DBFS and SecureFiles

Krishna Kunchithapadam, Wei Zhang, Amit Ganesh, Niloy Mukherjee

Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065

{Krishna.Kunchithapadam, Wei.Zhang, Amit.Ganesh, Niloy.Mukherjee}@Oracle.com

ABSTRACT

Modern enterprise, web, and multimedia applications are generating unstructured content at unforeseen volumes in the form of documents, texts, and media files. Such content is generally associated with relational data such as user names, location tags, and timestamps. Storage of unstructured content in a relational database would guarantee the same robustness, transactional consistency, data integrity, data recoverability and other data management features consolidated across files and relational contents. Although database systems are preferred for relational data management, poor performance of unstructured data storage, limited data transformation functionalities, and lack of interfaces based on filesystem standards may keep more than eighty five percent of non-relational unstructured content out of databases in the coming decades.

We introduce Oracle Database Filesystem (DBFS) as a consolidated solution that unifies state-of-the-art network filesystem features with relational database management ones. DBFS is a novel shared-storage network filesystem developed in the RDBMS kernel that allows content management applications to transparently store and organize files using standard filesystem interfaces, in the same database that stores associated relational content. The server component of DBFS is based on Oracle SecureFiles, a novel unstructured data storage engine within the RDBMS that provides filesystem like or better storage performance for files within the database while fully leveraging relational data management features such as transaction atomicity, isolation, read consistency, temporality, and information lifecycle management.

We present a preliminary performance evaluation of DBFS that demonstrates more than 10TB/hr throughput of filesystem read and write operations consistently over a period of 12 hours on an Oracle Exadata Database cluster of four server nodes. In terms of file storage, such extreme performance is equivalent to ingestion of more than 2500 million 100KB document files a single day. The set of initial results look very promising for DBFS towards becoming the universal storage solution for both relational and unstructured content.

1. INTRODUCTION

Content volumes are growing rapidly in both enterprise and consumer spaces as processors, storage devices, and physical hardware are growing in scale. According to an independent study [1], more than 500 exabytes of enterprise content had been ingested across all computer systems in 2008 alone. Analyst estimates demonstrate that more than eighty five percent of such content is unstructured in nature, which is accompanied by fifteen percent of relational content [2]. Besides enterprise applications, consumer multimedia services, higher availability of Internet access in emerging countries, and social networks are steadily contributing to the digital deluge. In 2009, more than 200,000 videos were uploaded per day using YouTube application [3]. More recent statistics from Facebook reveal that more than 60 million status updates are posted in a day, and more than 3 billion photographs are uploaded per month [4]. Estimates predict more than 20 quadrillion unstructured data objects will be created in the year 2011 alone [2].

Although database systems are equipped with more advanced and secure data management features such as transactional atomicity, consistency, durability, manageability, and availability, lack of high performance and throughput scalability for storage of unstructured objects, and absence of standard filesystem-based application program interfaces have been cited as primary reasons for content management providers to often prefer existing filesystems or devise filesystem-like solutions for unstructured objects [5][6].

Over the last two decades, several database researchers have envisioned an architectural unification of databases and filesystems. In 1996, Dr. David DeWitt had presented his vision on the confluence of Objects and Databases allowing large enterprises reaping the benefits of families of products that offer integrated solutions functioning scalably and robustly by the end of 2006 [7]. In his presentation at FAST 2005, Dr. Jim Gray mentioned that “Filesystem should borrow ideas from DB” [6].

We introduce Oracle Database Filesystem (DBFS) as a consolidated solution achieving the much-anticipated architectural unification through a cross-pollination of ideas from filesystem research to relational databases. DBFS is a pioneering shared-storage network filesystem client-server architecture built on Oracle SecureFiles [8][9], the high-performance unstructured data storage architecture within the Oracle RDBMS [10]. SecureFiles was primarily designed to provide filesystem-like or better storage throughput across all file sizes and types that scales with the scale of content-generating applications as well as with underlying hardware and storage systems. Besides performance and scalability aspects, several filesystem-like data transformation capabilities such as de-duplication, compression and encryption have been incorporated in SecureFiles to provide maximal data storage utilization and security. DBFS provides a client-server filesystem abstraction over SecureFiles allowing content management developers to perform typical network filesystem operations within the RDBMS using standard filesystem interfaces besides structured data management using standard database interfaces. Similar to traditional network filesystems,

Oracle Database Filesystem provides a transparent abstraction of a shared network filesystem as a local filesystem to end-user applications. Storage of unstructured data within the Oracle RDBMS extends the rich set of transactional, consistency, durability and temporal data management features to existing filesystem-based tools and applications.

The remainder of the paper is organized as follows. An overview of DBFS architecture is provided in section 2. Sections 3 to 7 detail the individual components of the DBFS architecture. Section 8 presents a preliminary performance evaluation of DBFS on filesystem storage and access operations using POSIX-standard filesystem commands. The paper is concluded in section 9.

2. ORACLE DBFS ARCHITECTURE

The architecture of Oracle DBFS comprises of filesystem client and server components, similar to traditional NFS [11]. Figure 1 demonstrates the client-server architecture of Oracle DBFS.

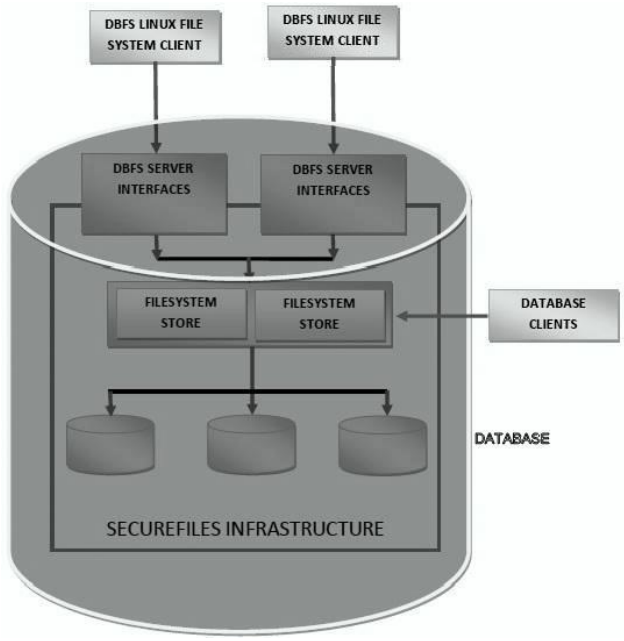

Figure 1. Oracle DBFS: Shared-Storage Filesystem within the Database

The RDBMS server [13] is the filesystem server for DBFS. The server consists of one or more filesystem stores that are accessed by a set of interfaces called DBFS Content API. A filesystem store is characterized by one or more database objects such as tables, table partitions, indexes, etc. Besides relational columns, these database objects consist of columns dedicated to filesystem metadata attributes and LOB datatypes. Oracle RDBMS allows these database objects to share storage in a multi-node distributed environment [12] thereby providing shared-storage filesystem capabilities to DBFS. File metadata operations, such as creation and listing of directories, results in modifications of tuples in these database objects. The SecureFiles architecture provides support for storage and access of file data as LOB datatypes in database storage devices using highly optimized algorithms that scale performance up on single multi-core processor systems as well as scale out on distributed systems.

