Práctica 6: Sistema multiagente

En la práctica de sistemas multiagentes se presenta la simulación del progreso de una enfermedad infecciosa en una población de agentes `n`, los cuales pueden ser contagiados sólo se encuentran dentro de un umbral de radio `r` y cuya probabilidad de infección `p` sea menor a la probabilidad de infectarse `pi`. La probabilidad `p` se calcula matemáticamente con la ecuación uno.

$$p = \begin{cases} 0, & Si\ d(i,j) \geq r \\ \frac{r-d}{r} & Si\ d(i,j) < r \end{cases} \qquad \text{(Ecuación1)}$$

Una vez infectados pueden recuperase en función a la probabilidad de recuperación `pr`. Esta simulación se lleva a cabo en un tiempo `Tmax` con valor de cien pasos, dentro de los cuales se muestra el desarrollo de esta enfermedad.

Meta

Evaluar en qué partes del código es posible realizar la ejecución del trabajo usando el paralelismo, y posteriormente, paralelizar el código de un sistema multiagente con el paquete *`parallel`* o *`doParallel`.*

Desarrollo del código

Con el objetivo de simplificar los códigos utilizados dentro de la práctica, se enumeran los tres archivos necesarios para ejecutar el código principal *'Cod_CompT.R'* junto con una explicación breve de las modificaciones realizadas. Estos códigos se encuentran en la esta liga.

1. Archivo *'Cod_CompT.R'*

El código '*Cod_CompT.R'* compara los tiempos entre el código original *'Cod_P6'* [1] y el código paralelizado *'Cod_P6_parSapply', además* se añade la variable de número de agentes *n* con el objetivo de observar un cambio notable dentro del

tiempo de ambos códigos. Incluye un gráfico tipo *boxplot* que muestra los cambios en función del código utilizado y el número de agentes.

```
> ciclos<-10
> resultados<-data.frame()
>
> for (n in seq (50,95,15)){
+ for (replicas in 1:ciclos){
+
+   source('~/GitHub/Simulacion/Simulacion/P6/Cod_P6_parSapply.R')
+
+   parS<-cbind("paralelo",t,n)
+
+   source('~/GitHub/Simulacion/Simulacion/P6/Cod_P6.R')
+
+   orginal<-cbind("original",t,n)
+
+   resultados<-rbind(resultados,parS,orginal)
+ }
+ }
>library(ggplot2)
> colnames(resultados)<-c("Tipo","Tiempo","Agentes")
> resultados$Tipo <- as.factor(resultados$Tipo)
> resultados$Agentes <- as.factor(resultados$Agentes)
> resultados$Tiempo<-as.numeric(levels(resultados$Tiempo))[resultados$Tie
mpo]
>
> png("Variacion_paralelo.png")
> ggplot(data=resultados, aes(x = Agentes, y= Tiempo,fill = Tipo)) +
+   geom_boxplot(position=position_dodge(1))
> dev.off()
```

2. *Archivo 'Cod_P6.R'*

Este archivo es utilizado como un sistema control, por lo tanto, las únicas modificaciones realizadas a este código fueron:

- Declarar una variable lógica `graficos`, la cual es utilizada dentro de un condicional *if* para crear o no gráficos dentro del código, según los objetivos del mismo.
- Crear una variable `t` para medir el tiempo que toma el código en ejecutarse.

3. *Archivo 'Cod_p6_parSapply'*

Debido a la larga extensión del código, sólo se presentan las modificaciones realizadas.

- Declarar una función *probabilidad.*

Probabilidad se encuentra en función de la variable `j`, la resultante *contagios* indicará la posición de los agentes susceptibles *"S"* que serán contagiados en el

siguiente valor de `tiempo`. Esta función se realizará de manera paralela, debido que se puede realizar el cálculo de las distancias euclidianas entre agentes infectados y agentes susceptibles de manera independiente. Además de identificar los agentes susceptibles a los cuales se les contagiará la enfermedad en el siguiente paso.

```
> probabilidad<-function(j){
+   contagios <- rep(FALSE, n)
+   for (i in 1:n) {
+     if (agentes[i,5] == "I") { # desde los infectados
+       if (!contagios[j]) { # aun sin contagio
+         if (agentes[j,5] == "S") { # hacia los susceptibles
+           dx <- agentes[i,1] - agentes[j,1]
+           dy <- agentes[i,2] - agentes[j,2]
+           d <- sqrt(dx^2 + dy^2)
+           if (d < r) { # umbral
+             p <- (r - d) / r
+             if (runif(1) < p) {
+               contagios[j] <- TRUE
+             }
+           }
+         }
+       }
+     }
+   }
+   return(contagios[j])
+ }
```

- Declarar una función `dEstado`.

