Evaluación de Modelos de Predicción de Carga para el Planificador de un Metasistema

Eduardo Izquierdo,†Mariela Curiel
Universidad Simón Bolívar, †Departamento de Computación y T.I.
Apartado Postal 89000, Caracas 1080, Venezuela
eduardo@cesma.usb.ve, mcuriel@ldc.usb.ve

Abstract

The use of metasystems, or systems comprised of other systems, for high performance computing is gaining considerable acceptance. The heterogeneity and dynamic nature of these environments make it challenging to achieve high performance. Scheduling is critical in this sense. Schedulers must base its decisions on the fraction of each resource that is available to it at the time the application will execute. This information is obtained by means of predictive models. In this article we outline the experiences and preliminary results on incorporating predictive models to SUMA metasystem. We focus on their ability to predict the CPU load and the available free memory. Mean based methods and time series models are used. While the best predictor for the CPU load average was an autoregressive integrated model, the last measurement provides the best prediction to the free memory.

Keywords: metasystems, scheduling, distributed systems, statistical forecasting

Resumen

El uso de metasistemas ha ganado considerable aceptación en el mundo de la computación de alto rendimiento. La heterogeneidad y características dinámicas de los metasistemas afectan en forma crítica el desempeño de las aplicaciones que en ellos se ejecutan. El *scheduling* juega un papel importante al momento de lograr el desempeño deseado. Con el objeto de determinar la planificación más adecuada, el *Scheduler* de un metasistema debe basar sus decisiones en el porcentaje del recurso a utilizar que estará disponible para el momento de ejecutar la aplicación. Para ello se vale de modelos predictivos, usualmente modelos estadísticos. En este artículo se describen los modelos estadísticos incorporados al Planificador del metasistema SUMA para predecir la carga del CPU y la disponibilidad de memoria, al momento de asignar una aplicación a una plataforma de ejecución. Se evalúan métodos basados en la media y modelos de series de tiempo. Para predecir la carga del CPU el mejor método de predicción resultó ser un modelo autoregresivo integrado. Para la disponibilidad de la memoria bastó un método sencillo: el predictor *último*, que ofrece como pronóstico para el próximo instante de tiempo, el valor medido en el instante de tiempo anterior.

Palabras claves: metasistemas, planificación, sistemas distribuidos, predicción estadística

1 Introducción

Los avances en la tecnología de redes y las mejoras observadas en la rapidez de los mecanismos de comunicacion entre procesos, han hecho posible el uso de computadores heterogéneos y geograficamente dispersos, como una sola y poderosa máquina virtual, también conocida como metacomputador o metasistema ([15]). Los metasistemas, han ganado considerable aceptación en el mundo de la computación de alto rendimiento. Entre los metasistemas más populares en la actualidad se encuentran Legion[9], Globus[8] y Netsolve[5].

Del concepto de metasistemas se deriva que los recursos que utiliza no están necesariamente dedicados a la ejecución de las aplicaciones. Aunque pudiera existir un subconjunto de plataformas de ejecución dedicadas al metasistema, la gran mayoría son máquinas conectadas en red, que el planificador del metasistema escoge de acuerdo a sus condiciones actuales de carga, sus características de *hardware* y *software*, y las restricciones impuestas por sus propietarios, entre otros factores. Por otro lado, la red que interconecta todos estos recursos dispersos es, en la mayoría de los casos, la Internet. Se trata, entonces, de un conjunto de recursos heterogéneos en sus características de *hardware* y *software*, y que son compartidos por múltiples usuarios ajenos al metasistema, los cuales introducen carga en forma dinámica; a esta carga la denominaremos en adelante "carga externa". Estas características de carga y heterogeneidad afectan en forma crítica el desempeño de las aplicaciones que se ejecutan. Una planificación efectiva de las aplicaciones en el metacomputador es uno de los elementos más importantes en el logro de un alto desempeño [15].

Nuestro proyecto de investigación propone la incorporación al metasistema SUMA de herramientas que permitan mejorar su algoritmo de planificación. Dicho algoritmo es actualmente muy sencillo, porque no considera factores tan importantes como las preferencias del usuario o la carga externa de la red y de las plataformas de ejecución (carga ajena al metasistema). Para mejorar el algoritmo de planificación se incorporarán a SUMA dos tipos de modelos, a saber:

- *Modelos Estadísticos*: A fin de lograr un buen desempeño, un Planificador debe basar sus decisiones en el porcentaje del recurso que estará disponible para el momento de ejecutar la aplicación [3]. Este porcentaje varía dinámicamente en el tiempo y con el uso de los recursos. Con el objeto de predecir las condiciones de carga que tendrán los nodos o la red para el momento en el que se ejecutará la aplicación, se utilizan modelos estadísticos. Estos modelos formarán parte del algoritmo de planificación. Los trabajos presentados en [14] y [15] describen el uso de modelos estadísticos (medias, medianas, modelos autoregresivos) en el sistema *Network Weather Service*, para predecir la carga en la red y el procesador. El *Network Weather Service* ofrece soporte a metasistemas o a planificadores de metasistemas tan importantes como Legion ([9]) o AppLeS ([3]).
- *Modelos de Colas*: Estos modelos estarán a disposición del usuario y le permitirán ganar conocimiento sobre su aplicación. En los modelos de colas estará representado el metasistema, y la idea es que al variar sus parámetros de forma planificada (usando técnicas como el diseño experimental), el usuario pueda descubrir qué es lo mejor para su aplicación entre, por ejemplo, un computador muy rápido con una carga externa muy variable, o un procesador más lento pero totalmente dedicado. La información que se obtenga de los modelos de colas llegará al Planificador como parte de las preferencias del usuario, y le servirá, con ayuda de otros parámetros y datos actuales sobre el estado del metasistema, para hacer la selección de plataformas de ejecución que produzca los mejores tiempos de respuesta.

Este artículo se enfoca en el desarrollo y evaluación de los modelos estadísticos. Se ajustaron e implementaron modelos de series de tiempo, junto con estadísticos más sencillos relacionados con la media. Estos modelos se usan para predecir la carga del CPU y la disponibilidad de memoria en las plataformas candidatas para ejecutar aplicaciones. Para probar la efectividad de cada uno de los métodos se reporta el error cuadrático medio de las predicciones. Un modelo autoregresivo integrado arrojó los mejores resultados en el caso de la carga de CPU. Debido a su menor variabilidad, para la memoria basta un predictor sencillo: el predictor *último*, que da como resultado el valor medido en el instante de tiempo anterior.