Besides providing high performance, SecureFiles provides advanced file data transformation capabilities that include filesystem compression to optimize utilization of cache and storage, automated de-duplication of files to prevent redundant file storage, and Transparent Data Encryption (TDE) semantics to both relational and file data. In addition to these advanced filesystem features, Oracle SecureFiles infrastructure was designed to provide several RDBMS capabilities, such as atomicity, consistency, isolation and durability semantics on unstructured data management operations, along with more advanced database features, such as consistent backup, point in time recovery, XML indexing, XML queries, temporal management and query ability of unstructured and relational data through complete historization of data [15]. DBFS filesystem stores inherit all the capabilities provided by SecureFiles storage infrastructure. The rich set of data transformation and management options allows applications to create filesystem stores with different combinations of such options.

The DBFS Content API provides PL/SQL interfaces that correspond to the complete set of POSIX filesystem access primitives such as create file, open, read, write, create directory, list directory, change directory, etc. Each filesystem store is characterized by application-specific implementations equivalent to these primitives within the DBFS Content API interfaces. A server-specific mount-point is associated with each filesystem store. Operations on files with pathnames relative to a server-specific mount-point are performed using the functionalities implemented for the corresponding filesystem store on database objects characterizing the store.

The DBFS client component utilizes Filesystem in User Space (FUSE) [14] kernel module that exposes filesystem calls from the OS kernel as function callbacks in user space. The client component transforms the function callbacks to the equivalent PL/SQL interfaces provided by Content API and places the calls to the RDBMS server over OCI or Oracle Call Interface connections. DBFS filesystem is mounted on the client machine with a client-specific mount-point. POSIX-based filesystem commands that are relative to the client-specific mount-point are converted to Content API functions. Based on the server-specific mount-point specified in the Content API interfaces, the target filesystem store is identified. The Content API therefore provides Linux VFS-like capabilities of mounting of multiple filesystems in a single database server. Besides a filesystem client , DBFS allows access of relational, filesystem metadata and file data directly through database clients such as PL/SQL, JDBC and OCI.

To summarize, the DBFS client server architecture provides the complete set of interfaces that transform filesystem calls from the client to database calls to RDBMS server. These calls are targeted to individual filesystem stores that employ store-specific interfaces to perform operations on database objects associated with them. Filesystem operations that involve storage and access of files are managed through the Oracle SecureFiles storage architecture. Each of the components will be discussed in detail in the subsequent sections.

3. SECUREFILES

SecureFiles [8][9] was introduced in 2007 as a high performance and scalable storage architecture for unstructured data in the database, breaking the performance barrier of unstructured data management in a database. File data manipulation and retrieval operations in Oracle DBFS filesystem are handled by the SecureFiles infrastructure. Figure 2 demonstrates the architecture of SecureFiles. The major components can be categorized based on their contributions in providing filesystem-like throughput and scalability, maximizing storage space utilization, and providing secure data management. The following subsection enumerates the major components of SecureFiles architecture.

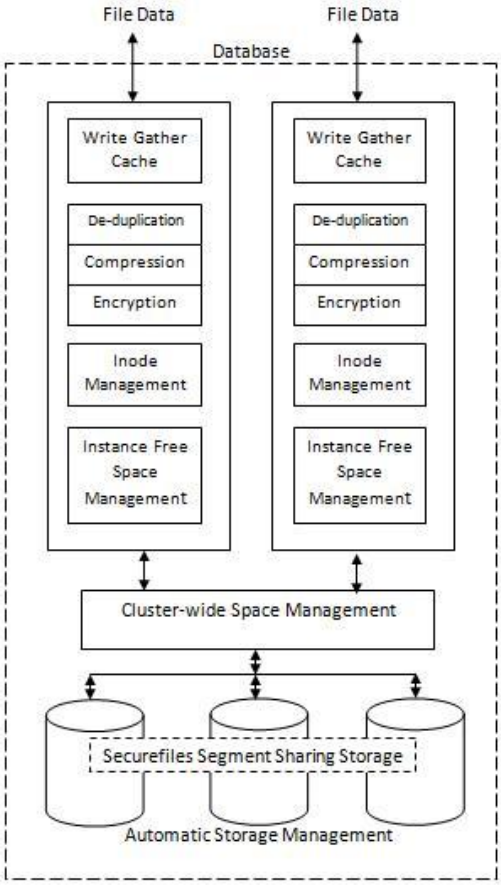

Figure 2. SecureFiles Architecture.

3.1 Performance and Scalability

The write gather cache, inode management, database space management and automatic storage management components are responsible for ensuring the scalability of file manipulation and retrieval throughput performance.

3.1.1 Write Gather Cache

The Write Gather Cache (WGC) is a subset of the database cache that can buffer large amounts of file data during write operations. The writes are checkpointed to the underlying storage system during file close operations and buffer overflows. This buffering of in-flight data allows for large contiguous storage space allocation leading to large contiguous storage I/O operations with reduced disk seek costs.

3.1.2 Inode Management

The inode management layer is responsible for initiating on-disk storage and access operations on files. During file checkpointing following write operations, the inode manager requests free space from underlying storage systems to initiate asynchronous I/O operations. Filesystem-like inode data structures are created and managed for individual DBFS files to maintain the array of contiguous physical offsets and lengths. This prevents single points of contention in concurrent environments during update, delete and append operations on files. Metadata maintained in the inode can remain extremely compact because the space management layer provides the support to return a set of variable sized chunks to store the data being written to disk. The metadata management structures can therefore scale to map terabyte-sized objects very efficiently. SecureFiles inode management layer contributes in further scale-up of read operations through an intelligent pre-fetching mechanism. The layer keeps track of access patterns and intelligently pre-fetches data before the request is actually made. Read latency is reduced by the overlap of network roundtrip with the disk I/O thereby scaling up read throughputs to greater extents.

3.1.3 Free Space Management

The free space management is one of the major components responsible for scalability of SecureFiles throughput during file manipulation operations. The layer dedicates one or more SecureFiles segment to a filesystem store database object, each segment being a set of contiguous blocks of underlying shared storage space. File operations such as writes, updates, appends and deletes result in allocation of logical free space from SecureFiles segments or de-allocating used space from files back to SecureFiles segments keeping the real density and seek amortization trend in mind.

The space management layer supports allocation of variable sized chunks. With SecureFiles objects being cached in the Write Gather Cache, the space management layer is able to meet larger space requests from the inode manager through more contiguous layout on disk, therefore providing more scalable read and write access.

The free space in SecureFiles segments is shared across all the instances in a distributed Oracle Real Application Cluster environment. To achieve maximum scalability in a distributed environment, a dedicated background space monitor process on each database server node performs load balancing of free space across the cluster. Each active database server node creates its private in-memory space dispenser shared by processes running on the same node but never across different nodes. As a result, free space allocations requested by server processes are met by the local database node, thereby reducing cluster wide network and storage traffic. The design of the in-memory dispenser allows space allocation operations to scale with the degree of concurrency on a single database node. Private in-memory space dispensers in individual nodes prevent the need for server processes to communicate across nodes in a shared-storage system to maintain free space metadata coherence. The design therefore

alleviates scalability bottlenecks of space allocation operations as the number of nodes in a cluster is scaled up.

