Implementing and Optimizing Parquet I/O in Chapel

Ben McDonald
Hewlett Packard Enterprise
USA
ben.mcdonald@hpe.com

ABSTRACT

This talk discusses changes made in Arkouda (a NumPy-like Python package with a Chapel backend server) to add support for Apache Parquet, which is a columnar file format designed for data science. The implementation uses Chapel's interoperability features to interact with Parquet's C++ API. Since Chapel does not have native support for C++ interoperability, a thin C-based wrapper library was constructed. To improve the performance of the implementation, optimizations were added to read data in batches and leverage Chapel's parallel programming features. The implementation is evaluated by comparing it against HDF5, a well-known file format for HPC workflows, which has native support in Arkouda. The results of the comparison show that, with these optimizations, Parquet can now outperform HDF5 when operating on datasets stored within a large number of files.

1 INTRODUCTION

Arkouda[1] is a NumPy-like Python package designed for data scientists to interactively use supercomputers and is implemented with a Chapel backend server. Traditional data science workflows rely on reading large amounts of structured data stored in standardized file formats. Apache Parquet[2] is a file format designed for data science and has recently generated significant interest in the Arkouda community. Since Parquet is a file format designed for data science and not HPC, there was concern that it would be a poor fit and perform sub-optimally in Arkouda. After implementing Parquet functionality, optimizations were made before comparing it against HDF5[3], which was designed for HPC systems. Through this comparison, we showed that Parquet in Arkouda performs well and even outperforms HDF5 when reading from multiple files.

2 CHAPEL PARQUET IMPLEMENTATION

Apache Parquet is supported by many different languages and, due to Chapel's ability to interoperate with C, the initial effort aimed to use the Parquet C API. Through this investigation, we discovered that the C API is not as feature rich as other implementations, lacking necessary queries for metadata. This led to experimentation with calling from Chapel to the C++ API, which is the standard Apache Parquet implementation.

Since Chapel does not support interoperability with C++, a thin C wrapper was created over the C++ Parquet library calls that Chapel could call into. Then, prior to compiling the Chapel program, the C/C++ code is compiled into an object file accessible to the Chapel code, allowing Chapel to call C++ functions without making any language changes.

3 OPTIMIZING PARQUET IN CHAPEL

While the C++ Parquet implementation showed significant performance improvements over the C Parquet implementation out of the box, the performance was still not competitive with HDF5. A goal for the Parquet implementation to perform at least half as well as the HDF5 implementation was set, and the first-place optimizations were looked for was in the Parquet API itself.

The initial optimization involved switching from the most simplistic Parquet read API to a lower-level batch reading API. This allowed reading many elements at once directly into the Chapel array, rather than reading into a temporary Arrow array and then copying into the Chapel array, which was the approach that was required with the simplistic, initial implementation. This optimization, while useful in removing the extra array copy, still resulted in only modest performance improvements.

After seeing the benefits from this optimization, the next step taken was to use some parallel features offered by Chapel to read multiple files simultaneously. This optimization was as simple as changing a "for" loop to a "forall" loop in Chapel code and resulted in outperforming HDF5 reads in Arkouda with the Parquet implementation when reading in datasets stored in multiple files.

4 PERFORMANCE RESULTS

Table 1 shows the Arkouda read performance of Parquet and HDF5 with 400 files, each containing 0.25 GiB of integer elements, which is representative of real data used by Arkouda data scientists. Through the optimizations using Chapel's parallel features discussed in section 3, Parquet performance improved ~9x on 1 locale and ~2x on 16 locales. This optimized code now outperforms HDF5 by ~2x on 16 locales and ~6x on 1 locale. For explanations of "Batch read" and "Parallel", please refer to section 3.

Version	1 Locale Throughput	16 Locale Throughput
Initial	0.85 GiB/s	10.75 GiB/s
Batch read	0.96 GiB/s	12.82 GiB/s
Parallel	7.46 GiB/s	23.26 GiB/s
HDF5	1.12 GiB/s	11.90 GiB/s

Table 1: Read performance on a Cray CS with HDR InfiniBand

5 CONCLUSION

This work covers the implementation and subsequent performance optimizations of Apache Parquet in Arkouda, written with Chapel, along with challenges faced through the process. Also demonstrated is one method of interoperating with C++ from Chapel. Finally, this work concludes that the Chapel Parquet implementation outperforms a similar HDF5 implementation while reading from multiple files simultaneously.

REFERENCES

[1] https://github.com/bears-r-us/arkouda
[2] https://parquet.apache.org/
[3] https://www.hdfgroup.org/solutions/hdf5/