each page, PAX groups all the values of a particular attribute together on a minipage, and fully utilizes the cache resources during a sequential data access because only a number of the required attribute's values are loaded into the cache. However, compared with DSM, PAX doesn't optimize the I/O between disk and memory.

The Multi-resolution Block Storage Model (MBSM) stores records in a partitioned way in super-blocks, and then organizes super-blocks on disk into mega-blocks [5]. MBSM is most suitable for decision-support workloads that frequently execute table scans. Compared to NSM and PAX, MBSM requests fewer disk pages. Compared to DSM, MBSM's scan performance is comparable, and its cache performance is better. Furthermore, MBSM has better insert/update I/O performance, and doesn't require a join to reconstruct the records.

The Clotho Storage Model (CSM) allows the DBMS to construct in-memory data pages that contain only the data required for a given query [6]. CSM provides the desired tradeoff between full- and partial-record access to save memory and disk bandwidth. The Atropos disk array logical volume manager enables efficient two-dimensional access to relations stored on disk, allowing data pages to be filled efficiently, regardless of the query [7]. The combination of Clotho and Atropos uses knowledge of the relation's schema and the characteristics of the underlying disk storage to enable a geometry-aware storage model (GASM) with the desired tradeoff: the performance of NSM for full-record access, the performance of DSM for single-record access, and a near-linear tradeoff for partial-record access. Many studies [7, 8, 9, 10] have taken the view that a storage device can provide relevant characteristics to applications to allow for optimized I/O access. But the required information about detailed knowledge of the mechanical parameters and geometry of the disk drives in the storage subsystem can be measured empirically, so it is difficult and fragile in practice, making the realization of GASM problematic.

Schlosser uses Object-Based Storage (OBS) interfaces to allow the database storage manager to cleanly communicate its storage requirements to the storage subsystem, where more information is available to make low-level optimizations [11]. In this way, an entire table is stored as a single object, and the schema of the relation is expressed through attributes assigned to that object, so storage managers can take advantage of heterogeneous storage devices without being burdened with different device parameters. Using the Object-based Storage Devices (OSD) specification [12], data can be addressed at an object granularity, allowing the storage manager to access individual records and fields of the relation directly. Furthermore, object-based storage devices have much richer semantic information about the data that they store, allowing them to optimize their performance "under the covers" much more effectively than current storage devices. The database no longer needs detailed information about the storage subsystem, but still can take advantage of geometry-aware data placement techniques such as CSM.

Schlosser only discussed how to organize relational database tables into objects using OSD, which are mainly used in Relational Database (RDBM) and Object-Relational Database (ORDBM). Sometimes applications evolve to object-oriented software development to keep the promise of flexibility and extensibility. OODB is well-suited, since it provides homogeneous means to store and to manage complex structures very efficiently. As for Object-Oriented Database (OODB), it usually uses Object-Oriented Data Model (OODM) to store the hierarchical classes in the pages based on the block storage devices such as disks [1]. In this way, the storage subsystem needs manage two buffers: pages buffer and objects buffer, and frequently change the storage mode between them. Its storage performance is low, and it is difficult to realize multilevel persistent objects storage.

In this paper, we present an Object-Based Storage Model (OBSM) for OODB using OSD. We store the hierarchical classes as objects in the OSD devices by mapping OODM to OBSM. The rest of the paper is organized as follows: Section 2 briefly introduces the OSD model and its advantages. Section 3 summarizes the OODM and the methods to map OODM to OBSM. Section 4 analyzes the benefits to OODB by using OBSM. Section 5 gives a summary for the paper and future work.

2 Object-based Storage

2.1 OSD model

An object is variable-length and can be used to store any type of data, such as files, database records, medical images, or multimedia, even be used to store an entire file system or database [13]. The OSD object abstraction is designed to re-divide the responsibility for managing the access to data on a storage device by assigning to the storage device additional responsibilities in the area of space management [12]. Figure 1 shows the relationship between the OSD model and a traditional SBC-based model for a file system.

