Análisis Comparativo de Algoritmos Genéticos Aplicados a Calendarización de Trabajos en un Grid Computacional

Victor Hugo Yaurima Basaldúa[1], Andrei Tchernykh[2], Moisés Torres Martínez[3]

[1] Universidad Estatal de Sonora, San Luis R.C., México, [2] Centro de Investigación CICESE, Ensenada, México, [3] Universidad de Guadalajara, Guadalajara, México,
victor.yaurima@ues.mx, chernykh@cicese.mx,
moises.torres@redudg.udg.mx

Resumen. Este artículo aborda la calendarización de trabajos paralelos en un Grid jerárquico con dos etapas. En esta configuración uno de los grandes retos es asignar las tareas de manera que permita un uso eficiente de los recursos, al mismo tiempo que satisface otros criterios. En general, los criterios de optimización a menudo están en conflicto. Para resolver este problema, proponemos un algoritmo genético bi-objetivo y presentamos un estudio experimental de seis operadores de cruzamientos, y tres operadores de mutación. Se determinan los parámetros más influyentes a través de un análisis estadístico de varianza multifactorial y comparamos nuestra propuesta con cinco estrategias de asignación conocidas en la literatura.

Palabras clave: Algoritmos genéticos, grid, calendarización.

1 Introducción

En este artículo presentamos un trabajo experimental de calendarización en un Grid computacional jerárquico de dos etapas [1, 2]. Uno de los grandes retos es obtener una calendarización que permita un uso eficiente de los recursos, al mismo tiempo que satisface otros criterios. Los criterios de optimización a menudo están en conflicto. Por ejemplo, los proveedores de los recursos y los usuarios tienen diferentes objetivos: los proveedores buscan una alta utilización de sus recursos, mientras que los usuarios están interesados en una rápida respuesta. En este trabajo se consideran ambos objetivos utilizando el método de agregación [3]. Para la primera etapa se desarrolla un algoritmo genético bi-objetivo para seleccionar recursos computacionales. Se examina el desempeño de un Grid computacional basado en datos reales de cinco días de actividad y se presenta un análisis comparativo de seis operadores de cruzamiento y tres operadores de mutación. Para calibrar el algoritmo genético se aplica un análisis de varianza multifactorial. El algoritmo genético calibrado es comparado con cinco estrategias conocidas.

Después de abordar el modelo del problema en la Sección 2, describimos en detalle el algoritmo genético en la Sección 3. En la Sección 4 calibramos el algoritmo genéti-

Advances in Computer Science and Engineering
Research in Computing Science 63, 2013 pp. 89–99
Paper Recived 22-09-2013 and Acepted 21-10-2013

co y presentamos la evaluación comparativa en la Sección 5. Finalmente concluimos en la Sección 6.

2 Modelo

Nos enfocamos en un modelo de calendarización fuera de línea: n trabajos paralelos $J_1, J_2, \dots, J_n$ deben ser calendarizados en m máquinas paralelas (sitios) $N_1, N_2, \dots, N_m$. Sea m_i el número de procesadores idénticos de la máquina N_i. Se asume que las máquinas están enlistadas en orden descendente de acuerdo al número de procesadores, esto es $m_1 \leq m_2 \leq \cdots \leq m_m$. Cada trabajo J_i se describe por una tupla $(size_j, p_j, p'_j)$: donde $1 \leq size_j \leq m_m$ representa el número de procesadores requeridos, también llamado grado de paralelismo, tiempo de ejecución p_j y tiempo de ejecución estimado p'_j. Todos los trabajos están disponibles antes de iniciar el proceso de calendarización. El tiempo de procesamiento del trabajo es desconocido hasta que el trabajo ha completado su ejecución. El tiempo de ejecución estimado p'_j es proporcionado por el usuario. Una máquina debe ejecutar un trabajo dedicando exactamente $size_j$ procesadores por un periodo ininterrumpido de tiempo p_j. Como no se permiten ejecuciones multi-sitios, un trabajo J_j puede ejecutarse solo en una máquina N_i si $size_j \leq m_j$.

Dos criterios son considerados: La terminación del calendario (*makespan*): $C_{max} = max(C_i)$, $i = 1,2,3, \dots, N_m$, donde C_i es el tiempo máximo de terminación de un trabajo en la máquina N_i, y el promedio de finalización (*mean turnaround time*): $TA = \frac{1}{n}\sum_{j=1}^{n} c_j$, donde c_j es el tiempo de terminación del trabajo J_j.

