PhilDB - The time series database with built-in change logging

Andrew MacDonald

PhilDB is an open-source time series database. It supports storage of time series datasets that are dynamic, that is recording updates to existing values in a log as they occur. Recent open-source systems, such as InfluxDB and OpenTSDB, have been developed to indefinitely store long-period, high-resolution time series data. Unfortunately they require a large initial installation investment before use because they are designed to operate over a cluster of servers to achieve high-performance writing of static data in real time. In essence, they have a 'big data' approach to storage and access. Other open-source projects for handling time series data that don't take the 'big data' approach are also relatively new and are complex or incomplete. None of these systems gracefully handle revision of existing data while tracking values that changed. Unlike 'big data' solutions, PhilDB has been designed for single machine deployment on commodity hardware, reducing the barrier to deployment. PhilDB eases loading of data for the user by utilising an intelligent data write method. It preserves existing values during updates and abstracts the update complexity required to achieve logging of data value changes. PhilDB improves accessing datasets by two methods. Firstly, it uses fast reads which make it practical to select data for analysis. Secondly, it uses simple read methods to minimise effort required to extract data. PhilDB takes a unique approach to meta-data tracking; optional attribute attachment. This facilitates scaling the complexities of storing a wide variety of data. That is, it allows time series data to be loaded as time series instances with minimal initial meta-data, yet additional attributes can be created and attached to differentiate the time series instances as a wider variety of data is needed. PhilDB was written in Python, leveraging existing libraries. This paper describes the general approach, architecture, and philosophy of the PhilDB software.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1488v1 | CC-BY 4.0 Open Access | rec: 9 Nov 2015, publ: 9 Nov 2015

PhilDB - The time series database with built-in change logging

Andrew MacDonald[1]

[1]**Melbourne, Australia**

ABSTRACT

PhilDB is an open-source time series database. It supports storage of time series datasets that are dynamic, that is recording updates to existing values in a log as they occur.
Recent open-source systems, such as InfluxDB and OpenTSDB, have been developed to indefinitely store long-period, high-resolution time series data. Unfortunately they require a large initial installation investment before use because they are designed to operate over a cluster of servers to achieve high-performance writing of static data in real time. In essence, they have a 'big data' approach to storage and access. Other open-source projects for handling time series data that don't take the 'big data' approach are also relatively new and are complex or incomplete. None of these systems gracefully handle revision of existing data while tracking values that changed. Unlike 'big data' solutions, PhilDB has been designed for single machine deployment on commodity hardware, reducing the barrier to deployment. PhilDB eases loading of data for the user by utilising an intelligent data write method. It preserves existing values during updates and abstracts the update complexity required to achieve logging of data value changes. PhilDB improves accessing datasets by two methods. Firstly, it uses fast reads which make it practical to select data for analysis. Secondly, it uses simple read methods to minimise effort required to extract data.
PhilDB takes a unique approach to meta-data tracking; optional attribute attachment. This facilitates scaling the complexities of storing a wide variety of data. That is, it allows time series data to be loaded as time series instances with minimal initial meta-data, yet additional attributes can be created and attached to differentiate the time series instances as a wider variety of data is needed.
PhilDB was written in Python, leveraging existing libraries.
This paper describes the general approach, architecture, and philosophy of the PhilDB software.

Keywords: time series, database, logging, python, data science

1 INTRODUCTION

This paper will explore existing time series database solutions. It will examine the need for a liberally licensed, open-source, easily deployed time series database, that is capable of tracking data changes, and look at why the existing systems that were surveyed failed to meet these requirements. Next, this paper will describe the architecture and features of the new system, PhilDB, that was designed to meet these outlined needs. Finally, a simple evaluation will be performed to compare PhilDB to the most promising alternative of the existing open-source systems.

2 BACKGROUND: EXISTING SYSTEMS

2.1 Proprietary systems

There are a number of proprietary solutions for storage of time series data that have been around since the mid-nineties to the early 2000s. Castillejos (2006) identified three proprietary systems of note, FAME, TimeIQ, and DBank, which have references that range from 1995 to 2000. There are other proprietary systems, such as kdb+[1], that are commercially available today. This shows that time series data storage is an existing problem. Ready access to open-source systems make them easier to evaluate and integrate with compared to proprietary systems and they are more fitting with the scientific Python ecosystem as described by Perez et al. (2011). Discussion on the need for an open-source system is further covered in Section 3. Therefore existing proprietary systems were not evaluated any further.