El artículo se estructura de la siguiente forma: En la Sección 2 se describe el vehículo de estudio para los modelos predictivos: el metasistema SUMA. Los métodos de predicción estudiados se explican con detalle en la Sección 3. Los pasos para el ajuste de los modelos de series de tiempo se detallan en la Sección 4. La Sección 5 se dedica a la descripción de los experimentos realizados para evaluar los métodos predictivos, y presenta los resultados obtenidos. En la Sección 6 se presentan las conclusiones y se describen actividades que se derivan del trabajo realizado.

2 El Metasistema SUMA

SUMA ([11]) es un metasistema para la ejecución de *bytecode* de Java, tanto secuencial como paralelo, con soporte adicional para cómputo científico. El objetivo principal de SUMA es el de extender el modelo

de máquina virtual de Java para proveer acceso continuo a recursos distribuidos de alto desempeño. El *middleware* de SUMA es orientado a objetos y está construido sobre CORBA.

De un modo general, SUMA presenta tres niveles de componentes (ver figura 1): los clientes (*Clients*), el núcleo del sistema (*Suma Core*) y los Agentes de Ejecución (*Execution Agents*). Los clientes son aplicaciones de muy diversos tipos y funcionalidades que se comunican con el *Suma Core* del sistema proporcionando la información necesaria para solicitar la ejecución de una aplicación Java. El *Suma Core* es el encargado de recibir tales peticiones y de encontrar un Agente de Ejecución adecuado para la aplicación; para ello se vale del *Scheduler* o Planificador. La meta del Planificador de SUMA es optimizar el desempeño de las aplicaciones; a este tipo de Planificadores también se les conoce como (*High Performance Schedulers*). Otros planificadores tratan de optimizar el número total de trabajos ejecutados en el metasistema (*High Throughput Schedulers*) o coordinan, de la forma más justa posible, múltiples accesos a un recurso determinado (*Resource Schedulers*). Los Agentes de Ejecución se encargan de ejecutar las aplicaciones de SUMA. Son procesos, a su vez, que funcionan en plataformas de alto rendimiento y gran capacidad de cómputo, con uno o varios procesadores.

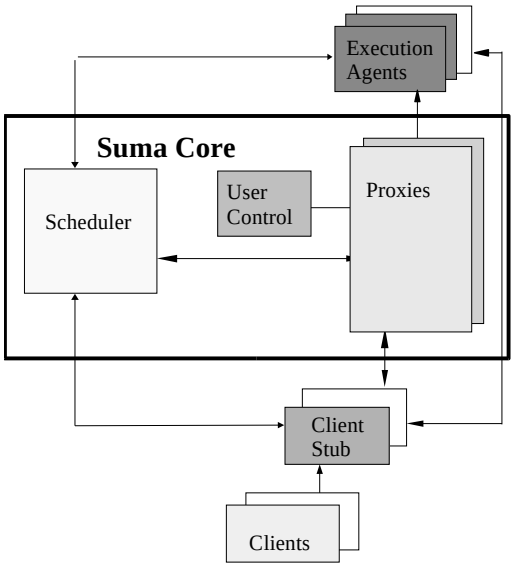

Figura 1: Arquitectura general de SUMA

El modelo de ejecución de programas Java en SUMA es muy simple. El usuario comienza la ejecución de un programa ejecutando un cliente en su máquina. Este cliente invoca el comando *sumaExecute*, correspondiente al modo de ejecución "en línea" (programas interactivos), o el comando *sumaSubmit*, que permite la ejecución "fuera de línea" (para ejecución en *batch*). Una vez que el *Suma Core* del metasistema recibe, por parte del usuario, la solicitud de ejecutar una aplicación, éste se encarga, transparentemente, de buscar una plataforma de ejecución a la que le envía la información necesaria para que comience la ejecución; antes de iniciar el presente trabajo, esta plataforma se seleccionaba únicamente atendiendo a las características de *hardware*. En la plataforma seleccionada por SUMA, el Agente de Ejecución inicia las actividades necesarias para ejecutar el programa.

2.1 El SumaMonitor

El *SumaMonitor* es un módulo integrado a una versión experimental de SUMA[1] que surge con la idea de monitorear y predecir la carga externa en las plataformas candidatas a ejecutar aplicaciones o en las redes que transportan los datos. Se compone de un *thread* que se levanta concurrentemente con cada Agente de Ejecución. Este *thread* se activa en forma periódica y toma el valor de ciertas variables del sistema que llevan estadísticas de rendimiento. Con el valor de estas variables se actualiza la estructura *Load*. Entre la información que recupera el *thread* (hasta ahora únicamente referente a los nodos de ejecución y no a la red)

[1]La versiones experimentales son versiones utilizadas para el desarrollo que deben probarse exhaustivamente antes de incorporarse a la versión de SUMA disponible al público.

se encuentra: el número de usuarios utilizando los recursos de la máquina; promedios de la carga del CPU para los últimos 1, 5 y 15 minutos (estos valores se expresan en el número promedio de procesos esperando por entrar al CPU); cantidad de memoria física disponible; memoria *swap* disponible; tiempos de CPU utilizados en modo usuario, kernel y *idle* (ocioso). Dado que Java no ofrece en la actualidad herramientas para realizar estas labores de monitoreo, la obtención de las estadísticas es dependiente del sistema operativo. Para el sistema operativo Linux, se toma toda la información de varios "pseudo" archivos que se encuentran en el sistema de archivos "/proc". En Solaris, las estadísticas se obtienen a través de JNI (*Java Native Interface*), utilizando las librerías *Kstat, Sysinfo* y *Swap* [12].

Conjuntamente con el *thread* mencionado, se crea un objeto *Predictor*. Este objeto fue concebido para la implementación de los modelos estadísticos. La clase *Predictor* posee, entre otros elementos, los métodos de predicción descritos en la Seccción 3 para tres variables de la estructura *Load*: la carga promedio del CPU (*Load Average*), la memoria física disponible (*Free Memory*) y el tiempo de CPU ocioso (*Idle Time*); el valor predicho de estas variables se almacena en la estructura *PredictedLoad*. El objeto *Predictor* también maneja los historiales para cada una de las tres variables y provee del método *update* encargado de actualizar tales historiales. El tamaño de los historiales se puede modificar a tiempo de compilación, y actualmente éstos almacenan hasta 10 observaciones.

