UNA ESTRATEGIA EVOLUTIVA DE PREDICCION PARA BALANCE DE CARGA EN ENTORNOS DE SISTEMAS DISTRIBUIDOS

ESQUIVEL, Susana C. y LEGUIZAMON, Guillermo M.
Grupo de Interés en Sistemas de Computación[1]
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106
5700 - San Luis
Argentina
E-mail:esquivel@inter2.unsl.edu.ar
legui@inter2.unsl.edu.ar
Teléfono: 54++ 652 20823
Fax : 54++ 652 30224

RESUMEN:

Este artículo propone una estrategia evolutiva de predicción, en un entorno de balance de carga, en Sistemas Distribuidos.
Los algoritmos de balance de carga intentan mejorar la *performance* del sistema recurriendo a la migración de procesos, pero presentan el problema adicional de la inestabilidad que ocurre cuando los procesos gastan una cantidad excesiva de tiempo migrando entre los diferentes nodos del sistema.
Con el fin de disminuir el tráfico de comunicaciones generadas por la estrategia de balance, se buscó reducir el número de requerimientos que un nodo debe efectuar cuando se encuentra sobrecargado, utilizando para ello una estrategia de predicción evolutiva que usa como base de la predicción el aprendizaje logrado, a través de la experiencia previa, por cada nodo.

PALABRAS CLAVES: Sistemas Distribuidos, Balance de Carga, Predicción, Aprendizaje, Algoritmos Genéticos.

[1]El grupo de Investigación está financiado por la Universidad Nacional de San Luis y el CONICET y su Director es el Msc. Raúl H. Gallard.

1. INTRODUCCION

Un conjunto de procesadores conectados en una red de área local (LAN) constituye un ejemplo clásico de un Sistema Distribuido.
El modelo considerado en este trabajo consta de un conjunto de nodos homogeneos e independientes, esto es, nodos que tienen una arquitectura, de hardware y software, semejantes pero donde cada uno de ellos tiene sus propios recursos de procesamiento y almacenamiento de la información.

En un sistema con tales características los usarios, desde diferentes puestos de trabajo, generan procesos autónomos que entre ellos no requieren más sincronización que la necesaria para competir por el uso de recursos críticos.
La carga de trabajo de un nodo de la red consiste del conjunto de demandas propias, y/o de los otros nodos, para el uso de todos o algunos de sus recursos. Esta demanda varía durante la ejecución de los procesos, pudiendo conducir a un estado de desequilibrio en el sistema; es decir, que parte de los nodos de la red pueden estar sobrecargados de requerimientos mientras otros están subocupados o, incluso, ociosos.

Con el objetivo de mejorar la performance del sistema, en cuanto a un uso más balanceado de sus recursos, el sistema utiliza, en forma dinámica y automatizada, una estrategia evolutiva de balance de carga .
En una presentación anterior [1], se mostró cómo una estrategia de balance de carga evolutiva, basada en Algoritmos Genéticos (cuyo objetivo fue optimizar el tiempo medio de permanencia de los procesos en el sistema, es decir su tiempo medio de respuesta, a través de una mejor utilización del recurso procesador) lograba mejores resultados cuando se la contrastó con otros algoritmos de balance de carga tradicionales.

El presente trabajo tiene por finalidad mostrar los resultados obtenidos al aplicar una estrategia de predicción, que asumiendo la existencia de una estrategia evolutiva de balance de carga, aprovecha el aprendizaje que cada nodo del sistema hace durante el proceso y trata de disminuir no ya el número de migraciones sino el número de requemientos que un nodo sobrecargado debe efectuar a los otros nodos de la red antes de poder migrar un proceso, logrando de este modo un aun mejor comportamiento del sistema.

2. DISEÑO DE LA ESTRATEGIA DE BALANCE DE CARGA

La fig.1 muestra los distintos componentes que implementan la estrategia de balance dentro de cada nodo del sistema.