De acuerdo a la notación de tres campos (α|β|γ) [4], nuestro problema de calendarización es caracterizado como $GP_m|size_j, p_j, p'_j|OWA$, donde GP_m es el modelo de Grid de *m* maquinas con procesadores idénticos, OWA es un criterio de optimización que concatena dos criterios por el método de Agregación ($OWA = w_1 C_{max} + w_2 TA$), y w_i es el peso asignado a cada criterio. Para el problema en la segunda etapa usamos la estrategia Easy Backfilling [5].

2.1 Trabajos relacionados

Los algoritmos de calendarización para modelos de Grid en dos etapas pueden ser divididos en calendarización global y calendarización local [6]. En la primera etapa seleccionamos una máquina para cada trabajo usando un algoritmo genético. En la segunda etapa, usamos una estrategia de ejecución local para los trabajos recibidos.

La administración de recursos de un Grid es influenciada por múltiples objetivos y pueden requerir apoyo en la toma de decisiones con múltiples criterios. En [7], se considera la calendarización de tareas en recursos tomando en cuenta dos criterios: tiempo máximo de completar una tarea y tiempo promedio de completar las tareas. En [8], se presenta una calendarización de tareas basada en negociación de recursos, reservación de antemano y preferencias de usuario. Para ayudar a escoger la mejor estrategia en [9], se desarrolla un análisis de diferentes métricas de acuerdo a la meto-

dología de sus degradaciones. El objetivo es encontrar una estrategia que se comporte con buenos resultados en todos los casos de estudio, considerando cargas de trabajo y configuraciones de Grid diferentes.

La metodología para una decisión multi-criterio puede basarse en la optimalidad de Pareto, sin embargo es muy difícil alcanzar soluciones en un tiempo razonable usando esta opción. En [3], usan el método de agregación para modelar las preferencias de los actores involucrados. En [10] se presentan estrategias de asignación a diferentes centros de cómputo y proponen un modelo de calendarización multi-criterio. Ellos concluyen que tal calendarización puede ser desarrollada eficientemente usando algoritmos genéticos. En [11], se aplican algoritmos genéticos con dos objetivos usando el método de agregación, comparando cinco operadores de cruzamiento.

3 Algoritmo genético

Los algoritmos genéticos son una técnica conocida, usada para encontrar soluciones a problemas de optimización combinatoria. Las soluciones candidatas son codificadas como cromosomas, también llamadas genomas o individuos. En nuestro caso, cada individuo o solución es codificada en una matriz $n \cdot m$. El número -1 en una celda significa que no existe un trabajo en esa posición de la cola local.

La fila $i = 0, \dots, m-1$ representa la cola local en la máquina N_i. El conjunto de máquinas disponibles para el trabajo J_j son las máquinas con índices $\{f_j \dots m\}$, donde f_j es el índice i más pequeño tal que $m_i \geq size_j$.

Una población está formada por un conjunto de individuos que estarán modificándose para mejorar su aptitud. Un valor de aptitud es la medida de la calidad de la solución de acuerdo al criterio de optimización usado. En este caso usamos dos criterios (Sección 2). Para que los dos criterios se evalúen simultáneamente, usamos el método de agregación: promedio ponderado ordenado (*Ordered Weighted Averaging - OWA*) [3]. Se contemplan valores o pesos que representan la relativa importancia de cada criterio:

$$OWA(x_1, x_2, \dots, x_n) = \sum_{c=1}^{k} w_c s(x)_{\sigma(c)} \tag{1}$$

Donde w_c es el peso, $c = 1, \dots, k$, x_c es un valor asociado con la satisfacción del criterio c. Se realiza la permutación de valores: $s(x)_{\sigma(1)} \leq s(x)_{\sigma(2)} \leq \dots s(x)_{\sigma(k)}$. Los pesos (w_c) son positivos y $\sum_{c=1}^{k} w_c = 1$. El objetivo es encontrar un esquema de pesos que provea el mejor valor medio de acuerdo a la conveniencia de los interesados y el más alto valor posible para el peor caso. Para alcanzar esto el peso w_1 debe ser relativamente grande, mientras que el peso w_k debe ser pequeño, k denota el número de criterios. Los pesos restantes son decrementados en valor desde w_1 hasta w_k de acuerdo a:

$$w_c = \begin{cases} 3/2k, & c = 1 \\ (3k - 2c - 1)/2n(k-1), & 1 < c \leq k \end{cases} \tag{2}$$

El valor mínimo de OWA corresponde a la mejor aptitud.

