GMQL: GenoMetrics Query Language

Simone Pallotta

2017-10-17

Contents

1 Introduction

Improvement of sequencing technologies and data processing pipelines is rapidly providing sequencing data, with associated high-level features, of many individual genomes in multiple biological and clinical conditions. For this purpose GMQL has been proposed a high-level, declarative GenoMetric Query Language (GMQL) and a toolkit for its use.

1.1 Purpose

GMQL operations focus on genomic domain-specific operations written as simple queries with implicit iterations over thousands of heterogeneous samples, computed in few minutes over servers (Marco, Ceri, and Kaitoua 2016). This package provides a set of functions to create, manipulate and extract genomic data from different datasources both from local and remote datasets. Also, these functios allow performing complex queries without knowledge of GMQL syntax.

2 Dataset

We usually distinguish two kinds of dataset layout:
These contains large number of information describing regions of genome.
Data are encoded in human readable format using plain text file.

- GMQL standard layout:

  GMQL dataset is a collection of samples with the same region schema, is composed basically of three type of file:
  1. region files usually terminating in .gtf or .gdm

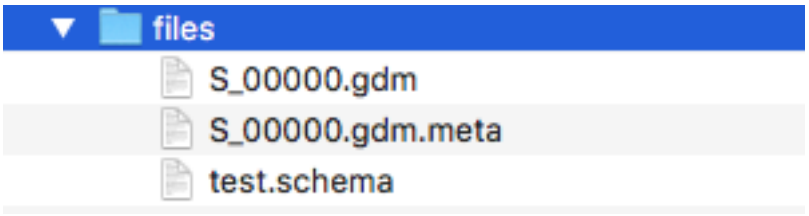

Figure 1: GMQL dataset folder

2. metadata files terminating in .meta
3. schema XML file containing regions attributes Each region sample file owns its metadata file. All these files must reside in unique folder called files.

- Generic text based dataset:

  Dataset composed by heterogeneous sample organised in simple text files probably stem from different medical, biological sytem Sample files are simply contained on a folder whose name must be specified as input on read function.

In our package dataset files are considered read-only. Once read, genomic information is represented in abstract structure inside package.

3 Genomic Data Model

The proposed Genomic Data Model (GDM) is based on the notions of datasets and samples; datasets are collections of samples, and each sample consists of two parts, the region data, which describe portions of the DNA, and the metadata, which describe sample general properties.(Marco, Ceri, and Kaitoua 2016).

4 Basic Requirements

The GMQL package requires:

- javaSE version 8
- java environment correctly set (i.e JAVA_HOME)
- scala version 2.11.8
- scala environment correctly set (i.e SCALA_HOME)
- network connectivity to web services (if required)

5 How to Install

The GMQL package can be installed by executing the following R expression.

```
source("https://bioconductor.org/biocLite.R")
biocLite("RGMQL")
```

6 Processing Environments

This package allows to create, manipulate and extract genomic data from different datasets using different processing modes both local and remote.

6.1 Local Processing

Query processing consumes computational power directly from local CPUs/system while managing datasets (both GMQL or generic text plain dataset).

6.1.1 Initialization

Load and attach the GMQL package in an R session using library function:

```
library('RGMQL')
```

Before starting using any GMQL operation we need to initialise the GMQL context with the following code:

```
initGMQL()
```

Calling initGMQL() with no parameters means we are initialising the context with GTF as output format for sample and metadata files. Details on this and all other functions are provided in the R documentation for this packag (e.g., help(GMQL)).

6.1.2 Datasource

After initialization we need to read the dataset. In the following section we show how getting data from different sources. We have four different cases:

1. Local GMQL dataset:
   As data are already in user computer, we simply execute:

```
gmql_dataset_path <- system.file("example","DATA_SET_VAR_GTF",package = "RGMQL")
data_out = readDataset("gmql_dataset_path")
```

2. Remote dataset / explicit download:
   User can download it locally using:

```
test_url <- "http://130.186.13.219/gmql-rest"
login.GMQL(test_url)
downloadDataset(test_url,"dataset_test",path = getwd())
```

where *test_url* is a R variable that define URL of remote server where web services are located.
Once local, these datatset behave like local dataset as written above.

3. Remote dataset (/ implicit download):

```
data_out = readDataset("dataset_name_on_repo")
```

There is no need to explicitally download data since execution will trigger download automatically.