La función `dEstado` se encuentra en función de la variable `i`, la cual permite actualizar el estado del agente con base en el vector `contagios` y a la probabilidad de recuperación `pr`. Es posible paralelizar esta parte del código en la cual, ya se tiene un vector `contagios` que te indica la posición de los agentes a los cuales se les realizará un cambio de estado de susceptibles "`S`" a infectados "`I`" o de infectados "`I`" a recuperados "`R`".

```
dEstado <-function(i){
+   if (enfermos[i]){agentes[i,5]="I"}
+   else { if (agentes[i,5]=="I" &
+               runif(1) < pr) {agentes[i,5] <- "R"}}
+   agentes[i,5]<-agentes[i,5]
+ }
```

- Creación del clúster

Con el objetivo de utilizar el paralelismo en esta práctica.

```
+ library(parallel)
+ cluster<-makeCluster(detectCores()-1)
```

```
+ clusterExport(cluster,"r")
+ clusterExport(cluster,"n")
+ clusterExport(cluster,"pr")
+
+ for (tiempo in 1:tmax) {…}
```

- Creación del vector `enfermos` y el vector remplazo de `agentes$estado`

```
+   clusterExport(cluster,"agentes")
+
+   enfermos<-parSapply(cluster,1:n,probabilidad)
+
+   clusterExport(cluster,"enfermos")
+
+   agentes$estado<-parSapply(cluster,1:n,dEstado)
```

- Cambios de posición para los agentes

Se decidió no paralelizar esta parte, ya que se pueden sumar o restar columnas directamente.

```
+   agentes$x <- agentes$x + agentes$dx
+   agentes$y <- agentes$y + agentes$dy
+
+   agentes[agentes$x < 0,1]<-agentes[agentes$x < 0,1]+l
+   agentes[agentes$x > l,1]<-agentes[agentes$x > l,1]-l
+   agentes[agentes$y < 0,2]<-agentes[agentes$y < 0,2]+l
+   agentes[agentes$y > l,2]<-agentes[agentes$y > l,2]-l
```

- Comprobación del código

  La gráfica 1 sirve para corroborar que el código paralelizado realiza lo mismo que el código original. En ella se puede observar que el porcentaje de infectados tiende al aumento conforme el tiempo avanza hasta llegar a un porcentaje de infección máxima y a partir del cual disminuye el porcentaje de agentes infectados debido a la probabilidad de recuperación

.

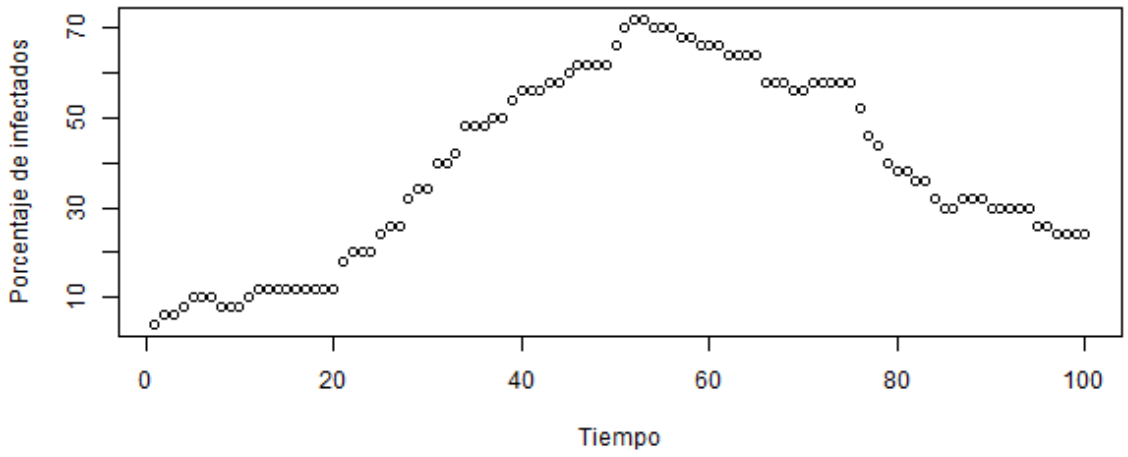

Figura 1 Gráfico que muestra el porcentaje de la población que se encontraba infectado en función del tiempo.

Resultados

En la figura 2 se muestra un comparativo entre ambos códigos, es claro que el código paralelizado es mucho más rápido para un número de agentes mayor a 65, si el número de agentes es menor el tiempo de ejecución es similar. Por lo tanto, se sugiere usar un código paralelizado para números de agentes grandes.

