UNIVERSIDAD AUTONOMA DE MADRID

excelencia Campus Internacional UAM CSIC

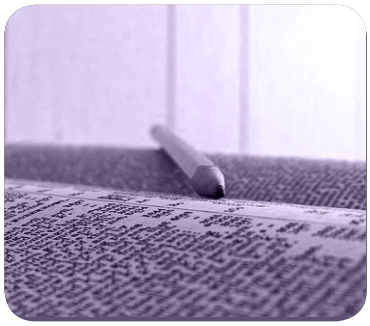
MÁSTERES de la UAM

Escuela Politécnica Superior / 16-17

Ingeniería de Telecomunicación

Desarrollo de una herramienta de tratamiento estadístico de distribuciones α-estables para su aplicación al análisis de tráfico de redes IP

Marta Martínez Redondo



UAM EDICIONES

UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Desarrollo de una herramienta de tratamiento estadístico de distribuciones α-estables para su aplicación al análisis de tráfico de redes IP

Máster Universitario en
Ingeniería de Telecomunicación

Marta Martínez Redondo

Tutor: Luis de Pedro Sánchez

Ponente: Jorge E. López de Vergara Méndez

Septiembre 2017

Desarrollo de una herramienta de tratamiento estadístico de distribuciones α-estables para su aplicación al análisis de tráfico de redes IP

AUTOR: Marta Martínez Redondo
TUTOR: Luis de Pedro Sánchez
PONENTE: Jorge E. López de Vergara Méndez

High Performance Computing and Networking

Dpto. de Tecnología Electrónica y de las Comunicaciones

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Septiembre 2017

AGRADECIMIENTOS

En primer lugar, me gustaría agradecerles a Luis y a Jorge, mi tutor y mi ponente, o como yo siempre me he referido a ellos, *mis tutores*, por su infinita paciencia y su ayuda. Gracias por la oportunidad de introducirme en la para mí tan desconocida investigación, y por los consejos y la disponibilidad que siempre habéis tenido para resolver mis problemas. Una gran enseñanza que me llevo de haber trabajado con vosotros, y que jamás olvidaré, es que no se debe *intentar pintar un león sin observar antes un gato*.

Allá por el año 2010, una chica de diecisiete años entraba en la Escuela Politécnica Superior con el miedo de no sobrevivir a lo que muchos decían que era imposible: el primer año de una ingeniería. Este camino ha tenido alguna que otra curva de las que marean, y los miedos, desde luego, han ido cambiando. Hoy, recojo en estas líneas los frutos de algo que sí ha perdurado desde entonces: el esfuerzo y el apoyo.

Respecto al apoyo, debo dar las gracias por el que me han brindado mis compañeros. Todos y cada uno de vosotros habéis logrado que este camino fuera más llevadero. Como ya dije en su día, gracias por hacer que el día de mañana pueda mirar orgullosa la orla y decir: "Sí, la universidad fue la mejor época de mi vida".

Muchas gracias a los de siempre, a Laura, Álvaro y Víctor, por ofrecerme la mano en mis tropiezos y por escuchar mis agobios con cosas que fingíais entender. El valor de una amistad verdadera como la vuestra, es sin duda el mayor apoyo y estimulante.

Gracias a Juan, por ser mi fuente de energía y no dejar de creer en mí. No podía haber elegido mejor compañero de vida.

Respecto al esfuerzo, debo dar las gracias de una manera muy especial a mi familia. Dejando a un lado que, por supuesto, el apoyo recibido por su parte ha sido el más incondicional de todos, a ellos les debo algo aún más importante: ser mi gran ejemplo a seguir. La niña que no quería ser lo mismo que papá, mirad dónde ha acabado. Gracias por enseñarme a no rendirme, y a que hace falta esfuerzo y dedicación para conseguir lo que uno quiere. Como siempre decimos, la suerte no va con los Martínez Redondo, pero a mí teneros ya me hace la más afortunada.

A todos, incluido a ti, lector, por invertir parte de tu tiempo en leer estas líneas, os dedico este trabajo.

RESUMEN

El tráfico de las redes IP actuales continúa creciendo de manera exponencial. Por ello conseguir un modelo del mismo que permita estimar y anticipar su comportamiento es una tarea cada vez más compleja, siendo un amplio campo de investigación en la gestión de red.

Últimamente, se ha planteado la utilización de modelos estadísticos basados en distribuciones α-estables que, como generalizaciones del caso gaussiano, están dando resultados prometedores en cuanto a la modelización de diferentes situaciones de tráfico en la red y su caracterización. Sin embargo, los requerimientos de cálculo que impone la estimación de los parámetros que definen estas distribuciones hacen que su utilidad como herramienta de análisis en tiempo real sea muy limitada en la actualidad.

En este Trabajo Fin de Máster se propone desarrollar una herramienta de tratamiento estadístico de distribuciones α-estables para su aplicación al análisis de tráfico de redes IP.

Para ello, será necesario analizar los algoritmos existentes para la estimación rápida de los parámetros de dichas distribuciones, procediendo a desarrollar uno de ellos en lenguaje C, y posteriormente, a su aceleración paralela.

Tras la fase de desarrollo en C y su implantación paralela, se aplica la herramienta en una serie de casos prácticos para extraer conclusiones sobre su uso en entornos reales de administración de redes y sistemas.

PALABRAS CLAVE

Modelado de tráfico de red, tiempo de establecimiento de la conexión TCP, distribuciones α-estables, estimación rápida de los parámetros, algoritmos paralelos.

ABSTRACT

Current IP network traffic continues to grow exponentially. This is why obtaining a model which allows to estimate and anticipate its behavior is an increasingly complex task, being a large field of investigation in network management.

Lately, the use of statistical models based on α-stable distributions is being proposed and, as generalizations of the gaussian case, they are giving promising results in terms of the modelling of different taffic situations in the network and its characterization. However, the calculus requirements that are imposed by the parameters estimation that define these distributions make their utility as a real time analysis tool very limited nowadays.

In this Master's Thesis a statistical treatment tool of α-stable distributions for IP network traffic is being proposed. For this, it will be necessary to analyze existing algorythms for fast parameter estimation of these distributions, proceeding to develop one of them in C language, and later, its parallel acceleration.

After the development phase in C and its parallel implementation, the tool is applied in a series of practical cases to extract conclusions about its use in real networks and systems management environments.

KEYWORDS

Network traffic modeling, TCP connection establishment time, α-estable distributions, fast parameter estimation, parallel algorithms.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1. Introducción

1.1. Motivación

La creciente demanda de redes de comunicaciones capaces de proporcionar cada vez un mayor número de servicios provoca que el tráfico transportado en la redes IP (Internet Protocol) actuales continúe creciendo de manera exponencial. Como consecuencia, conseguir una modelización del mismo que permita estimar y anticipar su comportamiento para logar un dimensionamiento óptimo de las infraestructuras presentes y futuras, se ha convertido en una tarea cada vez más compleja, siendo una amplia área de investigación en el campo de la gestión de red.

En los últimos años se ha estado planteando la utilización de modelos estadísticos basados en distribuciones α-estables, cuyas propiedades han permitido obtener resultados prometedores en cuanto al modelado de diferentes situaciones de tráfico en la red y su caracterización a partir de la estimación de los parámetros de dicho modelo estadístico. Además de caracterizar el tráfico y permitir predecir de forma precisa su comportamiento, es importante poder realizar la estimación de estos parámetros con la suficiente velocidad. La posibilidad de tener que enfrentarse a anomalías o intrusiones que intentan violar la privacidad de datos sensibles o causar interrupciones de servicio, obligan a un sistema de monitorización de red a detectarlos rápidamente y así poder resolver el problema lo antes posible.

Desafortunadamente, el coste computacional de los algoritmos de integración numérica que requiere la estimación de los parámetros de una distribución α-estable ha ocasionado que su uso como herramienta de análisis de tráfico que trabaje a tiempo real sea limitado en la actualidad.

Por otra parte, la computación paralela desde hace unos años, está experimentando un importante auge debido a las grandes posibilidades que ofrece su uso en la aceleración de algoritmos cuya estructura lo permite, y así abordar problemas que resultan ser costosos computacionalmente. Conforme a esto, gracias a la paralelización de la obtención de distribuciones α-estables, se prevé conseguir mejorar visiblemente el rendimiento, posibilitando su uso en sistemas predictivos en tiempo real.

1.2. Objetivos

El objetivo que se persigue con la elaboración del presente Trabajo Fin de Máster es desarrollar una herramienta de tratamiento estadístico que resuelva el problema de tiempo de cálculo en la modelización de tráfico de redes IP utilizando distribuciones α-estables. Se busca que de su aplicación en una serie de casos prácticos puedan extraerse conclusiones sobre su uso en entornos reales de administración de redes y sistemas. Para alcanzar este objetivo global, se van a considerar los siguientes objetivos parciales:

- Aprendizaje de la teoría y los conceptos subyacentes a la aplicación de las distribuciones α-estables en el modelado de tráfico de red.

- Análisis de los algoritmos existentes para la estimación de los parámetros de las citadas distribuciones, así como el desarrollo en lenguaje C de una herramienta que permita realizar dicha estimación de forma precisa y rápida, optando para ello por su aceleración mediante programación paralela.

- Evaluación de la herramienta desarrollada aplicándola a casos de medidas reales de red que permita probar su correcta adecuación y la extracción de conclusiones sobre su posible utilidad para la caracterización del tráfico en internet en entornos reales.

1.3. Fases de realización

Para facilitar el alcance de los objetivos planteados, este trabajo se realizará siguiendo una serie de fases estructuradas:

- Fase de investigación. En primer lugar, será necesario realizar un estudio y revisión del estado del arte de las tecnologías implicadas en el desarrollo del trabajo, como son los modelos estadísticos aplicados a tráfico de red, los algoritmos ya existentes para la estimación de los parámetros de las distribuciones α-estables que se adecuen a los requerimientos del proyecto, y las diferentes técnicas de paralelización.

- Fase de análisis previo de la magnitud a modelar. Sabiendo que existen referencias en la literatura que presentan interesantes resultados al aplicar un modelo α-estable en la caracterización del tráfico de red, se propone llevar a cabo un primer análisis que permita averiguar si es posible modelar una magnitud concreta del tráfico de red, como es el tiempo de respuesta del servidor en el establecimiento de la conexión TCP (Transport Control Protocol), con un modelo estadístico α-estable.

- Fase de desarrollo: Tras la selección del algoritmo pertinente que se adecue al problema, se procederá al desarrollo del código necesario para su aplicación, así como a su implementación en paralelo con el objetivo de conseguir una reducción

temporal en la estimación y lograr un correcto funcionamiento equivalente a la versión sin paralelizar.

- Fase de validación: Se van a llevar a cabo una serie de comprobaciones para validar la aceleración conseguida tras paralelizar el algoritmo. Además, se aplicará la herramienta a un conjunto de medidas reales de tráfico de red y se evaluarán los parámetros de la distribución α-estable para poder extraer conclusiones de su utilización como posible herramienta implantada en producción para la prevención de ataques en clientes reales.

1.4. Organización de la memoria

El presente documento sigue la siguiente estructura:

- **Capítulo 2:** Se describe el estado del arte referente al modelado de tráfico de red, poniendo de manifiesto las carencias observadas en los modelos tradicionales, las cuales son subsanadas por las distribuciones α-estables. Se presentan también las diferentes técnicas de paralelización que puedan permitir acelerar la estimación de sus parámetros.
- **Capítulo 3:** En este capítulo de plantea el problema a tratar, concretamente, la magnitud de tráfico que se busca modelar.
- **Capítulo 4:** Se analiza un algoritmo serie ya existente, y se desarrolla en lenguaje C como primera aproximación para su optimización.
- **Capítulo 5:** En este capítulo se procede a la aceleración paralela del algoritmo.
- **Capítulo 6:** Se aplica la herramienta desarrollada al análisis de tráfico.
- **Capítulo 7:** Finalmente, se resumen los resultados y las conclusiones extraídas a lo largo de este Trabajo Fin de Máster, y se exponen las posibles mejoras y propuestas de trabajo futuro que han surgido a partir de su realización.
- **Glosario.**
- **Referencias.**
- **Anexo:** Se incluyen las gráficas obtenidas en un estudio del capítulo 6.

Capítulo 2. Estado del arte

2.1. Introducción

En este capítulo se recogen los estudios previos necesarios para la realización del presente Trabajo Fin de Máster sobre el modelado de tráfico más empleado en la literatura y la situación actual en la que se encuentran los diferentes estándares de programación paralela. En concreto, se va a exponer una visión general sobre las capacidades de los modelos de tráfico para su adaptación a datos reales de tráfico de red. Dado que en la literatura hay evidencias de que el tráfico de red presenta alta variabilidad y dependencia a largo plazo, será importante contar con un modelo estadístico que se adapte lo más fielmente posible a estas propiedades, y por ello, partiremos de los prometedores resultados que se están alcanzando con las distribuciones α-estables. Sin embargo, aunque se han descrito varios métodos numéricos en la literatura tanto para calcular las densidades α-estables como para estimar sus parámetros, tienden a ser engorrosos y lentos de procesar, lo que motiva a explotar los beneficios de la paralelización.

2.2. Modelos de tráfico

La extracción de un conjunto de características representativas del tráfico a partir de los datos que circulan por la red ha logrado facilitar los trabajos de análisis, monitorización, evaluación y diseño de redes de comunicaciones. Una práctica común en la extracción de dichas características del tráfico es aplicar un modelo estadístico a los datos obtenidos de la red, de forma que el tráfico muestreado queda definido por los parámetros del modelo, simplificando así la descripción del mismo. Hoy en día, existen tecnologías que permiten la obtención de dichos datos de tráfico mediante sondas a velocidades adecuadas para no perder información como consecuencia del muestreo.

Es interesante que el modelo de tráfico empleado sea capaz de capturar y reproducir con la mayor fidelidad posible las principales características del tráfico real, ya que un buen modelado va a permitir la realización de simulaciones de redes ajustadas a la realidad que ayuden a mejorar el diseño de nuevas redes.

Además de su uso en la predicción de tendencias, una aplicación muy importante de los modelos de tráfico es la detección de anomalías en el tráfico agregado de redes IP [1]. En estos sistemas se cuenta con una fase de análisis de datos en la que se ajusta el vector de parámetros que describe el modelo estadístico a los datos observados, usándose dicho vector como conjunto de características extraídas de la muestra. Mediante inferencia estadística, basada comúnmente en el teorema de Bayes, estas características se emplean para decidir si el tráfico debe ser clasificado como normal o como anómalo.

Muchas son las aportaciones que se han ido haciendo a lo largo de los años en cuanto a la modelización del tráfico de red. El tráfico de red se ha venido considerando como un proceso estocástico que en cortos periodos de tiempo puede considerarse estacionario. Aunque son numerosas las aportaciones sobre modelos de tráfico, lo cierto es que no parece que exista un consenso que permita afirmar la evidencia de que alguno de los modelos existentes sea superior a los demás. Sin embargo, sí se tiene evidencia de que ciertas características del tráfico hacen que las predicciones hechas por los modelos tradicionales, no se ajusten fielmente a las características observables.

Una de estas características observables del tráfico de red que estos modelos no son capaces de reflejar es la propiedad de autosimilitud [2]. A grandes rasgos, esta propiedad implica que las características del tráfico observado a diferentes escalas parecen mantenerse, por lo que puede exhibir valores de dependencia a largo plazo. En la literatura [2][6], se aplican diferentes técnicas que introducen esta característica de la periodicidad a largo plazo, como pueden ser los modelos FARIMA (Fractional Auto Regressive Integrated Moving-Average) y FGN (Fractional Gaussian Noise).

Además de la dependencia a largo plazo, otra propiedad inherente al tráfico es la alta variabilidad o "efecto Noah" [9] que provoca la aparición de colas pesadas en la distribución. La presencia de colas pesadas no despreciables sugiere que los datos pueden ser mejor modelados usando modelos estadísticos de varianza infinita en lugar de otros modelos clásicos tales como el de Poisson, gaussiano o gamma [3].

2.2.1. Modelo de Poisson

El modelo de Poisson es el modelo de tráfico más antiguo que se encuentra en la literatura. Empezó a utilizarse hace más de medio siglo en el estudio y diseño de las primeras redes telefónicas.

Este modelo considera a los dispositivos de red como un sistema de colas, en el que en el proceso de llegada de paquetes, el tiempo que resta para la llegada del siguiente paquete es independiente del tiempo transcurrido desde la última llegada. Esto es, los tiempos entre llegadas de paquetes consecutivos se consideran independientes e idénticamente distribuidos según una VA (Variable Aleatoria) exponencial de parámetro λ (tasa de llegadas al sistema por unidad de tiempo), cuya función de distribución viene dada por:

$$P\{\tau \leq t\} = 1 - e^{-\lambda t}, \qquad t \geq 0,\ \lambda \geq 0 \tag{2.1}$$

Lo que implica que la función de densidad de probabilidad sea:

$$f_\tau(t) = \lambda e^{-\lambda t} \tag{2.2}$$

En estas condiciones, se puede demostrar [4] que el número de llegadas en un intervalo de tiempo t se distribuye según una VA de Poisson de media E{K} = λt y varianza Var{K} = λt, de forma que la probabilidad de que lleguen k paquetes en un intervalo t se define como:

$$P\{K = k\} = e^{-\lambda t}\frac{(\lambda t)^k}{k!}, \qquad k = 0,1,2 \ldots \tag{2.3}$$

En la figura 2.1 se ilustra una demostración del efecto que experimentan las distribuciones de Poisson al incrementar el valor de λ. Como se puede apreciar, para valores grandes del parámetro λ, la distribución converge a una gaussiana N (λt, λt). Esto es, que para tasas elevadas de paquetes por segundo el comportamiento de este modelo tenderá cada vez más a comportarse de forma parecida al modelo gaussiano, solo que con la limitación de contar con un solo grado de libertad.

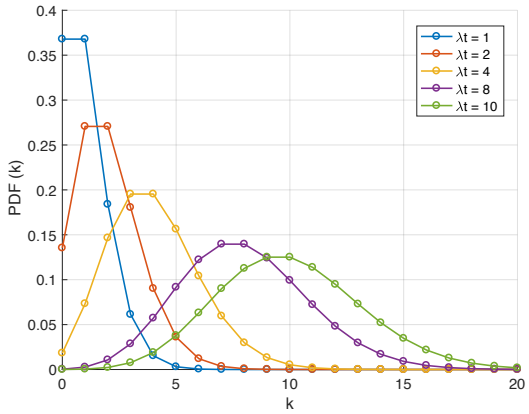


Figura 2.1 - Funciones de densidad de probabilidad de la distribución de Poisson para diferentes valores de λt.

A pesar de que la sencillez analítica que le confieren sus propiedades matemáticas ha apoyado tradicionalmente el uso de los procesos de Poisson en el modelado del tráfico de red, la distribución de Poisson presenta limitaciones a la hora de modelar en un entorno real. En concreto, el hecho de asumir que el tiempo entre la llegada de paquetes se ajusta a una distribución exponencial, carece de la variabilidad suficiente para representar el tráfico real, ya que se subestima la capacidad del tráfico de formar ráfagas, de las que se ha observado que su magnitud es altamente variable. Se contempla en la literatura que, si bien los procesos de Poisson modelan de forma adecuada los tiempos entre conexiones iniciadas por el usuario en servicios como TELNET o FTP, no es así para el caso de servicios como SMPT o WWW, ni para los tiempos entre llegadas de paquetes [5], ya que dicho tiempo no se distribuye según una variable aleatoria exponencial debido a la impulsividad del tráfico. El marcado comportamiento a ráfagas presente en el tráfico agregado no tiende a desaparecer en

intervalos temporales mayores, por lo que esta impulsividad del tráfico se relaciona íntimamente con la propiedad de la autosimilitud [6]. Además, el modelo de Poisson es un modelo estadístico para variables discretas, como es el caso del número de paquetes recibidos en un determinado intervalo temporal, por lo que se observan limitaciones al modelar datos de naturaleza continua.

En definitiva, el modelo de Poisson se presenta como un modelo incapaz de adaptarse a las trazas de tráfico de las redes modernas, haciendo necesario el desarrollo de modelos más avanzados basados en distribuciones con más grados de libertad.

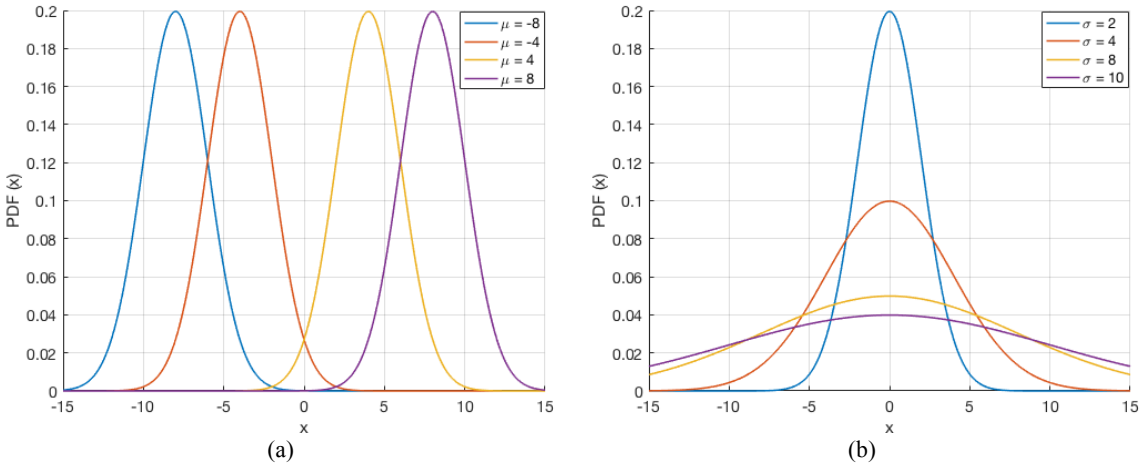
2.2.2. Modelo gaussiano

El modelo gaussiano es un modelo ampliamente conocido y utilizado en múltiples campos. La distribución normal describe de manera aproximada muchos fenómenos que ocurren en la naturaleza, la industria y la investigación. Muchos sucesos dentro del ámbito de las comunicaciones se caracterizan también con distribuciones gaussianas, y el tráfico de red no iba a ser menos, usándose éstas como modelo de primer orden para el tráfico de red agregado.

La función de densidad de probabilidad que caracteriza a una VA con distribución gaussiana de media μ y varianza σ^2 se expresa como:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{\frac{-(x-\mu)^2}{2\sigma^2}}, \qquad x, \mu \in \mathbb{R}, \sigma > 0 \tag{2.4}$$

A diferencia de la distribución de Poisson, la gaussiana cuenta con dos grados de libertad, siendo la media y la varianza dos parámetros independientes entre sí. Esta independencia hace que el modelo gaussiano aporte más libertad a la hora de modelar trazas de tráfico. Se puede ver en la figura 2.2 el efecto de la variación de cada parámetro manteniendo constante el otro: una variación en μ produce la misma función de densidad de probabilidad con posición desplazada en el eje x, mientras que σ regula la concentración de la distribución alrededor del valor μ determinado.



(a) (b)

Figura 2.2 - Funciones de densidad de probabilidad de la distribución gaussiana para diferentes valores de μ (figura a, σ = 2) y σ (figura b, μ = 0).

Además, el modelo gaussiano frente al modelo de Poisson, permite modelar magnitudes muestreadas del tráfico a partir de las cuales no sea posible deducir el parámetro λ, y también muestras de naturaleza continua ya que la función de densidad gaussiana está definida para todos los reales.

Sin embargo, el modelo gaussiano continúa siendo un modelo que al tener una varianza finita, no tiene en cuenta la alta variabilidad y limita su capacidad de adaptación a datos de tráfico muy impulsivos. Las distribuciones gaussianas presentan simetría, comportamiento que generalmente no se refleja en el tráfico de red. No obstante, la asimetría se puede intentar modelar como suma de gaussianas, mientras que la presencia de colas pesadas en las distribuciones del tráfico real continúa siendo un problema en su modelado con guassianas.

2.2.3. Modelo gamma

Buscando resolver la limitación que impone la simetría de las distribuciones expuestas anteriormente en el modelado de ciertos tipos de datos, cabe hablar de las distribuciones gamma que desempeñan un papel importante en la teoría de colas y de confiabilidad.

Una variable aleatoria continua X tiene una distribución gamma, con parámetros α y β, si su función de densidad viene dada por [7]:

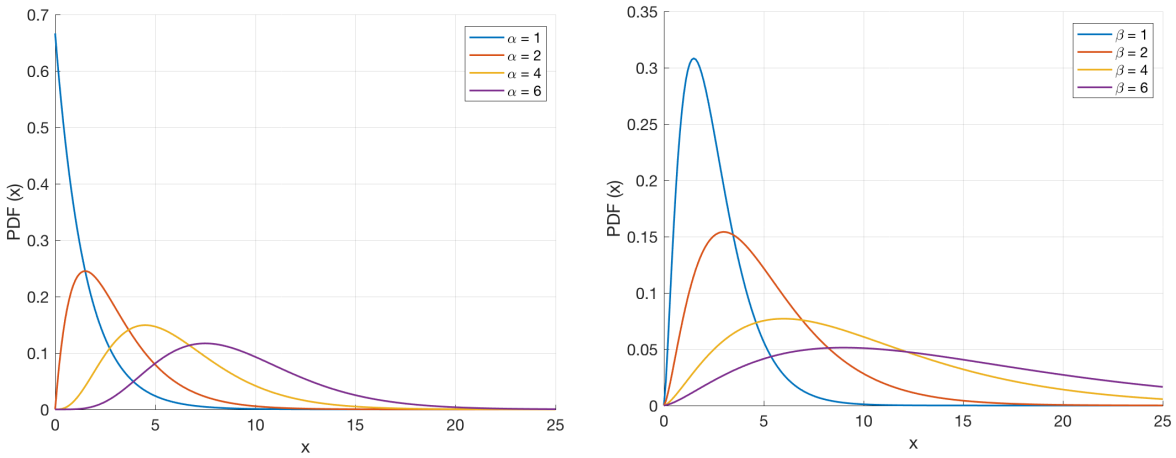
$$f(x) = \frac{1}{\beta^{\alpha}\,\Gamma(\alpha)}\; x^{\alpha-1}\; e^{-x/\beta}\;, \qquad x > 0,\ \alpha, \beta > 0 \tag{2.5}$$

Donde $\Gamma(\alpha)$ es la función gamma que se define como:

$$\Gamma(\alpha) = \int_0^{\infty} x^{\alpha-1}\; e^{-x}\; dx\,, \qquad \alpha > 0 \tag{2.6}$$

En la figura 2.3 se muestran funciones de densidad de probabilidad de la distribución gamma para varios valores de α y β, los cuales denotan la forma y la escala de la distribución. El caso especial para la que α = 1 es la distribución exponencial. Se puede ver que estas distribuciones permiten una mayor capacidad de adaptación del modelo gamma a, por ejemplo, datos de naturaleza asimétrica.

A pesar de esta ventaja con respecto a los modelos expuestos anteriormente, la distribución gamma cuenta con una limitación: su media ($E\{X\} = \alpha\beta$) y su varianza ($Var\{X\} = \alpha\beta^2$) están determinadas por los valores de sus parámetros de forma y escala, y esta dependencia no es deseable en la modelización de datos reales de tráfico de red, ya que va a limitar el modelado de las colas pesadas en la distribución de los datos.



(a) (b)

Figura 2.3 - Funciones de densidad de probabilidad de la distribución gamma para varios valores de α (figura a, β = 1.5) y β (figura b, α = 2.5).

2.2.4. Modelo α-estable

Algunos fenómenos no pueden ser descritos mediante la suposición gaussiana ya que presentan un grado de impulsividad mayor que la distribución normal, y se requiere un modelo que tenga en cuenta las colas pesadas que presenta su distribución. Para modelar de manera satisfactoria ese tipo de sucesos, en la literatura ha sido aplicada la distribución α-estable en diferentes disciplinas, tales como la economía, la hidrología, la astrofísica, la biología, el procesado de señal y las comunicaciones.

La principal propiedad de las distribuciones α-estables, a la que deben su nombre, es la propiedad de estabilidad, que establece que la suma de varias VA estables con el mismo valor de α es otra VA estable con el mismo valor de α [8].

La generalización de las gaussianas para el tratamiento estadístico del efecto de colas pesadas, hace que surjan este tipo de distribuciones. Por tanto, una de las principales justificaciones teóricas para usar la distribución gaussiana (el Teorema del Límite Central) también se cumple, en su versión generalizada, para las distribuciones α-estables, ya que la gaussiana es un caso particular dentro de las α-estables (α = 2). En términos generales, el Teorema del Límite Central Generalizado establece que la distribución de la suma de variables aleatorias cuya varianza no es finita, tiende a una distribución α-estable [8]. Hablar de varianza infinita en sucesos reales puede parecer poco intuitivo, sin embargo, en muchos de ellos se observa que un modelo de varianza infinita se adapta mejor a los datos que otros modelos existentes con varianza finita.

