Erasure Coding NIC Offload

API and design document

Modification History

0.1	Liran Oz		Initial revision

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to define the API and high level design for Mellanox Erasure coding Offload library used for Erasure Coding and RAID HW offload (Based on Erasure Coding Verbs API).

1.2 Definitions, Acronyms and abbreviations

RAID	Redundant array of inexpensive disks
EC	Erasure Coding
Verbs	API for applications working over an RDMA provider
API	Application Programmable Interface
HCA	Host Channel Adapter
HW	Hardware
SW	Software
IBV	RDMA verbs – common API prefix in verbs
IBV_MR element	Memory region object associated with the protection domain
IBV_SGE element	Verbs Scatter-Gather element {memory key, address, length}
PCI	Peripheral Component Interconnect Express
K	Denotes number of data blocks
M	Denotes number of code/parity blocks
W	Denotes Galois Field GF(2^w) symbol size (in bits)
HDFS	Hadoop Distributed File System

1.3 Reference

Erasure coding Verbs API	http://portalx.mellanox.com/sites/SW/Storage/General/Erasure%20Coding/Erasure%20Coding%20Offload%20Experimental%20Verbs%20API%20v1.0.2.pdf
Hadoop erasure coding	http://issues.apache.org/jira/browse/HDFS-7285 http://blog.cloudera.com/blog/2015/09/introduction-to-hdfs-erasure-coding-in-apache-hadoop/ http://aajisaka.github.io/hadoop-project/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html
Jerasure SW library	http://github.com/tsuraan/Jerasure http://web.eecs.utk.edu/~plank/plank/www/software.html

1.4 Dependencies and Prerequisites

1.4.1 Hardware

Mellanox ConnectX®-4 or ConnectX®-4 Lx.

1.4.2 Software

- MLNX_OFED v3.3-1.0.0.0 (or later).
- Firmware - v12.16.1006 for ConnectX®-4 (or later).
  v14.16. 1006 for ConnectX®-4 Lx (or later).
- Jerasure library v2.0.

1.5 Recommendations

- install Jerasure & GF-Complete using the following parameters:

```
./configure --prefix=/usr/ --libdir=/usr/lib64/
```

- It is highly recommended to install ConnectX®-4 on PCIe3.0 x16, and ConnectX®-4 Lx on PCIe3.0 x8 slot for better performance.
- Use block size aligned to 64 byte to avoid copying to remainder to internal buffers.

1.6 Limitations

- Thread safety - Single thread per encoder/decoder.
- Using mlx5_0 device as default.

2. Overview

Erasure coding (EC) is a method of data protection in which data is broken into fragments, expanded and encoded with redundant data pieces and stored across a set of different locations or storage media.

The goal of erasure coding is to enable data that becomes corrupted at some point in the disk storage process to be reconstructed by using information about the data that's stored elsewhere in the array. The drawback of erasure coding is that it can be more CPU-intensive, and that can translate into increased latency.

The simplest example is based on XOR, Assume we want to store two bits (X, Y), we can generate one parity bit $Z = X \oplus Y$ instead of copy both bits. When each of the bit is lost, it can be recovered using XOR on the other two bits.
Compered to naive approach (replication), we have the same data durability, half the storage overhead. But, as a tradeoff the recovery from a failure is slower.

As we mentioned above, data encoding/decoding is very CPU intensive and can be a major overhead when using Erasure coding. By using Mellanox EC Offload library, the calculation is done by the HCA which reduce dramatically the CPU consumption.

3. Common

3.1 Role

Describe the common methods and structures between the encoder and decoder.

3.2 Design

3.2.1 common structures

3.2.1.1 eco_context

```
/**
* Erasure Coding Offload context structure.
*
* @calc                  Verbs erasure coding engine context.
* @attr                  Verbs erasure coding engine initialization attributes.
* @alignment_mem         Verbs erasure coding memory layout context used for 64 bytes aligned buffers.
* @mrs_list              Simple doubly linked list of lbv_mr objects.
* @int_encode_matrx      Used for Jerasure to calculate the decode matrix.
* @remainder_buffers     [k + m] buffers each of size 64 bytes used for the remainder from 64 bytes.
* @remainder_mem         Verbs erasure coding memory layout context used for the remainder from 64 bytes.
* @block_size            Size of the input blocks.
* @async_mutex           Mutex used for async encode/decode operations.
* @async_cond            Condition used for async encode/decode operations.
* @async_ref_count       Reference count used for async encode/decode operations.
* @data                  Array of pointers to source input buffers.
* @coding                Array of pointers to coded output buffers.
* @alignment_comp        Erasure Coding Offload completion context used for 64 bytes aligned buffers.
* @remainder_comp        Erasure Coding Offload completion context used for the remainder from 64 bytes.
*/

struct eco_context {
        struct ibv_exp_ec_calc                  *calc;
        struct ibv_exp_ec_calc_init_attr        attr;
        struct ibv_exp_ec_mem                   alignment_mem;
        eco_list                                mrs_list;
        int                                     *int_encode_matrix;
        uint8_t                                 *remainder_buffers;
        struct ibv_exp_ec_mem                   remainder_mem;
        struct ibv_mr                           *remainder_mr;
        int                                     block_size;
        pthread_mutex_t                         async_mutex;
        pthread_cond_t                          async_cond;
        int                                     async_ref_count;
        uint8_t                                 **data;
        uint8_t                                 **coding;
        struct eco_coder_comp                   alignment_comp;
        struct eco_coder_comp                   remainder_comp;
};
```

