MABT – MultiAssay big table experiments

Vincent J. Carey, stvjc at channing.harvard.edu

Jan 2017

Contents

1 Introduction and authentication

It is challenging to authenticate in a non-interactive setting. All chunks here are eval=FALSE in the build package, but a PDF will be made available with a completed run.

Here's a setup.

```
suppressPackageStartupMessages({
library(GoogleGenomics)
})
apik = Sys.getenv("GOOGLE_API_KEY")
authenticate(apiKey=apik)
## Configured public API key.
getBQ2 = function() {
suppressPackageStartupMessages({
library(dplyr)
library(bigrquery)
})
my_billing = "cgc-05-0009" # replace billing info here with your own
src_bigquery("cgc-05-0009", "yriMulti", billing = my_billing)
}
```

```
suppressPackageStartupMessages({
library(MABT)
})
bq = getBQ2()
bq
## src:  bigquery [cgc-05-0009:yriMulti]
## tbls: banovichSE_expressionData, banovichSE_rowRanges, geuFPKM_colData,
##   geuFPKM_expressionData, geuFPKM_rowRanges
bano =
  RangedBT("cgc-05-0009", bq,
   "banovichSE_expressionData",
    "banovichSE_rowRanges")
bano
## rangedBT instance.
## assay colnames include:
##     cg_Methyly450 NA18498 ... NA18489 NA18909
## rangeData colnames include:
##     seqnames start ... probeEnd probeTarget
nrow(bano)
## [1] 329469
```

2 Considerations of the code base in 0.0.0

The current objective is to get some performance metrics on genomic computing with BigTable/BigQuery. We are not using the MultiAssayExperiment framework yet, as there is considerable detailed programming required to create objects compliant with the validating API.

2.1 RangedBT class

We have defined an S4 class RangedBT, to wrap instances of `tbl_bigquery`. Two such tables are assumed present, one for `assay` and one for `rangeData`. We also retain information on the Google Compute Platform (GCP) project name and the number of rows (implicitly equal for both tables).

```
getClass("RangedBT")
## Class "RangedBT" [package "MABT"]
##
## Slots:
##
## Name:         assay    rangeData      project  cached_nrow
## Class: tbl_bigquery tbl_bigquery    character    intOrNULL
```

The constructor is very simple and does not validate or initialize, though eventually it should. It takes as input the project name, the `src_bigtable` instance, and the table names in the src.

```
RangedBT
## function (project, src, assayname, rangename)
## {
##     new("RangedBT", assay = src %>% tbl(assayname), rangeData = src %>%
##         tbl(rangename), project = project, cached_nrow = NULL)
## }
## <environment: namespace:MABT>
methods(class="RangedBT")
## [1] assay     nrow      project   rowRanges show
## see '?methods' for accessing help and source code
```

The `nrow` method is of some interest.

```
getMethod("nrow", "RangedBT")
## Method Definition:
##
## function (x)
## {
##     if (!is.null(x@cached_nrow))
##         return(x@cached_nrow)
##     ans <- query_exec(query = paste("select count(*) from", assayname(x),
##         collapse = " "), project = project(x), default_dataset = paste(c(project(x),
##         datasetname(x)), collapse = ":"))
##     ans = as.integer(unlist(ans))
##     ans
## }
## <environment: namespace:MABT>
##
## Signatures:
##         x
## target  "RangedBT"
## defined "RangedBT"
```

2.2 Next steps

- define a “LabeledBT” class that can manage the sample data, and will answer ‘sampleNames’, ‘subsetBySamples’, ‘subsetByColumns’
- define subsetByRows, subsetByRanges for RangedBT
- establish a tiling of a chromosome using GenomicRanges::tileGenome and get performance on traversal of a chromosome, compare to Bioconductor with yriMulti package