

A peer-reviewed version of this preprint was published in PeerJ on 30 March 2016.

[View the peer-reviewed version](https://peerj.com/articles/cs-52) (peerj.com/articles/cs-52), which is the preferred citable publication unless you specifically need to cite this preprint.

MacDonald A. 2016. PhilDB: the time series database with built-in change logging. PeerJ Computer Science 2:e52
<https://doi.org/10.7717/peerj-cs.52>

1 PhilDB: The time series database with 2 built-in change logging

3 Andrew MacDonald¹

4 ¹Melbourne, Australia

5 ABSTRACT

PhilDB is an open-source time series database that supports storage of time series datasets that are dynamic, that is it records updates to existing values in a log as they occur. 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. It implements fast reads to make it practical to select data for analysis.

Recent open-source systems have been developed to indefinitely store long-period high-resolution time series data without change logging. Unfortunately such systems generally 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 avoid the 'big data' approach are also relatively new and are complex or incomplete. None of these
6 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 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 when a wider variety of data is needed. PhilDB was written in Python, leveraging existing libraries. While some existing systems come close to meeting the needs PhilDB addresses, none cover all the needs at once. PhilDB was written to fill this gap in existing solutions.

This paper explores existing time series database solutions, discusses the motivation for PhilDB, describes the architecture and philosophy of the PhilDB software, and performs a simple evaluation between InfluxDB, PhilDB, and SciDB.

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

8 1 INTRODUCTION

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

16 2 BACKGROUND: EXISTING SYSTEMS

17 2.1 Proprietary systems

18 There are a number of proprietary solutions for storage of time series data that have
19 been around since the mid-nineties to the early 2000s. Castillejos (2006) identified
20 three proprietary systems of note, FAME, TimeIQ, and DBank, that have references that
21 range from 1995 to 2000. There are other proprietary systems, such as kdb+¹, that are
22 commercially available today. This shows that time series data storage is an existing
23 problem. Compared to proprietary systems, open-source systems can generally be used
24 with the scientific Python ecosystem as described by Perez et al. (2011). Ready access
25 to open-source systems also make them easier to evaluate and integrate with. Therefore
26 existing proprietary systems were not evaluated any further. Discussion on the need for
27 an open-source system is further covered in section 3.

28 2.2 Open-source systems

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

32 2.2.1 'Big data' time series databases

33 Some of the most successful projects in the open-source time series database space are
34 OpenTSDB², Druid³, Kairosdb⁴, and InfluxDB⁵. The earliest start to development on
35 these systems was for OpenTSDB with an initial commit in April 2010. These systems
36 are designed to operate over a cluster of servers to achieve high-performance writing
37 of static data in real time. In essence, they have a 'big data' approach to storage and
38 access. The architectural approach to address big data requirements means a large initial
39 installation investment before use.

40 2.2.2 Alternate time series databases

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

¹<http://kx.com/software.php>

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

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

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

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

44 or require a moderate investment of time to operate. For example Timestore⁶ was, at
45 the time of writing, last modified August 2013 with a total development history of 36
46 commits. Some of the better progressed projects still only had minimal development
47 before progress had ceased, for example tsdb⁷ with a development start in January 2013
48 and the most recent commit at time of writing in February 2013 for a total of 58 commits.
49 Cube⁸ has a reasonable feature set and has had more development effort invested than
50 the other systems discussed here, with a total of 169 commits, but it is no longer under
51 active development according the Readme file. Searching GitHub for ‘tsdb’ reveals a
52 large number of projects named ‘tsdb’ or similar. The most popular of these projects
53 (when ranked by stars or number of forks) relate to the ‘big data’ systems described
54 earlier (in particular, OpenTSDB, InfluxDB, and KairosDB). There are numerous small
55 attempts at solving time series storage in simpler systems that fall short of a complete
56 solutions. Of the systems discussed here only Cube had reasonable documentation,
57 Timestore had usable documentation, and tsdb had no clear documentation.

58 **2.2.3 Scientific time series databases**

59 At present, the only open-source solution that addresses the scientific need to track
60 changes to stored time series data as a central principle is SciDB (Stonebraker et al. 2009
61 and Stonebraker et al. 2011). SciDB comes with comprehensive documentation⁹ that is
62 required for such a feature rich system. The documentation is however lacking in clarity
63 around loading data with most examples being based around the assumption that the
64 data already exists within SciDB or is being generated by SciDB. While installation on
65 a single server is relatively straight forward (for older versions with binaries supplied for
66 supported platforms) the process is hard to identify as the community edition installation
67 documentation is mixed in with the documentation on installation of the enterprise
68 edition of SciDB. Access to source code is via tarballs; there is no source control system
69 with general access to investigate the history of the project in detail.

