

LDDS - The LSST DAQ DDS Study

1 Introduction

The purpose of this document is to provide a high-level description of a study that will help evaluate the viability of DDS implementations for use in the LSST camera DAQ system. The particular area of interest within the DAQ system is the transfer of image data from the central DAQ crate(s) to the local (mountain/base camp) large processing farm.

This document provides information about our understanding of relevant camera components and basic requirements, our understanding of the problem at hand, and an initial discussion of the study that needs to be done.

2 Background Information

Provided here is a summary of what we understand about relevant aspects of the camera.

2.1 Camera

Hierarchical structure: rafts(21)/CCDs (9)/segments(16)

Each raft is formed into a tower that amounts to a 144 Megapixel camera unit.

The segment is the smallest addressable unit. This is the smallest part of an image that can be transferred or requested.

There are 3024 total segments in the camera, each should be producing about **2MB per image**. It appears that the value read from each pixel might fit in about **two bytes**.

What we heard during our visit with Mike Huffer was that when including overheads, the image size could be as high as 7GB.

The wave front and guider data are somewhat different in organization and size. These two types do not represent the bulk of the processing and will not be included as separate cases. The latency involved in delivery and processing this data, however, may be important to study at a later time.

2.2 Readout

Cadence for image acquisition is ~40 seconds, with two images (a visit) taken during this period from the 3.2 Gigapixel camera. Each image is read out in two seconds or **6GB per 2 seconds**. There is a goal to read out one image every second.

The important real-time constraint is that all the image data read out in two seconds be delivered to all client applications in the same two seconds.

Data rate due to images is **3GB/s** to the farm processing applications. Date rate is **~1MB/s** at the low end (3GB/3000 segments) and **~2MB/s** at the high end (6GB/3000) to any one segment processing application.

2.3 Early Processing

The DAQ hardware and software reads out entire images and does crosstalk corrections. It also buffers up to two days of raw images in flash memory. The major task is to deliver crosstalk-corrected detector segments in read-out time to the main processing farm of ~6000 cores outside the DAQ.

2.4 Farm processing

How the data is processed within base-camp applications is outside the scope of the DAQ. The DAQ system is designed to be insensitive to downstream processing. It is just required to deliver the data to clients using the DAQ network.

There are two types of downstream applications: dedicated segment processors (to gain parallelism in image processing) and other image processing applications. The processing model is stream based. The applications are long-running and receive their assigned segment and do processing on it. In other words, a particular application will be run on a particular core on the cluster. This application will continually receive the same segment number that it is subscribed to. Essentially a (multicast) path will be established from segment publisher to this subscriber application.

It is assumed that applications will not request entire images to be streamed. The problem of complete image assembly at the back end of the farm applications is not covered by DAQ. The problem is application specific.

3 Problem

A major challenge is to reliably deliver segments (and perhaps higher-level structures) to applications within the required time windows.

Another is to provide a good barrier between the main DAQ components and farm processing applications so that problems in the applications will not cause failures of the main data taking components.

DDS appears to be a good match for handling data transfer problems from the DAQ crates to the farm. Some of the main features of DDS that make it an attractive choice are:

- clean separation between data producers, data, and data consumers,
- full set of QoS options that can be used to satisfy real-time constraints,
- implementations that can exist in an embedded system environment.

DDS has two important concepts, which are topics and keys. The topic identifies a type of message and a key distinguishes instances of a message type. Mapping application messages to topics and keys is an important task.

Features of DDS that need consideration:

LDDS - The LSST DAQ DDS Study

- reliance on multicast technology,
- no notion of point-to-point communication
- there are many configuration options and system organizational and structural choices,
- topic and key assignment and performance implications,
- there are tradeoffs between robustness and performance.

One possible mapping of the LSST problem to DDS is to define each segment as a topic.

4 Study

A study is needed to help determine whether or not DDS technology can satisfy the DAQ processing and transfer requirements for LSST camera data. Critical factors are the number of subscribers and the number of publishers. DAQ publishers are expected to be relatively fixed at $O(1000)$. DM subscribers are expected to be one or more per DAQ publisher. The number of servers for science data is going to be fixed at about 33.

There are three quantities that need to be measured for this study:

- 1) performance: bytes per second sent by publishers and received by consumers.
- 2) efficiency: CPU, memory, and network resources required to send and receive data at full rate.
- 3) latency: The time it takes for a sent message to first appear at the receiver.

In this study we will measure these quantities for realistic combinations of publishers and subscribers running concurrently. The performance and efficiency numbers need to be reported at both the receiving and sending side. LSST is primarily interested in calculating (performance, efficiency, latency) for realistic combinations of (subscribers_per_publisher, total_publishers), given a network and node configuration. Since our test stands will only represent a slice or single partition of the real system, what we want from our evaluations is to confidently be able to predict the resource usage and performance of DDS in the full system.

