UN SISTEMA DISTRIBUIDO PARA EL PROCESAMIENTO PARALELO DE ALGORITMOS GENÉTICOS

ARREDONDO D., PRINTISTA M.[1]
GALLARD R.[2]

Grupo de Interés en Sistemas de Computación[3]
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106
5700 - San Luis
Argentina
E-mail:rgallard@inter2.unsl.edu.ar
gallardr@unslfm.edu.ar
Phone: 54++ 652 20823
Fax : 54++ 652 30224

RESUMEN

Un sistema distribuido, básicamente, permite que varias tareas se desarrollen en forma conjunta siguiendo un enfoque competitivo o cooperativo. El fundamento de esta última perspectiva está en alcanzar mayor rapidez en la resolución de un único problema. Este tipo de enfoque es el adecuado para aquellas aplicaciones que son inherentemente paralelas. Los algoritmos genéticos son, esencialmente, algoritmos de búsqueda intrínsecamente paralelos.

El presente trabajo propone una arquitectura distribuida que permite el procesamiento paralelo de algoritmos genéticos para la posterior evaluación de su performance.

Palabras Clave: Sistemas Distribuidos, Procesamiento Paralelo, Algoritmos Genéticos.

---

[1] Miembros del proyecto 338403 de la UNSL.

[2] Profesor Titular Exclusivo del Departamento de Informática. Director del Grupo de Investigación.

[3] El Grupo de Investigación es subvencionado por la Universidad Nacional de San Luis, el CONICET (Consejo Nacional de Investigaciones Científicas y Técnicas) y la SSID (Subsecretaría de Informática y Desarrollo).

1. INTRODUCCIÓN

Un sistema distribuido consiste de una colección de computadoras autónomas interconectadas por una red y equipadas con un software diseñado para ofrecer una facilidad de computación integrada. (Colouris [1], Tanenbaum [9]).
Un caso específico de sistema distribuido, conocido como *computadora MIMD*[1] *de memoria distribuida* consiste de múltiples nodos, los que a su vez poseen (al menos) un procesador y memoria (locales). El "program counter" del procesador recupera instrucciones desde la memoria local las que son ejecutadas sobre datos que también residen en esta memoria. Los nodos están interconectados por medio de algún tipo de conexión física conformando una red que hace posible la comunicación entre ellos. En esta arquitectura, el paralelismo se obtiene permitiendo que las computaciones de cada procesador progresen sobre los datos de su memoria local en simultaneidad con las ejecutadas en los demás procesadores. La comunicación y sincronización, eventuales, se manejan exclusivamente vía algún mecanismo (IPC) de comunicación entre procesos, utilizado a través de la red de interconexión.

Este trabajo propone una arquitectura distribuida apropiada para explotar el paralelismo subyacente en los algoritmos genéticos. Se describe además, una implementación paralela[2] de algoritmos genéticos donde cada componente de software se replica localmente, en cada nodo, y comienza el procesamiento de una subpoblación de la población inicial de individuos. Esporádicamente los mejores individuos migrarán de una subpoblación a otra, donde serán (o no) incorporados. Finalmente, la reagrupación de las subpoblaciones provee la población final que ha evolucionado a través de la ejecución paralela del algoritmo.

2. ALGORITMOS GENÉTICOS

Los algoritmos genéticos (GAs) desarrollados por Holland [6] son algoritmos de búsqueda basados en una analogía con la selección natural y la genética poblacional. Una de las aplicaciones mas destacadas de los GAs es la exploración de soluciones en problemas difíciles de optimización.
Opuestamente a otros métodos los GAs trabajan sobre una población de soluciones potenciales en lugar de operar sobre sólo una, lo cual permite una búsqueda altamente paralela (Michalewicz [8]).
Cada punto en el espacio de soluciones potenciales, se representa por un "string" de símbolos, llamado *individuo* o *cromosoma*, y la interpretación del significado de dicho "string" es específica del problema.