70 **3 MOTIVATION**

71 The author’s interest is derived from a need to handle data for exploratory purposes with
72 the intention to later integrate with other systems, with minimal initial deployment over-
73 head. It is assumed that the smaller time series database systems discussed previously
74 derive from similar needs. The author has found “[m]ost scientists are adamant about not
75 discarding any data” (Cudré-Mauroux et al. 2009). In particular, the author’s experience
76 in hydrology has found hydrological data requires the ability to track changes to it, since
77 streamflow discharge can be regularly updated through quality control processes or
78 updates to the rating curves used to convert from water level to discharge. Open-source
79 ‘big data’ time series database offerings don’t support the ability to track any changed
80 values out of the box (such support would have to be developed external to the system).
81 Their design targets maximum efficiency of write-once and read-many operations. When

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

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

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

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

82 streamflow data is used within forecasting systems, changes to the data can alter the
83 forecast results. Being able to easily identify if a change in forecast results is due to
84 data or code changes greatly simplifies resolving issues during development and testing.
85 Therefore, both requirements of minimal deployment overhead and logging of any
86 changed values rule out the current ‘big data’ systems.

87 While SciDB does address the data tracking need, recent versions of the community
88 edition are complex to install since they require building from source, a process more
89 involved than the usual ‘./configure; make; make install’. Older versions are more
90 readily installed on supported platforms, however the system is still complex to use,
91 requires root access to install, a working installation of PostgreSQL and a dedicated user
92 account for running. Installation difficulty isn’t enough to rule out the system being a
93 suitable solution, but it does diminish its value as an exploratory tool. SciDB is also
94 licensed under the GNU Affero General Public License (AGPL) that can be perceived as
95 a problem in corporate or government development environments. In these environments
96 integration with more liberally licensed (e.g. Apache License 2.0 or 3-clause BSD)
97 libraries is generally preferred with many online discussions around the choice of liberal
98 licences for software in the scientific computing space. For example, it can be argued
99 that a simple liberal license like the BSD license encourages the most participation and
100 reuse of code (Brown 2015, VanderPlas 2014, Hunter 2004).

101 Finally, SciDB has a broader scope than just storage and retrieval of time series data,
102 since “SciDB supports both a functional and a SQL-like query language” (Stonebraker
103 et al. 2011). Having SQL-like query languages does allow for SciDB to readily
104 support many high performance operations directly when handling large already loaded
105 data. These query languages do, however, add additional cognitive load (Sweller et al.
106 2011) for any developer interfacing with the system as the query languages are specific
107 to SciDB. If using SciDB for performing complex operations on very large multi-
108 dimensional array datasets entirely within SciDB, learning these query languages would
109 be well worth the time. The Python API does enable a certain level of abstraction
110 between getting data out of SciDB and into the scientific Python ecosystem.

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

117 The need of the author is purely to store simple time series of floating point values
118 and extract them again for processing with other systems.

119 **3.1 Use case**

120 To summarise, PhilDB has been created to provide a time series database system that is
121 easily deployed, used, and has logging features to track any new or changed values. It
122 has a simple API for writing both new and updated data with minimal user intervention.
123 This is to allow for revising time series from external sources where the data can change
124 over time, such as streamflow discharge data from water agencies. Furthermore, the
125 simple API extends to reading, to enable easy retrieval of time series, including the
126 ability to read time series as they appeared at a point in time from the logs.

127 **4 ARCHITECTURE**

128 PhilDB uses a central ‘meta-data store’ to track the meta information about time series
129 instances. Relational databases are a robust and reliable way to hold related facts. Since
130 the meta data is simply a collection of related facts about a time series, a relational
131 database is used for the meta-data store. Time series instances are associated with a user
132 chosen identifier and attributes and each time series instance is assigned a UUID (Leach
133 et al. 2005) upon creation, all of which is stored in the meta-data store. The actual time
134 series data (and corresponding log) is stored on disk with filenames based on the UUID
135 (details of the format are discussed in section 5.2). Information kept in the meta-data
136 store can then be used to look up the UUID assigned to a given time series instance
137 based on the requested identifier and attributes. Once the UUID has been retrieved,
138 accessing the time series data is a simple matter of reading the file from disk based on
139 the expected UUID derived filename.

140 **4.1 Architecture Philosophy**

141 The reasoning behind this architectural design is so that:

- 142 * A simple to use write method can handle both new and updated data (at the same
143 time if needed).
- 144 * Read access is fast and easy for stored time series.
- 145 * Time series are easily read as they appeared at a point in time.
- 146 * Each time series instance can be stored with minimal initial effort.

147 Ease of writing data can come at the expense of efficiency to ensure that create,
148 update or append operations can be performed with confidence that any changes are
149 logged without having to make decisions on which portions of the data are current or new.
150 The expectation is that read performance has a greater impact on use as they are more
151 frequent. Attaching a time series identifier as the initial minimal information allows for
152 data from a basic dataset to be loaded and explored immediately. Additional attributes
153 can be attached to a time series instance to further differentiate datasets that share
154 conceptual time series identifiers. By default, these identifier and attribute combinations
155 are then stored in a tightly linked relational database. Conceptually this meta data store
156 could optionally be replaced by alternative technology, such as flat files. As the data is
157 stored in individual structured files, the meta-data store acts as a minimal index with
158 most of the work being delegated to the operating system and in turn the file system.