As stated earlier, the expected data rate into a segment processor is $\sim 1\text{MB/s}$ or $\sim 2\text{MB/s}$. This segment processor is expected to reside on a single core of a multicore machine. Since the core count is expected to be 48, 96, or even higher, the tests we run should include rates up to 200MB/s of image traffic per node at the high end, and 50MB/s at the low end. At the low end, gigabit ethernet might suffice for some of the tests. The high end will require high speed networking.

Porting and testing under RTEMS has been identified as a critical task for this project and will need to be given sufficient priority up front.

Of course the CPU and network resources used will depend on the chosen NIC and the network and switching elements. We could benefit from producing a good metric for CPU utilization that allows the performance model to be independent of CPU architecture upgrades. Some of the obvious ones that may not have this property are: network processing time used within a time interval, cycles necessary for network processing during a time interval, or ratio of application work completed with network processing over without. The last would be a measure of typical application interference caused by network processing.

4.1 Additional requirements and goals

Subscribers can exist on any core of any node.

Subscribers must be able to subscribe to a single segment of the camera. In other words, each segment must form a unique data channel through use of topics and keys.

No data loss can be tolerated. This might be a difficult requirement to fulfill in a general way given the proposed one-way movement of data from mountain top to base camp. The buffering options within DDS may need to be tuned for the network and CPU core loads.

Use multicast as a transport protocol with datagrams. This is the appropriate protocol to use for this problem. The RCE (reconfigurable cluster element) support multicast. The COB (cluster on board) supports a special low-overhead implementation of the socket library on top of TCP/IP. Datagram sends can be performed with extremely low overhead.

Desire to build up the smallest packet (an MTU) for sends. This could possibly reduce the buffering overheads and packet sequencing and ordering overheads associated with the TCP/IP stack (mostly at the receiver side). This could also make the network traffic more easily tuned due to the finer granularity of requests at the user level.

Quickly (within the same one or two second window) report on and act on real-time violations (latency, transfer rates). For this study, this means local (on the node) recognition of the problem and creation of the data and the conditions that caused the failure.

The DM fabric will be separate from the DAQ fabric. In other words, the base camp nodes will be connected to two networks, the DM and the DAQ. The DAQ network will have well-known traffic pattern and will be heavily managed. The DM network will be unpredictable.

The DAQ will work in collaboration with the observatory and the DM on the DAQ fabric/network to take sure that the DAQ traffic patterns can be accommodated. This collaborative effort will ensure a robust and reliable transport of image data.

A secondary concern is that the DM needs access to buffered raw data and this raw data is needed at the base camp within 15 seconds of collection. Needs of DM are to continuously move the 6GB of image data pulled out of the buffered data pool within 15 seconds. The DM uses this for their 24 hour data product

cycle. The DM imposes the 15 second requirement. The DSI (data storage interface) does this data movement and is currently specified to be DDS. This could provide an additional load of 400MB/s to particular subscriber applications.

4.2 Test stand

We will establish a network organization that matches the performance and properties of a subset of the proposed DAQ network infrastructure. It was proposed during discussions with LSST that we start with approximately ten worker nodes connected via gigabit ethernet to several single-board computers (SBCs). We prefer to generate new test stands as the needs of the project grow. The sequence we propose is the following. The exact configurations will heavily depend on the existing equipment that is available when this project is ready.

- (1) For the RTEMS port, use VMs to run RTEMS (currently VirtualBox and qemu are supported). Use SBCs as a fall-back in case the VMs cannot be made to operate properly.
- (2) Develop study applications using standard x86_64 Linux nodes. Test these application on standard gigabit ethernet cluster nodes.
- (3) Acquire 8-16 existing nodes with 10G or better Infiniband NICs plus a 24 port switch. Use these nodes and network for improved (higher performance) throughput tests. Run tests using the code and VMs developed in (1). Use the IPoB drivers to run the standard TCP/IP stack over the Infiniband network. We suspect that these Infiniband-connected nodes will be readily available since they are in wide-spread use on some of the Fermilab projects.
- (4) Acquire two slot ATCA crate and COB equipment from SLAC for further RTEMS porting and testing.
- (5) Acquire an 8 RCE COB phase II board from SLAC. Use 10G ethernet ports available on Fermilab switches to connect this board to ~6-8 standard Linux nodes that could be made available with 10G ethernet NICs. It may be possible to use the RTM ethernet port to connect the 8 RCEs to external 10G ethernet equipment, or use the DTM switch connections to connect to each of the standard Linux nodes.

We may need to purchase/locate SFP modules to connect into existing equipment. If 10G ethernet NICs are difficult to locate, one possibility is to use newer Infiniband cards, with 10G ethernet firmware installed.

This plan allows us to incrementally move to more complicated platforms as the equipment becomes available and as we need the complexity. It also greatly minimizes the new equipment costs for this project.

Because multicast is used, we will need to be careful to test at large enough configurations to verify that we understand necessary partitioning constraints and also that we do not miss performance degradations due to node count.