Cada vez que se activa el *thread* del *SumaMonitor* para leer el valor de las variables de desempeño, también invoca el método *update* para actualizar los historiales y modifica los campos de la estructura *PredictedLoad*, aplicando el método de predicción predefinido para cada variable.

Aparte de los *threads* y objetos mencionados, la clase *SumaMonitor* ofrece métodos CORBA, que serán utilizados principalmente por el Planificador para acceder al contenido de la estructura *PredictedLoad* al momento del *Scheduling*; estos métodos también permiten modificar algunos parámetros, tales como el tiempo entre activaciones del *thread*, la cantidad de estadísticas deseadas o los métodos de predicción para cada una de las variables. Noten que las predicciones se hacen en el Agente de Ejecución cada vez que se despierta el *thread* y no cada vez que el *Scheduler* necesita una plataforma de ejecución. Esto reduce el *overhead* atribuido al metasistema al momento de colocar una aplicación en una plataforma de ejecución.

3 Métodos de Predicción de Carga

Los métodos de predicción que hasta ahora se han incorporado a SUMA son básicamente: métodos basados en la media y modelos de series de tiempo.

Entre los métodos más sencillos y de baja complejidad computacional, lo cual es importante para no degradar el desempeño de los Agentes de Ejecución (donde regularmente se están haciendo las predicciones), se encuentran los **métodos basados en la media**; éstos se describen en 3.1. Otras herramientas que dan soporte a planificadores de metasistemas, tales como el *Network Weather Service*, también hacen uso extensivo de estos métodos.

Los modelos de series de tiempo se incorporan a SUMA porque ya se han utilizados con buenos resultados para predecir la carga dinámica en *hosts* ([15],[7]) y en redes de interconexión (([15], [2]). En [15] se prueba únicamente con modelos autoregresivos (AR), variando sus parámetros, sin dar mayores detalles de cómo se obtienen estos modelos. Nosotros, sin embargo, decidimos estudiar primero la carga de trabajo típica de un conjunto de plataformas del metasistema, y con base en el comportamiento de la carga, tratamos de ajustar el mejor o los mejores modelos de la familia Box-Jenkins (AR, MA, ARMA, ARIMA). Estos modelos se incorporan después a SUMA. En [7] se explora la utilidad de modelos de series de tiempo para predecir la carga del CPU, concluyendo que los mejores modelos son los AR de alto orden; en el trabajo que presentamos en este artículo, los mejores resultados se obtuvieron con modelos autoregresivos integrados, también de orden alto.

Algunos de los modelos ajustados también se utilizaron con éxito para predecir la disponibilidad de la memoria. En la búsqueda bibliográfica realizada no se encontró ningún trabajo donde se emplearan los modelos con este fin. En la subsección 3.2 se describen brevemente los diferentes tipos de modelos Box-Jenkins. El lector interesado en más detalles puede consultar [4]. En la sección 4 se explica cómo se realizó el ajuste de estos modelos.

3.1 Modelos basados en la Media

3.1.1 Media(h)

Este método se basa en el promedio aritmético de los últimos h valores de la serie según la fórmula 1. Este método le da igual peso a todas las medidas.

$$pred(t) = \frac{1}{h} \sum_{i=1}^{h} valor(t - i + 1) \tag{1}$$

Donde $valor(k)$, es el valor real de la secuencia siendo monitoreada en el instante de tiempo k.

3.1.2 El Predictor Último

Este método es un caso particular del método anterior para $h = 1$. Puede ser un muy útil para predecir secuencias con baja variabilidad. Un trabajo reciente de Harchol-Balter y Downey ([10]) indica que este método es un predictor útil para recursos relacionados al CPU.

3.1.3 Media Adaptativa

El valor de h puede ser difícil de determinar a priori para cada recurso. Este método es una extensión dinámica de la media, donde el tamaño del historial, h, cambia dinámicante dependiendo si el error cuadrático medio mejoró o empeoró en la última y penúltima predicción. La fórmula predictiva se muestra a continuación:

$$pred(t) = \begin{cases} Media(h_{t-1} + mod) & \text{si } error(t) < error(t-1) \\ Media(h_{t-1} - mod) & \text{si } error(t) > error(t-1) \\ Media(h_{t-1}) & \text{si } error(t) = error(t-1) \end{cases} \tag{2}$$

Donde $mod = h_{t-1} - h_{t-2}$, y h_t es el tamaño del historial en el tiempo t. El error viene dado por la ecuación 3 del error cuadrático medio:

$$error(t) = \big(valor(t) - pred(t-1)\big)^2 \tag{3}$$

Donde $valor(t)$, es el valor real de la secuencia siendo monitoreada en el tiempo t y $pred(t-1)$ es el valor del pronóstico realizado en el tiempo $t - 1$.

3.1.4 Media Ponderada(h)

Este método es una modificación de Media(h), añadiéndole pesos a los históricos en orden creciente. Este método le da mayor peso a las medidas más recientes y menor peso a las últimas medidas del historial.

$$pred(t) = \frac{1}{sum_{j=1}^{h} j} \sum_{i=1}^{h} (h - i + 1) valor(t - i + 1) \tag{4}$$

3.1.5 Media Ponderada Adaptativa

En este método el historial se va adaptando dinámicamente al tamaño que arroje el menor error cuadrático medio. La forma de este método es similar a la de la media adaptativa:

$$pred(t) = \begin{cases} \text{Media Ponderada}(h_{t-1} + mod) & \text{si } error(t) < error(t-1) \\ \text{Media Ponderada}(h_{t-1} - mod) & \text{si } error(t) > error(t-1) \\ \text{Media Ponderada}(h_{t-1}) & \text{si } error(t) = error(t-1) \end{cases} \tag{5}$$

Donde $mod = h_{t-1} - h_{t-2}$, y h_t es el tamaño del historial en el tiempo t.

3.2 Modelos de Series Temporales Lineales

La idea principal que fundamenta el uso de modelos de series temporales en la predicción, es tratar una secuencia de medidas periódicas de la variable, (z_t), como la realización de un proceso estocástico que puede ser modelado como una fuente de ruido blanco (a_t) manipulado por un filtro lineal. Después de aplicar el filtro, la secuencia resultante, z_t, parcialmente predecible, exhibe una media μ y varianza σ^2. Los coeficientes del filtro lineal se pueden estimar utilizando observaciones pasadas de la secuencia ([7]). Los modelos de series temporales evaluados se describen brevemente a continuación.