Operations such as full overwrites / rewrites, updates and deletes of files in the DBFS server follow 'copy-on-write' semantics resulting in de-allocation of space previously occupied by the offsets affected by the operation. Users can set retention policies for such old data versions in their filesystem stores. Based on the retention policies, the de-allocated space is reused for future allocations once the retention period is over. Such 'copy-on-write' semantics allow extension of database properties to file data, as explained in Section 4.

3.1.4 Automatic Storage System Management

Automatic Storage System Management [16] assists manageability of underlying physical storage. SecureFiles extensively uses this feature to guarantee maximum I/O performance from raw asynchronous I/O operations across the storage system. The feature allows spreading the SecureFiles segment layout evenly across all available storage resources to scale performance and maximize storage utilization across the entire storage system. ASM provides mirroring options for protection against disk failures. Data Transformation Components

3.2 Memory and Storage Utilization

Data transformation components in Oracle SecureFiles allow for optimal utilization of storage space by the DBFS filesystem stores.

3.2.1 De-duplication

When a DBFS filesystem store has de-duplication [18] enabled, SecureFiles automatically detects duplicate files, and stores only a single physical copy on disk, thereby minimizing space usage. A secure hash is generated for a subset of the file data (prefix hash) and also for the whole file (full hash). During streaming writes, once generated, the prefix hash is compared to a set of prefix hashes stored in an index. If there is a prefix match, then the file associated with the original prefix hash (master version) is read and byte-by-byte comparison is performed across the buffered data and the master version. At the end of the write, if the full hash matches and the entire file matches on a byte-by-byte basis, then a reference pointer directing to the master version is maintained in the filesystem store. The component therefore contributes in scaling up throughput of applications that are required to store multiple instances of files, by preventing redundant physical I/O on the underlying storage system.

3.2.2 Compression

When compression option is enabled in a filesystem store, buffered writes from the write gather cache is compressed when it exceeds a configured boundary threshold. These compressed data chunks are referred to as compression subunits. Multiple contiguous compression subunits are encompassed within a larger unit. Compression is performed piecewise in such a way that efficient random access of large files is possible. Compression not only results in significant savings in storage but also improves performance by reducing I/O sizes, database buffer cache requirements, data logging for media recovery, and encryption overheads.

3.3 Secure File Data Management

SecureFiles uses Transparent Data Encryption (TDE) syntax for encryption of files along with the accompanying relational metadata. File buffers are encrypted/ decrypted on database block size units using one of several encryption algorithms, namely, Triple Data Encryption Standard with a 168-bit key size, Advanced Encryption Standard with a 128 bit key size, Advanced Encryption Standard with a 192-bit key size, or Advanced Encryption Standard with a 256-bit key size.

4. EXTENDING DATABASE FEATURES TO FILES

This subsection provides details of the some of the database features supported by Oracle SecureFiles infrastructure that are automatically inherited by Oracle DBFS.

4.1.1 Transaction Atomicity

Storage of files within Oracle RDBMS guarantees transactional atomicity for file and relational data operations in Oracle DBFS. Relational data in filesystem stores is managed using the transaction semantics associated with the relational database kernel. The database kernel implements these semantics by generating undo records for all data and metadata operations. The undo records are stored as first-class objects within the database and are used to roll back database operations during failures thereby maintaining transactional consistency in the database. Similar semantics are used for guaranteeing transactional atomicity of filesystem metadata manipulation operations in Oracle DBFS.

File data operations in DBFS undergo 'copy on write' semantics for overwrite and large update operations. Such a semantic alleviates the requirement to store previous object images, partial or entire, for rollback purposes. When a transaction aborts, the relational metadata associated with SecureFile objects, and space metadata roll back using the undo records. As a result, the SecureFile object locators point to the previous image of the inode metadata blocks that in turn point to the previous versions of the objects. Because of 'copy-on-write' semantics for large updates and overwrites, the rollback is not required to perform additional I/O to restore the previous object images. As a result, transaction recovery becomes independent of the sizes of the changes on the SecureFile objects. For smaller updates, SecureFile objects undergo in-place updates with traditional relational transaction undo, therefore avoiding unnecessary fragmentation. The transaction atomicity semantics guarantee transaction level consistency between files and their associated relational content in Oracle DBFS.

4.1.2 Read Consistency

Oracle RDBMS supports multi-version read consistency for relational data. Queries retrieve data by re-creating snapshots of modified data blocks as of the time of their issuances. The snapshots or versions of relational data blocks are created through application of undo records that were generated during data manipulation operations. While accompanying relational and filesystem metadata in DBFS filesystem use the above techniques to achieve read consistency, files stored as SecureFiles objects achieve this making use of 'copy-on-write' semantics. SecureFiles space management component maintains chunk metadata associated with object updates and deletes. The space freed during

the update and delete operations map to old versions of data. The space management component retains such freed up space for a user-specified amount of time. Depending on the expiration of the retention period, the space management component either retains such space or reuses them for future allocations. This technique guarantees users of Oracle DBFS to retrieve the most read-consistent version of file content along with the associated file and relational metadata at a point in time within the retention period.

4.1.3 Temporality

The read consistency mechanism described in the previous subsection is extended by the Oracle Flashback framework [15] to provide capabilities to query, retrieve, and recreate relational as well as unstructured data consistent as of any point in time in the past, ranging from several minutes to several years. Being first class RDBMS objects, the framework is automatically inherited by DBFS file and relational data management. Content management applications can set retention periods to SecureFiles segments in DBFS filesystem stores. If not explicitly specified by a user, previous versions of files are retained as long as their accompanying filesystem and relational metadata are retained. This ensures consistency of DBFS file data retrieval at any point in time as long as the accompanying filesystem and relational data can be retrieved. SecureFiles with Flashback Archive provide the option for content management applications to create tamper-proof temporal snapshot stores. Such stores can support creation and retrieval of critical content snapshots and accompanying relational data consistent as of several years in the past, extremely relevant to applications in content security and compliance areas.

5. FILESYSTEM STORES

As mentioned in section 2, a filesystem store is a container for files and relational content within the RDBMS. Physically, a filesystem store consists of one or more dedicated tables, referred to as store-tables, along with SecureFiles segments. The layout of a filesystem store is demonstrated through Figure 3.

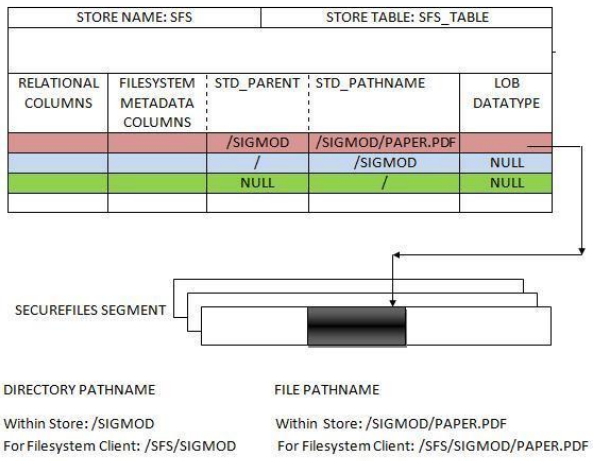

Figure 3: An example filesystem store.