Son cuatro los parámetros que caracterizan a estas distribuciones, aunque según se presenta en [25], no existe un aparente acuerdo en cuanto a la parametrización de este tipo de distribuciones, recibiendo diferentes nombres dependiendo del autor. A lo largo de este trabajo se empleará la nomenclatura α, β, σ y μ, presentando dichos parámetros las propiedades que se exponen a continuación [8][9]:

- α puede tomar valores en el intervalo (0,2] y es el que más caracteriza la forma de la distribución, desde la curva gaussiana cuando $\alpha = 2$ hasta una degenerada cuando $\alpha \to 0$. Es considerado el *índice de estabilidad* o *exponente característico* que, conceptualmente hablando, controla el grado de impulsividad o las colas pesadas de la distribución, acentuándose este fenómeno para valores bajos de α.

- β varía dentro del intervalo [-1,1] y determina la asimetría de la distribución. Se dice que la distribución es *totalmente asimétrica hacia la izquierda* si $\beta = -1$, *totalmente asimétrica hacia la derecha* si $\beta = 1$, y si $\beta = 0$ la distribución es simétrica, es decir, la probabilidad se reparte por igual a ambos lados.

- σ y μ son los clásicos parámetros de dispersión ($\sigma > 0$) y localización ($\mu \in \mathbb{R}$) de la distribución. Aunque debido a su analogía, algunos autores los dotan del mismo nombre que reciben en las distribuciones gaussianas, σ nunca es igual a la desviación típica, ni siquiera en el caso gaussiano de $\alpha = 2$, y μ solo coincide con la media de la distribución cuando $\alpha > 1$.

En la figura 2.4 se muestra la evolución de la función de densidad de probabilidad α-estable al variar el parámetro α, observándose que conforme menor es su valor, mayor es el grado de impulsividad de la distribución y por tanto, la función de densidad de probabilidad presenta colas más pesadas. Análogamente, en la figura 2.5 puede observarse el efecto del parámetro β sobre la función de densidad de probabilidad. Cabe destacar que, conforme α se aproxima a 2, el parámetro β va perdiendo relevancia, ya que para el caso en el que la distribución coincide con la gaussiana, ésta es simétrica por definición.

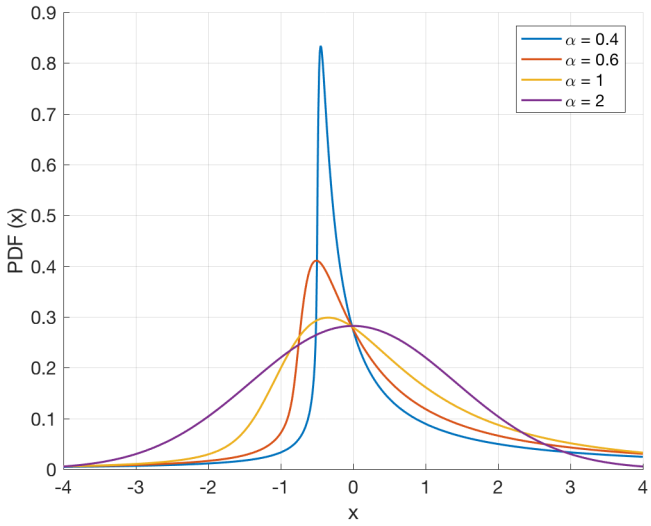


Figura 2.4 - Funciones de densidad de probabilidad de la distribución α-estable para diferentes valores de α ($\beta = 0.7$, $\sigma = 1$ y $\mu = 0$).

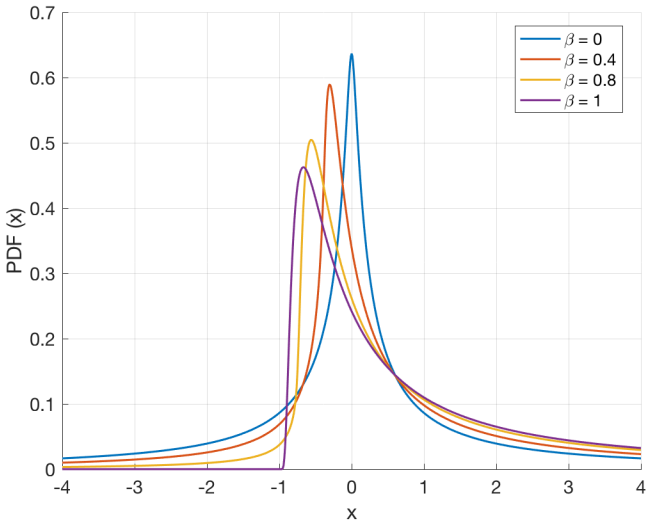


Figura 2.5 - Funciones de densidad de probabilidad de la distribución α-estable para diferentes valores de β (α = 0.5, σ = 1 y μ = 0).

Estos parámetros de forma, asimetría, dispersión y localización, dotan a las distribuciones α-estables de una gran flexibilidad. En lo que al tráfico de red respecta, este tipo de distribuciones son capaces de amoldarse a diversos grados de impulsividad y asimetría. Además, al contrario que la distribución gamma, el poder modelar de forma independiente las cuatro características que definen sus parámetros, hace que las distribuciones α-estables puedan adaptarse a diferentes situaciones de tráfico en la red.

Aunque las propiedades de la distribución α-estable apuntan a que su rango de aplicación es aún más amplio que el de la distribución gaussiana, su uso no está tan extendido como cabría esperar. El principal inconveniente es que la función de densidad de probabilidad α-estable existe y es continua, pero salvo para casos concretos, no cuenta con expresiones cerradas. Se las suele definir mediante su función característica, que se define como [9]:

$$E\{\exp(itX)\} = \begin{cases} \exp\left\{-\sigma^{\alpha}|t|^{\alpha}\left[1 - i\beta\tan\left(\frac{\pi\alpha}{2}\right)sign(t)\right] + i\mu t\right\} & \alpha \neq 1 \\ \exp\left\{-\sigma^{\alpha}|t|\left[1 - i\frac{2\beta}{\pi}sign(t)\log(|t|)\right] + i\mu t\right. & \alpha = 1 \end{cases} \tag{2.7}$$

Para que dicha expresión sea continua en todo el espacio de los parámetros, se han propuesto diferentes parametrizaciones de la misma. Una de las más empleadas en la literatura, y a lo largo de la realización de este trabajo, es la denominada *parametrización 0* ideada por John P. Nolan [10] basada en una serie de expresiones integrales que permiten el cálculo numérico de las funciones de distribución y de densidad de probabilidad α–estable. Dichas ecuaciones usan una transformación del parámetro de localización μ a μ_0, tal que:

$$\mu = \begin{cases} \mu_0 - \beta\tan\left(\frac{\alpha\pi}{2}\right)\sigma, & \alpha \neq 1 \\ \mu_0 - \beta\frac{2}{\pi}\sigma\ln\sigma, & \alpha = 1 \end{cases} \tag{2.8}$$

Sin embargo, en este procedimiento propuesto por Nolan, los requerimientos de cálculo que implica la estimación de los parámetros que caracterizan a estas distribuciones, hacen que su uso como herramientas de análisis de tráfico de red en tiempo real sea limitada.

Buscando resolver esta limitación, existen propuestas basadas en las distribuciones α-estables como las presentes en [11] y [12] que paralelizan los cálculos de la estimación de los parámetros mediante Pthreads y OpenCL respectivamente, consiguiendo una reducción del tiempo de estimación de los parámetros.

Con el mismo objetivo, el algoritmo serie expuesto en [13] presenta un método rápido para calcular una PDF (Probability Density Function) α-estable basándose en el precálculo de densidades en una rejilla apropiada del espacio de parámetros α-estable. Este método será el seguido en el desarrollo del presente trabajo, ya que según los resultados expuestos en [12], este algoritmo serie desarrollado en Matlab muestra un rendimiento prometedor.

2.3. Paralelización

Las restricciones tecnológicas en cuanto al aumento de velocidad de los procesadores y la búsqueda de sistemas cada vez más rápidos han dado paso al desarrollo de las arquitecturas paralelas como la principal alternativa. El sostenido avance de los microprocesadores en las últimas décadas ha mejorado notablemente las capacidades de los ordenadores, y con la finalidad de conseguir sistemas más veloces y eficientes, los avances en la tecnología se centran en arquitecturas basadas en múltiples CPU (Central Processing Unit) o núcleos en un mismo chip (procesadores multinúcleo). De esta forma se persigue mejorar las prestaciones habilitando varios núcleos para que ejecuten instrucciones de forma simultánea o paralela, e incrementar así su capacidad de cálculo, permitiendo trabajar con algoritmos cada vez más complejos.

Estos avances en el hardware implican un desarrollo en el software para el uso de la paralelización, hasta el punto de que algunas operaciones son paralelizadas automáticamente por el propio compilador. El desarrollo de arquitecturas paralelas precisa un cambio en los programadores a la hora de plantear los problemas a resolver. Esto ha dado lugar al nacimiento de ciertos algoritmos y estrategias de paralelización que se adaptan a la arquitectura utilizada y al problema en cuestión. De una forma general, los algoritmos en paralelo son más complejos que los algoritmos secuenciales, y los errores de programación son más difíciles de detectar.

La paralelización se trata de un campo muy amplio. Uno de los primeros pasos que se deben llevar a cabo es la comprensión de los distintos modelos, ya sean los basados en las arquitecturas paralelas tradicionales como en las emergentes tarjetas gráficas, y de las posibilidades que se pueden alcanzar utilizando los diversos estándares de programación paralela que han surgido a lo largo de los últimos años.

2.3.1. Modelos de programación paralela

La eficiencia de los algoritmos en paralelo depende de una serie de factores interrelacionados, entre los que por supuesto está el tipo de hardware disponible. En general, el hardware disponible para introducir paralelización [14] se divide en dos grandes grupos: los modelos de memoria compartida y los modelos de memoria distribuida. A este segundo grupo pertenecen por ejemplo los clústeres, mientras que el primero abarca la mayoría de los ordenadores personales actuales, que son los utilizados en este trabajo. De la misma manera en que existen distintos tipos de hardware para implementar la paralelización, también existen varios paradigmas orientados a la programación en paralelo.

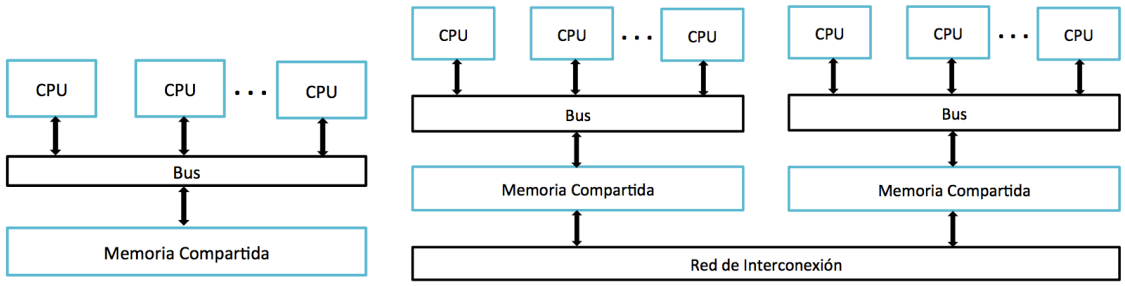
2.3.1.1. Modelo de memoria compartida

Un sistema de memoria compartida consta de un conjunto de procesadores que comparten un mismo espacio de memoria, de forma que cada procesador puede acceder a cada posición de memoria. En un sistema de memoria compartida, los procesadores se comunican entre ellos accediendo a estructuras de datos compartidas [15].

Dentro de estos sistemas, se encuentran dos tipos diferenciados, representados esquemáticamente en la figura 2.6:

1. Sistemas SMP (Symmetric Multi-Processing) o UMA (Uniform Memory Access), en los que todos los procesadores comparten una conexión común a la memoria principal y el acceso a cualquier zona de memoria se realiza a la misma velocidad.

2. Sistemas NUMA (Non-Uniform Memory Access), en los que algunos bloques de memoria pueden estar físicamente más estrechamente relacionados con algunos procesadores que otros, es decir, la memoria está físicamente distribuida de manera que la velocidad de acceso a un determinado bloque de memoria por parte de un procesador va a depender de la lejanía o cercanía de dicha memoria.



(a) (b)

Figura 2.6 - Sistemas de memoria compartida UMA (a) y NUMA (b).

Entre las múltiples API (Application Programming Interface) desarrolladas para crear y ejecutar programas paralelos en este tipo de sistemas de memoria compartida, en la literatura destacan Pthreads y OpenMP.

- **Pthreads**

En el inicio de la programación paralela cada fabricante lanzó al mercado su propia API para el manejo de hilos, lo que hizo que la portabilidad entre ellas fuera nula. Por este motivo, en 1995 se definió en el estándar IEEE POSIX 1003.1c. la interfaz Pthreads original. Actualmente, la librería Pthreads proporciona un conjunto bastante completo de rutinas para crear, destruir, gestionar y coordinar una colección de subprocesos, pudiendo así sincronizar los hilos e impedir que diferentes hilos intenten modificar los mismos valores a la vez.

- **OpenMP (Open Multi-Processing)**

La primera versión de OpenMP fue introducida en 1997 por un consorcio de diferentes empresas relacionadas con el desarrollo software y hardware llamado OpenMP ARB (OpenMP Architecture Review Board). Las recientes especificaciones de OpenMP [16] permiten crear y administrar programas paralelos escritos en Fortran, C y C ++ portables a arquitecturas de memoria compartida de diferentes proveedores, a través de una serie de directivas (denominadas *pragmas*) para el compilador, librerías y variables de entorno. Por lo tanto, OpenMP deja en manos del compilador la lógica y se basa en el uso de *pragmas* para marcar las zonas a ejecutar de forma paralela, y el programador debe elegir las variables que desea que sean compartidas entre todos los hilos que ejecutan dicha zona privada y aquellas que desea que sean privadas.

OpenMP proporciona una programación menos compleja que Pthreads, y hace que el código resultante no difiera sustancialmente de un programa secuencial equivalente. Con Pthreads, incluso las tareas simples se realizan a través de múltiples pasos, y por lo tanto un programa típico contendrá muchas llamadas a la librería. Por ejemplo, para ejecutar un bucle en paralelo, el programador debe declarar estructuras de subprocesos, crear y terminar los hilos individualmente, y calcular para cada hilo los límites del bucle. En comparación con Pthreads, las directivas de la API de OpenMP facilitan la especificación de la ejecución de bucle paralelo, la sincronización de subprocesos y la especificación de si se comparten o no los datos [14].

2.3.1.2. Modelo de memoria distribuida

En un sistema de memoria distribuida, cada procesador está emparejado con su propia memoria privada, y los pares de procesador-memoria se comunican a través de una red de interconexión, como se puede ver de forma esquemática en la figura 2.7. En estos sistemas los procesadores se comunican enviando mensajes o utilizando funciones especiales que proporcionan acceso a la memoria de otro procesador [15].

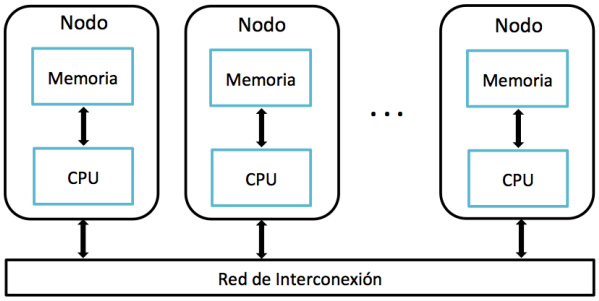


Figura 2.7 - Sistema de memoria distribuida.

Los nodos que componen los sistemas de memoria distribuida suelen ser sistemas de memoria compartida con uno o más procesadores multinúcleo. Para distinguir estos sistemas de los sistemas puros de memoria distribuida, se les denomina sistemas híbridos.

Para la comunicación entre procesos que se ejecutan en distintos procesadores se cuenta con un modelo de paso de mensajes, el cual requiere llamadas explícitas a funciones de envío/recepción y dificulta la programación paralela en los sistemas de memoria distribuida. En un entorno de comunicación de memoria distribuida en el que las rutinas de nivel superior y/o abstracciones se basan en rutinas de paso de mensajes de nivel inferior, los beneficios de la estandarización son particularmente evidentes.

- **MPI (Message Passing Interface)**

MPI ha sido desarrollado desde 1993 a través de un proceso abierto por una comunidad de proveedores de computación paralela, científicos y desarrolladores, con el fin de establecer un estándar práctico, portable, eficiente y flexible para el paso de mensajes. En dicha norma [17] se define la sintaxis y la semántica de un conjunto de rutinas contenidas en la interfaz de paso de mensajes para su uso en programas escritos en Fortran, C y C++, permitiendo implementaciones paralelas que se pueden utilizar en un entorno heterogéneo y en las plataformas de muchos proveedores.

Los programas MPI pueden ejecutarse en sistemas de memoria distribuida, alcanzando procesadores de diferentes nodos y llevando a cabo en cada uno de ellos una parte de la ejecución total. Dado que muchos sistemas tipo clúster presentan arquitecturas híbridas, MPI se combina cada vez más con OpenMP, pudiéndose así distribuir tareas entre los diferentes nodos gracias a MPI para abrir dentro de cada uno de los nodos múltiples hilos utilizando OpenMP.

Sin embargo, MPI requiere un esfuerzo de programación relativamente alto. Su dificultad se debe en gran parte a la mayor capacidad que tiene el programador de gestionar los procesos. El programador debe crear el código que será ejecutado por cada proceso, y esto implica una buena reorganización del código secuencial. La necesidad de reestructurar todo el programa no permite la paralelización incremental al igual que OpenMP. Además, puede ser difícil crear una única versión del programa que funcione eficientemente en muchos sistemas diferentes, ya que el coste relativo de comunicar datos y realizar cálculos varía de un sistema a otro [14].

2.3.2. Unidades de Procesamiento Gráfico

Al hablar de programación se tiende a pensar en la ejecución sobre CPUs. No obstante, podemos contar con unas unidades de procesamiento que están entrado con fuerza en la computación de altas prestaciones: las GPUs (Graphics Processor Unit). La estrategia de su uso persigue el fin de aprovechar la gran cantidad de núcleos existentes en una tarjeta gráfica para realizar un gran número de operaciones en coma flotante de forma paralela en cada núcleo, explotando así al máximo el paralelismo que se puede obtener en un equipo.

Las operaciones que pueden realizar en una GPU suelen ser básicas, por lo que son aconsejables en la paralelización a nivel de datos o ejecuciones tipo SIMD (Single Instruction Multiple Data) [18] aprovechando el paralelismo que ofrecen sus múltiples núcleos, que permiten el lanzamiento de un número muy alto de hilos simultáneos. En comparación, las CPUs poseen menor número de núcleos, pero las operaciones que pueden realizar son más complejas.

Existen tecnologías como CUDA y OpenCL que otorgan un nivel de abstracción que permite asignar ciertos trabajos a la tarjeta gráfica para ser procesados en paralelo, distribuyéndose de manera automática y transparente para el programador.

- **CUDA (Compute Unified Device Architecture)**

NVIDIA, importante proveedor mundial de GPUs, ha creado CUDA como ayuda a los desarrolladores para explotar las capacidades de sus propias tarjetas gráficas. Consta de un compilador y de un conjunto de herramientas de desarrollo que permiten usar una variación del lenguaje de programación C para codificar algoritmos en sus GPUs [19].

- **OpenCL (Open Computing Language)**

OpenCL, a diferencia de la plataforma propietaria anterior, consta de una API abierta y compatible con GPUs tanto del fabricante NVIDIA, como AMD o Intel. Permite crear aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse en CPUs, GPUs o FPGAs (Field-Programmable Gate Arrays) [20].

La realidad es que la utilización de las GPUs requiere un conocimiento específico del comportamiento y la arquitectura de las mismas. Llevar a cabo las modificaciones necesarias para que un código pueda ser ejecutado por una GPU tal vez no sea tan complicado por la codificación que se tiene que llevar a cabo, sino quizás por los conocimientos previos que debe tener el programador acerca del hardware que tiene a su disposición, entendiendo las estructuras de memoria disponibles en la GPU.

2.4. Conclusiones

En esta sección se han presentado los diferentes modelos de tráfico más utilizados en la literatura y su capacidad de adaptación a una de las propiedades que presenta el tráfico de red: la alta variabilidad. Según todo lo expuesto, las distribuciones α-estables se perfilan como buenas candidatas para hacer frente a esta propiedad y en general al comportamiento del tráfico real, alcanzándose en la literatura prometedores resultados con la elección de las distribuciones α-estables para el modelado del tráfico de red.

Sin embargo, las distribuciones α-estables son particularmente difíciles de manejar porque carecen de expresiones analíticas para su PDF (Probability Distribution Function). Como consecuencia, el uso de esta distribución en situaciones en las que de otra manera serían una familia de distribuciones muy flexible, no está tan extendido en la actualidad. Se han descrito métodos numéricos en la literatura, pero tienden a ser engorrosos de procesar. Por ello, pretendiendo alejar este problema, el método propuesto en [13] parte de una serie de funciones de densidad de probabilidad α-estables precalculadas sobre una rejilla del espacio de parámetros, cuyo algoritmo se analizará más a fondo en el capítulo 3, y cuyos resultados muestran una ganancia temporal sustancial respecto a la evaluación directa de las expresiones propuestas en la literatura previa, lo que ha motivado a su elección como punto de partida para este Trabajo Fin de Máster.

Por otro lado, a día de hoy los ordenadores personales suelen ser multinúcleo y el paralelismo en estos entornos es viable y asequible. De esta forma se persigue mejorar las prestaciones habilitando varios núcleos para que ejecuten instrucciones de forma paralela e incrementar así su capacidad de cálculo. Según lo estudiado en este capítulo, el paradigma de programación paralela más adecuado para el trabajo en cuestión y según las características del hardware disponible, es el de memoria compartida. La principal ventaja que se encuentra con este paradigma es la posibilidad de explotar el paralelismo de baja granularidad sin ser gravemente penalizados por la sobrecarga. Esto es así debido a la baja latencia de las operaciones de creación, sincronización y comunicación entre hilos. Se ha hablado de dos alternativas: Pthreads y OpenMP. La paralelización mediante hilos que facilita Pthreads tiene la ventaja de que el programador tiene un gran control sobre el programa. Sin embargo, requiere un gran número de instrucciones, incluso para tareas muy sencillas, resultando una programación más compleja y propensa a errores. Debido al tamaño relativamente pequeño del método que se estudiará en el capítulo 4, se ha desechado el uso de Pthreads. Si bien se han presentado otras tecnologías de programación paralela también muy utilizadas y que dotan de interesantes posibilidades al mundo del paralelismo, el presente trabajo se ha centrado en OpenMP y su facilidad de uso y aprendizaje.

Tras todo esto, y siguiendo el objetivo del presente proyecto de desarrollar una herramienta de tratamiento estadístico de distribuciones α-estables y en su aceleración paralela para lograr analizar tráfico de redes IP en tiempo real, se procede a presentar en el siguiente capítulo un análisis previo que permita averiguar si es posible modelar una magnitud concreta del tráfico de red con un modelo estadístico α-estable.

Capítulo 3. Planteamiento del problema

3.1. Introducción

Según lo expuesto en el capítulo anterior, en la literatura se cuenta con referencias que presentan resultados atractivos al caracterizar el tráfico de red aplicando un modelo basado en distribuciones α-estables. En concreto, en los trabajos como los presentes en [3] y [21], se confirma que, efectivamente, existen características en el tráfico de red que hacen que un modelo α-estable reproduzca con mayor fiabilidad su comportamiento. Una buena adaptación del modelo estadístico al tráfico de red real, va a facilitar determinados trabajos dentro de la gestión de red, como pueden ser la detección de anomalías, la predicción de tendencias, y el buen dimensionamiento de las infraestructuras.

El modelo α-estable, además de adaptarse fielmente al volumen del tráfico de red [3][21], otra magnitud interesante que ha demostrado también modelarse bajo un modelo estadístico basado en distribuciones α-estable es el RTT (Round Trip Time). Una predicción precisa del RTT es fundamental para evitar congestiones y retrasos. En [22] se plantea que la estimación actual del RTT basada en un modelo gaussiano, se aparta sustancialmente del valor real en los escenarios de cola pesada, y propone la distribución de Cauchy como aproximación útil para hacer las estimaciones, a falta de una herramienta para calcular la distribución α-estable, ya que la distribución de Cauchy se trata de un caso particular de la α-estable ($\alpha = 1$, $\beta = 0$). Adicionalmente, de los resultados obtenidos en [23], se deduce que las distribuciones α-estables se ajustan mejor al RTT que los modelos basados en otras distribuciones como Weibull y Lognormal, y plantea que, dado que la cola de la distribución está relacionada con la carga de tráfico, se puede establecer un umbral basado en los parámetros de la distribución α-estable que permita determinar retardos de paquetes.

Sirviendo de motivación los trabajos recién expuestos, en este capítulo se plantea una magnitud del tráfico de red de la que hasta el momento no se han encontrado referencias en cuanto a su modelado α-estable, y que ha resultado de interés como posible indicador de la carga del servidor: el tiempo de respuesta del servidor en el establecimiento de la conexión TCP.

3.2. Establecimiento de la conexión TCP

El protocolo TCP es un protocolo de la capa de transporte de Internet, fiable y orientado a la conexión. Se dice que TCP está orientado a la conexión porque antes de que un proceso de la capa de aplicación pueda comenzar a enviar datos a otro, ambos deben primero establecer una comunicación entre ellos; es decir, tienen que enviarse ciertos segmentos preliminares para definir los parámetros de la transferencia de datos que van a llevar a cabo a continuación.

Toda conexión TCP exitosa comienza con un acuerdo en tres fases o *three-way handshake*, representada en la figura 3.1, procediendo de la siguiente manera:

1) El protocolo TCP del lado del cliente envía un segmento SYN con su número de secuencia inicial.
2) El servidor extrae dicho segmento SYN, asigna los buffers y variables TCP a la conexión, y envía un segmento SYNACK de conexión concedida al cliente. Dicho segmento incluye un segmento SYN con su propio número de secuencia inicial y un segmento ACK para el número de secuencia inicial del cliente.
3) Al recibir el segmento SYNACK, el cliente también asigna buffers y variables a la conexión, y envía entonces al servidor otro segmento que confirma el segmento de conexión concedida del servidor.

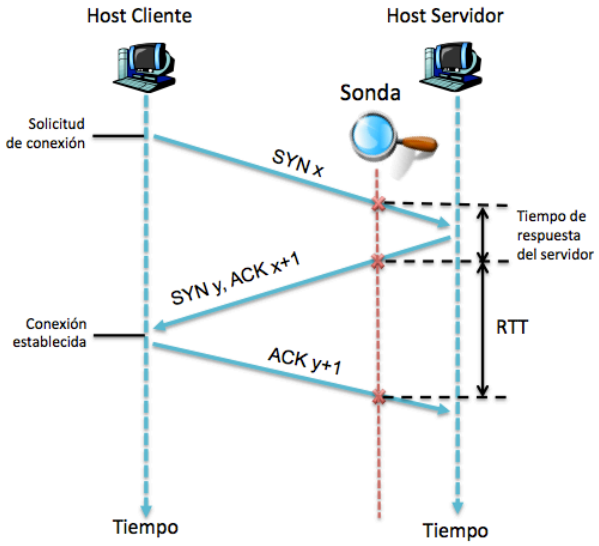


Figura 3.1 - Establecimiento de la conexión TCP.

En este escenario de la figura 3.1, el RTT se define como el tiempo que tarda un paquete de datos enviado desde el servidor en volver a este mismo habiendo pasado por el host cliente de destino. Por tanto, la implantación de una sonda entre ambos elementos permite tomar medidas de dicho RTT, así como de otras magnitudes del tráfico trasportado por la red, que facilitan importantes labores en la gestión de red, como puede ser la detección de problemas en el servidor y la planificación de su carga, con la gran ventaja de realizar las mediciones de forma no intrusiva en el servidor.

En este Trabajo Fin de Máster, se plantea el modelado de otra magnitud dentro del contexto del establecimiento de la conexión TCP: el tiempo de respuesta del servidor. Esto es, lo que el servidor tarda en responder a la solicitud del cliente y conceder por tanto la conexión, planteándose como un interesante indicador de lo cargado que se encuentra el servidor.

El tiempo de respuesta del servidor en el establecimiento de la conexión TCP se calcula en la práctica como el tiempo que transcurre desde el *timestamp* registrado del primer SYN del cliente al servidor, y el *timestamp* del primer SYN del servidor al cliente. Como primera aproximación para el modelado de esta magnitud, se realizó un ajuste con diferentes tipos de distribuciones, a partir del paquete *Statistics and Machine Learning Toolbox* de Matlab [26], sobre un conjunto de medidas de un servidor concreto, extraídas de los registros que se expondrán posteriormente en el capítulo 6. En concreto, estas medidas de tiempo de respuesta del servidor se corresponden con las medidas registradas a lo largo de una hora de alta actividad, un lunes de 07:00 a 08:00 AM UTC. En la figura 3.2 se muestra un ejemplo de ajuste de una traza de tráfico mediante diferentes tipos de distribuciones. De esta forma, la información de primer orden contenida en la traza (izquierda) queda caracterizada por los parámetros correspondientes a cada modelo (derecha).

