Astronomy and Computing 24 (2018) 117–128

Contents lists available at ScienceDirect

Astronomy and Computing

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

Full length article

Pipeline Collector: Gathering performance data for distributed astronomical pipelines

A.P. Mechev [a,*], A. Plaat [b], J.B. Raymond Oonk [a,c], H.T. Intema [a], H.J.A. Röttgering [a]

[a] Leiden Observatory, Niels Bohrweg 2, 2333 CA Leiden, The Netherlands
[b] Leiden Institute of Advanced Computer Science, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
[c] ASTRON, Oude Hoogeveensedijk 4, 7991 PD Dwingeloo, The Netherlands

ARTICLE INFO

Article history:
Received 18 April 2018
Accepted 21 June 2018
Available online 30 June 2018

Keywords:
Radio astronomy
Performance analysis
Profiling
High performance computing

ABSTRACT

Modern astronomical data processing requires complex software pipelines to process ever growing datasets. For radio astronomy, these pipelines have become so large that they need to be distributed across a computational cluster. This makes it difficult to monitor the performance of each pipeline step. To gain insight into the performance of each step, a performance monitoring utility needs to be integrated with the pipeline execution. In this work we have developed such a utility and integrated it with the calibration pipeline of the Low Frequency Array, LOFAR, a leading radio telescope. We tested the tool by running the pipeline on several different compute platforms and collected the performance data. Based on this data, we make well informed recommendations on future hardware and software upgrades. The aim of these upgrades is to accelerate the slowest processing steps for this LOFAR pipeline. The *pipeline_collector* suite is open source and will be incorporated in future LOFAR pipelines to create a performance database for all LOFAR processing.

1. Introduction

Astronomical data often requires significant processing before it is considered ready for scientific analysis. This processing is done increasingly by complex and autonomous software pipelines, often consisting of numerous processing steps, which are run without user interaction. It is necessary to collect performance statistics for each pipeline step. Doing so will enable scientists to discover and address software and hardware inefficiencies and produce scientific data at a higher rate. To identify these inefficiencies, we have extended the performance monitoring package *tcollector*[1] (Apache, 2017). The resulting suite, *pipeline_collector*, makes it possible to use *tcollector* to record data for complex pipelines. We have used a leading radio telescope as the test case for the *pipeline_collector* suite. The discoveries made with our software will help remove bottlenecks and suggest hardware requirements for current and future processing clusters. We summarize our findings in Table 1 in Section 3.

Over the past two decades, processing data in radio astronomy has increasingly moved from personal machines to large compute clusters. Over this time, radio telescopes have undergone upgrades in the form of wide band receivers and upgraded correlators (Broekema et al., 2018; Gupta et al., 2017). In addition, several aperture synthesis arrays such as the Low Frequency Array (LOFAR, Van Haarlem et al., 2013), Murchison Widefield Array (MWA Lonsdale et al., 2009; Tingay et al., 2013) and MeerKAT (Jonas, 2009) have begun observing the radio sky, leading to an increase of data rates by up to 3 orders of magnitude (Wu et al., 2013; Davidson, 2012).

As the data acquisition rate has increased, data size has entered the Petabyte regime, and processing requirements increased to millions of CPU-hours. In order for processing to match the acquisition rate, the data is increasingly processed at large clusters with high-bandwidth connections to the data. An important case where data processing is done at a high throughput cluster is the LOFAR radio telescope.

The LOFAR telescope is a European low frequency aperture synthesis radio telescope centered in the Netherlands with stations stretching across Europe. This aperture synthesis telescope requires significant data processing before producing scientific images (Van Weeren et al., 2016; Williams et al., 2016; Smirnov and Tasse, 2015; Oonk et al., 2014). In this work, we will use our performance monitoring utility, *pipeline_collector*,[2] to study the first half of the LOFAR processing, the Direction Independent (hereafter DI) pipeline.

* Corresponding author.
E-mail address: apmechev@strw.leidenuniv.nl (A.P. Mechev).

[1] https://github.com/OpenTSDB/tcollector.

[2] https://gitlab.com/apmechev/pipeline_collector.git.

https://doi.org/10.1016/j.ascom.2018.06.005
2213-1337/

Table 1
A table of all the results presented in Section 3.

Result #	Description
R1	Native compilation of the software performs comparably to pre-compiled binaries on two test machines.
R2	The processing steps do not appear to accelerate significantly on a faster processor or with larger cache size.
R3	Both calibration steps (*calib_cal* and *gsmcal_solve*) show linear correlation between speedup and memory bandwidth.
R4	Disk read/write speed does not affect the completion time of the slowest steps.
R5	Both calibration steps do not use large amounts of RAM despite processing data on the order of Gigabytes.
R6	The *calib_cal* step can suffer up to 20% of Level 1 Instruction Cache misses, while gsmcal only has 5% of these misses.
R7	Both calibration steps are impacted by Level 2 Cache eviction at comparable rates.
R8	The *calib_cal* step stalls on resources 70% of cycles while the gsmcal step only 30% of them.
R9	The *calib_cal* uses the CPU at full efficiency for only 10% of the CPU cycles.

