

ParSA: High-throughput Scientific-data Analysis Framework with Distributed File System

Tao Zhang

Tsinghua

Abstract

Scientific-data analysis and visualization has become a key component in nowadays large-scale simulations. Due to the rapidly increasing data volume and awkward I/O pattern among high-structured files, known serial methods/tools cannot scale well and usually lead to poor performance over traditional architectures. In this paper, we propose a new framework: ParSA (parallel scientific-data analysis) for high-throughput and scalable scientific analysis, with distributed file system. ParSA present the optimization strategies oriented for physical disk to maximize distributed I/O property of distributed file system as well as to maximize overlapping the data reading, processing and transferring during computation. Besides, ParSA provides the similar interfaces as the NCO (netCDF Operator), which is used in most of climate data diagnostic package, making it easy to port this framework. We utilize ParSA to accelerate well-known analysis methods for climate models on Hadoop Distributed File System (HDFS). Experimental results demonstrate the high efficiency and scalability of ParSA.

Keywords:

Data intensive, Scientific data analysis, Distributed file system

1. Introduction

In most of modern scitific applications, huge amounts of data are produced. Large-scale simulations, such as climate modeling, high-energy physics simulation and genome mapping, generate hundreds of terabytes data volumes(Tevfik, 2009; Hey, 2003). Additionally, it still increases as the high resolution model developping. As a consequence, analysis of scientific-data is data-intensive.

8 In fact, almost all of scientific-data are stored in high-structured files,
9 some of which provide parallel I/O interface, such as Network Common Data
10 Format version 4(NetCDF4), Hierarchical Data Format 5(HDF5) (hdfs) and
11 ADIOS BP data format (bp) (bp), and some of which only support serial
12 I/O interface, like Network Common Data Format version 3(NetCDF3). All
13 of these are self-describing, machine-independent data format.

14 In scientific-data analysis, large-scale scientific-data are stored in RAID-
15 5/6 or parallel file system. Yet the analysis methods/ tools are always central-
16 ized approaches, such as NCO and NCL, which are the most used in climate
17 applications for processing NetCDF files, leading to very poor scalability and
18 time-consuming performance.

19 Inspired by big data solution in Internet Port Data Center (IPDC), nu-
20 merous frameworks with distributed strategy have been developed. MapRe-
21 duce is a program framework for processing and generating large data sets,
22 provideing automatic parallel mechanism and build-in fault-tolerance on a
23 cluster. However, this solution with MapReduce requires that data first be
24 transformed into a text-based format. SciHadoop is a Hadoop plugin al-
25 lowing scientists to specify logical queries over array-based data models. It
26 executes queries as map/reduce programs defined over the logical data model.
27 It shows remarkable improvements for holistic functions of NetCDF data sets
28 for the following optimization goals: reduce total data transfers, reduce re-
29 mote reads and reduce unnecessary reads. Nevertheless, SciHadoop using
30 java language leads to the compatibility problem to the existing climate data
31 analysis tools, which is written by c shell scripts, NCL and NCO commands.
32 The SWAMP project [9] has provided the parallel NCO operations, but the
33 reading performance is still bottleneck.

34 In this paper, we propose a new framework - Parallel Scientific-data Anal-
35 ysis (ParSA). We utilize the distributed I/O property with Hadoop Dis-
36 tributed File System (HDFS) to improved data reading throughput. What
37 is more, ParSA optimizes the data layout schedule stored in the distributed
38 file system to overlap the data reading, processing and transferring. Besides,
39 it provides parallel NCO operations, cooperating with HDFS, making it easy
40 to use the efficient tool, without changing a lot for current climate analysis
41 package.

2. HDFS and scientific-data analysis

In this section, we will present the property of distributed file system HDFS, replicas and scheduler, which can be taken advantage of to optimize distributed I/O performance. We also present the character of scientific-data analysis, and discuss about the probability of analysis transportation onto HDFS.

2.1. HDFS

HDFS is an open source project, driven by Google File System (GFS). As a distributed, scalable and portable file system, HDFS is inherent for large-scale data-intensive process.

In HDFS, there are two types of node, Namenode and Datanode. Namenode maintains file system tree and metadata for all files or directories stored in HDFS, and Datanodes are where the data are actually stored. When a file are stored into HDFS, it will be split into file blocks as the storage unit of HDFS. For achieving fault-tolerance, HDFS stores three replicas, by default, for each file blocks in different datanodes. Therefore, even individual node halts down, all data, which are stored in the halted node, can be accessed from other replicas. All of I/O operates can be manipulated through Clientnode.

