`High level design of a data-block-allocator

By Hua Huang <hua.huang@clusterstor.com>

Date: 2010/6/20
Revision: 0.5

This document presents a high level design (HLD) of a data block allocator for Colibri C2 core. The main purposes of this document are: (i) to be inspected by C2 architects and peer designers to ascertain that high level design is aligned with C2 architecture and other designs, and contains no defects, (ii) to be a source of material for Active Reviews of Intermediate Design (ARID) and detailed level design (DLD) of the same component, (iii) to serve as a design reference document.

The intended audience of this document consists of C2 customers, architects, designers and developers.

0. Introduction

In Colibri Core, global objects comprise sub-components, and sub-components are stored in containers. A container is a disk partition, or volume, with some metadata attached. These metadata are used to describe the container, to track the block usage within the container, and for other purposes. Sub-components in the container are identified by an identifier. Sub-components are accessed with that identifier and its logical targeted offset within that sub-component. Sub-components are composed by data blocks from the container. A container uses some metadata to track the block mapping from sub-component-based logical block number to container-based physical block number. A container also need some metadata to track the free space inside its space. The purpose of this document is to have the high level design for such a data block allocator, which tracks the block usage in the container.

The data block allocator manages the free spaces in the container, and provides "allocate" and "free" blocks interfaces to other components and layers. The main purpose of this document is to provide an efficient algorithm to allocate and free blocks quickly, transactionally and friendly to I/O performance.

The mapping from logical block number in sub-components to physical block number is out of the scope of this document. That is another task, and will be discussed in another document.

1. Definitions

- Extent. Extent is used to describe a range of space, with "start" block number and "count" of blocks.
- Block. The smallest unit of allocation.
- Block Size. The number of bytes of a Block.
- Group. The whole space is equally divided into groups with fixed size. That means every group has the same amount of blocks. When allocating spaces, groups are iterated to search for the best candidate. Group is locked during this step. Having multiple groups in a container can reduce the lock contention.
- Super Group. Group 0 is an special group. It contains metadata of this container, and is not allocatable to ordinary objects.

2. Requirements

- The data-block-allocator should perform well for large continuous I/O, small object I/O.
- The data-block-allocator should survive node crash.
- DB5 should be used to track the block usage.
- The allocator should have similar strategy as the ext4 allocator.

- The allocator should be designed for concurrency by multiple processes
- The allocator should support container inclusion
- The allocator should support merging allocation data of sub-containers into that of their parents
- The allocator should leave FOL traces sufficient to support FOL driven fsck plugins which support all important operations fsck normally provides.
- Pre-allocation is supported

3. Design highlights

C2 data-block-allocator will use the same algorithm as that from ext4 to do block allocation. But instead of using bitmap to track the block usage, C2 will use extent to track the free space. These free space extent will be stored in database, and updated with transaction support.
Highlights in the C2 allocator (or derived from ext4 allocator) are:

- Container is equally divided into groups, which can be locked respectively.
- Extents are used to track the free spaces in groups.
- Every group has a statistic meta-data about this group, such as largest available extent, total free block count.
- All these information are stored in databases. We will use Oracle DB5 in our project.
- Object-based pre-allocation and group-based pre-allocation are both supported.
- Blocks are allocated in a bunch. Multi-blocks can be allocated at one time. Buddy algorithm is used to do allocation.

4. Functional specification

4.1 C2_mb_format

Format the specified container, create groups, initialize the free space extent.
int C2_mb_format(c2_mb_context *ctxt);

- ctxt: context pointer, including handle to database, transaction id, global variables. The allocation database is usually replicated to harden the data integrity.
- return value: if succeeded, 0 should be returned. Otherwise negative value will be returned to indicate error.

4.2 C2_mb_init

Init the working environment.
int C2_mb_init(c2_mb_context **ctxt);

- ctxt: pointer to context pointer. The context will be allocated in this function and global variable and environment will be set up properly.
- return value: if succeeded, 0 should be returned. Otherwise negative value will be returned to indicate error.