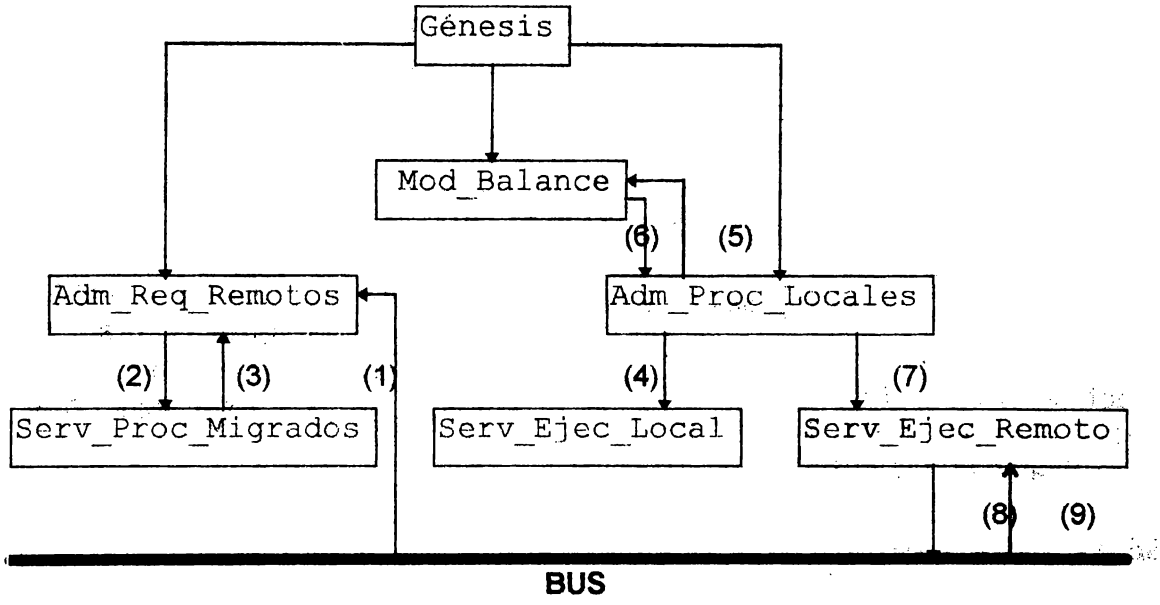

Fig.1: Módulos de la Estrategia de Balance

La funcionalidad de cada módulo se describe, brevemente, a continuación:

GENESIS, se ejecuta una única vez, en el momento de *booting* del nodo, y tiene por finalidad activar los tres módulos centrales del sistema.

MOD_BALANCE, este es el módulo que implementa la estrategia de balance de carga. En el sistema simulado contiene distintos algoritmos de balance para poder efectuar el estudio de la *performance*. En la próxima sección se describirá, con más detalle, la operación del Algoritmo Genético que implementa la estrategia de balance diseñada.

ADM_REQ_REMOTOS, cumple dos tareas principales, a saber:

- **(1)** da respuesta a los requerimientos de migración de los otros nodos de la red, brindándoles información acerca del estado de su carga (longitud de cola del procesador) o, eventualmente, si previamente recibió un proceso migrado desde otro nodo y éste finalizó su ejecución, informa de esta situación al nodo emisor.

- **(2)** Cuando el nodo está ocioso o con carga leve (**long_cola** ≤ **2**), es decir, en condiciones de aceptar procesos de otros nodos de la red que estén sobrecargados y, efectivamente, le arriba un proceso remoto entonces activa un proceso hijo servidor.

SERV_PROC_MIGRADOS, ejecuta localmente un proceso que ha migrado desde otro nodo, cuando finaliza la ejecución le envia una señal de aviso al administrador **(3)**.

ADM_PROC_LOCALES, cuando le arriba un proceso, es el encargado de verificar el estado de su carga, comparando la longitud de cola del nodo con el umbral que se haya establecido para determinar la sobrecarga, en función de esta comparación su actividad se diferencia según se indica a continuación:

- Si **long_cola** ó **umbral** entonces la tarea se ejecuta localmente para lo cual activa un módulo hijo ***SERV_EJEC_LOCAL*** **(4)**.