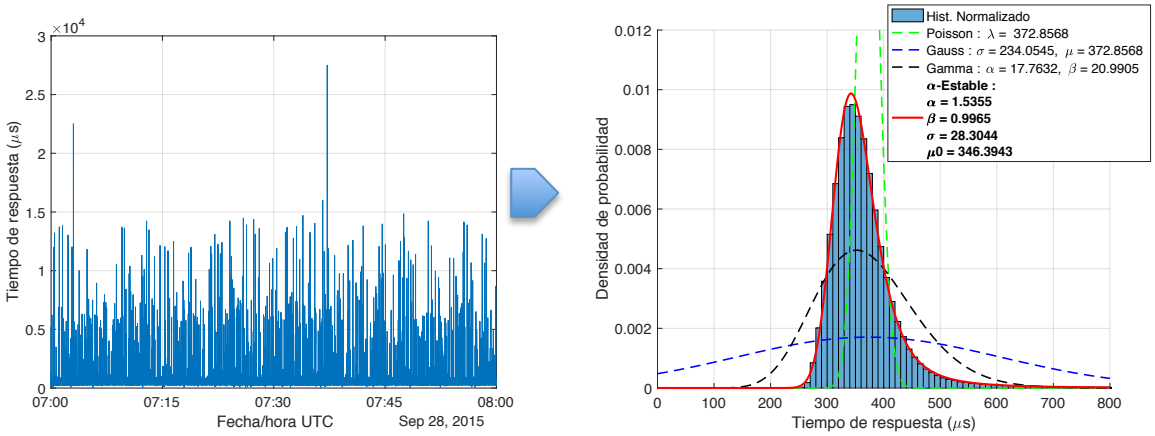


Figura 3.2 - Ejemplo de la distribución del tiempo de respuesta de un servidor en una ventana temporal de una hora y la PDF obtenida aplicando diferentes ajustes.

Por inspección, a partir de la comparativa gráfica del ajuste de diferentes distribuciones a las muestras de tráfico real adquiridas, la distribución α-estable se presenta como una buena candidata para el modelado del tiempo de respuesta de un servidor en el establecimiento de la conexión TCP. La presencia de colas pesadas en la distribución de los datos adquiridos, hacen que el ajuste con otro tipo de distribuciones se vea limitado.

3.3. Conclusiones

El crecimiento exponencial de las redes actuales implica una importante demanda en los servidores de red, y por tanto, la capacidad de medir el efecto de esta demanda es interesante para, por ejemplo, poder optimizar los distintos componentes que componen un servidor. Por ello, en este capítulo se ha presentado la magnitud de tráfico que se persigue modelar en este trabajo: el tiempo de respuesta del servidor en el establecimiento de conexión TCP.

Un primer ajuste realizado con la aplicación de Matlab, ha permitido inferir que la distribución α-estable se adapta mejor a las características de la distribución de los datos, la cual presenta una clara asimetría y la presencia de colas pesadas que otros modelos no son capaces de caracterizar.

Sin embargo, como ya se ha venido exponiendo hasta ahora, la ineficiencia de las herramientas existentes en la actualidad para la estimación de los parámetros de las distribuciones α-estables hacen que su uso en un entorno real de monitorización de tráfico de red se vea limitado. Sin ir más lejos, la aplicación de Matlab utilizada en este capítulo es especialmente lenta en la estimación de los parámetros de la distribución α-estable, lo que refuerza más aún el desarrollo que va a llevarse a cabo en este trabajo.

En este contexto, surge la necesidad de desarrollar una herramienta capaz de estimar los parámetros que caractericen el tiempo de respuesta del servidor de forma rápida, como se verá en el siguiente capítulo.

Capítulo 4. Optimización del algoritmo serie

4.1. Introducción

Tras la observación, en el capítulo anterior, de que las distribuciones α-estables se adaptan más fielmente que otros modelos tradicionales a las muestras del tiempo de respuesta del servidor en una conexión TCP, se procede a escoger un método existente en la literatura que permita desarrollar una herramienta para la estimación rápida de dichos parámetros.

Dadas las limitaciones actuales para el trabajo con distribuciones α-estables por la falta de expresiones analíticas para su PDF que dificultan la estimación de sus parámetros, se hace necesario contar con herramientas numéricas. En la literatura se han propuesto muchas técnicas de estimación de parámetros para estas distribuciones, pero todas sufren de alguna forma la carencia de una expresión cerrada para la PDF. En concreto, los métodos de estimación basados en su evaluación de la PDF como son los de máxima verosimilitud, requiere un gran número de evaluaciones para converger, lo que puede ser inviable cuando se necesita un análisis en tiempo real y por ello sea deseable acelerar su ejecución.

Han surgido diferentes estrategias con el objetivo de solventar esta limitación, centrándose el interés en este trabajo en el método desarrollado en [13]. Como se cita en dicho artículo, se propone un algoritmo rápido para el cálculo de las PDF α-estables basándose en el precálculo de funciones de densidad de probabilidad sobre una rejilla apropiada de puntos en el espacio de parámetros α-β, manteniendo tanto el uso de memoria como el nivel de distorsión a una cantidad limitada deseada. Los resultados presentes en el artículo muestran una ganancia temporal respecto a la evaluación directa de los algoritmos de integración propuestos por Nolan [10], haciendo que sea posible aplicar eficientemente la estimación de máxima verosimilitud a las distribuciones α-estables, sin incurrir en la penalización de rendimiento típicamente asociada con ellas.

En este capítulo se pretende analizar el método escogido desde un punto de vista constructivo, tratando de encontrar posibles vías de mejora que permitan su implementación en un entorno de análisis de tráfico de red en tiempo real.

Las limitaciones que podemos encontrar en este programa son relativas siempre al tiempo de ejecución necesario para obtener los resultados requeridos. La primera mejora que se plantea para aumentar la eficiencia computacional del método escogido, con el objetivo de contribuir al área del análisis de tráfico en tiempo real, es la portabilidad de código serie escrito en Matlab al lenguaje C, para posteriormente poder ser ejecutado en paralelo.

4.2. Análisis del algoritmo seleccionado

Como se mencionaba anteriormente, el algoritmo seleccionado propone calcular densidades α-estables a partir de unos valores precalculados y almacenados eficientemente. Dicho algoritmo ha sido programado en Matlab, cuyo código se encuentra disponible en [27]. En este método se pueden distinguir dos etapas, a desarrollar a continuación:

I. El precálculo de las densidades de probabilidad α-estables.
II. La estimación rápida de los parámetros, basada en los anteriores valores precalculados, y donde se han centrado los esfuerzos en este trabajo.

4.2.1. Etapa I: Precálculo de las densidades α-estables

Los criterios establecidos en esta primera etapa, basados en la subdivisión recursiva de la malla de valores α-β a partir de la definición de unas medidas de calidad ($Q_{\alpha\beta}$ y Q_x) que deben mantenerse por encima de unas tolerancias establecidas ($Tol_{\alpha\beta}$ y Tol_x), así como los algoritmos de muestreo del espacio α-β y de la dimensión x, pueden consultarse en [13]. Los detalles sobre la construcción de la rejilla están fuera del alcance de este documento, ya que para el desarrollo de la herramienta se tomará como punto de partida la malla disponible en [27] definida para $0{,}4 \leq \alpha \leq 2$, $-1 \leq \beta \leq 1$, $-10^8 \leq x \leq 10^8$, y $Tol_{\alpha\beta} = Tol_x = 30$dB. La figura 4.1 muestra dicha malla de valores α-β, en la que se puede ver que el número de puntos necesarios para mantener la suavidad entre puntos vecinos deseada aumenta conforme α se aproxima a 0, zona donde los cálculos tienden a ser más lentos y menos precisos.

La construcción de la malla a partir de la función *stabletabulate.m* es un proceso *off-line,* y por muy lento que pueda llegar a ser, no influye en el rendimiento del algoritmo en el tiempo de ejecución. Por este motivo, buscando acelerar el algoritmo para ser aplicado en tiempo real, no se ha investigado tan a fondo las funciones del código disponible encargadas del cómputo de la malla, ya que una vez calculada no es necesario hacerlo de nuevo.

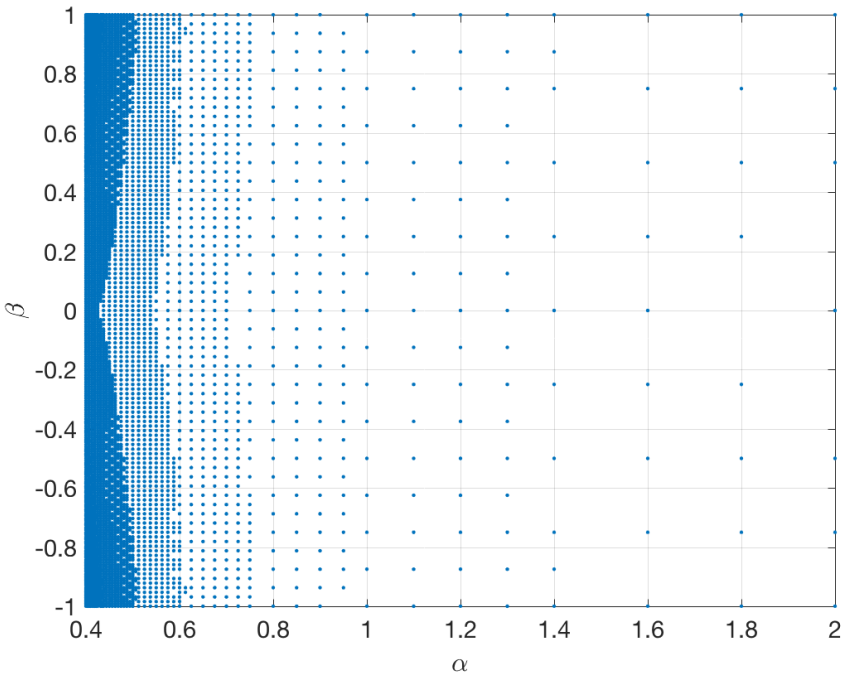


Figura 4.1- Malla de valores α-β precalculados disponible en [27].

La estructura de datos disponible en [27] almacena los resultados devueltos por dicha función, esto es:

- La rejilla α-β, representada en la figura 4.1, que consta de 6076 puntos en el intervalo $\alpha \in [0.4, 2]$ y $\beta \in [-1, 1]$, fijándose $Tol_{\alpha\beta}$ a 30 dB.
- Las coordenadas x-y de todas las PDF estándar ($\sigma = 1$ y $\mu = 0$) precalculadas a partir de un vector logarítmicamente espaciado en el intervalo $x \in [-10^8, 10^8]$ y ajustada Tol_x a 30 dB, habiendo generado el algoritmo de muestreo vectores x con 175.24 puntos de media.
- El objeto resultante de la posterior generación de una triangulación de Delaunay a partir de los valores α-β precalculados. La necesidad de la dicha triangulación en este método se detalla en el análisis de la etapa II.

Para la aplicación que en este trabajo se persigue, se considerará que esta estructura de datos proporcionada contiene puntos precalculados de densidades α-estables con la suficiente información y precisión para aproximar una densidad α-estable arbitraria con un vector cualquiera de parámetros $\{\alpha, \beta, \sigma, \mu_0\}$ (siempre que α, β y x caigan dentro del dominio precalculado), y no tener un problema de mayor tamaño que incremente el consumo de memoria al almacenar todos estos datos con formato en coma flotante de doble precisión.

4.2.2. Etapa II: Estimación rápida de los parámetros

El método de estimación del código Matlab de estudio se basa en un MLE (Maximum Likelihood Estimator) [7]. Este estimador supone una familia de densidades paramétricas, considerando los parámetros θ de dicha densidad cantidades deterministas desconocidas, y estima dichos parámetros del modelo maximizando la verosimilitud, es decir, la mejor estimación Likelihood es aquella que maximiza la probabilidad de obtener las muestras observadas. Para conseguir este máximo se emplea la función log-verosimilitud, la cual es creciente estricta. Por tanto, el MLE de un conjunto de datos $x_1, x_2, \ldots, x_n$ observado, independientes e idénticamente distribuidos, se define como:

$$\hat{\theta}_{MLE} = \arg\max_{\theta} \widehat{l}\,(\theta|\, x_1, \ldots, x_n)$$
$$\widehat{l}\,(\theta|x_1, \ldots, x_n) = \sum_{i=1}^{n} \ln f(x_i|\theta) \tag{4.1}$$

El estimador de máxima verosimilitud proporciona buenas propiedades de convergencia a medida que la cantidad de muestras aumenta, y es considerado el estimador más preciso disponible para las distribuciones α-estables [28]. Sin embargo, requiere de numerosas evaluaciones de la PDF para maximizar la probabilidad en el espacio de parámetros de cuatro dimensiones. Como se ha venido exponiendo, en las distribuciones α-estables, los métodos numéricos de Nolan para la aproximación de la PDF tienden a ser lentos, y por ello, contar con valores precalculados permite desarrollar un método rápido para la estimación de sus parámetros.

Partiendo de una malla de densidades α-estables almacenadas parece intuitivo el cálculo de una PDF arbitraria de parámetros (α, β) buscando el vecino más cercano dentro de la rejilla α-β precalculada en la etapa I. Sin embargo, tal y como se defiende en el artículo [13], este enfoque puede no ser adecuado para ciertos algoritmos de estimación de parámetros porque desconocerían el gradiente de densidades próximas y por tanto, podrían no converger al mínimo global. El método estudiado evita este problema interpolando una serie de densidades precalculadas cercanas al punto α-β requerido. Esto es lo que motiva a dividir el espacio de parámetros α-β en una malla de triángulos cuyos vértices son los puntos α-β obtenidos previamente en la etapa I, y de esta forma, los vértices del triángulo que contiene el punto (α, β) solicitado son considerados esos puntos cercanos en los que calcular las densidades interpoladas.

La triangulación escogida es la de Delaunay, cuyas propiedades geométricas dictan que [29] siendo P un conjunto de datos en el plano, y T una triangulación de P; T es una triangulación de Delaunay de P, si y solo si, la circunferencia circunscrita de cualquier triángulo de T no contiene puntos de P en su interior. En la figura 4.2 se representa la triangulación única de Delaunay generada a partir de los puntos α-β precalculados de los que partimos, formada por un total de 11709 triángulos. Se puede ver que todos los puntos están conectados entre sí y forman el mayor número de triángulos posibles sin que se crucen sus aristas.

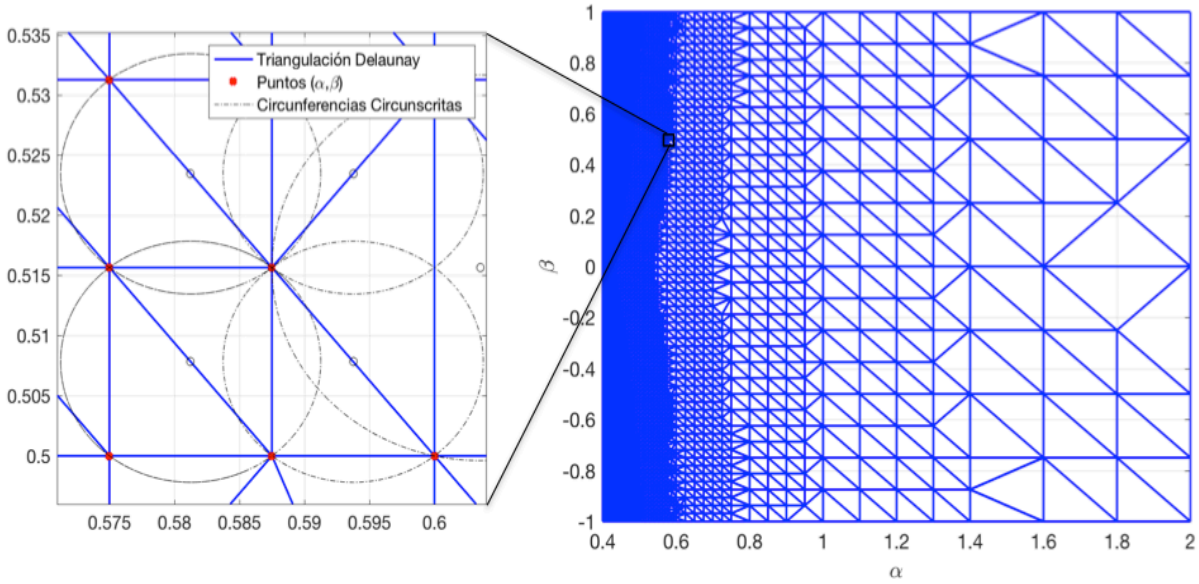


Figura 4.2- Triangulación de Delaunay aplicada en [27] a partir de los valores α-β precalculados.

El método de estimación de parámetros desarrollado en esta etapa II, primero realiza una estimación inicial a partir de las muestras para, posteriormente, refinar sucesivamente esa estimación aplicando un algoritmo de optimización a la función de verosimilitud. Para cada evaluación de la PDF que requiere dicha optimización, se distinguen varios pasos:

- **Paso 1:** Se obtiene el índice del triángulo que contiene el punto α-β solicitado y sus coordenadas baricéntricas. Esto se realiza mediante la función *pointLocation* de Matlab. Esta función parte de un objeto de triangulación generado en la etapa I mediante la función *DelaunayTri*, y se sirve de él para realizar un algoritmo de búsqueda optimizado. Lamentablemente, el código de esta función no es abierto y se desconoce el algoritmo concreto que realiza para la localización del punto solicitado en la triangulación dentro del dominio α-β precalculado.

- **Paso 2:** Se obtienen las coordenadas x-y de las tres PDF α-estables almacenadas para los puntos α-β de los vértices de dicho triángulo envolvente. El objeto de triángulos precalculado cuenta con una matriz de Nx3, de forma que para cada triángulo de los N totales que ha generado la triangulación única de Delaunay, se tienen tres valores que indican la fila de la matriz de puntos (α, β) al que corresponde cada vértice.

- **Paso 3:** Se desnormalizan las tres PDF de los vértices, ya que la información de la malla precalculada corresponde a densidades estándar ($\sigma = 1$ y $\mu_0 = 0$). Se calcula entonces la PDF a partir de su versión estándar, desplazando y escalando:

$$f(x;\alpha,\beta,\sigma,\mu_0) = \frac{1}{\sigma} f\left(\frac{x-\mu_0}{\sigma};\alpha,\beta\right) \qquad (4.2)$$

- **Paso 4:** Se realiza una interpolación cúbica de las PDF resultantes de cada vértice sobre el vector x solicitado. Esto se realiza mediante la función *interp1* de Matlab.

- **Paso 5:** La PDF resultante de todo este algoritmo pasa a calcularse como:

$$\hat{f}(x;\alpha,\beta) = \sum_{i=1}^{3} w_i \hat{f}(x;\hat{x_i},\alpha_i,\beta_i) \qquad (4.3)$$

  Siendo $\hat{f}(x;\hat{x_i},\alpha_i,\beta_i)$ las densidades α-estables precalculadas interpoladas en x para cada uno de los tres vértices del triángulo envolvente, obtenidas en el paso anterior, y siendo w_i los pesos de cada una de esas densidades vecinas al punto (α, β). Estos pesos corresponden a las coordenadas baricéntricas, las cuales son proporcionales a las áreas de los subtriángulos que se forman al tomar el punto requerido como vértice central, como se representa más adelante en la figura 4.5. Dada esta proporcionalidad, con este método se consigue dar más relevancia (mayor peso) a aquella PDF del vértice más cercano al punto (α, β) que está siendo evaluado.

- **Paso 6:** Finalmente, se calcula la función log-verosimilitud y el algoritmo de optimización valorará la necesidad o no de seguir evaluándola para maximizarla en el espacio de parámetros de cuatro dimensiones. Para esta fase de maximización, es utilizada la función *fmincon* de Matlab, obteniéndose los mejores resultados en las pruebas realizadas, tal y como se indica en [13], con el algoritmo SQP (Sequential Quadratic Programing) de entre los disponibles, cuyo código es cerrado.

En definitiva, se muestran de forma esquemática en la figura 4.3 los pasos del algoritmo iterativo implementado en esta etapa II del método de estudio, partiendo de los datos previamente calculados (*stable_table* y *tri*) en la etapa I, y tras una estimación inicial de los cuatro parámetros que el algoritmo iterativo de optimización va refinando.

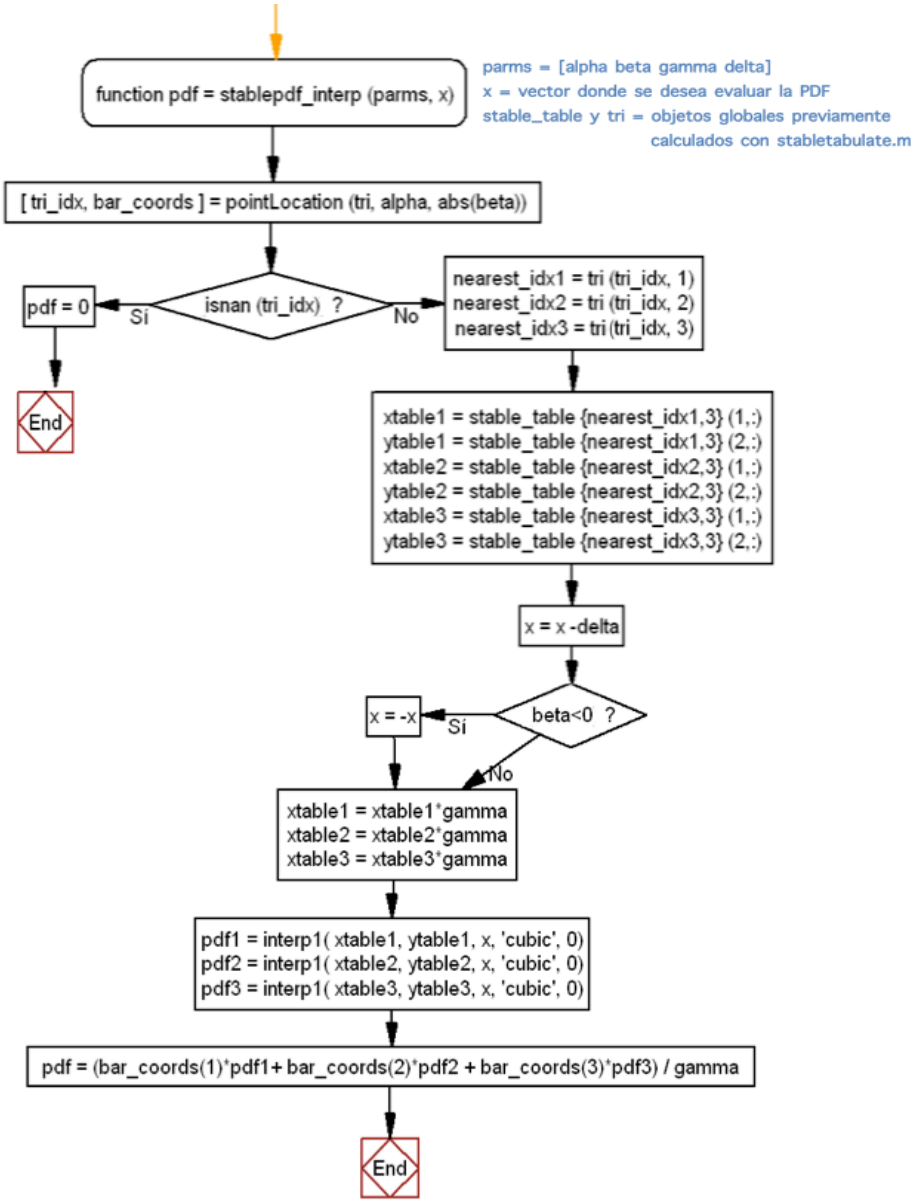


Figura 4.3- Diagrama de flujo del algoritmo [27] para la obtención de la PDF.

4.3. Desarrollo en C

Matlab es una herramienta muy utilizada en las fases iniciales de prototipado debido a la rapidez de desarrollo y depuración del código. Cuando ya se cuenta con un programa en Matlab que funciona correctamente, el código se puede escribir y depurar más rápidamente en C, manteniendo en Matlab solamente el pre y postprocesamiento.

El enlazado entre Matlab y C se puede realizar mediante funciones MEX o por medio de una implementación independiente, en la que los datos generados en Matlab se vuelcan en ficheros de texto o binarios para luego ser leídos desde el programa en C. Esta segunda alternativa de trabajo resulta beneficiosa cuando la aplicación depende de librerías externas, lo cual puede volver más complejo el proceso de compilación.

Como ya se ha mencionado, el desarrollo en cuestión va a centrarse en la etapa II del método, la estimación rápida de los parámetros, para la que se va a contar con la estructura de datos precalculada expuesta en la sección anterior. Recordemos que dicha estructura cuenta con la información de 6076 puntos de muestreo en el espacio de parámetros α-β, representada en la figura 4.1, dando lugar a una triangulación de Delaunay compuesta de 11709 triángulos, representada en la figura 4.2. Necesitaremos extraer toda la información de Matlab para que el programa en C pueda manejar la malla de triángulos y las PDF α-estables precalculadas. Para ello, se recurre a diversas estrategias que permitan mejorar la implementación básica escrita en C.

En primer lugar, se crean dos tipos de estructuras que almacenan la información de un triángulo, *tri_grid*, y la información de una PDF, *pdf_grid*, representadas en la figura 4.4. Estas estructuras permiten combinar bajo un mismo tipo, datos de distintos tipos y manejarlos como uno solo.

```
typedef struct {
    double x;
    double y;
} Point;


typedef struct {
    Point v1;
    Point v2;
    Point v3;
    double invDet;
} tri_grid;


typedef struct {
    int tam;
    double xtable[MAX_PDF]; Máxima longitud de los vectores de coordenadas x-y
    double ytable[MAX_PDF]; de las PDF precalculadas: 210
} pdf_grid;
```

Figura 4.4- Estructuras *tri_grid* y *pdf_grid*.

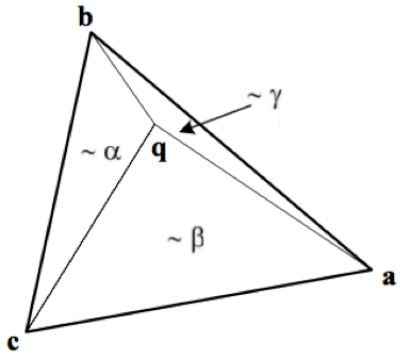
Para almacenar toda la información precalculada con la que contamos se declara una variable global, *GRID_TRI*, que contiene 11709 estructuras tipo *tri_grid* y una variable global, *GRID_PDF*, de 6076 estructuras tipo *pdf_grid*. En ambos casos, la declaración como variable global facilita que todas las funciones que hacen uso de ella comparten una copia única de esa variable.

Para la malla de triángulos se parte de la declaración de una matriz de 11709x3 enteros (*DelaunayTri*), cuyos elementos contienen el índice del vector de los 6076 puntos α-β al que corresponde cada vértice del triángulo en cuestión, incluido todo esto en los ficheros cabecera *tri_precal.h* y *grid_ab_precal.h*. Para la información de las PDF estándar disponibles en la estructura precalculada, se realiza un preprocesado en Matlab que escribe en un fichero binario el número de elementos, el vector de coordenadas x y el vector de coordenadas y de cada función. Esta disposición estratégica del contenido del fichero binario permite que, con la función *fseek*, podamos realizar un desplazamiento por las posiciones del fichero e ir volcando su contenido en

la variable global. La lectura de este fichero (*grid_pdf.bin*) se realiza inicialmente y de forma única, y todos los valores leídos se almacenan en una estructura única accesible a través de las distintas subrutinas de la implementación. De la formación de las variables *GRID_TRI* y *GRID_PDF* se encargan las funciones *create_global_grid_tri* y *create_global_grid_pdf*.

A continuación, siguiendo los mismos pasos que se han diferenciado en el análisis del algoritmo en la sección anterior, se detallan las funciones encargadas y las decisiones que se han tomado al desarrollarlas en lenguaje C:

- **Paso 1: Búsqueda del triángulo que contiene el punto α-β y cálculo de las coordenadas baricéntricas** (función *tri_location*).
  La coordenadas baricéntricas de un punto q son proporcionales a las áreas de los subtriángulos representados en la figura 4.5, y se definen como:



$$\mathbf{q} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$
$$\alpha = \mathcal{A}(\mathbf{bcq})/\mathcal{A}(\mathbf{abc})$$
$$\beta = \mathcal{A}(\mathbf{caq})/\mathcal{A}(\mathbf{abc})$$
$$\gamma = \mathcal{A}(\mathbf{abq})/\mathcal{A}(\mathbf{abc})$$
$$\alpha + \beta + \gamma = 1$$

Figura 4.5- Coordenadas baricéntricas (extraído de [30]).

Un punto con coordenadas baricéntricas (α, β, γ) está dentro del triángulo Δabc si y sólo si $0 \leq \alpha, \beta, \gamma \leq 1$ o alternativamente, si y sólo si $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$ y $\alpha + \beta \leq 1$. Por otro lado, se ha estudiado que el área de un triángulo Δabc se puede calcular de forma robusta y eficiente [30] mediante la evaluación del determinante:

$$DET(\Delta\text{abc}) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = \begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{vmatrix} = 2\mathcal{A}(\Delta\text{abc}) \qquad (4.4)$$

Conforme a esto, el objetivo de la función *tri_location* es encontrar el triángulo de la variable global *GRID_TRI* que contiene el punto solicitado de parámetros α, β de la distribución. Buscando reducir el número de operaciones lo máximo posible para optimizar el rendimiento de la búsqueda del triángulo envolvente, se siguen dos estrategias, como se puede deducir en la figura 4.6:

- Se precalcula para cada triángulo de la malla el valor de la inversa de su determinante siguiendo la expresión (4.4), ya que éste depende de los valores de sus vértices y no van a sufrir variaciones en toda la ejecución. Esto se refleja en la figura 4.4, incluyendo la estructura *tri_grid* un campo para almacenarlo.