3.2.1.1 eco_coder_comp

```
/**
 * Erasure Coding Offload completion context. Used for async encode/decode operations.
 *
 * @comp                    Completion context of EC calculation.
 * @eco_coder               Pointer to an encoder/decoder.
 * @is_remainder_comp       Boolean variable which determine the type of the completion context.
 */
struct eco_coder_comp {
        struct ibv_exp_ec_comp          comp;
        void                            *eco_coder;
        int                             is_remainder_comp;
};
```

3.2.2 Methods

mlx_eco_init

```
/**
 * Initialize verbs EC context object used for fast Erasure Coding HW offload.
 *
 * @param coder                     Pointer to an encoder/decoder.
 * @param k                         Number of data blocks.
 * @param m                         Number of code blocks.
 * @param use_vandermonde_matrix    Boolean variable which determine the type of the encode matrix:
 *                                  0 for Cauchy coding matrix else for Vandermonde coding matrix.
 * @param comp_done_func            Function handle of the EC calculation completion.
 * @return                          Pointer to an initialize EC contex object if successful, else NULL.
 */
struct eco_context *mlx_eco_init(void *coder, int k, int m, int use_vandermonde_matrix, void (*comp_done_func)(struct ibv_exp_ec_comp *));
```

Method Flow:

I. Open device using *ibv_open_device* method & Allocate verbs PD using *ibv_alloc_pd* method.
II. Create Vandermode/Cauchy distribuiton matrix used for encode/decode:
    Cauchy coding matrix is generated using Jerasure's *cauchy_original_coding_matrix(k, m, w)* method.
    Vandermonde coding matrix is generated using Jerasure's *reed_sol_vandermonde_coding_matrix(k, m, w)* method.
III. Allocate and initailaze needed resources for eco_context.
IV. Allocate and initailaze the context's calc using *ibv_exp_alloc_ec_calc* method.

mlx_eco_register

```
/**
 * Register buffers and update the memory layout context for future encode/decode operations.
 * Because the HW can perform encode/decode operations only on 64 bytes aligned buffers,
 * We will register only the aligned part of the buffers.
 * This function is optional - but it is recommended to use for better performance.
 *
 * @param eco_context       Pointer to an initialized EC context.
 * @param data              Array of pointers to source input buffers.
 * @param coding            Array of pointers to coded output buffers.
 * @param data_size         Size of data array (must be equal to the initial amount of data blocks).
 * @param coding_size       Size of coding array (must be equal to the initial amount of code blocks).
 * @param block_size        Length of each block of data.
 * @return                  0 successful, other fail.
 */
int mlx_eco_register(struct eco_context *eco_ctx, uint8_t **data, uint8_t **coding, int data_size, int coding_size, int block_size);
```

Method Flow:

I. If *block_size* < 64 return. Else, do the following on the 64 bytes aligned part.
II. For each given buffer (data or coding):
    a. Check if the memory layout context contains the buffer.
    b. Else, check if the buffer was registered before (using the *mrs_list* object).
    c. Else, register the buffer, update the memory layout context and add it to the list.
III. At the end of the function, the memory layout context should contain the aligned part of the input buffers.

mlx_eco_release

```
/**
 * Release all EC context resources.
 *
 * @param eco_context       Pointer to an initialized EC context.
 * @return                  0 successful, other fail.
 */
int mlx_eco_release(struct eco_context *eco_ctx);
```

Method Flow:

I. Release all allocated memory.
II. Deallocate calc using *ibv_exp_dealloc_ec_calc* method.
III. Deallocate verbs PD using *ibv_dealloc_pd* method & close device using *ibv_close_device* method.
IV. Free eco_context object.

4. Encode

4.1 Role

Generates redundancy blocks of encoded data as specified by the coding matrix of GF (2^4) coefficients.

4.2 Design

4.2.1 eco_encoder structure

```
/**
* @eco_ctx                          Erasure Coding Offload context.
*/
struct eco_encoder {
        struct eco_context          *eco_ctx;
}
```

4.2.2 Methods

mlx_eco_encoder_init

```
/**
 * Initialize verbs EC encoder object used for fast Erasure Coding HW offload.
 *
 * @param k                         Number of data blocks.
 * @param m                         Number of code blocks.
 * @param use_vandermonde_matrix    Boolean variable which determine the type of the encode matrix:
 *                                  0 for Cauchy coding matrix else for Vandermonde coding matrix -
 * @return                          Pointer to an initialize EC encoder object if successful, else NULL.
 */
struct eco_encoder *mlx_eco_encoder_init(int k, int m, int use_vandermonde_matrix);
```

Method Flow:

I. Call *mlx_eco_init* method.

mlx_eco_encoder_register

```
/**
 * Register buffers and update the memory layout context for future encode/decode operations.
 * Because the HW can perform encode/decode operations only on 64 bytes aligned buffers,
 * We will register only the aligned part of the buffers.
 * This function is optional - but it is recommended to use for better performance.
 *
 * @param eco_encoder       Pointer to an initialized EC encoder.
 * @param data              Array of pointers to source input buffers.
 * @param coding            Array of pointers to coded output buffers.
 * @param data_size         Size of data array (must be equal to the initial amount of data blocks).
 * @param coding_size       Size of coding array (must be equal to the initial amount of code blocks).
 * @param block_size        Length of each block of data.
 * @return                  0 successful, other fail.
 */
int mlx_eco_encoder_register(struct eco_encoder *eco_encoder, uint8_t **data, uint8_t **coding, int data_size,
int coding_size, int block_size);
```