- En otro caso, invoca al ***MOD_BALANCE*** **(5),** quién le indicará si puede enviar el proceso y a qué nodo **(6)** y crea un ***SERV_EJEC_REM*** **(7)** o, en su defecto, deberá ejecutarlo localmente (lo cual indicaría que los otros nodos del sistema están sobrecargados) y entonces se comporta como fue explicado en el párrafo anterior.

SERV_EJEC_LOCAL, es el encargado de ejecutar un proceso en dicho nodo.

SERV_EJEC_REM, es el encargado de efectuar la migración del proceso, **(8)** que le pasó el ***ADMIN_PROC_LOC,*** hacía el nodo que le indicó el módulo de balance y, luego, se bloquea a la espera de la respuesta de finalización de ejecución remota **(9).**

3. BREVE DESCRIPCION DE LA ESTRATEGIA DE BALANCE DE CARGA

La estrategia propuesta, ante la necesidad de migrar una tarea, procede de la siguiente manera:

a) El distribuidor de tareas del nodo, en primer lugar, envía un requerimiento de información de carga a un subconjunto de nodos de la red.

b) Los receptores mandaan una respuesta de aceptación o rechazo, **x** ó **0**, dependiendo de los estados de sus propias cargas. Donde $0 < \mathbf{x} \leq 2$ indica la longitud de cola del nodo que responde.

c) Si dos o más nodos le retornan un mensaje de aceptación, entonces el distribuidor de tareas del nodo sobrecargado elige, para hacer migrar la tarea, al nodo que respondió con menor longitud de cola.

La estrategia se implementa siguiendo un enfoque genético. Para ello, cada nodo tiene su propia población, sobre la cual se aplican los operadores genéticos habituales de selección, crossover, mutación y elitismo [2].
Cada individuo de la población está codificado como un vector binario $<p_0,p_1,....,p_n>$, donde cada uno representa al conjunto de procesadores a los cuales el nodo, actualmente sobrecargado, en alguna oportunidad anterior, envió requerimientos para migrar tareas. La semántica de cada valor posible, para cada p_i, es la siguiente:

- Si $p_i = 1 \Rightarrow p_i$ ***aceptó*** el pedido de migración de tarea la última vez que le fue requerido.

- Si $p_i = 0 \Rightarrow p_i$ ***rechazó*** el requerimiento por estar sobrecargado.

Cuando el módulo ***ADM_PROC_LOCALES*** de un nodo determina que está sobrecargado, efectúa un pedido de migración al módulo ***MOD_BALANCE*** y se libera para poder atender nuevos requerimientos.
El módulo ***MOD_BALANCE*** selecciona de su población un individuo (vector o cromosoma) con una probabilidad proporcional a su valor de adaptabilidad (fitness) y envía requerimientos de migración a los nodos indicados por dicho vector, luego se bloquea a la espera de una respuesta.
Cuando ésta arriba, si le llega más de una positiva,decide seleccionar al nodo que tenga menor longitud de cola, si más de uno tienen igual longitud de cola selecciona uno al azar y, finalmente, si todos los nodos, a los cuales le mandó un requerimiento, le responden negativamente (esto es, que no pueden aceptar una nueva tarea porque están sobrecargados) entonces le indica al ***ADM_PROC_LOCALES*** que deberá ejecutar la tarea localmente.
Luego de enviar la información al módulo ***SERV_EJEC_REMOTA*** para que efectúe la migración de la tarea, o en su defecto la haga ejecutar localmente, el ***MOD_BALANCE*** calcula el valor de fitness del cromosoma y crea una nueva población aplicando los operadores genéticos a la vieja población, los operadores utilizados son el elitismo, crossover de dos puntos y la mutación uniforme [2,3].
El proceso de aprendizaje se produce, precisamente, a través de la función de fitness, que tiene en cuenta los premios otorgados a los individuos de la población para determinar sus nuevos valores de adaptabilidad.
Este premio es proporcional al número de requerimientos enviados, es decir, el valor del premio será mayor para aquéllos individuos que envíen requerimientos a menos nodos, salvo que todos sean rechazados, en cuyo caso el premio es cero. Si esto último sucede el individuo obtendrá un valor de fitness cero, y no tendrá casi oportunidad de ser seleccionado la próxima vez.

