Departamento de Arquitectura de Computadores
Universidad de Málaga

TESIS DOCTORAL

Planificación de recursos en un sistema distribuido de VoD

Sonia González Navarro

Málaga, Abril 2006

Dr. Dña. M.ª Angeles González Navarro
Titular del Departamento
de Arquitectura de Computadores
de la Universidad de Málaga

Dr. D. Juan López Gómez
Titular del Departamento de
Arquitectura de Computadores
de la Universidad de Málaga

CERTIFICAN:

Que la memoria titulada *"Planificación de recursos en un sistema distribuido de VoD"*, ha sido realizada por Dña. Sonia González Navarro bajo nuestra dirección en el Departamento de Arquitectura de Computadores de la Universidad de Málaga y constituye la Tesis que presenta para optar al grado de Doctor en Ingeniería Informática.

Málaga, 9 de Marzo de 2006

Fdo: Dr. Dña. M.ª Angeles González Navarro
Codirectora de la Tesis Doctoral

Fdo: Dr. D. Juan López Gómez
Codirector de la Tesis Doctoral

Fdo: Dr. D. Emilio López Zapata
Director del Departamento de
Arquitectura de Computadores
de la Universidad de Málaga

A Francisco
A mi madre

Agradecimientos

Cuando ya estoy concluyendo este trabajo, me doy cuenta de que la parte más difícil es precisamente ésta: la de mirar hacia atrás y pensar en toda la gente que ha estado a mi lado hasta este momento.

En primer lugar, quisiera agradecer a D. Emilio López Zapata el que me diera la oportunidad de realizar este trabajo. Sin su confianza en mí y su apoyo no hubiera sido posible que yo hubiera llegado hasta aquí.

En segundo lugar, tengo que agradecerle mucho a mis dos directores de Tesis: Mª Angeles González y Juan López. A Mª Angeles por estar siempre dispuesta a enseñarme todo lo que yo no sabía, por su dedicación y sus buenos consejos para formarme como persona y como profesional. Ella es un ejemplo a seguir y puedo decir que soy quién soy gracias a ella. También tengo que agradecerle su infinita paciencia y el aguantarme en los momentos difíciles. Simplemente gracias.

Otra persona muy importante es Juan López. A él tengo también que agradecerle muchas cosas: su confianza, sus buenos consejos, su apoyo y su ánimo constante. Creo que ha sido un regalo el haberle tenido a mi lado y tengo que decir que más que un amigo, ha llegado a ser como un padre. Gracias por tu cariño.

Tengo que agradecerle también a Emilio el que me diera la oportunidad de conocer y trabajar con toda la gente maravillosa del departamento de Arquitectura de Computadores. Especialmente Rafa Asenjo que me ha ido animado y aconsejando por el camino; también a Francisco Corbera por su ayuda desinteresada, Gerardo, Mario, Eladio, Julián, Oscar, Mª Antonia, Sergio Romero, Luis Felipe, Guillermo, Andrés, Manuel Sánchez, Hormigo, Chema, Oswaldo, Manolo Ujaldón, Rafa Larrosa, Nico, Julio, Pablo y Pepe. Y cómo no, mis niñas: Mª Carmen, Magda y Carmen Donoso, es todo un lujo teneros de amigas y compañeras. No puedo dejar de mencionar a las nuevas incorporaciones: Sergio Ramírez, Adrián, Rosa, Alejandro, Antoliano, Antonio y a las personas que se han ido, pero que ha sido un placer conocer: Olga Pérez, Manolo Hidalgo, Mª Angeles Barragán, Jorge y Eva (actualmente en la Universidad de Granada). Aparte tengo que mencionar de nuevo a Eladio al que tengo que agradecer el que siempre esté dispuesto a ayudar en todo. Muchas gracias por tu soporte en Latex.

Quisiera agradecer asimismo el apoyo constante a mi familia. A mis padres Antonio y Pepa que siempre nos han impulsado a superarnos cada día. Mis

hermanos Mª Angeles, Juan, Jose y Noe, por su cariño y por todo lo que representan para mí. Sé que siempre estarán para todo.

A mis primas Belén y Mª José, buenas amigas y que siempre han estado ahí. Mis amigos Rocío, Raquel, Eva, Luciano y Juan Antonio, por su apoyo y por los momentos tan maravillosos. Juanmi que siempre me ha apoyado y que siempre intenta hacerme reír. A Jesús y Juanfe por sus ánimos en algunos momentos en los que mi confianza flaqueaba. Marta y Óscar que me animan desde Barcelona. La gente del Cañuelo con los que he pasado momentos buenos y la familia Guerrero López todo un placer el poder haberlos conocido.

A la familia Plaza Rosado, por haberme considerarme parte de la familia y darme su cariño.

Por último quiero mencionar a la persona más importante para mí. Mi pareja Francisco, que aparte de haber tenido el ánimo de leerse la tesis y decir que se ha enterado de algo, ha estado al pie del cañón y me ha aguantado durante todo este tiempo. Gracias por tu compresión y tu amor.

Índice

Índice de Figuras VI

Índice de Tablas VII

Prefacio IX

1.- Introducción **1**

1.1. Sistemas de vídeo bajo demanda 1

1.2. Descripción del sistema VoD propuesto 14

1.3. Distribución de los vídeos: popularidad y replicación parcial . . 16

1.4. Objetivos de esta tesis 24

2.- El algoritmo PRLS **27**

2.1. Introducción 27

2.2. Compartición de la carga en sistemas distribuidos de VoD . . . 30

2.3. Modelo analítico 33

2.4. Resultados experimentales 46

2.5. Conclusiones del capítulo 55

3.- Algoritmos Multicast: LTH y RLTH **57**

3.1. Introducción 57

3.2. Los algoritmos MFQL y Multicast con umbral 60

3.3. Nuestros algoritmos 64

3.4. Umbral óptimo 68

3.5. Modelo Analítico 72

3.6. Resultados experimentales 82

3.7. Conclusiones del capítulo 94

4.- Abandono en el sistema **97**

4.1. Introducción 97

4.2. El algoritmo BITM 99

4.3. Umbral óptimo con el algoritmo BITM 102
4.4. Resultados experimentales 109
4.5. Conclusiones del capítulo 117

Conclusiones **119**

Glosario **125**

Bibliografía **129**

Índice de figuras

1.1. Sistema de VoD. 3

1.2. Sistema centralizado de VoD. 11

1.3. Sistema centralizado con servidores de VoD. 12

1.4. Sistema distribuido de VoD. 12

1.5. Sistema con proxy. 13

1.6. Sistema distribuido de VoD. 15

1.7. Probabilidad de solicitar el vídeo i ($i = 1, ..., 10$) para distintos valores de ξ según la distribución Zipf. 17

1.8. Variación de la carga en el sistema a lo largo de un día laborable. 18

1.9. Ejemplo de distribución de los vídeos. 19

1.10. Distribución cíclica de los vídeos no populares. 20

1.11. Replicación moderada (R.M.) de los $M = 100$ vídeos ofertados por un sistema de 16 servidores: R.M. del 20 %. 21

1.12. Incremento de F_i en un sistema de 16 servidores que oferta $M = 100$ vídeos: (a) cuando se incrementa el porcentaje de replicación (con $\xi = 1$ fijo); (b) cuando aumenta el grado de popularidad ($\xi = 0{,}5$, $\xi = 1$ y $\xi = 1{,}8$) para los distintos porcentajes de replicación. 23

1.13. Probabilidad de solicitar un vídeo no local según distintas estrategias de distribución y distintos valores de ξ: (a) $\xi = 1$; (b) $\xi = 1{,}8$. 24

2.1. Tiempos de espera para los algoritmos GWQ y PRLS. 33

2.2. Sistema de cola cerrado. 36

2.3. Resultados del rendimiento del algoritmo PRLS y el modelo analítico para el 100 % de replicación: (a) cuando $\xi = 1$; (b) cuando $\xi = 1{,}8$. El eje X representa el número de servidores, y el eje Y son los tiempos de espera en segundos. 48

2.4. Resultados del rendimiento del algoritmo PRLS y el modelo analítico para el 75 % de replicación: (a) cuando $\xi = 1$; (b) cuando $\xi = 1{,}8$. El eje X representa el número de servidores, y el eje Y son los tiempos de espera en segundos. 49

2.5. Resultados del rendimiento del algoritmo PRLS y el modelo analítico para el 50 % de replicación: (a) cuando $\xi = 1$; (b) cuando $\xi = 1{,}8$. El eje X representa el número de servidores, y el eje Y son los tiempos de espera en segundos. 50

2.6. Comparación de N^{min}_{stream} frente a $N_{stream} = 50$ (línea sólida) para el 75 % y el 50 % de replicación: (a) cuando $\xi = 1$; (b) cuando $\xi = 1{,}8$. El eje X representa el número de servidores, y el eje Y está en Mbps. 52

2.7. Comparación de B^{min}_{link} frente a $B_{link} = 155\ Mbps$ (línea sólida) para el 75 % y el 50 % de replicación: (a) cuando $\xi = 1$; (b) cuando $\xi = 1{,}8$. El eje X representa el número de servidores, y el eje Y está en Mbps. 52

2.8. Comparación de B^{min}_{switch} frente a $B_{switch} = 1\ Gbps$ (línea sólida) para el 75 % y el 50 % de replicación: (a) cuando $\xi = 1$; (b) cuando $\xi = 1{,}8$. El eje X representa el número de servidores, y el eje Y está en Mbps. 53

2.9. Resultados del rendimiento cuando se incrementa sólo un parámetro del sistema ($B_{link}, B_{switch}, N_{stream}$): (a) 20 % de incremento; (b) 50 % de incremento. El eje X representa el número de servidores, y el eje Y son los tiempos de espera en segundos. 54

3.1. Planificación de servicios con el algoritmo MFQL. 61

3.2. Multicast con umbral. 62

3.3. Nuestro algoritmo multicast LTH. 65

3.4. Algoritmo LTH: En el instante t_s una cola remota del vídeo i, se planifica en el servidor S_k y el servidor S_j la sirve con un stream completo remoto. Posteriormente, en el instante t_{s+l}, llega al servidor S_k una nueva petición al vídeo i que es puesta en cola. 66

3.5. Algoritmo RLTH: En el instante t_s una cola remota del vídeo i, se planifica en el servidor S_k y el servidor S_j la sirve con un stream completo remoto. Posteriormente, en el instante t_{s+l}, $t_{s+l} \leq t_s + U_i$, llega al servidor S_n una nueva petición al vídeo i la cual es servida con un stream parcial por el servidor S_j. . . 67

3.6. Sensibilidad del Umbral para el vídeo más popular cuando $\xi = 1$ y M = 100. 71

3.7. Esquema con los principales parámetros del servidor. 74

3.8. Resultados del rendimiento de los algoritmos MFQL, LTH y RLTH. El eje X representa el número de servidores en el sistema. El eje Y representa el tiempo medio de espera en segundos. Porcentajes de replicación: (a) 75 %, (b) 50 %, (c) 25 %. En todos los casos $\xi = 1$. 85

3.9. Comportamiento de nuestro algoritmo y precisión del modelo analítico cuando: (a) $N_{term} = 150$; (b) $N_{term} = 200$; (c) $N_{term} = 300$. El eje X representa el número de canales físicos en el servidor, y el eje Y el tiempo de espera en segundos. 88

3.10. Tiempo de espera estimado cuando λ varía; T_{sleep} y T_{active} son 7200 segundos. 90

3.11. Resultados del rendimiento del algoritmo RLTH y del modelo analítico cuando: (a) $N_{term} = 150$; (b) $N_{term} = 200$; (c) $N_{term} = 300$. Porcentaje de replicación: 25 %, $\xi = 1$, $B_{link} = 622$ Mbps y $B_{switch} = 16$ Gbps. El eje X representa el número de servidores, y el eje Y son los tiempos de espera en segundos. 92

3.12. Resultados del rendimiento del algoritmo RLTH y el modelo analítico. Porcentaje de replicación 15 %, $\xi = 1$, $N_{term} = 300$, $B_{link} = 622$ Mbps, $B_{switch} = 16$ Gbps; T_{sleep} y T_{active} son 7200 segundos. 93

4.1. Nuestro algoritmo BITM. 101

4.2. Número de mini-grupos. 102

4.3. Umbral óptimo con el algoritmo BITM y el algoritmo RLTH con abandono. Parámetros: $M = 100$ vídeos, duración media de los vídeos 7200 seg, $\lambda_i = 0{,}102$ pet/seg y $B_{stream} = 15$ Mbps. . 107

4.4. Cota inferior del ancho de banda medio utilizado para el vídeo más popular con el algoritmo BITM y el algoritmo RLTH con abandono. Parámetros: $M = 100$ vídeos, duración media de los vídeos 7200 seg, $\lambda_i = 0{,}102$ pet/seg y $B_{stream} = 15$ Mbps. . . . 108

4.5. Probabilidad de abandono $P_i(t)$: fracción de peticiones al vídeo i que abandonan el sistema en un intervalo de longitud t. 109

4.6. Tiempos de espera del algoritmo BITM y de la versión del algoritmo RLTH con abandono cuando cada servidor tiene una capacidad de servicio de $N_{stream} = 50$ streams. El eje X representa el número de servidores en el sistema, y el eje Y representa el tiempo de espera en segundos. 111

4.7. Tiempos de espera del algoritmo BITM y de la versión del algoritmo RLTH con abandono cuando cada servidor tiene una capacidad de servicio de $N_{stream} = 150$ streams. El eje X representa el número de servidores en el sistema, y el eje Y representa el tiempo de espera en segundos. El porcentaje de replicación es del 25 %. 114

4.8. Tiempos de espera del algoritmo BITM y de la versión del algoritmo RLTH con abandono cuando cada servidor tiene una capacidad de servicio de $N_{stream} = 200$ streams. El eje X representa el número de servidores en el sistema, y el eje Y representa el tiempo de espera en segundos. El porcentaje de replicación es del 25 %. 114

4.9. Tiempos de espera del algoritmo BITM y de la versión del algoritmo RLTH con abandono cuando cada servidor tiene una capacidad de servicio de $N_{stream} = 250$ streams. El eje X representa el número de servidores en el sistema, y el eje Y representa el tiempo de espera en segundos. El porcentaje de replicación es del 25 %. 115

Índice de tablas

1.1. Formatos de compresión de vídeo y sus requisitos de ancho de banda. 5

2.1. Parámetros del modelo analítico del algoritmo PRLS. 38

3.1. Parámetros del modelo analítico del algoritmo RLTH. 73

3.2. Tiempos medios de espera, en segundos, cuando el porcentaje de replicación es del 25 % y los anchos de banda de la red son: $B_{link} = 155$ Mbps y $B_{switch} = 1000$ Mbps. 86

3.3. Tiempos medios de espera, en segundos, cuando el porcentaje de replicación es del 25 % y los anchos de banda de la red son: $B_{link} = 185$ Mbps y $B_{switch} = 1200$ Mbps. 87

3.4. Comparación de los tiempos de espera obtenidos mediante simulaciones y analíticamente. Fijamos $N_{term} = 300$. 90

4.1. Porcentaje de abandono del algoritmo BITM y de la versión del algoritmo RLTH con abandono: las filas impares representan el porcentaje de abandono de los vídeos más populares (**M.P.**) y las filas pares el porcentaje de abandono de los vídeos no replicados (**N.R.**). Las dos primeras columnas representan los porcentajes de abandono con un 75 % de replicación en el sistema; Las dos siguientes, los porcentajes con un 50 %; Las dos últimas, los correspondientes porcentajes de abandono con un 25 % de replicación en el sistema. 113

4.2. Porcentaje de abandono del algoritmo BITM y de la versión del algoritmo RLTH con abandono: las filas impares representan el porcentaje de abandono de los vídeos más populares (**M.P.**) y las filas pares el porcentaje de abandono de los vídeos no replicados (**N.R.**). Las dos primeras columnas representan los porcentajes de abandono con $N_{stream} = 150$; Las dos siguientes, con $N_{stream} = 200$; Las dos últimas, los correspondientes porcentajes de abandono con $N_{stream} = 250$. 116

Prefacio

En la actualidad hay un creciente interés en el diseño y explotación de sistemas de vídeo bajo demanda (VoD o *Video on Demand*). Un sistema de VoD consta, de manera general, de una amplia población de clientes dispersos, de proveedores de servicios y programas y de redes de interconexión. La finalidad de un sistema de VoD es ofrecer a los clientes los contenidos que demanden en el instante en el que lo soliciten. Otra característica que debe tener un sistema de VoD es la de ofrecer interactividad al usuario.

Hoy en día, el cliente para poder interactuar con el sistema necesita utilizar un canal de servicio (o *stream*) de forma exclusiva. Es decir, el sistema debe implementar una estrategia *unicast* para la administración de los recursos. Sin embargo si la población es numerosa, el coste del sistema se puede disparar si se quiere ofrecer servicios interactivos.

Una solución económicamente viable para este tipo de sistemas es diseñar un sistema distribuido en el que varios servidores de vídeo se conectan a través de una red específica, habitualmente conocida como *Content Distribution Network* o CDN.

El problema de la arquitectura basada en redes CDN es que los contenidos deben estar almacenados en todos los servidores, es decir, se debe copiar todos los contenidos en todos los servidores (lo que nosotros llamamos *replicar*). Obviamente replicar todos los contenidos requiere una alta capacidad de almacenamiento en cada servidor. Además, las redes CDN existentes están optimizadas para transmitir contenido de tipo Web mientras que nosotros estamos interesados en servicios de VoD. Por tanto, si se tiene que almacenar una copia de todos los vídeos -los cuales están codificados en MPEG-2- en todos los servidores, se necesitaría una gran cantidad de almacenamiento (por ejemplo, un vídeo de dos horas de duración, usando una compresión MPEG-2 HTDV (televisión de alta definición), requiere aproximadamente 18 Gigabytes). Una forma de abaratar el sistema sin degradar sus prestaciones consiste en no almacenar todos los vídeos en todos los servidores, sino sólo aquellos que sean más populares. Algunas propuestas utilizan arquitecturas en las que el control y los contenidos se distribuyen jerárquicamente a través de los servidores. De hecho, existen soluciones comerciales que permiten implementar arquitecturas centralizadas o distribuidas con distintas topologías.

En arquitecturas distribuidas es necesario el diseño de algoritmos eficientes de planificación de los recursos, que es precisamente el objeto de estudio de esta tesis. Una de las cuestiones a resolver por estos algoritmos es la de establecer

un mecanismo de asignación de peticiones y de compartición de la carga, que permita que aquellos servidores sobrecargados puedan dirigir las peticiones de los clientes a algún servidor con capacidad disponible. Existen propuestas en las que se asignan esas peticiones a un servidor central que almacena todos los vídeos, populares y menos populares. Sin embargo, normalmente los sistemas centralizados son poco escalables y poco robustos, ya que existe un único punto de fallo (el servidor central).

Otra cuestión a tener en cuenta es cuando el número de servidores en el sistema es elevado y el porcentaje de vídeos replicados es pequeño. En estos casos se genera un gran tráfico en la red de interconexión, que actúa como verdadero cuello de botella del sistema lo que repercute en la espera del servicio de los usuarios. Por tanto, el uso eficiente del ancho de banda de la red y del servidor juega un papel crucial.

Para reducir los tiempos de espera y el uso del ancho de banda de la red se utilizan técnicas de *multicast*. Sin embargo, aunque con estas técnicas se mejora el rendimiento del sistema lo hacen a costa de perder cierto grado de interactividad con el contenido. En la literatura se han propuesto numerosas aproximaciones que combinan distintas estrategias multicast para optimizar el uso del ancho de banda disponible. La mayoría de estas aproximaciones tienen como principal objetivo el de calcular y optimizar el ancho de banda. Para ello suponen siempre que los recursos disponibles son ilimitados. Sin embargo, en la realidad los recursos de los sistemas que se implementan son finitos.

Con el planteamiento de diseñar un sistema de VoD que sea de bajo coste y lo más real posible, en esta tesis proponemos un sistema distribuido de servidores de VoD donde los vídeos más populares están replicados en todos los servidores del sistema y los menos populares están distribuidos entre todos ellos. El objetivo es minimizar un parámetro de calidad bastante importante para los clientes, a la vez que se procura optimizar los recursos disponibles en el sistema. El parámetro de calidad mencionado es el tiempo de espera que percibe el cliente, es decir, el tiempo que transcurre desde que el cliente hace la petición hasta que el sistema empieza a servir el vídeo. Por tanto, la métrica principal de medida de prestaciones que consideramos en este tipo de sistemas es el tiempo de espera. Otro parámetro de calidad del sistema es el grado de satisfacción del cliente. El grado de satisfacción viene dado por la cantidad de usuarios que entran en el sistema y reciben el servicio. Si la espera del servicio es demasiado grande, pueden decidir abandonar el sistema sin haberlo recibido, con lo que el grado de satisfacción decrece. Analizaremos el comportamiento del sistema para distintos porcentajes de replicación cuando:

1. la estrategia de planificación de los recursos es de tipo unicast,
2. en el caso en el que sea más rentable utilizar una técnica multicast y
3. asumiendo que los clientes pueden abandonar el sistema.

En cualquiera de los casos, desarrollamos un modelo analítico que refleje las

características del sistema con los distintos algoritmos de planificación que proponemos en esta tesis. La importancia de estos modelos es que nos ayudan a la hora de analizar el comportamiento de los algoritmos y del sistema. Así mismo, son una herramienta útil, ya que nos sirven para estimar qué parámetro (y cuándo) va a ser el cuello de botella del sistema. Otra aplicación de los modelos es que nos pueden ayudar a estimar, en cuestión de segundos, el tiempo medio de espera para los distintos porcentajes de replicación, teniendo en cuenta la influencia de la popularidad de los vídeos. Estas estimaciones podrían ser utilizadas para decidir cuál sería el porcentaje de replicación apropiado de forma que garantice un tiempo de espera lo suficiente pequeño y razonable para los usuarios. Además, el modelo analítico puede ayudarnos para seleccionar el tamaño de la red, el número óptimo de servidores que debe tener el sistema y con el que se logre un tiempo de espera pequeño, y para predecir cuando la red se convierte en el cuello de botella.

Comenzamos nuestro trabajo describiendo brevemente en el Capítulo 1, los distintos elementos que componen un sistema de vídeo bajo demanda así como las distintas arquitecturas propuestas para diseñar un sistema de VoD. Veremos que, de entre las distintas propuestas, es siempre preferible una arquitectura distribuida a una centralizada debido a cuestiones de escalabilidad y fiabilidad. Así mismo, fijaremos las distintas características del sistema de VoD que proponemos y presentaremos y discutiremos la distribución que realizamos de los contenidos entre los distintos servidores que componen el sistema.

Una vez establecidas las características del sistema, en el Capítulo 2 proponemos un algoritmo unicast, que se encarga del control y compartición de la carga en el sistema. Además, presentamos un modelo analítico del algoritmo que captura el rendimiento del algoritmo y que nos permite relacionar dos factores y evaluar cuantitativamente el impacto que tendrán éstos en el tiempo de espera final del sistema: la popularidad de los vídeos y el porcentaje de replicación, es decir, cuantos vídeos van a ser replicados en todos los servidores. La validez del modelo así como el rendimiento del algoritmo propuesto se evalúan al final del capítulo a través de una serie de experimentos. Comprobaremos que, en casos de baja replicación y número elevado de servidores nos encontramos con una situación crítica, al convertirse la red en el cuello de botella de nuestro sistema.

Para optimizar el uso de la red y conseguir reducir el tiempo de espera, en el Capítulo 3 desarrollamos un algoritmo de planificación que combina los beneficios de distintas estrategias multicast (*batching* y *patching* con umbral). Al igual que en el caso unicast, desarrollamos y validamos un modelo analítico de nuestro algoritmo que nos permite resolver entre otras cuestiones que nos surgen al aplicar estas técnicas la siguiente: ¿Cuál sería ese umbral de tiempo que optimice el tiempo de espera y el ancho de banda en nuestro sistema?. Veremos que ese umbral dependerá de la popularidad del vídeo y del porcentaje de replicación en el sistema. Además, a partir de este modelo podemos estimar el rendimiento de un servidor y obtener expresiones para diseñar el sistema distribuido de forma que asegure que el tiempo de espera no se degrade por debajo de un valor fijado.

Por último, en el Capítulo 4 analizamos la problemática del abandono de los clientes. Para solucionarlo, proponemos un algoritmo que a la vez que optimiza el ancho de banda mantiene una tasa de abandono por debajo de un valor preestablecido por el sistema. Veremos que formando mini-grupos de clientes que se atienden con un mismo stream, se consigue reducir tanto la tasa de abandono como los tiempos de espera aun cuando los recursos del sistema son limitados. Desarrollamos un modelo para obtener los umbrales que se aplican en el algoritmo para conseguir que el uso del ancho de banda sea óptimo y que la tasa de abandono sea la prefijada. Al final del capítulo comprobaremos que el algoritmo consigue no sólo su objetivo, sino que mejora los tiempos de espera cuando se compara con un algoritmo de tipo patching con umbral.

Mencionar que hasta la fecha las publicaciones relacionadas con esta tesis son los trabajos [1], [2], [3], [4], [5], [6], [7] y [8].

Para finalizar, las aportaciones más significativas de esta tesis se exponen en el Capítulo de "Conclusiones y principales aportaciones", donde presentamos un breve resumen de las aportaciones que se han realizado a lo largo de este trabajo y de las líneas de investigación para trabajos futuros.

1 Introducción

1.1. Sistemas de vídeo bajo demanda

En los últimos años se ha producido una explosión en la demanda de nuevos servicios por parte del cliente residencial, debido a las nuevas posibilidades que introducen las tecnologías digitales y a las capacidades introducidas por las nuevas redes de comunicaciones.

Una de las características de la sociedad moderna es que ahora tenemos una mayor preocupación por disfrutar mejor de nuestro tiempo libre. Bien sea porque tenemos más tiempo de ocio para disfrutar o porque es un recurso escaso, la cuestión es que se observa desde hace unos años una necesidad creciente de espacios lúdicos y servicios de entretenimiento.

Entre las nuevas formas de ocio destacan internet y la televisión. A través del primero, los clientes consumen cada vez más contenidos multimedia, como por ejemplo música digital (archivos MP3), películas, juegos en red, etc. Hay que especificar que normalmente este consumo se realiza, en su inmensa mayoría, mediante la descarga gratuita de contenidos procedentes de redes *peer-to-peer*, lo que implica también una baja exigencia tanto en la calidad del contenido como en la velocidad de la descarga. También hay que destacar que es el usuario el que elige *qué* contenido consumir y *cuándo* hacerlo.

En cuanto al consumo de televisión (que sigue siendo el medio de ocio más utilizado), las plataformas de televisión de pago ofertan un mayor número de canales y son las que ofrecen la posibilidad de seleccionar contenidos de entre una serie de propuestas (cine, deportes, noticias, etc). Cuando se trata de elegir una película, el cliente puede también tener la opción de elegir la hora de emisión. Generalmente sólo se le permite elegir una de entre una serie de horas de comienzo de emisión ya predefinidos.

Sin embargo, estamos acostumbrados a utilizar las opciones de un reproductor de vídeo o de DVD doméstico, que permite detener la visualización, avanzar o rebobinar, entre otras opciones. El usuario de una plataforma de televisión no

tiene ningún tipo de control sobre estos contenidos. Simplemente se limita a sintonizar el canal que emite el contenido que le interesa, o acceder a un evento en el instante en el que esté planificado.

Hoy día, casi todos los espectadores quieren ser interactivos (mandan SMS para votar o responder preguntas en algunos programas). Pero les gustaría tener más interacción con los contenidos que visualizan. En otras palabras, existe una demanda latente de nuevos servicios de ocios más interactivos que la televisión convencional.

En los últimos años, el abaratamiento de costes de equipos y la consolidación de antiguos y nuevos estándares, permite la posibilidad de distribuir servicios de vídeo bajo demanda (VoD o *Video on Demand*). El servicio de vídeo bajo demanda permite a los clientes decidir *qué* contenido visualizar y *cuándo* hacerlo. Además, cada cliente tiene la posibilidad de controlar la reproducción de estos contenidos, mediante funciones interactivas (VCR) del tipo reproducir, pausa, rebobinado, avance rápido y parada. Todos estos aspectos son objeto de atención en la bibliografía reciente [9], [10], [11], [12], [13].

Es a partir de la década de los 90 cuando surge, en el entorno de la investigación, el interés por modelar y diseñar sistemas que sean capaces de llevar servicios de VoD al sector residencial [14], [15], [16], [17], [18].

En el diseño de estos sistemas de VoD están implicadas distintas áreas: estudio del comportamiento del usuario (tipos de servicios que demandan, duración del consumo, etc), arquitectura de los sistemas (centralizados, distribuidos, basados en proxies, etc), tipos de redes utilizadas (redes de cable, redes IP, híbridas, etc) protocolos de comunicaciones (TCP, RTP, RTSP, etc), formatos de compresión, algoritmos de gestión y planificación, sistemas de almacenamiento, calidad de servicio (QoS), etc.

En las siguientes secciones vamos a presentar una breve descripción de algunos de los elementos que intervienen en el diseño de un sistema de VoD. Veremos cómo es la estructura de un sistema genérico de VoD, los tipos de servicios que se pueden realizar, los formatos de transmisión empleados, así como algunas técnicas de servicios utilizadas para la planificación de las peticiones y la gestión de los recursos.

1.1.1. Componentes de un sistema de VoD

La estructura genérica de un sistema de VoD consiste, básicamente, en clientes, servidores de vídeo y una red que los conecte (Figura 1.1).

Pero en la creación de un sistema de VoD intervienen, además de los anteriores componentes, otros elementos. Entre los más importantes cabe destacar los siguientes:

- **El proveedor de programas o contenidos**. Esta figura realiza, entre otras, la tarea de negociar la comercialización de los contenidos con los

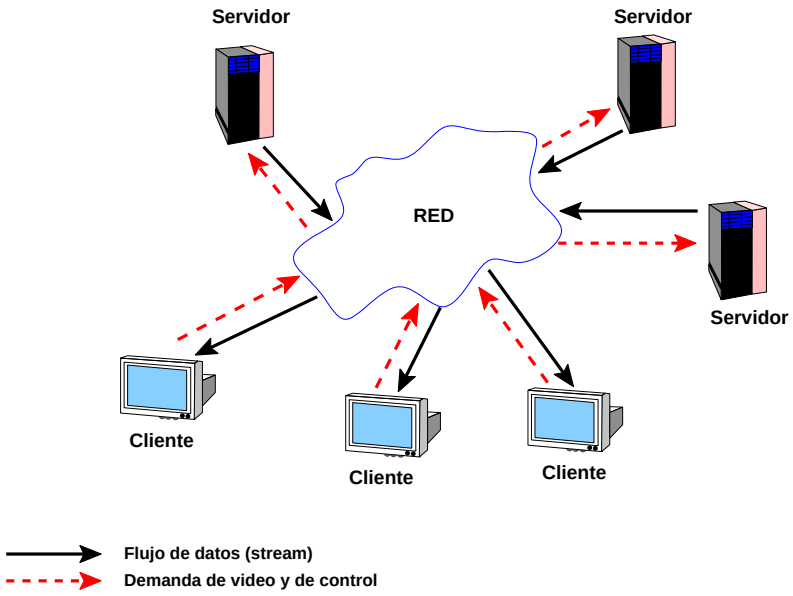


Figura 1.1 Sistema de VoD.

creadores y de proveer una plataforma tecnológica desde la que suministrar los contenidos a los proveedores del servicio.

- **El proveedor del servicio**. Es el encargado de suministrar al cliente final un servicio concreto. Para todos los efectos es la interfaz del servicio con el cliente, el encargado de suministrarle el producto, de atender al cliente y de facturarle por el uso del servicio.
- **El proveedor de red**. Es el encargado de proporcionar la red mediante la cual se distribuyen los contenidos.
- **El proveedor de acceso personalizado**. Es el proveedor del canal empleado por el usuario para solicitar un contenido y enviar las señales de control. También es el proveedor del canal para la distribución de los contenidos. Existen en la actualidad dos tendencias dominantes: las redes de cable y el ADSL (*Asimetric Digital Subscriber Line* o Línea de Abonado Digital Asimétrica).
- **El descodificador o *Set-top-box* (STB)**. Es la interfaz entre el cliente y el sistema. El STB se encarga de descodificar la señal que recibe del servidor y de enviar al servidor la acción seleccionada por el usuario.

De todos los componentes, los clientes son la pieza clave del sistema (como en la mayoría de los sistemas orientados a la realización de servicios). Los clientes realizan peticiones al proveedor de servicios, el cual planifica las peticiones, reserva los recursos, obtiene el vídeo solicitado del proveedor de programas y lo sirve al usuario a través de la red [19], [20]. Además, es deseable que los usuarios puedan interactuar con el servidor mediante algunas de las funciones VCR. Para ello y para otras necesidades que se describirán en el Capítulo 3,

la red del sistema y el STB deben soportar al menos dos canales. Uno por donde se transmita el contenido solicitado, y otro por donde el cliente envíe la petición del contenido y las órdenes de control sobre la sesión (funciones VCR).

En la siguiente sección vamos a ver los tipos de servicios interactivos que puede ofrecer un sistema de VoD.

1.1.2. Tipos de servicios interactivos

Dependiendo del grado de interacción permitido a los usuarios, podemos clasificar los servicios ofrecidos por un sistema de VoD en los siguientes:

- **Broadcast (No VoD)**. Es similar a la emisión de la televisión, en la que el usuario es un participante pasivo y no tiene ningún tipo de control o interacción sobre la sesión.
- **Pago por visión (Pay-per-view, PPV)**. Permite a los usuarios acceder a un programa con derechos específicos, pero a una hora determinada por el proveedor de servicios. En este tipo de servicios no se permite tampoco ningún tipo de interacción.
- **Cuasi vídeo bajo demanda (Quasi-VoD, Q-VoD)**. En este caso, los usuarios pueden decidir el momento en el que desean ver un vídeo. Normalmente, el proveedor de servicios agrupa a los usuarios que deseen ver el mismo vídeo al mismo tiempo y lo transmite a todos ellos. Estos servicios permiten un control rudimentario sobre la reproducción del vídeo (rebobinado, avance), mediante el cambio de grupos. Por ejemplo, se puede simular una función de rebobinado si existe un grupo que esté recibiendo o vaya a recibir el instante del vídeo al que se quiera ir. En este caso, un usuario cuya transmisión esté adelantada en el tiempo, puede rebobinar cambiándose a ese grupo que comenzó más tarde.
- **Vídeo bajo demanda casi a demanda (Near-VoD, N-VoD)**. El proveedor emite un mismo programa en distintos canales y a intervalos de tiempo regulares (del orden de cinco minutos), lo que permite a cualquier usuario comenzar la visualización en un tiempo de espera máximo de cinco minutos. Además, cambiando al canal adecuado se pueden garantizar las funciones de avance y rebobinado del programa.
- **Vídeo bajo demanda verdadero (True-VoD, T-VoD)**. Los usuarios deciden cuándo y qué contenido quieren ver y tienen total control sobre la sesión. La forma de realizar este servicio depende de la estrategia utilizada por el proveedor de servicios.

PPV y Q-VoD son servicios con interacción limitada, pero son los más fáciles de implementar. En estos casos, un descodificador puede filtrar y descodificar

los canales que se transmiten. El servicio T-VoD es, por el contrario, un servicio más complejo de implementar. De ahí el valor que tiene un sistema que implemente vídeo bajo demanda verdadero.

1.1.3. Formatos de codificación

El desarrollo de los sistemas de VoD es posible, entre otros factores, gracias a la evolución de los estándares de codificación.

Un sistema de VoD ofrece a sus usuarios vídeos con alta calidad de imagen. La digitalización de estos vídeos requiere servidores con una gran capacidad de almacenamiento y un gran ancho de banda para transmitir grandes volúmenes de datos. Por esta razón, los vídeos se codifican para así disminuir el ancho de banda requerido. Los formatos de codificación más utilizados para compresión de vídeos son los pertenecientes a la familia MPEG (*Moving Picture Experts Group* o Grupo de Expertos de Imágenes en Movimiento).

La familia de formatos de codificación MPEG comprende desde el estándar MPEG-1 (creado en 1990) hasta el estándar MPEG-4 (que es el sucesor del MPEG-2). El estándar MPEG-1 es un estándar genérico adecuado para todo tipo de vídeo, y tuvo una amplia repercusión. La idea inicial del estándar MPEG-1 era la de permitir el almacenamiento y reproducción en soporte CD-ROM, con un flujo de transmisión de datos del orden de 1.5 Mbits/s, transportando tanto imagen como sonido.

Debido a que la compresión de vídeo en el estándar MPEG-1 era de baja calidad se creó el estándar MPEG-2, que permite un flujo de transmisión de hasta 20 Mbit/s, transportando tanto imagen como sonido. El MPEG-2 es la base de los sistemas de televisión digital y del DVD.

El sucesor del formato MPEG-2 es el formato MPEG-4, que es mucho más versátil que su predecesor pero, a la vez, más complejo de implementar. Se prevé que el desarrollo de las diversas variantes del estándar MPEG-4 permitirá reducir aun más la velocidad de codificación de las señales de vídeo, si bien las opciones más avanzadas de este estándar aun no se han desarrollado comercialmente.

Los formatos anteriores, junto con sus requisitos de ancho de banda [21], aparecen en la Tabla 1.1.

Formato video	Ancho de banda
MPEG-1	1.5 Mb/s
MPEG-2 HDTV	19.4 Mb/s
MPEG-2 Dolby Digital	6-8 Mb/s
MPEG-4	< 1 Mb/s

Tabla 1.1 Formatos de compresión de vídeo y sus requisitos de ancho de banda.

1.1.4. Requisitos para realizar servicios de VoD

Las sesiones de VoD se caracterizan, por un lado, por la elevada cantidad de datos que se tienen que enviar durante un largo periodo de tiempo (la duración del vídeo), y por otro, por la naturaleza crítica que tienen las transmisiones (el envío y recepción de datos debe ser en tiempo real).

Por lo tanto, un sistema de VoD debe ser diseñado teniendo en cuenta una serie de requisitos que recogen la propia naturaleza de las sesiones de VoD.

A continuación, se describen brevemente algunos de estos requisitos.

- **Gran capacidad de almacenamiento**

  Para dar servicio a una gran población de usuarios y ofrecer una amplia variedad de programas, un sistema de VoD debe almacenar y administrar grandes volúmenes de datos. En consecuencia, el sistema de almacenamiento puede necesitar fácilmente espacio para decenas de Terabytes.

  Por ejemplo, un vídeo de dos horas de duración, usando una compresión MPEG-2 HTDV (televisión de alta definición), requiere aproximadamente 18 Gigabytes. Por lo tanto, un sistema de VoD que oferte 100 vídeos, puede requerir aproximadamente unos 1.8 Terabytes de almacenamiento.

- **Servicio en tiempo real**

  Un servicio de vídeo bajo demanda requiere que a cada cliente se le envíe un *stream*. Un *stream* es un flujo de datos que se envía a través de un canal al cliente que, en el caso del vídeo, debe cumplir una condición: que la transmisión de los paquetes de datos sea inmediata, ordenada y sin cortes.

  Este modo de funcionamiento tiene una consecuencia muy importante en las exigencias de red, especialmente en las relativas al retardo. Existe un tipo de retardo que afecta mucho a la calidad de los servicios que se realizan mediante streams. Este retardo se conoce como *jitter*, y se trata del retardo de unos paquetes respecto de otros y que hace que en un mismo flujo de datos la distancia entre paquetes sea variable. En consecuencia, puede que la recepción de los datos no llegue de forma ordenada. Para evitar este tipo de fallos, el servidor envía datos de forma adelantada a los usuarios de forma que se tenga un margen de tiempo para corregir los posibles retardos introducidos por la red. Los STB suelen disponer de un *buffer* donde almacenan los datos que llegan de forma adelantada.

- **Gran ancho de banda**

  Un sistema de VoD tiene que suministrar o transferir una gran cantidad de información a un número potencialmente grande de usuarios. Esto requiere que el sistema sea diseñado con una red capaz de soportar el tráfico generado por los usuarios.

- **Calidad de servicio (QoS)**

  Un aspecto a tener en cuenta en cualquier tipo de sistema orientado al servicio es el de proporcionar una calidad de servicio (*Quality of Service*, QoS) aceptable al usuario. En el caso de los sistemas de VoD, se puede requerir calidad de servicio desde el punto de vista de la red y del sistema.

  Desde el punto de vista de la red, los parámetros que determinan la calidad de servicio son:

  - la garantía de que no haya cortes en el envío de datos,
  - que no se produzca *jitter* en la presentación del vídeo,
  - e incluso la calidad de la imagen.

  Otros parámetros de QoS, desde el punto de vista del sistema son:

  - *Tiempo de espera del usuario*

    El tiempo desde que se solicita un vídeo hasta que empieza su visualización debería ser unos pocos minutos para que el cliente esté satisfecho.
  - *Abandono del sistema*

    Si el tiempo de espera del usuario es excesivo, los clientes podrían abandonar el sistema sin recibir el servicio esperado. Esto crea un malestar que hay que evitar o minimizar.
  - *Probabilidad de bloqueo*

    Si el sistema está saturado, las peticiones de servicio pueden ser rechazadas por el propio sistema. Esto también puede crear malestar al usuario si se repite con mucha frecuencia.

1.1.5. Protocolos de transmisión de datos en tiempo real

Como ya hemos mencionado antes, el servicio de vídeo bajo demanda debe entregar paquetes en tiempo real y de forma ordenada. Para que esto se verifique, se utilizan una serie de protocolos de control del flujo de datos. A continuación presentamos una breve descripción de los protocolos más utilizados para este fin.

- *RTP (Real Time Protocol)*

  RTP da la función básica de transferir paquetes de datos en tiempo real, pero no incluye mecanismos para asegurar ni la entrega de datos ni el orden en las transmisiones. Este protocolo, sin embargo, provee, entre otras funciones, la identificación de la fuente de transmisión y enumera las secuencias de entrega de los paquetes y permite así detectar las posibles pérdidas.

- *RTCP (Real Time Control Protocol)*

Este protocolo es una extensión del anterior. RTCP monitoriza el transporte de los paquetes RTP. Fue diseñado para asegurar la entrega de los paquetes enviados y para controlar la calidad del servicio (que la entrega de paquetes sea ordenada).

- *MPEG-2 Transport Stream*

  Es el estándar por defecto de transporte de vídeo. Permite agrupar diferentes programas en un único flujo. Sin embargo no implementa funcionalidades de control del flujo.

- *RTSP (Real Time Streaming Protocol)*

  Permite controlar la transmisión de datos en tiempo real y se usa especialmente en aplicaciones multimedia.

  En una sesión, el protocolo RTSP sólo se ocuparía del control, mientras que el transporte de los flujos de datos multimedia se realizaría mediante otro protocolo (por ejemplo, RTP o RTCP).

1.1.6. Infraestructura tecnológica

La transmisión de servicios multimedia requiere una infraestructura que soporte una elevada cantidad de transmisiones de datos sensibles al tiempo, así como un elevado grado de conexión entre los usuarios y los servidores.

Se han propuesto muchos protocolos de comunicación así como muchas arquitecturas de redes para conectar los diversos componentes de un sistema de VoD [22], [23].

La tecnología más apropiada para la transmisión de datos multimedia es la transmisión ATM (*Asynchronous Transfer Mode*) [22], [24]. Esta tecnología está diseñada para enviar distintos flujos de datos de forma simultánea y soporta altas velocidades de transmisión, lo que implica retardos de procesamiento muy pequeños. Asimismo, soporta la reserva de recursos y la conexión virtual de paquetes a través de las redes, lo que garantiza el servicio y la conexión. Otra característica de la transmisión ATM, es que soporta grandes anchos de banda (del orden de Gigabytes), lo que la hace bastante apropiada para un sistema de VoD.

La transmisión ATM no es la única que permite la transmisión de datos multimedia. Cabe destacar, entre otras, la transmisión ADSL. Esta tecnología ha sido desarrollada por las compañías de teléfono para transmitir los datos multimedia sobre las líneas telefónicas convencionales (de cobre). De hecho, en algunos países se han realizado experiencias con plataformas de vídeo sobre ADSL que ofrecen, entre otros servicios multimedia, el servicio de vídeo bajo demanda. Por ejemplo, en el Reino Unido cabe destacar la realizada por la compañía *Kingston Interactive TV*. En Italia, el producto de referencia para muchos operadores de banda ancha es *Fastweb*. En España, es la empresa Telefónica la que, a través de su producto *Imagenio*, [25], ofrece televisión, vídeo bajo demanda y acceso a Internet.

Otras experiencias a destacar son: en Alemania *T-Online Vision*, en Francia *MaLigne TV*, los operadores asiáticos *Yahoo! Broadband* y Korea Telecom, etc. Todas estas plataformas ofrecen acceso a una amplia oferta de canales de TV y radio, vídeo bajo demanda y acceso a Internet desde el PC y la TV.

