CMB-S4

DAQ/CONTROL TRADE STUDY 2021

CMBS4-doc-750-v1

Author(s)	Role/Organization	Date
Laura Newburgh	CMB-S4 DAQ L2 Lead	06/07/2021
Nathan Whitehorn	CMB-S4 DAQ L2 Deputy Lead	06/07/2021
Cosmin Deaconu	CMB-S4 DAQ L3	06/07/2021

REVISION HISTORY

Version	Revision Date	Description of Changes
v1	06/07/2021	first draft

This document has been officially released by the CMB-S4 project office if:

- A unique DocID number is printed on the bottom-right corner of each document page.
- The document approval page displays a full set of valid e-signatures.
- An e-signature audit trail is appended to this document.

APPROVALS

Name	Project Role	Signature	Date
John Corlett	Project Director		
Murdock Gilchriese	Deputy Project Director		
John Carlstrom	NSF Principal Investigator Project Scientist		
Matthaeus Leitner	DOE Project Manager		
Jeff Zivick	NSF Project Manager		
Robert Besuner	Project Engineer		
Brenna Flaugher	Technical Integration Scientist		
John Ruhl	Instrument Scientist		
Julian Borrill	Data Scientist		

TABLE OF CONTENTS

1. Purpose And Scope

This document outlines a comparison of current software packages and benchmarks them against the DAQ/Control/Monitoring/Alarm requirements.

2. References

References used within this document are detailed in the following subsections. Unless otherwise noted, the information contained in this document takes precedence over information contained in referenced materials.

2.1. Applicable Documents

The following documents are considered as part of this document to the extent specified herein. If not explicitly stated differently, the latest issue of the document is valid.

AD#	Document Title	Document No.	Version	Precedence

2.2. Reference Documents

The following documents contain additional information useful for providing history and context for the material contained in this document.

RD#	Document Title	Document No.	Version

2.3. Acronyms

Acronym	Full text

2.4. Definitions

Term	Definition

3. CMB-S4 DAQ/Control Trade Study

Executive Summary

The goal of this document is to provide a clear set of criteria to use in selecting the CMB-S4 DAQ package. We considered a variety of astronomical DAQ+control+monitoring software packages developed for current or future experiments, and compared them against our requirements. All packages met some or most of the requirements, and thus commonly met requirements were not used to distinguish between software packages. None of the packages considered provide clear advantages over the baseline design (SO OCS). The next steps are to solicit feedback from the collaboration for additional considerations we have not captured here, and use the combination of this document and collaboration feedback to form a ‘software downselect’. Collaboration input is critical to this decision because the user experience for the DAQ software is essential in widespread adoption. Not all packages have to (currently) meet requirements or user requests, work towards those requirements can be included in the DAQ work package going forward.

DAQ/Control Software Package Testing description

We considered 6 software packages, and some commercial packages:

- ALMA
- CLASS
- EPICS
- GCP
- LSST
- Simons Observatory (SO) OCS - Baseline design
- Commercial packages

There are three primary considerations we used when comparing software packages: requests from S4 users, requirements laid out by the DAQ team, and additional considerations (also from input from the collaboration). Currently we place the highest weight on meeting the requirements as our benchmarking criteria. A software package does not have to meet all requirements to be selected for CMB-S4 as long as there is a reasonable development path for meeting all requirements within the risk, budget, and schedule criteria for DAQ.

Each software package was installed by at least one of the core DAQ team members (if publicly available) and documentation read to assess which requirements were met (or not met) by the code. The full set of notes for each package are given as Appendices in this document; we drew from these notes to build our grading rubric and produce a soft recommendation.

Requests from S4 users

- Any clear dependencies on libraries that might be deprecated or poorly supported
- HK quantities plotted against each other

Additional Considerations

- Level of reuse : has it been adopted by others successfully?

Elucidation of Requirements

We compared each DAQ/control software package against a set of requirements first presented at the review (Directors Review Requirements). These requirements are broken into three categories: requirements for control, requirements for the data acquisition, and requirements for monitoring/alarms. Many requirements are met by all packages, however generally these were the requirements one might expect from any astronomical control/DAQ framework: centralized control, remote mode support, etc. Below we list all requirements.

Control :

Common requirements: The following requirements were essentially met by all packages considered.

