Processing data intensive Matlab jobs through Condor

Fanar M. Abed Stephen McGough

Newcastle University, Newcastle upon Tyne, NE1 7RU, UK
Email:{f.m.al-fadhly, stephen.mcgough}@newcastle.ac.uk

Abstract

Condor provides a powerful job invocation environment which is capable of successfully executing large sets of parameter sweep jobs. Though eviction of jobs from execution nodes can become expensive if the amount of data which needs to be sent to a node is large (such as Lidar data) and checkpointing and migration is not possible. In this paper we propose here a mechanism which can be used alongside Condor to convert the normal Condor job push model into a pull model where sweeps can be separated from data transfer allowing multiple sweeps on a node. This allows for smaller sub-jobs and better efficiency with respect to data transfer. We exemplify this work through the analysis of Lidar data processed using Matlab code.

1 Introduction

The Condor system [8] provides a high throughput environment for processing computationally independent runs of executions. Often referred to as parameter sweep operations where many similar jobs are run changing only the input parameters. Many Condor deployments exploit cycle stealing where idol execution time on computers normally used for other purposes (such as an open access cluster within a University) are used to run Condor jobs. This tends to lend itself well to sweeps of jobs which require little data transfer and short execution times as eviction of Condor jobs from a computer, as it reverts to normal use, will have less impact on the overall flow of the sweep. Conversely if the amount of time required to stage data to / from a computational resource is high there is a desire to perform the maximum amount of work on this resource to reduce effective overhead of transferring the data.

Condor provides the ability to perform checkpointing and migration of executions on remote computers along with file transparency where inputs and outputs from a users program are staged back to the submitting computer. This does however require that the user can compile their code against the Condor libraries and that you are running under a UNIX based operating system. This is something which is not always possible - such as when you are using a commercial package such as Matlab. However, it would be desirable to provide some equivalent functionality's to help reduce failed execution time.

In this abstract we propose an execution environment for use within Condor which provides the following benefits:

- When data is staged to a Condor computer it can be used many times
- Data generated on the Condor computer is staged back to the submission computer as soon as possible.

We therefore separate the data staging part of the Condor job submission from the job deployment phase and provide a mechanism for returning data to the user while code is still running on the remote computer. The user is required to provide new logic in the form of how to process the returned data and how to deal with incomplete returns where the job is evicted before it completes execution.

Our approach lends itself best to programs where a large data set is used repeatedly, which means that a large number of jobs are needed to be run concurrently on Condor.

2 General Architecture

Our general architecture separates the data staging and the job execution stages of a Condor submission. Figure 1 illustrates the overall architecture.

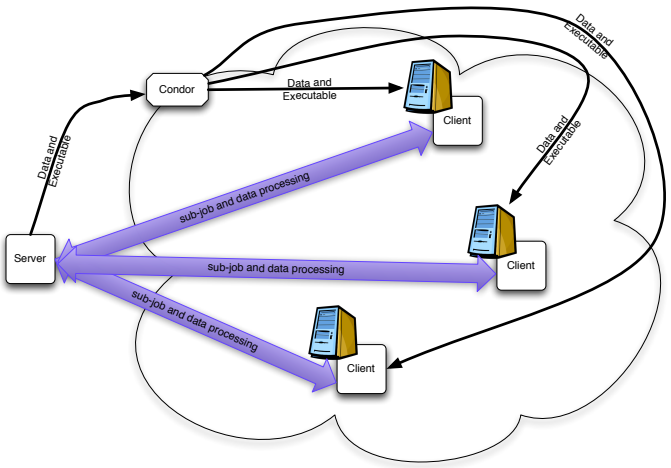


Figure 1: The general architecture

The server is invoked by the user wishing to submit a sweep of jobs. The application to run along with the data files to be processed are packaged into a compressed archive along with a script file of a given name which is used for invoking each run of the application. The Condor cluster at Newcastle is predominantly Windows XP computers thus the scripts are written as Windows Batch scripts. It would be an easy process to convert this over to Unix shell scripts. The 7zip archive format is used for data compression as this was already deployed across the Windows clusters.

