1 Parallel Reduction

1.1 Overview

Once the NeXus raw data file has been written (and is accessible to be read) an automated (user defined) process will be launched using appropriate computational resources. This processing job can either be a simple mantid 'serial' process, which may use thread parallelisation, or split up into separate parallel jobs.

We will need a set of strategies for intelligently splitting up larger jobs into smaller ones, this will also be of use in the future for running reduction procedures on machines of limited resources, for which case we can just run each of the jobs in serial when we don't have access to a parallel resource (but this use case is outside the scope of this project).

Various parts of the Mantid Framework will need modification in order to efficiently make use of any parallel resource (such as a cluster).

We will also look at how a user will easily submit and monitor jobs using a backend parallel compute resource.

1.2 Automated and meta-data driven reduction

In a nutshell: Build end-to-end skeleton to seamlessly process runs coming out of translation service, capture their meta-data in ICAT4, kick-off instrument specific reduction code, and attach reduced data files for the runs in ICAT4.

1.2.1 Prototype work

Set up infrastructure for automated reduction:

- Create ICAT4 production database schemas on snsdb1.sns.ornl.gov
- Deploy ICAT4 and download applications to the glassfish server on icat.sns.gov
- Set up read only data mounting on icat.sns.gov (e.g. /SNS/EQSANS)
- Install nxingest C++ application and mapping files on analysis machines. The nxingest code extracts metadata from a Nexus file to create an xml file, which can be ingested into ICAT4. The mapping defines the structure and content of the output file.
- Implement instrument specific reduction code and install it on analysis machines
- Write a python script to generate metadata xml file for the reduced data and install it on analysis machines
- Build an ICAT4 java client to call ICAT4 server to ingest raw and reduced meta-data
- Create a shell script and install it on analysis machines to catalog and reduce runs coming out of translation service automatically

Beamline specific tasks:

- Finalize automated reduction specific to EQSANS
- Finalize automated reduction specific to POWGEN

1.2.2 Mantid Integration

- Use the GSOAP toolkit to generate C++ source codes to glue the Mantid application to the SOAP/XML web service stack
- Enable Mantid users to search, browse, and download raw and reduced data (analysis data?) through ICAT4 once they are authenticated

- Enable Mantid user to locate datafiles for a given instrument and run number through ICAT4 without authentication
- Enable Mantid application to launch reduction jobs on analysis.sns.gov through ICAT4. That would be used to ask the system to perform the automated reduction another time on files for which meta-data has been added.

1.3 Configuration of Processing Task

As an initial step, the definition of what processing tasks will be run will be defined by a python script with a given fixed name that will be called as part of the standard end of run data flow. The contents of this script will change from case to case.

In the longer term, we need to look into how we better define a workflow.

1.4 Splitting up the Processing Task

1.4.1 Overview

We want to be able to split up large processing tasks into smaller chunks. In this case, the term 'large' is very dependent on the infrastructure you are trying to run on, e.g. a large task for a laptop will be different than a large task for a cluster.

1.4.2 What to split on ?

For any given data set there are many ways that we could split it up into smaller chunks.

- Detector Banks (i.e. based on parts of the instrument)
- Neutron Events (effectively collection time)
- Additional Parameters (e.g. for multi-rotation experiments, you may want to split on motor angle)

1.5 Parallel Extensions to Mantid

Here we discuss extensions to Mantid to allow the effective use of a processing cluster. There are a number of scenarios where this has the potential to overcome significant bottlenecks in the data analysis pipeline (reference to use cases?). The most urgent need at present relates to the processing of extremely large datasets that cannot be fully loaded into memory on even the most capable of the analysis machines - data from NOMAD being the main case in point. In these scenarios it is desirable to split up the run in the manner described in the previous section and distribute the 'chunks' across the cluster. There are further use cases where multiple related runs should be run in parallel (see below), or where long optimization jobs would benefit from distributed processing.