Los GAs trabajan asignando un valor a cada "string" de la población en concordancia con una función de adaptabilidad (*fitness function*), también específica del problema. Un proceso de selección, basado en la supervivencia de los más adaptables, escoge "strings" de la población actual acorde con su valor de adaptabilidad. Estos "strings" se recombinan utilizando operadores de cruzamiento (*crossover*) y mutación (*mutation*) produciendo un nueva generación de "strings" con

[1] Multiple Instruction Multiple Data
[2] Variante del modelo conocido como *island model*

posiblemente mayor adaptabilidad que los anteriores. A continuación se muestra un bosquejo de algoritmo genético.

```
ini_prog

t ← 0
inicializar    población( t )
evaluar        población( t )

repita         para cada generación
        seleccionar  población ( t +1) desde población ( t )
        recombinar   población ( t +1)
        evaluar      población ( t +1)
        t ← t+1
fin_repita

fin_prog
```

3. PARALELISMO Y ALGORITMOS GENÉTICOS

Como los GAs son algoritmos altamente paralelos, Holland [5], reconoció la eficiencia inherente del procesamiento paralelo en uno de sus primeros trabajos. Más recientemente Grefenstette [4] (1989), y Levine [7] (1994) reportaron importantes experiencias.
El punto central aquí es encontrar cómo, algoritmos inherentemente paralelos (GAs) pueden ser procesados en una arquitectura paralela para obtener ventaja de sus características intrínsecas. Las implementaciones paralelas típicas fueron puntualizadas por Grefenstette en su primer trabajo [3] como prototipos: master-slave sincrónicos y semisincrónicos, concurrentes-asincrónicos y procesamiento de red. Esta clasificación está principalmente basada en las arquitecturas paralelas disponibles ("Tightly-coupled" o "Loosely-coupled"). Ver Goldberg [2] para más detalles.

Los GAs pueden , globalmente, dividirse en secuenciales (SGAs) o paralelos (PGAs). Los primeros, también conocidos como convencionales, se ejecutan secuencialmente sobre una población evolutiva de cromosomas. Con respecto a los segundos, Levine observa la necesidad de distinguir entre un PGA como *modelo particular* de un GA, y un PGA como un *medio para implementar* un GA (secuencial o paralelo). En un *modelo* PGA la población total se distribuye en alguna de dos formas:

- Un conjunto de subpoblaciones autónomas de cierto tamaño (el caso extremo es el de tamaño 1).

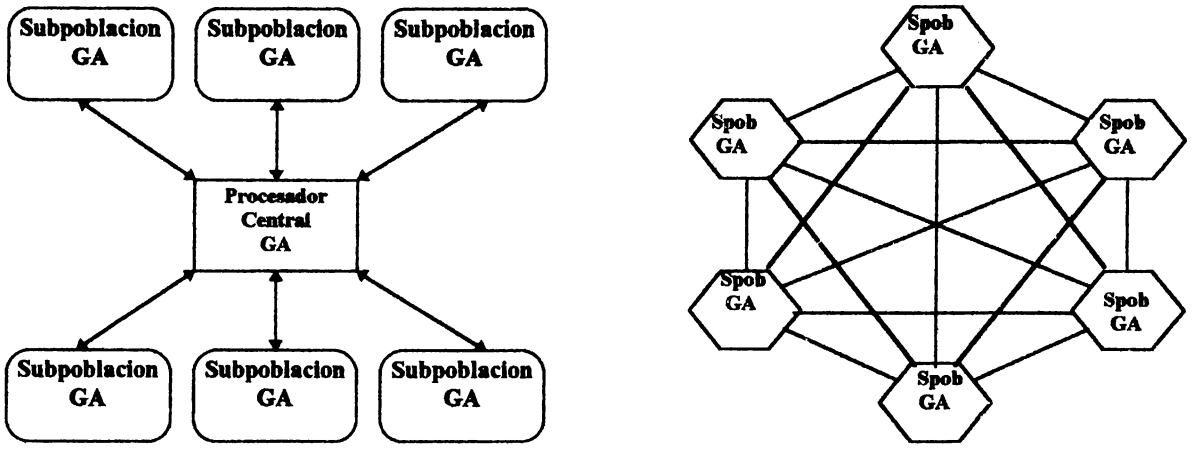
- Una sola población donde cada miembro de la misma interactúa sólo con un limitado subconjunto de miembros.

Bajo ambos enfoques el concepto subyacente es el de la existencia de diversas subpoblaciones que intercambian material genético. Esto es más cercano

al escenario donde la selección natural opera; poblaciones que consisten de varias subpoblaciones que eventualmente interactúan.