159 **5 IMPLEMENTATION**

160 PhilDB is written in Python because it fits well with the scientific computing ecosystem
161 (Perez et al. 2011). The core of the PhilDB package is the PhilDB database class¹⁰, that
162 exposes high level methods for data operations. These high level functions are designed
163 to be easily used interactively in the IPython interpreter (Perez and Granger 2007) yet
164 still work well in scripts and applications. The goal of interactivity and scriptability are

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

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

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

176 5.1 Features

177 Key features of PhilDB are:

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

186 5.2 Database Format

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

191 The meta-data store tracks attributes using a relational database, with the current
192 implementation using SQLite (Hipp et al. 2015). Actual time series data are stored
193 as flat files on disk, indexed by the meta-data store to determine the path to a given
194 series. The flat files are implemented as plain binary files that store a 'long', 'double',
195 and 'int' for each record. The 'long' is the datetime stored as a 'proleptic Gregorian
196 ordinal' as determined by the Python datetime.datetime.toordinal method¹¹ (van Rossum
197 2015). The 'double' stores the actual value corresponding to the datetime stored in the
198 preceding 'long'. Finally, the 'int' is a meta value for marking additional information
199 about the record. In this version of PhilDB the meta value is only used to flag missing
200 data values. Individual changes to time series values are logged to HDF5 files (The HDF
201 Group 1997) that are kept alongside the main time series data file with every new value
202 written as a row in a table, each row having a column to store the date, value, and meta
203 value as per the file format. In addition, a final column is included to record the date and
204 time the record was written.

¹¹<https://docs.python.org/2/library/datetime.html#datetime.date.toordinal>

205 6 EVALUATION

206 Of the open-source systems evaluated (as identified in section 2.2), InfluxDB came the
207 closest in terms of minimal initial installation requirements and feature completeness,
208 however, it doesn't support the key feature of update logging. Contrasting with InfluxDB,
209 SciDB met the requirement of time series storage with update logging but didn't meet
210 the requirement for simplicity to deploy and use. Both these systems were evaluated in
211 comparison to PhilDB.

212 To simplify the evaluation process and make it easily repeatable, the SciDB 14.3
213 virtual appliance image¹² was used to enable easy use of the SciDB database. This
214 virtual appliance was based on a CentOS Linux 6.5 install. The PhilDB and InfluxDB
215 databases were installed into the same virtual machine to enable comparison between
216 systems. The virtual machine host was a Mid-2013 Apple Macbook Air, with a 1.7 GHz
217 Intel Core i7 CPU, 8GB of DDR3 RAM and a 500GB SSD hard drive. VirtualBox 4.3.6
218 r91406 was used on the host machine for running the virtual appliance image with the
219 guest virtual machine being allocated 2 processors and 4GB of RAM.

220 Write performance was evaluated by writing all time series from the evaluation
221 dataset (described in section 6.1) into the time series databases being evaluated. This first
222 write will be referred to as the initial write for each database. To track the performance
223 of subsequent updates and reading the corresponding logged time series a further four
224 writes were performed. These writes will be referred to as 'first update' through to
225 'fourth update'. The update data was created by multiplying some or all of the original
226 time series by 1.1 as follows:

- 227 * First update: multiplied the last 10 values in the time series by 1.1 leaving the rest
228 of the record the same.
- 229 * Second update: multiplied the first 10 values by 1.1, resulting in reverting the
230 previously modified 10 values.
- 231 * Third update: multiplied the entire original series by 1.1 resulting in an update to
232 all values aside from the first 10.
- 233 * Fourth update: the original series multiplied by 1.1 again, which should result in
234 zero updates.

235 The SciDB load method used in this experiment did not support updating individual
236 values. The entire time series needed to be passed or the resulting array would consist
237 of only the supplied values. Due to this only full updates were tested and not individual
238 record updates or appends.

239 Performance reading the data back out of each database system was measured by
240 recording the time taken to read each individual time series, after each update, and
241 analysing those results.

242 As can be seen by figure 1, InfluxDB performance was a long way behind SciDB and
243 PhilDB. Given the performance difference and that InfluxDB doesn't support change
244 logging only the initial load and first read were performed for InfluxDB.

¹²<https://downloads.paradigm4.com/QuickStart/14.3/SciDB14.3-CentOS6-VirtualBox-4.2.10.ova>



Figure 1. Mean write/read time for 221 daily time series

245 Disk usage was measured by recording the size of the data directories as reported by
 246 the 'du' Unix command. The size of the data directory was measured before loading
 247 any data and subtracted from subsequent sizes. Between each data write (initial load
 248 and four updates) the disk size was measured to note the incremental changes.

249 For both PhilDB and SciDB the evaluation process described in this section was
 250 performed four times and the mean of the results analysed. Results between the four
 251 runs were quite similar so taking the mean gave results similar to the individual runs.
 252 Analysing and visualising an individual run rather than the mean would result in the
 253 same conclusions.

254 6.1 Evaluation dataset

255 The Hydrological Reference Stations (Zhang et al. 2014) dataset from the Australian
 256 Bureau of Meteorology¹³ was used for the evaluation. This dataset consists of daily
 257 streamflow data for 221 time series with a mean length of 16,310 days, the breakdown
 258 of the series lengths are in table 1 and visualised in figure 2.