It is true that in many cases the analysis chain is trivially parallelizable - for example the focussing of a single bank in a powder diffraction dataset - save for the desire to combine the outputs into a single Mantid workspace at the conclusion. In this situation, the need could be met by the implementation of a simple batch submission system (such as that described below). However, not only are the likely to be more complex use cases that are not perfectly independent, but the capability of handling the initial splitting and final combination within a single submitted job is highly attractive, and would be provided by the implementation of a solution based upon MPI. Such a solution would not prohibit the submission of simple batch jobs - with one advantage of such jobs being that they could run exactly the same scripts (workflow) as a 'desktop Mantid' could run.

1.5.1 MPI prototype

An initial prototype of an MPI-enabled Mantid is available within the Mantid codebase. A description of this prototype can be found at: https://github.com/mantidproject/documents/blob/master/Design/ParallelAnalysis/MPI_Prototype

We envisage proceeding along the direction that this has taken thus far, with continous evaluation of which particular technologies/implementations will best serve our needs (and continuation of the Mantid practice of placing implementation details behind abstract interfaces where possible).

Salient points of the implementation are:

- It consists of a special MPI-enabled build of the Mantid Framework (i.e. without the GUI layers).
- The submitted analysis jobs will be Python scripts.
- The MPI parts will be encapsulated within special MPI 'algorithms' (in the Mantid sense of the word) which mirror the simple MPI operations (e.g. broadcast, gather, reduce). (Eventually, it may become unavoidable to place MPI code within individual analysis algorithms, but this carries the risk that the path through the code will be different depending on where it is running.)
- The output of the job is a file.

GatherWorkspaces & BroadcastWorkspace Mantid currently has an two experimental algorithms called GatherWorkspaces and BroadcastWorkspace, which uses MPI to spread out work over several processes or nodes. The NOMAD_mpi_example.py script is an example of its use. The data loading and processing of NOMAD's 99 banks is spread over 99 processes. A small python script is run for each bank, processing the data. At the end of the script is a call to GatherWorkspaces which takes the spectra generated in each of the 99 MPI processes and collects them into one workspace. BroadcastWorkspace is used in the PG3_mpi_example.py script to distribute a normalization workspace to all of the processes.

Extensions to GatherWorkspaces & BroadcastWorkspace GatherWorkspaces currently only distributes data in its histogram form (i.e. the result is a Workspace2D). We will extend this (or write an alternative algorithm) to gather EventWorkspaces. This would require transfering event data back to the master. Often this will be less data; however in the case of NOMAD this would be significantly more data to transfer and a challenge will be to try and ensure that user jobs use the most appropriate mode to avoid unnecessarily large results files.

Further extensions will be to transmit the full characteristics of a workspace (i.e. including meta-data where appropriate). This is likely to be achieved by implementing workspace serialization in boost, which will enable the transfer of an entire workspace in a single MPI call.

1.5.2 Merging Multiple Runs

Our most demanding use case is a large inelastic neutron scattering data set, with measurements performed at many goniomieter angles.

Initial Conditions Let us take as an example an inelastic data set consisting of 100 separate runs sweeping through a range of goniometer angles. Let us assume that the data volume is approximately 1 GB for each run.

Using techniques described in the Automated Data Reduction in section 1.2, each run is converted to a reciprocal space MDWorkspace with a fixed box structure - that is, the box structure is common to all the runs. This requires that we have a reasonably good guess of the size (in reciprocal space) that will be covered by the data.

For example, we might split space into 1 million boxes. On average, this will be 1 kB of data per box, though in reality some boxes will contain much more than this, and many others will be empty. This processing can be performed as the experiment occurs, so that each file is ready a few seconds after the completion of the run (most likely bottleneck: file saving).

Merging Files Once all runs have been saved to 100 separate 1GB MDWorkspace files, it is then desired to combine these into one 100 GB MDWorkspace. The MergeMD algorithm, currently present in Mantid, can perform this task with < 100 GB of available memory by reading from disk and writing out to a single large 100 GB file. However, this operation is severely disk I/O limited.