Las máquinas paralelas son las plataformas apropiadas para implementar el paralelismo de los GAs. En el caso de una *implementación paralela* de un GA convencional, el problema consiste en encontrar una forma de paralelizar alguna componente del GA . Lo más simple es paralelizar el "loop" que crea una nueva población a partir de la actual. Muchos pasos de ese "loop", tal como la evaluación y la aplicación de operadores, pueden ejecutarse en paralelo. Este enfoque se ha implementado, a menudo, en computadoras de memoria distribuida. Sin embargo, el "overhead" de computación paralela, introducido por la frecuente distribución de estructuras de datos, la sincronización y la recolección de resultados parciales, menguan los beneficios esperados de la utilización de varios procesadores. Por ello para explotar el paralelismo a nivel de "loop" es recomendable hacerlo sobre máquinas de memoria compartida.

Por otro lado, los GA paralelos pueden dividirse en PGAs de gránulo[1] fino y de gránulo grueso. En un PGA de gránulo grueso, diversos procesadores corren, independiente y simultáneamente, un GA convencional sobre su propia subpoblación (modelo isla). Ocasionalmente los mejores individuos migran desde una subpoblación a otra. El destino de los individuos podría escogerse en forma random o bajo un cierto criterio elaborado. En un PGA de gránulo fino, la población global se subdivide de manera tal que un único individuo es asignado a un procesador. La selección, recombinación y reemplazo de cromosomas son ejecutadas por varios procesadores que interactúan (típicamente entre los vecinos).



a) Control Centralizado b) Control Distribuido

Fig 1. Implementaciones paralelas de un modelo de PGA

La figura 1 muestra implementaciones paralelas de un modelo PGA. Como un ejemplo, en este caso, la población total es dividida en un número de subpoblaciones asignadas a distintos procesadores, los cuales corren un GA sobre sus propias subpoblaciones. En 1.a existe una fase donde los mejores individuos son obtenidos por el procesador central, realizando luego una recombinación de

[1] Hacemos referencia a la granularidad del paralelismo. El gránulo de paralelismo mide la cantidad de computaciones paralelas que pueden realizarse en los procesadores involucrados antes de requerirse una comunicación entre los procesos, y en consecuencia determina el tipo de arquitectura del hardware que mejor se adecue al problema.

estos individuos. Los mejores individuos son insertados en las subpoblaciones. Este enfoque provee heterogeneidad entre las subpoblaciones (evitando caer en un mínimo local), pero depende de la confiabilidad del procesador central. En 1.b se muestra un enfoque descentralizado. Aquí pueden seguirse diversos criterios para distribuir el GA. (ej. cada procesador envía copias de sus mejores individuos a sus vecinos, los cuales reemplazan sus peores individuos por éstos, o periódicamente cada procesador asume el rol de controlador global).
Los modelos de PGA pueden ser implementados sobre una arquitectura "Tightly-coupled" o "Loosely-coupled", dependiendo de la frecuencia de comunicación entre procesadores (para "broadcasting" o migración de individuos).

4. EL SOPORTE PARA LA MIGRACIÓN

Ahora describiremos brevemente la arquitectura del sistema y el conjunto de rutinas que proveen el servicio que soporta el intercambio (*migración*) de cromosomas entre procesadores distribuidos en una red de área local, que ejecutan GAs sobre sus propias subpoblaciones. Estas rutinas son independientes del tipo particular de algoritmo genético paralelo que se ejecuta en los procesadores.

4.1 LA ARQUITECTURA DEL SISTEMA

La fig. 2 describe la arquitectura del sistema.

