bdgenomics.mango Documentation

Release 0.0.5

Big Data Genomics

Jun 29, 2021

Installation

CHAPTER 1

Introduction

Mango is a distributed visualization tool that enables visualization of genomic data on top of Apache Spark.

1.1 The Mango/Big Data Genomics Ecosystem

Mango builds upon the open source Apache Spark, Apache Avro, and Apache Parquet projects. Additionally, Mango can be deployed for both interactive and production workflows using a variety of platforms.

1.1.1 Building Mango from Source

You will need to have Java 8, Apache Maven version 3.1.1 or later, and npm version 4.0.0 or later installed in order to build Mango.

> **Note:** The default configuration is for Hadoop 2.7.3. If building against a different version of Hadoop, please pass `-Dhadoop.version=<HADOOP_VERSION>` to the Maven command. Mango will cross-build for both Spark 1.x and 2.x, but builds by default against Spark 1.6.3. To build for Spark 2, run the `./scripts/move_to_spark2.sh` script.

```
git clone https://github.com/bigdatagenomics/mango.git
cd mango
export MAVEN_OPTS="-Xmx512m -XX:MaxPermSize=128m"
mvn clean package -DskipTests
```

Outputs

```
...
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 04:30 min
[INFO] Finished at: 2017-12-11T10:35:57-08:00
[INFO] Final Memory: 61M/1655M
[INFO] ------------------------------------------------------------------------
```

Running Mango

Mango is packaged as an überjar and includes all necessary dependencies, except for Apache Hadoop and Apache Spark.

Building for Python

To build and test *Mango's Python bindings*, first set environmental variables pointing to the root of your Mango and Spark installation directories.

```
export SPARK_HOME=<PATH_TO_SPARK>
export MANGO_HOME=<PATH_TO_MANGO>
```

Next, build Mango jars without running tests, by running the following command from the root of the Mango repo install directory:

```
mvn clean package -DskipTests
```

Additionally, the PySpark dependencies must be on the Python module load path and the Mango JARs must be built and provided to PySpark. This can be done with the following bash commands:

```
PY4J_ZIP="$(ls -1 "${SPARK_HOME}/python/lib" | grep py4j)"
export PYTHONPATH=${SPARK_HOME}/python:${SPARK_HOME}/python/lib/${PY4J_ZIP}:$
↪{PYTHONPATH}
ASSEMBLY_DIR="${MANGO_HOME}/mango-assembly/target"
ASSEMBLY_JAR="$(ls -1 "$ASSEMBLY_DIR" | grep "^mango-assembly[0-9A-Za-z\_\.-]*\.jar$"␣
↪| grep -v javadoc | grep -v sources || true)"
export PYSPARK_SUBMIT_ARGS="--jars ${ASSEMBLY_DIR}/${ASSEMBLY_JAR} --driver-class-
↪path ${ASSEMBLY_DIR}/${ASSEMBLY_JAR} pyspark-shell"
```

Next, install dependencies using the following commands:

```
cd mango-python
make prepare
cd ..
cd mango-pileup
make prepare
cd ..
```

Finally, run `mvn package` again, this time enabling the `python` profile as well as tests:

```
mvn package -P python
```

This will enable the `mango-python` and `mango-pileup` module as part of the Mango build. This module uses Maven to invoke a Makefile that builds a Python egg and runs tests.

1.1.2 Installing Mango from Conda

Install Conda

Installing mango through conda requires conda to be installed. First, make sure you have installed conda.

Set up channels

After installing conda you will need to add other channels mango depends on. It is important to add them in this order so that the priority is set correctly.

```
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

Install Mango in a conda environment

To manage dependencies amongst other projects, you can install mango in a conda environment. First, create a new conda environment:

```
conda create -y -q -n MyEnv python=3.6 pip
```

Then install mango to the created environment:

```
conda install --name MyEnv mango
```

Then activate your environment:

```
conda activate MyEnv
```

Note: If not installing Mango into a conda environment, Mango requires SPARK_HOME environment variable to be set. If you are installing Mango into a conda environment and the SPARK_HOME environment variable is not set, activating the conda environment will automatically set SPARK_HOME to the pyspark site-packages path.

1.1.3 Running Mango from Distribution

Fetching Mango Distribution

Mango is packaged as an überjar and includes all necessary dependencies, except for Apache Hadoop and Apache Spark.

To fetch the Mango distribution, run:

```
VERSION=0.0.3

wget -O mango-distribution-${VERSION}-bin.tar.gz https://search.maven.org/
↪remotecontent?filepath=org/bdgenomics/mango/mango-distribution/${VERSION}/mango-
↪distribution-${VERSION}-bin.tar.gz
tar xzvf mango-distribution-${VERSION}-bin.tar.gz
```

From the distribution directory, you can run Mango notebook or Mango browser:

First, make sure your SPARK_HOME env variable is set:

```
export SPARK_HOME=<PATH_TO_SPARK>
```

Then run Mango notebook or Mango browser:

```
cd mango-distribution-${VERSION}
./bin/mango-notebook
./bin/mango-submit
```

Installing python modules

To run Mango in a python notebook, install bdgenomics.mango.pileup, a Jupyter Widget:

```
pip install bdgenomics.mango.pileup
jupyter nbextension enable --py --sys-prefix bdgenomics.mango.pileup  # can be␣
↪skipped for notebook version 5.3 and above
```

And bdgenomics.mango:

```
pip install bdgenomics.mango
```

1.1.4 Jupyter Widget Usage

Note: Python 2.7 is dropping support January 1, 2020. For this reason, Mango no longer supports Python 2.

The Mango widgets are Jupyter widgets built using pileup.js. The widgets support visualizations for alignments, features, variants, and genotypes in a Jupyter Notebook and Jupyter lab version >2.0.

Installation for Jupyter notebook

First, install and enable bdgenomics.mango.pileup, a Jupyter Widget:

```
pip install bdgenomics.mango.pileup

jupyter nbextension install --py --user bdgenomics.mango.pileup
jupyter nbextension install --py --user widgetsnbextension

jupyter nbextension enable --py --user widgetsnbextension
jupyter nbextension enable --py --user bdgenomics.mango.pileup
```

Note: If you are using an conda environment, install extensions using `--sys-prefix`:

```
jupyter nbextension install --py --sys-prefix bdgenomics.mango.pileup
jupyter nbextension install --py --sys-prefix widgetsnbextension

jupyter nbextension enable --py --sys-prefix widgetsnbextension
jupyter nbextension enable --py --sys-prefix bdgenomics.mango.pileup
```

This will install the bdgenomics.mango.pileup extension into your current conda environment.

Installation for Jupyter lab

To use the Mango widgets in Jupyter lab, you will need the following requirements:

- Jupyter lab version > 2.0
- node > 12

```
pip install bdgenomics.mango.pileup
jupyter labextension install @jupyter-widgets/jupyterlab-manager # install the␣
↪Jupyter widgets extension
jupyter labextension install bdgenomics.mango.pileup
```

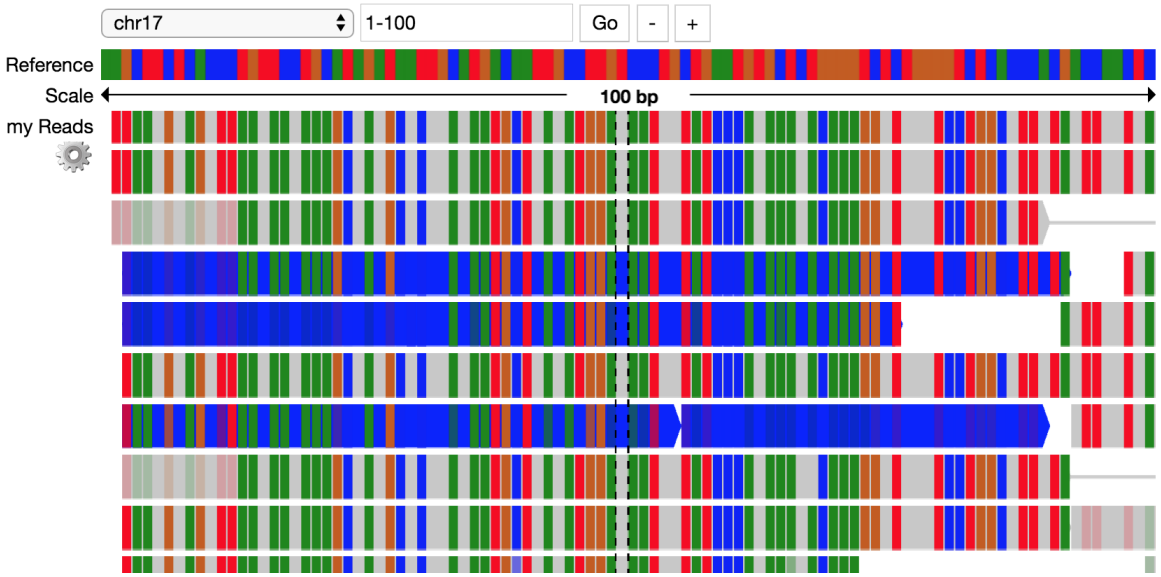
These tutorials show how to create a Jupyter pileup.js widget. An example notebook can be found in the Mango Github repository.

Pileup Example

This example shows how to visualize alignments through a Jupyter widget.

