Using SQLite to Manage a 16S RNA Sequencing Analysis Pipeline

Xu Junjie, Kevin

June 2014

Abstract

16S Ribosomal RNA Sequencing is used extensively in analzying bacterial phylogeny and taxonomy. This project attempts to streamline the 16S sequencing pipeline using local file databases to replace multiple flat sequence files used in the pipeline, to ease logistical burdens on the researcher and enable greater metadata analysis and accountability of experiments.

Contents

1 Introduction **2**

1.1 Reduction in intermediate work products 5

1.2 Reduction in overall disk usage 5

1.3 Faster pipeline runs 6
1.4 Improving reproducibility 6
1.5 Easier organization and collaboration 6

2 Data Management **7**
2.1 Semantic Database Structure 8
2.2 Data Compression 10
2.3 Data Transformation & Flow 10

3 Results **12**

4 Conclusion **15**

1 Introduction

The 16S small subunit of bacterial ribosomes are gene sequences found in all bacteria, which means that the differences within the 16S RNA profiles can be used as an analogue for species identity. Since the 16S gene contains both highly conserved and highly variable regions all interspersed together, the highly conserved regions cut, while the variable regions are amplified using PCR (Polymerase chain reaction) to classify the organism. HTS (High Throughput Sequencing) using Illumina sequencers are then used to sequence the PCR products from the prior step.

The output from HTS forms the start of the computational pipeline. In the preprocessing stage, poor quality reads are filtered and trimmed. Non-bacterial

sequences and other contaminants are then removed. The pipeline then attempts to produce phylogenetic classification based on sequence similarity (i.e. OTU Analysis). By comparing the filtered output from HTS against the already existing taxonomies of over 2 million species, the genus, family, and order of the sample can be determined.

The majority of the pipeline is executed on the University of Oregon's ACISS High-Performance Supercomputer Cluster, and utilizes PBS scripts (normal shell scripts with extra variables defined to manage job resources) to execute the different stages of the pipeline. Figure 1 illustrates the stages in the pipeline. The FASTQ output produced by the Illumina sequencer is run through the preprocessing stage of filtering, trimming, and demultiplexing. The PBS script for the preprocessing stage calls on the Demultiplexer, a Python script written by Rodger Voelker, which removes the primer attached to the sequences during the amplification process, and attaches barcodes (signifying sample origin) to both ends of the paired-end reads in the FASTQ file. After demultiplexing, the pipeline proceeds to use Trimmomatic v0.32, an open source tool to trim poor quality feeds from the reads. It uses sliding window trimming to cut out sequences when the average quality within a window falls below a certain threshold. Programs in the QIIME package are then used for clustering and taxonomic classification stages.

Every stage within a pipeline consumes a set of input files and produces a set of output files, so a single run of an experiment through the pipeline will produce a large number of work products, which must be managed manually