Method Flow:

I. Call *mlx_eco_register* method.

mlx_eco_encoder_encode

```
/**
 * Generates blocks of encoded data from the data buffers and store them in the coding array.
 * Using mlx_eco_encoder_register() if the buffers are not registered.
 *
 * @param eco_encoder       Pointer to an initialized EC encoder.
 * @param data              Array of pointers to source input buffers.
 * @param coding            Array of pointers to coded output buffers.
 * @param data_size         Size of data array (must be equal to the initial amount of data blocks).
 * @param coding_size       Size of coding array (must be equal to the initial amount of code blocks).
 * @param block_size        Length of each block of data.
 * @return                  0 successful, other fail.
 */
int mlx_eco_encoder_encode(struct eco_encoder *eco_encoder, uint8_t **data, uint8_t **coding, int data_size,
int coding_size, int block_size);
```

Method Flow:

I. Verify that memory layout context contains the input buffers (call *mlx_eco_register* if necessary).
II. Copy the 64 bytes remainder input data to internal buffers and perform encode using *ibv_exp_ec_encode_async method.*
III. Generates blocks of encoded data of the 64 bytes aligned part using *ibv_exp_ec_encode_async* method.
IV. Wait until both async methods will end (the remainder encoded data will be copy to the input buffers using the completion function).

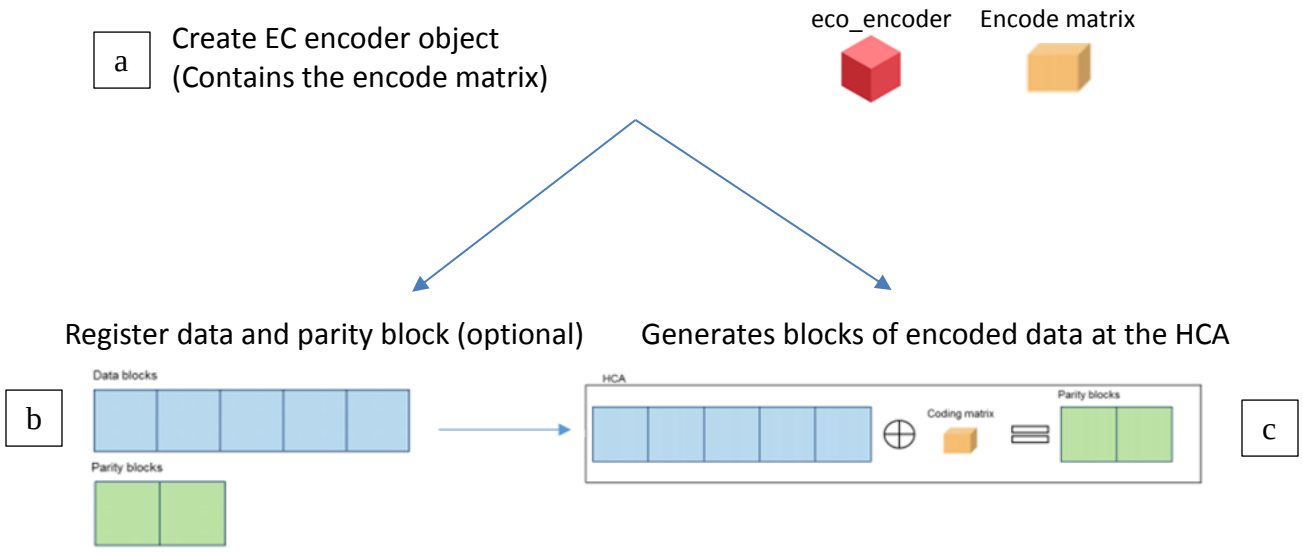
mlx_eco_encoder_release

```
/**
 * Release all EC encoder resources.
 *
 * @param eco_encoder          Pointer to an initialized EC encoder.
 * @return                     0 successful, other fail.
 */
int mlx_eco_encoder_release(struct eco_encoder *eco_encoder);
```

Method Flow:

I. Call *mlx_eco_relase* method.

4.3 Flow



a. Create EC encoder object using *mlx_eco_encoder_init* method**.**
b. [optionally] Register data and parity blocks using *mlx_eco_encoder_register* method.
c. Generates blocks of encoded data at the HCA using *mlx_eco_encoder_encode* method (call *mlx_eco_encoder_register* if neccesery).
d. Return to b/c for additional encode operations.
e. Release all resources by using *mlx_eco_encoder_release* method.

5. Decode

5.1 Role

Decode a given set of data blocks and code blocks and place into output recovery blocks.

5.2 Design

5.2.1 *eco_decoder structure*

```
/**
* @eco_ctx                Erasure Coding Offload context.
* @int_decode_matrix      Registered buffer [k * k] of the decode matrix used for Jerasure to calculate the
*                         decode matrix.
* @u8_decode_matrix       Registered buffer [k * m] of the decode matrix used decode.
* @int_erasures           Pointer to byte-map of which blocks were erased and needs to be recovered.
* @u8_erasures            Pointer to byte-map of which blocks were erased and needs to be recovered.
* @survived               Pointer to byte-map of which blocks were survived.
*/
struct eco_decoder {
        struct eco_context          *eco_ctx;
        int                         *int_decode_matrix;
        uint8_t                     *u8_decode_matrix;
        int                         *int_erasures;
        uint8_t                     *u8_erasures;
        int                         *survived;
};
```