Performance could be significantly improved by using a cluster; let's say that you have 10 nodes with > 10 GB each available. An initial step would be to distribute the events between these 10 nodes. We can relatively quickly read the header information of the 100 files in order to determine where the memory will be used. A simple way to distribute the memory is to randomly assign it to each node.

Through MPI, the root node assigns a list of boxes for each node to load. Each node loads that section of events into memory (POSSIBLE ISSUE: hammering the file system? 10 nodes will be reading from 100 files each). Once data is loaded, it can be reorganized effectively in memory using MDWorkspace's adaptive mesh refinement. Each MDWorkspace on each node will contain about 10 GB worth of events in a random distribution along Multi-dimensional space.

Using Merged MDWorkspace Once loaded on the 10 nodes, these 10 MDWorkspaces representing a single large MDWorkspace. Since all 10 nodes have the 10 GB of data in memory, further processing will be quite quick. The merging script should therefore take advantage of this opportunity to generate some desired visualisation data sets.

The BinMD algorithm bins MDWorkspaces into dense, multi-dimensional histograms (called MDHistoWorkspaces). Each node can perform binning of its (partial) MDWorkspace. The algorithm is parallelized using OpenMP and can effectively use all of the cores. Each MDHistoWorkspace created may be in the range of a few million voxels (from tens to a few hundred MB).

After binning, a version of GatherWorkspaces designed for MDHistoWorkspaces will be needed to gather all of the data. In this case, the MDHistoWorkspaces will need be summed into a final MDHistoWorkspace that represents the visualisation of the total MDWorkspace. This operation should be fairly quick once the data has been transferred.

Re-using MDWorkspaces For later processing, each node could save its partial MDWorkspace to a file for later re-loading. Depending on the performance of the file system, this could take a significant amount of time. A later job could then refer to the file to do further processing.

MDWorkspaces can be backed to a file - that is, the data is not entirely loaded into memory. Instead, only the box structure and a reference to the position of the data on file is loaded. Relevant data is loaded from file on demand. This means that a new call to BinMD, for example, will be relatively quick if it is only for part of the workspace, and therefore only touches a small fraction of the file. These follow-up calls would have to be run on the same number of nodes as the number of separate files.

1.5.3 Interactive Parallel Jobs

As mentioned above, loading/saving the very large MDWorkspaces to file will be a significant slow down. It may become useful in those cases to keep the MDWorkspaces loaded on each node and run Mantid in an interactive way. Although the initial plan (as discussed above) will not take us in this direction, we do not want to close the door on this possibility so it is recorded here.

Client-Server Interface One possible solution would be to run Mantid in an interactive way using a client-server interface. This would be similar to the client-server interface offerred by paraview. In paraview, the pvserver application is run in MPI (mpirun pvserver). The client application communicates with the root MPI node (only), sending rendering commands.

In Mantid, the client could send python commands that would then be evaluated by the MPI root node. Processing would be distributed as needed/possible by the MPI root node.

Alternatively, the communications could be reversed. The GUI client could listen on a port that the root node, or indeed any individual node, could connect back to (firewall issues notwithstanding). In this scenario, the job submission could remain essentially unchanged with respect to the non-interactive case, with a separate tool built into MantidPlot to deal with this case.

Possible Implementation ZeroMQ (http://www.zeromq.org/, aka zmq) is a very interesting option for this purpose. It is a socket library that abstracts of lot of the specifics of implementation. It can communicate through TCP, IPC or inproc so that it could handle a server on the local machine or a distant remote server.

Although it would be a significant redesign, the MantidPlot GUI could operate on the same client/server interface as well. For desktop applications, the server would run locally; or the client could connect to a large remote machine, or it could connect to the root node of a cluster.

Advantages

- Data remains in memory, avoiding constantly re-loading it.
- Much more responsive for a user.