4. DESCRIPCION DE LA ESTRATEGIA DE PREDICCION PROPUESTA

En la sección anterior se señaló que en el momento en que un nodo sobrecargado necesitaba efectuar la migración de un proceso el ***MOD_BALANCE*** seleccionaba el mejor cromosoma de su población local. Sea, por ejemplo, un cromosoma con el siguiente formato:

$< 1,0,-,1,1,1,1,1,0,0,1,1,1,1,1,1,1>$

indicaría que los procesadores 1, 4, 5, 6, 7, 8, 11, 12 , 13, 14, 15, 16 y 17 son los candidatos para recibir un requerimiento de migración y hacía ellos dirigía su pedido de migración.

Con la finalidad de disminuir el número de requerimientos que el nodo p_3, (en este caso) debe realizar se decidió implementar una estrategia de predicción que por un lado, aprovechase el conocimiento que el propio nodo tiene (subconjunto de procesadores que ya han aceptado un requerimiento) pero, que a la vez, sea más eficiente y seleccione, cada vez, no más del 50% del total de nodos candidatos para hacer requerimientos, por ejemplo, en este caso debería efectuar requerimientos a lo sumo a ocho nodos.

Para seleccionar a qué nodos efectuar solicitudes se decidió efectuar predicciones utilizando la técnica de Promedio Exponencial Ponderado [4], la cual permite predecir un valor (en este caso a qué procesador hacer un requerimiento de migración) sobre la base de valores que han ocurrido a lo largo del tiempo (dichos valores representando a procesadores que previamente aceptaron migraciones).

Teniendo ésto en cuenta la formulación más simple de la predicción sería:

$$V_{n+1} = 1 / n \bullet \sum_j P_i \text{ con } i = 1,2,...,n \quad (1)$$

donde:

V_{n+1} = Predicción de la respuesta (aceptación o no del requerimiento) de un procesador.

P_i = Respuesta efectiva del último requerimiento a ese procesador.

A los efectos de evitar recalcular la sumatoria completa cada vez, la ecuación (1) puede reescribirse como:

$$V_{n+1} = 1 / n \bullet P_n + (n - 1) / n \bullet V_n \quad (2)$$

En la ecuación (2) se da igual peso a cada uno de los valores observados. En realidad, para la situación que se modela es deseable, en general, dar más peso a la historia reciente porque ella reflejará, más probablemente, el comportamiento futuro del sistema. Entonces usando la formulación del promedio exponencial ponderado se tiene:

$$V_{n+1} = \alpha \bullet P_n + (1- \alpha) \bullet V_n \text{ , con } 0 \leq \alpha \leq 1 \quad (3)$$

Al usar en la ecuación (3) un valor constante para α, independiente del número de observaciones pasadas, sucede que todas ellas son consideradas pero la historia más antigua tiene menor peso. Esto puede visualizarse más claramente al realizar la expansión de la ecuación (3):

$$V_{n+1} = \alpha \bullet P_n + (1 - \alpha) \bullet \alpha \bullet P_{n-1} + \ldots + (1 - \alpha)^i \bullet P_{n-1} + \ldots + (1 - \alpha)^n \bullet V_0$$

