clustermq example

Michael Mayer

2023-06-21

Introduction

`clustermq` is an R package that uses the zeromq protocol for efficient inter-node/process communication. In contrast to `batchtools` that uses a disk-based registry, `clustermq` runs everything in-memory and hence has much better scalability.

Simple example

Using the `palmerpenguins` dataset we want to run some `glm`

```
compute <- function(n) {

  library(palmerpenguins)

  # Our dataset
  x <- as.data.frame(penguins[c(4, 1)])

  ind <- sample(344, 344, replace = TRUE)
  result1 <-
    glm(x[ind, 2] ~ x[ind, 1], family = binomial(logit))
  coefficients(result1)
}
```

Using this `compute()` function, we now simply can run

```
compute(1)
```

```
## (Intercept)   x[ind, 1]
##  12.8465988  -0.7036583
```

If we now want to run the same compute function say a 100 times, we can use the `sapply` function

```
res<-sapply(1:100,compute)
```

Let's run this function a 10, 100 and 1000 times and measure the compute time

```
library(microbenchmark)
microbenchmark(
  sapply(1:10,compute),
  sapply(1:100,compute),
  sapply(1:1000,compute),
  times=10)
```

```
## Unit: milliseconds
##                      expr        min         lq       mean     median         uq
##    sapply(1:10, compute)   18.15014   18.51515   19.47905   18.76681   18.94459
##   sapply(1:100, compute)  193.11657  194.36330  208.46590  197.86827  203.32704
```

```
##  sapply(1:1000, compute) 1969.63796 1982.80374 2017.84963 2012.77071 2031.35082
##        max neval
##    26.5317    10
##   297.3029    10
##  2106.9649    10
```

Enter clustermq

Now let's do the same with `clustermq`.

```
library(clustermq)
```

```
## * Option 'clustermq.scheduler' not set, defaulting to 'SLURM'
```

```
## --- see: https://mschubert.github.io/clustermq/articles/userguide.html#configuration
```

```
system.time(res<-Q(compute, n=1:100, n_jobs=1))
```

```
## Submitting 1 worker jobs (ID: cmq9235) ...
```

```
## Running 100 calculations (0 objs/0 Mb common; 1 calls/chunk) ...
```

```
## [-------------------------------------------------]   0% (1/1 wrk) eta:  ?s[>--------------------
```

```
##    user  system elapsed
##   0.374   0.034   3.240
```

Scaling up

```
microbenchmark(
  res<-Q(compute, n=1:1000, verbose=FALSE, n_jobs=1, chunk_size=10),
  res<-Q(compute, n=1:1000, verbose=FALSE, n_jobs=2, chunk_size=10),
  res<-Q(compute, n=1:1000, verbose=FALSE, n_jobs=4, chunk_size=10),
  res<-Q(compute, n=1:1000, verbose=FALSE, n_jobs=8, chunk_size=10),
  times=10
)
```

```
## Unit: seconds
##                                                                                 expr
##  res <- Q(compute, n = 1:1000, verbose = FALSE, n_jobs = 1, chunk_size = 10)
##  res <- Q(compute, n = 1:1000, verbose = FALSE, n_jobs = 2, chunk_size = 10)
##  res <- Q(compute, n = 1:1000, verbose = FALSE, n_jobs = 4, chunk_size = 10)
##  res <- Q(compute, n = 1:1000, verbose = FALSE, n_jobs = 8, chunk_size = 10)
##       min       lq     mean   median       uq      max neval
##  5.229201 5.331659 6.037618 5.581053 5.990311 8.135010    10
##  4.531817 4.736806 5.152440 5.054040 5.117577 6.448664    10
##  3.236773 3.659275 4.325724 3.948953 4.352783 6.494752    10
##  3.435474 3.743023 4.159678 4.027089 4.194330 6.122911    10
```

Note the use of `chunk_size` above that is used to chunk individual tasks together.

`foreach` loops

Now let's run the same thing via `foreach` loops

```
computeforeach_cmq <- function(samples, tasks) {
  library(foreach)
  library(palmerpenguins)
```

```
  # Register parallel backend to foreach
  register_dopar_cmq(
    n_jobs = tasks,
    log_worker = FALSE,
    verbose=FALSE,
    chunk_size=10
  )

  # Our dataset
  x <- as.data.frame(penguins[c(4, 1)])

  # Number of samples to simulate
  samples <- samples

  # Main loop
  foreach(i = 1:samples, .combine = rbind) %dopar% {
    ind <- sample(344, 344, replace = TRUE)
    result1 <-
      glm(x[ind, 2] ~ x[ind, 1], family = binomial(logit))
    coefficients(result1)
  }
}
```

Now, let's test this `computeforeach_cmq` function for a couple of scenarious

```
library(clustermq)
library(microbenchmark)
microbenchmark(
  computeforeach_cmq(1000,1),
  computeforeach_cmq(1000,2),
  computeforeach_cmq(1000,4),
  computeforeach_cmq(1000,8),
  times=10
)
```

```
## Unit: seconds
##                          expr      min       lq       mean   median       uq
##  computeforeach_cmq(1000, 1) 5.112868 5.519198   6.172361 5.696144 6.856463
##  computeforeach_cmq(1000, 2) 4.460139 4.494660 365.426522 5.193556 7.191374
##  computeforeach_cmq(1000, 4) 3.489098 3.800440   4.712625 4.124666 6.212004
##  computeforeach_cmq(1000, 8) 3.426540 3.969863   4.934567 4.455529 6.246246
##          max neval
##     8.239180    10
##  3604.555340    10
##     6.540094    10
##     6.782887    10
```

doFuture and future.batchtools

While there unfortunately is not yet a fully functional `future.clustermq` package, we have to make do with `future.batchtools` for the moment

```
library(doFuture)
```

```
## Loading required package: future
```

```
##
## Attaching package: 'future'
```

```
## The following object is masked from 'package:rmarkdown':
##
##     run
```

```
library(doRNG)
```

```
## Loading required package: rngtools
```

```
registerDoFuture()
library(future.batchtools)
```

```
## Loading required package: parallelly
```

Let's retry our `computeforeach` function again, but without the clustermq backend

```
computeforeach_future <- function(samples, tasks) {
  library(foreach)
  library(palmerpenguins)

  # Let's plan to have a maximum of tasks workers
  plan(batchtools_slurm,workers=tasks)

  # Our dataset
  x <- as.data.frame(penguins[c(4, 1)])

  # Number of samples to simulate
  samples <- samples

  # Main loop
  foreach(i = 1:samples, .combine = rbind,.options.future = list(chunk.size = 10)) %dorng% {
    ind <- sample(344, 344, replace = TRUE)
    result1 <-
      glm(x[ind, 2] ~ x[ind, 1], family = binomial(logit))
    coefficients(result1)
  }
}
```

Now, let's test this `computeforeach_future` function for a couple of scenarious

```
library(clustermq)
library(microbenchmark)
microbenchmark(
  computeforeach_future(1000,1),
  computeforeach_future(1000,2),
  computeforeach_future(1000,4),
  computeforeach_future(1000,8),
  times=10
)
```