Práctica No 3: Teoría de colas

Héctor Hernán Montes García

28 de agosto de 2017

Resumen

En la presente práctica se se estudian los tiempos de ejecución de un Código en R consistente en un método para descubrir qué números dentro de una secuencia de números enteros son primos. Como el problema no es puramente secuencial, los tiempos se estudian para diferentes formas de paralelización del código haciendo uso de la librería de R denominada *DoParallel*. Para el efecto, se divide la tarea completa en sub-tareas de diversa dificultad y se toma control de la cantidad de núcleos asignados a la tarea completa, así como del orden en que las sub-tareas son distribuidas entre cada núcleo. A través de pruebas estadísticas formales, se concluye que el orden de asignación de las sub-tareas, así como la cantidad de núcleos trabajando sobre la tarea completa no impacta significativamente en el tiempo total de ejecución del trabajo completo.

1. Planteamiento del problema

En esta práctica estudiamos el problema de identificar qué números son primos dentro de una secuencia de número enteros. Para decidir si un número n particular es primo, una alternativa es revisar si él es divisible entre algún entero menor que él. En general es posible acotar el grupo de enteros candidatos (por ejemplo enteros no superiores a $\sqrt{n}$).La tarea de encontrar todos los números que son primos dentro de una secuencia de enteros especificada pasa entonces por aplicar esta sencilla regla sobre cada entero n de la secuencia. Si ningún entero actúa como divisor exacto el número se declara primo. Por supuesto, para evaluar si un número de la secuencia es primo (digamos n_1), no se requiere esperar los resultados de la evaluación sobre otro (digamos n_2), así que las evaluaciones se pueden hacer en paralelo, y podemos disponer varios núcleos de la computadora para estudiar varios números en simultáneo.

El problema anterior tiene todas las características de un problema de teoría de colas, donde cada núcleo actúa como un servidor (prestador del servicio de identificar si un n dado es primo), la longitud completa de la cola en el tiempo t_0 es la longitud de la secuencia de números a etiquetar como primos o no. Aunque pudiera ser de interés evaluar cuánto tarda la tarea en ser completada conforme más números se agregan a la cola de trabajo a una tasa de llegada dada, y considerando la tasa de servicio de los núcleos (problema típico en teoría de colas), por lo pronto trataremos un problema más bien sencillo: consideraremos que no existen llegadas nuevas, y que nos interesa evaluar cuánto tarda el servicio completo en ser prestado, es decir, tiempo transcurrido entre la evaluación del primer número, y la evaluación del último medido a través de una instrucción

que registre el momento en que la función de etiquetado ha finalizado con todos los números. Esto dependerá, en general, del orden de llegada de los casos, y la forma cómo estos son enviados para ser atendidos por cada servidor, puesto que cada núcleo puede recibir una carga de trabajo diferente y no tienen porque finalizar en tiempos parejos.

En la presente situación práctica, nosotros asumimos no saber cómo opera internamente la librería de R *DoParallel* en lo que atañe a la distribución del esfuerzo de cómputo entre los núcleos (aunque siempre sea posible consultarlo por revisión cuidadosa del código original). La implementación actual de la instrucción de etiquetado es como sigue:

```
primo <- function(n)
{
  #Evaluamos si el entero es el primo 1 o 2 (casos triviales)
  if (n == 1 || n == 2)
  {
    return(TRUE)
  }
  #Evaluamos si es par, si lo es, etiquetamos falso (no primo)
  if (n %%2 == 0)
  {
    return(FALSE)
  }
  #Si es impar mayor a 3 y divisible entre un entero candidato,
      ponemos falso
  for (i in seq(3, max(3, ceiling(sqrt(n))), 2))
  {
    if ((i<n)&&(n %%i) == 0)
    {
      return(FALSE)
    }
  }
  return(TRUE)
}
```

Una mirada rápida al código nos hace conscientes de que el tiempo que tarda un núcleo en evaluar un número es variable, pues depende de factores como: pertenecer a un grupo trivial (por ejemplo: 1,2,3), ser éste un número par, o ser impar. Si es impar, la función de etiquetado evalúa los enteros candidatos a divisores, y el tiempo total de etiquetado dependerá del tamaño y naturaleza del número (números primos grandes son más tardados de evaluar).