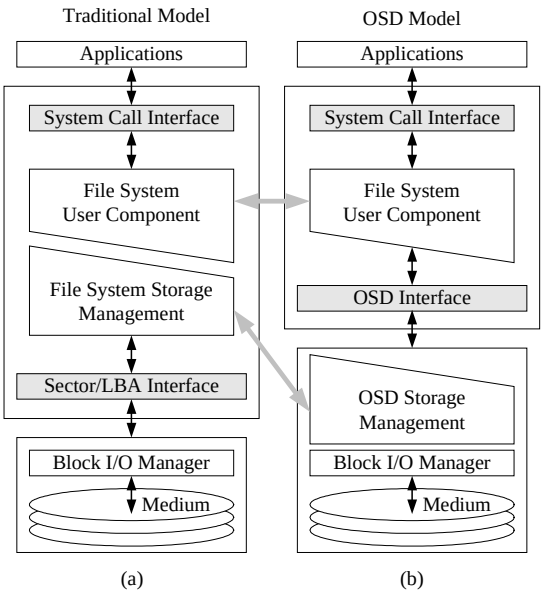

Fig. 1. Comparison of traditional and OSD storage models: (a) traditional storage model; (b) OSD storage model.

The user component of the file system contains such functions as hierarchy management, naming and user access control. The storage management component is focused on mapping logical constructs (e.g., files or database entries) to the physical organization of the storage media.

In the OSD model, the logical constructs are called user objects, which are composed of data, user-accessible attributes, and device-managed metadata, and act as data containers that abstract the physical disk layout details under the object interface. A user object is the basic object type, and the root object, partition objects and collection objects provide additional navigational aids for user objects. Each object, regardless of its type, has a unique name or Object ID (OID), a set of object attributes, and some meta-data. There is only a root object on each Object-Based Storage Device (OBSD), which encompasses all the other objects. Each user object is a member of only one partition object which allows for efficient addressing, capacity and quota management, and security management of sets of user objects. Within each partition there may be zero or more collection objects that are used to group user objects for fast indexing and multi-object operations. A user object can be a member of zero or more collection objects at the same time.

In addition to mapping data, the storage management component maintains other information about the OSD objects that it stores (e.g., size, usage quotas, and associated username) in attributes. The user component may have the ability to influence the properties of object data through the specification of attributes by accessing the object interface. The difference between an OBSD and a block-based device is the interface, not the physical media [13]. The OBSD makes the decisions as to where to allocate storage capacity for individual data entities and managing free space.

2.2 OSD advantages

Fig. 2 shows the architecture of the OSD. In order to separate access paths of control, management and data, the Client, the Metadata Server (MDS) and the OBSD are self-existent. The Client is the initiator of data access, the MDS manage the metadata of whole storage system and the OBSD are the storage devices to store objects [12]. In comparison with traditional storage architecture, the Object-based Storage System (OSS) possess some features such as intelligence of storage devices, distributed metadata, parallel data access, data sharing across platforms and security of data access [13,14].

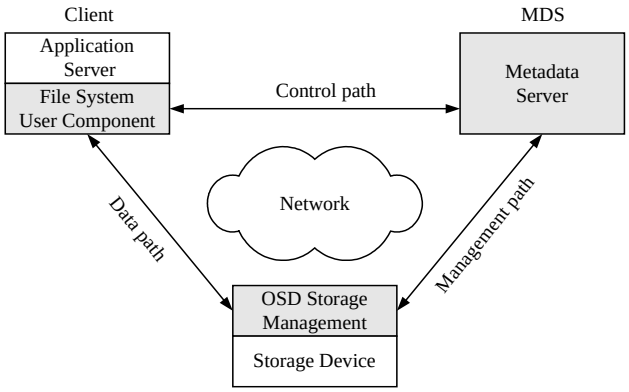

Fig. 2. OSD architecture

In the OSD model, the OBSD can understand some of the relationships between the blocks on the device, and can use this information to better organize the data and anticipate the storage requirements of whole system, because the object storage management component stands in the OBSD. The attributes of an object can contain static information about the object (e.g., creation time), dynamic information updated on each access (e.g., last access time), information specific to a storage application (e.g., filename, group, or user ids), and information specific to a current application. Attributes may also contain hints about the object's behaviors such as the expected read/write ratio, the most likely patterns of access (e.g., sequential or random), or the expected lifetime of the object [12, 13]. To access such various attributes enables the storage system to better store, organize and manage the data.