259 6.2 InfluxDB

260 Paul Dix (CEO of InfluxDB) found that performance and ease of installation were the
 261 main concerns of users of existing open-source time series database systems (Dix 2014).
 262 InfluxDB was built to alleviate both those concerns.

¹³<http://www.bom.gov.au/water/hrs/>

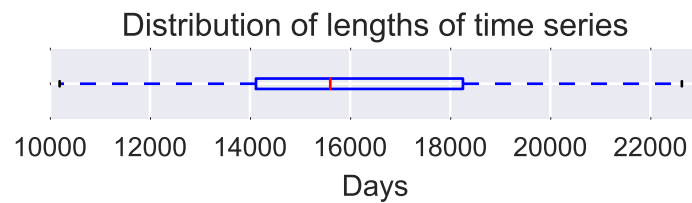


Figure 2. Distribution of time series length for the 221 time series in the evaluation dataset

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)

263 While InfluxDB is designed for high performance data collection, it is not designed
 264 for bulk loading of data. Searching the InfluxDB issue tracker on github¹⁴, it can be
 265 seen that bulk loading has been a recurring problem with improvement over time. Bulk
 266 loading performance is, however, still poor compared to SciDB and PhilDB, as seen
 267 later in the performance results (section 6.5). A key feature of interest with InfluxDB
 268 was the ability to identify time series with tags. This feature is in line with the attributes
 269 concept used by PhilDB, thereby allowing multiple time series to be grouped by a single
 270 key identifier but separated by additional attributes or tags.

271 **6.2.1 Installation**

272 InfluxDB is easily installed compared to the other open-source systems reviewed, as
 273 demonstrated by the short install process shown below. Installation of pre-built packages
 274 on Linux requires root access¹⁵. Installation of InfluxDB was performed in the CentOS
 275 Linux 6.5 based virtual machine containing the pre-installed SciDB instance.

```
276 wget http://influxdb.s3.amazonaws.com/influxdb-0.9.6.1-1.x86_64.rpm
277 sudo yum localinstall influxdb-0.9.6.1-1.x86_64.rpm
```

278 Starting the InfluxDB service with:

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

280 **6.2.2 Usage**

281 Loading of data into the InfluxDB instance was performed using the InfluxDB Python
 282 API that was straight forward to use. However, poor performance of bulk loads lead to a

¹⁴<https://github.com/influxdata/influxdb/issues>

¹⁵<https://influxdb.com/docs/v0.9/introduction/installation.html>

283 lot of experimentation on how to most effectively load large amounts of data quickly,
 284 including trying curl and the Influx line protocol format directly. The final solution
 285 used was to chunk the data into batches of 10 points using the Pandas groupby func-
 286 tionality before writing into InfluxDB using the InfluxDB Python API DataFrameClient
 287 write_points method, for example:

```
288 streamflow = pandas.read_csv(filename, parse_dates=True, index_col=0, header = None)
289 for k, g in streamflow.groupby(np.arange(len(streamflow))//100):
290     influx_client.write_points(g, station_id)
```

291 In addition to experimenting with various API calls, configuration changes were at-
 292 tempted resulting in performance gains by lowering values related to the WAL options
 293 (the idea was based on an older GitHub issue discussing batch loading¹⁶ and WAL
 294 tuning to improve performance). Despite all this effort, bulk data loading with InfluxDB
 295 was impractically slow with a run time generally in excess of one hour to load the 221
 296 time series (compared to the less than 2 minutes for SciDB and PhilDB). Reading was
 297 performed using the Python API InfluxDBClient query method:

```
298 streamflow = influx_client.query('SELECT_*_FROM_Q{0}'.format('410730'))
```

299 6.3 PhilDB

300 PhilDB has been designed with a particular use case in mind as described in section
 301 3.1. Installation of PhilDB is quite easy where a compatible Python environment exists.
 302 Using a Python virtualenv removes the need to have root privileges to install PhilDB
 303 and no dedicated user accounts are required to run or use PhilDB. A PhilDB database
 304 can be written to any location the user has write access, allowing for experimentation
 305 without having to request a database be created or needing to share a centralised install.

306 6.3.1 Installation

307 Installation of PhilDB is readily performed using pip:

```
308 pip install phildb
```

309 6.3.2 Usage

310 The experimental dataset was loaded into a PhilDB instance using a Python script. Using
 311 PhilDB to load data can be broken into three key steps.

312 First, initialise basic meta information:

```
313 db.add_measurand('Q', 'STREAMFLOW', 'Streamflow')
314 db.add_source('BOM_HRS', 'Bureau_of_Meteorology;_Hydrological_Reference_Stations_'
315             'dataset.')
```

316 This step only need to be performed once, when configuring attributes for the PhilDB
 317 instance for the first time, noting additional attributes can be added later.

