

Motivating Metabots

Matthew J. Barrick and Jack M. Pullikottil

February 13, 2007

The purpose of this document is to develop a better understanding HECURA proposal and outline a possible partitioning of the overall problem from UNM-SSL's point of view.

1 I/O Graphs

A fundamental concept in the context of our discussion is the I/O graph. An I/O graph can be conceptualized as a directed graph that describes the flow of data between different components of the application and the storage system connected by data streams. In reality, it is data structure generated at runtime which maps these application/storage service components to the available resources(MPP's compute and I/O nodes, the storage engine and the communication engine) while satisfying user specified data dependencies and other policy constraints. I/O graphs are generated by the XChange tool on the basis of specifications describing these dependencies and resource allocation policies as well as by monitoring the availability of resources during runtime.

There are two main benefits for structuring the flow of data in this manner:

- It eases the parallelizing of the I/O flow. Knowing the data dependencies, the I/O graph can be custom built so that individual flow paths are replicated. These replicated paths can then be scheduled more efficiently so that the computing resources in the overall setup are better utilized and less pressure is put on the I/O link between the MPP and the storage cluster.
- Tasks can be deployed to decrease the I/O bottleneck. For instance, in some cases, it maybe useful to keep the computation closer to the storage engine. This has the advantage of decreasing large movement of data which can otherwise result in I/O bottlenecks.

In order to enable the construction of these I/O graphs the programmer is tasked with the following responsibilities:

- Breakdown of the application into pieces of staged data transformations - where possible.
- Use some sort of mechanism - possibly a high level API or an XML language to specify the data flow dependencies between application components and the storage system and possibly where these application components are located and how they may be accessed.

Given this specification, XChange will use dynamic code generation and create a custom I/O Graph at runtime optimally using the available resources.

2 Out-of-Band Processing and LWFS

Smartly scheduling these application components is one way to alleviate the I/O bottleneck. There are two complimentary approaches that are being considered:

- Moving certain data transformations entirely out of band. Many HPC applications have considerable pre -and post processing stages that involve considerable I/O. In some cases, different applications or even different stages of the same application may need to be coupled to each other by exchanging data which could be in different formats. Such processing may involve data format transformation, filtering, querying, and reorganization on the scale of hundreds of terabytes. Moving these computations out of band can result in overall better utilization of the compute partitions of the MPP.
- Using light-weight storage systems like the Lightweight File System (LWFS) which allows for data storage at near disk speeds for in-band data streams.

The use systems such as LWFS has a direct bearing on the type of operations that are done out-of-band. In addition to the application data-centric operations that were discussed above, out-of-band operations also include offering file system-centric operations. For instance, data and metadata that is required by in-band components may have to be offered using file abstractions - which are not implicitly present in LWFS.

3 The Metabot Framework

The Metabot framework is responsible for realizing these out-of-band operations. Its roles and responsibilities include:

- Spawning software agents called Metabots, which will run on processors which are not part of the MPP.
- Managing and Scheduling Metabots: Because metabots represent out-of-band activity, they should not interfere with in-band application components making requests to the storage engine. Hence the Metabot framework should be able to reschedule and remove as and when needed.
- Managing QoS: The framework will also be responsible for negotiating and maintaining QoS with other runtime entities (XChange instances - if multiple clusters are accessing the same storage engine) so that expectations for data availability can be managed.

Metabots will operate on stored data streams (say in LWFS containers) and produce new data and/or metadata for different purposes. Since the storage cluster may be servicing multiple applications simultaneously, metabots have to be lightweight entities - so that a number of them can run on the storage engine

