MR
ISSN 1405-6690

Revista del Centro de Investigación

Universidad La Salle

Vol. 3 No. 11 Agosto 1998
$50.00

EDUCACIÓN

INGENIERÍA

Esta publicación tiene un tiraje de 1000 ejemplares y aparece semestralmente.
Impreso en Artes Gráficas Panorama, con domicilio en: Calle Avena No. 629 Col. Granjas México, CP 08400

REASIGNACIÓN DE TAREAS EN UN SISTEMA DISTRIBUIDO UTILIZANDO ALGORITMO GENÉTICO

Mario Farias[1] y Guillermo Morales[2]

[1]**Laboratorio del Centro de Investigación, Universidad La Salle**
Benjamin Franklin 47, Col. Hipódromo-Condesa, México DF 06170, email: mfarias@ci.ulsa.mx
[2]**Sección de Computación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN**

RESUMEN

Una de las características que debe tener un sistema distribuido es la capacidad de recuperación ante una eventualidad. Una falla de un procesador es una eventualidad de la cual el sistema debe de recuperarse lo más rápido posible y de manera satisfactoria, reasignanando las tareas de este procesador en los procesadores restantes. En la actualidad los métodos clásicos, como *recocido simulado*, son lentos para este tipo de situaciones. En este trabajo se muestra la utilización de la técnica de algoritmos genéticos para resolver el problema de reasignación de tareas, de tal forma que se obtiene como resultado una asignación que mantenga una carga de trabajo equilibrada entre los procesadores involucrados, así como una respuesta en un tiempo mucho menor que los métodos clásicos.

ABSTRACT

A characteristic that a distributed system should have is the ability of recovery after failure. A failure of one processor requires a fast and reliable recovery to reallocate the tasks enqueued into the failing processor among the remaining processors. The classic methods, as simulated annealing, are very slow to solve this problems. In this paper we introduce a solution for the task allocation problem by using genetic algorithms. We get as a result the task reallocation where the load is balanced among the processors with better time response that with the classic methods.

INTRODUCCIÓN

Uno de los primeros problemas encontrados en la operación de un sistema distribuido es la asignación de las tareas en los procesadores involucrados (1). El encontrar la mejor asignación de las tareas en los procesadores de un sistema distribuido puede ser formulado como un problema de optimización.

Los problemas de asignación se resuelven, por lo general, por un procedimiento de costo eficiente que encuentre la asignación óptima para instancias específicas del problema. Como una regla, los problemas de asignación tienden a ser computacionalmente intensos, es decir, muy tardados cuando se requiere de una respuesta rápida

En este trabajo, utilizamos la técnica de algoritmos genéticos para la reasignación de las tareas inmediatamente después de que ocurra una falla en algún procesador.

También se debe de tomar en cuenta que el desequilibrio de las cargas de trabajo en los procesadores afecta directamente el rendimiento del sistema (2).

DESARROLLO

El sistema consiste de *m* procesadores idénticos que ejecutan *n* tareas. Cada tarea posee *r* copias, o réplicas, cada una ejecutada por un procesador distinto. Cada procesador puede tener asignadas varias tareas, las cuales las atiende en diferentes segmentos de tiempo. Llamemos *frecuencia* de una tarea en un procesador al número de veces que el procesador pasa a atender esa tarea. Llamemos *densidad* de la tarea en el procesador al máximo número de instrucciones propias de esa tarea que ejecuta el procesador en los segmentos de tiempo que le asigna a esa tarea. Para cada tarea, sus frecuencias y densidades en los procesadores determinan un «peso específico de esa tarea» que en este contexto se llama

utilización de la tarea. Denotemos por u_j a la utilización de la *j*-ésima tarea.

Una asignación particular queda descrita por una matriz $\boldsymbol{V} = \left(V_{ij}\right)_{1 \le i \le m}^{1 \le j \le n}$ de orden $m \times n$, con entradas 0 ó 1. De hecho, para cada *i* y cada *j*, se tiene $V_{ij}=1$ si la *j*-ésima tarea está asignada al *i*-ésimo procesador, y $V_{ij}=0$ en otro caso. En estas condiciones la ecuación (Ec. 1) representa la *carga de trabajo* del *i*-ésimo procesador bajo la asignación ***V***:

$$p_i = \sum u_j V_{ij} \qquad \text{(Ec. 1)}$$

De acuerdo con (3), minimizar la suma de todos los valores p_i^2 minimiza también la varianza estadística de los p_i's, el cual viene a ser una medida del desequilibrio en las cargas de los procesadores. Es bien sabido que un desequilibrio de cargas afecta directamente el rendimiento del sistema.

