DAQBroker - A general purpose instrument monitoring framework

António Dias[1,2,*], *António* Amorim[2,], *António* Tomé[3], and *João* Almeida[1]

[1]CERN CH-1211 Geneva 23 Switzerland
[2]Faculdade de Ciências, Universidade de Lisboa, Campo Grande 016, 1749-016 Lisboa
[3]Universidade da Beira Interior Rua Marquês D'Ávila e Bolama 6201-001 Covilhã

Abstract. The current scientific environment has experimentalists and system administrators allocating large amounts of time for data access, parsing and gathering as well as instrument management. This is a growing challenge since there is an increasing number of large collaborations with significant amount of instrument resources, remote instrumentation sites and continuously improved and upgraded scientific instruments. DAQBroker is a new software framework adopted by the CLOUD experiment at CERN. This framework was designed to monitor CLOUD's network of various architectures and operating systems and collect data from any instrument while also providing simple data access to any user. Data can be stored in one or several local or remote databases running on any of the most popular relational databases (MySQL, PostgreSQL, Oracle). It also provides the necessary tools for creating and editing the meta data associated with different instruments, perform data manipulation and generate events based on instrument measurements, regardless of the user's know-how of individual instruments. This submission will present an overview of each of DAQBroker's components as well as provide preliminary performance results of the application running on high and low performance machines.

1 Introduction

Data acquisition (DAQ) systems are a constantly evolving medium, ubiquitous in any field of science [1–3]. All modern scientific instruments either possess or require some sort of DAQ system ranging from simple hardware buffers [4] to fully implemented control and monitoring systems [5]. Instruments with different measurement purposes are often used together to produce a better understanding of the underlying process being measured [6, 7]. This is becoming common as collaborations between different scientific institutes are increasing [8]. With the proliferation of Internet and Internet of Things (IoT) devices, information is now expected to be available with only a few clicks [9]. However, the growing number of instruments, multi-instrument sites and collaborations will often have data being stored in instrument files [10] and in a best-case scenario, using specific software tailored only to the instrument set available to the collaboration [11, 12].

An alternative approach to DAQ systems must exist to provide the following functionalities for any instrument user [13]:

[*]e-mail: amcbd89@gmail.com

- **Universal data storage** - identify, collect and store data from any instrument and ideally from any data source,
- **Universal data monitoring** - tools to visualize the instrument's data output even if users are not familiar with the instrument,
- **Primitives for instrument data access** - functions to integrate with existing systems for access of other instruments' data,
- **Primitives for instrument control** - allow the sending of specific commands, ideally via any of the instrument's communication channels, in order to change relevant instrument settings and operation method.

Several software packages exist that can be used for scientific instrument data acquisition [14–16]. However, they suffer from problems that make them unfit to be used as universal scientific instrument DAQ systems, be it limited scope, closed source, or not being designed with scientific instruments in mind. Thus, developing a single, open-source, device independent application for handling, collecting, storing and serving data from a universal set of instruments is not only a worthwhile endeavor, it will eventually become a necessity in a scientific world with massively increasing data, formats, hardware and protocols. This paper introduces the DAQBroker framework [17] as an attempt to tackle all the aforementioned requirements.

2 DAQBroker framework

DAQBroker is a Python-based [18] web framework that focuses on instrument monitoring and aims to provide monitoring and control methods for any instrument. The framework itself focuses on three major components: *Communication*, strategies for conveying information from an instrument to a centralized repository; *Storage*, methods for collecting, organizing and storing instrument data; *Interface*, solutions for data access and manipulation.

Despite their interconnectivity, each component presents clearly defined challenges for scientific instrument DAQ. Most of DAQBroker's code is written in Python with the exception of the supplied web interface, which uses the standard HTML, Javascript and CSS.

2.1 Communication

DAQBroker allows instrument data to be gathered from either a single machine or from a network of machines. Network communication is guaranteed via a server/client architecture that emphasizes minimizing the load on the client machine. The protocol used was built over TCP, but provides a more relaxed communication scheme to handle slow or malfunctioning machines. The communication protocol consists in communication between a server that consistently checks the experiments' meta data for new instruments/data to be collected and an agent that receives the requests and routes data back to the server, which in turn will also be responsible for its storage. This protocol is run over the high level distributed messaging library ZeroMQ [19] and is illustrated in Figure 1.