Si se compara el tiempo en función al número de agentes, se puede observar que conforme el número de agentes aumenta se alcanzan valores más altos de tiempo de ejecución, lo cual es de esperarse pues se traduce en un mayor número de cálculos. Si bien este aumento sucede para ambos códigos, se observa que el aumento es mayor para el código no paralelizado.

Otro aspecto que cabe resaltar es el largo de las cajas del bloxplot, ya que se observa este largo es mayor en el código original, por lo que se puede decir que el tiempo de ejecución en un código no paralelizado es demasiado variable. La explicación a los valores extremadamente pequeños se debe a que en el primer paso no se encontró ningún agente infectado, esto se debe a que el valor para comparar con la probabilidad de infección `pi` es un valor al azar.

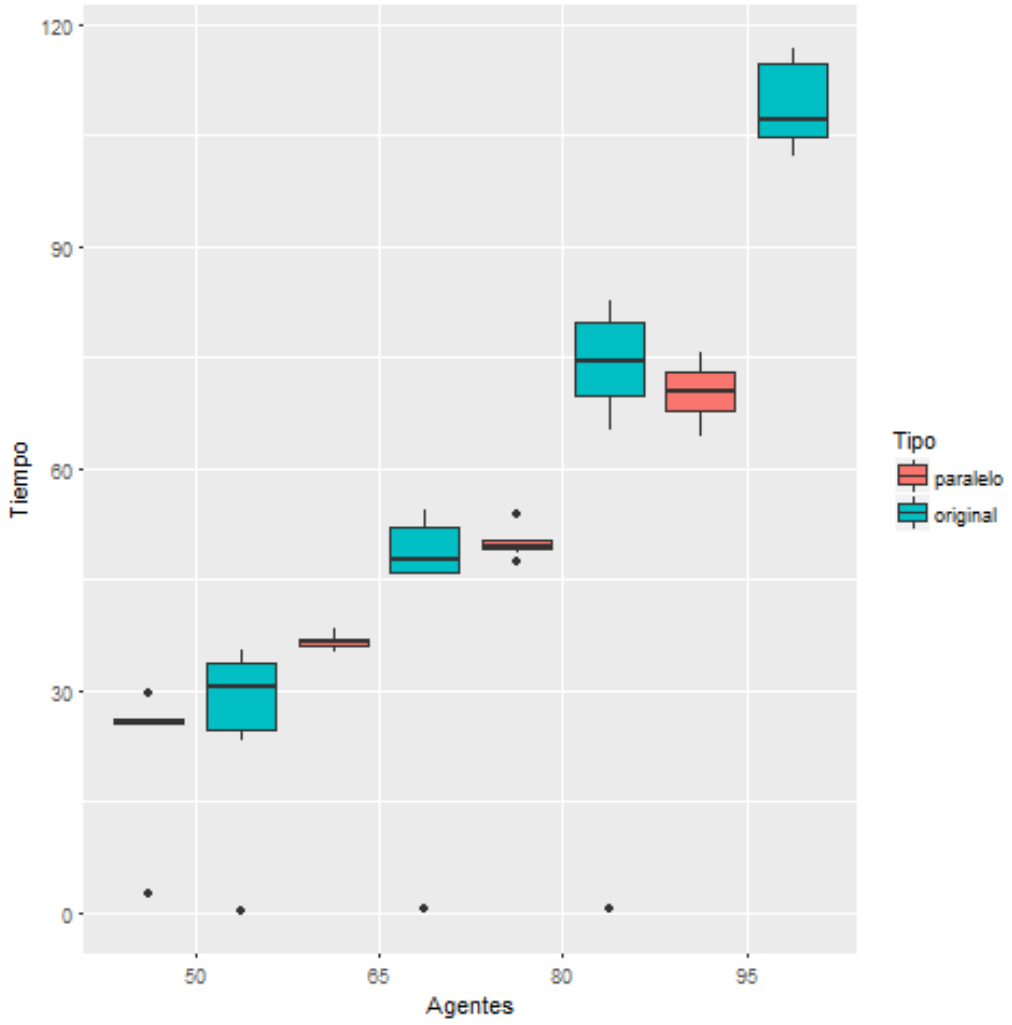

Figura 2 Comparación en tiempo entre un código paralelizado y sin paralelizar

Referencias

[1] Elisa.dyndns-web.com. (2017). P6 — R paralelo — Schaeffer. [online] Available at: http://elisa.dyndns-web.com/teaching/comp/par/p6.html [Accessed 13 Sep. 2017].