While separating the MDS, the Client can directly access the OBSDs, and 90% of the metadata management is offloaded from the metadata server to the OBSDs. In this way, it solves the bottleneck problem which results from using traditional network storage architectures, which are designed with a single monolithic metadata server. Each OBSD manages the layout and retrieval of the data that is presented to it. This brings improvement in an order of magnitude in the potential performance of the system's metadata management. Furthermore, adding more OBSDs to the system will not only increase the capacity of whole object-based storage system, but also increase

the resources of the metadata management. This also improves scalability of clusters greatly since hosts no longer need to coordinate metadata updates.

Direct accessing the OBSD enables a high-performance solution, as there are no potential bottlenecks in the system between the hosts and the storage devices. Owing to intelligence built in the OBSD, there is no need for a file server to intermediate the transaction. Further, if the file system stripes the data across a number of OBSDs, the aggregate I/O rates and data throughput rates scale linearly.

Objects introduce a mechanism in the OBSD that allows the device treat storage applications or clients individually. With objects, since metadata is offloaded to the OBSD, the dependency between the metadata and storage system/application is removed. Attributes improve data sharing by allowing storage applications to share a common set of information describing the data, and are the key to giving storage devices an awareness of how objects are being accessed. In this way, OSD removes the biggest obstacle to data sharing.

Security is perhaps the single most important feature of object-based storage that distinguishes it from block-based storage. Although security does exist at the device and fabric level for SANs, objects provide a finer granular security at a much lower cost in the OSD model [13]. In order to realize secure data sharing, a credential-based access control system is running in the OSD model, which is very different to realize in the traditional network storage architectures. In this security architecture, every access is authorized, and the authorization is done without communicating with a central authority that may slow the data path. The security manager may authenticate the Client, but the OBSD does not authenticate the Client. It is sufficient for the OBSD to verify the credential sent by the Client. The credential-based access control system may ensure the Client to use more effective and more achievable network such as Ethernet.

3 OBSM for OODB

The common data model supported by ODMG [15] implementations is based on the OMG Object Model, which was designed to be a common denominator for object request brokers, object database systems, object programming languages, and other applications. Generally, OODM includes interface, class, and structure (literal) data types. An interface specifies the abstract behavior of its instances; a class specifies the abstract state and behavior of its instances; a structure type is defined with a name and a set of attributes, and will be called a named structure type [16]. An instance of a named structure type has no identifier but values of attributes. The ODMG Object Model defines atomic literal types such as short and string.

Logically, OODB consists of a class hierarchy represented by a Directed Acyclic Graph (DAG). Each node in the DAG represents a class, and classes are named and associated with each class that is a set of instance variables. The OODB consists of instances of these classes, and every instance in the OODB belongs to its home class. Since the hierarchy represents an ISA relationship among the objects, an instance belongs not only to its home class but to all the superclasses related to it, and has a unique identifier to distinguish it from every other instance in the OODB.

While using OSD to store the objects in OODB, the primary problem is how to map OODM to OBSM. There are several methods to map OODM to OBSM: (1) direct mapping a Logical Object (LO) based on OODM to a Storage Object (SO) in OBSD, (2) mapping LOs clustering to SOs collection, (3) declustering a large LO to a series of sub-objects (several SOs).

3.1 Direct mapping

Primarily we map a LO to a SO in the object-based storage system. No need to change the structure of LO, just add a unique SO identifier (SOID), the length of LO and some storage attributes for the LO to construct a correlative SO (see Fig. 3). We also can add some other attributes to the end of SO. It is the general way to map a LO to a SO.