5.2.2 Methods

mlx_eco_decoder_init

```
/**
 * Initialize verbs EC decoder object used for fast Erasure Coding HW offload.
 *
 * @param k                         Number of data blocks.
 * @param m                         Number of code blocks.
 * @param use_vandermonde_matrix    Boolean variable which determine the type of the encode matrix:
 *                                  0 for Cauchy coding matrix else for Vandermonde coding matrix -
 * @return                          Pointer to an initialize EC decoder object if successful, else NULL.
 */
struct eco_decoder *mlx_eco_decoder_init(int k, int m, int use_vandermonde_matrix);
```

Method Flow:

I. Register buffers for decode matrix and erasures byte map.
II. Call *mlx_eco_init* method.

mlx_eco_decoder_register

```
/**
 * Register buffers and update the memory layout context for future encode/decode operations.
 * Because the HW can perform encode/decode operations only on 64 bytes aligned buffers,
 * We will register only the aligned part of the buffers.
 * This function is optional - but it is recommended to use for better performance.
 *
 * @param eco_decoder       Pointer to an initialized EC decoder.
 * @param data              Array of pointers to source input buffers.
 * @param coding            Array of pointers to coded output buffers.
 * @param data_size         Size of data array (must be equal to the initial amount of data blocks).
 * @param coding_size       Size of coding array (must be equal to the initial amount of code blocks).
 * @param block_size        Length of each block of data.
 * @return                  0 successful, other fail.
 */
int mlx_eco_decoder_register(struct eco_decoder *eco_decoder, uint8_t **data, uint8_t **coding, int data_size,
int coding_size, int block_size);
```

Method Flow:

I. Call *mlx_eco_register* method.

mlx_eco_decoder_generate_decode_matrix

```
/**
 * Generate k*k decoding matrix by taking the rows corresponding to k non-erased devices of the
 * distribution matrix and store it in the decoder context.
 * This method should be call before first decoding operation or when there is a change in the erasures bit-map.
 * This function is optional - but it is recommended to use for better performance.
 *
 * @param eco_decoder       Pointer to an initialized EC decoder.
 * @param erasures          Pointer to byte-map of which blocks were erased and needs to be recovered.
 * @param erasures_size     Size of erasures bit-map.
 * @return                  0 successful, other fail.
 */
int mlx_eco_decoder_generate_decode_matrix(struct eco_decoder *eco_decoder, int *erasures, int
erasures_size);
```

Method Flow:

I. Verify if decode matrix calculation in needed.
II. If yes, generate docode matrix using *jerasure_make_decoding_matrix* method.

mlx_eco_decoder_decode

```
/**
 * Decode a given set of data blocks and code_blocks and place into output recovery blocks.
 * Using mlx_eco_decoder_generate_decode_matrix() if the decode matrix is not compatible with the erasures.
 * Using mlx_eco_decoder_register() if the buffers are not registered.
 *
 * @param eco_decoder       Pointer to an initialized EC decoder.
 * @param data              Array of pointers to source input buffers.
 * @param coding            Array of pointers to coded output buffers.
 * @param data_size         Size of data array (must be equal to the initial amount of data blocks).
 * @param coding_size       Size of coding array (must be equal to the initial amount of code blocks).
 * @param block_size        Length of each block of data.
 * @param erasures          Pointer to byte-map of which blocks were erased and needs to
 *                          be recovered.
 * @param erasures_size     Size of erasures bit-map.
 * @return                  0 successful, other fail.
 */
int mlx_eco_decoder_decode(struct eco_decoder *eco_decoder, uint8_t **data, uint8_t **coding, int data_size,
int coding_size, int block_size, int *erasures, int erasures_size);
```

Method Flow:

I. Verify if decode matrix calculation in needed (call *mlx_eco_decoder_generate_decode_matrix*).
II. Verify that memory layout context contains the input buffers (call *mlx_eco_register*).
III. Copy the 64 bytes remainder from the valid blocks to internal buffers and perform decode using *ibv_exp_ec_decode_async* method.
IV. Recover failed blocks of the 64 bytes aligned part using *ibv_exp_ec_decode_async* method.
V. Wait until both async methods will end (the remainder decoded data will be copy to the input buffers using the completion function).