```
# imports
import bdgenomics.mango.pileup as pileup
from bdgenomics.mango.pileup.track import *
import pandas as pd
```

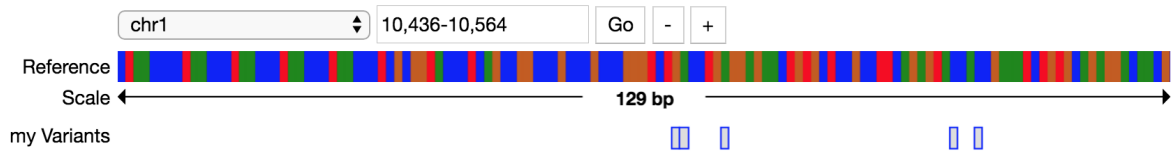
```
# read in JSON
readsJson = pd.read_json("./data/alignments.ga4gh.chr17.1-250.json")
GA4GHAlignmentJson = readsJson.to_json()

# make pileup track
tracks=[Track(viz="pileup", label="my Reads", source=pileup.sources.
↪GA4GHAlignmentJson(GA4GHAlignmentJson))]

# render tracks in widget
reads = pileup.PileupViewer(chrom="chr17", start=1, stop=100, reference="hg19",␣
↪tracks=tracks)
reads
```



Variant Example

This example shows how to visualize variants through a Jupyter widget.

```
# make variant track
tracks=[Track(viz="variants", label="my Variants", source=pileup.sources.
↪VcfDataSource("<path_to_file>/my_vcf.vcf"))]

# render tracks in widget
variants = pileup.PileupViewer(chrom="chr1", start=10436, stop=10564, reference="hg19
↪", tracks=tracks)
variants
```



Feature Example

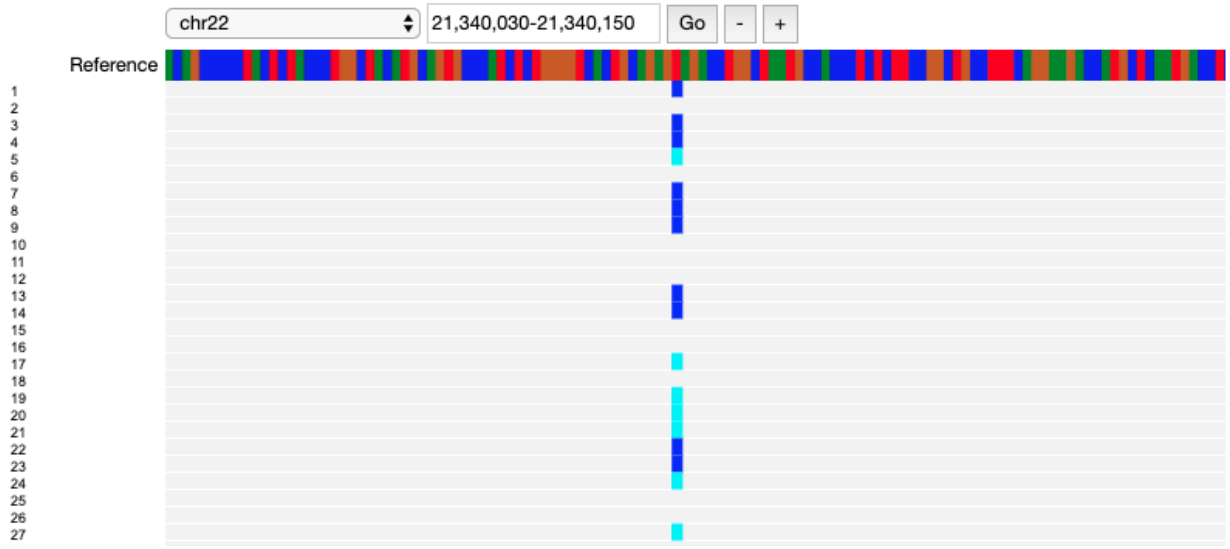
This example shows how to visualize features through a Jupyter widget.

```
featuresJson = pd.read_json("./data/features.ga4gh.chr1.120000-125000.json")
GA4GHFeatureJson = featuresJson.to_json()

# make feature track
tracks=[Track(viz="features", label="my Features", source=pileup.sources.
↪GA4GHFeatureJson(GA4GHFeatureJson))]

# render tracks in widget
features = pileup.PileupViewer(chrom="chr1", start=120000, stop=121000, reference=
↪"hg19", tracks=tracks)
features
```

Genotype Example

This example shows how to visualize genotypes through a Jupyter widget.

```
# make genotype track
tracks=[Track(viz="genotypes", label="my Genotypes", source=pileup.sources.
↪VcfDataSource("<path_to_file>/my_vcf.vcf"))]

# render tracks in widget
genotypes = pileup.PileupViewer(chrom="chr22", start=21340030, stop=21340150,␣
↪reference="hg19", tracks=tracks)
genotypes
```



1.1.5 Mango Browser Examples

Mango browser is an HTML based genome browser that runs on local, remote, and cloud staged files. The Mango Browser utilizes Apache Spark and scalatra.

See file support for file types that are supported in the Mango Browser.

Mango Browser Options

The Mango browser uses the mango-submit script to start an Apache Spark session and launch the Mango Browser. The mango-submit script can be found in the Mango Distribution or in the Mango Github repository.

To see options that can be run with the Mango Submit script, run:

```
./mango-submit -h
```

You will see a list of options:

```
genome                                                          : Path to compressed .
↪genome file. To build a new genome file, run bin/make_genome.
-cacheSize N                                                    : Bp to cache on␣
↪driver.
-coverage VAL                                                   : A list of coverage␣
↪files to view, separated by commas (,)
-debugFrontend                                                  : For debugging␣
↪purposes. Sets front end in source code to avoid recompilation.
-discover                                                       : This turns on␣
↪discovery mode on start up.
-features VAL                                                   : The feature files␣
↪to view, separated by commas (,)
-h (-help, --help, -?)                                          : Print help
-parquetIsBinned                                                : This turns on␣
↪binned parquet pre-fetch warmup step
-parquet_block_size N                                           : Parquet block size␣
↪(default = 128mb)
-parquet_compression_codec [UNCOMPRESSED | SNAPPY | GZIP | LZO] : Parquet compression␣
↪codec
```

(continues on next page)

(continued from previous page)

```
-parquet_disable_dictionary                                    : Disable dictionary␣
↪encoding
-parquet_logging_level VAL                                     : Parquet logging␣
↪level (default = severe)
-parquet_page_size N                                           : Parquet page size␣
↪(default = 1mb)
-port N                                                        : The port to bind to␣
↪for visualization. The default is 8080.
-prefetchSize N                                                : Bp to prefetch in␣
↪executors.
-preload VAL                                                   : Chromosomes to␣
↪prefetch, separated by commas (,).
-print_metrics                                                 : Print metrics to␣
↪the log on completion
-reads VAL                                                     : A list of reads␣
↪files to view, separated by commas (,)
-repartition                                                   : Repartitions data␣
↪to default number of partitions.
-test                                                          : For debugging␣
↪purposes.
-variants VAL                                                  : A list of variants␣
↪files to view, separated by commas (,). Vcf files require a
                                                                 corresponding tbi␣
↪index.
```

Note that a genome file is always required when running the Mango Browser. See how to build a genome.

Running Mango Browser Examples Locally

The Mango Github repository contains example scripts and data files for running Mango browser on region chr17:7500000-7515000 of a single aligment sample. Once Mango is built, you can run the following command from the root mango directory to view Mango browser:

```
./example-files/browser-scripts/run-example.sh
```

This script contains the following command:

```
bin/mango-submit ./example-files/hg19.genome \
  -reads ./example-files/chr17.7500000-7515000.sam \
  -variants ./example-files/ALL.chr17.7500000-7515000.phase3_shapeit2_mvncall_
↪integrated_v5a.20130502.genotypes.vcf
```

This script specifies the required genome reference file:

```
./example-files/hg19.genome
```

An optional alignment file:

```
-reads ./example-files/chr17.7500000-7515000.sam
```

An optional variant file:

```
-variants ./example-files/ALL.chr17.7500000-7515000.phase3_shapeit2_mvncall_
↪integrated_v5a.20130502.genotypes.vcf
```

Once the example script is running, navigate to localhost:8080 to view the Mango browser. Navigate to `chr17:7500000-7515000` to view data.

There is also another example script at `./example-files/browser-scripts/run-http-example.sh` that runs on remote files from Amazon S3.

This script contains the following command:

```
bin/mango-submit ./example-files/hg19.genome \
  -variants http://s3.amazonaws.com/1000genomes/phase1/analysis_results/integrated_
↪call_sets/ALL.chr1.integrated_phase1_v3.20101123.snps_indels_svs.genotypes.vcf.gz \
  -reads http://s3.amazonaws.com/1000genomes/phase1/data/NA19661/exome_alignment/
↪NA19661.mapped.illumina.mosaik.MXL.exome.20110411.bam
```

Running Mango Browser with Parameters

Mango can accept Apache Spark parameters, as well as Mango parameters shown above.

To run Mango browser with user specified Apache Spark parameters, run

```
./bin/mango-submit <Spark-parameters> -- <Mango-parameters>
```

`<Spark-parameters>` include Apache Spark specific configuration settings.

`<Mango-parameters>` are shown in the output of `./bin/mango-submit`.

Note that a genome file is required to run the Mango browser.

Running example files on a cluster with HDFS

The Mango browser can run files that are staged on Hadoop Distributed File System (HDFS).

To run the example files on a cluster with hdfs, first put example-files on hdfs:

```
hdfs dfs -put example-files
```

Then, run mango-submit:

```
./bin/mango-submit ./example-files/hg19.genome \
       -genes http://www.biodalliance.org/datasets/ensGene.bb \
       -reads hdfs:///<path_to_examples>/example-files/chr17.7500000-7515000.sam \
       -variants hdfs:///<path_to_examples>/example-files/ALL.chr17.7500000-7515000.
↪phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf \
```

Running on Apache YARN

YARN is a resource management system for clusters. The Mango browser can run on YARN clusters, and requires package `org.apache.parquet:parquet-avro:1.8.3`. To run the Mango browser on YARN, include parquet-avro as a package on start-up:

```
./bin/mango-submit --packages org.apache.parquet:parquet-avro:1.8.3 \
        --master yarn-client \
        <mango-parameters>
```

1.1.6 Building a Genome

The Mango browser requires a remote TwoBit reference file, a local gene file, and a local chromosome size file to run.

These files are downloaded from UCSC Genome Browser downloads.

Assembling a Default Genome File

To assemble a genome file, run:

```
./bin/make_genome <GENOME_NAME> <OUTPUT_LOCATION>
```

This will download required files for the genome build, compress them, and save them to <OUTPUT_LOCATION>. All builds accessible from the UCSC Downloads page are supported.

The file `hg19.genome` build exists in the `example-files/` folder as a reference.

Assembling a Custom Genome File

If you need to assemble a custom genome file that is not supported by the `make_genome` executable, you can assemble one as follows.

First specify a folder, <YOUR_GENOME>.genome, and include the following files:

- cytoBand.txt: UCSC formatted cytoBand file
- <YOUR_GENOME>.chrom.sizes: tab delimited file of chromosome names and sizes
- refGene.txt: UCSC formatted tab delimited text file of gene information
- properties.txt: File containing metainformation for your genome

The `properties.txt` file should be formatted as follows:

```
sequenceLocation=http://<path_to_remote_2bit_file>.2bit
refGene=refGene.txt
chromSizes=<YOUR_GENOME>.chrom.sizes
cytoband=cytoBand.txt
```

Note that the sequenceLocation parameter must link to a remote TwoBit file.

1.1.7 Mango Python Examples

Note: Python 2.7 is dropping support January 1, 2020. For this reason, Mango no longer supports Python 2.

Running Mango Notebook Locally

Once Mango and Mango python is built, you can run the following command to view Mango notebook.

```
./bin/mango-notebook
```

Mango notebook depends on Jupyter notebook. To install all dependencies for Mango notebook in a virtual environment, see installation instructions.

One Mango notebook is running, you can view local results at localhost:<port>, where <port> is the open port assigned by Jupyter notebook. There are three notebooks that can be viewed as examples in the Mango repository:

- example-files/notebooks/mango-python-alignment.ipynb
- example-files/notebooks/mango-python-coverage.ipynb
- example-files/notebooks/mango-python-variants.ipynb
- example-files/notebooks/mango-pileup.ipynb

Running the Mango Notebook with Parameters

The Mango Notebook can be run with Apache Spark parameters and Jupyter notebook parameters. To run Mango notebook with user specified parameters, run

```
./bin/mango-notebook <Spark-parameters> -- <Jupyter-notebook-parameters>
```

Running the Mango Notebook on YARN

YARN is a resource management system for clusters. The Mango notebook can run on YARN clusters, and requires jars for org.apache.parquet:parquet-hadoop:1.8.3. To run the Mango browser on YARN, download parquet-hadoop jar:

Then include the jar in spark.driver.extraClassPath:

```
wget http://central.maven.org/maven2/org/apache/parquet/parquet-hadoop/1.8.3/parquet-
↪hadoop-1.8.3.jar
```