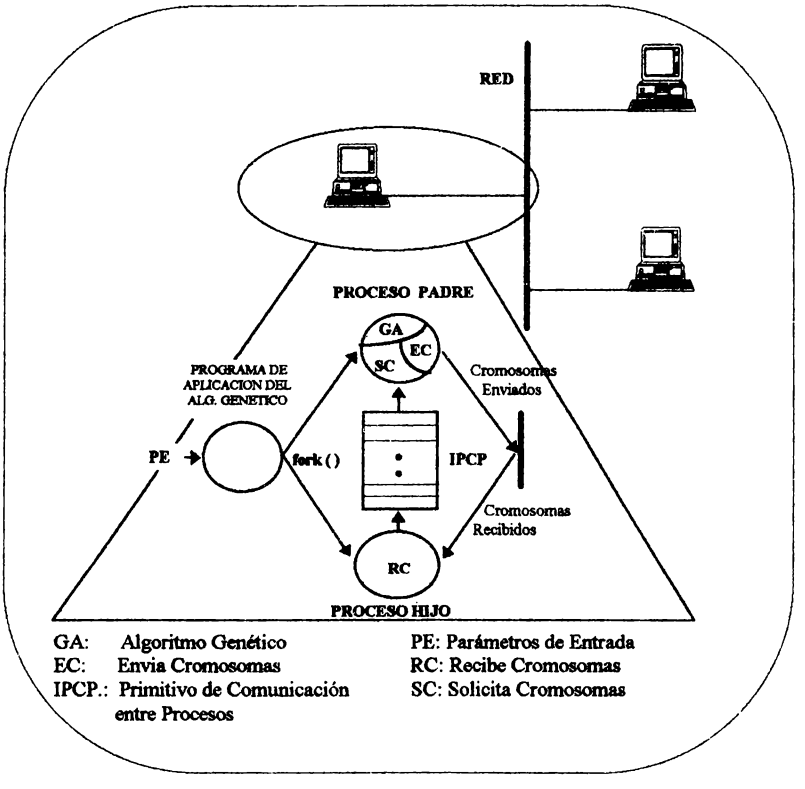


Fig.2 ARQUITECTURA DEL SISTEMA PARA SERVICIO DE MIGRACIÓN.

El proceso padre será el encargado de ejecutar el algoritmo genético y solicitar los servicios de migraciones invocando las rutinas definidas a tal fin. El proceso hijo por su parte, será el encargado de recibir desde la red los cromosomas que arriban desde otros procesadores (subpoblaciones) y de transferirlos al proceso padre local a través de un *primitivo de comunicación entre procesos* (IPCP: pipe, fifo ,etc.).

4.2. ALGUNAS CONSIDERACIONES DE IMPLEMENTACIÓN

Un punto importante a considerar es el hardware del sistema. Básicamente se pueden encontrar dos tipos de sistemas de computadoras paralelas. "Tightly-coupled systems", donde demoras despreciables son experimentadas en la comunicación entre procesadores, y la velocidad de transmisión de datos es alta. Estos sistemas son aptos para trabajar en un único problema. Tienden a ser usados en diseños que involucran muy frecuente comunicación entre procesos paralelos. Por otro lado los "Loosely-coupled systems", son aptos para ejecutar procesos sin relación que comparten (o compiten) por recursos, o bien si cooperan para la solución de un problema, lo hacen a través de comunicaciones circunstanciales.

Otro punto relevante es el software del sistema. Un "Tightly-coupled software" está dedicado a correr un único programa en paralelo explotando el paralelismo inherente a su estructura. Aquí el objetivo central es incrementar la velocidad de ejecución. Un "Loosely-coupled software", permite que máquinas y usuarios de sistemas distribuidos sean independientes unos de otros con un grado limitado de comunicación si fuese necesario. Aquí el objetivo esencial es permitir la compartición de recursos.
En ambos casos, dependiendo del diseño, se puede incrementar también el nivel de confiabilidad del sistema.

Este trabajo fue desarrollado sobre equipos SUN, conectados en una red de área local, bajo una topología de hardware "Loosely-coupled". El proyecto global de esta aplicación, que involucra el procesamiento paralelo de GAs y su posterior análisis de resultados, requiere desde el punto de vista del software, un tratamiento muy cercano al "Tightly-coupled". Esta decisión se basa en que la aplicación requería un software que permitiera:

1. Que al comienzo de la ejecución del programa, el usuario ingresara el tamaño de la población, la condición de finalización para cada proceso paralelo, y otros parámetros propios de los GAs. El sistema debía entonces generar la población total de individuos, producir la división en subpoblaciones y finalmente distribuir esta información junto con los parámetros para la migración, a los distintos procesos replicados en los procesadores destinados a tal fin.
2. Que cada procesador ejecutase el mismo GA sobre su propia subpoblación. Además, esporádica e independientemente, cada proceso corriendo sobre una subpoblación migraría cromosomas a otros procesos ejecutando en otros procesadores. De la misma forma, los procesos requerirían cromosomas emitidos desde otros procesadores y los integrarían en su subpoblación. De esta forma se establece una actividad cooperativa entre los distintos procesos distribuidos.

3. Que por último, cuando todos los programas ejecutándose en paralelo hubieran finalizado, retornaran sus resultados y los datos estadísticos relevantes para su registro y evaluación.