Como se ha mencionado anteriormente, el acceso a estos servicios se basa en una infraestructura de acceso ADSL. Desde el punto de vista de la infraestructura, los clientes acceden a todos los servicios utilizando la línea convencional analógica de toda la vida y por el que actualmente ya reciben los servicios de voz.

Entre las principales desventajas de este tipo de infraestructuras, podemos destacar que la calidad de este servicio está determinada por la distancia desde el hogar hasta la central telefónica, y por el estado del par de cobre que se utiliza para la transmisión. Si el domicilio del cliente está a una distancia muy grande de la central telefónica, es imposible poder tener un servicio de calidad, ya que la señal se suele cortar constantemente.

Otra desventaja es que la tecnología ADSL tiene una topología punto a punto. Este sistema sólo permite el envío de un canal hasta el usuario y el ancho de banda que puede ofrecerse es menor al que se emplea en otros sistemas de transmisión de contenidos multimedia. Además, es una tecnología no escalable. Cuando crece la demanda en una central telefónica, el ancho de banda de la red puede no ser suficiente. En estos casos se tendría que redimensionar la red cada vez que aumentase el número de clientes, o incluso crear más centrales, para poder atender la demanda.

1.1.7. Políticas de servicios

Las políticas de servicios son las encargadas de decidir cómo se gestionan tanto las peticiones que generan los usuarios, como los recursos del sistema.

Dependiendo del tipo de comunicación empleado, las políticas de planificación de los servicios se pueden clasificar en tres tipos: broadcast, unicast y multicast.

- **Broadcast**

  La estrategia de esta política es servir a un número ilimitado de usuarios utilizando un único canal. El contenido se emite en modo difusión (para que sea recibido por cualquier usuario), con una planificación predefinida. Si un usuario está interesado en el contenido, sólo tiene que sintonizar el canal por el que se emite para visualizarlo. Esto implica que el stream que se está enviando consume el mismo ancho de banda de la red tanto si el contenido es muy demandado por los usuarios como si no lo es.

  Esta política tiene sentido utilizarla cuando los contenidos que se emiten son muy populares. Se puede servir a un gran número de usuarios usando una cantidad constante de ancho de banda. Sin embargo, si la popularidad del vídeo es media o baja, el ancho de banda se malgasta. Además,

como hemos señalado anteriormente, esta estrategia no soporta ninguna de las funciones VCR. Otro inconveniente de esta política es que los clientes que soliciten un contenido que ya ha comenzado tienen que esperar al siguiente instante de planificación del contenido para poder visualizarlo.

- **Unicast**

  Esta política es la más sencilla de implementar, ya que a cada petición de un usuario se le dedica un canal para suministrar el contenido solicitado.

  Al disponer cada cliente de un canal dedicado, puede realizar cualquiera de las funciones VCR. Es decir, con la política unicast se puede soportar vídeo bajo demanda verdadero (T-VoD). Sin embargo, con esta estrategia no se puede optimizar el uso del ancho de banda disponible y para poder dar servicio a una población numerosa se necesitan un buen dimensionamiento del sistema y grandes anchos de banda.

- **Multicast**

  Con esta política un grupo de usuarios interesados en el mismo contenido se sirve con un mismo stream. Esta política es mucho más eficiente que las anteriores, en el sentido de que optimiza el uso del ancho de banda del servidor; es evidente que con la estrategia multicast se atienden a más clientes con un mismo recurso que si se hubiera empleado la técnica unicast. Y, en comparación con la estrategia broadcast, la política multicast asegura que al menos un cliente esté haciendo uso del canal empleado.

  Entre las distintas estrategias multicast, *batching* y *patching* son las más usadas. La idea básica de la estrategia *batching* [26], [27], [28] es ir agrupando durante un cierto intervalo de tiempo las peticiones a un mismo contenido. Una vez transcurrido el intervalo, se sirven las peticiones utilizando el mismo stream. La principal ventaja de esta solución es la simplicidad a la hora de implementarla, pero su principal desventaja es que el retraso del comienzo del servicio puede ser muy grande aun en el caso de que haya disponible suficiente ancho de banda. Esto puede incrementar el grado de insatisfacción del usuario, que incluso puede desestimar la petición.

  En una estrategia *patching* [29], se optimizan los recursos de la siguiente forma. Con la llegada de una petición al servidor, se comprueba si existe un stream que esté transmitiendo el contenido solicitado desde su inicio (stream completo). En caso afirmativo, lo que se hace es añadir el usuario que cursó la petición al canal que está transmitiendo el contenido, e iniciar un nuevo stream en un nuevo canal con el trozo inicial de vídeo que le falta (stream parcial). De esta forma, el usuario es servido de forma inmediata y el ancho de banda que se ocupa con los primeros instantes del vídeo está ocupado menos tiempo que si se le hubiera asignado el canal para la emisión completa del vídeo. Para implementar esta estrategia, se necesita que el usuario disponga de un STB que soporte la recepción simultánea de, al menos, dos canales. En el STB se van almacenando los

datos que transmite el stream completo, mientras el usuario visualiza los datos recibidos del stream parcial.

Algunos autores han sugerido distintas combinaciones de políticas para conseguir un balanceo entre coste y rendimiento. En [30] se combinan las políticas de servicio unicast y broadcast, [31] propone una combinación de servicios unicast, multicast y broadcast, mientras que [32] combinan multicast, unicast y batching. Todas estas políticas se analizarán detenidamente en el Capítulo 3.

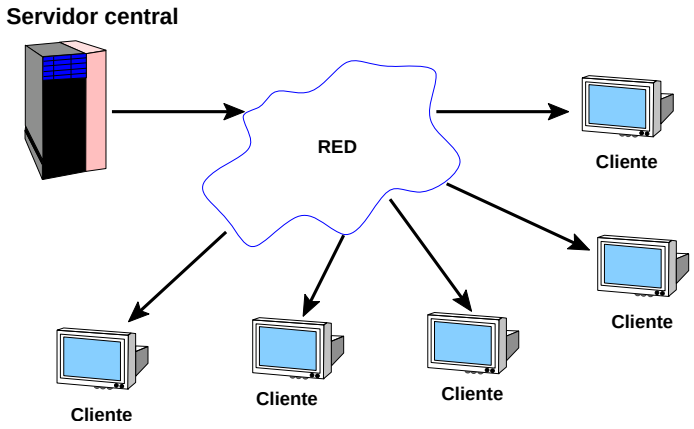


Figura 1.2 Sistema centralizado de VoD.

1.1.8. Arquitecturas propuestas

Un sistema de VoD puede ser diseñado como un sistema centralizado o como un sistema distribuido [23].

En un sistema centralizado existe un único coordinador (el servidor central), que es el que gestiona todas las actividades del sistema y almacenada todos los contenidos. Los usuarios del sistema están conectados al servidor central mediante una red principal que comparten. La Figura 1.2 ilustra la arquitectura de un sistema centralizado.

Los sistemas centralizados son bastante simples de administrar pero, normalmente, son poco escalables y poco robustos, ya que existe un único punto de fallo (el servidor central). Además, debido a que todas las peticiones del sistema tienen que ser transmitidas por el servidor central, la red suele convertirse en el cuello de botella.

Una forma de mejorar el rendimiento de un sistema centralizado es añadir múltiples nodos de servicio (servidores locales) a la red (ver Figura 1.3). Los servidores locales pueden almacenar los vídeos más populares, de forma que su transmisión llegue más rápidamente a los clientes. Si se solicita un vídeo que no está en un servidor local, éste solicita el vídeo al servidor central, que lo transmite al usuario a través de la red. Sin embargo, uno de los inconve-

nientes de este tipo de sistemas es que encarece el sistema al introducir nuevos servidores.

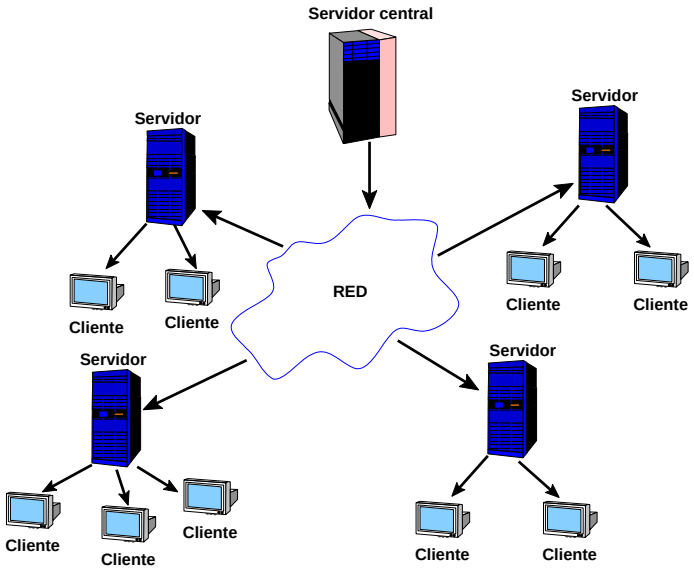


Figura 1.3 Sistema centralizado con servidores de VoD.

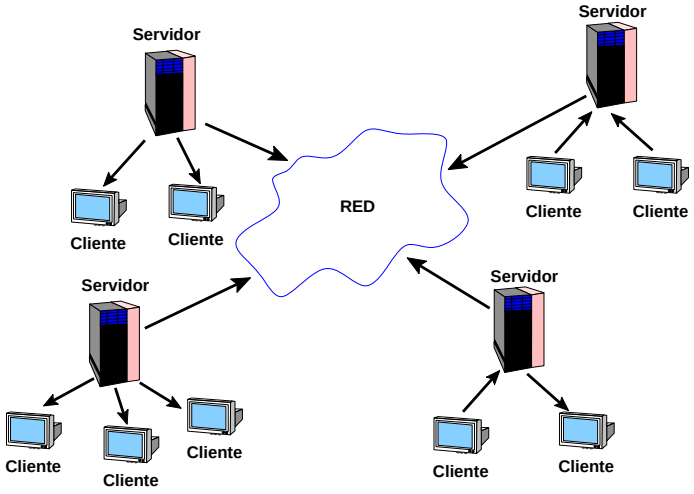


Figura 1.4 Sistema distribuido de VoD.

Otra opción es una arquitectura distribuida de servidores conectados entre sí (como muestra la Figura 1.4), cada uno con su propia población de usuarios. Cada servidor realiza la gestión de sus recursos y planifica las peticiones generadas por sus clientes. En otras palabras, cada servidor actúa de forma independiente (no existe la figura del servidor central). Pero para conseguir esta autonomía, todos los usuarios de cualquiera de los servidores tienen que tener acceso a todos los contenidos. Una posibilidad es almacenar todos los contenidos en cada uno de los servidores, lo que equivaldría a múltiples servi-

dores centralizados independientes y no haría falta una red que los conectara entre sí. Sin embargo, existen propuestas en que los servidores están conectados para permitir que en caso de fallo o si, de forma eventual, un servidor está saturado, pueda redirigir las peticiones a otro servidor del sistema [33]. El servicio, en este caso, se realizaría a través de la red principal.

Una de las ventajas de este tipo de arquitectura es que tienen una mejor escalabilidad que las arquitecturas centralizadas. Además, los sistemas distribuidos son más tolerante a los fallos; si un servidor tiene problemas, sus clientes pueden recibir servicio de otros servidores.

Uno de los inconvenientes de estas arquitecturas es el elevado coste asociado con el sistema de almacenamiento, debido a que se debe almacenar en cada servidor del sistema todos los videos ofertados.

Una forma de reducir los tiempos de espera en estos sistemas es instalar entre un servidor y sus clientes un *servidor-proxy* (ver Figura 1.5). Estos servidores-proxy se comportan como una caché del servidor, pudiendo almacenar los contenidos más populares [34], [35], [36], [37], [38]. Una estrategia muy utilizada para optimizar los recursos del proxy es la conocida como *caching*. Esta técnica consiste en almacenar el fragmento inicial de cada vídeo popular en el proxy (*prefix-caching* [36]). Esta política tiene la ventaja de que los clientes que solicitan vídeos cuyo prefijo almacena el proxy no sufren espera para obtener el servicio, ni sufren problemas de *jitter*.

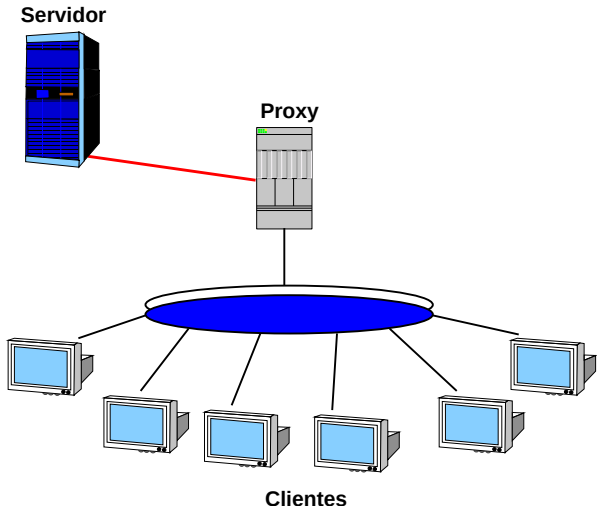


Figura 1.5 Sistema con proxy.

La estrategia *caching* suele combinarse con distintas políticas de servicio: en [39], se combina prefix-caching con broadcast periódico, en [40], [41] se combinan prefix-caching y las estrategias de servicios *batching* y *patching*.

En resumen, la elección de la arquitectura del sistema dependerá, entre otros factores, del almacenamiento necesario, los sistemas de comunicación que se deban emplear, de los costes asociados a la implementación física del sistema

y de la escalabilidad que se exija.

En general, es preferible una configuración distribuida de servidores a una configuración centralizada debido a dos factores [42]:

1. buena escalabilidad (el sistema puede escalarse más fácilmente), y
2. gran fiabilidad (el sistema puede dar servicio aun cuando alguno de los servidores del sistema falle).

Debido a estas razones, en esta tesis: i) nos centraremos en el estudio de un sistema distribuido de VoD; ii) estudiaremos la viabilidad de un sistema basado en servidores con baja capacidad de almacenamiento, conectados mediante una red donde; iii) los vídeos que ofrece el sistema están distribuidos entre los servidores, y iv) analizaremos distintas estrategias de distribución "equitativas" de los vídeos.

1.2. Descripción del sistema VoD propuesto

Debido a los problemas de costes asociados a la creación de un sistema de VoD a gran escala, una solución económicamente viable consiste en diseñar un sistema distribuido con varios servidores pequeños de VoD, conectados entre sí a través de una red de capacidad limitada. Nuestro objetivo es investigar el rendimiento de un sistema distribuido genérico de VoD con las siguientes características:

- Los servidores que tengan ancho de banda disponible ayudan a otros servidores que estén temporalmente saturados (*load sharing*).
- No *todos* los vídeos son almacenados en todos los servidores del sistema.

Uno de los objetivos de esta tesis es conseguir un modelo analítico del sistema que nos permita asegurar que se ofrece un nivel adecuado en los servicios en términos de tiempo de respuesta del sistema y determinar los requisitos de ancho de banda, capacidad de almacenamiento y otros parámetros del sistema para garantizar un tiempo de respuesta preestablecido. Con el modelo podemos considerar y evaluar distintas alternativas de los parámetros del sistema sin necesidad de esperar a implementarlo para estudiar su rendimiento. Asimismo, podemos predecir el rendimiento que tendrá en un futuro un sistema ya implementado, y si hace falta, mejorar oportunamente sus prestaciones.

En la Figura 1.6 se muestra la arquitectura genérica del sistema distribuido de VoD que vamos a estudiar. A continuación se describen los parámetros del sistema a los que se asignarán los correspondientes valores en las simulaciones que se presentan en los siguientes capítulos (ver tabla 2.1). El sistema que se analiza en esta memoria consiste en N_{serv} servidores conectados a un

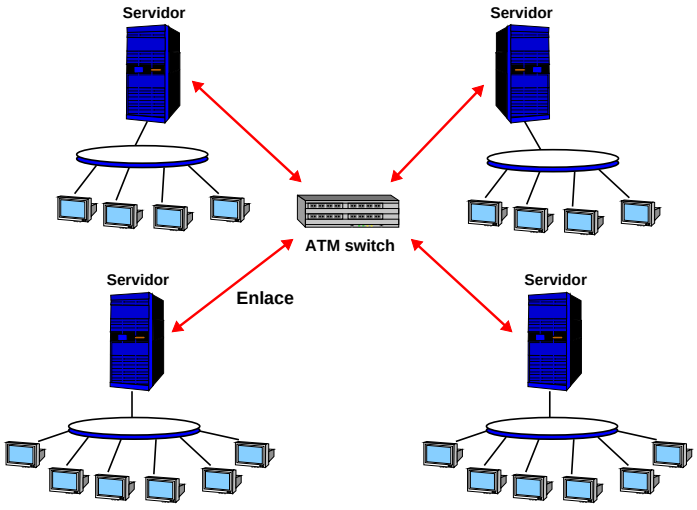


Figura 1.6 Sistema distribuido de VoD.

conmutador ATM por medio de un enlace de dos sentidos, uno de entrada al servidor y otro de salida. El ancho de banda del conmutador es B_{switch} medido en Megabits por segundo (Mbps), y cada uno de los enlaces tiene un ancho de banda B_{link} Mbps en cada sentido. Cada servidor tiene conectado una población de N_{term} clientes, y tiene capacidad para servir N_{stream} canales a la vez. Por estos canales se transmitirán, a través de un stream, los datos que se necesitan para que los clientes visualicen el vídeo solicitado. El número de vídeos que ofrece el proveedor de programas a los clientes es M.

El ancho de banda requerido para servir un stream de vídeo por canal es de B_{stream} Mbps. Un servicio depende de la calidad que se tenga contratada. En esta tesis suponemos B_{stream} fijo para todos los vídeos del sistema.

En nuestra investigación suponemos que todos los servidores del sistema son homogéneos, es decir, suponemos que tienen el mismo número de clientes, la misma capacidad de servicio, etc. Hemos supuesto esta homogeneidad para facilitar el desarrollo de los modelos analíticos del sistema que presentaremos en los siguientes capítulos. Aunque, como veremos, las expresiones que obtenemos pueden ser utilizadas aun cuando los servidores no sean homogéneos.

La primera característica del sistema que estudiamos en esta memoria es la compartición de la carga entre los servidores. Se produce compartición de la carga cuando un servidor con ancho de banda disponible puede ser requerido para realizar un servicio solicitado a otro servidor que está temporalmente saturado.

No se debe confundir compartición de la carga con balanceo de la carga. Un sistema que balancea la carga intenta equilibrar la realización de servicios entre todos sus servidores. Por tanto, el sistema puede asignar peticiones efectuadas en un servidor a otro servidor del sistema (servidor remoto). Para realizar estos servicios se usa parte del ancho de banda de la red de conexión. Esto implica

un uso innecesario de los recursos. Sin embargo, en un sistema que comparte la carga, el planificador asigna servicios a otro servidor sólo cuando un servidor está saturado. En esta situación se utiliza la red de conexión.

1.3. Distribución de los vídeos: popularidad y replicación parcial

Puesto que proponemos un sistema de servidores pequeños, su capacidad de almacenamiento será limitada y no podemos suponer que puede almacenar todos los vídeos en cada servidor, sino que habrá que distribuir los vídeos entre los distintos servidores del sistema. Para garantizar al usuario un tiempo de espera pequeño parece lógico almacenar una copia de los vídeos más populares en todos los servidores. El resto de los vídeos que se ofertan, se distribuyen siguiendo algún esquema de distribución.

En esta sección describimos cómo se distribuyen los vídeos entre los servidores y definimos dos conceptos importantes en nuestro trabajo: la *popularidad de los vídeos* y el *porcentaje de replicación*.

Como vamos a estudiar el comportamiento de un sistema distribuido pequeño donde se permite compartir la carga, nos centramos en redes locales o de un área metropolitana, que es donde más beneficio podemos obtener de la funcionalidad de la compartición de la carga [33].

En este contexto, parece razonable suponer que todos los usuarios de los servidores tienen preferencias similares, y por esta razón vamos a trabajar suponiendo la popularidad de todos los vídeos de forma global. En cualquier caso, el esquema de compartición de la carga que proponemos más adelante puede manejar distintos comportamientos de los usuarios, así como distintas popularidades de los vídeos en cada servidor.

Cualquiera de los M vídeos ofertados por el proveedor de programas puede ser solicitado por cualquier cliente del sistema. Además, la popularidad de los vídeos viene dada por el comportamiento de los usuarios.

Definición 1.1 Definimos la **popularidad** de un vídeo en el sistema como la frecuencia relativa de las peticiones a ese vídeo de todos los usuarios del sistema.

En los estudios de mercado se utilizan varios tipos de distribuciones de probabilidad, que intentan aproximar el comportamiento de la popularidad de un producto o de un servicio (como la distribución uniforme). Sin embargo, en la bibliografía encontramos que es la distribución Zipf [43] la mayormente utilizada para modelar la popularidad de los vídeos [44], [11], [28], [45], [46]. Esto es porque la distribución Zipf es una distribución caracterizada por modelar eventos muy frecuentes (por ejemplo, películas de estreno) y eventos poco frecuentes (como las películas que pertenecen al cine clásico). Por tanto, para mantener nuestra propuesta lo suficientemente general, suponemos que la popularidad de los vídeos en el sistema sigue una distribución Zipf.

1.3.1. Función de probabilidad Zipf

Para un espacio muestral de tamaño M, la distribución de probabilidad Zipf está dada por

$$p_i = \frac{c}{i^{\xi}} \tag{1.1}$$

donde $i \in \{1, \ldots, M\}$, p_i es la probabilidad de que un cliente del sistema solicite el vídeo i, $c = \frac{1}{\sum_{k=1}^{M} 1/k^{\xi}}$ es una constante de normalización y el parámetro ξ es el que influye en los valores de las probabilidades asociadas a los distintos vídeos.

Definición 1.2 Al parámetro ξ de la función de distribución Zipf, lo denominamos como **grado de popularidad**.

De la expresión 1.1 se deduce que las probabilidades asignadas por la distribución Zipf decrecen con los índices de la secuencia de vídeos, es decir, $p_1 > p_2 > \ldots > p_M$. Por otro lado, cuando el parámetro ξ crece, la probabilidad de solicitar los vídeos más populares crece.

Ejemplo 1.1 La Figura 1.7 muestra en el eje de abscisas 10 vídeos ordenados por orden decreciente de probabilidades, y en el eje Y la probabilidad de solicitar el vídeo 1, el vídeo 2, etc, para distintos grados de popularidad ($\xi = 1{,}8$, $\xi = 1$, $\xi = 0$) cuando $M = 10$. Como se observa, los primeros vídeos tienen más probabilidad conforme aumenta el grado de popularidad. Por ejemplo, entre el vídeo 1 y el vídeo 2 acumulan casi el 80 % de la probabilidad de ser solicitados cuando $\xi = 1{,}8$. Esta figura muestra, además, que el parámetro ξ hace que unos vídeos sean más frecuentemente solicitados (los vídeos muy populares) que otros (los no populares).

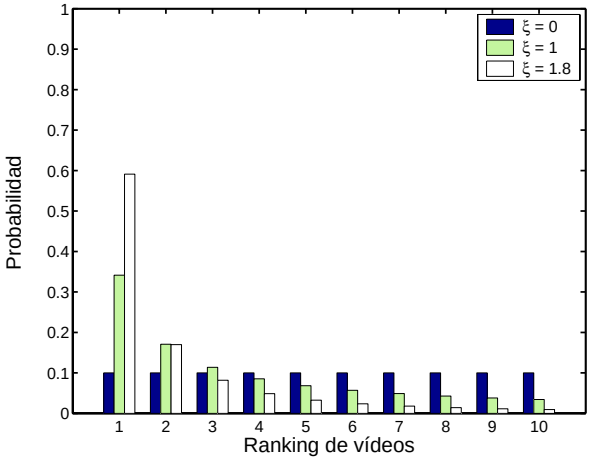


Figura 1.7 Probabilidad de solicitar el vídeo i ($i = 1, ..., 10$) para distintos valores de ξ según la distribución Zipf.

1.3.2. Distribución de los vídeos

Obviamente, las preferencias de los clientes no serán uniformes a lo largo de un período de 24 horas. Típicamente, uno podría esperar de diferentes grupos de usuarios distintas elecciones de programas. Por ejemplo, los telediarios son los más vistos en los mediodías mientras que los vídeos infantiles son más populares durante las primeras horas de la tarde. No sólo la popularidad de los vídeos fluctúa, sino que la carga (o número de peticiones por unidad de tiempo) podría ser de baja a moderada a lo largo del día y podría, sin embargo, aumentar gradualmente a lo largo de la tarde-noche para más tarde caer de nuevo durante la noche [22]. La Figura 1.8 muestra cómo puede variar la demanda a lo largo de un día laborable [47]. Es más, el consumo de contenidos audiovisuales varía a lo largo de la semana, existiendo una clara diferenciación entre los días laborables y el fin de semana. El proveedor del sistema debería caracterizar tales periodos de carga (tiempo de inicio, duración, etc). De forma diaria, para cada periodo de carga, un algoritmo de administración podría actualizar las popularidades de los vídeos. Una vez hecho esto, el administrador de recursos debería aprovechar las horas con menos actividad (para no colapsar la red) para redistribuir y copiar los vídeos entre los servidores, de forma que estén disponibles en las horas con mayor demanda [22].

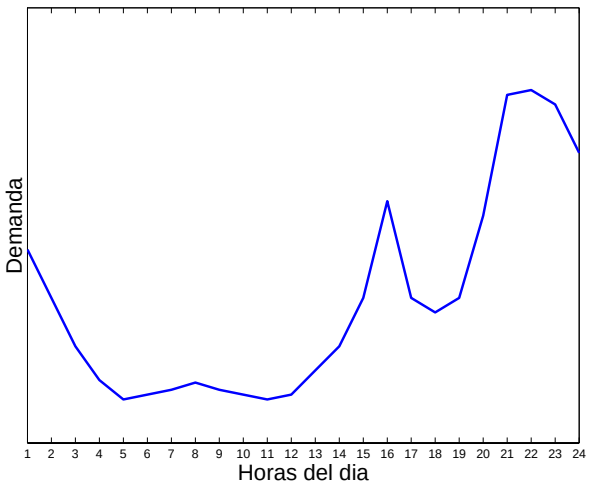


Figura 1.8 Variación de la carga en el sistema a lo largo de un día laborable.

De ahora en adelante, centraremos nuestro estudio en uno de estos periodos de carga, para el cual consideramos que son fijos tanto el número estimado de peticiones por unidad de tiempo como la popularidad de los vídeos.

En nuestro estudio, a la hora de distribuir y colocar los vídeos en los servidores, nos hemos centrado en aquellas políticas que sean "equitativas" desde el punto de vista de la distribución de los recursos y de los usuarios.

Definición 1.3 Una política es equitativa si distribuye los vídeos de forma uniforme entre los servidores, de forma que el cliente pueda esperar un rendi-

miento similar sea cual sea el servidor al que está conectado.

Con respecto a la distribución de los vídeos, suponemos que todos los servidores van a almacenar los vídeos más populares.

Definición 1.4 Definimos el **porcentaje de replicación** como el porcentaje de vídeos que se encuentran almacenados en **todos** los servidores. Llamamos **vídeo replicado** aquel vídeo que está almacenado en **todos** los servidores. Si el porcentaje de replicación es menor que el 100 %, diremos que existe **replicación parcial** que es como se denomina al hecho de copiar **sólo** los vídeos más populares en **todos** los servidores.

Veamos en más detalle esta cuestión: los M vídeos que oferta el proveedor de programas se ordenan según su popularidad, es decir, establecemos una secuencia ordenada de los vídeos. Fijado un porcentaje de replicación, establecemos el número s de vídeos que van a ser almacenados en todos los servidores, es decir, los primeros s vídeos de la secuencia ordenada de vídeos.

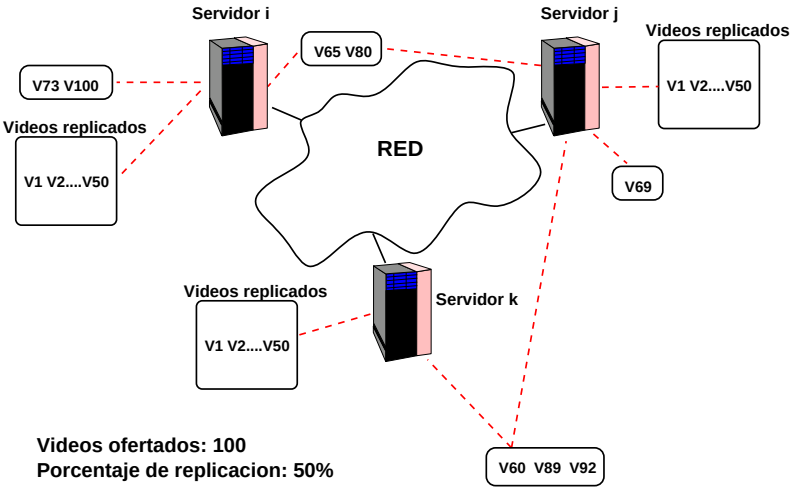


Figura 1.9 Ejemplo de distribución de los vídeos.

En cuanto a los $M - s$ vídeos restantes, podemos utilizar distintos esquemas de distribución.

Ejemplo 1.2 La Figura 1.9 muestra un ejemplo de distribución de los vídeos. Hemos supuesto que el número de vídeos en el sistema es $M = 100$ vídeos, y un porcentaje de replicación del 50 %. En este caso, los $M = 100$ vídeos se ordenan según su popularidad y los primeros $s = 50$ vídeos de la secuencia ordenada son los que van a ser almacenados en todos los servidores. A continuación se distribuyen los restantes $M - s = 50$ vídeos entre todos los servidores siguiendo algún esquema de distribución. En este ejemplo, le ha correspondido almacenar al servidor i los vídeos 73 y 100, al servidor j sólo el vídeo 69 y al servidor k los vídeos 60, 89 y 92. Los restantes vídeos están almacenados en el resto de servidores que componen el sistema.

Un esquema de distribución de los vídeos puede ser, por ejemplo, ordenar los $M-s$ vídeos por orden de popularidad y distribuirlos de forma cíclica entre los servidores, con lo cual cada uno estaría almacenado en un único servidor. En este caso hablamos de una **distribución cíclica** de los vídeos no populares.

Ejemplo 1.3 En la Figura 1.10 mostramos un ejemplo de distribución cíclica de los vídeos menos populares, suponiendo un sistema con 16 servidores, $M = 100$, y un porcentaje de replicación del 50 %. Por lo tanto, se almacenan los primeros $s = 50$ vídeos en todos los servidores y se distribuyen los $M - s = 50$ restantes vídeos de forma cíclica entre los todos los servidores (14 servidores tendrán 53 vídeos y 2 servidores, 52).

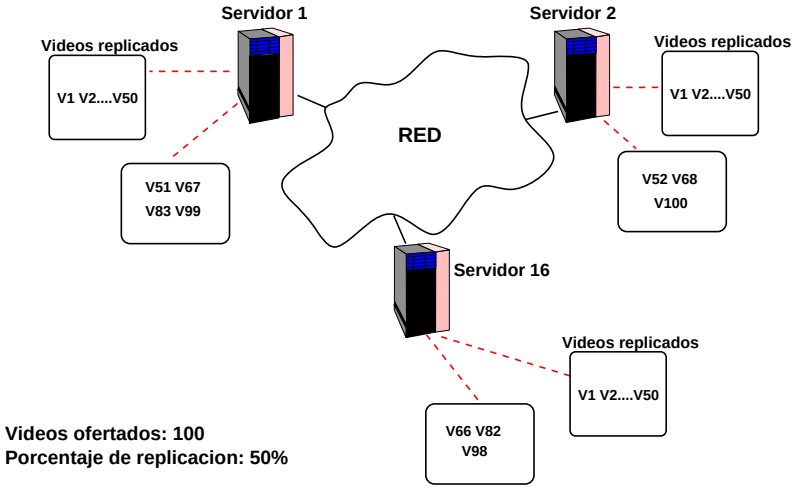


Figura 1.10 Distribución cíclica de los vídeos no populares.

Otro esquema de distribución posible sería seleccionar algunos vídeos (los que tengan más probabilidad) de entre los $M - s$ vídeos no replicados y almacenarlos en algunos grupos de servidores. En este caso hablamos de realizar una *replicación moderada (R.M.)* de estos vídeos.

Un caso de replicación moderada sería almacenar de forma uniforme entre grupos de k servidores los a vídeos que sean más populares de los $M - s$ vídeos y por último, repartir de forma uniforme entre todos los servidores los restantes vídeos (los menos populares). Por ejemplo, podríamos seleccionar el primer vídeo de la secuencia ordenada $M - s$, y distribuirlo en un grupo de servidores ($servidor_1$,$servidor_2$,...,$servidor_k$). A continuación podríamos distribuir en otro grupo de servidores ($servidor_{k+1}$,$servidor_{k+2}$,...,$servidor_{2k}$) el siguiente vídeo de la secuencia, y seguir este proceso para todos los a vídeos que hemos seleccionado. Los restantes vídeos menos populares, $M - s - a$ se distribuyen de forma uniforme entre los servidores.

Ejemplo 1.4 En la Figura 1.11 presentamos un ejemplo de replicación moderada del 20 % de los $M - s$ vídeos (20 % R.M.) y consideramos grupos de $k = 8$ servidores. Al igual que en el ejemplo anterior, suponemos un sistema

con 16 servidores, $M = 100$ vídeos, y un porcentaje de replicación del 50 %. Por tanto, los primeros 50 vídeos se almacenan en todos los servidores.

El 20 % de los restantes $M-s = 50$ vídeos, corresponde con $a = 10$. Es decir, los vídeos $V51, V52, \ldots, V60$ son distribuidos equitativamente en 2 grupos. Seleccionamos el primer vídeo de esa secuencia, $V51$ y lo almacenamos en un primer grupo de 8 servidores ($servidor_1$,$servidor_2$,...,$servidor_8$). A continuación copiamos el siguiente vídeo de la secuencia $s + a$, (que en este ejemplo es el vídeo $V52$), en otro grupo de servidores ($servidor_9$,$servidor_{10}$,...,$servidor_{16}$). Los siguientes vídeos de la secuencia $V51, V52, \ldots, V60$ se copian de manera similar entre los distintos grupos de servidores, mientras que los restantes $M - s - a = 40$ vídeos se distribuyen de forma uniforme entre todos los servidores como muestra la Figura 1.11.

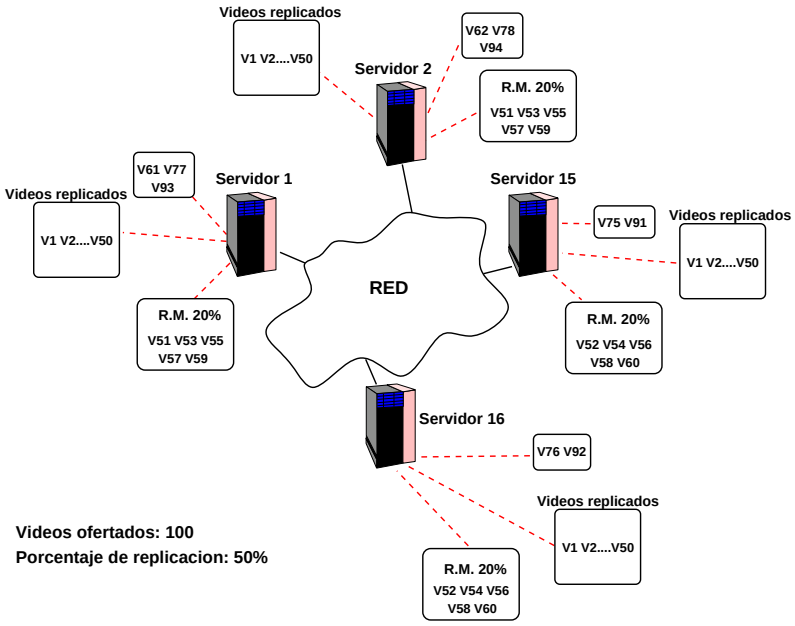


Figura 1.11 Replicación moderada (R.M.) de los $M = 100$ vídeos ofertados por un sistema de 16 servidores: R.M. del 20 %.

Existen trabajos previos que consideran la replicación parcial de los vídeos más populares en un sistema distribuido multimedia [48], [49], [50], [51]. Estas aproximaciones proponen la asignación de un vídeo a uno o más servidores de tal forma que la probabilidad de bloqueo (es decir, la probabilidad de que una petición para un servicio de VoD sea rechazada porque el servidor no tiene streams disponibles), sea lo más pequeña posible. Además, en los trabajos reseñados sólo se optimiza la relación coste/ganancia del sistema según la política de replicación seleccionada. Es decir, no se tiene en cuenta el posible impacto de estas políticas en la calidad de servicio que se ofrece a los clientes (por ejemplo, en el tiempo de espera de los usuarios).

1.3.3. Caracterización de una distribución de vídeo equitativa

Cualquiera que sea la estrategia elegida para distribuir los vídeos en el sistema, está claro que siempre es mejor tener almacenados los vídeos más populares en todos los servidores. La estrategia a seguir para distribuir los restantes vídeos dependerá del criterio que más le convenga al distribuidor de programas y de la capacidad de almacenamiento disponible de los servidores que compongan el sistema. En todo caso, el acceso a cualquiera de los vídeos (aunque no estén en el servidor en el que se realiza la petición) será transparente para los clientes, siendo el planificador de los servicios el encargado dar el servicio.

Definición 1.5 Decimos que un vídeo es **local** a un servidor si se encuentra almacenado en él.

Sea s el número de vídeos replicados en todos los servidores y sea $\mathcal{K}_i$ el conjunto de los *vídeos no replicados* asignados al servidor i ($\forall i \in \{1, \ldots, N_{server}\}$). La **probabilidad F_i de solicitar un vídeo local** en el servidor i, se calcula como

$$F_i = \sum_{j=1}^{s} p_j + \sum_{\forall k \in \mathcal{K}_i} p_k \tag{1.2}$$

El primer término de la ecuación 1.2 representa la probabilidad de solicitar un vídeo replicado. De la ecuación 1.2 se deduce que:

(a) Si el porcentaje de replicación se incrementa entonces s aumenta y de la ecuación 1.2 se deduce que F_i aumenta.

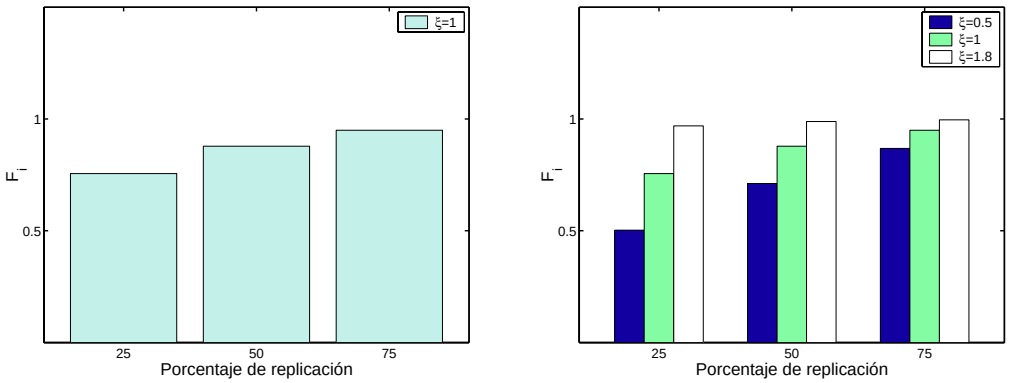
(b) Si el grado de popularidad ξ aumenta, entonces la popularidad de los vídeos más populares aumenta y como consecuencia, también lo hace F_i.

Estas observaciones las usaremos más adelante en el siguiente capítulo.

Ejemplo 1.5 En la Figura 1.12(a) mostramos un ejemplo donde se puede ver cómo aumenta F_i cuando se incrementa el porcentaje de replicación en un sistema compuesto por 16 servidores y que oferta $M = 100$ vídeos. En este caso hemos fijado el grado de popularidad a $\xi = 1$. En la Figura 1.12(b) hemos variado el grado de popularidad: $\xi = 0{,}5$, $\xi = 1$ y $\xi = 1{,}8$; podemos ver cómo, efectivamente, aumenta F_i conforme aumenta el grado de popularidad y para los distintos porcentajes de replicación.

Definición 1.6 Se define la **probabilidad media F de solicitar un vídeo local** en cualquier servidor, como

$$F = \frac{\sum_{i=1}^{N_{server}} F_i}{N_{server}} \tag{1.3}$$



(a) (b)

Figura 1.12 Incremento de F_i en un sistema de 16 servidores que oferta $M = 100$ vídeos: (a) cuando se incrementa el porcentaje de replicación (con $\xi = 1$ fijo); (b) cuando aumenta el grado de popularidad ($\xi = 0{,}5$, $\xi = 1$ y $\xi = 1{,}8$) para los distintos porcentajes de replicación.

Obviamente, la probabilidad media de solicitar un vídeo no local es $1 - F$.

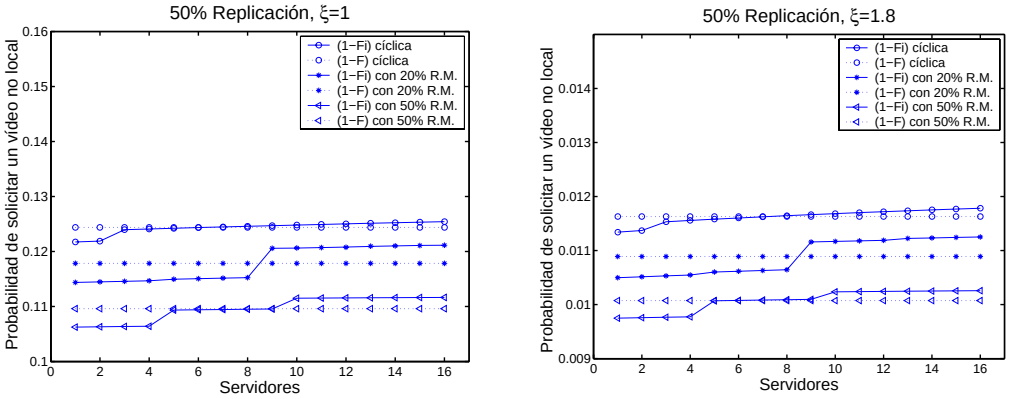
Ejemplo 1.6 Si todos los vídeos están almacenados en todos los servidores (en cuyo caso tenemos que el porcentaje de replicación es el 100 %) se deduce que $F = 1$ y $1 - F = 0$.

Como mencionamos antes, nuestra investigación estudia esquemas de distribución de los vídeos que sean "equitativas". Desde el punto de vista del cliente, nos interesa dar un rendimiento similar independientemente del servidor al cual está conectado el cliente. En otras palabras, nosotros consideramos que una política de distribución de vídeos es equitativa cuando la probabilidad de solicitar un vídeo local (o no local) en cualquier servidor es aproximadamente igual a la probabilidad de solicitar un vídeo local (o no local) en el sistema. Es decir, F_i ($\forall i \in \{1, ..., N_{server}\}$) puede ser aproximado por F, o equivalentemente, $1 - F_i$ ($\forall i \in \{1, ..., N_{server}\}$) puede ser aproximado por $1 - F$.

Los esquemas de distribución de vídeos que hemos descrito anteriormente verifican esta importante propiedad. Para ilustrar esta cuestión hemos seleccionado las distribuciones de los vídeos realizadas en los ejemplos 1.3 y 1.4 y hemos medido los valores de $1 - F_i$ (ecuación 1.2) y $1 - F$ (ecuación 1.3) en cada una de ellas y para cada uno de los diferentes servidores. La Figura 1.13 muestra esas probabilidades desde el servidor 1 al servidor 16, suponiendo un sistema con 16 servidores, $M = 100$, y un porcentaje de replicación del 50 %.

Como estamos interesados en el impacto que tiene el grado de popularidad en nuestra aproximación, hemos medido las probabilidades de solicitar vídeos no locales en dos casos: cuando $\xi = 1$ y cuando $\xi = 1{,}8$. Como podemos ver en la Figura 1.13, las diferencias para cada esquema de distribución entre $1 - F_i$ y $1 - F$ son pequeñas, especialmente cuando el grado de popularidad es alto ($\xi = 1{,}8$). Por ejemplo, la máxima diferencia entre $1 - F_i$ y $1 - F$ para cualquier esquema de distribución es siempre más pequeña que 10^{-3} para $\xi = 1$, mientras

que para $\xi = 1{,}8$ es más pequeña que 10^{-4}. Creemos que esas diferencias son lo suficientemente pequeñas para justificar la aproximación de $1-F_i$ por $1-F$ cuando se utiliza un esquema de distribución de los vídeos equitativa, como los que proponemos en esta memoria.



(a) (b)

Figura 1.13 Probabilidad de solicitar un vídeo no local según distintas estrategias de distribución y distintos valores de ξ: (a) $\xi = 1$; (b) $\xi = 1{,}8$.

