Key Value Framework Programming Howto

Version 1.0

Dec 30,2014

Please send comments to luoqingchao@huawei.com

Revision History

Date	Revision	Change Request	Description	Authors
2014-12-30	1.0	None	Initial Version	luoqingchao@huawei.com thunder.zhang@huawei.com

Preface

Documents Introduction

The spec describes architecture and interface of Key Value Framework.

Catalogue

List of abbreviations

Abbreviations	Full spelling
SCSI	Small Computer System Interface
LBA	Logical Block Address
KVS	Key Value Storage
KVF	Key Value Functions
KV-LIB	Key Value Library
VFS	Virtual File System
DHT	Distributed Hash Table
LLM API	Lower Layer Manage API
ULA API	Upper Layer Access API

1 Introduction

1.1 Overview

This section describes basic Key Value Framework (KVF) concept and the overview, and the interface definition. Through KVF it defines unified data model, function, and parameters.

This Framework allows different Key Value Storage (KVS, also known as Key Value Store) regisgter and unregister, like Virtual File System (VFS) manages ext3, jfs, brtfs, etc.

And this framework provides unified API for upper application to call, to avoid write multiple adapters for different KVS, like call adapter for different vendors KVSs.

1.2 Key Value Storage Introduction

Traditional storage is based on SCSI architecture, clients uses Logical Block Address (LBA) to read and write data. Upon block provides more services, such as file system, data base, etc.

As an alternative, Key Value Storage (also known as Object storage system) proveide Key-Vaule pair operation. Applicatin store value by specify key name, and key name maybe a variable string, also value is a variable data buffer. Then application is easy to store data, need no complicated layout design based on linear block space, only focuses on key name policy design and how to store key (like use Distributed Hashb Table to store it) . Upon KVS, it's easy to build file, swift, hds, no-sql, even block service, shown as Figure_1.

Actually, 512 or 4096 byte based block device is a simple kind KVS, its key name is LBA and its value is 512 or 4096 data. Since its simplicity, application needs complicated layout design; and variable key value store is smarter, so application will be easy to store data on it. This is a kind of trade-off.

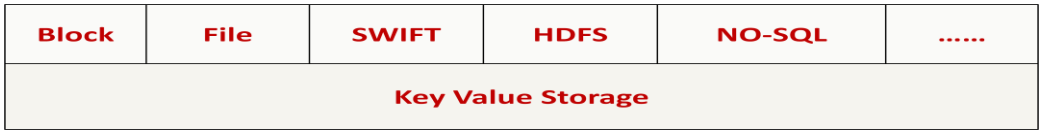

Figure_1 Key Value Storage Application Scenario

Based on the redundancy, there are three kinds of KVS:

- Standalone KVS

Standalone KVS specifies an application scenario where application client read and write date directly from one standalone device (such as single hard disk, single flash card, etc). Related products include Huawei UDS Smart Disk, HGST libzbc with KV, Seagate Kinetic and SanDisk Fusionio etc.

- Distributed KVS

  Distribute KVS specifies an application scenario where data has redundancy among servers and racks in one data center, like replica / erasure code. Related products include Ceph RADOS, mechcached, mongoDB, Huawei UDS SoD (Sea of Disks), etc.

- Multiple Data Center KVS

  Multiple Data Center KVS specifies an application scenario where data has redundancy among multiple data center, also has replica / erasure code feature. Related products include amazon S3, SWIFT, etc.

  Three kinds of KVS are connected, and CAN be mutual dependant. Like Multiple Data Center KVS may be based on Distributed KVS, and Distributed KVS may be based on Standalone KVS.

1.3 Key Value Framework (KVF) Data Model

Based on Key Value Framework (KVF), different vendors can register its Key Value Library (KV-LIB) to manage it Key Value Storage (KVS). These KV-LIBs should follow KVF data model, KVS provides pools, and pool provides objects. And pool supports nesting design, which means one pool can be on top of another pool, Shown as Figure_2.