2.2 Open-source systems

In recent years the development of open-source time series databases has taken off, with most development beginning within the last five years. This can be seen by the number of projects discussed here along with noting the initial commit dates.

2.2.1 'Big data' time series databases

Some of the most successful projects in the open-source time series database space are OpenTSDB[2], Druid[3], Kairosdb[4], and InfluxDB[5]. The earliest start to development on these systems was for OpenTSDB with an initial commit in April 2010. These systems are designed to operate over a cluster of servers to achieve high-performance writing of static data in real time. In essence, they have a 'big data' approach to storage and access. The architectural approach to address big data requirements means a large initial installation investment before use.

2.2.2 Alternate time series databases

In contrast to the 'big data' time series systems, some small dedicated open-source code bases are attempting to address the need for local or single server time series data storage. These systems, however, have stalled in development, are poorly documented,

[1] http://kx.com/software.php

[2] OpenTSDB initial commit: 2010-04-11; https://github.com/OpenTSDB/opentsdb

[3] Druid initial commit: 2012-10-24; https://github.com/druid-io/druid/

[4] Kairosdb initial commit: 2013-02-06; https://github.com/kairosdb/kairosdb

[5] InfluxDB initial commit: 2013-04-12; https://github.com/influxdb/influxdb

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1488v1 | CC-BY 4.0 Open Access | rec: 9 Nov 2015, publ: 9 Nov 2015

or require a moderate investment of time to operate. For example Timestore[6] was, at the time of writing, last modified August 2013 with a total development history of 36 commits. Some of the better progressed projects still only have minimal development before progress has ceased, for example tsdb[7] with a development start in January 2013 and the most recent commit at time of writing in February 2013 for a total of 58 commits. Cube[8] has a reasonable feature set and has had more development effort invested than the other systems discussed here, with a total of 169 commits, but it is no longer under active development according the Readme file. Searching GitHub for 'tsdb' reveals a large number of projects named 'tsdb' or similar. The most popular of these projects (when ranked by stars or number of forks) relate to the 'big data' systems described earlier (in particular, OpenTSDB, InfluxDB, and KairosDB). There are numerous small attempts at solving time series storage in simpler systems that fall short of a complete solutions. Of the systems discussed here only Cube had reasonable documentation, Timestore had usable documentation, and tsdb had no clear documentation.

2.2.3 Scientific time series databases

At present, the only open-source solution that addresses the scientific need to track changes to stored time series data as a central principle is SciDB (Stonebraker et al. 2009 and Stonebraker et al. 2011). SciDB comes with comprehensive documentation[9] which is required for such a complex system. Access to source code is via tarballs (there is no source control system with general access to investigate the history of the project in detail).

3 WHY ANOTHER TIME SERIES DATABASE?

An interest in the smaller time series database systems is likely derived from a need to handle data for exploratory purposes with the intention to later integrate with other systems, with minimal initial deployment overhead, as was needed by the author. Open-source 'big data' time series database offerings don't support the ability to track any changed values out of the box (such support would have to be developed external to the system). Their design targets maximum efficiency of write-once and read-many operations. "Most scientists are adamant about not discarding any data" (Cudré-Mauroux et al. 2009) is a statement the author has found to be true. Therefore, both requirements of minimal deployment overhead and logging of any changed values rule out the current 'big data' systems.

While SciDB does address the data tracking need, it is complex to install and use, thus falling into the same installation category as the 'big data' systems in terms of set up. Installation difficulty isn't enough to rule out the system being a suitable solution, but it does diminish its value as an exploratory tool. SciDB is also licensed under the GNU Affero General Public License (AGPL) which can be perceived as a problem in corporate or government development environments. In these environments integration

[6] Timestore http://www.mike-stirling.com/redmine/projects/timestore; https://github.com/mikestir/timestore initial commit 2012-12-27

[7] tsdb initial commit: 2013-01-11; most recent commit at time of writing: 2013-02-17; https://github.com/gar1t/tsdb