Tomando en cuenta lo anterior y la (Ec. 1), el problema puede definirse como la minimización de:

$$\sum_{i=1}^{m}\left(\sum_{j=1}^{n} u_{ij}V_{ij}\right)^2 \qquad \text{(Ec. 2)}$$

restringido a:

$$\sum_{i=1}^{m} V_{ij} = r_j \quad \forall\, 1 \le j \le n \qquad \text{(Ec. 3)}$$

El problema de asignación de tareas puede modelarse por una ecuación que es la suma de las ecuaciones (Ec. 2) y (Ec. 3) quedando como:

$$E^* = a\sum_{j=1}^{n}\left(\sum_{i=1}^{m} V_{ij} - r\right)^2 + b\sum_{i=1}^{m}\left(\sum_{j=1}^{n} u_j V_{ij}\right)^2 \qquad \text{(Ec. 4)}$$

donde los parámetros involucrados tienen los significados descritos a continuación:

n es el número de tareas existentes,
m es el número de procesadores disponibles,
r es el número de réplicas de cada tarea,
V_{ij} es el valor de asignación de la j-ésima tarea al i-ésimo procesador, es decir,

$$V_{ij} = \begin{cases} 1 & \text{si la tarea } j \text{ esta en el procesador } i \\ 0 & \text{en otro caso} \end{cases}$$

u_j es la utilización de la tarea *j*, y
a,b son ponderaciones a los sumandos de la función E^* en la (Ec. 4).

*Puede verse que el primer término de la función E^* en la (Ec. 4) está motivado por el número de réplicas para cada tarea. Este término alcanza un mínimo si cada tarea es ejecutada exactamente por *r* procesadores. El segundo término, por otro lado, representa a la función de costo, que como ya se mencionó antes, es una medida del desequilibrio de las cargas de los procesadores.

La técnica de *recocido simulado (simulated annealing)* es un procedimiento de optimización que produce aproximaciones a valores óptimos, dada una función *objetivo*. Veamos la manera en que procede:

Procedimiento RS: Dado un punto inicial en el espacio de búsqueda con su respectivo valor para la función objetivo, a éste se le considera como punto actual. En cada iteración de este procedimiento, se busca un vecino en forma aleatoria, si dicho vecino genera un valor en la función objetivo menor al del punto inicial, éste nuevo vecino toma el lugar del punto inicial, en caso contrario se obtiene un valor de cambio basado en el exponente de la variación y se genera en forma aleatoria una probabilidad, si el cambio es mayor a la probabilidad, el punto vecino se convierte en el punto inicial. Esto se repite en tanto no se arribe a condiciones terminales.

La técnica de algoritmos genéticos (*genetic algorithms*) es un procedimiento, que en este caso, se utiliza para optimización que produce aproximaciones a valores óptimos, con una probabilidad alta de que las aproximaciones sean, en efecto, los óptimos globales. Dada una función, llamada *objetivo*, busca el valor mínimo de esa función (4). A grandes rasgos la forma en que procede es:

Procedimiento AG: Dado un conjunto de puntos en el espacio de búsqueda, llamado *seminal*, se evalúa la función objetivo en ellos y se escoge los puntos correspondientes a los mejores valores. Estos puntos forman una muestra de mejores individuos a «reproducirse», es decir forman una clase de *padres*. En un proceso de *entre-cruzamiento* para generar *hijos*, se obtiene a estos últimos

de los padres. A los hijos se les muta con probabilidades determinadas y se actualiza el conjunto seminal considerando padres e hijos. Se repite este ciclo «selección - entrecruzamiento - mutación» hasta obtener ya sea el valor mínimo de la función o bien hasta un número fijo de generaciones (5).

En la Figura 1 se muestra la forma en que se programó el algoritmo.

