ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THBPL03

A NEW ACS BULK DATA TRANSFER SERVICE FOR CTA*

Mauricio Araya†, Leonardo Pizarro, Rodolfo Stephano Castillo,Marcelo Ignacio Jara, Horst von Brand
Universidad Técnica Federico Santa María, Valparaíso, Chile
Igor Oya
DESY Zeuthen, Zeuthen & Humboldt University Berlin, Berlin, Germany
Etienne Lyard
University of Geneva, Geneva, Switzerland

Abstract

The ALMA Common Software (ACS) framework provides Bulk Data Transfer (BDT) service implementations that need to be updated for new projects that will use ACS, such as the Cherenkov Telescope Array (CTA) and other projects, most of them having quite different requirements than ALMA. We propose a new open-source BDT service for ACS based on ZeroMQ, that meets CTA data transfer specifications while maintaining retro-compatibility with the closed-source solution used in ALMA. The service uses the push-pull pattern for data transfer, the publisher-subscriber pattern for data control, and Protocol Buffers for data serialization, having also the option to integrate other serialization options easily. Besides complying with ACS interface definition to be used by ACS components and clients, the service provide an independent API to be used outside the ACS framework. Our experiments show a good compromise between throughput and computational effort, suggesting that the service could scale up in terms of number of producers, number of consumers and network bandwidth.

INTRODUCTION

The Cherenkov Telescope Array (CTA) control software decided to reuse ALMA Common Software (ACS) [1], which is a distributed framework for high-level control specially tailored for array control [2]. ALMA itself is not developing new features nor upgrading actively their technologies because the observatory is currently in production phase. Further development has been taken over by the ACS Community Branch [3], a fully open source branch of ACS, with the objective of upgrading the software without the ALMA constraints, replacing closed components by open source alternatives and working on packaging to streamline installation and simplify future development.

Even though CTA will use ACS as its base distributed-system broker, the requirements are different from ALMA. Besides the straight-forward differences (e.g., hardware drivers, wavebands, two sites), the CTA instruments will generate much larger volumes of data in bursts (events), and not more or less continuously like in ALMA antennas. Therefore, the CTA software development group needs to upgrade this framework to avoid obsolescence and to meet the specific requirements of the planned instruments that CTA will deploy. Specifically, ACS must support the Observation Execution System (OES) work package, which is the control system of CTA that includes high-level control and coordination of the detectors, instrument control and configuration, triggering system for detecting target events, data acquisition pipeline and graphical interfaces for science and engineering use cases [4].

Between these new challenges for the framework, we found much more demanding bulk-data transfer (BDT) requirements, because CTA will use cameras with very large readout targets, and their respective servers will be not designed to store the data locally (e.g., [5]). Moreover, concurrent events in the array are expected (indeed events need to be stereoscopic to be considered as a detection), and simulations predict large event rates (> 10 KHz) [6]. Consequently, data needs to be transferred rapidly to a data center that will store the results. This paper presents our proposal for a new BDT service that uses state-of-the-art technologies to achieve this goal.

ACS BULK DATA TRANSFER SERVICE

The current next-generation BDT service (BDT-NG) used in ALMA is based on RTI Distributed Streaming System [7], and it is tailored for their transfer rates and the package sizes [8], which differ substantially compared to the needs of CTA. Also, it is closed-source solution which involves licensing issues. A previous open-source version was available, but it is considered deprecated and it is not supported any more, leaving the ACS community branch without an implementation of a key feature of its architecture.

Despite these technical differences, the core characteristics of the BDT service remain similar. The control software architecture of CTA, specifically the Data Acquisition subpackage (DAQ) [9], defines the requirements of the BDT service that suits CTA, and proposes using the ZeroMQ library for this task. ZeroMQ would be used for the data transference among servers, with the use of Protocol Buffers for data serialization and language abstraction, through its Interface Description Language (IDL) interface[1].

