



PARALLEL PROGRAMMING IN R

Package foreach

Hana Sevcikova

University of Washington



What is foreach for?

- Developed by Rich Calaway and Steve Weston.
- Provides a new looping construct for repeated execution.
- Supports running loops in parallel.
- Unified interface for sequential and parallel processing.
- Greatly suited for embarrassingly parallel applications.



foreach looping construct

```
foreach(...) %do% ...
```

```
library(foreach)
```

```
foreach(n = rep(5, 3)) %do% rnorm(n)
```

```
[[1]]  
[1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078
```

```
[[2]]  
[1] -0.8204684  0.4874291  0.7383247  0.5757814 -0.3053884
```

```
[[3]]  
[1]  1.5117812  0.3898432 -0.6212406 -2.2146999  1.1249309
```

Iteration variables

```
foreach(n = rep(5, 3), m = 10^(0:2)) %do% rnorm(n, mean = m)
```

```
[[1]]  
[1] 0.3735462 1.1836433 0.1643714 2.5952808 1.3295078  
  
[[2]]  
[1] 9.179532 10.487429 10.738325 10.575781 9.694612  
  
[[3]]  
[1] 101.51178 100.38984 99.37876 97.78530 101.12493
```



Combining results

```
foreach(n = rep(5, 3), .combine = rbind) %do% rnorm(n)
```

```
      [,1]      [,2]      [,3]      [,4]      [,5]  
result.1 -0.6264538 0.1836433 -0.8356286 1.5952808 0.3295078  
result.2 -0.8204684 0.4874291 0.7383247 0.5757814 -0.3053884  
result.3 1.5117812 0.3898432 -0.6212406 -2.2146999 1.1249309
```

```
foreach(n = rep(5, 3), .combine = '+') %do% rnorm(n)
```

```
[1] 0.06485897 1.06091561 -0.71854449 -0.04363773 1.14905030
```



List comprehension

```
foreach(x = sample(1:1000, 10), .combine = c) %:%  
  when(x %% 3 == 0 || x %% 5 == 0) %do% x
```

```
[1] 372 906 201 894 940 657 625
```



PARALLEL PROGRAMMING IN R

Let's practice!



PARALLEL PROGRAMMING IN R

foreach & parallel backends

Hana Sevcikova

University of Washington



Popular backends

- **doParallel** (`parallel`)
- **doFuture** (`future`)
- **doSEQ** (for consistent sequential interface)

Package doParallel (Rich Calaway et al.)

- Interface between **foreach** and **parallel**
- Must register via `registerDoParallel()` with cluster info
- Quick registration:

```
library(doParallel)
registerDoParallel(cores = 3)
```

- using *multicore* functionality for Unix-like systems (fork)
- using *snow* functionality for Windows systems



Package doParallel (Rich Calaway et al.)

- Register by passing a cluster object:

```
library(doParallel)
cl <- makeCluster(3)
registerDoParallel(cl)
```

- will use *snow* functionality



Using doParallel

Sequential:

```
library(foreach)
foreach(n = rep(5, 3)) %do% rnorm(n)
```

Parallel:

```
library(doParallel)
cl <- makeCluster(3)
registerDoParallel(cl)

foreach(n = rep(5, 3)) %dopar% rnorm(n)

[[1]]
[1] -1.16719198 -0.03600075 -0.59728324  1.03807353 -0.05085617

[[2]]
[1]  0.3700061 -0.4193585  0.1311767  0.6566272 -0.0371627

[[3]]
[1]  0.9872227 -1.1697387  0.3992779 -0.1556074 -1.0345713
```



Package doFuture (Henrik Bengtsson)

- On top of the **future** package
- How to **plan** the future:
 - sequential
 - cluster
 - multicore
 - multiprocess
- `future.batchtools`: run processes on HPC clusters (Torque, Slurm, SGE etc.)



Using doFuture

```
library(doFuture)
registerDoFuture()
```

Cluster plan:

```
plan(cluster, workers = 3)

foreach(n = rep(5, 3)) %dopar% rnorm(n)
```



Using doFuture

Multicore plan:

```
plan(multicore)  
foreach(n = rep(5, 3)) %dopar% rnorm(n)
```



PARALLEL PROGRAMMING IN R

Let's practice!