Como se ve, este procedimiento tiene varias particularidades. Criterios específicos para realizar las operaciones de selección, de entrecruzamiento y de mutación, así como para detener el procedimiento, dan algoritmos genéticos con comportamientos diversos. En un problema particular, la fijación de esos criterios es un factor importantísimo para resolver eficientemente el problema con estos algoritmos.

Posteriormente se realiza una búsqueda por vecinos. Esto es debido a que los Algoritmos Genéticos y el Recocido Simulado son algoritmos de aproximación y muchas veces no alcanzan el valor óptimo global, una forma de alcanzar el valor óptimo es realizando una búsqueda por vecinos.

Para ello considérese la matriz $V = \left(V_{ij}\right)_{1\le i\le m}^{1\le j\le n}$ de orden $m \times n$ con entradas *ceros* o *unos*. Sea $A \in V$

$$A^{ij} = \left(a_{kl}^{ij}\right) \Rightarrow \forall k,l:$$

$$a_{kl}^{ij} = \begin{cases} a_{kl} & si\ (k,l) \neq (i,j) \\ \overline{a}_{kl} & si\ (k,l) = (i,j) \end{cases} \qquad \text{(Ec. 5)}$$

El algoritmo procede como sigue:

Procedimiento BV: Dado un punto inicial en el espacio de búsqueda con su respectivo valor para la función objetivo, a éste se le considera como punto *actual*. En cada iteración de este procedimiento, se busca entre los posibles vecinos del punto actual a un vecino cuyo valor en la función objetivo sea menor al del actual, si se localiza a tal vecino a éste se le considera como el punto actual con fines de una nueva iteración. Esto se repite en tanto no se llegue a condiciones terminales.

También en este caso, las particularidades que determinan a un algoritmo específico en esta clase están dadas por la noción de *vecindad* de un punto, por la selección de un buen vecino y por las condiciones de terminación.

```
Entrada:   func      : Función objetivo
           probmut   : Probabilidad de mutación
           n x m     : Orden de la matriz de asignación
           maxgen    : Número máximo de generaciones
           np        : Número de padres
           nc        : Número de entrecruzamientos
Salida:    V         : Matriz de asignación
Algoritmo:
   {       nh = np^(nc+1)
           generarpoblacion(hijos,nh);
           obtenerpadres(padres,hijos,np);
           for gen = 0 to gen = maxgen {
                   generarhijos(hijos,padres,nh);
                   mutarpoblacion(hijos,probmut);
                   agregarpadres(hijos,padres);
                   obtenerpadres(padres,hijos,np);
           }
   }
```

Figura 1. Esquema del Algoritmo Genético.

```
Entrada:    func        : Función objetivo
            Aini        : Matriz inicial
Salida:     V           : Matriz de asignación
Algoritmo:
    {       Anueva = Aini;
            Vini = func(Aini);
            Anueva = MayorVariacion(Aini);
            Vnueva = func(Anueva);
            while (Vnueva < Vini) {
                    Vini = Vnueva;
                    Anueva = MayorVariacion(Aini);
                    Vnueva = func(Anueva);
            }
    }
```

Figura 2. Esquema de la búsqueda por vecinos

En nuestro problema de asignación el espacio de búsqueda está conformado por las matrices de orden $m \times n$, con entradas 0 o 1. Para el procedimiento AG hacemos la selección evaluando la función objetivo en las matrices seminales, y para un número *K* prefijado de *padres*, elegimos aquellas *K* matrices que den los valores mínimos. En nuestro caso *K=4*. En el entrecruzamiento se parte cada matriz padre en *n bloques de m entradas contiguas por renglones*. Obtenemos pues $K \cdot n$ bloques. A la nueva generación de matrices la obtenemos recombinando estos bloques en posiciones correspondientes, es decir, si un bloque aparece en la posición *k*, en una matriz padre, ha de aparecer en esa misma posición cuando aparezca en un hijo. Esto genera K^n matrices *hijas*. Finalmente, para la mutación, se hace un cambio en cada entrada de las matrices *hijas* por su valor complementario, pero sólo en cierto porcentaje de las entradas totales de la población.