Distributions	The software shall be versioned, collaborative via eg git, issue tracking, reviews
Deliverable	Shall provide software for Controls/DAQ compatible test stands (ie Detector Modules, LAT, SAT (TBR))
Telescope interface	The telescope vendor shall provide telescope control such that S4 control will be high-level (at the level of az/el itinerary delivered by network/serial)
Refrigerator commanding	DAQ shall provide a carefully considered plan for command of the refrigerators, with equipment safety made a particularly high priority, and in consultation with the needs of the collaboration and the refrigerator vendor.
Centralized control	The Telescope and Site shall include an Observatory Control System (OCS) shall be the central coordination facility that controls the delivery of high quality data from scan sequencing, providing continuous set-points for the relevant hardware devices and provides the operator with the necessary feedback to efficiently and safely monitor the system operation.
Local or remote control	The OCS shall support local and remote user interaction with the telescopes, receivers, and other hardware.
Support Observer or scheduled commands	Each telescope, receiver, and associated hardware shall be controlled through OCS either directly by a telescope operator or through a scripted set of commands.
Telescope tracking	The OCS shall control the tracking/guiding function of each telescope within the performance specified for the Tracking requirements.
Support multiple modes	OCS shall be designed and constructed to support the following operational modes: fully automated, calibration, manual observing, engineering and maintenance
Receiver commanding	The OCS shall command each receiver's bolometer readout crate, such as calibration, tuning, fast cadence modes, and data acquisition
Other' commanding	OCS shall support commanding other components, including but not limited to: calibration equipment, heaters, compressors, hardware configuration`
Commonality	Identical software system between Chile and SP, and labs
Scheduling	DAQ shall support the execution of observations, which include ordered operations such as move to a location, tune detectors, begin a scan pattern, etc
Modular	The software shall allow configuration of DAQ/Control objects per lab and changeable by the user
Firmware-free	The DAQ system shall not involve firmware
All hardware interfaces	The DAQ interface to all hardware shall be digital

Distinguishing requirements: The following were not necessarily common across all packages and so will be used to distinguish between them.

Scalability	The DAQ software must work at a variety of scales: single object, lab, testing, receiver, full observatory
User friendly	Framework and support shall be structured so that control/Acq and monitoring for new components can be added by any qualified collaboration member (eg middle-ware is generally hidden from user)
Broad	The DAQ software shall be written so it can be used on capable lab computer, with compatible OS,

hardware/computer applicability	package manager, compiler/library version, etc on the native system
Open source	Software shall be open source with no licensed libraries, packages, etc
Code language	Code language shall be limited to a few well-known options, including C++ and Python, with the expectation that python shall be the primary language for lab developers
User access control	Access to observatory control by users shall be managed and prioritized so that competing commands will not be possible
Messaging layer	The messaging and routing layer for the control architecture shall be commercial and designed for distributed systems

Areas for future development/work: We did not scale test each package. Although we do not have concerns for most packages, this testing is required in validating the selected software package.

Downtime/uptime	The system must not cause more than 0.1% loss of data from downtime
Verified to work at scale	The system shall be verified to work at S4 scale, acceptable via emulators and/or simulations

Data Acquisition :

Common requirements: The following requirements were essentially met by all packages considered (or are hardware requirements and thus not necessarily tied to the choice of software).

Data Description	The OCS shall acquire all required housekeeping data, including metadata, needed for the scientific analysis of the survey data as well as, at a minimum, the following: health of operating systems, temperatures, pressures, loads, status, rate, weather.
Data Rate	The Data acquisition system shall support data rates up to 4 (TBR) Gbits/s per site
Data Loss	Data loss shall be less than 0.001% per packet
Timing	DAQ shall provide absolute timing and any required clocks for synchronization to all systems, sent via fiber to each telescope platform, within the timing phase noise jitter requirements specified by the detector readout system. The timing breakout at each telescope may include IRIG, 10MHz, PPS.
Error trace/logs	Logs and error tracing shall be recorded
Network	The DAQ shall provide a network design sufficient for transferring up to 10Gbps from telescopes to the recording computers
Network	The domains for high speed and low-speed DAQ may be segregated
Synchronicity	The software architecture shall be built to support asynchronous data collection of all data types

Timing	The timing system shall meet the requirements from detector readout for phase noise of XX dBc/Hz (requirement not yet received from readout)

Distinguishing requirements: The following were not necessarily common across all packages and so will be used to distinguish between them.

Data format	DAQ shall specify the data format, and the OCS shall record data in that format.
Time Stamp	DAQ shall provide PTP absolute timing across fiber to all systems, that time stamp must be propagated through the DAQ/control system.
Network	The network shall use TCP/IP as a universal protocol
Meta-data	Data shall include meta-data and shall be recorded (such as bolometer tuning results)

Monitoring and Alarms :

Common requirements: The framework and approach for live monitoring and alarms was generally quite different between the packages due to differing requirements between the various experiments. As a result, all of our requirements are distinguishing.

Distinguishing requirements: The following were not necessarily common across all packages and so will be used to distinguish between them.

Health & Monitoring	OCS shall provide real-time monitoring of all housekeeping and diagnostic data, which may also include real-time monitoring of detector statistics (on transition, rms). The monitoring shall at minimum allow visualization of temporal changes in monitored quantities. Decimated views over the entire history will also be supported.
Health & Monitoring	The OCS system shall support remote monitoring by authorized personnel using a web browser compliant device, including, but not limited to, smart phones, laptops, and tablets.
Alarms	DAQ shall provide an alarm system based on housekeeping data and any detector statistics provided by the readout crate metadata.
Alarms	Alarm notifications shall be hierarchical depending on the severity of the alarm, and site dependent based on notification systems already in place.
Health and Monitoring	The monitor shall support real-time data with update rates of no longer than 5s
Health and Monitoring	The monitor shall support decimated views of historical data, decimated at cadences of >1 week
Health and Monitoring	The monitor must be able to support monitoring quantities of up to 100,000 fields/sec
Monitoring	The graphical monitoring software must be easily configurable for lab, observatory, and testing from the browser itself

Areas for future development/work:

One request is to monitor one quantity against another, instead of just a time stream of quantities (eg, a live monitor that can plot one housekeeping variable against another). This requires, in all cases, co-sampled synchronous data, which is not part of the raw data acquisition. For any package, enabling this feature in a live monitor will need to be an area of development for DAQ if the request is elevated to a requirement on the DAQ package.