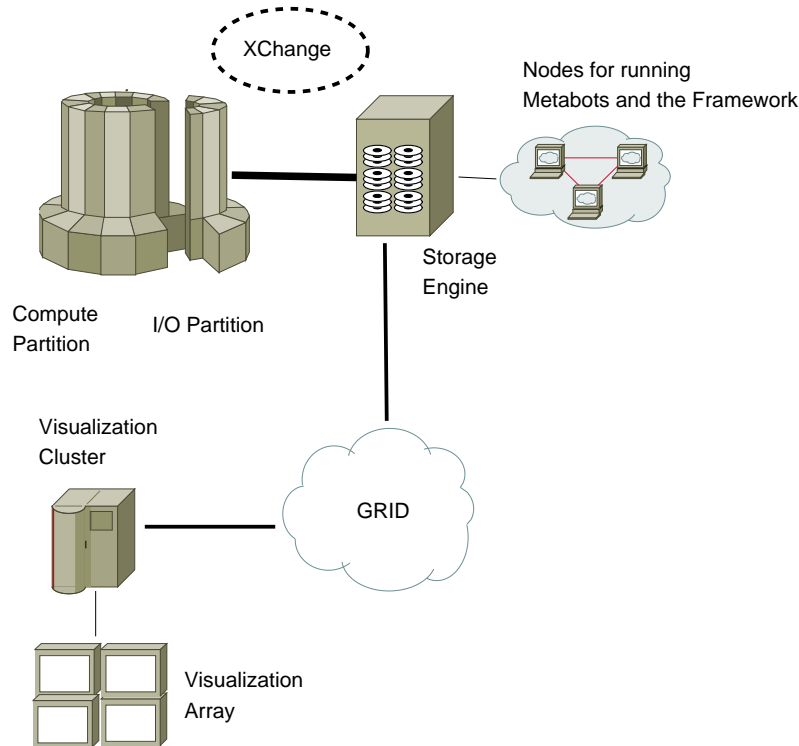
at any given time.

Metabot will offer the following types of services:

- Application Data Centric Services For ex.,
 - Transformation - producing new application data and metadata
 - * Converting File Formats - say from NetCDF to HDF4 (File Translation)
 - * Extracting A submatrix from a single matrix based on different parameters for another application
 - Querying - producing new data and or metadata.
 - * Produce an index of images from a dataset of images based on some criteria. This would be new application specific metadata.
 - * If an index to downsampled versions of the images is what is required, then new data would have to be produced by some specified downsampling function and then index metadata would have to be generated.
- File System Centric Services. For ex.,
 - Reorganizing Data for access by in-band components exploiting full hardware capacity
 - * An offline application specific query or data transformation may produce new data which needs to be organized in LWFS containers so that it may accessed at the fastest possible speed possible.
 - Facilitating File System abstractions - creating file system metadata
 - * Say, the contents of a container is actually a dataset of n files which needs to be accessed individually by some application downstream. So a metabot can be tasked to interpret the contents and produce file system like metadata mapping a set of file-names to container addresses and offsets. A library implementing a filesystem interface may then be able to use this metadata by the same application.
- I/O Graph Centric Services
 - Each in-band stage of an XChange-aware application may have associated with it a customized I/O graph produced by XChange based on specifications of data flow dependencies between different application components in the stage. When these stages are coupled by offline transformations, Metabots are a mechanism to submit these specifications to XChange. So Metabots, can be programmed to store these specifications (in an XML language) physically and logically close to the input data stream of the next application stage, so that it is easily accessed by XChange when the stage begins.

4 A possible sequence of events

We have the following components:



1. The Metabot framework activates one or more Metabots to prepare data and save the XML-specs for XChange.
2. Upon completion, the Metabot framework informs XChange that the next stage may commence along with information regarding the addresses of required data and metadata.
3. XChange reads the I/O Graph relevant metadata and creates and maps the customized I/O graph thereby scheduling the in-band stage.
4. The stage terminates with the data stream being streamed to storage.
5. XChange informs the Metabot framework that the data stream is being stored and negotiates QoS requirements for out-of-band processing following this stage and preceding the next.
6. The Metabot framework activates one or more metabots and the cycle continues from the first step.

5 Issues, Thoughts and Conclusions

- Specifying Metabots - Since these will be specified by the application programmer, we do not want the programmer to be overloaded with low

level details. We need a reasonably simple specification mechanism that abstracts the underlying operations and architecture but which can be converted to produce efficient parallelized executables.

- What services are already available with LWFS? Metabots will have to be a convenient way of working with layered services.
- What kind of QoS commitments do we see the metabot framework accepting? Whats a reasonable way of specifying them.
- How do we maintain QoS in environments when non XChange aware computations are also being executed?
- Do want all I/O to the Storage Engine to run through our service? Such a service could pick up on the extra scheduling information that XChange and Metabot-based applications can provide, and empirically monitor other applications on the cluster. It would then be tasked with handling the I/O and QoS scheduling which will make this all work.