A ZeroMQ-based Bulk Data Transfer Service

Both ZeroMQ and Protocol Buffers are off-the-shelf tools that would allow in combination to transfer CTA's bulk-data

* Work supported by Centro CientÍfico Tecnológico de ValparaÍso (CONICYT FB-0821) and Advanced Center for Electrical and Electronic Engineering (CONICYT FB-0008)
† mauricio.araya@usm.cl

[1] This IDL is different from the one used in CORBA-based systems such as ACS.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THBPL03

over distributed systems without using the nowadays outperformed CORBA communication proposed by the general ACS architecture nor using proprietary tools like in ALMA. Moreover, this combination also provides support for multiple programming languages, replicating one of the main advantages of ACS.

Therefore, we developed a ZeroMQ-based Bulk Data Transfer Service (BDT-Z) prototype for ACS towards meeting the CTA requirements. The main characteristics of our prototype are:

- It is an open-source alternative for the ACS community branch, completing a missing key service of the ACS architecture.
- It is built under DAQ sub-package requirements, allowing the sub-package to focus on protocol buffer serialization and data types, while BDT-Z manages data transfer.
- It can work as a standalone BDT or be integrated to any other project that needs a BDT service, because besides providing an ACS interface it includes an independent C++ API's included.

The BDT-Z prototype has been developed in C++ in order to address the producer-consumer problem faced by BDT. It has two connection points among Sender (i.e., producer) and Worker (i.e., consumer). The first is the data transfer, where the Push-Pull pattern was selected. Previously, Router-Dealer pattern was implemented, but proposed changes in ZeroMQ will deprecate it in future versions. For data control, Publisher-Subscriber pattern is used, allowing the sender to notify every worker when the data stream has ended. It could also be used to notify other kind of events, but ACS has its own notification service.

The architectural design of BDT-Z was based on BDT-NG, where the notion of data transfer are called `Flows`, which are grouped into `Streams`. This grouping can be done regarding the different data each byte is related to. To operate data serialization, BDT-Z can send raw byte arrays or protocol buffer serialized messages. This two initial types can be easily extended to other serialization options, as the base data type is raw content.

Currently, we have not yet fully integrated BDT-Z to ACS. While we comply with the basic interface proposed by ACS for BDT, we have not mounted the appropriate test environment to validate our implementation. However, we have tested the service using stand-alone executable files. In the next section, we present these results.

PERFORMANCE EXPERIMENTS

The key concern in a BDT service is the throughput. The objective is to obtain a high bit per second count while minimizing the number of messages, in order to harness the most of the packet size (related to the MTU). The higher the bit per second speed is, the more data we can send through the "wire", while having a precise count of messages, optimizes the CPU use. In theory, a higher packet size gives more throughput and less CPU usage, but in practice this is not always true, and the packet size is not something we can always choose.

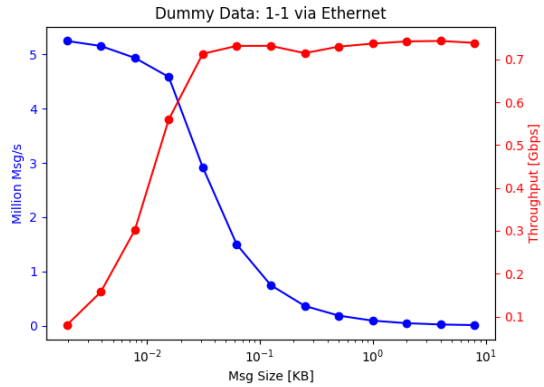

Figure 1: Throughput and msg/s of dummy data between 2 servers for different packet sizes.

The preliminary experiments of this section provide insights on the behaviour of the BDT-Z service by measuring the throughput and messages per second passed. Please note that firewalls were kept up and the CentOS 7.3 operating system was used in all the servers.

1-1 over Ethernet

The first experiment consist in sending 1 million messages over a Gbit Ethernet network as fast as possible with a 1-to-1 configuration (one sender, one worker). We have used two general-purpose desktop workstations for this experiment. The test was run for different messages size (2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 and 8192 bytes). The whole test was performed 3 times.

Dummy Data To establish a baseline throughput we sent randomly generated "'dummy" messages of various sizes between two idle systems, with a standard network configuration (MTU: 1500).