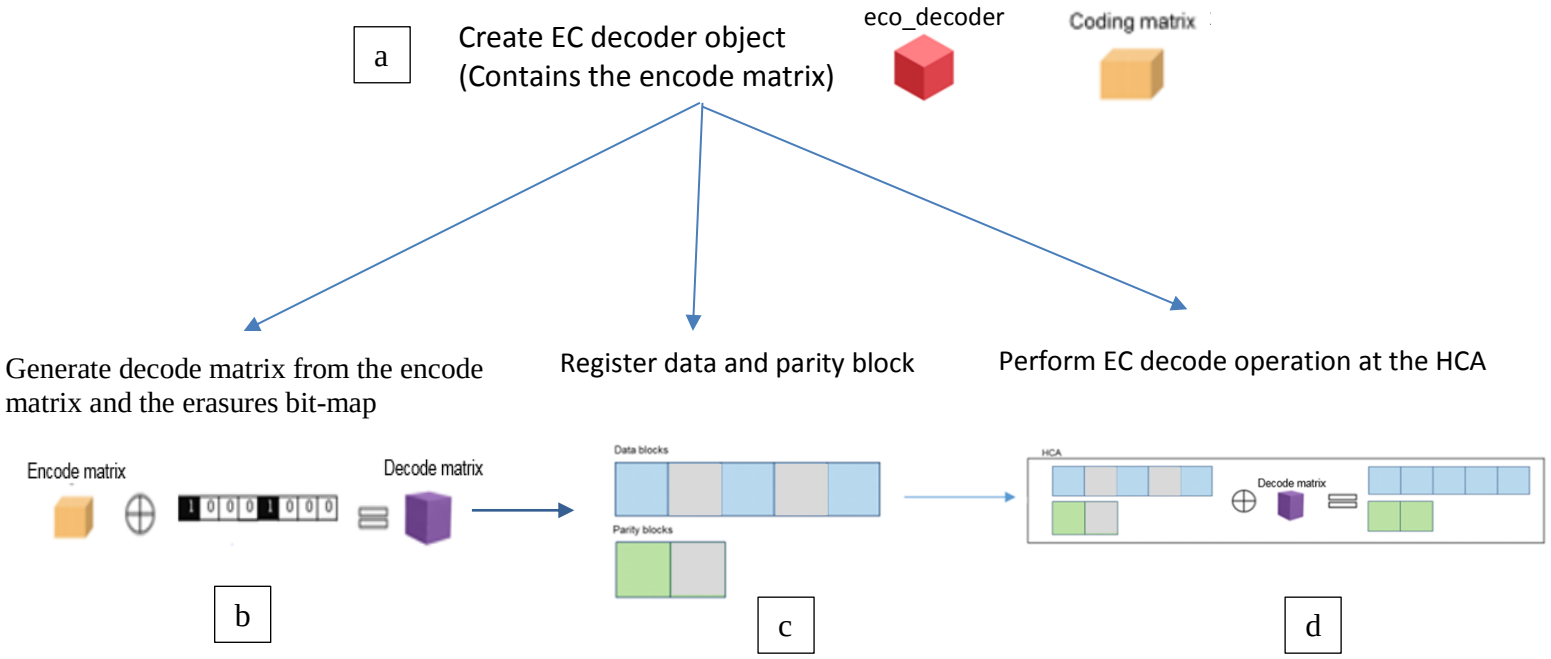
mlx_eco_decoder_release

```
/**
 * Release all EC decoder resources.
 *
 * @param eco_decoder       Pointer to an initialized EC decoder.
 * @return                  0 successful, other fail.
 */
int mlx_eco_decoder_release(struct eco_decoder *eco_decoder);
```

Method Flow:

I. Call *mlx_eco_relase* method.
II. Release the decode matrix and erasures byte map buffers.

5.3 Flow



a. Create EC decoder object using *mlx_eco_decoder_init* method**.**
b. [optionally] Generate decode matrix from the encode matrix and the erasures bit-map using *mlx_eco_decoder_generate_decode_matrix* method**.**
c. [optionally] Register data and parity blocks using *mlx_eco_decoder_register* method.
d. Decode the data and coding blocks at the HCA using *mlx_eco_decoder_decode* method (call *mlx_eco_decoder_generate_decode_matrix* and/or *mlx_eco_decoder_register* if neccesery).
e. For additional decoding operations:
    - Return to b if there is a change in the erasures bit-map.
    - Else, return to b/c/d.
f. Release all resources by using *mlx_eco_decoder_release* method.

6. Appendix a – HDFS Erasure Coding Offload Plugin

6.1 Overview

HDFS by defaults using replication to store its blocks, namely, each given block is replicated serval time (the default number is 3) and stored in the HDFS Datanodes. Replication provides a simple from to deal with most failure scenarios.
Using Erasure Coding, we can reduce the storage overhead by approximately 50% while maintaining the same data durability.
Erasure Coding operation such as encode and decode are very CPU intensive actions and can be a major overhead compared to the replication approach.
HDFS-EC is currently targeted for release in Hadoop 3.0 (Hadoop Road-map).
HDFS Erasure Coding Offload Plugin is based on Erasure Coding NIC Offload Library described above.

6.2 Purpose

The purpose of this appendix is to define the HDFS Erasure Coding plugin design and modules.

6.3 Prerequisites

1. Erasure Coding NIC Offload library.
2. Hadoop 3.0.0 based on commit 5b7078d.
   Including the following patches:
   a. HADOOP-11996-v3.patch (HADOOP-11996).
   b. HADOOP-11540-v2.patch (HADOOP-11540).
   c. HDFS-8668-v2.patch (HDFS-8668).

6.4 Limitations

1. Same as the library described above.
2. Decode also redundant blocks.

6.5 Installation

1. Clone the repository from Github.
2. cd HDFS
3. Install/Build

- Install

```
ant install -DJAVA_HOME=/path/to/java/home/dir -DHADOOP_HOME=/path/to/hadoop/home/dir
```

This will build and copy MellanoxECOffload.jar and libHdfsEcOffload.so to the preferable locations in your Hadoop home directory.

- Build

```
ant build -DJAVA_HOME=/path/to/java/home/dir -DHADOOP_HOME=/path/to/hadoop/home/dir
```

a. Place your MellanoxECOffload JAR file (located in build/jar/MellanoxECOffload.jar):
The preferable location is where all the Hadoop Common JARs are already found:

```
$HADOOP_HOME/share/hadoop/common/lib
```

