240 lines (177 sloc) 10.9 KB Raw Blame History

UTC-0500

UTC-500 team submission for the AI Genomics Hackathon, 23-25 June 2017.

The data for the hackathon include a whole genome sequence data set for NF2: a genetic condition that causes tumors to grow from Schwann cells. The dataset has a tumor/normal pair and a sibling. All three samples were whole genome sequenced at 60-90X, aligned to the hg19 human genome assembly, and made available as BAM files. The dataset was acquired in the context of finding the driver mutations for a particular NF2 patient.

This submission plans to transform the data in VCF and BAM formats into the Big Data Genomics Parquet + Avro format, partitioned for efficient performance on an Apache Spark cluster. Then as time allows, re-align the reads against the most recent human genome assembly (GRCh38), and re-call variants with FreeBayes via Cannoli and Avocado.

Transform VCF and BAM data into Big Data Genomics Parquet + Avro format

Using the ADAM command line (which extends `spark-submit`, provided by Apache Spark) and the ADAM interactive shell (which simarily extends `spark-shell`), the data in VCF and BAM formats were converted to `Genotype` and `AlignmentRecord` Avro records, respectively, and written to disk in Parquet format.

Gene features from Ensembl in GFF3 format were also converted to `Feature` Avro records and written to disk in Parquet format.

Table 1. Relative sizes of data in VCF, BAM, and GFF3 formats compared to Big Data Genomics Parquet + Avro format

Resource	Size (on	Relative	Format

Table 1. Relative sizes of data in VCF, BAM, and GFF3 formats compared to Big Data Genomics Parquet + Avro format

Resource	Size (on S3)	Relative Size	Format
G91716.vcf.gz	589.2 MB	100%	GZIP VCF
G91716.genotypes.adam	602.5 MB	102%	`Genotype` Avro records in Parquet
G97552.vcf.gz	544.4 MB	100%	GZIP VCF
G97552.genotypes.adam	564.1 MB	103%	`Genotype` Avro records in Parquet
GSN79Tumor_normal.vcf.gz	7.4 MB	100%	GZIP VCF
GSN79Tumor_normal.genotypes.adam	9.3 MB	125%	`Genotype` Avro records in Parquet
NF2_XY_s.bam	253.2 GB	100%	BAM
NF2_XY_s.alignments.adam	193.8 GB	76%	`AlignmentRecord` Avro records in Parquet
OF_010116NF2_a.bam	207.7 GB	100%	BAM
OF_010116NF2_a.alignments.adam	158.6 GB	76%	`AlignmentRecord` Avro records in Parquet
OF_112015SJIA_2.bam	207.2 GB	100%	BAM
OF_112015SJIA_2.alignments.adam	159.7 GB	77%	`AlignmentRecord` Avro records in Parquet
Homo_sapiens.GRCh38.89.chr.gff3.gz	35.5 MB	100%	GZIP GFF3
Homo_sapiens.GRCh38.89.chr.features.adam	29.9 MB	84%	`Feature` Avro records in Parquet

Methods

```
import org.bdgenomics.adam.rdd.ADAMContext._
sc.loadGenotypes("G91716.vcf.gz").sort().saveAsParquet("G91716.genotypes.adam")
sc.loadGenotypes("G97552.vcf.gz").sort().saveAsParquet("G97552.genotypes.adam")
```

`htsjdk` complained about the format of some missing attributes in GSN79Tumor_normal.vcf

```
$ cat GSN79Tumor_normal.vcf | \
    sed -e 's/MAX_ED=.;//g' | \
    sed -e 's/MIN_ED=.//g' > GSN79Tumor_normal.edit.vcf
```

```
import org.bdgenomics.adam.rdd.ADAMContext._
sc.loadGenotypes("GSN79Tumor_normal.edit.vcf").sort().saveAsParquet("GSN79Tumor_normal.genotypes.adam")
```

Apache Spark configuration parameters (such as `--num-exectors`, `--executor-memory`) have been left out for clarity.

```
$ adam-submit transformAlignments NF2_XY_s.bam NF2_XY_s.alignments.adam
$ adam-submit transformAlignments OF_010116NF2_a.bam OF_010116NF2_a.alignments.adam
$ adam-submit transformAlignments OF_112015SJIA_2.bam OF_112015SJIA_2.alignments.adam
```

```
import org.bdgenomics.adam.rdd.ADAMContext._

val features = sc.loadFeatures("Homo_sapiens.GRCh38.89.chr.gff3")
features.saveAsParquet("Homo_sapiens.GRCh38.89.chr.features.adam")

val geneFeatures = features.transform(_.filter(f => f.featureType == "gene"))
geneFeatures.saveAsParquet("Homo_sapiens.GRCh38.89.chr.geneFeatures.adam")
```

Conductor, also based on Apache Spark, was used for efficient, distributed transfer of data between S3 and HDFS.

Apache Parquet is an compressed, efficient columnar data storage format. Columns are compressed using standard columnar techniques (e.g. RLE, dictionary encoding). Not only are genomic data in Parquet format often smaller on

Conductor, also based on Apache Spark, was used for efficient, distributed transfer of data between S3 and HDFS.