[8] Cube initial commit: 2011-09-13; https://github.com/square/cube

[9] http://www.paradigm4.com/HTMLmanual/15.7/scidb_ug/

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1488v1 | CC-BY 4.0 Open Access | rec: 9 Nov 2015, publ: 9 Nov 2015

with more liberally licensed (e.g. Apache License 2.0 or 3-clause BSD) libraries is generally preferred with many online discussions around the choice of liberal licences for software in the scientific computing space. For example, it can be argued that a simple liberal license like the BSD license encourages the most participation and reuse of code (Brown 2015, VanderPlas 2014, Hunter 2004). Finally, SciDB has a broader scope than just storage and retrieval of time series data, since "SciDB supports both a functional and a SQL-like query language" (Stonebraker et al. 2011). These query languages add additional cognitive load for any developer interfacing with the system as the query languages are specific to SciDB.

Of the other existing systems discussed here, none support logging of changed values. Limited documentation makes them difficult to evaluate, but from what can be seen and inferred from available information, the designs are targeting the 'write once read many' style of the 'big data' time series systems at a smaller deployment scale. These systems were extremely early in development or yet to be started at time the author began work on PhilDB in October 2013.

PhilDB has been created to provide a time series database system that is easily deployed and has logging features to track any new or changed values.

4 ARCHITECTURE

PhilDB uses a central 'meta-data store' to track the meta information about time series instances. Each time series instance is assigned a UUID (Leach et al. 2005) upon creation. Time series instances are associated with an identifier and attributes using the meta-data store. The identifier and attributes can then be used to distinguish between time series instances. Using the meta-data store a UUID can be looked up for a given combination of identifier and attributes. The UUID then maps to a file on disk containing the time series data or a file containing the related log of changes to the time series.

4.1 Architecture Philosophy

The reasoning behind this architectural design is so that a time series instance can be stored with minimal initial effort. Attaching a time series identifier as the initial minimal information allows for data from a basic dataset to be loaded and explored immediately. Additional attributes can then be attached to a time series instance to further differentiate datasets that share conceptual time series identifiers. By default these identifier and attribute combinations are then stored in a tightly linked relational database. This meta data store could optionally be replaced by alternative technology such as flat files. As the data is stored in individual structured files, the meta-data store acts as a minimal index with most of the work being delegated to the operating system.

5 IMPLEMENTATION

PhilDB is written in Python because it fits well with the scientific computing ecosystem (Perez et al. 2011). The core of the PhilDB package is the PhilDB database class[10], which exposes high level methods for data operations. These high level functions are designed to be easily used interactively in the IPython interpreter (Perez and Granger

[10] http://phildb.readthedocs.org/en/latest/api/phildb.html#module-phildb.database

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1488v1 | CC-BY 4.0 Open Access | rec: 9 Nov 2015, publ: 9 Nov 2015

2007) yet still work well in scripts and applications. The goal of interactivity and scriptability are to enable exploratory work and the ability to automate repeated tasks (Shin et al. 2011). Utilising Pandas (McKinney 2012) to handle complex time series operations related to different frequencies simplifies the internal code that determines if values require creation or updating. Returning Pandas objects from the read methods allows for data analysis to be performed readily without further data munging. Lower level functions are broken up into separate modules for major components such as reading, writing, and logging, which can be easily tested as individual components. The PhilDB class pulls together the low level methods, allowing for the presentation of a stable interface that abstracts away the hard work of ensuring that new or changed values, and only those values, are logged.

Installation of PhilDB is performed easily within the Python ecosystem using the standard Python setup.py process, including installation from PyPI using 'pip'.

5.1 Features

Key features of PhilDB are:

* A single method accepting a pandas.Series object, data frequency and attributes for writing or updating time series.
* A read method for reading a single time series based on requested time series identifier, frequency and attributes.
* Advanced read methods for reading collections of time series.
* Support for storing regular and irregular time series.
* Logging of any new or changed values.
* Log read method to extract a time series as it appeared on a given date.

5.2 Database Format

The technical implementation of the database format, as implemented in version 0.6.1 of PhilDB (MacDonald 2015), is described in this section. Due to the fact that PhilDB is still in the alpha stage of development the specifics here may change significantly in the future.