The store-table allows applications to define optional columns that contain relational metadata associated with the file, e.g., location information associated with photograph image files. Besides optional user-defined relational columns, they contain mandatory scalar columns to store standard, well-defined filesystem-specific metadata based on the POSIX standard namespace, such as STD_PATHANME, STD_PARENT, etc. Directories do not have a well-defined length and stores are free to set this property to zero, null, or any other value they choose), std_modification_time, and so on.

Besides the above columns, a store-table consists of one or more attributes of the LOB datatype. For a row in the store table that corresponds to a file record, the LOB datatype column in the row contains reference pointer to the file content that is stored and accessed from the associated SecureFiles segment. The SecureFiles segment is shared by all file records contained in the associated store-table. Storage parameters of SecureFiles segment can be configured to enable different flavors to filesystem stores. For example, a filesystem store targeted towards personal documents may enable SecureFiles compression for storage utilization benefits while a filesystem store targeted towards mission-critical content may enable SecureFiles encryption.

Logically, a filesystem store is characterized by a store-name that contains a collection of files, each identified by a unique absolute path name (that is, a "/" followed by one or more "component names" separated by "/"). Some stores may implement only a flat namespace, others might implement "directories" (or "folders") implicitly, while still others may implement a comprehensive filesystem-like collection of entities: hierarchical directories, files, symbolic links (or just links), hard links (or references), and so on, along with a rich set of relational metadata (or "properties") associated with files.

DBFS allows creation of multiple stores within the same database. The RDBMS allows database transactions, read consistency and other ACID properties to span relational, filesystem metadata and file data in a filesystem store. The store-table being a database object allows access of relational, filesystem metadata and file data through database client interfaces such as PL/SQL, JDBC and OCI.

6. FILESYSTEM SERVER

The Oracle Database Filesystem Server consists of a set of interfaces within the database that provide filesystem-like abstraction of DBFS stores to the clients. Figure 4 demonstrates the components of the DBFS server. The topmost component of the server interface is called the DBFS ContentAPI (CAPI). The DBFS ContentAPI is a collection of interfaces that correspond corresponding to POSIX-standard filesystem access primitives such as create, open, read, write, list directory, change directory, etc. The ContentAPI defines a PL/SQL interface for every POSIX-standard filesystem call interface. The complete set of interfaces defined by the ContentAPI is used by the filesystem client to access underlying filesystem stores.

The DBFS Store Provider API, DBFS SPI, follows the Content API. The SPI allows registration of several user-defined PL/SQL packages or Store Providers, each of which inherits and implements the set of PL/SQL interfaces defined by the CAPI. The DBFS ContentAPI implements a VFS abstraction based on store providers and the SPI. Store providers manage the low-level details of data storage and retrieval and can do so in arbitrary way. This allows applications to create multiple instances of the same filesystem interface through multiple store providers. For

example, a read-only application may implement the PL/SQL interface for write system call to return an error message, while a read-write application would implement the same interface to actually store the data in the SecureFiles segment. A temporal filesystem application may choose to implement the read interface to retrieve file data consistent as of a fixed time in the past, while a traditional filesystem application would implement the same to retrieve the most current versions of file data.

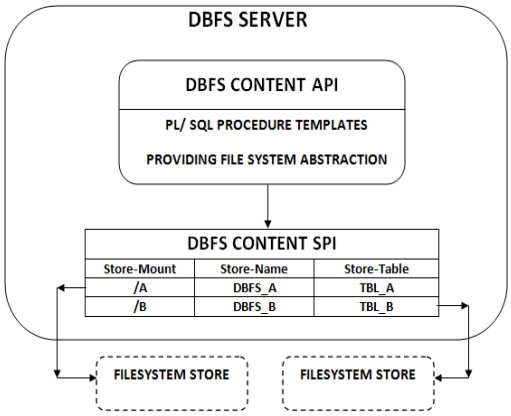

Figure 4: DBFS Server Interfaces

6.1 Registering Filesystem Stores with DBFS Server

As mentioned in the previous section, a DBFS filesystem store is defined by one or more store-tables and is identified by a store-name. Registration of a filesystem store to the DBFS server takes place in three steps. The first step is to create a Store Provider with implementations of the Content API methods. The second step is to associate the Store Provider with the DBFS store-name. The final step is to associate the store with a mount point or store-mount. The store-mount is used to expose the DBFS filesystem store to the filesystem application running on the client.

A filesystem store therefore gets defined as a four-attribute tuple (store-name, store-provider, store-mount, and store-table). The filesystem client accesses files or directories in underlying DBFS stores using a full absolute pathname (a single string): such as "/<store-mount>/<store-specific-path-name>". DBFS manages the namespace and dispatch of end-user filesystem operations based on pathnames.

The following example explains the flow of interfaces on a 'chmod' filesystem call {int chmod(const char *path, mode_t mode)}, where path is "/<store-mount>/<store-specific-path-name>". The DBFS Content API provides an equivalent interface: DBFS_CONTENT.chmod(<store-mount>/<store-specific-path-name>, mode). From the store-mount, the store provider DBFS_<sp> is identified, which inherits the interface as: DBFS_<sp>.chmod(<store-specific-path-name>, mode, store-table) and may implement as "update <store-table> set std_mode = mode where std_pathname = <store-specific-path-name>".

7. FILESYSTEM CLIENT

The DBFS client is built on the Filesystem in User Space (FUSE) infrastructure, as demonstrated in Figure 5. FUSE [14] is a framework for implementing filesystems outside the operating system kernel in a separate protection domain in a user process. The Fuse library interface closely resembles the in-kernel virtual filesystem interface. The DBFS client is an OS user level client that registers function callbacks with FUSE kernel module, which get executed once the OS kernel issues a corresponding request. The function callbacks within DBFS Linux Filesystem Client receive standard filesystem calls from the FUSE kernel module, translate them into the equivalent Content API interfaces and transfer them over to the DBFS server using OCI connections. The client interfaces have been implemented to scale with the number of applications accessing the filesystem. The Linux Filesystem Client dynamically maintains a pool of OCI connections thereby avoiding creation of network connections on every call. Use of write caching, read pre-fetching and load balancing across a pool of database OCI network connections are some of the other optimizations that remove client side latencies.

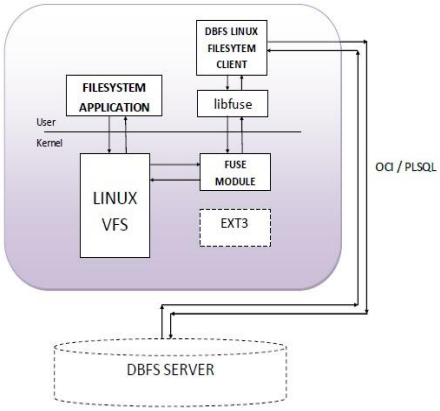

Figure 5: DBFS Linux Filesystem Client Architecture

The Linux Filesystem Client allows mounting DBFS on local hosts, similar to a NFS mount. The client allows multiple mounts of the same filesystem server on a single machine. Multiple clients can run in an Oracle cluster environment that may share the same database and therefore the same filesystem. Applications in the client machines issue standard filesystem calls and commands. The Linux Filesystem Client interfaces get invoked when file, directory, or link pathnames associated with the kernel calls are prefixed with the appropriate DBFS mounts.

