The next generation of low-latency data distribution and stream processing for LIGO

Jameson Rollins

CI Compass meeting
March 22, 2022

Introduction

LIGO is embarking on an effort to **modernize**, **improve**, and **extend** it's low-latency data delivery and stream processing systems.

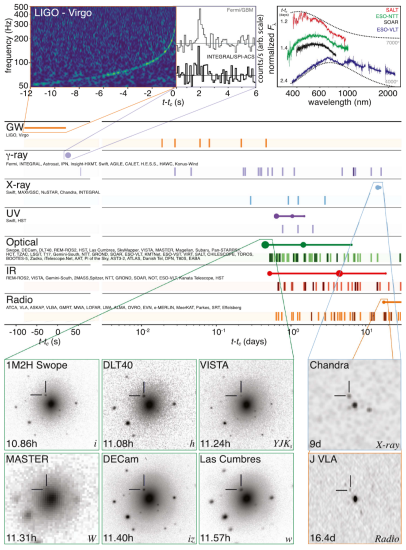
LIGO is acknowledging the importance of the low-latency search/alert pipeline to our overall science goals, and is making a significant investment in the components of the system it is responsible for.

Background

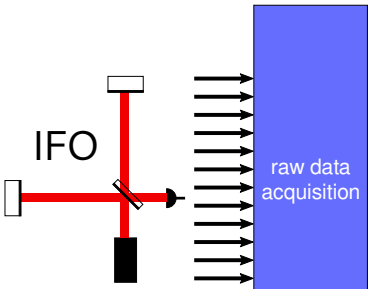
First “low latency” searches developed for Initial LIGO’s sixth science run (S6) in 2009. Came to fruition in Advanced LIGO, leading to the multi-messenger observation of **GW170817** →

Our ability to tell astronomers where to look for exceptional transient electromagnetic events is now maybe our most important reason d’être.

Direct correlation between the speed with which we can issue alerts and the quality of the science.



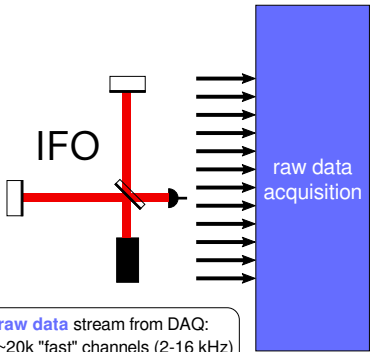
Overview of low-latency data flow



raw data stream from DAQ:
~20k "fast" channels (2-16 kHz)
~200k "slow" channels (16 Hz)

1/16 second block size
~70 MBps

Overview of low-latency data flow



raw data stream from DAQ:
~20k "fast" channels (2-16 kHz)
~200k "slow" channels (16 Hz)

1/16 second block size
~70 MBps

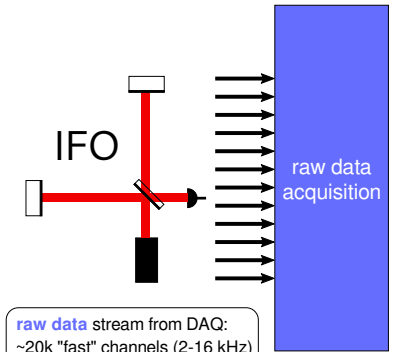
arbitrary number of
analysis applications
produce *asynchronous*
event streams or other
data products