Concluyendo, se convino que un enfoque "Tightly-coupled software" sobre un "Loosely-coupled hardware" era el más adecuado para el tipo de aplicación que se proponía realizar, adaptándose a las necesidades mencionadas anteriormente.

4.3 LOS PARÁMETROS DE ENTRADA AL SISTEMA

Además de los parámetros propios del algoritmo (operadores, tamaño de la población, número de generaciones, etc.), cada componente del algoritmo genético paralelo ejecutando sobre cada uno de los procesadores, deberá proveer los parámetros utilizados en los servicios de migraciones. Entre estos parámetros se encuentran:

- el *número de cromosomas* a enviar en cada migración.
- el *intervalo de tiempo* entre sucesivas migraciones de cromosomas, y
- el *intervalo de tiempo* entre aceptación de cromosomas.

Estos parámetros pueden ser una función dependiente del tiempo (enfoque estático) o del estado de la población (enfoque dinámico).

4.4 LAS RUTINAS PARA EL SERVICIO DE MIGRACIONES

Las rutinas que proveen el servicio de migración son independientes de los intervalos entre envíos y entre aceptaciones. También son independientes de la política a adoptarse con los cromosomas (ej. selección de cromosomas para migrar). Todas estas responsabilidades corresponden al Algoritmo Genético que utilice éstas rutinas. Las siguientes rutinas son provistas por el sistema:

enviar_cromosomas (EC): es invocada por el GA cuando el proceso local desea migrar cromosomas a los procesos remotos asociados en la red. Previo al llamado de esta rutina el GA deberá haber escogido los cromosomas candidatos (probablemente sus mejores individuos) que serán distribuidos a los otros componentes.
El número de cromosomas involucrados en cada transacción es variable. Este dato junto con un vector de índices que apunte a los cromosomas candidatos deberán ser pasados a esta rutina como parámetro para concretar su envío. Finalmente los cromosomas serán enviados dentro de un paquete con el siguiente formato:

Cantidad de Cromosomas	Longitud de Cromosomas	Lista de Cromosomas

Para ésta aplicación inicial, se asume que todos los cromosomas tienen la misma longitud. Sin embargo, el sistema contempla la posibilidad de migrar cromosomas de longitud variable (estructuras multicromosómicas), en paquetes de distinto formato. Esta rutina mantiene un registro de las acciones de envío que almacena

información acerca del número de cromosomas enviados para compararla con los módulos de recepción.

recibir_cromosomas (RC): el trabajo de esta rutina consiste en ir almacenando en un IPCP[1], a medida que arriben, los cromosomas provenientes de procesos remotos que corren en otros procesadores. Para ello permanece en escucha de la red. RC es un proceso (hijo) independiente del proceso (padre) que corre el GA quién ocasionalmente requerirá cromosomas externos, por medio de otra rutina (SC). Finalmente obtiene información sobre el número de cromosomas recibidos y de donde provienen. Con esto se realiza un control de la comunicación (cromosomas perdidos en el medio, etc.).

solicitar_cromosomas (SC): esta rutina es la encargada de obtener cromosomas desde un IPCP que conecta el proceso padre con el proceso hijo.

Luego de cada generación, el proceso padre (encargado de correr el GA) si lo desea, solicitará a EC el envío de cromosomas y requerirá a SC cromosomas que han arribado desde otros GAs, ahora almacenados en un IPCP. Al retornar el control al GA, éste deberá escoger el número y cromosomas candidatos a reemplazar. Se asume que los paquetes recibidos tienen el formato descripto anteriormente.

4.5. LA COMUNICACIÓN ENTRE PROCESOS

La performance de un sistema distribuido depende en gran medida de la eficiencia alcanzada por el sistema de comunicación entre procesos. Este afecta a los procesos que envían y reciben mensajes por dos razones fundamentales:

a) Los procesos EC y RC son los componentes que van a permitir la interacción con otro u otros procesos del sistema distribuido. Para esto se debió escoger un mecanismo de comunicación apropiado que permitiese la transferencia de datos desde el ambiente del proceso emisor al de los procesos receptores. Claramente existe una situación en la que la comunicación involucra a múltiples procesos (cada algoritmo genético migra sus cromosomas al resto de los algoritmos genéticos). Para implementar esta forma de comunicación, de uno a muchos, se consideró "broadcasting message" como el método más apropiado. Estas rutinas utilizan una interfase "socket" como abstracción del mecanismo de comunicación entre procesos. Se decidió utilizar un protocolo sin conexión (UDP) por varias razones:

- Esencialmente, para evitar el "overhead" involucrado en un protocolo con conexión.
- Al estar ejecutando sobre una LAN, la pérdida o corrupción de paquetes es muy poco probable, lo cual ha sido comprobado en las pruebas realizadas hasta el momento. Aún cuando existiese alguna pérdida de cromosomas en la comunicación, la ejecución del GA no se vería sensiblemente afectada.
- El flujo de datos no es continuo, más bien es esporádico, de pocos datos y no requiere reconocimiento.