Apache Parquet is an compressed, efficient columnar data storage format. Columns are compressed using standard columnar techniques (e.g. RLE, dictionary encoding). Not only are genomic data in Parquet format often smaller on disk than compressed native formats (e.g. BAM or CRAM), they are partitioned for efficient distributed I/O across an Apache Spark cluster.

For example, the following took 378 seconds on a BAM file

```
val alignments = sc.loadAlignments("NF2_XY_s.bam")
alignments.rdd.count()
```

```
res0: Long = 1721483180
```

whereas the same took only 93 seconds on `AlignmentRecord` Avro records in Parquet

```
val alignments = sc.loadAlignments("NF2_XY_s.alignments.adam")
alignments.rdd.count()
```

```
res2: Long = 1721483180
```

Re-align reads to GRCh38 with BWA

In order to re-align reads to the GRCh38 homan genome assembly, the alignment records were transformed to unaligned fragments and written as paired FASTQ format. These fragments were also written out as `Fragment` Avro records in Parquet format.

Re-align reads to GRCh38 with BWA

In order to re-align reads to the GRCh38 homan genome assembly, the alignment records were transformed to unaligned fragments and written as paired FASTQ format. These fragments were also written out as `Fragment` Avro records in Parquet format.

Table 2. Relative sizes of unaligned reads in FASTQ format compared to Big Data Genomics Parquet + Avro format

Resource	Size (on S3)	Relative Size	Format
NF2_XY_s_1.fq.gz, NF2_XY_s_2.fq.gz		100%	Paired GZIP FASTQ files
NF2_XY_s.unaligned.fragments.adam	138.6 GB		`Fragment` Avro records in Parquet
OF_010116NF2_a_1.fq.gz, OF_010116NF2_a_2.fq.gz		100%	Paired GZIP FASTQ files
OF_010116NF2_a.unaligned.fragments.adam	115.9 GB		`Fragment` Avro records in Parquet
OF_112015SJIA_2_1.fq.gz, OF_112015SJIA_2_2.fq.gz		100%	Paired GZIP FASTQ files
OF_112015SJIA_2.unaligned.fragments.adam	124.3 GB		`Fragment` Avro records in Parquet

Methods

```
import org.bdgenomics.adam.rdd.ADAMContext
```

Methods

```
import org.bdgenomics.adam.rdd.ADAMContext._

val alignments = sc.loadAlignments("NF2_XY_s.alignments.adam")
alignments.saveAsPairedFastq("NF2_XY_s_1.fq", "NF2_XY_s_2.fq", outputOriginalBaseQualities = true)

val unalignedReads = sc.loadAlignments(pathName = "NF2_XY_s_1.fq", optPathName2 = Some("NF2_XY_s_2.fq"))
unalignedReads.toFragments.saveAsParquet("NF2_XY_s.unaligned.fragments.adam")
```

And similar for the other samples.

The next step would be to use the Cannoli library, which wraps the Apache Spark pipe API to stream data already partitioned across the executor nodes in a cluster to and from external bioinformatics applications. A wrapper for running BWA via Docker is provided.

Again, Apache Spark configuration parameters have been left out for clarity.

```
#!/bin/bash

set -e -x -v

export SAMPLE=NF2_XY_s
export ADAM_MAIN=org.bdgenomics.cannoli.Cannoli

adam-submit \
    --jars target/cannoli_2.11-spark2-0.1-SNAPSHOT.jar \
    -- \
    bwa \
    ${SAMPLE}.unaligned.fragments.adam \
    ${SAMPLE}.bwa.hg38.alignments.adam \
    ${SAMPLE} \
    -index hg38.fa \
    -sequence_dictionary hg38.dict \
    -fragments \
    -use_docker \
    -docker_image heuermh/bwa \
    -add_indices
```

Results TBD.

RDD.

Interactive genomic region joins in ADAM shell

ADAM supports efficient genomic region joins between various genomic data types, supporting analysis on interactive time scales.

For example, in ADAM shell

```
import org.bdgenomics.adam.rdd.ADAMContext._

val geneFeatures = sc.loadFeatures("Homo_sapiens.GRCh38.89.chr.geneFeatures.adam")
val genotypes = sc.loadGenotypes("G91716.genotypes.adam")
val variants = genotypes.toVariantContextRDD.toVariantRDD
val variantsByGeneFeatures = geneFeatures.shuffleRegionJoinAndGroupByLeft(variants)
val variantCountsByGeneFeatures = variantsByGeneFeatures.rdd.map(f => (f._1.getName, f._2.size))

variantCountsByGeneFeatures.first
```

```
(String, Int) = (DDX11L1,12)
```

After writing out to a text file and sed/awk/sort, the top few variant counts by gene feature

```
$ head G91716.variantCountsByGeneFeature.sorted.txt
13698 CSMD1
9204 RBFOX1
8397 PTPRD
7451 CNTNAP2
6601 LRP1B
5796 CDH13
5689 FHIT
5659 PCDH15
5588 WWOX
5435 MACROD2
```

Brought to you by