porque α y $(1 - \alpha)$ son menores que 1, cada término sucesivo en la expansión precedente tiene menos peso que su sucesor.
La ecuación (3) permite controlar, en nuestra predicción, el peso relativo de la información actual y antigua.
Por ejemplo, si α tiende a cero se estaría considerando que la historia reciente no tiene efecto (las condiciones actuales son transitorias); en tanto que si α es igual a 1 sólo interesa la historia reciente y se asume que la historia antigua es obsoleta o irrelevante.
Volviendo al cromosoma del ejemplo bajo este nuevo enfoque el ***MOD_BALANCE***, en el paso a) de la sección anterior, no enviará un requerimiento a todos los procesadores que tengan el correspondiente bit del cromosoma en 1 sino que sucesivamente hará predicciones acerca de cuál es el mejor subconjunto (a lo sumo el 50% de los nodos candidatos) para enviarles su solicitud de migración.
Para implementar esta política sólo hizo falta adicionar una tabla, con m entradas, siendo m la cantidad de procesadores del sistema, dónde en cada una de ellas se almacena el valor de V_n y P_n se obtiene de la respuesta del nodo.

5. DESCRIPCION DE LOS EXPERIMENTOS REALIZADOS

5.1. Software de Simulación

El Sistema Distribuido, que implementa la estrategia, fue simulado utilizando una herramienta de modelización de sistemas de computación, PARASOL [5], que está orientada hacía los modernos sistemas de computación distribuidos o paralelos.
PARASOL consiste de un conjunto de bibliotecas escritas en lenguaje C; además de las facilidades típicas encontradas en otros sistemas de simulación discreta, PARASOL ofrece un conjunto de constructores que facilitan la definición del hardware del sistema y su topología de red. PARASOL corre bajo sistemas operativos UNIX o semejantes.

5.2. Parámetros del Sistema Distribuido

- Número de nodos: 10, 16 y 20.
- Cada nodo ejecuta procesos en forma concurrente con una disciplina *round-robin*, manteniendo una cola de procesos en condiciones de ser ejecutados pero que están esperando por su *time-slice*.
- Topología de la red: *Ethernet*.
- Tipo de procesos: intensivos en el uso del procesador.
- Tiempo de servicio : fijo para todos, de 1seg. de procesador.
- Tamaño de los procesos: fijo para todos en 64 Kb.
- Velocidad de transferencia de 10 Mbits.
- Los procesos que arriban en cada nodo se generan con una distribución Poisson , con valor medio λ. Distintos valores de λ fueron utilizados a los efectos de analizar distintos niveles de carga en el sistema.

- La simulación finaliza cuando se ejecutan 10000 procesos en los diferentes nodos de la red.

5.3. Parámetros para los Algoritmos Genéticos

- Tamaño de la población en cada nodo: 10 individuos.
- Probabilidad de crossover: 0,5.
- Probabilidad de mutación: 0,05.

Estos valores de los parámetros fueron determinados después de varias corridas experimentales, realizadas con la finalidad de "poner a punto" el algoritmo.

5.4. Estrategias de Balance de Carga Comparadas

Como ya fue mencionado, para que el ***ADM_PROC_LOCALES*** de un nodo pueda determinar si puede o no atender a una tarea que le arriba, utiliza la longitud de cola del procesador del nodo. En tal sentido, se fijaron los siguientes criterios para la carga de cada nodo:

- **long_cola** > 4 $\Rightarrow$ el nodo está sobrecargado
- 3 ó **long_cola** ó 4 $\Rightarrow$ carga media
- **long_cola** ó 2 $\Rightarrow$ carga baja, puede aceptar migraciones..

Las estrategias de balance comparadas son dos: una, la **Est_Evol_1** presentada en [1] en la que cuando un procesador necesita hacer una migración envía requerimientos a todos los nodos candidatos y, la otra **Est_Evol_2** , propuesta en el presente trabajo, que predice a qué subconjunto de los nodos candidatos deberá hacerle el requerimiento.
Para la **Est_Evol_2 ,** luego de varias corridas se fijo un $\alpha = 0.5$ y un valor de predicción inicial $V_0 = 0$ para todos los nodos.