Each Datanode can mount several hard disks and it manages these hard disks by itself. By default, Datanode will store each block into the hard disks in a round-robin way. For example, when a file “a” will be stored into HDFS via Clientnode, Namenode will add “a” into the file system tree. Then Clientnode begin writing the content into HDFS. Once Clientnode detects that current writing size exceeds the block size, 64MB by default, it will ask Namenode for a new block with unique block number. Simultaneously, Namenode need recode mapping relation between “a” and block numbers. Since three replicas are used in HDFS for fault tolerance, Namenode will select three Datanodes to store a block in the file “a”. In Datanode, HDFS should choose a hard disk to each replica of a block. As shown in Figure 1, the last block “k” is stored in disk “2”, then the new one “j” will locate at disk “0” according to the round-robin rule.

Blocks of files are distributed in the two-tiered storage architecture, Datanode and Datanode’s disk. It will make full use of the collective bandwidth of HDFS if each replica of blocks is appropriately chosen and scheduled to reduce remote reading among Datanodes.

MapReduce is usually introduced as a computation model cooperating with HDFS. It can process the data with good locality. However, due to the traits of scientific data and the operation pattern, MapReduce cannot be directly utilized on scientific-data analysis with perfect performance.

2.2. Scientific-data Analysis

Scientific-data are usually stored in high-structured files, a kinds of special binary format, which can not be read directly by MapReduce. Each file contain multi logical units, and each unit are corresponding to its own physical meaning. The size of each logical varies a lot in one file and only same logical units can be manipulated in most of analysis operations

In this paper, climate model data is used, the total size of which is extremely huge. The high resolution ocean model, a sub-component model in climate model, has 48TB for 100 simulated years, with monthly output. These data are organized by thousands of files with timestamp named in NetCDF format, called history result. Each file contains the same dozens of physical variables. Analysis operations need to process the whole or part of the history results. These operations include computing average, combining the same variables in different files, producing the remapping file and fast Fourier transform (FFT), etc. They require to handle whole or part of variables in multi-files.

2.3. Problems and Challenges

Problem/Challenge 1: How to define the operation unit considering the storage unit Logical unit size varies a lot from a few bytes to gigabytes. Then it can not be used as the operation unit. Otherwise, the workflow of each task will be imbalance.

Problem/Challenge 2: How to schedule tasks among many files to increase the locality and reduce the data network transfer? If all logical units with same meanings locate on the same Datanode, all data access will proceed locally. However, default block placement policy of HDFS cannot ensure that, which leads huge amount data network transfer, as the above mentioned.

Problem/Challenge 3: How to best utilize the disk I/O?

3. Parallel scientific-data analysis

In order to solve the problems mentioned in previous section to take full use of distributed I/O performance, we propose a new framework - Parallel Scientific-data Analysis (ParSA).

113 3.1. Logical Unit Split and Group among File Blocks

114 No matter how a file is split, all data in one block are stored contiguously
115 in a physical location, since file block is the storage unit of HDFS. Thus the
116 block of HDFS can be used as the basic operation unit to keep continuity in
117 disk and improve I/O performance by proper scheduling to reduce network
118 read. In a block, we can group the small size of logical units or split the big
119 size one.

120 The logical unit split and group approach is shown in Figure 2. All logical
121 units located in one block are distributed into one group. If a logical unit
122 spans two or more blocks, this logical unit will split to several parts. For
123 example, LU1 spans three blocks. It will split into three split parts - LU1
124 (1), LU1 (2) and LU1 (3), unique identifier assigned to each part. Then LU1
125 (1) and LU0 locate in file block 0 and form a operation unit group. LU1 (2)
126 itself is distributed as a group. The remaining LU1 (3), LU2 and LU3 are
127 distributed to the third group. The mapping relationship between groups
128 and file blocks should be records, as shown in Figure 3.

129 Variables in NetCDF, used in climate data, can be viewed as multi-
130 dimension array. An array section is defined as a contiguous rectangular
131 block specified by two vectors - index vector and count vector. The index
132 vector indicates the start offset of the element in the corner closest to the
133 origin. The count vector gives the lengths of the edges along each of the
134 variable's dimension.

135 For instance, the variable given in Figure 4 (a) is a two-dimension slab.
136 If we access the whole variable, index vector is (0, 0) and count vector is (3,
137 5). However, if this variable locates on two file blocks stored with HDFS,
138 and variable are split between value (1, 1) and value (1, 2), it cannot be split
139 into these two pieces. The reason is that each of part is not a rectangular
140 shape, which can not use an index vector and count vector to represent. In
141 this situation, we should split the variable into four pieces, as Figure 4(b)
142 shows.

