Online Data Analysis at European XFEL

Hans Fangohr
Control and Analysis Software Group
Senior Data Analysis Scientist

DESY, 25 January 2018
Outline

- Introduction & European XFEL status
- Overview online analysis
- “Karabo Bridge” (0MQ)
- Summary
Introduction

Hans Fangohr

- Diplom in Physics (1999), Hamburg
- PhD in High Performance Computing & Computer simulation (2002) in Computer Science, Southampton (UK)
- Lecturer (2002), Senior Lecturer (2006) in Computational Methods, Engineering, Southampton
- Professor of Computational Modelling (2010), Southampton
  - Head of Computational Modelling Group (2010-2017)
- Since September 2017 Data Analysis Scientist at XFEL

Research interests:
- Data Analysis & Computational Modelling
- Software for Science
- Use of software for science
European XFEL

- Official opening 1 September 2017
- 2 of 6 scientific instruments live
- First experiments started 14 Sept 2017
- 12 proposals collected ~450 TB raw data
- Positive feedback

Prof. Dr. Johanna Wanka, Bundesministerin für Bildung und Forschung, visits SPB hutch
Data analysis infrastructure

- **Hardware: “Online cluster”**,
  - 8 nodes x (20 cores, 256GB RAM) dedicated to users
  - Additional nodes for control and XFEL provided calibration and processing

- **Hardware: “Offline cluster” = Maxwell cluster (DESY)**
  - 80 nodes/3200 cores (Intel Xeon E5-2698v4)
  - ~112 TFlops
  - 512GB RAM each node
  - +20 nodes with other spec
  - 7 GPU nodes available
Data management online -> offline

During measurement (run)
- Calibrated and raw data available in hutch (GUI, online)

Data migration after each run
- After each run, data manager decides on quality of the data: “good”, “unclear”, “not interesting”
- “good” and “unclear” data transferred to “Offline cluster”
- Migration triggers computation of calibrated data at online cluster

After experiment
- Raw and calibrated data available
- Analysis on ”Offline cluster” (Maxwell @ DESY)

No automatic online data reduction
Online Analysis
Online data analysis
Data Acquisition

Various data sources
- Detectors
- Cameras
- Sensors
- Actuators

Various protocols
- Message broker
- TCP
- UDP

Interesting sources are gathered in the DAQ system
Synchronized by train ID
Stored to file (HDF5)
Streamed over TCP

Control network

On the fly control feedback
Data Acquisition (DAQ)

- Various data sources
  - Detectors
  - Cameras
  - Sensors
  - Actuators
  - Computing
  - ...

- Interesting sources are gathered in the DAQ system
  - Synchronized by train ID
  - Stored to file (HDF5)
  - Streamed over TCP
Karabo Data Pipeline

Various data sources
- Detectors
- Cameras
- Sensors
- Actuators
- Computing

Various protocols
- Message broker
- TCP
- UDP

Interesting sources are gathered in the DAQ system
Synchronized by train ID
(\texttt{daq schema})

Stored to file (HDF5)
(\texttt{storage schema})

Streamed over TCP
(\texttt{pipeline})
Karabo processing pipeline example

Karabo [1] is framework for control and data.
- Data tokens pass through pipeline.
- Processing units called “devices”.
- Devices can be distributed over hardware.
- Simplified example in figure: calibration for detector modules carried out in parallel.

Karabo Data Pipeline

- Peer-to-peer model with **TCP** protocol
- Direct **data channels** between Karabo devices
  - Can dispatch data 1-to-n / n-to-1
  - Copy data to n clients
  - Policy on busy client: wait, queue, drop, exception
- Standardized format and **data container** (Karabo Hash)
- Provides data source and timestamp
GUI ‘scenes’

- Experiment
  - Detectors
  - Sensors

- Data Management

- Pipeline processing

- Live scenes

- Karabo bridge
  - Data stream input adapter
  - Data stream output adapter

- Control network

- Raw data files (HDF5) (XFEL)
- Metadata (XFEL)
- Data files (XFEL)

- Data analysis tool A
- Data analysis tool B

- Data files

Various data sources
- Detectors
- Cameras
- Sensors
- Actuators

Various protocols
- Message broker
- TCP
- UDP

Interesting sources are gathered in the DAQ system
- Synchronized by train ID (\texttt{daq schema})