4.3 C2_mb_allocate_blocks

Allocate blocks from the container.
int C2_allocate_blocks(c2_mb_context *ctxt, c2_mb_allocate_request * req);

- ctxt: context pointer, including handle to database, transaction id, global variables.
- req: request, including object identifier, logical offset within that object, count of

blocks, allocation flags, preferred block number (goal), etc.
- return value: if succeeded, physical block number in the container. Otherwise negative value will be returned to indicate error.

4.4 C2_mb_free_blocks

Free blocks back to the container.
int C2_free_blocks(c2_mb_context *ctxt, c2_mb_free_request * req);
- ctxt: context pointer, including handle to database, transaction id, global variables.
- req: request, including object identifier, logical offset within that object, physical block number, count of blocks, free flags, etc.
- return value: if succeeded, 0 should be returned. Otherwise negative value will be returned to indicate error.

4.5 C2_mb_enforce

Modify the allocation status by enforce: set extent as allocated or free.
int c2_mb_enforce(c2_mb_context *ctx, bool alloc, c2_extent *ext);
- ctxt: context pointer, including handle to database, transaction id, global variables.
- alloc: true to set the specified extent to be allocated, or false to set them free.
- ext: user specified extent.
- return value: if succeeded, 0 should be returned. Otherwise negative value will be returned to indicate error.

5. Logical specification

All blocks of data only have two state: allocated, or free. Free data blocks are tracked by extents. No need to track allocated in this layer. Allocated data will be managed by object block mapping or extent mapping metadata. This will be covered by other components.

The smallest allocation and free unit is called a block. Block is also the smallest read/write unit from/to this layer. For example, a typical ext4 file system would have the block size as 4096 bytes.

The container is divided into multiple groups, which have the same sizes of blocks. To speedup the space management and maximize the performance, lock is imposed on the granularity of groups. Groups are numbered starting from zero. Group zero, named "Super Group", is reserved for special purpose, used to store container metadata. It will never be used by ordinary objects.

Every group has a group description, which contains many useful information of this group: largest block extent, count of free blocks, etc. Every group description is stored in database as a respective table.

Free space is tracked by extent. Every extent has a "start" block number and "count" of blocks. Every group may have multiple chunks of free spaces, which will be represented by multiple extents. These extents belonging to a group will be stored in a database table. Every group has its own table. Concurrent read/write access to the same table is controlled by lock per group.

Allocation of blocks are using the same algorithm with that of ext4: buddy-style. Various flags are passed to the allocator to control the size of allocation. Different applications may need different allocation size and different block placement, e.g. stream data and striped data have different requirements. In all the following operations, FOL log will be generated and logged, and these logs may help to do file system checking (fsck-ing).

- C2_mb_format. This routine creates database, group description tables, free space extent tables for container. Every container has a table called super_block, which contains container-wide information, such as block size, group size, etc. Every group has two tables: description table and free extent table. They are used to store group-wide information and its allocation states.
- c2_mb_init. This routine creates a working environment, reading information about the container and its groups from the data tables.
- c2_mb_allocate_blocks. This routine searches in groups to find best suitable free spaces. It uses the in-memory buddy system to help the searching. And then if free space is allocated successfully, updates to the group description and free space tables are done within the same transaction.
- c2_mb_free_blocks. This routine updates the in-memory buddy system, and then update the group description and free space tables to reflect these changes. Sanity checking against double free will be done here.
- c2_mb_enforce. This routine is used by fsck or other tools to modify block allocation status forcibly.

Comparison of C2 data-block-allocator and Ext4 multi-block allocator:

	Ext4 Multi-block Allocator	C2 data-block-allocator
on-disk free block tracking	bitmap	extent
in-memory free block tracking	buddy	buddy with extent
block allocation	multi-block buddy	multi-block buddy
pre-allocation	per-inode, per-group	per-object, per-group
cache	bitmap, buddy all cached	limited cache

These metadata for the free space tracking and space statistics are stored in database, while database themselves are stored in regular files. These files are stored in some metadata containers. The high availability, reliability and integrity of these database files rely on these metadata containers. The metadata containers usually are striped over multiple devices, with parity protection. These databases may also use replication technology to improve data availability.

5.1. Conformance

- Every group has its own group description and free space extent table. Locks have group granularity. This reduces lock contention, and therefore leads to good performance.

