SDAC: Porting Scientific Data to Spark RDDs

Tian Yang, Kenjiro Taura, Liu Chao

SDAC: Porting Scientific Data to Spark RDDs

Tian Yang[1,2] ✉, Kenjiro Taura[2], and Liu Chao[1]

[1] School of Computer Science and Engineering
Beihang University, Beijing 100191, China
[2] Graduate School of Information Science and Technology
The University of Tokyo, Tokyo 113-0033, Japan
yang@eidos.ic.i.u-tokyo.ac.jp

Abstract. Scientific data processing has exposed a range of technical problems in industrial exploration and specific-domain applications due to its huge input volume and data format diversity. While Big Data analytic frameworks such as Hadoop and Spark lack their native supports for processing increasing heterogeneous scientific data efficiently. In this paper, we introduce our work named SDAC (Scientific Data Auto Chunk) for porting various scientific data to RDDs to support parallel processing and analytics in Apache Spark framework. With the integration of auto-chunk task granularity-specify method, a better-planned theoretical pipeline can be derived to navigate data partitioning and parallel I/O. We showcase performance comparison with H5Spark within 6 benchmarks in both standalone and distributed mode. Experimental results showed SDAC module achieved an overall improvement of 2.1 times over H5Spark in standalone mode, and 1.34 times in distributed mode.

Keywords: Scientific data, Spark, RDDs, HDF5

1 Introduction

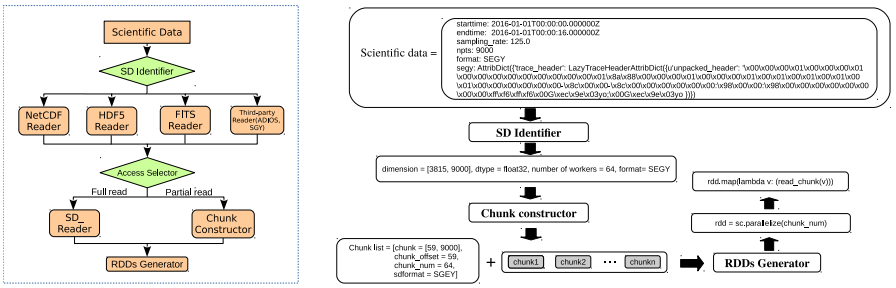
Science is increasingly becoming data-driven [1]. Nowadays, with exponentially proliferating of scientific data volume generated from scientific instruments and computer simulations, the storage capacity, processing efficiency and analytical accuracy are becoming critical challenging. To address these issues, ad-hoc frameworks such as Hadoop and Spark are taken into account. With the seamless integration of MapReduce programming paradigm, rapidly manipulating large amounts of data in parallel becomes feasible. Meanwhile a range of scientific data formats such as HDF5 (Hierarchical Data Format 5)[2] and NetCDF (The Network Common Data Format) [3] are put forward with the similar purpose of solving high volume data storage and platform-independent processing, and have been well proved for specific-domain study and analytics. However, when it comes to utilize parallel frameworks such Spark for processing scientific data, some technical defects are exposed such as lacking methods to specify semantic indexing delimiters as pointed by “Scientists need a way to use intelligent indices and data organizations to subset the search ”in[1].

In this paper, we introduce our module SDAC (Scientific Data Auto Chunk) to bridge the gap between scientific data and Spark RDDs. In order to be better fitted for MapReduce paradigm in Spark, we propose an auto-chunk algorithm to improve the data parallelism level by partitioning the data layout into predefined chunks. SDAC is available at http://github.com/TYoung1221/SDAC.

2 Methodology

2.1 Overview of SDAC

In this paper, we design and implement a module named SDAC (Scientific Data Auto Chunk) to enable various scientific data processing atop Spark framework. The architecture of SDAC module is illustrated in Figure 1(a).



(a) Overview of SDAC stack

(b) Overview of scientific data porting process in Spark

Fig. 1: SDAC Stack and Porting Process

In order to seamlessly integrate scientific data with Spark RDDs, we propose 3 components to implement the porting process:

1. SD Identifier: Recognize scientific data format and map with corresponding read method.
2. Access Selector: Decide data access strategy to optionally process the file entirely or to process in parallel for performance improvement.
3. RDDs Generator: Implement the bridge to port scientific data to RDDs by first parallelizing a collection of total chunk numbers from auto-chunk output. This porting process is shown in Figure 1 (b). Note that the input scientific data has 3815 rows and 9000 columns and is in SGY format.

In SDAC, we reference the Spark vanilla way of RDDs generation by parallelizing the total number of chunks from auto-chunk calculated output. Then the generated RDDs will be mapped with corresponding chunks to generate sub RDDs for distributing among workers.

