RUM 2 (RNA-Seq Unified Mapper)

Mike DeLaurentis

University of Pennsylvania

August 15, 2012

Agenda

1. Intro
2. The RUM pipeline
3. Tour of the new RUM
   - Installation
   - Command-line interface
   - Job state management
   - Work distribution
4. Demo
   - Web resources
   - Run tiny job
5. Wrap up

Agenda

Who am I?

- Working at Penn (ITMAT) since January
- Software engineering background
- Experience in a variety of languages, (Perl, Java, Clojure (lisp), Python, C, Ruby)
- And in a lot of different environments
- Made some big changes to RUM recently

Agenda

What is RUM?

- *"RUM is an alignment, junction calling, and feature quantification pipeline specifically designed for Illumina RNA-Seq data"*
- Written by Gregory Grant (ggrant@grant.org)
- Runs on Linux / UNIX / Mac
- Distributed (optionally)

What is RUM?

Inputs

- RNA-Seq reads: FASTA or FASTQ, paired or single
- Organism-specific index

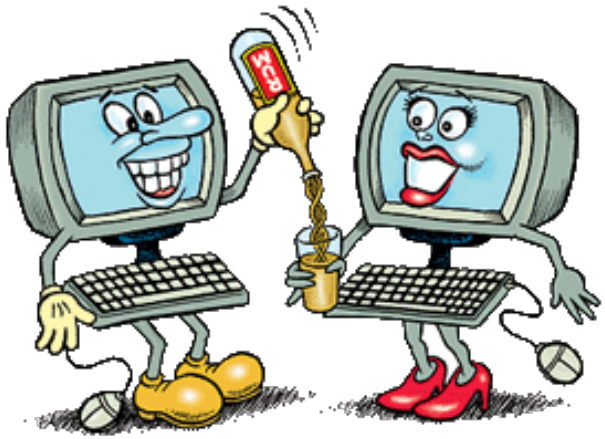
What is RUM?

Inputs

- RNA-Seq reads: FASTA or FASTQ, paired or single
- Organism-specific index

Outputs

- Unique and non-unique alignments
- Coverage plots
- Feature quantifications
- Junction calls
- List of novel inferred internal exons



RUM has three distinct phases:

- Preprocessing
- Processing
- Postprocessing

Phase 1: Preprocessing

- Perform some quality checks on reads
- Split reads into N chunks
- Allows reads to be processed by N nodes on a cluster

Phase 2: Processing

- Align all reads against genome using Bowtie
- Align all reads against transcriptome using Bowtie
- Merge genome and transcriptome alignments and identify unmapped reads
- Align unmapped reads against genome using BLAT
- *"This leverages the advantages of both genome and transcriptome mapping as well as combining the speed of Bowtie with the sensitivity and flexibility of Blat."*
- Merge Bowtie and Blat alignments

Phase 3: Postprocessing

- Merge alignments for all chunks together
- Produce coverage plots, junction files
- Find novel internal exons
- Generate some reports

Agenda

Big changes for RUM 2

- Focus on code reuse / modularity
- Standardize installation process
- Revamp command-line interface
- Improve inter-process communication
- More robust job state management
- Automated tests
- Extensible support for clusters

Installing RUM

- Uses standard Perl Makefile.PL
- Should be familiar to system administrators
- Download tarball from `https://github.com/PGFI/rum/downloads`
- `perl Makefile.PL`
- `make install` (optional)
- Then install indexes...

Installing Indexes

- RUM needs an index for each organism you want to align against
- Index includes genome, gene annotations, and binary index files for Bowtie
- Index installation is now separate from code installation / upgrade
- Run `rum_indexes`; it will guide you through the process

Command-line interface

Usage is `rum_runner ACTION [OPTIONS]` where action is one of:

- `align` - Run an alignment
- `status` - Check the status of a job
- `stop` - Stop a job (can be restarted later)
- `kill` - Stop a job and clean it up (to restart from scratch)
- `clean` - Remove output files for a job
- `help` - Get help
- `version` - Show version number

Running an alignment

Use "`rum_runner align`" to run an alignment:

```
rum_runner align \
  --output ~/sample123/results \
  --index  ~/rum_indexes/hg19 \
  --name   TestJob \
  --chunks 25 \
  ~/sample123/forward.fq ~/sample123/reverse.fq
```

Job status

- Use `rum_runner status` to check on the status of a running job.

```
$ rum_runner status -o ~/sample123/results

Processing in 25 chunks
-----------------------
XXXXXXXXXXXXXXXXXXXXXXXXX Run bowtie on genome
XXXXXXXXXXXXXXXXXXXXXXXXX Parse genome Bowtie output
X XXX  XX XX XXXX  XXXXX  Run bowtie on transcriptome
X XXX  XX XX XXXX  XXXXX  Parse transcriptome Bowtie output
X XXX  XX XX XXXX  XXXXX  Merge unique mappers together
X XXX  XX XX XXXX  XXXXX  Merge non-unique mappers together
X XXX  XX XX XXXX  XXXXX  Make unmapped reads file for blat
X XXX  XX XX XXXX  XXXXX  Run blat on unmapped reads
X XXX  XX XX XXXX  XXXXX  Run mdust on unmapped reads
X XXX  XX XX XXXX  XXXXX  Parse blat output
X XXX  XX XX XXXX  XXXXX  Merge bowtie and blat results
X XX   XX XX  XXX   XXXX  Clean up RUM files
X XX   XX XX  XXX   XXXX  Produce RUM_Unique
X XX   XX XX  XXX   XXXX  Sort RUM_Unique by location
X X    XX XX  XXX   XXXX  Sort cleaned non-unique mappers by ID
X X    XX XX  XXX   XXXX  Remove duplicates from NU
X X    XX XX  XXX   XXXX  Create SAM file
X X    XX XX  XXX   XXXX  Create non-unique stats
X X    XX XX  XXX   XXXX  Sort RUM_NU
```