143 3.2. Scheduler

144 As discussed in section 2.3, quite a few operations are only executed on
145 the same logical units among multi-files. In principle, blocks containing these
146 units should be scheduled in one Datanode as many as possible to increase
147 the access locality.

148 At first, information about file distribution should be gathered. As dis-
149 cussion in section 2.1, file block has three replicas by default. In this paper,

150 we define the tuple (N_i, D_j) as the location of a file block, which means
 151 Datanode number and hard disk number, respectively. Assume we get the
 152 following block location information for the first file block of all three files.

153 For the nine replicas for B0 of F1, F2 and F3, Datanode N0 covers 2
 154 replicas. N1 contains 2 replicas and other Datanode only contain 1 for each.
 155 Then Datanode N0 and N1 contain the same number of replicas. Therefore,
 156 Datanode N0 is chosen to process the B0 block of F1 and F2, instead of N1
 157 to process B0 of F0 and F2. The reason is that B0 of F0 and F2 locates
 158 on the same hard disk in N1 Datanode, which is more likely to impact the
 159 performance (explained in section 3.3). Although N1, N6 and N7 all have
 160 the B0 replica of F2, N1 is selected. Because when disk 0 of N0 is broken,
 161 the replica B0 of F0 can be accessed in N1. According to this principle, other
 162 blocks can be scheduled as the Figure 5(b) shown.

163 3.3. Workflow of ParSA

164 Although we utilize the file block location in HDFS to schedule which
 165 block replica are selected among Datanodes, I/O bandwidth of disks may not
 166 be efficiently utilized intra-Datanode. For example, the file block B0s of F0
 167 and F1 are assigned to N0 in Figure 5. Even though B0 of F0 and B0 of F1
 168 locate on different disks - D0 and D1 respectively, the two disks are accessed
 169 sequentially instead of parallelly, if the operations are implemented from logical
 170 level at Datanode N0 as following:

Algorithm 1 Sequential operation on disk in the same Datanode

```

for each logical-unit LU in B0 do
  for each file Fx whose block B0 locates on N0 do
    read LU from B0 of Fx
    operate on LU
  end for
end for

```

171 In this paper, we propose a new workflow to efficiently access the data
 172 on HDFS and maximize utilizing all disk bandwidth in one Datanode.

173 In ParSA, the framework of hybrid of MPI and pthread is used. MPI is
 174 responsible for parallel and communication among Datanodes. There is one
 175 MPI process in each Datanode. Pthead is responsible to implement the detail
 176 operations within each Datanode, constituting of three major components:
 177 reading-thread, processing-thread and receiving-thread.

178 **Reading-thread** If more than one thread dedicatedly access one disk,
 179 it cannot get a higher performance than only one thread. Thus we invoke
 180 one thread for each disk. This is defined as reading-thread and use $Tread_i$
 181 to present the i_{th} reading-thread. For each block, $Tread_i$ reads the queue
 182 of logical units, shown in Figure 3. After reading the logical unit, it is
 183 marked as LOCAL FLAG, distinguishing with receiving data. The workflow
 184 of reading-thread is given below:

Algorithm 2 reading-thread

```

inputs:
datanode_number:the Datanode where the reading-thread locates.
disk_number:the disk number to be accessed.
scheduler_result:task scheduler result, given in Figure 5(b).
LU_groups_map:mapping relations between logical unit groups and file
blocks in Figure 3.

for each tuple (node_no, disk_no) in scheduler_result do
  if node_no = datanode_number and disk_no = disk_number then
    get the file-index and block-index according to the index of tuple,
    shown in Figure 5(a).
    for each lu in LU_groups_map[block_index] do
      read values of lu of file_index file
      store file-index and values into lu
      mark lu as LOCAL_TAG
      insert (lu) into data queue
      notify processing-thread
    end for
  end if
end for
send ending-flag to processing-thread

```

185 **Processing-thread** After reading-threads read logical units into queue,
 186 the processing-thread will be waked up. The processing- thread takes in
 187 charge of calling user-defined function local_data_process to process these
 188 logical units. Users only focus on implementing operations on logical unit.
 189 Since each Datanode only covers part of entire dataset, the major process
 190 should reduce the intermediate results, computed with each process in each
 191 Datanode. This operation is another user-defined recv_data_process function.