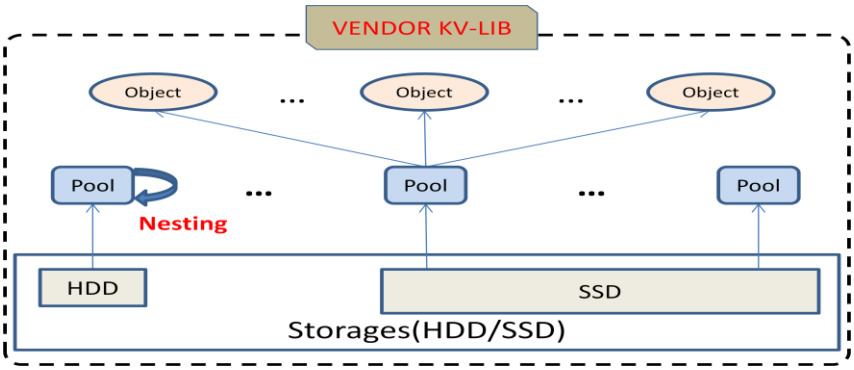

Figure_2 Key Value Framework Data Model

1.4 Key Value Framwork Architecture

Key Value Framework (KVF) provides 2 layer interfaces: Lower Layer Manage (LLM) API and Upper Layer Access (ULA) API, shown as Figure_3.

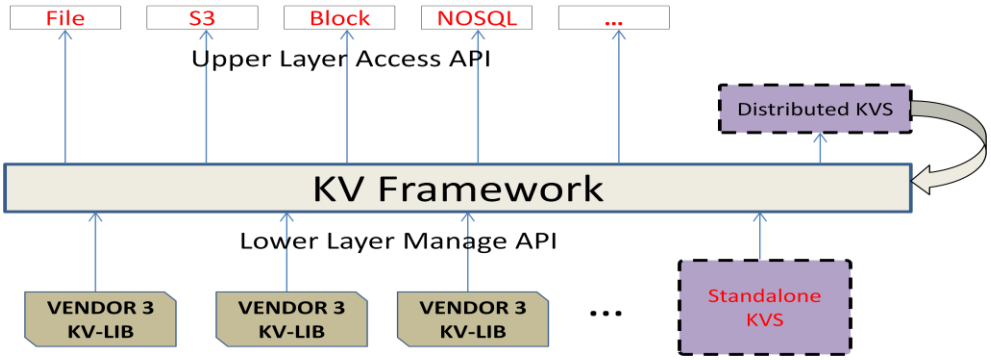

Figure_3 Key Value Framework Architecture

LLM(Lower Level Manage) provides register and unregister functions, and has 3 kinds of API: Object, Pool, and KV-LIB(Key Value Libraray).

ULA(Upper Layer Access) provides unified key value interface, then it makes application program simple and compatible.

Also some applications may provide key value service by leveraging ULA interface based on other KVS, and then it can be re-registered into key value framework, this make nesting design easily.

2 KVF Lower Layer Manage API

2.1 Overview

This section describes the LLM (Lower Layer Manage) APIs, includes KVF interface, Pool interface, Object interface.

KVF function starts with a “KVF_” header; different companies can replace the header of KVF function. E.g. Huawei can change to “Huawei_”; Vendor1 can change to “Vendor1_”, in this way to efficiently identify the function supplier.

2.2 Basic Data Structure

2.2.1 Basic Data Type

Original Type	New Defined Name
signed char	s8
unsigned char	u8
signed short	s16
unsigned short	u16
signed int	s32
unsigned int	u32
signed long long	s64
unsigned long long	u64

Original Type	New Defined Name
unsigned int	size_t

2.2.2 String Type

Type	Structure Definition	Description
String	typedef struct string { u32 len; s8* data; }string_t;	len: the length of the string data: a pointer which points to the string(UTP-8)

More data structure definition, please take look of the code.

2.3 KVF register/unregister API

These 2 API let vendors register its own KVS into the framework or unregister, API related parameter definition please take look of the code.

