Algoritmo paralelo para el cálculo de matrices de probabilidades de transición: aplicación a la modelación de yacimientos lateríticos mediante cadenas de Markov

Parallel algorithm to compute probability transition matrices: application to the modeling of lateritic deposits using Markov chains

Dannier Trinchet. M.Sc.[1] & Asnay Guirola. M.Sc.[2]
1. Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.
2. Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.
trinchet@uci.cu, aguirola@uci.cu

Recibido para revisión 04 de junio de 2010, aceptado 28 de junio de 2011, versión final 28 de junio de 2011

***Resumen*— En el presente trabajo se propone un algoritmo paralelo para la obtención de matrices de probabilidades de transición. El algoritmo propuesto es aplicado a la modelación de yacimientos lateríticos a partir de un modelo matemático basado en cadenas de Markov. El modelo genera un hipercubo de probabilidades condicionales, cuya cantidad de dimensiones queda fijada por la cantidad de variables que intervienen en el modelado. Se realiza un análisis teórico del algoritmo y se implementa en dos variantes: usando MPI para su ejecución sobre un cluster Beowulf y a partir de un sistema distribuido para su ejecución en una red local de estaciones de trabajo heterogéneas. Los resultados teóricos y prácticos obtenidos demostraron que el algoritmo es escalable y óptimo en cuanto a Ganancia de Velocidad y Eficiencia. Se propone además, una representación matricial adecuada para el almacenamiento de hipercubos dispersos que persigue un ahorro significativo de memoria con el menor comprometimiento posible de tiempo durante la ejecución del algoritmo.**

***Palabras claves*— Computación paralela y distribuida, Matrices de probabilidades de transición, Modelación de yacimientos lateríticos.**

***Abstract*— The present paper proposes a parallel algorithm to compute transition matrixes and it's application in the modeling of lateritic deposits using Markov Chains. We makes a theoretical analysis of the algorithm and implemented it in two variants, using MPI for execution over a Beowulf cluster and using a distributed system to run on a network of workstations. The analytical and empirical results shows that the algorithm is scalable and optimal in terms of speed-up and efficiency. It also proposes a matrix representation for store the results of modeling pretending reduces memory needed attend to time required for algorithm's execution.**

***Keywords*— Parallel and distributed computing, High throughput computing, Deposits lateritic modeling.**

I. INTRODUCCIÓN

Las matrices de probabilidades de transición (en lo adelante MPT) son ampliamente usadas dentro de la modelación matemática de procesos. Muchas veces son aplicadas como simples herramientas para determinar posibles comportamientos del proceso que se modela i.e Economía, Ecología, Ciencias Sociales, Marketing [1]-[4]; y en otras ocasiones como parte de modelos más complejos i.e Teoría de Autómatas Probabilísticos y Modelado de Procesos Estocásticos [5],[6]. Formalmente una matriz de probabilidades de transición P se define como:

$$P = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{pmatrix}$$

donde n es la cantidad de estados en los cuales puede estar el proceso a modelar, y p_{ij} la probabilidad de que el sistema pase del estado i al j, esta matriz cumple que:

$$\forall i, j \in [1, n] : 0 \leq p_{ij} \leq 1 \quad (1)$$

$$\forall i \in [1, n] : \sum_{j=1}^{n} p_{ij} = 1 \quad (2)$$

Suponiendo que se tiene un conjunto O de observaciones del proceso a modelar, y en cada una de ellas el estado en el cual se encuentra, entonces para ese proceso, *P* puede obtenerse mediante el **Algoritmo 1**. Donde el procedimiento *ajustar_probabilidad_en_P(e_1,e_2)*, actualiza las probabilidades $p_{e1,e2}$ y $p_{e2,e1}$, así como todas las probabilidades que estén en la fila e_1 y la e_2, nótese que debe garantizarse el cumplimiento de la condición (2).

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

Algoritmo 1 Algoritmo secuencial

Entrada: Registro O de n observaciones
Salida: Matriz P de probabilidad de transición

```
crear P    // matriz de n × n donde ∀i, j ∈ [1, n] : p_ij = 0
for i := 1 to n − 1 do    //ciclo (1)
  for j := i + 1 to n do    //ciclo (2)
    e_O ← estado_del_proceso(O_i)
    e_D ← estado_del_proceso(O_j)
    if existe_transicion(e_O, e_D) then
      ajustar_probabilidad_en_P(e_O, e_D)
    end if
    if existe_transicion(e_D, e_O) then
      ajustar_probabilidad_en_P(e_D, e_O)
    end if
  end for
end for
return P
```

El tiempo de este algoritmo secuencial y denotado por T_S queda expresado por la siguiente fórmula:

$$T_S = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} c\,Flops = \sum_{i=1}^{n-1} c(n-i)\,Flops \approx \frac{cn^2}{2}Flops \quad (3)$$

Como puede verse la complejidad temporal de este algoritmo es cuadrática, por lo que aplicarlo en situaciones donde el tamaño de la entrada sea pequeño no representa un inconveniente práctico, sin embargo, existen situaciones donde *n* es considerablemente grande y/o se deben calcular muchas matrices de este tipo para resolver un mismo problema, ejemplo de esto son los Autómatas Probabilísticos donde por cada símbolo del alfabeto de entrada del autómata existe una MPT.

II. ALGORITMO PARALELO

El conjunto de observaciones O puede verse, sin pérdida de generalidad, como una tabla (lista de elementos), la cual contiene un registro por cada observación realizada. En lo adelante se referirá *comparación* del registro *i* con el registro *j* a la operación necesaria para reflejar en P la transición del estado *i* al *j*.