search

analysis

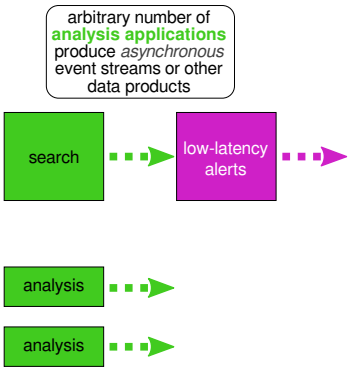
analysis

Overview of low-latency data flow

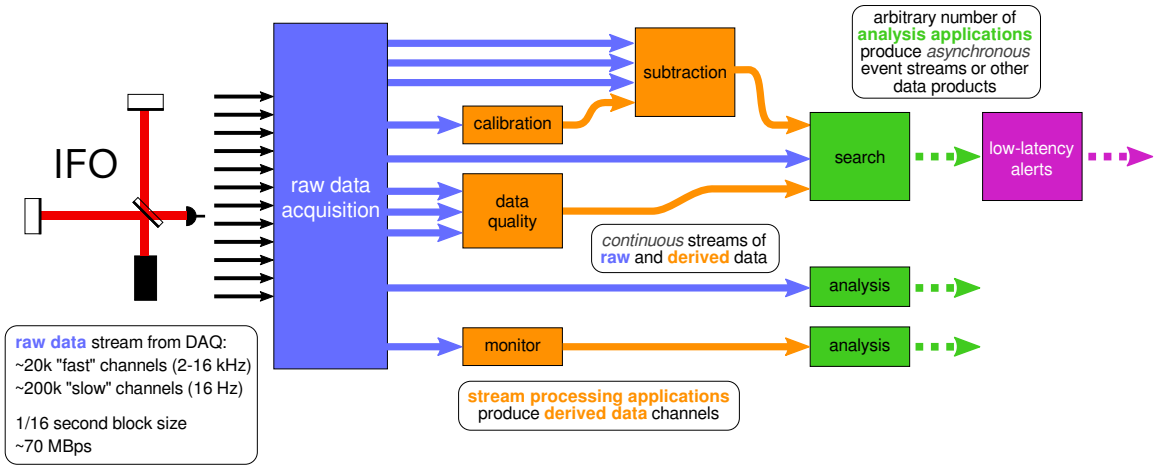


raw data stream from DAQ:
~20k "fast" channels (2-16 kHz)
~200k "slow" channels (16 Hz)

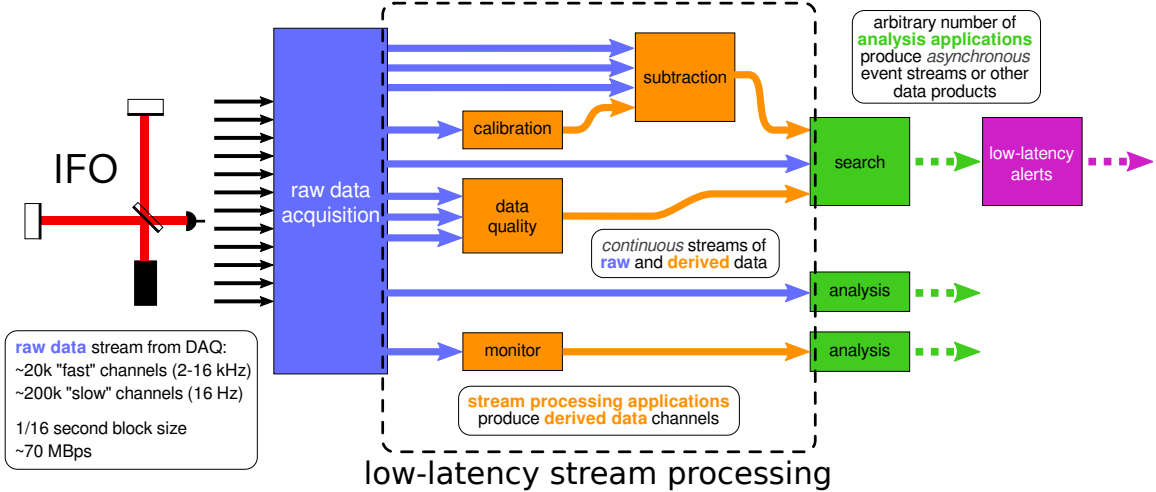
1/16 second block size
~70 MBps



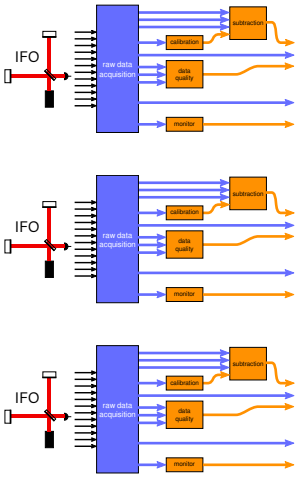
Overview of low-latency data flow



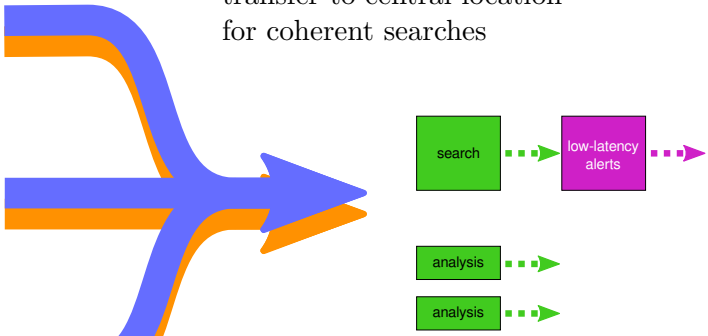
Overview of low-latency data flow



Overview of low-latency data flow



transfer to central location
for coherent searches



Existing stream processing and data distribution

stream processing: Data Monitoring Tool (DMT) receives data from the DAQ in one-second long frames. Data passed between elements via shared memory. Applications must run in special environment, on a single machine.