2.3.1 kvf_register()

Register a vendor specific Key Value Library into KVF, implement must.

s32 kvf_register(kvf_type_t * kvf)

2.3.2 kvf_unregister()

Unregister a vendor specific Key Value Library into KVFS, implement must.

s32 kvf_unregister(kvf_type_t * kvf)

2.4 KV-LIB APIs

KV-LIB is vendor specific key value library, through which manages the key value storage, following is the KV-LIB related APIs, please take look of the code for detail parameter definition.

2.4.1 kvf_init()

Callback to let vendor specific key value library init, implement recommended.

```
s32 kvf_init(const char * config_file)
```

2.4.2 kvf_shutdown ()

Callback to let vendor specific key value library init, implement recommended.

```
s32 kvf_shutdown()
```

2.4.3 kvf_set_prop ()

Callback to let vendor specific key value library set property, implement optional.

```
s32 kvf_set_prop(const char* name, const char* value)
```

2.4.4 kvf_get_prop ()

Callback to let vendor specific key value library get property, implement optional.

```
s32 kvf_get_prop(const char* name, char** value)
```

2.4.5 kvf_alloc_buf ()

Callback to let vendor specific key value library allocate buffer, for zero copy purpose, implement optional.

```
void* kvf_alloc_buf (size_t size, s32 flag)
```

2.4.6 kvf_free_buf ()

Callback to let vendor specific key value library free buffer, for zero copy purpose, implement optional.

```
void kvf_free_buf (void** buf)
```

2.4.7 kvf_get_errstr ()

Callback to let vendor specific key value library return specific error string information, implement optional.

```
const char* kvf_get_errstr (s32 err_code)
```

2.4.8 kvf_get_stats ()

Callback to let vendor specific key value library get specific statistics information, implement optional.

```
s32 kvf_get_stats (kvf_stats_t* kvfstats)
```

2.4.9 kvf_trans_start ()

Callback to let vendor specific key value library start a transaction, implement optional.

s32 kvf_trans_start(kv_trans_id_t ** t_id)

2.4.10 kvf_trans_commit ()

Callback to let vendor specific key value library commit a transaction, implement optional.

s32 kvf_trans_commit(kvf_trans_id_t* t_id)

2.4.11 kvf_trans_abort ()

Callback to let vendor specific key value library abort a transaction, implement optional.

s32 kvf_trans_abort(kvf_trans_id_t* t_id)

2.5 POOL API

Pool can be manged after KV-LIB registeredand initiated; following is the pool related APIs, please take look of the code for detail parameter definition.

2.5.1 pool_create ()

Callback to create pool, implement must.

s32 pool_create (const char* name, const char* config_path, pool_t * pool)

2.5.2 pool_destroy ()

Callback to destroy pool, implement must.

s32 pool_destroy (pool_t * pool)

2.5.3 pool_open ()

Callback to open pool, implement must.

s32 pool_open (pool_t * pool)

2.5.4 pool_close ()

Callback to close pool, implement must.

s32 pool_close (pool_t* pool)

2.5.5 pool_set_prop ()

Callback to set property of specific pool, implement optional.

s32 pool_set_prop(const pool_t* pool, const char* name, const char* value)

2.5.6 pool_get_prop ()

Callback to get property of specific pool, implement optional.

s32 pool_get_prop(const pool_t* pool, const char* name, char** value)

2.5.7 pool_get_stats ()

Callback to get statistics information of pool, implement optional.

s32 pool_get_stats (pool_stats_t * stats)

2.6 Object KV interface

After a pool is created successfully, we can operate the objects that belong to the pool. Following is the Object related APIs, please take look of the code for detail parameter definition.