PARALLEL PROGRAMMING IN R

Packages future and future.apply

Hana Sevcikova

University of Washington



Package future

- Developed by Henrik Bengtsson (now also funded by R Consortium)
- Uniform way to evaluate R expressions asynchronously
- Provides a unified API for sequential and parallel processing of R expressions
- Processing via a construct called **future**
- An abstraction for a value that may be available at some point in the future

What is a future?

Example in plain R:

```
x <- mean(rnorm(n, 0, 1))  
y <- mean(rnorm(n, 10, 5))  
print(c(x, y))
```

Via implicit futures:

```
x %<-% mean(rnorm(n, 0, 1))  
y %<-% mean(rnorm(n, 10, 5))  
print(c(x, y))
```

Via explicit futures:

```
x <- future(mean(rnorm(n, 0, 1)))  
y <- future(mean(rnorm(n, 10, 5)))  
print(c(value(x), value(y)))
```



Sequential and parallel futures

Sequential:

```
plan(sequential)

x %<-% mean(rnorm(n, 0, 1))
y %<-% mean(rnorm(n, 10, 5))
print(c(x, y))
```

Parallel:

```
plan(multicore)

x %<-% mean(rnorm(n, 0, 1))
y %<-% mean(rnorm(n, 10, 5))
print(c(x, y))
```



Package future.apply

- Developed by Henrik Bengtsson
- Provide parallel API for all the apply functions in base R using futures
- *Sibling* to `foreach`
- **Functions:** `future_lapply()`, `future_sapply()`, `future_apply()`, ...



Example of future.apply

Using `lapply()`:

```
lapply(1:10, rnorm)
```

Using `future_lapply()` sequentially:

```
plan(sequential)  
future_lapply(1:10, rnorm)
```

Using `future_lapply()` on a cluster:

```
plan(cluster, workers = 4)  
future_lapply(1:10, rnorm)
```



my_cool_R_package

```
cool_function <- function(...) {
```

```
...
```

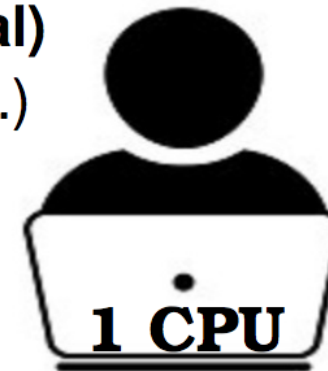
```
  future_lapply()
```

```
...
```

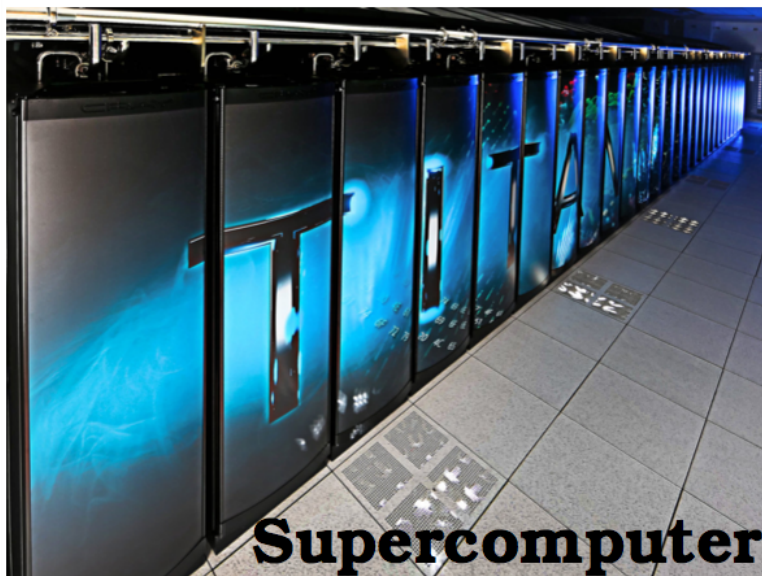
```
}
```



```
plan(sequential)  
cool_function(...)
```

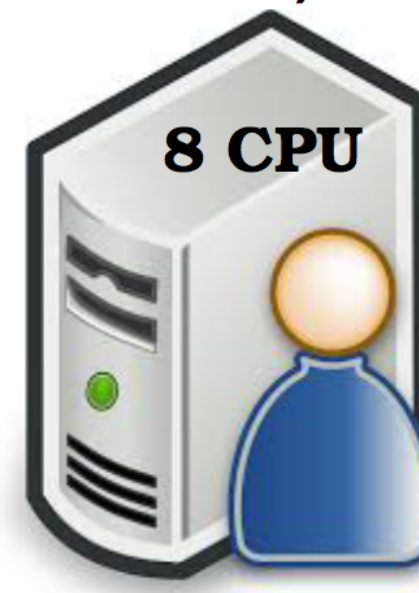


```
plan(cluster, workers = 10000)  
cool_function(...)
```



Supercomputer

```
plan(multicore, workers = 8)  
cool_function(...)
```





PARALLEL PROGRAMMING IN R

Let's practice!