De la Figura 1.13 podemos deducir que para cualquier grado de popularidad ξ, la distribución cíclica es la única en la que las diferencias entre las probabilidades de solicitar un vídeo no local son significativas. Por tanto en ese caso, el número de peticiones para vídeos no locales (las cuales consumirán recursos extras de la red) será mayor que con otro esquema de distribución, y como veremos en el siguiente capítulo, esto significa que la disponibilidad de recursos para la compartición de la carga se verá bastante reducida. Por lo tanto, escogemos esta estrategia de distribución de los vídeos para así poder estudiar lo que nosotros consideramos el escenario con las condiciones más adversas.

1.4. Objetivos de esta tesis

Una vez descrito el sistema distribuido de VoD que queremos analizar, resumimos los objetivos de esta tesis en los siguientes puntos:

1. Diseñar algoritmos de planificación de los recursos (de tipo unicast y multicast) que optimicen el tiempo de espera y/o el ancho de banda en el contexto de nuestra arquitectura distribuida de VoD.

2. Implementar un modelo analítico de nuestros algoritmos. Este modelo se usará como una herramienta que facilite el diseño y control de los parámetros del sistema.

3. Analizar de forma experimental la validez de los modelos analíticos y determinar el rendimiento de los algoritmos en función del porcentaje de replicación y del grado de popularidad.

4. En particular, se quiere estudiar cómo influyen los distintos parámetros del sistema en el tiempo medio de espera de los clientes en situaciones críticas, las cuales ocurren cuando no hay vídeos mucho más populares que otros y el porcentaje de replicación es bajo.

5. Diseñar y analizar un algoritmo que tenga en cuenta la impaciencia de los usuarios en el sistema y, obtener cotas del ancho de banda requerido para realizar los servicio mientras se mantiene un valor prefijado de probabilidad de abandono.

A lo largo del resto de esta memoria desarrollaremos los puntos aquí expuestos.

2 El algoritmo PRLS

2.1. Introducción

Un sistema de VoD permite a los usuarios recibir y reproducir contenidos solicitados explícitamente e interactuar con ellos. Los clientes solicitan los contenidos cuando ellos quieren (de forma aleatoria), y no a horas preprogramadas, y el planificador de servicios los atiende conforme recibe las solicitudes y tiene recursos para realizar los servicios.

En la actualidad se están implementando sistemas interactivos donde el cliente tiene total libertad para interactuar con el contenido que ha solicitado. Puede detener temporalmente la visualización del vídeo o avanzar o retroceder a su antojo. Pero para que esto ocurra es obligatorio que los flujos de vídeo (streams) sean exclusivos e independientes [25]. En otras palabras, se necesita que el sistema implemente una estrategia de servicio de tipo *unicast*.

Como ya se ha mencionado en el capítulo anterior, la estrategia unicast es la más fácil de implementar. Se trata de asignar un canal exclusivo a cada usuario que solicite un vídeo. Evidentemente un sistema con, por ejemplo, 1000 clientes, tiene que disponer de suficientes canales de servicio para poder afrontar la demanda que éstos generen. Una forma muy simple de dar servicio a esta población sería construir un sistema que tuviera 1000 canales de servicio. Teniendo en cuenta que la red de comunicación es uno de los componentes más caros del sistema, esta solución podría resultar prohibitiva. Pero como no siempre van a estar conectados todos los usuarios a la vez, se puede diseñar un sistema con menos canales. Ahora bien, se requiere un mecanismo que controle cuántos canales son necesarios para que el sistema no sea muy costoso pero de forma que el tiempo de espera de los clientes no sea muy elevado.

Una herramienta poderosa para realizar el análisis de un sistema en el que ocurren procesos aleatorios, (como es el caso de un sistema de VoD) es la teoría de colas. La teoría de colas es una formulación matemática para la optimización de sistemas en los que interactúan dos procesos normalmente aleatorios: el proceso de llegada de clientes y el proceso de servicio a los clientes. Con esta

herramienta el diseñador del sistema puede analizar el comportamiento del sistema y asegurar que el tiempo de espera de los clientes esté por debajo de un nivel establecido.

En las secciones 1.2 y 1.3 del Capítulo 1 se describe el sistema distribuido que proponemos, el cual contempla la compartición de la carga (*load sharing*) y la replicación parcial de los vídeos. En este capítulo proponemos un modelo analítico que permite determinar el tiempo de espera en función de los parámetros del sistema.

Nuestra investigación se centra en la minimización del tiempo de espera explotando la compartición de la carga. Suponemos además que los clientes están dispuestos a esperar a recibir el servicio, es decir, que no abandonan el sistema. Es lo que se define como un sistema sin pérdidas.

La funcionalidad de la compartición de la carga se comporta como sigue: si el vídeo solicitado por un cliente de un servidor está almacenado en él, y hay suficiente ancho de banda en el servidor para atender la petición, entonces se inicia lo que nosotros denotamos como *servicio local*. Por el contrario, cuando un servidor está ocupado, es decir, no tiene suficientes recursos para atender una petición, lo que hace es ponerla en cola. En este caso, el servidor inicia un diálogo con los otros servidores que almacenan el vídeo solicitado para determinar si hay un servidor en el sistema con recursos disponibles que se pueda hacer cargo de la petición. Si alguno de esos servidores tiene ancho de banda disponible, entonces uno de ellos sirve la petición pero de forma remota. Esto es lo que nosotros denotamos como *servicio remoto*. Debemos señalar también que un cliente conectado a un servidor que solicita un vídeo que no se encuentra almacenado en él (lo que denotamos como *vídeo no local*), también genera un servicio remoto. Hay que señalar que un servicio remoto consume más recursos que un servicio local, ya que consume: i) ancho de banda local en el servidor remoto, ii) ancho de banda del enlace de salida del servidor remoto, iii) ancho de banda del conmutador ATM, y iv) ancho de banda en el enlace de entrada del servidor local.

En un trabajo previo, Tay y Pang [33] propusieron un algoritmo llamado GWQ (*Global Wait Queue*) para minimizar el tiempo medio de espera utilizando la estrategia de compartir la carga entre los servidores. La estrategia de servicio en este algoritmo es de tipo *unicast*, es decir, se dedica un canal de servicio o stream para servir el objeto solicitado por cada una de las peticiones que llegan. Estos autores también desarrollaron en [33] un protocolo para controlar la compartición de la carga en arquitecturas distribuidas. Además, presentan un modelo basado en teoría de colas que caracteriza el rendimiento del algoritmo GWQ con bastante exactitud, el cual es validado por una serie de simulaciones. Los autores demostraron a través de simulaciones que el rendimiento de su algoritmo de compartición de la carga es casi óptimo.

En [33] los autores, para simplificar, impusieron dos suposiciones: i) los vídeos son solicitados de forma uniforme (es decir, tienen la misma probabilidad de ser solicitados); y ii) todos los vídeos están almacenados en todos los servidores. Estas dos simplificaciones se aplican principalmente a su modelo analítico y al

análisis de las simulaciones, no al algoritmo en sí.

En esta memoria, proponemos un nuevo algoritmo de tipo unicast para compartir la carga, que llamamos *Popularity and Partial Replication Load Sharing* (PRLS a partir de ahora), en el cual introducimos dos nuevas características: i) los vídeos tienen distintas probabilidades de ser solicitados, es decir, en cada servidor son solicitados de acuerdo con su *popularidad*; y ii) sólo los vídeos más populares están almacenados en todos los servidores. Esto es lo que nosotros llamamos *replicación parcial*.

Estas nuevas características permiten presentar un algoritmo unicast más realista para compartir la carga en un sistema distribuido de VoD. Como ya hemos mencionado, el algoritmo GWQ y las infraestructuras CDN se centran en la compartición de la carga en arquitecturas distribuidas, suponiendo que todos los contenidos (los vídeos) ofrecidos por el proveedor de programas están replicados en todos los servidores del sistema. Estas aproximaciones no están limitadas intrínsecamente sólo a la replicación total del contenido, pero no tienen en cuenta el impacto que la replicación parcial puede suponer en el rendimiento del sistema.

Una de las aportaciones de esta tesis es precisamente dar soluciones a un sistema distribuido de VoD donde sí hay replicación parcial del contenido y donde la estrategia de servicio es de tipo unicast. Obviamente, la replicación total podría suponer que la capacidad de almacenamiento de un servidor individual debería ser lo suficientemente grande para guardar todos los programas. En este capítulo demostraremos que nuestro algoritmo todavía mantiene tiempos de espera razonables usando menos capacidad de almacenamiento en los servidores, ya que sólo replicamos los vídeos más populares.

Una diferencia importante entre los trabajos que consideran replicación parcial [48], [49], [50], [51] y nuestra propuesta es que ellos no consideran la compartición de la carga cuando un servidor está sobrecargado. Estas aproximaciones consideran la capacidad de servicio (el número de streams) que tiene el servidor y la capacidad de almacenamiento como restricciones en sus planteamientos. Sin embargo, como en estos estudios no se considera la funcionalidad de la compartición de la carga entre los servidores, no consideran la restricción que podría suponer la capacidad de la red. Uno de los objetivos de este capítulo es precisamente considerar un sistema donde haya compartición de la carga y estudiar el impacto que el algoritmo PRLS tiene en todos los parámetros del sistema.

En este capítulo presentamos un modelo analítico basado en teoría de colas que captura el rendimiento del algoritmo PRLS. La importancia de este modelo es que nos ayuda a la hora de analizar el comportamiento del algoritmo y del sistema. Así mismo, es una herramienta útil, ya que nos sirve para estimar qué parámetro (y cuándo) va a ser el cuello de botella del sistema. Otra aplicación del modelo es que nos permite utilizar ecuaciones para estimar, el tiempo medio de espera para los distintos porcentajes de replicación, teniendo en cuenta la influencia de la popularidad de los vídeos. Estas estimaciones podrían ser utilizadas para decidir cuál sería el porcentaje de replicación apropiado de

forma que garantice un tiempo de espera lo suficiente pequeño y razonable. Además, el modelo analítico puede ayudarnos para seleccionar el tamaño de la red, el número óptimo de servidores que debe tener el sistema para lograr un tiempo de espera pequeño, y para prevenir que la red se convierta en el cuello de botella. En resumen, la utilidad que vamos a obtener de este modelo, aparte de que nos permite poder estimar el tiempo de espera de los usuarios, es que nos permite ver la relación entre la popularidad y la replicación parcial con el rendimiento del sistema. Podremos explotar esta relación para ahorrar costes sin reducir el rendimiento con sólo replicar en todos los servidores los vídeos populares apropiados.

En las siguientes secciones presentamos detalladamente el algoritmo que proponemos, el modelo analítico que captura el rendimiento del algoritmo y, en la sección de los experimentos, validaremos el modelo con simulaciones.

2.2. Compartición de la carga en sistemas distribuidos de VoD

El principal objetivo de los algoritmos de compartición de la carga en un sistema distribuido de VoD es minimizar el tiempo de espera permitiendo que servidores con recursos disponibles, ayuden a otros servidores que se encuentren momentáneamente saturados de trabajo. Mientras que los sistemas convencionales requieren que los algoritmos de compartición de la carga sean capaces de balancear la carga [52], este requisito es de menor consideración en sistemas de VoD porque los servicios remotos consumen ancho de banda en la red (en los enlaces de los servidores locales y remotos así como en el conmutador ATM) y la red es el elemento que se convierte en el cuello de botella. En [33] los servicios remotos sólo se deben a la compartición de la carga. Sin embargo, como en nuestro sistema existe replicación parcial, un usuario puede solicitar un vídeo que no esté almacenado de forma local. En este caso también se generaría un servicio remoto.

Como resaltamos en la introducción de este capítulo, nosotros proponemos un algoritmo de compartición de la carga que hemos llamado PRLS que está basado en uno previo llamado *Global Wait Queue* (GWQ) propuesto por Tay y Pang en [33]. En las siguientes secciones exponemos ambos algoritmos, explicamos las diferencias entre ambos e indicamos cuáles son nuestras principales aportaciones.

2.2.1. El algoritmo GWQ

GWQ es un algoritmo para compartir la carga en un sistema distribuido de servidores como el que muestra la Figura 1.6 del Capítulo 1. Este algoritmo administra una única cola global de peticiones pendientes de todos los servidores desde la que los servidores no saturados obtienen trabajos adicionales. La implementación de GWQ requiere dos colas en cada servidor: una Cola Local para las peticiones realizadas por los clientes conectados al servidor (a partir

de ahora **peticiones locales**) y una Cola Remota para las peticiones que se reciben de otros servidores saturados (a partir de ahora **peticiones remotas**). A continuación describimos las principales características del algoritmo GWQ:

1. Las peticiones locales a un servidor son servidas por él mientras tenga recursos locales disponibles.
2. Un servidor genera una petición remota sólo si recibe una petición local y está totalmente sobrecargado, es decir, no tiene suficientes recursos para atender la petición.
3. La prioridad de cada servidor es servir las peticiones de la Cola Local. Las peticiones de la Cola Remota se atienden sólo cuando la Cola Local está vacía.
4. Cada petición es servida por el mismo servidor durante todo el tiempo de servicio.

Como ya se mencionó en la introducción del capítulo y en [3], el algoritmo anterior presenta algunas desventajas:

- En cada servidor hay una copia de cada uno de los vídeos. La replicación total no es un prerrequisito para este algoritmo, pero el caso de la replicación parcial no es analizado por los autores del algoritmo.
- No se considera el tema de la popularidad de los vídeos, ya que todos los vídeos se solicitan con la misma probabilidad.
- En el caso de existir varios servidores candidatos para realizar un servicio remoto, éste se asigna al primer servidor que devuelve una respuesta positiva. De esta forma la carga actual de cada servidor potencialmente candidato no se tiene en cuenta. Por lo tanto, el servicio remoto asignado puede saturar la capacidad del servidor seleccionado, lo que podría incrementar la posibilidad de que éste genere nuevas peticiones remotas.

En este último caso, se podría llegar a saturar la red con la generación de nuevos servicios remotos y, al no haber recursos disponibles, el tiempo de espera aumenta. Esto se puede evitar con una asignación adecuada. Por ejemplo, supongamos que dos servidores, S_j y S_k, pueden realizar el servicio de una petición remota al vídeo i y que el servicio de esta petición no satura el ancho de banda local del servidor S_j pero sí el del servidor S_k. Si este último servidor realiza el servicio, la próxima petición local al servidor S_k, por ejemplo al vídeo n, se convierte en una nueva petición remota (ya que el servidor se ha quedado sin recursos) que será atendida a través de la red por otro servidor que pueda atenderla. Evidentemente, esta nueva petición remota consume recursos de la red, pero este consumo se podría haber evitado si se permite que el servidor S_j realice el servicio remoto de la petición al vídeo i.

2.2.2. El algoritmo de compartición de la carga PRLS

Nosotros proponemos un nuevo algoritmo de compartición de la carga que denotamos como PRLS, inspirado en el algoritmo GWQ y que mantiene las características 3 y 4 descritas en la Sección 2.2.1. Este nuevo algoritmo está diseñado para considerar la replicación parcial, la popularidad de los vídeos y para balancear la carga cuando se hacen necesarios los servicios remotos.

La replicación parcial introduce dos nuevas características:

i) La primera es que cuando un servidor saturado recibe una petición local inicia un diálogo con el resto de servidores para asignar esta petición a uno de ellos. Sin embargo, debido al hecho de la replicación parcial, puede ocurrir que no todos los servidores remotos tengan una copia del vídeo solicitado y por lo tanto sólo se deberían avisar a los servidores apropiados. Para este fin suponemos que cada servidor conoce la localización de cada vídeo en el sistema usando, para ello, una tabla local con una entrada para cada vídeo que indica qué servidor mantiene una copia. Esta tabla es lo que llamamos *mapa de localización* de los vídeos. Sólo se solicita ayuda a aquellos servidores que mantienen una copia del vídeo solicitado y se elige el que tiene menor carga. Esto reduce el número de mensajes que se generan con el protocolo de asignación de los servicios remotos.

ii) La segunda es que una petición local a un vídeo no local genera una petición remota y, por lo tanto, ésta deberá ser atendida por un servidor remoto. De nuevo, en este caso se debe utilizar el mapa de localización para direccionar los mensajes de ayuda a los servidores que tienen una copia del vídeo. Obviamente, una consecuencia de la replicación parcial es que el número total de servicios remotos se incrementa debido a las peticiones de los vídeos no locales.

Como se indica en el anterior apartado i), nuestro algoritmo PRLS utiliza la carga actual de los servidores como criterio a la hora de seleccionar el servidor remoto. El servicio se asigna al servidor que esté menos saturado. Para medir el impacto de este criterio en el algoritmo PRLS frente al criterio usado en [33] descrito en la sección 2.2.1, presentamos el siguiente experimento. Suponemos que el número de clientes conectados a cada servidor es $N_{term} = 90$ mientras que el número de streams que se pueden utilizar a la vez en cada servidor es $N_{stream} = 50$. Cada cliente genera peticiones a intervalos de tiempo que obedecen a una distribución con un tiempo medio de $T_{sleep} = 7200$ segundos. Los tiempos de servicio (la duración de los vídeos) se distribuyen de forma uniforme en el intervalo [3600,10800] segundos. El número total de vídeos ofrecidos por el sistema es $M = 100$, el porcentaje de replicación es del 100 % y el grado de popularidad es $\xi = 0$. Con estos parámetros los algoritmos GWQ y PRLS difieren sólo en el criterio de asignación de los servicios remotos. Luego los dos algoritmos están en las mismas condiciones y podemos medir el impacto que tiene balancear la carga en el tiempo de espera de los clientes.

Los resultados de este experimento se muestran en la Figura 2.1. Podemos observar que para cualquier número de servidores, los tiempos de espera obtenidos por el algoritmo PRLS son siempre menores que los obtenidos por el algoritmo GWQ. Por tanto, balancear la carga ayuda a mantener los servidores con el mismo nivel de carga y, como consecuencia, los tiempos de espera se reducen.

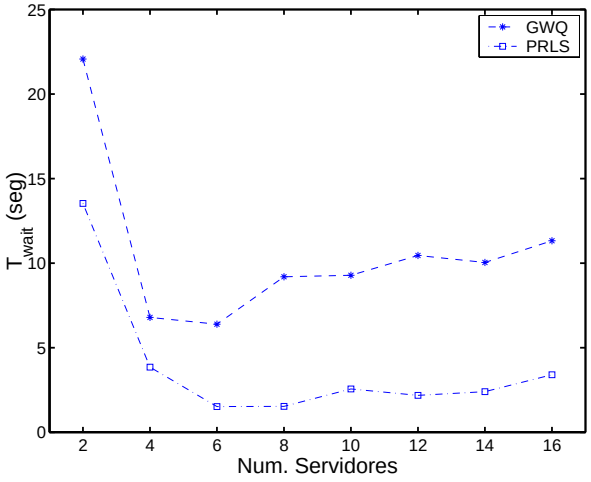


Figura 2.1 Tiempos de espera para los algoritmos GWQ y PRLS.

2.3. Modelo analítico

Una de las razones para obtener el modelo analítico de nuestro algoritmo PRLS es entender cómo interactúan los recursos, así como estudiar cómo influye la compartición de la carga en el rendimiento del sistema. Además, el modelo nos sirve como una herramienta de configuración del sistema para encontrar el número óptimo de servidores en el sistema, y la capacidad adecuada de los servidores y de la configuración de la red.

Debemos hacer notar aquí que hemos basado nuestro modelo en los tres pasos presentados en [33], para el algoritmo GWQ. Nuestra contribución es que hemos incorporado la característica de la replicación parcial en ese modelo y podemos cuantificar el impacto que tendrá el porcentaje de replicación en el rendimiento del sistema. Otro efecto que captura nuestro modelo es la influencia del grado de popularidad. Como veremos en las siguientes secciones, nuestro modelo podría también sernos útil para decidir, dado un grado de popularidad, cuál es el porcentaje de replicación adecuado en el sistema.

2.3.1. Modelos de colas utilizados

Antes de presentar el modelo analítico, vamos a introducir la herramienta que utilizamos para ello: la teoría de colas.

La teoría de colas es una formulación matemática para la optimización de sistemas en que interactúan dos procesos normalmente aleatorios: un proceso de llegadas de clientes y un proceso de servicio a los clientes, en los que existen fenómenos de acumulación de clientes en espera del servicio, y donde existen reglas definidas (prioridades) para la prestación del servicio.

Para describir un sistema de colas, se emplea la notación de Kendall que es un estándar en la literatura de la teoría de colas. Esta notación consiste en un grupo de letras separadas mediante barras de la forma, $A/B/c/m/d$, y donde cada una de las letras tiene el siguiente significado:

- **A** designa el proceso de llegadas; más concretamente describe el tipo de distribución del tiempo entre llegadas.
- **B** designa el proceso de servicio; es decir, describe la distribución del tiempo de servicio. En todos los casos supondremos que la duración del tiempo de servicio es independiente de la distribución de las llegadas.
- **c** es el número de canales disponibles para realizar servicios.
- **m** es el número máximo de usuarios simultáneos (que incluye el número de usuarios en cola y los que están recibiendo servicio) que se admiten en el sistema. Si se excede esta capacidad, se rechazan las peticiones que intentan entrar en el sistema (bloqueo de las peticiones). Si esta capacidad es infinita, se omite de la notación.
- **d** designa la disciplina de la cola, es decir, el proceso de decisión de cuál de los usuarios en espera va a pasar a recibir servicio. Se suele considerar una disciplina del tipo FCFS (*First Come First Served*, o lo que es lo mismo, "el primero que llega, el primero que se sirve") que se suele omitir en la notación.

Existen una serie de símbolos estándares para denotar algunas funciones de distribución más usuales y que sustituyen a la notación de las distribuciones A y B. De entre ellos, cabe destacar la notación asignada a la distribución de probabilidad exponencial, que se denota por M (M proviene de proceso de Markov, [53]).

El estado del arte actual cubre una amplia serie de sistemas de colas o modelos de colas. De estos modelos se conocen, dependiendo del tipo de sistema, o todas o casi todas las medidas del rendimiento del sistema. Estas medidas se obtienen a partir de los parámetros conocidos del sistema de colas (tasa de llegada de peticiones (λ), tiempo medio de servicio (T_{active}), número de canales (c), etc). De entre las medidas del rendimiento de un sistema de colas caben destacar:

- **El factor de utilización, ρ**

  El *factor de utilización* de un sistema de colas tiene la siguiente expresión

$$\rho = \frac{\lambda \cdot T_{active}}{c} \tag{2.1}$$

Este parámetro tiene varias interpretaciones. Por un lado se puede interpretar como la media de tiempo que cada canal de servicio está ocupado, o como la probabilidad de que un canal dado esté ocupado. También se puede interpretar como la probabilidad de que el sistema esté ocupado. Aunque también se puede interpretar como el número medio de peticiones que están siendo atendidas en un instante dado. En cualquier caso, este parámetro nos dice cómo de sobrecargado está el sistema de colas. Es decir, ρ es una medida de la congestión del sistema.

- **El tiempo medio de espera en cola,** T_{wait}

  Esta medida nos dice cuánto tiempo (de media) está el usuario en cola esperando para recibir servicio. En esta memoria este es el parámetro que vamos a minimizar.

- **El tiempo medio del usuario en el sistema,** W

  Esta medida nos dice cuanto tiempo (de media) está el usuario dentro del sistema. Este tiempo es el periodo que pasa desde que el usuario entra en el sistema hasta que sale de él. Comprende los periodos de tiempo de espera en cola (T_{wait}) y de realización de servicio (T_{active}). Es decir, $W = T_{wait} + T_{active}$.

- **Número medio de los usuarios en cola,** L_q

  Nos dice cuántos clientes (de media) están en cola. En esta medida no están incluidos el número medio de clientes que están recibiendo servicio (L_s).

- **Número medio de usuarios en el sistema,** L

  Es el número medio de clientes que están en cola y que están recibiendo servicio. Es decir, $L = L_s + L_q$.

Todas estas medidas se pueden calcular a partir de la conocida fórmula o ley de Little. La ley de Little establece que *el número medio de clientes en un sistema de colas es igual a la tasa de llegada de los clientes al sistema por el tiempo medio que el usuario pasa en ese sistema.* Es decir,

$$L = \lambda \cdot W \tag{2.2}$$

A partir de esta fórmula podemos obtener otras relaciones como, $L_q = \lambda \cdot T_{wait}$ y $L_s = \lambda \cdot T_{active}$.

Si conocemos λ y T_{active}, la ley de Little permite calcular las medidas de rendimiento, L, L_q, W y T_{wait}, si conocemos alguna de ellas. Por ejemplo, si conocemos W, entonces podemos calcular $L = \lambda \cdot W$, $T_{wait} = W - T_{active}$ y $L_q = \lambda \cdot T_{wait}$.

De entre todos los modelos de colas, en esta sección vamos a presentar los modelos que se utilizan para modelar nuestro sistema distribuido de VoD, y que son las colas $M/M/c$ y $M/M/c/K/K$.

Las colas de tipo $M/M/c$ y $M/M/c/K/K$

De acuerdo con la notación de Kendall, las colas tipo $M/M/c$ corresponden a sistemas donde, tanto la distribución del tiempo entre las llegadas de las peticiones, como la distribución del tiempo de servicio son exponenciales (ambas independientes). Este sistema tiene c canales para realizar los servicios y no tiene restricción en el número de usuarios que se admiten en el sistema, es ecir, que el número de clientes que pueden acceder al sistema es infinito ($m = \infty$). De esta cola se pueden obtener las siguientes medidas del rendimiento: L, L_q, T_{wait} y W.

En el modelo de cola anterior, la población fuente es infinita. Sin embargo, existen sistemas de colas donde sólo hay un número finito de usuarios y se suele denotar como un modelo de cola cerrada o incluso cíclico. Este modelo es uno de los más útiles de todos los modelos de colas. En la Figura 2.2 se muestra un sistema cerrado. Como podemos ver, hay una población finita de usuarios que son los únicos que entran al sistema para recibir servicio. Una vez finalizado el servicio, el usuario sale del sistema y cuando necesite un nuevo servicio entrará de nuevo.

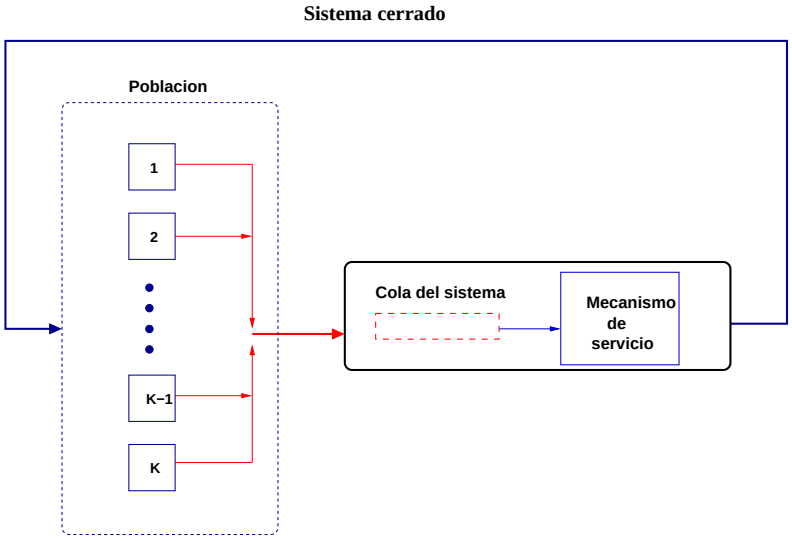


Figura 2.2 Sistema de cola cerrado.

El sistema cerrado que utilizamos en nuestro modelo es del tipo $M/M/c/K/K$, que consiste en K usuarios que entran al sistema, siguiendo cada uno, una distribución exponencial con la misma media $E[O]$. El tiempo de servicio también es exponencial y tanto el número de usuarios permitidos dentro del sistema como la población de clientes es K. En este modelo el número de peticiones por unidad de tiempo (*throughput*) o tasa de llegadas al sistema, λ se calcula mediante la expresión

$$\lambda = \frac{K}{E[O] + T_{wait} + T_{active}} \tag{2.3}$$

Aplicando la ley de Little obtenemos

$$T_{wait} = L_q/\lambda = \frac{L_q}{K} \cdot (E[O] + T_{wait} + T_{active}) \tag{2.4}$$

Despejando T_{wait} de la ecuación 2.4 obtenemos

$$T_{wait} = \frac{L_q}{K - L_q} \cdot (E[O] + T_{active}) \tag{2.5}$$

Este sistema de colas modela bastante bien a cada uno de los servidores del sistema (vistos de forma individual) como veremos en la siguiente subsección.

Notación del modelo

En la Tabla 2.1 se muestran los parámetros de nuestro modelo. Como en la mayoría de estudios previos [33], hemos supuesto que los tiempos entre las llegadas de dos clientes consecutivos a un servidor, siguen una distribución exponencial y que el tiempo de servicio de un vídeo sigue otra distribución exponencial. Hemos hecho esta suposición porque la distribución exponencial se usa mucho para simular el tiempo entre llegadas, cuando las llegadas son completamente aleatorias (como es el caso de las demandas que se producen en un sistema de VoD) y de tiempos de servicio. Ejemplos de procesos que han sido modelados mediante distribuciones exponenciales: la duración de conversaciones telefónicas, el tiempo entre fallos en ciertos elementos, tiempo entre interrupciones en la CPU de un ordenador, etc.

Un servidor de nuestro sistema puede ser modelado como una cola cerrada, es decir, puede ser modelado como la cola $M/M/N_{stream}/N_{term}/N_{term}$ con N_{term} clientes que acceden al servidor, con N_{stream} canales de servicio y con una capacidad máxima permitida de N_{term} clientes (clientes en cola más clientes recibiendo servicio). Sustituyendo los parámetros correspondientes en la ecuación 2.3, tenemos que la expresión de la tasa de llegadas al sistema es

$$\lambda = \frac{N_{term}}{T_{sleep} + T_{wait} + T_{active}} \tag{2.6}$$

Los vídeos y documentales típicamente tiene una duración mayor de 1500 segundos. En contraste los usuarios podrían tolerar sólo tiempos de espera pequeños (por ejemplo menos de 100 segundos). Por tanto, esperamos que se verifique en la práctica que $T_{wait} << T_{active}$. Por tanto, de la ecuación 2.6 se tiene que

$$\lambda \approx \frac{N_{term}}{T_{sleep} + T_{active}} \tag{2.7}$$

que representa el valor de λ en el resto de esta memoria.

Suponemos que los servidores son homogéneos, es decir, que tienen el mismo número de clientes N_{term}, la misma capacidad de servicio (mismo número de canales que corresponde con el número de streams del servidor) N_{stream} y distribuciones exponenciales similares para modelar la llegada de peticiones (de media T_{sleep} segundos) y de tiempos de servicios (de media T_{active} segundos).

Un parámetro importante en nuestro sistema es el *factor de utilización*. Como mencionamos anteriormente, este valor tiene varias interpretaciones. En este caso, nosotros lo interpretamos como la probabilidad de que el sistema esté ocupado. El factor de utilización de un servidor i puede ser calculado aproximadamente como (ver ecuación 2.1)

$$\rho_i = \frac{\lambda \cdot T_{active}}{N_{stream}} \tag{2.8}$$

2.3.2. Modelo analítico del algoritmo PRLS

Nuestro modelo está basado en dos observaciones establecidas en [33]: 1) la compartición de la carga no afecta significativamente al factor de utilización; y 2) el efecto de la compartición de la carga en el rendimiento de un servidor es similar al de añadir más capacidad al servidor. La primera observación nos dice que el factor de utilización es común para cualquier servidor (es decir, $\forall i \in \{1, \ldots, N_{server}\}, \rho_i = \rho$). Este hecho nos permite estimar el factor de

Parámetro	Definición	Valor por defecto
N_{server}	Nº servidores en el sistema	-
B_{switch}	Ancho banda conmutador ATM	1000 Mbps
B_{link}	Ancho banda de un enlace (en cada dirección)	155 Mbps
N_{stream}	Nº máx. canales por servidor (streams)	50
S_i	Capacidad mín. de almacenamiento en servidor i	ecua. 2.15
N_{term}	Número de clientes conectados a cada servidor	90
T_{sleep}	Tiempo entre peticiones	distr. expon. (media 7200 seg.)
T_{active}	Duración servicio de un vídeo	distr. expon. (media 7200 seg.)
B_{stream}	Ancho de banda que ocupa cada stream	15 Mbps
M	Nº de vídeos	100
ξ	Grado de popularidad	1, 1.8
p_j	Probabilidad de solicitar el vídeo j	ecua. 1.1
L_j	Duración del vídeo j	-
$1 - F$	Probabilidad media de solicitar un vídeo no local	ecua. 1.2, 1.3
T_{wait}	Tiempo de espera	-

Tabla 2.1 Parámetros del modelo analítico del algoritmo PRLS.

utilización para cualquier algoritmo de compartición de la carga simplemente calculando ρ para un único servidor. Por tanto, nuestro modelo del sistema se puede obtener desde el punto de vista de uno de los servidores.

La observación 2) captura la compartición de la carga mediante la adición de más capacidad al servidor gracias a que los demás servidores cooperan en la compartición de la carga.

Además, basándonos en los resultados de numerosas simulaciones de nuestro algoritmo, hemos obtenido una nueva observación: 3) el hecho de que no todos los vídeos están replicados en el sistema puede ser capturado mediante la sustracción de parte de la capacidad de la red. Esto es debido a que algunos de los servicios remotos deben ser realizados obligatoriamente (los debidos a las peticiones a vídeos no locales) lo que obviamente va a sustraer recursos para la compartición de la carga. La incorporación de estas tres observaciones nos permite organizar el modelo analítico en tres pasos:

Paso 1 $M/M/N_{stream}/N_{term}/N_{term}$ es el modelo analítico de un servidor aislado. Para este modelo, calculamos el throughput λ (ecuación 2.6) y el factor de utilización ρ_i (ecuación 2.8).

El objetivo de este paso es calcular la tasa de llegada de las peticiones a servicios remotos. En el algoritmo PRLS sabemos que una petición es candidata para un servicio remoto obligatorio cuando la petición es para un vídeo que no está almacenado en el servidor (una petición a un vídeo no local). Por tanto una estimación de la tasa de llegada de peticiones a servicios remotos obligatorios λ_r puede ser,

$$\lambda_r = \lambda \cdot (1 - F) \tag{2.9}$$

donde $(1 - F)$ es la probabilidad de solicitar un vídeo no local.

Paso 2 $M/M/n$ es el modelo para calcular la totalidad de capacidad remota disponible para la compartición de la carga. Elegimos este modelo porque asumimos que no hay abandono en el sistema [28], es decir, los usuarios esperan hasta que son servidos. Desde la perspectiva de un servidor, la capacidad total remota disponible puede ser vista como una capacidad adicional de n streams. Es aquí donde usamos la tasa de llegada λ_r para calcular estos n streams. n se determina en función de: los anchos de banda del enlace (B_{link}) y del conmutador (B_{switch}), del factor de utilización y del número de servidores remotos y de la probabilidad de solicitar un vídeo no local. Consideramos esta probabilidad porque las peticiones a vídeos no locales deben ser servidas remotamente. Estos servicios remotos obligatorios también consumen algunos recursos en el servidor remoto, además de ancho de banda del enlace y del conmutador. En consecuencia estos recursos ocupados no se pueden utilizar para compartir la carga.

Vamos a describir cómo calculamos n. Un servidor remoto r que funciona de forma independiente tiene de media una capacidad disponible de

$$(1 - \rho_r) \cdot N_{stream}$$

streams.

Sin embargo, parte de esta capacidad está ocupada en servir los servicios remotos obligatorios, que corresponden a los vídeos no replicados que están almacenados en el servidor r (es decir, los vídeos no replicados pero que son locales en ese servidor). Suponiendo un escenario en el que todos los servidores en el sistema solicitan vídeos no locales (la probabilidad de solicitar vídeos no locales será $\sum_{j=1}^{N_{server}}(1-F_j)$), la media de peticiones que corresponden a los vídeos almacenados en el servidor r se calcula como

$$\frac{\sum_{j=1}^{N_{server}}(1-F_j)}{N_{server}} = \frac{N_{server} - \sum_{j=1}^{N_{server}}(F_j)}{N_{server}} = 1 - F$$

Esto es debido a que hemos supuesto una estrategia de distribución equitativa de los vídeos en la que todos los vídeos no replicados se distribuyen de forma uniforme entre todos los servidores. Luego podríamos esperar que las peticiones a servicios remotos se harán de forma uniforme. Por tanto, esos servicios remotos ocupan una capacidad en el servidor r de

$$(1-\rho_r)\cdot N_{stream}\cdot(1-F)$$

Es decir, podemos suponer que el servidor r ofrece una capacidad disponible aproximada de

$$(1-\rho_r)\cdot N_{stream} - (1-\rho_r)\cdot N_{stream}\cdot(1-F)$$

streams. Por tanto el servidor i ve disponible (de promedio) una capacidad total de $n_{idle}(i)$, que es la suma de la capacidad disponible de los otros servidores $r \neq i$

$$n_{idle}(i) = \sum_{r=1, r\neq i}^{N_{sever}} \Big((1-\rho_r)\cdot N_{stream} - (1-\rho_r)\cdot N_{stream}\cdot(1-F) \Big)$$

De la observación 1) sabemos que el factor de utilización es común para cualquier servidor (es decir, $\forall r \in \{1,\ldots,N_{server}\}, \rho_r = \rho$). Como consecuencia, calculamos el promedio de la capacidad disponible que ve un servidor, n_{idle}, como

$$\begin{aligned} n_{idle} &= \Big[(1-\rho)\cdot N_{stream} - (1-\rho)\cdot N_{stream}\cdot(1-F) \Big] \cdot (N_{server}-1) \\ &= (1-\rho)\cdot N_{stream}\cdot F\cdot(N_{server}-1) \end{aligned} \tag{2.10}$$

Sin embargo sólo parte de esta capacidad disponible puede ser utilizada para la compartición de la carga, básicamente porque tenemos que tener en cuenta tanto el ancho de banda del enlace como del conmutador.

El número de streams que pueden ser transmitidos a través de uno de los enlaces conectado al servidor i es como mucho

$$n_{link}^{max} = B_{link}/B_{stream}$$

Sin embargo, de nuevo parte de esos streams no pueden ser utilizados para compartir la carga, porque algunos de ellos se dedican para servir los vídeos que son locales en el servidor i y que no están replicados en otros servidores (los servicios remotos obligatorios). Suponiendo el peor de los casos donde todos los servidores (excepto el servidor i) solicitan vídeos no locales y suponiendo que esas peticiones sólo pueden ser servidas por el servidor i, la probabilidad de solicitar un vídeo que sólo se encuentre en el servidor i es ahora

$$\sum_{j=1,j\neq i}^{N_{server}} (1 - F_j)$$

Por tanto usando esta estimación y la observación 3) podemos deducir que el número de streams que se pueden utilizar del enlace conectado al servidor i para la compartición de la carga se calcula como

$$n_{link}(i) = \frac{B_{link}}{B_{stream}} - \frac{B_{link}}{B_{stream}} \cdot \sum_{j=1,j\neq i}^{N_{server}} (1 - F_j)$$

Como nos interesa la capacidad media del enlace, n_{link}, podemos aproximar la probabilidad de solicitar un vídeo no local en un servidor $1-F_j$ por la probabilidad de solicitar un vídeo no local cuya expresión es $1-F$. Por tanto, n_{link} se calcula como

$$\begin{aligned} n_{link} &= \frac{B_{link}}{B_{stream}} - \frac{B_{link}}{B_{stream}} \cdot (1 - F) \cdot (N_{server} - 1) \\ &= \frac{B_{link}}{B_{stream}} \cdot [1 - (1 - F) \cdot (N_{server} - 1)] \end{aligned} \tag{2.11}$$

Por otro lado, el número de streams por servidor que pueden ser transmitidos a través del conmutador es como mucho

$$n_{switch}^{max} = B_{switch}/(B_{stream} \cdot N_{server})$$

Siguiendo una deducción similar a la anterior, sabemos que parte de esos streams no se pueden ofrecer para poder compartir la carga, ya que algunos de ellos se dedicarán a servir los vídeos que son locales en el servidor i y que no están almacenados en ninguno de los otros. Por tanto el número de streams del conmutador que se pueden utilizar para compartir la carga se calcula como

$$n_{switch}(i) = \frac{B_{switch}}{B_{stream} \cdot N_{server}} - \frac{B_{switch}}{B_{stream} \cdot N_{server}} \cdot \sum_{j=1, j \neq i}^{N_{sever}} (1 - F_j)$$

Como estamos interesados en la capacidad media del conmutador, n_{switch}, podemos aproximar la probabilidad de solicitar un vídeo no local en un servidor $1 - F_j$ por la probabilidad de solicitar un vídeo no local cuya expresión es $1 - F$. Por tanto, n_{switch} se calcula como

$$\begin{aligned} n_{switch} &= \frac{B_{switch}}{B_{stream} \cdot N_{server}} - \frac{B_{switch}}{B_{stream} \cdot N_{server}} \cdot (1 - F) \cdot (N_{server} - 1) \\ &= \frac{B_{switch}}{B_{stream} \cdot N_{server}} \cdot [1 - (1 - F) \cdot (N_{server} - 1)] \end{aligned} \tag{2.12}$$

Una vez que hemos calculado n_{idle}, n_{link} y n_{switch} podemos deducir el número medio de streams que están disponibles remotamente y que denotamos como n,

$$n = min(n_{idle}, n_{link}, n_{switch}) \tag{2.13}$$

Una vez calculado n consideramos el modelo de cola $M/M/n$ con un tasa de llegada dada por λ_r (de la ecuación 2.9) y un tiempo medio de servicio de T_{active} segundos. Recordemos que n es el número de canales (streams en el caso del servidor de VoD) disponibles que tiene el sistema de colas para realizar servicios. Con este modelo calculamos m que es el número medio de streams activos. Una observación importante aquí es que para este modelo de cola, el cálculo de m es independiente de n [53]. Esta m es nuestra estimación del número medio de peticiones a vídeos no locales que se sirven de forma remota. La expresión para calcular m viene dada por [53],

$$m = \lambda_r \cdot T_{active} = \lambda \cdot (1 - F) \cdot T_{active} \tag{2.14}$$

Debemos señalar aquí que las ecuaciones 2.10, 2.11 y 2.12 dependen de F. Luego la influencia de la replicación se puede medir en nuestro sistema. Claramente podemos ver que cuando el porcentaje de replicación aumenta, lo hacen también s y F (ecuaciones 1.3 y 1.2) y por tanto aumentan n_{idle}, n_{link} y n_{switch}. En otras palabras, más replicación significa que se producirán menos peticiones que tengan que ser servidas de forma remota y por esta razón habrá más recursos disponibles para compartir la carga.

Otro efecto que se captura con nuestro modelo es la influencia del grado de popularidad. De la Sección 1.3 sabemos que cuanto mayor es el grado de popularidad, los vídeos más populares se solicitan con más frecuencia.

Por esta razón F aumenta de nuevo. Y como explicamos antes n_{idle}, n_{link} y n_{switch} aumentan. Esto significa que cuanto más grande es el grado de popularidad más peticiones se realizan de los vídeos replicados (que son vídeos locales) y por tanto se realizan menos peticiones de servicios remotos y por esta razón habrá más recursos disponibles para compartir la carga. Todas estas cuestiones las analizaremos con más detalle en la Sección 2.4.

Paso 3 $M/M/N_{stream} + v/N_{term}/N_{term}$ es nuestro modelo para caracterizar la compartición de la carga, como mencionamos en la observación 2). De hecho a través de las simulaciones, hemos verificado que el rendimiento de nuestro algoritmo PRLS se puede aproximar con un modelo de cola del tipo $M/M/N_{stream}+v/N_{term}/N_{term}$, es decir, cada servidor se comporta como si tuviera una capacidad (virtual) añadida de v streams.

Lo primero es calcular v. Este v es el número medio de recursos disponibles para compartir la carga y se estima como $v = n - m$. Esta es la diferencia entre el número medio de streams disponibles remotamente (n) y el número medio de peticiones que se sirven obligatoriamente de forma remota (m). En otras palabras, v es la capacidad remota que no está ocupada con los servicios obligatorios y que por lo tanto está disponible para compartir la carga. Una vez que hemos calculado v, de las ecuaciones de la cola $M/M/N_{stream} + v/N_{term}/N_{term}$ [53] calculamos el tiempo medio de espera en nuestro sistema (T_{wait}).

Debemos tener cuidado con el valor de v ya que puede tomar valores positivos y negativos. Cuando v es positivo nos indica que hay capacidad remota disponible para realizar la compartición de la carga y por tanto esta capacidad se puede añadir al servidor. Cuando v es negativo lo que ocurre es que hay más peticiones a los vídeos no locales que recursos en la red para atenderlas. Como estas peticiones consumen obligatoriamente ancho de banda del enlace y del conmutador, no hay capacidad disponible para poder compartir la carga y lo que ocurre es que el servidor tiene que reservar $|v|$ streams para atender el tráfico de servicios remotos obligatorios, lo que en realidad repercute en una disminución de la capacidad para atender las peticiones locales.

