Analisis de un Algoritmo en un Ambiente Distribuido.

Resumen

Estudiamos y presentamos los elementos del Modelo de Computo Distribuido, en el cual se definió de manera formal sus elementos, asi como se formalizo lo que significa la ejecución del algoritmo en un modelo de computo Distribuido
Se presentaran y definirán las nociones del análisis de un Algoritmo Distribuido, como son:

- Complejidad del Algoritmo,i.e Complejidad temporal.
- La exactitud o correccion del Algoritmo.

Entonces con estas nociones presentaremos un problema en el contexto de un ambiente distribuido el cual sera resulto diseñando un algoritmo.

Definiciones y Conceptos

Definicion de complejidad de un Algoritmo ejecutado en un modelo de computo: $LOCAL$.

Como en un modelo secuencial la medida de complejidad en el sentido temporal es el numero de pasos que toma del inicio al final, entonces inspirado en esa noción de complejidad secuencial definimos el siguiente concepto

Complejidad Temporal de un Algoritmo

Definición:
Sea G una gráfica y π un algoritmo entonces definimos la complejidad de π de manera *Sincrona* como el numero de rondas que se generaron durante la ejecución de π en G

De igual manera se puede definir la complejidad de una manera *Asincrona* como las unidades de tiempo que se generan desde el inicio del algoritmo π en G hasta el final de su ejecución.

Noción de exactitud o correcion del Algoritmo.
Definición

Diremos que un Algoritmo es correcto si:

- Resuelve el problema computacional por el cual fue diseñado
- Para cada entrada, produce la salida deseada
- Termina en un tiempo de ejecución finito

Definicion(Notacion Big-Oh): Si f y g son funciones de $\mathbb{Z}$ en $\mathbb{Z}$ entonces decimos que:

- $f = O(g)$ si existe una constante c tal que: $f(n) < c\dot{g}(n)$ para todo n suficientemente grande.
- Diremos que $f = \theta(g)$ si $f = O(g) y g = O(g)$

Entonces con base a esa definición podemos encapsular la complejidad(Temporal) del Algoritmo en términos de las definiciones anteriores, entonces podemos decir que la complejidad esta dada de la siguiente manera: *como la complejidad temporal es dada con sintaxis y semántica*:

$Time(\pi, G) = O(g(n)$ donde g dependerá de la naturaleza del algoritmo en cuestión.

Presentacion del Problema a resolver y su respectivo Algoritmo

Tenemos en contexto el siguiente Problema:
Dado una red de procesadores con cierta topologia, a saber la topologia de un arbol, donde cada nodo tiene un entero n, entonces el problema a solucionar es encontrar el elemento mas repetido en toda la red, i.e encontrar estadisticamente el modo de la red; para eso debemos implementar un protocolo de comunicacion en el Arbol para determinar dicho valor. Que en terminos formales es diseñar un algoritmo que sera una tupla de n protocolos los cuales se ejecutaran en el Arbol para solucionar dicho problema.

Definicion Conceptualmente tendremos los siguientes tipos de mensajes que se enviaran en el protocolo que se enuciara mas adelante:

- Mensajes del tipo 1: $M(\uparrow, A_{i1}, A_{i2})$
- Mensajes del tipo 2: $M(\uparrow, (x_{modo}), max(A_{i2}))$

Donde A_{i1} es un arreglo de los valores locales de los nodos del sub-arbol enraizado en el nodo p_i y A_{i2} es un arreglo de las frecuencias de los valores locales en A_{i1}

Definicion
Sean $M(\uparrow, A_{i1}, A_{i2})$, $M(\uparrow, A_{k1}, A_{k2})$ mensajes del tipo 1, entonces podemos definir la operacion $Sum(M(\uparrow, A_{i1}, A_{i2}), M(\uparrow, A_{k1}, A_{k2}))$ y la definimos de la siguiente manera:

1. Sea x en A_{i1}, si x es tal que es distinto a todo z en A_{j1} entonces
   $A_{r1} = A_{i1}[y] \star A_{k1}$ y $A_{r2} = A_{i2}[y] \star A_{k2}$
   donde $\star$ denota la concatenacion del array formado por el elemento $x = A_{i1}[y]$ con A_{k1} y tambien denota la concatenacion del elemento $f_x = A_{i2}[y]$ con el array A_{k2}, es decir estamos generando un array que contiene a los que estamos operando bajo $\star$
2. Sea x en A_{i1}, si es tal que $x = A_{i1}[y] = A_{k1}[y']$ entonces $A_{r1}[y''] = A_{i1}[y] = A_{i2}[y']$ y $A_{r2}[y''] = A_{i2}[y] + A_{k2}[y']$

Y denonamos a $Sum(M(\uparrow, A_{i1}, A_{i2}), M(\uparrow, A_{k1}, A_{k2})) = M(\uparrow, A_{r1}, A_{r2})$

En las siguientes lineas se muestra el Algoritmo.