Comparison of Distinguishing Requirements

We begin by scoring each software package against the distinguishing requirements. Although not all requirements should be weighted equally, this provides a raw ranking of the packages and allows us to focus on the higher-ranked packages that meet more requirements and hence would be likely to need less work to be adopted by CMB-S4.

Numerical scoring:

0=Yes
1=Partial/some concerns
2=No

For this scoring rubric, a ***lower*** score is preferred.

	ALMA	CLASS	EPICS	GCP	LSST	SO OCS
Scalability	0	1	0		0	0
User friendly	2	1	2	2	0	0
Broad hardware/computer applicability	2	1	1	2	0	0
Open source	1	2	0		0	0
Code language	1	2	1		0	0
User access control	2	0	0		0	0
Messaging layer	1	1	1		0	0
Data format	1	1	1		1	1
Time Stamp	1	1	1		1	0
Network	1	0	0		0	0
Meta-data	0	1	1		0	0
Health & Monitoring (real time+historical)	2	1	0		0	0

Health & Monitoring (web browser)	2	1	1		0	0
Alarms (det stats)	0	1	0		0	0
Alarms (hierarchical)	0	1	0		0	0
Health and Monitoring (<5s)	2	1	0		1	0
Health and Monitoring (decimated)	2	1	1		1	0
Health and Monitoring (100k fields/s)	1	1	0		1	0
Monitoring (configurable)	2	1	0		1	0
TOTAL	**23**	**19**	**10**		**6**	**1**

Some comments on least competitive packages

All packages met the majority of requirements, so these comments are focused on the disadvantages compared to the baseline design. None had clear advantages over the baseline design.

ALMA:

- Development began over 10 years ago, as a result various choices (eg for messaging layer) are outdated or proprietary and substantial development would be required to use it for another ~20 years. This work may be undertaken by ALMA but won't be driven by CMB-S4 requirements.
- Poor user-friendliness, documentation, installation experience. Appears unlikely that an 'average' user would be able to easily add new data streams.
- Monitoring was inadequate and would need substantial development.

CLASS:

- Code infrastructure would need significant development: the code is not open source, documentation is sparse, and various things are hardcoded which would need to be tracked down and generalized (file paths and network addresses, etc.)
- Timing propagation would need to be updated to meet CMB-S4 architecture
- Data is written to dirfiles, which do not scale to CMB-S4 data requirements. It is unclear how embedded that choice is in the rest of the architecture.
- Graphical monitoring based on KST, which is 'maintained' by a single member of the CMB community. This would need substantial development, or wholesale adoption of a new monitoring scheme.

EPICS:

- EPICS is widely adopted, but as a result is complex, has many legacy components, and its own specialized jargon (IOC, PV, CWS…). As a result, a significant amount of training with a steep learning curve seems to be required to understand how to develop a piece of it. The large ecosystem also means there are many modules that do nearly the same thing but the tradeoffs are not always apparent and so recommendations (and eventual implications of a given choice) may not be clear.
- User-friendliness and ability to easily add new hardware is problematic

GCP:

Comments on most competitive packages

Two packages are clearly competitive compared to the rest: Simons Observatory OCS and LSST/Vera Rubin Telescope. We'll use the notes from the testing to more directly compare these two:

LSST/Vera Rubin Telescope:

Commercial software dependencies: Communication middleware is commercial DDS from OpenSplice by PrismTech (free and paid version). This may limit us to CentOS 7.

Data storage: Not discussed. As an imaging telescope with a heavy emphasis on real-time map-based CCD data products, FITS files are the storage of choice, it was not clear how embedded that choice is in the software. Metadata storage for LSST is captured by their Engineering and Facility Database, unclear how embedded the choice is for us as well.

Scaling between lab and observatory: While it is clear this would work at a full observatory scale, It is unclear if this system is easy to stand up and maintain in ~20 different testing labs. Is there a compact version of this software? Setup seems quite involved at a first glance. Deployment at the site is entirely done through a CI/CD pipeline. Speculate: perhaps the integration testing aspect of this fills the role of running elements in the lab for testing before deployment?

Monitoring and Alarms: Monitoring through the LSST Operators visualization environment, but the 'refresh' rate, historical data perusal, maximum number of fields per second, and ease of configurability were not described and this could be a concern. They have a 'watcher' component that is written so the alarm rules are easy to write and understand. Alarms do need to be acknowledged.

Time stamp: Timing was not well described in the documentation. Although I am certain they are propagating excellent GPS-based timing, how data sets get time tagged and how that is propagated into the data streams is not clear.