```
./bin/mango-notebook --master yarn \
  --conf spark.driver.extraClassPath=<path_to_jar>/parquet-hadoop-1.8.3.jar \
   -- <Jupyter-notebook-parameters>
```

1.1.8 Mango Pyspark API Documentation

bdgenomics.mango Package

bdgenomics.mango provides hooks for visualizing genomic data on top of Apache Spark. Available visualizations are supported for alignments, coverage, variants and features.

Alignments

AlignmentSummary(spark, ac, alignmentDataset)	AlignmentSummary class.
FragmentDistribution(ss, alignmentDataset[, ...])	FragmentDistribution class.
MapQDistribution(ss, alignmentDataset[, sample])	MapQDistribution class.
IndelDistribution(ss, alignmentDataset[, ...])	IndelDistribution class.

bdgenomics.mango.alignments.AlignmentSummary

class bdgenomics.mango.alignments.**AlignmentSummary**(*spark*, *ac*, *alignmentDataset*, *sample=1.0*)

AlignmentSummary class. AlignmentSummary provides scrollable visualization of alignments based on ge-

nomic regions.

__init__(*spark*, *ac*, *alignmentDataset*, *sample=1.0*)
Initializes an AlignmentSummary class.

Args:

param spark SparkSession

param ac ADAMContext

param alignmentDataset bdgenomics.adam.Dataset.AlignmentRecoDatasetataset

param sample fraction of reads to sample from

Methods

__init__(spark, ac, alignmentDataset[, sample])	Initializes an AlignmentSummary class.
getCoverageDistribution([bin_size])	Computes coverage distribution for this AlignmentDataset.
getFragmentDistribution()	Computes fragment distribution for this AlignmentDataset.
getIndelDistribution([bin_size])	Computes insertion and deletion distribution for this AlignmentDataset
getMapQDistribution()	Computes mapping quality distribution for this AlignmentDataset.
viewPileup(contig, start, end[, reference, ...])	Visualizes a portion of this AlignmentDataset in a scrollable pileup widget

bdgenomics.mango.alignments.FragmentDistribution

class bdgenomics.mango.alignments.**FragmentDistribution**(*ss*, *alignmentDataset*, *sample=1.0*)
FragmentDistribution class. Plotting functionality for visualizing fragment distributions of multi-sample cohorts.

__init__(*ss*, *alignmentDataset*, *sample=1.0*)
Initializes a FragmentDistribution class. Computes the fragment distribution of a AlignmentDataset. This Dataset can have data for multiple samples.

Args:

param ss global SparkSession.

param alignmentDataset bdgenomics.adam.Dataset.AlignmentDataset

param sample Fraction to sample AlignmentDataset. Should be between 0 and 1

Methods

__init__(ss, alignmentDataset[, sample])	Initializes a FragmentDistribution class.
plotDistributions([normalize, cumulative, ...])	Plots final distribution values and returns the plotted distribution as a Counter object.

Attributes

pre_sampled
rdd
sample
seed
ss

bdgenomics.mango.alignments.MapQDistribution

class bdgenomics.mango.alignments.**MapQDistribution**(*ss*, *alignmentDataset*, *sample=1.0*)

MapQDistribution class. Plotting functionality for visualizing mapping quality distributions of multi-sample cohorts.

__init__(*ss*, *alignmentDataset*, *sample=1.0*)

Initializes a MapQDistribution class. Computes the mapping quality distribution of an AlignmentDataset. This Dataset can have data for multiple samples.

Args:

param ss global SparkSession.

param alignmentDataset A bdgenomics.adam.dataset.AlignmentDataset object.

param sample Fraction to sample AlignmentDataset. Should be between 0 and 1

Methods

__init__(ss, alignmentDataset[, sample])	Initializes a MapQDistribution class.
plotDistributions([normalize, cumulative, ...])	Plots final distribution values and returns the plotted distribution as a Counter object.

Attributes

pre_sampled
rdd
sample
seed
ss

bdgenomics.mango.alignments.IndelDistribution

class bdgenomics.mango.alignments.**IndelDistribution**(*ss*, *alignmentDataset*, *sample=1.0*, *bin_size=10000000*)

IndelDistribution class. IndelDistribution calculates indel distributions on an AlignmentDataset.

__init__(*ss*, *alignmentDataset*, *sample=1.0*, *bin_size=10000000*)

Initializes a IndelDistribution class. Computes the insertiona and deletion distribution of alignmentDataset.

Args:

param SparkSession the global SparkSession

param alignmentDataset A bdgenomics.adam.dataset.AlignmentDataset object

param bin_size Division size per bin

Methods

__init__(ss, alignmentDataset[, sample, ...])	Initializes a IndelDistribution class.
plot([testMode, plotType])	Plots final distribution values and returns the plotted distribution as a counter object.

Coverage

CoverageDistribution(ss, coverageDataset[, ...])	CoverageDistribution class.

bdgenomics.mango.coverage.CoverageDistribution

class bdgenomics.mango.coverage.**CoverageDistribution**(*ss*, *coverageDataset*, *sample=1.0*, *bin_size=10*, *pre_sampled=False*)

CoverageDistribution class. Plotting functionality for visualizing coverage distributions of multi-sample cohorts.

__init__(*ss*, *coverageDataset*, *sample=1.0*, *bin_size=10*, *pre_sampled=False*)

Initializes a CoverageDistribution class. Computes the coverage distribution of a CoverageRDD. This RDD can have data for multiple samples.

Args:

param ss global SparkSession.

param coverageRDD bdgenomics.adam.ds.CoverageDataset

param sample Fraction to sample CoverageRDD. Should be between 0 and 1

Methods

__init__(ss, coverageDataset[, sample, ...])	Initializes a CoverageDistribution class.
plotDistributions([normalize, cumulative, ...])	Plots final distribution values and returns the plotted distribution as a Counter object.

Attributes

pre_sampled
rdd
sample
seed
ss

Features

FeatureSummary(ac, dataset)	FeatureSummary class.

bdgenomics.mango.features.FeatureSummary

class bdgenomics.mango.features.**FeatureSummary**(*ac*, *dataset*)

FeatureSummary class. FeatureSummary provides scrollable visualization of features based on genomic regions.

__init__(*ac*, *dataset*)

Initializes a GenomicRDD viz class.

Args:

param ac bdgenomics.adam.damContext.ADAMContext

param dataset bdgenomics.adam.ds.FeatureDataset

Methods

__init__(ac, dataset)	Initializes a GenomicRDD viz class.
viewPileup(contig, start, end[, reference, ...])	Visualizes a portion of this FeatureDataset in a scrollable pileup widget

Variants

VariantSummary(ac, dataset)	VariantSummary class.

bdgenomics.mango.variants.VariantSummary

class bdgenomics.mango.variants.**VariantSummary**(*ac*, *dataset*)

VariantSummary class. VariantSummary provides scrollable visualization of variants based on genomic regions.

__init__(*ac*, *dataset*)

Initializes a GenomicDataset viz class.

Args:

param ac bdgenomics.adamContext.ADAMContext

param dataset bdgenomics.adam.ds.VariantDataset

Methods

__init__(ac, dataset)	Initializes a GenomicDataset viz class.
viewPileup(contig, start, end[, reference, ...])	Visualizes a portion of this VariantRDD in a scrollable pileup widget

Genotypes

GenotypeSummary(spark, ac, genotypeDataset)	GenotypeSummary class.
VariantsPerSampleDistribution(ss, ...[, sample])	VariantsPerSampleDistribution class.
HetHomRatioDistribution(ss, genotypeDataset)	HetHomRatioDistribution class.
GenotypeCallRatesDistribution(ss, ...[, sample])	GenotypeCallRatesDistribution class.

bdgenomics.mango.genotypes.GenotypeSummary

class bdgenomics.mango.genotypes.**GenotypeSummary**(*spark*, *ac*, *genotypeDataset*, *sample=1.0*)

GenotypeSummary class. GenotypeSummary provides visualizations for genotype based summaries.

__init__(*spark*, *ac*, *genotypeDataset*, *sample=1.0*)

Initializes an GenotypeSummary class.

Args:

param spark SparkSession

param ac bdgenomics.adam.damContext.ADAMContext

param genotypeDataset bdgenomics.adam.ds.GenotypeDataset

param sample fraction of reads to sample from

Methods

__init__(spark, ac, genotypeDataset[, sample])	Initializes an GenotypeSummary class.
getGenotypeCallRatesDistribution()	Computes a distribution of variant CallRate (called / total_variants) per sample
getHetHomRatioDistribution()	Computes a distribution of (Heterozygote/Homozygote) ratios per sample
getVariantsPerSampleDistribution()	Computes distribution of variant counts per sample.
viewPileup(contig, start, end[, reference, ...])	Visualizes a portion of this GenotypeRDD in a scrollable pileup widget

bdgenomics.mango.genotypes.VariantsPerSampleDistribution

class bdgenomics.mango.genotypes.**VariantsPerSampleDistribution**(*ss*, *genotypeDataset*, *sample=1.0*)

VariantsPerSampleDistribution class. VariantsPerSampleDistribution computes a distribution of the count of variants per sample.

__init__(*ss*, *genotypeDataset*, *sample=1.0*)

Initializes a VariantsPerSampleDistributionn class. Computes the coverage distribution of a CoverageDataset. This dataset can have data for multiple samples.

Args:

param ss global SparkSession.

param genotypeDataset bdgenomics.adam.ds.GenotypeDataset

param sample Fraction to sample GenotypeDataset. Should be between 0 and 1

Methods

__init__(ss, genotypeDataset[, sample])	Initializes a VariantsPerSampleDistributionn class.
plotDistributions([normalize, cumulative, ...])	Plots final distribution values and returns the plotted distribution as a Counter object.

Attributes

pre_sampled
rdd
sample
seed
ss

bdgenomics.mango.genotypes.HetHomRatioDistribution

class bdgenomics.mango.genotypes.**HetHomRatioDistribution**(*ss*, *genotypeDataset*, *sample=1.0*)

HetHomRatioDistribution class. HetHomRatioDistribution computes a distribution of (Heterozygote/Homozygote) ratios from a genotypeDataset.

__init__(*ss*, *genotypeDataset*, *sample=1.0*)

Initializes HetHomRatioDistribution class. Retrieves the call rate and missing rate for each sample in a genotypeDataset

Args:

param ss global SparkSession.

param genotypeDataset bdgenomics.adam.ds.GenotypeDataset

param sample Fraction to sample GenotypeDataset. Should be between 0 and 1

Methods

__init__(ss, genotypeDataset[, sample])	Initializes HetHomRatioDistribution class.
plot([testMode])	Plots final distribution values and returns the plotted distribution as a list

bdgenomics.mango.genotypes.GenotypeCallRatesDistribution

class bdgenomics.mango.genotypes.**GenotypeCallRatesDistribution**(*ss*, *genotypeDataset*, *sample=1.0*)

GenotypeCallRatesDistribution class. GenotypeCallRatesDistribution computes a distribution of per-sample

genotype call rates from a genotypeDataset.

__init__(*ss*, *genotypeDataset*, *sample=1.0*)

Initializes a GenotypeCallRatesDistribution class. Retrieves counts of called and missing genotypes from a genotypeDataset.

Args:

param ss SparkContext

param genotypeDataset bdgenomics.adam.ds.GenotypeDataset

param sample Fraction to sample GenotypeDataset. Should be between 0 and 1

Methods

__init__(ss, genotypeDataset[, sample])	Initializes a GenotypeCallRatesDistribution class.
plot([testMode])	Plots final distribution values and returns the plotted distribution as a list

1.1.9 Jupyter Widget API Documentation

bdgenomics.mango.pileup Package

Each widget instance calls the *PileupViewer* to draw an interactive widget for genomic data.

PileupViewer

PileupViewer(**kwargs)	Widget wrapper for pileup.js viewer in Jupyter notebooks.

bdgenomics.mango.pileup.pileupViewer.PileupViewer

class bdgenomics.mango.pileup.pileupViewer.**PileupViewer**(***kwargs*)

Widget wrapper for pileup.js viewer in Jupyter notebooks.

__init__(***kwargs*)

Public constructor

Methods

__init__(**kwargs)	Public constructor
add_class(className)	Adds a class to the top level element of the widget.
add_traits(**traits)	Dynamically add trait attributes to the Widget.
class_own_trait_events(name)	Get a dict of all event handlers defined on this class, not a parent.
class_own_traits(**metadata)	Get a dict of all the traitlets defined on this class, not a parent.
class_trait_names(**metadata)	Get a list of all the names of this class' traits.

Continued on next page

Table 22 – continued from previous page

class_traits(**metadata)	Get a dict of all the traits of this class.
close()	Close method.
close_all()	
getSVG()	Sends request to javascript to convert to svg Needs to run separately from saveSVG because js cannot message to kernel until cell is completed.
get_manager_state([drop_defaults, widgets])	Returns the full state for a widget manager for embedding
get_state([key, drop_defaults])	Gets the widget state, or a piece of it.
get_view_spec()	
goto(chrom, start, stop)	Redirects widget view to new genomic locus.
handle_comm_opened(comm, msg)	Static method, called when a widget is constructed.