Se aplican tres operadores genéticos: selección, cruzamiento y mutación. La selección identifica los individuos que van a cruzarse para producir la siguiente generación. Utilizamos la selección por torneo binario, donde dos individuos son elegidos aleatoriamente de la población, el que tenga mejor aptitud, gana. Este proceso se repite dos veces con el fin de seleccionar dos padres. Los individuos se desarrollan hasta que el criterio de paro se cumple. En nuestro caso, el algoritmo se detiene si la aptitud del mejor individuo encontrado no mejora en 10 generaciones.

3.1 Operadores de cruzamiento

El operador de cruzamiento se aplica bajo cierta probabilidad (P_c). En este trabajo son considerados seis operadores que a continuación se describen.

One Segment Crossover for Matrix (OSXM). Está basado en el operador de cruzamiento OSX - *One Segment Crossover* [14]. En este operador, se seleccionan aleatoriamente dos puntos en la matriz, S1 y S2 desde 0 hasta el máximo índice usado. El hijo hereda las columnas del padre 1 desde la posición 0 hasta S1. Hereda las columnas del padre 2 desde S1 hasta S2, considerando solamente aquellos elementos que no han sido copiados del padre 1. Finalmente el hijo hereda el resto de los elementos del padre 1.

Two Point Crossover for Matrix (TPM). Está basado en el operador de cruzamiento *Two Point Crossover* [15]. En este operador, dos posiciones de la matriz son elegidas aleatoriamente *S1* y *S2*. Las columnas desde la posición 0 hasta *S1* y desde *S2* hasta el final son copiados del padre 1. El resto de los elementos son copiados del padre 2.

Order Based Crossover for Matrix (OBXM). Está basado en el operador de cruzamiento *OBX - Order Based Crossover* [12]. Se usa una máscara binaria generada aleatoria e uniformemente de acuerdo al número de columnas en la matriz. Los valores de la máscara binaria iguales a uno indican que sus correspondientes columnas son copiadas al hijo del padre 1. El resto de los elementos son copiados del padre 2 (Fig. 1).

Precedence Preservative Crossover for Matrix (PPXM). Está basado en el operador de cruzamiento *PPX - Precedence Preservative Crossover* [13]. Se usa una máscara binaria generada aleatoria e uniformemente de acuerdo al número de columnas en la matriz. La columna cuyo valor corresponde a la máscara igual a uno, es copiada al hijo del padre 1. La siguiente columna si la máscara es igual a cero, es copiada del padre 2, de lo contrario es copiada del padre 1, así sucesivamente son copiados los elementos. Se toma en cuenta el orden de izquierda a derecha en cada iteración y que los elementos no hayan sido copiados al hijo de alguno de los padres.

Order Segment Crossover for Matrix with Setup (OSXMS). Está basado en el operador de cruzamiento *OSX-Order Segment Crossover* [11, 14]. Similar al OSXM, pero los elementos son ordenados ascendentemente de acuerdo al número de procesadores requeridos antes de ser copiados al hijo, tomando en cuenta que la capacidad de la máquina en cuanto a número de procesadores sea suficiente para ejecutar el trabajo.

Order Based Crossover for Matrix with Setup (OBXMS). Está basado en el operador de cruzamiento OBX - *Order Based Crossover* [12]. Se usa una máscara binaria generada aleatoria e uniformemente de acuerdo al número de columnas en la

matriz. Los valores de la máscara binaria iguales a uno indican que sus correspondientes columnas son copiadas al hijo del padre 1 tomando en cuenta un orden ascendente de los trabajos de acuerdo al número de procesadores requeridos. El resto de los elementos son copiados del padre 2.