Job status

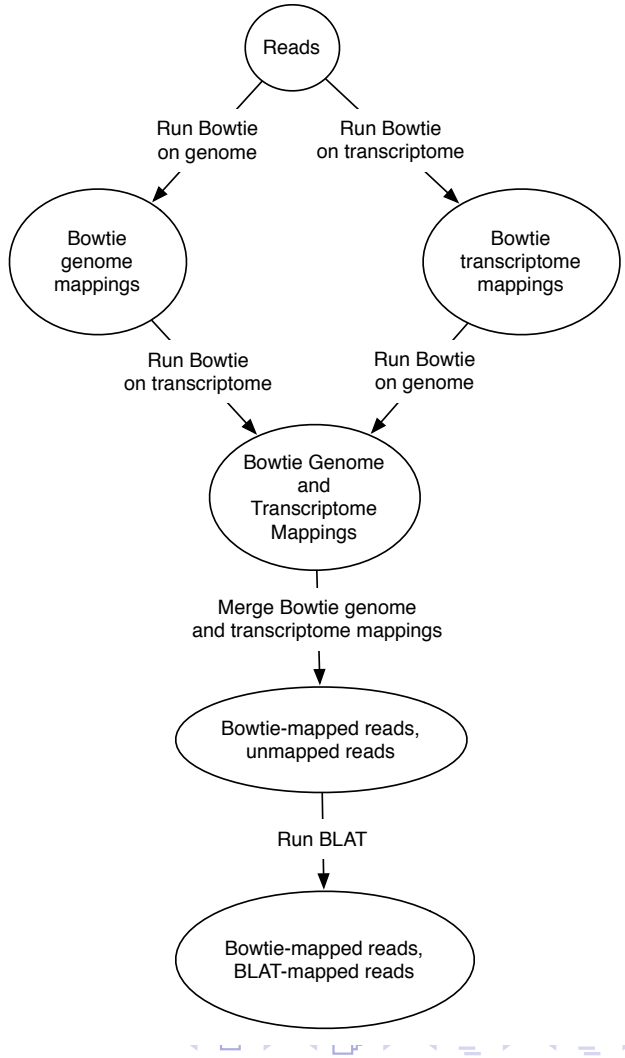
```
Postprocessing
--------------
X Merge RUM_NU files
X Make non-unique coverage
X Merge RUM_Unique files
X Compute mapping statistics
X Make unique coverage
X Finish mapping stats
X Merge SAM headers
X Concatenate SAM files
X Merge quants
  make_junctions
  Sort junctions (all, bed) by location
  Sort junctions (all, rum) by location
  Sort junctions (high-quality, bed) by location
  Get inferred internal exons
  Quantify novel exons

All the chunk error log files are empty. That's good.
Main error log file is empty. That's good.

RUM is running (job ids 815718, 815720).
```

Job state management

- Model the workflow as a state machine
- A completed step transitions the job from one state to another
- Stitch steps together into a workflow
- Determine the state of a job by looking at which output files exist
- Similar to GNU Make (but simpler)
- Basis for a lot of additional features



Recovering from errors

- In case of failure...
- RUM 2 allows easier recovery
- Running "`rum_runner align`" again, RUM will determine what state the job was in when it failed
- Just resumes at the next uncompleted step
- Can save *a lot* of time when recovering from infrastructure failure

Killing a job

- To stop a job and remove all of its output:

```
rum_runner kill -o dir
```

- Useful if you've run a job with incorrect parameters and need to start over

Work distribution

- Automatic support for one multi-core machine (will run each chunk in a separate process by default)
- Built-in support for Sun Grid Engine, with `--qsub` option
- Easily extensible for other platforms

Work distribution

For other platforms, you can either extend a Perl class to provide support, or set up some scripts to run the parts of the job on different machines:

Work distribution

For other platforms, you can either extend a Perl class to provide support, or set up some scripts to run the parts of the job on different machines:

- Run preprocessing alone:

```
rum_runner -o <dir> --preprocess
```

Work distribution

For other platforms, you can either extend a Perl class to provide support, or set up some scripts to run the parts of the job on different machines:

- Run preprocessing alone:

  ```
  rum_runner -o <dir> --preprocess
  ```

- Run chunks one at a time:

  ```
  rum_runner -o <dir> --process --chunk 8
  ```

Work distribution

For other platforms, you can either extend a Perl class to provide support, or set up some scripts to run the parts of the job on different machines:

- Run preprocessing alone:

  ```
  rum_runner -o <dir> --preprocess
  ```

- Run chunks one at a time:

  ```
  rum_runner -o <dir> --process --chunk 8
  ```

- Run postprocessing alone:

  ```
  rum_runner -o <dir> --postprocess
  ```

Agenda

Web resources

Main github page	`https://github.com/PGFI/rum`
User guide	`https://github.com/PGFI/rum/wiki`
Issues	`https://github.com/PGFI/rum/issues`
Downloads	`https://github.com/PGFI/rum/downloads`

To the command line...

Agenda

Image credits

- http://cbil.upenn.edu/RUM/rumpouring2.gif
- http://upload.wikimedia.org/wikipedia/commons/2/29/Cut_sugarcane.jpg
- http://upload.wikimedia.org/wikipedia/en/9/98/Rum_in_barrels_at_travellers_distillery.jpg
- http://theprosperityproject.blogspot.com/2010/04/if-at-first-you-dont-succeed.html
- http://muppet.wikia.com/wiki/Doozers