II.A Algoritmo

Teniendo en cuenta que el algoritmo recibe como entrada una tabla de *n* registros los cuales deben ser comparados entre sí, puede descomponerse (funcionalmente) el problema dividiendo *O* en *2p* bloques de igual tamaño, cada uno formado por *n/2p* registros consecutivos donde *p* es el total de procesadores, en lo adelante se denotará por B_i al (i+1) -ésimo bloque.

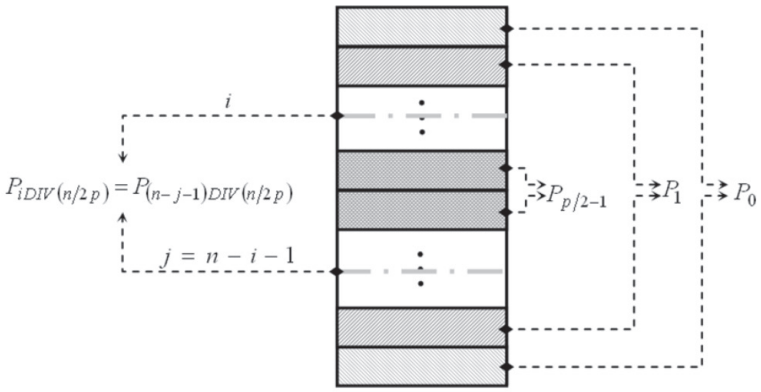

Figura 1. Asignación de bloques extremos (Fase I del algoritmo)

Con el objetivo de balancear la carga se define una tarea t_k como el cómputo necesario para llevar a cabo todas las comparaciones requeridas por los bloques B_k y B_{2p-k-1} $\forall k \in [0, p-1]$, y cuyo resultado será una matriz P_k^0. La *K*-ésima tarea consistirá entonces en las comparaciones correspondientes a los registros

$$\frac{nk}{2p}, \frac{nk}{2p}+1, \ldots, \frac{n(k+1)}{2p}-1$$

y

$$n\left(1-\frac{k+1}{2p}\right), n\left(1-\frac{k+1}{2p}\right)+1, \ldots, n\left(1-\frac{k}{2p}\right)-1$$

. Por lo que la comparación de los registros $c_i \forall i < n/2,$ así como las correspondientes a los registros $c_j \forall j \geq n/2$ serán realizadas en el procesador

$$P_{iDIV\left(\frac{n}{2}\right)} = P_{n-j-1DIV(n/2p)}$$

.

De esta forma se persigue que todos los procesadores tengan exactamente la misma carga, al menos durante la ejecución de esta Fase del algoritmo. Sea p_r el procesador que ejecuta la tarea t_r consistente en realizar las comparaciones correspondientes a los bloques B_r y B_{2p-r-1}. Si $c_i \in B_r$ con $i < n/2$ entonces

$c_{n-i-1} \in B_{2p-r-1}$, que sería el otro bloque a procesar en p_r, si se compara c_i con $c_j \forall j > i$ entonces se realizan $n-i-1$ comparaciones, de forma análoga se calcula que, para c_{n-i-1}, se realizan $n-(n-i-1)$, que en su conjunto suman $n-1$.

Teniendo en cuenta que todos los bloques son de tamaño *n/2p* se concluye que, en toda tarea t_r se realiza un total de *n(n-1)/2p* comparaciones independientemente de los bloques que les sean asignados a p_r.

Una vez realizada las comparaciones, en cada procesador p_k se tiene una matriz P_k^0, por lo que se necesita aun realizar la combinación de todas para obtener la MPT final $P = \sum_{i=0}^{p-1} P_i^0$. Este proceso puede realizarse haciendo una suma en forma de árbol, tal y como se muestra en la Figura 2, donde los nodos hojas son las matrices correspondiente a cada uno de los *p* procesadores: $P_k^0 \forall k \in [0, p-1]$, y cada nodo intermedio $P_k^j \forall j \in [1, \log p]$ la matriz resultado en el paso *j* de la suma de las matrices obtenidas en los procesadores p_k y p_r con $r = k + p - 2^{j-1} MOD\ p$ en el paso *j-1*, o sea $P_k^j = P_k^{j-1} + P_r^{j-1}$. De esta forma al final del proceso, log p ésimo paso, en p_0 quedará $P = \sum_{i=0}^{p-1} P_i^0 = P_0^{\log p}$ que es la MPT objetivo.

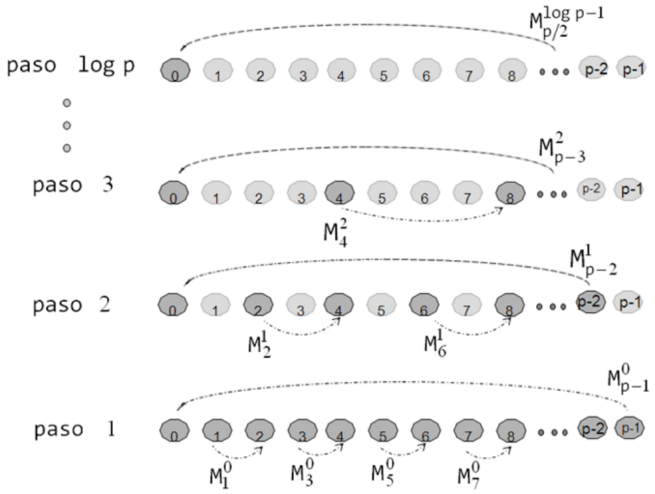

Figura 2. Suma de matrices (Fase II del algoritmo)

II.B Descripción formal

En **Algoritmo 2** se formaliza el procedimiento antes descrito, el cual está basado en una descomposición funcional mediante asignación de bloques extremos de *n/2p* registros consecutivos.