Ahora bien, en el procedimiento BV, teniendo una matriz actual, para obtener una matriz vecina se copia la matriz actual y sólo en una entrada se le asigna el valor complementario. Este se realiza para cada entrada de la matriz, y aquella que dé la mayor diferencia en el valor de la función objetivo es la que se selecciona como la matriz vecina.

El espacio de búsqueda posee $2^{m \times n}$ elementos, que para números *m=7* y *n=14*, utilizados en nuestras prácticas, es sumamente grande. De manera típica, nuestro procedimiento AG utiliza un conjunto seminal de 2^{28} matrices y calcula 40 generaciones. En un sistema SPARC de 50 MHz. el tiempo de cálculo rondaba los 75 minutos, tomando en cuenta que la probabilidad de mutación varía desde *0.05* hasta *0.95* en incrementos de *0.05*, y en cada probabilidad se calculan 40 generaciones.

Una característica que nos es importante en nuestra plataforma de experimentación es la «estabilidad del problema». Así, nos interesa calcular asignaciones óptimas habiendo perturbado los parámetros del problema, particularmente el vector de «utilizaciones», partiendo de la solución óptima del problema sin perturbaciones. De manera que no nos alejásemos mucho de los óptimos calculados, procedimos por el algoritmo BV para calcular los óptimos correspondientes a los problemas perturbados.

Un factor importante que se observó es que la probabilidad de mutación que dio mejores resultados fue de 0.35.

RESULTADOS

A continuación se presentan los resultados de algunas simulaciones realizadas, tanto con AG como con RS. Para cada simulación se mantienen algunos parámetros fijos, con el fin de poder comparar los resultados y los tiempos de ejecución.

Los valores que se mantienen para todas las simulaciones son:

No. de réplicas (r) = 3
No. de tareas (*n*) = 14

No. de procesadores (m) = 7
Ponderaciones de:
$a=1200;\ b=200$
Probabilidad de mutación (*probmut*)=0.35
Máximo de generaciones (*maxgen*)=40
No. de padres (np)=4
No. de entrecruzamientos (nc)=6

Primera Simulación

Vector de utilización:

u = {0.849396, 0.822232, 0.632769, 0.822278, 0.391899, 0.630836, 0.902279, 0.348117, 0.540706, 0.365684, 0.958876, 0.692888, 0.319022, 0.0528932}

Matriz de asignación por RS:

$$V = \begin{bmatrix} 0&1&0&1&0&0&0&0&1&0&0&1&1&1 \\ 1&0&1&1&1&0&0&0&1&0&0&0&1&0 \\ 1&0&0&0&0&1&0&1&0&1&1&0&1&0 \\ 0&0&0&0&0&0&1&1&1&1&0&1&0&0 \\ 0&1&1&0&1&1&0&0&0&0&1&0&0&1 \\ 0&0&1&1&0&0&1&0&0&0&0&1&0&0 \\ 1&1&0&0&1&1&0&1&0&1&0&0&0&1 \end{bmatrix}$$

Evaluando la función objetivo (Ec. 4) con la matriz obtenida tenemos:

$$E^* = 17768.514870$$

Matriz de asignación por AG

$$V = \begin{bmatrix} 0&1&0&1&1&1&1&0&1&0&1&1&0&0 \\ 0&1&0&0&0&0&0&0&1&1&0&1&0&0 \\ 1&0&0&0&1&0&1&1&1&1&0&1&1&1 \\ 0&1&1&1&0&1&1&0&1&1&0&0&1&0 \\ 0&0&0&1&1&1&0&1&1&1&0&1&1&0 \\ 0&0&1&0&1&1&1&1&0&0&1&1&0&0 \\ 0&1&1&0&1&1&1&0&1&1&1&1&1&0 \end{bmatrix}$$

Posteriormente a dicha matriz se le aplicó el algoritmo de búsqueda por vecinos, dando como resultado la matriz:

$$V = \begin{bmatrix} 1&1&0&1&1&1&0&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0&1&1&0&1&0&1 \\ 1&0&0&0&0&0&1&1&1&0&0&1&0&1 \\ 0&0&1&1&0&0&1&0&1&0&0&0&1&1 \\ 0&0&0&1&1&1&0&1&0&1&0&1&1&0 \\ 0&0&1&0&0&1&1&1&0&0&1&0&0&0 \\ 0&1&1&0&1&0&0&0&0&1&1&0&1&0 \end{bmatrix}$$

Evaluando la función objetivo (Ec. 4) con la matriz obtenida tenemos:

$$E^* = 17713.651138$$

Segunda Simulación

Vector de utilización:

$u2$ = {0.318318, 0.817773, 0.904486, 0.763688, 0.439411, 0.673254, 0.139050, 0.653115, 0.0294785, 0.653488, 0.141683, 0.378823, 0.489752, 0.443381}

Los demás parámetros son iguales al ejemplo anterior.

