Algoritmo Rushdi Optimizado
Software RMES

Centro de Desarrollo de Gestión Empresarial.
Poniente 1206 – Viña del Mar, Chile.
Fono:(56) (32)688987 – Fax:(56) (32)2684079
empresa@mes.cl

Noviembre, 2009

1. Introducción

Este algoritmo es usado para el cálculo de confiabilidad de sistemas en redundancia o fraccionamiento. Estos sistemas se conocen como k-out-of-n:G, lo cual significa que el sistema funciona sí y sólo sí k de sus n componentes están operativos. La nomenclatura complementaria, k-out-of-n:F, se refiere a que el sistema falla sí y sólo sí k de los n componentes fallan. RMES utiliza la primera definición.

El presente trabajo utiliza las definiciones existentes en el documento "Algoritmo Rushdi", por lo cual, es necesario leer este documento antes de abordar el actual.

2. El algoritmo optimizado

Para implementar una optimización del algoritmo original, es necesario comprender a cabalidad el original.

Para ello se explicará el algoritmo de forma gráfica, de modo de entenderlo paso a paso. Por ejemplo, digamos que se tiene un subsistema 3-out-of-4:G. Supongamos que los valores de la confiabilidad de los 4 subsistemas se conocen y se almacenan en el vector "P":

$$P = \left[\begin{array}{cccc} p_0 & p_1 & p_2 & p_3 \end{array} \right]$$

El algorimo original crearia la siguiente matriz A:

$$A = \left(\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right)$$

Y luego igualaria la primera fila a 1.

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Luego, se comenzaria por calcular la celda $a_{1,1}$ usando los valores de las celdas $a_{0,0}$ y $a_{1,0}$ junto con el valor p_0 del vector:

$$a_{1,1} = p_0 \cdot a_{0,0} + (1 - p_0) \cdot a_{1,0}$$

Notar que $a_{0,i} = 1$ y $a_{1,0} = 0$, por lo tanto, para este caso particular $a_{1,1} = p_0$

$$A = \begin{pmatrix} a_{0,0} & 1 & 1 & 1 & 1 \\ a_{1,0} & a_{1,1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Continuando con el algoritmo, se procede a calcular la siguiente celda, $a_{1,2}$, el cual se obtendra de las celdas $a_{0,1}$ y $a_{1,1}$ junto con el valor p_1 del vector:

$$a_{1,2} = p_1 \cdot a_{0,1} + (1 - p_1) \cdot a_{1,1}$$

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & 1 & 1 & 1 \\ a_{1,0} & a_{1,1} & a_{1,2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Finalmente las ultimas celdas se calcularian de la siguiente manera:

$$a_{1,3} = p_2 \cdot a_{0,2} + (1 - p_2) \cdot a_{1,2}$$
$$a_{1,4} = p_3 \cdot a_{0,3} + (1 - p_3) \cdot a_{1,3}$$

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & 1 \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

En la siguiente iteración, el calculo comienza en la celda $a_{2,2}$ y algoritmo es idéntico:

$$a_{2,2} = p_1 \cdot a_{1,1} + (1 - p_1) \cdot a_{2,1}$$

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & 1 \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ 0 & a_{2,1} & a_{2,2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$a_{2,3} = p_2 \cdot a_{1,2} + (1 - p_2) \cdot a_{2,2}$$

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & 1 \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ 0 & a_{2,1} & a_{2,2} & a_{2,3} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$a_{2,3} = p_4 \cdot a_{1,3} + (1 - p_3) \cdot a_{2,3}$$

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & 1 \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ 0 & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Y en la siguiente y última iteración el calculo comienza en la celda $a_{3,3}$ hasta llegar a la celda $a_{3,4}$, valor que retorna el algoritmo.

$$a_{3,3} = p_2 \cdot a_{2,2} + (1 - p_2) \cdot a_{3,2}$$

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & 1 \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ 0 & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ 0 & 0 & a_{3,2} & a_{3,3} & 0 \end{pmatrix}$$

$$a_{3,4} = p_3 \cdot a_{2,3} + (1 - p_3) \cdot a_{3,3}$$

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & 1 \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ 0 & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ 0 & 0 & a_{3,2} & a_{3,3} & a_{3,4} \end{pmatrix}$$