Algoritmo 2 Algoritmo paralelo
Entrada: Registro O de n observaciones
Salida: Matriz P de probabilidad de transición

```
EN PARALELO:
for pr = 1, 2, ··· , p − 1 do
  EN pr :
  //FASE I: Cada procesador pr calcula su matriz P^0_pr
  //Iteración sobre los registros del 1er bloque(B_pr) que
  conforma a τ_pr
  for i := pr n/2p to (pr + 1)n/2p do
    for j := i + 1 to n do
      e_O ← estado_del_proceso(O_i)
      e_D ← estado_del_proceso(O_j)
      if existe_transicion(e_O, e_D) then
        ajustar_probabilidad_en_P(e_O, e_D)
      end if
      if existe_transicion(e_D, e_O) then
        ajustar_probabilidad_en_P(e_D, e_O)
      end if
    end for
  end for

  //Iteración sobre los registros del 2do bloque(B_{2p−pr−1})
  que conforma a τ_pr
  for i := n(1 − (k + 1)/2p) to n(1 − k/2p) − 1 do
    for j := i + 1 to n do
      e_O ← estado_del_proceso(O_i)
      e_D ← estado_del_proceso(O_j)
      if existe_transicion(e_O, e_D) then
        ajustar_probabilidad_en_P(e_O, e_D)
      end if
      if existe_transicion(e_D, e_O) then
        ajustar_probabilidad_en_P(e_D, e_O)
      end if
    end for
  end for

  //FASE II: Cálculo de P = Σ_{i=0}^{p−1} P^0_i = P^{log p}_0
  //En cada uno de los log p pasos se realizan sumas
  parciales acumulativas
  //En el paso i−ésimo se realizan p/2^i sumas
  for i := 1 to log n do
    if pr MOD 2^i = 0 then
      Recibir Matriz
      P^i_pr = P^{i−1}_pr + Matriz
    else
      Enviar P^i_pr a p_k donde k = (pr + 2^{(i−1)}) MOD p
      Terminar
    end if
  end for
  if pr = 0 then
    return P^{log p}_pr
    Terminar
  end if
end for
```

II.C Análisis de prestaciones

II.C1 Parámetros Absolutos: El tiempo paralelo del algoritmo T_P se determina [7], para sistemas paralelos con memoria distribuida, como $T_P = T_A + T_C + T_{SOL}$ donde T_A

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

es el tiempo aritmético, T_C el tiempo de comunicación y T_{SOL} es el tiempo de solapamiento. Debido a la complejidad del cálculo de T_{SOL} se hace la aproximación $T_P \approxeq T_A + T_C$, en cuyo caso se está buscando una cota superior para T_P. Para encontrar el T_A se puede realizar el análisis en la misma forma en que está diseñado el algoritmo, el cual queda dividido en dos Fases fundamentales, en la Fase I se realiza el cálculo de la matriz correspondiente al procesador pr, esta se obtiene haciendo $n-1$ comparaciones por cada par de registro $(O_i, O_j) : \forall i \in \left[\frac{npr}{2p}, n\left(1-\frac{pr}{2p}\right)-1\right]$ y $\forall j \in [\frac{n(pr+1)}{2p}-1, n\ (1-\frac{pr+1}{2p})]$, teniendo en cuenta que el total de registros asignados a p_r es de *n/p* y cada comparación tiene un costo de *c Flops* la Fase I tiene un costo total de $cn(n-1)/2p$, puede comprobarse mediante la expresión:

$$\sum_{i=n\ pr/2p}^{\frac{n(pr+1)}{2p}-1} \sum_{j=i+1}^{n} c\ Flops + \sum_{i=n\left(1-\frac{pr}{2p}\right)-1}^{n\left(1-\frac{pr+1}{2p}\right)-1} \sum_{j=i+1}^{n} c\ Fops$$

En la Fase II se realizan *p-1* sumas, cada una de costo constante *m Flops*, en *log p* etapas por lo que el tiempo de la Fase II es *m log p Flops*. En consecuencia $T_A = \frac{cn^2}{2p} + m\ \log p\ Flops$. Para determinar el tiempo de comunicación hay que tener en cuenta la cantidad de envíos de mensajes que se realizan en el algoritmo, sus tamaños y el tipo de red de interconexión. En general [7] $T_C = d(N_\tau + \beta)$ donde *d* es el número de enlaces a cruzar (diámetro de la topología), τ el tiempo de envío de una palabra y β el tiempo de latencia. En el algoritmo (Fase II) se envían $p-1$ mensajes $(1+2+\ldots+2^{k-1}-1=2^k-1=p-1)$ cada uno de tamaño *m* (cantidad de elementos de la matriz), considerando que la topología de la red de interconexión es la de estrella el tiempo de comunicación del algoritmo es $T_C = 2(p-1)(m\tau+\beta)$. Por consiguiente:

$$T_P = \frac{cn^2}{2p} + m\log p + 2(p-1)(m\tau+\beta)\ Flops \qquad (4)$$

En la sección *III Implementación* se dan algunas consideraciones de la constante *m*.