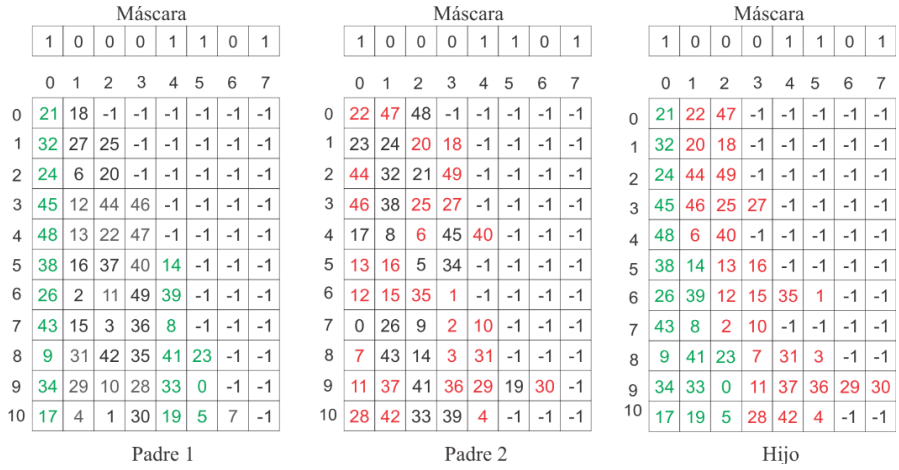


Fig. 1. Order Based Crossover for Matrix (*OBXM*)

3.2 Operadores de mutación

Los operadores de mutación producen pequeños cambios en los individuos de acuerdo a una probabilidad P_m. Este operador ayuda a prevenir caer en óptimos locales y a extender el espacio de búsqueda del algoritmo. Se consideran tres operadores de mutación adaptados para una matriz de dos dimensiones: (1) *Queue_Insert***.** Dos posiciones *S1* y *S2* son seleccionadas aleatoriamente. Los elementos de la columna *S2* son insertados en la columna *S1*, recorriendo el resto de los elementos. (2) *Queue_Swap* selecciona aleatoriamente dos columnas en la matriz e intercambia sus elementos; (3) *Queue_Switch*, selecciona una columna aleatoriamente e intercambia sus elementos con la columna adyacente.

4 Calibración del algoritmo genético

4.1 Carga de trabajo

Dos aspectos fundamentales deben abordarse para configurar un entorno de simulación para la evaluación del desempeño. Por un lado, se necesitan registros representativos de carga de trabajo para producir resultados confiables. Por otro lado, un buen entorno de prueba debe ser configurado para obtener resultados reproducibles y comparables. Se consideran cuatro registros de PWA (Parallel Workloads Archive) [16] correspondientes a: Cornell Theory Center, High Performance Computing Center North, Swedish Royal Institute of Technology y Los Alamos National Lab; y un

registro de GWA (Grid Workloads Archive) [17] correspondiente a: Advanced School for Computing and Imaging. Se tomaron los registros correspondientes a cinco días.

4.2 Calibración de parámetros

Se usa una adaptación del método de diseño de experimentos propuesto por [18], donde se consideran los siguientes pasos: (a) se ejecuta cada carga de trabajo con todas las posibles combinaciones de parámetros; (b) se obtiene la mejor solución; (c) se calcula la diferencia relativa de cada algoritmo sobre la mejor solución (d) Se aplica el Análisis de Varianza Multifactorial (ANOVA) para encontrar el parámetro que más influye en la solución y para seleccionar el conjunto de los mejores valores de cada parámetro que constituirá el algoritmo adecuado para el problema abordado. La Tabla 1 muestra los parámetros usados para la calibración.

Tabla 1. Parámetros de calibración

Parámetros	Niveles
Operadores de cruzamiento:	OSXM, TPM, OBXM, PPXM, OSXMS, OBXMS
Operadores de mutación:	Queue_Insert, Queue_Swap, Queue_Switch
Probabilidades:	Cruzamiento: 0.9, Mutación: 0.01
Población:	100 individuos.
Número de trabajos por individuo:	Día 1: 1375; Día 2: 646; Día 3: 564; Día 4: 1041; Día 5: 1083.
Operador de selección:	Torneo binario.
Tamaño de sitios:	4, 4, 4, 8, 8, 8, 16, 16, 32, 32
Criterio de paro:	Si la aptitud no mejora en 10 generaciones.