The server then starts to submit Condor jobs containing the above archive file and a Java client to the cluster. To prevent excessive load on the server the number of jobs that can be launched at one time and the frequency at which these are launched is limited. As each job starts to request sub-jobs from the server the server can deploy new jobs into Condor until the pre-defined limit is reached.

When the Java client executes on the remote computer it calls back to a server running on the submit computer. This changes the normal Condor job push model into a pull model. This is similar to the pilot jobs within the EGEE Grid [7].

The Java client can now request the next piece of work from the server side. As we have now broken the link between data transfer and execution, this sub-job can be as small as possible with the client asking for further tasks without having to go through the process or re-downloading the data set. The execution of the original application is invoked by Java which is able to stream the output of the execution back to the server and return the results of this sub-job immediately on completion. The client is now able to contact the server for further sub-jobs and will terminate only when instructed to by the server or due to eviction of the Condor job from the computer.

The user is required to provide five (simple) pieces of code; the first is the script to be run by the client to invoke each execution of the application. The second is Java code run by the Client to indicate which files should be packed and returned at the end of execution and which files to monitor during execution. The remaining three codes are to be used by the server. These are to describe each of the new sub-jobs to be executed, code to describe how to process the results sent back from a successful invocation of a sub-job and code to describe how to process the partial output from a sub-job which was evicted during execution. Currently these latter three codes need to be provided as Java classes, however, it is hoped in the future to provide a simpler coding interface. The number of sub-jobs is provided as an argument to the invocation of the server.

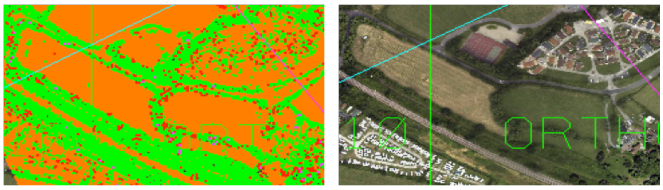
Once the server has successfully completed each of the sub-jobs, either through the initial submission or through re-execution of partially completed sub-jobs it will instruct each of the clients to terminate (successfully). This will be seen by Condor as successful job completion and thus remove the job from the Condor queueing system. Alternatively if the client was evicted during execution Condor will attempt to execute it again on a new node of the Condor cluster.

3 Example Case: Lidar analysis using Matlab

Lidar (LIght Detection And Ranging) is an active remote sensing technology which can acquire highly accurate 3D data of the earths surface [2, 9, 5]. Although all available commercial aerial (Figure 2) Lidar systems can provide users with 3D information of the earths terrain surface, the new generation of full-waveform lidar systems can supply significantly more physical information [1]. This information is crucial for classification and segmentation purposes as they can help to distinguish between different land cover features (e.g. vegetation, buildings, roads. etc.) according to their reflectivity and roughness [4].

Figure 2: Airborne laser scanning technique

Handling raw Lidar data is more challenging than 3D point clouds, and their data processing is a time consuming procedures especially with the new generation of full-waveform lidar data that comprises both geometric and physical properties of the ground features. However, each single flightline strip contains millions of points that need to be processed to obtain the required information (sometimes billions with dense datasets), a real need for an effective processing strategy has emerged. Due to high accuracy requirements, a novel, Rigorous Gaussian pulse Detection method (RGD) has been developed at Newcastle University and successfully applied for full waveform point cloud post processing [3]. This method is an iterated technique that based on the most popular Gaussian decomposition pulse detection method which plays a crucial role in full-waveform lidar processing [6]. It needs multiple seconds to process an individual point, which translates to months to process only one flightline with millions of points using Matlab code. Therefore, adopting a Condor-based approach was essentially in order to feasibly process the whole project dataset. Figure 3 shows the result of processing part of one of these flightlines (a) along with the equivalent aerial view (b).



