FILE SERVICE in DISTRIBUTED SYSTEMS

- one of the most important services provided by a distributed system is the file service, other services rely on file service (print service, name service etc.)

- users contact the distributed system through the file service

- one of the primary functions of an operating system is providing file storage capacity

- there are two models of distributed system:

1.integrated model of distributed computing:

- all users has the same view of the system, each computer runs a complete (but highly configurable) set of standard software which provides all the needed facilities

2. client-server model:

- services are designated to a computers (=servers), servers run special purpose server software, clients are a priori unknown (and placed on the local or remote machine)

Basic aspects of file services and servers

service: a software entity running on one ore more machines

server: a service software running on a single machine

Services provided by the file server

1. sharing data and databases

- automatic backup and recovery against user errors and storage media failure

2. user mobility

- users may want to use different computers at different times: e.g. computer fail, user often changes his/her location

3. diskless workstations

-low-cost solution

-can be used in an extreme environment (noise, dust, heat)

Classifying file servers

1. true file server:

- file names, directory structure

2. storage server:

- only unstructured object, primitive identifiers

3. simple file server:

- unit of data access, locking

4. database management file server:

unit of data access, locking: record or variable

-traditional file service:

- higher performance

-robust file service:

- provides extremely high reliability

-explicit file transfer:

- FTP

- distributed file system:

- file location is transparent to the client

Design goals of a distributed file system

1. Diversity of application:

- file system should support all the applications

2. File system semantics:

- should be easy to understand

3. Matching user requirements:

- a file system should provide only the services to the user needed by him/her, the user don't want to pay for the unnecessary features

4. Consistent naming:

- every user and process in a distributed system uses the same *text* name for accessing a file regardless of the location of the object.

- This results in the possibility of *identifying* and *accessing* a particular file anywhere in the distributed system.

5. Location transparency:

- a file should be movable in a distributed system without changing it's name.

- the user don't have to bind the specified file to a particular computer.

6. Transparent access:

- processes should not *need* or *able to be* recognize whether a file is local or remote,

- they can access them in the same way.

7. Data consistency:

- file system must guarantee the integrity of data, changes to files must never be lost. (concurrency control problem)

8. Consistent authentication:

- users should recognize the identity of the others, users should be treated in the same way regardless of current physical location

9. Information protection:

- information stored in a file system should be protected against non-authorized access.

10. security issues:

- safe communication, built-in protection mechanisms (e.g. each file usually belongs to its owner who created it, and the system provides different levels of user granularity: owner, group, other)

11. User mobility:

- users should be able to access their files from any workstation in a distributed system.

- the performance should be about the same between own and other computers.

12. Performance:

- users should not be able to recognize the existence of distributed system on the grounds of performance.

- It means that the level of file system performance should be at least as good as that of an equivalent time-sharing system.

13. Availability:

- The failure of a computer or an element of the network should not affect the *whole* distributed system.

14. Crash recovery:

- servers and user processes must be able to recover easily from computer crashes or network problems, recovery mechanisms should be reliable

15. Scalability:

- The distributed system should be changed (enlarged) without any loss of performance or service.

16. Supporting different storage media

Architecture of a centralized file system

Basic services:

-disk (block) service

-file service

-directory service

remark: these services are not usually separated

Architecture of a distributed file system

Basic services:

-disk (block) service

-file service

-directory service

Additional services for reasons of distribution, reliability and performance:

-replication and multiple copy update

-transactions and atomic update control

-naming of distributed objects

-placement of files

Modules of a distributed file system

<table>
<tr><td colspan="3">NAMING/DIRECTORY SERVICE</td></tr>
<tr><td colspan="2">Directory module (11)</td><td>Access control module (12)</td></tr>
<tr><td>Placement module (13)</td><td>Location module(14)</td><td>Relocation module (15)</td></tr>
<tr><td colspan="3">REPLICATION SERVICE</td></tr>
<tr><td colspan="3">Data consistency and multiple copy update module (10)</td></tr>
<tr><td colspan="3">TRANSACTION SERVICE</td></tr>
<tr><td>Transaction op. m. (7)</td><td>Recovery module (8)</td><td>Concurrency control m. (9)</td></tr>
<tr><td colspan="3">FILE SERVICE</td></tr>
<tr><td>File module (4)</td><td>File access m. (5)</td><td>Attribute access m. (6)</td></tr>
<tr><td colspan="3">BLOCK (DISK) SERVICE</td></tr>
<tr><td>Block module (2)</td><td>Device module (1)</td><td>Caching module (3)</td></tr>
</table>

1. Device module:

performs I/O and buffering

2. Block module:

- accesses and allocates disk blocks

3. Caching module:

- caches recently obtained results of disk operations

4. File module:

- relates file identifiers (IDs) to particular files

5. File access module:

- reads and writes the file *data*

6. Attribute access module:

- reads and writes the file *attributes*

7. Transaction operations module:

- provides transaction mechanism.