2.6.1 put ()

Callback to put an object, implement must.

s32 put(pool_t* pool,const string_t* key, const string_t* value,const kv_props_t* props, const put_options_t* putopts);

2.6.2 get ()

Callback to get an object, implement must.

s32 get(pool_t* pool, const string_t* key, string_t* value, const kv_props_t* props, const get_options_t* getopts)

2.6.3 del ()

Callback to delete an object, implement must.

s32 del(pool_t* pool, const string_t* key, const kv_props_t* props, const del_options_t* delopts)

2.6.4 mput ()

Callback to put multiple objects, implement optinal.

s32 mput(pool_t* pool, kv_array_t* kvarray, const kv_props_t* props, const put_options_t* putopts)

2.6.5 mget ()

Callback to get multiple objects, implement optinal.

s32 mget(pool_t* pool, kv_array_t* kvarray, const kv_props_t* props, const get_options_t* getopts)

2.6.6 mdel ()

Callback to delete multiple objects, implement optinal.

s32 kv_mdel(pool_t* pool, array_t* kvarray, const kv_props_t* props, const del_options_t* delopts)

2.6.7 iter_open ()

Callback to open an iterator with specific regular expression, implement optinal.

s32 iter_open(const pool_t* pool, const string_t* key_regex, s32 limit, s32 timeout, kv_iter_t* it)

2.6.8 iter_next ()

Callback to get next object based on specific iterator, implement optinal.

s32 iter_next(pool_t* pool, kv_iter_t* it, kv_array_t* kvarray)

2.6.9 iter_close ()

Callback to close an iterator, implement optinal.

s32 iter_close(pool_t* pool, kv_iter_t* it)

2.6.10 xcopy ()

Callback to let one KVS copy KV pairs to another KVS by specific regular expression, to offload upper layer data copy, implement optional.

s32 xcopy(const pool_t* src, const pool_t* dest, const string_t* regex)

2.7 KV-LIB example

Example to show how to register and unregister.

2.7.1 Register vendor KV-LIB

```
pool_operations_t xxx_pool_ops = {
        .create = xxx_pool_create;
        .destroy = xxx_pool_destroy;
        .open = xxx_pool_open;
        .close = NULL;
        .set_prop = NULL;
        .get_prop = NULL;
        .get_stats = NULL;
};
kvf_operations_t xxx_kvlib_ops = {
        .init = xxx_kvlib_init;
        .shutdown = xxxs_kvlib_shutdown;
        .set_prop = NULL;
        .get_prop = NULL;
        .alloc_buf = NULL;
        .free_buf = NULL;
        .get_errstr = NULL;
        .get_stats = NULL;
        .trans_start = NULL;
        .trans_commit = NULL;
        .trans_abort = NULL;
};
kvf_type_t xxx_kv_lib = {

        .magic = KVF_MAGIC_XXX;
        .flag = 0;
        .name = "xxx_kvlib";
        .kvf_ops = xxx_kvlib_ops;
        .pool_ops = xxx_pool_ops;
};
int xxx_init(void)
{
        return kvf_register(xxx_kv_lib);
}
```

During xxx_pool_create() funciont, specify related kv_operations_t.

```
kv_operations_t xxx_kv_ops = {
        .put = xxx_kv_put;
        .get = xxxs_kv_get;
```

```
    .del = NULL;
    .mput = NULL;
    .mget = NULL;
    .mdel = NULL;
    .iter_open = NULL;
    .iter_next = NULL;
    .iter_close = NULL;
    .inc_ref = NULL;
    .dec_ref = NULL;
    .get_ref = NULL;
    .xcopy = NULL;
};

s32 xxx_pool_create(const char* name, const char* config_path, pool_t* pool)
{
    s32 ret;
    xxxkvsdev_t *kvsdevice;

    //1. Call vendor kv lib to create pool
    ret = xxxkvsdev_create(kvsdevice);
    if(!ret)
    {
        printf("create xxx pool fail\n");
        return ret;
    }

    //2. Init pool data structure
    pool->pool_private = kvsdevice;
    pool->pool_name = name;
    pool->kvf = xxx_kv_lib;
    pool->pool_location->pool_loc_type = PL_LOCATION_DEV;
    pool->pool_location->pool_loc.loc_dev = 0x0803;       //According to config_path get the dev_no
    pool->pool_physical_capacity = 0x10000;     ////According to dev_no get cap
    pool->pool_physical_used = 0;
    pool->pool_physical_free = pool->pool_physical_capacity;
    pool->pool_capacity = pool->pool_physical_capacity;
    pool->pool_used = 0;
    pool->pool_free = pool->pool_capacity;
    pool->pool_availability->pool_redundancy_type = PL_REDUNDANCY_SINGLE;
    pool->pool_latency = 100; //100ms according to dev desgin.
    pool->pool_throughput = 60*1024*1024; //60MB/s according to dev desgin.
    pool->pool_obj_cksum_type = 0;
    pool->pool_obj_cksum_lengh = 0x10;
    pool->pool_obj_compress_type = 0;
    pool->kv_ops = xxx_kv_ops;

    //3. write to configure file
    ret = kvf_write_pool_conf(pool);
    if(!ret)
    {
        printf("write pool to configure file fail\n");
        return ret;
```

```
        }
        return KV_OK;
}
```

