Using the Streamer classes to count genomic overlaps with `summarizeOverlaps`

Nishant Gopalakrishnan, Martin Morgan

April 23, 2023

1 Introduction

This vignette illustrates how users can make use of the functionality provided by the *Producer*, *Consumer* and *Stream* classes in the *Streamer* package to process data in a streaming fashion. The users have the option of quickly being able to create their own class to stream process data by inheriting from the classes provided by the *Streamer* package.

This example illustrates a simple *BamInput* class that inherits from the *Producer* class and a *CountGOverlap* class that inherits from the *Consumer* class. These classes allows us to count the number of hits in a BAM file corresponding to the ranges specified by the user and return the hits in a streaming manner on a per sequence basis. Finally, the results for each sequence is collated and reordered using a helper function so they appear in the same order as the ranges provided by the user. The classes that we are going to develop in this example make use of the reference class system available in R.

We first load the *GenomicAlignments* and *Streamer* packages.

```
> library(GenomicAlignments)
> library(Streamer)
```

2 *BAMInput* class

The *BAMInput* class will be used to read gapped alignments from a file specified by the user in a streaming manner. i.e reads will be read one sequence at a time.

The two inputs specified by the user are

- file: a character string specifying the file from which alignments are to be read.
- ranges: the ranges from which alingments are to be

Like the design of the other classes in the *Streamer* package, the *BamInput* class will have an `initialize` and a `yield` method. The `initialize` method will be used to initialize the fields of the *BamInput* class and is called automatically when objects are instantiated from this class.

The `yield` method does not take any inputs. Each call to the `yield` method returns a *GAlignments* object for a single sequence within the ranges specified by the user until all the sequences have been read from the BAM file at which point, an empty *GAlignments* object will be returned.

```
> .BamInput <-
+     setRefClass("BamInput",
+     contains="Producer",
+     fields=list(
+         file="character",
+         ranges="GRanges",
+         .seqNames="character"))
> .BamInput$methods(
+     yield=function()
+     {
+        "yield data from .bam file"
+        if (verbose) msg("BamInput$.yield()")
+        if(length(.self$.seqNames))
+        {
+             seq <- .self$.seqNames[1]
+             .self$.seqNames <- .self$.seqNames[-1]
+             idx <- as.character(seqnames(.self$ranges)) == seq
+             param <- ScanBamParam(which=.self$ranges[idx],
+                                   what=character())
+             aln <- readGAlignments(.self$file, param=param)
+             seqlevels(aln) <- seq
+        } else {
+            aln <- GAlignments()
+        }
+        list(aln)
+     })
>
```

The constructor for the *BamInput* class takes the file and ranges as input and returns and instance of the *BamInput* class.

```
> BamInput <- function(file, ranges,...)
+ {
+     .seqNames <- names(scanBamHeader(file)[[1]]$target)
+     .BamInput$new(file=file, ranges=ranges, .seqNames=.seqNames, ...)
+ }
>
```

3 *CountGOverlap* class

The second class we are going to develop is a *Consumer* class that processes the data obtained from the *BamInput* class. The class calls the `summarizeOverlaps`

method with the *GAlignments* object, user supplied ranges and additional arguments to control the behaviour of the `summarizeOverlaps` method.

The *CountGOverlap* class has an `initialize` method and a `yield` method. The `initialize` method initializes the class with the options to be passed in to the `countGenomicOverlaps` method as well as some variables for keeping track of the order of the hits to be returned by the *CountGOverlap* class.

The `yield` method returns a *DataFrame* with the number of hits. The rownames of the result returned correspond to the order of the results in the original ranges supplied by the user. (These are subsequently used to reorder the results for the hits after collating results for all the sequences)

```
> .CountGOverlap <-
+     setRefClass("CountGOverlap",
+     contains="Consumer",
+     fields=list(ranges="GRanges",
+                 mode="character",
+                 ignore.strand="logical"))
> .CountGOverlap$methods(
+     yield=function()
+     {
+         "return number of hits"
+         if (verbose) msg(".CountGOt$yield()")
+         aln <- callSuper()[[1]]
+         df <- DataFrame(hits=numeric(0))
+         if(length(aln))
+         {
+             idx <- as.character(seqnames(.self$ranges)) == levels(rname(aln))
+             which <- .self$ranges[idx]
+             olap <- summarizeOverlaps(which, aln, mode=.self$mode,
+                                       ignore.strand=.self$ignore.strand)
+             df <- as(assays(olap)[[1]], "DataFrame")
+             dimnames(df) <- list(rownames(olap), seqlevels(aln))
+         }
+         df
+     })
> CountGOverlap <-
+     function(ranges,
+              mode = c("Union", "IntersectionStrict",
+                "IntersectionNotEmpty"),
+              ignore.strand = FALSE, ...)
+ {
+     values(ranges)$pos <- seq_len(length(ranges))
+     .CountGOverlap$new(ranges=ranges, mode=mode,
+                        ignore.strand=ignore.strand, ...)
+ }
>
```

4 Stream with *BamInput* and *CountGOverlap*

Instances of the *BamInput* and *CountGOverlap* classes can be created using their respective constructors and can subsequently be hooked up to form a stream using the `Stream` function provided by the *Streamer* package. For our example we shall make use of a BAM file available in the *Rsamtools* package and create a *GenomicRanges* object for the ranges that we are interested. A *Stream* can then be created by passing these objects as the arguments to the `Stream` function.

A call to the `yield` function of the *Stream* class will yield the results obtained by calling yield first on the *BamInput* class and subsequently on the *CountGOverlap* class for the first sequence in the ranges provided.

```
> galn_file <- system.file("extdata", "ex1.bam", package="Rsamtools")
> gr <-
+     GRanges(seqnames =
+             Rle(c("seq2", "seq2", "seq2", "seq1"), c(1, 3, 2, 4)),
+             ranges = IRanges(rep(10,1), width = 1:10,
+               names = head(letters,10)),
+             strand = Rle(strand(rep("+", 5)), c(1, 2, 2, 3, 2)),
+             score = 1:10,
+             GC = seq(1, 0, length=10))
> bam <- BamInput(file = galn_file, ranges = gr)
> olap  <- CountGOverlap(ranges=gr, mode="IntersectionNotEmpty")
> s <- Stream(bam, olap)
> yield(s)

DataFrame with 4 rows and 1 column
       seq1
  <integer>
g         0
h         0
i         0
j        32

>
```

5 Collate results

Each call to the `yield` function of the stream process data for one sequence. It would be convenient to have a function that processed data for all the sequences in the ranges provided and collated the results so that they are ordered correctly. (same order as the ranges provided). We proceed to create this helper `overlapCounter` function that takes a *BAMInput* and *CountGOverlap* class objects as inputs.

```
> overlapCounter <- function(pr, cs) {
+     s <- Stream(pr, cs)
```

```
+     len <- length(levels(seqnames(pr$ranges)))
+     lst <- vector("list", len)
+     for(i in 1:len) {
+         lst[[i]] <- yield(s)
+         names(lst[[i]]) <- "Count"
+     }
+     do.call(rbind, lst)[names(cs$ranges), ,drop=FALSE ]
+ }
> bam <- BamInput(file = galn_file, ranges = gr)
> olap  <- CountGOverlap(ranges=gr, mode="IntersectionNotEmpty")
> overlapCounter(bam, olap)

DataFrame with 10 rows and 1 column
      Count
  <integer>
a         0
b         0
c         0
d         0
e         0
f        87
g         0
h         0
i         0
j        32
```