3.2.1 *Modelos Autoregresivos (AR(p))*

La clase de modelos AR(p) son de la forma:

$$z(t) = \frac{1}{\phi(B)} a_t + \mu \tag{6}$$

Donde $\phi(B)$ tiene p coeficientes. Intuitivamente la salida de este modelo es una suma ϕ-ponderada de los p valores previos medidos.

3.2.2 *Modelos de Promedios Móviles (MA(q))*

La clase de modelos MA(q) son de la forma

$$z(t) = \theta(B) a_t \tag{7}$$

Donde $\theta(B)$ tiene q coeficientes. Intuitivamente, el valor de salida es la suma θ-ponderada de los q errores previos calculados a partir de la predicción.

3.2.3 *Modelos ARMA(p, q)*

La clase de modelos ARMA(p, q) de Box y Jenkins son combinaciones de los dos conceptos anteriores. Estos modelos son de la forma:

$$z(t) = \frac{\theta(B)}{\phi(B)} a_t + \mu \tag{8}$$

Donde $\phi(B)$ tiene p coeficientes y $\theta(B)$ tiene q coeficientes. Intuitivamente, la salida es la suma ϕ-ponderada de los p valores medidos previamente y θ-ponderada de los q errores previos.

3.2.4 *Modelos ARIMA(p, d, q)*

La clase de modelos ARIMA(p, d, q) (modelos autoregresivos integrados de promedio móvil) son de la forma:

$$z(t) = \frac{\theta(B)}{\phi(B)(1-B)^d} a_t + \mu \tag{9}$$

Para $d = 1, 2, ...$ Intuitivamente, el componente $(1-B)^d$ genera una d-integración de la salida de un modelo ARMA(p, q). Este filtro permite modelar secuencias no-estacionarias. Los modelos ARIMA(p, d, q) se ajustan diferenciando la serie d veces y ajustando a la secuencia resultante un modelo ARMA(p, q).

4 Ajuste de los Modelos de Series de Tiempo

Antes de incorporar a SUMA los modelos de series de tiempo, se decidió estudiar la carga de trabajo en las plataformas de ejecución que utiliza actualmente el metasistema, de forma tal de ajustar los mejores modelos para las trazas obtenidas directamente en estas plataformas. En las próximas subsecciones se explica el proceso de obtención de trazas y la metodología para ajustar los modelos.

4.1 Medidas para obtener las trazas

Las trazas se tomaron de un subconjunto de máquinas que forman parte de la gran máquina virtual SUMA. Se trata de un grupo de máquinas del Laboratorio Docente de Computación (LDC) de la Universidad Simón Bolívar. Las máquinas del laboratorio son empleadas usualmente por estudiantes de cursos de Ingeniería de la Computación. Típicos cursos son: Sistemas de Operación, Redes de Computadoras, Computación Gráfica, Lenguajes de Programación y Sistemas de Programas. La carga en las máquinas no siempre es la misma, suele ser más intensa a partir de la semana 3, en un trimestre de 12 semanas, que es cuando los estudiantes comienzan a trabajar en sus asignaciones.

Las trazas se tomaron en 30 estaciones de trabajo. La herramienta utilizada fue el propio *SumaMonitor* con pequeñas modificaciones para almacenar las trazas. De las 30 máquinas, 20 tienen un procesador Pentium III a 800 MHz, con 256 MB de memoria principal y tienen como sistema operativo Linux Mandrake 7.2. Las 10 máquinas restantes tienen un procesador Ultra Sparc de 144 MHz con 128 MB de memoria principal; tienen como sistema operativo Solaris 8.

Las mediciones se hicieron durante un período de tres semanas de carga intensa (semanas 7, 8 y 9 del trimestre Marzo-Julio). Cada día de la semana entre lunes y viernes, se tomaban medidas entre las 8:00 am

y las 7:00 pm, horas en las cuales el laboratorio presenta una carga importante. Las medidas se tomaron cada minuto, pero la idea es ajustar modelos para intervalos de 1 y 5 minutos. La escogencia de este intervalo de tiempo está aún en la fase de entonación. Estos valores iniciales se inspiraron de los resultados de [13], [7] y [6]. En [13] se encontró que una ventana de 5 minutos tenía una alta correlación, i.e., era muy probable que la conducta observada en los próximos 5 minutos, fuera muy cercana al comportamiento de los 5 minutos pasados. Adicionalmente, Dinda y O'Hallaron ([7], [6]), encontraron que el comportamiento del CPU es estable por un período de tiempo de entre 150 y 450 segundos; en [7], particularmente, se realizan predicciones en intervalos de 1 minuto.

Las variables registradas en la traza fueron la carga promedio del CPU (*Load Average)* y la memoria física disponible (*Free Memory*). Posteriormente se estudiará el comportamiento de otras variables registradas por el *SumaMonitor*. Con respecto a las trazas obtenidas se pudo observar lo siguiente:

- En general la variable *Load Average* presenta una variabilidad alta. Se pueden observar *outliers* intermitentes de corta duración. La media está muy cercana a cero con una desviación estándar alta, y valores máximos de hasta un orden de magnitud superioriores a la media. La figura 2 muestra una traza de 600 minutos donde se puede apreciar el comportamiento típico de esta variable; la media de la traza es 0.05 y la desviación es de 0.13. La variable *Free Memory* mostró una menor variabilidad, con una media distinta de cero y desviación estándar menor que la media. Su comportamiento se puede observar en la figura 5.

- Dado que no se observaron diferencias importantes en las trazas de los distintos días en las distintas semanas, se decide realizar el ajuste sobre una traza que resulta de promediar trazas similares tomadas en distintos días.

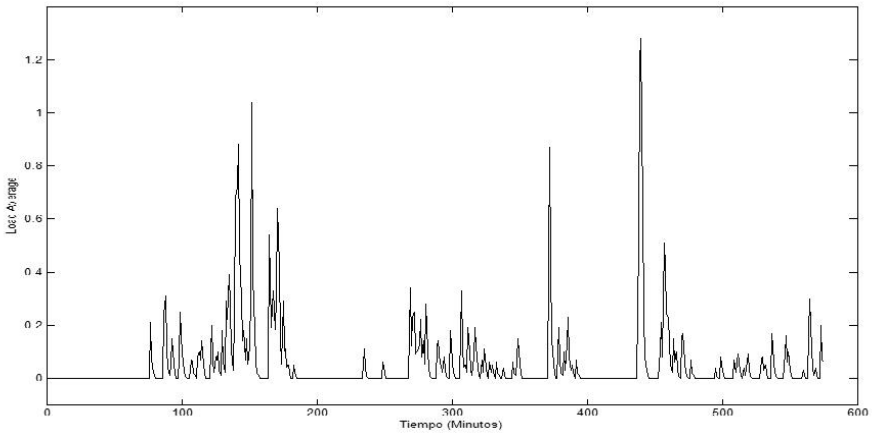

Figura 2: Traza de la variable *Load Average*

4.2 Ajuste

El ajuste de los modelos de series de tiempo se realizó usando los pasos de la metodología de Box, Jenkins y Reinsel [4], que se enuncian a continuación: Identificación de modelos tentativos, Estimación de los parámetros y Diagnóstico del modelo. Es importante destacar que, por razones de tiempo, el ajuste de los modelos se realizó únicamente sobre la variable *Load Average*, aunque en su fase predictiva los modelos también se probaron con la variable *Free Memory*.