The meta-data store is a relational database to track attributes with the current implementation using SQLite (Hipp et al. 2015). Actual time series data are stored as flat files on disk, indexed by the meta-data store to determine the path to a given series. The flat files are implemented as plain binary files that store a 'long', 'double', and 'int' for each record. The 'long' is the datetime stored as a 'proleptic Gregorian ordinal' as determined by the Python datetime.datetime.toordinal method[11] (van Rossum 2015). The 'double' stores the actual value corresponding to the date stored in the preceding 'long'. Finally, the 'int' is a meta value for marking additional information about the record. In this version of PhilDB the meta value is only used to flag missing data values. Individual changes to time series values are logged to HDF5 files (The HDF Group

[11] https://docs.python.org/2/library/datetime.html#datetime.date.toordinal

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1488v1 | CC-BY 4.0 Open Access | rec: 9 Nov 2015, publ: 9 Nov 2015

1997) that are kept alongside the main time series data file with every new value written as a row in a table, each row having a column to store the date, value, and meta value as per the file format. In addition, a final column is included to record the date and time the record was written.

6 EVALUATION

Of the open-source systems evaluated (as identified in section 2.2), InfluxDB came the closest in terms of minimal initial installation requirements and feature completeness, however, it doesn't support the key feature of update logging. Paul Dix (CEO of InfluxDB) found that performance and ease of installation were the main concerns of users of existing open-source time series database systems (Dix 2014). InfluxDB was built to alleviate both those concerns. An illustrative comparison between InfluxDB and PhilDB was performed by loading a sample dataset — consisting of daily data for 221 time series — into both systems, noting configuration requirements and performance along the way. Reading the data back out from both systems was also measured for performance. The sample dataset of 221 time series had a mean length of 16310 days, with the breakdown of the series lengths in Table 1.

mean	16310 days
std	2945 days
min	10196 days
25%	14120 days
50%	15604 days
75%	18256 days
max	22631 days

Table 1. Breakdown of length of time series in sample dataset (all values rounded to nearest day)

While InfluxDB is designed for high performance data collection, it is poorly designed for bulk loading of data. Two complications arose while trying to load the sample dataset. First, the number of files created by InfluxDB while writing data points resulted in overloading the Ext4 file system of the test machine, with journal writes causing performance degradation, to the point that InfluxDB failed to respond to the client loading the data with an HTTP 500 error. This was worked around by reducing the amount of data loaded by for each time series in the dataset to 5 years (1825 days) along with specifying a batch_size of 100 to the write_point method of the InfluxDB Python client. While file system tuning or using alternate file systems could solve this, this was not attempted because the idea behind PhilDB is that it should be easily used with default system configuration. Second, the failure to write any data with a date prior to 1970-01-01. This may be a bug in the InfluxDB Python client rather than a limitation of InfluxDB itself, as there is no documentation that specifies dates must be from 1970 onwards. To work around this problem the data being loaded was restricted to data after 1970-01-01.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1488v1 | CC-BY 4.0 Open Access | rec: 9 Nov 2015, publ: 9 Nov 2015

6.1 Installation

InfluxDB is easily installed compared to the other open-source systems evaluated as demonstrated by the short install process shown below. Installation of pre-built packages on Linux requires root access[12] . Installation of InfluxDB was performed on a 64-bit Fedora 19 Linux desktop machine using the pre-built RPM of InfluxDB version 0.9.3 as follows:

```
wget http://influxdb.s3.amazonaws.com/influxdb-0.9.3-1.x86_64.rpm
sudo yum localinstall influxdb-0.9.3-1.x86_64.rpm
```

Starting the InfluxDB service with:

```
sudo /etc/init.d/influxdb start
```

Installation of PhilDB is readily performed using pip:

```
pip install phildb
```

Using a Python virtualenv removes the need to have root privileges to install PhilDB.

6.2 Performance

Due to the limitations of loading data into InfluxDB the dataset was restricted to 5 years worth of each time series from 1970 (which will be referred to as the 'partial dataset'). Performance tests were also done for PhilDB with the entire time series dataset loaded (the 'complete dataset'). It should be noted that PhilDB supports local write, which is advantageous for performance, compared to InfluxDB which only supports network access. InfluxDB was hosted locally, which prevents network lag, but the protocol design still reduced performance compared to the direct write as done by PhilDB. Write performance was measured by writing each of the 221 time series into the database under test, recording the time spent per time series and calculating the average (Figure 1 and Table 2). InfluxDB write performance was two orders of magnitude slower than PhilDB with the equivalent dataset and four times slower compared to PhilDB with the complete dataset, as can be seen in Figure 1 and Table 2. This shows PhilDB has a significant performance advantage over InfluxDB for bulk loading of time series data.