One major project for the LOFAR telescope is the Surveys Key Science Project (SKSP) (Shimwell et al., 2017). This project consists of more than 3000 observations of 8 h each, 600 of which have been observed. These observations need to be processed by a DI pipeline, the results of which are calibrated by a Direction Dependent (DD) pipeline. The DI pipeline is implemented in the software package *prefactor*.[3] The *prefactor* pipeline is itself split into four stages and implemented at the SURFsara Grid location at the Amsterdam e-Science centre (SURF, 2018; Mechev et al., 2017). The automation and simple parallelization has decreased the run time per dataset from several days to six hours, making it comparable to the observation rate. To better understand and optimize the performance of the *prefactor* pipeline, we require detailed performance information for all steps of the processing software. We have developed a utility to gather this information for data processing pipelines running on distributed compute systems.

In this work, we will use the *pipeline_collector* utility to study the LOFAR *prefactor* pipeline and suggest optimization based on our results. To test the software on a diverse set of hardware, we will set up the monitoring package on four different computers and collect data on the pipeline's performance. Using this data, we discuss several aspects of the LOFAR software which we present in Table 1. Finally we discuss the broader context of these optimizations in relation to the LOFAR SKSP project and touch on the integration of *pipeline_collector* with the second half of the data processing pipeline, the DD calibration and imaging.

1.1. Related work

Scientific fields that need to process large datasets employ some type of data processing pipelines. Such pipelines include e.g. solar imaging (Centeno et al., 2014), neuroscience imaging (Strother et al., 2004) and infrared astronomy (Ott, 2010). While these pipelines often log the start and finishing times of each step (using tools such as pegasus-kickstart (Vöckler et al., 2006)), they do not collect detailed time series performance data throughout the run.

At a typical compute cluster the performance of every node in a distributed systems is monitored using utilities, such as Ganglia (Massie et al., 2004). These tools only monitor the global system performance. If one is interested in specific processes, then the Linux procfs (Bowden, 2009) is used. The procfs system can be used to analyze the performance of individual pipeline steps. Likewise, the Performance API (PAPI, Mucci et al., 1999) is a tool which collects detailed low level information on the CPU usage per process. Collecting detailed statistics at the process level is required to understand and optimize the performance of the LOFAR pipeline and we will integrate PAPI into *pipeline_collector* in the future. Finally, DTrace (Gregg and Mauro, 2011) is a Sun Microsystems tool which makes it possible to write profiling scripts that access data from the kernel and can be used to monitor process or system performance at run time with minimal overhead. As DTrace was not installed on either of the processing clusters, we have not used it to monitor the pipeline's performance.

The Linux procfs system and PAPI record data which is already made available by the Linux kernel. This option incurs insignificant overhead as it uses data the kernel and processor already log. Likewise PAPI reads performance counters that the CPU automatically increments during processing. These profiling utilities can run concurrently with the scientific payload without using more than 1%–2% of system resources. Their low overhead is why we choose to use them to collect performance data.

Other tools for performance analysis such as Valgrind (Nethercote and Seward, 2007) collect very detailed performance information. This comes at the expense of execution time: running with Valgrind, the processing time slows by up to two orders of magnitude. As such, we do not use Valgrind along the LOFAR software.

2. Measuring LOFAR pipeline performance with pipeline_collector

We developed the package *pipeline_collector* as an extension of the performance collection package *tcollector*. *pipeline_collector* makes it possible to collect performance data for complex multi-step pipelines. Additionally, it makes it easy to record performance data from other utilities. A performance monitoring utility that we plan to integrate in the future is the PAPI tools described in Section 1.1. The resulting performance data was recorded in a database and analyzed. For our tests, we used the LOFAR *prefactor* pipeline, however with minor modifications, any multi-step pipeline can be profiled.

tcollector is a software package that automatically launches 'collector' scripts. These scripts are samples of the specific system resource and send the data to the main tcollector process. This process then sends the data to the dedicated time series database. We created custom scripts to monitor processes launched by the *prefactor* pipeline (Appendix A.1).

In this work, we use a sample LOFAR SKSP dataset as a test case. A particular focus was to understand the effect of hardware on the bottlenecks of the LOFAR data reduction. To gain insight into the effect of hardware on *prefactor* performance, the data was processed on four different hardware configurations (Table 2). As

[3] Available at https://github.com/lofar-astron/prefactor.