2.7.2 Unregister vendor KV-LIB

```
init hgst_exit(void)
{
        return kvf_unregister(hgst_kv_lib);
}
```

3 KVF Upper Layer Access API

3.1 Overview

This section describes the Upper Layer Access API, includes KVF interface, Pool interface, Object interface also.

Since this layer API's purpose is to unify the interface, so most APIs just adjust the function name, please refer chapter 2 for more detail of definition.

3.2 Access KV-LIB APIs

3.2.1 get_kvf()

Get specific KV-LIB handle based on the name.

s32 kvf_init(const char * config_file) kvf_type_t* get_kvf(const char* name)

3.2.2 init_kvf()

Callback to let vendor specific key value library init, implement recommended.

s32 init_kvf(const char * config_file)

3.2.3 shutdown_kvf ()

Callback to let vendor specific key value library init, implement recommended.

s32 shutdown_kvf()

3.2.4 set_kvf_ prop ()

Callback to let vendor specific key value library set property, implement optional.

s32 set_kvf_ prop(const char* name, const char* value)

3.2.5 get_kvf_ prop ()

Callback to let vendor specific key value library get property, implement optional.

s32 get_kvf_ prop(const char* name, char** value)

3.2.6 alloc_kvf_ buf ()

Callback to let vendor specific key value library allocate buffer, for zero copy purpose, implement optional.

void* alloc_kvf_ buf (size_t size, s32 flag)

3.2.7 free_kvf_ buf ()

Callback to let vendor specific key value library free buffer, for zero copy purpose, implement optional.

void free_kvf_ buf (void** buf)

3.2.8 get_kvf_ errstr ()

Callback to let vendor specific key value library return specific error string information, implement optional.

const char* get_kvf_ errstr (s32 err_code)

3.2.9 get_kvf_ stats ()

Callback to let vendor specific key value library get specific statistics information, implement optional.

s32 get_kvf_ stats (kvf_stats_t* kvfstats)

3.2.10 start_ kvf_trans ()

Callback to let vendor specific key value library start a transaction, implement optional.

s32 start_kvf_trans (kv_trans_id_t ** t_id)

3.2.11 commit_kvf_trans ()

Callback to let vendor specific key value library commit a transaction, implement optional.

s32 commit_kvf_trans (kvf_trans_id_t* t_id)

3.2.12 abort_kvf_trans ()

Callback to let vendor specific key value library abort a transaction, implement optional.

s32 abort_kvf_trans (kvf_trans_id_t* t_id)

3.3 Access POOL API

3.3.1 create_pool ()

Callback to create pool, implement must.

s32 create_pool (const char* name, const char* config_path, pool_t * pool)

3.3.2 destroy_pool ()

Callback to destroy pool, implement must.

s32 destroy_pool (pool_t * pool)

3.3.3 open_pool ()

Callback to open pool, implement must.

s32 open_pool (pool_t * pool)

3.3.4 close_pool ()

