BIOINFORMATICS

Vol. 00 no. 00 2013
Pages 1–2

Wormtable: A data structure for genome scale tabular data

Jerome Kelleher, Daniel L. Halligan and Robert W. Ness
University of Edinburgh, King's Buildings, West Mains Road, EH9 3JT, UK

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT
Summary: Biological research generates vast quantities of tabular data, which must be processed to answer biological questions. These tables are produced by programs and exported to plain text files, which are often tens of gigabytes long. Researchers must process these files line-by-line, parsing values encoded as plain text into native types so that calculations may be done. This is an enormously inefficient process. Furthermore, there is no simple means of indexing these files so that arbitrary data values within rows can be quickly found. We introduce a new data format and software library called Wormtable, which provides efficient access to tabular data using languages such as Python. Wormtable avoids the overhead of repeatedly parsing text, random access to rows, and also provides efficient indexing of arbitrary columns within these tables.
Availability: Package available at http://pypi.python.org/pypi/wormtable
Contact: jerome.kelleher@ed.ac.uk

1 INTRODUCTION

The large volumes of data that present day bioinformatics generates is well documented. The improvement in sequencing technologies has lead to an exponential increase in the volume of data that is available to biologists. Perfectly standard studies require the storage and processing of hundreds of gigabytes of data, and this volume of data is only set to increase as sequencing technologies continue to improve. [*This intro is intended to set the stage: there is shit loads of data out there, and this is only set to increase*].

Despite the increasing importance of data processing, however, the methods used in Bioinformatics are crude and inefficient being almost entirely based around the use of text files. Plain text files have many advantages: they are the most portable format available, and can be processed on any platform with the minimum of library dependencies. Text files also have many disadvantages however; the principle difficulties for our purposes here are, they are slow to parse and difficult to index.

A popular file format for Bioinformatics use is the Variant Call Format [**?**], or VCF. In VCF, information about variant sites in a genome is encoded as tab-delimited rows in a text file. To access information about a particular site, one must proceed through the file line-by-line until the desired record is found. There are methods available to index this file [**?**], but these have no semantic understanding of the structure of the file and so are limited in the fields that can be indexed. One cannot, for example, build an index to quickly obtain all records in which the quality field is greater than some value. Such queries can take a considerable time on large VCF files, which discourages testing and checking for the effects of arbitrary thresholds in analyses.

One may, of course, load VCF data into a relational database, and use SQL to retrieve the records of interest. This approach has several disadvantages, however. Firstly, one must first examine the VCF header and construct an appropriate SQL schema to hold the columns of interest. It is feasible to do this for more recent versions of VCF, but is certainly not a trivial process, particularly if the schema must be portable across different relational database platforms. Once the schema has been constructed, the data must be inserted into this database line-by-line by constructing SQL code, which is a time-consuming and error prone process. Even assuming that the data has been inserted into a table with an appropriate schema, there are still difficulties to be addressed. Maintaining a database server is a difficult task and when the volumes of data become large, a specialised administrator is usually required to ensure the server works correctly.

Much of the time, a database server is far more complicated than we require for large scale biological data. In the case of VCF, the file is written once by an application such as [gatk? references?] and is not subsequently expected to change. Thus, storing this information in a relational database with its sophisticated concurrency control and guarantees of consistency in the face of asynchronous updates is entirely unnecessary. Even the client-server architecture of a typical relational database is wasteful in this case: protocol commands must be read and parsed, and responses must be encoded, written to a socket and subsequently converted back to native types, all of which constitutes a significant overhead when streaming large quantities of data. The limitations of relational database servers have been acknowledged for some time now, and has led to a variety of 'NoSQL' database systems [**?**].

Databases do, however, have significant advantages over plain text files. Data is typically encoded in an efficient binary format, saving substantial space and giving better precision when dealing with floating point formats. Records are stored in a B-tree type structure, leading to fast access times even over very large datasets. Indexing over columns within rows is also very powerful, allowing us to quickly retrieve rows with particular features. Ideally, we would like a data format that combines these useful features of databases without the drawbacks of a relational database server outlined above.

Wormtable is a new data format and software library designed specifically to deal with the challenges of data processing in bioinformatics. It provides a portable, compact, read-only store for tabular data with high-performance access to rows via indexes. Tables can be either written directly via the software library, or converted from existing formats via command line tools. Tables can be read concurrently by any number of processes, and are accessed by a simple API in Python.

2 WORMTABLE

To take advantage of the data processing capabilities of a database without the overheads and disadvantages of a relational database server, Wormtable is built on top of Berkeley DB [**?**]. Berkeley DB is an open source embedded database toolkit, that provides a scalable key-value store. It is a mature and stable platform, and is currently the most widely deployed database toolkit in the world [**?**].

Berkeley DB (or simply DB) differs from relational database servers in several ways. Firstly, it is a software library that a program links against, and so there is no need for a separate database server process. Secondly, DB provides only a key-value store, and so has no knowledge of the structure of the records that it stores, treating them simply as an array of bytes. While there are sophisticated mechanisms within DB for locking, transactions and so on, these are all optional, allowing a very lightweight read-only store to be built using the library.