Se consideraron 18 diferentes algoritmos. Se realizaron 30 ejecuciones para la carga de trabajo en cada combinación, en total 18 x 30 = 540 experimentos. El desempeño de cada algoritmo es calculado como el porcentaje del incremento relativo sobre la mejor solución obtenida (*IRMS*), calculado con la siguiente fórmula: $IRMS = (Heu_{sol} - Best_{sol})/Best_{sol} \cdot 100$. Donde Heu_{sol} es el valor de la función objetivo obtenido por el algoritmo considerado y $Best_{sol}$ es el mejor valor obtenido durante la ejecución de todas las posibles combinaciones de los parámetros.

4.3 Análisis de varianza.

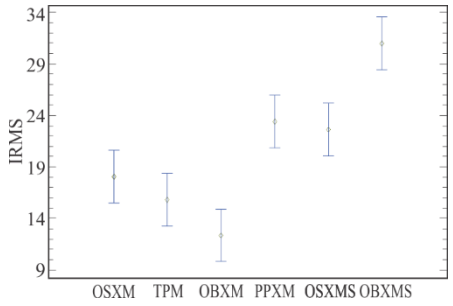
El análisis de varianza se aplica para evaluar la diferencia estadística entre los resultados experimentales y observar el efecto de los parámetros sobre la calidad de los resultados. Se usa para determinar los factores que tienen un efecto significativo y saber cuáles son los factores más importantes (Tabla 2).

Tabla 2. Análisis de varianza para IRMS. 5 días.

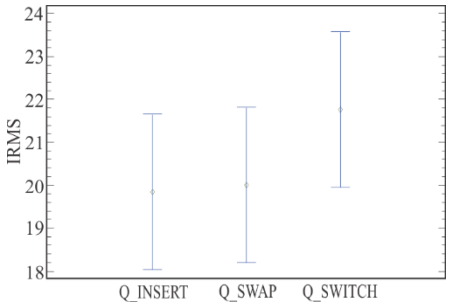
		(a) *Cmax*		(b) *TA*	
Días	**Factores**	**Razón-F**	**Valor-P**	**Razón-F**	**Valor-P**
Día 1	Cruzamiento	16.43	**0.0002**	11.76	**0.0006**
	Mutación	0.88	0.4452	2.23	0.1580
Día 2	Cruzamiento	32.25	**0.0000**	7.26	**0.0041**
	Mutación	1.64	0.2416	0.77	0.4874
Día 3	Cruzamiento	23.70	**0.0000**	11.00	**0.0008**
	Mutación	0.82	0.4681	0.99	0.4056
Día 4	Cruzamiento	26.49	**0.0000**	7.50	**0.0036**
	Mutación	0.35	0.7104	0.90	0.4372
Día 5	Cruzamiento	51.64	**0.0000**	13.34	**0.0004**
	Mutación	1.48	0.2742	0.61	0.5628

Los resultados de ANOVA descomponen la variabilidad de IRMS en contribuciones debidas a varios factores. Puesto que se ha escogido la suma de cuadrados Tipo III (por omisión), la contribución de cada factor se mide eliminando los efectos de los demás factores.

La Fig. 2 (a) muestra los resultados obtenidos en el primer día para el factor de operador de cruzamiento. El eje vertical es el valor del *IRMS*. Podemos ver que el operador de cruzamiento *OBXM* es mejor. La Fig. 2 (b) muestra los resultados para los operadores de mutación. El operador *Queue_Insert* resultó ser ligeramente mejor. Este comportamiento es similar en los cinco días analizados. Con el criterio *TA* los operadores de mutación tienen el mismo comportamiento y respecto al operador de cruzamiento *OSXM* es mejor.



(a) Operadores de cruzamiento. Cmax



(b) Operadores de mutación. Cmax

Fig. 2. Resultados Día 1

Hasta este punto, hemos calibrado el algoritmo genético, obteniendo como operador de cruzamiento: *OBXM* para el criterio *Cmax* y *OSXM* para el criterio *TA*. Siendo que en la siguiente sección de comparación usaremos el criterio *Cmax*, seleccionamos el operador *OBXM*. Como operador de mutación proponemos el mejor en ambos casos: *Queue_Insert*. Usamos 0.9 como probabilidad de cruzamiento de acuerdo a [11]. A este algoritmo calibrado, llamaremos *Calib_GA*.

5 Evaluación comparativa

Después de obtener la calibración del algoritmo genético, en esta sección procedemos a comparar *Calib_GA* con 5 estrategias que se presentaron en [9] (Tabla 3).