4.2.1 Identificación de modelos tentativos

Esta primera actividad busca identificar una subclase adecuada de modelos (AR, MA, ARMA, ARIMA) que pueda ser usada para representar una serie temporal. En la búsqueda del modelo más adecuado se siguen los siguientes pasos:

1. Diferenciar la serie z_t cuantas veces sea necesario para generar estacionaridad en la secuencia. A fin de poder estimar las características transversales (medias, varianzas, etc.) de la serie, a partir de su trayectoria, debemos suponer que las propiedades son transversales a lo largo del tiempo, es decir que son procesos estacionarios (no hay componentes estacionales ni de tendencia). Desafortunadamente, la gran mayoría de las series que observamos no son estacionarias y su media varía con el tiempo. No

obstante, es frecuente lograr la estacionaridad al diferenciarla (considerar el proceso de los incrementos). Llamaremos proceso primera diferencia del proceso z_t a un nuevo proceso, w_t, obtenido mediante: $w_t = z_t - z_{t-1}$.

2. Identificar el proceso ARMA resultante

Las principales herramientas utilizadas en las dos actividades anteriores son la función de autocorrelación (acf) y la función de autocorrelación parcial estimadas (pacf) (ver detalles en [4]). La forma como los coeficientes de estas funciones decrecen a cero (en forma exponencial, sinusoidal, lentamente, etc.) nos permite tener una primera idea del modelo tentativo o nos indica la presencia de componentes estacionales o tendencias.

Al observar la acf y pacf de una de nuestras series temporales z_t (aquella que tenía observaciones cada minuto), notamos que la función de autocorrelación decrece con gran lentitud hacia cero, lo cual es una señal de falta de estacionaridad. La recomendación es entonces diferenciar para tratar de obtener una serie estacionaria. Se analizaron las acf y pacf de las primeras tres diferencias y la primera diferencia proporcionó los mejores resultados. Esto se puede observar en la rapidez con que decrece la acf hacia el valor cero. Al diferenciar una vez se obtuvo un proceso integrado (ARIMA) de orden 1 (parámetro d del modelo). La estimación del resto de los parámetros del modelo se explica en los próximos párrafos. Para lograr estacionaridad en la serie con observaciones cada 5 minutos, también fue necesario diferenciar la serie 1 vez.

4.2.2 Estimación de los parámetros

Una vez que se ha identificado como modelo tentativo un modelo ARIMA, todavía resta encontrar el valor de los parámetros p y q, y el valor de los coeficientes del modelo. Para los modelos ARIMA, la estimación se hace por máxima verosimilitud. Con el objeto de encontrar el modelo ARIMA más adecuado se utilizó la instrucción de **R** (*software* estadístico, http://www.r-project.org):

arima0(x, order=c(0,0,0),...)

Donde x es la serie que deseamos estimar y *order* el tipo de modelo que se desea ajustar. Entre los elementos que devuelve esta instrucción se encuentran: los coeficientes de las componentes AR y MA del modelo (son los coeficientes de los polinomios), los residuales, el valor del AIC correspondiente a la log-verosimilitud (proporcionado por Akaike [1]) y un indicador de convergencia; si este último es cero, el algoritmo de ajuste convergió sin problemas. Para estimar los valores p y q se invocó la función varias veces con combinaciones de estos parámetros, escogiendo finalmente los modelos con el menor valor de AIC. El proceso se aplicó para trazas de 1 y 5 minutos. Los mejores modelos fueron: el ARIMA$(8,1,0)$ y el ARIMA$(1,1,2)$, para las trazas de 1 minuto, y el ARIMA$(4,1,0)$ para las trazas de 5 minutos.

4.2.3 Diagnóstico del modelo

Para diagnosticar los modelos resultantes se utilizan los residuos del ajuste. Si el modelo es adecuado, se esperaría que los residuos se comportasen como un ruido blanco gaussiano. La presencia de autocorrelación o de falta de estacionaridad en los residuos implicaría que el modelo no es adecuado. En general, el análisis de residuos incluye:

- **Un gráfico de los residuos en el tiempo**. Este gráfico permite identificar valores atípicos, falta de homogeneidad en la varianza y presencia de estructuras en el tiempo.
- **Un análisis de la función de autocorrelación de los residuos**. Sean $\hat{\gamma}_k$ las autocorrelaciones estimadas para los residuos, si el modelo es adecuado, los $\hat{\gamma}_k$ deberían ser no correlacionados y aproximadamente normales con media cero. La presencia de autocorrelaciones grandes indica que el modelo puede ser inadecuado.
- **Pruebas de hipótesis sobre las autocorrelaciones**. En lugar de examinar los $\hat{\gamma}_k$ individualmente, es útil basar una prueba de diagnóstico en un conjunto de autocorrelaciones (típicamente entre 10 y 20). Se define el estadístico de Box-Pierce (también llamado *portmanteau*) como $Q = n\sum_{k=1}^{K}\hat{\gamma}_k^2$ donde K es el máximo número de retardos y n es el número de observaciones usadas para el cálculo de la verosimilitud. Si el modelo ajustado es apropiado, los valores en la gráfica deben estar por encima de un margen calculado según el número de parámetros del modelo.