The data flow from the client to the filesystem store is enumerated below on the 'chmod' example presented in Section 6.1.

1. Filesystem application issues a chmod: chmod (/DBFS_mount/<store-mount>/<store-specific-path-name>, mode)
2. DBFS client converts it to a Content API interface: DBFS_CONTENT.chmod(/<store-mount>/<store-specific-path-name>, mode) and transfers to the server

3. The DBFS server retrieves the Service Provider DBFS_<sp> to select the appropriate store API and converts it to the method: DBFS_<sp>.chmod(<store-specific-path-name>, mode, <store-table>)
4. DBFS_<sp>.chmod(/<store-specific-path-name>, mode, <store_table>) issues 'update <store-table> set std_mode = mode where std_pathname = <store-specific-path-name>'

8. PRELIMINARY PERFORMANCE EVALUATION

The motivation behind the introduction of an industry-strength database filesystem had been the sub-optimal performance and scalability of storage and access of files compared to filesystems. The section presents a set of preliminary performance evaluation of DBFS primarily focused on read and write operations of files across various sizes. The experiments are conducted on a Sun Oracle Database Machine [17], a state-of-the-art database SMP server and storage cluster system introduced in 2009. The experiment set has been designed to demonstrate and verify the scale of throughput of file data storage and access achievable by DBFS on a high-end server and storage system.

8.1 Objective

The performance evaluation comprises of three sets of experiments with the following objectives. Firstly, throughput and file read and write operations are evaluated on a single database node to observe the scalability of DBFS on a multi-core SMP machine. Once the concurrency configuration providing the maximum scale is determined, the second set of experiments scale out the operations over the entire cluster. The objective is to observe whether DBFS scales out file storage and access over a cluster of servers and shared storage.

The first and second sets of experiments are performed on an empty filesystem store. The third set of experiments extends the cluster-wide set, repeating them for a period to 12 hours. The experiment set comprises of multiple iterations of write/read operations followed by removal of stored files iterations interleaved by removal of files and directories. The objective is to observe and verify whether DBFS is able to reproduce high performance in steady-state.

8.2 System Setup

Figure 6 illustrates the system configuration used in the experiment. The hardware is a half-rack Sun Oracle Database Machine [17] comprising of 4 database server nodes and 7 Exadata storage server nodes.

Each Sun Oracle Exadata Storage Server comprises of twelve 2 TB Serial Advanced Technology Attachment (SATA) disks that provide up to 7 TB of uncompressed user data capacity, and up to 0.85 GB/second of raw data bandwidth. The database machine uses a state of the art InfiniBand interconnect between the servers and storage. An Exadata storage server has dual port Quad Data Rate (QDR) InfiniBand connectivity for high availability. Each InfiniBand link provides 40 Gigabits of bandwidth - many times higher than traditional storage or server networks. The InfiniBand network has the flexibility of a LAN network, with the efficiency of a SAN. The same InfiniBand network also provides a high performance cluster interconnect for the Oracle Database Real Application Cluster (RAC) nodes. Industry standard Oracle The database servers are equipped with two Intel Xeon (Nehalam) dual-socket quad-core E5540 processors running at 2.53 GHz processors, 72 GB RAM, four 146 GB SAS drives, dual port InfiniBand Host Channel Adapter (HCA), four 1 Gb/second Ethernet ports, and dual-redundant, hot-swappable power supplies.

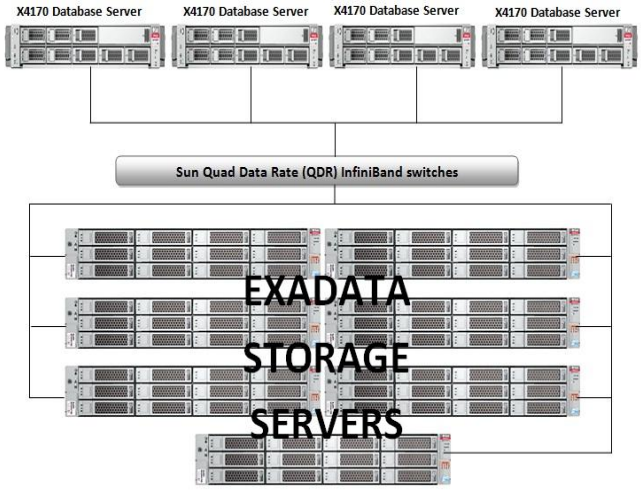

Figure 6. Sun Oracle Database Machine

8.3 Experiment Configuration

A single filesystem store was used for the scope of the experiments. The store comprised of a single non-partitioned database table. The table comprised of the mandatory filesystem metadata columns, a single column of LOB data type, and zero relational columns.

The SecureFiles segment associated with the filesystem store was configured to be shared across all storage servers. The storage disks were configured as raw block devices with Automatic Storage Management. Incoming write operations were configured to issue direct and asynchronous I/Os for the file data to the underlying storage bypassing the database buffer cache. Additional logging of file data was disabled as the writes were configured for direct I/O. Data transformation options were disabled on the SecureFiles segment. The minimum block size was set to 8KB

The Oracle RDBMS contains a reference PL/SQL package, DBFS_SFS, containing the implementations of the DBFS Content API interfaces. Interfaces involving filesystem metadata operations are implemented as transactionally managed inserts/updates/deletes of file and directory records in the store table. The read interface is implemented to return file data consistent as of the current point in time.

The database filesystem client is used to mount DBFS on all four server machines. The common Unix/Linux program 'dd' is used in the experiments to issue file copies from /dev/zero to DBFS and from DBFS to /dev/null, thereby evaluating a more "pure" DBFS-only storage and retrieval performance profile. The experiments were configured to issue filesystem commands with

file pathnames referencing the same filesystem store from all four servers.

8.4 Single Node Experiments

This subsection reports evaluations of scalability of reads and writes operations on a single node. The set of experiments perform file read and write operations with varying degrees of concurrency and varying file sizes to observe DBFS scalability in such environments. Seven experiments are performed, each experiment writing/reading a certain file size. File sizes are varied from 10KB, 100KB, 1MB, 10MB, 100MB, 1GB and 10GB respectively.

8.4.1 10KB File Sizes

The experiment application initiates multiple simultaneous threads, each thread performing a set of serial filesystem 'dd' operations, each operation writing 10KB from /dev/zero to an output file targeted to the DBFS filesystem store within the database. The total number of files inserted in 1.6 millions. The number of simultaneous threads is varied from 32 to 128 in steps of 8. For each concurrency configuration, the number of files written per thread is set as 1.6 million divided by the number of simultaneous threads. Average throughput is measured using the elapsed time for the entire application to reach completion. As evident from figure 7, file writes scale up with the degree of concurrency on a single database server node. However, the workload becomes entirely CPU bound as the underlying physical I/Os comprise of maximal 2 8KB contiguous data blocks, and therefore saturates after the number is increased from 64 upwards. The maximum throughput observed from the experiment is 59.6MB/sec or more than 6000 10KB files per second.

Once all files are stored in DBFS filesystem store, a read application initiates multiple simultaneous threads, each thread performing a set of serial filesystem dd commands, each command issuing a 10KB read of an input file from the DBFS filesystem store and writing to /dev/null. The number of simultaneous threads is varied from 32 to 128 in steps of 8. The throughput behavior is observed to be similar to that of file writes. The throughput saturates after the number of threads increases from 64 onwards. The throughput saturates to 95.7 MB/sec or more than 9600 10KB files per second, as shown in figure 7.