- Se implementa el cálculo de las coordenadas baricéntricas con sentencias *if* anidadas, es decir, si y solo si la primera coordenada baricéntrica cumple la condición de pertenecer al intervalo [0,1], se procede a calcular la segunda coordenada baricéntrica, y lo mismo con la tercera.

```
Para i = 0 Hasta DT_NUM_TRI  Número de triángulos de la malla precalculada: 11709
q: punto de parámetros α-β a localizar en la malla
invDet: valor precalculado DET(Δabc) para cada triángulo
  BaryCoord1 = DET(Δbcq) * GRID_TRI[i].invDet
    Si 0 ≤ BaryCoord1 ≤ 1
      BaryCoord2 = DET(Δcaq) * GRID_TRI[i].invDet
        Si 0 ≤ BaryCoord2 ≤ 1
          BaryCoord3 = DET(Δabq) * GRID_TRI[i].invDet
            Guardar i como índice del Δ que contiene a q
            Guardar coordenadas baricéntricas
            Triángulo encontrado -> Parar búsqueda
        Fin Si
    Fin Si
Fin Para
```

Figura 4.6- Seudocódigo de la búsqueda del triángulo envolvente.

Lo que se persigue con esta anidación es el ahorro de cálculos innecesarios para toda la malla precalculada de triángulos, ya que para la obtención de la PDF solo interesan las coordenadas baricéntricas del punto (α, β) respecto al triángulo envolvente en cuestión.

- **Paso 2: Obtención de las coordenadas x-y de las PDF α-estables precalculadas en los tres vértices del triángulo envolvente.**
  Localizado ya el índice del triángulo que contiene el punto (α, β) requerido, este paso se implementa como simples accesos a través de dicho índice a las variables y estructuras estratégicamente precargadas, como se puede apreciar en el diagrama de la estructura del programa de la figura 4.8.

- **Paso 3: Desnormalización de las tres densidades α-estables de los vértices** (función *unnormalize_stored_pdf*).
  Se aplica a la PDF estándar precalculada de cada vértice el ajuste visto en (4.2).

- **Paso 4: Interpolación cúbica de las PDF resultantes de cada vértice sobre el vector x** (función *cubic_interp*).
  Para la interpolación cúbica se acude a funciones matemáticas externas que resuelven estas operaciones con la máxima eficiencia, como es el caso de la librería GSL (GNU Scientific Library) escrita en C que proporciona diferentes métodos de

interpolación: lineal, polinómica (la cual introduce grandes oscilaciones), e interpolación por medio de spline, entre otros. El método escogido ha sido este último (*gsl_interp_cspline*) [31], ya que el valor interpolado en un punto de consulta se basa en una interpolación cúbica, como en el código de partida.

- **Paso 5: Cálculo de la PDF resultante.**
  Habiéndose obtenido en el paso 1 las coordenadas baricéntricas del punto (α, β) solicitado respecto al triángulo que lo contiene, y las densidades α-estables interpoladas en x para cada uno de los tres vértices en el paso 4, se halla la PDF resultante sobre el vector x aplicando el sumatorio ponderado visto en (4.3).

- **Paso 6: Optimización de la función log-verosimilitud.**
  En la implementación del MLE, definido en (4.1), se busca que el algoritmo de optimización de la función no requiera costosos cálculos. El algoritmo SQP empleado en el método de estudio por la función *fmincon* de Matlab, se basa en aproximaciones del Hessiano de la función mediante evaluaciones del gradiente [32]. Con el objetivo de reducir la carga computacional que implica un gradiente, se pretende utilizar un método más eficiente mediante la evaluación directa de la función, evitando así la problemática asociada al cálculo de derivadas.

  Este es el caso del método Nelder-Mead [33], que optimiza una función de N variables a través de la comparación de los valores que toma al evaluarse en los N+1 vértices de un simplex[1], el cual se va modificando de forma que en los nuevos vértices la función objetivo mejore, es decir, disminuya su valor. El simplex se va deformando y variando en base a cuatro operaciones que se muestran en el ejemplo del tetraedro de la figura 4.7: reflexión, expansión, contracción y contracción multi-dimensional. Una secuencia apropiada de estas transformaciones converge a un mínimo de la función. Esto puede parecer no ser muy eficiente en términos del número de evaluaciones de la función que requiere [30], pero es con frecuencia el mejor método si el objetivo es conseguir un funcionamiento rápido para un problema cuya carga computacional es pequeña.

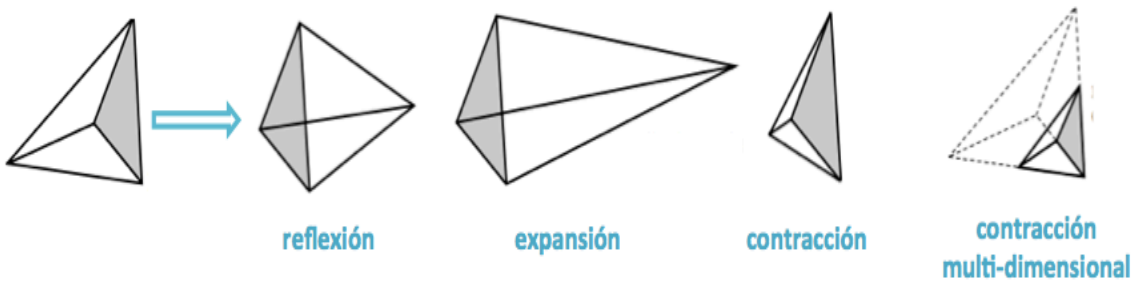


Figura 4.7- Estrategias del método Nelder-Mead en tres dimensiones.

[1] Un simplex es la figura geométrica que, en N dimensiones, consiste de N+1 puntos o vértices y todos se encuentran interconectados por segmentos, caras poligonales, etc. En dos dimensiones, un simplex es un triángulo. En tres dimensiones, es un tetraedro.

El algoritmo de optimización ha de acotarse dentro de todos los posibles valores que pueden tomar los parámetros en el espacio de cuatro dimensiones $\{\alpha, \beta, \sigma, \mu_0\}$ para asegurar la convergencia al mínimo deseado. Además, dependiendo del punto donde se inicialice el algoritmo se alcanzará el mínimo global o un mínimo local. Algo que se ha observado en el uso de la función *fmincon* y el método de búsqueda no lineal SQP es la importancia de una buena estimación inicial. En las pruebas realizadas, se ha observado que la estimación es especialmente sensible al valor inicial de μ_0 y σ.

Conforme a esto, se opta por tomar como referencia las rutinas presentes en la librería *libstable* [11], desarrollada en la Universidad de Valladolid, para el tratamiento de distribuciones α-estables. En concreto, ésta cuenta con la implementación de un MLE (función *stable_fit_mle*) que, realizando una transformación previa a los parámetros, logra interesantes resultados de precisión y rendimiento. La maximización la realiza mediante el método Nelder-Mead proporcionado por la librería GSL (*gsl_multimin_fminimizer_nmsimplex2rand*), y utiliza como aproximación inicial el estimador ideado por McCulloch (función *stable_fit_init*) que estima los cuatro parámetros de la distribución α-estable a partir de percentiles de los datos y de valores tabulados [34]. Pese a que en el algoritmo que se ha tomado como punto de partida cuenta con una estimación inicial diferente que se puede consultar en [13], tomar en las pruebas realizadas la estimación de McCulloch como punto inicial de la búsqueda ha dotado a la herramienta de una mejor convergencia en el MLE y de mayor velocidad, ya que este estimador tiene un costo computacional muy bajo.

Con el fin de esclarecer el procedimiento expuesto en esta sección, se representa en la siguiente figura 4.8 un esquema del flujo de datos y de instrucciones que se llevan a cabo en cada uno de los pasos de la herramienta implementada para la estimación de los parámetros de una distribución α-estable.

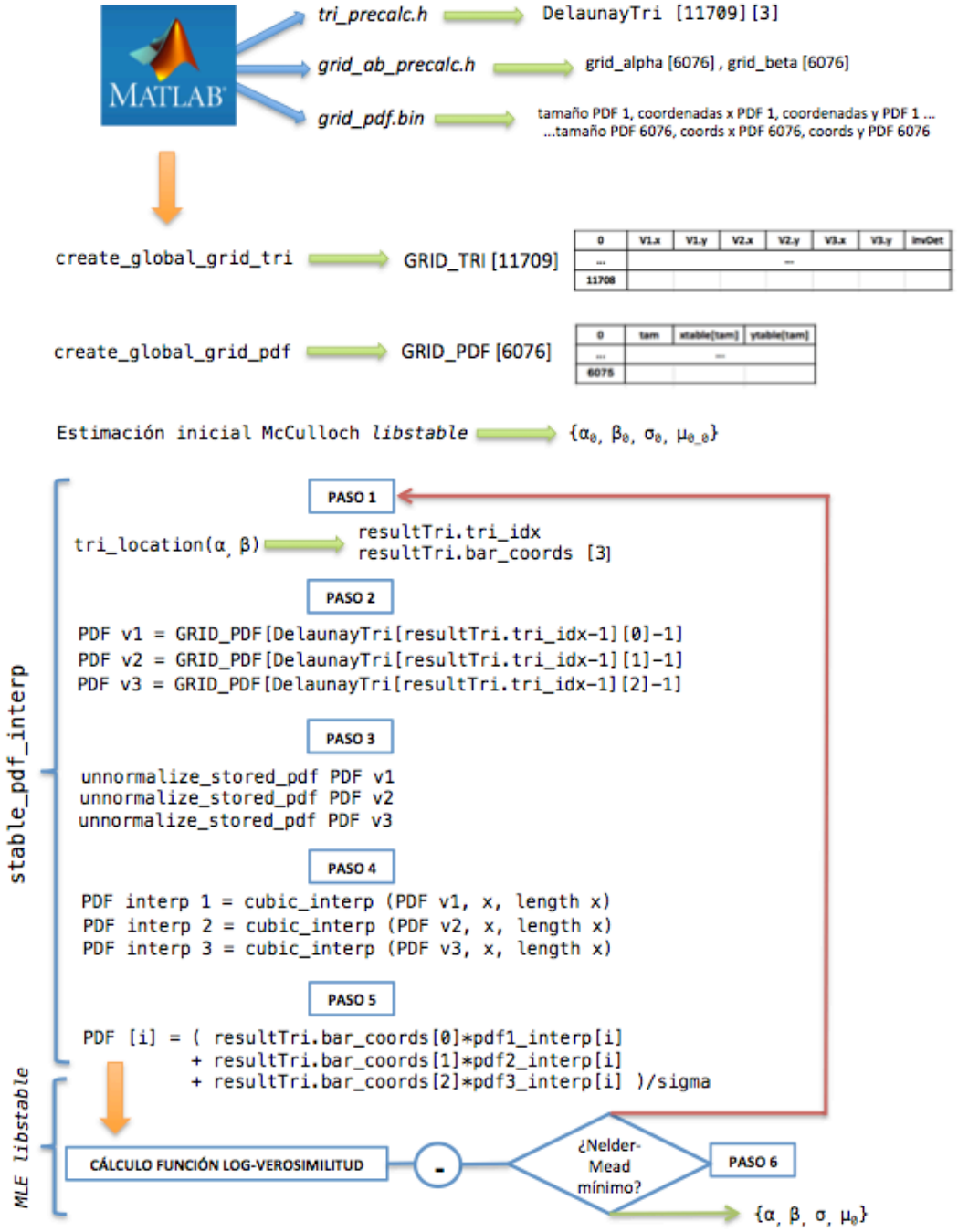


Figura 4.8- Esquema del desarrollo serie de la herramienta.

4.4. Evaluación

Para la evaluación del software desarrollado, se realiza en primer lugar una validación de los resultados de la función encargada de la obtención de una PDF concreta (*stable_pdf_interp*) partiendo de la malla construida y la comparación de los tiempos que implica dicho algoritmo. El rendimiento de esta función va a depender del valor de los parámetros, según su posición en la triangulación, y del número de muestras del vector donde se desea evaluar la función.

En la tabla 4.1 se muestran los resultados obtenidos para diferentes valores de los parámetros fijados dentro de los intervalos $\alpha \in [0.4, 2]$, $\beta \in [0,1]$, $\sigma = 1$, $\mu_0 = 0$, y un vector equiespaciado de 1000 muestras $x \in (-100, 100)$. Se analiza el tiempo consumido, como resultado medio de 100 experimentos, en los pasos 1 y 4 del algoritmo desarrollados en la sección anterior, es decir, en la búsqueda del triángulo envolvente y en la interpolación cúbica de las PDF desnormalizadas de cada vértice sobre el vector x, respectivamente.

Las comparaciones se han realizado con el código de Matlab que se ha tomado como punto de partida para asegurar que se obtienen los mismos resultados para cada etapa, ejecutados ambos en un Intel Core i7-6700HQ con 4 núcleos a 2.60 GHZ, y compilado el programa en C mediante GCC 5.4.0 [35] con las opciones de optimización -O3 -march=native, y enlazándolo con la librería GSL versión 2.3, y el código de referencia, se ha ejecutado en la versión 2016a de Matlab. Pese a que la comparativa no es del todo justa dada la diferencia entre el consumo y la precisión de los comandos *tic* y *toc* (ms) de Matlab, y la función *gettimeofday* utilizada en C para las mediciones de tiempo, esto puede servir como primera impresión de la eficiencia del código implementado.

Como se puede ver en los ejemplos de la tabla 4.1, en el lenguaje C sin paralelizar se han obtenido reducciones del tiempo total respecto a los tiempos de Matlab, observándose especialmente esta reducción en la interpolación cúbica (paso 4). Respecto a la búsqueda del triángulo (paso 1), se puede ver que los tiempos de *pointLocation* se mantienen más o menos constantes independientemente de los valores a localizar, mientras que en nuestra implementación el tiempo consumido es proporcional a la posición del triángulo en la malla. Como se ha comentado anteriormente, el algoritmo de búsqueda de esta función de Matlab no está disponible, y replicarlo de forma idéntica en C no ha sido posible, pero en media, se han obtenido tiempos similares.

Por otro lado, el uso de las coordenadas baricéntricas como pesos gestiona automáticamente los casos de puntos que caen exactamente sobre un vértice del triángulo, porque solo el vértice coincidente es distinto de cero, independientemente del triángulo elegido. En nuestro código la búsqueda finaliza en el primer triángulo que encuentra, mientras que en Matlab la función *pointLocation* parece seguir la búsqueda por toda la malla de triángulos, quedándose con el último que lo cumple. Para el caso gaussiano ($\alpha = 2$), dado que antes de comenzar la búsqueda del triángulo se fuerza a que

β sea 0 porque en este caso su comportamiento no afecta, el triángulo que se obtiene en el desarrollo en C es independiente del valor de β.

α	β	Paso 1 (ms)	Paso 4 (ms)	Triángulo	Coordenadas Baricéntricas	Total (ms)	
0.4	0	0.04013	0.82269	4546	(0, 1, 0)	0.92913	**Matlab**
0.4	0.5	0.02694	0.79790	1282	(1, 0, 0)	0.91617	
0.4	1	0.02583	0.76077	326	(0, 1, 0)	0.85501	
0.85	0	0.02270	0.77993	9487	(1, 0, 0)	0.86835	
0.85	0.5	0.02348	0.77758	8874	(0, 0, 1)	0.86157	
0.85	1	0.02871	0.76776	8706	(1, 0, 0)	0.85202	
1.55	0	0.02229	0.76567	8896	(0, 0.75, 0.25)	0.85302	
1.55	0.5	0.02257	0.76752	9410	(0.75, 0.25, 0)	0.86364	
1.55	1	0.02199	0.76346	8701	(0, 0.75, 0.25)	0.84402	
2	0	0.02384	0.77498	9465	(0, 0, 1)	0.83941	
2	0.5	0.02297	0.76879	8921	(0, 1, 0)	0.84057	
2	1	0.02196	0.75536	2146	(0, 0, 1)	0.83151	
0.4	0	0.05683	0.05612	4546	(0, 1, 0)	0.11810	**C**
0.4	0.5	0.01522	0.05329	1282	(1, 0, 0)	0.07279	
0.4	1	0.00318	0.04485	326	(0, 1, 0)	0.05213	
0.85	0	0.10944	0.04977	9443	(1, 0, 0)	0.16386	
0.85	0.5	0.02600	0.04775	2305	(0, 1, 0)	0.07792	
0.85	1	0.08694	0.03788	8706	(1, 0, 0)	0.12887	
1.55	0	0.02226	0.03678	2319	(0.25, 0.75, 0)	0.06275	
1.55	0.5	0.08719	0.03717	8918	(0.75, 0, 0.25)	0.12776	
1.55	1	0.08440	0.03652	8701	(0, 0.75, 0.25)	0.12458	
2	0	0.08162	0.03497	8754	(1, 0, 0)	0.12009	
2	0.5	0.08053	0.03410	8754	(1, 0, 0)	0.11820	
2	1	0.07950	0.03397	8754	(1, 0, 0)	0.11678	

Tabla 4.1- Comparación entre el algoritmo de obtención de la PDF implementado en Matlab [27] y en C evaluada en $x \in (-100, 100)$.

Por otro lado, con respecto al programa de Matlab, se ha hecho uso de otro método de estimación, partiendo de una estimación inicial y una optimización de la función log-verosimilitud (paso 6) aparentemente más eficientes disponibles en la librería *libstable*. Por eso, es necesario también validar el algoritmo de estimación midiendo el sesgo. Con este fin, se generan valores aleatorios α-estables con la rutina *stable_rnd* de la librería *libstable* de parámetros $\alpha \in [0.5, 2]$ y $\beta \in [-1, 1]$, en pasos de 0.05, y $\sigma = 1$ y $\mu_0 = 0$, y se procede a estimar sus valores a partir de las muestras generadas. Se han generado muestras con un tamaño de 1000, repitiendo cada experimento 100 veces. En la figura 4.9 se muestran los resultados del sesgo obtenido para cada parámetro en función de los diferentes valores de α y β, calculándose dicho sesgo como la diferencia entre el valor del parámetro obtenido tras la estimación y el parámetro con el que se han generado las muestras. En general, se puede apreciar que el comportamiento del estimador es

correcto, ya que presenta un sesgo bajo en la mayoría de los casos. Los extremos del espacio de parámetros α-β y el parámetro β muestran los peores resultados, apreciándose un mayor sesgo. Sin embargo, cuando α se aproxima a 2 (caso Gaussiano) la influencia del parámetro β en la forma de la distribución es nula, por lo que no resulta un caso problemático.

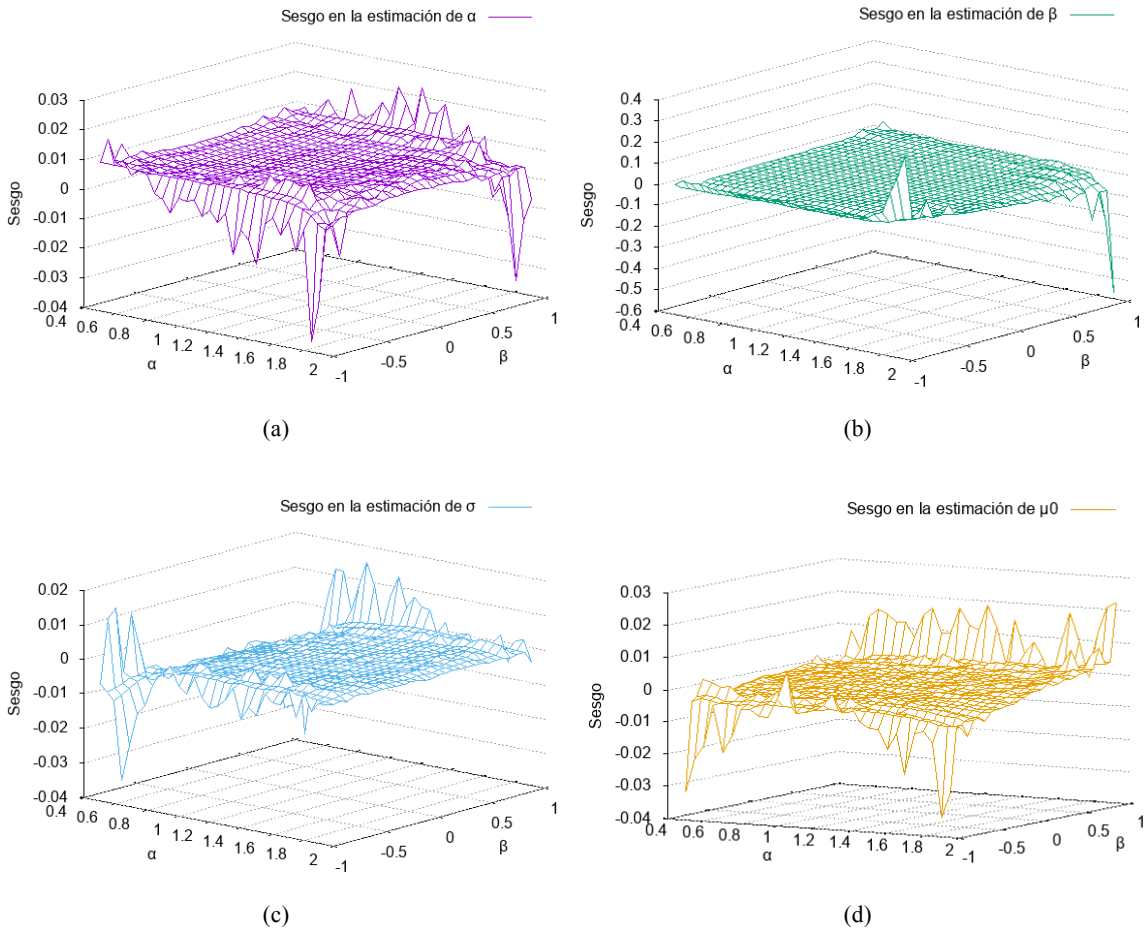


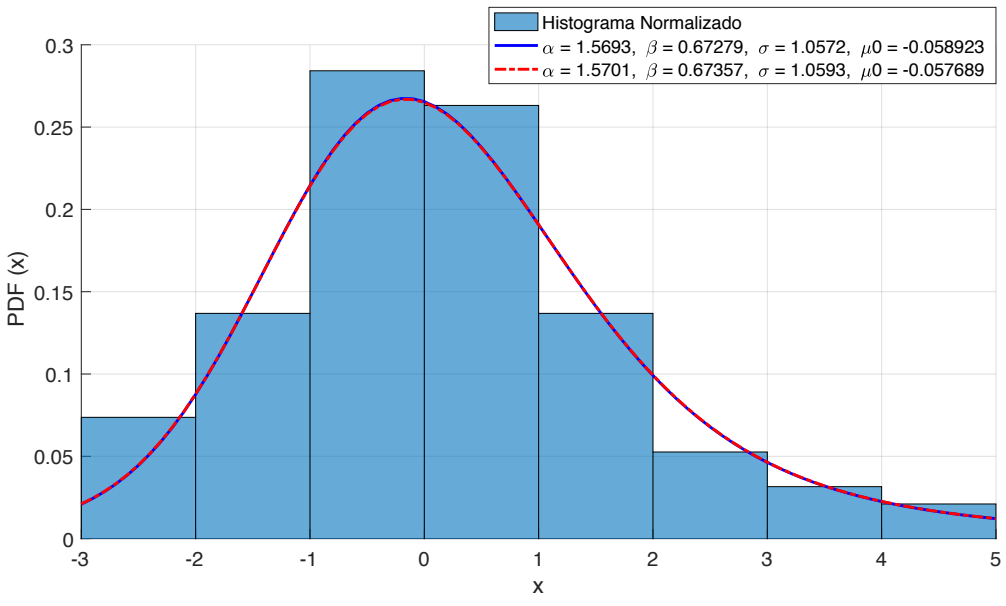
Figura 4.9- Sesgo de los parámetros α (a), β (b), σ (c) y μ_0 (d) obtenidos a partir de 1000 muestras sintéticas.

Por último, se quiere evaluar la mejora temporal en la estimación respecto al código de Matlab. Para ello, se han generado 100, 1000 y 10000 muestras α-estables aleatorias, como en el caso anterior, a partir parámetros $\alpha \in [0.5, 1.5]$, $\beta \in [0, 1]$, $\sigma = 1$ y $\mu_0 = 0$, y se ha estimado su valor, repitiendo cada experimento 100 veces, mostrándose los resultados medios obtenidos. Se puede observar en la tabla 4.2 que los tiempos para la estimación de los parámetros en el desarrollo en C, como cabía esperar, son más bajos, reduciéndose la ganancia obtenida según aumenta el número de muestras. Hay que tener en cuenta en estos resultados la aleatoriedad existente en la generación de las muestras, y las restricciones y convergencia de cada optimizador, que pueden no haber alcanzado un mínimo global según la aproximación inicial de la que parten.

		N = 100 muestras		N = 1000 muestras		N = 10000 muestras	
α	**β**	**Estimación Matlab (ms)**	**Estimación C (ms)**	**Estimación Matlab (ms)**	**Estimación C (ms)**	**Estimación Matlab (ms)**	**Estimación C (ms)**
0.5	0	251.7517	16.6493	294.8409	31.7032	455.2161	114.4741
0.5	0.5	230.8133	20.3836	248.6464	32.3254	426.6794	137.5329
0.5	1	234.0094	24.8461	301.2995	44.1466	456.8699	271.5081
0.75	0	220.7986	15.0898	243.2288	22.2294	425.0195	103.6211
0.75	0.5	212.4591	15.8053	229.6141	27.1882	364.5942	119.2500
0.75	1	205.1053	19.7698	264.0732	35.2864	296.1673	231.7387
1.25	0	156.9877	10.5044	213.7619	18.6342	309.8475	80.3046
1.25	0.5	142.5128	13.8303	191.1137	20.9275	301.8435	98.1120
1.25	1	105.9063	19.9779	145.9302	34.7949	216.4566	194.5313
1.5	0	119.4136	15.3382	152.8721	20.1564	203.7799	98.5564
1.5	0.5	116.9432	16.9262	146.5194	30.6120	200.1985	129.0430
1.5	1	110.6133	22.6337	96.2759	47.2837	174.5858	112.8489

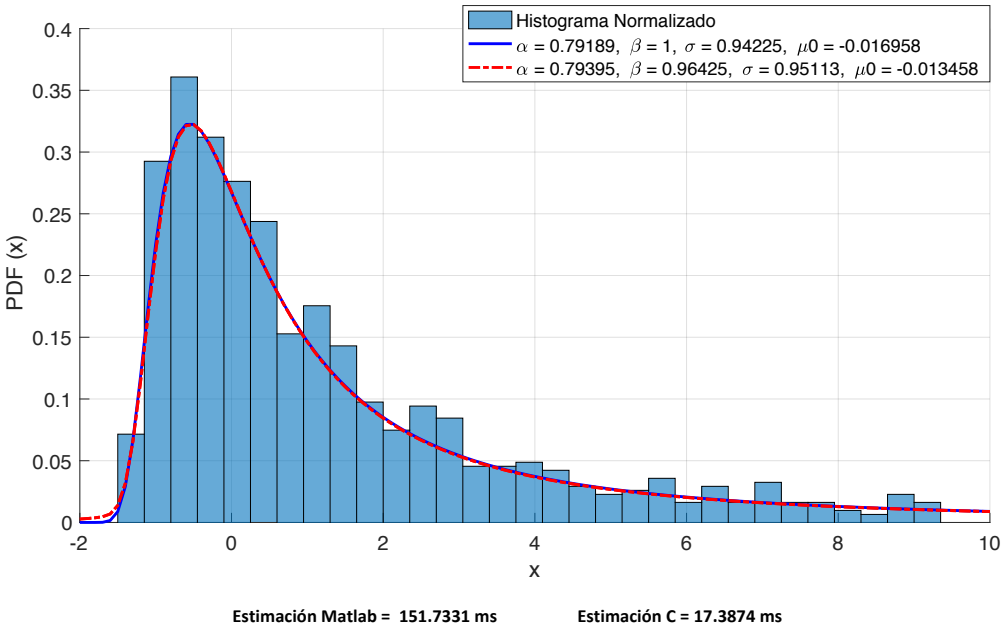
Tabla 4.2- Comparación entre el algoritmo de estimación implementado en Matlab [27] y en C.

En la figura 4.10 se representan tres ejemplos concretos de estos casos generados, apreciándose gráficamente que pese a no estimar parámetros idénticos partiendo del mismo conjunto de muestras, la PDF obtenida a partir de ellos en la implementación en C presenta una similitud bastante razonable con respecto a la conseguida con el algoritmo desarrollado en Matlab, además de una mejora sustancial en el rendimiento de la estimación.