II.C2 Parámetros Relativos: El **Speed-Up** (S_P) del algoritmo, que indica la ganancia de velocidad de este algoritmo comparado con el mejor algoritmo secuencial conocido, está dado [8] por:

$$S_P = \frac{T_S}{T_P} = \frac{cn^2\ Flops}{\frac{cn^2}{2p} + m\log p + 2(p-1)(m\tau+\beta)\ Flops}$$

$$\lim_{n\to\infty} S_p = p \qquad (5)$$

Como puede verse el algoritmo tiene un **Speed-Up** óptimo, al menos teóricamente, pues la ganancia de velocidad tiende a ser directamente proporcional al número de procesadores a medida que *n* aumenta. Lo que significa que si se aumenta el tamaño del problema se obtiene mayor ganancia de velocidad hasta llegar al óptimo teórico que es *p*. La **Eficiencia** (E_p) de este algoritmo, que expresa el grado de utilización de un sistema multiprocesador [8], y calculable mediante $E_p = \frac{S_p}{p}$ es:

$$E_p = \frac{1}{1 + \frac{2\,p\,m\ \log p + 4p(p-1)(m\tau+\beta)}{cn^2}}$$

$$\lim_{n\to\infty} E_p = 1 \qquad (6)$$

Lo que indica que a medida que aumenta el tamaño del problema el algoritmo usa los procesadores de forma óptima. Otro parámetro muy importante de analizar en el algoritmo es cuán equilibrado está el trabajo entre los *p* procesadores del sistema, para ello se calcula el **Desequilibrio de Carga** [7] que está dado por la expresión $D_q = \left|\frac{T_s}{p} - T_A\right|$ de donde:

$$D_q = \left|\frac{cn^2/2}{p} - cn^2/2p - m\log p\right| = m\log p\ Flops \qquad (7)$$

como puede notarse el desequilibro de la carga es constante respecto al tamaño del problema, esto es debido a que en la Fase I del algoritmo la cantidad de trabajo en cada uno de los procesadores es la misma, pero en la Fase II hay un desbalance, no dependiente del tamaño del problema sino de la cantidad de procesadores(cantidad de matrices a sumar) pues en el primer paso trabajan todos pero en los pasos siguientes va disminuyendo el número de procesadores que lo hacen. Teniendo que $T_p = cn^2/2p + m\log p + 2(p-1)(m\tau+\beta)\ Flops$, el **número óptimo de procesadores** puede determinarse haciendo que derive un tiempo paralelo mínimo. $\frac{\partial T_p}{\partial p} = 0$

$$(cn^2/2p + m\log p + 2(p-1)(m\tau+\beta))' = 0$$

$$-\frac{cn^2}{2\left(\frac{1}{p^2}\right)}+m\left(\frac{1}{p}\right)+2\left(m\tau+\beta\right)=0$$

$$2\left(m\tau+\beta\right)p^2+mp-\frac{cn^2}{2}=0$$

Donde:

$$p=\frac{\sqrt{m^2+4cn^2\left(m\tau+\beta\right)}-m}{4\left(m\tau+\beta\right)} \quad (8)$$

Obviamente, en la práctica se usarán los procesadores con los que se cuente. Dado los altos valores de *n*, es muy difícil disponer de *n* procesadores, que es el valor que aproximadamente puede dar esta expresión.

III. IMPLEMENTACIÓN

El algoritmo fue aplicado a la modelación de yacimientos lateríticos a partir del siguiente modelo:

$$X\left(t,r\right)=\sum_{i=1}^{n}\pi\left(i\right)\rho(t,r) \quad (9)$$

propuesto en [9] y basado en la existencia de tipos o clases de materiales patrones en yacimientos de este tipo, asumiendo que el conjunto de estas contiene todas las variables presentes en ellos, y que además representa el espacio muestral. Cada clase se asume como uno de los estados en un proceso de Markov.

Las matrices $\pi(i)$ son MPTs que en (9) están asociadas a características temporales y geoespaciales $\rho(t,r)$, por lo que para toda combinación *r* de valores posibles de las variables que describen un punto de muestreo[1] en el momento *t* se tiene una MPT elemental de $k\times k$, donde *k* es la cantidad de clases que caracterizan a los yacimientos lateríticos según [10]. Las variables que se consideraron en *r* en la propuesta inicial de (9) fueron: dirección horizontal (α_H) y vertical (α_V), profundidad (*H*) y distancia entre los puntos de muestreo (*P*).

De esta forma (9) puede verse como una MPT 6-dimensional o, lo que es equivalente en nuestro caso a, $|\alpha_H|\times|\alpha_V|\times|H|\times|P|$ matrices de probabilidades de transición donde $|x|$ representa la cardinalidad de la variable *x*. Como puede apreciarse, estas variables son continuas por lo que se hace necesario discretizarlas, lo ideal desde el punto de vista de costo computacional sería transformarla a una variable discreta con cardinal pequeño, sin embargo, este problema (modelación de yacimientos lateríticos) establece límites ya que se puede ver comprometida la exactitud con que se necesitan los resultados, o sea la eficacia de (9), así como por las características mineralógicas y geoespaciales propias de los yacimientos lateríticos.

Por otro lado, la cantidad de muestras (tamaño del problema) está en el orden de los cientos de miles, incluso cantidades mayores si se combinan para un mismo yacimiento los registros de múltiples muestreos. Ambas cosas en su conjunto provocan que la ejecución de este algoritmo en una máquina convencional no sea viable en términos prácticos.

El algoritmo secuencial originalmente fue implementado en [11] donde el tiempo para *n=28511* fue de 17 horas 10 minutos y 16 segundos en una PC dedicada con procesador Intel Core 2 Duo a una velocidad de 2.66 GHz y una memoria RAM de 2 GB, luego en [12] se realizaron optimizaciones basadas fundamentalmente en el desenrollado de ciclos y los tiempos fueron mejorados, para ese mismo tamaño de la entrada y PC, hasta 7 horas 43 minutos y 16 segundos. Si se tiene en cuenta que *n=28511* puede considerarse pequeño respecto al tamaño del problema que genera comúnmente la modelación de yacimientos lateríticos reales, se puede verificar mediante (3) que los tiempos son considerablemente grandes. En la descripción del algoritmo se propone una estrategia general que, en función del tamaño del problema, persigue obtener ganancia de tiempo, sin embargo, como puede verificarse en (4), hay otros parámetros que pueden influir en su rendimiento, tal es el caso del costo de realizar una comparación (*c*) y la cantidad de elementos (*m*) de cada una de las matrices obtenidas en la Fase I del algoritmo, que determina directamente el tiempo requerido para sumar dos de estas, a continuación se hacen algunas propuestas con el objetivo de optimizar la implementación considerando dichos parámetros.