low-latency data distribution: Derived data frames copied out of shared memory and published in bulk into Kafka for distribution to cluster nodes. Node Kafka consumers copy frame files back into shared memory for processing by searches.

bulk offline access: Frame files on shared file systems.

general access: Network Data Service (NDS) provides network access (via custom protocol) to all offline data, and some online channels, via a simple interface. (Much work has gone into improving this system in recent years.)

Time for an update

These existing systems have evolved only incrementally over 20+ years, accumulating much technical debt along the way.

The data distribution and stream processing domains have also obviously evolved a *lot* in recent years, so we should now be able to leverage more modern existing projects.

We fortunately have a mandate from the LIGO laboratory for a completely green field re-imagining of the whole space.

Time for an update

Desired features:

- **Fast**: get data to the processes that need it as fast as possible.
- **Scalable**: need to accommodate hundreds of producers, thousands of consumers.
- **Distributed**: accessible from all relevant computing centers.
- **Flexible/Extensible**: simple plug-and-play architecture.
- **Usable**: low barrier to entry for users.

User centered design

Starting the design process by considering the interfaces that should be provided to users.

- Simple to subscribe to any raw or derived channel from anywhere.
- Simple to produce new derived channels that are available to any other application in the system.
- Archive derived channels as first class data products alongside with raw channels.
- Simple playback of both raw and derived channels from the archive.
- Simple to create and manage stream processing elements.

Prototype interface for users

A simple interface to read data:

```
for data_block in iter_data(channels):
    my_algorithm(data_block)
```

The existing NDS system provides an interface essentially identical to this.

This same interface should work for both online and offline (archived) data (with the addition of GPS start/stop times).

Prototype interface for users

A simple interface to produce derived channels into the system:

```
producer = register_producer(my_channel)

while True:
    new_data_block = my_data_generator()
    producer.publish(new_data_block)
```

Prototype interface for users

Putting them together for a stream processor:

```
producer = register_producer(my_channel)

for data_block in iter_data(channels):
    new_data_block = my_data_transform(data_block)
    producer.publish(new_data_block)
```

Straightforward development→production life cycle

It should easy to develop and test new stream processing applications without a lot of overhead or coordination with administrators or gatekeepers.

Once applications are mature/reviewed/etc., it should be straightforward to "promote" them into production, so that:

- their output derived channels are acquired into the production data management system and made available to other production stream processing applications and via the archive,
- the application processes themselves are supervised and monitored by a production process management system.

Current status

Evaluating existing technologies, and prototyping some concepts.

Message passing frameworks:

- Kafka
- MQTT
- Redis

Time series and OLAP databases:

- Materialize
- ClickHouse
- InfluxDB
- TimescaleDB

Most stream processing frameworks seem overly integrated for our needs (all processing elements in a single application).

Current status

A new high-speed, high-throughput connection between the DAQs and the site clusters, and recent refactorizations/improvements of the DAQ software, provide a basis for the new system.

All channels from the DAQ are now being delivered to the site LDAS cluster with a 1/16 second stride.

First real application: The LHO NDS2 server is currently serving fast online streams (16 Hz stride) of every channel acquired from H1. Working on getting this setup at LLO as well.

Major challenge: channel synchronization

Raw and derived channels will be produced with very different latencies. But clients want the channel data *GPS synchronized*. How do we synchronize multiple data streams produced at different latencies?

Consumer application developers should to not have to worry about buffering.

This issue has a large effect on the underlying architecture of the system.

Major challenge: channel synchronization

Client-side buffering

Client interface receives all requested channels as they come, buffering appropriately until synchronized data blocks can be provided to the application.

Flexible, minimal server requirements, but puts a lot of burden on the client.

Server-side buffering

All data is inserted into some sort of database, clients interact with a service that provides synchronized data buffers (existing NDS model).

A service for providing archived data (synchronized) will need to exist anyway...

Development roadmap

Kafka with message index based on GPS of message data?

Broker synchronization?

Development roadmap

Want new system in production for **O5** (2025)

Requires production readiness at the end of O4 (early 2024).

Hope to have something ready for testing at the end of this year (maybe some usage during O4??).