2.3.3. Estimación de la capacidad del sistema

Una característica importante en el diseño del sistema es la capacidad de almacenamiento, S, que debe ofrecer un servidor. Sea L_j la duración del vídeo j, s el número de vídeos replicados y K_i el conjunto de vídeos no replicados pero que están almacenados en el servidor i. La capacidad mínima de almacenamiento que se requiere en el servidor i, S_i, se puede calcular como

$$S > S_i = \sum_{j=1}^{s} L_j \cdot B_{stream} + \sum_{\forall k \in \mathcal{K}_i} L_k \cdot B_{stream} \tag{2.15}$$

La selección del esquema de distribución influirá en gran mediada en este parámetro. Los requisitos de almacenamiento del disco serán menores si elegimos un esquema de distribución con un bajo porcentaje de replicación y en el que los vídeos que no están replicados son distribuidos de forma uniforme entre todos los servidores, por ejemplo el esquema de distribución cíclica que presentábamos en la Sección 1.3.2 del Capítulo 1. En este caso S_i^{min} es

$$S_i^{min} = \sum_{j=1}^{s} L_j \cdot B_{stream} + \sum_{k=s+i:N_{server}:M} L_k \cdot B_{stream} \tag{2.16}$$

Para otro esquema de distribución, el S_i que se requiere puede ser mayor. Por otro lado, el tamaño actual del almacenamiento de los servidores, dado un esquema de distribución, fijará el porcentaje máximo de replicación y obviamente el número de vídeos totalmente replicados s.

Otros parámetros que podrían estimarse son la capacidad de streams en los servidores (N_{stream}) y el tamaño de la red (B_{link} y B_{switch}) para evitar que cualquiera de esos elementos se convierta en los cuellos de botella del sistema. Para calcular estas capacidades podemos usar el modelo analítico que hemos propuesto.

El Paso 2 de nuestro modelo calcula m (ecuación 2.14) que es el número medio de peticiones de servicios remotos obligatorios. Una de las primeras restricciones para prevenir el colapso en nuestro sistema es que el número medio de peticiones que se sirven de forma obligatoria debe estar acotado y debe ser menor que el número medio de streams remotos disponibles (n). Es decir, se debe cumplir $n > m$.

Desde el punto de vista de un servidor, de la ecuación 2.13 podemos definir N_{stream}^{min}, es decir, la capacidad mínima de streams que debe ofrecer un servidor para prevenir que un servidor se convierta en el cuello de botella del sistema. En este caso un servidor tiene que tener una capacidad de streams disponible que verifique $n_{idle} > n_{idle}^{min} = min(n_{link}, n_{switch})$. La capacidad media disponible de un servidor para compartir con el resto de los servidores viene dada por la ecuación 2.10. Además de las observaciones anteriores, el número mínimo de streams disponibles en el servidor debe ser mayor que m si queremos evitar el colapso en el sistema. Es decir,

$$n_{idle} > n_{idle}^{min} = min(n_{link}, n_{switch}) > \lambda \cdot (1 - F) \cdot T_{active} \tag{2.17}$$

De las ecuaciones 2.17 y 2.10 encontramos que

$$N_{stream} > N_{stream}^{min} = \frac{\lambda \cdot (1 - F) \cdot T_{active}}{(1 - \rho) \cdot F \cdot (N_{server} - 1)} \tag{2.18}$$

Si estudiamos la ecuación 2.13, podemos definir el tamaño mínimo del enlace que necesitamos para prevenir que éste se convierta en el cuello de botella

como, $n_{link} > n_{link}^{min} = min(n_{idle}, n_{switch})$. Además, este número mínimo de streams disponibles de forma remota a través del enlace tiene que ser mayor que m si no deseamos que el enlace sea el que colapse el sistema. En otras palabras,

$$n_{link} > n_{link}^{min} = min(n_{idle}, n_{switch}) > \lambda \cdot (1 - F) \cdot T_{active} \tag{2.19}$$

De las ecuaciones 2.19 y 2.11 encontramos que

$$B_{link} > B_{link}^{min} = \begin{cases} \frac{\lambda \cdot (1-F) \cdot T_{active}}{1-(1-F) \cdot (N_{server}-1)} \cdot B_{stream} & \text{if } C < 1 \\ \infty & \text{if } C \geq 1 \end{cases} \tag{2.20}$$

donde $C = (1 - F) \cdot (N_{server} - 1)$.

Nos gustaría señalar aquí que $(1-F) \cdot (N_{server} - 1)$ podría ser mayor que 1. En tal caso se podría obtener un valor negativo de B_{link}^{min} (si usamos la primera parte de la ecuación 2.20), pero este resultado no tendría sentido físico. Lo que ocurre es que cuando F decrece y N_{server} aumenta, el número de servicios remotos obligatorios aumenta significativamente, con lo que $(1 - F) \cdot (N_{server} - 1)$ puede ser mayor o igual que 1. En otras palabras, el número de peticiones de servicios remotos supera a los recursos disponibles. Nosotros expresamos esta situación con un comportamiento asintótico de B_{link} mediante el término ∞ en la segunda parte de la ecuación 2.20.

Si la cota de la ecuación 2.20 se viola, entonces el enlace es el cuello de botella a la hora de diseñar la red. En este caso, el tiempo de espera se podría reducir incrementando el ancho de banda del enlace y sin necesidad de cambiar ningún otro parámetro.

De forma similar, de la ecuación 2.13 podemos deducir que el tamaño mínimo del conmutador que necesitamos para prevenir que éste se convierta en el cuello de botella debe verificar que, $n_{switch} > n_{switch}^{min} = min(n_{idle}, n_{switch})$. Ahora este número mínimo de streams disponibles de forma remota a través del conmutador tiene que ser mayor que m si no deseamos que el conmutador sea el que colapse el sistema. En otras palabras,

$$n_{switch} > n_{switch}^{min} = min(n_{idle}, n_{link}) > \lambda \cdot (1 - F) \cdot T_{active} \tag{2.21}$$

De las ecuaciones 2.21 y 2.12 encontramos que

$$B_{switch} > B_{switch}^{min} = \begin{cases} \frac{\lambda \cdot (1-F) \cdot T_{active}}{1-(1-F) \cdot (N_{server}-1)} \cdot N_{server} \cdot B_{stream} & \text{if } C < 1 \\ \infty & \text{if } C \geq 1 \end{cases} \tag{2.22}$$

donde $C = (1 - F) \cdot (N_{server} - 1)$.

De nuevo $(1 - F) \cdot (N_{server} - 1)$ podría ser mayor o igual que 1. Como explicamos anteriormente esto significa que se necesitan más recursos para los servicios remotos debido a que el número de peticiones de servicios remotos es desproporcionada con respecto a los recursos de los que disponemos. Nosotros expresamos esta situación mediante un comportamiento asintótico de B_{switch} el cual representamos con el término ∞ en la segunda parte de la ecuación 2.22.

Si se viola la cota de la ecuación 2.22 entonces el conmutador se convierte en el cuello de botella y por lo tanto el tiempo de espera se podría reducir con sólo incrementar el ancho de banda del conmutador.

La expresión 2.18 y las estimaciones 2.20 y 2.22 nos dan unas ecuaciones para calcular la capacidad de streams necesarios tanto para el servidor como para la red. Obviamente estas expresiones pueden ser muy útiles en la fase de planificación de los recursos. Por ejemplo, en tales expresiones podemos ver el impacto de la replicación parcial y el número de servidores en el sistema. Cuando el porcentaje de replicación es del 100 %, entonces $1 - F = 0$ y $N^{min}_{stream} = B^{min}_{link} = B^{min}_{switch} = 0$, esto es, no hay servicios obligatorios que se tengan que servir de forma obligatoria y por tanto no es necesario reservar recursos ni en los servidores ni en la red de interconexión. Sin embargo, cuando el porcentaje de replicación decrece, la probabilidad de solicitar un servicio local (esto es, $1 - F$) aumenta y en consecuencia el divisor en la ecuación 2.18 y en la primera parte de las ecuaciones 2.20 y 2.22 tienden a disminuir y por tanto N^{min}_{stream}, B^{min}_{link} y B^{min}_{switch} tienden a aumentar. Este crecimiento de los recursos del servidor/red es más brusco cuando el número de servidores en el sistema aumenta. De hecho los valores asintóticos de B^{min}_{link} y B^{min}_{switch} se podrían alcanzar. Aquí tenemos una situación donde grandes valores de $1 - F$ (debido a un porcentaje de replicación bajo o incluso una baja popularidad) podrían colapsar el enlace o el conmutador. En la próxima sección analizaremos este problema en más profundidad.

2.4. Resultados experimentales

En esta sección presentamos varios experimentos para medir el rendimiento de nuestro algoritmo PRLS. Además, validamos el modelo analítico a través de estos experimentos. Así mismo, estudiamos la influencia de la replicación y del grado de popularidad en el tiempo de espera de los clientes. Empezamos con una descripción de los parámetros que usamos en estas simulaciones, a continuación describimos los experimentos que hemos realizado, examinamos la validez del modelo y analizamos los resultados.

2.4.1. Parámetros de las simulaciones

En las simulaciones utilizamos los parámetros representados en la Tabla 2.1. El número de clientes conectados a cada servidor es $N_{term} = 90$ mientras

que el número de streams que se pueden utilizar a la vez en cada servidor es $N_{stream} = 50$. Dependiendo del momento del día en el que se realizan las peticiones y del cliente, puede suceder que entre que termine un vídeo y que el cliente demande otro, pasen unos minutos o pasen horas. Incluso puede suceder que en cada hogar ("cliente"), haya distintos usuarios (adultos, adolescentes, niños) y cada uno de ellos quiera ver un contenido distinto a los que el resto de su familia consume. Por tanto, podemos suponer que, de media, cada cliente va a generar una petición cada dos horas. Por eso, en nuestras simulaciones suponemos que cada cliente genera peticiones a intervalos de tiempo que obedecen a una distribución con un tiempo medio de $T_{sleep} = 7200$ segundos. Los tiempos de servicio (la duración de los vídeos) se distribuyen también de forma exponencial con un tiempo medio de servicio de $T_{active} = 7200$ segundos. Hemos elegido este tiempo medio de servicio ya que las películas suelen durar entre una y tres horas, es decir, entre 3600 y 10800 segundos. El número total de vídeos que ofrece el proveedor del sistema es $M = 100$. La estrategia de distribución seleccionada consiste en replicar en todos los servidores los vídeos más populares y distribuir de forma uniforme los restantes vídeos entre todos los servidores. Es decir, elegimos la distribución cíclica. Como mencionamos en la Sección 1.3.2 este esquema de distribución de los vídeos es el escenario más pesimista desde el punto de vista de la compartición de la carga y nos ayuda a simplificar nuestro análisis y a comprender el impacto de la replicación parcial y el grado de popularidad en el rendimiento del sistema, así como a identificar donde se encuentran los cuellos de botella.

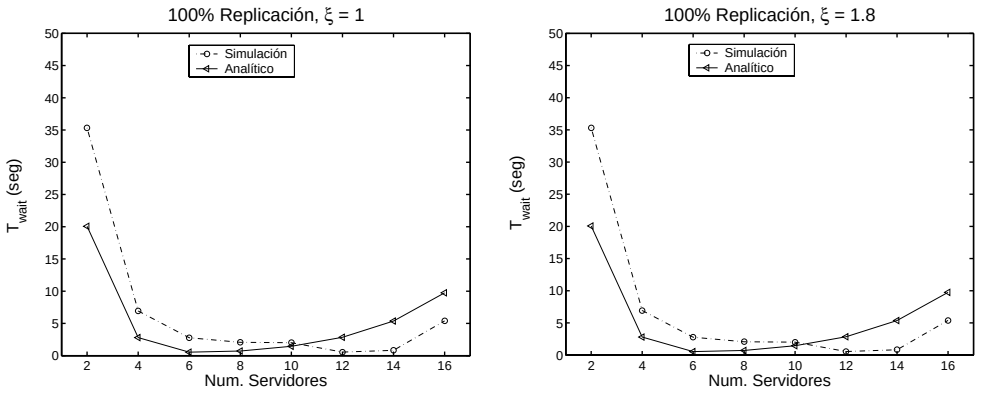
Para cada experimento simulado permitimos que se realizaran al menos 5000 servicios en cada servidor. Los primeros 1000 servicios se descartaron para así alcanzar un estado estadístico estacionario.

Para examinar la influencia del porcentaje de replicación realizamos un primer conjunto de experimentos donde fijamos $\xi = 1$ (esto corresponde a la llamada distribución pura de Zipf) y variamos el porcentaje de replicación entre el 100 %, el 75 % y el 50 %.

En el segundo conjunto de experimentos fijamos $\xi = 1{,}8$ (esto nos da una distribución de probabilidad donde unos pocos vídeos más populares son muy demandados). De nuevo variamos el porcentaje de replicación entre el 100 %, el 75 % y el 50 %. El objetivo de este nuevo conjunto de experimentos es medir el impacto del grado de popularidad en el rendimiento del sistema.

2.4.2. La precisión del modelo analítico

Las Figuras 2.3, 2.4 y 2.5 representan el rendimiento de nuestro algoritmo PRLS obtenido mediante simulaciones (las marcas circulares). Recordemos que nuestra medida de rendimiento es el tiempo de espera del cliente en el sistema. El eje X representa el número de servidores en el sistema, N_{serv}, y el eje Y representa el tiempo medio de espera en segundos. Además hemos representado los resultados obtenidos con el modelo analítico (las marcas triangulares).

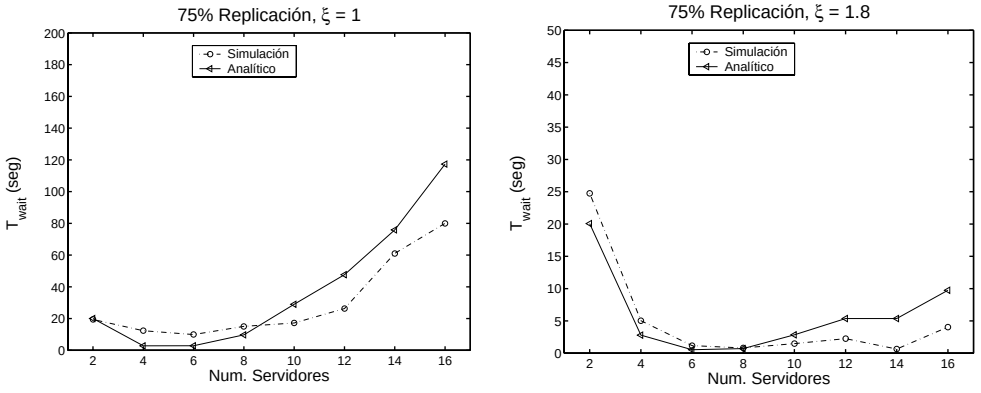


(a) (b)

Figura 2.3 Resultados del rendimiento del algoritmo PRLS y el modelo analítico para el 100 % de replicación: (a) cuando $\xi = 1$; (b) cuando $\xi = 1{,}8$. El eje X representa el número de servidores, y el eje Y son los tiempos de espera en segundos.

De las Figuras 2.3, 2.4 y 2.5 podemos observar que el modelo analítico representa de forma bastante precisa el rendimiento del algoritmo PRLS, en cuanto a tiempo de espera. Podemos apreciar que para menos de 6 servidores en el sistema los tiempos de espera del modelo analítico son ligeramente más pequeños que los simulados. Pensamos que esto es porque nuestro modelo sobreestima el número de streams disponibles cuando son calculados mediante la ecuación 2.10. Para un número pequeño de servidores, el promedio de streams que están disponibles de forma remota (n, ecuación 2.13) viene determinado por n_{idle}. Nuestra estimación de n_{idle} supone que la capacidad ocupada en un servidor debido a los servicios remotos, puede ser calculada como un promedio de todas las peticiones remotas. Sin embargo en el sistema real sólo una parte de las peticiones remotas pueden ser atendidas por un servidor. Esto significa una sobreestimación de n, lo que produce mayores valores de v (el promedio de streams disponibles para compartir la carga) en el modelo que en el sistema real. Por tanto en nuestro modelo la sobreestimación del número de streams disponibles en un servidor para compartir la carga provoca tiempos de espera más pequeños.

Por otro lado, para más de 10 servidores los tiempos de espera del modelo analítico son ligeramente mayores que los simulados. La razón es que el modelo analítico asume un peor escenario cuando calcula el número de servicios remotos obligatorios (los debidos a las peticiones a los vídeos no locales). Como explicamos en la Sección 2.3 el modelo asume que todos los servidores solicitarán simultáneamente peticiones de servicios remotos al servidor i. Esta suposición, que usamos cuando calculamos las ecuaciones 2.11 y 2.12, provocarán que la estimación analítica de los streams disponibles en el enlace y en el conmutador sea más pequeña que el número de streams disponibles en el sistema real, especialmente cuando el número de servidores es elevado. Sin embargo, en el sistema real todas las peticiones a vídeos no locales no se asignarán a un único servidor. Lo que se hará es distribuir las peticiones de vídeos no locales entre todos los servidores ya que los vídeos no locales se han distribuido



(a) (b)

Figura 2.4 Resultados del rendimiento del algoritmo PRLS y el modelo analítico para el 75 % de replicación: (a) cuando $\xi = 1$; (b) cuando $\xi = 1{,}8$. El eje X representa el número de servidores, y el eje Y son los tiempos de espera en segundos.

de forma uniforme en el sistema. Por tanto, en nuestro modelo la subestimación del número de streams disponibles en la red provoca que los tiempos de espera sean mayores.

Como alternativa, hemos realizado el mismo conjunto de experimentos para una distribución uniforme en el intervalo $(3600, 10800)$ segundos del tiempo de servicio. En estos experimentos obtuvimos resultados similares; lo único destacable es que los tiempos de espera obtenidos con 2 servidores en el sistema eran ligeramente más pequeños que los obtenidos en el caso de una distribución exponencial (los presentados en las Figuras 2.3, 2.4 y 2.5).

2.4.3. Efecto de la replicación y de la popularidad en el tiempo de espera

Ahora vamos a estudiar la influencia del porcentaje de replicación y del grado de popularidad en los tiempos de espera de los clientes.

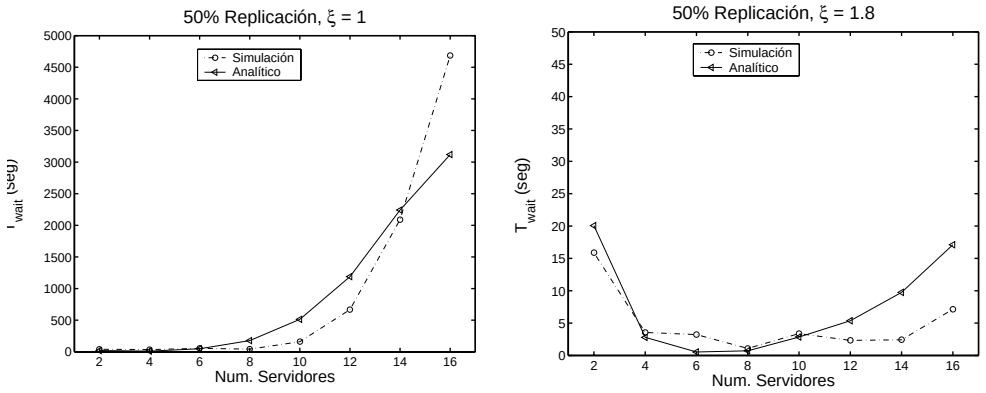
Como ya explicamos en el primer conjunto de experimentos fijamos $\xi = 1$, en cuyo caso aproximadamente la mitad de las peticiones se dirigen a los ocho vídeos más populares en cada servidor. En el segundo conjunto de experimentos, fijamos el grado de popularidad a $\xi = 1{,}8$, en cuyo caso aproximadamente el 80 % de las peticiones que se realizan en cada servidor son para los cuatro vídeos más populares.

En sendos conjuntos de experimentos se observa que los tiempos de espera que medimos aumentan cuando el porcentaje de replicación disminuye, especialmente cuando en el sistema $\xi = 1$. Esto es debido al aumento de servicios remotos obligatorios ya que menor replicación significa que un gran número de peticiones se harán a los vídeos no locales, especialmente cuando $\xi = 1$. Por ejemplo, el tiempo de espera con 10 servidores en el sistema aumenta cuando fijamos el grado de popularidad, y cuando el porcentaje de replicación

varía del 100 %, al 75 % y al 50 %. Esto se puede observar en las Figuras 2.3(a), 2.4(a), 2.5(a) (cuando $\xi = 1$). Sin embargo, en las Figuras 2.3(b), 2.4(b), 2.5(b) (cuando $\xi = 1{,}8$) el crecimiento del tiempo es muy pequeño. Vamos a ver esta cuestión con más detalle.

Cuando examinamos las gráficas de los tiempos de espera para $\xi = 1$ y distintos porcentajes de replicación (Figura 2.4(a) que representa los tiempos de espera para el 75 % de replicación y Figura 2.5(a) que representa los tiempos de espera para el 50 % de replicación) vemos que los tiempos de espera se mantienen pequeños de 4 a 6 servidores pero siguen un comportamiento exponencial cuando el número de servidores aumenta a partir de 8 servidores. Esto es debido al hecho de que cuando el número de servidores aumenta el número de servicios remotos obligatorios y en consecuencia la demanda de la red aumenta. Sin embargo, para $\xi = 1{,}8$ (Figura 2.4(b) que representa los tiempos para el 75 % de replicación y Figura 2.5(b) para el 50 % replicación), las gráficas de los tiempos de espera son similares. En este caso, los tiempos son muy pequeños de 4 a 6 servidores (de hecho los tiempos son similares a los obtenidos con el 100 % de replicación como vemos en la Figura 2.3(b)) y sólo a partir de 8 servidores los tiempos aumentan ligeramente cuando el número de servidores aumenta y el porcentaje de replicación disminuye. Obviamente la demanda de servicios remotos en este caso es baja. En otras palabras, *el efecto del porcentaje de replicación es más importante cuando el grado de popularidad es pequeño.*

Obsérvese la similitud de las Figuras 2.3(a) y 2.3(b). Ambas representan los tiempos de espera para el 100 % de replicación pero para distintos ξ. Podemos deducir fácilmente que para el 100 % de replicación el grado de popularidad no tiene influencia en los tiempos de espera ya que todos los servicios remotos se deben a la sobrecarga de los servidores en ambos casos.



(a) (b)

Figura 2.5 Resultados del rendimiento del algoritmo PRLS y el modelo analítico para el 50 % de replicación: (a) cuando $\xi = 1$; (b) cuando $\xi = 1{,}8$. El eje X representa el número de servidores, y el eje Y son los tiempos de espera en segundos.

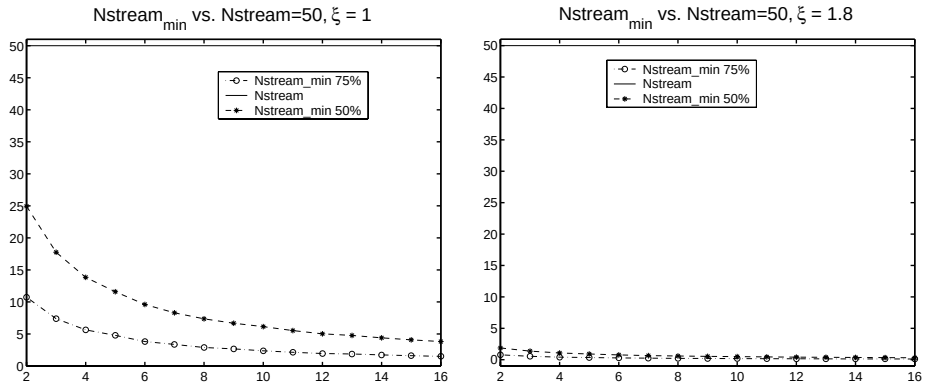
Sin embargo, cuando comparamos la Figura 2.4(a)) y la Figura 2.4(b)) que representan los tiempos de espera para el 75 % de replicación pero con distintos ξ, observamos que el grado de popularidad tiene ahora un importante impacto

como ya mencionamos antes. Cuando el grado de popularidad aumenta, los tiempos de espera se mantienen con valores pequeños incluso si el número de servidores aumenta. Sin embargo, cuando $\xi = 1$, los tiempos de espera aumentan con el número de servidores a causa de la contienda en la red provocada por el incremento de los servicios remotos. Un argumento similar se puede hacer para las Figuras 2.4(b) y 2.5(b), que representan los tiempos de espera para el 50 % de replicación pero con distintos ξ.

En la Figura 2.4(a) observamos que para un número de servidores entre 8 y 14 el tiempo de espera aumenta en un factor de 2. Sin embargo, en la Figura 2.4(b) para un número de servidores entre 8 y 14 el tiempo de espera aumenta en un factor mayor que 10. Esto es debido al aumento del número de servicios remotos obligatorios. De hecho, nuestro modelo predice que hay un punto en el cual el número de peticiones que se tienen que servir remotamente (m) es mayor que el número de streams disponibles en la red (n). Este punto (que es el cuello de botella) se alcanza cuando el número de servidores es más grande que $N_{server} = 12$ para el 75 % de replicación (Figura 2.4(a)), y más grande que $N_{server} = 6$ para el 50 % de replicación (Figura 2.4(b)). Este punto no se alcanza para $\xi = 1{,}8$. Una conclusión de este análisis es que cuando *el grado de popularidad es elevado, el porcentaje de replicación tiene un impacto muy pequeño en el rendimiento del sistema.*

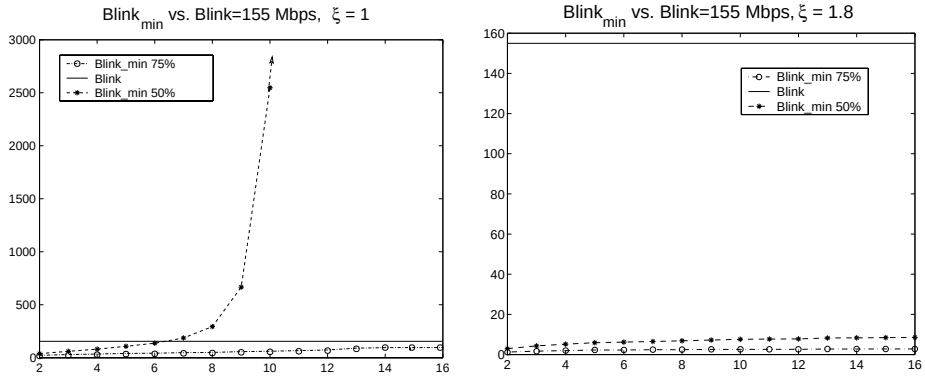
El punto en el que los servidores o la red se convierten en el cuello de botella se puede estimar con nuestro modelo. Como ya indicamos en la Sección 2.3, las ecuaciones 2.18, 2.20 y 2.22 nos permiten calcular los valores de N^{min}_{stream}, B^{min}_{link} y B^{min}_{switch} para evitar que los servidores o la red se conviertan en el cuello de botella. Por ejemplo, en la Figura 2.6(a) comparamos los valores de N^{min}_{stream} y de la capacidad actual de stream de un servidor, $N_{stream} = 50$, cuando $\xi = 1$ y el porcentaje de replicación es el 75 % y el 50 % respectivamente. La Figura 2.6(b) compara N^{min}_{stream} frente a $N_{stream} = 50$, cuando $\xi = 1{,}8$ y el porcentaje de replicación es 75 % y 50 %. Estas figuras nos permiten ilustrar una idea intuitiva: cuando el número de servidores aumenta la capacidad disponible de streams debida al número de servidores en el sistema aumenta y como las peticiones de servicios remotos obligatorios se repartirán de forma uniforme entre esta capacidad disponible, entonces el N^{min}_{stream} requerido en cada servidor disminuye para asegurar la realización de estos servicios remotos obligatorios. Claramente la cota de $N_{stream} = 50$ no se alcanza en ningún caso; en otras palabras la capacidad de los servidores no es el cuello de botella en el sistema.

Ahora estudiamos la red. En la Figura 2.7(a) comparamos los valores de B^{min}_{link} y el ancho de banda actual del enlace en nuestro sistema, $B_{link} = 150Mbps$, cuando $\xi = 1$ y el porcentaje de replicación es del 75 % y del 50 %. En la Figura 2.8(a) representamos los valores de B^{min}_{switch} junto con el ancho de banda del conmutador de nuestro sistema $B_{switch} = 1000Mbps$ cuando $\xi = 1$ y el porcentaje de replicación es del 75 % y del 50 %. Podemos ver que cuando el número de servidores aumenta, aumentan la capacidad requerida de B^{min}_{link} y B^{min}_{switch}. En el caso del 75 %, la cota del enlace de 155 Mbps no se alcanza. Sin embargo, podemos observar en la Figura 2.8(a) que la cota del conmutador se alcanza y se supera de forma notable cuando el número de servidores es mayor



(a) (b)

Figura 2.6 Comparación de N_{stream}^{min} frente a $N_{stream} = 50$ (línea sólida) para el 75 % y el 50 % de replicación: (a) cuando $\xi = 1$; (b) cuando $\xi = 1{,}8$. El eje X representa el número de servidores, y el eje Y está en Mbps.

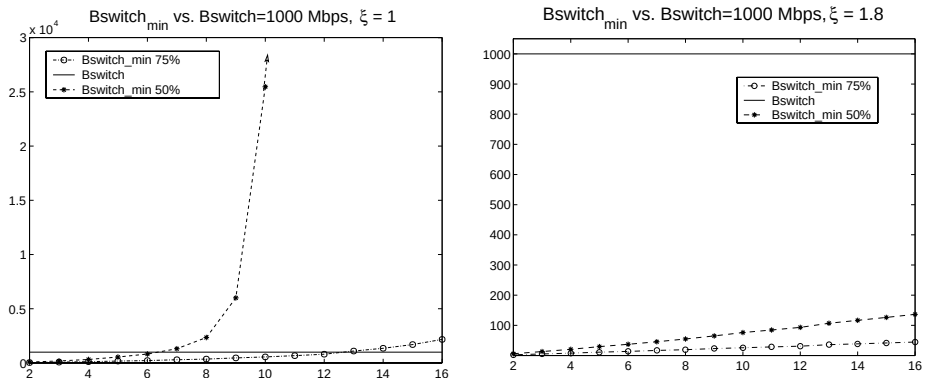


(a) (b)

Figura 2.7 Comparación de B_{link}^{min} frente a $B_{link} = 155\ Mbps$ (línea sólida) para el 75 % y el 50 % de replicación: (a) cuando $\xi = 1$; (b) cuando $\xi = 1{,}8$. El eje X representa el número de servidores, y el eje Y está en Mbps.

de 12. En otras palabras, como ya hemos deducido de las gráficas del tiempo de espera, con más de 12 servidores en el sistema, la red se convierte en el cuello de botella, en concreto el ancho de banda del conmutador. En el caso del 50 % de replicación las cotas del enlace y del conmutador se alcanzan cuando el número de servidores es mayor de 6. Además de esto, en esta situación las figuras muestran que los anchos de banda requeridos para B_{link}^{min} y B_{switch}^{min} tienen un comportamiento asintótico cuando el número de servidores es mayor de 10. Señalar que estas asíntotas provienen de la segunda parte de las ecuaciones 2.20 y 2.22 y representan el punto donde el número de peticiones de servicios remotos obligatorios explota.

En contraste, las Figuras 2.7(b) y 2.8(b) representan B_{link}^{min} y el valor actual de B_{link} y B_{switch}^{min} y el valor actual de B_{switch} respectivamente, para el 75 % y el 50 % de replicación y $\xi = 1{,}8$. En estos casos las cotas para el ancho de banda del enlace y el ancho de banda del conmutador no se alcanzan para ningún



(a) (b)

Figura 2.8 Comparación de B_{switch}^{min} frente a $B_{switch} = 1$ *Gbps* (línea sólida) para el 75 % y el 50 % de replicación: (a) cuando $\xi = 1$; (b) cuando $\xi = 1{,}8$. El eje X representa el número de servidores, y el eje Y está en Mbps.

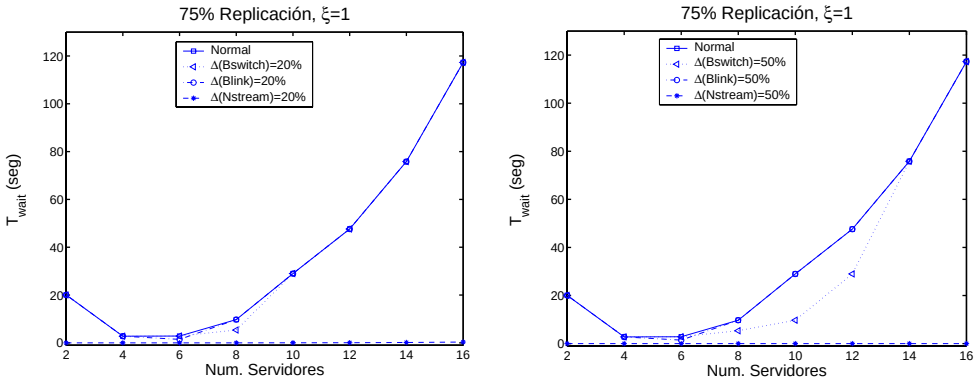
número de servidores. Por lo tanto, como hemos deducido anteriormente, la red no es el cuello de botella cuando el grado de popularidad es elevado. Resumiendo los resultados para $\xi = 1$ y $\xi = 1{,}8$ podemos decir que *la red es el cuello de botella para un número pequeño de servidores cuando el grado de popularidad es pequeño y el porcentaje de replicación disminuye.*

Otra cuestión interesante a explorar es analizar, fijado un porcentaje de replicación dado y un grado de popularidad, qué recurso del sistema influye en el tiempo de espera.

Por ejemplo, hemos usado el modelo analítico para medir los tiempos de espera cuando aumentamos uno de los recursos del sistema en un 20 % -ver Figura 2.9(a)-, o cuando aumentamos uno de los recursos en un 50 % -ver Figura 2.9(b). Es decir, aumentamos o el ancho de banda del enlace, B_{link}, o el ancho de banda del conmutador, B_{switch}, o el número de streams en cada servidor, N_{stream} en 20 % o en un 50 % cada uno.

En ambos casos estudiamos el sistema con un 75 % de replicación y con $\xi = 1$. En las figuras, `Normal` representa los tiempos de espera con los parámetros iniciales sin incremento, (los parámetros que hemos utilizado para verificar el modelo analítico y que aparecen en la Tabla 2.1). Las leyendas $\triangle(B_{switch})$ y $\triangle(B_{link})$ significan, respectivamente, que los parámetros de los anchos de banda del conmutador y del enlace se han incrementado, mientras que la leyenda $\triangle(N_{stream})$ significa que lo que se ha incrementado en los servidores es el número de canales para realizar los servicios (o lo que es lo mismo, el número de streams). Claramente podemos ver que el incremento en la capacidad de streams de los servidores ($\triangle(N_{stream})$) es el parámetro con más influencia en la reducción del tiempo de espera. De hecho, es suficiente con un incremento de cerca del 20 % (esto es, $N_{stream} = 60$) porque no se obtienen más beneficios aunque tengamos más capacidad de streaming cualquiera que sea el número de servidores. Sin embargo, los parámetros de la red tienen un comportamiento distinto. Por ejemplo, aumentando la capacidad del conmutador ($\triangle(B_{switch})$)

en un 20 % (esto es, $B_{switch} = 1{,}2Gbps$) se produce una pequeña reducción en el tiempo sólo con 6-8 servidores y no hay ningún tipo de repercusión para el resto de número de servidores. La reducción en los tiempos es más significativa cuando incrementamos la capacidad del conmutador en un 50 % ($B_{switch} = 1{,}5Gbps$). Sorprendentemente, aumentando la capacidad del enlace $\triangle(B_{link})$) en un 20 % ($B_{link} = 186Mbps$) o en un 50 % ($B_{link} = 232Mbps$) no tiene efecto en la reducción de los tiempos de espera. Lo que ocurre aquí es que incluso con los incrementos que hemos propuestos, es el conmutador el parámetro que limita la capacidad de streaming en la red (de hecho todavía es el cuello de botella para 12 y 14 servidores, ver la Figura 2.7(a) o la ecuación 2.20).



(a) (b)

Figura 2.9 Resultados del rendimiento cuando se incrementa sólo un parámetro del sistema (B_{link}, B_{switch}, N_{stream}): (a) 20 % de incremento; (b) 50 % de incremento. El eje X representa el número de servidores, y el eje Y son los tiempos de espera en segundos.

Otra cuestión interesante es encontrar el número de servidores para el que el sistema logra un tiempo mínimo de espera (ver de nuevo las Figuras 2.3, 2.4, 2.5). Para el 100 % de replicación y cualquier ξ, el tiempo mínimo se obtiene con 6-8 servidores. Sin embargo, para $\xi = 1$, cuando el porcentaje de replicación disminuye, el mínimo se logra cuando el número de servidores tiende a ser 4 (Figuras 2.4(a) y 2.5(a)). Por otro lado, para $\xi = 1{,}8$ cuando el porcentaje de replicación disminuye, el mínimo se alcanza cuando el número de servidores tiende a 8 (Figuras 2.4(b) y 2.5(b)). Otra aplicación del modelo analítico podría ser estimar el porcentaje de replicación apropiado (y el esquema de distribución de los vídeos) para obtener el tiempo mínimo de espera. Si conocemos los parámetros del sistema (N_{server}, N_{stream}, S_i, B_{link}, B_{switch}) y los parámetros del vídeo (p_j, L_j), podríamos formular un problema de optimización cuya función objetivo sea la minimización del tiempo medio de espera y cuyas restricciones sean las dadas por la ecuación 2.15 (la limitación del almacenamiento del disco) y las ecuaciones 2.18, 2.19, 2.21 (las capacidades mínimas de streaming en los servidores y la red).

2.5. Conclusiones del capítulo

En este capítulo hemos presentado un nuevo algoritmo unicast de compartición de la carga para sistemas distribuidos de VoD que tiene en cuenta que los vídeos se solicitan según sea su popularidad y que pueden estar parcialmente replicados en los servidores. Para capturar las características claves de este algoritmo hemos desarrollado un modelo analítico y hemos demostrado con simulaciones que nuestro modelo representa de forma bastante exacta el rendimiento del algoritmo, a pesar de los escenarios relativamente pesimistas que hemos considerado a la hora de distribuir los vídeos y de calcular los servicios remotos obligatorios.

Hemos analizado la influencia del porcentaje de replicación y del grado de popularidad en los tiempos de espera con el modelo analítico y con los resultados de los experimentos. La principal conclusión es que el tiempo de espera aumenta cuando el porcentaje de replicación disminuye, especialmente si el grado de popularidad es menor que 1. Cuando este grado está alrededor de 2, el crecimiento de los tiempos es muy pequeño. En otras palabras, el porcentaje de replicación tiene más efecto cuando el grado de popularidad es pequeño. Sin embargo, un grado de popularidad elevado significa que la mayoría de las peticiones son para los vídeos más populares y si son vídeos locales, esto podría estar enmascarado por una baja replicación de los otros vídeos. En consecuencia, en este caso, podemos ahorrar espacio de almacenamiento sin reducir el rendimiento con sólo replicar los vídeos más populares (que son pocos) en todos los servidores.

Otra importante conclusión es que el modelo analítico nos permite estimar para distintos porcentajes de replicación y grados de popularidad, algunas características básicas del comportamiento de nuestro algoritmo PRLS. Estas estimaciones nos podrían permitir seleccionar el tamaño de la red, el número óptimo de servidores para mantener un tiempo de espera pequeño y predecir cuándo la red se convierte en el cuello de botella.

3 Algoritmos Multicast: LTH y RLTH

3.1. Introducción

Actualmente las aplicaciones de VoD están limitadas por consideraciones económicas relacionadas, principalmente con el ancho de banda. Es bien sabido que en tales sistemas la capacidad de la red y del servidor son los recursos más caros. Por tanto, el uso eficiente del ancho de banda de la red y del servidor juega un papel crucial. Precisamente, el punto de partida de este capítulo es la optimización de estos recursos de un servidor de VoD.

La forma más intuitiva de optimizar el uso del ancho de banda de un servidor es atender todas las peticiones a un vídeo con un único stream, es decir, usar una estrategia multicast [54]. Entre las distintas estrategias multicast, *broadcast*, *batching*, y *patching* son las más usadas.

En una estrategia broadcast, el vídeo es emitido a todos los usuarios con un canal dedicado exclusivamente a la emisión de ese vídeo y con una planificación predefinida. Esta estrategia es muy adecuada para el caso de vídeos muy populares, ya que puede servir a un número ilimitado de peticiones a un vídeo muy popular, usando una cantidad constante de ancho de banda. Sin embargo, el ancho de banda es malgastado si la popularidad del vídeo es media o baja. Además, los clientes tienen que esperar hasta el momento en que el vídeo está planificado para poder recibir el servicio. Para reducir este retraso, se han propuesto algunas estrategias que mejoran la situación como *pyramid* [55], *permutation-based pyramid* [56] y *skyscraper* [57]. La idea innovadora en estas estrategias es que los datos de cada objeto se dividen en fragmentos, que son emitidos en intervalos predefinidos en diferentes canales. Los usuarios deben de ser capaces de recibir de forma simultánea dos o más canales, y de almacenar un fragmento recibido antes de que sea necesaria su reproducción (en otras palabras, el cliente necesita un *set-top box* o STB). En resumen, con estas estrategias se consigue una reducción del ancho de banda sólo cuando la tasa de las peticiones es elevada (es decir, se puede ahorrar ancho de banda con estas estrategias siempre y cuando los vídeos sean muy populares). Además estas estrategias no pueden dar servicio inmediato. Otro inconveniente es que

las operaciones en estos sistemas son bastantes complicadas para el usuario. Por ejemplo, el cliente es responsable de cambiar a los canales apropiados y en los instantes precisos, para poder ver el vídeo.

La idea básica del *batching* [26], [27], [28] es poner en cola las peticiones al mismo vídeo durante un cierto tiempo (que va a corresponder con el intervalo batching o de agrupamiento) y se sirven utilizando el mismo stream. Por tanto, se reduce el ancho de banda (más mientras mayor sea el intervalo de agrupamiento) pero su principal desventaja es que el retraso del comienzo del servicio puede ser muy grande aunque haya suficiente ancho de banda, y esto puede incrementar el grado de insatisfacción del usuario si el intervalo de espera es demasiado grande. De entre las políticas de planificación de tipo *batching* [26], [58], [27], el algoritmo MFQL o *Maximum Factored Queue Length* [28] planifica los vídeos en función de un factor que depende de la longitud de la cola. El vídeo cuyo factor sea máximo es el que se planifica en primer lugar. Este parámetro se obtiene aplicando un peso a la longitud de la cola de cada vídeo, peso que disminuye conforme la popularidad del vídeo aumenta. Los autores demostraron que esta política da un mayor rendimiento que otras políticas de *batching* [26]. Además, consigue excelentes resultados en términos de tiempos de espera, abandono del sistema y de imparcialidad en la planificación de los vídeos.

En una estrategia *patching* [29], la primera petición a un vídeo inicia un canal regular que transmite un stream completo. Un stream completo es el flujo de información que transmite el vídeo íntegramente. Una petición posterior al vídeo se une al canal regular para recibir y almacenar el vídeo que se está transmitiendo, mientras se inicia un canal patching que transmite un stream parcial con sólo los primeros minutos o segundos que se ha perdido del vídeo. Para ello, se necesita que el usuario tenga un STB donde se va almacenando el stream completo mientras el usuario visualiza los datos recibidos del stream parcial.

La principal ventaja de una estrategia *patching* es que no introduce retraso. En [59] se propone una estrategia interesante de tipo *patching* denominada *Threshold-based multicast* o Multicast con umbral, que introduce un **umbral** para controlar la frecuencia a la que se inician los canales regulares con un stream completo. En una estrategia *patching*, la duración de un stream parcial aumenta conforme va aumentando el tiempo de emisión del stream completo al que está asociado. Los autores del algoritmo Multicast con umbral demostraron que empezar la transmisión de un nuevo canal regular cuando se alcanza un determinado umbral es más eficiente que continuar lanzando streams parciales del último canal regular. Los autores demostraron también que la estrategia Multicast con umbral optimiza el ancho de banda usado y que el tiempo de espera disminuye.