Algorithm 1 First-dis(T_{r_0}, ID, x_i)

$A_{first}(p_i, ID, x_i \in \mathbb{N})$
if p_i es una hoja **then**
  $A_{r1} = (x_{p_i})$
  $A_{r2} = (1)$
  send $M_{p_i} = M(\uparrow, A_{r1}, A_{r2})$ a su padre
else
  wait $M_1, \ldots, M_n$ mensajes de sus hijos
  $M_j = M(\uparrow, A_{j1}, A_{j2})$
  for all $l, s \in \{1, \ldots, n+1\}$ con $l \neq s$ **do**
    $Sum(M(\uparrow, A_{l1}, A_{l2}), M(\uparrow, A_{s1}, A_{s2}))$
  end for
  $M(\uparrow, A_{r1}, A_{r2}) = Sum(M(\uparrow, A_{11}, A_{12}), \ldots, M(\uparrow, A_{n1}, A_{n2}))$
  $A_{p_i 1} = A_{r1}$ // Es el array después de las operaciones de los mensajes de sus Hijos
  $A_{p_i 2} = A_{r2}$ // Es el array después de las operaciones de los mensajes de sus Hijos
  if p_i es la raiz **then**
    $M_{p_i} = M(\downarrow, (x_{modo}), max(A_{r2}))$
    return x_{modo}
  else
    send $M_{p_i} = M(\uparrow, A_{r1}, A_{r2})$ a su padre
  end if
end if

Correccion del Algoritmo y complejidad de los algoritmos.

En esta parte del documento se mostrara que en efecto A_{first} hace lo que tiene que hacer, que en terminos formales es decir que A_{first} es correcto, para ello diremos que es invariante en el siguiente sentido: Que para cada ronda $r \in \{1, \dots, h_{r_0}\}$ se cumple que el nodo v de altura h_v ha calculado correctamente su variable local a saber su correspondiente M_v que es un mensaje de la lista de tipos.

Theorem 1 *Sea $\pi = A_{first}$ entonces tiene la propiedad de que para cada $r \in \{1, \dots, h_{r_0}\}$ al final de la ronda r el nodo v de altura h ha calculado correctamente su variable local M_v y ademas en el sub-arbol T_v que denota que esta enraizado en v tiene en su variable local las entradas de los nodos de T_v y con posibles repeticiones.*

Demostracion:

Sea $h = 1$ entonces al final de la ronda correspondiente, v es una hoja con base al init del algoritmo

$\therefore M_{v_{h=0}} = M(\uparrow, A_{r1}, A_{r2})$

Suponemos que es cierto para el paso $h = j$ y demostremos que es cierto para $h = j+1$ como es cierto para el paso $h = j$ entonces tiene el subarbol T_{v_j} calculada la variable local correspondiente, luego por la construccion de A_{first} se tiene que en v de altura $j+1$ tambien tiene calculada su variable local que es por la construcción del algoritmo es:

$\therefore v_{h=j+1} = M(\uparrow, A_{r1}, A_{r2})$

Del razonamiento anterior se desprende que el nodo que calcula el maximo es en efecto la raiz del Arbol

Theorem 2 *Para el Algoritmo A_{first} en la ronda r el nodo v esta calculando de manera implicita el elemento mas repetido, mas aun en la ronda $r = h_{r_0}$ es el que toma la decision final de cual es elemento mas repetido y disemina el mensaje a manera de reponse en el resto del arbol*

Demostracion:

Sea r una ronda, entonces notamos que en la correspondiente ronda el vertice r podemos hacer la operacion en la variable local M_v que tiene una representacion $M(\uparrow, A_{r1}, A_{r2})$ por las lineas de codigo del algoritmo, en particular podemos calcular $max(A_{r2})$ que por el mapeo implicito entre A_{r1} y A_{r2} le corresponde el elemento mas frecuente en A_{r2} a saber: x_{modo} Y observamos que si $h = h_{r_0}$ y seguimos las lineas de codigo para ese caso, entonces para $x_{modo_{r_0}}$ es el valor que se toma finalmente y es en la ronda en la que se disemina dicho mensaje.

Ahora tenemos que analizar y deducir la complejidad del Algoritmo en el sentido temporal y de mensajes. Por un lado la complejidad temporal se puede desprender observando la demostracion del teorema anterior: que para la altura correspondiente a la raiz denotada como h_{r_0} sera la complejidad temporal pero solo en la parte del algoritmo **request**, pues en el **response** tardara exactamente lo mismo hasta diseminar el mensaje en el resto de los nodos del arbol, entonces lo decimos mas formalmente en el siguiente enunciado:

Theorem 3 *Sea* $\pi = A_{first}$ *entonces la complejidad temporal denotada como* $Time(T_{first}) = O(h_{r_0})$ *y la complejidad de mensajes es exactamente* $Message(A_{first}) = O(m)$ *donde* m *denota el numero de aristas en* T_{r_0}

Demostracion:

Por construccion del algoritmo y por la correccion del algoritmo, en particular por la propiedad de Loop-Invariant podemos observar que se disemina el mensaje hasta el paso de la induccion que es h_{r_0} *correspondiente a la altura de de la raiz, y luego en el proceso de diseminar el mensaje en todo el arbol de manera de* ***response*** *es la misma altura, entonces la complejidad temporal es* $O(h_{r_0})$
Por otro lado, afirmamos que hay tantas rondas como canales de comunicacion, pues en esencia estamos mapeando la cantidad de mensajes con los canales de comunicacion, pero eso solo en el proceso de ***request*** *por el diseño del algoritmo, en el proceso de* ***response*** *que es la diseminacion del mensaje es el mismo mapeo que en el proceso de* ***request*** *entonces hay tantas rondas como el doble de canales de comunicacion en el proceso de* ***request, response****, mas formalmente lo escribimos de la siguiente manera:* $Message(A_{first}) = O(m)$ *donde* m *denota el numero de aristas en en el Arbol en el que estamos ejecutando el algoritmo.*

Observacion:
Podemos obsevar que existe un subarbol en el que su raiz toma la decision de hacer:
return x_{modo}
Es decir podemos decir que se puede optimizar A_{first} sujeto a la altura h, y apartir de ese subarbol diseminar el mensaje a manera de **response** en todo el arbol T_{r_0}