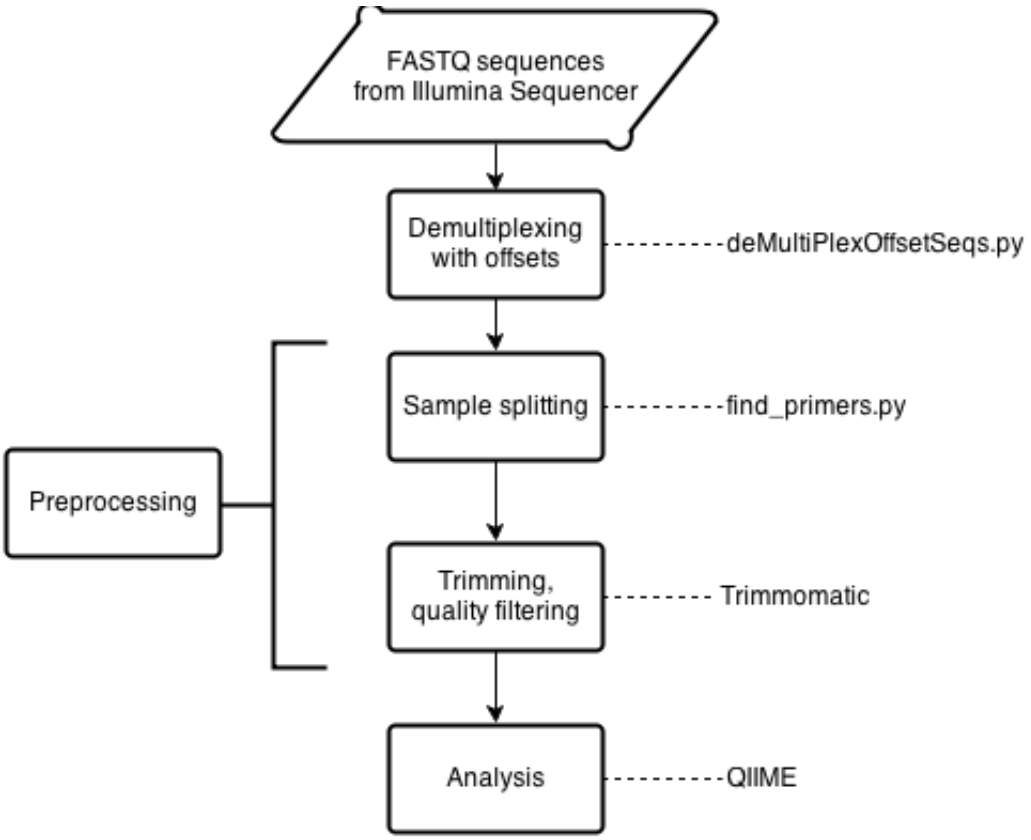

Figure 1: 16S pipeline stages

by the user of the pipeline. Table 1 shows the number of files that will accumulate through running through the stages in a pipeline.

Stage	# of Input Files	# of Output Files
Trimmomatic	2	5
Bowtie2	3	7
QIIME (Assign multiplex reads)	3	3

Table 1: Number of files involved in 16S pipeline stages

PIPPING was developed as an extension to PIP[6] (*P*ipeline *I*nterface

*P*rogram), a tool that manages bioinformatics pipelines using MySQL databases. The project focused on the preprocessing (primer splitting, trimming, quality filtering) stages of the pipeline, preparing the work products from the raw sequencing data produced by the Illumina NGS machines to be fed into the analysis (clustering, alignment, phylogenic inference) stages. Instead of using MySQL databases, PIPPING explored the feasibility of storing work products in a SQLite database, with the following goals.

1.1 Reduction in intermediate work products

Every stage in a 16S pipeline requires a set of input and output files, which increases the organizational workload for the user running the pipeline. By feeding inputs and consuming outputs of stages, a PIPPING database can contain all data within a pipeline in a single file, reducing the amount of file management required of the user.

1.2 Reduction in overall disk usage

Reducing the number of intermediate work products in a pipeline also serves to reduce the amount of disk space consumed by running a pipeline. Temporary work products produced by converting between one file format to another can be reduced or eliminated by performing data transformations in PIPPING and feeding the output to the next stage directly.

1.3 Faster pipeline runs

For programs such as Trimmomatic (which requires only one pass over input data) there is a minor increase in processing speed due to reduced disk access. Additionally, since format conversions can be performed on-the-fly without disk access, conversion stages within a pipeline can be skipped, reducing the overall number of stages in a pipeline.

1.4 Improving reproducibility

A PIPPING database is a container that holds not just the original sequences from the Illumina Sequencer, but also results of transformations performed by stages in the pipeline, and the parameters used in those transformations. Since OTU clustering strategies are sensitive to length and quality filtering parameters upstream, preserving settings used in earlier stages can aid analysis in later stages.

1.5 Easier organization and collaboration

With all the data on an experiment contained in a single file, users can easily share PIPPING databases with collaborators without needing to manage dozens of files for every experiment. The semantic structure of the SQLite tables inside the database provide an open platform for developers to extend Pip, such as streaming adapters for currently unsupported programs.

2 Data Management

Management of data in a pipeline is conceptually divided into four tasks: creating semantic structure within the database; storing sequence data in compressed form; transforming stored data into formats required by tools in the pipeline; logging settings and parameters of transformations and related metadata.