318 Second, add an identifier for a time series and a time series instance record based on
 319 the identifier and meta information:

```
320 db.add_timeseries(station_id)
321 db.add_timeseries_instance(station_id, 'D', '', measurand = 'Q', source = 'BOM_HRS')
```

322 Multiple time series instances, based on different combinations of attributes, can be
 323 associated with an existing time series identifier. Once a time series instance has been
 324 created it can be written to and read from.

325 Third, load the data from a Pandas time series:

¹⁶<https://github.com/influxdata/influxdb/issues/3282>

```
326 streamflow = pandas.read_csv(filename, parse_dates=True, index_col=0, header = None)
327 db.write(station_id, 'D', streamflow, measurand = 'Q', source = 'BOM_HRS')
```

328 In this example the Pandas time series is acquired by reading a CSV file using the Pandas
329 read_csv method, but any data acquisition method that forms a Pandas.Series object
330 could be used. Reading a time series instance back out is easily performed with the read
331 method:

```
332 streamflow = db.read(station_id, 'D', measurand = 'Q', source = 'BOM_HRS')
```

333 The keyword arguments are optional provided the time series instance can be uniquely
334 identified.

335 6.4 SciDB

336 SciDB, as implied by the name, was designed with scientific data in mind. As a result
337 SciDB has the feature of change logging, allowing past versions of series to be retrieved.
338 Unfortunately SciDB only identifies time series by a single string identifier, therefore
339 storing multiple related time series would require externally managed details about what
340 time series are stored and with what identifier. Due to the sophistication of the SciDB
341 system it is relatively complex to use with two built in languages, AFL and AQL, that
342 allow for two different approaches to performing database operations. This, in turn,
343 increases the amount of documentation that needs to be read to identify which method
344 to use for a given task (such as writing a time series into the database). While the
345 documentation is comprehensive in detailing the available operations, it is largely based
346 on the assumption that the data is already within SciDB and will only be operated on
347 within SciDB, with limited examples on how to load or extract data via external systems.

348 6.4.1 Installation

349 SciDB does not come with binary installers for newer versions and the build process is
350 quite involved. Instructions for the build process are only available from the SciDB
351 forums using a registered account¹⁷. Installation of older versions is comparable to
352 InfluxDB with the following steps listed in the user guide:

```
353 yum install -y https://downloads.paradigm4.com/scidb-14.12-repository.rpm
354 yum install -y scidb-14.12-installer
```

355 Same as InfluxDB, SciDB requires root access to install and a dedicated user account
356 for running the database. A PostgreSQL installation is also required by SciDB for
357 storing information about the time series data that SciDB stores. Unlike InfluxDB,
358 SciDB has authentication systems turned on by default that requires using dedicated
359 accounts even for basic testing and evaluation.

360 Only Ubuntu and CentOS/RHEL Linux variants are listed as supported platforms in
361 the install guide.

362 6.4.2 Usage

363 It took a considerable amount of time to identify the best way to load data into a SciDB
364 instance, however once that was worked out, the actual load was quick and effective
365 consisting of two main steps.

366 First, a time series needs to be created:

¹⁷<http://paradigm4.com/forum/viewtopic.php?f=14&t=1672&sid=6e15284d9785558d5590d335fed0b059>

```
367 iquery -q "CREATE_ARRAY_Q${station}_<date:datetime,_streamflow:double>_[i  
368 =0:*,10000,0];"
```

369 It is worth noting that datetime and double need to be specified for time series storage,
370 since SciDB can hold many different array types aside from a simple time series.
371 Additionally, SciDB identifiers can not start with a numeric character so all time series
372 identifiers were prefixed with a 'Q' (where 'Q' was chosen in this case because it is
373 conventionally used in the hydrological context to represent streamflow discharge).

374 Second, the data is written using the iquery LOAD method as follows:

```
375 iquery -n -q "LOAD_Q${station}_FROM_ '/home/scidb/S${station}.scidb';"
```

376 This method required creating data files in a specific SciDB text format before hand
377 using the csv2scidb command that ships with SciDB.

378 Identifying the correct code to read data back out required extensive review of
379 the documentation, but was quick and effective once the correct code to execute was
380 identified. The SciDB Python code to read a time series back as a Pandas.DataFrame
381 object is as follows:

```
382 streamflow = sdb.wrap_array('Q' + station_id).todataframe()
```

383 A contributing factor to the difficulty of identifying the correct code is that syntax errors
384 with the AQL based queries (using the SciDB iquery command or via the Python API)
385 are at times uninformative about the exact portion of the query that is in error.

386 6.5 Performance

387 It should be noted that PhilDB currently only supports local write, which is advantageous
388 for performance, compared to InfluxDB that only supports network access. InfluxDB
389 was hosted locally, which prevents network lag, but the protocol design still reduced
390 performance compared to the direct write as done by PhilDB. Although SciDB has
391 network access, only local write performance (using the SciDB iquery command) and
392 network based read access (using the Python API) were evaluated. SciDB was also
393 accessed locally to avoid network lag when testing the network based API. For a
394 comparable network read access comparison the experimental PhilDB Client/Server
395 software was also used.

396 6.5.1 Write performance

397 Write performance was measured by writing each of the 221 time series into the database
398 under test and recording the time spent per time series.