2.2 Storage

Storage of data in DAQBroker is divided into two types. User and server data is stored in a local file database and instrument data is stored in one or several user provided databases running any type major consumer SQL engine[1]. Instrument data comes in a wide variety of

[1]Currently tested using, MySQL, PostgresSQL and OracleDB

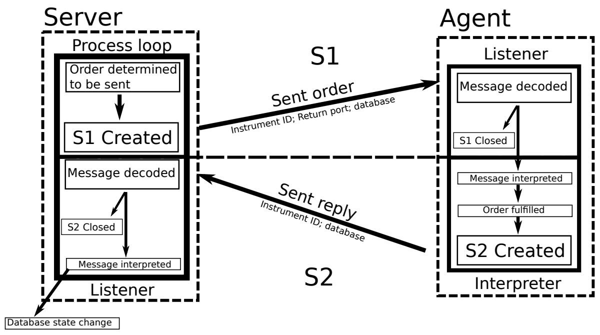

Figure 1. Illustration of the communication between a server and agent applications. The full communication requires the creation of 2 sockets (S1 & S2) between a total of 4 independent processes evenly divided between the server and agent applications.

formats, sources and granularity. However, there is one common quantity to all data gathering instruments, the time at which measurements were taken. Time allows data from different instruments to be compared, or even manipulated to create new measurements. Time is thus the best common parameter to base data storage and searches on, within a single or multiple instrument set. To that end, a top-down design concept of instrument data warehousing is introduced where the instrument is the *atomic unit*. The basic model for the instrument is divided into smaller more specific blocks:

- **Instrument block** - The most general block that encompasses all others. Includes the basic information about an instrument (name, operator info and contact details). This block allows separation between different instruments' data.
- **Data source block** - Provides information about a specific data source from an instrument. This block allows DAQBroker to decide what data gathering method to use and what information to feed to said method (ex: file data gathering requires a folder, file format and extension info).
- **Data channel block** - The most specific block of an instrument, each channel is associated to a data source block and is related to a single stream of unit information (ex: number or string). Data for each channel are associated with a specific time value or range.

It is also worth noting that a DAQBroker database is not geographically bound to a location as long as different instrument locations have access to a specific database engine (e.x: AWS RDS, institution server).

The most difficult task of this framework is to accommodate data collection from any source. This is a near impossible task with ever changing and emerging data and time formats, transfer protocols and instrument hardware to name a few of the challenges that DAQBroker faces. The open source nature of DAQBroker and the underlying Python language allows users to easily integrate their own data collection methods with the storage and visualization methods already existing in DAQBroker.

2.3 Interface

The last major component of the DAQBroker framework is one of great importance, as it provides users with the tools and visual methods of interacting with the instrument environment provided by DAQBroker. Since in a shared instrument environment, most users will not be familiar with all instruments, a simple interface is required to access the relevant data from each instrument in a universal way. In order to provide off-the-shelf functionalities to users from a wide spectrum of programming experience, a web application was designed to query and interact with DAQBroker databases. This web application consists of the following elements:

- **Front-end interface** - the main user interface contains several different tabs that provide users with the ability to query and interact with the instrument environment viewable by the server machine as well as tabs with settings and/or administrative tools. Depending on the type of user that is connected, some tabs may have their functionalities limited or even be off limits, raising an error to the user in case he decides to attempt access.
- **RESTful API** - a comprehensive API based on HTTP calls to manage and alter the instrument environment. The endpoints of the API are divided into separate types of actions to be preformed. These actions range from viewing a single instrument's information to collecting a wide range of data from several instrument data channels. Each call is required to provide proper user authentication before any changes can be attempted. The most recent version of the API can be found at http://daqbroker.com/documentation.html.

3 DAQBroker performance

As discussed previously, an application with the scope of monitoring sets of scientific instruments requires not only a comprehensive and simple to use interface, but is also required to be optimized to handle a changing and often growing set of instruments and variable hardware requirements. This section will be dedicated to testing the performance of the current version of DAQBroker by applying a set of tests designed to mimic examples of real-world instrument sets. Two machines were chosen for the following tests to represent low (LP) and high (HP) computational power machines, which are compared in Table 1:

Table 1. Computational resources of each testing machine

Resource	High-power (HP)	Low-power (LP)
CPU	8 core hyperthreading, 3.00GHz	4 core, 1.2GHz
RAM	32GB DDR4 (3200 MHz)	1GB LPDDR2 (900 MHz)
Network	Intel Gigabit LAN	10/100 Ethernet
ROM	500GB SSD	16GB microSD

When relevant, tests will be duplicated first using a local database engine and a remote AWS (**A**mazon **W**eb **S**ervices) free tier database [20] to study the effect of decentralizing the database engine. All machines are running on the same network served by a TP-LINK AC1200 Gigabit Router [21]. Randomized instrument data will be collected from a separate low power machine. The results of these tests and more can be found on an interactive application hosted on https://daqbroker.com/benchmark.