This is the approach taken in Wormtable. We assume that tabular data is written once, by some program generating the data, and subsequently read many times. Under these assumptions, the primary database holding the rows can be opened in a read-only state, allowing any number of processes to read it concurrently in a lock free manner.

Since DB is agnostic to the structure of data stored in rows, we are free to design our own formats. In Wormtable, the row format is a compact and flexible binary format holding integer, floating point or character values of various sizes. Each column in a row may contain either a fixed or variable number of elements (each element being of a fixed type and size). This row format enables two important properties: firstly, values from any column can be accessed in constant time; and secondly, values in columns can be converted directly into native machine values, ready for calculations.

Indexing of columns is achieved via DB also. An index is simply a database mapping values in a column (or list of columns) to rows in the primary database. Indexes allow us to find rows with particular properties very quickly, and are a very useful method of excluding rows with certain properties from our analyses.

3 EXAMPLES

Wormtable is designed to make working with genome scale data more efficient, and in this section we describe some examples to illustrate the improvements over existing methods. To do this, we converted a VCF file with XXX rows from the YYY project into wormtable format. The (uncompressed) VCF file was XX gigabytes, and the converted wormtable YY gigabytes, which required X hours.

Values are stored in wormtable in a binary format, so that no parsing is required when reading in rows. Methods using the VCF directly must parse each row before it can be used, and this parsing is by far the most time consuming part of processing VCF data. To illustrate this advantage, we wrote a script to count the number of transitions and transversions in the dataset, using wormtable and using PyVCF[citation?]. Both of these methods proceeded row-by-row, checking the ALT and REF columns. The PyVCF method required X hours to complete, while the wormtable based script needed only Y seconds. This stark difference is not a criticism of the PyVCF library, which provides an efficient parser for VCF files; it is rather a measure of the inherent slowness of parsing text in large volumes.

It is well known, of course, that plain text cannot be retrieved efficiently at large scales. The BVCF format is an binary version of VCF which is intended to avoid this overhead [citation?]. This binary format does alleviate the problem of parsing overhead, but does not solve the problem of indexing *within* rows.

To illustrate the use of indexes in wormtable, we created an index on the ALT and REF columns, and another on the IN, UT, SY and NS columns. These indexes required X and Y MB of disc space and X and Y minutes to build, respectively. The ALT+REF index provides a much more efficient means of counting the transitions and transversions, and another script to calculate these required x seconds. Using the IN+UT+SY+NS index, we calculated the mean nucleotide diversity for SNPs (not indels) at synonymous, nonsynonymous, intronic and UTR sites. This required X seconds. Computing the same values with PyVCF required Y hours.

The examples in this section are intended to illustrate the magnitude of performance gains that wormtable can provide when working with large datasets. The gains are many: we do not have to parse the information before it is used; we can quickly seek to arbitrary rows within a table; and we can use indexes to intelligently sort and aggregate our data to make it more useful.

4 CONCLUSIONS

Text files are the oldest and most reliable means of exchanging information between computers. Plain text, however, does not provide an efficient means of storing and processing large volumes of data. Current practice in bioinformatics is to encode data in more and more complex plain text data formats. While these formats are excellent from the perspective of minimising software dependencies, they are very inefficient in terms of storage space, the time required to parse them, and the linear seek times to find particular rows.

Wormtable is not intended to replace text files as the universal interchange format. It is intended to provide a persistent data structure that efficiently holds data in a form that can be efficiently processed and searched. Using this data structure, researchers with no knowledge of database systems can take full advantage of sophisticated data management techniques, and write simple code that processes this data in a very high performance manner. In our example above, we [blah blah blah]. A systematic application of these techniques can substantially increase the researcher's productivity and ability to explore their data.

Wormtable is an open and collaborative project seeking feedback and contributors. The project is in its early stages, currently providing only a Python interface. A C library, Perl and Java interfaces are also planned for Wormtable, which should make the format available to the majority of bioinformatics projects. With such broad support, it is hoped that other software tools may use these libraries to support direct output of data in Wormtable format.

ACKNOWLEDGEMENT

Funding: JK is supported by EPSRC grant EP/I013091/1; DLH is supported by XXXXX; and RWN is supported by XXXX.

REFERENCES

Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, Richard Durbin, and 1000 Genomes Project Analysis Group. The variant call format and VCFtools. *Bioinformatics*, 27(15): 2156–2158, 2011.

Heng Li. Tabix: fast retrieval of sequence features from generic TAB-delimited files. *Bioinformatics*, 27(5):718–719, 2011.

Adam Marcus. The NoSQL ecosystem. In Amy Brown and Greg Wilson, editors, *The Architecture of Open Source Applications*, volume 1. ISBN 978-1-257-63801-7, 2012.

Michael A. Olson, Keith Bostic, and Margo Seltzer. Berkeley DB. In *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*, 1999.

Margo Seltzer and Keith Bostic. Berkeley DB. In Amy Brown and Greg Wilson, editors, *The Architecture of Open Source Applications*, volume 1. ISBN 978-1-257-63801-7, 2012.