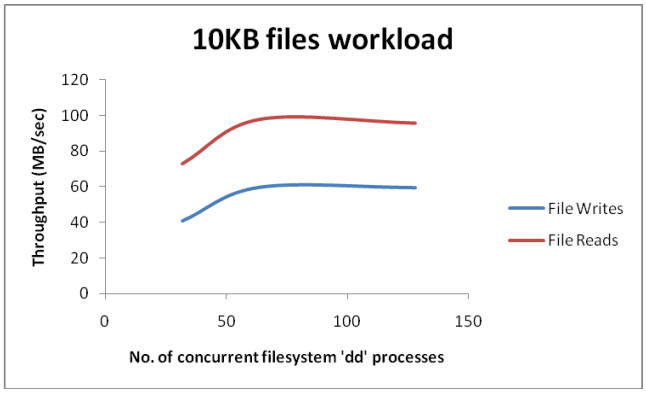

Figure 7. Read and write throughput for 10KB file sizes

8.4.2 100KB File Sizes

The same application described in section 8.4.1 is reused for this experiment, but with different parameters. The total number of files inserted is 1 million. The number of threads is varied from 32 to 128 in steps of 8. As evident from figure 8, throughput of writes scales up with the degree of concurrency on the server machine. DBFS throughput for this specific workload still remains entirely CPU bound, as the underlying physical I/Os comprise of maximal 64 contiguous data blocks. Following a scale up, throughput therefore saturates after the number of processes is increased from 64. The throughput for the document archiving application saturates around 346.3MB/sec, implying data ingestion rate of more than 3200 100KB files per second.

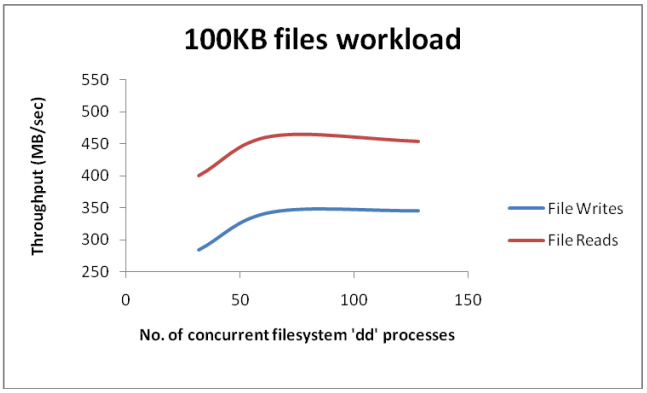

Figure 8. Read and write throughput for 100KB file sizes

For file reads, the number of simultaneous threads is increased from 64 to 128. The read throughput saturates from 64 threads onwards to 454.2MB/sec implying more than 4600 100KB files per second.

8.4.3 1MB to 100MB File Sizes

For 1MB file sizes, the experiment application writes a total of 240,000 files resulting in total ingestion of 234GB. The number of threads is varied from 24 to 96 in steps of 8. For each concurrency configuration, the number of files written per thread is set as 240,000 divided by the number of simultaneous threads. The same configuration is set for the read experiment following the writes. It is observed from figure 9 that DBFS throughput starts becoming I/O bound at lower degrees of concurrency and saturates near the hardware limit of 1GB/sec. The throughput saturates from concurrency levels of 32 onwards and maximizes at 920.4 MB/sec. The read operations in DBFS are also observed to be I/O bound resulting in a throughput saturating at 931.6 MB/sec.

For 10MB file sizes experiment, the experiment application writes a total of 24,000 files. The number of threads is varied from 12 to 48. Both DBFS read and write throughputs saturate around 970 MB/sec, very near to the hardware limit. The 100MB file sizes experiment performs dd writes on 2400 files varying the degree of concurrency from 8 to 32 in steps of 8. The write and read throughputs are entirely I/O bound and saturate near 962 MB/sec, as shown in figure 9.

As the file sizes increase, both read and write throughputs shift from being CPU bound towards being I/O bound. Buffering

optimizations from SecureFiles Write Gather Cache leads to allocation of more contiguous storage space resulting in larger I/O requests that fully utilize the underlying storage bandwidth from a single server machine.

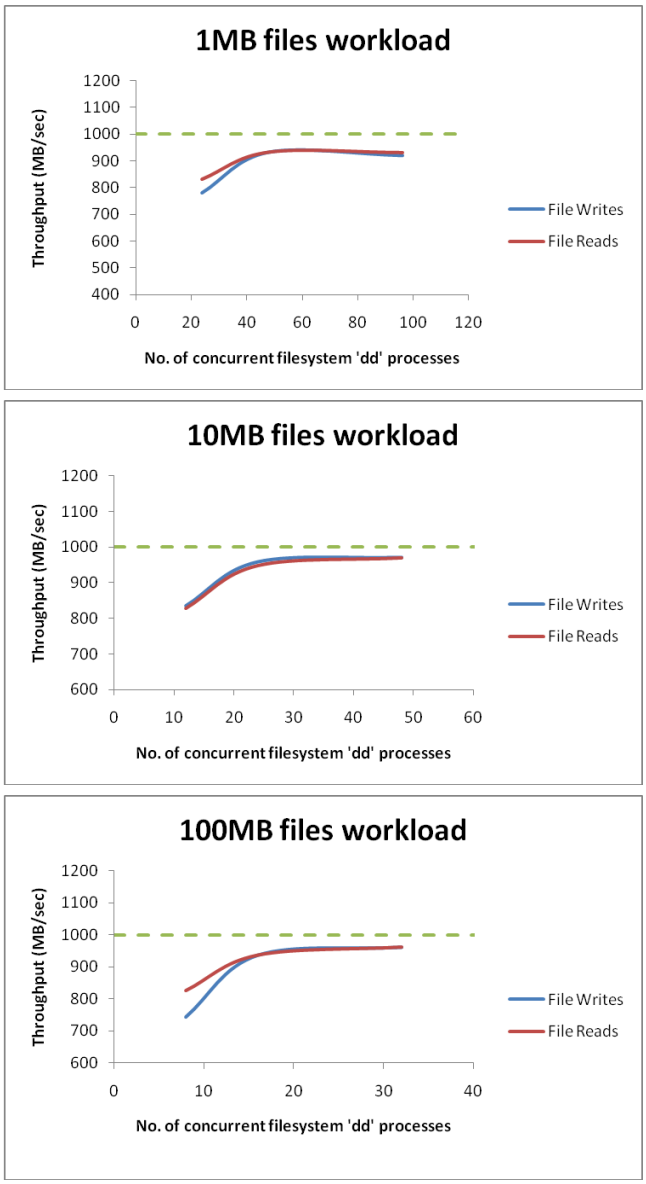

Figure 9. Read and write throughputs for 1MB, 10MB, and 100MB file sizes.