Ambas son emisor_iniciadas, es decir, el pedido de migración lo efectúa el nodo que está sobrecargado, a diferencia de las receptor_iniciadas, en los cuales los nodos ociosos requieren a los otros si tienen procesos para remitirles.

5.5. Escenarios para la Realización de los Experimentos

Los experimentos fueron realizados sobre tres scenarios diferentes, usando un tiempo medio entre arribos $1/\lambda$, con $\lambda = 0.1, 0.2, ..., 0.9, 1$.

- **Escenario 1:** Un subconjunto de nodos, 40%, recibe procesos con igual velocidad de arribo. mientras al resto de los nodos de la red no arriba ningún proceso. Esto permite simular un sistema con un grupo de nodos con un nivel de carga alto.

- **Escenario 2:** Un 40% de los nodos recibe procesos con una velocidad (λ) de arribo baja y el 60% restante con una velocidad alta (2λ).
- **Escenario 3:** Cada nodo tiene una velocidad de arribo diferente y su valor establecido en función del tiempo, es decir, $\lambda_i(t)$.

El escenario 1, trató de reflejar un aspecto de la realidad, tal es que no siempre todos los nodos de una red reciben procesos con un nivel de carga semejante.
Los escenarios 2 y 3, son parecidos, en el sentido en que en todos los nodos se producen arribos, la diferencia radica en que en el escenario 3 las velocidades de arribo son dependientes del tiempo en que se produce el arribo en cada nodo para reflejar, por ejemplo, que en determinada horas del día los requerimientos varian.

5.6. Resultados de la Simulación

5.6.1. *Escenario 1*

En esta situación es donde si no existiese un algoritmo de balance de carga el sistema degrada su performance a medida que el tiempo entre arribos de los procesos disminuye, puesto que existe un subconjunto de nodos que rápidamente se sobrecarga al recibir más procesos de los que puede manejar y, en consecuencia, aumenta el tiempo que los procesos permanecen en el sistema.
La fig.2 a) muestra que si bien las dos estrategias de balance mejoran el tiempo medio de respuesta, la estrategia Est_Evol_2 es la que mejor se comporta.
Un comportamiento semejante se obsevó para 10, 16 y 25 nodos.

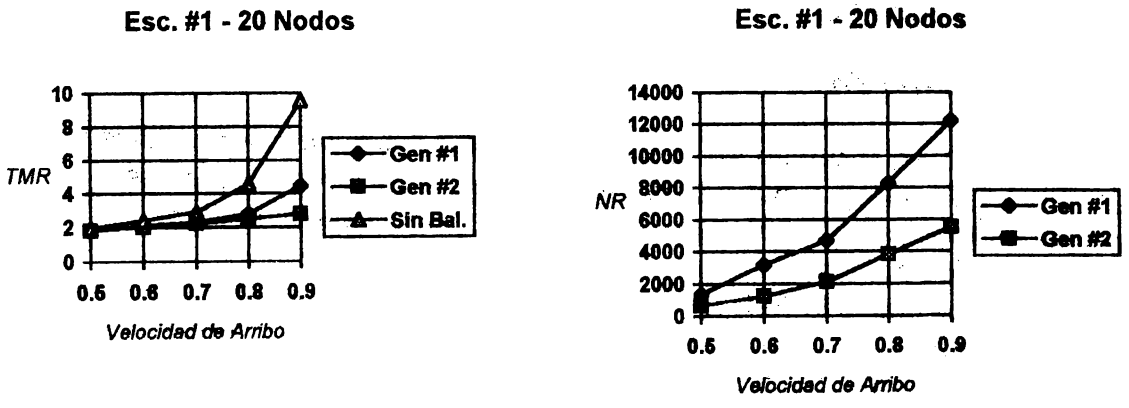

Fig.2: a) TMR = Tiempo medio de Respuesta; b) NR = Nº de requerimientos

La fig. 2 b), muestra como la **Est_Evol_2,** utilizando su capacidad predictiva disminuye considerablemente la cantidad de requerimientos efectuada respecto **de la Est_Evol_1**.