En la figura 3 se pueden observar las gráficas del diagnóstico del modelo ARIMA$(8,1,0)$. Los gráficos que se muestran son: residuos vs. tiempo, funciones de autocorrelación y autocorrelación parcial para los residuos y un gráfico de los p-valores del test Box-Pierce. En la gráfica de los residuos en el tiempo se

puede apreciar una secuencia sin ninguna tendencia de crecimiento o decrecimiento fuerte, es decir, ninguna estructura o tendencia en el tiempo; se observan muy pocos valores atípicos y una varianza sufientemente homogénea. En las funciones de autocorrelación y autocorrelación parcial de los residuos se observan, en general, valores pequeños y dentro de los umbrales (líneas punteadas). Se observa, no obstante, en la pacf, valores muy por encima del umbral en los retardos 13 y 20, haciendo posible la existencia de periodicidades no consideradas por este modelo. Sin embargo, el periodograma y el periodograma acumulativo de los residuos confirmaron que no se estaba pasando por alto algún comportamiento estacional y que el modelo se ajustaba correctamente a la serie. En la última gráfica, los p-valores son claramente mayores al margen delimitado por la línea punteada, lo que indica que no se puede rechazar la hipótesis de que las autocorrelaciones estimadas son normales con media cero.

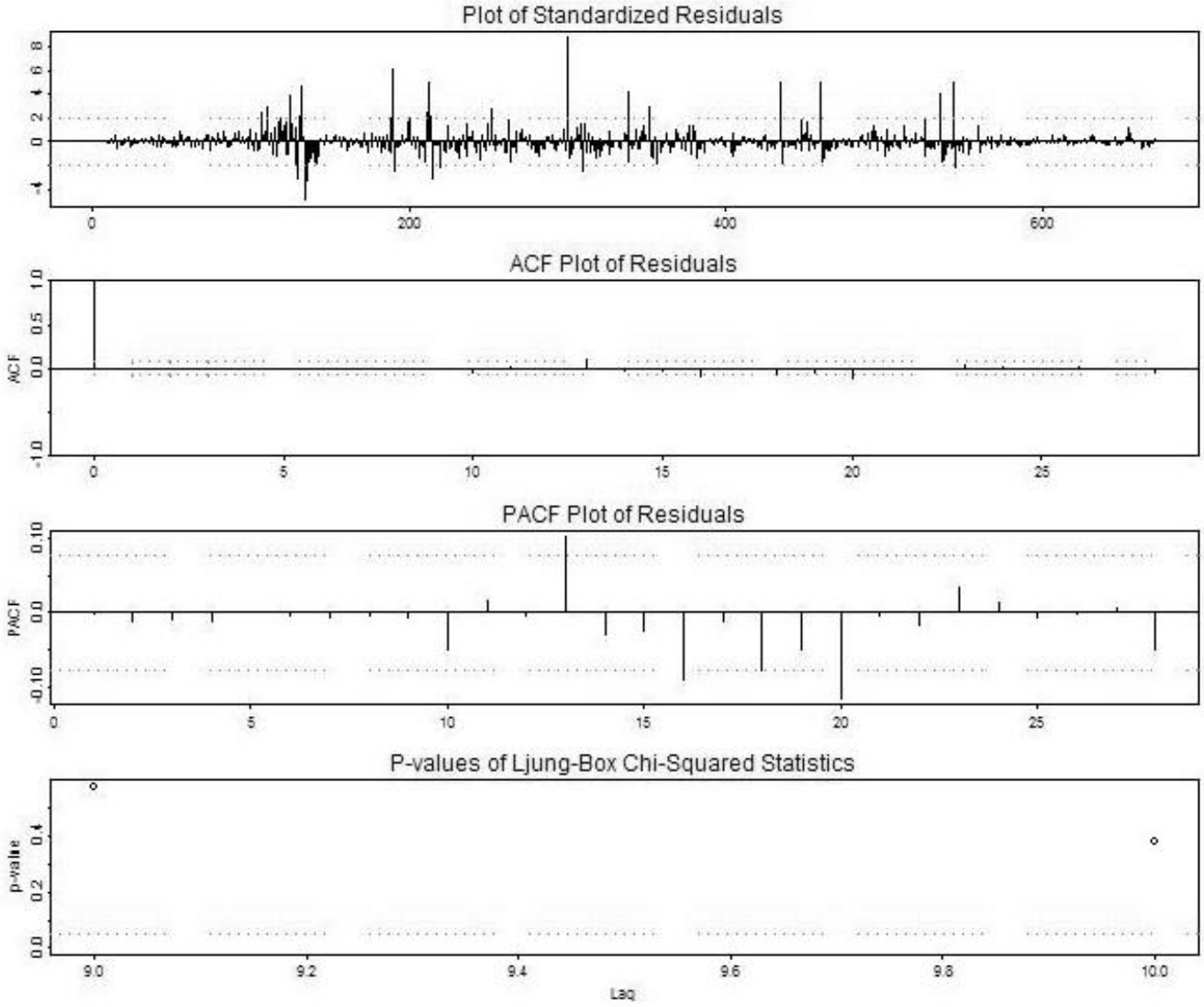

Figura 3: Diagnóstico del modelo ARIMA(8,1,0)

4.3 Implementación del modelo para la predicción

Después del ajuste, se construyó, para cada uno de los modelos ajustados, un predictor con los coeficientes resultantes. El predictor consiste en el modelo convertido en una ecuación, el valor a predecir y una cola que contiene los últimos $(p+d)$ valores de la secuencia. Cuando el valor de la última medida está disponible, se inserta en la cola y se evalúa el modelo ($O(p+d)$ operaciones) para producir un nuevo valor de pronóstico para la secuencia. A continuación se presenta la ecuación de predicción para el modelo ARIMA$(8,1,0)$:

$$z_t = 0.7914z_{t-1} - 0.0837z_{t-2} + 0.06906z_{t-3} + 0.04195z_{t-4} - 0.02581z_{t-5}$$

$$+0.10393z_{t-6} - 0.00381z_{t-7} - 0.03924z_{t-8} + 0.14622z_{t-9}$$

5 Experimentos

Todos los modelos de predicción descritos se implementaron en Java y se incluyeron en la clase *Predictor* dentro del paquete *SumaMonitor*. Se incorporaron todos los predictores a fin de evaluarlos con las dos variables seleccionadas. Aunque los resultados de un determinado predictor no sean del todo satisfactorios, se dejan en el *SumaMonitor* porque pudieran ser útiles con otras variables o en otros entornos de carga.

Para probar la efectividad de los predictores se toman medidas de las variables en el tiempo t, y se predice, con los distintos métodos, el valor de la variable en el tiempo $t+k$, donde k es 1 ó 5 minutos. Una vez que se tiene el valor real de la carga en el instante $t+k$, se calcula el error cuadrático medio (ecuación 3). Las medidas se tomaron en el LDC, durante tres días seguidos con la carga usual del laboratorio en semanas de mucho trabajo. Los Agentes de Ejecución junto con el *SumaMonitor* se levantaron en 6 máquinas: en tres máquinas se hacían predicciones para la variable *Load Average* y en tres máquinas para la variable *Free Memory*. Se hizo de esta forma para reducir el *overhead* introducido por el monitor, ya que para cada variable se estaban probando todos los métodos.