399 As can be seen in figure 1, SciDB and PhilDB have a significant performance
400 advantage over InfluxDB for bulk loading of time series data. SciDB write performance
401 is comparable to PhilDB, so a closer comparison between just SciDB and PhilDB write
402 performance is shown in figure 3.

403 It can be seen that while PhilDB has at times slightly better write performance,
404 SciDB has more reliable write performance with a tighter distribution of write times.
405 It can also be seen from figure 3 that write performance for SciDB does marginally
406 decrease as more updates are written. PhilDB write performance while more variable
407 across the dataset is also variable in performance based on how much of the series
408 required updating. Where the fourth update writes the same data as the third update it
409 can be seen that the performance distribution is closer to that of the initial load than the
410 third load, since the data has actually remained unchanged.

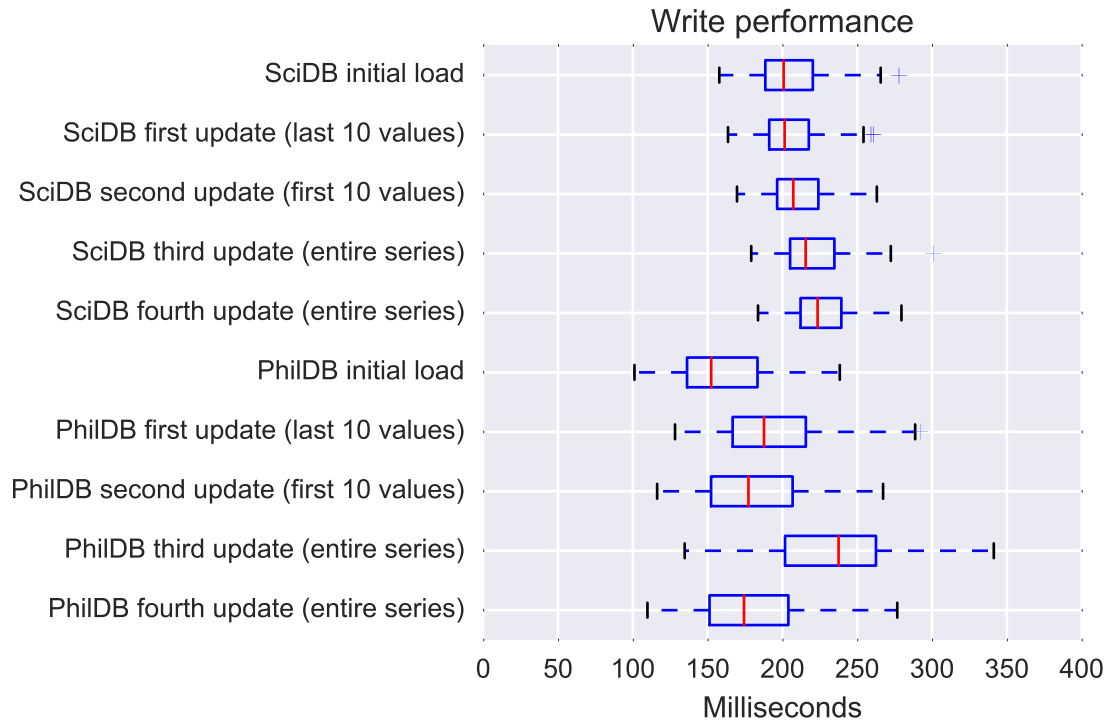


Figure 3. Distribution of write times for 221 time series

411 Both SciDB and PhilDB perform well at loading datasets of this size with good write
412 performance.

413 **6.5.2 Read performance**

414 InfluxDB read performance is adequate and SciDB read speed is quite good, however
415 PhilDB significantly out-performs both InfluxDB and SciDB in read speed, as can be
416 seen in figure 1. Even the PhilDB server/client model, which has yet to be optimised for
417 performance, out-performed both InfluxDB and SciDB. Read performance with PhilDB
418 is consistent as the time series are updated, as shown in figure 4, due to the architecture
419 keeping the latest version of time series in a single file. Reading from the log with
420 PhilDB does show a decrease in performance as the size of the log grows, but not as
421 quickly as SciDB. While PhilDB maintains consistent read performance and decreasing
422 log read performance, SciDB consistently decreases in performance with each update
423 for reading both current and logged time series.

424 **6.5.3 Disk usage**

425 After the initial load InfluxDB was using 357.21 megabytes of space. This may be
426 due to the indexing across multiple attributes to allow for querying and aggregating
427 multiple time series based on specified attributes. This is quite a lot of disk space being
428 used compared to SciDB (93.64 megabytes) and PhilDB (160.77 megabytes) after the
429 initial load. As can be seen in figure 5, SciDB disk usage increases linearly with each
430 update when writing the entire series each time. In contrast, updates with PhilDB only
431 result in moderate increases and depends on how many values are changed. If the time



Figure 4. Distribution of read durations for the 221 time series from the evaluation dataset

432 series passed to PhilDB for writing is the same as the already stored time series then
 433 no changes are made and the database size remains the same, as can be seen between
 434 update 3 and 4 in figure 5.