PARALLEL PROGRAMMING IN R

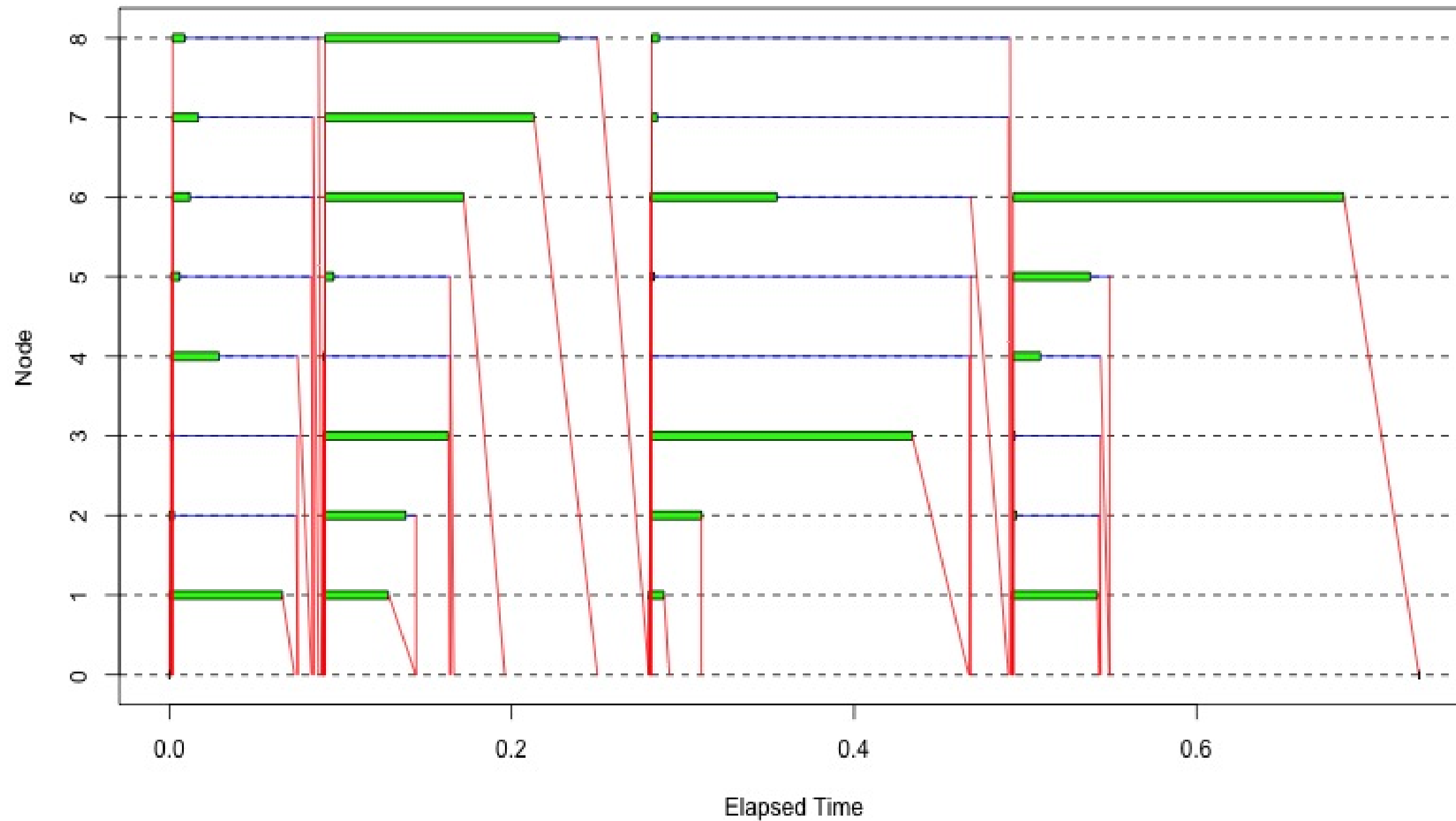
Scheduling and Load Balancing

Hana Sevcikova

University of Washington

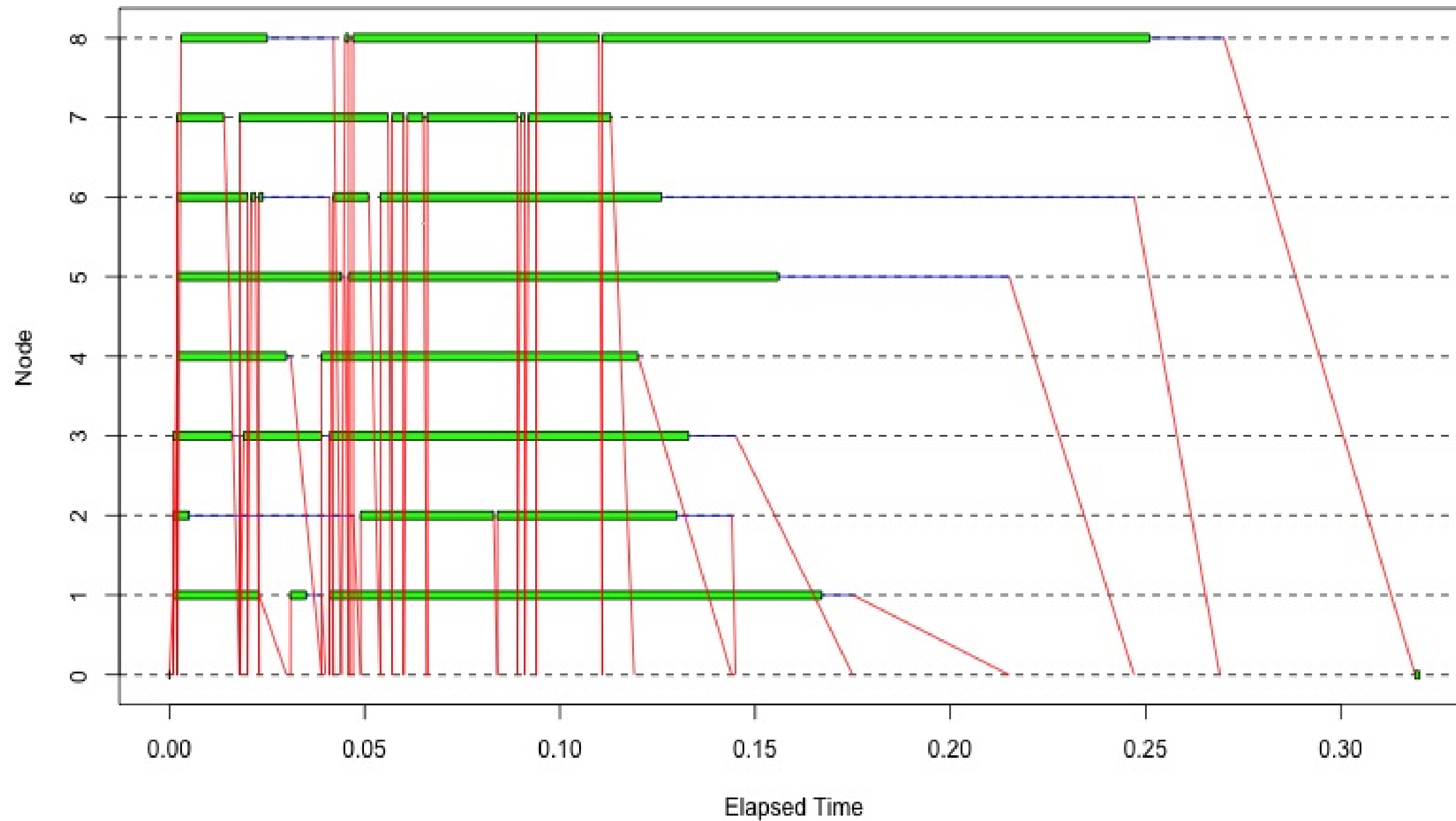


Usage with clusterApply



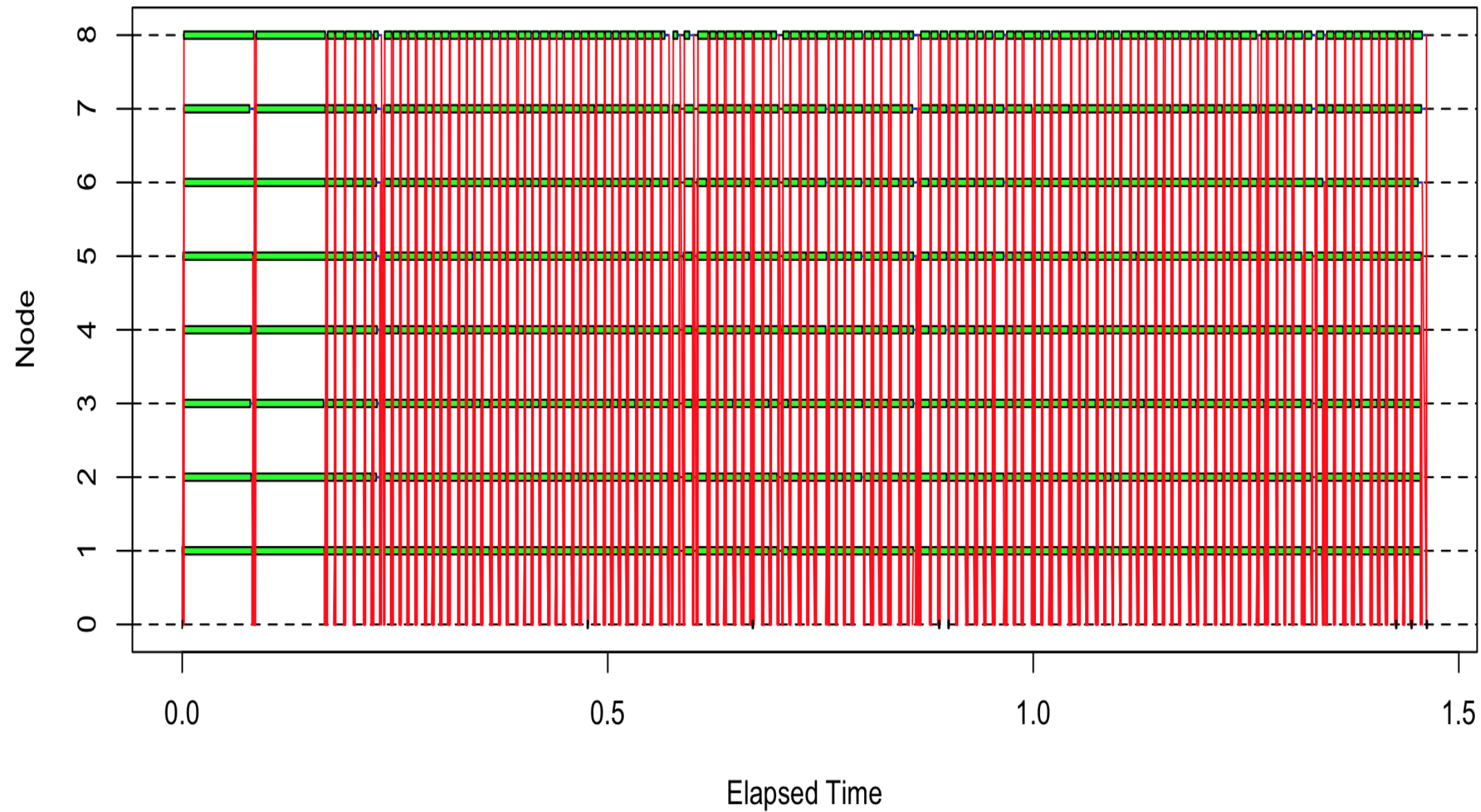


Usage with clusterApplyLB



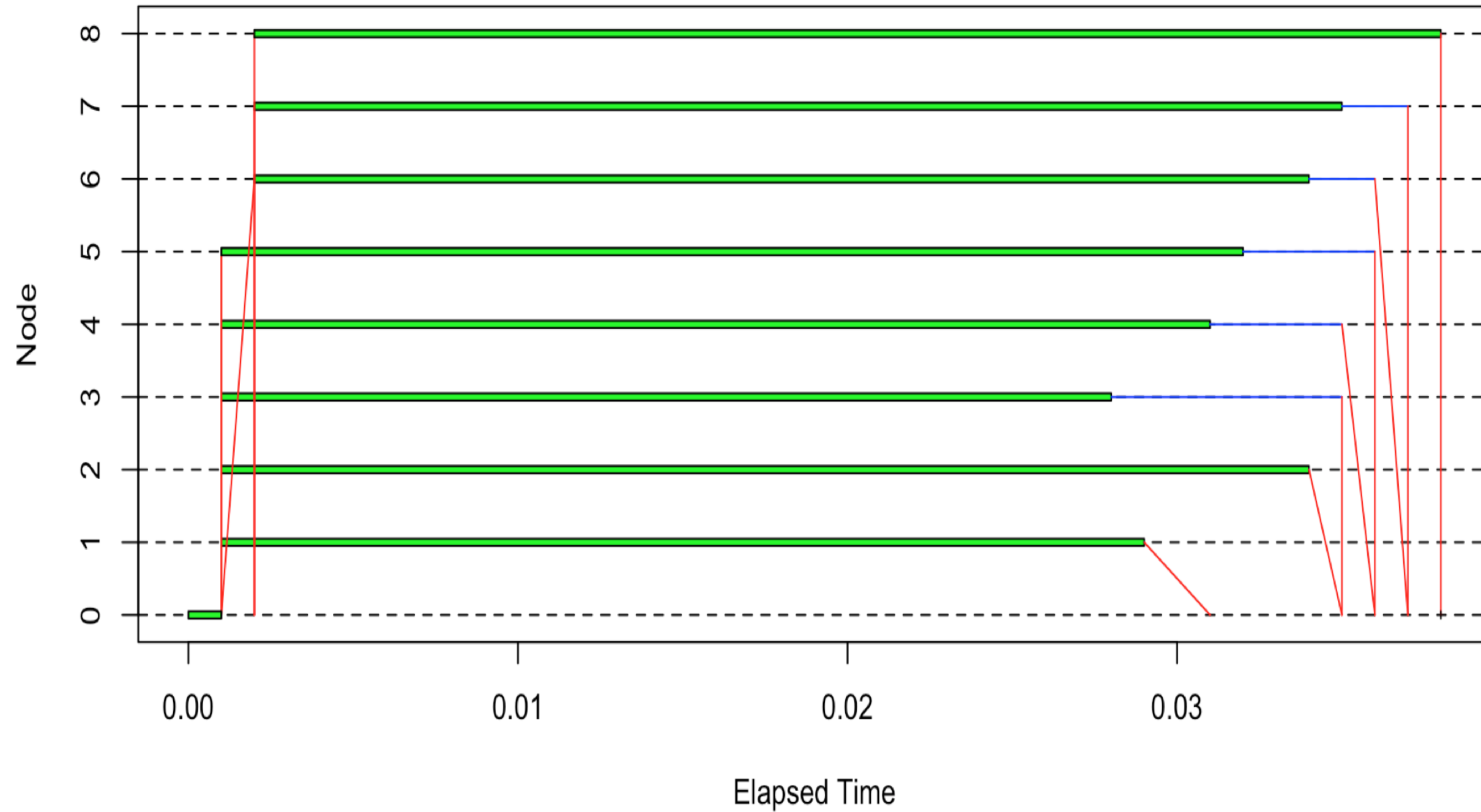


1000 tasks





8 chunks with 1000 tasks



How to chunk in parallel?

Group 10 tasks into 2 chunks using the `parallel` package:

```
splitIndices(10, 2)
```

```
[[1]]  
[1] 1 2 3 4 5
```

```
[[2]]  
[1] 6 7 8 9 10
```

```
clusterApply(cl, x = splitIndices(10, 2), fun = sapply, "*", 100)
```

```
[[1]]  
[1] 100 200 300 400 500
```

```
[[2]]  
[1] 600 700 800 900 1000
```

Built into functions `parApply()` and friends (arg. `chunk.size` for R \geq 3.5)



How to chunk in foreach and future.apply?

For `foreach`, use functions from the `itertools` package, e.g.:

```
foreach(s = isplitVector(1:10, chunks = 2)) %dopar% sapply(s, "*", 100)
```

For `future.apply`, use argument `future.scheduling`, e.g.

- one chunk per worker (default):

```
future_sapply(1:10, `*`, 100, future.scheduling = 1)
```

- one chunk per task:

```
future_sapply(1:10, `*`, 100, future.scheduling = FALSE)
```



PARALLEL PROGRAMMING IN R

Let's practice!