AdiosStMan Manual

v0.4

Jason Wang

September 23, 2014

International
Centre for
Radio
Astronomy
Research

1 Introduction

This software is a parallel casacore (CASA - Common Astronomy Software Applications) storage manager using ADIOS (Adaptive IO System) as the underlying storage back-end. It is motivated by the fact that the previous CASA storage managers have been all based on the CASA built-in file format, which can only be operated in serial, while there is an increasing need for parallelizing the IO part of casacore to manage larger datasets for next generation radio telescopes. In terms of the parallelism paradigm, this storage manager uses MPI to coordinate IO operations from multiple processes. This provides the potential to be scaled extensively on large scale clusters or supercomputers.

2 Prerequisites

2.1 MPI

MPI is required in multiple stages to get AdiosStMan compiled and run, including compiling ADIOS, compiling AdiosStMan and running AdiosStMan powered casacore applications. Generally, any implementations following the MPI standard should work, but it is recommended to use the latest version of OpenMPI. The MPI implementation / version used to compile ADIOS should be identical to that to compile and run AdiosStMan. When used on a supercomputer, the MPI compiler wrapper needs to be configured to use GNU C++ compiler by ensuring gcc module is loaded correctly. This is because there are known issues of compiling casacore applications using Intel compiler or Cray compiler.

2.2 ADIOS

The current version of AdiosStMan is developed using ADIOS 1.7.0. Older versions might not work as there have been slight interface changes in recent versions. Users are directed to ADIOS User Manual for information to compile and install ADIOS.

2.3 casacore

The current version of AdiosStMan is developed using casacore 1.7.0. Version 1.5.0 also proved to work, but versions older than that have not been tested. Users are directed to the casacore documentation for information to compile and install casacore.

3 Compile

As a prototype project, AdiosStMan does not use any automatic Al build tools at this stage. Consequently, users need to ensure dependent libraries are exported to corresponding system environmental variables before compile AdiosStMan from makefile. The following table shows the environmental variables needed for making AdiosStMan, where users are directed to corresponding user manuals for more information for libraries that are not required

directly by AdiosStMan.

Environmental Variable	Value	Required by
CPATH	$ADIOS_ROOT_DIR/include	AdiosStMan
CPATH	$MXML_ROOT_DIR/include	ADIOS
CPATH	$CASACORE_ROOT_DIR/include/casacore	AdiosStMan
CPATH	$CFITSIO_ROOT_DIR/include	casacore
CPATH	$WCSLIB_ROOT_DIR/include	casacore
LIBRARY_PATH & LD_LIBRARY_PATH	$ADIOS_ROOT_DIR/lib	AdiosStMan
LIBRARY_PATH & LD_LIBRARY_PATH	$MXML_ROOT_DIR/lib	ADIOS
LIBRARY_PATH & LD_LIBRARY_PATH	$CASACORE_ROOT_DIR/lib	AdiosStMan
LIBRARY_PATH & LD_LIBRARY_PATH	$CFITSIO_ROOT_DIR/lib	casacore
LIBRARY_PATH & LD_LIBRARY_PATH	$WCSLIB_ROOT_DIR/lib	casacore
PATH	$MPI_ROOT_DIR/bin	ADIOS & AdiosStMan

4 API

AdiosStMan is developed in the hope that it is compatible with any other casacore storage managers when working in serial, while providing an additional multi-process parallel mode. Therefore, any existing code is supposed to work with AdiosStMan by simply changing the definition of the storage manager to AdiosStMan, as long as all CASA table features the code uses are in the support list, which is shown below.

Datatypes	All except String and String Array
Column Types	Only Scalar columns and direct array columns
Write	Yes
Rewrite	No
Read	Yes
Add Rows	No
Add Columns	No

4.1 Serial

The following listing shows a simple case of creating an AdiosStMan object and bind it to a CASA table containing a scalar column 'index' and a direct array column 'data'. For the full example code please refer to **AdiosStMan/examples/wAdiosStMan.cc**.

```
54    // define a storage manager
55    AdiosStMan stman;
56
```

```
57    // define a table description & add a scalar column and an array column
58    TableDesc td("", "1", TableDesc::Scratch);
59    td.addColumn (ScalarColumnDesc<int>("index"));
60    td.addColumn (ArrayColumnDesc<float>("data", data_pos, ColumnDesc::Direct));
61
62    // create a table instance, bind it to the storage manager
63    SetupNewTable newtab(filename, td, Table::New);
64    newtab.bindAll(stman);
```

4.2 Parallel

AdiosStMan supports writing a single CASA table from multiple MPI processes simultaneously. In order to maximize the IO throughput, this is done through bypassing the CASA table lock mechanism. More specifically, to get AdiosStMan writing in parallel, only the master MPI rank should be directed to the intended CASA table files, while slave ranks should be fooled to write their table files into a temporary path. In the meanwhile, once AdiosStMan is instantiated and bound to a CASA table in a multi-process scenario, the master MPI rank will broadcast the path and name of the ADIOS file associated with the intended CASA table files to slave ranks. In such a way, the CASA table system is working as if every MPI rank is handling completely independent CASA tables, although actually the data coming from all ranks is put into the same ADIOS file that is associated with the master rank's CASA table. Once the table is finished, the temporary table files generated by slave ranks can be removed, since the master rank's table files have contained everything needed to rebuild the CASA table.

AdiosStMan is aware of being instantiated in a multi-process MPI code in both implicit and explicit cases. The implicit case means that by the time AdiosStMan is instantiated, no MPI related code has been executed at all. Then AdiosStMan will initialize MPI and pass any necessary configurations to the ADIOS library. In this case when an AdiosStMan object is destructed, it will finalize MPI as well. On the other hand, if MPI has been already initialized when AdiosStMan is being instantiated, then AdiosStMan will carry on using the existing MPI configuration. In this case the AdiosStMan object will not finalize MPI while being destructed.

The following listing shows the section of a parallel code that is different from the serial usage.

```
71 void write_table(){
72
73    AdiosStMan stman;
74
75    // define a table description & add a scalar column and an array column
76    TableDesc td("", "1", TableDesc::Scratch);
77    td.addColumn (ScalarColumnDesc<uInt>("index"));
78    td.addColumn (ArrayColumnDesc<Float>("data", data_pos, ColumnDesc::Direct));
79
```

```
80    // create a table instance, bind it to the storage manager & allocate rows
81    SetupNewTable newtab(filename, td, Table::New);
82    newtab.bindAll(stman);
83    Table casa_table(newtab, NrRows);
84
85    // define column objects and link them to the table
86    ScalarColumn<uInt> index_col(casa_table, "index");
87    ArrayColumn<Float> data_col(casa_table, "data");
88
89    // each mpi rank writes a subset of the data
90    for (uInt i=mpiRank; i<NrRows; i+=mpiSize) {
91        index_col.put (i, i);
92        data_col.put (i, data_arr);
93    }
94
95 }
96
97
98 int main (){
99
100   MPI_Init(0,0);
101   MPI_Comm_rank(MPI_COMM_WORLD, &mpiRank);
102   MPI_Comm_size(MPI_COMM_WORLD, &mpiSize);
103
104   // generate filenames for slave processes
105   // these files are not used later on, so just put them
106   // into /tmp and clean them up when job is finished
107   if(mpiRank>0){
108       stringstream filename_s;
109       filename_s << "/tmp/v" << mpiRank << ".casa";
110       filename = filename_s.str();
111   }
112
113   // put some data into the data array
114   indgen (data_arr);
115
116   write_table();
117
118   MPI_Finalize();
119
120   return 0;
```

```
121 }
```