Las predicciones y los errores correspondientes a 1 y 5 minutos se almacenaron en históricos diferentes. La tabla 1 muestra los errores de los métodos predictivos para las variables *Load Average* y *Free Memory*. Los resultados de la variable *Free Memory* se deben multiplicar por $(1.0e^8)$.

	Load Average		*Free Memory*	
	1 minuto	5 minutos	1 minuto	5 minutos
Último (*Last*)	0.0117	0.0163	**0.2600**	**0.8853**
Media Adaptativa	0.128	0.0178	0.6981	1.4162
Media Ponderada Adaptativa	0.0129	**0.0141**	0.4770	1.0871
ARIMA(1, 1, 2)	0.0157	0.0243	0.3122	1.183
ARIMA(8, 1, 0)	**0.0103**	0.0146	0.3141	0.8914
ARIMA(4, 1, 0)	0.0120	0.0149	0.3230	0.9453

Tabla 1: Errores para cada uno de los métodos

Se puede observar, en todos los casos, que los errores son más pequeños cuando el pronóstico se hace cada minuto. Para la variable *Load Average*, el método predictivo que reportó el error más pequeño en intervalos de un minuto fue el modelo $ARIMA(8,1,0)$; el modelo $ARIMA(1,1,2)$ produce las predicciones con los errores más grandes. Espaciando las predicciones en intervalos de 5 minutos el mejor predictor resultó ser la media ponderada adaptativa, aunque las diferencias con los modelos $ARIMA(8,1,0)$ y $ARIMA(4,1,0)$ no son muy significativas; recordemos que el modelo ARIMA(4,1,0) fue el ajustado para la traza de 5 minutos.

La figura 4 muestra la carga real y las predicciones, cada minuto, de los métodos que reportaron el error más bajo (izquierda) y más alto (derecha); la traza abarca un intervalo de tiempo de 100 minutos. La línea sólida representa el estado real del sistema y la línea punteada, el resultado del método de predicción. El modelo $ARIMA(1,1,2)$ predice, en general, los *outliers* intermitentes con una mayor magnitud que los que aparecen en la carga real. El $ARIMA(8,1,0)$ es más conservador en los *outliers* de mayor magnitud, pero en los picos más pequeños suele ser bastante más acertado.

Tanto para un minuto, como para cinco minutos, el mejor predictor para la variable *Free Memory* es el *último*. El comportamiento de esta variable es distinto al de la variable *Load Average*; suele mantenerse más tiempo dentro de ciertos rangos (en su gráfica se observan menos picos) lo cual la hace más predecible. Las predicciones de los modelos ARIMA son superiores a las predicciones de la Media Adaptiva, y en algunos casos (ARIMA(8,1,0) en 5 minutos), comparables a los resultados del predictor *último* a pesar de que este tipo de modelos no fue ajustado para las trazas de esta variable; pudieran obtenerse resultados distintos si se repite con la variable *Free Memory* el procedimiento utilizado para la variable *Load Average*.

La figura 5, muestra el mejor y el peor predictor para la variable *Free Memory* en intervalos de 1 minuto. La media adaptativa no parece comportarse muy bien en los cambios bruscos de la variable. La gráfica del predictor *último* es idéntica al valor real, salvo que desplazada por una unidad de tiempo. Este predictor es de mucha mayor utilidad en este caso porque el valor de la variable es más estable en períodos más largos de tiempo (varios minutos).

El próximo paso es comprobar si se obtienen resultados similares, ajustando modelos autoregresivos a trazas de la variable *Free Memory*. Adicionalmente, en otros entornos de cargas, donde el comportamiento de esta variable fuese menos estable, se pudieran tratar de ajustar modelos de función de transferencia. En este tipo de modelos, si se logran determinar las relaciones dinámicas entre dos series de tiempo X_t y