Tabla 3. Estrategias de asignación

Estrategia	Descripción
MIN LB	Asigna el trabajo j al sitio con los menores recursos consumidos por procesador. $min_{i=1..m}\left\{\sum_{k=1}^{l_i}\frac{size_k \cdot p'_k}{m_i}\right\}$
MIN CT	Asigna el trabajo j al sitio con el menor tiempo de finalización del Grid $min\{C^i_{max}\}$,
MIN WT	Asigna el trabajo j al sitio con el menor promedio de tiempo de espera de los trabajos. $min_{i=1..m}\left\{\sum_{g_k=i}\frac{t^k_w}{n_i}\right\}$
MIN ST	Asigna el trabajo j al sitio que pueda iniciar más temprano su ejecución. $min_{\ i=1..m}\{s^i_j\}$
MIN TA	Asigna el trabajo j al sitio con el menor tiempo de permanencia de los trabajos $min_{i=1..m}\left\{\sum_{g_k=i}\frac{c^k_w}{n_i}\right\}$

Del total de trabajos hemos tomado la cantidad correspondiente al registro de 5 días: 1375, 646, 564, 1041, 1083 trabajos por día. Hemos realizado 30 ejecuciones y tomado el promedio de éstas para efecto de la comparación.

El desempeño de los algoritmos es calculado usando el *IRMS*. Para todos los casos el algoritmo Calib_GA resultó mejor. El *IRMS* de este algoritmo está en el rango 2.7% - 12.7% en promedio. Mientras que los otros algoritmos tienen una distancia de 9.5% - 118.7% respecto a la mejor solución. La Tabla 4 muestra los promedios del *IRMS* respecto a los 30 experimentos para cada día. El algoritmo Calib_GA se desempeñó aproximadamente cuatro veces mejor con un promedio de 9.3 comparado con las otras estrategias que resultaron en el rango de 30.3 a 69.8.

Tabla 4. Promedio del *IRMS*

Estrategia	Día 1	Día 2	Día 3	Día 4	Día 5	Prom.
MIN LB	49.8	25.6	85.4	22.8	33.7	43.4
MIN CT	54.8	30.0	24.1	33.3	9.5	30.3
MIN WT	52.5	24.2	54.6	35.3	36.9	40.7
MIN ST	77.7	25.7	52.4	28.2	11.2	39.0
MIN TA	93.5	44.9	62.1	118.7	29.9	69.8
Calib_GA	11.9	12.7	4.8	14.3	2.7	9.3

La Tabla 5 muestra los tiempos de ejecución. Las estrategias clásicas tuvieron un tiempo de ejecución de menos de 3 segundos, mientras que el algoritmo Calib_GA consumió un tiempo mayor, con un promedio de 644 segundos (10.7 minutos).

Tabla 5. Tiempo de ejecución en segundos.

Estrategia	Día 1	Día 2	Día 3	Día 4	Día 5
MIN LB	1	< 1	< 1	< 1	< 1
MIN CT	< 1	1	1	1	1
MIN WT	< 1	1	2	2	2
MIN ST	< 1	1	1	1	2
MIN TA	< 1	1	1	3	1
Calib_GA	942	276	218	495	1290

6 Conclusiones

Se realizó un estudio comparativo de un algoritmo genético y estrategias conocidas, enfocados a optimizar la ejecución de trabajos computacionales en un grid jerárquico con dos etapas, considerando una carga de centros de cómputo reales. Primeramente se calibró el algoritmo genético. La calibración fue realizada comparando seis operadores de cruzamiento y tres operadores de mutación. Así, se analizaron 18 diferentes algoritmos genéticos con 30 ejecuciones para cada uno. En total, se evaluaron 540 experimentos. Usamos el método de agregación de criterios para considerar dos objetivos: C_{max} y *TA*. Para evaluar la diferencia estadística entre los resultados experimentales y observar el impacto de diferentes parámetros sobre la calidad del resultado, se aplicó la técnica de ANOVA. De los seis operadores de cruzamiento comparados, se seleccionó el operador *OBXM*. Después, se comparó con cinco estrategias conocidas en la literatura. Los resultados muestran que nuestro algoritmo es 480% mejor, sin embargo, consumió más de 10 minutos en promedio en comparación con menos de 3 segundos de las otras estrategias. Concluimos que para los casos donde la calendarización en ambientes dinámicos es administrada por lotes, la calendarización realizada con algoritmos genéticos puede ser muy eficiente. Los resultados obtenidos pueden servir como punto de partida para algoritmos de calendarización que pueden ser implementados en un ambiente de Grid computacional real donde es posible agrupar los trabajos en lotes.

