Author: Mahzad Khoshlessan
email: mkhoshle@asu.edu
institution: Arizona State University
Author: Oliver Beckstein
email: obeckste@asu.edu
institution: Arizona State University
corresponding:

Parallel Analysis in MDAnalysis using the Dask Parallel Computing Library

Abstract

The analysis of biomolecular computer simulations has become a challenge because the amount of output data is now routinely in the terabyte range. We evaluate if this challenge can be met by a parallel map-reduce approach with the Dask parallel computing library for task-graph based computing coupled with our MDAnalysis Python library for the analysis of molecular dynamics (MD) simulations [Gowers2016]. We performed a representative performance evaluation, taking into account the highly heterogeneous computing environment that researchers typically work in together with the diversity of existing file formats for MD trajectory data. We found that the the underlying storage system (solid state drives, parallel file systems, or simple spinning platter disks) can be a deciding performance factor that leads to data ingestion becoming the primary bottle neck in the analysis work flow. However, the choice of the data file format can mitigate the effect of the storage system; in particular, the commonly used "Gromacs XTC" trajectory format, which is highly compressed, can exhibit strong scaling close to ideal due to trading a decrease in global storage access load against an increase in local per-core cpu-intensive decompression. Scaling was tested on single node and multiple nodes on national and local supercomputing resources as well as typical workstations. In summary, we show that, due to the focus on high interoperability in the scientific Python eco system, it is straightforward to implement map-reduce with Dask in MDAnalysis and provide an in-depth analysis of the considerations to obtain good parallel performance on HPC resources.

Keywords

MDAnalysis, High Performance Computing, Dask, Map-Reduce, MPI

Introduction

MDAnalysis is a Python library that provides users with access to raw simulation data that allows structural and temporal analysis of molecular dynamics (MD) trajectories generated by all major MD simulation packages [Gowers2016]. The size of these trajectories is growing as the simulation times is being extended from micro-seconds to milli-seconds and larger systems with increasing numbers of atoms are simulated. Thus the amount of data to be analyzed is growing rapidly (into the terabyte range) and analysis is increasingly becoming a bottleneck. Therefore, there is a need for high performance computing (HPC) approaches to increase the throughput. MDAnalysis does not yet provide a standard interface for parallel analysis; instead, various existing parallel libraries are currently used to parallelize MDAnalysis-based code. Here we evaluate performance for parallel map-reduce type analysis with the Dask parallel computing library for task-graph based distributed computing on HPC and local computing resources. As the computational task we perform an optimal structural superposition of the atoms of a protein to a reference structure by minimizing the RMSD of the Cα. A range of commonly used MD file formats (CHARMM/NAMD DCD, Gromacs XTC, Amber NetCDF) and different trajectory sizes are benchmarked on different HPC resources including national supercomputers (XSEDE TACC Stampede and SDSC Comet), university supercomputers (ASU Research computing center (Saguaro)), and local resources (Gigabit networked multi-core workstations). All resources architectures are parallel and heterogeneous with different CPUs, file systems, high speed networks and are suitable for high-performance distributed computing at various levels of parallelization. Such a heterogeneous environment creates a challenging problem for developing high performance programs without the effort required to use low-level, architecture specific parallel programming models for our domain-specific problem. Different storage systems such as solid state drives (SSDs), hard disk drives (HDDs), and the parallel Lustre file system (implemented on top of HDD) are also tested to examine effect of I/O on the performance. The benchmarks are performed both on a single node and across multiple nodes using the multiprocessing and distributed schedulers in Dask library. A protein system of N = 3341 atoms per frame but with different number of frames per trajectory which corresponds to different trajectory sizes of (50GB, 150GB, 300GB) for Dask multiprocessing and (100GB, 300GB, 600GB) for Dask distributed. All the results for Dask distributed are obtained across three nodes on different clusters. Results are compared across all file formats, trajectory sizes, and machines. Our results show strong dependency on the storage system because a key problem is competition for access to the same file from multiple processes. However, the exact data access pattern depends on the trajectory file format and a strong dependence on the actual data format arises. Some trajectory formats are more robust against storage system specifics than others. In particular, analysis with the Gromacs XTC format can show strong ideal scaling over multiple nodes because this highly compressed format effectively reduces (global) I/O at the expense of increasing (local) per-core work for decompression. Our results show that there can be other challenges aside from the I/O bottleneck for achieving good speed-up. For instance, with numbers of processes matched to the available cores, contention on the network may slow down individual tasks and lead to poor load balancing and poor overall performance. In order to identify the performance bottlenecks for our Map-Reduce Job, we have tested and examined several other factors including striping, oversubscribing, Dask Scheduler, and Scheduler Plugin. In addition, lower level tools like MPI for python is used to be compared with high level Dask parall library. This will be specially helpful to identify the possible underlying factors that may lead to low performance. In summary, Dask together with MDAnalysis makes it straightforward to implement parallel analysis of MD trajectories within a map-reduce scheme. We show that obtaining good parallel performance depends on multiple factors such as storage system and trajectory file format and provide guide lines for how to optimize trajectory analysis throughput within the constraints of a heterogeneous research computing environment.