[1] De ahora en más, no se hará distinción entre el primitivo de comunicación entre procesos propiamente dicho y sus partes componentes (procesos o estructuras de datos).

- Por último, UDP, por ser un protocolo sin conexión, permite realizar una comunicación del tipo "broadcast".

b) Cada interacción u operación de comunicación entre uno o varios procesos provoca la necesidad de sincronizar las actividades de envío con las de recepción. Básicamente se debió considerar la necesidad de introducir una clase de IPCP no bloqueante. De esta forma el GA puede seguir ejecutando aún cuando no se hayan recibido cromosomas en la IPCP desde otros procesos.

Otro punto a considerar y que afecta la eficiencia del sistema de comunicación es la *granularidad del paralelismo* para las computaciones involucradas. Aplicaciones que permiten gran número de computaciones independientes en cada procesador antes de requerir una comunicación entre procesos remotos, se denominan de *gránulo grueso*. La arquitectura expuesta anteriormente (fig. 2) se adecua a este tipo de sistemas dado que las migraciones se realizan esporádicamente.

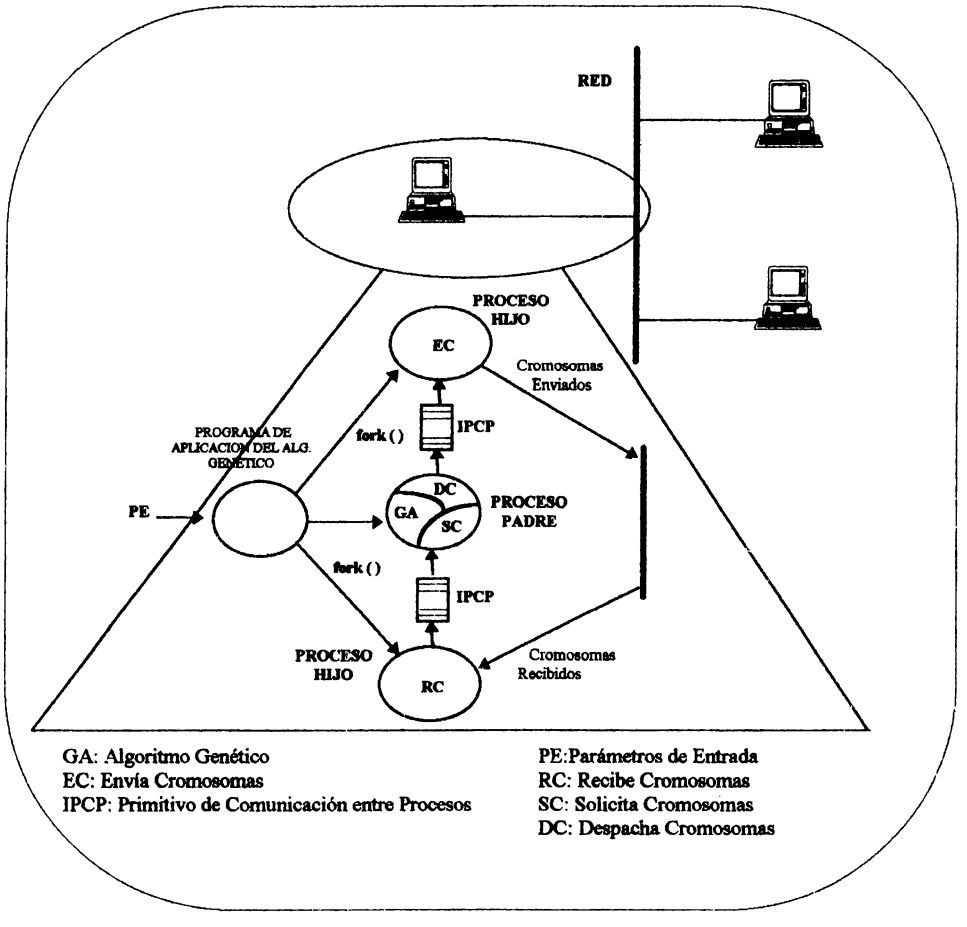


Fig.3 ARQUITECTURA ALTERNATIVA DEL SISTEMA PARA SERVICIO DE MIGRACIÓN

Para los casos en que las aplicaciones necesitaran realizar mayor número de (o más frecuentes) migraciones, se pensó en una arquitectura alternativa para contrarrestar algunos de los efectos negativos de la disminución de la granularidad del paralelismo. La opción mostrada en fig. 3 tiende a evitar las demoras en que podría incurrir el proceso padre, encargado de correr el GA, al intentar enviar sus paquetes a la red.

Con esta variante sería posible que, en lugar de esperar por el envío, continuase procesando una nueva generación. Estas demoras pueden ocurrir si el medio estuviese saturado por el tráfico producido por otros procesadores en situación similar.

Brevemente, este nuevo enfoque propone que el proceso inicial se divida en tres procesos; el proceso padre, encargado de ejecutar esencialmente el algoritmo genético[1], y dos procesos hijos. Uno, encargado de recibir los cromosomas que arriban desde otros procesos por la red y mantenerlos disponibles en un IPCP. El otro, encargado de enviar los cromosomas locales hacia otros procesos. Ambos hijos estarían comunicados por medio de un IPCP con el proceso padre.

5. CONCLUSIONES