III.A Representación de la matriz

El modelo a obtener es representado mediante un hipercubo de probabilidades condicionales, resultado de la suma de las matrices obtenidas en cada procesador. Si estas matrices son representadas en memoria como bloques lineales de elementos por cada dimensión (Variante Clásica), entonces la cantidad de elementos de cada una de ellas es el mismo e igual a

$$m=\left(d_1=\left|\propto_H\right|\right)\times\left(d_2=\left|\propto_V\right|\right)\times\left(d_3=\left|H\right|\right)\times$$

$$\left(d_4=\left|P\right|\right)\times\left(d_5=k\right)\times\left(d_6=k\right)$$

en consecuencia, el tiempo de para sumar dos de estas, está acotado por *T(m)=m*. Sin embargo, si existen altos valores de dispersión se puede considerar la posibilidad de sólo almacenar los valores distintos de cero y para cada uno de ellos el lugar que ocuparían en el modelo original. Supóngase que las matrices son representadas como colección de pares de la forma (*Elemento,Coordenadas*), así el componente *Coordenadas* debe almacenar los 6 índices que expresan el lugar que ocupa *Elemento* dentro del hipercubo original, haciendo:

1. Proceso de toma de muestras del suelo que persigue refinar la búsqueda de determinados minerales.

$$Coordenadas = \sum_{i=1}^{6} indice_i \times a_i \begin{cases} a_i = \prod_{k=i-1}^{6} d_k \ \ \forall i < 6 \\ a_6 = 1 \end{cases}$$

puede alcanzarse este objetivo; realizando el proceso inverso se logra obtener los índices a partir del valor *Coordenadas*. Suponiendo que el total de elementos distintos de cero es m_c, esta representación es más eficiente (respecto a la Variante Clásica) en cuanto a espacio ssi $m_c < m/2$. Sin embargo el tiempo para acceder a un determinado elemento no es constante, pues se requiere su búsqueda a lo largo de la colección, que si es representada como una lista entonces es $T(m_c) = O(m_c)$ pero si se representa como un árbol AVL (Variante AVL) entonces el tiempo es $T(m_c) = O(\log m_c)$.

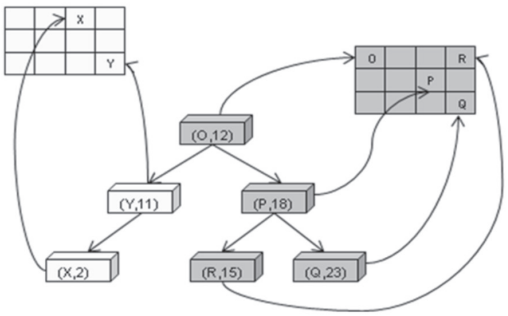

Figura 3. Variante AVL para un hipercubo de 3 dimensiones

Como ya se mencionaba, en la Fase II del algoritmo se realizan sumas matriciales, que siguiendo la Variante Clásica toma un tiempo *T(m)*, pero si se escoge la Variante AVL es $T(m_c) = O(m_c \log m_c)$, lo que establece que esta Fase del algoritmo será más rápida solo cuando $m_c \log m_c < m$, o lo que es lo mismo

$$m_c < \frac{m}{W(m)} \qquad (10)$$

donde *W* es la función *W de Lambert*[2]. Sea $f(m) = m / W(m)$ una función que acota superiormente a m_c. Como puede notarse *f(m)* es monótona creciente, por lo que si se puede encontrar un valor *x* tal que *f(m)<m/x* se puede afirmar que la Variante AVL es más óptima, en cuanto a tiempo respecto a Variante Clásica, siempre que se cumpla que m_c sea a lo sumo un *x%* de m; y en cuanto a espacio siempre que $m_c < m/4$ pues se debe almacenar por cada nodo del árbol, el valor de la probabilidad, su posición dentro del hipercubo, además de los dos punteros a los nodos hijos. El análisis antes planteado está referido a la Fase II del algoritmo, pues como puede verificarse en (11) llevando al límite el tamaño del problema, el tiempo empleado para la ejecución del algoritmo completo usando una representación clásica(T_p^{CL}) es menor que utilizando la Variante AVL(T_p^{AVL}).

$$\lim_{n\to\infty} \frac{T_p^{AVL}}{T_p^{CL}} = 1 + \log \sqrt[c]{m_c} \qquad (11)$$

IV. RESULTADOS Y DISCUSIÓN

Para la realización de los experimentos se implementó el algoritmo secuencial empleando las dos formas de representación de la matriz referidas en la sección III.A, pero debido a que la Variante Clásica secuencial es la más óptima en cuanto a tiempo, todas los análisis realizados sobre este parámetro están considerados respecto a dicha variante. Las pruebas fueron realizadas para la obtención de (9) a partir de datos de yacimiento lateríticos reales y fijando $|\alpha_V| = 6$, $|\alpha_H| = 16$, $|H| = 52$, $|P| = 9$ y $|K| = 6$ que es la propuesta inicial de sus autores [9], esto determina que el total de matrices de probabilidades de transición a calcular es $44928 = 6 \times 16 \times 52 \times 9$, cada una de $256(16 \times 16)$ elementos.

