EOS Erasure Coding plug-in as a case study for the XRootD client declarative API
Outline

- Motivation
- The EC use case
- Summary
Motivation

- Use case: erasure coding plug-in for EOS
  - Executing multiple operations on multiple remote files (stripes) in parallel
- Problem with asynchronous operation composability and code readability
  - Asynchronous `Open() + Write() + Close()` in the code is only visible as an `Open()` (rest of the workflow is in the callbacks)
Case study: Write erasure coded block of data

We would like to implement a ECWrite() method based on XRootD client API

- Write one block **striped to** \( n \) **data chunks** and \( m \) **parity chunks**
Case study: Write erasure coded block of data

• We need to open all stripes, write to all stripes, set extended attributes on all stripes (e.g. checksum), close all stripes
• Ideally, for performance we would like to use only asynchronous APIs
• The write operation and setting extended attributes should be done in parallel
Case study: Write erasure coded block of data

Update of a single stripe/chunk with standard XrdCl API ...

```cpp
using namespace XrdCl;

/*
 * Write to a single chunk
 */
void ECWrite(uint64_t offset,
             uint32_t size,
             const void *buff,
             ResponseHandler *userHandler)
{
    // translate arguments to chunk specific parameters
    // ...
    File *file=new File();
    OpenHandler *handler=
        new OpenHandler(file, userHandler, /*long list of arguments*/);
    // although we do a write in here we only see an open call,
    // all the logic is hidden in the callback and the workflow
    // is unclear
    file->Open(url, flags, handler);
}
```
Case study: Write erasure coded block of data

... also all this boilerplate code is needed!

```cpp
using namespace XrdCl;

class CloseHandler : public ResponseHandler {
    CloseHandler(File *file, /* other arguments */) {
        /*...*/
    }

    void HandleResponse(XRootDStatus *st, AnyObject *rsp) {
        // 1. validate status and response first
        // ...
        // 2. call the end-user handler
        userHandler->HandleResponse(st, rsp);
    }

    // members
    // ...
};

class XAttrHandler : public ResponseHandler {
    XAttrHandler(File *file, /* other arguments */) {
        //...
    }

    void HandleResponse(XRootDStatus *st, AnyObject *rsp) {
        // 1. validate status and response first
        // ...
        // 2. proceed to the next operation
        CloseHandler *handler = new CloseHandler(file, /*...*/)
        file->Close(handler);
    }

    // members
    // ...
};

class OpenHandler : public ResponseHandler {
    OpenHandler(File *file, /* other arguments */) {
        //...
    }

    void HandleResponse(XRootDStatus *st, AnyObject *rsp) {
        // 1. validate status and response first
        // ...
        // 2. proceed to the next operation
        WrHandler *handler = new WrHandler(file, /*...*/)
        file->Write(offset, size, buffer, handler);
    }

    // members
    // ...
};
```
Case study: Write erasure coded block of data

What do we have so far:

• We updated **only one chunk**

• Write and SetXAttr happen **sequentially** (we would need **yet another handler-class** to aggregate the result of parallel execution)

• The amount of **boilerplait code is SIGNIFICANT!!!**

• To update all data stripes and parity stripes we will need **yet another handler-class** to cope with parallel execution

• The boilerplait code is very repetitive!
Case study: Write erasure coded block of data

We extracted the repeating patterns, applied significant amount of template meta-programming and got a new declarative API:

- Asynchronous operation composability
- Code readability
- Clear workflow
- In line with modern C++ (ranges v3 inspired, support for lambdas, std::futures)
- Released in 4.9.0 but more complete set of features available only in 5.0.0
Case study: Write erasure coded block of data

Using declarative API:

```cpp
using namespace XrdCl;

// Write erasure coded block
void ECWrite(uint64_t offset,
             uint32_t size,
             const void *buffer,
             ResponseHandler *userHandler)
{
    std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
    for (size_t i = 0; i < nbchunks; ++i)
    {
        // calculate offset, size and buffer for each stripe/chunk
        // ...
        File *file = new File();
        Pipeline p = Open(file, url, flags)
        | Parallel(Write(file, choff, chsize, chbuff),
                   SetXAttr(file, "xrdcc.cksum", checksum))
        | Close(file) >> [file](XRootDStatus&){delete file;}
    }
    // Execute the workflow!
    Async(Parallel(wrts) >>
          [userHandler](XRootDStatus& st)
            {userHandler->HandleResponse(new XRootDStatus(st, 0));});
```
using namespace XrdCl;

// Write erasure coded block
void ECWrite(uint64_t offset,
              uint32_t size,
              const void *buffer,
             ResponseHandler *userHandler)
{
  std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
  for(size_t i=0;i<nbchunks;++i)
  {
    // calculate offset, size and buffer for each stripe/chunk
    File *file=new File();
    Pipeline p=Open(file,url,flags)
    Parallel(Write(file,choff,chszie,cbuff),
             SetXAttr(file,"xrdenc.cksum",checksum))
    Close(file)>>[file](XRoadDStatus&){delete file;}
  }
  // Execute the
  Async(Parallel(userHandler,
             userHandler,0)(new XRootDStatus(st),0));

Composse operations with | operator!
Case study: Write erasure coded block of data

Using declarative API:

```c++
using namespace XrdCl;

// Write erasure coded block
void ECWrite(uint64_t offset,
             uint32_t size,
             const void *buffer,
             ResponseHandler *userHandler)
{
    std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
    for (size_t i=0; i<nbchunks; ++i)
    {
        // calculate offset, size and buffer for each stripe/chunk
        // ...
        File *file=new File();
        Pipeline p=Open(file, url, flags);
        Parallel(WRITE(file, choff, chsize, chbuff),
        SetXAttr(file, ”xrdce_cksum”, checksum))
        Close(file)>>[file](XRootDStatus&){delete file;}
    }
    // Execute the workflow
    Async(Parallel(wrts) >>
    [userHandler](XRootDStatus st)
    {userHandler->handleError(st);});
```
using namespace XrdCl;

