
CERN

Michal Simon

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

Outline

- Motivation
- The EC use case
- Summary

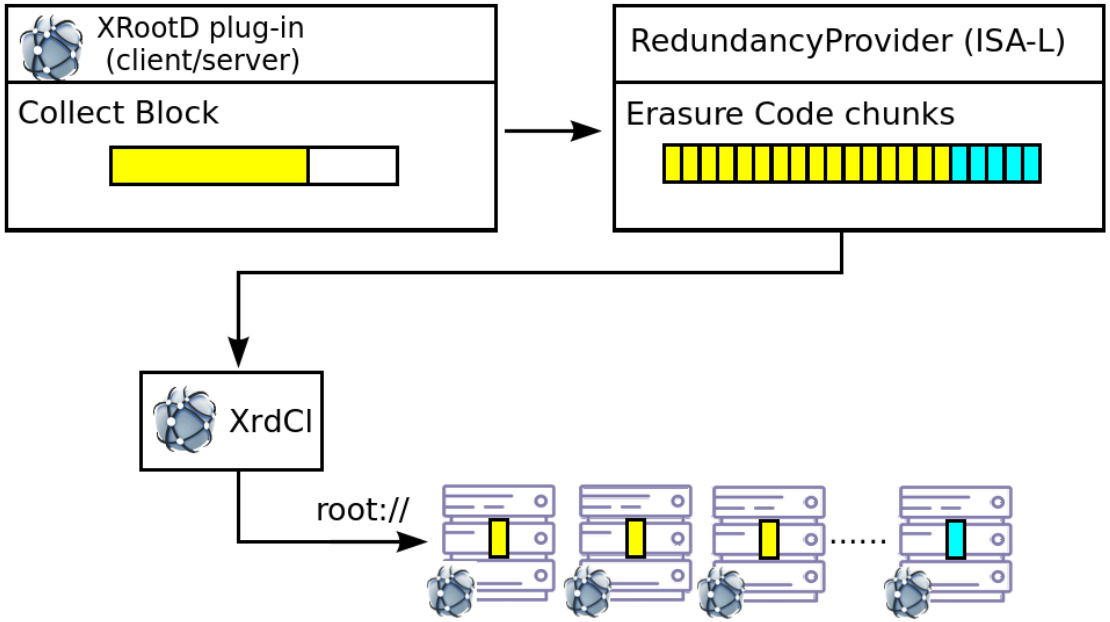
Motivation

- 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 ...

```
1
2  using namespace XrdCl;
3
4  /*
5   * Write to a single chunk
6   */
7  void ECWrite(uint64_t           offset,
8               uint32_t           size,
9               const void        *buff,
10              ResponseHandler *userHandler)
11 {
12   // translate arguments to chunk specific parameters
13   // ...
14   File *file=new File();
15   OpenHandler *handler=
16     new OpenHandler(file,userHandler,/*long list of arguments*/);
17   // although we do a write in here we only see an open call,
18   // all the logic is hidden in the callback and the workflow
19   // is unclear
20   file->Open(url,flags,handler);
21 }
22
```

Case study: Write erasure coded block of data

… also all this boilerplate code is needed!

```
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 WrtHandler : public ResponseHandler
{
  WrtHandler(File *file,/*other arguments*/){ //... }

  void HandleResponse(XRootDStatus *st, AnyObject *rsp)
  {
    // 1. validate status and response first
    // ...
    // 2. proceed to the next operation
    XAttrHandler *handler = new XAttrHandler(file,/*...*/)
    file->SetXAttr("xrdec.chsum",checksum,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
    WrtHandler *handler = new WrtHandler(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 t**o 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:

```
1
2  using namespace XrdCl;
3
4  // Write erasure coded block
5  void ECWrite(uint64_t          offset,
6               uint32_t          size,
7               const void       *buffer,
8               ResponseHandler *userHandler)
9  {
10   std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
11   for(size_t i=0;i<nbchunks;++i)
12   {
13     // calculate offset, size and buffer for each stripe/chunk
14     // ...
15     File *file=new File();
16     Pipeline p=Open(file,url,flags)
17               |Parallel(Write(file,choff,chsize,chbuff),
18                         SetXAttr(file,"xrdec.cksum",checksum))
19               |Close(file)>>[file](XRootDStatus&){delete file;}
20   }
21   // Execute the workflow!
22   Async(Parallel(wrts) >>
23         [userHandler](XRootDStatus& st)
24         {userHandler->HandleResponse(new XRootDStatus(st),0);});
25 }
26
```

Case study: Write erasure coded block of data

Using declarative API:

```
1
2  using namespace XrdCl;
3
4  // Write erasure coded block
5  void ECWrite(uint64_t          offset,
6               uint32_t          size,
7               const void       *buffer,
8               ResponseHandler *userHandler)
9  {
10   std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
11   for(size_t i=0;i<nbchunks;++i)
12   {
13     // calculate offset, size and buffer for each stripe/chunk
14     // ...
15     File *file=new File();
16     Pipeline p=Open(file,url,flags)
17               |Parallel(Write(file,choff,chsize,chbuff),
18                         SetXAttr(file,"xrdec.cksum",checksum))
19               |Close(file)>>[file](XRootDStatus&){delete file;}
20   }
21   // Execute the
22   Async(Parallel
23         [userHa                                    t)
24         {userHa                                  new XRootDStatus(st),0);});
25 }
26
```

Compose operations with | operator!

Case study: Write erasure coded block of data

Using declarative API:

```
1
2  using namespace XrdCl;
3
4  // Write erasure coded block
5  void ECWrite(uint64_t          offset,
6               uint32_t          size,
7               const void       *buffer,
8               ResponseHandler *userHandler)
9  {
10   std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
11   for(size_t i=0;i<nbchunks;++i)
12   {
13     // calculate offset, size and buffer for each stripe/chunk
14     // ...
15     File *file=new File();
16     Pipeline p=Open(file, url, flags)
17               |Parallel(Write(file,choff,chsize,chbuff),
18                         SetXAttr(file,"xrdec.cksum",checksum))
19               |Close(file)>>[file](XRootDStatus&){delete file;}
20   }
21   // Execute the workflow
22   Async(Parallel(wrts) >>
23         [userHandler](X
24         {userHandler->H                tDStatus(st),0);});
25 }
26
```

Parallel execution of operations!

Case study: Write erasure coded block of data

Using declarative API:

```
1
2  using namespace XrdCl;
3
4  // Write erasure coded block
5  void ECWrite(uint64_t          offset,
6               uint32_t          size,
7               const void       *buffer,
8               ResponseHandler *userHandler)
9  {
10   std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
11   for(size_t i=0;i<nbchunks;++i)
12   {
13     // calculate offset, re     uffer for each stripe/chunk
14     // ...
15     File *file=new F
16     Pipeline p=Oper
17                |Par                    ,chsize,chbuff),
18                                        xrdec.cksum",checksum))
19                |Close(file)>>[file](XRootDStatus&){delete file;}
20   }
21   // Execute the workflow!
22   Async(Parallel(wrts) >>
23         [userHandler](XRootDStatus& st)
24         {userHandler->HandleResponse(new XRootDStatus(st),0);});
25 }
26
```

Parallel execution of a container of operations

Case study: Write erasure coded block of data

Using declarative API:

```
1
2  using namespace XrdCl;
3
4  // Write erasure coded block
5  void ECWrite(uint64_t          offset,
6               uint32_t          size,
7               const void       *buffer,
8               ResponseHandler *userHandler)
9  {
10   std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
11   for(size_t i=0;i<nbchunks;++i)
12   {
13     // ... offset, size and buffer for each stripe/chunk
14
15     ...e();
16     ...e,url,flags)
17     ...(Write(file,choff,chsize,chbuff),
18                 SetXAttr(file,"xrdec.cksum",checksum))
19               |Close(file)>>[file](XRootDStatus&){delete file;}
20   }
21   // Execute the workflow!
22   Async(Parallel(wrts) >>
23         [userHandler](XRootDStatus& st)
24         {userHandler->HandleResponse(new XRootDStatus(st),0);});
25 }
26
```

Specify async callback with >> operator

Case study: Write erasure coded block of data

Using declarative API:

```
1
2  using namespace XrdCl;
3
4  // Write erasure coded block
5  void ECWrite(uint64_t          offset,
6               uint32_t          size,
7               const void       *buffer,
8               ResponseHandler *userHandler)
9  {
10   std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
11   for(size_t i=0;i<nbchunks;++i)
12   {
13     // ... offset, size and buffer for each stripe/chunk
14
15     ...();
16     ..., url, flags)
17     ...(Write(file, choff, chsize, chbuff),
18                 SetXAttr(file, "xrdec.cksum", checksum))
19                 |Close(file)>>[file](XRootDStatus&){delete file;}
20   }
21   // Execute the workflow!
22   Async(Parallel(wrts) >>
23         [userHandler](XRootDStatus& st)
24         {userHandler->HandleResponse(new XRootDStatus(st),0);});
25 }
26
```

Use lambdas (or std::future) as callbacks

Case study: Write erasure coded block of data

Using declarative API:

```
1
2  using namespace XrdCl;
3
4  // Write erasure cod
5  void ECWrite(uint64_
6               uint3
7               const
8               Respon                    ier)
9  {
10   std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
11   for(size_t i=0;i<nbchunks;++i)
12   {
13     // calculate offset, size and buffer for each stripe/chunk
14     // ...
15     File *file=new File();
16     Pipeline p=Open(file,url,flags)
17               |Parallel(Write(file,choff,chsize,chbuff),
18                         SetXAttr(file,"xrdec.cksum",checksum))
19               |Close(file)>>[file](XRootDStatus&){delete file;}
20   }
21   // Execute the workflow!
22   Async(Parallel(wrts) >>
23         [userHandler](XRootDStatus& st)
24         {userHandler->HandleResponse(new XRootDStatus(st),0);});
25 }
26
```

First prepare the workflow

Case study: Write erasure coded block of data

Using declarative API:

```
1
2  using namespace XrdCl;
3
4  // Write erasure coded block
5  void ECWrite(uint64_t          offset,
6               uint32_t          size,
7               const void       *buffer,
8               ResponseHandler  *userHandler)
9  {
10   std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
11   for(size_t i=0;i<nbchunks;++i)
12   {
13     // calculate offset, size and buffer for each stripe/chunk
14     // ...
15     File *file=new File();
16     Pipeline p=C                    (file,flags)
17                                     e(file,choff,chsize,chbuff),
18                                     (file,"xrdec.cksum",checksum))
19                                     ](XRootDStatus&){delete file;}
20   }
21   // Execute t
22   Async(Parallel(wrts) >>
23         [userHandler](XRootDStatus& st)
24         {userHandler->HandleResponse(new XRootDStatus(st),0);});
25 }
26
```

Execute later!

Case study: Write erasure coded block of data

Using declarative API:

```
1
2  using namespace XrdCl;
3
4  // Write erasure cod
5  void ECWrite(uint6
6               uint3
7               const
8               ResponseHandler *userHandler)
9  {
10   std::vector<Pipeline> wrts; wrts.reserve(nbchunks);
11   for(size_t i=0;i<nbchunks;++i)
12   {
13     // calculate offset, size and buffer for each stripe/chunk
14     // ...
15     File *file=new File();
16     Pipeline p=Open(file,url,flags)
17               |Parallel(Write(file,choff,chsize,chbuff),
18                         SetXAttr(file,"xrdec.cksum",checksum))
19               |Close(file)>>[file](XRootDStatus&){delete file;}
20   }
21   // Execute the workflow!
22   Async(Parallel(wrts) >>
23         [userHandler](XRootDStatus& st)
24         {userHandler->HandleResponse(new XRootDStatus(st),0);});
25 }
26
```

only ~15 lines of code,
no boilerplate code!

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/xrdcldocs.html#x1-600005

Questions?