Effect of I/O Environment

In MDAnalysis library, trajectories from MD simulations are a frame by frame description of the motion of particles as a function of time. To allow the analysis of large trajectories, MDAnalysis only loads a single frame into memory at any time [Gowers2016]. Some file systems are designed to run on a single CPU while others like Network File System (NFS) which is among distributed file systems are designed to let different processes on multiple computers access a common set of files. These file systems guarantees sequential consistency which means that it prevents any process from reading a file while another process is reading the file. Distributed parallel file systems (Lustre) allow simultaneous access to the file by different processes; however it is very important to have a parallel I/O library; otherwise the file system will process the I/O requests it gets serially, yielding no real benefit from doing parallel I/O. For XTC file format, file size is smaller as compared to the other formats because of in-built compression. In addition, MDAnalysis implements a fast frame scanning algorithm for XTC files. This algorithm computes frame offsets and saves the offsets to disk as a hidden file once the trajectory is read the first time. When a trajectory is loaded again then instead of reading the whole trajectory the offset is used to seek individual frames. As a result, the time it takes a process to load a frame into memory is short. In fact, each frame I/O will be followed by decompressing of that frame as soon as it is loaded into memory. Thus, as soon as the frame is loaded into memory by one process, the file system will let the next process to load another frame into memory. This happens while the first process is decompressing the loaded frame. Figure [] show the I/O pattern compared between different file formats. For XTC file format, sort of pipelining is happening [] which means that as soon as one process finishes frame I/O, and start decompressing it, the other process can start another frame I/O. However, this is not the case for DCD and netCDF file format []. There is no in- built compression for these types of file formats and as a result file sizes are larger. This will result in higher I/O time (Which is the bottleneck here) and therefore, the whole time one process is loading a frame into memory other processes are waiting. The I/O time is larger for netCDF file format as compared to DCD file format. This is since netCDF has a more complicated file format. Reading an existing netCDF dataset involves opening the dataset; inquiring about dimensions, variables, and attributes; reading variable data; and closing the dataset [ref]. In fact, netCDF has a very sophisticated format, while DCD has a very simple file format. This is why DCD is showing a weak scaling by increasing parallelism whereas netCDF file format is being scaled reasonably well by increasing parallelism across many cores.

Effect of File Format

Figures [] and [] show comparison of job execution time, total compute and I/O time averaged over all processes and the difference between these two times for 300X and 600X trajectories and for all file formats respectively. As can be seen, job execution time does not scale very well across parallelisms from 1 to 72 for all formats. XTC and NCDF file formats reveals much better scaling as compared to DCD file format. As shown in Figure [], the results from different machines lie on top of each other for total compute and IO time for XTC and NCDF file formats; however, this is not the case for job execution time. Unlike job execution time, total compute and I/O time averaged over all processes reveals a reasonable scaling. The same behavior can be seen for other trajectory sizes as shown also in Figures 36 to 41. Based on the present result, there is a difference between job execution time, and total compute and I/O time averaged over all processes. This difference increases with increase in trajectory size for all file formats for all machines. This time difference is much smaller for Comet and Stampede as compared to other machines. In order to find the underlying reasons for this difference, web interface of Dask is used to obtain information about the amount of time spent on the communication between workers, and different computations at the worker level in the Map-reduce job.

Challenges for Good HPC Performance

Performance Optimization

Effect of Striping

Effect of Oversubscribing

Examining Scheduler Overhead

Scheduler Plugin Results

Comparison of Performance of Map-Reduce Job Between MPI for Python and Dask Frameworks

References

Gowers2016(1, 2, 3) R. J. Gowers, M. Linke, J. Barnoud, T. J. E. Reddy, M. N. Melo, S. L. Seyler, D. L. Dotson, J. Doman■ski, S. Buchoux, I. M. Kenney, and O. Beckstein. MDAnalysis: A Python package for the rapid analysis of molecular dynamics simulations. In S. Benthall and S. Rostrup, editors, Proceedings of the 15th Python in Science Conference, pages 102 – 109, Austin, TX, 2016. SciPy. URL http://mdanalysis.org.

Khoshlessan2017 Khoshlessan, Manzad; Beckstein, Oliver (2017): Parallel analysis in the MDAnalysis Library: Benchmark of Trajectory File Formats. figshare. doi:10.6084/m9.figshare.4695742