3.1 Number of instruments

For the LP machine, each *instrument* produces a single source 10 channel file and updates data every second. For the HP machine, another single source, this time with 100 channels, is considered. Each instrument data container is pre-filled with entries of artificially generated data, to emulate an instrument already containing data. Each instrument is also set up to collect new data every 10 seconds. For each set of instruments, 30 minutes of continuous monitoring is preformed, during which CPU utilization, RAM usage, disk I/O and network requests are recorded.

Figure 2 presents the collection time of instrument data as a function of the number of instruments being monitored. This figure shows that under a local database engine, there is a decrease in the average collection time of instrument data when more instruments are monitored, with that decrease being more pronounced on the LP machine. This behavior is counter-intuitive to the expected behavior of the application, although the variability of the collection time increases with the number of monitored instruments, thus it is possible that with more instruments being monitored the system is less stable in terms of these metrics, which is more in line with the expected behavior of the application.

Under a remote database engine there is no discernible trend as a function of monitored instruments for the LP machine, while in the HP machine there seems to be a gradual increase of the collection time. This behavior is more in line with the expected behavior of the application but may be connected to the underlying cloud service used and the limitations of the database engine and storage tier used.

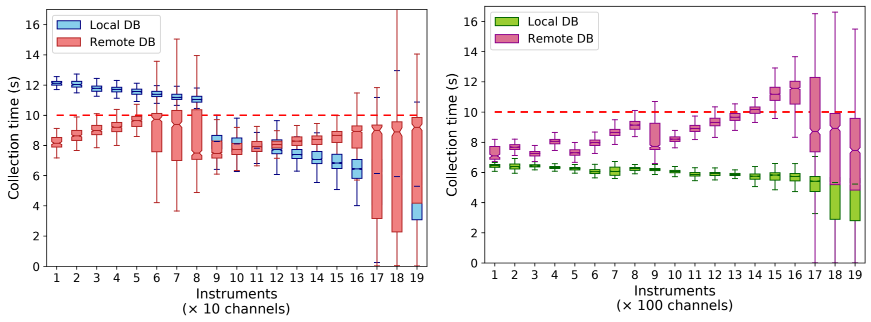

Figure 2. Instrument collection time versus number of monitored instruments in a single DAQBroker server on a LP machine (*left*) and a HP machine (*right*).

Figure 3 shows the disk usage of as a function of monitored instruments. Regardless of machine and database engine, a clear increasing trend exists for both read and writes per second (RPS and WPS, respectively), with a local database producing more variability and overall larger values of both metrics, which would be expected of using local resources over remote ones. This trend seems to show that DAQBroker is IO-bound for both machines. This is not surprising, as the bulk of DAQBroker's operations are reading and writing to files or file-like resources.

3.2 Rate of data generation

For this test, a single instrument from a remote machine with a single file source of 10 data channels is created with variable data creation periods ranging from 1 to 10^{-3} s. For each period a fixed 10 minutes of monitoring is preformed using DAQBroker. During this time a program running on the instrument's machine will simultaneously record every second the timestamp of the last record in the reference file and the timestamp of the last value stored in the DAQBroker database. While DAQBroker is able to store the instrument's data in a timely fashion, the difference between the aforementioned timestamps should always be smaller