435 **6.5.4 Performance summary**

436 Each database has different design goals that results in different performance profiles.
 437 InfluxDB is not well suited to this use case with a design focusing on high performance
 438 writing of few values across many time series for metric collection, leading to poor
 439 performance for bulk loading of individual time series.

440 SciDB fares much better with consistent read and write performance, with slight
 441 performance decreases as time series are updated, likely due to design decisions that
 442 focus on handling large multi-dimensional array data for high performance operations.
 443 Design decisions for SciDB that lead to consistent read and write performance appear to
 444 also give the same read performance when accessing historical versions of time series.
 445 Achieving consistent read and write performance (including reading historical time
 446 series) seems to have come at the expense of disk space with SciDB consuming more
 447 space than PhilDB and increasing linearly as time series are updated.

448 PhilDB performs quite well for this particular use case, with consistently fast reads
 449 of the latest time series. This consistent read performance does come at the expense of
 450 reading historical time series from the logs, which does degrade as the logs grow. Write
 451 performance for PhilDB, while variable, varies due to the volume of data changing.

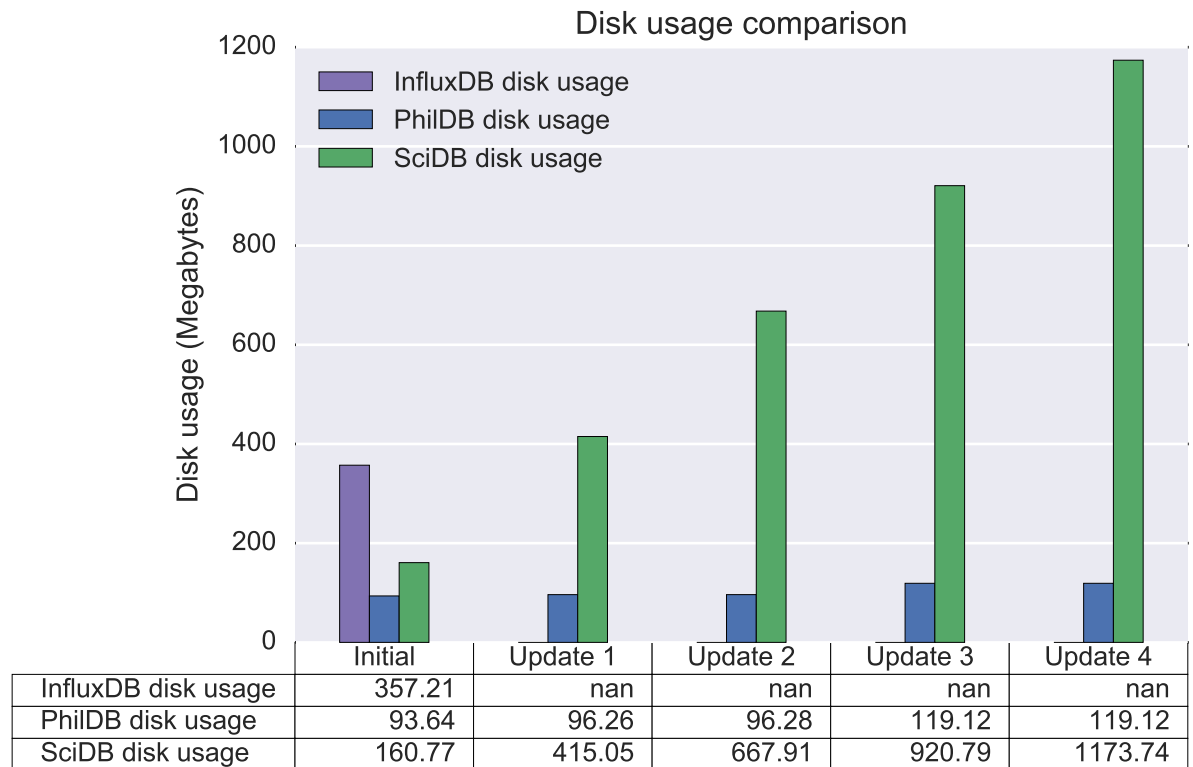


Figure 5. Disk usage after initial data load and each subsequent data update

452 The performance of PhilDB (particularly the excellent read performance) compared
 453 to SciDB for this use case was unexpected since the design aimed for a simple API at
 454 the expense of efficiency.

455 7 FUTURE WORK

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

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

465 **8 CONCLUSION**

466 In conclusion, there is a need for an accessible time series database that can be deployed
467 quickly so that curious minds, such as those in our scientific community, can easily
468 analyse time series data and elucidate world-changing information. For scientific
469 computing, it is important that any solution is capable of tracking subsequent data
470 changes.

471 Although InfluxDB comes close with features like tagging of attributes and a clear
472 API, it lacks the needed change logging feature and presently suffers poor performance
473 for bulk loading of historical data. InfluxDB has clearly been designed with real-time
474 metrics based time series in mind and as such doesn't quite fit the requirements outlined
475 in this paper.