- transaction is a group of connected, elementary operations

8. Recovery module:

- provides recovery mechanism after failure

9. Concurrency control module:

- provides concurrency control mechanisms.

- there are two requirements:

a. ensuring that object consistency is preserved and

b. each action is completed in finite time

10. Data consistency and multiple copy update module:

- improves performance and reliability.

- maintains data consistency and multiple copy update mechanisms.

11. Directory module:

- provides mapping between user *text* names and references to files. (=resolves user names)

12. Access control module:

- checks rights for the requested operations

13. Placement module:

- selects a server to hold a newly created file

14. Location module:

- finds the location of (the server managing) the referenced file

15. Relocation module:

- improves the performance by relocating files (by current needs).

DISK SERVICE

- provides a logical view of storage system

- reading or writing raw disk blocks without regard to how they are organized

- in many distributed file systems the disk service is separated from the file service. This separation makes it easy to combine different methods of storage, and different storage media.

- combines high speed with high reliability

Advantages of block service

- differently-specified file servers can coexist and share the same disk storage device in a single distributed system

- a variety of disks and other media can be used

- the implementation of file service independent from hardware concerns

- a file can be stored in a non-contiguous set of blocks of storage (this implies the need for recording a sequence of pointers (file index) to the blocks or other method.

- By the aid of file index, both sequential and random access to the items of a file are supported. File index also contains another item: file length)

Caching system

- the performance of any computer system partially rely on the number of disk access operations and remote operations, it is necessary to avoid them whenever it is possible

cache is a set of local copies of blocks or remote data

Problems to solve about cache

- how to maintain consistency between blocks cached in more than one location

- how to maintain reliability in case of system crashes

- how to avoid overloading

- how to avoid duplicated caching (performance and consistency issues)

Solutions to the cache consistency

1. Passive:

- do nothing, it is the responsibility of the users to be careful

2. Server maintains cached objects:

- the server keeps track of which client has which blocks and files in its cache.

- when a client modifies a block or a file, it informs the file server.

- the file server informs the other clients that the particular block or file is invalid and has to be removed from their cache.

3. Server monitors object access:

- clients can only cache *files*.

- a client informs the file server when opening a file indicating whether this is to be opened for reading or writing.

- server must react only if one of the processes writes the particular file and there is another process reading that file at the same time. In this case the server asks the writing process to upload the modified file that can be accessed by the others.

- Only efficient if the shared files are rarely updated (in most cases they are).

Design decisions on caching

- granularity of cached data

- location of client's cache: main memory or local disk

- the way of propagating of modifications on cached objects

- the ways of determining of consistency of cached data

Service functions provided by a disk service

- disk service basically provides allocating new blocks, reading or writing and releasing blocks.

- disk services:

1. allocate_block(): allocates a new block

2. free_block(block): releases a block that are no longer required

3. get_block(block): the specified block is read from disk and stored in a free buffer of the cache, and a copy is sent to the calling process. If the block is already in the cache, no access is made.

4. put_block(block, data): data stored in the cache is copied to disk at the given address

5. flush_block(): removes block from the cache.

- during read and write operations checksums are used to detect errors.

Service design issues:

- size of the block pointer, it determines the maximum number of blocks

- size of the block: it affects the optimal utilization of disk storage and performance

- duplicating blocks: each block data can be stored at two separate disk blocks - it involves reliability

FILE SERVICE

- file service provides an abstraction of files, each of which is a linear sequence of data items (records)

- data items (records) can be system-defined or user-defined

- the user is not concerned with how and/or where the data in the file are stored or with any relationships between files

File storage

file:

- unstructured sequence of data - this is the simplest model.

- sequence of modifiable data items of a given size

- files also have attributes that describe them (e.g. file name, allowed operations, access control data, date and time of creation, last modification and last read, owner, encryption key)

- each attribute has a name, type and value

- files are stored in addressable blocks of storage whose size is fixed by the storage device (block module - block service)

Unique file identifiers

- every file maintained by a file system has at least one user name and a unique system name (usually a long integer of fixed length)

- different ways to generate unique file identifier (UFID): e.g. server identifier + file number + time stamp + random number

File location

- users don't need to know the location of the files.

