Distributed Genome Preprocessing Using Apache Hadoop

Charles W. Kazer[1], Richard Platania[2], Sayan Goswami[2], Arghya Kusum Das[2], Seung-Jong Park[2]

1.Swarthmore College, Swarthmore, PA 19081

2.Center for Computation & Technology at Louisiana State University, Baton Rouge, LA 70803

Background

Genome preprocessing is the practice of analyzing and correcting genome read data, short snippets of genetic code, before attempting to assemble the reads into a genome. Sequencers are error prone and usually include a quality score with each read. Preprocessing is a well studied practice that has been proven to improve genome assembly accuracy and efficiency[1]. Modern genome preprocessing relies on the fact that many of the reads for a particular genome sample usually overlap so that a statistical approach can be applied to determine trustworthy reads.

One popular method of genome assembly involves generating a structure called a De Bruijn graph[2] that relies on *k*-mers, a substring of a read of length *k*. Because of the popularity of the De Bruijn graph approach, many preprocessors analyze reads based on *k*-mers, including the one developed in this project.

Most preprocessing techniques have been designed for running sequentially, on a single machine. This project's goal was to design preprocessing tools that could run on multiple machines in a parallel, distributed fashion, in particular using Hadoop, a distributed computing framework developed by Apache.

```
@SRR067577.10004404/2
CCATAGATGCCAGAATCTATCCCTGCCCTCGGCGTGAGACCTCTGCTGGGGAACTGGTGTTTGCCGTATCAGACCACAGCAAGGTGTCGTAATGGAGTCTC
+
IIIIIIIIIGIIIGIHHIIGIIIHIHIIGDGEBGBIIIIHIHHGIHBHE<EDIEDDD=DBA@DD@FBEBDDDBEBEDB@FDB>B4?08?;?;>?>B@9D##
@SRR067577.10029246/1
CTGAATCCCTTTTTCCATGGATAGCTTTCTGGTAGACCATAAATGAAAGCATGGCCACAGCATCTTGGCAGCCAGCACAGACCCATGGAACTCTTGGCCAG
+
HHGHGHHHHGHHHDHGHHEB<FBEG<BGGBDDAGFHHHBHHHHGBDD3GDGEBF@HHHBHGDDF<D>DGBDFFEFFBEEHCDF8E38?=>DADA#######
```

Examples of read data in fastq fromat. Each read is four lines long. The lines are as follows: Identifier, Read, Optional Description, Quality Score

Method

Our general approach was to filter reads based on *k*-mer frequency, first counting the number of appearances of each *k*-mer then dropping reads with low *k*-mer frequencies. In practice, this required a four stage pipeline of mapreduce jobs:

1. List all *k*-mers and count their appearances among all reads
2. Map the reads back to their *k*-mers along with their counted appearances
3. Filter out reads with low *k*-mer counts, returning the IDs of trusted reads
4. Apply the filter to the original data

```
CCATAGATGCCAGAATCTATCCCTGCCCTCGGCGTGAGACCTC
CCATAGA
 CATAGAT
  ATAGATG
   TAGATGC
```

Example of splitting up a read into k-mers with k = 7

Optimizations

- A Perl script was used to reformat each read into a single line, so Hadoop could a whole read at a time. This greatly reduced the time taken by the first step.
- After step one, all references to full reads and specific *k*-mers are dropped, only preserving read identifiers and frequencies. This makes adding step four a necessity, however the runtime of step four is negligible compared to the time saved in the shuffle phases of steps two and three because there is less data being sent between different nodes in the cluster.

Results

Tests were run using a cluster of servers with 32 GB of main memory, 8 cores, and 1 TB hard drives, connected by 10 Gb links. We used raw read data representing Human Chromosome 14[5] which is about 8GB in size. The *k*-mer length was set to 31, and reads that contained over 39 *k*-mers which only appeared once were dropped. Two major tests were run, see Figure 1 and Table 1.