De todo el proceso se pueden desprender que en cada paso unitario del algoritmo se necesita únicamente saber el valor de la celda superior izquierda y de la celda inmediatamente izquierda respecto de la actual. Mientras el algoritmo avanza, los valores que se calcularon para las primeras filas no se vuelven a utilizar. Esto lleva a pensar de que sólo es necesario utilizar un vector de tamaño n (la cantidad de subsistemas hijos) que almacene los valores por cada iteración.

Teniendo presente este esquema, es necesario aislar la primera iteración del resto, debido a que los valores iniciales de las celdas superiores deben ser igual a 1, lo que implica que la fórmula puede ser modificada para este caso particular.

Ubicaremos gráficamente un vector V dentro la matriz A para observar el comportamiento del algoritmo modificado. El vector V almacenará los valores del algoritmo.

$$V = \begin{bmatrix} v_0 & v_1 & v_2 & v_2 \end{bmatrix}$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & v_0 & v_1 & v_2 & v_3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Para la primera iteración se cumple que $a_{0,i} = 1$, por lo tanto la fórmula puede ser modificada de la siguiente manera:

$$a_{1,j} = p_{j-1} + (1 - p_{j-1}) \cdot a_{1,j-1} \quad \forall_{j=1\ldots n}$$

Reemplazando para el vector V:

$$v_j = p_j + (1 - p_j) \cdot a_{1,j} \quad \forall_{j=0\ldots n-1}$$

Para eliminar la dependencia con la matriz A, el valor de la celda $a_{1,j}$ simplemente es el valor de la celda calculada en el paso anterior, la cual puede ser referenciada directamente. Para primer elemento v_0, v_{-1} no existe y se considera con valor 0, por ende, la fórmula puede ser aún más concisa.

$$v_0 = p_j$$
$$v_j = p_j + (1 - p_j) \cdot v_{j-1} \quad \forall_{j=1\ldots n-1}$$

Gráficamente, el algoritmo seria el siguiente:

$$v_0 = p_0$$
$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & v_0 & - & - & - \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$v_1 = p_1 + (1 - p_1) \cdot v_0$$
$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & v_0 & v_1 & - & - \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$v_2 = p_2 + (1 - p_2) \cdot v_1$$
$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & v_0 & v_1 & v_2 & - \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$v_3 = p_3 + (1 - p_3) \cdot v_2$$
$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & v_0 & v_1 & v_2 & v_3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Para la siguiente iteración, es necesario modificar el criterio anterior. El valor de la celda v_1 es el que debe ser calculado, entonces, siguiendo la fórmula:

$$v_1 = p_1 \cdot \widehat{v_0} + (1 - p_1) \cdot v_0$$
$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & \widehat{v_0} & - & v_2 & v_3 \\ 0 & v_0 & v_1 & - & - \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

En donde $\widehat{v_i}$ se refiere al valor antiguo de la celda i del vector V antes de ser calculada. Particularmente, para el primer paso de la iteración, no se realiza cálculo sobre v_0, además, su valor es $v_0 = 0$, y se debe de recordar que $\widehat{v_i}$ tiene un valor calculado en la iteración anterior, por lo tanto, sólo es necesario modificar la fórmula:

$$v_1 = p_1 \cdot \widehat{v_0}$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & \widehat{v_0} & - & v_2 & v_3 \\ 0 & 0 & v_1 & - & - \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