- Free space is represented in extent. This is efficient in most cases.
- Update to the allocation status is protected by database transactions. This insures the data-block-allocator survive from node crash.
- Operations of the allocator is logged by FOL. This log can be used by other components, i.e. fsck.

5.2. Dependencies

Some dependencies on container. But simulation of simple container will be used to avoid this.

5.3. Security model

N/A

5.4. Refinement

N/A

6. State

6.1. States, events, transitions

Every block is either allocated, or free. Tracking of free space is covered by this component. Tracking is allocated block is managed by object block mapping. That is another component. Blocks can be allocated from container. Blocks can also be freed from objects.
Allocated blocks and free blocks should be consistent. They should cover the whole container space, without any intersections. This will be checked by fsck-like tools in Colibri Core.
Allocation databases are usually replicated, so that this can improve the metadata integrity.

6.2. State invariants

N/A

6.3. Concurrency control

Concurrent read access to group description and free space extents are permitted. Write (update) access should be serialized.
Concurrent read/write access to different group description and free space extents are permitted. This enables parallel allocation in SMP systems.

7. Use cases

7.1. Scenarios

[This sub-section enumerates important use cases (to be later used as seed scenarios for ARID) and describes them in terms of logical specification.]

Scenario	[usecase.data-block-allocator.format]
Relevant quality attributes	
Stimulus	Initialize a container

Stimulus source	User/Admin
Environment	Container
Artifact	Fully formatted container, ready for use
Response	Initialize the metadata in the db
Response measure	Container is in its initial status, ready for use
Questions and issues	

Scenario	[usecase.data-block-allocator.init]
Relevant quality attributes	
Stimulus	Container init/startup
Stimulus source	System bootup, user/admin start the container services
Environment	Container
Artifact	Working environment
Response	Setup the working environment, including db handle, buddy information, etc.
Response measure	All data structures are properly setup
Questions and issues	

Scenario	[usecase.data-block-allocator.allocate]
Relevant quality attributes	concurrent, scalability should be good
Stimulus	object write or truncate
Stimulus source	object
Environment	container
Artifact	blocks allocated to object
Response	free blocks becomes allocated. Free space tables updated.
Response measure	correct extent updated to reflect this allocation
Questions and issues	

Scenario	[usecase.data-block-allocator.free]
Relevant quality attributes	concurrent, scalability
Stimulus	object delete, truncate
Stimulus source	object
Environment	container
Artifact	allocated blocks become free, usable again by other objects
Response	mark blocks as free, add them into free space tables.
Response measure	correctly update the free space tables. sanity check passed.
Questions and issues	

Scenario	[usecase.data-block-allocator.recovery]
Relevant quality attributes	fault tolerance
Stimulus	node failure
Stimulus source	power down accidentally, software bugs
Environment	container
Artifact	free space and allocated space are consistent
Response	recover the database and object metadata within same transaction
Response measure	consistent space.
Questions and issues	

Scenario	[usecase.data-block-allocator.fscking]
Relevant quality attributes	fault tolerance
Stimulus	consistency checking
Stimulus source	user/admin
Environment	container
Artifact	consistent container
Response	fix issues found in the checking
Response measure	container is consistent or not
Questions and issues	

Scenario	[usecase.data-block-allocator.fixup]
Relevant quality attributes	fault tolerance
Stimulus	consistency fixup
Stimulus source	user/admin
Environment	container, data or metadata
Artifact	unusable container fixed and usable again
Response	prevent to mount before fix. Fixes are: delete those objects who are using free space, or mark free space as used.
Response measure	fix should remove this inconsistency
Questions and issues	

7.2. Failures

Failures cases need recovery or fsck. This is described in the above use cases.

8. Analysis

8.1. Scalability

Lock per group enables concurrent access to the free space extent tables and description tables. This improves scalability.

8.2. Other

N/A

8.2. Rationale

N/A

9. Deployment

9.1. Compatibility

9.1.1. Network

9.1.2. Persistent storage

9.1.3. Core

9.2. Installation

10. References

[0]Ext4 multi-block allocator

11. Inspection process data

11.1. Logt

	Task	Phase	Part	Date	Planned time (min.)	Actual time (min.)	Comments

11.2. Logd

No.	Task	Summary	Reported by	Date reported	Comments
1					
2					
3					
4					

5					
6					
7					
8					
9					