Callback to close pool, implement must.

s32 close_pool (pool_t* pool)

3.3.5 set_pool_ prop ()

Callback to set property of specific pool, implement optional.

s32 set_pool_prop(const pool_t* pool, const char* name, const char* value)

3.3.6 get_pool_ prop ()

Callback to get property of specific pool, implement optional.

s32 get_pool_prop(const pool_t* pool, const char* name, char** value)

3.3.7 get_pool_stats ()

Callback to get statistics information of pool, implement optional.

s32 get_pool_stats (pool_stats_t * stats)

3.4 Access Object KV interface

3.4.1 put ()

Callback to put an object, implement must.

s32 put(pool_t* pool,const string_t* key, const string_t* value,const kv_props_t* props, const put_options_t* putopts);

3.4.2 get ()

Callback to get an object, implement must.

s32 get(pool_t* pool, const string_t* key, string_t* value, const kv_props_t* props, const get_options_t* getopts)

3.4.3 del ()

Callback to delete an object, implement must.

s32 del(pool_t* pool, const string_t* key, const kv_props_t* props, const del_options_t* delopts)

3.4.4 mput ()

Callback to put multiple objects, implement optinal.

s32 mput(pool_t* pool, kv_array_t* kvarray, const kv_props_t* props, const put_options_t* putopts)

3.4.5 mget ()

Callback to get multiple objects, implement optinal.

s32 mget(pool_t* pool, kv_array_t* kvarray, const kv_props_t* props, const get_options_t* getopts)

3.4.6 mdel ()

Callback to delete multiple objects, implement optinal.

s32 mdel(pool_t* pool, array_t* kvarray, const kv_props_t* props, const del_options_t* delopts)

3.4.7 open_iter ()

Callback to open an iterator with specific regular expression, implement optinal.

s32 open_iter (const pool_t* pool, const string_t* key_regex, s32 limit, s32 timeout, kv_iter_t* it)

3.4.8 next_iter ()

Callback to get next object based on specific iterator, implement optinal.

s32 next_iter (pool_t* pool, kv_iter_t* it, kv_array_t* kvarray)

3.4.9 close_iter ()

Callback to close an iterator, implement optinal.

s32 close_iter (pool_t* pool, kv_iter_t* it)

3.4.10 xcopy_obj()

Callback to let one KVS copy KV pairs to another KVS by specific regular expression, to offload upper layer data copy, implement optional.

s32 xcopy_obj(const pool_t* src, const pool_t* dest, const string_t* regex)

3.5 Application Access Example

```
#include "kvf-api.h"

int main()
{
    kvf_type_t t;
    char *confile;
    char prop_value[64];
    pool_t pl;
    string_t key, value;
    kv_props_t kvprops;

    //1. Get kv-lib handle and init.
    get_kvf("xxx_kvlib", &t);
    init_kvf(t, "/etc/xxx_kvlib-conf");

    //2. Create Pool and open it.
    create_pool(t, "test", confile, &pl);
    open_pool(&pl);
```

```
    //3. Put key, get key and delete key.
    key.data = "hello"
    key.len = 5;
    value.data="0123456789";
    value.len = 10;

    put (&pl, key, &value, kvprops, NULL);
    get (&pl, key, &value, kvprops, NULL);
    del (&pl, key, kvprops, NULL);

    //4. Close pool and destroy pool.
    close_pool(&pl);
    destroy_pool(&pl);

    //5. Shutdown xxx_kvlib by this application
    shutdown_kvf(t);

    return 0;
}
```

4 Appendix

4.1 Reference

A. http://docs.ceph.com/docs/v0.80.5/rados/api/librados/#api-calls

B. https://developers.seagate.com/display/KV/Client+Library+APIs+and+Simulator

C. http://opennvm.github.io/nvmkv-documents/#nvmkv api.htm

D. https://github.com/hgst/libzbc

E. https://github.com/Seagate/kinetic-c

F. https://github.com/westerndigitalcorporation/SMR-Simulator