2.2 Auto-chunk Algorithm

We then propose an auto-chunk algorithm to calculate a better-planned task granularity to navigate in parallel operation. First, a dimension array to specify the chunk unit size is calculated collaboratively by total amount of input multi-dimensional array and available computer resources. Once the chunk dimension size is determined, a B-tree structure will be generated which contains chunk index and chunk offset for retrieving in parallel. The details of the auto-chunk algorithm is shown as follows:

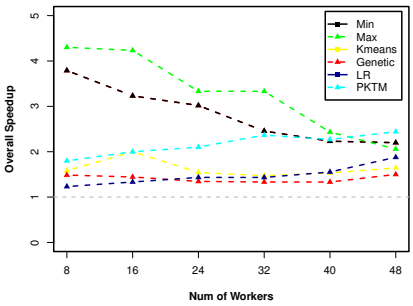
Algorithm 1 Auto-chunk Algorithm

Precondition: n_w is number of workers in Spark, n_d is dimension of scientific data.
function auto_chunk(n_w, n_d)
  if ALL $i \in n_d \bmod n_w \neq 0$ **then**
    $min_{dim} = \text{argmin}_i(\text{abs}(i - n_w))$
    id = dimension.index(min_{dim})
    if $min_{dim} \geq n_w$ **then**
      chunk[id] = int(min_{dim}/n_w)
    else
      chunk[id] = 1
    end if
  else
    $min_{dim} = \text{argmin}_i(i \bmod n_w == 0)$
    id = dimension.index(min_{dim})
    chunk[id] = int(min_{dim}/n_w)
  end if
end function

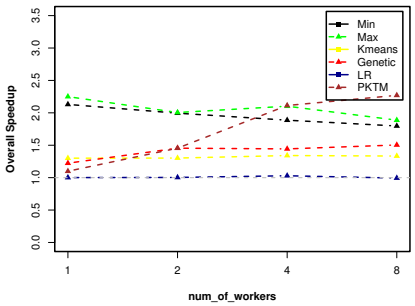
3 Experimental Evaluation

We evaluate SDAC performance comparing with H5Spark [5] via 6 benchmarks which are evaluated on one single machine with 4 AMD Opteron(tm) Processor 6380 CPUs with 64 cores and on 8 worker nodes with a 8-core 2.10GHz Inter E5-2620v4, 16GB of RAM and 4 Spark executor threads in each node. And 3 datasets including 14.15GB HDF5 and 14.37GB NetCDF data are involved in evaluation.

Figure 2(a) shows the overall speedups relative to H5Spark in standalone mode. In which, the benchmark results of Max, Min and PKTM draw speedups between 1.8x and 4.3x, and LR, Genetic and K-means draw smaller speedups between 1.2x and 2.0x. And Figure 2(b) draws the overall speedups over H5Spark in distributed mode. In which the benchmark result of Max, Min and PKTM draw speedups between 1.2x and 2.3x, and Genetic and K-means get relatively smaller speedups between 1.0x and 1.5x.



(a) The overall speedup compared with H5Spark with the number of cores ranging from 8 to 48



(b) The overall speedup compared with H5Spark with the number of workers ranging from 1 to 8

Fig. 2: Overall Speedups comparison between SDAC and H5Spark in standalone and distributed mode

4 Conclusion

In this paper, we propose a light-weight module named SDAC(Scientific Data Auto Chunk) atop Apache Spark framework to bridge the gap between heterogeneous scientific data to Spark RDDs. We describe our efforts in supporting scientific data formats such as HDF5, NetCDF, ADIOS, SGY and FITS. And an auto-chunk algorithm is integrated to navigate parallel I/O by offering a more meticulous strategy to determine task granularity by partitioning the input dataset into pre-defined chunks. We showcase the performance gains across 6 benchmarks compared with H5Spark in both standalone and distributed mode. As future work, we plan to exploit Spark GraphX analytics and machine learning library (MLlib) to make deeper survey of scientific data analytics.

Acknowledgments. This work was in part supported by Grant-in-Aid for Scientific Research (A) 16H01715, Japan.

References

1. Gray, J., Liu, D.T., Nieto-Santisteban, M., Szalay, A., DeWitt, D. J., and Heber, G.: Scientific data management in the coming decade. In: ACM SIGMOD Record, 34(4), pp. 3441 (2005).
2. The hdf group, hierarchical data format version 5,. `http://www.hdfgroup.org/HDF5`.
3. Rew, R., and Davis, G., 1990. Netcdf: an interface for scientific data access. In: IEEE Computer Graphics and Applications, 10, pp. 7682 (1990).
4. Liu, J., Racah, E., Koziol, Q., and Canon, R. S.: H5spark: Bridging the I/O Gap between Spark and Scientific Data Formats on Hpc Systems. In: Cray User Group. (2016)