De acuerdo con lo anterior, nuestro objetivo será, no sólo estudiar los tiempos de etiquetado, sino también realizar conjeturas sobre el mecanismo usado por la librería *DoParallel* para distribuir entre los núcleos la tarea de etiquetar toda un secuencia completa de números.Para ello convendrá tomar control de las secuencias de números que se le pasarán al programa, en forma tal que se conozca previamente la dificultad de evaluación de los números suministrados y el orden en que se están pasando al comando *foreach*. Cada número puede interpretarse como una sub-tarea de complejidad variable, y de lo que se trata es de evaluar el impacto que diferentes ordenamientos y diferentes cantidades de núcleos tienen sobre los tiempos totales de ejecución, así como explicar las razones de estas diferencias y establecer por métodos estadísticos formales si éstas son significativas. Los objetivos precisos se formulan en la siguiente sección.

2. Objetivos

1. Examinar cómo cambian las diferencias en los tiempos de ejecución de los diferentes ordenamientos cuando se varía el número de núcleos asignados a la tarea.

2. Argumentar, apoyándose con visualizaciones, las posibles causas de las diferencias en los tiempos de ejecución y razonar cómo y por qué le afecta el número de núcleos disponibles a la diferencia.

3. Aplicar en R pruebas estadísticas para determinar si las diferencias observadas entre los tres ordenamientos son significativas.

3. Metodología

Para atacar cada objetivo se procede con un método diferente. Las diferencias de métodos son esencialmente en el grado de control que se toma sobre las secuencias de números que se van a evaluar (tanto en orden como en complejidad), y la cantidad de núcleos asignados.

- En el caso del primer objetivo, sólo nos interesa la descripción del problema, así que el control que se aplicará sobre la secuencia de número es mínimo y sólo consistirá en 3 formas diferentes de ordenar una secuencia de números a estudiar.

- En el caso del segundo objetivo, interesa una aproximación vía simulación y análisis gráfico a la explicación de las diferencias detectadas en el primer objetivo, así que esto requerirá proceder en una forma un poco más sistemática en el diseño y asignación de secuencias, con el fin de conjeturar sobre la forma de asignación de tareas a la cantidad de núcleos disponibles y sus efectos en los tiempos de ejecución.

- Finalmente el último objetivo añade formalismo a la explicación del segundo objetivo, así que será necesario el planteamiento de hipótesis sobre los tiempos de ejecución y su contraste a través de procedimientos con el debido rigor estadístico.

3.1. Método para el Objetivo 1

El objetivo 1 lo atacamos construyendo una rutina en R que evalúa los tiempos de ejecución de una labor consistente en etiquetar los números enteros entre 10000 y 30000. Una caracterización previa de estos números se realizó construyendo una pequeña rutina de conteo de pares, impares y primos, sobre la secuencia mencionada.

El código resulta de ligeras modificaciones al código original que se ofreció en clase para medir tiempos de ejecución del etiquetado de números. En lugar de simplemente etiquetar el número como primo o no primo, usando como valor de retorno de la función: *false* o *true*, retornamos al usuario los textos *Trivial*, *Par*, *Impar no primo* e *Impar primo* para caracterizar cada entero, aprovechando todos los núcleos de la máquina (excepto uno que se conserva para mantener el sistema responsivo). Posteriormente almacenamos los valores de los retornos