Referencias

1. Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., Röblitz, T., Ramírez-Alcaraz, J.M., González-García, J.L.: Multiple Parallel Workflow Scheduling Strategies on a Grid. Journal of Grid Computing, 10(2), 325-346 (2012).
2. Quezada-Pina, A., Tchernykh, A., González-García, J.L., Hirales-Carbajal, A., Ramírez-Alcaraz, J.M., Schwiegelshohn, U., Yahyapour, R., Miranda-López, V.: Adaptive parallel job scheduling with resource admissible allocation on two-level hierarchical grids. Future Generation Computer Systems. 28(7), 965-976 (2012).
3. Kurowski, K., Nabrzyski, J., Oleksiak, A. and Węglarz, J.: A multicriteria approach to two-level hierarchy scheduling in Grids. Journal of Scheduling, 11(5), 371-379 (2008).

4. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Hammer, P.L., Johnson, E.L., Korte, B.H. (eds.) Annals of Discrete Mathematics 5. Discrete Optimization II, North-Holland, 287–326 (1979).
5. Lifka, D. A.: The ANL/IBM SP Scheduling System. In IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, London, UK: Springer-Verlag, 295-303 (1995).
6. Tchernykh A., Schwiegelsohn U., Yahyapour R., Kuzjurin N.: Online Hierarchical Job Scheduling on Grids with Admissible Allocation. Journal of Scheduling, Springer-Verlag, 13(5), 545-552 (2010).
7. Dutot, P., Eyraud, L., Mounie, G., Trystram, D.: Models for scheduling on large scale platforms: which policy for which application? 18th International Symposium on Parallel and Distributed Processing, pp.172 (2004).
8. Siddiqui, M., Villazon, A., Fahringer, T.: Grid Capacity Planning with Negotiation-based Advance Reservation for Optimized QoS. In: Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC '06). ACM (2006).
9. Ramírez-Alcaraz J.M, Tchernykh A., Yahyapour R., Schwiegelshohn U, Quezada-Pina A., González-García, J.-L., Hirales-Carbajal A.: Job Allocation Strategies with User Run Time Estimates for Online Scheduling in Hierarchical Grids. J. Grid Computing, Springer, 9, 95-116 (2011).
10. Lorpunmanee, S., Noor, M., Hanan, A., Srinoy, S.: Genetic algorithm in Grid scheduling with multiple objectives. Proceedings of the 5th WSEAS int. Conf. on Artificial Intelligence. Knowledge Engineering and Data Bases. Madrid, Spain, 429-435 (2006).
11. Yaurima-Basaldúa, V. H, Tchernykh, A., Castro-García, Y., Villagómez-Ramos, V.M., Burtseva, L.: Genetic algorithm calibration for two objective scheduling parallel jobs on hierarchical Grids. In Parallel Processing and Applied Mathematics, Wyrzykowski et al. (Eds.). LNCS, Springer-Verlag, 7204, 61-70 (2012).
12. Gen, M., Cheng, R.: Genetic algorithms & engineering optimization. John Wiley & Sons, New York, 512 pp. (1997).
13. Bierwirth, C., Mattfeld, D. and Kopfer, H.: On permutation representations for scheduling problems. En: Voigt, H.-M., W. Ebeling, I. Rechenberg, y H.-P. Schwefel, editores, Parallel Problem Solving from Nature - PPSN IV, Berlin, Germany, LNCS. Springer, 1141, 310-318 (1996).
14. Guinet, A., Solomon, M.: Scheduling Hybrid Flowshops to Minimize Maximum Tardiness or Maximum Completion Time, Int. J. Production Research, 34(6), 1643-1654 (1996).
15. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolutions Programs. 3rd edn. Springer-Verlag, Berlin Heidelberg New York (1996).
16. PWA. Parallel Workloads Archive, http://www.cs.huji.ac.il/labs/parallel/workload/
17. GWA. Grid Workloads Archive, http://gwa.ewi.tudelft.nl/pmwiki/
18. Ruiz, R., Maroto, C.: A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research, 169, 781-800 (2006).