(a) (b)

Figure 3: The processed point clouds data using RGD method and its equivalent orthogonal aerial view

Each flightline strip equates to data sets of multiple Gigabytes which can take up to ten minutes to transfer across the network even after compression. Thus the desire to process as much data on a Condor node as possible for each data transfer. Each datapoint can be made into a separate sub-job and executed as a Matlab run using the pre-devleoped code. As each datapoint only requires a few seconds for processing and no intermediate stages are exposed from the Matlab code there is no partially completed sub-job state, thus failed executions of the sub-job will be re-executed from the start. Thus both the sub-job generation code and the evicted-sub-job code will generate the index of the datapoint to process. On successful completion of the sub-job the data generated will be added to the set of successful runs which once all sub-jobs have completed successfully will be concatenated together, and pre-prended with a header file, to give the final result file (along with an index file).

The Matlab code has been compiled into a binary executable using the Matlab compiler. This code requires a number of configuration arguments. The data file to process, the index file for this data file, the datapoint within this file to process, where to write the output and where to write the index data for this output. All of these parameters will remain the same except for the datapoint to process. The shell script is passed this index before invoking the Matlab executable. The code written for the client takes the output file and the index entry for this output file and returns them after successful execution. As no intermediary output data is generated the code does not indicate any files to steam back during execution.

4 Deployment environment and results

We are using the Newcastle Condor cluster which has (an average of) 1200 Windows based Condor nodes available. There is a high degree of heterogeneity in these computers ranging from single through to quad core Intel processors with between one and four Gigabytes of memory. Hard disc sizes vary from 120Gb up to 400Gb.

We have demonstrated the ability to reduce execution times for processing an entire flightline from over three months on a single computer to just over 24-30 hours using our approach presented, limited to no more than 100 concurrently running Clients. While this elapsed time need to process individual flightline within used dataset, could be eliminated to be just approximately 1-2 hours if we submit more than 1000 job at time.

5 Conclusion

We have described here a technique for reversing the normal Condor Push job model into a client based pull model. This is particularly useful in situations where the user has large data sets which require significant time to distribute to worker nodes allowing nodes which already have the data to keep on requesting sub-jobs until either evicted or all sub-jobs have been completed.

A full comparison of the benefits and improvements to efficiency will be provided in the final work.

References

[1] Mallet C. Bretar F. Full-waveform topographic lidar: State-of-the-art. *ISPRS Journal of Photogrammetry and Remote Sensing*, 64:1–16, 2008.

[2] Bretar F. Chauve A. Mallet C. Jutzi B. 'managing full waveform lidar data: a challenging task for the forthcoming years. *International Archives of photogrammetry, Remote sensing and spatial information science*, XXXVII, (Part B1):415–419, 2008.

[3] Lin Y.-C. Mills J. P. Smith-Voysey S. Rigorous pulse detection from full-waveform airborne laser scanning data. *International Journal of Remote Sensing*, 2009. in press.

[4] Reitberger J. Schnorr C. Krzystek P. Stilla U. 3d segmentation of single trees exploiting full waveform lidar data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2009. In press.

[5] Persson A. Soderman U. Topel J. Ahlberg S. Visualization and analysis of full-waveform airborne laser scanner data. *International Archives of photogrammetry, Remote sensing and spatial information science*, WG III/3:103–108, 2005.

[6] Wagner W. Ullrichb A. Ducica V. Melzera T. Studnickab N. Gaussian decomposition and calibration of a novel small-footprint full-waveform digitising airborne laser scanner. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60:100112, 2006.

[7] EGEE. Pilot jobs using centralized storage. `http://wiki.egee-see.org/index.php/Pilot_jobs_using_centralized_storage`, May 2010.

[8] Condor Team. Condor Project Homepage. `http://www.cs.wisc.edu/condor`.

[9] Liu X. Airborne lidar for DEM generation: some critical issues. *Progress in Physical Geography*, 32(1):31–49, 2008.