The result of the baseline measurement is shown in Figure 1. This correspond to the ideal behavior: the messages per second drop from 5 millions using a few bytes to less than 1 million with packet sizes around a kilobyte. On the other hand the throughput increases until saturation near 700 Mbps. We suspect that the main reason for not arriving to the network limit was the moderate power of the workstation's cores.

Image Data A more realistic scenario is presented in Figure 2, where we generate an image of an equivalent size to the constant one million messages depending on the packet sizes. Now, the best throughput is not when using the larges packet size, but only 512 kilobytes, reaching near the 70% of the bandwidth anyway. Between the multiple causes of the degradation of the throughput curve for the large packet sizes we found:

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THBPL03

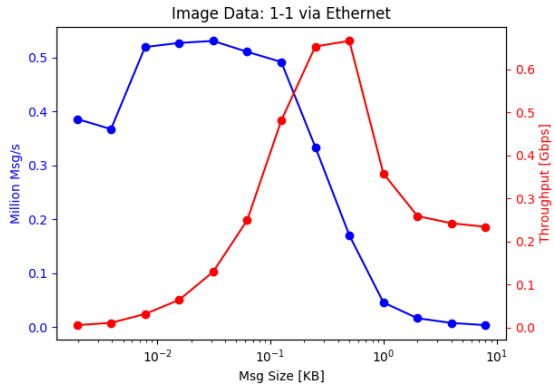

Figure 2: Throughput and msg/s of an image between 2 servers for different packet sizes.

- By using a file rather than dummy messages, the `fread` function produce a growing impact on disk I/O as the packet becomes larger.
- The CPU use for creating packages that might have surpassed de MTU 1500 size, considering protocol overhead, causes a slower production of packages to be sent over the network and therefore, a lower throughput for byte sizes over 1024.

Even though these issues will only improve in more realistic scenarios (e.g., SSD, fiber cables and jumbo-frame switches), the experiment allow us to observe how a throughput-curve degradation looks like.

N-N over Infiniband

The experimental configuration of the past section is clearly not very realistic: CTA's camera servers will need a much more performant network and consuming servers. In this section we present the bulk-data transfer experiments between two rackable servers with 24 cores (1200 MHz) and 128 GB of RAM each, connected through an 40/60 Infiniband network over FDR. The servers are also connected through Ethernet interfaces for normal traffic, while Infiniband is only used for bulk-data transfer. In this paper, we show only experiments using a symmetric number of senders and workers for simplicity. We have run the same experiment than the previous section 24 times in this setup, increasing by one the number of cores used at each iteration.

Infiniband vs Ethernet Just for completeness, we report in Figure 3 the 1-1 results for this high-performance setup. Comparing to Figure 2, we can see that we achieve more that 7 Gbps when using very large packet sizes of 1 MB in Infiniband, but comparable values in terms of msg/s and throughput are found at the 1KB level (and below) but without degradation.

Two Workers If we increase the number of workers to two (and the senders also to two), we can observe the natural throughput increment in Figure 4. While the messages also

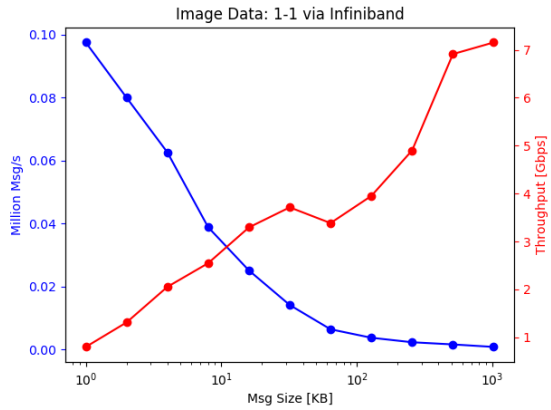

Figure 3: 1-1 throughput and msg/s using different packet sizes through Infiniband.