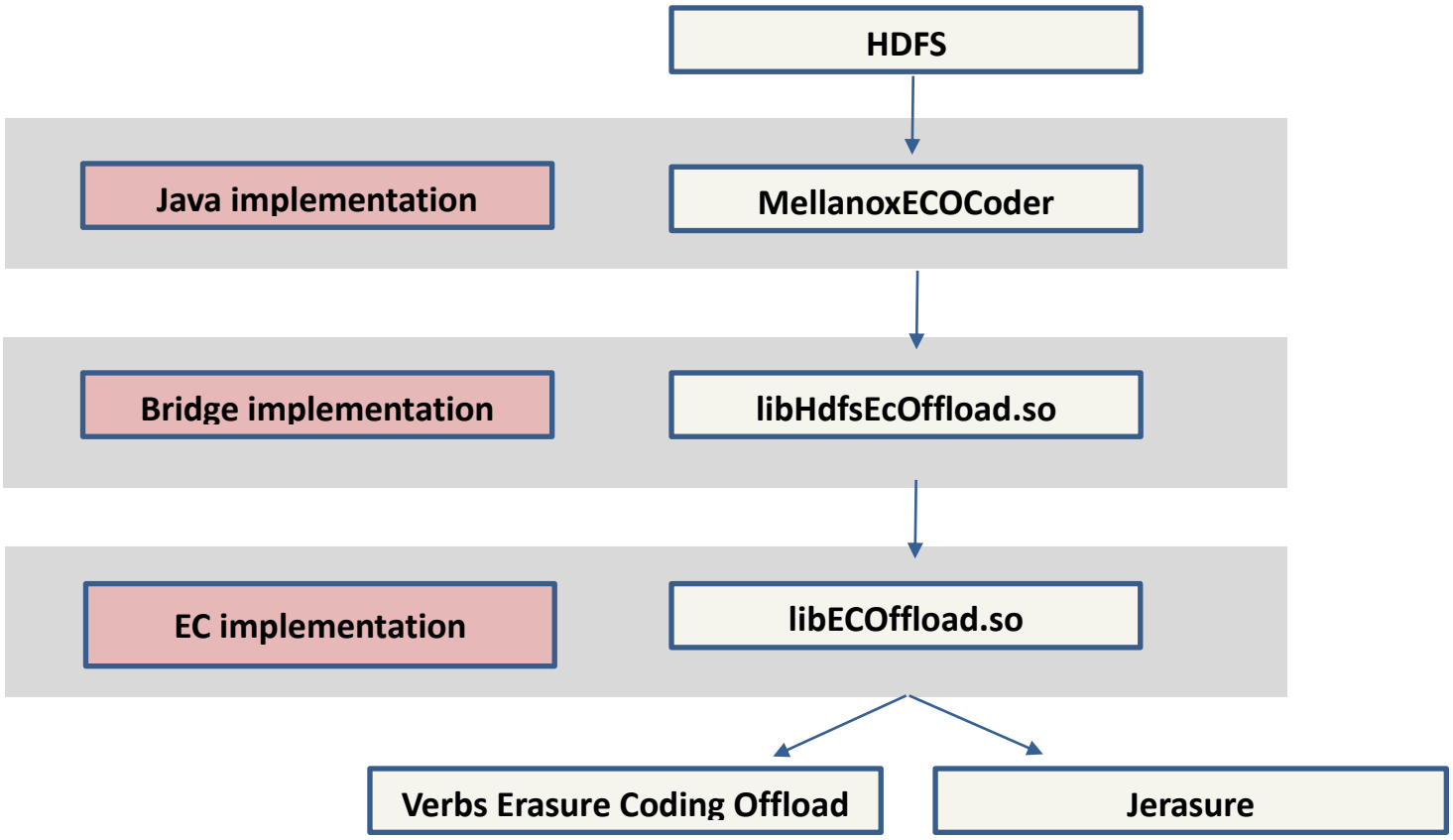
b. Place your libHdfsEcOffload.so file (located in build/lib/libHdfsEcOffload.so) :
The preferable location is where the Hadoop native libraries are already found:

```
$HADOOP_HOME/lib/native/
```

4. Configure HDFS Erasure Coding Offload plugin in hdfs-site.xml as follows:

```
<property>
<name>io.erasurecode.codec.rs.rawcoder</name>
<value>com.mellanox.erasurecode.rawcoder.MellanoxRSRawErasureCoderFactory</value>
</property>
```

6.6 High Level Design



6.7 Modules

6.7.1 MellanoxECOCoder

This module is responsible for the connection between the HDFS and the Erasure Coding Offload implementation. Written in Java.

MellanoxRSRawErasureCoderFactory

Role:
Creates Mellanox Erasure encoder/decoder used for Erasure Coding HCA Offload.

Methods:
public RawErasureEncoder createEncoder(int numDataUnits, int numParityUnits)
public RawErasureDecoder createDecoder(int numDataUnits, int numParityUnits)

MellanoxRSRawEncoder

Role:
Encode with inputs and generates outputs.

Methods:
void performEncodeImpl(ByteBuffer[] inputs, int[] inputOffsets, int dataLen, ByteBuffer[] outputs, int[] outputOffsets)
void release()

MellanoxRSRawDecoder

Role:
Decode given chunks of input data.

Methods:
void performDecodeImpl(ByteBuffer[] inputs, int[] inputOffsets, int dataLen, int[] erased, ByteBuffer[] outputs, int[] outputOffsets)
void release()

MellanoxECLibraryLoader

Role:
Load the Mellanox Erasure Coding Offload Library.

Methods:
static boolean isNativeCodeLoaded()
static void checkNativeCodeLoaded()
static String getLoadingFailureReason()

6.7.2 libHdfsEcOffload.so

This module is a bridge between Java to C. Written in C.

MellanoxRSRawEncoder.c

Role:
Perform encode operations using Mellanox EC Offload Library.

