Firehose: An Analysis Infrastructure

Douglas Voet[1], Pei Lin[1], Lee Lichtenstein[1], Robert Zupko II[1], Gordon Saksena[1], Michael Noble[1], Kristian Cibulskis[1] , Stacy Gabriel[1], Eric Lander[1], Lynda Chin[2], Gad Getz[1]

[1]Broad Institute of MIT and Harvard, Cambridge, MA; [2]MD Anderson Cancer Center, Houston, TX, USA

Abstract

We present Firehose, an analysis infrastructure developed at The Broad Institute to coordinate the flow of terabyte-scale datasets through scores of quantitative algorithms. Implemented primarily as a Java web application, the chief aims of Firehose are reproducibility, automation, and high throughput. To achieve the first of these goals Firehose version-tracks samples, algorithms, and the logic which binds them together, storing this and related execution information in a persistent RDBMS for provenance; this makes it easy for users to identify, over arbitrarily large time spans, what has been run, what can be run, what needs to be run again if data or codes change, or what cannot be run due to unsatisfied dependencies.

Automation and throughput are facilitated in several ways: First, while Firehose is most often used interactively from a browser, it also offers an an extensive API for programmatic control of routine tasks and scalability to multiple workspaces and datasets. Next, by encapsulating data and algorithm parameters within abstract annotations, instead of only literal values or explicit file system references, Firehose is able to execute analyses in data-blind manner across a wide variety of inputs, without modification or onerous bookkeeping for end users. This has proven to be a powerful metaphor for interacting with TCGA data, for example, because once a code is in Firehose it can run on either a single tumor type or all of them with equal ease. Third, encapsulating jobs within an abstract execution engine (presently GenePattern, but support for others is under development) enables them to be transparently dispatched to a single machine or across many compute nodes, again without algorithm modification or extensive user tuning; in-depth knowledge of the underlying operating system or HPC task scheduler is not required, as entire analysis workflows for all samples of interest can be executed at the click of a button or via one API call. Finally, provenance stored for reproducibility also increases throughput by allowing new job requests that match previous requests to be completely avoided; in these cases Firehose simply returns the prior results.

Although still evolving, Firehose has become a valuable piece of The Broad Institute computing infrastructure; it is used daily by dozens in the Cancer Genome Analysis group, to perform all TCGA GDAC and GSC analyses, managing hundreds of thousands of jobs on tens of thousands of samples, spread over hundreds of compute nodes and a 400+ TB file system.