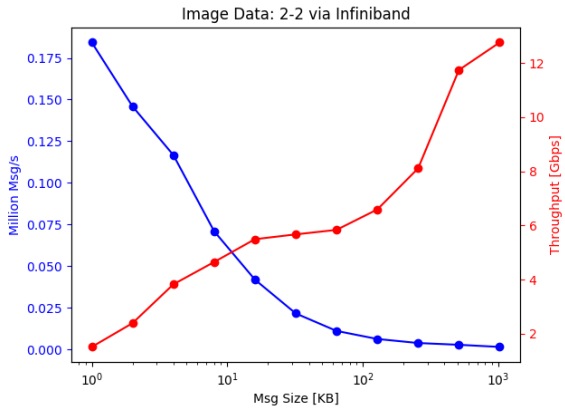

Figure 4: 2-2 throughput and msg/s using different packet sizes through Infiniband.

increases, now we have two senders to generate and two workers to process them. We can observe that both curves are very similar to the 1-1 experiment, showing no degradation.

Twelve Workers This behaviour remains similar for the following experiments, as can be shown in Figure 5, where we can observe that the maximum throughput has increased, but the curves remain very similar to 1-1 or 2-2.

Thirteen Workers In Figure 6 we observe that for 1 MB packet sizes, the throughput started to fall compared to 512 KB. The same behaviour can be found consistently in 10 of the 11 next experiments, while not important perturbations are found in the msg/s curve. Please note that in this experiment we are using (at most) the 70% of the bandwidth, so the network limitation is not the problem. Indeed, with more than 20 workers (and 20 senders) and with packet sizes of 512 KB we approach to the 100% of network usage. Therefore, we need to study the nature of the degradation of the throughput when using too large packet sizes, in order to setup the appropriate size depending on the number of workers and senders available.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THBPL03

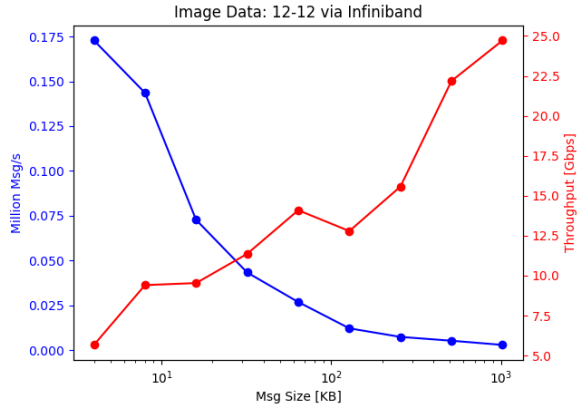

Figure 5: 12-12 throughput and msg/s using different packet sizes through Infiniband.

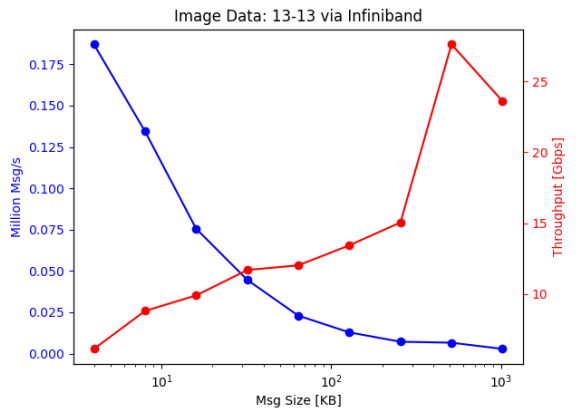

Figure 6: 13-13 throughput and msg/s using different packet sizes through Infiniband.

Workers and Packet Sizes In Figure 7 we summarize the results for the last five points in the previous figures. In other words, we show the evolution of different packet sizes as we increase the number of workers/senders. The degradation effect described above can be clearly observed as the 512 KB curve surpass the 1 MB one at the 13-13 experiment. Moreover, We can observe an interesting valle for the throughput near the 12-12, not only for the 1 MB packet size, but for all of them. Also, the clearly logarithmic behaviour of the msg/s is also perturbed near 12-12.

NEXT STEPS

First of all, we need to complete the integration to ACS, modifying ACS's database (i.e., the CDB), to run BDT-Z with ACS on startup and measure the overhead that RPC communication will cost. Also, some modifications to the IDL governing the BDT-Z need to be suggested to the ACS community branch. Once the integration is complete, new measurements can be taken to complete a full behavioral analysis of the BDT-Z performance.