// Write erasure coded block
void ECWrite(uint64_t offset, uint32_t size, const void *buffer, ResponseHandler *userHandler)
{
    std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
    for (size_t i=0; i<nbchunks; ++i)
    {
        // calculate offset, create buffer for each stripe/chunk
        // ...
        File *file=new File;
        Pipeline p=Open(file, " Parallel|s, chsize, chbuff),
        // xrdc.cksum", checksum));
        Close(file) >> [file](XRootDStatus&){ delete file;}
    }

    // Execute the workflow!
    Async(Parallel(wrts) >>
    [userHandler](XRootDStatus &st)
    {userHandler->HandleResponse(new XRootDStatus(st), 0);});
Case study: Write erasure coded block of data

Using declarative API:

```cpp
using namespace XrdCl;

// Write erasure coded block
void ECWrite(uint64_t offset,
              uint32_t size,
              const void *buffer,
              ResponseHandler *userHandler)
{
    std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
    for (size_t i = 0; i < nbchunks; ++i)
    {
        // Specify offset, size and buffer for each stripe/chunk
        wrts[i] = Write(file, choff, chsize, chbuff);
        SetXAttr(file, "xrdsec.cksum", checksum)
        [Close(file) >> file](XRootDStatus&){ delete file;}
    }

    // Execute the workflow!
    Async(Parallel(wrts) >>
           [userHandler](XRootDStatus& st)
           {userHandler->HandleResponse(new XRootDStatus(st),0);});
}
```
Case study: Write erasure coded block of data

Using declarative API:

```cpp
using namespace XrdCl;

// Write erasure coded block
void ECWrite(uint64_t offset,
             uint32_t size,
             const void *buffer,
             ResponseHandler *userHandler)
{
    std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
    for (size_t i=0;i<nbchunks;++i)
    {
        offset, size and buffer for each stripe/chunk
        wrts.push_back(
        (Write(file, choff, chsize, chbuff),
         SetXAttr(file, "xrdcr.cksum", checksum))
        | Close(file) >> [file](XRootDStatus&){ delete file;});
    }

// Execute the workflow!
Async(Parallel(wrts) >>
    [userHandler](XRootDStatus& st)
    { userHandler->HandleResponse(new XRootDStatus(st),0);});
}```
Case study: Write erasure coded block of data

Using declarative API:

```cpp
using namespace XrdCl;

void ECWrite(uint64_t numchunks, const Response & response) {
    std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
    for (size_t i = 0; i < nbchunks; ++i) {
        // calculate offset, size and buffer for each stripe/chunk
        // ...
        File *file = new File();
        Pipeline p = Open(file, url, flags)
            | Parallel(Write(file, choff, chsize, chbuff),
                      SetXAttr(file, "xrdchecksum", checksum))
            | Close(file) >> [file](XRootDStatus & st) { delete file; }
    }

    // Execute the workflow!
    Async(Parallel(wrts) >>
          [userHandler](XRootDStatus & st)
            { userHandler->HandleResponse(new XRootDStatus(st, 0)); });

24/11/2019
Michal Simon
```
Case study: Write erasure coded block of data

Using declarative API:

```cpp
using namespace XrdCl;

// Write erasure coded block
void ECWrite(uint64_t offset,
              uint32_t size,
              const void *buffer,
              ResponseHandler *userHandler)
{
    std::vector<Pipeline> wrts; wrts.reserve(nbcunks);
    for (size_t i=0; i<nbcunks; ++i)
    {
        // calculate offset, size and buffer for each stripe/chunk
        // ...
        File *file=new File();
        Pipeline p=Pipeline(file, choff, chsize, chbuff),
        Pipeline q=Pipeline(file, "xrdec.cksum", checksum));
    }
    // Execute wrts
    AsyncParallel(wrts) >>
    [userHandler](XRootDStatus& st)
    {userHandler->HandleResponse(new XRootDStatus(st), 0);};
}
```
Case study: Write erasure coded block of data

Using declarative API:

```cpp
using namespace XrdCl;

// Write erasure code
void ECWrite(uint64_t n, uint32_t const * userHandler)
{
    std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
    for (size_t i = 0; i < nbchunks; ++i)
    {
        // calculate offset, size and buffer for each stripe/chunk
        // ...
        File *file = new File();
        Pipeline p = Open(file, url, flags)
            | Parallel(Write(file, choff, chsize, chbuff),
                    SetXAttr(file, "xrd_ecksum", checksum))
            | Close(file) >> [file](XRootDStatus&){ delete file; }
    }
    // Execute the workflow!
    Async(Parallel(wrts)) >>
        [userHandler](XRootDStatus& st)
        { userHandler->HandleResponse(new XRootDStatus(st), 0); });
```
Summary

• Constraints: available only as a **private API**
  • No template export available in gcc 4.8.5 (cc7), so making it public would effectively mean we won’t be able to change a thing

• Future work
  • Once XRootD protocol supports request bundling we will be able to **translate pipelines to bundled requests** (hopefully at compile-time) in order to save some RTTs
  • Exposing it in **Python bindings**
  • Documentation: [http://xrootd.org/doc/xrdcl-docs/www/xrdclldocs.html#x1-600005](http://xrootd.org/doc/xrdcl-docs/www/xrdclldocs.html#x1-600005)
Questions?