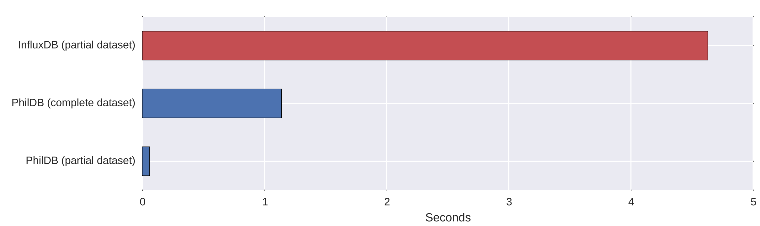

Figure 1. Mean write time

PhilDB direct access is capable of reading a (partial) time series two orders of magnitude faster than InfluxDB as seen in Figure 2. A fairer comparison of read access is using the experimental PhilDB server and client, which provides the same API as PhilDB with a JSON over HTTP data transfer. Read performance was measured using

[12] https://influxdb.com/docs/v0.9/introduction/installation.html

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1488v1 | CC-BY 4.0 Open Access | rec: 9 Nov 2015, publ: 9 Nov 2015

the Python timeit module to perform 10 iterations of a read using InfluxDB, PhilDB (partial and complete dataset), and the PhilDB server/client combo (partial and complete dataset) for a sample of ten time series. The mean of the ten runs yielded the results given in Figure 2 and Table 2.

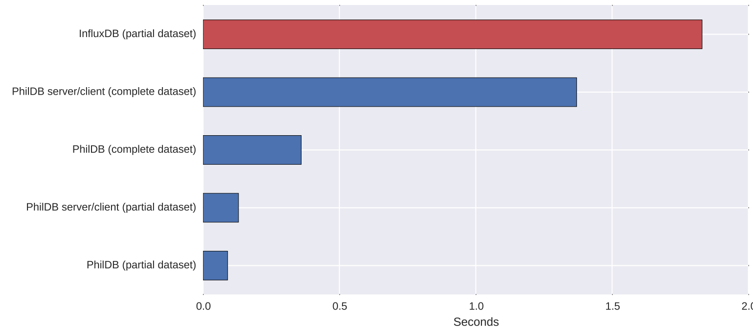

Figure 2. Mean read time

PhilDB out-performs InfluxDB in read speed. Even the server/client model, which has yet to be optimised for performance, out-performed InfluxDB even in the case where InfluxDB contained a smaller dataset than PhilDB did.

	Write	Read
InfluxDB (partial dataset)	4.63 seconds	1.83 seconds
PhilDB (partial dataset)	0.06 seconds	0.09 seconds
PhilDB (complete dataset)	1.14 seconds	0.13 seconds
PhilDB server/client (partial dataset)	N/A	0.36 seconds
PhilDB server/client (complete dataset)	N/A	1.37 seconds

Table 2. Read and write performance results

Finally, the resulting PhilDB database containing the partial test dataset is 13 megabytes while InfluxDB required 69 megabytes. It is worth noting at this point that PhilDB does have full duplication of the initial data due to the current implementation of the logging mechanism. Therefore PhilDB takes significantly less space than InfluxDB to store an equivalent (small) quantity of data.

6.3 Summary

While InfluxDB was the most promising of the evaluated open-source systems, it fell short in terms of performance, ease of use, and ease of deployment. InfluxDB appears well suited for the data and usage patterns it was designed for, but ill suited for the use case PhilDB is aimed at. The lack of support for change logging means that such a feature would need to be developed alongside the deployment. It also suggests that any attempt to integrate such a feature would result in further performance losses. PhilDB's design has yielded good performance in terms of speed and space requirements, while at the same time being simple to install and use. This makes it a suitable tool for handling time series data in the scientific context.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1488v1 | CC-BY 4.0 Open Access | rec: 9 Nov 2015, publ: 9 Nov 2015

7 FUTURE WORK

PhilDB is still in its alpha stage. Before reaching the beta stage, the author shall investigate:

* Complete attribute management to support true arbitrary attribute creation and attachment.
* Possible alternative back ends, using alternative data formats, disk paths, and relational databases.
* More sophisticated handling of time zone meta-data.
* Storage of quality codes or other row level attributes.
* Formalisation of UUID usage for sharing of data.