8.4.4 1GB and 10GB File Sizes

The experiment application is the same described in the previous subsections. However, the total number of files written and read is set to 640 and 64 for 1GB and 10GB experiments, resulting in total ingestion and retrieval of 640GB in both cases. Degree of concurrency is varied from 4 to 16 in steps of 2. Similar to above experiments, the number of files read and written is equally distributed across the threads. The read and write throughputs saturate towards the hardware limit of 1GB/sec from the concurrency level of 6 itself and remains so as the levels are increased, as demonstrated in figure 10.

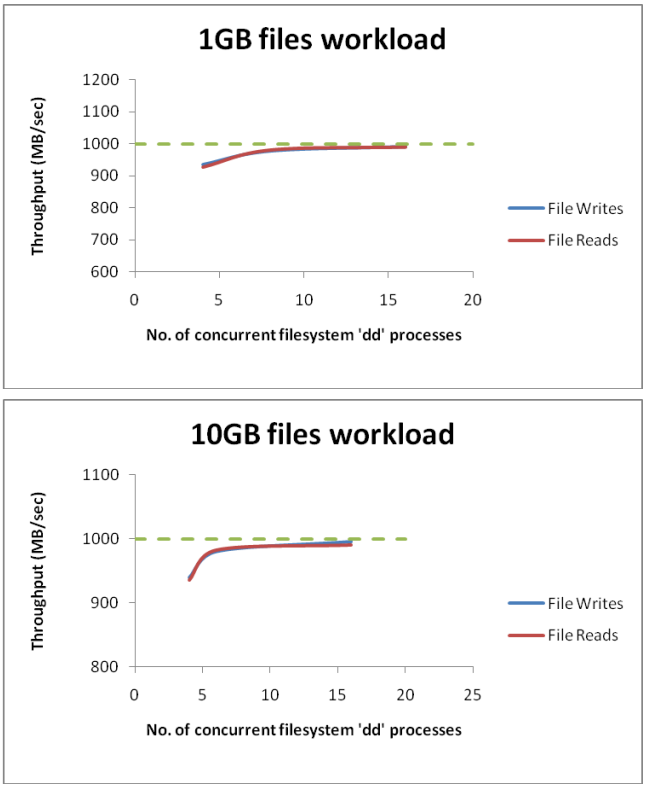

Figure 10. Read and write throughputs for 1GB and 10GB file sizes.

8.5 Cluster-wide Experiments

Results from the previous subsection demonstrate that DBFS performance scales with the degree of concurrency of filesystem write and read operations in a single database server. To observe whether DBFS performance scales out on a shared storage cluster-wide system, multi-node experiments are conducted on two and four database server nodes in the database machine.

The same experiment application described in subsection 8.4.1 is issued from each individual database server. Experiments are conducted on file sizes of 1MB and more as they involve more hardware I/O bound operations. For each file size, the degree of concurrency chosen per node is the one that generates the maximum throughput in the single node experiments. The total number of files written and read is scaled up with the number of nodes for each file size. To summarize, the 1MB experiment ingests and retrieves 240,000 files per node using 96 parallel threads; the 10MB experiment ingests and retrieves 24,000 files per node using 48 parallel threads; the 100MB experiment ingests and retrieves 2400 files per node using 32 parallel threads; the 1GB experiment ingests and retrieves 640 files per node using 16 parallel threads; and, the 10GB experiment ingests and retrieves 64 files per node using 16 parallel threads.

As evident from figure 11, DBFS write and read throughputs scale out with the number of database nodes across all workloads. Even though a single DBFS filesystem store is shared across the DBFS clients and server nodes and the filesystem store itself shares underlying physical storage, free space management algorithms, consistent read mechanism, and cluster–wide read sharing contribute to the scale out of throughput. DBFS filesystem write operations are driven with more than 3.4 GB/sec ingestion rate while the reads generate a throughput of more than 3.5 GB/s across the cluster.

To summarize, DBFS achieves more than 10TB/hr file ingestion and retrieval rates in a four node database cluster on an empty filesystem.

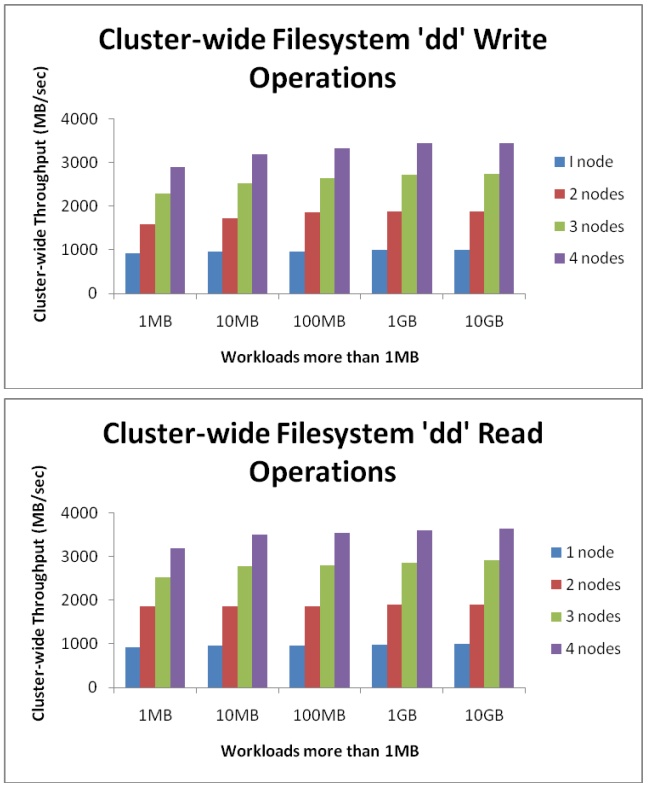

Figure 11. Cluster-wide scale out of file data operations throughput

8.6 Steady-state Reproducibility Experiments

The next set of experiments is performed to observe whether DBFS is capable to consistently reproduce such high throughputs in a steady-state filesystem. This set of experiments extends on the multi-node experiment application used in section 8.5. The multi-node experiments performed for file sizes ranging from 1MB to 10GB are iterated multiple times. Between each iteration, all inserted files are deleted through a single 'rm –rf' command targeted to the filesystem store.

The iterations are performed continuously for 12 hours to observe the decay in throughput performance with time. Figure 12 demonstrates the rate of decay for all file sizes tested. The throughput values reported are averaged over read and write operations after each workload before removing the files. It is observed that decay in throughput increases with the file size with 1MB file sizes demonstrating the least (1.8%) and 10GB sizes demonstrating the greatest (9.3%). It is important to note that a steady-state behavior that is eventually reached for every file size. The observation can be explained in terms of filesystem fragmentation. The probability of allocating bigger contiguous chunks decreases with the number of iterations, at a rate faster than the probability of allocating smaller contiguous ones.

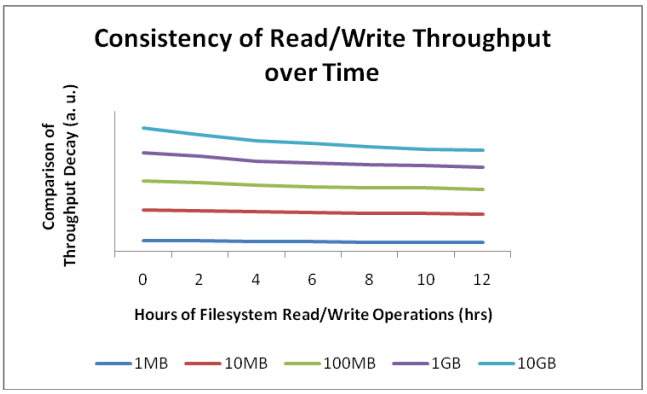

Figure 12. Consistency of DBFS cluster-wide read/write throughput

Although DBFS throughput for larger file sizes decays with time, there is a steady-state behavior that is eventually reached. The highest observed 9.3% decay after 12 hours of continuous ingestion and deletion of files still implies more than 10TB/hr of steady-state throughput of file read and write operations. The results demonstrate the capability of Oracle DBFS to provide file read and write throughput that consistently scales with the underlying hardware system in a steady state environment.