has_trait(name)	Returns True if the object has a trait with the specified name.
hold_sync()	Hold syncing any state until the outermost context manager exits
hold_trait_notifications()	Context manager for bundling trait change notifications and cross validation.
notify_change(change)	Called when a property has changed.
observe(handler[, names, type])	Setup a handler to be called when a trait changes.
on_displayed(callback[, remove])	(Un)Register a widget displayed callback.
on_msg(callback[, remove])	(Un)Register a custom msg receive callback.
on_trait_change([handler, name, remove])	DEPRECATED: Setup a handler to be called when a trait changes.
on_widget_constructed(callback)	Registers a callback to be called when a widget is constructed.
open()	Open a comm to the frontend if one isn't already open.
remove_class(className)	Removes a class from the top level element of the widget.
saveSVG(filepath)	Saves svg to filepath
send(content[, buffers])	Sends a custom msg to the widget model in the front-end.
send_state([key])	Sends the widget state, or a piece of it, to the front-end, if it exists.
set_state(sync_data)	Called when a state is received from the front-end.
set_trait(name, value)	Forcibly sets trait attribute, including read-only attributes.
setup_instance(**kwargs)	This is called **before** self.__init__ is called.
trait_defaults(*names, **metadata)	Return a trait's default value or a dictionary of them
trait_events([name])	Get a dict of all the event handlers of this class.
trait_has_value(name)	Returns True if the specified trait has a value.
trait_metadata(traitname, key[, default])	Get metadata values for trait by key.
trait_names(**metadata)	Get a list of all the names of this class' traits.
trait_values(**metadata)	A dict of trait names and their values.
traits(**metadata)	Get a dict of all the traits of this class.
unobserve(handler[, names, type])	Remove a trait change handler.
unobserve_all([name])	Remove trait change handlers of any type for the specified name.
zoomIn()	Zooms widget view in by a factor of 2.
zoomOut()	Zooms widget view out by a factor of 2.

Attributes

chrom	A trait for unicode strings.
comm	A trait whose value must be an instance of a specified class.
cross_validation_lock	A contextmanager for running a block with our cross validation lock set to True.
id	
keys	An instance of a Python list.
layout	An instance trait which coerces a dict to an instance.
log	A trait whose value must be an instance of a specified class.
model_id	Gets the model id of this widget.
msg	A trait for unicode strings.
reference	A trait for unicode strings.
start	An int trait.
stop	An int trait.
svg	A trait for unicode strings.
tracks	An instance of a Python list.
widget_types	
widgets	

Sources

Sources specify where the genomic data comes from. Sources can come from a url, a GA4GHDatasource, or a JSON string of GA4GH formatted data.

BamDataSource(url[, indexUrl])	Initializes file source from bam file endpoint.
VcfDataSource(url[, indexUrl])	Initializes file source from vcf file endpoint.
TwoBitDataSource(url[, indexUrl])	Initializes file source from twoBit file endpoint.
BigBedDataSource(url[, indexUrl])	Initializes file source from big bed (.bb) file endpoint.
GA4GHAlignmentJson(json)	Initializes GA4GH Alignment JSON.
GA4GHVariantJson(json)	Initializes GA4GH variant JSON.
GA4GHFeatureJson(json)	Initializes GA4GH feature JSON.
GA4GHAlignmentSource(endpoint, readGroupId)	Initializes GA4GHAlignmentSource.
GA4GHVariantSource(endpoint, readGroupId[, ...])	Initializes GA4GHSource.
GA4GHFeatureSource(endpoint, readGroupId[, ...])	Initializes GA4GHFeatureSource.

bdgenomics.mango.pileup.sources.BamDataSource

class bdgenomics.mango.pileup.sources.**BamDataSource**(*url*, *indexUrl=None*)

Initializes file source from bam file endpoint.

Args:

param str url to file

param str indexUrl to index file

__init__(*url*, *indexUrl=None*)
Initializes file sources.

Args:

param str url to file

param str indexUrl to index file

Methods

__init__(url[, indexUrl])	Initializes file sources.

Attributes

dict_
name — name that pileup.js uses to identify sources

bdgenomics.mango.pileup.sources.VcfDataSource

class bdgenomics.mango.pileup.sources.**VcfDataSource**(*url*, *indexUrl=None*)
Initializes file source from vcf file endpoint.

Args:

param str url to file

param str indexUrl to index file

__init__(*url*, *indexUrl=None*)
Initializes file sources.

Args:

param str url to file

param str indexUrl to index file

Methods

__init__(url[, indexUrl])	Initializes file sources.

Attributes

dict_
name — name that pileup.js uses to identify sources

bdgenomics.mango.pileup.sources.TwoBitDataSource

class bdgenomics.mango.pileup.sources.**TwoBitDataSource**(*url*, *indexUrl=None*)
Initializes file source from twoBit file endpoint.

Args:

param str url to file

__init__(*url*, *indexUrl=None*)
Initializes file sources.

Args:

param str url to file

param str indexUrl to index file

Methods

__init__(url[, indexUrl])	Initializes file sources.

Attributes

dict_
name

bdgenomics.mango.pileup.sources.BigBedDataSource

class bdgenomics.mango.pileup.sources.**BigBedDataSource**(*url*, *indexUrl=None*)
Initializes file source from big bed (.bb) file endpoint.

Args:

param str url to file

__init__(*url*, *indexUrl=None*)
Initializes file sources.

Args:

param str url to file

param str indexUrl to index file

Methods

__init__(url[, indexUrl])	Initializes file sources.

Attributes

dict_	
name	name that pileup.js uses to identify sources

bdgenomics.mango.pileup.sources.GA4GHAlignmentJson

class bdgenomics.mango.pileup.sources.**GA4GHAlignmentJson**(*json*)

Initializes GA4GH Alignment JSON.

Args:

param str json in GA4GH format

__init__(*json*)

Initializes GA4GH JSON.

Args:

param str json in GA4GH format

Methods

__init__(json)	Initializes GA4GH JSON.

Attributes

dict_	
name	name that pileup.js uses to identify sources

bdgenomics.mango.pileup.sources.GA4GHVariantJson

class bdgenomics.mango.pileup.sources.**GA4GHVariantJson**(*json*)

Initializes GA4GH variant JSON.

Args:

param str json in GA4GH format

__init__(*json*)

Initializes GA4GH JSON.

Args:

param str json in GA4GH format

Methods

__init__(json)	Initializes GA4GH JSON.

Attributes

dict_	
name	name that pileup.js uses to identify sources

bdgenomics.mango.pileup.sources.GA4GHFeatureJson

class bdgenomics.mango.pileup.sources.**GA4GHFeatureJson**(*json*)

Initializes GA4GH feature JSON.

Args:

param str json in GA4GH format

__init__(*json*)

Initializes GA4GH JSON.

Args:

param str json in GA4GH format

Methods

__init__(json)	Initializes GA4GH JSON.

Attributes

dict_	
name	name that pileup.js uses to identify sources

bdgenomics.mango.pileup.sources.GA4GHAlignmentSource

class bdgenomics.mango.pileup.sources.**GA4GHAlignmentSource**(*endpoint*, *readGroupId*, *callSetIds=None*)

Initializes GA4GHAlignmentSource.

Args:

param str url endpoint

param str read group id

__init__(*endpoint*, *readGroupId*, *callSetIds=None*)

Initializes GA4GHSource.

Args:

param str url endpoint

param str read group id

param str optional call set ID for variants

Methods

__init__(endpoint, readGroupId[, callSetIds])	Initializes GA4GHSource.

Attributes

dict_	
name	name that pileup.js uses to identify sources

bdgenomics.mango.pileup.sources.GA4GHVariantSource

class bdgenomics.mango.pileup.sources.**GA4GHVariantSource**(*endpoint*, *readGroupId*, *callSetIds=None*)

Initializes GA4GHSource.

Args:

param str url endpoint

param str call set ID

param str optional call set ID for variants

__init__(*endpoint*, *readGroupId*, *callSetIds=None*)

Initializes GA4GHSource.

Args:

param str url endpoint

param str read group id

param str optional call set ID for variants

Methods

__init__(endpoint, readGroupId[, callSetIds])	Initializes GA4GHSource.

Attributes

dict_	
name	name that pileup.js uses to identify sources

bdgenomics.mango.pileup.sources.GA4GHFeatureSource

class bdgenomics.mango.pileup.sources.**GA4GHFeatureSource**(*endpoint*, *readGroupId*, *callSetIds=None*)

Initializes GA4GHFeatureSource.

Args:

param str url endpoint

__init__(*endpoint*, *readGroupId*, *callSetIds=None*)

Initializes GA4GHSource.

Args:

param str url endpoint

param str read group id

param str optional call set ID for variants

Methods

__init__(endpoint, readGroupId[, callSetIds])	Initializes GA4GHSource.

Attributes

dict_	
name	name that pileup.js uses to identify sources

Track

Tracks specify what visualization will be drawn.

Track(**kwargs)	A trait for a pileupTrack, requires a viz string of (coverage, pileup, features, variants, genome, genes, scale, or location) and a DataSource.
track_to_json(pyTrack, manager)	Serialize a Track.
track_from_json(js, manager)	Deserialize a Track from JSON.
tracks_to_json(pyTracks, manager)	Serialize a Python date object.
tracks_from_json(js, manager)	Deserialize a list of Tracks from JSON.

bdgenomics.mango.pileup.track.Track

class bdgenomics.mango.pileup.track.**Track**(***kwargs*)

A trait for a pileupTrack, requires a viz string of (coverage, pileup, features, variants, genome, genes, scale, or location) and a DataSource.

__init__(***kwargs*)

Initializes track.

Args:

param kwargs Should contain viz, optional source, optional label.

Methods

__init__(**kwargs)	Initializes track.
class_init(cls, name)	Part of the initialization which may depend on the underlying HasDescriptors class.
default([obj])	The default generator for this trait
default_value_repr()	
error(obj, value[, error, info])	Raise a TraitError
from_string(s)	Get a value from a config string

Continued on next page

Table 46 – continued from previous page

get(obj[, cls])	
get_default_value()	DEPRECATED: Retrieve the static default value for this trait.
get_metadata(key[, default])	DEPRECATED: Get a metadata value.
info()	
init_default_value(obj)	DEPRECATED: Set the static default value for the trait type.
instance_init(obj)	Part of the initialization which may depend on the underlying HasDescriptors instance.
set(obj, value)	
set_metadata(key, value)	DEPRECATED: Set a metadata key/value.
subclass_init(cls)	
tag(**metadata)	Sets metadata and returns self.

Attributes

allow_none	
default_value	
info_text	
label	Label for this track.
metadata	
name	
read_only	
source	Datasource consumed by pileup.js.
sourceOptions	Options for source.
this_class	
viz	visualization specified in pileup.js.

bdgenomics.mango.pileup.track.track_to_json

bdgenomics.mango.pileup.track.**track_to_json**(*pyTrack*, *manager*)

Serialize a Track. Attributes of this dictionary are to be passed to the JavaScript Date constructor.

Args:

param Track Track object

param any manager. Used for widget serialization.

Returns: dict of Track elements (viz, source, sourceOptions and label)

bdgenomics.mango.pileup.track.track_from_json

bdgenomics.mango.pileup.track.**track_from_json**(*js*, *manager*)

Deserialize a Track from JSON.

Args:

param (str) json for Track containing viz, source, sourceOptions and label

param (any) manager. Used for widget serialization.

Returns: Track: pileup Track built from json

bdgenomics.mango.pileup.track.tracks_to_json

bdgenomics.mango.pileup.track.**tracks_to_json**(*pyTracks*, *manager*)

Serialize a Python date object. Attributes of this dictionary are to be passed to the JavaScript Date constructor.

Args:

param (List) List of Tracks

param (any) manager. Used for widget serialization.

Returns: List of dict of Track elements (viz, source, sourceOptions and label)

bdgenomics.mango.pileup.track.tracks_from_json

bdgenomics.mango.pileup.track.**tracks_from_json**(*js*, *manager*)

Deserialize a list of Tracks from JSON.

Args:

param (str) json for list of Tracks containing viz, source, sourceOptions and label

param (any) manager. Used for widget serialization.

Returns: List: List of pileup Track built from json

1.1.10 Supported Files

Supported File Types

Mango supports the following file types:

Data Type	Supported File Types
Alignments	Parquet, bam, indexed bam, sam
Variants	Parquet, vcf, indexed vcf, vcf.gz
Features	Parquet, bed, narrowPeak
Genome	Parquet, twoBit*, fa, fasta

*TwoBit files must be staged locally for access.

Accessing http files through Mango

Mango can copy and read http files. To do so, when running `mango-submit`, set `spark.local.dir` to a path in the user's home directory:

```
./bin/mango-submit
    --conf spark.local.dir=<user home>/spark-tmp
```

This will allow Spark to access temporary http files.

Accessing s3a files through Mango

To access s3a files when running on AWS, you need the `net.fnothaft:jsr203-s3a` package, and the bam splitter to be enabled:

```
./bin/mango-submit \
        --packages org.apache.parquet:parquet-avro:1.8.2 \
        --packages net.fnothaft:jsr203-s3a:0.0.2 \
        --conf spark.hadoop.hadoopbam.bam.enable-bai-splitter=true \
        -- hg19.2bit \
        -reads s3a://1000genomes/phase1/data/NA12878/exome_alignment/NA12878.mapped.
↪illumina.mosaik.CEU.exome.20110411.bam
```

1.1.11 Running Mango from Docker

Running Mango from Docker requires Docker to be installed.

Mango is available on Docker and is available at quay.io.

To pull the Mango docker container, run:

```
docker pull quay.io/bigdatagenomics/mango:latest
```

Running Mango Browser on Docker

To run Mango browser example files on Linux in Docker run:

To run Mango browser on local data, you must first mount these files with the `Docker -v` flag. For example, if you have local files stored at `<example-file-path>`:

```
LOCAL_EXAMPLE_FILES=<path_to_example_files>
DOCKER_EXAMPLE_FILES=<path_in_docker_container>

docker run -it -p 8080:8080 \
    -v $LOCAL_EXAMPLE_FILES:$DOCKER_EXAMPLE_FILES \
    --entrypoint=mango-submit \
    quay.io/bigdatagenomics/mango:latest \
    -- $DOCKER_EXAMPLE_FILES/<genome_build_filepath> \
    -reads $DOCKER_EXAMPLE_FILES/<alignment_filepaths> \
    -variants $DOCKER_EXAMPLE_FILES/<variant_filepaths>
```

To create a genome build (<genome_build_filepath>), see *Building a Genome*.

Running Mango Notebook on Docker

To run Mango notebook on Linux in Docker run:

```
docker run --net=host -it -p 8888:8888 \
    --entrypoint=mango-notebook \
    quay.io/bigdatagenomics/mango:latest \
    -- --ip=0.0.0.0 --allow-root
```

Note: To run the Mango notebook on OS X, remove `--net=host`.

To view a number of ipython notebook examples, see our github.

Creating a Mango genome using Docker

To run create a mango genome on Linux in Docker run:

```
LOCAL_LOCATION=<host_src>
DOCKER_LOCATION=<docker_src>
GENOME_BUILD=<genome_name> # i.e. hg19, mm10, etc.