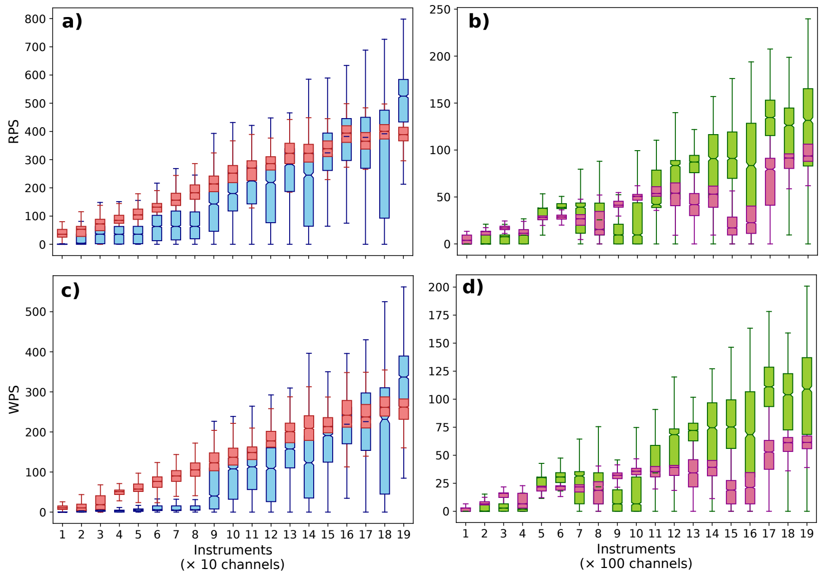

Figure 3. Disk usage versus number of monitored instruments in a single DAQBroker server on a LP machine (*a)* and *c)* and a HP machine (*b)* and *d)*. The top plots (*a)* and *b)* show writes per second and the bottom plots show reads per second (*c)* and *d)*.

than the data collection period (10 s) of that instrument.

Figure 4 shows the timestamp differences as a function of time for the different data generation rates. It can be seen that for the LP machine, a local database engine struggles to monitor instruments with creation periods lower or equal to 10^{-2} seconds (100 Hz). This limitation is mitigated by the use of a remote database, allowing data generation periods of 10^{-2} seconds to be monitored. For the HP machine, however, a local database engine struggles to keep up with periods as low as 10^{-3} seconds (1000 Hz) while a remote database engine simply cannot handle the same data generation period.

Figures 2 and 4 illustrate the need for carefully choosing the database engine location. A remote engine, while freeing local resources to handle more computation costs, trades more CPU performance for less network performance which can be at times more damaging for the monitoring of very fast data generating instruments, while at other times, freeing resources is what is needed to accommodate large amounts of slower DAQ.

4 Conclusions and future work

This contribution has introduced DAQBroker as a framework that tackles the growing need for open source universal instrument data acquisition and monitoring tools and the challenges posed by modern scientific instrument sets. The framework has been performance tested under low and high powered machines. It was shown that the framework can handle acquisition and monitoring of over 20 instruments with 100 data channels each under a high computational power machine while under a lower powered machine the framework begins to struggle at 16 instruments with 10 data channels each. Regarding data acquisition

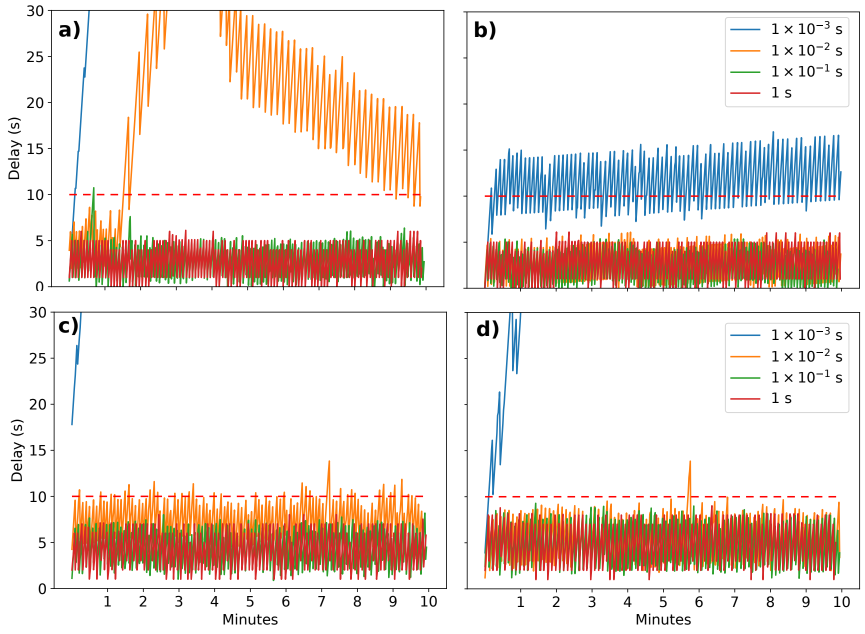

Figure 4. Current to stored timestamp differences versus time for different machines and data generation periods. The left plots (*a)* and *c)* show the LP machine performance and the right plots (*b)* and *d)* show the HP machine performance. The top plots (*a)* and *b)* show the use of a local database engine while the bottom plots (*c)* and *d)* show the use of a remote database engine.

speed the framework has the ability to handle real time data acquisition of up to 1000 Hz on the high powered machine, while on the lower powered machine it is limited to 100 Hz. The use of a remote database engine can be used to increase the number of instruments monitored, as it was shown that the framework itself is mostly IO bound. The same use of the remote database engine, however, can be detrimental when fast real time acquisition is required since the high powered machine cannot handle 1000 Hz using a remote database. DAQBroker has been implemented in the CLOUD experiment at CERN and was used to collect data for it's 2017 experimental efforts.