In terms of the experiments, we need to explore first the impact of using simultaneous streams for several senders

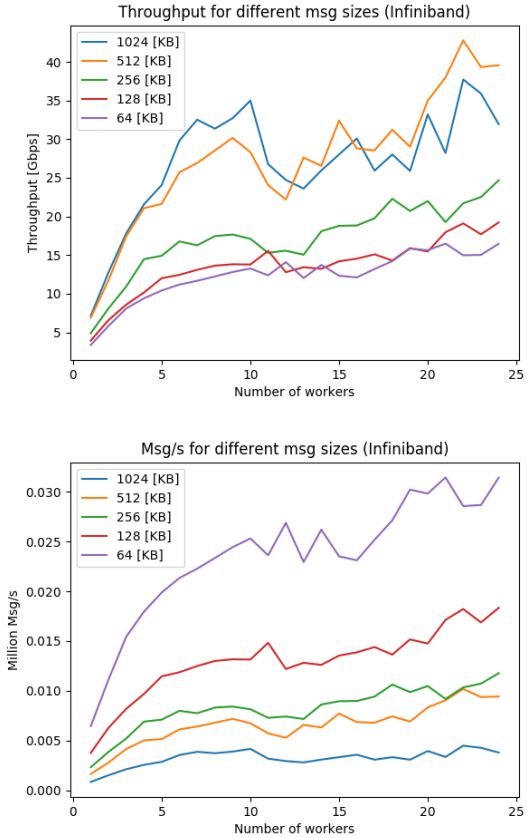

Figure 7: Summary of throughput and msg/s for large packet sizes with respect to the workers.

nodes and several worker nodes. Also we need to understand the impact of asymmetrical senders and workers, to estimate the number of worker nodes that we need with respect to the actual number of camera servers given a network infrastructure.

Alternative approaches using Distributed Streaming System such as Apache Kafka has been considered, but concerns regarding overhead have discouraged it in favor of ZeroMQ. However, the use a Distributed File System such as HDFS (based on Google File System) is a promising direction that should be considered and measured against the ZeroMQ solution in the future.

ACKNOWLEDGMENT

This research was possible due to CONICYT-Chile fundings, specifically through the Basal Project FB-0821 and Basal Project FB-0008.

REFERENCES

[1] I. Oya *et al.*, "The software architecture to control the cherenkov telescope array," *Proc.SPIE*, vol. 9913, p. 15, 2016.

[2] G. Chiozzi *et al.*, "The ALMA common software: a developer-friendly CORBA-based framework," in *Advanced Software, Control, and Communication Systems for Astronomy*, H. Lewis and G. Raffi, Eds., vol. 5496, Sep. 2004, pp. 205–218.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THBPL03

[3] *ACS Community Branch*, http://acs-community.github.io/.

[4] M. Füßling *et al.*, "Status of the array control and data acquisition system for the cherenkov telescope array," *Proc.SPIE*, vol. 9913, pp. 1–12, 2016. DOI: 10.1117/12.2233174.

[5] V. Conforti *et al.*, "Software design of the astri camera server proposed for the cherenkov telescope array," *Proc.SPIE*, vol. 9913, pp. 1–9, 2016. DOI: 10.1117/12.2230016.

[6] M. Actis *et al.*, "Design concepts for the cherenkov telescope array cta: An advanced facility for ground-based high-energy gamma-ray astronomy," *Experimental Astronomy*, vol. 32, no. 3, pp. 193–316, Dec. 2011. DOI: 10.1007/s10686-011-9247-0.

[7] *RTI Distributed Streaming System*, https://www.rti.com/products/dds/.

[8] B. Jeram, G. Chiozzi, R. Tobar, R. Amestica, and M. Watanabe, "Reimplementing the bulk data system with DDS in ALMA ACS," in *Proceedings of ICALEPCS2013*, 2013.

[9] E. Lyard *et al.*, "Modern middleware for the data acquisition of the Cherenkov Telescope Array," *ArXiv e-prints*, Aug. 2015. arXiv: 1508.06473 [astro-ph.IM].