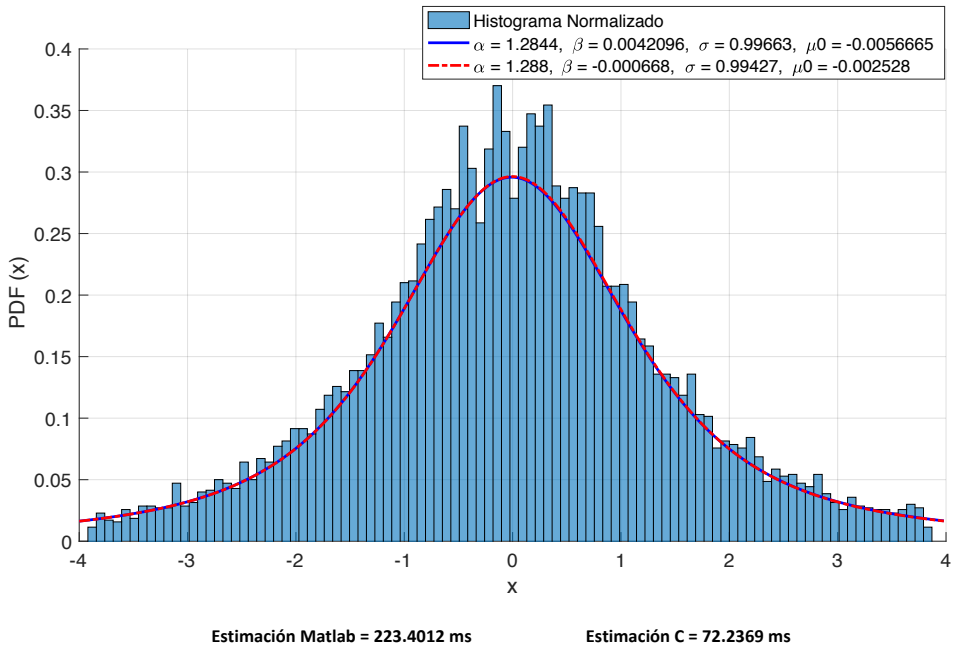


Estimación Matlab = 110.0536 ms **Estimación C = 14.6220 ms**

(a) 100 muestras generadas con $\alpha = 1.5$, $\beta = 0.5$, $\sigma = 1$ y $\mu_0 = 0$.



(a) 1000 muestras generadas con $\alpha = 0.75$, $\beta = 1$, $\sigma = 1$ y $\mu_0 = 0$.



(c) 10000 muestras generadas con $\alpha = 1.25$, $\beta = 0$, $\sigma = 1$ y $\mu_0 = 0$.

Figura 4.10- PDF de parámetros estimados aplicando el código de Matlab [27] (gráficas azules continuas) y la implementación en C (gráficas rojas discontinuas) a partir de muestras generadas aleatoriamente.

4.5. Conclusiones

A lo largo de este capítulo se ha expuesto el procedimiento seguido para conseguir una herramienta lo más rápida posible que permita una precisa estimación de los parámetros de una distribución α-estable que, según se veía en el capítulo anterior, modele el comportamiento del tiempo de respuesta del servidor en una conexión TCP.

Se ha optado por el interesante método presente en [13] para una rápida estimación basada en la interpolación de valores precalculados, y a partir de la malla y el código disponible para Matlab [27], se ha procedido a su traspaso al lenguaje C, comprobando que se disponía de resultados correctos para cada etapa del código, y que las funciones desarrolladas en C han logrado reducciones de tiempo respecto a Matlab.

En este proceso, ha sido de especial utilidad la librería *libstable* [11]. En concreto, su estimador de máxima verosimilitud, el cual se ha modificado para que en vez de invocar a la rutina encargada de la evaluación de la PDF a partir de métodos de integración numérica, llame a la función que obtiene la PDF a través del método descrito. El elevado costo computacional de la evaluación numérica de la PDF en *libstable* mediante la aplicación de las ecuaciones de Nolan, hace que la maximización en el espacio de parámetros de cuatro dimensiones sea lenta, incluso cuando se restringe la búsqueda a distribuciones estables estandarizadas. Sin embargo, aplicando el método visto en esta sección, esto deja de ser problemático incluso para valores de α próximos a 0 en los que el cálculo numérico de la PDF es más lento y menos preciso. Esto supone una ventaja en el campo del análisis de tráfico de red, ya que se tienen evidencias de que los datos muestran valores de α variando ampliamente entre 2 y 0,5 (o incluso menos) dependiendo del número de fuentes agregadas y la intensidad del tráfico [1].

La implementación en C permite alejarse de un mero prototipo, además de que los resultados obtenidos en cuanto a precisión y rendimiento motivan su uso como herramienta de análisis de tráfico de red, ya que el aumento de eficiencia que se obtiene al pasar de Matlab a C es fundamental. Aún así, hoy en día puede resultar insuficiente para el análisis en tiempo real, se prevé que paralelizar este código permitirá alcanzar resultados razonables con tiempos de ejecución lógicos en el ámbito del tráfico de red.

Capítulo 5. Aceleración paralela del algoritmo

5.1. Introducción

Una vez que se cuenta con un código en C que permite la estimación precisa de los parámetros de las distribuciones α-estables, en este capítulo se consideran sus limitaciones en cuanto a eficiencia computacional, y se plantean las posibles vías de paralelismo para poder alcanzar tiempos de ejecución que se adapten al análisis en tiempo real de las muestras del tiempo de respuesta del servidor en una conexión TCP, y poder extraer conclusiones de la evolución de dichos parámetros estimados.

Gracias al lenguaje C, junto con la optimización preliminar que proporcionan las opciones del compilador GCC, se ha visto en el capítulo anterior una importante reducción de los tiempos de ejecución comparado con Matlab. Aun así, puede resultar interesante conseguir mejores tiempos para su uso en el análisis de tráfico de red, ya que dadas las exigencias de las redes actuales, su caracterización debe hacerse rápidamente para detectar posibles anomalías en la red con la suficiente antelación. Por ello, se plantea portar el código implementado para que pueda ser ejecutado por diferentes hilos, obteniendo así una mejora en el tiempo de ejecución acorde con los núcleos disponibles.

Las dos tareas más importantes en el diseño de un algoritmo para su ejecución en paralelo son la división del problema a resolver en subproblemas y su asignación a cada uno de los procesadores disponibles. Esta es la tarea de descomposición [36], y puede ser por tareas o por datos. La descomposición por tareas implica asignar a cada núcleo del procesador una tarea que pueda ejecutarse de manera simultánea con otras. Por otro lado, la paralelización por datos genera tareas en función de los datos con los que cada una de ellas va a trabajar y cuyo resultado es independiente del resto de los datos. Generalmente, las tareas en este tipo de paralelización son las mismas en los distintos núcleos, pero aplicadas a distintas porciones de datos. La elección del tipo de paralelización está casi siempre determinada por la naturaleza de los algoritmos a paralelizar, como se verá en este capítulo.

Por tanto, a la hora de paralelizar, es imprescindible analizar el código implicado para poder explotar al máximo sus posibilidades y lograr un rendimiento superior a la ejecución en serie. Además, cualquier modificación cuando se lleva a cabo un proceso de paralelización, debe ser comprobada rigurosamente para ver que los resultados, tanto de tiempos de ejecución como, sobre todo, el resultado final del cálculo, no se vean afectados. Por este motivo, a lo largo de todo este capítulo, se tomará como solución de referencia en los experimentos realizados un mismo conjunto de parámetros. Se seleccionan unos valores que se encuentren el un triángulo lo suficientemente alejado de los valores extremos de la rejilla α-β (ver figura 4.2), para aislar los problemas de alto sesgo y convergencia del algoritmo en dichos extremos:

$$\alpha = 1.5,\ \ \beta = 0.5,\ \ \sigma = 1,\ \ \mu_0 = 0 \tag{5.1}$$

5.2. Tiempo y estructura del funcionamiento en ejecución

Un primer paso en el análisis del código que se desea paralelizar es un estudio de los tiempos de ejecución. Para ello, se han evaluado los resultados de la estimación de los parámetros a partir de un conjunto de muestras sintéticas generadas con el mismo procedimiento que en el capítulo anterior, con los parámetros (5.1) y aumentando su tamaño, obteniéndose los resultados que se muestran en la tabla 5.1. Las ejecuciones mostradas en esta tabla, se han llevado a cabo, como en el capítulo anterior, en un Intel Core i7-6700HQ con 4 núcleos a 2.60 GHZ.

Tamaño muestral	Tiempo (ms)	Iter.	$\hat{\alpha}$	Δα	$\hat{\beta}$	Δβ	$\hat{\sigma}$	Δσ	$\hat{\mu}_0$	$\Delta\mu_0$
10^2	13.6369	78	1.5295	0.0295	0.3381	0.1619	1.0432	0.0432	-0.1441	0.1441
10^3	24.4848	47	1.4798	0.0202	0.5070	0.0070	1.0021	0.0021	0.0410	0.0410
10^4	149.5662	54	1.5088	0.0088	0.5152	0.0152	1.0112	0.0112	0.0070	0.0070
10^5	661.8559	24	1.5017	0.0017	0.5047	0.0047	0.9941	0.0059	0.0001	0.0001

Tabla 5.1- Resultados de la estimación para conjuntos de datos de diferentes tamaños.

Podemos inferir de la tabla 5.1 que el método de estimación se vuelve computacionalmente más demandante para conjuntos de muestras de grandes tamaños, como puede ocurrir en la magnitud que deseamos modelar al analizar un servidor muy activo, lo que hace aun más necesaria esta optimización.

Un segundo paso en el análisis del código es disponer de herramientas que ayuden a profundizar en la programación, como los analizadores de código (*profilers*). Este estudio del comportamiento del programa al ser ejecutado, conocido como perfilado o *profiling*, va a proporcionar los porcentajes de cómputo que implican cada una de las funciones en la ejecución total del programa y poder así analizar las funciones que llevan el mayor peso en cuanto a tiempo de ejecución se refiere.

La ley de Amdahl [37] plantea que la máxima aceleración o *speed-up* (S) que se puede alcanzar en una aplicación completa viene acotada por la parte secuencial o no paralelizable que contenga, y se enuncia de la siguiente forma:

$$S = \frac{1}{(1-F) + \frac{F}{A}} \tag{5.2}$$

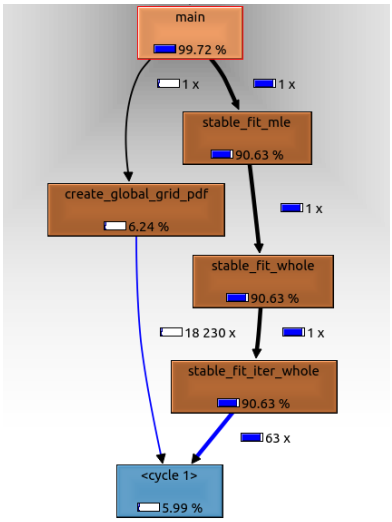
Donde F es la fracción de tiempo del tiempo total de la ejecución que consume el algoritmo a paralelizar, y A es la mejora parcial aplicada durante esa fracción de tiempo F. Es decir, si por ejemplo un 30% del código no puede ser paralelizado, el *speed-up* máximo alcanzable es debido al 70% restante. Por ello, aunque sea deseable conseguir paralelizar la mayor parte del código, debe considerarse el esfuerzo en conseguir una mejora apreciable y la mejora real que conseguimos. Conocer en qué rutina se emplea más tiempo permite hacer una idea de dónde focalizar los esfuerzos en la paralelización.

Por otro lado, la ley de Gustafson [38] proporciona otro punto de vista al propuesto por Amdahl, ya que éste supone una solución a un problema con un tamaño determinado, mientras que Gustafson propone no limitar el tamaño, y aprovechar la parte paralelizable para alcanzar una aceleración significativa que varía linealmente con el número de procesadores, y que así sea posible la solución de mayores problemas en el mismo intervalo de tiempo, utilizando al máximo la capacidad disponible del equipo donde se ejecute.

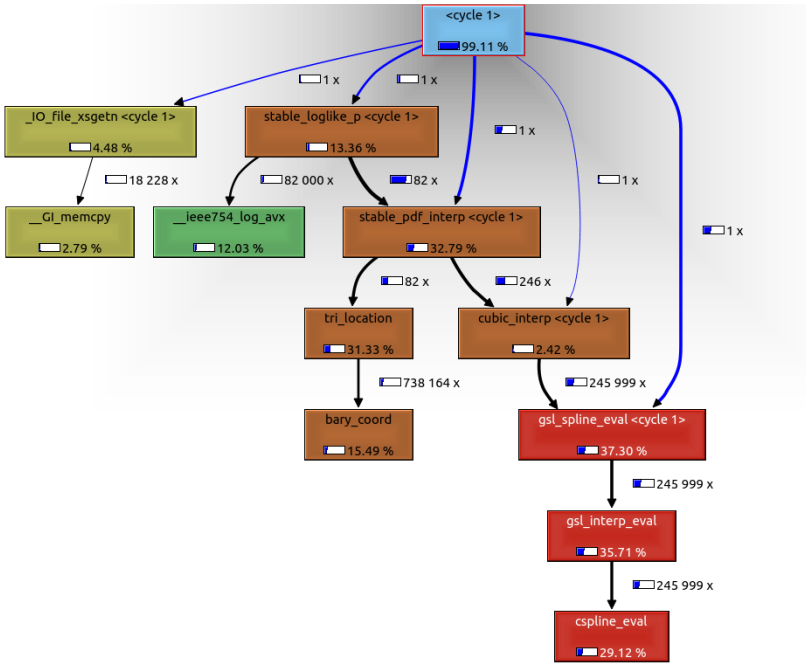
En definitiva, el necesario perfilado del programa lo llevamos a cabo con las herramientas Valgrind-Callgrind [39] y Kcachegrind [40] para visualizar sus resultados. El estudio se ha realizado para diferentes escenarios de tamaños de muestras, y los resultados de las llamadas con mayores porcentajes dentro del proceso de la estimación, se recogen en la tabla 5.2. En la figura 5.1 se muestra como ejemplo el gráfico obtenido tras el perfilado para el caso de la estimación de parámetros de 1000 muestras.

	100 muestras		**1000 muestras**		**10000 muestras**		**100000 muestras**	
Función	**%**	**Número de llamadas**	**%**	**Número de llamadas**	**%**	**Número de llamadas**	**%**	**Número de llamadas**
stable_fit_mle	87.51%	1	90.63%	1	91.07%	1	90.82%	1
stable_pdf_interp	57.74%	138	32.79%	82	7.26%	97	2.56%	45
tri_location	56.75%	138	31.33%	82	5.21%	97	0.52%	45
bary_coord	28.08%	772958	15.49%	738164	2.58%	881774	0.26%	401301
stable_loglike_p	3.89%	138	13.36%	82	22.01%	97	22.43%	45
gsl_spline_eval	11.02%	41400	37.30%	246000	61.60%	2910000	62.83%	13500000

Tabla 5.2- Resultados del perfilado recogido por Valgrind para la estimación de parámetros de muestras de diferentes tamaño.



(a) Main



(b) Ciclo de búsqueda del mínimo

Figura 5.1- Perfilado obtenido para la estimación de parámetros de 1000 muestras.

De estos resultados recién expuestos se puede deducir que la función que interesa paralelizar es *stable_fit_mle*, ya que consume gran parte del tiempo del programa y, sustituyendo en la ley de Amdahl (5.2) F por el tiempo de cómputo de dicha parte del código y asumiendo $A \rightarrow \infty$, se deduce que puede alcanzarse una aceleración máxima teórica aproximada de x10, la cual en la práctica podría ser inalcanzable debido a las limitaciones de la arquitectura.

Dicha estimación basada en la máxima verosimilitud (4.1), consta de un proceso iterativo, y en dicho ciclo de la búsqueda del mínimo, dependiendo del número de muestras, va a merecer la pena focalizar los esfuerzos de la paralelización en una zona u en otra, por lo que viendo los resultados del perfilado, se centrarán los esfuerzos de paralelización sobre las subrutinas que han mostrado los mayores porcentajes. Dado que

el problema al que aquí se trata no tiene un tamaño determinado, sino que son el número de muestras a modelar y la precisión de la malla precalculada los que imponen el tamaño del mismo, resulta interesante paralelizar el estimador de máxima verosimilitud distinguiendo tres niveles posibles que se presentan en la siguiente sección, de forma que el rendimiento que obtengamos justifique la utilización de múltiples procesadores.

5.3. Fases de la implementación paralela con OpenMP

Los resultados del perfilado recogidos en la tabla 5.2, han permitido conocer las rutinas que consumen el mayor porcentaje del tiempo, dependiendo del número de muestras que se desean modelar. Por ello, basándose en ese análisis previo realizado, se proponen tres fases o niveles de posible paralelización:

1) Paralelizar la obtención de la PDF.
2) Paralelizar el cálculo de la verosimilitud.
3) Paralelizar el método de optimización.

Con la finalidad de centrarse en la propia problemática, un factor importante a considerar en la paralelización es la creación de un número de hilos no superior al provisto por el hardware disponible. Para paralelizar una determinada aplicación es importante diferenciar entre lo núcleos virtuales (tecnología *Hyper-Threading*) y los núcleos físicos que posee el procesador utilizado, ya que el hecho de utilizar más hilos de los que se dispone físicamente aumenta el costo computacional debido a que los hilos a nivel de software comparten una cantidad de recursos. Sin embargo, los núcleos físicos disponen de recursos propios y pueden ejecutarse de forma independiente y simultánea en cada núcleo, obteniendo así una paralelización efectiva.

A lo largo de esta sección, todas las pruebas han sido realizadas en un clúster, accesible mediante SSH (Secure Shell) desde equipos de la Universidad Autónoma de Madrid, cuyo nodo sobre el que se han ejecutado los experimentos es un equipo con dos procesadores AMD Opteron 6128 de 8 núcleos a 2 GHz, por lo que escala a un total de 16 núcleos, siendo éste el número máximo de hilos a utilizar. Acorde con el modelo de sistema paralelo de memoria compartida de este hardware, y por los motivos ya expuestos en el capítulo 2, en este trabajo la tarea de paralelización va a llevarse a cabo mediante OpenMP.

OpenMP emplea un modelo de ejecución en paralelo *fork-join*, de forma que se crean diferentes hilos que se ejecutan en los diferentes núcleos el código definido en la región paralela, para finalmente unirse de nuevo en la barrera de sincronización implícita al final de dicha región. La forma de iniciar una de estas regiones paralelas en un programa escrito en C es mediante la directiva *#pragma omp parallel*, que cuenta con diferentes construcciones para distribuir el trabajo entre los hilos, describiendo las que usaremos en este trabajo en la subsección correspondiente a cada fase donde se aplique. De esta forma, su uso se limita a añadir una opción para que el compilador

incluya las librerías de OpenMP (*-fopenmp* para el compilador GCC) y se puedan así ejecutar las zonas paralelas, y asignar a la variable de entorno *OMP_NUM_THREADS* el número de hilos que se desean crear.

5.3.1. Paralelización de la obtención de la PDF

El estimador de máxima verosimilitud puede requerir de muchas evaluaciones de la función de densidad de probabilidad, y por ello paralelizar su obtención puede llegar a permitir evaluar dicha función rápidamente. Recordemos que la función encargada de obtener la PDF es *stable_pdf_interp*, que según se veía en la tabla 4.1, independientemente del valor de los parámetros, la mayor parte del tiempo consumido en esta función es en los pasos 1 y 4 del algoritmo, es decir, en la búsqueda del triángulo que contiene el punto α-β y en la interpolación cúbica de las tres PDF de los vértices sobre el vector x donde se desea evaluar.

Cabe destacar, que este enfoque se partía ya de una malla de triángulos y de PDF estándar precalculadas, constando de un total de 11709 triángulos formados tras aplicar la triangulación de Delaunay a 6076 puntos α-β, cada uno con su PDF correspondiente. De esta forma, aparentemente el reparto del tiempo de ejecución entre *tri_location* y *cubic_interp* lo marca el número de muestras sobre las que evaluamos, de forma que si el número de muestras es muy elevado, será la interpolación cúbica la que más tiempo consuma, mientras que si se quiere modelar un conjunto de muestras más pequeño, en este caso, la búsqueda en 11709 triángulos acapararé más tiempo del total. Dados estos dos posibles enfoques en el problema a resolver, se plantea estudiar la mejora obtenida mediante la paralelización de la búsqueda del triángulo y de la interpolación cúbica de la PDF de cada vértice.

5.3.1.1. Búsqueda del triángulo

Recordemos que la función encargada de la localización del triángulo que contiene el punto α-β solicitado es la función *tri_location*, en la cual, tal como se veía en la figura 4.6, se realizan operaciones matemáticas sencillas (sumas, restas, multiplicaciones y divisiones) con los datos precalculados de la triangulación. Pese a que dichas operaciones sean sencillas y por lo tanto su consumo computacional no sea muy elevado, una posibilidad que aquí surge es que si se paraleliza esta búsqueda, la malla de puntos pueda ser más densa, es decir, pueda contar con más puntos α-β, y por tanto, con más triángulos, haciendo que el resultado final de la estimación sea más preciso y, gracias a esta paralelización de su búsqueda, más eficiente.

La paralelización de esta función se lleva a cabo mediante la directiva *#pragma omp for*, la cual distribuye las iteraciones del bucle *for* entre los hilos. Para permitir que el compilador ejecute esta optimización se debe evitar la dependencia de datos en el

bucle que impida la ejecución simultánea de las operaciones requeridas para el cálculo de las coordenadas baricéntricas. Tal y como se había almacenado estratégicamente en la versión serie la variable global *GRID_TRI*, el reparto de los triángulos a tratar entre los hilos disponibles va a permitir que cada hilo compruebe la pertenencia o no del punto α-β solicitado al interior de los triángulos que se le asigna, sin interferir en los triángulos de los demás hilos, y siendo solo un hilo el que lo encuentre.

	1 núcleo	**2 núcleos**	**4 núcleos**	**8 núcleos**	**12 núcleos**	**16 núcleos**
Tiempo medio *tri_location* (ms)	0.3112	0.1550	0.0791	0.0410	0.0319	0.0269
Speed-up tri_location	-	2.0077	3.9343	7.5902	9.7555	11.5688
Tiempo total estimación (ms)	61.5630	48.0285	41.7464	38.7218	38.1290	40.3819
***Speed-up* total**	-	1.2818	1.4768	1.5899	1.6146	1.5245

Tabla 5.3- Tiempos de ejecución tras paralelizar la búsqueda del triángulo para 1000 muestras.

En la tabla 5.3 se muestran los resultados obtenidos en la estimación de 1000 muestras. Se puede ver que pese a que se ha conseguido una mejora parcial de *tri_location* proporcional al número de núcleos, la mejora del estimador completo no ha sido así. En la tabla 5.1 se veía que esta rutina representaba un 31.33% del tiempo total, por lo que acorde a la mejora parcial obtenida, el *speed-up* máximo teórico sustituyendo en (5.2) estos datos, es de 1.4, pese a no ser una mejora muy atractiva. Dadas las características del problema, tal y como se mencionaba anteriormente, esta paralelización será beneficiosa en la aceleración total del algoritmo en casos en los que se quiera contar con una malla precalculada más densa y alcanzar así más precisión en los resultados.

5.3.1.2. Interpolación cúbica

La interpolación cúbica sobre el vector en el que se quiere evaluar la PDF se realiza tres veces, una por cada vértice del triángulo. Es en este punto donde se plantea la siguiente paralelización: una vez almacenado en una variable común a todos los hilos el identificador del triángulo envolvente, dividir la gestión de las PDF de los vértices en 3 hilos, encargándose cada uno de ellos de la desnormalización e interpolación de la PDF precalculada en cada vértice, ya que la malla de las PDF (*GRID_PDF*) es una variable global a la que todos los hilos pueden acceder. Esta mejora por tanto, va a estar limitada a 3 núcleos, pero el paralelizar en sí el ya eficiente algoritmo en serie de la interpolación cúbica basado en la librería GSL, no ha sido posible dada su fuerte dependencia en la evaluación de la función (*gsl_spline_eval*).

Esta tarea de paralelización se realiza con la directiva *#pragma omp section*, la cual distribuye secciones de trabajo independientes entre los hilos, tal y como se muestra en la figura 5.2. Las variables declaradas dentro de cada sección son privadas, de modo que cada hilo tendrá una copia independiente de su PDF a interpolar. Las variables declaradas fuera de la directiva son compartidas por defecto (parámetros de la distribución, vector de muestras y su tamaño) por todos los hilos, siendo en este caso sólo datos de lectura, por lo que los hilos no van a interferirse entre sí.

```
#pragma omp parallel sections
{
   #pragma omp section
   {
       Hilo 0: Extrae del GRID_PDF la PDF estándar precalculada
       correspondiente al vértice 1 del triángulo envolvente.
       Hilo 0: Desnormaliza la PDF del vértice 1.
       Hilo 0: Realiza la interpolación cúbica de la PDF
       desnormalizada del vértice 1.
   }

   #pragma omp section
   {
       Hilo 1: Extrae del GRID_PDF la PDF estándar precalculada
       correspondiente al vértice 2 del triángulo envolvente.
       Hilo 1: Desnormaliza la PDF del vértice 2.
       Hilo 1: Realiza la interpolación cúbica de la PDF
       desnormalizada del vértice 2.
   }

   #pragma omp section
   {
       Hilo 2: Extrae del GRID_PDF la PDF estándar precalculada
       correspondiente al vértice 3 del triángulo envolvente.
       Hilo 2: Desnormaliza la PDF del vértice 3.
       Hilo 2: Realiza la interpolación cúbica de la PDF
       desnormalizada del vértice 3.
   }

 } //Barrera implícita al finalizar la construcción paralela
```

Figura 5.2- Estructura de la implementación paralela de la interpolación cúbica.

En la tabla 5.4 se recogen los resultados obtenidos en la estimación de 1000 muestras. Se puede ver que pese a que se ha conseguido una mejora parcial muy próxima a 3, la aceleración del estimador completo, como ocurría en el caso anterior, sigue alejándose mucho de la máxima alcanzable. Teniendo en cuenta que en la tabla 5.2 se veía que el porcentaje de tiempo dedicado a la interpolación para 1000 muestras era de un 37.30%, el *speed-up* teórico total con 16 núcleos aplicando (5.2) es de aproximadamente 1.5, el cual se alcanza.

	1 núcleo	**2 núcleos**	**4 núcleos**	**8 núcleos**	**12 núcleos**	**16 núcleos**
Tiempo medio interpolación (ms)	0.3239	0.2171	0.1130	0.1126	0.1149	0.1140
***Speed-up* interpolación**	-	1.4919	2.8584	2.8766	2.8189	2.8412
Tiempo total estimación (ms)	55.3345	42.7381	34.1959	34.1372	34.3127	34.5533
***Speed-up* total**	-	1.2947	1.6182	1.6209	1.6126	1.6014

Tabla 5.4- Tiempos de ejecución tras paralelizar la interpolación cúbica de los tres vértices para 1000 muestras.

En definitiva, la paralelización de la obtención de la PDF en el algoritmo total de la estimación de los parámetros no parece un buen planteamiento, en cuanto a alcanzar la máxima aceleración teórica del algoritmo completo se refiere. En la figura 5.3 se representan los resultados de la aceleración total obtenida en el algoritmo de estimación de parámetros, aplicando conjuntamente estos dos métodos recién expuestos, sobre datos de diferentes tamaños.

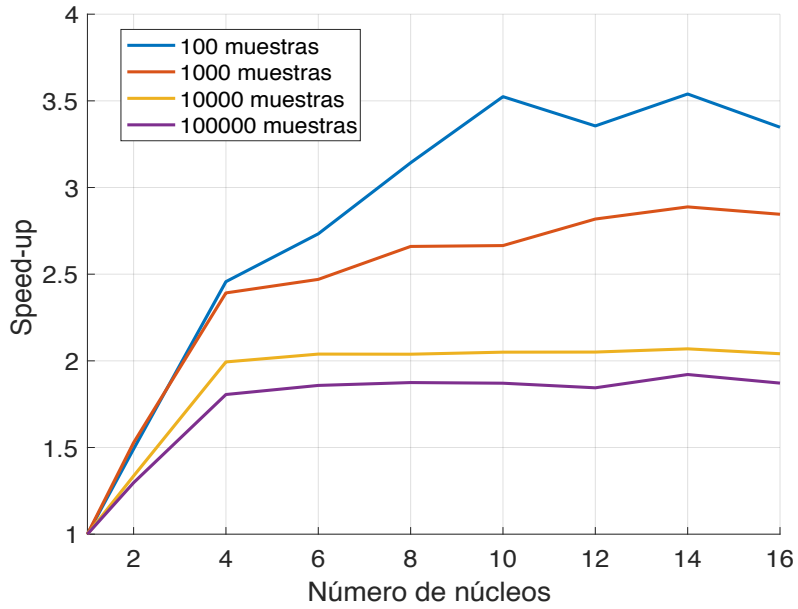


Figura 5.3- Aceleración obtenida tras paralelizar la obtención de la PDF para diferentes tamaños de muestras.

Tanto la paralelización de la búsqueda del triángulo como la interpolación, claramente optimizan los tiempos parciales de *stable_interp_pfd,* pero al evaluarse la PDF de manera cíclica a lo largo del algoritmo de estimación, se tratan de procesos poco costosos computacionalmente en los que el hecho de estar abriendo y cerrando hilos tan frecuentemente puede estar ocasionando que se pierda cualquier mejora en tiempo que se obtenga con la paralelización, lo que lleva a analizar otra alternativa.