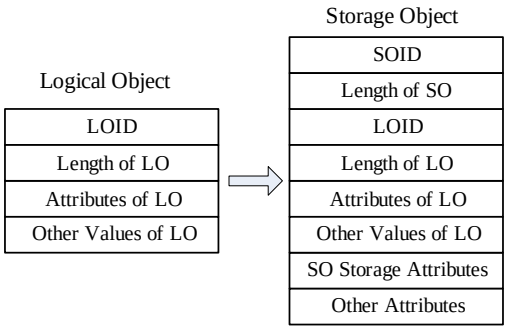

Fig. 3. Direct mapping

3.2 Mapping clustering to collection

Clustering is a container for the objects which attributes are similar such as all the instances of some class, and is the ability to store logically related objects close together. When applications access an object, applications can find the object or other correlative objects using clustering. Using OBSM, we can map a clustering to a Collection Object (CO) stored in the OBSD. In this way, each LO in the clustering is mapped to a relevant SO using the method described in 3.1, and the CO has its own unique object identifier and attributes (see Fig. 4). The SOs belong to the CO logically, but not the part of it. The common attributes of all the SOs can be drew out to become the part of the attributes of the CO. Each SO also has its own unique object identifier and attributes. Applications can access the CO, and also directly access any SO in the CO respectively.

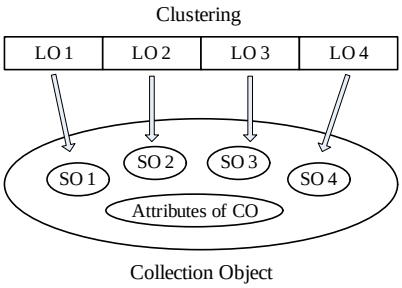

Fig. 4. Mapping clustering to collection

3.3 Declustering a large object to a series of sub-objects

In the block storage device, a large logical object which size is bigger than one page must be stored in several pages using some strategy. In the OSD, a large logical object can be mapped to a SO using the method in 3.1. When the size of a large object is too big to affect the access efficiency mapping a SO stored in an OBSD, we can split the large logical object to several slices, and each slice is mapped to a relevant SO (see Fig. 5). In this way, we can use some strategies to distribute the series of sub-objects in different OBSDs to increase the access I/O bandwidth using the characteristic of parallel access of OSD. The procedure is hided in the object storage system, and the applications don't need to know the processing details. For applications, accessing a large object and accessing an average object are in the same way.

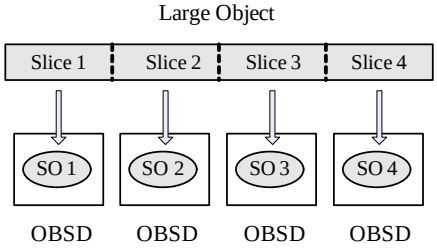

Fig. 5. Declustering a large object to a series of sub-objects

4 Benefits

In the OSD, the storage devices have the intelligence according to offloading the storage management component into the storage devices, which brings many benefits to the object management in the OODB.

4.1 Providing storage functionalities offloading

Using OBSM, the storage management component of the storage subsystem is offloaded into the object-based storage devices. In this way, some functionalities of storage management can be offloaded into the object-based storage system, and the storage manager of OODB focuses on the transaction processing.

4.2 Providing objects sharing pool

The object-based storage system can provide an objects sharing pool for OODB, and provide access control at object granularity. Synchronization of metadata is realized in the object-based storage system, so various OODBs can share the objects sharing pool to construct a parallel distributed Object-Oriented DataBase System (OODBS). The metadata server of the object-based storage system can provide a uniform DAG for all the OODBs.

4.3 Providing integrative object persistence

The storage subsystem of OODB can use unified object interface to access the objects storage sharing pool without format conversion, and the uniform DAG provides efficiency navigation. Furthermore, persistence objects and temporary objects have the same object storage format, which provides an integrative management for objects in OODB.