476 While SciDB has the important feature of change logging and performs quite well,
477 it doesn't have a simple mechanism for tracking time series by attributes. SciDB is
478 well suited for handling very large multi-dimensional arrays, which can justify the steep
479 learning curve for such work, but for simple input/output of plain time series such
480 complexity is a little unnecessary.

481 PhilDB addresses this gap in existing solutions, as well as surpassing them for
482 efficiency and usability. Finally, PhilDB's source code has been released on GitHub¹⁸
483 under the permissive 3-clause BSD open-source license to help others easily extract
484 wisdom from their data.

485 **9 ACKNOWLEDGEMENTS**

486 I would like to thank Di MacDonald for her editorial advice on early drafts, my fiancée
487 Katrina Cornelly for her support and editorial advice, and my colleague Richard Lauge-
488 sen for his valuable review comments on an earlier draft. PhilDB was named in memory
489 of my father Phillip MacDonald.

¹⁸<https://github.com/amacd31/phildb>

490 **REFERENCES**

- 491 Brown, C. T. (2015). On licensing bioinformatics software: use the BSD, Luke.
492 <http://ivory.idyll.org/blog/2015-on-licensing-in-bioinformatics.html>.
- 493 Castillejos, A. M. (2006). *Management of Time Series Data*. PhD thesis, University of
494 Canberra.
- 495 Cudré-Mauroux, P., Kimura, H., Lim, K.-T., Rogers, J., Simakov, R., Soroush, E.,
496 Velikhov, P., Wang, D. L., Balazinska, M., Becla, J., and others (2009). A demon-
497 stration of SciDB: a science-oriented DBMS. *Proceedings of the VLDB Endowment*,
498 2(2):1534–1537.
- 499 Dix, P. (2014). InfluxDB: One year of InfluxDB and the road to
500 1.0. <https://influxdb.com/blog/2014/09/26/one-year-of-influxdb-and-the-road-to-1.0.html>.
- 501
- 502 Hipp, D. R., Kennedy, D., and Mistachkin, J. (2015). SQLite.
503 <https://www.sqlite.org/download.html>.
- 504 Hunter, J. D. (2004). Why we should be using BSD.
505 http://nipy.sourceforge.net/nipy/stable/faq/johns_bsd_pitch.html.
- 506 Leach, P. J., Mealling, M., and Salz, R. (2005). A Universally Unique Identifier (UUID)
507 URN Namespace.
- 508 MacDonald, A. (2015). phildb: Version 0.6.1. <http://dx.doi.org/10.5281/zenodo.32437>.
- 509 McKinney, W. (2012). *Python for Data Analysis: Data Wrangling with Pandas, NumPy,*
510 *and IPython*. "O'Reilly Media, Inc."
- 511 Perez, F. and Granger, B. E. (2007). IPython: a system for interactive scientific comput-
512 ing. *Computing in Science & Engineering*, 9(3):21–29.
- 513 Perez, F., Granger, B. E., and Hunter, J. D. (2011). Python: an ecosystem for scientific
514 computing. *Computing in Science & Engineering*, 13(2):13–21.
- 515 Shin, D., Schepen, A., Peatey, T., Zhou, S., MacDonald, A., Chia, T., Perkins, J., and
516 Plummer, N. (2011). WAFARi: A new modelling system for Seasonal Streamflow
517 Forecasting service of the Bureau of Meteorology, Australia. In *MODSIM 2011,*
518 *Modelling and Simulation Society of Australian and New Zealand*.
- 519 Stonebraker, M., Becla, J., DeWitt, D. J., Lim, K.-T., Maier, D., Ratzesberger, O., and
520 Zdonik, S. B. (2009). Requirements for Science Data Bases and SciDB. In *CIDR*,
521 volume 7, pages 173–184.
- 522 Stonebraker, M., Brown, P., Poliakov, A., and Raman, S. (2011). The Architecture of
523 SciDB. In Cushing, J. B., French, J., and Bowers, S., editors, *Scientific and Statistical*
524 *Database Management*, number 6809 in Lecture Notes in Computer Science, pages
525 1–16. Springer Berlin Heidelberg.
- 526 Sweller, J., Ayres, P. L., and Kalyuga, S. (2011). *Cognitive load theory*. Explorations in
527 the learning sciences, instructional systems and performance technologies. Springer,
528 New York.
- 529 The HDF Group (1997). Hierarchical Data Format, version 5.
530 <http://www.hdfgroup.org/HDF5/>.
- 531 van Rossum, G. (2015). Python Programming Language. www.python.org.
- 532 VanderPlas, J. (2014). The Whys and Hows of Licensing Scientific
533 Code. [http://www.astrobetter.com/blog/2014/03/10/the-whys-and-hows-of-licensing-](http://www.astrobetter.com/blog/2014/03/10/the-whys-and-hows-of-licensing-scientific-code/)
534 [scientific-code/](http://www.astrobetter.com/blog/2014/03/10/the-whys-and-hows-of-licensing-scientific-code/).

535 Zhang, S. X., Bari, M., Amirthanathan, G., Kent, D., MacDonald, A., Shin, D., and
536 others (2014). Hydrologic reference stations to monitor climate-driven streamflow
537 variability and trends.