Stored to file (HDF5) (\texttt{storage schema})

Streamed over TCP (\texttt{pipeline})
Online data analysis: Rapid feedback through GUI
Karabo Bridge

Interesting sources are gathered in the DAQ system synchronized by train ID. Stored to file (HDF5). Streamed over TCP.
Karabo Bridge collaboration

DATA ANALYSIS SUPPORT IN KARABO AT EUROPEAN XFEL


European XFEL GmbH, Holzkoppel 4, 22869 Schenefeld, Germany

S. Aplin, A. Barty, M. Kuhn, V. Mariani
Centre for Free Electron Laser Science, DESY, Notkestrasse 85, 22607 Hamburg, Germany

T. Kluyver
University of Southampton, SO17 1BJ Southampton, United Kingdom

Abstract

We describe the data analysis structure that is integrated into the Karabo framework to support scientific experiments and data analysis at European XFEL. The photon science experiments have a range of data analysis requirements, including online (i.e. near real-time during the actual measurement) and offline analysis. The challenge is to handle very high data rates, i.e. on the order of 1 terabyte per second per detector at European XFEL [2] demand an efficient concurrent approach of performing experiments and data analysis: Data analysis must already start whilst data is still being acquired and initial analysis results must immediately be usable to feedback into and re-adjust the current experiment setup. The Karabo control system [3] has been developed to support these requirements.
Export Data Pipeline – Karabo Bridge

- We provide an interface to listen to Karabo pipelines
  - Integrate existing (complex) user provided tools
  - Quick (dirty) specific scripts to use during an experiment

- Karabo Bridge requirements
  - Loosely coupled Interface between Karabo and external programs
  - Export data in a generic container
  - Using straightforward network interface
  - Low latency

- Development in collaboration with CFEL Chapman Group” (S. Aplin, A. Barty, M. Kuhn, V. Mariani from CFEL)
Karabo Bridge Client

- Install the client

  pip install -e git+https://github.com/European-XFEL/karabo-bridge-py.git#egg=karabo-bridge-py

  Import Karabo bridge client

```python
In [1]: from karabo_bridge import KaraboBridge
```

- How to use it

  ```python
  In [2]: help(KaraboBridge)
  Help on class KaraboBridge in module karabo_bridge.KaraboBridge:

  class KaraboBridge(builtins.object)
   Karabo bridge client for Karabo pipeline data.
   This class can request data to a Karabo bridge server.
   Create the client with:
     krb_client = KaraboBridge("tcp://153.0.55.21:12345")
   then call `data = krb_client.next()` to request next available data
   container.

   Parameters
   ----------
   endpoint : str
       server socket you want to connect to (only support TCP socket).
   sock : str, optional
       socket type - supported: REQ.
   ser : str, optional
       Serialization protocol to use to decode the incoming message (default
       is msgpack) - supported: msgpack,pickle.
  ```
Karabo Bridge Client

- Connection to a server
  - At object instantiation, the client connects to the karabo bridge server.
  - `In [3]: kb = KaraboBridge('tcp://max-exfl093:45632')`

- Request data
  - `In [4]: train = kb.next()`
  - The data container is a dictionary.
  - `In [5]: type(train)`
  - `Out[5]: dict`
  - It contains all data sources in this data pipeline for an XRAY train
  - `In [6]: train.keys()`
  - `Out[6]: dict_keys(['detector', 'DETLAB_LAB_LPD-1/FPGA/FEM_Q2M0', 'DETLAB_LAB_LPD-1/FPGA/FEM_Q1M0', 'DETLAB_LAB_LPD-1/FPGA/FEM_Q0M0', 'DETLAB_LAB_LPD-1/FPGA/FEM_Q3M0'])`
Karabo Bridge Client

- Each data source is a dictionary
  - It contains device parameters
  - And source metadata
- All data are python built-in types
- Big array are Numpy array
- Requesting data will return the latest available train in the pipeline
Karabo Bridge Client

- You can instantiate many clients
- Data can be dispatched among them

You can create as many clients as you need (data will be distributed over the different clients).

```
In [52]: client_2 = KaraboBridge('tcp://max-exfl093:45632')
       data = client_2.next()
       print(data['detector']['trainId'])
1516380752
```

```
In [53]: client_3 = KaraboBridge('tcp://max-exfl093:45632')
       data = client_3.next()
       print(data['detector']['trainId'])
1516380753
```

- Or copy to all
  - PUB-SUB sockets
Karabo Bridge Client – Try this at home!