docker run --net=host -it -p 8888:8888 \
    -v $LOCAL_LOCATION:$DOCKER_LOCATION \
    --entrypoint=make_genome \
    quay.io/bigdatagenomics/mango:latest $GENOME_BUILD $DOCKER_LOCATION
```

The genome file will be saved to `<host_src>`.

Note: To run the make_genome on OS X, remove `--net=host`.

1.1.12 Running Mango on Google Cloud

Cloud Dataproc provides pre-built Hadoop and Spark distributions which allows users to easily deploy and run Mango.

This documentation explains how to configure requirements to connect with Google Cloud on your local machine, and how to run Mango on GCP.

Before you Start

Commands in this section will require users to:

1. Create an account on Google Cloud
2. Install the gcloud cli

Creating a Dataproc Cluster

The necessary initialization scripts are available at the Google Cloud Storage bucket located at gs://mango-initialization-bucket/

In order to access this bucket through the cloud dataproc cluster, it is necessary to give billing account manager permissions to dataproc-accounts.iam.gserviceaccount.com and compute@developer.gserviceaccount.com through the IAM console.

Create the Cloud Dataproc Cluster (modify the fields as appropriate) with the below installation script:

```
gcloud dataproc clusters create <cluster-name> \
    --project <project_id> \
    --bucket <optional_bucket_name> \
    --metadata MINICONDA_VARIANT=2 \
    --master-machine-type=n1-standard-1 \
    --worker-machine-type=n1-standard-1 \
    --master-boot-disk-size=50GB \
    --worker-boot-disk-size=10GB \
    --initialization-actions \
        gs://mango-initialization-bucket/install.sh
```

After the above steps are completed, ssh into the master node with the following command:

```
gcloud compute ssh <cluster-name>-m
```

1.1.13 Running Mango on GCE through Docker

Before Mango can run, it is recommended to stage datasets into HDFS if you are trying to view specific files. The created container will share the same hadoop file system with the root master user.

```
hdfs dfs -put /<local-machine-path> /<hdfs-path>
```

Public datasets can be accessed by referencing google cloud storage at gs://genomics-public-data/.

More information about available public datasets on Google cloud can be found online

More information on using the dataproc cluster's Spark interface is available through Google Cloud documentation

Running the Mango Notebook on GCE

An example docker startup script is available in the Mango GCE scripts directory for running mango notebook.

```
wget 'https://raw.githubusercontent.com/bigdatagenomics/mango/master/bin/gce/run-
↪notebook.sh'

bash run-notebook.sh
```

Note: root permissions may be necessary for Docker.

Once the notebook is running, connect to Mango by setting up a tunnel to your local computer via the exposed port in the master node:

```
gcloud compute ssh <cluster-name>-m -- -N -L localhost:<local-port>:localhost:8888
```

You can navigate to the notebook through your local browser by pointing it towards http://localhost:<local-port>/.

Running the Mango Browser on GCE

To run Mango Browser on GCE on top of Docker, you will first need to configure a reference. To create a reference, see Building a Genome.

Simply run:

```
sudo docker run \
      -i \
      -t \
      -v <OUTPUT_DIR>:<OUTPUT_DIR> \
      --entrypoint=make_genome \
      quay.io/bigdatagenomics/mango:latest \
       <GENOME_NAME> <OUTPUT_DIR>
```

This will save a file called `<GENOME_NAME>.genome` to your `<OUTPUT_LOCATION>`. Now that you have a reference, you can run Mango browser.

An example docker startup script for the Mango browser is available in the Mango GCE scripts directory for running mango browser.

```
wget 'https://raw.githubusercontent.com/bigdatagenomics/mango/master/bin/gce/run-
↪browser.sh'

bash run-browser.sh \
    --  \
    <path_to_genome>/hg19.genome \
    <MANGO_ARGS>
```

1.1.14 Running Mango from Amazon EMR

Amazon Elatic Map Reduce (EMR) provides pre-built Hadoop and Spark distributions which allows users to easily deploy and run Mango. This documentation explains how to configure requirements to connect with AWS on your local machine, and how to run Mango on AWS.

Before you Start

You will first need to set up an EC2 key pair and configure the AWS Command Line Interface (AWS CLI) using your key. This will allow you to create clusters and ssh into your machine from the command line.

First, follow the following configuration steps:

1. Set up an EC2 key pair. This keypair allows you to securely access instances on AWS using a private key.
2. Install the AWS CLI
3. Configure the AWS CLI

This will allow you to access AWS using your credentials.

1.1.15 Running Mango through Docker

This section explains how to run the Mango browser and the Mango notebook through Docker on Amazon EMR. Using Docker allows users to quickly get Mango up and running without needing to configure different environment variables on their cluster. If you need more flexibility in configuration, see Running Mango *standalone*.

Creating a Cluster

First, you must configure an EMR cluster. This can be done using the AWS CLI.

Through the command line, create a new cluster:

```
aws emr create-cluster
--release-label emr-5.27.0 \
--name 'emr-5.27.0 Mango example' \
--applications Name=Hadoop Name=Hive Name=Spark  \
--ec2-attributes KeyName=<YOUR_EC2_KEY>,InstanceProfile=EMR_EC2_DefaultRole \
--service-role EMR_DefaultRole \
--instance-groups \
  InstanceGroupType=MASTER,InstanceCount=1,InstanceType=c3.4xlarge \
  InstanceGroupType=CORE,InstanceCount=4,InstanceType=c3.4xlarge \
--region <your_region> \
--log-uri s3://<your-s3-bucket>/emr-logs/ \
--bootstrap-actions \
Name='Install Mango', Path="s3://bdg-mango/install-bdg-mango-docker-emr5.sh"
```

In the code chunk above, set your EC2 key pair name:

```
--ec2-attributes KeyName=<YOUR_EC2_KEY>
```

Note the instance counts:

```
InstanceGroupType=CORE,InstanceCount=4,InstanceType=c3.4xlarge
```

In this example, we have set the number of instance counts, or the number of workers, to 4. If you are using larger or smaller workloads, you should scale this number up or down accordingly. Note that more instances will cost more money.

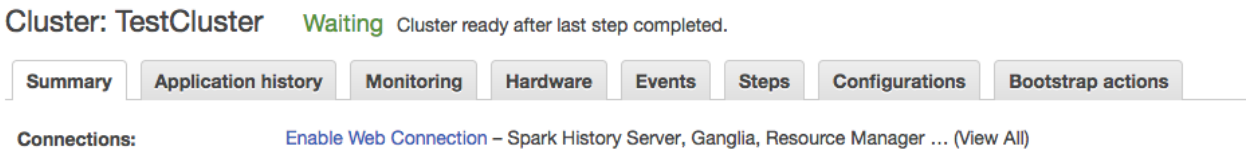
The bootstrap action:

```
--bootstrap-actions \
Name='Install Mango', Path="s3://bdg-mango/install-bdg-mango-docker-emr5.sh"
```

will download docker and required scripts. These scripts will be available on your EMR master node in the directory `/home/hadoop/mango-scripts`.

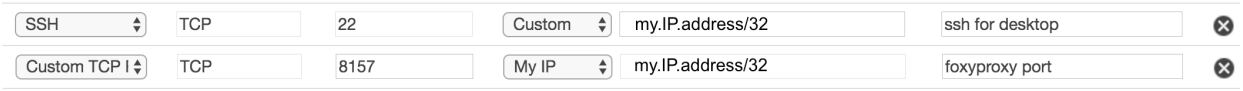
Enabling a Web Connection

To view the Spark UI, notebook, and browser, you must setup a web connection for the cluster. To do so, navigate to your Amazon EMR Clusters page, click your started cluster, and click on **Enable Web Connection** and follow the instructions for enabling a connection.



These instructions will require you to install the FoxyProxy addon in your web browser. Note that for accessing the recommended 8157 port using FoxyProxy (as well as port 22 for ssh), you will have to expose these ports in the security group for the master node.

To expose required ports on the master node, navigate to **Security and access** in your Cluster EMR manager. Click on **Security groups for Master**. Add a inbound new rule for ssh port 22 and a new TCP rule for the port configured in FoxyProxy inbound to <YOUR_PUBLIC_IP_ADDRESS>/32.



Connecting to Cluster

To ssh into your cluster, navigate to your EMR cluster in AWS console and click on `ssh`. This will give you the command you need to ssh into the cluster.

Accessing the Web UI

Click on **Enable Web Connection** in the AWS cluster console and run the ssh command for accessing the UIs through your browser. The command line argument will look like this:

```
ssh -i ~/MyKey.pem -ND <PORT_NUM> hadoop@<PUBLIC_MASTER_DNS>
```

Where <PORT_NUM> is the configured port in FoxyProxy (default is 8157), and hadoop@<PUBLIC_MASTER_DNS> is the address you use to ssh into the master cluster node. Let this run throughout your session.

Testing your Configuration

You should now be able to access the Hadoop UI. The Hadoop UI is located at:

```
<PUBLIC_MASTER_DNS>:8088
```

You can access Spark applications through this UI when they are running.

Running the Mango Browser on EMR with Docker

To run Mango Browser on EMR on top of Docker with the hg19 genome run:

```
GENOME_BUILD_FILE=<path_to_genome_file> # ie hg18.genome, hg19.genome, mm10.genome,␣
↪etc.