- Un campo promisorio de los Sistemas Distribuidos es la explotación del poder computacional disponible en red, obteniendo mejoras en los tiempos de ejecución y la calidad de los resultados de programas paralelizables. Como los Algoritmos Genéticos son algoritmos intrínsecamente paralelos se espera que su performance pueda mejorarse cuando corren en una arquitectura de máquina paralela.
- Este trabajo presenta una arquitectura factible para un sistema que soporte la migración de cromosomas para procesamiento paralelo de Algoritmos Genéticos. Se adoptó un enfoque de Algoritmo Genético Paralelo de granularidad gruesa, de acuerdo con el esquema de clasificación de Levine, para procesamiento de GAs en una red de computadoras.
- Actualmente el sistema está en estado experimental y es el paso inicial de un estudio para encontrar, en cada caso, la arquitectura de memoria distribuida más apropiada para distintos modelos de algoritmos genéticos paralelos.

6. AGRADECIMIENTOS

Agradecemos la colaboración de los restantes integrantes de la línea de investigación por el aporte de críticas constructivas a las ideas aquí expuestas. Asimismo al CONICET y la UNSL de quienes recibimos continuo apoyo.

7. BIBLIOGRAFÍA

[1] Colouris G., Dollimore J., Kindberg T. -Distributed Systems: Concept and Design - Addison-Wesley, 1994.
[2] Goldberg, D.E. - Genetic Algorithms in Search, Optimization and Machine Learning- Addison-Wesley, Reading, MA, 1989.
[3] Grefenstette J. J. - Parallel Adaptive Algorithms for Function Optimisation - Technical report Nr CS-81-19, Nashville: Vanderbilt University, Computer Science Dept, 1981.

[1] El proceso padre también contiene la rutina **despachar_cromosomas** (DC), encargada de transferir los cromosomas que se desea migrar al IPCP.

[4] Grefenstette J. J., Baker J. - A Critical Look at Implicit Parallelism - Proceedings of the Third International Conference on Genetic Algorithms, pp 20-27, San Mateo, 1989, Morgan Kauffman.
[5] Holland J. H. - Outline for a Logical Theory of Adaptive Systems - Journal of the ACM, 3, pp 297-314, 1962.
[6] Holland, J.H. - Adaptation in Natural and Artificial Systems. Ann Arbor; The University of Michigan Press. 1975.
[7] Levine D. - A Parallel genetic Algorith for the Set Partitioning Problem - Ph D Thesis, Illinois Institute of Technology and Argonne national Lab (ANL-94/23), 1994.
[8] Michalewicz, Z. - Genetic Algorithms + Data Structures = Evolution Programs - Springer-Verlag, 1994.
Tanenbaum, Andrew S. - Sistemas Operativos Modernos - Prentice Hall, 1992.