Algunos autores han sugerido distintas combinaciones de estrategias para conseguir un balanceo entre el coste y el rendimiento. En [30] se combinan las estrategias de servicio unicast y broadcast, mientras que [31] propone una combinación de servicios unicast, multicast y broadcast. Sin embargo, recien-

temente, el retraso en la hora de comienzo y los requisitos de ancho de banda para broadcast y *patching* ya han sido mejorados. Algunas soluciones incluyen dynamic skyscraper [60], piggybacking [61], la solución stream merging- HMSM [62] y una estrategia *patching* de dos-niveles [63] que utiliza dos tipos de umbrales para controlar los streams parciales. Entre estas soluciones, en términos de ancho de banda requerido, la estrategia *patching* con dos niveles rinde mejor que la estrategia piggybacking, y también es mejor que la estrategia dynamic skyscraper, aun sobre un amplio rango de tasa de peticiones, y se acerca al rendimiento de la estrategia HMSM. Además, las estrategias *patching* son más fáciles de implementar comparadas con el sistema de transmisión de la estrategia skyscraper y HMSM. Es por estas razones que nos centramos en la estrategia *patching*.

En este capítulo presentamos un modelo analítico que captura las características principales de una estrategia *patching* con umbral. Nosotros nos diferenciamos de los trabajos anteriores en que nos centramos en el cálculo y optimización del tiempo de espera de los usuarios, mientras que todos los trabajos anteriores se han centrado solamente en el cálculo y optimización del ancho de banda. Creemos que el tiempo de espera es una métrica más exacta que el ancho de banda usado. Precisamente, el modelo analítico que proponemos en este capítulo nos permitirá calcular el tiempo de espera en el sistema y nos ilustrará en cómo usar este modelo para controlar y optimizar esta métrica del rendimiento. Otra diferencia es que tenemos en cuenta el hecho de que hay un número finito de canales en el servidor, lo que pensamos que es una aproximación más realista. Demostraremos que nuestro modelo tiene dos funciones:

1. Puede estimar de forma precisa el rendimiento de un servidor cualesquiera que sean sus características (incluyendo el comportamiento del usuario) en cuestión de segundos (comparado con las simulaciones que podrían tardar horas).

2. Puede dar unas guías para diseñar el sistema a fin de asegurar que el rendimiento no se degrade por debajo de un valor especificado cuando algunos de los parámetros cambien. Por ejemplo, la tasa de peticiones puede cambiar a lo largo del tiempo, y el servidor podría ser diseñado para que sea capaz de atender las peticiones en cualquier caso, garantizando que el tiempo máximo de espera sea menor que un tiempo de espera especificado.

En el Capítulo 2 hemos analizado el rendimiento del sistema de VoD (descrito en el Capítulo 1), considerando que se dedica un canal para realizar los servicios de cada cliente, es decir, los servicios se realizan de forma unicast. En ese capítulo, vimos que un bajo porcentaje de replicación en el sistema produce un elevado número de servicios remotos. Esto incrementa el uso de la red, la cual se convierte en el cuello de botella. Por otro lado, como los servidores en nuestro sistema son pequeños, sólo pueden soportar un número pequeño de streams a la vez, con lo que es importante una utilización eficiente de la capacidad de los servidores.

En este capítulo abordamos la optimización del uso de la red y del ancho de banda de un servidor utilizando una aproximación multicast para gestionar las peticiones.

De entre las distintas aproximaciones multicast [54], es bastante frecuente utilizar las técnicas *batching* y *patching*.

Aunque tanto las estrategias MFQL como Multicast con umbral fueron desarrolladas originalmente para un sistema con un único servidor, en este capítulo proponemos dos algoritmos híbridos (inspirados por estos algoritmos) diseñados para tener en cuenta la naturaleza de nuestro sistema distribuido de VoD. En nuestros algoritmos, la estrategia MFQL (de tipo *batching*) se utiliza para planificar las colas de los vídeos, es decir, para decidir qué cola de vídeo será servida iniciando una sesión multicast (que equivale a lanzar un stream completo) cuando haya recursos disponibles. La estrategia Multicast con umbral, se aplica para controlar la transmisión tanto de los streams completos, como de los streams parciales que se lanzan dentro del umbral del vídeo durante la emisión de una sesión multicast.

Resumiendo, nuestros algoritmos han sido diseñados específicamente para planificar de forma eficiente todas las peticiones (locales como remotas) del sistema distribuido de VoD propuesto, es decir, optimizando el uso de los recursos de los servidores y la red, y manteniendo un tiempo de espera aceptable.

3.2. Los algoritmos MFQL y Multicast con umbral

En esta sección explicamos con más detalle las características de los algoritmos MFQL y Multicast con umbral.

De entre las políticas de planificación del tipo *batching* [26] [58] [27] existen dos implementaciones comunes. Una es la política *First Come First Served (FCFS)*, (o lo que es lo mismo, "el primero que llega, el primero que se sirve"), que planifica el vídeo que tenga el cliente con mayor tiempo de espera en cola. Otra implementación es la estrategia *Maximum Queue Lenght (MQL)* (o "longitud máxima de la cola"), que selecciona el vídeo que tenga el mayor número de peticiones esperando en cola. Sin embargo, estas estrategias no conducen a unos resultados satisfactorios. MQL tiende a ser demasiado injusto al planificar casi siempre los vídeos más populares, ya que sólo considera la longitud de la cola. Por otro lado, FCFS se comporta justo al revés, ya que ignora completamente la longitud de la cola y se centra sólo en el tiempo de llegada para reducir el abandono de los usuarios.

Para compensar los algoritmos FCFS y MQL se propuso una estrategia llamada *Maximum Factored Queue Length* (MFQL) [28]. Este esquema planifica los vídeos en función de un factor que depende de la longitud de la cola y de la popularidad de los vídeos. Este parámetro se obtiene aplicando un peso a la longitud de la cola de cada vídeo, de forma que el factor disminuye cuando la popularidad del vídeo aumenta. En [28] se demuestra que los factores óptimos

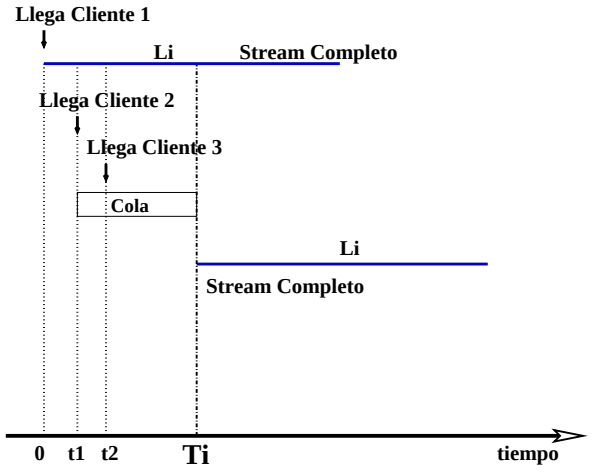


Figura 3.1 Planificación de servicios con el algoritmo MFQL.

son inversamente proporcionales a las raíces cuadradas de las probabilidades de los distintos vídeos, y vienen dada por la expresión

$$\frac{q_i}{\sqrt{p_i}} \tag{3.1}$$

$\forall i \in \{1, \ldots, M\}$, donde q_i es la longitud de la cola del vídeo i y p_i es la probabilidad de solicitar dicho vídeo.

En la Figura 3.1 se muestra un esquema del algoritmo MFQL. Supongamos que, en el instante 0, se inicia un stream completo para el vídeo i (la duración del vídeo i es L_i). A partir del instante 0, todas las peticiones que llegan al vídeo i se ponen en cola (como les ocurre al Cliente 2 y Cliente 3). Supongamos que se queda disponible en el servidor capacidad para un nuevo stream, en el instante T_i. Entonces, el algoritmo MFQL planifica el vídeo que tenga el mayor factor (ver ecuación 3.1). Supongamos que el vídeo con mejor factor es el vídeo i. Entonces, las peticiones que están en la cola del vídeo i se sirven con un nuevo stream multicast completo en el instante T_i.

Los autores [28] demostraron que MFQL consigue excelentes resultados en términos de tiempos de espera y baja tasa de abandono de clientes en el sistema. Sin embargo, como MFQL es un algoritmo de tipo *batching*, los clientes se ven obligados a esperar hasta la siguiente planificación del vídeo. Por tanto, no hay una latencia de servicio máxima garantizada. Como nuestro objetivo es minimizar y acotar el tiempo de espera de los clientes en el sistema, buscamos otras estrategias multicast que no retrasen a los clientes haciendo un mejor uso de los anchos de banda del servidor y de la red.

Para ello, hemos de recurrir a estrategias de tipo *patching*. En concreto, consideramos la estrategia de tipo *patching* propuesta en [59] y llamada *Threshold-*

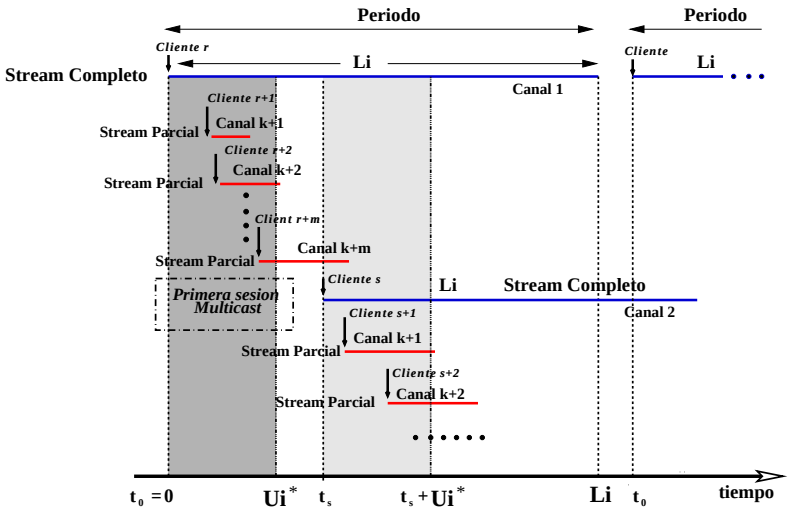


Figura 3.2 Multicast con umbral.

based multicast o Multicast con umbral. La idea clave del algoritmo Multicast con umbral es como sigue: cuando una petición a un vídeo (supongamos de nuevo que es el vídeo i) llega al sistema y es la primera, se asigna inmediatamente (en este estudio se supone un número infinito de canales en el servidor y por tanto, todas las peticiones se sirven inmediatamente) un canal regular que transmite un stream completo multicast de este vídeo. A cualquier petición posterior a este vídeo que llegue dentro de un intervalo de tiempo, (es decir, dentro de un umbral de tiempo) se le permite unirse a la sesión multicast iniciada para este vídeo. Además, se inicia un stream unicast parcial para enviar los primeros minutos del vídeo y así ponerse al corriente con el stream completo. Mientras se transmite el stream parcial, el STB del cliente recibe y almacena los datos recibidos del stream completo. Este algoritmo minimiza el ancho de banda usado para realizar los servicios de los vídeos. Pero una limitación importante en este estudio es (como ya se ha mencionado) que suponen un número infinito de canales en el servidor.

La Figura 3.2 muestra un ejemplo del algoritmo Multicast con umbral. Supongamos que el *Cliente* r solicita el vídeo i en el instante 0 y que es la primera petición que solicita el vídeo. Entonces se asigna un canal regular (el Canal 1 en la figura) para transmitir el stream completo, es decir, se inicia una sesión multicast de duración L_i. Por otro lado, U_i^* representa la duración de la ventana de tiempo, llamada umbral, que empieza a la vez que el stream completo del vídeo i. Este umbral controla la frecuencia a la que se transmite un nuevo stream completo del vídeo i. Supongamos que llega una petición del *Cliente* $r+1$ antes de U_i^*. Entonces el cliente se une inmediatamente a la sesión multicast (al Canal 1). Además, el servidor transmite un stream parcial con los primeros minutos del vídeo a través de un canal de patching (el Canal k+1). Cualquier otra petición al vídeo dentro del umbral U_i^* se procesará de la misma forma (como se puede ver en la Figura 3.2 las peticiones del *Cliente* $r+2$ al *Cliente* $r+m$ son servidos con los canales Canal k+2 al Canal

k+m respectivamente). Por otro lado, cualquier petición que llegue fuera del umbral U_i^*, será servida empezando un nuevo stream completo multicast en un nuevo canal regular (ya que suponen recursos infinitos). Tal es el caso del *Cliente s* que inicia un stream completo en el Canal 2 en el instante t_s. De nuevo se abre una ventana U_i^*, y cualquier petición que llegue dentro de ella, será servida con un stream parcial a través de un canal de patching.

En [59] se obtiene un umbral óptimo para cada vídeo, U_i, que minimiza el ancho de banda del servidor. Modelando el sistema como un proceso de renovación, se calcula el umbral óptimo en función del tamaño del buffer del cliente, la tasa de llegada de peticiones y la popularidad de los vídeos.

Los autores demuestran que la media de ancho de banda requerido por el servidor para servir al vídeo i siguiendo una estrategia tipo *batching* (como MFQL) crece de forma lineal en función de la tasa de llegadas (ya que cada petición se sirve con un stream completo) y tiene como expresión

$$B_i = L_i \cdot \lambda_i \cdot B_{stream} \tag{3.2}$$

donde λ_i es la tasa de peticiones al vídeo i ($\lambda_i = \lambda \cdot p_i$, siendo λ el número de peticiones por unidad de tiempo y p_i la probabilidad de solicitar el vídeo i) y B_{stream} es el ancho de banda requerido para servir un vídeo por canal (ver Tabla 3.1).

Sin embargo, con la estrategia Multicast con umbral, el ancho de banda medio requerido para servir un vídeo es una función de la raíz cuadrada de la tasa de peticiones y la duración del vídeo [59], y tiene como expresión

$$B_i = \left(\sqrt{2 \cdot L_i \cdot \lambda_i + 1} - 1\right) \cdot B_{stream}$$

Esta reducción en el ancho de banda requerido tiene un importante impacto en el tiempo de espera de los usuarios. De hecho, veremos en la Sección 3.6 que nuestros algoritmos reducen significativamente los tiempos de espera.

Como en [59] suponen un número infinito de canales en el servidor, no se formarán nunca colas. Por el contrario, en nuestra investigación suponemos que hay un número finito de canales (N_{stream}) en el servidor, que es una aproximación más realista. Esta restricción impone un cambio en el comportamiento del algoritmo Multicast con umbral: eventualmente, cuando los canales del servidor estén ocupados, las peticiones que lleguen tendrán que ser puestas en cola y habrá que establecer un criterio de planificación de estas colas.

Otros trabajos como [64] y [65] proponen aproximaciones donde el uso del ancho de banda es una función logarítmica.

3.3. Nuestros algoritmos

En esta sección introducimos dos algoritmos de tipo *patching* con umbral, que proponemos para planificar tanto las peticiones locales como las remotas en nuestro sistema. Los hemos llamado *Local Threshold (Umbral Local, LTH)* y *Remote and Local Threshold (Umbral Remoto y Local, RLTH)*. Ambos algoritmos, LTH y RLTH, comparten las siguientes características:

- Cada servidor mantiene una cola por vídeo.
- Cuando en un servidor hay varias colas de vídeos y se libera un canal, se planifican las colas utilizando el algoritmo MFQL, que funciona de la siguiente forma: se calculan en el servidor los factores de las colas de los M vídeos ofertados dados por la ecuación 3.1. Una vez obtenidos los factores, éstos se organizan en orden decreciente y se planifica una stream multicast completo para la cola del vídeo con el máximo factor. Supongamos que el vídeo que se planifica es el vídeo i y que tiene una duración de L_i segundos.

3.3.1. Servicios locales

En nuestro sistema, un importante porcentaje de peticiones corresponde a los vídeos locales, especialmente de los vídeos que están replicados en todos los servidores, ya que éstos son los más populares. Por tanto, podemos reducir el uso del ancho de banda del servidor, y en consecuencia el tiempo de espera de estas peticiones locales, si utilizamos la estrategia Multicast con umbral. Recordemos que esta estrategia utiliza un umbral para controlar la frecuencia a la que se inician los canales regulares con un stream completo y que controla la emisión de los streams parciales.

Si el vídeo i que se ha planificado es local y hay suficiente ancho de banda en el servidor, todas las peticiones pendientes de ese vídeo que están en cola, se sirven con el mismo stream multicast completo a través de un canal regular.

Sea U_i el umbral que controla la transmisión de los streams completos del vídeo i. Todas las peticiones que se reciben del vídeo y que llegan dentro del umbral del vídeo U_i se sirven con streams parciales si existe un stream completo inicializado. En cualquier otro caso, las peticiones se ponen en cola hasta que se planifique el próximo stream multicast completo para el vídeo i. El periodo de tiempo que transcurre desde la última planificación de un stream multicast completo del vídeo i hasta el próximo, es lo que llamamos el intervalo de agrupamiento, T_i.

La Figura 3.3 muestra el esquema básico de nuestros algoritmos. En el ejemplo, en el instante 0 todas las peticiones pendientes en la cola del vídeo i se sirven con el mismo canal regular (Canal 1), el cual transmite un stream completo multicast. Una vez iniciada una sesión multicast, durante el umbral U_i se inician

streams parciales para atender las peticiones que llegan al vídeo i, como es el caso del *Cliente* $r+1$ al *Cliente* $r+m$ que son servidos con los canales de patching Canal k+1 al Canal k+m.

En nuestras propuestas, la primera petición al vídeo i que llega después del umbral podría iniciar una nueva sesión multicast si hubiese recursos disponibles en el servidor para realizar una sesión multicast (como ocurre para el *Cliente* s en la Figura 3.2). Por el contrario si no hay recursos disponibles, las peticiones al vídeo i que llegan después del umbral se pondrán en la cola hasta que se planifique la próxima sesión multicast del vídeo i. Esto es lo que ocurre en el ejemplo de la Figura 3.3, donde las peticiones de los clientes *Cliente* s, *Cliente* s+1, ... se ponen en cola hasta el instante T_i. Este instante de tiempo es el momento en el que se liberan recursos que se utilizan para iniciar una sesión multicast del vídeo i planificada por el algoritmo MFQL. Entonces, todas las peticiones pendientes al vídeo i se servirán con el mismo canal regular (Canal 2) que transmite un stream multicast completo. De nuevo, una ventana de duración U_i empieza y todas las peticiones que lleguen antes del umbral serán servidas a través de canales de patching que transmiten streams parciales. La selección del umbral es clave para mejorar el rendimiento del sistema. En la Sección 3.4 obtendremos un umbral que optimiza el uso de los recursos.

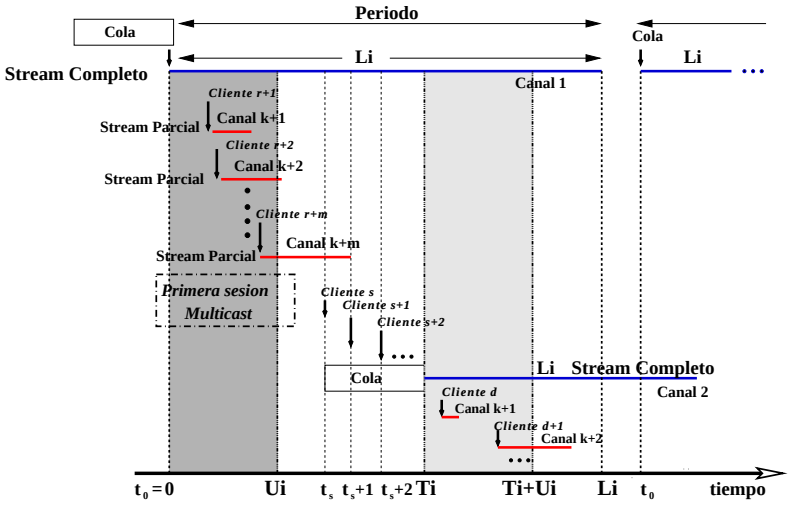


Figura 3.3 Nuestro algoritmo multicast LTH.

En este punto, nos encontramos con la siguiente cuestión: ¿qué pasa cuando el vídeo planificado por el algoritmo MFQL está almacenado en un servidor remoto?. Recordemos que en el caso de una petición remota *sólo un servidor* almacena el vídeo. En este caso LTH y RLTH siguen estrategias distintas como veremos a continuación.

3.3.2. Servicios remotos con el algoritmo LTH

La Figura 3.4 ilustra cómo procesa el algoritmo LTH las peticiones remotas. En este algoritmo se aplica una estrategia de planificación simplista. Supongamos

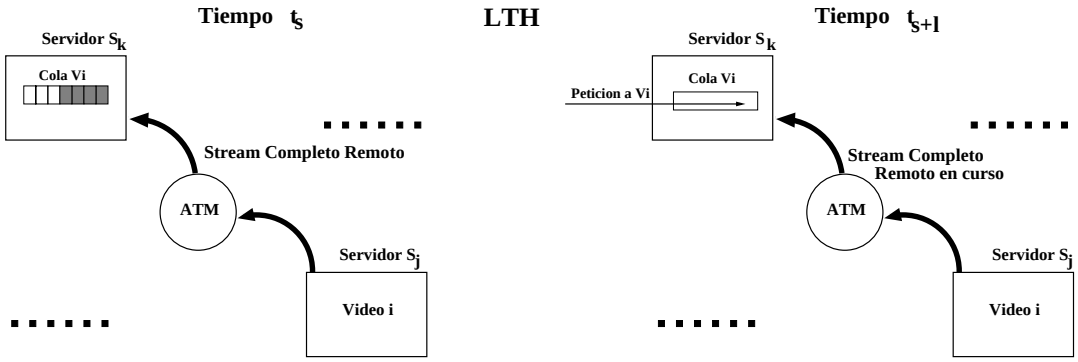


Figura 3.4: Algoritmo LTH: En el instante t_s una cola remota del vídeo i, se planifica en el servidor S_k y el servidor S_j la sirve con un stream completo remoto. Posteriormente, en el instante t_{s+l}, llega al servidor S_k una nueva petición al vídeo i que es puesta en cola.

que en el instante t_s se planifica en el servidor S_k la cola del vídeo remoto i. Este servidor dialoga con el único servidor que almacena el vídeo, supongamos que se trata del servidor remoto S_j. Si hay recursos en la red (es decir, hay ancho de banda disponible en los enlaces de entrada/salida, B_{link}, de los servidores S_k y S_j, y ancho de banda en el conmutador B_{switch}) así como ancho de banda local B_{stream} en el servidor S_j, entonces se sirven todas las peticiones que estén en la cola del vídeo i del servidor S_k con el mismo stream completo suministrado por el servidor S_j. Si más tarde llega una nueva petición remota al vídeo i en el servidor S_k en el instante t_{s+l}, ésta se pone en cola. La nueva cola del vídeo se planificará en el próximo intervalo de agrupamiento T_i. En otras palabras, a las peticiones remotas no se les aplica la estrategia *patching*.

Como podemos ver, en el algoritmo LTH la estrategia MFQL se usa para planificar las peticiones locales y remotas, mientras que la estrategia de Multicast con umbral se usa para servir solamente a las peticiones locales.

3.3.3. Servicios remotos con el algoritmo RLTH

Nuestro segundo algoritmo es un poco más agresivo que el anterior, y pretende resolver el siguiente problema: cuando el porcentaje de replicación disminuye, el tráfico a través de la red de interconexión aumenta debido al gran número de servicios remotos. De hecho, la red puede convertirse rápidamente en el cuello de botella del sistema. Para reducir este tráfico explotamos las ventajas del algoritmo Multicast con umbral para reducir ahora el uso del ancho de banda de la red de interconexión. Para ello, modificamos el algoritmo LTH dando lugar al algoritmo *Remote and Local Threshold (Umbral Remoto y Local)* (RLTH).

En la Figura 3.5 podemos ver como se procesan las peticiones remotas con este nuevo algoritmo. Cuando en el instante t_s se planifica la cola de peticiones remotas del vídeo i en el servidor S_k, éste se pone en contacto con el servidor que almacena el vídeo (que para facilitar la explicación vamos a suponer que

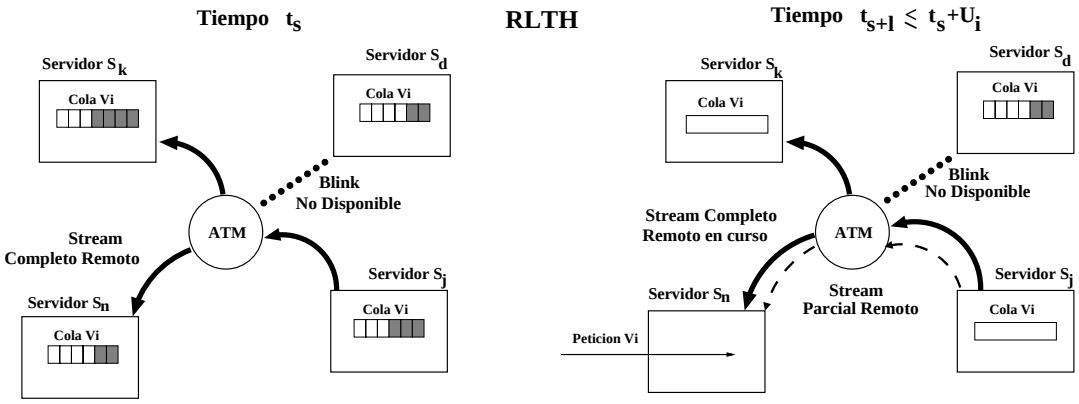


Figura 3.5: Algoritmo RLTH: En el instante t_s una cola remota del vídeo i, se planifica en el servidor S_k y el servidor S_j la sirve con un stream completo remoto. Posteriormente, en el instante t_{s+l}, $t_{s+l} \leq t_s + U_i$, llega al servidor S_n una nueva petición al vídeo i la cual es servida con un stream parcial por el servidor S_j.

se trata de nuevo del servidor S_j). Cuando el servicio remoto se acepta porque hay recursos disponibles en el servidor S_j y en la red, entonces el servidor S_k notifica a los demás servidores del sistema la siguiente información:

1. que se va a iniciar un stream remoto para el vídeo i,
2. la identificación del servidor que va a realizar el servicio (el servidor S_j) y,
3. el umbral del vídeo i, U_i.

Si los restantes servidores, incluido el servidor S_j, tienen peticiones pendientes en sus respectivas colas del vídeo i y hay recursos disponibles (en concreto, ancho de banda disponible en los enlaces de entrada/salida), estas peticiones se sirven también con el stream multicast completo que se va a inicializar. Es decir, todas las peticiones pendientes del vídeo i *en el sistema* se pueden servir con el mismo stream multicast. Los servidores que comparten el mismo stream multicast completo constituyen lo que llamamos un *grupo multicast*. En el caso representado en la Figura 3.5, las colas del vídeo i en los servidores S_k y S_n (y S_j) se sirven con el mismo stream multicast completo. Sin embargo, las peticiones en cola del servidor S_d, no se sirven porque no hay ancho de banda de entrada del enlace, B_{link}, en ese servidor.

Si más tarde, en el instante t_{s+l}, llega una nueva petición al vídeo i en cualquier servidor del grupo multicast (S_k, S_n o S_j) dentro del umbral (es decir, si $t_{s+l} - t_s \leq U_i$) (por ejemplo, en la Figura 3.5 al servidor S_n llega una petición remota del vídeo i la petición llega dentro del umbral), y hay ancho de banda disponible en la red y en el servidor S_j, entonces se sirve la petición con un stream parcial remoto del servidor S_j.

Resumiendo, la característica que distingue LTH y RLTH es la forma en la que se sirven las peticiones remotas. Eso da lugar a que con LTH sólo se optimice el ancho de banda local (el del servidor) gracias al uso del umbral, mientras que con RLTH, tanto el ancho de banda local como el de la red se optimizan, y como consecuencia, el tiempo de espera se reduce aun más como veremos en la Sección 3.6.

3.4. Umbral óptimo

Como mencionamos en la Sección 3.3, la selección del umbral es clave para optimizar los anchos de banda del servidor y la red.

Para obtener el umbral óptimo con nuestros algoritmos, nos centramos en el vídeo i y suponemos que las peticiones al vídeo se generan siguiendo una distribución exponencial de media $1/\lambda_i$, donde

$$\lambda_i = \lambda \cdot p_i$$

siendo p_i la probabilidad de solicitar el vídeo i y λ la tasa de peticiones al servidor. Para simplificar hemos supuesto que todos los servidores tienen la misma tasa de peticiones λ.

En nuestro análisis hemos considerado que los servidores del sistema tienen una capacidad limitada y que todos los vídeos almacenados en un servidor comparten la misma capacidad de $N_{streams}$ streams. Por lo tanto, algunas peticiones (a un mismo vídeo) pueden que sean agrupadas con peticiones anteriores a ellas. También tenemos que destacar que para atender la demanda que genera un vídeo en un instante en particular, puede usar más (o menos) ancho de banda del servidor que el ancho de banda requerido para servirlo. Pero midiendo estadísticamente el consumo de ancho de banda (durante toda la duración de la transmisión de un vídeo), éste consume, de media, el ancho de banda requerido para servirlo. Por tanto, es razonable utilizar el ancho de banda medio para servir el vídeo como parámetro de rendimiento para obtener nuestro umbral óptimo.

Por otro lado, hay que destacar que cuando un usuario solicita el vídeo i y llega t segundos después de que se haya iniciado un stream completo del vídeo i, el cliente recibe t segundos del vídeo en un stream parcial y los restantes $L_i - t$ segundos a través del stream completo. Por tanto, el cliente necesita un buffer para almacenar al menos $\min\{t, L_i - t\}$ minutos del vídeo. Suponemos que el tamaño del buffer en el STB del cliente es suficientemente grande y que no va a ser una restricción.

Obtenemos el umbral óptimo modelando el sistema como un proceso de renovación. Estamos interesados en el proceso $\{S(t) : t > 0\}$ donde $S(t)$ es el ancho de banda total usado desde el instante 0 al instante t para servir el vídeo i. En particular, nos interesa el ancho de banda medio utilizado y una forma de

obtenerlo es

$$B_i = lim_{t\to\infty} S(t)/t \tag{3.3}$$

Sea $\{T_j\}_{j=0}^{\infty}$ ($T_0 = 0$) los tiempos en los que el sistema planifica un stream completo para el vídeo i. Estos son los tiempos de batching o de agrupamiento en nuestro algoritmo, y además, son puntos de renovación en el sentido de que el comportamiento del sistema para $t \geq T_j$ no depende del comportamiento pasado.

Consideramos el proceso $\{S_j, N_j\}$ donde S_j es el ancho de banda usado durante el j-ésimo intervalo de renovación $[T_{j-1}, T_j)$ y N_j es el número total de clientes servidos en el mismo intervalo. Como es un proceso de renovación, podemos omitir el subíndice j (no depende del pasado), y por tanto, obtenemos una expresión equivalente del ancho de banda medio utilizado como

$$B_i = \lambda_i \cdot \frac{E[S]}{E[N]} \tag{3.4}$$

donde $E[S]$ es la media de ancho de banda usado durante un intervalo de renovación y $E[N]$ el número medio de clientes servidos durante el mismo intervalo.

A continuación vamos a calcular los parámetros $E[S]$ y $E[N]$.

$E[N]$ viene dado por la expresión

$$E[N] = T_{wait_i} \cdot \lambda_i + \lambda_i \cdot U_i$$

donde el término $T_{wait_i} \cdot \lambda_i$ corresponde con los clientes que llegaron antes de la planificación que se va a realizar del vídeo, y que serán servidos con el mismo stream completo. El término $\lambda_i \cdot U_i$ corresponde con los clientes que llegarán en el umbral U_i y que serán servidos con streams parciales.

A continuación calculamos $E[S]$. Sea K el número de llegadas que ocurren en un intervalo de longitud U_i. Estas llegadas siguen una distribución exponencial, por tanto,

$$P[K = k] = \frac{(\lambda_i \cdot U_i)^k \cdot e^{-\lambda_i \cdot U_i}}{k!} \tag{3.5}$$

Como consecuencia del hecho de que k llegadas en un intervalo de longitud U_i tienen la misma probabilidad cualquiera que sea el instante en que llegan dentro del intervalo, tenemos que

$$E[S|K=k] = \left(L_i + \frac{k \cdot U_i}{2}\right) \cdot B_{stream} \tag{3.6}$$

Por tanto, de las ecuaciones 3.5 y 3.6, se tiene que el ancho de banda medio utilizado del servidor, $E[S]$, viene dado por

$$E[S] = B_{stream} \cdot L_i + \frac{\lambda_i \cdot B_{stream} \cdot U_i^2}{2}$$

Una vez que hemos determinado $E[S]$ y $E[N]$ los sustituimos en la ecuación 3.4 y obtenemos el ancho de banda medio requerido para servir al vídeo i como

$$B_i = \frac{2 \cdot L_i + \lambda_i \cdot U_i^2}{2 \cdot (T_{wait_i} + U_i)}$$

El valor de U_i que minimiza esta expresión es

$$U_i = \sqrt{T_{wait_i}^2 + 2 \cdot L_i/\lambda_i} - T_{wait_i} \tag{3.7}$$

y el valor correspondiente del ancho de banda medio requerido B_i es

$$B_i = \lambda_i \cdot \left(\sqrt{T_{wait_i}^2 + 2 \cdot L_i/\lambda_i} - T_{wait_i}\right) \cdot B_{stream} \tag{3.8}$$

El umbral obtenido en la ecuación 3.7 es el valor que minimiza el uso del ancho de banda del servidor. Suponiendo que esta minimización lleva al tiempo mínimo de espera, podemos esperar entonces que T_{wait_i} tienda a 0. En este caso, la expresión del umbral óptimo se aproxima por,

$$U_i = \sqrt{2 \cdot L_i/\lambda_i} \tag{3.9}$$

Una vez obtenido el umbral para cada vídeo, el valor correspondiente del ancho de banda medio requerido para servir a un vídeo cuando T_{wait_i} tiende a 0, se calcula de la ecuación 3.8 como,

$$B_i = \left(\sqrt{2 \cdot L_i \cdot \lambda_i}\right) \cdot B_{stream} \tag{3.10}$$

De la ecuación 3.9, podemos observar que el umbral óptimo aumenta cuando la popularidad del vídeo disminuye. Otro punto importante que podemos observar de la ecuación 3.10, es que nuestros algoritmos requieren un ancho de banda que crece en función de la raíz cuadrada de la duración del vídeo (L_i) y la tasa de peticiones al vídeo i (λ_i). Comparado con el ancho de banda del servidor requerido por la estrategia básica de *batching* (ecuación 3.2)

$$B_i = L_i \cdot \lambda_i \cdot B_{stream}$$

podemos confirmar que nuestras técnicas reducen el ancho de banda requerido para realizar un servicio.

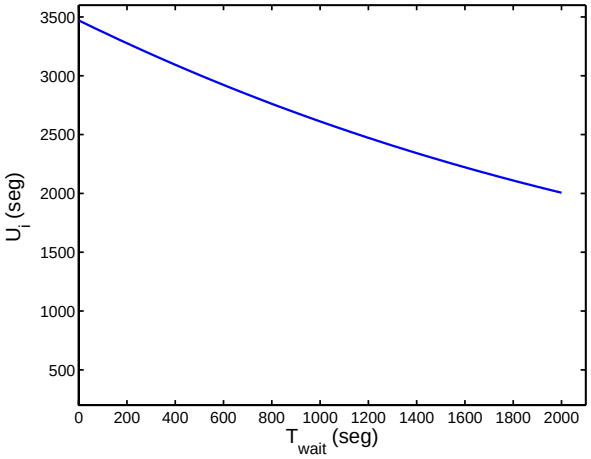


Figura 3.6 Sensibilidad del Umbral para el vídeo más popular cuando $\xi = 1$ y M = 100.

Una cuestión interesante es la sensibilidad que tiene la aproximación del umbral calculada según la ecuación 3.9, cuando el tiempo de espera para el video i, T_{wait_i}, aumenta. Para ilustrar esta cuestión representamos en la Figura 3.6 el umbral exacto calculado según la ecuación 3.7 para el vídeo más popular en el sistema, y cuando el tiempo de espera varía de 0 a 2000 segundos. Nos centramos en el intervalo de 0-300 segundos, porque consideramos que es un intervalo de espera aceptable para el cliente. En este caso, podemos ver que el rango de variación para el umbral U_i va de 3500 a 3300 segundos, lo que representa aproximadamente un 5 % de variación del valor calculado en la ecuación 3.9. Es decir, se trata de una modificación bastante pequeña de tal valor. Esto es debido a que el término T_{wait_i} es mucho más pequeño que el término L_i/λ_i en la ecuación 3.7, por lo que podemos ignorar sin ningún problema tal término cuando calculamos el umbral en nuestros algoritmos.

3.5. Modelo Analítico

En esta sección veremos que el modelo se puede usar como una herramienta de configuración para dimensionar tanto un servidor aislado, como el sistema. Al igual que en el caso unicast (el algoritmo PRLS cuyo modelo analítico estudiamos en el Capítulo 2), nuestro modelo del sistema implementando nuestro algoritmo de *patching* con umbral, RLTH, se basa en las dos siguientes observaciones: 1) la compartición de la carga no afecta significativamente al factor de utilización de un servidor; 2) el efecto de la compartición de la carga en el rendimiento de un servidor es similar al de añadir más capacidad al servidor; 3) el hecho de que no todos los vídeos están replicados en el sistema puede ser capturado mediante la sustracción de parte de la capacidad al servidor.

La primera observación nos dice que el factor de utilización es común para cualquier servidor, por tanto, al igual que en el caso unicast, podemos obtener el modelo desde el punto de vista de uno de los servidores.

La observación 2) captura la compartición de la carga mediante la adición de más capacidad al servidor gracias a que los demás servidores cooperan en la compartición de la carga.

La tercera observación indica que, aunque el algoritmo RLTH optimiza los recursos de la red (da la oportunidad de compartir los streams remotos), se tiene que reservar parte de los recursos para realizar estos servicios remotos obligatorios.

La incorporación de estas tres observaciones nos permite organizar el modelo analítico en tres pasos:

- En el primer paso obtendremos el modelo analítico de un servidor aislado. Este modelo recoge las características del algoritmo RLTH entre las que destacamos: por un lado, la asignación de un mismo recurso a más de una petición, y por otro, la diferencia en los tiempos de ocupación de los recursos (debidos a los streams parciales y a los completos).

- En el segundo paso (al igual que en el caso unicast), calculamos la capacidad remota disponible para compartir la carga entre los distintos servidores.

- En el último paso obtenemos el modelo completo del sistema.

Antes de presentar el modelo en tres pasos del sistema, obtendremos y validaremos a través de simulaciones el modelo analítico de un servidor aislado. El objetivo de este modelo es entender cómo interaccionan los recursos, así como estudiar cómo influye el algoritmo RLTH en el sistema. Los principales parámetros del modelo y del rendimiento aparecen en la Tabla 3.1.

Parámetro	Definición	Valor por defecto
N_{server}	Nº servidores en el sistema	-
B_{switch}	Ancho banda conmutador ATM	1000 Mbps, 16 Gbps
B_{link}	Ancho banda de un enlace (en cada dirección)	155, 622 Mbps Mbps
N_{stream}	Nº máx. canales por servidor (streams)	-
N_{term}	Número de clientes conectados a cada servidor	90, 150, 200, 300
T_{sleep}	Tiempo entre peticiones	distr. expon. (media 7200 seg.)
T_{active}	Duración servicio de un vídeo	distr. unif [3600,10800] seg.
B_{stream}	Ancho de banda que ocupa cada stream	15 Mbps
M	Nº de vídeos	100
ξ	Grado de popularidad	1
p_j	Probabilidad de solicitar el vídeo j	ecua. 1.1
$1-F$	Probabilidad media de solicitar un vídeo no local	ecua. 1.2, 1.3
T_{wait}	Tiempo de espera	-

Tabla 3.1 Parámetros del modelo analítico del algoritmo RLTH.

3.5.1. Modelo analítico de un servidor

El modelo que obtenemos en esta sección es válido tanto para el algoritmo Multicast con umbral (ver Figura 3.2) así como para nuestra estrategia *patching* con umbral (ver Figura 3.3). La principal diferencia está en la forma de obtener el umbral: U_i^* en el algoritmo Multicast con umbral (para más detalle ver [59]) y U_i en nuestra propuesta (ver [4]). En algunas expresiones de nuestro modelo analítico, usaremos el parámetro del umbral U_i porque nos centramos en el estudio de nuestra estrategia *patching*. Pero en vez de U_i, podríamos usar el valor de U_i^* si estudiásemos la estrategia Multicast con umbral. La misma consideración se tiene que hacer cuando se utilice cualquier otro algoritmo de *patching* con umbral.

La Figura 3.7 muestra esquemáticamente algunos de los parámetros del modelo de un servidor aislado. Como hemos estado haciendo a lo largo de nuestra investigación, suponemos que los tiempos entre las llegadas de dos clientes consecutivos a un servidor, siguen una distribución exponencial. Suponemos que el tiempo de servicio de un vídeo (o lo que es lo mismo, la duración de un vídeo L_i) sigue una distribución uniforme. En cualquier caso, el servidor VoD se puede modelar como una cola cerrada con *h canales de servicio paralelos*, del tipo $M/U/h/N_{term}/N_{term}$ con N_{term} clientes que acceden al servidor y con una capacidad máxima permitida de N_{term} clientes en el sistema.

Recordemos que el *throughput* (peticiones por unidad de tiempo), λ se calcula como (ecuación 2.6)

$$\lambda = \frac{N_{term}}{T_{sleep} + T_{wait} + T_{active}}$$

y que el factor de utilización (ecuación 2.8) tiene la expresión

$$\rho = \frac{\lambda \cdot T_{active}}{N_{stream}}$$

la cual nos da una buena estimación de la utilización del servidor. Para que el sistema esté en estado estacionario se tiene que verificar que ρ sea menor que uno, [66]. Si el término derecho de la expresión anterior excede de uno, entonces podemos esperar un elevado valor de T_{wait}. Por tanto, como el lado derecho tiene que ser menor que uno se tiene que

$$\lambda \cdot T_{active} < h \tag{3.11}$$

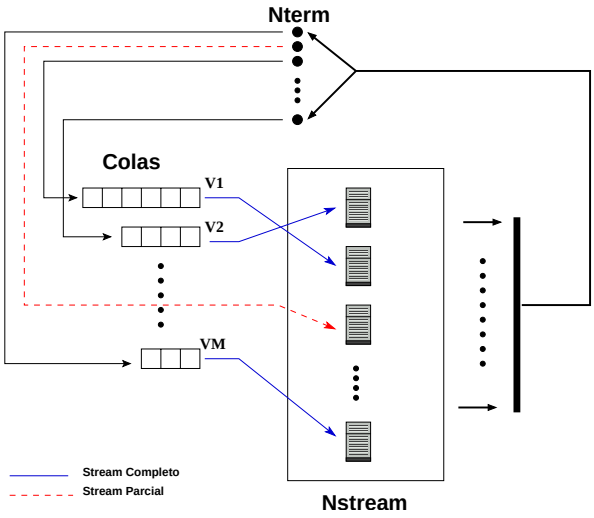


Figura 3.7 Esquema con los principales parámetros del servidor.

La validez de la ecuación 3.11 está confirmada en un amplio rango de experimentos [33]. Si la desigualdad se viola, entonces se puede esperar que T_{wait} sea inaceptablemente grande, como ya hemos mencionado. Por tanto, de las características de los vídeos (T_{active}), el perfil del usuario (T_{sleep}) y la configuración del sistema (N_{term} o número de usuarios), el diseñador del sistema puede determinar de antemano el mínimo número de canales de servicio, h, que el servidor debe tener para dar un rendimiento razonable.

En una estrategia multicast los h canales de servicio mencionados anteriormente, no representan directamente el número de canales de servicio que el servidor soporta físicamente (esto es, los N_{stream} canales en nuestro servidor). De hecho, una estrategia *patching* con umbral crea a los clientes la *ilusión* de que están siendo servidos con canales individuales, así que ellos piensan que hay h canales de servicio efectivos, siendo en realidad el número de canales físicos (N_{stream}) más pequeños que h.

A partir de un gran número de simulaciones hemos encontrado que

$$h = N_{stream} + n_v \tag{3.12}$$

siendo n_v un número de canales virtuales que se añaden a los canales físicos. h nos da el número de canales de servicio que nuestro servidor provee cuando se implementa una estrategia *patching* con umbral. Los n_v canales modelan la compartición de los recursos. De hecho, la idea básica de cualquier algoritmo multicast, incluido del tipo *patching*, es servir más de un cliente con un canal de servicio regular (el que transmite un stream completo), mientras que cada cliente tiene la ilusión de que el servidor está usando exclusivamente un canal para atenderlo.

Nuestro objetivo ahora es calcular n_v. En nuestro sistema, los vídeos tienen probabilidades distintas de ser solicitados, luego tendrán diferente uso de los recursos. Suponemos que las peticiones al vídeo i se generan siguiendo una distribución exponencial de media de tiempo entre las llegadas de $1/\lambda_i$, donde

$$\lambda_i = \lambda \cdot p_i$$

siendo λ la tasa de peticiones y p_i la probabilidad de solicitar el vídeo i. Claramente, los vídeos más populares se beneficiarán de la estrategia Multicast con umbral. En otras palabras, es razonable considerar que la estrategia Multicast con umbral se podría aplicar a los vídeos que verifican que

$$\lambda_i \cdot U_i > 1 \tag{3.13}$$

es decir, aquellos vídeos tales que al menos un cliente llega dentro del umbral y por tanto puede ser servido con un stream parcial a través de un canal de patching. En nuestras simulaciones, hemos encontrado que la expresión de la ecuación 3.13 es una condición un tanto pesimista.