Several features are planned to be introduced to DAQBroker in the future. One such feature would increase the flexibility of the virtual instrument with the creation of a "skeleton instrument file" . This file would contain all the information relevant to creating an identical instrument container in a server running DAQBroker, allowing instruments to be easily moved between sites running DAQBroker. Another set of features consist in the introduction of several statistical methods that could be applied to stored time series data in order to provide users with knowledge of stationarity of the data, comparison of different time series for similarity and even information events occurring in time series.

In conclusion, DAQBroker provides a solution for data acquisition and monitoring of an ever expanding and interchangeable set of scientific instruments. It has been proven to operate in machines with comparable differences in computational power. This makes DAQBroker ideal for use in sites that are constrained in computational power or in large networks of instruments and even in sites that require large interchangeability of their instruments.

References

[1] P. Agnes, I. Albuquerque, T. Alexander, A. Alton, K. Arisaka, D.M. Asner, M. Ave, H.O. Back, B. Baldin, K. Biery et al., Journal of Instrumentation **12**, P12011 (2017)

[2] M. Ergeneci, K. Gokcesu, E. Ertan, P. Kosmas, IEEE transactions on biomedical circuits and systems **12**, 68 (2018)

[3] B. Li, J. Li, X. Lan, Y. An, W. Gao, Y. Jiang, International journal of medical informatics **112**, 114 (2018)

[4] D. Svirida, D. Collaboration et al., Physics of Particles and Nuclei **49**, 84 (2018)

[5] R. Abbasi, M. Ackermann, J. Adams, M. Ahlers, J. Ahrens, K. Andeen, J. Auffenberg, X. Bai, M. Baker, S. Barwick et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **601**, 294 (2009)

[6] J.P. Grotzinger, J. Crisp, A.R. Vasavada, R.C. Anderson, C.J. Baker, R. Barry, D.F. Blake, P. Conrad, K.S. Edgett, B. Ferdowski et al., Space science reviews **170**, 5 (2012)

[7] X. Liu, L.G. Huey, R.J. Yokelson, V. Selimovic, I.J. Simpson, M. Müller, J.L. Jimenez, P. Campuzano-Jost, A.J. Beyersdorf, D.R. Blake et al., Journal of Geophysical Research: Atmospheres **122**, 6108 (2017)

[8] C.S. Wagner, T.A. Whetsell, L. Leydesdorff, Scientometrics **110**, 1633 (2017)

[9] J. Oliveira e Sá, J.C. Sá, J.L. Pereira, F. Pimenta, M. Monteiro (2017)

[10] T. Hey, S. Tansley, K.M. Tolle et al., *The fourth paradigm: data-intensive scientific discovery*, Vol. 1 (Microsoft research Redmond, WA, 2009)

[11] C. Currey, A. Bartle, C. Lukashin, C. Roithmayr, J. Gallagher, Remote Sensing **8**, 902 (2016)

[12] D. Tescaro, A. López-Oramas, A. Moralejo, D. Mazin et al., arXiv preprint arXiv:1310.1565 (2013)

[13] Z. Hons, arXiv preprint arXiv:1508.01379 (2015)

[14] N. Instruments, *What is labview?*, accessed: Mar 2018, `http://www.ni.com/en-us/shop/labview.html`

[15] G. Labs, *Grafana - the open platform for analytics and monitoring*, accessed: Mar 2018, `https://grafana.com/`

[16] OPeNDAP, *Opendap - advanced software for remote data retrieval*, accessed: Mar 2018, `https://www.opendap.org/`

[17] A. Dias, *Daqbroker*, accessed: Mar 2019, `https://www.daqbroker.com/`

[18] P.S. Foundataion, *Python*, accessed: Mar 2019, `https://www.python.org/`

[19] P. Hintjens, *ZeroMQ: messaging for many applications* (" O'Reilly Media, Inc.", 2013)

[20] Amazon, *Amazon rds free tier - amazon web services (aws)*, accessed: Mar 2018, `https://aws.amazon.com/rds/free/`

[21] TP-Link, *Archer c5 | ac1200 wireless dual band gigabit router*, accessed: Mar 2018, `https://www.tp-link.com/us/products/details/cat-9_Archer-C5.html`