5 Conclusion

In this paper, we introduce a new storage model called OBSM for OODM. We have explored the methods to store the objects of OODB by using object-based storage devices to improve the interaction between the database storage manager and the storage subsystem. OBSM brings some benefits to OODB such as providing storage functionalities offloading, providing objects sharing pool, providing integrative object persistence. OBSM provides an integrative management for objects in OODB.

OSD has much richer semantic information about the data that they store, allowing them to optimize their performance in the underlying storage system, so it is much more effectively than current storage devices. OSD already includes provision for free space management, which is currently handled by the database storage manager. With better semantic information about the free space requirements of the database, the storage subsystem could better optimize its layout and set aside free space for future inserts and deletes. Our future work focuses on investigating these opportunities for OODB applications.

Acknowledgments. This paper is supported by the National Key Basic Research and Development Program of China (No.2004CB318206). Every member in our project

team has made contribution to this project. Especially thanks Dr. Sheng Zheng for his helpful advice.

References

1. R. Ramakrishnan, J. Gehrke: Database management systems. Number 3rd edition. McGraw-Hill (2003)
2. G. P. Copeland, S. Khoshafian: A decomposition storage model. In ACM SIGMOD International Conference on Management of Data, ACM Press (1985) 268–279.
3. R. Ramamurthy, D. J. DeWitt, Q. Su: A case for fractured mirrors. In International Conference on Very Large Databases, Morgan Kaufmann Publishers, Inc. (2002) 430–441
4. A. Ailamaki, D. J. DeWitt, M. D. Hill, M. Skounakis: Weaving relations for cache performance. In Proceedings of VLDB Conference (2001)
5. J. Zhou, K. A. Ross: A multi-resolution block storage model for database design. Proceedings of the Seventh International Database Engineering and Applications Symposium (IDEAS'03), Asunción, Paraguay (2003)
6. J. Schindler, S. W. Schlosser, M. Shao, A. Ailamaki, G. R. Ganger: Atropos: A disk array volume manager for orchestrated use of disks. In Conference on File and Storage Technologies. USENIX Association (2004)
7. M. Shao, J. Schindler, S. W. Schlosser, A. Ailamaki, G. R. Ganger: Clotho: Decoupling memory page layout from storage organization. In International Conference on Very Large Databases (2004) 696–707
8. J. Schindler, A. Ailamaki, G. R. Ganger: Lachesis: Robust database storage management based on device-specific performance characteristics. In International Conference on Very Large Databases, Morgan Kaufmann Publishing, Inc. (2003) 706–717
9. J. Schindler, J. L. Griffin, C. R. Lumb, and G. R. Ganger. Track-aligned extents: Matching access patterns to disk drive characteristics. In Conference on File and Storage Technologies, USENIX Association (2002) 259–274
10. R. VanMeter: SLEDs: Storage latency estimation descriptors. In IEEE Symposium on Mass Storage Systems. USENIX (1998)
11. Steven W. Schlosser, Sami Iren: Database storage management with object-based storage devices. Proceedings of the First International Workshop on Data Management on New Hardware (DaMoN 2005), Baltimore, MD, USA (2005)
12. SCSI Object-Based Storage Device Commands-2 (OSD-2), http://www.t10.org. October (2004)
13. Mike Mesnier, Gregory R. Ganger, Erik Riedel: Object-based Storage. IEEE Communications Magazine, Vol. 41(8) (2003) 84–90
14. Azagury A, Dreizin V, Factor M: Towards an Object Store. 20th IEEE Symposium on Mass Storage Systems (2003)
15. R. Cattel, D. Barry, M. Berler, J. Eastman, D. Jordan, C. Russel, O. Schadow, T. Stanienda, F. Velez: The Object Data Standard: ODMG 3.0. Morgan Kaufgrnann, Publishers, Inc. (1999)
16. Liwu Li: On ODMG Data Types. 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS 39) (2001) 219–228