ScrubJay Documentation

Release 1.0

Alfredo Gimenez

Oct 28, 2018

Contents

ScrubJay is a framework for automatic and scalable data integration. Describe your input datasets (files, formats, database tables), then describe the semantics of the dataset(s) you desire, and let ScrubJay derive it for you in a consistent and reproducible way.

ScrubJay was originally developed for analyzing the supercomputing facilities at Lawrence Livermore National Laboratory, but is not specifically tied to any one kind of data.

CHAPTER 1

Install

You can use a precompiled version of ScrubJay by downloading a release jar and adding it to your classpath.

todo: Create a github release jar and link to it here.

For building ScrubJay, see *Build*.

For example usage, see *Example*.

CHAPTER 2

Build

ScrubJay uses sbt. Compile and test using the `compile` target:

```
sbt compile
```

Create a fat jar (skipping tests) using the sbt `assembly` target:

```
sbt assembly
```

CHAPTER 3

Example

We have three datasets:

1. Job queues describing what jobs ran on what nodes over what time ranges
2. Data describing which nodes reside on which racks
3. Collected FLOPs at different points in time for each node

Now, lets say we want to analyze how the FLOPs of different jobs are distributed over a rack. This requires integrating all three datasets in a non-trivial way. With ScrubJay, we specify the data columns that we want and ask ScrubJay to generate solutions containing them.

First, we load a *DataSpace* which describes the dimensions and datasets we are using.

```
val dataSpace = DataSpace.fromJsonFile("jobAnalysis.sj")
```

Then, we create a query target, which is just the schema of the dataset that we want, and use it to create a query in the created dataspace.

```
val querySchema = ScrubJaySchema(Array(
  ScrubJayField(domain = true, dimension = "rack"),
  ScrubJayField(domain = false, dimension = "flops")
))

val query = Query(dataSpace, querySchema)
```

todo: Queries may be generated using the sjql.

We find solutions to the query (there may be multiple) using:

```
val solutions = query.solutions
```

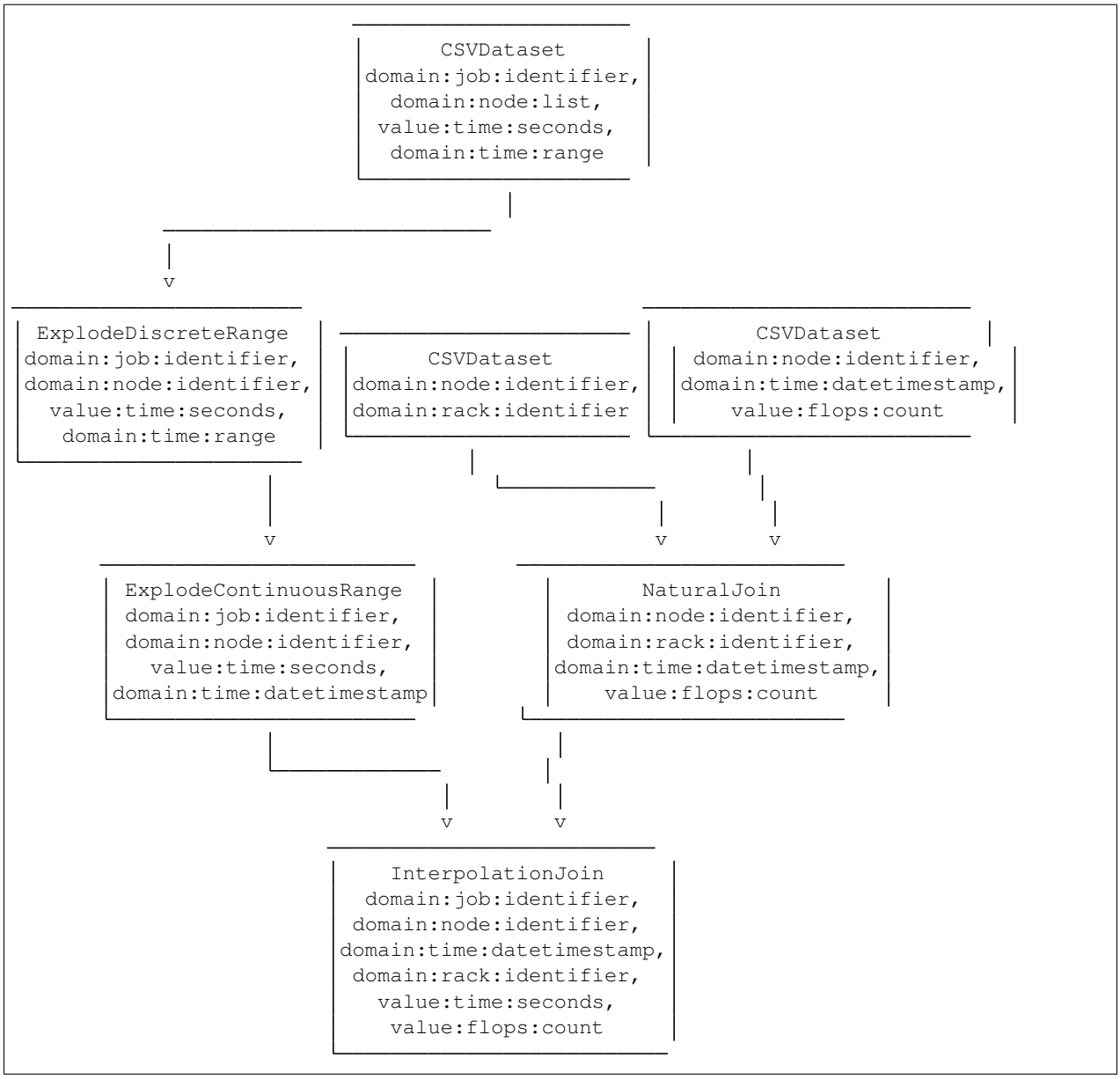
This gives us all solutions as a lazily-evaluated iterator. A solution is a *DatasetID*, which describes the resulting dataset and how to derive it. To derive a solution as a Spark DataFrame, run the `realize` function on it, specifying the dimension space used to create the query. For example, to derive the first solution:

```
val realizedDataFrame = solutions.head.realize(dataSpace.dimensions)
```