Disadvantages

- The required number of nodes must be available on the cluster. This may mean that the user has to wait until an interactive session is available.
- Resources are "claimed" even when they are not used.

1.6 Mantid Task Submission

1.6.1 Overview

We want to be able to submit and monitor parallel jobs from within MantidPlot. This implies some kind of network communication between the client running MantidPlot and the parallel cluster.

Condor is a cluster-management package from the University of Wisconsin (http://research.cs.wisc.edu/condor/) that should fit our needs quite well. It is designed to schedule jobs on otherwise unused computing resources (for example, idle computers in a student lab), but will function just fine on a dedicated cluster. More importantly for the Mantid project, Condor is designed to allow remote job submission. It has a complete API for submitting jobs, monitoring jobs and retrieving results over the network via SOAP calls. That means that we only need to write a minimal amount of code for MantidPlot to actually call the Condor API's.

Another option for a job scheduler is MOAB. MOAB has the advantage that it's used elsewhere at NCCS and NCCS personnel are familiar with it. The latest version of MOAB - due to be released in February - will have an entirely new remote API and we will be evaluating a beta version to see if this API meets our needs. If it does, then we will probably use it instead of Condor.

1.6.2 GUI

We envision creating a new box on the right side of the MantidPlot window below the algorithms box. This one would be called “Remote Algorithms” and would list the tools that Russell has developed (see 1.5). Clicking on one of these remote algorithms would open a dialog box containing user-selectable options (number of processes, input files, etc...) and a “Submit” button.

The list of available algorithms, along with the necessary parameters to invoke them would be read from a configuration file. We would like to fetch this file from a well-known location on the cluster. That way, when changes are made to the software on the cluster, they would automatically be distributed to every MantidPlot user.

There should also be a “Job Monitor” button (or perhaps a menu choice) that will bring up a window showing current and completed jobs.

1.6.3 Data Transfer

Wherever possible, we want to avoid moving data between the workstation running MantidPlot and the parallel cluster. Ideally, all data will reside on a parallel filesystem that’s shared by both systems. In such cases, when a parallel job is submitted from MantidPlot, all that needs to be included is the name of the input file and the parallel cluster can read it locally.

In cases where a parallel filesystem is not available (such as a user working remotely), then some kind of file transfer mechanism will be necessary. Condor’s API’s allow for transferring the input data to the cluster and retrieving the output. We aren’t sure what MOAB’s API’s allow, so if we decide to use MOAB, it might be necessary to set up a separate transfer mechanism such as scp or gridFTP.

1.6.4 Setup and Configuration

We will need a dialog box in MantidPlot to hold various configuration options, such as the address of the cluster head node and the location of the algorithm configuration file mentioned in 1.6.2. Given that there is only one cluster right now and that there will probably never be more than a few, it seems practical to include these details in a config file that’s included in the MantidPlot distribution. This will allow users to just select the cluster from a drop-down list. Moreover, given the prerequisite of having a remote API that MantidPlot can communicate with, we don’t expect it to be practical for the user to specify their own cluster.

1.6.5 User Authentication and Authorization

Condor accepts a number of different authentication/authorization options (and we assume that MOAB will as well). We would like something that would allow the user to authenticate once yet allow multiple connections to the cluster. For example, forcing the user to enter a password every time he/she wanted to check on the status of a running job would not be ideal.

One possible option is to use X.509 certificates. We can set up a server (either the cluster head node or some other convenient server) that will issue X.509 certificates. The user logs in and obtains a certificate that is valid for a reasonable amount of time (12 hours, for example). This certificate is then used for all of the Condor API calls. We would expect to integrate the act of obtaining the certificate into MantidPlot so that users needn’t even be aware that certificates are in use.

1.7 Parallel IO

In order to make most efficient use of a parallel file system we may need to implement addition techniques and methodology for some IO tasks within Mantid. Until we have access to a Parallel File System, it is impossible to determine what these changes are and if indeed they are necessary. This will be revisited once we have done sufficient testing and parameterisation using a parallel file system.