Astronomy and Computing 16 (2016) 146–154

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

AdiosStMan: Parallelizing Casacore Table Data System using Adaptive IO System

R. Wang [a,*], C. Harris [b], A. Wicenec [a]

[a] International Centre for Radio Astronomy Research, The University of Western Australia, M468, 35 Stirling Hwy, Crawley, Western Australia 6009, Australia
[b] Pawsey Supercomputing Centre, 26 Dick Perry Ave, Kensington, Western Australia 6151, Australia

ARTICLE INFO

Article history:
Received 21 October 2015
Accepted 5 May 2016
Available online 21 May 2016

Keywords:
Data format
Casacore
ADIOS
Storage manager
Parallel I/O
CASA

ABSTRACT

In this paper, we investigate the Casacore Table Data System (CTDS) used in the casacore and CASA libraries, and methods to parallelize it. CTDS provides a storage manager plugin mechanism for third-party developers to design and implement their own CTDS storage managers. Having this in mind, we looked into various storage backend techniques that can possibly enable parallel I/O for CTDS by implementing new storage managers. After carrying on benchmarks showing the excellent parallel I/O throughput of the Adaptive IO System (ADIOS), we implemented an ADIOS based parallel CTDS storage manager. We then applied the CASA MSTransform frequency split task to verify the ADIOS Storage Manager. We also ran a series of performance tests to examine the I/O throughput in a massively parallel scenario.

Contents

1. Introduction

Modern radio astronomy is entering the era of big data. With the radio interferometry technique, the data production of a telescope scales quadratically with the number of antennas (Wang and Harris, 2013). Next generation radio telescopes, such as the Square Kilometre Array (SKA), will consist of thousands of antennas. Together with the prospective high frequency resolution and long baselines, it will produce petabytes of data per day (Dewdney, 2013). The traditional method of processing radio astronomy data on desktops or workstations will have to be replaced by large scale clusters or supercomputers, in order to tackle the compute and data challenge.

* Corresponding author.
E-mail addresses: jason.wang@icrar.org (R. Wang), chris.harris@pawsey.org.au (C. Harris), andreas.wicenec@icrar.org (A. Wicenec).

http://dx.doi.org/10.1016/j.ascom.2016.05.003
2213-1337/

However, most software packages and data formats traditionally used in radio astronomy data processing are poorly parallelized. For instance, one of the most widely used software package for radio astronomy, the Common Astronomy Software Applications (CASA) library, used to only have OpenMP level parallelism that can collaboratively work on a maximum of a single node. Introducing inter-node parallelism has long been planned, but it is only very recently that CASA has started to support MPI for some of its algorithms. For data I/O, the Casacore Table Data System (van Diepen, 2015a) (CTDS) built-into the casacore library, used to only allow one process to write data to a table at a time. It can be worked around by virtually concatenating multiply physical tables into a single logical table, so that from the logical table's perspective, it can be written in parallel. However, there still has not been concrete solutions for parallel writing at the physical table level, which could be important for optimizing the I/O performance at the SKA scale.

Therefore, considering the data volume that next generation radio telescopes will generate, there is a possibility that the current data I/O technique becomes invalid. It could be because of the enormous amount of files beyond the capability of any filesystems. Or it could be because that data is too large to duplicate or move around, but can only be put into as few tables as possible where any applications can get access simultaneously. This paper focuses on parallel data I/O issues of the Casacore Table Data System. We will first look into the CTDS architecture, and then summarize the parallel I/O techniques that can potentially be used to parallelize CTDS without breaking the underlying architecture. Following this, we will present our parallel CTDS storage manager design and implementation, as well as related testing results.

2. Casacore Table Data System (CTDS) & storage backends

The casacore library (van Diepen, 2015b) is a set of common radio astronomy functions implemented in C++, originally derived from the AIPS++ library. As it forms the core algorithm component of the CASA (Common Astronomy Software Applications (McMullin et al., 2007)) library, it is currently one of the most widely used radio astronomy libraries. Casacore has a well designed data I/O subsystem, namely the Casacore Table Data System (CTDS), which can handle terabyte-scale astronomy data efficiently. The Casacore Table Data System provides an abstract layer to data by defining the storage manager interface. Each column, or a group of columns, of a CASA table can be assigned with a storage manager that handles the actual I/O operations interacting with the storage backends. This essentially allows the substitution of the built-in storage managers with custom ones that are based on third-party storage backends.