IV.A Pruebas en un entorno paralelo

El algoritmo se implementó en sus dos variantes en el lenguaje C usando la librería MPI, y los experimentos fueron realizadas en un Cluster Beowulf de 8 PC con procesador Intel(R) Core(TM)2 Duo CPU E4500 2.20GHz y memoria principal de 1(2 x 512) GB de RAM con sistema operativo Linux Debian Lenny 5.0 y una red a 100 Mbps.

Consideraciones de tiempo. Las ganancias de velocidad y eficiencias obtenidas para ambas variantes se muestran en la figura 4 y 5 respectivamente.

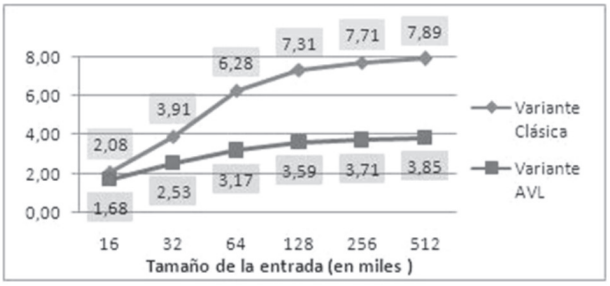

Figura 4: Speed-up usando 8 procesadores

2. Se define como la función inversa de $f(W) = W e^W$ o lo que es equivalente, la función *W(z)* que satisface la ecuación $W(z) e^{W(z)} = z$ para todo valor complejo *z*.

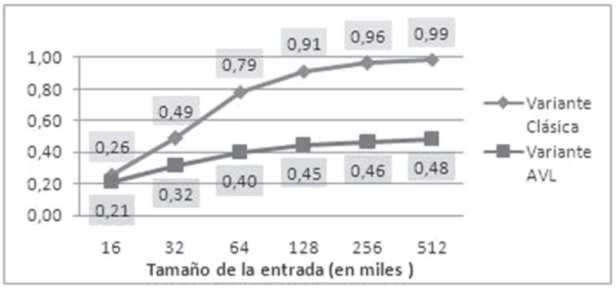

Figura 5. Eficiencia usando 8 procesadores

Como puede verse la Variante Clásica muestra mejores prestaciones que la Variante AVL, tendiendo a un **Speed-Up** y **Eficiencia** óptimos a medida que aumenta el tamaño del problema, no obstante puede comprobarse que la Variante AVL también crece en prestaciones bajo estas condiciones. El **Speed-up** de la Variante Clásica se comportó desde *1.2(n=16 mil)* hasta *2.07(n=512 mil)* veces mayor que la Variante AVL. La mayor ganancia de velocidad estuvo dada para *n=512 mil*, donde se disminuyó el tiempo desde 9 horas 30 minutos y 19 segundos a 1 hora 18 minutos 5 segundos.

En la figuras 6 y 7 se muestra el desempeño del algoritmo en su Variante Clásica en función del tamaño del problema usando 2, 4 y 8 procesadores. Como puede verse tanto el **Speed-Up** como la **Eficiencia** aumenta al tiempo en que lo hace el número de procesadores y el tamaño de la entrada, tendiendo a su valor óptimo teórico.

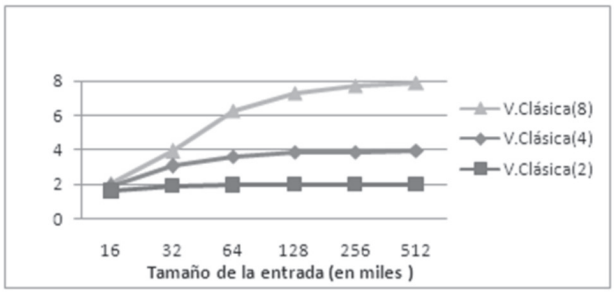

Figura 6: Speed-up usando 2, 4 y 8 procesadores

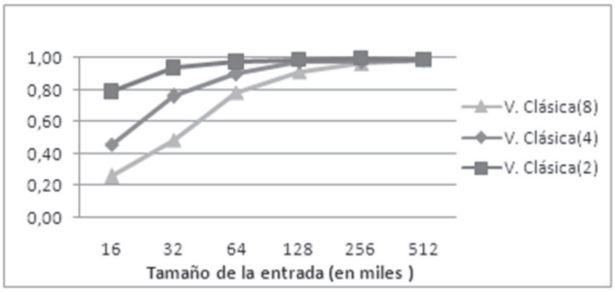

Figura 7: Eficiencia usando 2, 4 y 8 procesadores

Veamos cómo se comporta la Escalabilidad del algoritmo, según [8] un sistema es escalable si es posible tener incrementalmente valores de n y p para los cuales la Eficiencia se mantiene constante (al menos). Como puede observarse en la tabla I es posible alcanzar este objetivo si se combinan adecuadamente estos parámetros.

Tabla I. Eficiencia en función de *n* y *p*

n	*p=2*	*p=4*	*p=8*
16 mil	0,789	0,460	0,260
32 mil	0,935	0,760	0,489
64 mil	**0,973**	0,899	0,785
128 mil	0,989	**0,973**	0,914
256 mil	0,991	0,976	0,964
512 mil	0,993	0,985	**0,983**

Consideraciones de memoria. Como se mencionaba en la sección III.A el espacio requerido para representar el hipercubo siguiendo la Variante Clásica es constate. Para los parámetros considerados en las pruebas y referidos en el primer párrafo de esta sección, la memoria necesaria es de *43,875* MB. Sin embargo cuando se emplea la Variante AVL se observó que el grado de dispersión del modelo fluctuó de un 2 a un 3% con un nivel de confianza de un 99 %, lo que provocó una disminución significativa del total de memoria empleado para su obtención, puede verificarse mediante (10) que con un grado de dispersión, en este caso, menor al 13% se obtienen ganancias tanto en tiempo durante la Fase II del algoritmo, como en espacio.