4.3 Failure Conditions

In addition to performance goals, we want study how DDS applications respond to failure conditions and the effects of other system constraints. The most important ones include:

- equipment and process failures
 - death of node
 - death of a producer or consumer application
 - death of an internal DDS daemon
 - out of resource problems (memory, disk)
 - network outage
- installation and management of the software
- Data loss and data errors during continuous and bursty operation

4.4 Vendors to be Tested

The testing we will be able to achieve will depend on the availability of source code and how complex the porting task becomes. RTI and OpenDDS are both available under vxWorks and possibly under QNX. The one-tier architecture of these products makes them good candidates for RTEMS porting. We are told that obtaining source code from RTI is very difficult. We are also told that OpenDDS might be the least performant DDS implementation. OpenSplice uses a two-tier architecture to maintain access to the network resources. This configuration may be well-suited for the Linux base-camp cluster.

We propose first investigating OpenDDS for the RTEMS port and running OpenSplice DDS on the Linux cluster nodes. After initial work is completed with these products, we will test using RTI in place of OpenSplice on the Linux nodes, and investigate porting the OpenSplice code to RTEMS. We suspect that this port will be more difficult than that of OpenDDS due to the two process model reduced to running under one address space within RTEMS.

We are hoping the trial licenses of RTI will be enough to perform all necessary tests. For OpenSplice, we will use the licenses we currently have and the community edition of the source code.

5 Additional Notes

The metrics defined in the previous section will be affected by the QoS configuration of DDS and the organization of the messages. The affects of these configuration options will need to be better understood. In addition to the ones specified in the previous section, we will include information on how available DDS implementations scale (memory, CPU, network use) with:

- number of available topics and keys
- number of nodes

LDDS - The LSST DAQ DDS Study

- number of network domains and partitions (multicast usage)
- message size

All of these factors will impact the three main measurements reported to LSST.

Questions to be considered during the investigation:

- what sort of configuration is needed (e.g. topology, QoS)?
- what is the structure of the topics and keys?
- what DDS architecture was chosen for this problem? (multi-tiered or self contained)
- what is the target MTU size, datagram size, and IDL message structure size?

One of the target payload data rates will be segments transferred at a rate of **1MS/s** and at **2MB/s** to DM subscribers.

The DDS architecture might have a large influence on the robustness and scaling. A two-tier approach, such as the one offered by OpenSplice, might produce the coalescing and message aggregation that is necessary to achieve the desired throughput and latency.

Since DDS configuration will affect performance and efficiency, we will likely need to optimize this configuration for each (subscribers_per_publisher, total_publisher) combination. This includes setting buffer limit at sender, receiver, and within the network elements. It also includes during the delivery rate out of senders and adjusting the packet, datagram, and message structure sizes.

Because of the zero data loss requirement, we may need to consider what it would take to maintain a redundant copy of the data at the receivers. This can be done using the internal structures of DDS implementations or delivery to multiple receivers. Packet drops due to slow or faulty applications can be recovered at the receiver end with this scheme, possibly with little overhead.

One of the overall goals here is to construct a system that reliably pushes data out of RCEs, through the network, and input complaint base-camp applications without network protocol feedback and flow control. A second goal is to consume a predictable, fixed amount of memory resources on the RCE, that does not depend on the number of applications interested in receiving the data.

We are likely going to want the largest possible MTU based on conversation with people using 10G ethernet at Fermilab.

System requirement: RTEMS must support multicast sockets. It is essential to the efficient use of DDS. We are told that RTEMS has an excellent TCP/IP stack with POSIX interfaces and that the DAQ has already demonstrated the use of this stack and of the multicast features.

6 Project Plan

Phases, deliverables, and tasks are defined in this section. The text in a cell for a given release and deliverable described what will be provided during that release.

Sheet1

Release		V1	V2	V3	V4	V5	V6	V7
Phase	Deliverables							
Porting	Dev Environment	i386 VM				RCE		
	DDS port	VM OpenDDS			Feature additions	RCE OpenDDS	Feature additions	Second vendor
	Tests	Functionality (multicast)			Startup, Shutdown, configuration			
Facilities	GigE test stand	Linux 1x1	Linux/RTEMS 1x1	Linux 8x8 node	Linux/RTEMS 8x8 node			
	IPoB test stand				Linux/RTEMS 10Gib 8x8			
	RCE test stand					Linux/RTEMS 10Ge 1x1	Linux/RTEMS 10Ge 1x8	
Experiment Development	Definitions	Parameters, metrics, messages, and performance model	Metrics and output data formats	Failure modes, fault injector, QoS violation reporting				
	Tools	Configuration generator	program launcher	configuration optimizer				
	Publisher	Simple Data Generator Application	RTEMS release	Full App	Changes, Full+QoS checks	Changes	Changes	
	Subscriber	Simple Results collector Application		Full App	Changes, Full+QoS checks	Changes	Changes	
Data Analysis	Report, Runs, Tools		summary scripts	validation scripts	Runs, Report	Runs, Report	Runs, Report	Report

j