5.3.2. Paralelización del cálculo de la verosimilitud

Una vez obtenida la PDF como la interpolación resultante según se veía en (4.3), se debe calcular la función log-verosimiltud para que el algoritmo de estimación MLE converja. Esta función se calcula como se presentaba en (4.1), es decir, se trata de un sumatorio del logaritmo aplicado a la PDF punto a punto. La paralelización en esta fase se centra pues, en la distribución de dichos sumandos entre los diferentes hilos disponibles. Este sumatorio se calcula en la función de libstable [11] stable*_loglike_p*, accediendo punto a punto a la PDF resultante, devuelta en el caso de este método propuesto, por *stable_pdf_interp.*

Para la paralelización de dicho sumatorio se utiliza la directiva *#pragma omp parallel for reduction(+:l),* como se muestra en la figura 5.4. En este tipo de construcción paralela, cada hilo lanzado en el bucle *for* privatiza la variable *l,* y al finalizar la directiva, los hilos actualizan la variable de la que deriva la copia realizando la operación suma.

```
#pragma omp parallel for reduction(+:l)
     Para i = 0 Hasta Tamaño de la PDF
       Si (PDF[i] > 0.0)
             Cada hilo calcula sobre las iteraciones i que se le asignan:
             l+= log(PDF[i])
     Fin Para //Barrera implícita y actualización de l
```

Figura 5.4- Estructura de la implementación paralela del cálculo de la verosimilitud.

En la tabla 5.5 se recogen los resultados obtenidos en la estimación de 1000 muestras, aplicando solo esta paralelización en aras de poder compararla con las obtenidas en la subsección anterior. Se puede comprobar que en este caso la optimización de este sumatorio no ha proporcionado una aceleración parcial superior a 6, y la mejora temporal del estimador completo no ha alcanzado con solo esta paralelización valores superiores a los que se tenían con anterioridad. En la tabla 5.2 se veía que para 1000 muestras esta función representaba solamente un 13.36% del tiempo de ejecución, por lo que aplicando (5.2) no cabía esperar una aceleración superior a 1.2.

	1 núcleo	**2 núcleos**	**4 núcleos**	**8 núcleos**	**12 núcleos**	**16 núcleos**
Tiempo medio log-verosimilitud (ms)	0.0949	0.0501	0.0302	0.0171	0.0159	0.0157
***Speed-up* log-verosimilitud**	-	1.8942	3.1424	5.5498	5.9686	6.0446
Tiempo total estimación (ms)	57.0468	47.5560	45.9195	44.6457	44.5306	44.7392
***Speed-up* total**	-	1.1996	1.2423	1.2778	1.2811	1.2751

Tabla 5.5- Tiempos de ejecución paralelizando el cálculo de la verosimilitud para 1000 muestras.

Sin embargo, si este sumatorio tuviese que realizarse sobre una PDF con más elementos, se puede deducir que esta paralelización sí va a cobrar importancia en la aceleración total del algoritmo. De esta forma, se puede conseguir un beneficio conjuntamente de las tres paralelizaciones desarrolladas hasta el momento. En la figura 5.4 se representan los resultados de la aceleración total obtenida en el algoritmo de estimación de parámetros aplicando las dos fases expuestas, sobre datos de diferentes tamaños.

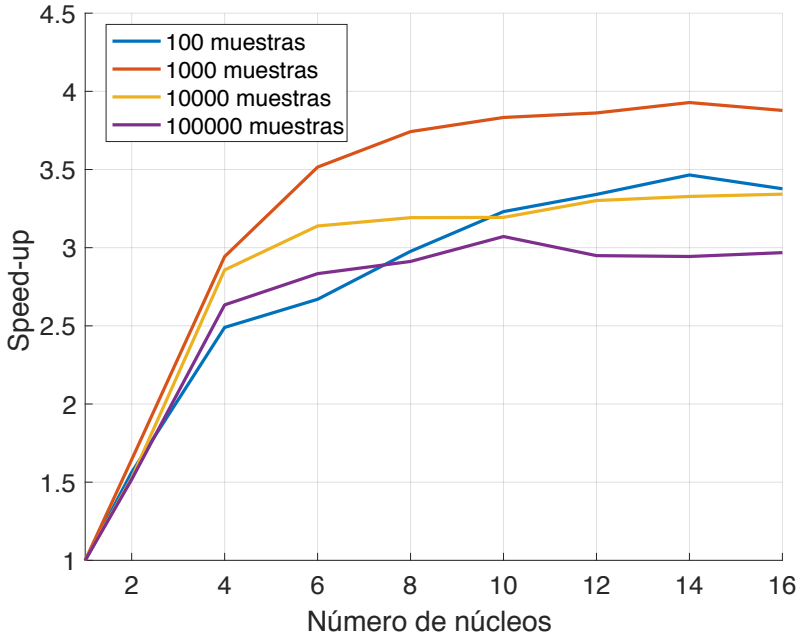


Figura 5.4- Aceleración obtenida tras paralelizar la obtención de la PDF y el cálculo de la verosimilitud para diferentes tamaños de muestras.

Comparando esta última figura con la figura 5.3 obtenida en la primera fase de paralelización, recogidos ambos resultados de aceleración en la tabla 5.6, se puede inferir que el conjunto de ambas paralelizaciones justifica el uso de varios procesadores para poder alcanzar una aceleración total del algoritmo de aproximadamente 3.3, incluso para conjuntos muestrales de gran tamaño.

Número de muestras	2 núcleos	4 núcleos	8 núcleos	12 núcleos	16 núcleos
	Fase 1				
100	1.4916	2.4570	3.1427	3.3553	3.3478
1000	1.5277	2.3919	2.6601	2.8181	2.8457
10000	1.3356	1.9937	2.0383	2.0507	2.0410
100000	1.2969	1.8058	1.8747	1.8445	1.8716
	Fase 1 + Fase 2				
100	1.5633	2.4900	2.9767	3.3408	3.3769
1000	1.6443	2.9432	3.7424	3.8618	3.8780
10000	1.5264	2.8574	3.1920	3.3015	3.3423
100000	1.5173	2.6341	2.9119	2.9492	2.9682

Tabla 5.6- Comparativa de las aceleraciones obtenidas para diferentes tamaños de muestras.

5.3.3. Paralelización del método de optimización

Con el objetivo de poder aproximarse más a la máxima aceleración alcanzable, la última fase planteada de paralelización es respecto al método de optimización de la verosimilitud. Cabe recordar que para ello se ha hecho uso del método Nelder-Mead proporcionado por la librería GSL (*gsl_multimin_fminimizer_nmsimplex2rand*). Para todo este estudio, ha sido necesaria la extracción de las funciones involucradas de la librería e incluirlas en el código propio para realizar las modificaciones pertinentes.

En términos generales, este método parte de la construcción de n+1 vectores a partir de una base ortonormal aleatoria, siendo n las dimensiones del problema (en nuestro caso 4 dimensiones, correspondientes a los parámetros α, β, σ y μ_0 de la distribución), a partir de los valores iniciales estimados y de los pasos $s = (s_0, s_1,..., s_{n-1})$. Suponiendo la estimación inicial $x = (x_0, x_1,..., x_{n-1})$, los vértices del simplex de n dimensiones están formados por los n+1 vectores:

$v_0 = (x_0, x_1, ..., x_{n-1})$

$v_1 = (x_0 + s_0, x_1, ..., x_{n-1})$

...

$v_i = (x_0, x_1, ..., x_{i-1} + s_{i-1}, ..., x_{n-1})$

$v_n = (x_0, x_1, ..., x_{n-1} + s_{n-1})$

En cada iteración del algoritmo se intenta mejorar el vector de parámetros v_i correspondiente al valor de la función más alto mediante transformaciones geométricas, que se presentaban en la figura 4.7: reflexión, expansión, contracción y contracción múltiple. Mediante estas transformaciones el simplex se mueve a través del espacio de parámetros hacia el mínimo. Esta serie de modificaciones del simplex en la implementación de GSL, se realiza acorde a los siguientes pasos:

- **Paso 1:** Se ordenan los vértices de forma que los valores de la función objetivo evaluada en cada vértice estén ordenados de forma creciente. Dado que el objetivo es minimizar la función, se toma el peor punto *hi,* donde se ha obtenido el valor más alto, el segundo punto *s_hi* más alto, y el punto más bajo *lo.*

- **Paso 2:** Se refleja el vector correspondiente a *hi* respecto al centroide del simplex que forman los n puntos restantes, y se evalúa la función en ese vector reflejado *r*.
  - Si *f(r) < f(lo)*, se salta al paso 3.
  - Si *f(r) > f(s_hi)*, *r* pasa a ser el nuevo vértice y se salta al paso 4.
  - Si *f(r) < f(s_hi), r* pasa a ser el nuevo vértice y se salta al paso 6.
  -
- **Paso 3:** Si el punto reflejado es más bajo, se expande el simplex doblando la distancia del vector *hi* al centroide, y se evalúa la función en ese vector expandido *e.*
  - Si *f(e) < f(lo), e* pasa a ser el nuevo vértice del simplex y se salta al paso 6.
  - Si *f(e) > f(lo), r* pasa a ser el nuevo vértice del simplex y se salta al paso 6.

- **Paso 4:** Si el punto reflejado no ha mejorado las cosas lo suficiente, se contrae el simplex por la mitad de la distancia del vector *r* al centroide del simplex, y se evalúa la función en ese punto *c*.
  - Si *f(c) < f(hi)*, *c* pasa a ser el nuevo vértice del simplex y se salta al paso 6.
  - Si *f(c) > f(hi)*, se salta al paso 5.

- **Paso 5:** Se lleva a cabo la contracción múltiple, es decir, se contrae el simplex reduciendo a la mitad la distancia de cada vértice respecto al vértice *lo* que produce el valor de la función más pequeño.

- **Paso 6:** Se itera hasta que el criterio de parada del algoritmo se cumpla, es decir, hasta que se alcance el mínimo o se supere un máximo de iteraciones, en este caso en concreto, configurado a 200 iteraciones máximas.

Dada la naturaleza del algoritmo, no todos los pasos mejoran el mejor vector de parámetros que se está evaluando, por ello normalmente se requieren varias iteraciones. Para el caso de la estimación de 1000 muestras generadas con parámetros (5.1) que se viene analizando a lo largo de esta sección, se estudia que en este caso concreto, el algoritmo requiere de 47 iteraciones hasta hallar el mínimo, de las cuales en 16 solo realiza la reflexión, y en las 30 restantes, la reflexión y una segunda transformación, ya sea la expansión o la contracción. Esto es lo que lleva a plantear la ejecución paralela de dichas operaciones, de forma que en aquellas iteraciones en las que se requiera el uso de dos transformaciones del simplex, ya se tenga la segunda transformación precalculada simultáneamente junto con la reflexión.

Aquí surge el problema de que, tal y como lo implementa GSL, la contracción (paso 4) es completamente dependiente de los valores modificados previamente por la reflexión, ya que parte del punto reflejado. Sin embargo, en el caso de la expansión se deduce cierta independencia y que sí puedan calcularse de forma simultánea.

Esta tarea de paralelización se realiza como en el caso de la interpolación cúbica, con la directiva *#pragma omp section*, de forma que distribuya dos secciones de trabajo independientes, una encargada de la reflexión y otra de la expansión. Estas tareas de reflexión y expansión conllevan una evaluación de la función objetivo, que en este caso se trata de la función log-verosimilitud, y a su vez por lo tanto, de la evaluación de la PDF en ese vector de parámetros que forman el vértice del simplex. Se busca por tanto, poder beneficiase de las construcciones paralelas desarrolladas en los apartados anteriores junto con esta, gracias a la posibilidad que brinda OpenMP de anidar regiones paralelas dentro de otras regiones.

Lamentablemente, los esfuerzos dedicados en la comprensión y paralelización de este algoritmo, han quedado en un mero ejercicio didáctico, ya que se lograron tiempos equiparables a la versión serializada. Se pudo observar que en ciertas iteraciones del algoritmo sí se conseguía una mejora temporal apreciable, pero en otras se incrementaba el tiempo respecto a la versión serie, por lo que realmente la carga entre el hilo

encargado de la reflexión y el encargado de la expansión no estaba siendo equilibrada y se producía por tanto un desequilibrio en los cálculos de cada procesador que repercutía negativamente. La realidad es que son muchos los factores que pueden intervenir en una correcta y eficiente paralelización, como la sincronización entre hilos, la utilización de secciones críticas, los bloqueos entre hilos, etc. Dada la complejidad de estos factores en el método de optimización Nelder-Mead de la librería GSL, su aceleración no ha sido posible, y con el fin de disminuir su influencia negativa en la aplicación, ha sido descartada.

5.4. Conclusiones

En este capítulo se ha analizado en primer lugar, el tiempo y la estructura de la versión serie de la herramienta desarrollada para el tratamiento estadístico de las distribuciones α-estables, con el objetivo de acelerarlo atendiendo en la mayor medida de lo posible a las leyes de Amdahl y Gustafson. Se ha visto que la función que consume la mayor parte del tiempo del programa es el método MLE, dentro del cual no existe un claro paso que consuma la mayor parte de su tiempo como para focalizar en sólo un punto nuestro interés en la paralelización, ya que varía según el número de muestras. Por ello, se han planteado tres fases o niveles donde se preveía que pudiese alcanzarse una aceleración apreciable.

La independencia de las estructuras principales *GRID_TRI* y *GRID_PDF* ha hecho que las evaluaciones independientes de las coordenadas baricéntricas y de cada uno de los tres vértices del triángulo envolvente, haya permitido repartir el trabajo de la evaluación de la PDF entre los hilos disponibles. Esta idea, junto con la paralelización del cálculo de la función log-verosimilitud, han permitido alcanzar una aceleración entre 2.5 y 3 con 4 núcleos. Dicha aceleración aumenta con el número de procesadores, pero sin alcanzar valores superiores a 3.8, dado que una de las fases de paralelización implementada, la distribución del tratamiento de las tres PDF de los vértices a tres hilos independientes, hace que hablar por tanto de más de 3 procesadores no sea muy beneficioso. Se debe tener en cuenta que se ha querido mantener la formulación secuencial previa, paralelizando las partes de código cuyo cálculo fuese independiente, y aunque esta metodología requiere introducir pocos cambios en el código, el aumento de eficiencia conseguido es bajo. El trabajo al que aquí nos enfrentamos no tiene un tamaño fijado, ya que va a depender de las muestras, de la densidad de la malla precalculada y de la precisión de los cálculos en la optimización de la máxima verosimilitud, de forma que, si el tamaño del problema es lo suficientemente grande, podemos explotar el paralelismo que se ha encontrado.

En el siguiente capítulo, se procede a utilizar esta versión acelerada del algoritmo para realizar estimaciones de los parámetros α-estables sobre muestras de tráfico real que se desean modelar, obteniendo los resultados en un tiempo razonable.

Capítulo 6. Aplicación al análisis de tráfico

6.1. Introducción

Como se ha presentado en el capítulo 3, el tiempo de respuesta de un servidor en el establecimiento de una conexión TCP se puede modelar con una distribución α-estable, y si la estimación de sus parámetros se ha logrado acelerar con las prácticas desarrolladas a lo largo de los capítulos 4 y 5, su aplicación en un entorno real se plantea como una interesante aportación.

La precisión de la estimación y el tiempo de ejecución se han evaludado hasta el momento a lo largo de este trabajo con muestras α-estables sintéticas y distribuciones estandarizadas, es decir, con $\sigma = 1$ y $\mu_0 = 0$. En este capítulo, se describen las pruebas y los experimentos realizados para la evaluación de la herramienta desarrollada sobre registros de tráfico real.

Con el objetivo de contextualizar este capítulo, en la figura 6.1 se representa de forma esquemática el flujo completo del escenario de aplicación de la herramienta en trazas de tráfico de red. En naranja se muestran las fases llevadas a cabo en este capítulo, a partir de unos registros ya proporcionados.

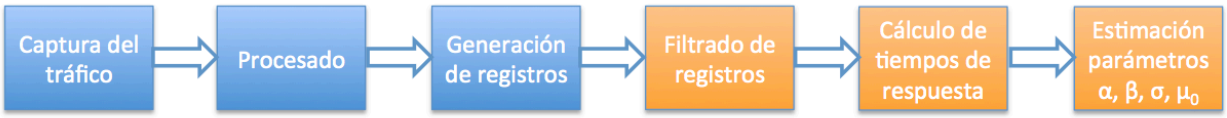


Figura 6.1- Flujo del escenario de aplicación de la herramienta.

A continuación, se van a describir los datos utilizados, y seguidamente se muestran los experimentos y las mediciones realizadas para, finalmente, realizar una discusión sobre los resultados obtenidos y las posibilidades de esta herramienta en su implantación en entornos de red reales.

6.2. Análisis previo de los registros utilizados

Para la realización de este trabajo se ha partido de una serie de registros correspondientes al monitoreo de toda una semana de una entidad privada grande con un gran número de sucursales, generados en el año 2015 con el sistema M3Omon [24], encargándose dicha herramienta de la captura de tráfico, su procesado y la generación de los registros (bloques azules representados en la figura 6.1), a partir de los cuales se han realizado los experimentos que se expondrán a continuación. Dada la gran batería de información con la que se contaba, aproximadamente un total de 140 GB, se han filtrado de entre todos los registros los correspondientes al servidor más popular, es decir, el servidor que presentaba más conexiones a lo largo de la semana (de ahora en adelante, S1) del lunes 28 de septiembre al domingo 4 de octubre de 2015, representadas sus conexiones por segundo en la figura 6.2.

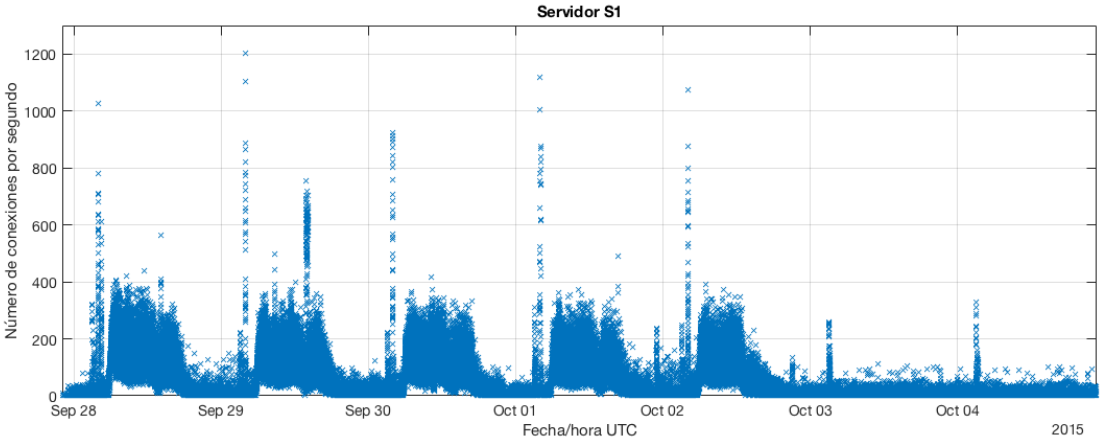


Figura 6.2 - Conexiones por segundo a lo largo de la semana para el servidor S1.

En el escenario presentado en el capítulo 3, se veía que el tiempo de respuesta del servidor en el establecimiento de la conexión TCP se puede calcular en la práctica como el tiempo que transcurre desde el *timestamp* registrado del primer SYN del cliente al servidor, y el *timestamp* del primer SYN del servidor al cliente. Para cada establecimiento de conexión TCP, una hipotética sonda de red ubicada próxima al servidor en cuestión, registra el tiempo de llegada de los paquetes que recibe en su interfaz de red, y sobre esa información, se ha procedido a su tratamiento. La resolución temporal que se ha obtenido es del orden de microsegundos. En la figura 6.3 se puede ver la evolución del tiempo medio de respuesta por hora que se ha obtenido para este servidor de estudio.

Se puede inferir de las figuras 6.2 y 6.3 que, efectivamente, el tiempo de respuesta del servidor en la conexión TCP está estrechamente relacionado con la cantidad de conexiones que tiene que atender en ese instante el servidor. Aquellos momentos del día donde se percibe un mayor número de conexiones por segundo, el tiempo medio de respuesta del servidor en el establecimiento de la conexión TCP se incrementa. Este hecho se puede apreciar más claramente en la figura 6.4, donde se extrae un determinado día de la semana para el caso del servidor S1. Se observa que en horas

punta, como la correspondiente a las 06:00-07:00 UTC, hora aproximada de comienzo de la jornada laboral en dicha entidad (ubicada en zona UTC+2), los nacimientos de conexiones de los diferentes clientes hacia S1, experimentan una subida, que se corresponde con un pico en su tiempo medio de respuesta en dicha hora. De la misma manera, al medio día y al finalizar la jornada laboral, se percibe una bajada de dicho tiempo medio de respuesta en el servidor S1, ocasionado por la baja carga de solicitudes de conexión que tiene que atender en esos instantes, de forma que puede responder más rápidamente a los clientes.

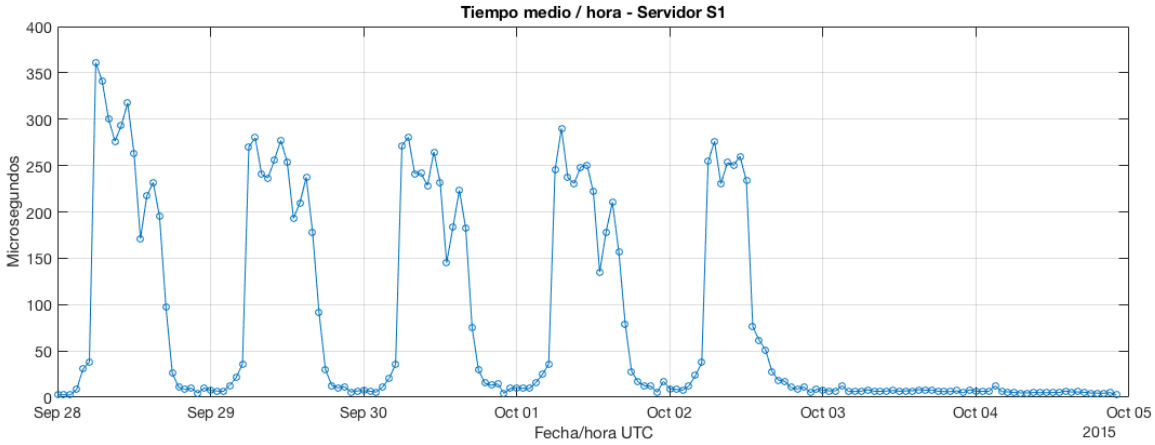


Figura 6.3 - Evolución por horas del tiempo medio de respuesta en el establecimiento de la conexión TCP.

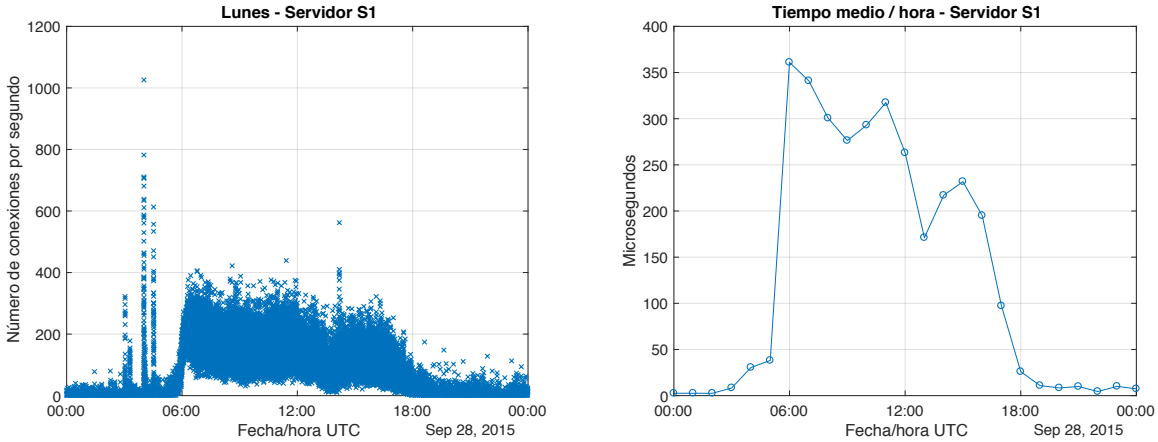


Figura 6.4 - Evolución diaria del tiempo medio de respuesta en S1.

Tras analizar la aparente potencialidad del tiempo de respuesta del servidor en una conexión TCP como indicador de su carga, en este punto, se opta por un análisis multiresolución con ventanas de 1, 5, 15 y 60 minutos, de forma que pueda apreciarse la aparición de saturaciones en el servidor que a otras escalas mayores no se logren apreciar. En la figura 6.5 se muestra un ejemplo de este efecto, correspondiente a los días laborables de la semana de estudio.

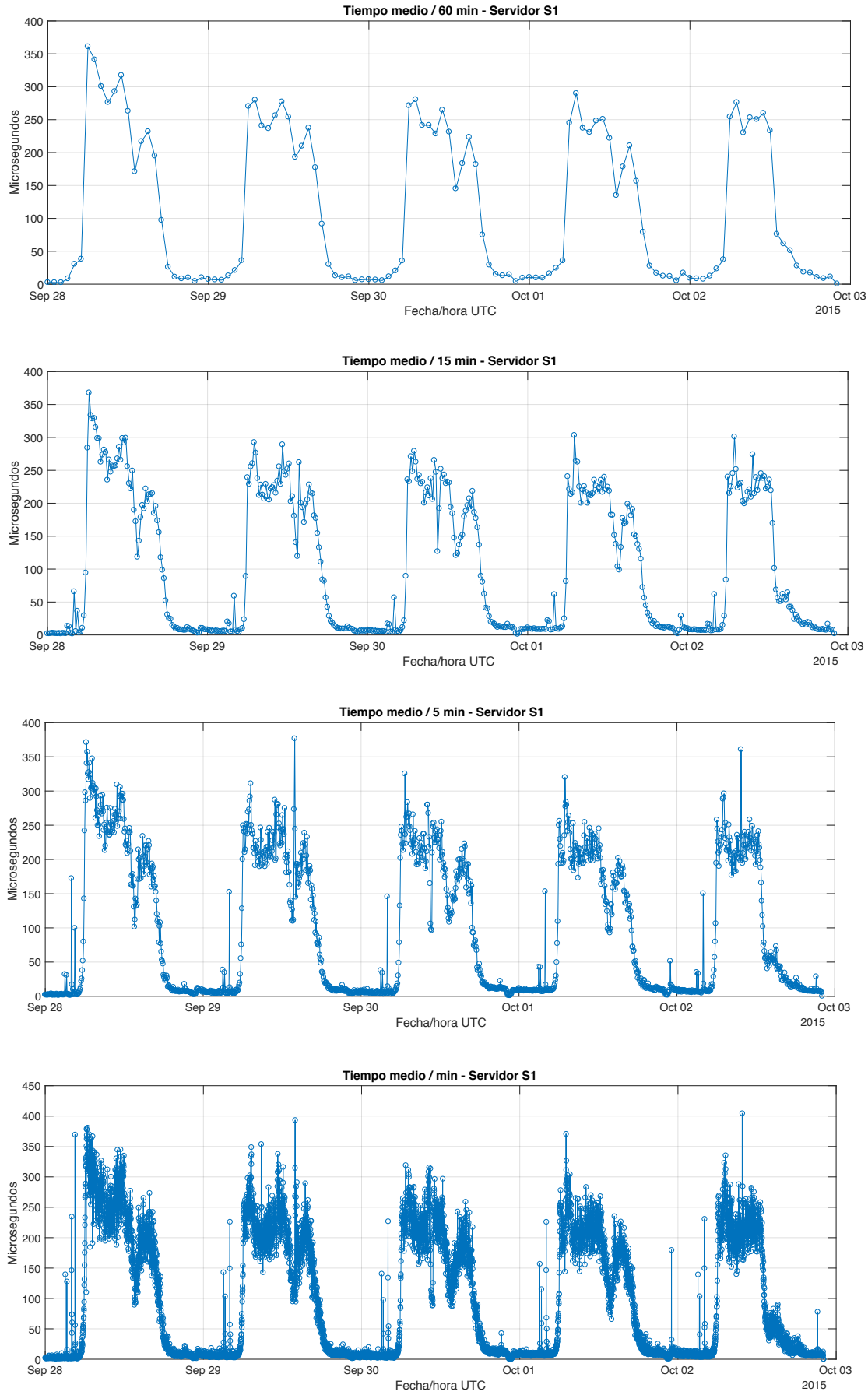


Figura 6.5 - Análisis multiresolución del tiempo medio de respuesta del servidor S1 en la semana de estudio.

6.3. Resultados aplicando la herramienta desarrollada

Para caracterizar con la herramienta implementada el tiempo de respuesta del servidor en el establecimiento de conexión TCP del servidor de estudio, se ha realizado la estimación de los parámetros siguiendo un enfoque multiresolución, con ventanas de 1, 5, 15 y 60 minutos.

En esta sección se van a considerar los siguientes cuatro casos concretos, recopilados en la tabla 6.1:

- **Caso 1:** Intervalo de 1, 5, 15 y 60 minutos en una hora de alta actividad de un día laborable, en concreto, las 07:00-08:00 AM UTC del lunes 28 de septiembre de 2015.

- **Caso 2:** Intervalo de 1, 5, 15 y 60 minutos en una hora de baja actividad de un día laborable, en concreto, las 00:00-01:00 AM UTC del lunes.

- **Caso 3:** Intervalo de 1, 5, 15 y 60 minutos en la misma hora que el caso 1, las 07:00-08:00 AM UTC, pero de un día no laborable, como es el domingo.