Other notes:
May need to actively discourage labview and java. The software itself is heavily Java based (based on EXO software).

Simons Observatory OCS (Baseline design):

Commercial software dependencies: Communication middleware is Crossbar.io, additionally uses grafana for live monitoring display and (optionally, though preferred) docker for containerization.

Data storage: Data format for all data (bolometer, HK, metadata) is SPT-3G (custom), with a serialization software dependency (Cereal). Writing to this format requires some packages that may be frustrating to the average user (boost in particular), however that is containerized in docker. Because the format is custom, average lab users will require read-in scripts to parse the files correctly.

Monitoring and Alarms: Decimation is currently handled by influx; this would likely change if a different back-end is adopted to allow more flexible plotting from DAQ (HK quantity against HK quantity).

Conclusion

We considered a variety of astronomical DAQ+control+monitoring software packages developed for current or future experiments, and compared them against our requirements. All packages met some or most of the requirements, and so commonly met requirements were not used to distinguish between software packages. None of the packages considered provide clear advantages over the baseline design (SO OCS). For other considerations (not quite elevated to requirements):

- Reuse: the two most competitive packages are fairly new and have not been re-used beyond the experiment they were designed for, and so this was not used as a distinguishing criteria.
- Software dependencies: LSST/Vera Rubin software had one concerning dependency for it's middleware.
- Flexible HK plotting: This would need to be developed for any of the packages considered.

The goal of this document is to provide a clear set of criteria to use in selecting the CMB-S4 DAQ package. The next steps are to solicit feedback from the collaboration for additional

considerations we have not captured here, and use the combination of this document and collaboration feedback to form a 'software downselect'. Collaboration input is critical to this decision because the user experience for the DAQ software is essential in widespread adoption. As noted, not all packages have to currently meet requirements or user requests, those can be included in the DAQ work package going forward.

Appendices:

Appendix A: ALMA

DAQ Reviewers: Abby Crites, Cosmin Deaconu

Relevant documentation:

https://www.almaobservatory.org/en/alma-software/
https://confluence.alma.cl/display/ICTACS/ICT+ALMA+Common+Software
(in particular the workshop material)

Requirements met:

Control

- **Centralized Control:** ALMA Common Software (ACS) provides centralized control of distributed software. Current "notification channel" backend is CORBA, which they correctly recognize as ancient and basically deprecated. Seems to be a current project to replace CORBA with ActiveMQ while keeping the same ACS API.
- **Local or remote control:** Yes, at least via CLI and some Java applications.
- **Support Observer or Scheduled Commands:** Yes, a client can be interactive or long-running loading scheduled commands.
- **Support Multiple Modes**: Yes, by choosing which components to run.
- **Receiver Commanding, Other commanding:** Yes, support for device components.
- **Scalability:** Yes, scales from single client to large observatory (ALMA). However, it seems bulk data transfer does not use CORBA for a lot of components, in favor of some proprietary thing called DDS from RTI (which is also C++ only).
- **Commonality:** Yes, the same base can be used (with different components)
- **Scheduling:** Yes, can have scheduler components.
- **Modular:** Yes, different sets of services etc. can be configured.
- **Firmware-free:** No firmware.

- **All-digital:** yes, that's possible
- **Verified to work at scale**: Yes, by ALMA

DAQ

- **Data description:** Device controllers have a common way of identifying what properties can be archived.
- **Data format:** ACS provides a way to access things to be archived and receive bulk data, but doesn't define how anything is actually archived (so can be however).
- **Error trace/logs:** Very robust logging infrastructure,
- **Network:** Supports bulk data transfer over fast networks and separation of networks
- **Synchronicity:** Yes, everything can be queried asynchronously.
- **Meta-data:** Yes, all properties of devices can be queried / recorded.

Monitoring/Alarms

- **Alarms:** A robust alarm infrastructure within the logging system, with severity levels and priority.

Requirements partially met:

Control

- **Scheduling:** Yes, can have scheduler components.
- **Open source:** Most components are open source, but some proprietary middleware is required for the "improved" bulk data interface (RTI DDS).
- **Code language:** C++, Java and Python are supported for most things, but some interfaces are only available in C++. Moreover, must use DSL for interface definitions and understand Makefiles even for Python modules. Python3 support seems in progress.
- **Messaging layer:** CORBA is designed for distributed systems, but fell out of vogue a while ago and they're trying to replace it.

DAQ

- **Data Rate**: The (proprietary) "new technology" bulk data transfer can reach the required throughput (based on a plot in a presentation), although it was shown for infiniband. The legacy one seems to have problems scaling. (Design requirement is only ~0.5 Gbps).

Requirements not met:

Control

- **User-Friendly:** Setting up ACS is difficult, easier to use pre-configured virtual machine. Writing a new module requires generation of strange directory structure (not so bad) and use of C-like DSL for defining the interface of each module ("Interface Definition Language", IDL). Lots of paths need to be set, etc and overall lots of boilerplate required.