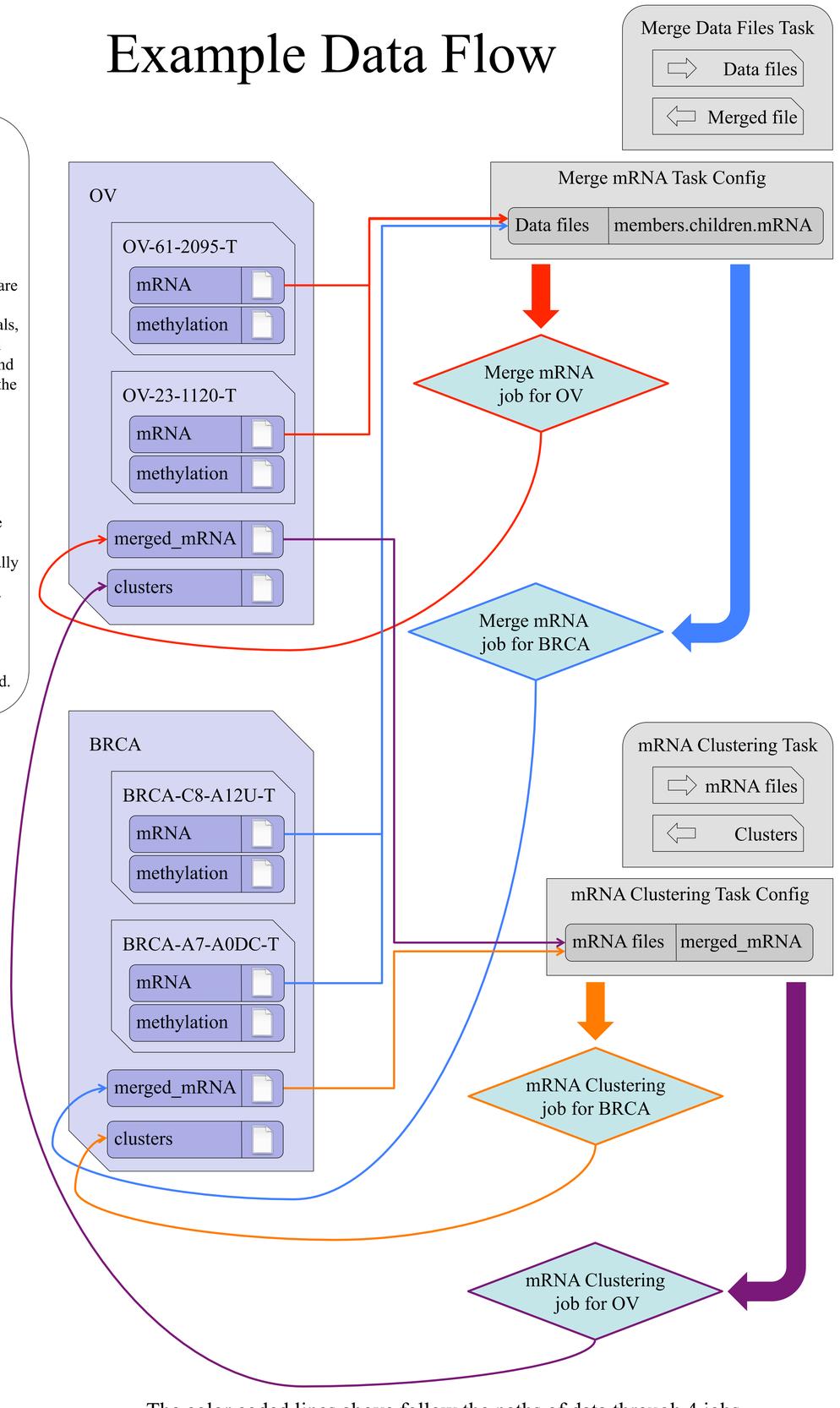
Data

Entities & Annotations

Entities are the basic unit about which information is tracked and on which jobs are run. There are 4 types of entities, Samples, Individuals, Sample Sets and Individual Sets. In the example, OV and BRCA are individual sets, the other entities are samples. Individuals are omitted for visual clarity.

Each datum that is tracked about an entity is called an annotation. Annotations are used as inputs to analyses. Annotations may be manually uploaded, populated by connectors or the results of analyses. Last change information and source are tracked. New types of annotations are easily added.

Example Data Flow



The color coded lines above follow the paths of data through 4 jobs.

Analyses

Tasks & Task Configurations

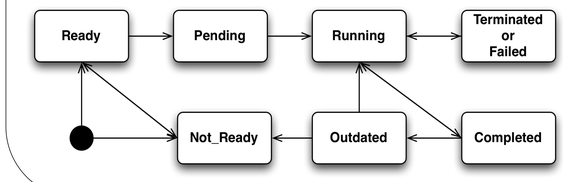
Tasks keep track of a particular version of an executable, the input parameters ⇨, and the outputs ⇦. New Tasks are easily added.

Task configurations tell Firehose
- what entity type to analyze
- how to bind annotations to parameters of analyses
- when analyses are ready to run
- what to do with outputs of analyses

Expressions
- simple: mRNA
- dependents: members.children.mRNA
- more complex via an expression language

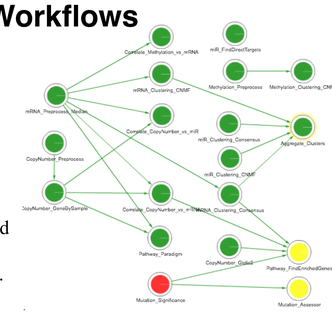
Jobs

Entities, Annotation and Tasks come together through Task Configurations to specify an analysis job. By tracking the task version and all inputs, Firehose can determine
- whether a job can be started or not
- when a completed job needs to be rerun
- when a new job request is satisfied by an existing job



Workflows

Workflows are directed acyclic graphs (DAGs) of task configurations. Automated execution of workflows ensures that analyses are run in dependency order. Pictured to the right is the Broad GDAC analysis workflow.



Architecture

Java Web Application