- **Caso 4:** Intervalo de 1, 5, 15 y 60 minutos durante la misma hora que el caso 2, 00:00-01:00 AM UTC, del domingo 4 de octubre de 2015.

	Caso 1 Lunes 28 de septiembre de 2015		Caso 2 Lunes 28 de septiembre de 2015	
ΔT	Hora UTC	Número de muestras	Hora UTC	Número de muestras
1 min	07:00-07:01 AM	11536	00:00-00:01 AM	100
5 min	07:00-07:05 AM	53359	00:00-00:05 AM	402
15 min	07:00-07:15 AM	165585	00:00-00:15 AM	1130
60 min	07:00-08:00 AM	619355	00:00-01:00 AM	4999
	Caso 3 Domingo 4 de octubre de 2015		Caso 4 Domingo 4 de octubre de 2015	
ΔT	Hora UTC	Número de muestras	Hora UTC	Número de muestras
1 min	07:00-07:01 AM	171	00:00-00:01 AM	246
5 min	07:00-07:05 AM	768	00:00-00:05 AM	1018
15 min	07:00-07:15 AM	2537	00:00-00:15 AM	2913
60 min	07:00-08:00 AM	9029	00:00-01:00 AM	12604

Tabla 6.1 - Casos de estudio para la estimación de los parámetros aplicando la herramienta desarrollada sobre diferentes escalas temporales.

El criterio para la elección de estos casos ha sido evaluar dos escenarios diferentes, que permita extraer conclusiones sobre los cambios de los parámetros de las distribuciones α-estables que modelan el tiempo de respuesta del servidor en condiciones dispares, como puede ser un lunes y un domingo, o una hora punta y una hora de poco tráfico. Además, se puede ver en la tabla 6.1 que el tamaño de muestras varía notoriamente entre el lunes y el domingo, y en el caso del lunes, entre periodos diurnos y nocturnos. Esto nos va a permitir validar que los tiempos de ejecución en la versión paralela se aceleran, aún para diferentes volúmenes de datos.

6.3.1. Validación de la aceleración paralela

En primer lugar, se busca hacer una evaluación del tiempo de ejecución que requiere la estimación de los parámetros de las muestras seleccionadas, siendo ejecutado, como en el capítulo 4, en un equipo compuesto por dos procesadores AMD Opteron 6128 de 8 núcleos a 2 GHz. Acorde a los resultados de aceleración que se han obtenido en el capítulo 4, a pesar de que este nodo escala hasta 16 núcleos, la versión paralela de la herramienta ha sido ejecutada con 4 núcleos para una mayor eficiencia del sistema. En la tabla 6.2 se recogen los resultados de tiempo de ejecución de la herramienta desarrollada al estimar los parámetros para cada uno de los cuatro casos de estudio. Se puede ver que la versión paralela del algoritmo proporciona una aceleración próxima a 3, ejecutándose con 4 núcleos y sobre datos, como se presentaba en la tabla 6.1, de diferentes tamaños.

Hora UTC AM	Serie (ms)	4 núcleos (ms)	*Speed-up*	Serie (ms)	4 núcleos (ms)	*Speed-up*
	Caso 1 - Lunes 28 de septiembre de 2015			Caso 3 - Domingo 4 de octubre de 2015		
07:00-07:01	253.9160	96.2290	2.6387	143.6011	50.9729	2.8172
07:00-07:05	1185.1611	444.8821	2.6640	120.8940	39.8269	3.0355
07:00-07:15	3279.5349	1244.9749	2.6342	460.7900	162.2390	2.8402
07:00-08:00	14176.6880	5230.0869	2.7106	363.7249	127.8589	2.8447
	Caso 2 - Lunes 28 de septiembre de 2015			Caso 4 - Domingo 4 de octubre de 2015		
00:00-00:01	129.2981	40.7100	3.1761	151.3040	47.7190	3.1707
00:00-00:05	164.7891	57.7598	2.8530	257.9880	86.6431	2.9776
00:00-00:15	75.2280	25.6741	2.9301	519.7620	184.4380	2.8181
00:00-01:00	787.4431	281.8501	2.7938	1913.8818	688.6760	2.7791

Tabla 6.2 - Tiempos de ejecución para la estimación de parámetros sobre los casos de estudio.

6.3.2. Validación del modelo estadístico

Como segunda validación de los resultados obtenidos al aplicar la herramienta desarrollada sobre datos reales de tiempos de respuesta de un servidor concreto, se analizan a continuación los ajustes que se han obtenido en la estimación de los parámetros de cada uno de los casos de estudio. En la tabla 6.3 se recogen los parámetros estimados para cada caso.

Hora UTC AM	$\hat{\alpha}$	$\hat{\beta}$	$\hat{\sigma}$	$\hat{\mu}_0$	$\hat{\alpha}$	$\hat{\beta}$	$\hat{\sigma}$	$\hat{\mu}_0$
	Caso 1 Lunes 28 de septiembre de 2015				Caso 3 Domingo 4 de octubre de 2015			
07:00-07:01	1.5958	0.9525	26.8847	340.7718	1.6000	1.0000	27.1075	334.8756
07:00-07:05	1.5846	0.9497	26.7477	341.1272	1.6000	1.0000	24.8663	333.7425
07:00-07:15	1.5836	0.9566	27.7540	345.8145	1.6000	1.0000	25.3513	332.2634
07:00-08:00	1.5925	0.9807	28.2489	347.6127	1.5999	0.9999	25.4163	332.2193
	Caso 2 Lunes 28 de septiembre de 2015				Caso 4 Domingo 4 de octubre de 2015			
00:00-00:01	1.4675	1.0000	28.4215	329.5119	1.7843	1.0000	31.0127	345.4805
00:00-00:05	1.4519	1.0000	26.7922	332.1550	1.4275	1.0000	24.9230	329.1896
00:00-00:15	1.4306	0.9997	26.2047	330.1811	1.5708	1.0000	25.4751	331.7727
00:00-01:00	1.4000	1.0000	26.6305	328.5113	1.4962	1.0000	24.1418	329.4470

Tabla 6.3 - Parámetros estimados sobre las muestras de los diferentes casos de estudio y escalas temporales.

En los resultados obtenidos en el análisis de estos casos concretos de estudio se pueden observar dos hechos relevantes:

1. El parámetro β muestra una simetría total hacia la derecha (β = 1) en la distribución de los datos de prácticamente todos los casos, o bien muy próximo a dicho valor.

2. Para el día laborable, se había visto en la figura 6.4 que el tiempo medio por hora a las 07:00-08:00 AM UTC era de 361 μs y a las 00:00-01:00 AM de 2.585 μs, apreciándose por tanto un importante cambio entre dos horas de alta y baja actividad, respectivamente. En estos casos del lunes aplicando la ventana de 60 minutos (en azul en la tabla 6.3), los resultados muestran que los parámetros α, σ y μ_0 presentan una variación, especialmente notoria en el parámetro μ_0, siendo de 19.1014. Esta variación del valor de μ_0 entre una y otra hora se reduce para el caso del domingo a 2.7723, dado que ambas horas para el caso del un día no laborable pueden considerarse de bajo tráfico.

Para posibilitar un futuro estudio y extracción de conclusiones sobre la estimación multiresolución, aplicando ventanas de 1, 5, 15 y 60 minutos, se recogen las gráficas de la evolución de dichos parámetros a lo largo de todo el lunes 28 de septiembre de 2015 en el anexo de este documento. En las figuras 6.6 y 6.7 se representa el modelo obtenido para las muestras de los casos 1 y 2.

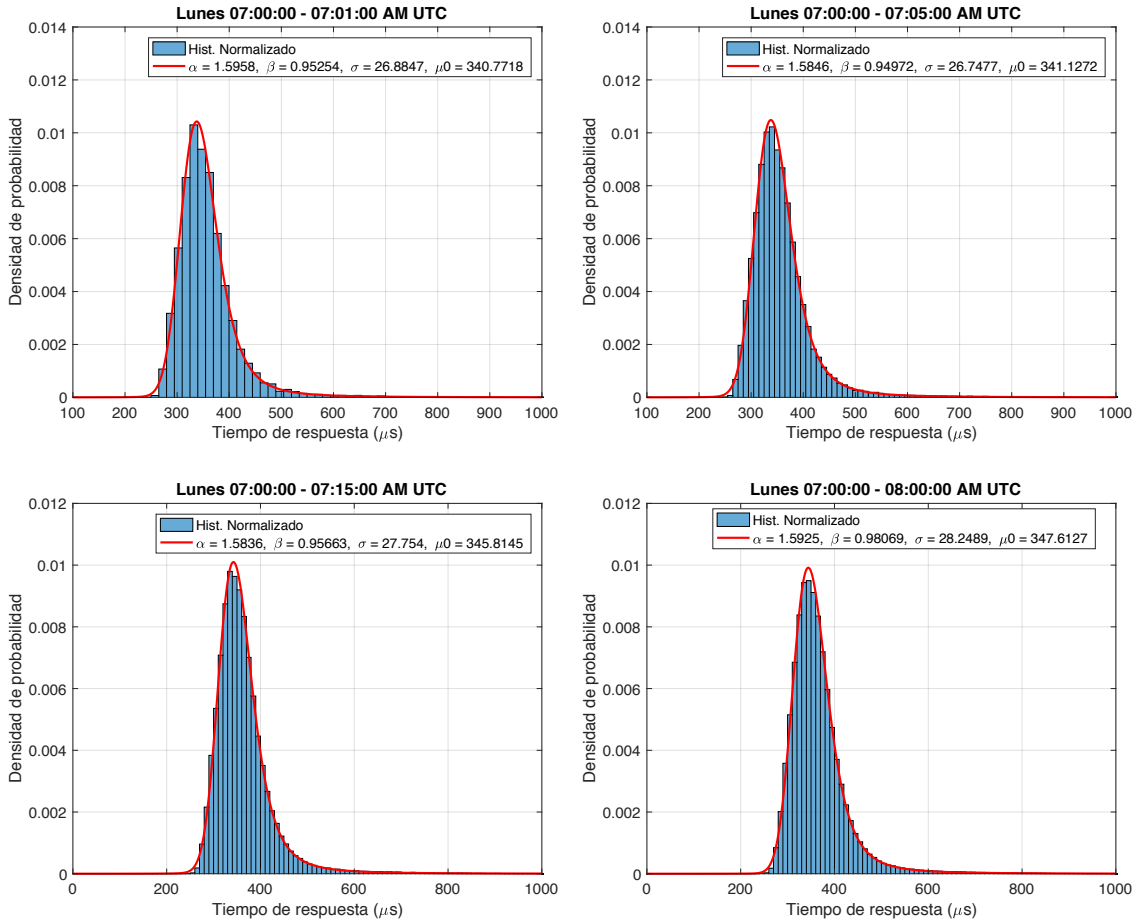


Figura 6.6 - Distribución del tiempo de respuesta de S1 y su modelo α-estable aplicando la herramienta desarrollada sobre escalas temporales de 1, 5, 15 y 60 minutos (caso 1).

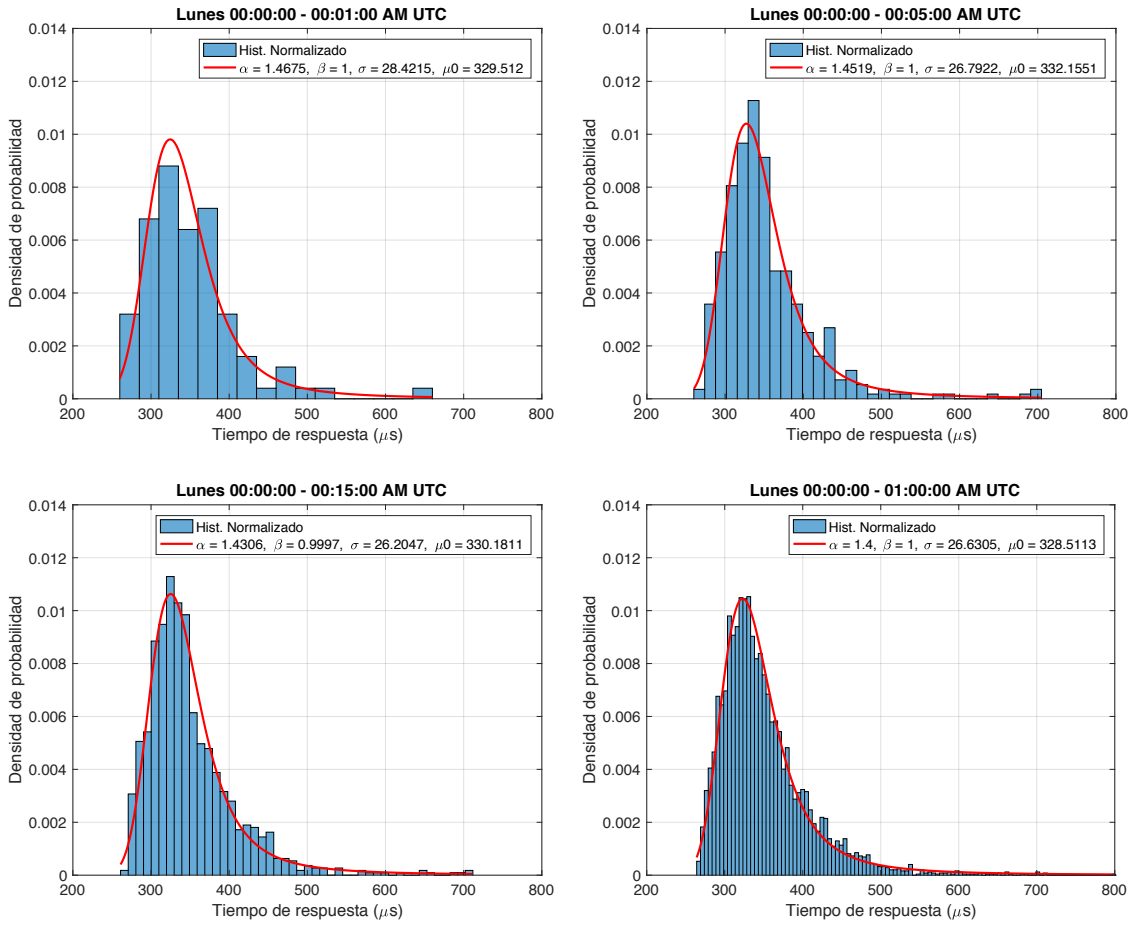


Figura 6.7 - Distribución del tiempo de respuesta de S1 y su modelo α-estable aplicando la herramienta desarrollada sobre escalas temporales de 1, 5, 15 y 60 minutos (caso 2).

Por último, se escoge como ejemplo la ventana de 15 minutos correspondiente a los cuatro casos de estudio, y se busca su comparación con otros modelos diferentes. En la figura 6.8 y 6.9 se muestra una comparativa cualitativa entre los diferentes modelos aplicando el ajuste disponible en Matlab [26] para todos los demás casos de distribuciones, y la distribución acumulada estimada a partir de la PDF α-estable obtenida con la herramienta desarrollada en este Trabajo Fin de Máster.

Tras estos últimos resultados se puede observar que, como se ve en las figuras 6.6 y 6.7, el tiempo de respuesta del servidor en una conexión TCP muestra poder modelarse correctamente con una distribución α-estable, incluso a diferentes escalas temporales. Además, en las figuras 6.8 y 6.9 se ha comparado la densidad de probabilidad acumulada aplicando diferentes distribuciones, típicamente utilizadas en el análisis de tráfico. Los modelos tradicionales no son capaces de modelar correctamente la presencia de colas pesadas en la distribución de las medidas de tiempos de respuesta del servidor. Sin embargo, se puede apreciar que la gráfica correspondiente a la CDF calculada a partir del modelo aquí desarrollado, por inspección y métodos heurísticos, se aproxima más a la distribución de las muestras de tráfico tratadas en este capítulo.

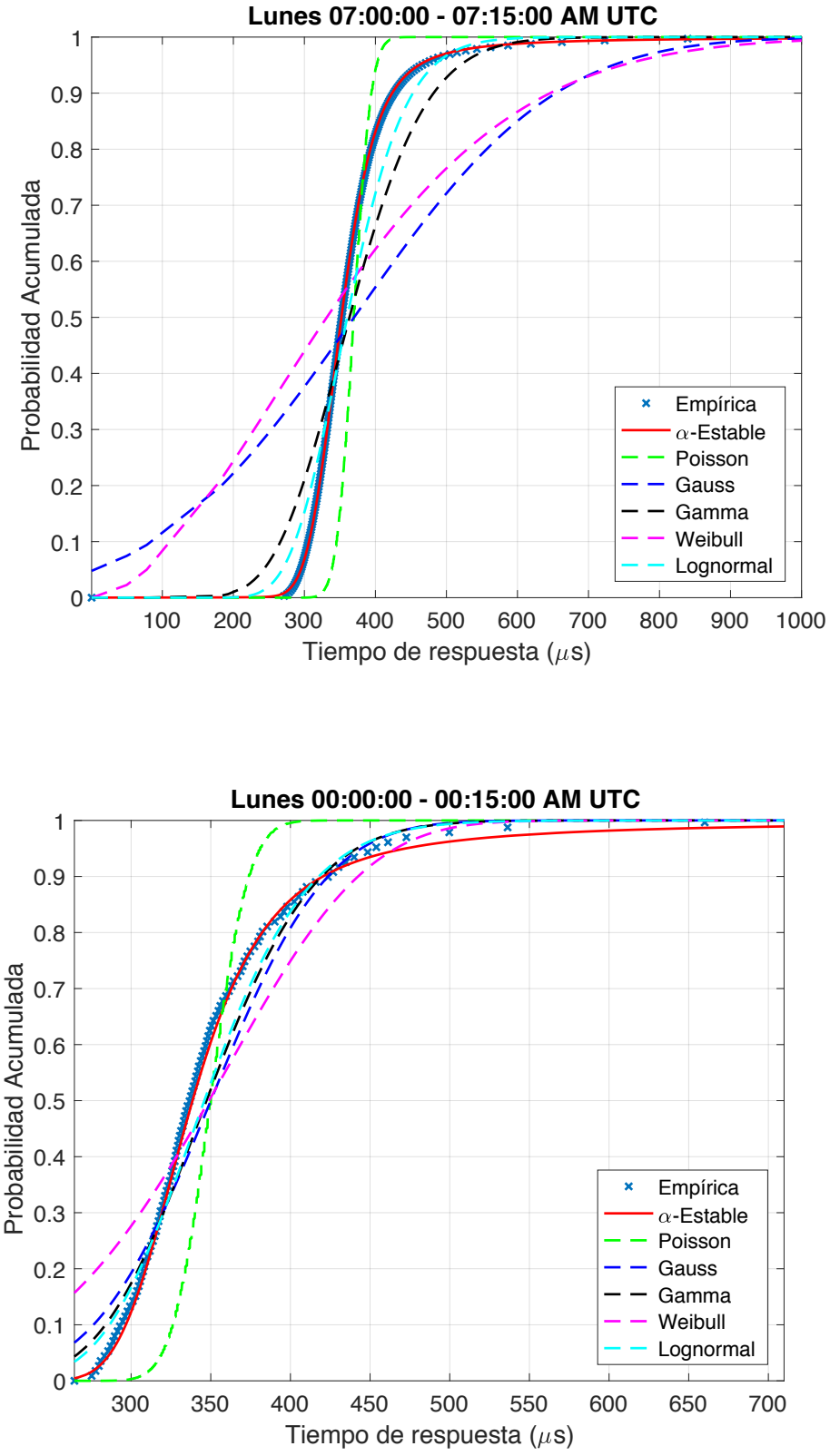


Figura 6.8 - Comparativa de la CDF calculada a partir de las muestras, y la CDF obtenida a partir de los diferentes ajustes, incluido el desarrollado en este trabajo, para el caso de estudio del lunes.

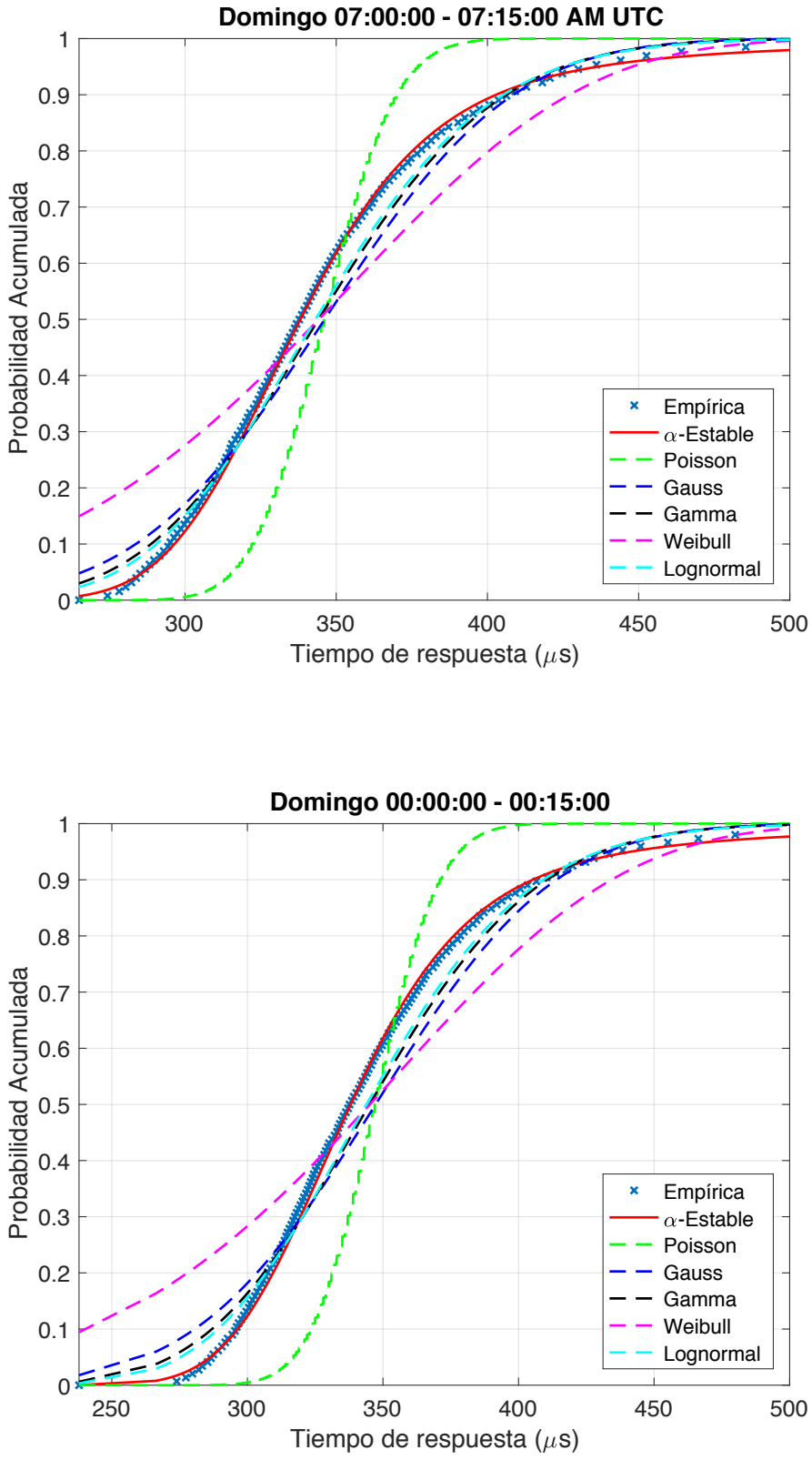


Figura 6.9 - Comparativa de la CDF calculada a partir de las muestras, y la CDF obtenida a partir de los diferentes ajustes, incluido el desarrollado en este trabajo, para el caso de estudio del domingo.

6.4. Conclusiones

A lo largo de este capítulo se han presentado una serie de resultados obtenidos al aplicar la herramienta desarrollada sobre la medida de red que se buscaba modelar. Para ello, se contaba con una serie de registros sobre los cuales se han filtrado aquellos que correspondiesen al servidor con más conexiones a lo largo de la semana de estudio.

Como se ha visto en este capítulo, a partir de las series temporales para este servidor, se han elegido cuatro longitudes de ventana temporal diferentes: 1, 5, 15 y 60 minutos. Con este análisis multiresolución, se ha podido observar que el estudio de la estacionariedad resulta relevante, ya que aparentemente para periodos más largos se mantiene cierta estacionariedad del tiempo de respuesta del servidor entre intervalos consecutivos. Sin embargo, con una ventana de 1 minuto se pueden apreciar la aparición de saturaciones en el servidor, que a escalas mayores no logran apreciarse, lo que refuerza la necesidad de estimar los parámetros a escalas pequeñas. Esto plantea la búsqueda de un buen equilibrio entre un periodo de tiempo en el cual se pueda considerar el tráfico de red estacionario y el número de muestras comprendidas en dicho intervalo que doten de más robustez al modelo.

Se ha podido comprobar que el número de muestras de tiempo de respuesta del servidor en el establecimiento de conexión TCP , es mayor cuando se estudian períodos del la semana con alta actividad, y que por lo tanto, el tiempo de ejecución en la estimación de los parámetros puede llegar a incrementarse. En cualquier caso, gracias a la aceleración paralela desarrollada en este trabajo, se ha logrado procesar miles de muestras en cuestión de milisegundos.

Se ha confirmado cualitativamente que el tiempo de respuesta del servidor en el establecimiento de conexiones TCP parece ser una magnitud de tráfico candidata a modelarse adecuadamente con distribuciones α-estables. De la evolución de sus parámetros, incluida en el anexo, no se ha llegado a obtener una clara conclusión, salvo que la distribución es claramente asimétrica y que μ_0 parece estar estrechamente relacionado con la carga del servidor, dado que su valor se incrementa en períodos de alta actividad, en este caso, en períodos correspondientes a la duración de la jornada laboral. Dado la buena adecuación del modelo en comparación con otros, parece interesante el modelar el tiempo de respuesta del servidor con α-estables y, que a partir de su tratamiento estadístico y evaluación de los parámetros, lleguen a tomarse decisiones o incluso anticiparse a incidencias que puedan producirse en el servidor.

Este capítulo deja abiertas toda una serie de posibilidades e investigaciones futuras que se plantean en el siguiente capítulo.

Capítulo 7. Conclusiones y trabajo futuro

7.1. Conclusiones

En este Trabajo Fin de Máster, se ha seguido un proceso de análisis de un algoritmo y su optimización, con el objetivo de una futura aplicación en entornos reales de monitorización de tráfico de red.

Como punto de partida, se ha tomado un algoritmo serie desarrollado en Matlab que en la literatura contaba con prometedores resultados. Se ha analizado este método, el cual estima los parámetros de las distribuciones α-estables de una forma rápida, ya que la obtención de la PDF se basa en una malla de densidades de probabilidad precalculadas sobre una rejilla de parámetros α-β. Esto lo aleja de la problemática, en tiempo de ejecución, que presentan las distribuciones α-estables derivada de la falta de expresión cerrada para su densidad de probabilidad.

Con el traspaso del método de Matlab a C, lidiando con las dificultades que se han presentado al tratarse de algoritmos propietarios de código no disponible, se ha alcanzado una importante optimización en los tiempos, llegando en algunos casos a ser hasta 10 veces más rápido.

Tras esta primera optimización, se ha llevado a cabo la aceleración del algoritmo mediante programación paralela, en concreto, mediante OpenMP. Se han propuesto tres niveles o fases en la paralelización. Excepto descartando una de ellas, para la cual no se ha podido concluir una aceleración temporal, aplicando las paralelizaciones propuestas se ha alcanzado una aceleración de aproximadamente 3, ejecutándolo sobre 4 núcleos de un equipo AMD Opteron 6128 a 2 GHz.

Conseguida una herramienta capaz de estimar los parámetros hasta tres veces más rápido que la versión equivalente en serie, se ha probado su validez sobre muestras de tráfico de red real. De manera cualitativa, se ha podido comprobar que los tiempos de respuesta de un servidor de estudio en el establecimiento de conexión TCP, se ajustan mejor con el modelo desarrollado que otros modelos de tráfico típicamente utilizados.

7.2. Trabajo futuro

Se resumen exponen las posibles líneas de trabajo futuro que han surgido a partir de la realización de este trabajo:

- La implantación de esta algoritmo en una arquitectura tipo GPU. En este trabajo se ha llevado a cabo la portabilidad de un programa Matlab a C, y posteriormente a su aceleración mediante OpenMP. Del código C es previsible que se pueda obtener beneficio de esta implementación, ya que la forma de gestionar los datos se puede beneficiar de las características de las GPU, donde debemos intentar minimizar la interacción entre GPU y CPU, debido a que obtener datos de la memoria de la CPU para ser utilizados por la GPU puede causar una grave penalización en los tiempos de ejecución. Una primera aproximación, aprovechando el uso de las directivas utilizadas con OpenMP, puede ser el uso de OpenACC [41].

- La paralelización de método de optimización Nelder-Mead. En este trabajo no se ha logrado optimizar el enfoque de GSL del que partía la versión serie del algoritmo. Sin embargo, en la literatura se han encontrado métodos [42] para su futura implementación paralela.

- La paralelización de la primera etapa del algoritmo analizado en el capítulo 2. A lo largo de este trabajo se ha contado con ya un malla de densidades precalculadas, ya que su obtención se trataba de un preceso *offline,* pero se presenta como un algoritmo con interesantes posibilidades de paralelización.