As the first stage in a 16S pipeline, PIPPING reads in a configuration file, primer sequences, barcodes, offsets, and raw Illumina NGS (Next-Gen Sequencing) FASTQ files and uses them to initialize its database structure. PIPPING encapsulates the data with adapters to transform the compressed data into usable input for subsequent stages, and provides metadata logging. Typically, an Illumina NGS machine uses *Paired-End Sequencing*[3] to produce a pair of FASTQ files where each read is saved in the following format:

```
@HWI-ST0747:277:D1M96ACXX:6:1101:1232:2090 1:N:0:
GGATAGTACTAGGGTATCTAATCCTGTTTGCTCCCCACGCT...
+
ACCCFFDDFH#FHHIGIJJFJJJJJJJJJJJIIJJJGIJJG...
```

Line 1 is the *defline* containing information about the Illumina Sequencer which produced the sequence, line 2 contains all the bases in the sequence, line 3 is the separator between the sequence and quality scores, and line 4 contains the corresponding quality score for each of the bases in the sequence. Each character in the file ias an ASCII character taking up 8 bits. The format uses five characters to represent the four DNA-bases (A - adenine, T - thymine, C

- cytosine, G - guanine) and N to represent an errorneous read. Each base is paired with a quality score that has 42 values (0 to 41).

2.1 Semantic Database Structure

Instead of juggling multiple input files such as raw Illumina FASTQ sequences, barcodes, and offsets, PIPPING creates several SQLite tables which impart a well-defined semantic structure to the data and contains them within a single file.

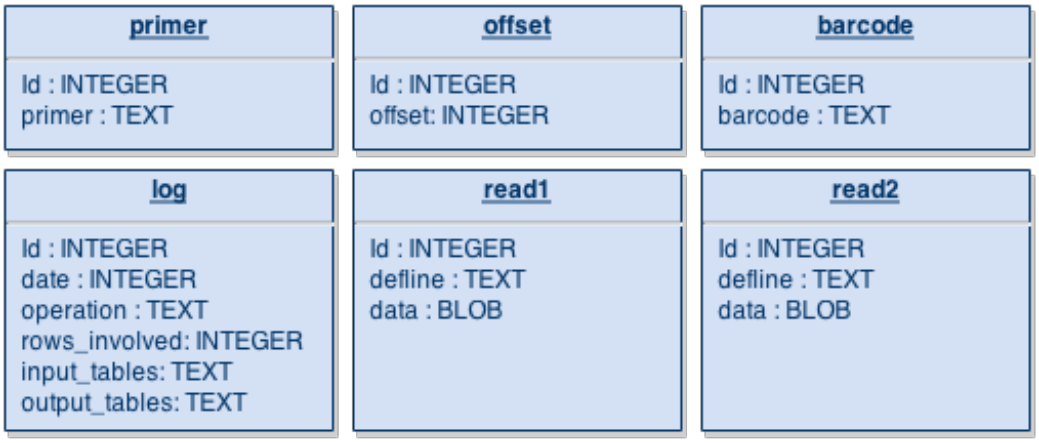

Figure 2: PIPPING database structure

Figure 2 shows the structure of a PIPPING database, initialized with six tables:

primer holds all primer sequences used in the experiment

offset randomized offsets to aid 16S sequencing

barcode experiment identifier

log stores all metadata related to data, experiment, or pipeline

read1 stores the first part of paired-end sequence data

read2 stores the second part of paired-end sequence data

The *primer*, *offset*, and *barcode* tables link information about the experiment to the actual sequences to be stored in the *read1* and *read2* tables. Each sequence in the FASTQ input is stored as a row in the read tables, with the bases and quality scores compressed into a binary BLOB.

The *log* table stores metadata about every operation done in Pip. The settings used in the operation, along with the time and sequences affected are stored in the *log* table. Since the results of OTU Analysis can be affected by parameters used by the stages upstream in a pipeline, having a reliable record of operations and transformations to the data will aid in analysis stages downstream.