8 CONCLUSION

In conclusion, PhilDB provides for an accessible time series database that can be deployed quickly so that curious minds, such as those in our scientific community, can easily analyse time series data and elucidate world-changing information. For scientific computing, it is important that any solution is capable of tracking subsequent data changes. PhilDB addresses this gap in existing solutions, as well as surpassing them for efficiency and usability. Finally, PhilDB's source code has been released on GitHub under the 3-clause BSD open-source license to help others easily extract wisdom from their data.

9 ACKNOWLEDGEMENTS

I would like to thank Di MacDonald for her editorial advice on early drafts, my fiancée Katrina Cornelly for her support and editorial advice, and my colleague Richard Laugesen for his valuable review comments on a later draft.

REFERENCES

Brown, C. T. (2015). On licensing bioinformatics software: use the BSD, Luke. http://ivory.idyll.org/blog/2015-on-licensing-in-bioinformatics.html.

Castillejos, A. M. (2006). *Management of Time Series Data*. PhD thesis, University of Canberra.

Cudré-Mauroux, P., Kimura, H., Lim, K.-T., Rogers, J., Simakov, R., Soroush, E., Velikhov, P., Wang, D. L., Balazinska, M., Becla, J., and others (2009). A demonstration of SciDB: a science-oriented DBMS. *Proceedings of the VLDB Endowment*, 2(2):1534–1537.

Dix, P. (2014). InfluxDB: One year of InfluxDB and the road to 1.0. https://influxdb.com/blog/2014/09/26/one-year-of-influxdb-and-the-road-to-1_0.html.

Hipp, D. R., Kennedy, D., and Mistachkin, J. (2015). SQLite. https://www.sqlite.org/download.html.

Hunter, J. D. (2004). Why we should be using BSD. http://nipy.sourceforge.net/nipy/stable/faq/johns_bsd_pitch.html.

Leach, P. J., Mealling, M., and Salz, R. (2005). A Universally Unique IDentifier (UUID) URN Namespace.

MacDonald, A. (2015). phildb: Version 0.6.1. http://dx.doi.org/10.5281/zenodo.32437.

McKinney, W. (2012). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. "O'Reilly Media, Inc.".

Perez, F. and Granger, B. E. (2007). IPython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3):21–29.

Perez, F., Granger, B. E., and Hunter, J. D. (2011). Python: an ecosystem for scientific computing. *Computing in Science & Engineering*, 13(2):13–21.

Shin, D., Schepen, A., Peatey, T., Zhou, S., MacDonald, A., Chia, T., Perkins, J., and Plummer, N. (2011). WAFARi: A new modelling system for Seasonal Streamflow Forecasting service of the Bureau of Meteorology, Australia. In *MODSIM 2011, Modelling and Simulation Society of Australian and New Zealand*.

Stonebraker, M., Becla, J., DeWitt, D. J., Lim, K.-T., Maier, D., Ratzesberger, O., and Zdonik, S. B. (2009). Requirements for Science Data Bases and SciDB. In *CIDR*, volume 7, pages 173–184.

Stonebraker, M., Brown, P., Poliakov, A., and Raman, S. (2011). The Architecture of SciDB. In Cushing, J. B., French, J., and Bowers, S., editors, *Scientific and Statistical Database Management*, number 6809 in Lecture Notes in Computer Science, pages 1–16. Springer Berlin Heidelberg.

The HDF Group (1997). Hierarchical Data Format, version 5. http://www.hdfgroup.org/HDF5/.

van Rossum, G. (2015). Python Programming Language. www.python.org.

VanderPlas, J. (2014). The Whys and Hows of Licensing Scientific Code. http://www.astrobetter.com/blog/2014/03/10/the-whys-and-hows-of-licensing-scientific-code/.