/home/hadoop/mango-scripts/run-browser-docker.sh <SPARK_ARGS> -- $GENOME_BUILD_FILE \
  -reads s3a://1000genomes/phase1/data/NA19685/exome_alignment/NA19685.mapped.
↪illumina.mosaik.MXL.exome.20110411.bam
```

Note You must first create a GENOME_BUILD_FILE. To do so, see *creating a reference genome in docker*.

Navigate to <PUBLIC_MASTER_DNS>:8081 to access the browser. In the browser, navigate to `TP53`, `chr17-chr17:7,510,400-7,533,590` to view exome data.

Creating a reference genome on EMR with Docker

To run the Mango browser, you must first create a reference genome. For example, to create an hg18 genome, run:

```
/home/hadoop/mango-scripts/make-genome-docker.sh hg18 <output_directory>
```

This script will create a `.genome` file and save it to <output_directory> on the master host.

You can then run the Mango browser using your new genome:

```
/home/hadoop/mango-scripts/run-browser-docker.sh <SPARK_ARGS> -- <output_directory>/
↪hg18.genome
```

The `run-browser-docker.sh` script mounts the location of your new genome file, making it accessible to the docker container.

Note: s3a latency slows down Mango browser. For interactive queries, you can first transfer s3a files to HDFS.

You can then run the Mango browser on HDFS files:

```
/home/hadoop/mango-scripts/run-browser-docker.sh <SPARK_ARGS> -- <path_to_genome_file>
↪/hg19.genome \
  -reads hdfs:///user/hadoop/NA19685.mapped.illumina.mosaik.MXL.exome.20110411.bam
```

Note: The first time Docker may take a while to set up.

Running Mango Notebook on EMR with Docker

To run the Mango Notebook on EMR on top of Docker, run the `run-notebook-docker` script:

```
# Run the Notebook
/home/hadoop/mango-scripts/run-notebook-docker.sh <SPARK_ARGS> -- <NOTEBOOK_ARGS>
```

Where <SPARK_ARGS> are Spark specific arguments and <NOTEBOOK_ARGS> are Jupyter notebook specific arguments. Example Spark arguments are shown in the following example:

```
./run-notebook.sh --master yarn --num-executors 64 --executor-memory 30g --
```

Note: It will take a couple minutes on startup for the Docker configuration to complete.

Navigate to <PUBLIC_MASTER_DNS>:8888 to access the notebook. Type in the Jupyter notebook token provided in the terminal. An example notebook for EMR can be found on the Mango GitHub.

Accessing files in the Mango notebook from HDFS

Mango notebook and Mango browser can also access files from HDFS on EMR. To do so, first put the files in HDFS:

```
hdfs dfs -put <my_file.bam>
```

You can then reference the file through the following code in Mango notebook:

```
ac.loadAlignments('hdfs:///user/hadoop/<my_file.bam>')
```

1.1.16 Running Mango Standalone

This section explains how to run the Mango browser and the Mango notebook without Docker on EMR.

Creating a Cluster

Through the AWS command line, create a new cluster with the latest Mango version:

```
aws emr create-cluster
--release-label emr-5.27.0 \
--name 'emr-5.27.0 Mango example' \
--applications Name=Hadoop Name=Hive Name=Spark Name=JupyterHub  \
--ec2-attributes KeyName=<YOUR_EC2_KEY>,InstanceProfile=EMR_EC2_DefaultRole \
--service-role EMR_DefaultRole \
--instance-groups \
  InstanceGroupType=MASTER,InstanceCount=1,InstanceType=c3.4xlarge \
  InstanceGroupType=CORE,InstanceCount=4,InstanceType=c3.4xlarge \
--region <your_region> \
--log-uri s3://<your-s3-bucket>/emr-logs/ \
--bootstrap-actions \
Name='Install Mango', Path="s3://bdg-mango/install-bdg-mango-dist-emr5.sh"
```

Where $VERSION specifies the Mango version available in the Maven central repository.

This bootstrap action will download Mango distribution code, and an example notebook file for the Mango notebook will be available at `/home/hadoop/mango-distribution-${VERSION}/notebooks/aws-1000genomes.ipynb`.

Finally, make sure you set your SPARK_HOME env:

```
export SPARK_HOME=/usr/lib/spark
```

Running Mango Browser on EMR

To run Mango Browser on EMR on top of Docker, you will first need to configure a reference. To create a reference, see Building a Genome.

Simply run:

```
make_genome <GENOME_NAME> <OUTPUT_LOCATION>
```

This will save a file called `<GENOME_NAME>.genome` to your `<OUTPUT_LOCATION>`. Now that you have a reference, you can run Mango browser:

```
/home/hadoop/mango/scripts/run-browser-emr.sh \
           --  \
            <path_to_genome>/hg19.genome \
           -reads s3a://1000genomes/phase1/data/NA19685/exome_alignment/NA19685.
↪mapped.illumina.mosaik.MXL.exome.20110411.bam \
           -port 8080
```

To visualize data in the NA19685 exome, navigate to `chr17:7,569,720-7,592,868`. Here, you will see reads surrounding TP53.

Note: Pulling data from s3a has high latency, and thus slows down Mango browser. For interactive queries, you can first transfer s3a files to HDFS. The package `net.fnothaft:jsr203-s3a:0.0.2` used above is required for loading files from s3a. This is not required if you are only accessing data from HDFS.

If you have not *established a web connection*, set up an ssh tunnel on the master node to view the browser at port 8081.

In the browser, navigate to a `TP53, chr17:7,510,400-7,533,590` with exome data to view results.

Running Mango Notebook on EMR

To run Mango Notebook on EMR, run the mango-notebook script:

```
/home/hadoop/mango/scripts/run-notebook-emr.sh \
      -- <NOTEBOOK_ARGS>
```

If you have *established a web connection*, you will now be able to access the Mango notebook at `<PUBLIC_MASTER_DNS>:8888`.

1.1.17 Development notes for the Mango Browser

Please follow the Mango source installation requirements before continuing.

Debugging the Mango browser frontend

Mango browser uses scalatra as a web server. To interactively modify the frontend browser while running scalatra, use the "-debugFrontend" flag:

```
./bin/mango-submit <args> -debugFrontend
```

This allows scalatra to directly access ssp, css and javascript resources without packaging Mango.

- genindex
- search

CHAPTER 2

References

Massie, Matt, Frank Nothaft, Christopher Hartl, Christos Kozanitis, André Schumacher, Anthony D Joseph, and David A Patterson. 2013. "ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing." UCB/EECS-2013-207, EECS Department, University of California, Berkeley.

Morrow, Alyssa, Anthony Joseph, and Nir Yosef. 2017. "Distributed Visualization for Genomic Analysis." UCB/EECS-2017-82, EECS Department, University of California, Berkeley.

McKenna, Aaron, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, et al. 2010. "The Genome Analysis Toolkit: A MapReduce Framework for Analyzing Next-Generation DNA Sequencing Data." *Genome Research* 20 (9). Cold Spring Harbor Lab: 1297–1303.

Melnik, Sergey, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. "Dremel: Interactive Analysis of Web-Scale Datasets." *Proceedings of the VLDB Endowment* 3 (1-2). VLDB Endowment: 330–39.

Nothaft, Frank A, Matt Massie, Timothy Danford, Zhao Zhang, Uri Laserson, Carl Yeksigian, Jey Kottalam, et al. 2015. "Rethinking Data-Intensive Science Using Scalable Analytics Systems." In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*. ACM.

Schadt, Eric E, Michael D Linderman, Jon Sorenson, Lawrence Lee, and Garry P Nolan. 2010. "Computational Solutions to Large-Scale Data Management and Analysis." *Nature Reviews Genetics* 11 (9). Nature Publishing Group: 647–57.

Vanderkam, Dan, B. Arman Aksoy, Isaac Hodes, Jaclyn Perrone, and Jeff Hammerbacher. 2016. "pileup.js: a JavaScript library for interactive and in-browser visualization of genomic data." In *Bioinformatics*, 32 (15).

Vavilapalli, Vinod Kumar, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, et al. 2013. "Apache Hadoop YARN: Yet Another Resource Negotiator." In *Proceedings of the Symposium on Cloud Computing (SoCC '13)*, 5. ACM.

Zaharia, Matei, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, and Ion Stoica. 2012. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for in-Memory Cluster Computing." In *Proceedings of the Conference on Networked Systems Design and Implementation (NSDI '12)*, 2. USENIX Association.

Python Module Index

b

Index

Symbols

A

B

C

F

G

H

I

M

P

T

V