Methods:
void Java_com_mellanox_erasurecode_rawcoder_MellanoxRSRawEncoder_initImpl(JNIEnv *env, jobject thiz, jint numDataUnits, jint numParityUnits)
void Java_com_mellanox_erasurecode_rawcoder_MellanoxRSRawEncoder_encodeImpl(JNIEnv *env, jobject thiz, jobjectArray inputs, jintArray inputOffsets, jint dataLen, jobjectArray outputs, jintArray outputOffsets)
void Java_com_mellanox_erasurecode_rawcoder_MellanoxRSRawEncoder_destroyImpl(JNIEnv *env, jobject thiz)

MellanoxRSRawDecoder.c

Role:
Perform decode operations using Mellanox EC Offload Library.

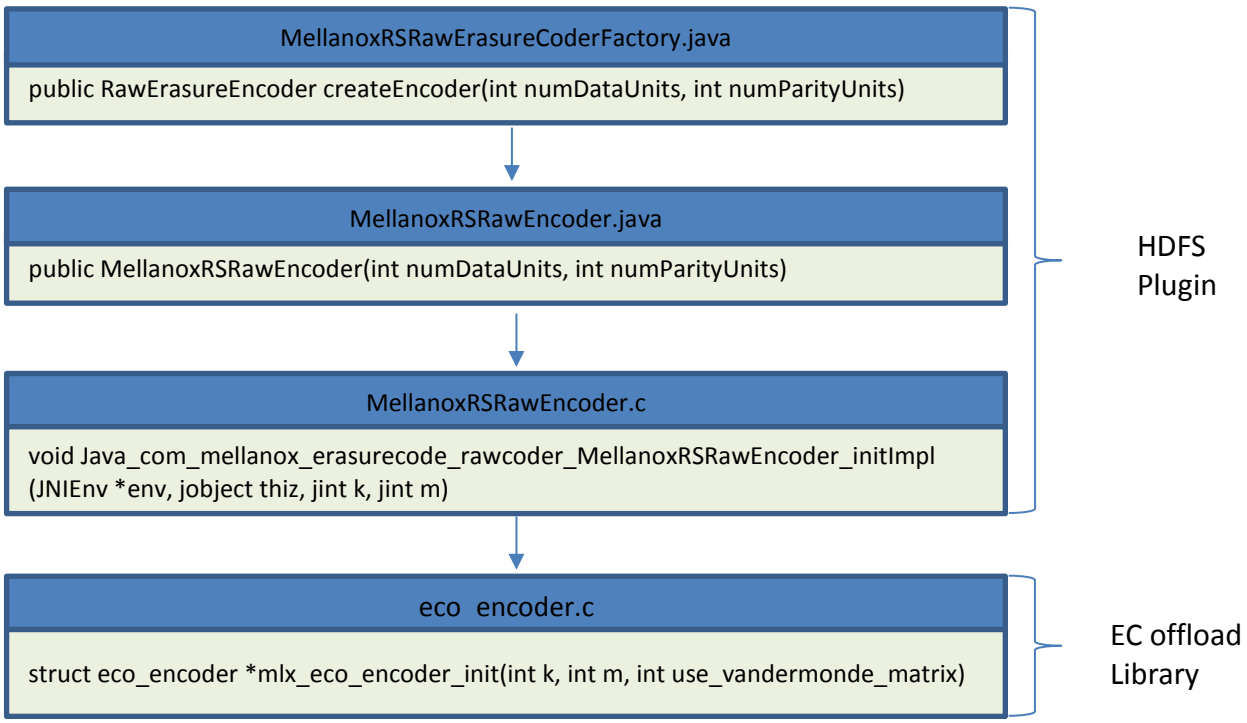
Methods:
void Java_com_mellanox_erasurecode_rawcoder_MellanoxRSRawDecoder_initImpl(JNIEnv *env, jobject thiz, jint numDataUnits, jint numParityUnits)
void Java_com_mellanox_erasurecode_rawcoder_MellanoxRSRawEncoder_decodeImpl(JNIEnv *env, jobject thiz, jobjectArray inputs, jintArray inputOffsets, jintArray erasedIndexes, jint dataLen, jobjectArray outputs, jintArray outputOffsets)
void Java_com_mellanox_erasurecode_rawcoder_MellanoxRSRawDecoder_destroyImpl(JNIEnv *env, jobject thiz)