- La comprobación cuantitativa de la adecuación del método desarrollado al modelar muestras de tiempo de respuesta del servidor en el establecimiento de conexión TCP, con por ejemplo un test de bondad como Kolmogórov-Smirnov, el cual por limitaciones de tiempo, no ha llegado a aplicarse.

- Estudio de la correlación de la evolución de los parámetros α-estables con la carga de CPU del servidor, para conocer cual de ellos puede permitir su predicción. Esto permitirá una aplicación industrial directa de la herramienta, de manera que su implantación en una sonda permita predecir la carga del servidor para detectar incidencias de manera no intrusiva, y tendencias del servidor antes de un posible colapso. Para que esta gestión de la capacidad y de posibles cuellos de botella tenga sentido, debe hacerse rápidamente.

Glosario

AM:	Antes del Mediodía
FARIMA:	Auto Regressive Integrated Moving-Average
FGN:	Fractional Gaussian Noise
CDF:	Cumulative Distribution Function
CPU:	Central Processing Unit
FTP:	File Transfer Protocol
FPGA:	Field-Programmable Gate Arrays
GB:	Gigabyte
GPU:	Graphics Processor Unit
GSL:	GNU Scientific Library
IEEE:	Institute of Electrical and Electronics Engineers
IP:	Internet Protocol
MLE:	Maximum Likelihood Estimator
MPI:	Message Passing Interface
NUMA:	Non-Uniform Memory Access
OpenCL:	Open Computing Language
OpenMP:	Open Multi-Processing
OpenMP ARB:	OpenMP Architecture Review Board
PDF:	Probability Density Function
POSIX:	Portable Operating System Interface
RTT:	Round Trip Time
SIMD:	Single Instruction Multiple Data
SMP:	Symmetric Multi-Processing

SMTP:	Simple Mail Transfer Protocol
SQP:	Sequential Quadratic Programing
SSH:	Secure Shell
TELNET:	Teletype Network
TCP:	Transport Control Protocol
UMA:	Uniform Memory Access
UTC:	Coordinated Universal Time
VA:	Variable Aleatoria
WWW:	World Wide Web

Referencias

[1] Federico Simmross Wattenberg, Directores: Juan Ignacio Asensio Pérez y Marcos Martín Fernández. Detección de anomalías en el tráfico agregado de redes IP basada en inferencia estadística sobre un modelo α-estable de primer orden. Tesis Doctoral, Universidad de Valladolid, Julio 2009.

[2] Mark E. Crovella, Azer Bestavros. Self-similarity in World Wide Web traffic: evidence and possible causes. *IEEE/ACM Transactions on networking*. 5(6), 835-846, 1997.

[3] Ioannis A. Dimitriadis, Carlos Alberola-López, Marcos Martín-Fernández, Pablo Casaseca-de-la-Higuera, Federico Simmross-Wattenberg, Juan Ignacio Asensio-Pérez. Anomaly detection in network traffic based on statistical inference and α-stable modeling. *IEEE Transactions on Dependable and Secure Computing*, 8(4), 494-509, 2011.

[4] Athanasios Papoulis. Probability, Random Variables, and Stochastic Processes. McGraw Hill, Nueva York, NY, EEUU, tercera edición, 1991.

[5] Michela Becchi. From Poisson processes to self-similarity: a survey of network traffic models. Washington University in St. Louis, Tech. Rep, 2008.

[6] Will E. Leland, Murad S. Taqqu, Walter Willinger, Daniel V. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on networking*, 2(1), 1-15, 1994.

[7] Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers, Keying Ye. Probability and Statistics for Engineers and Scientists. Pearson Education, Boston, MA, EEUU, novena edición, 2012.

[8] Gonzalo R. Arce, Nonlinear Signal Processing: A Statistical Approach, John Wiley and Sons, Inc., Hoboken, NJ, EEUU, 2005.

[9] Gennady Samorodnitsky, Murad S. Taqqu, Stable non–Gaussian random processes. Stochastic models with infinite variance. Chapman & Hall/CRC, Boca Raton, FL, EEUU, 1994.

[10] John P. Nolan. Numerical calculation of stable densities and distribution functions. *Communications in Statistics. Stochastic Models*, 13(4), 759-774, 1997.

[11] Javier Royuela-del-Val, Federico Simmross-Wattenberg, Carlos Alberola-López. Libstable: Fast, Parallel, and High-Precision Computation of α-Stable Distributions in R, C/C++ and MATLAB. *Journal of Statistical Software*, 78(1), 1-25, 2017.

[12] Guillermo Julián-Moreno, Jorge E. López de Vergara, Iván González, Luis de Pedro, Javier-Royuela-del-Val, Federico Simmross-Wattenberg. Fast parallel α-stable distribution function evaluation and parameter estimation using OpenCL in GPGPUs. *Statistics and Computing, Springer*, 27(5), 1365-1382, 2017.

[13] Federico Simmross-Wattenberg, Marcos Martín-Fernández, Pablo Casaseca-de-la-Higuera, Carlos Alberola-López. Fast calculation of alpha-stable density functions based on off-line precomputations. Application to ML parameter estimation. *Digital Signal Processing*, 38, 1-12, 2015.

[14] Barbara Chapman, Gabriele Jost, Ruud van der Pas. Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation). The MIT Press, 2007.

[15] Peter Pacheco. An Introduction to Parallel Programming. Morgan Kaufmann Publishers Inc., San Francisco, CA, EEUU, 2011.

[16] OpenMP Architecture Review Board. OpenMP Application Programming Interface. Version 4.5. Noviembre 2015. www.openmp.org

[17] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard. Version 3.1. Junio 2015. www.mpi-forum.org

[18] David B. Kirk, Wen-mei W. Hwu. Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, EEUU, segunda edición, 2013.

[19] CUDA Zone, www.developer.nvidia.com/cuda-zone

[20] OpenCL Official Website, www.khronos.org/opencl

[21] Anestis Karasaridis, Dimitrios Hatzinakos, Network heavy traffic modeling using α-stable self-similar processes. *IEEE Transactions on Communications*, 49(7), 1203-1214, 2001.

[22] L. Rizo-Dominguez, D. Munoz-Rodriguez, C. Vargas-Rosales, D. Torres-Roman, J. Ramirez-Pacheco. RTT Prediction in Heavy Tailed Networks, *IEEE Communications Letters*, 18(4), 700-703, 2014.

[23] Esteban Carisimo, Sebastián Grynberg, José Ignacio Alvarez-Hamelin. Revisiting RTT models. In 6th PhD School on Traffic Monitoring and Analysis (TMA), Belgium, Louvain La Neuve, April 2016.

[24] Victor Moreno, Pedro M. Santiago Del Río, Javier Ramos, David Muelas, José Luis Garcia-Dorado, Francisco J. Gomez-Arribas, Javier Aracil. Multi-granular, multi-purpose and multi-Gb/s monitoring on off-the-shelf systems. *International Journal of Network Management*, 24(4), 221-234, 2014.

[25] Peter Hall. A comedy of errors: the canonical form for a stable characteristic function. Bulletin of the London Mathematical Society, 13, 23-27, 1981.

[26] MATLAB. Statistics and Machine Learning Toolbox. https://www.mathworks.com/products/statistics.html

[27] Federico Simmross-Wattenberg, Marcos Martín-Fernández, Pablo Casaseca-de-la Higuera, Carlos Alberola-López. Fast calculation of stable density functions based on off-line precomputations. Sample implementation, http://es.mathworks.com/matlabcentral/fileexchange/44576-fast-calculation-of-stable-density-functions-based-on-off-line-precomputations

[28] Szymon Borak, Adam Misiorek, Rafał Weron. Models for Heavy-Tailed Asset Returns. In *Statistical Tools for Finance and Insurance*, 21–55. Springer-Verlag, Berlin, 2011.

[29] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. Computational Geometry: Algorithms and Applications. Springer-Verlag, Santa Clara, CA, EEUU, tercera edición, 2008.

[30] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery. Numerical Recipes: The Art of Scientific Computing. Cambridge University Press, New York, NY, EEUU, tercera edición, 2007.

[31] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Patrick Alken, Michael Booth, Fabrice Rossi, Rhys Ulerich. GNU Scientific Library Reference Manual. Version 2.3. Diciembre 2016. www.gnu.org/software/gsl

[32] Jorge Nocedal, Stephen J. Wright. Numerical Optimization. Springer Series in Operations Research, Springer-Verlag, Nueva York, NY, EEUU, segunda edición, 2006.

[33] John A. Nelder, Roger Mead. A simplex method for function minimization. *The Computer Journal*, 7(4), 308-313, 1965.

[34] J. Huston McCulloch. Simple Consistent Estimators of Stable Distribution Parameters. *Communications in Statistics - Simulation and Computation*, 15(4), 1109-1136, 1986.

[35] GCC, the GNU Compiler Collection. https://gcc.gnu.org/

[36] Ananth Grama, George Karypis, Anshul Gupta, Vipin Kumar. Introduction to Parallel Computing. Addison-Wesley, segunda edición, 2003.

[37] Gene M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *AFIPS Conference Proceedings*, 30, 483-485, 1967.

[38] John L. Gustafson. Reevaluating Amdahl's Law. *Communications of the ACM*, 31(5), 532-533, 1988.

[39] Callgrind: a call-graph generating cache and branch prediction profiler. http://valgrind.org/docs/manual/cl-manual.html

[40] KCachegrind, Call-Graph Viewer. http://kcachegrind.sourceforge.net/html/Home.html

[41] OpenACC. More Science, Less Programming. https://www.openacc.org/

[42] Donghoon Lee, Matthew Wiswall. A parallel implementation of the simplex function minimization routine. Computational Economics, 30(2), 171-187, 2007.

Anexo. Evolución de los parámetros

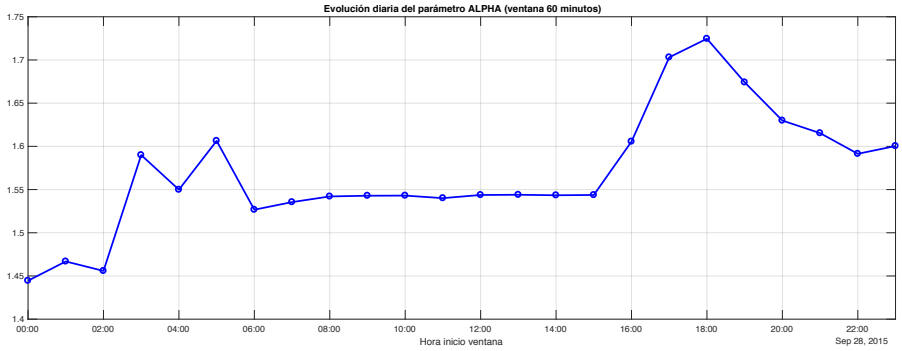


Figura A.1 - Evolución diaria del parámetro α aplicando una ventana de 60 minutos.

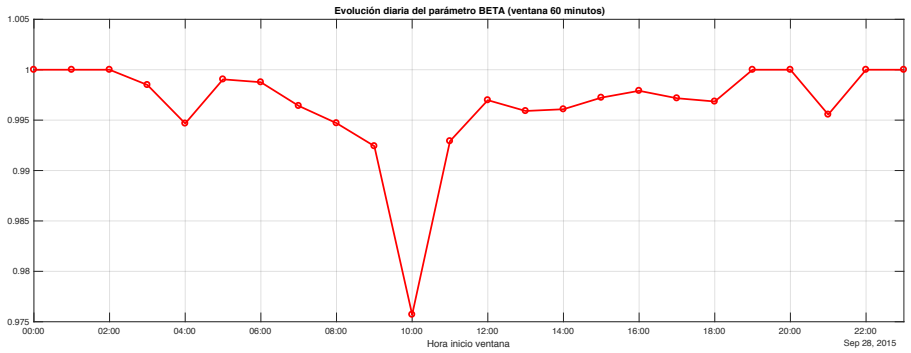


Figura A.2 - Evolución diaria del parámetro β aplicando una ventana de 60 minutos.

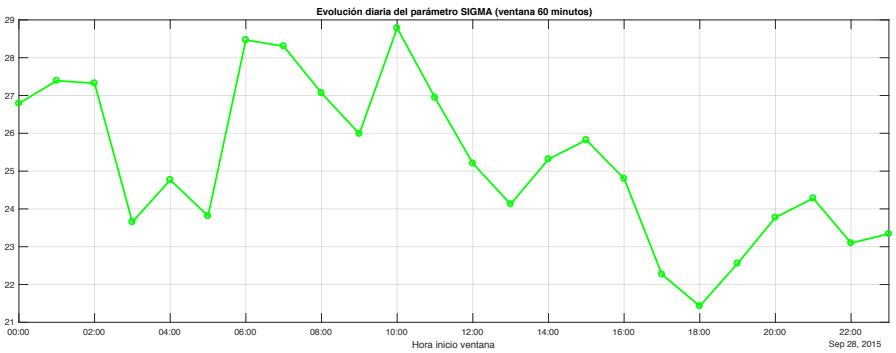


Figura A.3 - Evolución diaria del parámetro σ aplicando una ventana de 60 minutos.

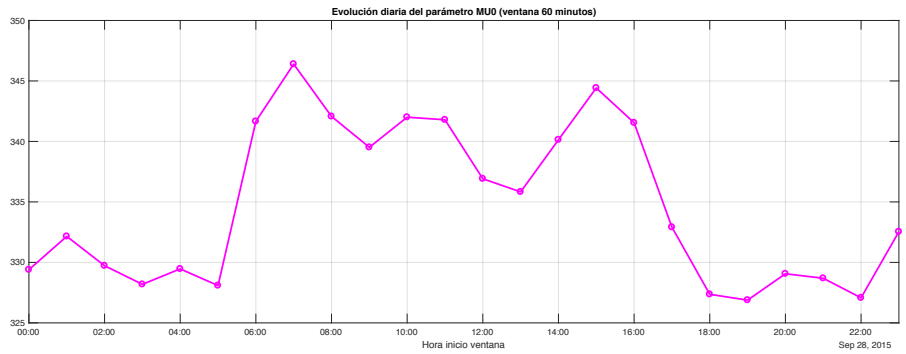


Figura A.4 - Evolución diaria del parámetro μ_0 aplicando una ventana de 60 minutos.

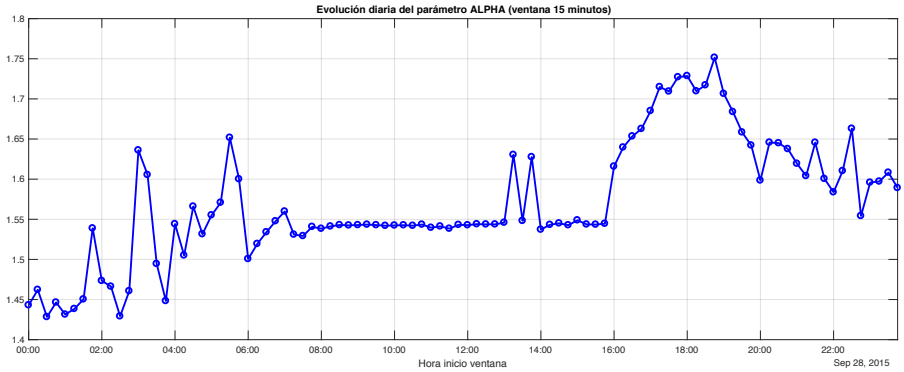


Figura A.5 - Evolución diaria del parámetro α aplicando una ventana de 15 minutos.

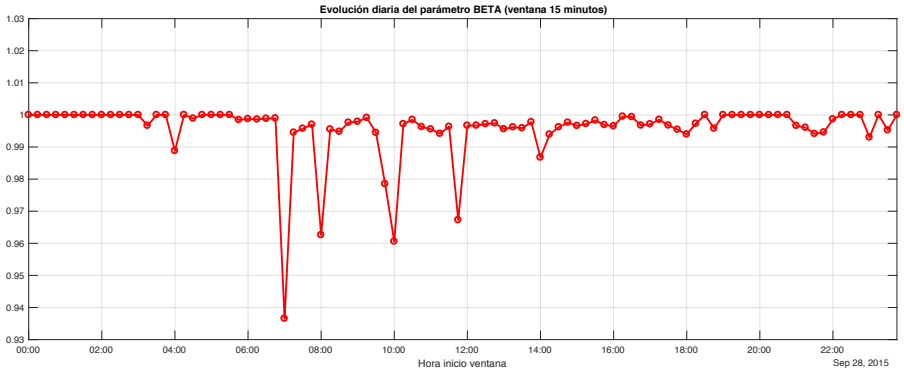


Figura A.6 - Evolución diaria del parámetro β aplicando una ventana de 15 minutos.

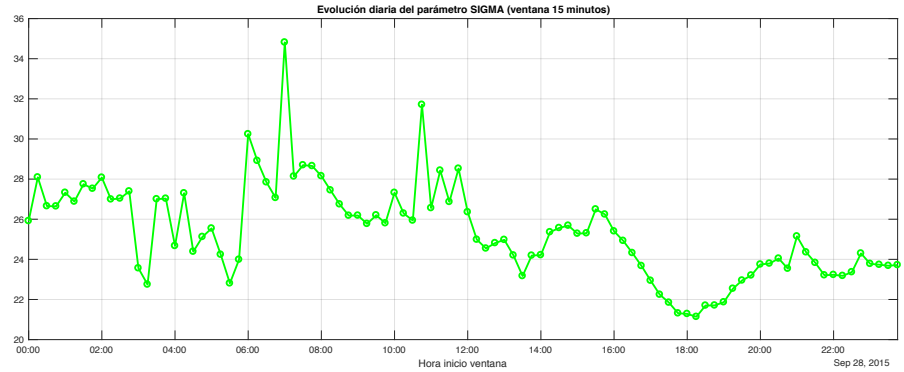


Figura A.7 - Evolución diaria del parámetro σ aplicando una ventana de 15 minutos.

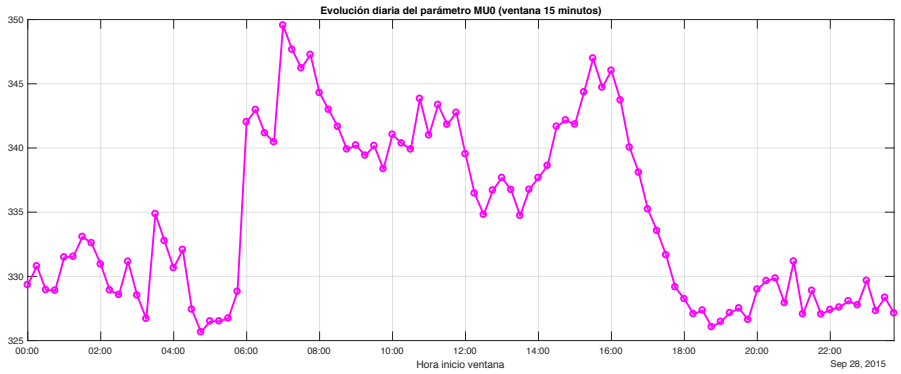


Figura A.8 - Evolución diaria del parámetro μ_0 aplicando una ventana de 15 minutos.

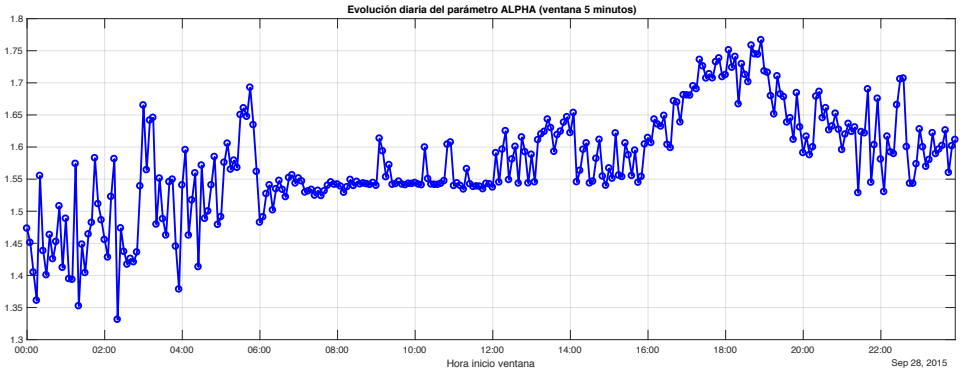


Figura A.9 - Evolución diaria del parámetro α aplicando una ventana de 5 minutos.

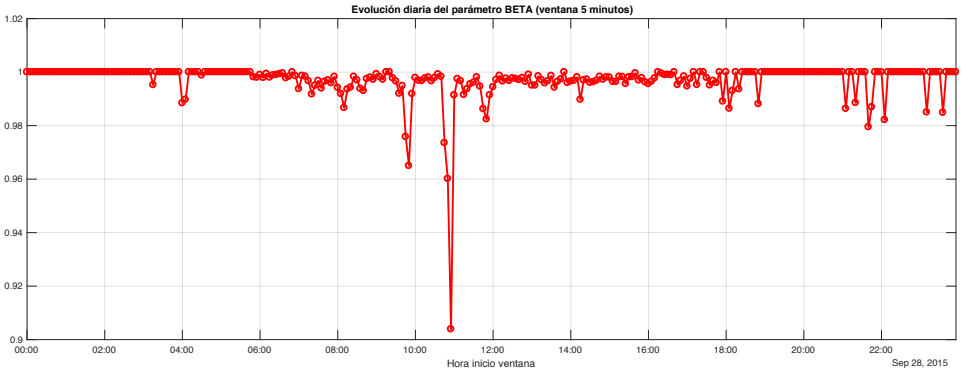


Figura A.10 - Evolución diaria del parámetro β aplicando una ventana de 5 minutos.

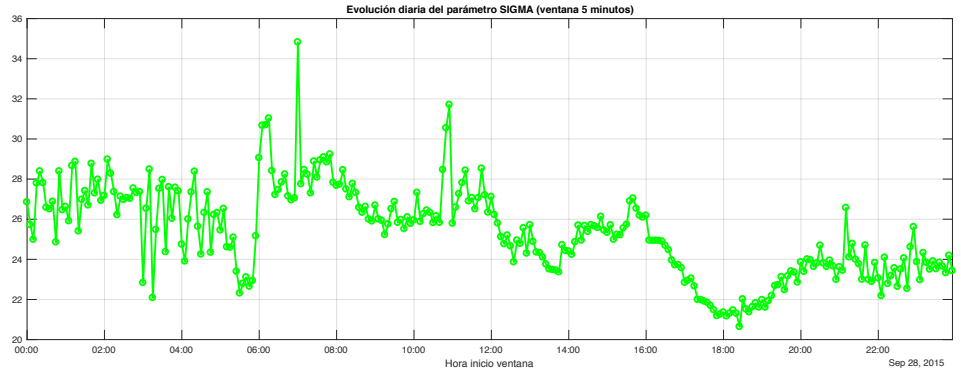


Figura A.11 - Evolución diaria del parámetro σ aplicando una ventana de 5 minutos.

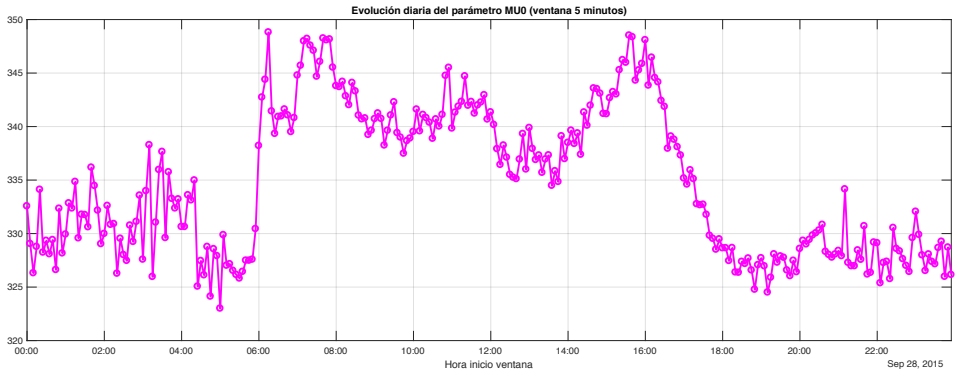


Figura A.12 - Evolución diaria del parámetro μ_0 aplicando una ventana de 5 minutos.

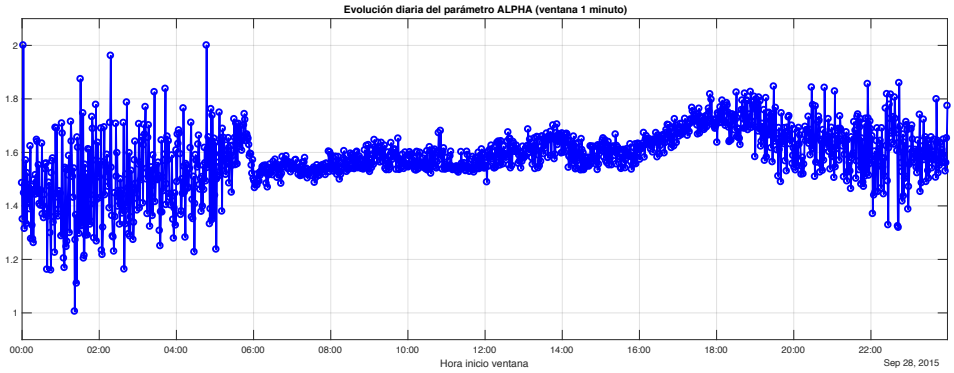


Figura A.13 - Evolución diaria del parámetro α aplicando una ventana de 1 minuto.

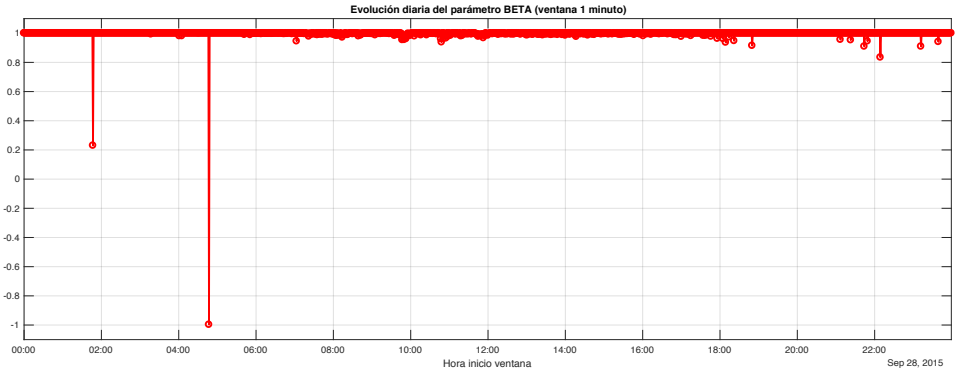


Figura A.14 - Evolución diaria del parámetro β aplicando una ventana de 1 minuto.

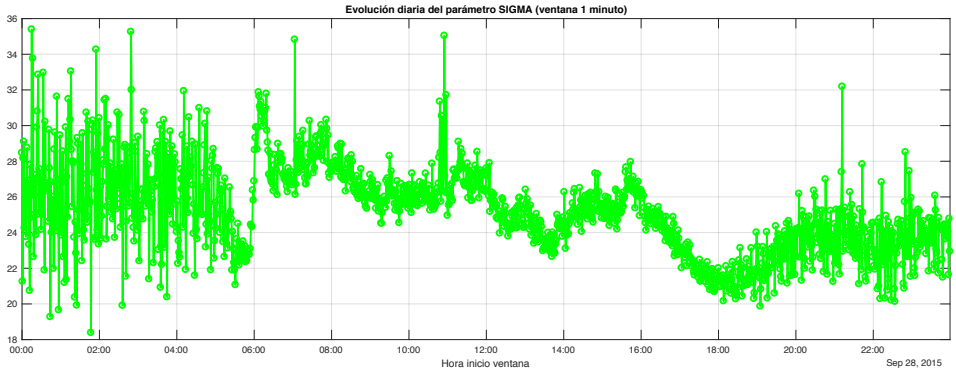


Figura A.15 - Evolución diaria del parámetro σ aplicando una ventana de 1 minuto.

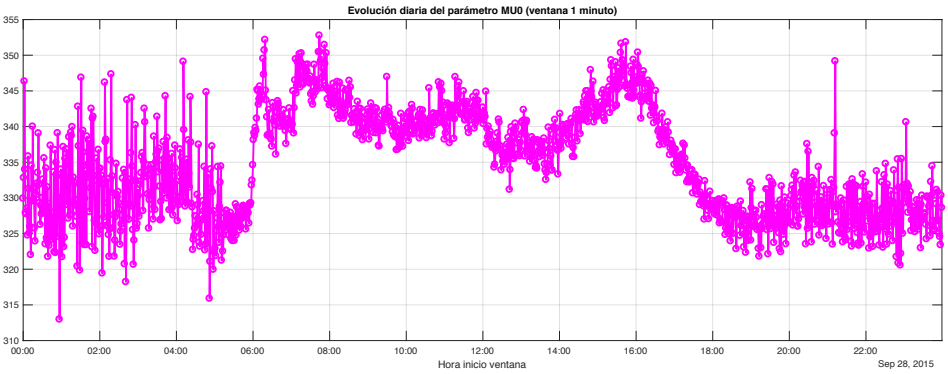


Figura A.16 - Evolución diaria del parámetro μ_0 aplicando una ventana de 1 minuto.