- Karabo Bridge server simulation
- Does not require Karabo
- Helps testing integration of the client to your tool

```python
# server.py
from karabo_bridge import server_sim

# start a simulated karabo bridge server
# and bind a socket on port 4545 of this machine (localhost).
server_sim(4545)

# client.py
from karabo_bridge import KaraboBridge

# connect the client to localhost if running on the same machine as the server.
client = KaraboBridge('tcp://localhost:4545')

while True:
    data = client.next()
    det_data = data['SPB_DET_AGIPD1M-1/DET/detector']
    print("Client : received train ID", str(det_data['header.trainId']))
    print("Client : - detector image shape is {}, {} Mbytes".format(
        det_data['image.data'].shape, det_data['image.data'].nbytes/1024**2))
```
Karabo Bridge – technical details
Networking library

- ZeroMQ
  - Intelligent socket library for messaging
  - Many kind of connection patterns
  - Multiplatform, multi-language (30+)
  - Open source LGPL
  - Large user community (including Jupyter)

- Message blobs of 0 to N bytes
- One socket to many socket connection
- Queuing at sender and receiver
- Automatic TCP (re)connect
- Easy to use

**ØMQ Hello World**

```java
import org.zeromq.ZMQ;
public class hwclient {
   public static void main (String[] args){
      ZMQ.Context context = ZMQ.context (1);
      ZMQ.Socket socket = context.socket (ZMQ.REQ);
      socket.connect ("tcp://localhost:5555");
      socket.send ("Hello", 0);
      System.out.println (socket.recv (0));
   }
}
```

```java
import org.zeromq.ZMQ;
public class hwserver {
   public static void main (String[] args) {
      ZMQ.Context context = ZMQ.context (1);
      ZMQ.Socket socket = context.socket (ZMQ.REP);
      socket.bind ("tcp://*:5555");
      while (true) {
         byte[] request = socket.recv (0);
         socket.send ("World", 0);
      }
   }
}
```
Karabo Bridge – technical details
Message serialization

- Serialization
  - Pickle
  - boost::serialization
  - MessagePack
  - Protobuf
  - ...

- MessagePack
  - Simple and open source design: https://github.com/msgpack/msgpack/
  - JSON-like binary format
  - But faster and smaller
  - Multi-language (80+ implementation available)
  - Easy implementation if need to support new language

Source: https://wtanaka.com/node/8100
Summary Karabo Bridge

- Network interface to access scientific data during experiment in near real time
  - Keep the same data structure and names as in Karabo Hash and HDF5 files
  - Easy set-up to export any data pipeline from Karabo

- Client implementation and simulator
  - Python: https://github.com/European XFEL/karabo-bridge-py
  - C++: implementation existing

- Successful use during first experiments
  - OnDA, Hummingbird, CASS, custom

- Performance (SPB experiment, AGIPD detector)

![Diagram showing data flow and performance metrics]
Containers & Jupyter

- Jupyter Notebook
  - Executable document
  - Code, output, interpretation

- Jupyter Ecosystem
  - Docker, Binder
  - Reproducibility -> better science

- Potential to support Online DA?
Summary

- Outlined basics of online data analysis at European XFEL
  - Quasi real time analysis within Karabo
  - Online GUI elements
  - Lightweight (0MQ) interface to integrate external applications

- Very early stages

- Development of growing set of open source tools

- Contact
  - Thomas.Michelat@xfel.eu, Hans.Fangohr@xfel.eu, Sandor.Brockhauser@xfel.eu

- Acknowledgements: CFEL, XFEL groups Detectors, ITDM, SPB, FXE, CAS

- Reference
Online Analysis performance

Early User Experiment (S Hauf)

- Feeds user-provided online tools via a Karabo-Bridge device
- 3-5 Hz rate at 64 cells measured
- 2-4s latency with 64 memory cells

Latency includes:
- Data acquisition
- Data formatting on DAQ
- Data forwarding to pipelines
- Data selection at pipeline entry points:
  - Every nth train
- Combining of 16 streams from modules
- Data advertising on ZMQ