- a translation is necessary from user name to UFID, and from UFID to block pointer

- if UFID contains the first block of the given file index, no file location map is required

Mutable and immutable files

- a mutable file is a container for variable information

- an immutable file cannot be modified once it has been created except to be deleted

- the name of an immutable file cannot be reused (it implies that the name of that file signifies the contents of the file)

- sharing only immutable files makes it easy to support consistent sharing

- each file is represented by a history of immutable versions. On update, the immutable file is not updated but a new version is created. (Sometimes only a record of difference between versions is stored, to minimize the storage required)

- old versions are removed when there is a shortage of disk space or when a version count reaches its allowed maximum

File service operations on data

- a file contains data and has attributes. It implies that there are two separate sets of operations, one to deal with the data and the other to deal with attributes

file services:

1. **create(file): creates a new file of length 0 with the specified name *file*, establishes its default attributes, and delivers a UFID for it.**

2. **open(file):opens the existing *file*, that can be used to perform different operations on it**

3. **copy(source, destination): copies *source* file to *destination***

4. **get_size(file): return the *file* size**

5. **read(file, offset, count, buffer): reads up to *count* bytes of data from *file* starting *offset* bytes and puts them contiguously into *buffer***

6. **write(file, offset, count, buffer): writes *count* bytes of data from *buffer* to *file* beginning *offset* bytes from the beginning of the file**

7. **close(file): terminates access for actions to be taken on**

8. **delete(file): removes the *file* from the file store.**

File service operations on attributes

- the file service maintains a set of attributes for each file, such as date and time of creation, last modification, last read access, owner, access list, allowed operations

- file services operation on attributes:

1. read_attribute(file): returns the values of the attributes of the *file*

2. write_attributes(file, attribute): modifies (writes) the existing value of the attribute

Operational properties of file systems

- there are two properties of operations said to be important in a distributed system: idempotency and being stateless:

Idempotency:

- an operation is idempotent if its effect when executed more than once is the same as for a single execution.

- non-idempotent operations must be prevented by the server by implementing duplicate suppression.

Stateless file servers:

- stateless server is a server which doesn't maintain its internal state. It implies that the parameters necessary to carry out an operation are stored by the client, and is passed to the server in a request message.

- the server does not keep track of any past request. After opening a file, the index to this entry is returned to the client for use in subsequent request. (other example: read-write pointer)

-a stateless server also satisfies crash recovery: when a server crashes, the client retransmits only its request message until a response message is received. When the client crashes, no recovery is necessary.

-recovery using a stateful server is much harder: the client must detect server crashes on order to rebuild the server state, and the server must detect client crashes in order to discard any state holding for the client.

-a stateless server should provide support for atomic transactions

-it is difficult to implement a normal file system in a fully stateless way (e.g. deleting an open file, change permissions on file)

TRANSACTION SERVICE

- if a basic file service is used concurrently by a number of processes to read from or write data to the same file, unpredictable effects on the file can result.

- if a client or file server fails, undetected errors can be produced.

- to avoid these problems, the atomic transaction service should be provided

- it means that operations (create, open, write, close…) can be grouped into a transaction (using begin_transaction and end_transaction)

- each transaction must be recoverable: when the *client or server* stops for whatever reason, the transaction must either be completed or all the updated files must be returned to their state before the transaction was started.

- if the transaction has progressed normally the client is informed by the server that the transaction is committed. **Otherwise, the transaction is** aborted.

- the fact that a transaction can be aborted implies that all of the file operations must be performed in a tentative manner.

- Solutions to reserve consistency and to recover after failures: intentions list approach, file version approach

file version approach:

- a file have a sequence of versions which are the results of file modifications. Each of these versions is written only once and is tentative.

- when the server commits a transaction, the most recently committed version of the file becomes the current version (note that a committed transaction cannot be undone)

Transaction service operations

- when the client sends a begin_transaction() **request, the server creates a new transaction and returns a unique transaction identifier (transID), all**

further messages within that transaction must carry that ID.

- after completing all the operations grouped into the transaction, the client sends the end_transaction(transID) **request to close (and commit) the transaction. The returned result indicates whether the transaction has committed or failed, or whether the transaction is inactive because it is aborted.**

- abort_transaction(transID)**: client aborts the transaction, all updated files are undone and the state of the transaction is set to inactive**

Concurrency control

- concurrency control problems can be solved using different methods:

1. Locking:

- the server sets a lock (labeled with the transaction identifier) on each data item (file) before it is accessed. Only the transaction which locked the data item can access it, other transactions have to wait until the data item is unlocked.