Sequence data stored in *read1* and *read2* tables in the database are fed as input to later stages in the pipeline. Correspondingly, output from those stages are normalized into new tables in the database. For example, running Trimmomatic with PIPPING *trimmed1* and *trimmed2* tables which stores the coordinates of trimmed bases from the results of trimming and filtering operations performed by Trimmomatic. This prevents unnecessary data duplication and reduces the file size of a PIPPING database.

2.2 Data Compression

Compression works by representing each base with an integer range (A: 0 to 49, T: 50-99, C: 100-149, G: 150-199, N: 200-255) and adding the quality score to the beginning of that range. Thus, base 'A' with a quality score of '#' (value 0) can be represented by the value 0, while base 'G' with a quality score of 'J' (value 41) can be represented by 191 (150 + 41). The resulting values fit within an 8-bit character, which is a 50% reduction in space usage. Since both compression and decompression require only addition operators, the scheme is fast and lossless. However, the overall size of a PIPPING file is slightly more than 50% of the size of the original FASTQ file due to table index structures and other database overhead.

2.3 Data Transformation & Flow

A central part of managing the sequence data in the 16S pipeline is the manipulation and transferring of sequences between the various tools. There is often a need to modify the structure or format of the data to fit the varying input requirements of those tools. Table 2 shows the various input formats that tools in the pipeline require.

Traditionally, the way to manage data in the pipeline was to store distinct sets of input and output files for every stage. For example, the first stage of the pipeline usually involves using Trimmomatic to filter poor quality reads and to trim all the reads to a certain length. Doing this requires the user to

Tool	Expected input format
Trimmomatic	FASTQ/Gzipped FASTQ
Bowtie2	FASTQ/FASTA
QIIME	454-FASTA/454-Quality Scores

Table 2: Expected input formats for various tools

provide 2 paired-end FASTQ sequence files to Trimmomatic, which will then produce 5 output files: unpaired and paired sequences 1 and 2, and a trimming log. The output files have sizes similar to the input, which results in the space usage nearly doubling after the first stage in the pipeline. Subsequent stages will then take the output of the first stage and create more output. Thus, both the number of files and their file size will increase very quickly due to the number of stages within the 16S pipeline. Additionally, certain stages in a pipeline only exist to convert sequences from one format to another (i.e. the need to convert Illumina FASTQ to 454-FASTA and 454-Quality formats required by QIIME), increasing the length of the pipeline and the number of intermediate work products.

Using a plug-in architecture, PIPPING can "stream" sequence data to tools in the pipeline without writing to disk. PIPPING uses UNIX Named Pipes to feed data to external programs such as Trimmomatic and Bowtie2 while simultaneously consuming the output they produce back into the database. UNIX Named Pipes allows programs to transfer data through memory instead of disk, and therefore do not require reading and writing to disk. However, named pipes appear as size zero, regular files on a filesystem, thus tools

can read and write from those files without needing any changes to the software. In the case of Trimmomatic, PIPPING provides a named pipe to capture the contents of the trimming log, and uses the details within it (sequence number, position of trim, number of characters trimmed) and stores them in a "trimmed" table. Since this table only stores the sequences which were operated on, instead of entire copies of the original sequences, the size increase of the PIPPING database after the first stage is almost negligible. Additionally, instead of needing to manage multiple output files and their naming, the data is now ready for the next stage of the pipeline.

3 Results

PIPPING was benchmarked on a 2.7 GHz Intel Core i7 processor with 16GB of RAM, running OS X 10.9. Table 3 shows the time taken for the specified number of sequence inserts into a new PIPPING database. Figure 3 plots the number of actual raw inserts per second into SQLite, showing consistent insertion rates between 96,000 and 102,000 raw inserts per second.