- **Broad hardware/computer applicability:** Only a few operating systems are supported for deployment (officially just EL7, with EL8 under test). More are supported for development.
- **User access control:** Seems to be no attempt at avoiding concurrent access and components must deal with this themselves by making methods reentrant.
- **Network:** Some components seem to use UDP? Hard to tell though.

DAQ

- **Timing:** ALMA design has 48 ms hardware pulse for synchornization of real-time systems, otherwise use NTP.

Monitoring/Alarms

- Built-in monitoring is extremely limited. There are ways to query properties at desired rates, but as far as I can tell, only rudimentary java plotting tools are available as part of the distribution, but this only collects data while running. More complex tools would have to be developed. Also, the rate of monitoring is limited, and latency can be high. It says "up to 1 kHz for a few components" Details: http://www.eso.org/~almamgr/AlmaAcs/Releases/ACS_8_0/Docs/ACS_Sampling_System.pdf

Unknown if requirements met:

Control

- **Uptime**: Not sure. The fact that kill everything (as opposed to stop) is an option in the java control GUI is maybe not promising?

DAQ:

- **Data loss:** ???

Comments on feasibility of meeting all requirements:

The biggest issues I see are:

1. Poor user-friendliness. There are these workshops with tons of material and a lot of documentation, but the user experience seems poor to me, and much of the documentation is very old. I feel like this is unlikely to improve.

2. Use of either outdated (CORBA) or proprietary (RTI DDS) technology. It looks like there is some attempt to address this, but might be too late on our timescale. The current non-proprietary bulk data streams seem to have reliability / throughput problems.

3. Inadequate monitoring features. Required monitoring services would need to be integrated by us, and even so, we can't reach the desired throughput and latency with the system design.

Additional notes:

- May be difficult to get new data streams without new development?
- 2 FTE working on the ALMA side to keep this software working

Appendix B: CLASS

DAQ Reviewers: Brian Koopman
Relevant documentation:

https://arxiv.org/abs/2012.08433
Summary Slides from 2021-02-12 DAQ Call

Software doesn't appear to be open source based on lack of repos available on their GitHub Page.

Requirements met:

Control

- Centralized control - centralized command script and scheduler that orchestrates individual control scripts using PYRO4.
- Local and remote control - Both through CLI and/or web interface
- Support Observer or schedule commands
- Telescope tracking
- Support multiple modes - Seems to be true
- Receiver commanding - interfaces with MCEs
- Other commanding - interfaces with thermometry, custom VPM hardware, etc.
- Scalability - Has been deployed across both CLASS mounts and 3 if not 4 telescopes by now
- Commonality - not sure how to evaluate this here, but it's been tested in Chile, presumably could also be deployed to SP
- User friendly - web interface seems friendly, not sure about things like adding systemd control scripts with PYRO4 exposed interfaces
- Scheduling
- Broad hardware/computer applicability - Seems runnable on systems with systemd and Python3.5. Can only really confirm running on Ubuntu 16.04 though
- Code language - Some C, but mostly Python, occasional Bash
- User access control - It appears user login to web interface is per user, based on the web interface showing you other users currently on the page. Not sure if limiting controls is in here.
- Modular - Seems true

- Messaging layer - Data seems to be directly written by an NFS server. Then packaged/compressed and sent to NA

DAQ

- Data description
- Data format - Dirfiles
- Error trace/logs - All errors logged to metadata CouchDB
- Synchronicity
- Meta-data - All recorded to CouchDB

Monitoring and Alarms

- Health and Monitoring - Status page on web interface to display real time 24h view of HK data. For longer time periods probably KST on dirfiles is the solution? Not 100% sure though. Remote viewing possible. KST viewing likely through VNC?
- Alarms - Noted that they currently only send to Slack channels, though target could be changed.

Requirements not met:

Control

- Open source - Doesn't seem to be open source

DAQ

- Timing - All derived via GPS timing on CLASS ACU, used in combination with MCE syncbox pulse to assign timestamps to detector data. Noted as a limitation on other hardware accepting PTP timing, since MCE hardware doesn't accept PTP.
- Time stamp - see above

Monitoring and Alarms

- None - Some unknown, see below.

Requirements unsure of:

Control

- Firmware-free
- All hardware interfaces
- Downtime/uptime
- Verified to work at scale - Works on 4 telescopes, not sure about scaling past that?

DAQ

- Data Rate
- Data Loss
- Network - Not sure about 10 Gbps support/network segmentation compatibility. Does use UDP for data sending, TCP for commanding.
- Timing - Sufficient to meet phase noise of ??? dBc/Hz

Monitoring and Alarms

- Alarms - Not sure about hierarchical nature of alarms or details about selected group alerts, etc.
- Health and Monitoring - Is KST sufficient for decimated views of historical data? 100,000 fields/sec?
- Monitoring - Ease of graphical monitoring configuration unknown. How difficult is it to add a new HK field to the status page?

Comments on feasibility of meeting all requirements:

After getting in touch with the authors, they are open to the idea of making it public, but not without cleaning up some things and writing documentation first. That said, the code does hardcode the CLASS infrastructure (file paths and network addresses, etc.), which would need some amount of effort to generalize. Documentation is also sparse, so getting up and running from the current state is likely not straightforward.