De hecho, encontramos que el sistema está bien modelado considerando que la estrategia *patching* con umbral funciona de forma efectiva con aquellos vídeos que verifican que

$$\lambda_i \cdot U_i > 0{,}56 \tag{3.14}$$

Teniendo en cuenta estas consideraciones, obviamente la compartición de los recursos que una estrategia *patching* con umbral ofrece es debida a los vídeos que verifican la condición expresada en la ecuación 3.14. Luego, volviendo a la ecuación 3.12, podemos calcular el número de canales de servicio h que el servidor ofrece cuando implementa una estrategia *patching* con umbral como

$$h = N_{stream} + \sum_{\forall i/\lambda_i \cdot U_i > 0{,}56} n_{v_i} \tag{3.15}$$

donde n_{v_i} es el número de canales virtuales proporcionados para el servicio de los vídeos que efectivamente usan la estrategia *patching* con umbral.

Centremos nuestro análisis en el vídeo i. Volvamos a las Figuras 3.2 y 3.3 para entender el uso de los canales en una estrategia *patching* con umbral. Los canales de servicio pueden ser claramente clasificados en dos tipos:

- Los canales regulares que transmiten un stream completo; Por ejemplo, el Canal 1 y el Canal 2 en las Figuras 3.2 y 3.3.
- Los canales de patching que transmiten los streams parciales; Por ejemplo, Canal k+1, ..., Canal k+m en las Figuras 3.2 y 3.3.

Típicamente, una estrategia *patching* con umbral se puede modelar como un proceso de renovación [59], [4], [63], esto es, toda la duración de las peticiones se dividen en un conjunto de subintervalos llamados *Periodos*, y cada nuevo *Periodo* es independiente del *Periodo* anterior. Estamos interesados primeramente, en el cálculo del uso de los canales, teniendo en cuenta la compartición de los canales en las distintas sesiones multicast. Por ejemplo, en las Figuras 3.2 y 3.3 ese *Periodo* corresponde con L_i. En nuestro modelo, cada *Periodo* es además dividido en conjuntos de subintervalos, que son las ventanas correspondiente al umbral donde tiene lugar el *patching* (por ejemplo, las ventanas U_i^* y U_i de las Figuras 3.2 y 3.3, respectivamente). El primer stream completo en un *Periodo* ocupa un canal regular e inicia la *Primera sesión Multicast*. Cada uno de los próximos streams completos en el *Periodo*, ocupa también un canal regular e inician nuevas sesiones multicast. En la primera ventana del umbral, cada petición inicia un stream parcial que ocupa un canal durante un tiempo que es la diferencia entre el tiempo de llegada de la correspondiente petición y el tiempo en el que el stream completo se inició.

En los ejemplos, en el instante 0 se planifica la *Primera sesión Multicast* del vídeo i. En esta sesión, se ocupa un canal regular, el Canal 1, para transmitir un stream completo. El canal se ocupa durante todo el tiempo de servicio, L_i. Las peticiones que llegan dentro del umbral U_i (o U_i^*) de esta sesión multicast se sirven a través de los canales de patching. Estos canales de patching forman lo que llamamos los streams parciales asociados a la *Primera sesión Multicast*. Estas peticiones se sirven de forma inmediata con los Canales k+1, k+2, ..., pero estos canales se usan durante un periodo menor que U_i y como consecuencia, menor que el tiempo de servicio. Una vez que las peticiones han alcanzado al stream completo, se liberan los canales. Suponemos que estos canales utilizados para transmitir los streams parciales en la *Primera sesión Multicast* podrían ser reutilizados para servir los streams parciales asociados de la próxima sesión multicast para el mismo vídeo, como les ocurre al *Cliente* $s+1$ y al *Cliente* $s+2$ en la Figura 3.2 o al *Cliente* d y al *Cliente* $d+1$ en la Figura 3.3. Esta reutilización de los canales de patching ocurre de una manera cíclica, cualquiera que sea la sesión multicast iniciada. Esta observación nos dice que, durante un *Periodo*, parte de los canales físicos (los canales de patching) se comparten entre los usuarios de la segunda y sucesivas sesiones multicast.

Podemos calcular fácilmente el número de canales físicos de patching reutilizados como

$$n_{p_i} = \lambda_i \cdot U_i \tag{3.16}$$

que representa el número de peticiones que llegan dentro del umbral y que se sirven con streams parciales.

De la Figura 3.3 podemos claramente encontrar otra fuente de compartición de los recursos. Ocurre cuando una cola con clientes comparten el mismo stream completo. Por ejemplo, los clientes que están en la cola del vídeo cuando se inicia la primera sesión del vídeo i en el instante 0. Estos clientes comparten un canal regular (Canal 1). Otro ejemplo, *Cliente* s, *Cliente* $s+1$, ... están esperando en la cola del vídeo i hasta que se planifique la próxima sesión multicast del vídeo. Cuando esta sesión empieza, todas las peticiones pendientes se sirven con el mismo canal regular (Canal 2 en el ejemplo). De nuevo, durante un *Periodo*, parte de los canales físicos (los canales regulares) se comparten por los usuarios cada vez que se inicia una sesión multicast. Podemos calcular ahora el número de canales regulares reutilizados como

$$n_{r_i} = \lfloor \frac{L_i}{U_i} \rfloor \tag{3.17}$$

donde hemos asumido el peor caso desde el punto de vista de la ocupación de los canales. En otras palabras, el caso en el que durante un *Periodo*, cada U_i segundos, se inicia una nueva sesión multicast.

De las ecuaciones 3.17 y 3.16 obtenemos $n_{r_i} + n_{p_i}$, que es el número de canales físicos que debe tener el servidor para proporcionar los recursos que el algoritmo de *patching* con umbral necesita para el vídeo i.

Tenemos que subrayar que en nuestro modelo el algoritmo de *patching* con umbral crea la ilusión de que todos los clientes que llegan durante un *Periodo* se sirven con canales de forma individual. En otras palabras, que se sirven $\lambda_i \cdot L_i$ clientes con $\lambda_i \cdot L_i$ canales de servicio. Sin embargo, como hemos discutido previamente, durante ese *Periodo* los canales físicos de patching n_{p_i} y los canales regulares físicos n_{r_i} serán, de hecho, los canales ocupados en el servidor. Por tanto, podemos calcular un número de canales virtuales que representan aquellos clientes que no ocupan canales físicos como

$$n_{v_i} = \lambda_i \cdot L_i - (n_{r_i} + n_{p_i}) \tag{3.18}$$

Tenemos que destacar que estos n_{v_i} modelan los canales virtuales para el vídeo i, que los clientes creen que están utilizando de forma exclusiva para ellos, y cuando son añadidos a los canales físicos nos da el número de canales de servicio efectivos que provee el servidor. Esta ilusión sólo ocurre con los vídeos que trabajan de forma efectiva bajo el algoritmo de *patching* con umbral. En

este punto, podemos calcular el número total de canales de servicio h, que el servidor ofrece cuando se implementa el algoritmo de *patching* con umbral. Este número de canales h se obtiene de las ecuaciones 3.15 y 3.18 como

$$h = N_{stream} + \sum_{\forall i/\lambda_i \cdot U_i > 0{,}56} \lambda_i \cdot L_i - (n_{r_i} + n_{p_i}) \tag{3.19}$$

Aplicaciones del modelo

En esta sección analizamos el modelo para obtener algunas directrices para dimensionar un servidor aislado. La ecuación 3.11 nos da una expresión para calcular el número mínimo de canales físicos ($N_{stream_{min}}$) que el servidor debe tener para ofrecer un rendimiento razonable. Como h se puede calcular de la ecuación 3.19, tenemos que

$$N_{stream} > \lambda \cdot T_{active} - \left(\sum_{\forall i/\lambda_i \cdot U_i > 0{,}56} \lambda_i \cdot L_i - (n_{r_i} + n_{p_i}) \right) = N_{stream_{min}} \tag{3.20}$$

Ahora mismo necesitamos conocer alguna información del perfil del usuario (peticiones por unidad de tiempo, λ, y popularidad de los vídeos p_i), algunas características de los vídeos (duración de cada tiempo de servicio de los vídeos, L_i). A partir de estas informaciones, calculamos el umbral U_i (ecuación 3.9), el número de canales regulares n_{r_i} (ecuación 3.17) y el número de canales de patching n_{p_i} (ecuación 3.16).

El valor de $N_{stream_{min}}$ que acabamos de calcular es una cota inferior del número estimado de canales físicos que el servidor debe tener para evitar que se sature. Como veremos en la Sección 3.6 cuando el número de canales en el servidor está próximo a $N_{stream_{min}}$, los tiempos de espera, aunque razonables (hemos garantizado que el sistema no se sature), están lejos de ser cero. De hecho, uno de los objetivos del diseñador podría ser obtener el número de canales físicos que garantice que todos los usuarios sean servidos inmediatamente, es decir, que no tengan que esperar ($T_{wait} = 0$). Podemos obtener con nuestro modelo de un servidor aislado, una expresión exacta para asegurar este objetivo. Denotaremos al número de canales físicos requeridos para obtener $T_{wait} = 0$ como N_{stream_0}. Esta cuestión la estudiamos con más profundidad a continuación.

De la discusión del modelo en la Sección 3.5.1, hemos encontrado que los vídeos más populares, y más precisamente aquellos que verifican que $\lambda_i \cdot U_i > 0{,}56$, son los que llevan a cabo las sesiones multicast y se benefician de la reutilización de los canales. De hecho, en el cálculo de $N_{stream_{min}}$ hemos tenido en cuenta el número de canales n_{r_i} y n_{p_i}, ocupados por los servicios de estos vídeos populares durante un *Periodo*. A la inversa, los vídeos menos populares que son aquellos que verifican que $\lambda_j \cdot U_j \leq 0{,}56$ son los que probablemente serán servidos a través de sesiones unicast durante la duración de su servicio (o

Periodo). En otras palabras, cada uno de estos vídeos serán servidos a través de un canal regular. Podríamos contar el número de canales regulares ocupados por el servicio de un vídeo menos popular [66] como

$$n_{r_j} = \lambda_j \cdot L_j \tag{3.21}$$

Si incorporamos el número de canales regulares requeridos para realizar los servicios unicast de los vídeos menos populares a $N_{stream_{min}}$, obtenemos el número completo de canales que el servidor debe de tener para poder transmitir tanto las sesiones multicast como las unicast para todos los vídeos

$$N_{stream_0} = N_{stream_{min}} + \sum_{\forall j/\lambda_j \cdot U_j <= 0{,}56} n_{r_j} \tag{3.22}$$

Precisamente la ecuación 3.22 representa el número de canales que garantiza que $T_{wait} = 0$, ya que hemos contado todos los canales que el servidor debe tener para transmitir todos los tipos de servicios, tanto las sesiones multicast como las unicast. En la Sección 3.6 validaremos estas expresiones a través de varios experimentos.

Como las peticiones por unidad de tiempo, λ (o incluso la popularidad de los vídeos) puede cambiar a lo largo del tiempo, podríamos dimensionar el servidor para asegurar que, para unos valores dados de λ, el tiempo de espera no se degrade por debajo de un valor fijado T_{max}, y por tanto, podríamos asegurar este parámetro de Calidad de Servicio (QoS). Este puede ser otro de los objetivos en nuestro diseño. Tenemos que recordar que en muestro modelo el tiempo de espera se estima con un modelo de cola del tipo $M/U/h/N_{term}/N_{term}$. El tiempo de espera para este sistema se puede calcular como

$$T_{wait} = \frac{L_q \cdot (T_{sleep} + T_{active})}{N_{term} - L_q} \tag{3.23}$$

donde L_q representa el número medio esperado de clientes en cola y depende de los parámetros h y λ [66]. De la ecuación 3.23 podemos encontrar el valor de h que asegura que $T_{wait} < T_{max}$, para los distintos valores de λ. Aplicando la ecuación 3.19, podemos obtener de forma exacta el número de canales físicos, N_{stream}, que debe tener el servidor para asegurar el valor fijado de QoS, es decir, un tiempo de espera inferior a cierto T_{max}, que se calcula como

$$N_{stream} = h - \left(\sum_{\forall i/\lambda_i \cdot U_i > 0{,}56} \lambda_i \cdot L_i - (n_{r_i} + n_{p_i}) \right) \tag{3.24}$$

De nuevo, los parámetros que dependen del algoritmo de *patching* con umbral, como el umbral U_i, n_{r_i} y n_{p_i} se calculan de las ecuaciones 3.9, 3.17 y 3.16. En la Sección 3.6 hemos realizado una serie de experimentos para ilustrar esta aplicación de nuestro modelo.

3.5.2. Modelo analítico del sistema distribuido

Una vez que tenemos el modelo de un único servidor pasamos a presentar el modelo del sistema distribuido completo. Los pasos que se siguen son muy parecidos a los seguidos en el caso del algoritmo PRLS. Recordemos que al igual que en el caso unicast, las tres observaciones establecidas en el capítulo anterior nos permite organizar el modelo en tres pasos:

Paso 1 $M/U/h/N_{term}/N_{term}$ es el modelo analítico obtenido en la sección anterior de un servidor aislado y h se calcula según la ecuación 3.19.

El objetivo de este paso es calcular la tasa de llegada de las peticiones a servicios remotos. En el algoritmo RLTH sabemos que una petición es candidata para un servicio remoto obligatorio cuando la petición es para un vídeo que no está almacenado en el servidor (una petición a un vídeo no local). Por tanto una estimación de la tasa de llegada de servicios remotos obligatorios λ_r puede ser,

$$\lambda_r = \lambda \cdot (1 - F) \tag{3.25}$$

donde $(1 - F)$ es la probabilidad de solicitar un vídeo no local.

Paso 2 $M/U/n$ es el modelo para calcular la capacidad remota disponible para la compartición de la carga. Elegimos este modelo porque suponemos que no hay abandono en el sistema [28], es decir, los usuarios esperan hasta que son servidos.

En este paso nos diferenciamos del caso unicast en la forma de obtener n. n es la capacidad total remota disponible para realizar los servicios remotos. n se determina en función de: los anchos de banda del enlace (B_{link}) y del conmutador (B_{switch}), del factor de utilización y del número de servidores remotos y de la probabilidad de solicitar un vídeo no local. A continuación describimos como calculamos n.

Un servidor remoto que funciona de forma independiente tiene de media una capacidad disponible de

$$(1 - \rho) \cdot h$$

streams.

Como la utilización del servidor remoto casi no se ve afectada por la compartición de la carga, (como se ha establecido en la observación 1)), un servidor ve (de promedio) una capacidad disponible n_{idle} de

$$n_{idle} = (1 - \rho) \cdot h \cdot (N_{server} - 1) \tag{3.26}$$

ya que cada servidor ve la capacidad disponible en distintos instantes de tiempo. Sin embargo, sólo parte de esta capacidad disponible puede

ser utilizada para la compartición de la carga, básicamente porque tenemos que tener en cuenta tanto el ancho de banda del enlace como del conmutador.

El número de streams que pueden ser transmitidos a través de uno de los enlaces conectado al servidor es como mucho

$$n_{link}^{max} = B_{link}/B_{stream}$$

Igualmente, cada servidor ve esta capacidad en instantes distintos, por tanto, el número de streams que se pueden utilizar para compartir la carga, n_{link} coincide con n_{link}^{max}, con lo que

$$n_{link} = \frac{B_{link}}{B_{stream}} \tag{3.27}$$

Por otro lado, el número de streams por servidor que pueden ser transmitidos a través del conmutador es como mucho

$$n_{switch}^{max} = B_{switch}/(B_{stream} \cdot N_{server})$$

Siguiendo una deducción similar a la anterior, tenemos que el número de streams que se pueden utilizar del conmutador para compartir la carga es la que nos da la expresión n_{switch}^{max}. Por tanto tenemos que

$$n_{switch} = \frac{B_{switch}}{B_{stream} \cdot N_{server}} \tag{3.28}$$

Una vez que hemos calculado n_{idle}, n_{link} y n_{switch} podemos deducir el número medio de streams que están disponibles remotamente y que denotamos como n de la siguiente forma

$$n = min(n_{idle}, n_{link}, n_{switch}) \tag{3.29}$$

Una vez calculado n, consideramos el modelo de cola $M/U/n$ con un tasa de llegada dada por λ_r (ecuación 3.25) y un tiempo medio de servicio de T_{active} segundos. Con este modelo calculamos m que es el número medio de streams activos. Este m es nuestra estimación del número medio de peticiones a vídeos no locales que se sirven de forma remota. En otras palabras, m es el número de servicios remotos obligatorios realizados. La expresión para calcular m viene dada por [53],

$$m = \lambda_r \cdot T_{active} = \lambda \cdot (1 - F) \cdot T_{active} \tag{3.30}$$

Paso 3 $M/U/h+v/N_{term}/N_{term}$ es nuestro modelo para caracterizar la compartición de la carga, como mencionamos en la observación 2). De hecho a través de las simulaciones, hemos verificado que el rendimiento de nuestro algoritmo RLTH se puede aproximar con un modelo de cola del tipo $M/M/h + v/N_{term}/N_{term}$, es decir, cada servidor se comporta como si

tuviera una capacidad (virtual) añadida de v streams gracias a la compartición de la carga.

Lo primero es calcular v. Este v es el número medio de recursos disponibles para compartir la carga y al igual que en el caso unicast v se estima como $v = n - m$. Esta es la diferencia entre el número medio de streams disponibles remotamente (n) y el número medio de peticiones que se sirven obligatoriamente de forma remota (m). En otras palabras, v es la capacidad remota que no está ocupada con los servicios obligatorios y que por lo tanto está disponible para compartir la carga. Una vez que hemos calculado v de las ecuaciones de la cola $M/U/h + v/N_{term}/N_{term}$ [53] calculamos el tiempo medio de espera en nuestro sistema (T_{wait}).

Debemos tener cuidado con el valor de v ya que puede tomar valores positivos y negativos. Cuando v es positivo lo que significa es que hay capacidad remota disponible para realizar la compartición de la carga y por tanto esta capacidad se puede añadir al servidor. Es decir, el flujo de servicios remotos obligatorios (m), no supera la capacidad disponible remota (n). En este caso, puede ocurrir que el número de canales que se añaden al servidor ($v = n - m$) sea mayor que el número de servicios remotos reales que se realizan gracias a la funcionalidad de la compartición de la carga. El número de servicios remotos debidos a la compartición de la carga, m', se calcula como

$$m' = \lambda \cdot p \cdot R \cdot T_{active}$$

donde p es la probabilidad de que el servidor esté ocupado y $R = \sum_{j=1}^{s} p_j$ es la probabilidad de solicitar un vídeo totalmente replicado.

Por tanto, en este caso v se estima como

$$\text{mín}\{m', v\}$$

Cuando v es negativo lo que ocurre es que hay más peticiones a los vídeos no locales que recursos en la red para atenderlas. Como estas peticiones consumen obligatoriamente ancho de banda del enlace y del conmutador, no hay capacidad disponible para poder compartir la carga y lo que ocurre es que el servidor tiene que reservar $|v|$ streams para atender el tráfico de servicios remotos obligatorios (observación 3)).

3.6. Resultados experimentales

En esta sección presentamos varios experimentos para medir el rendimiento de los algoritmos que proponemos así como para analizar la validez de nuestro modelo analítico. Estos experimentos nos han ayudado para conseguir una nueva percepción de como interactúan los recursos. Empezamos con una descripción de los parámetros que utilizamos en las simulaciones, seguido de la descripción de los distintos experimentos que hemos realizado. Seguidamente, examinamos la validez del modelo y analizamos los resultados.

3.6.1. Parámetros de la simulación

En las simulaciones hemos asumido los siguientes parámetros que aparecen también en la Tabla 3.1. Cada cliente genera peticiones con unos tiempos entre llegadas que obedecen una distribución exponencial de tiempo medio $T_{sleep} = 7200$ segundos. El ancho de banda de cada vídeo stream es $B_{stream} =$ 15 Mbps. El número total de vídeos ofrecidos por el servidor es $M = 100$. La distribución de los vídeos entre los servidores se realiza según lo que se ha explicado en la Sección 1.3.2 del Capítulo 1. La duración de los vídeos sigue una distribución uniforme en el intervalo [3600,10800] segundos. Luego, el tiempo medio de servicio es $T_{active} = 7200$ segundos. Para asignar una popularidad a cada vídeo, utilizamos la distribución pura Zipf ($\xi = 1$). En estos experimentos, hemos estudiado el comportamiento de nuestro algoritmo de *patching* con umbral para el que se tiene en cuenta los efectos de tener un ancho de banda limitado. Como vimos en la Sección 3.5.1, esto significa que en la implementación de nuestra estrategia, los servidores mantienen una cola por vídeo para que las peticiones se pongan en ella cuando no haya recursos en el servidor. Hemos usado la ecuación 3.9 para calcular el umbral U_i para cada vídeo, ya que no sabemos el tiempo de espera de antemano.

La principal métrica en estos experimentos es el tiempo de espera T_{wait}, que es uno de los principales parámetros que caracteriza la QoS en estos sistemas.

Los parámetros que definen la capacidad de la red de interconexión, B_{link} y B_{switch}, así como el número de clientes conectados al servidor, N_{term}, y el número de streams que se pueden realizar a la vez (que corresponde con el número de canales físicos, N_{stream}) variarán dependiendo del experimento realizado. Para calcular el número de canales físicos necesarios para cada experimento, utilizamos las ecuaciones 3.18 y 3.15.

3.6.2. Comportamiento de los algoritmos LTH y RLTH

En esta sección presentamos los resultados de las simulaciones que realizamos para evaluar el comportamiento de nuestros algoritmos. Además comparamos nuestras propuestas, LTH y RLTH, con una versión distribuida del algoritmo MFQL [28].

Este estudio nos dará información significativa sobre el impacto que tienen los algoritmos que implementan umbrales en un sistema distribuido VoD cuando se comparan con una estrategia *batching*, como en el caso de MFQL. También nos dará una comparación cuantitativa de los servicios cuando sólo se les aplica de forma local el umbral y cuando se les aplica además el umbral de forma remota.

Para estos experimentos suponemos que el número de clientes conectados a cada servidor es $N_{term} = 90$; el número de streams que se pueden utilizar a la vez en cada servidor es $N_{stream} = 50$. El ancho de banda de cada enlace del servidor al conmutador es $B_{link} = 155$ Mbps mientras que el ancho de banda

que suponemos para el conmutador es $B_{switch} = 1000$ Mbps. El porcentaje de replicación en el sistema varía del 75 %, al 50 % y finalmente al 25 %.

Para cada simulación, el número de servicios que se realizó en cada servidor fue al menos 5000. Rechazamos las primeras 1000 muestras a fin de lograr un estado estadístico estacionario.

Análisis de los resultados

La Figura 3.8 representa los resultados del rendimiento de los algoritmos MFQL (representado con las líneas sólidas), LTH (con líneas punteadas) y RLTH (líneas rayadas) en nuestro sistema distribuido de VoD.

El eje X representa el número de servidores en el sistema, N_{serv}, y el eje Y representa el tiempo medio de espera en segundos.

De la Figura 3.8 podemos observar que en cualquier caso de replicación, tanto LTH como RLTH tienen tiempos de espera más pequeños que MFQL. Como era de esperar, RLTH tiene los tiempos de espera más pequeños. Esto es porque tanto el uso del ancho de banda local como el de la red de interconexión, se optimizan con esta última propuesta de algoritmo. Esta es la razón por la que nos hemos decantado por el estudio analítico de este algoritmo, claramente el más eficiente de los analizados. Una observación importante es que aunque los tiempos obtenidos con el algoritmo LTH son más pequeños que los de MFQL, la diferencia de tiempos entre MFQL y LTH se vuelve más pequeña conforme disminuye el porcentaje de replicación. La razón de este comportamiento se analiza con más detalle en cada caso de replicación.

Primero vamos a considerar el 75 % de replicación (Figura 3.8(a)). Para este porcentaje de replicación podemos ver que los resultados están alrededor de un segundo para LTH y para RLTH, cualquiera que sea el número de servidores en el sistema, mientras que los resultados obtenidos con MFQL están alrededor de los veinte segundos.

El tiempo de espera es mayor para MFQL ya que los otros algoritmos implementan la estrategia *patching* con umbral para los servicios locales. Como la mayoría de las peticiones son a vídeos locales, ya que hay un 75 % de replicación, estas peticiones son servidas inmediatamente con un stream parcial mientras que con MFQL, estas peticiones tienen que esperar en cola hasta que sean planificadas.

Podemos ver que los tiempos de espera para LTH y RLTH son similares. Esto es debido al porcentaje de replicación que estamos considerando. En este caso, no hay mucho tráfico en la red de interconexión porque no se producen muchos servicios remotos. Claramente en esta situación, la clave está en minimizar el uso del ancho de banda local, lo que LTH hace bastante bien.

El rendimiento en el caso del 50 % de replicación está representado en la Figura 3.8(b). En general, el tiempo medio de espera de los usuarios no excede de

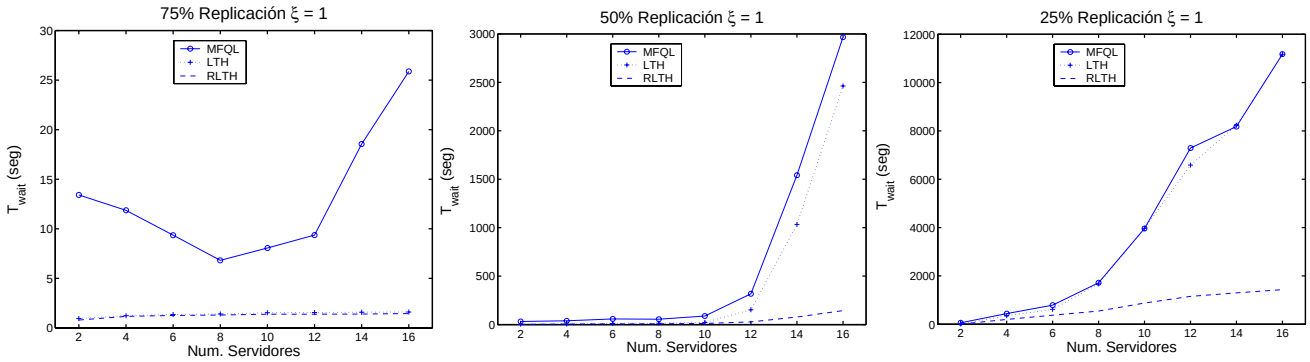


Figura 3.8 Resultados del rendimiento de los algoritmos MFQL, LTH y RLTH. El eje X representa el número de servidores en el sistema. El eje Y representa el tiempo medio de espera en segundos. Porcentajes de replicación: (a) 75 %, (b) 50 %, (c) 25 %. En todos los casos $\xi = 1$.

los cinco minutos para los algoritmos MFQL y LTH si el sistema tiene hasta 12 servidores, mientras que para RLTH tenemos que de 2 a 16 servidores los tiempos no son mayores de tres minutos. Aunque no aparece en la figura, cuando el número de servidores en el sistema está entre 2 y 12, el tiempo medio de espera obtenido con LTH son más pequeños que los producido por MFQL en un factor de 4. De nuevo, podemos ver que la estrategia de *patching* con umbral optimiza de forma efectiva el uso del ancho de banda local. Sin embargo, para ambos algoritmos LTH y RLTH, podemos ver que la forma de los tiempos de espera crecen rápidamente a partir de 12 servidores. La explicación es que la red está colapsada porque hay un incremento en el número de servicios remotos debido a las peticiones remotas. Se producen más servicios remotos que locales en el sistema y por tanto, la ventaja de la estrategia con umbral aplicada a los vídeos locales no tiene suficiente impacto en este caso. Luego, la red se convierte en el cuello de botella a partir de 12 servidores para el 50 % de replicación. Por el contrario, podemos observar que la forma de los tiempos de espera para el algoritmo RLTH es la más baja y que se mantiene por debajo de los tres minutos para cualquier número de servidores. Esto es porque el algoritmo RLTH implementa la estrategia *patching* con umbral con los servicios remotos. Como consecuencia, se reduce significativamente el uso del ancho de banda de la red de interconexión evitando que se convierta en el cuello de botella del sistema.

Hay que resaltar que MFQL y LTH sirven las colas de peticiones remotas en distintos instantes de tiempo y que usan un stream completo por cola, lo que incrementa el uso de la red de interconexión y como consecuencia, se incrementan los tiempos de espera. Sin embargo, RLTH permite que todas las colas del mismo vídeo en el sistema se sirvan a la vez con sólo un stream completo cada vez que se planifica un servicio remoto para ese vídeo.

En general, con el 50 % de replicación, el tiempo medio de espera de los clientes no excede de cinco minutos para las estrategias MFQL y LTH si el sistema no

tiene más de 12 servidores, mientras que para RLTH tenemos que de 2 a 16 servidores el tiempo de espera no es mayor de tres minutos.

Nº serv.	**MFQL**	**LTH**	**RLTH**
2	61	11	7
4	441	350	201
6	792	618	375
8	1718	1670	548
10	3964	3969	878
12	7288	6586	1155
14	8181	8231	1299
16	11174	11169	1432

Tabla 3.2 Tiempos medios de espera, en segundos, cuando el porcentaje de replicación es del 25 % y los anchos de banda de la red son: $B_{link} = 155$ Mbps y $B_{switch} = 1000$ Mbps.

Por último, vamos a analizar los resultados obtenidos con el 25 % de replicación que aparece representado en la Figura 3.8(c). También mostramos los tiempos de espera de esta figura en la Tabla 3.2. En este caso, tenemos que la red se convierte en el cuello de botella a partir de 8 servidores para los algoritmos MFQL y LTH. Cuando el número de servidores en el sistema está entre 2 y 8, los tiempos medios de espera obtenidos con LTH son ligeramente más pequeños que los producidos con MFQL. Pero de 8 a 16 servidores, MFQL y LTH tienen tiempos similares. Aparentemente, la estrategia Multicast con umbral tiene muy poco efecto en la optimización del uso del ancho de banda local. Sin embargo, lo que ocurre es que hay un importante incremento en el número de servicios remotos y este incremento enmascara la reducción en el uso del ancho de banda local que consigue LTH. En esta situación no es tan importante reducir el uso del ancho de banda local. La clave está en minimizar el uso del ancho de banda de la red. Por esta razón, el algoritmo RLTH tiene mejores resultados que las otras estrategias, de hecho el cuello de botella aparece más tarde, a partir de 12 servidores. Con este último algoritmo, los tiempos de espera están por debajo de los cinco minutos cuando el número de servidores es menor que 6. Sin embargo, los tiempos de espera aumentan rápidamente a partir de 8 servidores y se vuelven en insoportables, cuando el número de servidores es mayor de 12. Sin embargo, de la figura podemos observar que la tasa de degradación en los tiempos es menor en el algoritmo RLTH cuando la comparamos con las tasas de MFQL y LTH.

Una forma de reducir los tiempos de espera cuando el grado de replicación es bajo, consiste en incrementar el ancho de banda de la red, que es el cuello de botella en el sistema como ya hemos mencionado antes.

Para testear el comportamiento de nuestro algoritmo RLTH y encontrar dónde aparece el cuello de botella cuando aumentamos la capacidad de la red, hemos realizado un experimento en el que aumentamos un 20 % los recursos de la red; ahora B_{link} es 185 Mbps y B_{switch} es 1200 Mbps. Para los tres algoritmos hemos medido de nuevo los tiempos de espera para el 25 % de replicación. Los

resultados de este experimento los mostramos en la Tabla 3.3.

Podemos observar que el cuello de botella aparece ahora a partir de 10 servidores para los algoritmos MFQL y LTH. Sin embargo, para el algoritmo RLTH, aunque hay un incremento en los tiempos de espera a partir de 10 servidores, los tiempos son aun aceptables, es decir, no hay cuello de botella.

Nº serv.	**MFQL**	**LTH**	**RLTH**
2	27	6	3.7
4	207	89	44
6	358	212	96
8	559	465	153
10	2870	2249	367
12	5236	4783	589
14	6268	6114	744

Tabla 3.3 Tiempos medios de espera, en segundos, cuando el porcentaje de replicación es del 25 % y los anchos de banda de la red son: $B_{link} = 185$ Mbps y $B_{switch} = 1200$ Mbps.

De estos experimentos hemos aprendido que nuestras estrategias *patching* con umbral, LTH y RLTH, mejoran un algoritmo basado en la estrategia *batching* como es el caso de MFQL, especialmente porque optimizan el uso del ancho de banda local de los servidores cuando el porcentaje de replicación es elevado. Sin embargo, esta optimización de los recursos locales no es muy útil cuando el porcentaje de replicación es pequeño. Otra lección es que el algoritmo RLTH mejora drásticamente los tiempos de espera de los clientes, pero esta mejora no puede evitar la degradación del comportamiento del sistema cuando la red se convierte en el cuello de botella, situación que aparece cuando el porcentaje de replicación es pequeño y el número de servidores en el sistema es elevado. Sin embargo, con un pequeño incremento en los recursos de la red se puede fácilmente resolver este problema.

3.6.3. Precisión del modelo analítico de un servidor

Para examinar la precisión de nuestro modelo analítico de un servidor, hemos realizado dos conjuntos de experimentos. En el primer conjunto, variamos el número de clientes conectados al servidor (N_{term}), es decir, variamos la carga del servidor (λ y ρ, ver ecuaciones 2.6 y 2.8). El objetivo de estos experimentos fue verificar que para cualquier carga en el servidor, el modelo analítico predice los resultados de la simulación. En el segundo conjunto de experimentos, fijamos el número de clientes $N_{term} = 300$ y variamos los valores de los parámetros T_{sleep} y T_{active}. El objetivo de estos experimentos fue verificar que para cualquier objeto multimedia, el modelo analítico se ajusta de nuevo a los resultados de la simulación.

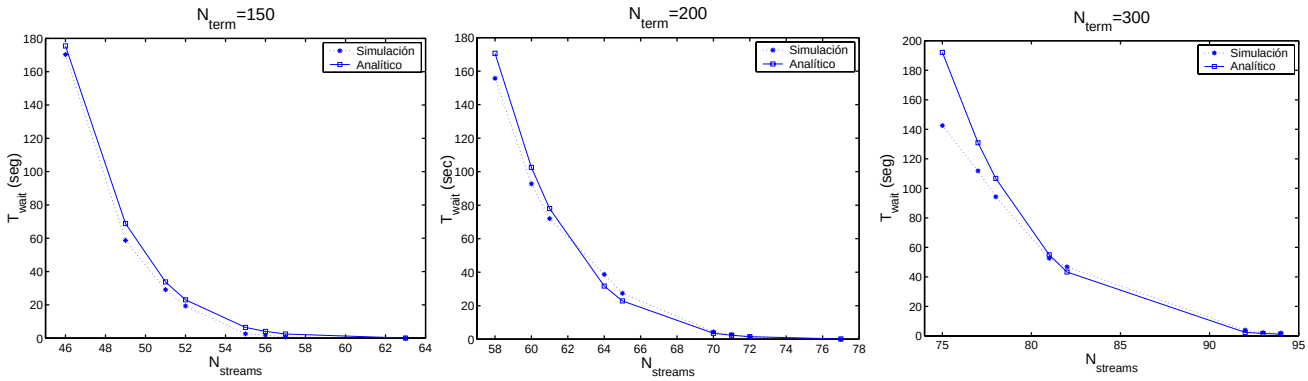
Para cada simulación el número de servicios que se realizó fue al menos 3000. Los primeros 750 servicios se descartaron a fin de lograr un estado estadístico

estacionario.

En el primer conjunto de experimentos establecimos: a)$N_{term} = 150$, más tarde lo fijamos a b) $N_{term} = 200$ y finalmente asignamos c) $N_{term} = 300$. Para cada uno de estos tres casos, variamos el número de canales físicos en el servidor (N_{stream}), de $N_{stream_{min}}$ a N_{stream_0}. En estos experimentos medimos a través de las simulaciones, los tiempos medios de espera cuando implementamos nuestra propuesta de algoritmo y los comparamos con los tiempos de espera analíticos obtenidos de la ecuación 3.23. Recordemos que estos tiempos de espera corresponden con los que se obtienen según el modelo de cola $M/U/h/N_{term}/N_{term}$ donde h se calcula según la ecuación 3.19.

De la ecuación 3.20 se puede calcular $N_{stream_{min}}$ que representa la cota inferior de los canales físicos que tiene que tener el servidor para transmitir con un rendimiento razonable. Para cada caso, tenemos: a) $N_{stream_{min}} = 44$, b) $N_{stream_{min}} = 56$, y c) $N_{stream_{min}} = 74$, respectivamente. Por otro lado, de la ecuación 3.22 podemos obtener N_{stream_0} y, en nuestro modelo, representa el número de canales que garantiza que $T_{wait} = 0$. De nuevo para cada caso tenemos: a) $N_{stream_0} = 63$, b) $N_{stream_0} = 77$ y c) $N_{stream_0} = 94$, respectivamente.

La Figura 3.9 representa los resultados obtenidos mediante las simulaciones (las marcas con forma de asteriscos) para cada caso. El eje X representa el número de canales N_{stream} en el servidor, y el eje Y representa el tiempo medio de espera en segundos. Además, representamos los tiempos de espera obtenidos del modelo analítico (las marcas cuadradas).



(a) (b) (c)

Figura 3.9 Comportamiento de nuestro algoritmo y precisión del modelo analítico cuando: (a) $N_{term} = 150$; (b) $N_{term} = 200$; (c) $N_{term} = 300$. El eje X representa el número de canales físicos en el servidor, y el eje Y el tiempo de espera en segundos.

De la Figura 3.9 podemos observar que el modelo analítico representa de forma exacta el rendimiento del algoritmo. Observamos que en todos los casos los resultados analíticos se ajustan a los simulados. Sin embargo, podemos ver en la Figura 3.9(c) que en el caso de una tasa elevada de peticiones y un número de canales cercano al mínimo, es decir, cercano a $N_{stream_{min}}$, hay una pequeña diferencia entre los tiempos de espera medidos y los esperados analíticamente.

De hecho, en esta situación, nuestro modelo tiende a sobreestimar los tiempos de espera de los clientes. Lo que ocurre es que nuestro modelo asume un escenario pesimista en el que los canales de patching (n_{p_i}) se ocupan, en una sesión multicast, durante toda la duración del umbral U_i. Sin embargo, en las medidas reales, algunos de los canales de patching están ocupados sólo durante una parte del umbral U_i, y este número de canales de patching es especialmente significativo cuando la tasa de peticiones es elevada. En otras palabras, en el caso de una tasa elevada, hay algunos canales que no se ocupan durante toda la duración del umbral y podrían dar la oportunidad de que otros clientes lo reutilizasen en la misma sesión multicast. Por tanto, en este caso, el número de canales de patching necesarios para transmitir streams parciales puede ser menor que el número que estimamos en la ecuación 3.16. Esta reutilización de los canales de patching para este caso en particular no es capturado por nuestro modelo, y por esta razón los tiempos esperados son mayores que los medidos. Por otro lado, cuando el número de canales aumenta, los tiempos analíticos se ajustan con bastante exactitud a los medidos. En estas otras situaciones, la reutilización de los canales de patching no es tan importante porque hay suficiente recursos disponibles en el servidor. En estos casos, el uso de los canales es bastante similar al que hemos descrito en la Sección 3.5.1 y nuestro modelo parece capturar de forma exacta el comportamiento del algoritmo.

De todas formas, creemos que la ecuación 3.20 nos da aun una guía válida del número mínimo de canales que tiene que tener el servidor para evitar la saturación. En las figuras, los tiempos de espera cuando $N_{stream} = N_{stream_{min}}$ están siempre por debajo de 200 segundos, y este tiempo podría ser razonable en algunos sistemas.

Por otro lado, con respecto al número de canales que garantiza que $T_{wait} = 0$, esto es, N_{stream_0}, podemos ver en las Figuras 3.9(a),(b),(c) que nuestro modelo predice este parámetro con bastante precisión. Este resultado demuestra la validez de la ecuación 3.22. De hecho, podemos observar que con menos canales que N_{stream_0} los tiempos de espera son muy pequeños. Por tanto, podríamos ahorrar recursos y seleccionar un número de canales menor que N_{stream_0} y aun conseguir un buen rendimiento. Veremos este tema con más detalle en la Sección 3.6.3.

Realizamos un segundo conjunto de experimentos donde variamos otros parámetros, para verificar de nuevo la precisión de nuestro modelo. En estos experimentos, ejecutamos otro grupo de simulaciones donde fijamos $N_{term} = 300$ y variamos los parámetros T_{sleep} y T_{active}. En otras palabras, cambiamos el comportamiento de los clientes y de los objetos multimedia. En la Tabla 3.4 mostramos los resultados obtenidos. Seleccionamos los valores de N_{stream} de forma que eran un poco mayores que $N_{stream_{min}}$, en cada caso. Las dos últimas columnas representan los tiempos de espera (en segundos) medidos en las simulaciones (T_{wait_s}) y los tiempos de espera calculados de nuestro modelo analítico (T_{wait_a}), para cada caso. Como podemos ver, en todos los casos los tiempos simulados y los analíticos están muy cercanos. Estos resultados confirman que nuestro modelo analítico captura con bastante precisión el rendimiento de un algoritmo de *patching* con umbral, teniendo en cuenta el comportamiento del

usuario (T_{sleep}) y cualquiera que sea la duración del servicio del objeto multimedia (T_{active}).

T_{sleep}	T_{active}	N_{stream}	T_{wait_s}	T_{wait_a}
7200	5000	70	46	48
7200	5000	73	20	22
7200	5400	74	32	35
7200	5400	77	10	15
9000	7200	74	57	59
9000	7200	77	25	30

Tabla 3.4: Comparación de los tiempos de espera obtenidos mediante simulaciones y analíticamente. Fijamos $N_{term} = 300$.

Dimensionado de un servidor

Como indicamos en la Sección 3.5.1 podríamos utilizar nuestro modelo analítico para dimensionar el servidor y para seleccionar un número aproximado de canales físicos que garantice un tiempo de espera menor que un valor fijado T_{max}, cuando alguno de los parámetros cambia. Para ilustrar esta cuestión, suponemos que λ (el número de peticiones por unidad de tiempo) es el parámetro que cambia a lo largo del tiempo. En la Figura 3.10 representamos los tiempos de espera (en segundos) que predice nuestro modelo cuando λ (ver ecuación 2.6) varía de 0,625 peticiones por minuto (el número de clientes conectados al servidor es $N_{term} = 150$) a 1,25 peticiones por minuto (ahora el número de clientes conectados al servidor es $N_{term} = 300$). Calculamos los tiempos de espera según el modelo de colas $M/U/h/N_{term}/N_{term}$, siendo $T_{sleep} = 7200$ segundos y el tiempo medio de servicio $T_{active} = 7200$ segundos. Los tiempos de espera se calculan para $h = 95$, $h = 113$, $h = 133$ y $h = 155$ canales de servicio.

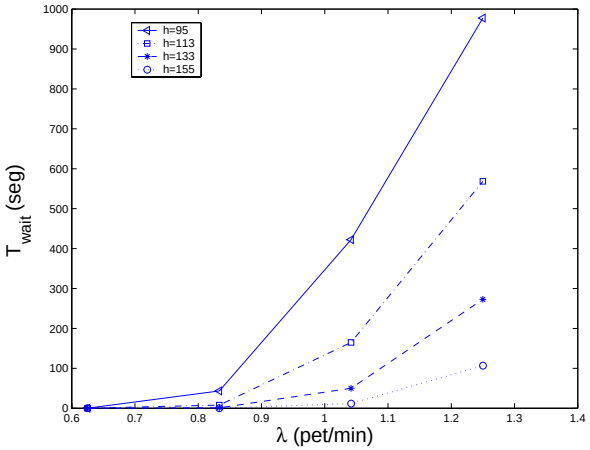


Figura 3.10 Tiempo de espera estimado cuando λ varía; T_{sleep} y T_{active} son 7200 segundos.

De la Figura 3.10 podemos observar que los tiempos de espera tienen un comportamiento exponencial cuando aumenta λ. Por ejemplo, iniciamos el estudio con $h = 95$, y aplicando la ecuación 3.24 tenemos que $N_{stream} = 63$. De hecho, este es el número de canales físicos $N_{stream_{min}}$ (ver ecuación 3.20) cuando λ es mínimo (λ = 0,625 pet/min). En este caso, los tiempos de espera se degradan muy rápidamente a partir de $\lambda > 0{,}8$ pet/min. La razón de este comportamiento es que en este caso se viola la desigualdad 3.11, es decir, el factor de utilización del servidor es mayor que uno, y como mencionamos en la Sección 3.5.1 esto significa unos tiempos de espera muy elevados. Sin embargo, podemos ver que cuando aumentamos el número de canales de servicio (h), la pendiente de las curvas tiende a ir hacia abajo suavemente. Por ejemplo, en el caso $h = 155$ (que corresponde con $N_{stream} = 78$ de acuerdo con la ecuación 3.24), el tiempo de espera es siempre menor que 100 segundos (el cual podría ser T_{max}). Claramente, para el amplio rango de variación de λ, este número de canales físicos podría representar un tamaño válido para nuestro sistema. En otras palabras, podríamos dimensionar el servidor con $N_{stream} = 78$ canales físicos para asegurar que el tiempo de espera no se degrada por debajo de un valor fijado $T_{max} = 100$ segundos, y por tanto podríamos asegurar este parámetro de QoS.