Number of sequences (paired-end)	Time taken (seconds)
500,000	10.35
1,000,000	19.57
2,000,000	39.22
4,000,000	78.67
8,000,000	160.84
16,000,000	319.38
32,000,000	644.90
64,000,000	1,286.26

Table 3: Insertion speeds into SQLite using Pip

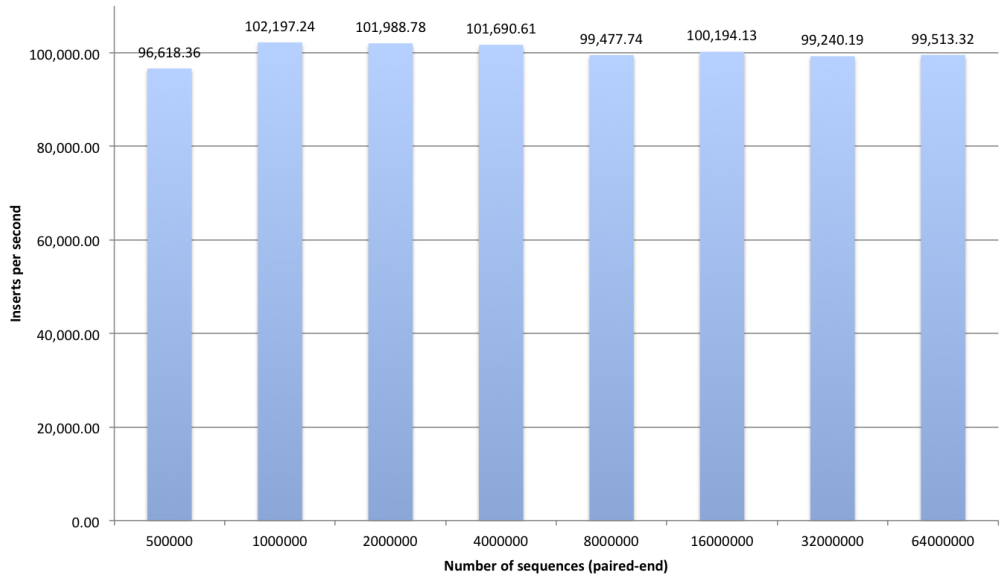

Figure 3: Inserts per second into SQLite using PIPPING

Table 4 shows the size of the raw FASTQ sequence file compared to the resulting database file after inserts. Figure 4 shows that PIPPING databases are consistently 55% of the size of the original FASTQ input.

Table 5 shows runtime reductions in a stage in the test pipeline, as a

Number of sequences (paired-end)	Input FASTQ size (MB)	PIPPING database (MB)
500,000	371.40	206.90
1,000,000	743.00	413.90
2,000,000	1,486.00	828.00
4,000,000	3,051.52	1,699.84
8,000,000	6,082.56	3,389.44
16,000,000	12,165.12	6,789.12
32,000,000	24,350.72	13,568.00
64,000,000	48,701.44	27,146.20
91,000,000	69,754.88	35,061.76