4. GrangesList:
   Also, for better integration in R environment and with other packages, we provide a function to read from GrangesList, for example:

```
library("GenomicRanges")
library("RGMQL")
initGMQL()
gr1 <- GRanges(seqnames = "chr2",
ranges = IRanges(103, 106),
strand = "+",
score = 5L, GC = 0.45)

gr2 <- GRanges(seqnames = c("chr1", "chr1"),
ranges = IRanges(c(107, 113), width = 3),
strand = c("+", "-"),
score = 3:4, GC = c(0.3, 0.5))

grl <- GRangesList("txA" = gr1, "txB" = gr2)
data_out <- read(grl)
```

Every read function return a result object a value containing internal details used for executing the subsequent GMQL operation.

6.1.3 Queries

GMQL is not DDL/DML traditional query language: With "query" we intend a group of operation that together produce result; in that sense GMQL query are more similar to SQL script. GMQL programming consist of a series of select, union, project, difference (and so on...) command.

If you want to persist result, you can materialize as last step. Let's see a short example:

```
initGMQL("gtf")
test_path <- system.file("example","DATA_SET_VAR_GTF",package = "RGMQL")
input = readDataset(test_path)

## it selects from input data samples of patients younger than 70 years old,
## based on filtering on sample metadata attribute Patient_age
#s=select(input,"Patient_age < 70")

## it counts the regions in each sample and stores their number as value of the new metadata
## RegionCount attribute of the sample.
e = extend(input_data = s, list(RegionCount = COUNT()))

## materialize the result dataset on disk
#m = materialize(e)
```

6.1.4 Execution

GMQL processing does not store results: They remain in the environment until you invoke *execute* function.

```
execute()
```

execute can be issued only if at least one *materialize* is present in GMQL query, otherwise an error is generated. Data are saved in the path specified in every *materialize*. Besside *execute* we can use

```
g <- take(input_data = m, rows = 45)
```

to extract data as GRangesList format and execute all *materialize* commands. NOTE: GRangesList are contained in R environment and are not saved on disk.

rows parameter specified how many rows will be exported

6.2 Remote Processing

Query processing consumes computational power from remote clusters/system while managing datasets that are only GMQL dataset.

Remote processing exits in two flavour:

- BATCH execution:
  Similar to local execution; user read data and the system automatically upload it on remote system: once loaded you can isue R function to manage remote data.
- REST web services:
  user can write GMQL queries to be executed remotely on remote data (or local data previous upload)

6.2.1 REST web services

This package allows to invoke rest services implementing the commands specified at link.

6.2.1.1 Initialization Rest servicies required login so the first step is to perform logon using user and password or as guest. Upon succesfull logon you get a request token must use in every subsequent REST call. Login can be performed using function:

```
library("RGMQL")

test_url = "http://130.186.13.219/gmql-rest"
login.GMQL(test_url)
## [1] "your Token is 8a52eba2-09a5-469b-8271-b7f61a223aa6"
```

that saves token in Global R environment with variable named *authToken*.
wit this token you can call all the funciton in web services suite.

6.2.1.2 Execution User can write the query as in the following example, as the second parameter of *runQuery*.

```
test_url = "http://130.186.13.219/gmql-rest"
login.GMQL(test_url)
## [1] "your Token is d585ee8e-7a73-44c6-af26-59c76bac9e61"
runQuery(test_url, "query_1", "DATA_SET_VAR = SELECT() HG19_TCGA_dnaseq;
         MATERIALIZE DATA_SET_VAR INTO RESULT_DS;", output_gtf = FALSE)
## $id
## [1] "job_query_1_guest_new685_20171017_123807"
##
## $status
## [1] "PENDING"
##
## $message
## [1] ""
##
## $datasets
## $datasets[[1]]
```

```
## $datasets[[1]]$name
## [1] "job_query_1_guest_new685_20171017_123807_RESULT_DS"
```

Once run, query continues on the server while *runQuery* returns immediately. User can extract from result the job_id and status. jod_id can be used to invoke log and trace calls both in this R package.

6.2.2 Batch execution

This function is similar to local processing (syntax, function and so on . . .) except: 1. if data is local is uploaded on repository implicitly 2. materialized data only on repository

References

Marco, Masseroli, Stefano Ceri, and Abdulrahman Kaitoua. 2016. "Data Management for Heterogeneous Genomic Datasets." http://ieeexplore.ieee.org/document/7484654/.