3.6.4. Precisión del modelo analítico para el sistema distribuido

En esta sección presentamos algunos experimentos para medir la validez de nuestro modelo analítico del sistema distribuido implementando el algoritmo RLTH.

Como hemos visto en la Sección 3.6.2, el caso más crítico (y el que se obtiene peores tiempos de espera) es cuando en el sistema hay un 25 % de replicación. Hemos visto también que en este caso, la red de interconexión es el cuello de botella y que con un pequeño incremento en los recursos de la red, los tiempos mejoran. Por tanto, para validar el modelo analítico hemos realizado un conjunto de experimentos en un sistema con un 25 % de replicación.

Los parámetros establecidos son los siguientes: cada cliente genera peticiones con unos tiempos entre llegadas que obedecen una distribución exponencial de tiempo medio $T_{sleep} = 7200$ segundos; la duración de los vídeos sigue una distribución uniforme en el intervalo [3600,10800] segundos. El ancho de banda (en cada sentido) del enlace que une a un servidor con el conmutador ATM es de 622 Mbps, mientras que el del conmutador es de 16 Gbps. El número de clientes conectados a cada servidor son: a)$N_{term} = 150$ en el primer conjunto de experimentos, en el segundo lo fijamos a b) $N_{term} = 200$ y en el último conjunto de experimentos realizados asignamos c) $N_{term} = 300$. Para cada uno de estos tres casos, escogimos un número de canales físicos en un servidor (N_{stream}) superior a $N_{stream_{min}}$, de forma que los tiempos de espera fueran menores que 20 segundos (ver Figura 3.9). Los números de canales físicos escogidos en cada caso fueron: a) $N_{stream} = 54$, b)$N_{stream} = 66$ y c) $N_{stream} = 86$, respectivamente.

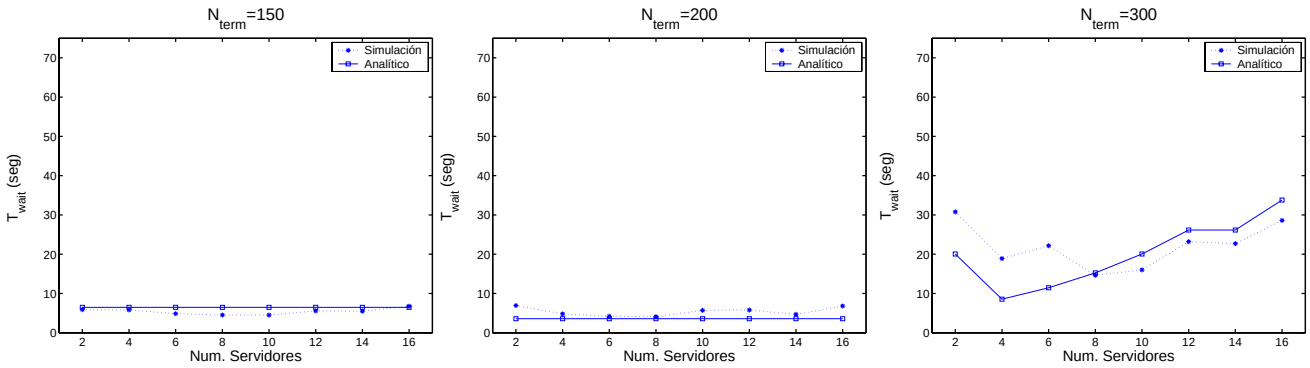


Figura 3.11 Resultados del rendimiento del algoritmo RLTH y del modelo analítico cuando: (a) $N_{term} = 150$; (b) $N_{term} = 200$; (c) $N_{term} = 300$. Porcentaje de replicación: 25 %, $\xi = 1$, $B_{link} = 622$ Mbps y $B_{switch} = 16$ Gbps. El eje X representa el número de servidores, y el eje Y son los tiempos de espera en segundos.

La Figura 3.11 representa el rendimiento de nuestro algoritmo RLTH obtenido mediante simulaciones (las marcas con forma de asterisco) para cada caso. El eje X representa el número de servidores en el sistema, N_{serv}, y el eje Y representa el tiempo medio de espera, T_{wait}, en segundos. Además, representamos los tiempos de espera obtenidos del modelo analítico (las marcas cuadradas).

De la Figura 3.11 podemos observar que el modelo analítico representa de forma bastante precisa el rendimiento del algoritmo RLTH en el sistema distribuido. Observamos que en todos los casos los resultados analíticos se ajustan a los simulados. Podemos apreciar en la Figura 3.11(c), que hasta 8 servidores los tiempos analíticos son menores que los simulados. Esto se debe a que el modelo sobreestima el número de streams disponibles para compartir la carga. Para un número pequeño de servidores en un sistema con una tasa elevada de peticiones ($N_{term} = 300$), el número de peticiones a vídeos no locales no supera al número de streams remotos disponibles, con lo que el modelo añade recursos para poder compartir la carga. Sin embargo, en las simulaciones se realizarán más servicios remotos de vídeos locales que recursos disponibles para ellos. En consecuencia, se ocuparán los recursos destinados a los servicios remotos de vídeos no locales (los servicios obligatorios), con lo que algunos de ellos tendrán que esperar. Sin embargo, a partir de 8 servidores los tiempos simulados son ligeramente menores que los analíticos. Como aumenta el número de servidores en el sistema, aumenta el número de peticiones a vídeos no locales a través de la red. En este caso, el modelo reserva parte de los recursos del servidor para realizar los servicios remotos obligatorios. Sin embargo, una de las características del algoritmo RTHL es que optimiza el uso de los recursos de la red, con lo que da la oportunidad de que se realice algún servicio remoto de peticiones a vídeos locales. Es decir, se puede compartir la carga entre los servidores, con lo que los tiempos de espera disminuyen.

En los experimentos anteriores hemos aumentado tanto la tasa de peticiones ($N_{term} = 150$, $N_{term} = 200$ y $N_{term} = 300$), como la capacidad de la red

($B_{link} = 622$ Mbps y $B_{switch} = 16$ Gbps) en un sistema con un bajo porcentaje de replicación (25 %). Hemos visto que conforme aumenta la tasa peticiones el tiempo de espera aumenta, pero los tiempos de espera aun son razonables (menos de un minuto).

Para analizar el comportamiento de nuestro algoritmo en un sistema con unas condiciones más pesimistas, realizamos un conjunto de experimentos donde el porcentaje de replicación es del 15 % (por tanto, la capacidad de almacenamiento es pequeña), y la tasa de peticiones es elevada ($N_{term} = 300$). Los demás parámetros del sistema son iguales a los de los experimentos anteriores.

La Figura 3.12 representa los resultados de las simulaciones (las marcas con forma de asterisco). El eje X representa el número de servidores en el sistema, N_{serv}, y el eje Y representa el tiempo medio de espera en segundos. Además, representamos los tiempos de espera obtenidos del modelo analítico (las marcas cuadradas).

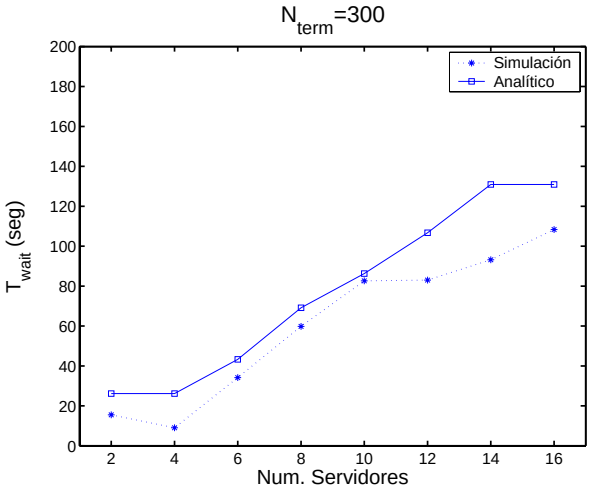


Figura 3.12 Resultados del rendimiento del algoritmo RLTH y el modelo analítico. Porcentaje de replicación 15 %, $\xi = 1$, $N_{term} = 300$, $B_{link} = 622$ Mbps, $B_{switch} = 16$ Gbps; T_{sleep} y T_{active} son 7200 segundos.

Como podemos observar, aunque estamos en un escenario pesimista, los tiempos de espera permanecen por debajo de los 100 segundos, con lo que los tiempos de espera no se degradan mucho. También podemos observar que el modelo analítico representa de forma bastante precisa el rendimiento del algoritmo RLTH. Podemos apreciar que para más de 10 servidores en el sistema los tiempos del modelo analítico son mayores a los simulados. Como hemos mencionado antes, esto se debe a que al crecer el número de servidores en el sistema, el flujo de peticiones a vídeos no locales aumenta. En este caso, el modelo reserva parte de los recursos del servidor para realizar los servicios remotos obligatorios, con lo que limita el número de recursos disponibles en la red. Pero el algoritmo RTHL al optimizar el uso de los recursos de la red, no usa todos los recursos reservados por el modelo, con lo que se pueden utilizar para

servir de forma remota peticiones a vídeos locales (es decir, se puede compartir la carga entre los servidores). En otras palabras, nuestro modelo subestima el número de streams disponibles en la red lo que provoca que los tiempos de espera sean ligeramente mayores.

De este conjunto de experimentos hemos aprendido que aun teniendo una replicación pequeña y una tasa elevada de peticiones, nuestro algoritmo RLTH, optimiza el uso de los recursos consiguiendo una calidad de servicio bastante aceptable en términos de tiempo de espera.

Finalmente, debemos comentar otra forma que puede mejorar el rendimiento del sistema, especialmente en el caso de una replicación pequeña. La idea sería intentar influenciar el comportamiento del usuario provocando una mayor demanda demanda de los vídeos replicados, asignándoles a éstos un precio más pequeño. Esto podría conducir a un ahorro en los recursos de la red, y como consecuencia, reducir los tiempos de espera.

3.7. Conclusiones del capítulo

En este capítulo hemos propuesto dos algoritmos multicast (basándonos en los algoritmos MFQL y Multicast con umbral), para planificar tanto los vídeos locales, como los remotos en el sistema distribuido descrito en el Capítulo 1. Aplicamos la estrategia MFQL para seleccionar la cola de las peticiones que se va a planificar y seguimos la estrategia de tipo *patching* con umbral para realizar tanto los servicios locales como los remotos en un sistema distribuido de VoD. Hemos desarrollado un modelo analítico en dos etapas. En la primera etapa desarrollamos y validamos un modelo analítico que captura las principales características de la estrategia *patching* con umbral, aplicada en un solo servidor. En un segundo paso obtenemos el modelo del sistema distribuido.

Nos diferenciamos de anteriores propuestas en que centramos nuestro estudio en el cálculo y optimización del tiempo de espera del usuario, en vez de en el ancho de banda medio. Otra diferencia es que tenemos en cuenta el hecho de que hay un número finito de canales en el servidor, que es una aproximación más realista. Como hemos mostrado en la sección anterior, nuestro modelo ha sido verificado a través de simulaciones, para distintos valores de los parámetros del sistema. Hemos encontrado que nuestro modelo es bastante preciso en sus predicciones numéricas y conclusiones analíticas. A partir de este modelo podemos estimar el rendimiento de un servidor y obtener expresiones para diseñar el sistema de forma que asegure que el rendimiento no se degrade por debajo de un valor fijado.

Además, a través de simulaciones, hemos demostrado que nuestros algoritmos, especialmente RLTH, mejoran significativamente el rendimiento de la estrategia básica (independientemente del porcentaje de replicación y del número de servidores en el sistema). Las mejoras obtenidas en los tiempos de espera son especialmente significativas en los casos del 75 % y del 50 % de replicación. Si

la replicación es del 25 %, la red es el cuello de botella y los tiempos de espera no son tolerables, pero este problema se puede resolver con un pequeño incremento en los recursos de la red.

También hemos visto que con un pequeño incremento en los recursos de la red, podemos optimizar el almacenamiento de los vídeos y conseguir con nuestro algoritmo RLTH, aun con una tasa elevada, un tiempo de espera cercano al minuto.

4 Abandono en el sistema

4.1. Introducción

Para facilitar el modelado del sistema, en los capítulos anteriores hemos supuesto que los clientes siempre estaban dispuestos a esperar para recibir el servicio. Sin embargo, en la realidad puede ocurrir que si un cliente no recibe el vídeo solicitado en un plazo de tiempo, aborte la petición y abandone el sistema.

Como uno de los objetivos de esta tesis es proponer un sistema lo más real posible, en este capítulo proponemos una estrategia de servicio que tiene en cuenta la impaciencia de los usuarios.

Como ya se mencionó en la introducción de esta tesis, un aspecto que se tiene que tener en cuenta en cualquier tipo de sistema orientado al servicio es el de proporcionar una calidad de servicio (QoS) a los usuarios. Entre los distintos parámetros que se utilizan para garantizar unas calidades mínimas, son significativos el tiempo de espera de los clientes y la insatisfacción que se puede generar en éstos si la espera supera un umbral de tolerancia. Si un cliente tarda en recibir un servicio que ha solicitado puede decidir cancelar la petición que realizó abandonando, por tanto, el sistema. Sin una buena planificación de los recursos el sistema puede encontrarse con una elevada tasa de abandono.

A continuación describimos brevemente algunos trabajos previos que tienen en cuenta la impaciencia y, en consecuencia el abandono del sistema, de los usuarios. Gelman y Halfin [58] consideran un sistema en el que los canales del servidor se distribuyen entre los vídeos que ofrece el sistema, y analiza un conjunto de estrategias batching en función de la tasa de abandono de los usuarios. Dan y cols. consideran en [67] que los canales del servidor se comparten entre los vídeos ofertados y estudian la tasa de abandono y el tiempo de espera de los usuarios. Aggarwal y cols. proponen en [28] una estrategia batching (MFQL) que pondera el número de clientes que están en cola con un factor dependiente de la popularidad de los vídeos que se ofertan, de forma que las tasas de abandono de los distintos vídeos sean uniformes. En [68], se estudia minimizar la tasa de abandono dado un número de canales de un servidor y

conocido el comportamiento de los usuarios.

Chan y Tobagi en [69] y [70] hacen un estudio del balance entre coste de ancho de banda usado para servir un vídeo y las ganancias que se consiguen, dependiendo de la estrategia batching que se analiza. En estos análisis se tiene en cuenta el comportamiento impaciente de los usuarios y proponen una estrategia mixta con las mejores cualidades de las estrategias analizadas. Sin embargo, estos estudios suponen que los canales se adjudican conforme son solicitados y que la probabilidad de que una petición no pueda ser atendida por falta de canales es lo suficientemente pequeña como para ser ignorada.

En [46], da Fonseca y Facanha proponen un esquema de planificación que maximiza el número de clientes que se pueden atender con una misma sesión multicast teniendo en cuenta el tiempo de abandono de los clientes que están esperando en cola. Además, analizan los beneficios de integrar una estrategia batching con una de tipo piggybacking en un sistema donde la tasa de peticiones es elevada.

En [71], Gupta y Ammar analizan el rendimiento de varias estrategias multicast en función del tiempo de espera de los clientes y de la tasa de abandono. Basándose en MFQL, proponen una estrategia batching cuyo objetivo es ofrecer tiempos de esperas similares independientemente de la popularidad de los vídeos, y de forma que el abandono por parte de los clientes sea mínimo. Este algoritmo se comporta de forma similar a MFQL. La diferencia es que introduce un factor que obliga a los vídeos más populares a esperar más que los menos populares.

Poon y cols. proponen en [31] y [72] un algoritmo que planifica de forma dinámica la asignación de los canales disponibles para los distintos tipos de vídeo. Si el vídeo es muy popular, se asigna un canal multicast mientras que para los vídeos menos populares se les asigna un canal broadcast. Desarrollan un modelo analítico para seleccionar que tipo de canal se va asignar en función de la popularidad del vídeo solicitado y de la impaciencia de los usuarios. El objetivo de estos trabajos es minimizar el ancho de banda utilizado a la vez que se asegura una mínima calidad de servicio en términos de porcentaje de abandono.

En todos los anteriores trabajos, las estrategias consideradas son de tipo batching. Aunque plantean soluciones donde se combinan batching con otra estrategia de servicio (incluida la estrategia patching), la combinación de batching y patching con umbral no se tiene en cuenta. Además, todos los análisis se realizan suponiendo un sistema compuesto por un único servidor y no se tiene en cuenta que los recursos del sistema son limitados.

En los capítulos anteriores nos hemos centrado en desarrollar unos algoritmos que optimizan los recursos y minimizan el tiempo de espera del cliente. En este capítulo buscamos un algoritmo que mantenga unos tiempos de espera aceptables y, además, mantenga la tasa de abandono en el sistema por debajo de cierto parámetro de calidad. En las siguientes secciones proponemos un algoritmo que combina las estrategias batching y patching con umbral y veremos

que el cálculo del umbral óptimo depende del comportamiento impaciente de los clientes.

La estrategia que proponemos está inspirada en la técnica multicast con umbral RLTH, que presentamos en el capítulo anterior y la hemos denominado *Batching Intervals in a Threshold based Multicast* (BITM). En nuestra aproximación, la idea principal es aplicar la siguiente estrategia a los vídeos más populares: una vez que un stream completo se inicializa, las nuevas peticiones que llegan a continuación no se sirven de forma inmediata, sino que se retrasan unos cuantos segundos para formar un mini-grupo. El mini-grupo de peticiones que se forma se servirá mediante un stream parcial multicast (para transmitir los primeros instantes del vídeo solicitado). De esta forma, los clientes agrupados en un mini-grupo comparten el mismo stream parcial reduciendo así el uso del ancho de banda.

Un punto crítico en nuestra aproximación es la selección del periodo de tiempo para formar los mini-grupos. En la Sección 4.3 determinaremos este periodo con el objetivo de que el número de clientes en los mini-grupos sea lo más grande posible pero de forma que no aumente la probabilidad de abandono. Además, obtenemos el umbral que controla cuándo se inicia un nuevo stream completo que, de hecho, depende del periodo de agrupamiento. El objetivo de este umbral es optimizar el uso del ancho de banda. También en esta sección presentamos un modelo para calcular los parámetros que minimizan el ancho de banda usado y que garantiza que la probabilidad de abandono de los usuarios sea menor que un valor prefijado. En la Sección 4.4 presentamos los resultados de un conjunto de experimentos. El objetivo de estos experimentos es evaluar el comportamiento del algoritmo propuesto en este capítulo. Además, se compara el algoritmo BITM con una versión del algoritmo RLTH donde se tiene en cuenta el abandono de los usuarios para analizar cuál de los algoritmos es mejor implementar en el sistema, dependiendo de la calidad de servicio que se quiera ofrecer, y/o de los recursos disponibles en el sistema.

4.2. El algoritmo BITM

A continuación presentamos las principales características del algoritmo que proponemos y que hemos denotado como *Batching Intervals in a Threshold based Multicast* (BITM).

Esta estrategia se basa en el algoritmo RLTH, y por tanto, es un algoritmo multicast con umbral. Sea U_i' el umbral que controla la transmisión de los streams completos del vídeo i. La idea del algoritmo BITM es crear grupos de usuarios que compartan un mismo recurso. Pero como los clientes pueden abandonar el sistema sin haber recibido el servicio solicitado, el intervalo de tiempo que se les obligue a esperar no puede superar un umbral de tiempo. Por tanto, con este algoritmo se puede optimizar el ancho de banda maximizando el número de clientes que comparten un mismo recurso.

Las características del algoritmo BITM son las siguientes:

1. Suponemos que los clientes pueden impacientarse. Si su tiempo de espera supera un umbral de tolerancia, pueden abandonar el sistema sin recibir el servicio que habían solicitado.

2. Al igual que en el algoritmo RLTH, para planificar las colas de los vídeos se sigue el criterio del algoritmo MFQL [28]. Recordemos que en cada servidor hay una cola por vídeo para almacenar las peticiones hasta que reciben servicio. La planificación de las colas se realiza cada vez que hay usuarios esperando en cola y recursos disponibles. Recordemos, además, que el algoritmo MFQL planifica la cola de vídeos en función de un factor que depende de la cantidad de peticiones que haya en cola y del tiempo que haya transcurrido en ella. La cola que tenga el mayor factor es la que se atiende cuando hay recursos disponibles en el servidor.

3. Una vez que se planifica una cola (para facilitar la explicación, suponemos que se planifica la cola de peticiones al vídeo i), se inicia una sesión multicast para servir a los clientes de la cola. Para ello, el planificador de servicios del servidor reserva un canal regular para transmitir un stream multicast completo del vídeo.

4. Supongamos que después de iniciar una sesión multicast del vídeo i, la siguiente petición a este vídeo que llega al sistema ocurre dentro del umbral U_i'. En ese instante se inicia un intervalo de tiempo de duración t_d segundos durante el cual se hace esperar a todas las peticiones que llegan al vídeo i dentro de ese periodo de tiempo. De esta manera se forma un mini-grupo de peticiones que esperan recibir el servicio solicitado. Una vez transcurrido el intervalo de tiempo t_d segundos, se sirven todas las peticiones que componen el mini-grupo. Para ello, al igual que en el algoritmo RLTH, los clientes se unen inmediatamente a la sesión multicast ya iniciada. Además, el servidor transmite un stream parcial con los primeros instantes del vídeo a través de un canal multicast patching. Hay que hacer notar que la planificación de los mini-grupos tiene prioridad absoluta, es decir, no compiten con otros servicios pendientes cuando se libera un recurso en el sistema.

   Definición 4.1 Al intervalo de tiempo t_d lo denotamos como intervalo de agrupamiento.

5. La siguiente petición del vídeo i que llegue después de atender a un mini-grupo, y que llegue además dentro del umbral U_i', definirá un nuevo mini-grupo que será servido como se ha descrito en el punto anterior. En principio, los componentes de los mini-grupos esperan como mucho t_d segundos salvo en el caso de que no haya recursos disponibles (en el que esperan más de t_d segundos), o si una petición llega en un instante de tiempo dentro del umbral pero que, al sumarle el intervalo de espera t_d para formar un mini-grupo, supere el umbral U_i'. En este caso, sólo se forma un mini-grupo con aquellos clientes que lleguen hasta el umbral.

Este mini-grupo se unirá a la sesión multicast justo U_i' segundos después de que se haya iniciado, y para recibir estos primeros instantes del vídeo, el servidor utilizará de nuevo un canal patching para enviar un stream parcial de duración U_i' segundos.

6. Finalmente, cualquier petición al vídeo i que llegue fuera del umbral, se pone en la cola hasta que se planifique una nueva sesión multicast para el vídeo i (determinada por el algoritmo MFQL).

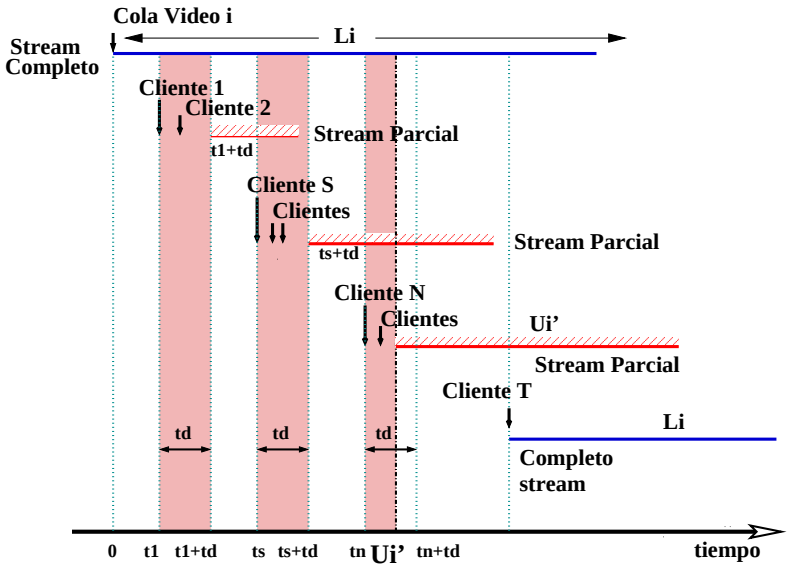


Figura 4.1 Nuestro algoritmo BITM.

Ejemplo 4.1 La Figura 4.1 muestra un ejemplo de cómo se sirven las peticiones según el algoritmo BITM. Supongamos que en el instante $t = 0$ se inicia una sesión multicast para atender las peticiones en cola del vídeo i mediante un stream multicast completo. Después de iniciar la sesión multicast, la primera petición que llega al vídeo i es la que va a definir el primer mini-grupo de la sesión multicast iniciada, y corresponde con la que realiza el *Cliente* 1 en el instante t_1. En ese momento se inicia un intervalo de agrupamiento de duración t_d segundos. Las próximas peticiones que lleguen dentro del intervalo $[t_1, t_1 + t_d]$ (como ocurre con el *Cliente* 2), esperan hasta el instante $t_1 + t_d$. En ese instante, el primer mini-grupo que está compuesto por los usuarios *Cliente* 1 y *Cliente* 2, se unen a la sesión multicast iniciada anteriormente (en el instante $t = 0$). Al mismo tiempo, se inicia un stream parcial multicast (que comparten las peticiones que llegaron en el subintervalo $[t_1, t_1 + t_d]$) para enviar los primeros segundos o minutos del vídeo. El *Cliente S* realiza la primera petición del vídeo i después de que se haya servido el primer mini-grupo. Como esta petición está en el intervalo $(t_1 + t_d, U_i']$, definirá un nuevo mini-grupo que será servido como en el caso del mini-grupo definido por el *Cliente* 1. Sin embargo, cuando el *Cliente N* llega al sistema (que ocurre en el instante t_n), el mini-grupo generado en este caso, estará restringido a los clientes que lleguen dentro del intervalo $[t_n, U_i']$.

Por último, cabe destacar que si suponemos que la tasa de llegadas de peticiones del vídeo i a un servidor es λ_i, entonces el número medio de mini-grupos, n_g, que se forman durante el umbral (ver Figura 4.2) es

$$n_g = \frac{U_i'}{1/\lambda_i + t_d} \tag{4.1}$$

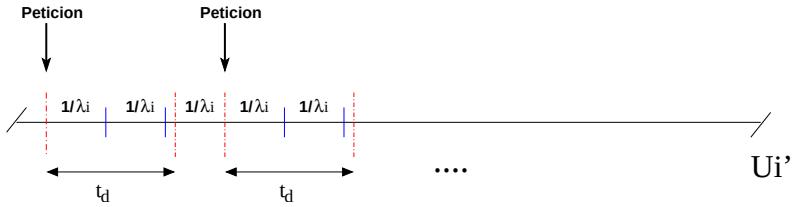


Figura 4.2 Número de mini-grupos.

4.3. Umbral óptimo con el algoritmo BITM

El objetivo del agrupamiento de clientes en mini-grupos es reducir el uso del ancho de banda. Si comparamos los algoritmos RLTH y BITM, podemos ver que, con el primero, cada una de las peticiones que llegan una vez iniciada una sesión multicast se sirven empleando un canal patching exclusivo. Sin embargo, con el segundo utilizamos un único canal patching (que envía un stream parcial multicast) para servir a todas las peticiones que componen un mini-grupo, ahorrándonos por tanto, los canales patching que se utilizarían si el servidor gestionara las peticiones con un algoritmo como RLTH. Es lógico que cuanto mayor sea el intervalo de agrupamiento t_d, habrá más oportunidad de reducir el uso del ancho de banda (se hacen menos streams parciales). El inconveniente lo encontramos en el comportamiento del usuario. Cuanto mayor sea t_d, habrá más usuarios que renuncien al servicio (porque no están dispuestos a esperar tanto) y abandonen el sistema. Luego hay que buscar un compromiso entre la calidad de servicio (que el porcentaje/probabilidad de abandono sea pequeño), y el consumo del ancho de banda.

El objetivo de esta sección es determinar el umbral óptimo U_i' y el máximo intervalo de agrupamiento t_d, con los que se consigan utilizar el mínimo ancho de banda para realizar los servicios de las peticiones que llegan dentro del umbral de una sesión multicast, pero de forma que, la probabilidad de abandono en el intervalo de agrupamiento sea menor que un valor prefijado.

Definición 4.2 El intervalo de agrupamiento t_{max}^{α} es el que verifica que:

- el número de clientes que llegan dentro de él (para formar mini-grupos) sea máximo, pero sin que

- la probabilidad de abandono de estos clientes (que esperan durante el intervalo) sea menor que un valor prefijado α.

Aunque este intervalo depende de α, para simplificar la notación de ahora en adelante lo denotaremos como t_{max}.

A continuación, calculamos t_{max}. Una vez que obtengamos t_{max}, calcularemos el umbral U_i' que minimiza el ancho de banda utilizado. Como veremos, el umbral óptimo U_i' depende de t_{max}.

4.3.1. Cálculo de t_{max}

Como en los capítulos anteriores, suponemos que el tiempo entre las peticiones que llegan a cualquier servidor sigue una distribución exponencial con una tasa λ. Por tanto, las peticiones al vídeo i llegan a cada servidor siguiendo una distribución exponencial de tasa λ_i, donde $\lambda_i = \lambda \cdot p_i$, siendo p_i la probabilidad de solicitar el vídeo i.

Además, suponemos que:

1. Cada cliente admite distintos tiempos de tolerancia a los retrasos. Este comportamiento se puede describir como una secuencia de variables aleatorias mutuamente independientes y con la misma función de densidad $(1/MRT) \cdot e^{-x/MRT}$ donde $1/MRT$ es la tasa de abandono [31], [53]. Por tanto, el comportamiento de abandono del sistema por parte de los usuarios puede modelarse por una distribución exponencial de parámetro $1/MRT$ (o lo que es lo mismo, de media MRT segundos de espera antes de abandonar el sistema).
2. Como los vídeos tienen distintas probabilidades de ser solicitados, tienen distintos consumos del ancho de banda. Por lo tanto, para facilitar el análisis del consumo de los recursos, centramos nuestra discusión en un vídeo particular i de duración L_i segundos y un umbral de U_i' segundos.

Sea $F(x)$ la función de distribución exponencial que define el comportamiento de abandono por parte de los clientes. Suponiendo que el vídeo se sirve al final del intervalo t, sea $P_i(t)$ la fracción de peticiones al vídeo i que abandonan el sistema en un intervalo de tiempo de longitud t. Como el tiempo entre peticiones se distribuye de forma exponencial, las peticiones llegan al sistema siguiendo una distribución de Poisson, es decir, las peticiones llegan de forma uniforme en el intervalo $[0, t]$. Por tanto, la probabilidad de que los clientes que han solicitado el vídeo i abandonen el sistema si tienen que esperar durante un intervalo de tiempo de longitud t, $P_i(t)$, se calcula como

$$P_i(t) = \frac{1}{t} \int_0^t F(x)dx \tag{4.2}$$

Como hemos supuesto que la función de distribución de abandono es exponencial se tiene que,

$$P_i(t) = \frac{1}{t}\int_0^t F(x)dx = 1 - MRT \cdot \frac{1 - e^{-t/MRT}}{t} \tag{4.3}$$

Queremos mantener la probabilidad de abandono, $P_i(t)$, lo suficientemente pequeña, supongamos que más pequeña que un valor fijo α. Por tanto, el intervalo máximo t_{max} que queremos obtener debe verificar que

$$P_i(t_{max}) \leq \alpha \tag{4.4}$$

Aproximando la función $e^{-t/MRT}$ por el polinomio de Taylor de orden 2, tenemos que $P_i(t_{max}) \approx \frac{t_{max}}{2 \cdot MRT} \leq \alpha$; sustituyendo la aproximación en la ecuación 4.4 y despejando de t_{max} se tiene que

$$t_{max} \leq \alpha \cdot 2 \cdot MRT \tag{4.5}$$

El objetivo de implementar intervalos de agrupamientos en el algoritmo de planificación, es dar la oportunidad de formar mini-grupos de peticiones de forma que compartan un mismo stream parcial. Por consiguiente, el intervalo t_{max} que hemos calculado da la oportunidad de que se forme un mini-grupo de peticiones al vídeo i, si se verifica que

$$\lambda_i \cdot t_{max} > 1 \tag{4.6}$$

es decir, llega al menos una petición al vídeo i en el intervalo t_{max}.

De la anterior ecuación podemos observar que sólo los vídeos muy populares (aquellos que verifican la ecuación 4.6), podrán beneficiarse de la optimización de los recursos.

Las ecuaciones anteriores son una herramienta útil para encontrar el compromiso entre el grado de satisfacción de los usuarios (calidad de servicio), y el rendimiento que se desee tener en el sistema. Por un lado, dada una probabilidad de abandono α, la ecuación 4.5 nos da una cota superior de t_{max}. Por otro lado, la ecuación 4.6 determinará los vídeos a los que vamos a aplicar nuestra estrategia, es decir, los vídeos para los que se forman los mini-grupos.

Con estas ecuaciones podemos implementar una estrategia de planificación de los servicios, diseñada para alcanzar el nivel de calidad de servicio deseada, a la vez que se optimizan los recursos (en la medida en la que el parámetro de la calidad de servicio lo permite).

Ejemplo 4.2 Si se quiere que el número de clientes impacientes sea lo más pequeño posible, podemos escoger como probabilidad de abandono $\alpha = 0{,}05$, la cual es lo suficientemente pequeña para considerar insignificante el número de clientes que abandonan el sistema durante t_{max}. Este número de clientes es menor que $\lambda_i \cdot t_{max} \cdot 0{,}05$. Por ejemplo, si $t_{max} = 60$ segundos y $\lambda_i = 0{,}102$ peticiones por segundo, el número de clientes que llegan durante t_{max} es $\lambda_i \cdot t_{max} = 6$, mientras que el número de usuarios que abandonan durante el mismo periodo es menor que 0,306.

4.3.2. Cálculo del umbral óptimo U_i'

Una vez que hemos determinado t_{max}, nos centramos en obtener el umbral óptimo que reduce el uso del ancho de banda. En nuestro análisis, utilizamos el ancho de banda del servidor requerido para servir las peticiones que llegan durante una sesión multicast, y que denotamos como B_i', como parámetro de rendimiento para encontrar el umbral que optimiza el uso de los recursos. De forma similar al estudio realizado en el Capítulo 3, para obtener este umbral óptimo modelamos el sistema como un proceso de renovación [59]. Estamos interesados en el proceso $\{S(t) : t > 0\}$ donde $S(t)$ es el total del ancho de banda del servidor usado desde el instante 0 al instante t. En particular, nos interesa el ancho de banda medio del servidor

$$C = lim_{t\to\infty} S(t)/t \tag{4.7}$$

Sean $\{t_j\}_{j=0}^{\infty}$ $(t_0 = 0)$ los tiempos en los que el sistema planifica un stream completo para cada vídeo i. Estos son puntos de renovación en el sentido de que el comportamiento del sistema para $t \geq t_j$ no depende del comportamiento pasado. Consideramos el proceso $\{S_j, N_j\}$ donde S_j es el ancho de banda usado y N_j es el número total de clientes servidos durante el j-ésimo intervalo de renovación $[t_{j-1}, t_j)$. Como es un proceso de renovación (el comportamiento del sistema no depende del pasado), omitimos el subíndice j y por tanto obtenemos el siguiente resultado

$$B_i' = \lambda_i \cdot \frac{E[S]}{E[N]} \tag{4.8}$$

donde $E[S]$ es la media de ancho de banda usado durante un intervalo de renovación y $E[N]$ el número medio de clientes servidos durante el mismo intervalo.

A continuación vamos a calcular los parámetros $E[S]$ y $E[N]$.

En un sistema sin abandono, el número medio de clientes que son servidos durante el umbral U_i' se calcula como $\lambda_i \cdot U_i'$. Sin embargo, estamos suponiendo un sistema con abandono, luego algunas de las peticiones que lleguen durante el umbral pueden marcharse del sistema sin recibir servicio. Como queremos que la probabilidad de abandono sea lo suficientemente pequeña para que el número de peticiones que abandonen sea insignificante, podemos aproximar $E[N]$ como

$$E[N] \approx \lambda_i \cdot U_i'$$

que representa los clientes que llegarán en el intervalo U_i' y que serán servidos con streams parciales multicast.

Calculamos ahora $E[S]$. Como nuestro sistema tiene recursos finitos, algunas peticiones no podrían ser servidas con streams parciales. Por tanto, en el caso de recursos limitados podemos obtener una cota superior de $E[S]$. Esta cota es

$$E[S] \leq \left(L_i + \sum_{j=1}^{\lfloor n_g' \rfloor} \left(\frac{1}{\lambda_i} + t_{max} \right) \cdot j + U_i' \right) \cdot B_{stream} \tag{4.9}$$

El primer término de la ecuación , $L_i \cdot B_{stream}$, denota el ancho de banda requerido para servir el stream completo iniciado para servir el vídeo i. En el segundo término, el límite superior de la suma $\lfloor n_g' \rfloor = \lfloor \frac{U_i'}{1/\lambda_i + t_{max}} \rfloor$ es el número de mini-grupos que se pueden formar (ver ecuación 4.1) en el intervalo de longitud U_i'. La suma denota todo el ancho de banda requerido por los streams parciales que servirán a los distintos mini-grupos. El tercer término de la ecuación, $U_i' \cdot B_{stream}$, corresponde al ancho de banda requerido por el stream parcial utilizado para servir a los clientes que llegan en un intervalo del tipo $(t_n, U_i']$ del ejemplo 4.1.

Una vez que hemos determinado $E[S]$ y $E[N]$, los reemplazamos en la ecuación 4.8 y obtenemos una cota superior del ancho de banda requerido para servir al vídeo i como,

$$B_i' \leq \frac{L_i + \frac{1}{2} \cdot \frac{U_i' \cdot (U_i' + 1/\lambda_i + t_{max})}{1/\lambda_i + t_{max}} + U_i'}{U_i'} \cdot B_{stream} \tag{4.10}$$

El valor del umbral óptimo, U_i', que minimiza el segundo término de esta expresión es

$$U_i' = \sqrt{2 \cdot L_i \cdot (1/\lambda_i + t_{max})} \tag{4.11}$$

Por tanto el valor correspondiente de la cota inferior del ancho de banda medio utilizado es

$$B_i' \leq \left(\sqrt{\frac{2 \cdot L_i}{1/\lambda_i + t_{max}}} + \frac{3}{2} \right) \cdot B_{stream} \tag{4.12}$$

En este punto, nos gustaría evaluar el comportamiento de nuestra propuesta. Para ello comparamos los algoritmos RLTH considerando abandono (los clientes no son retrasados para formar mini-grupos) y el algoritmo BITM. En el Capítulo 3 hemos obtenido el modelo analítico de este algoritmo. Recordemos que con este esquema el umbral óptimo U_i viene dado por la expresión,

$$U_i = \sqrt{2 \cdot L_i / \lambda_i}$$

y que el ancho de banda medio utilizado B_i viene dada por la expresión,

$$B_i = \left(\sqrt{2 \cdot L_i \cdot \lambda_i} \right) \cdot B_{stream}$$

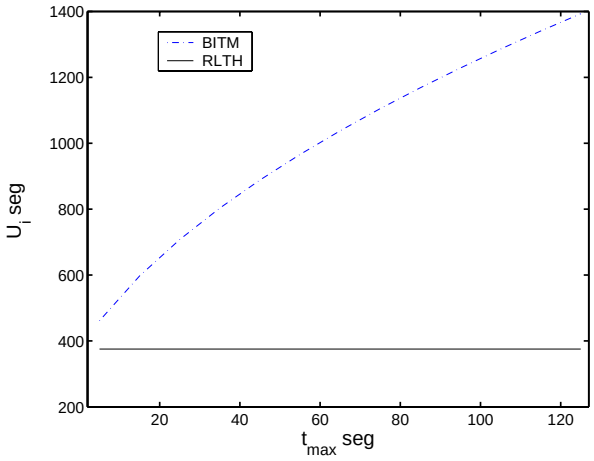


Figura 4.3 Umbral óptimo con el algoritmo BITM y el algoritmo RLTH con abandono. Parámetros: $M = 100$ vídeos, duración media de los vídeos 7200 seg, $\lambda_i = 0{,}102$ pet/seg y $B_{stream} = 15$ Mbps.

Podemos observar que los valores analíticos de U_i' y B_i' obtenidos para el algoritmo BITM dependen del parámetro t_{max}. En las Figuras 4.3 y 4.4 hemos

representado respectivamente el umbral óptimo y la mínima cota superior del ancho de banda usado por ambos algoritmos y para el vídeo más popular entre los 100 vídeos ofrecidos por el sistema. Los parámetros que hemos usado son los mismos que los que usaremos en la Sección 4.4 donde discutimos algunos resultados experimentales. Como podemos observar en la Figura 4.3, el umbral óptimo para BITM es siempre mayor que el umbral óptimo obtenido para RLTH. Cuando t_{max} aumenta, U_i' aumenta y como podemos observar en la Figura 4.4, el ancho de banda requerido por BITM disminuye. En la Figura 4.5 hemos representado la probabilidad de abandonar en un intervalo de longitud $t = t_{max}$ de acuerdo con la ecuación 4.3. Las Figuras 4.4 y 4.5 sugieren que la selección del parámetro t_{max} debe equilibrarse ya que, como podemos observar en la Figura 4.4, el ancho de banda requerido por BITM disminuye con t_{max} y sin embargo, de la Figura 4.5, podemos observar que la probabilidad de abandono aumenta cuando así lo hace t_{max}.

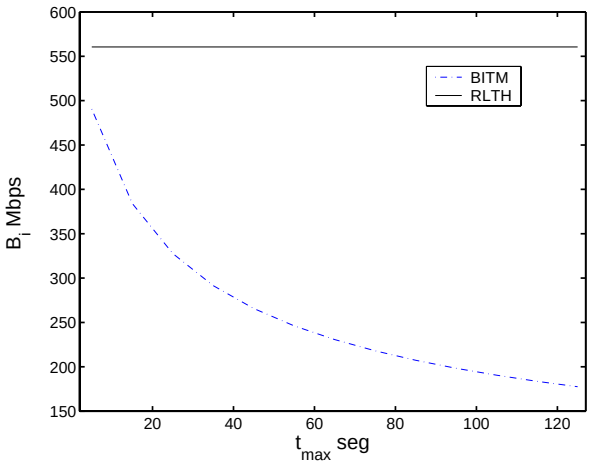


Figura 4.4 Cota inferior del ancho de banda medio utilizado para el vídeo más popular con el algoritmo BITM y el algoritmo RLTH con abandono. Parámetros: $M = 100$ vídeos, duración media de los vídeos 7200 seg, $\lambda_i = 0{,}102$ pet/seg y $B_{stream} = 15$ Mbps.

El ancho de banda total utilizado por los vídeos replicados que se encuentran almacenados en cada servidor está acotado por el número de streams que se pueden servir a la vez en un servidor local, es decir,

$$\sum_{i=1}^{s} B_i' \leq \sum_{i=1}^{s} \left(\sqrt{\frac{2 \cdot L_i}{1/\lambda_i + t_{max}}} + \frac{3}{2} \right) \cdot B_{stream} \leq N_{stream} \cdot B_{stream} \qquad (4.13)$$

Por otro lado, el ancho de banda de la red de conexión utilizado por los servicios remotos está acotado por

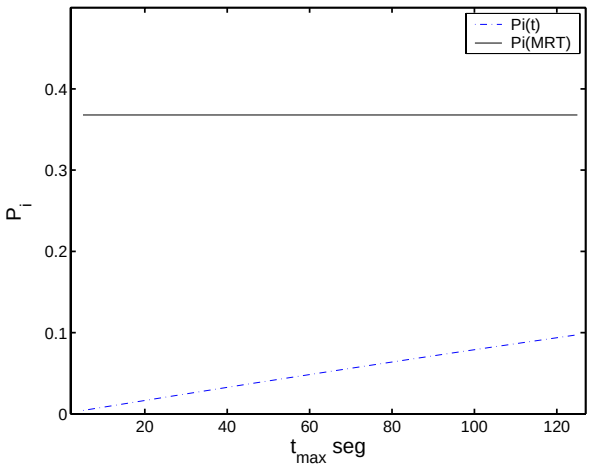


Figura 4.5 Probabilidad de abandono $P_i(t)$: fracción de peticiones al vídeo i que abandonan el sistema en un intervalo de longitud t.

$$\sum_{i=s+1}^{M} B'_i \leq \sum_{i=s+1}^{M} \left(\sqrt{\frac{2 \cdot L_i}{1/\lambda_i + t_{max}}} + \frac{3}{2} \right) \cdot B_{stream} \leq min(B_{link}, \frac{B_{switch}}{N_{serv}}) \quad (4.14)$$

De las ecuaciones 4.11, 4.12, 4.13 y 4.14 obtenemos una relación entre los parámetros del sistema (B_{stream}, B_{link}, B_{switch}, N_{serv}), los parámetros del algoritmo (U'_i, t_{max}) y el comportamiento del usuario (λ_i). Estas ecuaciones se pueden utilizar como guías para redimensionar tanto el tamaño de la red como los servidores.