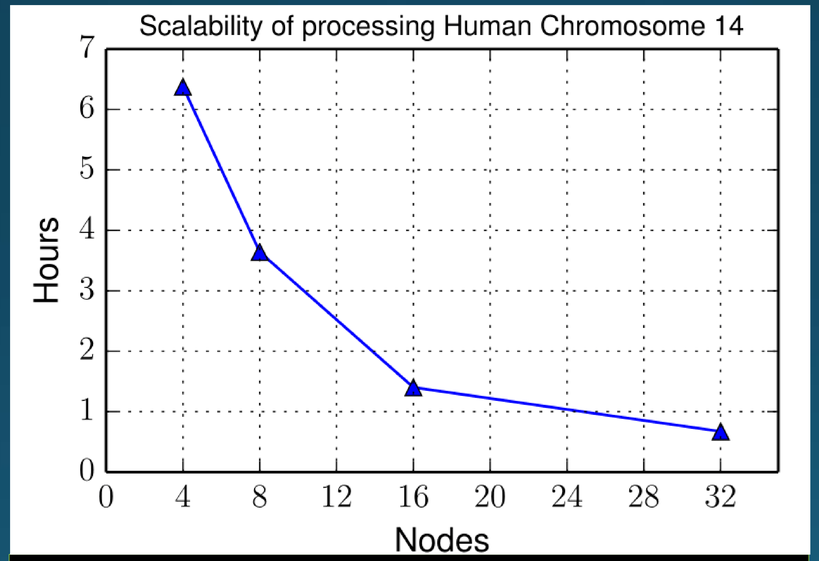

Figure 1: The preprocessor was run using 4, 8, 16, and 32 slave nodes to test scalability as nodes were added.

Assembly of Read Data

	Unfiltered	Filtered
Nodes in Graph	4419219	2792604
N50 Length	249	599
Max Length	6760	9524

Table 1: The Velvet assembler[3] was run on both unfiltered read data and reads filtered by our preprocessor. This is a summary of the major statistics.

Discussion and Further Work

- Speed improved significantly as more nodes were added. The time required to run was nearly cut in half each time the number of nodes used doubled, implying scalability.
- The output of the assembler had significant improvements when working on the filtered data rather than the unfiltered data. The complexity of the graph was greatly reduced, and the N50 length also more than doubled, indicating a better assembly.

Although the trimmer implemented and tested here is naïve, its distributed approach is promising for implementing more advanced preprocessing techniques. As of the completion of this poster, we've already implemented other trimmers:

- Average quality filter
- Sliding window filter
- *q*-mer counter filter (based on *k*-mer count and quality scores)[4]

However these were not thoroughly tested.

We're also looking into preprocessing using Apache Spark, which has a more flexible distributed model than the simplistic and rigid one of Hadoop.

Summary

We built a distributed preprocessing tool to run over Apache Hadoop on a cluster of arbitrary size with commodity hardware. The tool uses a simple *k*-mer counting approach to filter reads. Initial testing show that the tool is both scalable and effective. Each time the number of nodes used doubles, performance almost doubles as well. The filtering improved the accuracy and execution time of the well known genome assembler Velvet. We've already used this distributed approach to build more advanced preprocessing tools, and in-memory distributed frameworks, like Spark, promise to allow for even more complex preprocessing.

References

1. Chen et al.: Software for pre-processing Illumina next generation sequencing short read sequences. Source Code for Biology and Medicine 2014 9:8.
2. Sharafat, Ali. "De Novo Assembly." *Stanford Artificial Intelligence Laboratory*. Web. 29 May 2015.
3. Zerbino, D. R., and E. Birney. "Velvet: Algorithms for De Novo Short Read Assembly Using De Bruijn Graphs." Genome Research 18.5 (2008): 821-29. Web.
4. Kelley, David R., Michael C. Schatz, and Steven L. Salzberg. "Quake: Quality-aware Detection and Correction of Sequencing Errors." Genome Biology 11.11 (2010): n. pag. Web.
5. Salzberg et al. "GAGE: A Critical Evaluation of Genome Assemblies and Assembly Algorithms." *Genome Research* 22.3 (2012): 557-67. Web.

Acknowledgments

This material is based upon work supported by the National Science Foundation under award OCI-1263236 with additional support from the Center for Computation & Technology at Louisiana State University.