8.7 Summary of Preliminary Evaluations

The preliminary results, summarized in Tables 1 and 2, demonstrate the capability of Oracle Database Filesystem to break the performance barrier across sizes ranging from tens of kilobytes to tens of gigabytes that has been discouraging storage of unstructured file-like content in databases.

It is observed that DBFS demonstrates scale up with the number of filesystem read/write application processes on a single database/filesystem server. For smaller file sizes, DBFS throughput gets bound by CPU of the server, while for sizes around IMB and larger; throughput scales up and gets bound by the I/O capacity of the underlying storage system. On a cluster of 4 database servers of Sun Oracle Database Machine, DBFS scales out to provide more than 10TB/hr throughput for filesystem write operations and read operations. Furthermore, DBFS demonstrates the capability to reproduce the scale out in a steady state filesystem, consistently generating more than 10TB/hr cluster-wide throughput under long-running filesystem read/write applications.

Considering a single DBFS server being able to ingest an equivalent of more than 8320 million 10KB small size files a single day and a cluster of four such servers being able to ingest

an equivalent of 25 million 100MB large size files at the same rate, the current set of performance results implies huge potential for DBFS as the consolidated storage solution for the entire spectrum of existing and future industry-strength content management applications.

Table 1. Summary of preliminary DBFS evaluation (single database server)

File Size	No. of Files	Aggregate Write Throughput (MB/s)	Aggregate Read Throughput (MB/s)
10 KB	1,600,000	59.6	95.7
100 KB	1,000,000	346.3	454.2
1 MB	240,000	920.4	931.6
10 MB	24,000	970.0	970.0
100 MB	2,400	962.0	962.0
1 GB	640	970.0	970.0
10 GB	64	970.0	970.0

Table 2. . Summary of preliminary DBFS evaluation (four database servers)

File Size	No. of Files	Aggregate Write Throughput (MB/s)	Aggregate Read Throughput (MB/s)
1 MB	960,000	2897.6	3184.6
10 MB	96,000	3193.6	3503.1
100 MB	9,600	3334.2	3542.2
1 GB	2,560	3443.7	3594.8
10 GB	256	3450.1	3634.2

9. Conclusion

Content management applications across enterprises, internet, research, and healthcare industries are generating volumes of data at rates doubling year after year. More than eighty five percent of such data volumes are unstructured or file-like in nature and the rest comprise of accompanying relational content. Although the database is a preferred choice for relational data storage, poor performance of unstructured data storage and lack of filesystem interfaces have deterred content management application developers to use the database as a single storage solution for all data. This may result in less than fifteen percent of all data stored outside database management systems in the coming decades. The paper introduces Oracle Database Filesystem, the latest filesystem client-server architecture within the RDBMS kernel that allows content management applications to transparently store and organize files using standard filesystem interfaces, in the same database that stores associated relational content. The server component of DBFS is based on Oracle SecureFiles infrastructure that provides high performance storage and access of unstructured content in the database through POSIX-standard filesystem interfaces, without compromising on database management robustness features. The architecture opens the database to existing and future content generating applications, offering a no-compromise alternative to filesystems for unstructured data storage.

A preliminary performance evaluation successfully demonstrates the potential of Oracle DBFS to provide very high scalability of files storage and access operations in massive data management environments. Immediate future work on Oracle DBFS includes efforts to conduct exhaustive performance evaluations on all kinds of filesystem operations not limited to file data related ones using standard filesystem benchmarks, and compare them against existing traditional non-database network filesystems.

10. ACKNOWLEDGEMENTS

We are thankful to Mr. Juan Loaiza for giving us the opportunity to employ the Sun Oracle Database Machine setup for the preliminary performance evaluation. We also thank Kevin Closson for assisting us with the hardware setup as well as conducting the performance tests on DBFS. We acknowledge all members of Oracle SecureFiles and Oracle DBFS virtual teams for their contributions in the entire product lifecycle, from brainstorming to product design, product development, and quality assurance.

11. REFERENCES

[1] Lewis. M. *Information 2.0*. An EMC^2 White Paper, 2008.

[2] Lallier, J. *Storage Management in the Year 2010*. Computer Technology Review, September 2004.

[3] *You Tube Fact Sheet*. A YouTube White Paper, 2009.

[4] *Facebook*. http://www.facebook.com/ statistics, 2009.

[5] Sears, R., Ingen, C., Gray, J. To *BLOB or not to BLOB*: Large object Storage in a database or a Filesystem? Microsoft Research Technical Report, MSR-TR-2006-45, 2006.

[6] Gray, J. *Greetings! From a Filesystem User*. 4th USENIX Conference on File and Storage Technologies, San Francisco, CA, 2005.

[7] Carey, M. J., Dewitt, D. *Of Objects and Databases: A Decade of Turmoil*. Proceedings of the 22nd Very Large Data Bases Endowment, 3-14, 1996.

[8] Mukherjee, N., Aleti, B., Ganesh, A. et. al. *Oracle SecureFiles System.* Proceedings of the 34th Very Large Data Bases Endowment, 1(2), 1301-1312, 2008.

[9] Mukherjee, N., Ganesh, A., Kunchithapadam, K., Muthulingam, S. *Oracle SecureFiles - A Filesystem Architecture in Oracle Database Server.* ICSOFT (SE/MUSE/GSDCA), 60-63, 2008.

[10] Mukherjee, N., Ganesh, A. et. al. *Oracle SecureFiles: Prepared for the Digital Deluge.*. Proceedings of the 35th Very Large Data Bases Endowment, 2009.

[11] *The NFS Version 4 Protocol.* A Sun Solaris 10 White Paper, 2000.

[12] Lahiri, T., Srihari, V., Chan, W., Macnaughton, N., Chandrasekaran, S. *Cache Fusion: Extending Shared-Disk Clusters with Shared Caches*, Proceedings of the 27th VLDB conference, 2001.

[13] Cryan, M. *Oracle Database Concepts*. An Oracle White Paper, 2003.

[14] Szeredi, M. *Filesystem in USErspace.* http://fuse.sourceforge.net/.

[15] Rajamani, R. *Oracle Total recall/ Flashback Data Archive*. An Oracle White Paper, June 2007.

[16] Manning, P. *Automatic Storage Management technical Overview*. An Oracle Technical White Paper, 2003.

[17] Weiss, R. A Technical Overview of the Sun Oracle Exadata Storage Server and Database Machine. An Oracle Technical White Paper, 2009.

[18] Biggar, H. *Experiencing Data De-Duplication: Improving Efficiency and Reducing Capacity Requirements*. A SearchStorage.com White Paper, Feb 2007.