En la tabla II se muestra el consumo de memoria para la representación del modelo siguiente la Variante AVL para diferentes grados de dispersión obtenidos durante las pruebas.

Tabla II. Memoria requerida por la Variante AVL

Dispersión (%)	*Espacio(MB)*
2.0	3.51
2.3	4.04
2.5	4.39
2.7	4.74
3.0	5.27

En ella se muestra como la Variante AVL necesitó como promedio 10.19 veces menos memoria que la Variante Clásica. Es evidente el compromiso entre la ganancia en tiempo y la ganancia en memoria, el uso de una variante o la otra queda condicionada entonces a la prioridad establecida a estos parámetros.

IV.B Pruebas en un entorno distribuido

Es bien conocido que las redes de área local aglutinan estaciones de trabajo cuyo poder de cómputo no es totalmente explotado, algunas investigaciones han estudiado con precisión cual es el uso de las CPUs de estas estaciones de escritorio

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

[13]–[15], otras muestran que el aprovechamiento de los ciclos ociosos de estas últimas es una alternativa importante para alcanzar elevadas prestaciones [16]–[18] y en consecuencia resolver problemas complejos.

Partiendo de esta premisa, el algoritmo fue implementado además siguiendo el paradigma *High Throughput Computing* mediante el uso del sistema distribuido T-arenal, el cual incluye un grupo de adaptaciones y mejoras [19], [20] respecto al presentado originalmente en [21], [22]. T-arenal está desarrollado sobre Java y como objetivo fundamental tiene el aprovechamiento de los recursos computacionales conectados mediante una red de área local, independientemente de su heterogeneidad. Las pruebas fueron realizadas usando hasta 30 estaciones de trabajo conectadas mediante una red local de 100 Mbps, todas con un procesador Intel(R) Core(TM)2 Duo CPU E4500 2.20GHz, memoria principal de 1(2 x 512) GB de RAM, y sistema operativo Windows XP.

Como puede verificarse en las Figuras 8 y 9, que muestran respectivamente, las mayores ganancia de velocidad y eficiencia alcanzadas por el sistema durante las pruebas[3], los tiempos fueron reducidos hasta 11.3 veces (de 21 horas con 5 mins a 112 mins) y la eficiencia llegó hasta un 51%.

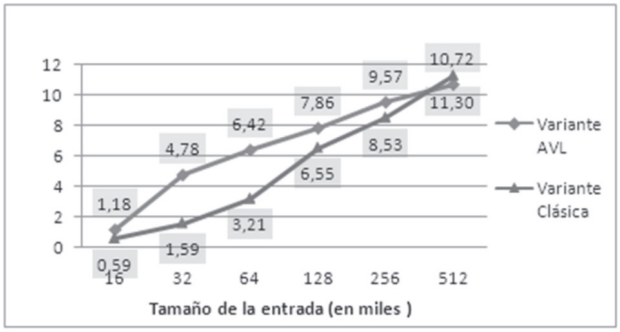

Figura 8: Speed-Up Máximo

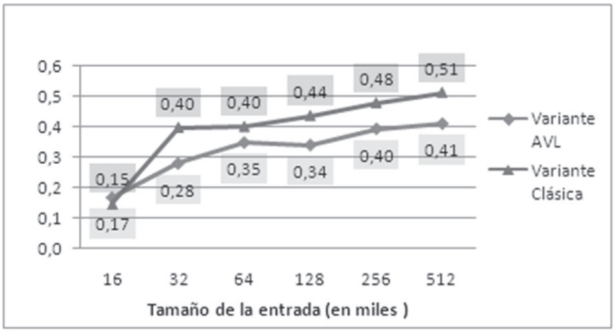

Figura 9. Eficiencia Máxima

Como puede apreciarse para ambos parámetros su valor aumentó a medida que lo hacía el tamaño de la entrada, la misma tendencia que se muestra en la sección IV.A, lo que significa que el algoritmo propuesto es más ventajoso aplicarlo, y usa los procesadores disponibles de forma más efectiva, para problemas de mayor dimensión, hecho este que confirma lo planteado en la sección II.C. Otro parámetro interesante de analizar para sistemas de este tipo, debido a que las unidades de cómputo son no dedicadas, es la desviación estándar observada en la cantidad de PC utilizadas (Figura 10) durante la ejecución del algoritmo, la cual aumentó a medida que lo hacía el tamaño del problema, evidentemente la probabilidad de perder estaciones de trabajo aumenta a medida en que lo hace el tiempo necesario para resolver el problema.

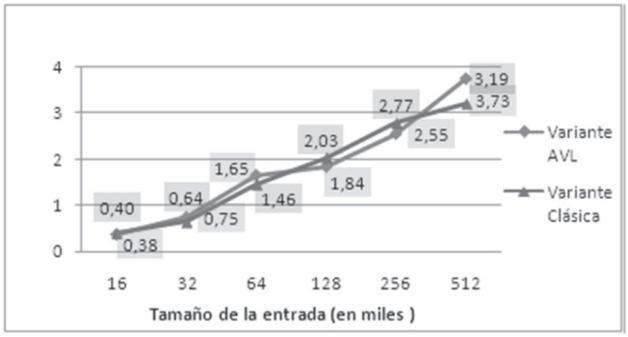

Figura 10: Número de PC Utilizadas (DesvEst)

En general, al usar un árbol AVL para representar las matrices se obtuvieron mejores resultados, esto no representa una contradicción con lo visto en la sección III.A, allí se hizo la salvedad que a pesar de que el tiempo total del algoritmo empleando el árbol sería mayor, el tiempo de la Fase II disminuiría bajo ciertas condiciones. La razón principal de estos resultados, aparejado al cumplimiento de estas condiciones, está en la arquitectura del sistema empleado: cliente-servidor, pues y teniendo en cuenta que la complejidad de esta Fase es constante respecto a la Fase I, se decidió realizar las sumas matriciales a medida que respondían los clientes, esto provocó que el grado de dispersión de la matriz influyera de forma determinante en el tiempo total para resolver el problema.