192 **Receiving-thread** Receiving-thread is a component in major process. It
193 receives logical inits from worker processes, and puts the logical units into the
194 receiving queue. Then the processing-thread is notified to fetch these data.

195 The workflow of ParSA is shown in Figure 6. Major process locates in
196 Datanode 0, and a worker process is presented as Datanode2. Each read-
197 ing thread reads the logical units which is assigned to this thread. After
198 $Tread_0$ in Datanode 2 reading the logical LU1 (3) of F0, it reads LU2 of
199 F0, inserts the values into queue and notifies processing-thread to wake up.
200 The processing-thread processes all the values in queue sequentially. When
201 processing-thread detects that LU1 (3) are all processed in Datanode 2, it will
202 send the intermediate results to the major process. Once Receiving-thread
203 in major process gets the intermediate results, it inserts the results into the
204 logical unit queue. Since the logical unit queue is the shared resource, all
205 threads must access exclusively.

206 ParSA is the framework for scientific-data access on HDFS. Users can
207 only put their focus on logical operations. ParSA is implemented using MPI
208 and Pthread, providing more flexible configuration with regard to performance
209 than MapReduce. ParSA can achieve load-balance based on current block
210 replicas layout information. However, if the layout itself is unbalance, the
211 performance is also impacted. In the next section, how to further improve
212 the performance by optimizing layout will be discussed.

213 4. Block layout optimization

214 In section 3.2, the scheduler of ParSA utilizes the location information of
215 file blocks. In principle, file blocks containing the same logical units in the
216 one Datanode should be scheduled as many as possible for the operations
217 involving multi-files, while the default block layout strategy of HDFS cannot
218 guarantee this.

219 In HDFS, if Clientnode is one of the Datanodes, the first replica of each
220 file block will be stored in Clientnode, and other replicas are chosen random
221 Datanodes except for Clientnode. If the Clientnode is not Datanode, the
222 first replica will be randomly stored in a Datanode.

223 Although the scheduler can assign tasks based on current file block repli-
224 cas distribution, the distribution should be control according to the principle
225 for task assignment in order to achieve load balancing. Since blocks of files
226 are stored in Datanode and Datanode's disk, it needs to optimize the layout
227 in these the two tiered architecture.

228 4.1. Data layout inter-Datanodes

229 Taken into account the characteristic of scientific-data analysis, tightly
230 coupled data as above described should be assigned in one Datanode to
231 increase locality of processing data. Simultaneously, each Datanode should
232 have equal workload, with a pretty load-balance performance.

233 However, it may indeed exists a few analysis operations different from the
234 characteristic. ParSA expose an interface for users to determine where the
235 first replicas of blocks shoule locate. What’s more, the second replicas are
236 stored with uniform distribution automaticlly. The third abide by the HDFS
237 default strategy.

238 4.2. Data layout intra-Datanodes

239 As discussed in section 3.3, the workflow of ParSA invokes one thread
240 for one disk. If blocks are imbalance among disks in one Datanode, it will
241 waste the bandwidth of other disks. The load balance is not only the equal
242 number of file blocks in one Datanode, but also the distribution of blocks
243 containing the same logical units. The former can be guaranteed by HDFS
244 default disk allocation strategy with a round-robin way, while the later can
245 not be satisfied all the time. ParSA provides a round-robin mechanism for
246 these blocks distribution, as shown in Figure 7. The disk will be used for a
247 block, if it does not store one with the same logical units.

248 5. Evalution

249 5.1. Experimental setting

250 To demonstrate the performance portability of ParSA framework, we
251 choose two HDFS clusters with different configuration, as shown in Table
252 1 and Table 2.

253 In order to demonstrate the efficiency of ParSA, we use climate data
254 generated by a ocean model, POP, a component of climate model. The
255 dataset has 100 NetCDF files, and each of file contains 60 two-dimension
256 variables (logical units) and 18 three-dimension variables, with 320x384 and
257 320x384x40 resolution, which has 309 MB datasize. The total size of this
258 dataset is 30.9 GB.

259 The major analysis operation is calculating time average using ncea com-
260 mand of NCO package, which is the basic preprocessing operation for almost
261 all of climate data analysis.

262 To evaluate the scalability of ParSA, we compare the speedup and through-
263 put performance of different number of Datanodes and different number of
264 disks per node. Cluster 1 is used for the former experiment since the number
265 nodes of cluster 1 is more than cluster 2. While cluster2 is used for the later
266 experiment since each node has more disks in cluster 2.