2. Optimistic concurrency control:

- the server allows to run concurrently and before a transaction is committed, the server checks whether the transaction has used the same data items as an earlier transaction. In this case the transaction must be aborted and restarted.

3. Timestamps:

- the server records the most recent time of reading and writing of each data item. Each transaction compares its own timestamp with that of the data item currently accessed.

REPLICATED FILE SERVICES

- in a distributed system there can be more than one file server (for reliability, performance or other reasons)

- each file can be placed at more than one file server

- all these servers must collaborate to provide an integrated file service, which is transparent to location, distribution and replication of files

- there are two classes of collaborating file servers:

1. each file server is managing a discrete partition of the total set of files (no replication)

2. file servers store replicated files.

The master/slave strategy

- there is one distinguished master server, and one or more slave servers for each replicated file

- the master server services all the update requests, after update the slaves get the new version of the file

- efficient if the file is rarely updated (YP)

Distributed update control

- the true solution

- any file server can accept an update request on a replicated file

- difficult to implement, first of all, because of performance issues

NAMING & DIRECTORY SERVICE

- naming service manages user names and their mappings into the system names

- system names should be unique (UFID), file service should be transparent to location, distribution and replication of the files

Naming schemes

1. Simple naming:

- files are named by a combination of their server name and local name, this guarantees the uniqueness

2. Mounting remote directories:

- mounting remote directories to local directories provides transparent access only for the files placed in the mounted directory (NFS)

3. Single global name space:

- a single name is unique in the whole system

Directory service

- the directory service is concerned with naming and protecting files

- a directory is usually a file and a set of functions which distinguishes it from an ordinary data file. It seems to be a fixed size record file, each record corresponds to an entry in the directory

- directory service is a client of file service

- each directory has a UFID

- the directory service protects the files against unauthorized access

- directories usually organized into a tree, it guarantees uniqueness of names and efficient name resolution

- directory service operations are as follows:

create_dir(dir, parentdir):

- creates a new directory *dir* in the directory *parentdi*

open_dir(dir):

- opens the directory *dir*

read_dir(dir, name, accessmode):

- returns the UFID of the file *name*, checking whether the client making the request is authorized to access that file in the manner specified in *accessmode*

insert(dir, name, file):

- adds the *(name, file)* pair to directory *dir*

un_name(dir, name):

- removes the entry containing *name* from directory *dir*

change(olddir, oldname, newdir, newname):

- changes the entry from the old place to the new one

get_names(dir, pattern):

- returns the set of the names in directory *dir* that mach the regular expression given by *pattern*

close_dir(dir):

- closes the directory *dir*

delete_dir(dir, parentdir):

- removes the empty directory *dir* from the directory *parentdi*

***r*ch_dirAttr(dir, attributes):**

- changes the access control attributes of directory *dir*

Access control

- access control requires the directory service to access and update the file attributes

- each file has an owner, and ownerID is recorded in the attributes

- directory service should provide operations which allow the owners of the files to grant and revoke access rights to themselves and to other users

- accessing and updating directories requires a special treatment of attributes

Placement and location of files in a distributed file service

- in a distributed system there can be more than one file server (for reliability, performance or other reasons)

- each file can be placed at more than one file server

- all these servers must collaborate to provide an integrated file service, which is transparent to location, distribution and replication of files

- there are two classes of collaborating file servers:

1. each file server is managing a discrete partition of the total set of files (no replication)

2 file servers store replicated files.

- when file servers hold partitions of the total set of files, usually each partition is associated with a particular application (or set of applications). In this case the given file server manages its partition quite independent from the other servers

- the user in not aware of the location of the server holding his/her file

- three basic problems in location and placement of files:

1. placement problem:

- a server must be chosen to hold the file when it is created (the selection is performed automatically to minimize communication costs between the client and the server, or specified by the client)

2. location problem:

- the server which holds the given file must be found when the client wishes to refer to it (there are different location algorithms: e.g., there is a server which maps each UFID to the address of the server holding that file)

3. relocation problem:

- this problem arises when files need to be relocated for performance or other reasons

- to support placement and location the following primitives are given:

1. place(file):

- finds the best file server for *file*

2. locate(file):

- finds the address of the server holding *file*