Matriz de asginación por RS:

$$V = \begin{bmatrix} 1&0&1&0&0&0&1&0&0&1&0&1&1&1 \\ 0&1&0&1&1&1&0&0&0&0&0&0&0&1 \\ 1&1&0&1&0&0&1&0&1&1&0&1&0&0 \\ 0&1&0&0&1&1&0&0&1&0&1&0&1&0 \\ 0&0&1&0&0&0&0&1&1&1&0&0&1&0 \\ 0&0&1&0&0&1&1&1&0&0&1&1&0&0 \\ 1&0&0&1&1&0&0&1&0&0&1&0&0&1 \end{bmatrix}$$

Evaluando la función objetivo (Ec. 4) con la matriz obtenida tenemos:

$$E^* = 12133.658048$$

Se obtiene una matriz de asignación por AG

$$V = \begin{bmatrix} 0&1&0&0&0&0&1&0&1&1&1&1&1&1 \\ 1&0&0&0&1&1&0&0&1&1&0&1&1&0 \\ 1&1&1&0&1&0&0&1&0&0&1&0&0&1 \\ 1&1&0&1&1&1&1&0&0&0&0&0&1&0 \\ 0&1&1&1&1&1&1&1&0&1&1&1&1&1 \\ 1&0&0&0&0&0&0&1&0&1&1&0&0&1 \\ 1&1&0&1&1&1&0&0&0&0&1&1&0&0 \end{bmatrix}$$

Aplicando el algoritmo de búsqueda por vecinos:

$$V = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Evaluando la función objetivo (expresión 4) con la matriz obtenida tenemos:

$$E^* = 12067.763320$$

En estos ejemplos, el tiempo necesario para obtener el resultado por RS es de 2 horas, con 9216 iteraciones, en el caso de AG solo tomo 7 minutos obtener el resultado.

Se puede observar en la evaluación de la función objetivo, los valores obtenidos por AG es menor que los obtenidos por RS, por lo que podemos decir que el algoritmo AG tiene mejor rendimiento que el RS, en este tipo de problemas en particular.

CONCLUSIONES

Una de las cosas que se observo es la rapidez de los AG's para solucionar un problema de reasignación de tareas, esto se puede ver comparando los tiempos que tomaron los algoritmos aquí es presentados para llegar a una solución.

Cabe mencionar que el tiempo de respuesta tanto de los AG's y del RS dependen directamente de la cantidad de tareas y procesadores que se introducen a la simulación. Es evidente que entre mayor número de procesadores y de tareas se introduzcan a la simulación, mayor será el tiempo necesario para obtener la solución al problema.

REFERENCIAS

1. Chu, W.W., Holloway, L.J., Lam, M.-T. Efe, K.: Task Allocation in Distributed Data Processing. *IEEE Comput.* 13, 67-69 (1980).

2. Chou, T.C.K., Abraham, J.A.: Load Balancing in Distributed Systems. *IEEE Trans Software Engrg.* SE-8, 401-412 (1982).

3. Bennister, J.A. & Trivedi, K.S. *Task allocation in fault-tolerant distributed systems. In Hard Real-Time Systems* (Tutorial), J.A. Stankovic and K. Ramamritham, Eds. IEEE Computer Society Press, 1988, pp. 256-272.

4. Goldberg, G.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

5. E. Gómez-Ramírez, A. Sanchez De Tagle Hort & M. Alencastre Miranda; Forecasting time series using a polynomial artificial neural netwrok; *Workshop Artitifial Neural Networks: Current Trends and Applications*.Mexico city, March 16-20, 1998.