The authors call out lack of PTP integration to MCE hardware limiting their decisions on accepting PTP timing in other hardware like the VPM. Their solution uses GPS timing from the ACU, so if the S4 ACU accepts PTP, a similar solution could be put in place?

I think adding new control scripts for pieces of hardware wouldn't be too bad, but am unsure about the ease of adding new graphical monitoring for those new pieces of hardware.

In terms of scaling, there were concerns about the scalability of the data acquisition all writing to a common network file system directory that gets zipped to a dirfile. Since we don't want to use dirfiles, a replacement mechanism for writing to the file format of choice would need to be implemented.

In order to meet requirements the scalability of writing data to disk needs to be solved, it would have to be open sourced, and documentation should be written (likely part of the authors open sourcing the code.)

Additional notes:

During the collaboration meeting parallel session Ryan Herbst pointed out that PYRO4 doesn't scale particularly well, as it requires a great deal of bi-directional sockets for each mirrored instance.

Appendix C: EPICS

DAQ Reviewers: Sasha Rahlin, Cosmin Deaconu

Relevant documentation:

The “best” current documentation seems to be at https://docs.epics-controls.org/en/latest/ Due to how long EPICS has been around and the fact that there are multiple supported versions, there is a lot of documentation in various places, but it is not always easy to tell which of the many bits of documentation lying around are relevant.

Requirements met:

Control:

- **Centralized Control:** All components talk to each other using the specialized EPICS wire protocol. Historically this is “Channel Access” but new versions of EPICS have a new replacement called “pvAccess” which is apparently higher throughput (see e.g. http://epics-pvdata.sourceforge.net/talks/2015/ICALEPCS2015_WEA3O02_TALK.pdf , they claim it can saturate a 10 Gb link!). These protocols support pub/sub, introspection, and all the normal things you’d expect.
- **Local and Remote Control:** Channel Access and pvAccess are network transparent, with a wide variety of clients available.
- **Supported Observer or Scheduled commands:** Many ways to control, including a large variety of client interfaces , from Eclipse-based (*Control Studio*) to Motif-based tools. There is also a domain specific language for a control state machine (*Sequencer)*, and surely other ways to schedule.
- **Telescope tracking:** Can’t find details, but e.g. Keck uses EPICS (and likes it! At least based on this. Keck servo is 40 Hz.
- **Supports multiple modes:** Yes, too many.
- **Receiver Commanding:** Yes, via a “configuration” interface
- **Other Commanding:** Yes.
- **Scalability:** Yes, will scale from a test setup on a laptop to an entire accelerator.
- **Commonality:** Yes, can use in different configurations.
- **Scheduling:** Yes, this is at least possible using with *Sequencer* DSL and I think also using some database interface (which may have some flowcharty GUI).
- **Broad hardware/computer applicability:** The base system (and likely lot of common modules) seem to run fine on a variety of Linux systems as well as Windows and MacOS. But it’s a big ecosystem, surely some modules/extensions have more stringent requirements
- **Open Source:** Yes, at least the base. Some modules for specific hardware support may not be.
- **Code language:** EPICS seems to be implemented in a mix of C and C++, but bindings available in every conceivable language (Python, Perl, JavaScript, Java, C#, MATLAB), and some inconceivable ones (IDL, LabView, PHP).
- **User Access Control:** It is possible to have user access control in EPICS to limit permissions by certain users. It also appears to be possible to lock control fields to avoid concurrent access from multiple clients.
- **Modular:** Yes, (though not necessarily user-friendly)

- **Firmware-Free:** No firmware.
- **All hardware interfaces:** There is support for digital hardware interfaces (???)
- **Messaging Layer:** Uses its own bespoke thing, but is widely-deployed and designed for distributed systems.
- **Downtime/Uptime:** No apparent complaints about reliability
- **Verified to work at scale:** Entire accelerator complexes (e.g. SLAC, J-PARC) with thousands of devices can be controlled by EPICS.

DAQ:

- **Data Description:** EPICS can monitor and record all types of housekeeping values.
- **Data Rate:** Apparently can fill a 10 Gb link with PvAccess
- **Data Loss:** Seems to use TCP for everything but discovery, as far as I can tell.
- **Timing:** Seems possible to deploy precise timing using EPICS (see e.g. https://epics.anl.gov/meetings/2001-05/psi/SLStimingEPICS1.pdf)
- **Time Stamp:** There do appear to be some EPICS users successfully using PTP, based on a cursory search. I don't see any explicit hardware support for PTP modules in the database, but I might be missing it. Some commercial vendors of PTP hardware may support EPICS.
- **Error trace/logs:** There is a way to output/log error information within EPICS
- **Network:** Can support 10 Gb and complex network topologies.
- **Synchronicity:** It seems like almost all input output controllers use the Asyn module for asynchronous data collection
- **Meta-data**: Anything can be exposed as a process variables

Monitoring and Alarms:

- **Health & Monitoring:** A plethora of monitoring clients are available, including React-based clients which should work on mobile devices. There is a huge amount of customizability here.
- **Alarms:** At least one very configurable alarm system available, with severity options.
- **Update latency:** It seems like near-realtime monitoring is standard
- **Update rate:** It is possible to monitor individual fields at up to 10 kHz, I think.
- **Monitoring flexibility:** The monitoring system can be configured for all sorts of settings.