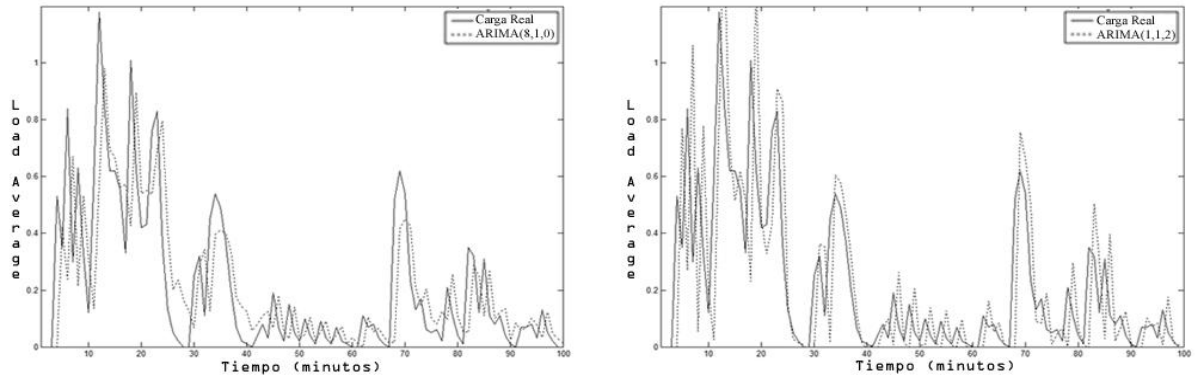

Figura 4: Predicciones sobre la variable *Load Average*

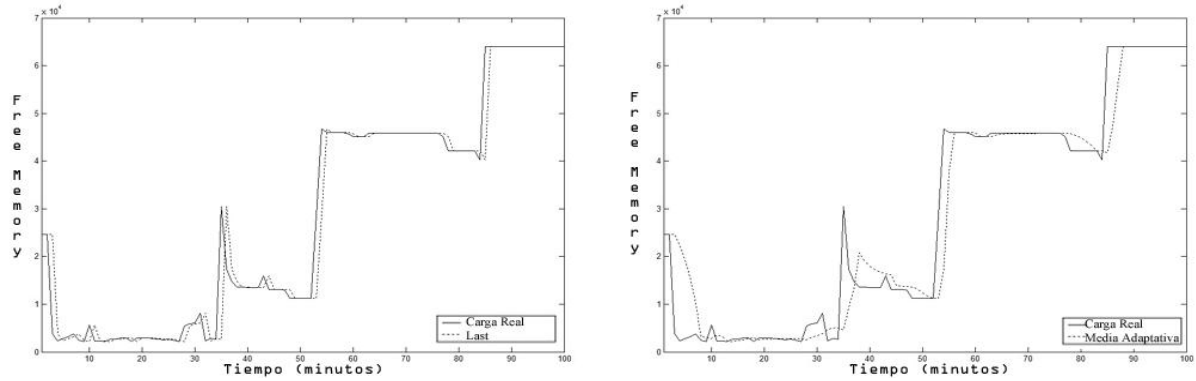

Figura 5: Predicciones sobre la variable *Free Memory*

Y_t, los valores pasados de ambas series se pueden utilizar para predecir el valor de Y_t; en muchos casos este enfoque contribuye a aumentar el poder predictivo. En nuestro problema, se pudieran agregar como variables exógenas, relacionadas con la cantidad de memoria libre, variables como el número total de procesos en el sistema o la cantidad de paginación.

6 Conclusiones y Trabajo Futuro

Del trabajo expuesto en este artículo se concluye:

- Los modelos ARIMA resultan de mucha utilidad para predecir la carga del CPU en intervalos cortos de tiempo (1 minuto). Que el mejor modelo obtenido tenga en la parte autoregresiva un polinomio de alto grado (grado 8) concuerda con los resultados presentados por Dinda y O'Hallaron [7]. En intervalos de 5 minutos, la media ponderada adaptativa y los modelos ARIMA(8,1,0) y ARIMA(4,1,0) ofrecen los mejores resultados. Definitivamente, para un tipo de variable como *Load Average* el valor de las observaciones pasadas tiene importancia en la efectividad del predictor.

- En el caso de la memoria, al mostrar la variable menos cambios bruscos, un predictor sencillo como el *último* parece suficiente. El peor comportamiento de este predictor es precisamente cuando ocurren estos cambios.

- De los errores reportados por los modelos, se infiere que la carga de trabajo estudiada es más estable en períodos cortos, más cercanos a 1 minuto que a 5 minutos. Este período de tiempo hay que estudiarlo con mayor mayor cuidado así como explorar qué otra información puede ser útil para el Planificador junto con la predicción. Si la aplicación va a ocupar la plataforma de ejecución por una gran cantidad de tiempo (varios minutos o incluso horas), es probable que sean de gran utilidad estadísticos como la media y la varianza, que puedan dar una mejor idea de cómo se comportará la carga externa mientras se ejecute la aplicación.

- Todos los métodos fueron exitosamente implementados en Java e incorporados a SUMA. De momento se utilizarán los métodos que ofrecieron los mejores resultados para las variables *Load Average* y *Free*

Memory. El resto de los métodos se implementa porque pudieran ser útiles para otros entornos de carga u otras variables. Si el entorno de carga cambia drásticamente, la idea es aplicar los pasos descritos en este artículo para ajustar nuevos modelos de series de tiempo. Otra meta a futuro es que dinámicamente se escoga el método de predicción que más se ajuste al entorno de carga de trabajo.

Las actividades que siguen pretenden continuar explorando métodos de predicción para las variables aquí estudiadas y otras variables que describan, no sólo la carga en las plataformas de ejecución, sino en las redes de interconexión. En el caso de la carga del procesador, cuyo comportamiento es más complejo, se pudiera explorar el uso de métodos de predicción basados en redes neurales.

7 Agradecimientos

Los autores quisieran agradecer a la Prof. Lelys Bravo del Centro de Estadística y Matemática su valiosa colaboración en el ajuste de los modelos de series de tiempo.

Referencias

[1] H. Akaike: A New Look at the Statistical Model Identification. IEEE Trans. Automatic Control, AC-19, 716-723, 1974.

[2] S. Basu, A. Mukherjee y S. Kilvansky: Time Series Models for Internet Traffic. Reporte Técnico GIT-CC-95-27, Georgia Institute of Technology, 1996.

[3] F. Berman y R. Wolsky: The AppLeS Project: a Status Report. Proceedings of NEC Symposium on Metacomputing, 1997.

[4] G. Box, G Jenkins, G Reinsel. Time Series Analysis. Forecasting and Control. Third Edition, 1994.

[5] H. Casanova y J. Dongarra: Netsolve: A Network Server for Solving Computational Science Problems. Reporte Técnico, Universidad de Tenessee, Noviembre, 1995.

[6] P. A. Dinda: The Statistical Properties of Host Load. Proc. of 4th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers. (LCR'98), vol. 1511 de Lecture Notes in Computer Science, Springer-Verlag, pp. 319-334, 1998.

[7] P. Dinda y D. O'Hallaron: An Evaluation of Linear Models for Host Load Prediction. Reporte Técnico, CMU-CS-TR-98-148, Carnegie Mellon University, Noviembre 1998.

[8] I. Foster y C. Kesselman: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputing Applications, 11(2): 115-128, 1997.

[9] A. S. Grimshaw, W. A. Wulf, y the Legion Team: The Legion Vision of a Worldwide Virtual Computer. Communications of the ACM, 40(1), Enero, 1997.

[10] M. Harchol-Balter y A. Downey: Exploiting Process Lifetime Distributions for Dynamic Load Balancing. Proceedings of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Science, 1996.

[11] E. Hernández, Y. Cardinale, C. Figueira y A. Teruel: SUMA: A Scientific Metacomputer. Proceedings of PARCO'99, Amsterdam, The Netherlands, 1999.

[12] G. Parra: Mejoras a los Servicios de SUMA que Soportan la Evaluación del Rendimiento. Proyecto de Grado de la Universidad Simón Bolívar, Noviembre 2001.

[13] J. M. Schopf: Performance Prediction and Scheduling for Parallel Applications on Multi-Users Clusters. Tesis de Doctorado, Universidad de California, San Diego, 1998.

[14] R. Wolski, N. Spring y C. Peterson: Implementing a Performance Forecasting System for Metacomputing: The Network Weather Service. UCSD Reporte Técnico TR-CS97-540, Mayo 20, 1997.

[15] R. Wolski: Dinamically Forecasting Network Performance Using the Network Weather Service. UCSD Reporte Técnico TR-CS96494 y publicado en Cluster Computing: Network, Software Tools and Applications, Universidad de California en San Diego, Enero, 1998.