We can also see how the solution was derived in a DAG, using `toAsciiGraphString`:

```
DatasetID.toAsciiGraphString(solution)
```

which produces:



CHAPTER 4

Data Semantics

ScrubJay uses a semantic language to define how to load datasets, how to transform them, what they contain, and properties of the dimensions over which they are defined.

4.1 DataSpace

A DataSpace describes the datasets and dimensions available for ScrubJay to derive new datasets from. Every ScrubJay query must be made in the context of a DataSpace.

DataSpaces do not actually encode the data, but rather definitions called *DatasetID* instances that describe how to create the data. The dimensions of a DataSpace are defined as a single dimensionspace . DatasetID columns each have an associated dimension in the DimensionSpace, and ScrubJay uses this information to compare different columns to one another.

4.2 Domains and Values

When data is collected, some entity is being measured, and some measurement is being made. For example, a thermometer measures temperature at some time and place. We define the entity being measured (the time and place) as the *domain*, and the measurement itself (the temperature) as the *value*.

Each column in a DatasetID must be defined as either a domain or a value column. This way, ScrubJay can determine whether two different measurements are measuring the same entity.

4.3 DatasetID

A DatasetID identifies how to create data from one or more sources. An "original" DatasetID describes how to load data from a single source (e.g., a CSV file or database table), without performing any transformations, and a "derived" DatasetID describes how to derive a DatasetID from one or more original DatasetIDs and one or more transformations (e.g., join operations).

Every DatasetID contains the following fields:

type For an original DatasetID, this describes the type of source, e.g., `CSVDatasetID`. For a derived DatasetID, this describes the type of derivation, e.g., join.

sparkSchema The low-level Spark schema of the data, describing data types of each column, e.g. `int`, `array<string>`

scrubJaySchema The high-level ScrubJay schema of the data, describing the dimensions and units of columns, as well as whether each column represents a domain or a value (see *Domains and Values*).

Different types of DatasetIDs contain additional parameters. For example, a CSVDatasetID contains the path of the CSV file. Derived DatasetIDs contain parameters for transformations as well. .. _dimensionspace:

4.4 DimensionSpace

A DimensionSpace specifies a set of dimensions and their properties. Each dimension contains the fields:

name A uniquely identifying name for the dimension.

ordered Whether the dimension has an ordering, such that there is a less-than relation between two points on that dimension.

continuous Whether the dimension is continuous, i.e., there always exists another point between any two points on that dimension.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search