El problema de esto es que se pierde el valor original de v_1, el cual se utiliza en el paso siguiente de la iteración. Por lo tanto, el valor de v_1 debe ser almacenado en una variable auxiliar antes de realizar el calculo. Esta variable se llamará "actualOldValue", o AOV

$$AOV = v_1$$

$$v_1 = p_1 \cdot \widehat{v_0}$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & \widehat{v_0} & AOV & v_2 & v_3 \\ 0 & 0 & v_1 & - & - \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Para la siguiente iteración, es necesario utilizar el valor AOV y v_1 para calcular el nuevo valor de la celda v_2, pero también es necesario almacenar el valor de la celda v_2 anterior al cálculo, dado que se necesita el valor antiguo para el cálculo de la siguiente celda en la iteraciíon. Se utilizará la variable AOV para almacenar el valor de v_2 previo al calculo, pero ademas, se debe almacenar el valor que AOV tenía almacenado. Para ello, se utilizará la variable "upperOldValue", o UOV. Esta se utilizará para realizar el cálculo de la celda actual v_2. Entonces:

$$UOV = AOV$$

$$AOV = v_2$$

$$v_2 = p_2 \cdot UOV + (1 - p_2) \cdot v_1$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & \widehat{v_0} & UOV & AOV & v_3 \\ 0 & 0 & v_1 & v_2 & - \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Para la siguiente iteración, no es necesario agregar más variables, sólo actualizar las existentes:

$$UOV = AOV$$

$$AOV = v_3$$

$$v_3 = p_3 \cdot UOV + (1 - p_3) \cdot v_2$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & \widehat{v_0} & - & UOV & AOV \\ 0 & 0 & v_1 & v_2 & v_3 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Para la última iteración, se ocupa el mismo criterio anterior. Sabemos que $v_1 = 0$, entonces el primer paso de la iteración sería:

$$AOV = v_2$$

$$v_2 = p_2 \cdot \widehat{v_1}$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & - & - & - & - \\ 0 & - & \widehat{v_1} & AOV & v_3 \\ 0 & - & 0 & v_2 & 0 \end{pmatrix}$$

El ultimo paso de la iteración sería:

$$UOV = AOV$$

$$AOV = v_3$$

$$v_3 = p_3 \cdot UOV + (1 - p_3) \cdot v_2$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & - & - & - & - \\ 0 & - & \widehat{v_1} & UOV & AOV \\ 0 & 0 & 0 & v_2 & v_3 \end{pmatrix}$$

El codigo de este algoritmo es el siguiente:

```
private static double rushdiGeneral(double[] ssR, int k) {

  int n = ssR.length;
  double upperOldValue;
  double actualOldValue;
  double array[] = new double[n];
  int i, j;

  array[0] = ssR[0];
  for (j = 1; j < n; j++) {
    array[j] = ssR[j] + (1 - ssR[j]) * array[j - 1];
  }

  for (i = 1; i < k; i++) {
    actualOldValue = array[i];
    array[i] = ssR[i] * array[i - 1];

    for (j = i + 1; j < n; j++) {
      upperOldValue = actualOldValue;
      actualOldValue = array[j];
      array[j] = ssR[j] * upperOldValue + (1 - ssR[j]) * array[j - 1];
    }
  }

  return array[n - 1];

}
```

En donde ssR es el vector de contiene la confiabilidad de los hijos de la maquina y k es la mñima cantidad de equipos en operación. El valor de k puede ir entre 1 y n, donde n es la cantidad de elementos del vector ssR.

Para el caso particular $k = 1$ se implementó un algoritmo recursivo que tiene mejor performance que el presente algoritmo, pero únicamente para un valor de $k = 1$, para cualquier otro valor, su performance es pobre y de hasta menor calidad que el algoritmo rushdi original.

El codigo se basa en la siguiente fórmula:

$$R(i, j) = p_j R(i - 1, j - 1) + q_j R(i, j - 1)$$

Y esta es su implementación:

```
private static double rushdiOneMinToWork(double[] ssR, int n)
  if (n == 0) {
    return 0;
  }
  return ssR[n - 1] + (1 - ssR[n - 1]) * rushdiOneMinToWork(ssR, n - 1);
}
```

En donde ssR es el vector de contiene la confiabilidad de los hijos de la maquina y n un parámetro que varía con la recursión, inicialmente es la cantidad de elementos del vector ssR.