6.7.3 libECOffload.so

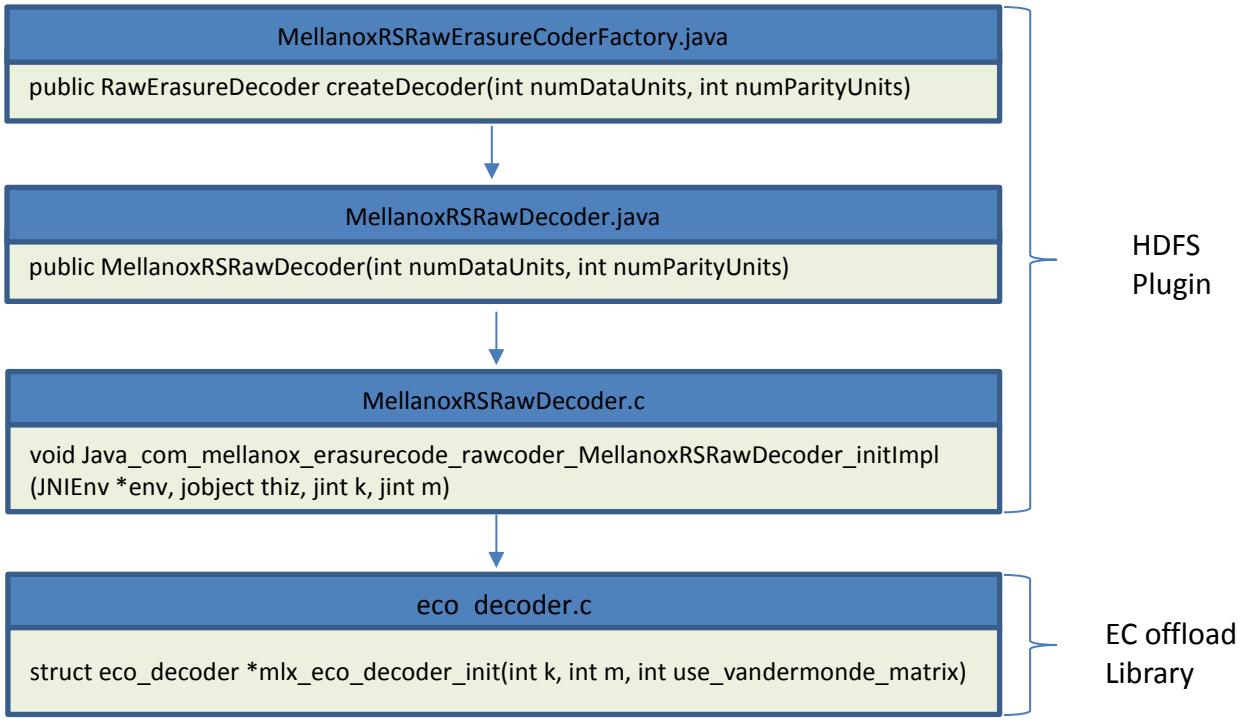
This module is responsible for all the Erasure Coding calculations.
More info describe above.

6.8 Flow Diagram

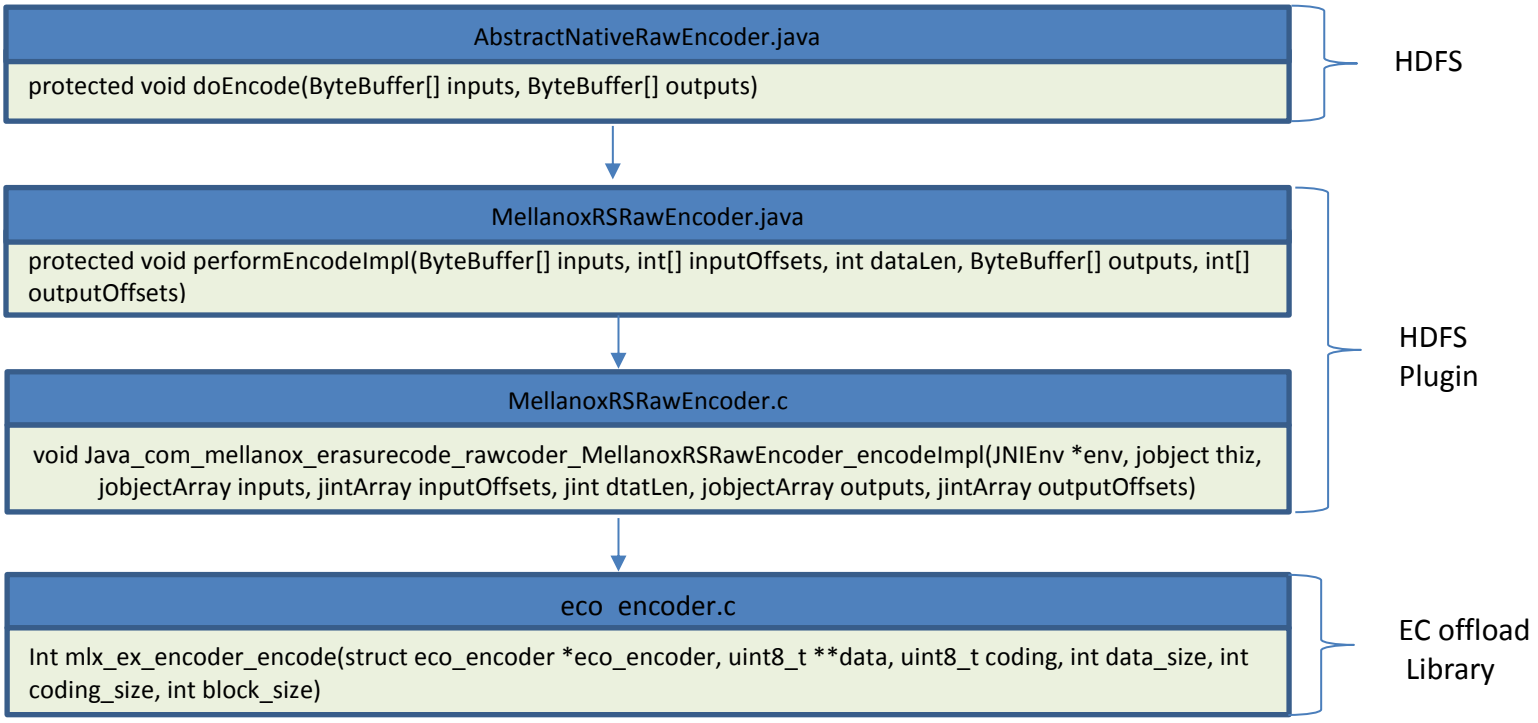
Encoder Initialization flow



Decoder Initialization flow



Encode implementation Flow



Decode implementation Flow