Table 4: Comparison of input file sizes against PIPPING database sizes

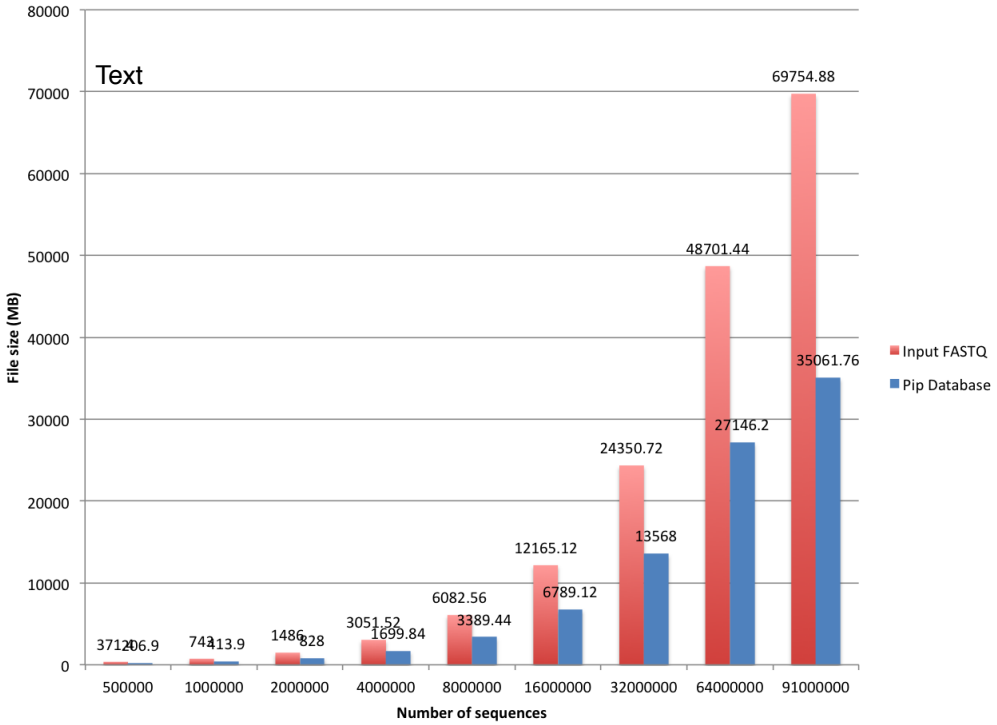

Figure 4: File sizes of PIPPING database against original FASTQ files

result of streaming data directly from PIPPING to Trimmomatic, instead of accessing the disk.

Pipeline Stage	Raw time (seconds)	PIPPING time (seconds)
Trimmomatic	46.43	22.89

Table 5: Time needed to process 500,000 sequences through 16S pipeline stages with and without PIPPING

4 Conclusion

The use of PIPPING within the 16S pipeline shows promise in reducing the logistical effort of researcher and interoperability headaches of multiple tools. PIPPING currently operates on the preprocessing stages of a 16S pipeline and can be extended to later stages in the pipeline easily through a plug-in architecture. The number of intermediate work products in a pipeline has been reduced to a single PIPPING database file, and there are measurable reductions in pipeline run times. There appears to be negligible overhead in streaming data compared to transferring data over the filesystem using intermediate files, and the underlying SQLite implementation is able to deal with datasets of almost 100 million rows without slowdowns.

However, PIPPING currently does not perform complex queries such as subtable queries in its current pipeline so we do not have data on possible scalability issues. SQLite itself might also be a limiting factor due to its single-access model[2], so it might not work well for distributed or parallel workflow

projects such as PaPy[5] or Parallel-META[8]. Additionally, programs that require multiple passes over input data or utilize memory-mapped I/O would not work with PIPPING due to the First-In-First-Out nature of UNIX Named Pipes.

Future work on PIPPING include extending the streaming adapters to support other pipelines such as UPARSE[7] or QIIME[1]. A fully-developed version of PIPPING can potentially encompass an entire 16S sequencing pipeline, completely eliminating unwanted intermediate work products. As sequencing datasets get bigger, SQLite may fail to scale as well as expected, thus exploration into alternative backend systems such as Parallel HDF5[4] may be beneficial.

References

[1] 454 overview tutorial: de novo otu picking and diversity analyses using 454 data. `http://qiime.org/tutorials/tutorial.html`. Accessed: 2014-06-13.

[2] Appropriate uses for sqlite. `http://www.sqlite.org/whentouse.html`. Accessed: 2014-06-13.

[3] Paired-end sequencing assay. `http://www.illumina.com/technology/next-generation-sequencing/paired-end-sequencing_assay.ilmn`. Accessed: 2014-06-13.

[4] Parallel hdf5. http://www.hdfgroup.org/HDF5/PHDF5/. Accessed: 2014-06-13.

[5] Marcin Cieslik and Cameron Mura. A lightweight, flow-based toolkit for parallel and distributed bioinformatics pipelines. *BMC Bioinformatics*, 12(1):61, 2011.

[6] S. Conery, M. Catchen, and Michael Lynch. Rule-based workflow management for bioinformatics. *The VLDB Journal*, 14(3):318–329, September 2005.

[7] Robert Edgar. Uparse pipeline. http://www.drive5.com/usearch/manual/uparse_pipeline.html. Accessed: 2014-06-13.

[8] Xiaoquan Su, Jian Xu, and Kang Ning. Parallel-meta: efficient metagenomic data analysis based on high-performance computation. *BMC Systems Biology*, 6(Suppl 1):S16, 2012.