267 5.2. Scalability with different disks per Datanode

268 ParSA can get about 1.3 GB/s aggregation throughput if per Datanode
269 uses 5 disks, as shown Figure 8(a). The throughput has a remarkable
270 scalability if per Datanode from 1 to 4 disks. However, it has little perfor-
271 mance improvement with 5 disks. The reason is that there are some serial
272 operations, such as writing results and communication, which impact on the
273 performance. Since a series of strategies propused by ParSA is aimed at I/O,
274 Figure 8(b) shows the speedup with only reading part. It's scalability is close
275 to the ideal speedup.

276 5.3. Scalability with different Datanodes

277 Two disks per node are used in this evaluation with cluster 1. It almost
278 keeps the linear speedup until 9 Datanodes as shown Figure 9. However, the
279 performance improvment is limited when the datanodes are greater than 9
280 due to the increasing overhead of communication and the critical section con-
281 trolled by mutex, as shown Figure 6. Although data processing is overlapped
282 with data accessing, the time non-overlapped occupies almost 38.89% on 16
283 datanodes, and this proportions are only 0.02% and 7.96% when datanodes
284 are 9 and 12 respectively.

285 In Figure 9(a), it can get about 800MB/s maximum throughput on 12
286 datanodes, with 24 disks totally. While the same number of disks in Figure
287 8(a), 4 disks per datanode, it can get about 1.2GB/s. The same performance
288 only need 2 disks per datanode, with 12 disks totally. The reason is that, on
289 one hand, the disks and network are difference between the two cluters; on
290 the other hand, cluster 1 requires more network access due to the few disks
291 per node.

292 The speedup of only reading is presented as Figure 9(b), which shows an
293 exllcent scalability with different datanodes. The reason why the speedup is
294 higher than the ideal may result from performance variability.

295 5.4. *Throughput comparison with different task optimization strategies*

296 HDFS block is the basic operation unit in ParSA, splitting and grouping
297 the logical unit. We select the block replicas in different datanodes to reduce
298 network reading, optimize the workflow to support overlapping and paral-
299 lelism of reading and processing, as well as the optimization of blocks layout
300 for both inter-datanodes and intra-datanodes.

301 To present the improving performance, two additional strategies are com-
302 pared to ParSA. The first one takes logical unit as the operation unit. The
303 disk utilization is shown as Figure 10(a). Since the size of logical unit varies
304 quite a lot, a serious load imbalance exists among tasks. The second strat-
305 egy takes all logical units located in one file block as an operation unit, while
306 those logical unit which locate on more than one block do not be split and
307 treat the big size logical units as other operation unit. In this situation, it
308 can not guarantee that an entire operation unit stores in a disk. When a
309 thread try to read this unit, it requires read a part of value from the other
310 disk. In the mean time, the other thread is reading the same disk, leading
311 to disk resource competition. This utilization performance is presented as
312 Figure 10(b).

313 Compared with throughput to evaluate the speedup, ParSA outperforms
314 the above two strategies by a factor of 3.34X and 1.14X respectively.

315 6. Conclusion and future work

316 A high-throughput and scalable scientific framework combined with HDFS
317 distributed file system and NCO analysis tool has been designed, imple-
318 mented and evaluated. Maximize distributed I/O performance has been
319 gained by scheduling block replicas in different datanodes, optimizing the
320 blocks layout for both inter-datanodes and intra-datanodes, as well as over-
321 lapping the data reading, processing and transferring. Instead of Hadoop
322 framework, we use multi-process with MPI to parallel the NCO basic oper-
323 ations, without leading to compatible problem. In future work, we plan on
324 testing the framework on some complex climate data analysis, such as FFT,
325 principal component analysis and Wavelet transform.

Algorithm 3 processing-thread

inputs:

num_tasks:total number of **ending-flag** to wait

intermediate_results:store the intermediate results

cur_num_tasks = 0

while **cur_num_tasks** < **num_tasks** **do**

 wait until receiving notification

if receive **ending-flag** **then**

cur_num_tasks++

 fetch the logical unit queue

for each **lu** in logical unit queue **do**

 get **interm_res** according to **lu identifier** from **intermediate_results**

if **lu's flag** == **LOCAL_FLAG** **then**

 call **local_data_process** (**lu**, **interm_res**)

end if

if **lu's flag** == **RECV_FLAG** **then**

 call **recv_data_process** (**lu**, **interm_res**)

end if

if is not major process **then**

if all logical unit are processed in this Datanode **then**

 store **interm_res** into **lu**

 mark **lu** as **RECV_TAG**

 send (**lu**) to major process

end if

end if

end for

end if

end while