5.6.2. *Escenario Dos*

En la fig. 3 a) puede observarse un comportamiento similar, en el sentido que para cargas bajas y medias las estrategias se comportan, en general, adecuadamente manteniendo bajo el tiempo medio de respuesta del sistema. Ya para niveles de cargas medias a altas, la **Est_Evol_2** reduce el Tiempo Medio de Respuesta a casi la mitad de la **Est_Evol_1**.

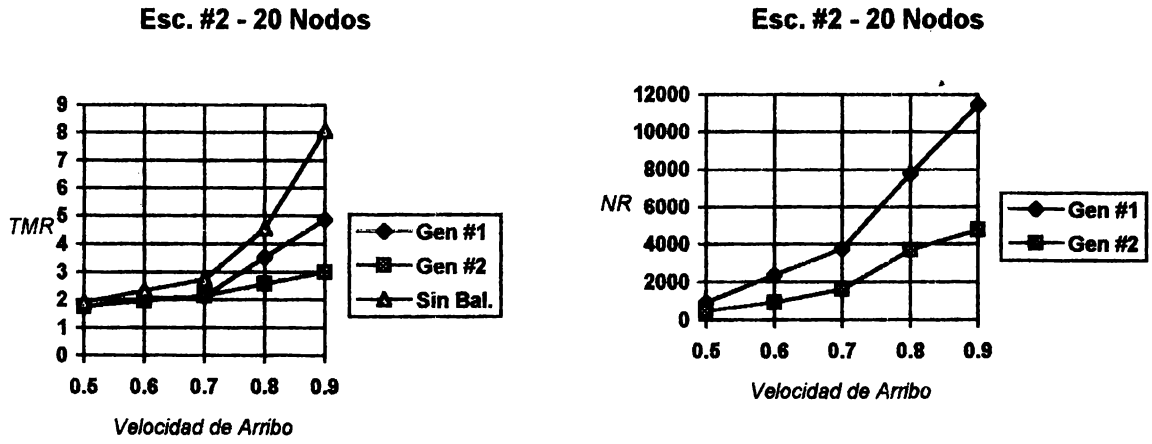

Fig.3: a) TMR = Tiempo Medio de Respuesta; b) NR = N° de Requerimientos

Nuevamente, puede verse en la fig. 3 b que con la estrategia predictiva se reduce el número de requerimientos en más de la mitad. Los mismos resultados se mantienen para 10, 16 y 25 nodos.

5.6.3. *Escenario Tres*

En la tabla 1 se resumen los resultados para las simulaciones, utilizando 10, 16, 20 y 25 nodos, y donde las velocidades de arribo de los procesos a los nodos es diferente para cada nodo de la red. Aqui nuevamente se mantienen los resultados analizados en el escenario anterior.
En todos los casos la **Est_Evol_2** , obtiene un tiempo medio de respuesta del sistema notablemente inferior, a la vez que el número de requerimientos se reduce en aproximadamente entre un 20 y 30 por ciento.
Puede observarse que en el caso de una red con 10 nodos si bien el número de migraciones fallidas (MF) aumenta, esto no se reflejó en una degradación de la performance sino todo lo contrario. Es de hacer notar que al ser sólo 10 nodos el tamaño del cromosoma será de 10, y al reducir el subconjunto de nodos a requerir migraciones a lo sumo al 50% de los nodos candidatos, en el

mejor de los casos sólo se hará un *broadcast* a 5 nodos, en consecuencia un mayor número de migraciones fallidas era esperable.
En los casos que aumenta el número de nodos la performance de la estrategia mejora en cuanto al número de fallas de migraciones comportándose de manera análoga a la **Est_Evol_1**, la que no reduce el número de requerimientos.
También un mayor número de nodos mejora el aprendizaje realizado por cada nodo, ya que aumenta el tamaño de los cromosomas, esto da lugar a una mayor diversidad genética de la población lo cual facilita la tarea del algoritmo genético subyacente y, consecuentemente, al tener un cromosoma de partida de mejor calidad mejora la predicción efectuada.