V. CONCLUSIONES

Se presentó un algoritmo paralelo escalable, óptimo en cuanto a Ganancia de Velocidad y Eficiencia, para la obtención de matrices de probabilidades de transición, se determinó además el Desequilibrio de Carga y el Número Óptimo de Procesadores para su ejecución en un Sistema Multiprocesador. Se implementó sobre un cluster Beowulf y distribuido sobre una red local de estaciones de trabajo, donde se corroboró de forma práctica los resultados obtenidos teóricamente. Se constató las potencialidades que ofrece la *High Throughput Computing*, que aunque inicialmente estuvo dirigida a resolver problemas de alto

3. Se realizaron 360 ejecuciones (180 por variante), 60 por cada tamaño de la entrada.

costo computacional a largo plazo, con el rápido crecimiento del poder de computo de las estaciones de trabajo es posible obtener elevadas ganancias de velocidad y en consecuencia reducir considerablemente el tiempo necesario para resolver problemas de este tipo. Se propuso una representación matricial basada en árboles AVL para hipercubos dispersos y se aplicó en la obtención del modelo planteado en [9], determinándose además las cotas o valores de dispersión límites para los cuales su uso en el algoritmo presentado es más eficiente, en cuanto a tiempo y espacio, que la representación estándar.

REFERENCIAS

[1] P. Bierzychudek, "Looking backwards: assessing the projections of a transition matrix model," 1999.

[2] D. T. Crouse, L. B. Crowder, and H. Caswell, "A stage-based population model for loggerhead sea turtles and implications for conservation," 1987.

[3] N. Enright and J. Ogden, "Applications of transition matrix models in forest dynamics: Araucaria in papua new guinea and nothofagus in new zealand," 1979.

[4] M. L. Morrison, B. G. Marcot, and R. W. Mannan, Wildlife-habitat relationships: concepts and applications, Island Pr, 2006.

[5] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," 1970.

[6] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," 1990.

[7] F. Almeida, D. Gim´enez, J. M. Mantas, and A. M. Vidal, Introducción a la programación paralela, Thomson Paraninfo, 2008.

[8] V. Kumar, Ananth Grama, Anshul Gupta, and George Karypis, Introduction to Parallel Computing. Design and Analysis of Algorithms, The Benjamin/Cummings Pub. Company, Redwood City, California, 1994.

[9] R. Pe˜na, "Modelo matemático para la optimización de las redes de exploración y explotación en yacimientos lateríticos," in II Convención Cubana de Ciencias de la Tierra, La Habana, Cuba, 2007.

[10] R Peña, Matos Elias L, Ortiz Romero E, and Robles Labacena V, "Propuesta de clases patrones en yacimientos lateríticos ferro-niquelíferos," in II Convención Cubana de Ciencias de la Tierra, La Habana, Cuba, 2007.

[11] R Peña, "Algoritmo de conteo para modelos markovianos en yacimientos lateríticos," in COMPUMAT, La Habana, Cuba, 2007.

[12] A M Ramírez, R E Peña, and L Y Broscat, "Mejora de un algoritmo de conteo para modelos markovianos en yacimientos later´ıticos," in COMPUMAT, La Habana, Cuba, 2009.

[13] R.H. Arpaci, A.C. Dusseau, A.M. Vahdat, L.T. Liu, T.E. Anderson, and D.A. Patterson, "The interaction of parallel and sequential workloads on a network of workstations," in Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems. ACM, 1995, pp. 267–278.

[14] A. Acharya, G. Edjlali, and J. Saltz, "The utility of exploiting idle workstations for parallel computation," ACM SIGMETRICS Performance Evaluation Review, vol. 25, no.1, pp. 225–234, 1997.

[15] S. Smallen, H. Casanova, and F. Berman, "Tunable on-line parallel tomography," in Proceedings of SuperComputing 01, Denver, Colorado, 2001.

[16] H. Casanova, D. Zagorodnov, F. Berman, and A. Legrand, "Heuristics for scheduling parameter sweep applications in grid environments," in Proceedings of the 9th Heterogeneous Computing Workshop (HCW 00). Published by the IEEE Computer Society, 2000, p. 349.

[17] D. Kondo, M. Taufer, CL Brooks, H. Casanova, and AA Chien, "Characterizing and evaluating desktop grids: An empirical study," in Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, 2004.

[18] P Domingues, P Marques, and L Silva, "Resource usage of windows computer laboratories," in Parallel Processing, 2005. ICPP. International Conference Workshops on Parallel Processing, Oslo, Norway, 2005, pp. 469–476.

[19] L. Aguilera, Sistema de cómputo distribuido aplicado a la Bioinformática, Tesis de maestría, Universidad de las Ciencias Informática, 2008.

[20] C R; Jacas, L; Aguilera, and D Mariño, "Platform of distributed task v2.0," Laboratorio Nacional de Computación, Petrópolis, Rio de Janeiro, Brasil, 2010.

[21] Thomas Keane, A General Purpose Heterogeneous Distributed Computing System, Msc thesis, National University of Ireland, 2004.

[22] A; Page, T; Keane, R; Allen, T. J; Naughton, and J Waldron, "Multitiered distributed computing platform.," in Second International Conference on the Principles and Practice of Programming in Java, J. F. Power editions and J. T. Waldron, Eds., Kilkenny City, Ireland, 2003, p. 191.

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663