A CTDS table can be operated from multiple writers, but only in serial. While one writer is writing data into a CTDS table, it locks the table and prevents others from writing until it is finished. The built-in storage managers of casacore are also designed to comply with this serial writing mechanism. As of now, in the newest version of casacore, 2.0.3, the lock mechanism can be disabled at compile time. However, errors are still seen when multiple processes operate on a single table through this non-locking access. There has been another workaround in CASA 4.5, which virtually concatenates multiple physical CTDS tables into one logical table, and thus enables parallel writing at the logical table layer. At the physical table level, there still has not been concrete solutions for parallel writing. Taking advantage of the storage manager interface, a promising direction would be to implement a custom storage manager based on a parallel storage backend and enabling parallelism at the physical table layer.

There are several categories of parallel storage backends that potentially work with CTDS, including filesystem based data formats, databases, database engines, and distributed object stores. Traditionally, filesystem based data formats have been the most effective way of dealing with scientific data. This is because they usually target a particular science scenario, or an abstract group of science scenarios. A reflection of this is that these data formats are usually designed and optimized for numerical arrays, where other storage backend categories traditionally give very little consideration. Moreover, without involving overheads by reserving functionalities for index, query, security and so on, data files can be accessed for reading and writing quite efficiently, sometimes approaching the theoretical I/O bandwidth of storage hardware. Some good examples of filesystem based data formats are FITS (Flexible Image Transport System (Wells et al., 1981)), HDF5 (Hierarchical Data Format Version 5 (Folk et al., 1999)) and ADIOS (Adaptive IO System (Jin et al., 2008)). In these data formats, FITS has been most commonly used in radio astronomy. However, there has long been a debate on how far FITS can still go since it does not support parallel I/O (Wells, 1997; Price et al., 2014). HDF5 and ADIOS are more suitable solutions for future systems that desire a parallel storage backend. Especially, ADIOS is essentially designed for large scale parallel IO, and in some cases, proved to be 1000 times faster than other parallel I/O libraries (Lofstead et al., 2009).

Over the last decade, another trend for managing scientific data is to use databases or a hybrid database and data file approach. This is because using the traditional filesystem based data formats, the management of metadata is usually relying on directory hierarchies and file names, which may not be easily scalable when it comes to petabyte scale data (Gray et al., 2005). However, very few traditional databases are actually optimized for, or even compatible with, the major form of scientific data, numeric arrays. Our preliminary investigation shows that one of the few databases that target scientific data, SciDB (Paradigm4 Inc., 2015), performs one to two orders of magnitude slower than ADIOS or the casacore built-in storage manager, when given hundreds of large floating point arrays. An improved solution is to directly use high performance database engines, or storage engines. A good example is WiredTiger (MongoDB, Inc., 2015b), which has been recently adopted in MongoDB (MongoDB, Inc., 2015a) as one of the optional underlying storage engines. This bypasses the database interface layer, and thus could possibly provide higher throughput than using a fully functioned database.

Similar ideas can also apply to filesystem based approaches. Currently some parallel distributed filesystems, Lustre for instance, are essentially based on object stores, while modern object stores, such as Ceph (Red Hat, Inc., 2015), do also provide filesystem interfaces. This implies the possibility that higher throughput could also be expected by bypassing the filesystem interface and directly using object stores as the storage backend. One difficult problem of this approach is that data centres do not always provide such low-level interfaces to end users, but rather usually a filesystem interface only. In the meanwhile, setting up and maintaining such an object store on dedicated facilities at a sensible scale could be a considerable expense. This largely limits the universality of a pure object store based storage backend model.

In this paper, we mainly focus on filesystem based data formats, as they proved to achieve relatively high throughputs at a reasonable cost in terms of both software development and hardware requirement. For the selection of the storage backend technique, we will summarize one of our representative preliminary investigations and present in the next section.