	Est_Evol_1			*Est_evol_2*		
Nodos	*TMR*	*NR*	*MF*	*TMR*	*NR*	*MF*
10	11.49	21877	10	4.13	9117	745
16	18.95	33847	2	5.40	13621	3
20	29.05	47071	0	9.92	20332	0
25	37.44	56029	0	13.03	24965	0

Tabla 1: Escenario 3

Es de hacer notar que se obtuvieron resultados semejantes variando la semilla del generador de números random.

Observar que en todos los escenarios no se pudieron graficar los resultados para valores de $\lambda > 0.9$, esto debido a que aumentaba rápidamente la sobrecarga de los nodos, como puede observarse en las diferentes figuras , produciéndose *overflow* en las colas del sistema de simulación.

6. CONCLUSIONES

En este artículo se han presentado los resultados de la comparación de dos estrategias evolutivas de balance de carga en sistemas distribuidos.
Ya en un artículo anterior [1] se mostró que el tiempo medio de respuesta del sistema distribuido mejora siempre que una estrategia de balance se aplica, en todos los experimentos de simulación llevados a cabo, se observó que los dos enfoques genéticos obtuvieron mejores resultados que algoritmos tradicionales de balance.
Sin embargo, en este estudio anterior sólo se buscaba minimizar el *overhead* producido por los procesos que migran por el sistema.
La óptica con que se encaró el trabajo presente no es sólo atender a la migración propiamente dicha de los procesos sino también tratar de disminuir el número de requerimientos que un procesador sobrecargado debe efectuar antes de migrar un proceso. En otras palabras reducir el tráfico en el sistema cuando se aplica una estrategia de balance.
Para cumplir este objetivo se enriqueció la estrategia evolutiva incorporándole al módulo de balance una función de predicción, que aprovechando el

conocimiento del nodo, permitiese reducir el número de nodos a consultar ante la presencia de sobrecarga.
Los resultados experimentales obtenidos a través de la simulación muestran la bondad de la estrategia propuesta.
Si bien la estrategia de balance presentada se aplicó en un ambiente de procesos independientes, los autores ya han trabajado con un enfoque genético para un esquema de distribución de procesos paralelos que interactuan en un sistema distribuido con un enfoque estático [6], por lo tanto, un trabajo futuro será el de adaptar la estrategia genética de balance de carga aqui presentada para que pueda aplicarse en ambientes donde ambos tipos de procesos convivan y contrastándola además con algoritmos que pertenezcan al modelo receptor-iniciados.
Además también es trabajo futuro ajustar la estrategia para que trabaje con distintas métricas [7] y no sólo la longitud de cola del procesador.

7. REFERENCIAS BIBLIOGRAFICAS

[1] Esquivel S. y Leguizamón G. - "Una Estrategia Evolutiva de Balance de Carga para un Sistema Distribuido", a ser publicado en los Anales de las 25 JAIIO, Setiembre de 1996.

[2] Goldberg D. E., - " Genetic Algorithms in Search, Optimization & Machine Learning", Addison Wesley, 1989.

[3] Michalewicz Z. - " Genetic Algorithms + Data Structures = Evolutions Programs", Springer-Verlag, Second, Extended Edition, 1994.

[4] Stallings William - "Operating Systems", MacMillan Publishing Company, New York, 1992.

[5] Neilson J., - " Parasol User's Manual ", School Of Computer Science, Carleton University, Canada.

[6] Esquivel S., Leguizamon G., Gallard R. - "An Evolutionary Approach for Cluster Allocation of Parallel Program Tasks in Distributed Systems ", Proceeding of the 10º International Conference on Control Systems and Computer Science, Universidad Politécnica de Bucarest, Rumania , Mayo de 1995.

[7] Mehra Pankaj - "Automated Learning of Load Balancing Strategies for a Distributed Computer System", PhD Thesis, University of Illinois at Urbana,Champaign, 1993.