en un vector usando la sentencia *foreach* en lugar de guardar el tiempo total de ejecución. Finalmente un llamado a la función *table* sobre el vector creado, da toda la información necesaria para construir la Tabla 1. El código usado para el conteo de números es el siguiente:

```
suppressMessages(library(doParallel))
nucleos<-(detectCores()-1)
conteos <- function(n)
{
  if (n == 1 || n == 2)
  {
    return("Trivial")
  }
  if (n %%2 == 0)
  {
    return("Par")
  }
  for (i in seq(3, max(3, ceiling(sqrt(n))), 2))
  {
    if ((i<n)&&(n %%i) == 0)
    {
      return("Impar no primo")
    }
  }
  return("Impar primo")
}
desde <- 10000
hasta <-  30000
original <- desde:hasta
registerDoParallel(makeCluster(nucleos))
ot<-numeric()
ot <- foreach(n = original, .combine=c) %dopar% conteos(n)
stopImplicitCluster()
table(ot)
```

Así mismo, el procesamiento de la información contenida en una base de datos disponible en internet[1] que provee los primeros 50 millones de números primos, nos permitió confirmar la consistencia de nuestros resultados.

Tabla 1: Conteo de números según tipo y tamaño

Tipo de número	Tamaño				Totales
	[10000-15000]	(15000-20000]	(20000-25000]	(25000-30000]	
Impar	2500	2500	2500	2500	10000
No primo	1992	2000	2017	1975	7984
Primo	508	500	483	525	2016
Par	2500	2500	2500	2501	10001
Totales	5000	5000	5000	5001	20001

Finalmente, la siguiente rutina paraleliza el conteo de tiempos de ejecución de etiquetado para diferentes cantidad de núcleos asignados, y diferente órdenes en el paso de la secuencia de números.

Muchas hipótesis se pueden hacer respecto a la distribución de tareas a los núcleos, tres razonables son:

[1] https://primes.utm.edu/lists/small/millions/

1. A cada núcleo que se desocupa se le asigna el siguiente elemento en la lista de espera.

2. La tarea completa se distribuye desde el inicio en proporciones iguales (o casi iguales) entre los núcleos, partiendo el vector de casos a evaluar en el orden de aparición en la lista.

3. La tarea completa se distribuye desde el inicio en proporciones iguales (o casi iguales) pero tomando de la lista subconjuntos de casos donde los elementos de cada subconjunto son elegidos al azar.

En los últimos dos casos asumimos que los núcleos quedan cargados con una cierta cantidad de trabajo desde el inicio, y que no se balancean las cargas a lo largo del proceso, en el primero por el contrario asumimos balanceo gradual de la carga porque las tareas no se asignan en bloque sino de la forma más granular posible, es decir, un caso a la vez.

De acuerdo con lo anterior, para los casos 2 y 3 simplemente esperaríamos que si un núcleo termina primero todo su lote de números, tendrá que esperar a que los demás finalicen la tarea asignada, y el tiempo total de ejecución lo decidirá principalmente el subconjunto que haya sido de más tardado procesamiento (asumiendo rendimiento igual de cada núcleo).

Para cotejar esta diversidad de hipótesis, y poder caracterizar el mecanismo subyacente a la asignación de tareas a los núcleos, conviene seguir un enfoque de diseño de experimentos. Esto supone el planteamiento de escenarios de experimentación que pongan a prueba cada hipótesis. Los escenarios consistirían en diseños de configuraciones de lista de sub-tareas, donde la dificultad de evaluación de cada sub-tarea en la lista ordenada es previamente conocida. Note que la construcción de escenarios no puede obviar el eventual efecto de la cantidad de núcleos asignados a la tarea completa.

Registrados los tiempos de ejecución totales para cada escenario diseñado, evaluamos si la información incorpora evidencia estadística fuerte a favor de cada uno de los esquemas de asignación de tareas hipotetizados, usando pruebas estadísticas formales. Para esto es necesario tener muy claro el comportamiento en los tiempos de ejecución total que se esperarían para una mezcla particular de escenarios de orden de dificultad de sub-tareas, cantidad de núcleos, e y esquema de asignación de tareas hipotetizado.

Para sistematizar estas ideas, a continuación se describe el comportamiento esperado para cada mezcla de orden subtareas-núcleos-hipótesis, antes mencionado.....

Referencias

[1]