Requirements partially met:

DAQ:

- **Data Format:** The wire protocol is meant to be used with a schema in its specified format (https://github.com/epics-base/pvAccessCPP/wiki/Protocol-Encoding) . It may be possible to encode the DAQ data format in this schema. Even if it's not, it's possible to send arrays of bytes
- **Network:** Data transfer is TCP/IP but broadcast/discovery is UDP

Requirements not met:

Control:

- **User-friendly:** EPICS is a big, complicated system with lots of legacy components and a lot of specialized jargon (IOC, PV, CWS…). A significant amount of training seems to be required to understand how to develop a piece of it. A lot of reliance on domain-specific languages. There are also a lot of ways to do the same thing (you can set up a control interface using React, Eclipse, QT, Motif, among others), and a lot of potential modules for "input output controllers" to be be based off, which is not only not very discoverable, but could also lead to a very heterogenous system without careful planning. It's also difficult from the outside to figure out the relevant merits of different modules/extensions that purport to do the same thing, which means a lot of trial and error may be necessary.

  For reference, here is the tutorial on how to create an IOC (i.e. the piece that talks to some hardware): https://docs.epics-controls.org/projects/how-tos/en/latest/getting-started/creating-ioc.html
  While it's easy enough to follow the directions, it does not strike me as easy to understand.

Comments on feasibility of meeting all requirements:

The main concern here is the complexity of deployment and usability by non-experts. In theory, this can be abstracted away by building some middle-ware on top of EPICS, but at that point, it seems like it defeats the point of using the framework. Maybe there already exist some libraries to do this but if so, it's not obvious from the outside.

Additional notes:

EPICS is obviously capable of running CMB-S4 and very likely exceeds all other considered options in capability, if the right components are chosen. But there is a significant cost even to implementers just to understand all the various components enough to start to make a reasonable design within EPICS. The fact that there are so many ways to do similar things is also a bit concerning, since the probably vary in capability, but there doesn't seem to be an obvious way to find out ahead of time and a poor decision here can have a bad impact.

Appendix D: Generic Control Program (GCP)

DAQ Reviewers: Nathan Whitehorn
Relevant documentation:

"South Pole Telescope Software Systems: Control, Monitoring, and Data Acquisition"
https://arxiv.org/abs/1210.4966

Requirements met:

Requirements not met:

Comments on feasibility of meeting all requirements:

Additional notes:

Appendix E: LSST

DAQ Reviewers: Sasha Rahlin, Brian Koopman

Relevant documentation:

SPIE Proceedings - LSST control software component design (this seems to a bit outdated, but isn't behind the SPIE paywall)
Control Software Architecture Document - Control Software Architecture (the more useful of the two documents)
https://github.com/lsst-ts - Entirely separate Github Organization found for telescope and site software! (Note: 261 repositories!)
Observatory Control Documentation - https://obs-controls.lsst.io/
Python Control Agent Base Object Code - https://github.com/lsst-ts/ts_salobj
Python Control Agent Documentation - https://ts-salobj.lsst.io/

Requirements met:

Control

- Centralized control - Control is centralized on the "Service Abstraction Layer", a communication middleware built on top of a Data Distribution Service (DDS), specifically OpenSplice by PrismTech.
    - Some notes about OpenSplice:
        - There seems to be a free and paid version of this
        - The paid version comes from a company called AdLink - https://www.adlinktech.com/en/dds-community-software-evaluation.aspx

- Local and remote control - Local and Remote control called out as a requirement in section 2.7.1.1 - Access Control, and remote monitoring is called out in section 2.8.1.6 - Remote Monitoring of this requirements document.
- Support Observer or scheduled commands - These are both possible, and there is a scheduler.
- Telescope tracking
- Support multiple nodes - This is supported, each component has its own Commandable SAL Components (CSC). And configuration of nodes is all managed centrally in a fully automated manner.
- Receiver commanding - Should be possible with a separate CSC
- Other commanding - Should be possible with a separate CSC
- Scalability - Seems to scale to many nodes. Unsure about things like data output scalability. I think they write to FITS files, and have a high data rate, as their images are quite large, but data storage isn't discussed in the documents about the control software.
- User friendly - Seems like new CSC are Python objects with documentation. There are many examples available on github here: https://github.com/lsst-ts
- Scheduling - There is a scheduler that can perform these actions.
- Open source - all control code available at https://github.com/lsst-ts
- Code language - Python and C++ are options. Labview and Java are also options (so I've noted that in "requirements not met")
- User access control - This is managed by allow lists that each CSC checks before sending data on the network, and it won't send unless it's allowed to.
- Modular
- Messaging layer - Uses PrismTech OpenSplice for messaging layer

DAQ

- Data rate - It's high bandwidth fiber between sites, so I imagine this would be sufficient, but no numbers are given.
- Data format - Seems to be FITS file based: https://github.com/lsst-ts/pythonFitsfile
- Error trace/logs - every action is logged
- Network - 10 Gbps to recording computers. I imagine this is true, as fiber connects the networks
- Network - Use TCP/IP, this is true
- Network - Domains for high and low-speed DAQ may be segregated. I believe this is true
- Synchronicity - Main part of code uses python's asyncio to support asynchronous data collection
- Meta-data - I think metadata could easily be captured by their Engineering and Facility Database

Monitoring and Alarms

- Health & Monitoring real time monitoring - Yes, all provided via the LSST Operators Visualization Environment

- Health & Monitoring remote monitoring - Yes, provided by the same real time monitoring interface
- Alarms - There is a “watcher” component that sends alerts. The goal being that alarm rules are easy to write and understand
- Alarms (hierarchical severity) - The system does seem to handle this. Description of the watcher system includes details on what happens if alarms occur rapidly after acknowledgements, and other scenarios. All errors must be acknowledged, but in certain situations can be suppressed if they are not resolved yet, but still need to be re-acknowledged.

Requirements not met:

Control

- Code language - Beyond Python and C++ there are options for Labview and Java. So it is not “limited to a few well-known options” as described in the requirement.
- Broad hardware/computer applicability - Not sure on hardware recommendations, however this python SalObj (and thus ADLink OpenSplice) installation page specifies that as of December 2020, only CentOS 7 is supported.

Requirements unsure of:

Control

- Commonality - It’s not clear to me that this deployment is used in individual labs. Perhaps individual components are, or software at a lower level that is used in components.
- Firmware-free
- All hardware interfaces
- Downtime/uptime - not sure how to assess this from documentation available
- Verified to work at scale - Seems likely it would scale, but unconfirmed

DAQ

- Data description - Data format isn’t discussed in detail, though every action is logged to a database. I imagine this is met, just can’t say for sure
- Data loss - unsure how to assess loss based on limited documents
- Timing - not discussed in control docs
- Time stamp - not discussed in control docs
- Timing (requirement for phase noise of XX dBc/Hz) - not discussed in docs I found

Monitoring and Alarms

- Health and monitoring - Not sure of a few of these, how quickly updates are coming in (if faster than 5s, how far back historical data goes, how many fields it can handle, or it’s ease of configurability.

Comments on feasibility of meeting all requirements:

A lot of the architectural design features are similar to other control systems being considered -- there's some middleware layer that does the message passing and communication. Separate "agents" are developed for individual hardware and added to the system by communicating over this middle layer. It does seem like most of these "agent" components are written in Python.

There are several aspects that are unclear to me though, listed in the "Requirements unsure of" section. One being the commonality, I'm not sure how much of this is being used in individual labs. Setup seems quite involved at a first glance. Deployment at the site is entirely done through a CI/CD pipeline. Perhaps the integration testing aspect of this fills the role of running elements in the lab for testing before deployment, though this is speculation on my part.

Additional notes:

Heavily Java based, based on EXO software

Appendix F: Simons Observatory OCS

DAQ Reviewers: Cosmin Deaconu, Abby Crites

Relevant documentation:

https://arxiv.org/abs/2012.10345

Requirements met:

Control

- Open source -
  - yes, easy to install, with clear instructions and also Docker options (does not play well with Podman, but now Docker supports cgroupsV2 so mostly a non-issue)
  - Extensive documentation on installation, using and creating agents
- Centralized control
- Local or remote control
- Support Observer or scheduled commands
- Telescope tracking
- Support multiple modes
- Receiver commanding
- Other' commanding
- Refrigerator commanding
- Scalability -- High data rates tested by Cosmin
  - Verified to work at scale
  - 60,000 bolos for SO
- Commonality?

- User friendly”
  - Yes, test code can be written for hardware by non-DAQ expert
- Scheduling
- Broad hardware/computer applicability: yes
- Distributions
- Code language - Python (usually twisted dialect for clients). In principle, it is possible to expand support to write clients in C++ or JavaScript (through other autobahn libraries).
- User access control
- Deliverable
- Firmware-free - yes
- Telescope interface - ACU agent pushed to the OCS feed
- All hardware interfaces -- yes,
- Messaging layer?
- Downtime/uptime
- Modular -- yes, ocs Agents and ocs Clients (easy to add agents for new hardware)
- Monitoring: live monitoring of housekeeping data and alerting, Grafana

DAQ

- Data Description
- Data Rate
- Data Loss
- Data format - SPT3G data format, serializable frames
- Timing
- Time Stamp
- Health & Monitoring - OCSweb
- Alarms
- Error trace/logs
- Network
- Synchronicity
- Health and Monitoring
- Timing
- Monitoring
- Meta-data

Requirements not met:

None

Comments on feasibility of meeting all requirements:

Additional notes:

Appendix G: Alternative Commercial Solutions

DAQ Reviewers: Nathan Whitehorn, John Joseph, Ryan Herbst

LZ (ignition)
https://inductiveautomation.com/scada-software/
https://inductiveautomation.com/pricing/ignition
Notes from CD: Completely java based. Scripting with jython (2.5?)…