4.4. Resultados experimentales

En esta sección, evaluamos el rendimiento que tiene el algoritmo BITM en el sistema distribuido propuesto en nuestro trabajo. Este rendimiento lo medimos a través de dos métricas relacionadas con la Calidad de Servicio de nuestro sistema: la probabilidad de abandono, que medimos como el porcentaje de clientes que abandonan el sistema antes de recibir servicio, y el tiempo de espera (T_{wait}). Los tiempos de espera de las peticiones que abandonan no se consideran a la hora de calcular T_{wait}. Por otro lado, recordemos que sólo consideramos para formar mini-grupos a los vídeos más populares, que son aquellos que verifican la ecuación 4.6.

En estos experimentos comparamos nuestro algoritmo BITM con una versión del algoritmo RLTH, donde las peticiones no se retrasan para formar mini-grupos, pero donde los usuarios sí pueden abandonar el sistema.

4.4.1. Parámetros de las simulaciones

En los siguientes experimentos suponemos que cada uno de los servidores del sistema tiene conectados $N_{term} = 3000$ clientes. Hemos elegido este número de clientes ya que, a través de numerosas simulaciones, hemos visto que con este valor, al menos los seis vídeos más populares verifican la ecuación 4.6. Como estamos suponiendo que algunos clientes pueden abandonar el sistema si no reciben el servicio, suponemos que cada cliente genera peticiones siguiendo una distribución exponencial con una media de parámetro $T_{sleep} = 3600$ segundos, lo que va a generar un mayor número de peticiones al sistema. Un cliente abonado a un sistema de vídeo bajo demanda puede estar acostumbrado a que, en ocasiones, tenga que esperar al menos un poco para recibir el servicio solicitado. Dependiendo del usuario, el umbral de tolerancia de esta espera puede variar desde un minuto a cerca de veinte minutos. Pensando en una población que no sea ni muy impaciente ni muy benévola, podemos suponer que de media los usuarios esperan como mucho diez minutos antes de abandonar el sistema. Por tanto, fijamos el parámetro MRT (recordemos que es la media de tiempo de espera antes abandonar) a 600 segundos. Como queremos que la probabilidad de abandono de los clientes que integran un mini-grupo sea pequeña, fijamos el parámetro α a 0,05. Por tanto, de las ecuaciones 4.3 y 4.5, el máximo retraso que pueden sufrir los clientes que solicitan los vídeos más populares, es $t_{max} = 60$ segundos.

El sistema ofrece $M = 100$ vídeos cuyas duraciones siguen una distribución uniforme en el intervalo de tiempo de [3600,10800] segundos. Estos vídeos se distribuyen en el sistema como se explicó en el Capítulo 1. Para modelar la popularidad de los vídeos, utilizamos la ley de Zipf [43], [28] con un parámetro $\xi = 1$ (que es lo que se conoce como ley de Zipf pura). El comportamiento de los algoritmos lo evaluamos a través de dos conjuntos de experimentos. En el primero de ellos suponemos que cada servidor tiene una capacidad de servicio de $N_{stream} = 50$ streams y que el porcentaje de replicación varía entre el 75 %, el 50 % y el 25 %. Con este conjunto de experimentos veremos cómo influye el porcentaje de replicación en la calidad del servicio. En el segundo conjunto de experimentos, variamos el parámetro N_{stream} de 150, 200 a 250 streams mientras que el porcentaje de replicación lo fijamos al 25 %. El objetivo de este segundo conjunto de experimentos es estudiar el comportamiento de los algoritmos cuando se incrementan los recursos en el sistema.

Para cada experimento hemos supuesto que el sistema está en un periodo de *prime time*, es decir, en horario de máxima audiencia, con lo que el tiempo del sistema simulado es aproximadamente de 4 horas.

4.4.2. Análisis de los resultados

En esta sección vamos a analizar, por un lado, la influencia del porcentaje de replicación tanto en los tiempos de espera como en el abandono de los usuarios. Por otro lado, estudiaremos el comportamiento de los algoritmos

cuando fijamos el porcentaje de replicación y aumenta la capacidad de servicio.

Influencia del porcentaje de replicación

En la Figura 4.6 comparamos los tiempos de espera para los distintos porcentajes de replicación. El eje X representa el número de servidores en el sistema, N_{serv}, y el eje Y representa el tiempo medio de espera (T_{wait}) expresado en segundos.

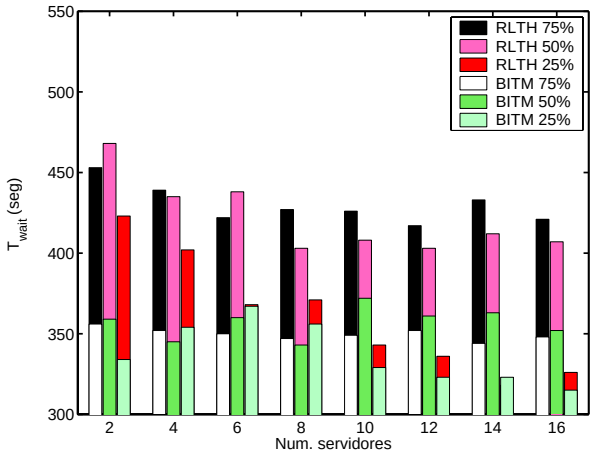


Figura 4.6 Tiempos de espera del algoritmo BITM y de la versión del algoritmo RLTH con abandono cuando cada servidor tiene una capacidad de servicio de $N_{stream} = 50$ streams. El eje X representa el número de servidores en el sistema, y el eje Y representa el tiempo de espera en segundos.

En la Tabla 4.1 hemos representado el porcentaje de abandono de los vídeos más populares, es decir, de aquellos vídeos que verifican la ecuación 4.6 (que con los parámetros escogidos se trata de los seis primeros vídeos) y que denotamos como los Más Populares (**M.P.**) así como el porcentaje de abandono debido a las peticiones remotas (es decir, debido a los vídeos No Replicados, **N.R.**). La forma de obtener estos porcentajes es como sigue: para cada vídeo se calcula el número de peticiones que recibe el sistema y el número de éstas que dejan el sistema sin recibir servicio. El porcentaje de abandono de los vídeos más populares se calcula como la suma del número de peticiones a los vídeos que verifican la ecuación 4.6 que han abandonado el sistema entre el número total de peticiones (a todos los vídeos) que ha recibido el sistema. El porcentaje de abandono debido a las peticiones remotas se calcula como la fracción entre la suma de las peticiones a los vídeos que no están replicados en todos los servidores que han abandonado el sistema y el número total de peticiones (a todos los vídeos) que ha recibido el sistema. Ambos porcentajes se han calculado para los distintos sistemas y porcentajes de replicación.

En la Figura 4.6 podemos ver que en casi todos los casos los tiempos de espera de BITM son siempre los más pequeños. Esto es porque algunas peticiones

de los vídeos más populares se ven obligadas a esperar para formar los mini-grupos, y así ser servidas con un mismo stream parcial. Este hecho hace que el ancho de banda utilizado por el algoritmo BITM sea menor que el empleado por el algoritmo RLTH. Luego, ésto confirma que nuestro algoritmo BITM reduce el uso de los recursos en el caso de que éstos sean muy limitados. Además, podemos observar que T_{wait} es casi constante cualquiera que sea el porcentaje de replicación. Es sólo con un 25 % de replicación que T_{wait} disminuye y lo hace a partir de 10 servidores en el sistema. Esto se debe a dos razones.

La primera es que para el 25 % de replicación, hay un mayor abandono de las peticiones a los vídeos no replicados (como se puede observar en la Tabla 4.1). Como hay menos peticiones remotas que efectivamente serán servidas, hay menos competencia por la red de interconexión. Por tanto, las peticiones remotas que no abandonan el sistema esperan menos tiempo.

La segunda razón está relacionada con la distribución de los vídeos que hemos supuesto. Con esta distribución, si un sistema tiene un número pequeño de servidores (2, 4, 6 servidores), éstos tienen que almacenar más vídeos no replicados que los servidores de un sistema con un mayor número de servidores (10, 12, 14, 16 servidores). En consecuencia, los servidores de un sistema compuesto por un gran número de servidores, almacenan menos vídeos. Por tanto, estos servidores no tienen que compartir su ancho de banda local entre tantos vídeos. Como hay menos competencia por el ancho de banda local, los clientes que no abandonan el sistema esperan menos que los clientes (en la misma situación) de un sistema con menos servidores.

De la Tabla 4.1 podemos observar que el porcentaje de abandono debido a los vídeos no replicados **N.R.** aumenta cuando el porcentaje de replicación disminuye. Con un porcentaje de replicación bajo (por ejemplo el 25 %) un sistema tiene que administrar más servicios remotos, luego existe una competitividad por los recursos de la red. Por tanto, los clientes tienen que esperar necesariamente para ser servidos. Pero cuantos más servicios remotos, más tienen que esperar los usuarios y, como consecuencia, el porcentaje de abandono aumenta. Por otro lado, de los resultados representados en la Tabla 4.1 podemos observar que para ambos algoritmos el porcentaje de abandono de peticiones a vídeos locales permanece constante cualquiera que sea el porcentaje de replicación en el sistema. De la misma forma podemos observar que fijado un porcentaje de replicación, los porcentajes de abandono son igualmente constantes independientemente del número de servidores que tenga el sistema. Es decir, el número de servidores en el sistema tiene poca influencia en el abandono, a diferencia del parámetro T_{wait}.

De la Tabla 4.1 también podemos observar que con BITM, los clientes que solicitan vídeos locales abandonan menos que si fueran planificados con el algoritmo RLTH. Es decir, retrasar unos cuantos segundos los clientes para formar mini-grupos, hace que se optimice el ancho de banda local y, por tanto, se sirven más clientes y se abandona menos.

Nº Serv.	75 %		50 %		25 %		Porcent. Abandono
	RLTH	BITM	RLTH	BITM	RLTH	BITM	
	0.26	0.18	0.24	0.19	0.25	0.19	**M. P.**
2	0.03	0.04	0.1	0.1	0.2	0.21	**N. R.**
	0.24	0.18	0.24	0.19	0.24	0.19	**M.P.**
4	0.04	0.04	0.1	0.11	0.2	0.22	**N.R.**
	0.24	0.18	0.24	0.19	0.24	0.18	**M.P.**
6	0.04	0.04	0.1	0.11	0.23	0.23	**N.R.**
	0.21	0.19	0.24	0.19	0.24	0.19	**M.P.**
8	0.04	0.04	0.1	0.11	0.23	0.24	**N.R.**
	0.24	0.19	0.25	0.19	0.24	0.17	**M.P.**
10	0.04	0.04	0.11	0.11	0.23	0.24	**N.R.**
	0.24	0.19	0.24	0.19	0.24	0.18	**M.P.**
12	0.04	0.04	0.11	0.11	0.23	0.24	**N.R.**
	0.23	0.19	0.24	0.19	0.23	0.17	**M.P.**
14	0.04	0.04	0.11	0.11	0.23	0.24	**N.R.**
	0.23	0.19	0.24	0.19	0.24	0.16	**M.P.**
16	0.04	0.04	0.11	0.11	0.23	0.24	**N.R.**

Tabla 4.1 Porcentaje de abandono del algoritmo BITM y de la versión del algoritmo RLTH con abandono: las filas impares representan el porcentaje de abandono de los vídeos más populares (**M.P.**) y las filas pares el porcentaje de abandono de los vídeos no replicados (**N.R.**). Las dos primeras columnas representan los porcentajes de abandono con un 75 % de replicación en el sistema; Las dos siguientes, los porcentajes con un 50 %; Las dos últimas, los correspondientes porcentajes de abandono con un 25 % de replicación en el sistema.

Comportamiento de BITM

A continuación vamos a analizar el comportamiento de los algoritmos BITM y RLTH cuando se aumenta la capacidad de servicio en los servidores. Del estudio anterior hemos visto que si el sistema tiene un porcentaje de replicación pequeño (del 25 %), los tiempos de espera que se obtienen son los más pequeños, mientras que el porcentaje de abandono de los vídeos más populares es similar a los obtenidos con los demás porcentajes de replicación pero sin embargo, aumenta de manera importante el porcentaje de abandono para los vídeos no replicados. Por tanto, para este análisis nos centramos en un sistema con un porcentaje de replicación del 25 %.

En las Figuras 4.7, 4.8, 4.9 comparamos los tiempos de espera cuando varía la capacidad de servicio de $N_{stream} = 150$, $N_{stream} = 200$ a $N_{stream} = 250$ streams. El eje X representa el número de servidores en el sistema, N_{serv}, y el eje Y representa el tiempo medio de espera expresado en segundos. Como podemos observar de las figuras, se tiene que conforme aumenta la capacidad de servicio los tiempos de espera disminuyen (lo que es bastante lógico). También podemos observar que los tiempos de espera del algoritmo BITM son menores que los obtenidos con el algoritmo RLTH, si la capacidad de servicio en cada servidor es menor de 200 streams. Sin embargo, cuando $N_{stream} = 250$ los

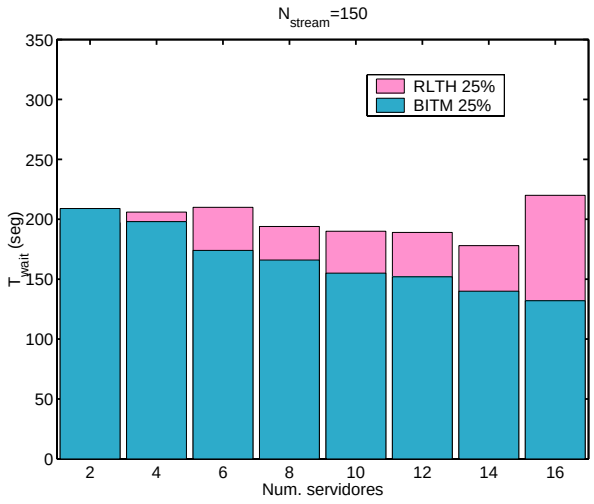


Figura 4.7 Tiempos de espera del algoritmo BITM y de la versión del algoritmo RLTH con abandono cuando cada servidor tiene una capacidad de servicio de $N_{stream} = 150$ streams. El eje X representa el número de servidores en el sistema, y el eje Y representa el tiempo de espera en segundos. El porcentaje de replicación es del 25 %.

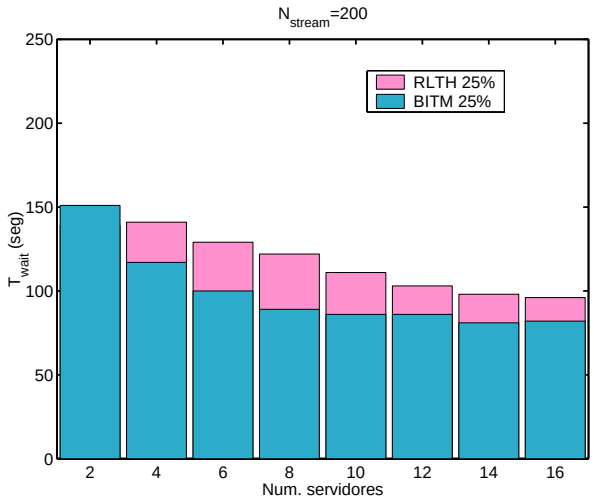


Figura 4.8 Tiempos de espera del algoritmo BITM y de la versión del algoritmo RLTH con abandono cuando cada servidor tiene una capacidad de servicio de $N_{stream} = 200$ streams. El eje X representa el número de servidores en el sistema, y el eje Y representa el tiempo de espera en segundos. El porcentaje de replicación es del 25 %.

papeles se invierten. Esto es debido que el sistema tiene más canales de servicio que los necesitados para atender la demanda que recibe. Por tanto, no le hace falta optimizar los recursos. En el caso del algoritmo BITM, se está obligando a esperar a usuarios (para formar los mini-grupos) que podrían haberse servido

de forma inmediata. Es precisamente esta espera la que aparece reflejada en los resultados (Figura 4.9), ya que como podemos ver, los tiempos de espera son del orden del intervalo de agrupamiento $t_{max} = 60$ segundos. Una consecuencia de tener suficiente capacidad de servicio para atender la demanda, es que con el algoritmo RLTH se produce menos abandono de los vídeos más populares que con el algoritmo BITM. Esto se puede apreciar en la Tabla 4.2, donde para el caso de $N_{stream} = 250$ streams, el porcentaje de abandono de los vídeos más populares tiende a 0 cuando las peticiones se planifican con el algoritmo RLTH.

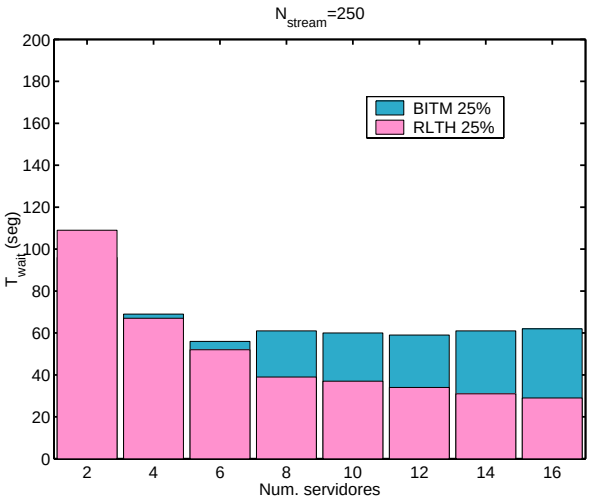


Figura 4.9 Tiempos de espera del algoritmo BITM y de la versión del algoritmo RLTH con abandono cuando cada servidor tiene una capacidad de servicio de $N_{stream} = 250$ streams. El eje X representa el número de servidores en el sistema, y el eje Y representa el tiempo de espera en segundos. El porcentaje de replicación es del 25 %.

En la Tabla 4.2 hemos representado el porcentaje de abandono de los vídeos más populares (los que verifican la ecuación 4.6) y que denotamos como los Más Populares (**M.P.**) así como la porcentaje de abandono debido a las peticiones remotas (es decir, debido a los vídeos No Replicados, **N.R.**).

Como podemos observar, los porcentajes de abandono de los vídeos más populares van disminuyendo conforme aumenta el número de streams disponibles en el sistema. Esta disminución es mayor con el algoritmo RLTH lo que, como se mencionó anteriormente, es debido al hecho de que este algoritmo no obliga a algunos clientes a esperar para formar mini-grupos. Como hay más recursos, se atienden más peticiones de forma inmediata, y por tanto, se produce menos abandono.

En cuanto al porcentaje de abandono de los vídeos no replicados, podemos observar que son similares. Esto es debido a que ambos algoritmos planifican las peticiones a vídeos no replicados de la misma forma.

Hay que señalar que para todos los experimentos realizados verificamos que la probabilidad de abandono de los clientes que integran un mini-grupo era del

Nº Serv.	150 st.		200 st.		250 st.		Porcent. Abandono
	RLTH	BITM	RLTH	BITM	RLTH	BITM	
	0.34	0.39	0.28	0.38	0.17	0.36	**M. P.**
2	0.22	0.2	0.2	0.16	0.15	0.13	**N. R.**
	0.34	0.38	0.24	0.35	0.09	0.34	**M.P.**
4	0.21	0.19	0.18	0.17	0.15	0.15	**N.R.**
	0.33	0.38	0.19	0.35	0.03	0.34	**M.P.**
6	0.2	0.2	0.18	0.17	0.15	0.19	**N.R.**
	0.3	0.35	0.16	0.36	0.02	0.33	**M.P.**
8	0.21	0.2	0.18	0.18	0.16	0.2	**N.R.**
	0.3	0.37	0.14	0.34	0.01	0.33	**M.P.**
10	0.21	0.2	0.18	0.18	0.17	0.21	**N.R.**
	0.3	0.35	0.13	0.35	0	0.35	**M.P.**
12	0.21	0.2	0.19	0.18	0.17	0.21	**N.R.**
	0.3	0.35	0.12	0.34	0	0.34	**M.P.**
14	0.21	0.2	0.19	0.19	0.17	0.21	**N.R.**
	0.3	0.36	0.11	0.35	0	0.34	**M.P.**
16	0.21	0.2	0.19	0.190	0.18	0.21	**N.R.**

Tabla 4.2 Porcentaje de abandono del algoritmo BITM y de la versión del algoritmo RLTH con abandono: las filas impares representan el porcentaje de abandono de los vídeos más populares (**M.P.**) y las filas pares el porcentaje de abandono de los vídeos no replicados (**N.R.**). Las dos primeras columnas representan los porcentajes de abandono con $N_{stream} = 150$; Las dos siguientes, con $N_{stream} = 200$; Las dos últimas, los correspondientes porcentajes de abandono con $N_{stream} = 250$.

orden prefijado, es decir, se verifica que la probabilidad de abandono de los clientes es alrededor de $\alpha = 0{,}05$.

De los resultados anteriores podemos concluir que, dependiendo de la calidad de servicio que se quiera ofrecer, y/o de los recursos disponibles en el sistema, sería preferible utilizar la estrategia BITM o la estrategia de servicio RLTH.

Si por cuestiones económicas el sistema debe ser diseñado contando con poca capacidad de servicio, desde el punto de vista del comportamiento del abandono, la opción más atractiva sería diseñar un sistema distribuido de VoD basado en el mayor número de servidores que verifiquen la restricción expresada con la ecuación 4.14, y usar un porcentaje de replicación pequeño. En este sistema, para conseguir los mejores resultados posibles, habría que planificar las peticiones utilizando el algoritmo BITM. Sin embargo, debemos considerar el compensar la selección del porcentaje de replicación, porque como vimos en la Tabla 4.1 cuanto menos replicación, más abandono de las peticiones a vídeos no replicados.

Sin embargo, si se puede invertir en ancho de banda, se debe estudiar cuál de las estrategias de servicio produce menos tiempo de espera, y/o porcentaje de abandono.

4.5. Conclusiones del capítulo

En este capítulo hemos propuesto una estrategia de servicio para un sistema distribuido de Vídeo bajo Demanda que optimiza el uso de los recursos. A través de un estudio analítico y experimental del sistema, hemos visto que retrasar algunos usuarios durante un corto intervalo de tiempo conduce a la optimización del uso de los recursos, mientras que se mantiene un valor prefijado de probabilidad de abandono. Además, hemos visto que cuando los recursos del sistema son limitados, BITM reduce tanto las probabilidades de abandono como los tiempos de espera T_{wait} y, además, mejora los resultados obtenidos por el algoritmo propuesto en el capítulo anterior (RLTH con abandono). En otras palabras, BITM mejora la calidad de servicio de un sistema con pocos recursos.

Conclusiones y principales aportaciones

Para finalizar esta memoria, a continuación se incluye un resumen de las principales aportaciones de nuestro trabajo junto con una descripción de las futuras líneas de investigación.

La motivación de este trabajo se ha basado en el estudio y diseño de un sistema de VoD - *Video on Demand* o Vídeo bajo Demanda- que sea escalable y económico. Muchas de las consideraciones realizadas en este estudio son válidas para otro tipo de sistemas que suministran contenidos multimedia bajo demanda (MoD). De entre las distintas arquitecturas de sistemas de VoD que se han propuesto en la última década, se ha comprobado que es mejor una arquitectura distribuida a una centralizada debido sobre todo a cuestiones de escalabilidad y de tolerancia a fallos. Además, cuando se quiere dar servicio a una gran población de usuarios es económicamente viable dividir ésta en subáreas, de forma que sean atendidas por medio de un nodo de servicio. En la mayoría de las arquitecturas propuestas, estos nodos son grandes servidores. Como tienen que dar servicio a cientos de usuarios, estos servidores tienen que suministrar un gran ancho de banda y soportar un gran sistema de almacenamiento (para almacenar todos los vídeos que ofertan), con lo que al final el sistema sale bastante costoso. Además, al ser un único servidor, en caso de fallo se tiene que redirigir toda la población a otros nodos del sistema, con el consiguiente aumento de tráfico a través de la red de interconexión.

Con todo esto, es natural plantearse una alternativa de diseño de estos nodos de servicio. Nosotros proponemos que la arquitectura de estos nodos sea también un sistema distribuido. En este trabajo hemos propuesto un sistema distribuido de servidores de pequeña capacidad y de bajo coste donde los vídeos más populares están almacenados en todos los servidores.

Por otro lado, a la hora de diseñar un sistema hay que tener en cuenta todos los factores que están implicados en el funcionamiento de un sistema de VoD: comportamiento de los usuarios, tipo de servicios así como calidad de servicio que se va a ofrecer, o planificación de los recursos entre otros. En concreto, en esta tesis nos hemos centrado sobre todo en el estudio de algoritmos de planificación de los recursos que optimicen algún parámetro de calidad del sistema distribuido. Para ello es necesario disponer de algún modelo analítico

del sistema. Sin embargo, casi todas las herramientas analíticas que se han desarrollado, han tenido como objetivo optimizar parámetros concretos del sistema y no en obtener unas expresiones con las que se conozca por adelantado el rendimiento del sistema, qué parámetros se van a convertir en los cuellos de botella y cuándo, etc. En este trabajo hemos desarrollado una herramienta matemática que es útil para que el diseñador tenga unas guías con las que poder dimensionar el sistema. De esta forma, el diseñador del sistema puede encontrar un equilibrio entre los parámetros de calidad que se quiera ofrecer al usuario final y las ganancias que el sistema genere.

Más detalladamente, las principales aportaciones se exponen como sigue:

- Hemos propuesto un sistema con replicación parcial, es decir, un sistema en el que todos los servidores almacenen los vídeos más populares (donde la popularidad se define como el porcentaje de peticiones a un determinado vídeo) frente a la mayoría de las aproximaciones que consideran la replicación total de los vídeos en todos los servidores. Hemos estudiado que la popularidad de los vídeos permite que el sistema pueda ahorrar costes en almacenamiento, ya que una gran cantidad de las peticiones que se generan en el sistema es debida a los vídeos más populares.

- Hemos desarrollado un algoritmo de planificación de los recursos de tipo unicast -para ofrecer interactividad- llamado PRLS, que permite la compartición de la carga en caso de que algún servidor esté sobrecargado (*load sharing*). Este algoritmo tiene en cuenta tanto el comportamiento de los usuarios (tasa de peticiones, tiempo de servicio), la popularidad de los vídeos, así como el porcentaje de replicación de los vídeos en el sistema. Además hemos desarrollado este algoritmo de forma que los recursos -en nuestro caso, el ancho de banda de los servidores y de la red- sean optimizados al máximo. Hemos comprobado que la calidad de servicio, en términos de tiempo de espera, no se degrada demasiado cuando el sistema distribuido pasa de tener una replicación del 100 % (es decir, todos los vídeos están almacenados en todos los servidores) a tener una replicación del 50 %.

- Hemos desarrollado un modelo analítico que captura las características claves de este algoritmo. Este modelo analítico nos permite estimar para distintos porcentajes de replicación y grados de popularidad, algunas características básicas del comportamiento de nuestro algoritmo PRLS. Estas estimaciones nos pueden permitir seleccionar el ancho de banda y la capacidad de almacenamiento de un servidor, así como el tamaño de la red y el número óptimo de servidores para mantener un tiempo de espera pequeño y predecir cuándo la red se convierte en el cuello de botella.

- Por otro lado, en el caso en el que se prime las ganancias del sistema se puede optar por un algoritmo de planificación de tipo en el que se implemente una estrategia multicast. Este tipo de propuesta es adecuada si se tiene un sistema de almacenamiento pequeño (donde el porcentaje de replicación sea inferior al 50 %) o si se quiere optimizar al máximo el

uso de los recursos, es decir, si se prefiere servir más usuarios con un mismo canal que ofrecer servicios interactivos. Para ello, hemos propuesto dos algoritmos multicast basados en las estrategias batching y patching con umbral, que hemos llamado LTH y RLTH. Estos algoritmos han sido diseñados específicamente para planificar de forma eficiente todas las peticiones (locales como remotas) del sistema distribuido de VoD propuesto, es decir, han sido diseñados para optimizar el uso de los recursos de los servidores y la red, y de forma que los tiempos de espera sean pequeños.

- Además, hemos demostrado que nuestros algoritmos, especialmente el algoritmo RLTH, mejoran significativamente el rendimiento de una estrategia de tipo patching con umbral (independientemente del porcentaje de replicación y del número de servidores en el sistema). Las mejoras obtenidas en los tiempos de espera son especialmente significativas en los casos del 75 % y del 50 % de replicación.

- Para capturar las características de los algoritmos propuestos hemos desarrollado un modelo analítico en dos etapas. Nuestra aportación es que nos diferenciamos de anteriores propuestas en que centramos nuestro estudio en el cálculo y optimización del tiempo de espera del usuario, en vez de en el ancho de banda medio. Otra diferencia es que tenemos en cuenta el hecho de que hay un número finito de canales en el servidor frente a las estrategias previas que suponían un número ilimitado de canales, lo que supone que nuestra aproximación es más realista.

  Este modelo se ha desarrollado en dos etapas. En la primera etapa desarrollamos un modelo analítico que captura las principales características de la estrategia patching con umbral, aplicada en un solo servidor. A partir de este modelo podemos estimar el rendimiento de un servidor y obtener expresiones para diseñarlo de forma que asegure que el tiempo de respuesta no se degrade por debajo de un valor fijado. En un segundo paso hemos obtenido el modelo del sistema distribuido. Hemos encontrado que nuestro modelo es bastante preciso en sus predicciones numéricas y conclusiones analíticas. Las principales aportaciones del modelo son que da dos funciones:

  1. Puede estimar el rendimiento de un servidor sea cual sea su diseño (teniendo en cuenta el comportamiento del usuario) en cuestión de segundos (comparado con las simulaciones que podrían tardar horas).
  2. Puede dar expresiones para diseñar el sistema de forma que asegure que el tiempo de respuesta no se degrade por debajo de un valor fijado, cuando varían algunos parámetros del sistema.

- Hemos propuesto un nuevo algoritmo que planifica los recursos llamado BITM y que tiene en cuenta la impaciencia de los usuarios de un sistema distribuido. En este algoritmo se han introducido dos tipos de umbrales. El primer tipo de umbral genera un subintervalo de tiempo

durante el cual se agrupan a los clientes que vayan solicitando un mismo vídeo. Una vez alcanzado el final del subintervalo generado, todas las peticiones agrupadas se sirven utilizando un mismo canal. El segundo tipo de umbral sirve para controlar cuántas veces se generan los subintervalos durante la emisión de un vídeo. El objetivo de este algoritmo ha sido no sólo reducir el uso del ancho de banda mientras se consigue un porcentaje de abandono inferior a un valor prefijado, sino que mantiene e incluso mejora los tiempos de espera de los clientes de un sistema con un porcentaje de replicación pequeño y con escasos recursos cuando se compara con una estrategia de tipo patching con umbral. Nuestra propuesta permite un análisis más detallado del sistema ya que la mayoría de las aproximaciones realizadas se han centrado sobre todo en balancear el coste de ancho de banda usado para servir un vídeo y las ganancias que se consiguen, o bien sólo en asegurar una mínima calidad de servicio en términos de porcentaje de abandono.

Como líneas de trabajo futuras, creemos que serían interesantes los siguientes trabajos:

- Estudiar el impacto que nuestros algoritmos puedan tener en otras topologías distribuidas.
- Estudio del comportamiento de nuestros algoritmos en una topología híbrida (conectando grupos de servidores descentralizados a través de una jerarquía de servidores). Este análisis puede darnos más elementos de juicio para diseñar un sistema distribuido de VoD eficiente, económicamente viable y escalable.
- Realizar un modelo analítico del sistema teniendo en cuenta la impaciencia de los usuarios. Es decir, desarrollar la herramienta matemática que nos permita dimensionar el sistema sin necesidad de recurrir a las simulaciones.

Glosario

Glosario

ADSL	*Asimetric Digital Subscriber Line*
ATM	*Asynchronous Transfer Mode*
BITM	*Batching Intervals in a Threshold based Multicast*
DVD	*Digital Versatile Disc*
GWQ	*Global Wait Queue*
HDTV	*High Definition Television*
IP	*Internet Protocol*
LTH	*Local Threshold*
MP3	*MPEG-1 Layer 3*
MPEG	*Moving Picture Experts Group*
N-VoD	*Near Video on Demand*
PPV	*Pay-per-view*
PRLS	*Popularity and Partial Replication Load Sharing*
QoS	*Quality of Service*
Q-VoD	*Quasi Video on Demand*
RM	Replicación Moderada
RLTH	*Remote and Local Threshold*
RTCP	*Real Time Control Protocol*
RTP	*Real Time Protocol*
RTSP	*Real Time Streaming Protocol*
SMS	*Short Message Service*
STB	*Set-top-box*
T-VoD	*True Video on Demand*
VCR	*Video Control Remote*
VoD	*Video on Demand*

Bibliografía

Bibliografía

[1] S. González, A. Navarro, J. Lopez, and E. Zapata, "Planificación de la carga en sistemas distribuidos de vídeo bajo demanda (vod)," in *XII Jornadas de Paralelismo*, Valencia, España, 3-4 Septiembre 2001, pp. 93–98.

[2] S. González, A. Navarro, J. Lopez, and E. Zapata, "Load sharing in distributed vod (video on demand) systems," in *International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet*, L'Aquila, Italy, 21-27 January 2002.

[3] S. González, A. Navarro, J. Lopez, and E. Zapata, "Load sharing based on popularity in distributed video on demand systems," in *IEEE International Conference on Multimedia*, Lausanne, Switzerland, 26-29 August 2002.

[4] S. González, A. Navarro, J. Lopez, and E. Zapata, "Two hybrid multicast algorithms for optimizing resources in a distributed vod system," in *10th International Multi-Media Modelling Conference*, Brisbane, Australia, 5-7 January 2004.

[5] S. González, A. Navarro, J. Lopez, and E. Zapata, "A batching driven reneging technique in a distributed vod system," in *XV Jornadas de Paralelismo. Computación de altas prestaciones*, Almería, España, 15-17 Septiembre 2004, pp. 465–470.

[6] S. González, A. Navarro, J. Lopez, and E. Zapata, "A threshold based multicast technique in a distributed vod system with customer reneging behavior," in *Tenth International Conference on Distributed Multimedia Systems*, San Francisco, CA, 8-10 September 2004, pp. 145–150.

[7] S. González, A. Navarro, J. Lopez, and E. Zapata, "A case study of load sharing based on popularity in distributed vod systems," *IEEE Transactions on Multimedia*, 2006, en prensa.

[8] S. González, A. Navarro, J. Lopez, and E. Zapata, "Analytical model of a threshold-based patching approach in a video on demand system," *IEEE Transactions on Broadcasting*, en revisión.

[9] M.S. Chen, D. Kandlur, and P. Yu, "Storage and retrieval methods to support fully interactive playout in a disk-array based video servers," *Multimedia Systems*, vol. 3, no. 3, pp. 126–135, 1995.

[10] J. Dey-Sircar, J. Salehi, J. Kurose, and D. Towsley, "Providing vcr capabilities in large-scale video servers," in *Second ACM Multimedia Conference*, 1994, pp. 25–32.

[11] K.C. Almeroth and M.H. Ammar, "The use of multicast delivery to provide a scalable and interactive video-on-demand service," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 6, pp. 1110–1122, 1996.

[12] W. W.-F. Poon and K.-T. Lo, "Design of multicast delivery for providing vcr funcionality in interactive video-on-demand systems," *IEEE Transactions on Broadcasting*, vol. 45, no. 1, pp. 141–148, March 1999.

[13] M. Y. Y. Leung, J. C. S. Lui, and L. Golubchik, "Use of analitical performance models for system sizing and resource allocation in interactive video-on-demand systems employing data sharing techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 3, pp. 615–637, May/June 2002.

[14] D.W. Brubeck and L.A. Rowe, "Hierarchical storage management in a distributed vod system," *IEEE Multimedia*, pp. 37–47, Fall 1996.

[15] H. Pang, B. Jose, and M.S. Krishnan, "Resource scheduling in a high-performance multimedia server," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 2, pp. 303–320, March-April 1999.

[16] J.C-Lliu, D.H.C. Du, S.S.Y Shim, J. Hsieh, and M. Lin, "Design and evaluation of a generic software architecture for on-demand video servers," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 3, pp. 406–423, May-June 1999.

[17] N.J. Sarhan and C.R. Das, "Adaptive block rearrangement algorithms for video-on-demand servers," in *International Conference on Parallel Processing*, 3-7 September 2001, pp. 452–459.

[18] E.W.M. Wong and S.C.H. Chan, "Performance modeling of video-on-demand systems in broadband Networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 7, pp. 848–859, July 2001.

[19] E. Souza, H.R. Gail, L. Golubchik, and J.C.S. Lui, "Analytical models for mixed workload multimedia storage servers," *Performance Evaluation an International Journal*, , no. 36–37, pp. 185–211, 1999.

[20] R. Boutaba and A. Hafid, "A generic platform for scalable access to multimedia-on-demand systems," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 9, pp. 1599–1613, September 1999.

[21] C. Shahabi, R. Zimmermann, K. Fu, and S-Y. D. Yao, "Yima: A second-generation continuous media server," *Computer*, vol. 35, no. 2, pp. 56–64, 2002.

[22] T.D.C. Little and D. Venkatesh, "Prospects for interactive video-on-demand," *IEEE Multimedia*, vol. 1, no. 3, pp. 14–24, 1994.

[23] M. Ko and I. Koo, "An overview of interactive video on demand system," Tech. Rep., University of British Columbia, December 1996.

[24] C. Vassilakis, M. Paterakis, and P. Triantafillou, "Video placement and configuration of distributed video servers on cable tv networks," *Multimedia Systems*, vol. 8, no. 2, pp. 92–104, 2000.

[25] Telefónica, *Imagenio: La Nueva Forma de Ocio a la Carta*, División de Relaciones Corporativas y Comunicación, 2004.

[26] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proceedings of the Second ACM International Conference on Multimedia*, 1994, pp. 15–24.

[27] H. Shachnai and P. Yu, "Exploring wait tolerance in effective batching for video-on-demand scheduling," *Multimedia Systems*, vol. 6, no. 6, pp. 382–394, 1998.

[28] C.C. Aggarwal, J.L. Wolf, and P.S. Yu, "The maximum factor queue length batching scheme for video-on-demand systems," *IEEE Transactions on Computers*, vol. 50, no. 2, pp. 97–109, 2001.

[29] K.A. Hua, Y. Cai, and S. Sheu, "Patching: a multicast technique for true on-demand services," in *6th ACM International Conference on Multimedia*, September 1998, pp. 191–200.

[30] J.Y.B. Lee, "On a unified architecture for video-on-demand services," *IEEE. Transactions on Multimedia*, vol. 4, no. 1, pp. 38–47, March 2002.

[31] W.-F. Poon, K.-T. Lo, and J. Feng, "A hybrid delivery strategy for a video-on-demand system with customer reneging behavior," *IEEE Transactions on Broadcasting*, vol. 48, no. 2, pp. 140–150, 2002.

[32] L. H. Chang and K.-C. Lai, "Near video-on-demand systems with combining multicasting and unicasting batching," in *IEEE TENCON*, August 2001, pp. 198–201.

[33] Y.C. Tay and H.H. Pang, "Load sharing in distributed multimedia-on-demand systems," *IEEE Transactions on Knowledge and Data Engeneering*, vol. 12, no. 3, pp. 410–428, June 2000.

[34] Luigi Rizzo and Lorenzo Vicisano, "Replacement policies for a proxy cache," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 158–170, 2000.

[35] R. Tewari, H. M. Vin, A. Dan, and D. Sitaram, "Resource-based caching for web servers," in *SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[36] S. Sen, J. Reexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *IEEE INFOCOM*, April 1999, pp. 1310–1319.

[37] J. Almeida, D. Eager, and M. Vernon, "A hybrid caching strategy for streaming media files," in *SPIE/ACM Conference on Multimedia Computing and Networking*, January 2001.

[38] Y. Wang, Z.-L. Zhang, D. Du, and D. Su, "A Network conscious approach to end-to-end video delivery over wide area Networks using proxy servers," in *IEEE INFOCOM*, April 1998.

[39] Y. Guo, S. Sen, and D. Towsley, "Prefix caching assisted periodic broadcas: Framework and techniques to support streaming for popular videos," Tech. Rep., University of Massachusetts, Amherst, MA, USA, 2001.

[40] P. Frossard and O. Verscheure, "Batched patch caching for streaming media," *IEEE Communications Letters*, vol. 6, no. 4, pp. 159–161, April 2002.

[41] S. Ramesh, I. Rhee, and K. Guo, "Multicast with cache (mcache): An adaptive zero delay video-on-demand service," in *IEEE INFOCOM*, 2001, pp. 85–94.

[42] D. N. Serpanos and A. Bouloutas, "Centralized versus distributed multimedia servers," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 8, pp. 1438–1449, December 2000.

[43] G. Zipf, *Human Behavior and the Principle of Least Effort*, Addison-Wesley, Cambridge, Massachussetts, 1949.

[44] A.L. Chervenak, *Tertiary Storage: An Evaluation of New Applications*, Ph.D. thesis, University of California at Berkeley, December 1994, U.C. Berkeley Technical Report UDB/CSD 94/847.

[45] L. A. Adamic, "Zipf, power-laws, and pareto - a ranking tutorial," http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html.

[46] N.L.S. da Fonseca and R.D.A. Facanha, "The look-ahead-maximize-batch batching policy," *IEEE Transactions on Multimedia*, vol. 4, no. 1, pp. 114–120, March 2002.

[47] M. Sainz, *Manual Básico de Producción en Televisión*, IORTV, 1995.

[48] T.D.C. Little and D. Venkatesh, "Popularity-based assignment of movies to storage devices in a video-on-demand system," *Multimedia Systems*, vol. 2, pp. 280–287, 1995.

[49] A.N. Mourad, "Issues in the design of a storage server for video-on-demand," *Multimedia Systems*, vol. 4, pp. 70–86, 1996.

[50] X. Zhou and R. Luling, "Dynamic adaptive mapping of videos in a hierarchical tv-anytime server network," in *Ninth International Conference on Computer Communications and Networks*, October 2000, pp. 402–409.

[51] Y.W. Leung and R.Y.T. Hou, "Assignment of movies to heterogeneous video servers," Tech. Rep. COMP-03-009, Dept.of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong, March 2003.

[52] J. Kee-Yin Ng, S. Xiong, and H. Shen, "A multi-server video-on-demand system with arbitrary rate playback support," *The Journal of Systems and Software*, , no. 51, pp. 217–227, 2000.

[53] D. Gross and C. Harris, *Fundamentals of Queueing Theory*, Wiley, 3rd edition, 1998.

[54] P.J. Shenoy, P. Goyal, and H.M. Vin, "Issues in multimedia server design," in *ACM Comput. Surv.*, December 1995.

[55] S. Viswanathan and T. Imielinaki, "Metropolitan area video-on-demand service using pyramid broadcasting," *Multimedia Systems*, vol. 4, no. 4, pp. 197–208, August 1996.

[56] C.C. Aggarwal, J.L. Wolf, and P.S. Yu, "A permutation-based pyramid broadcasting scheme for video-on-demand systems," in *IEEE International Conference Multimedia Computing and Systems*, June 1996, pp. 253–258.

[57] K.A. Hua and S. Sheu, "Skyscraper broadcasting: A New broadcasting scheme for metropolitan video-on-demand systems," in *SIGCOMM'97*, France, September 1997, pp. 899–910.

[58] A.D. Gelman and S. Halfin, "Analysis of resource sharing in information providing services," in *IEEE Global Telecommunications Conference and Exhibition (GLOBECOM '90)*, December 1990, pp. 312–316.

[59] L. Gao and D. Towsley, "Threshold-based multicast for continuous media delivery," *IEEE Transactions on Multimedia*, vol. 3, no. 4, pp. 405–414, December 2001.

[60] D.L. Eager, M.C. Ferris, and M.K. Vernon, "Optimized regional caching for on-demand data delivery," in *SPIE Conference Multimedia Computing and Networking (MMCN'99)*, San Jose, CA, January 1999, pp. 301–316.

[61] S.W. Lau, J.C.S. Liu, and L. Golubchic, "Merging video streams in a multimedia storage server: complexity and heuristics," *ACM Multimedia Systems Journal*, vol. 6, no. 1, pp. 29–42, 1998.

[62] D.L. Eager, M.K. Vernon, and J. Zahorjan, "Minimizing bandwidth requirements for on-demand data delivery," *IEEE Transactions on Knowledge and Data Engineering*, 2001.

[63] D. Guan and S. Yu, "A two-level patching scheme for video-on-demand delivery," *IEEE Transactions on Broadcasting*, vol. 50, no. 1, pp. 11–15, March 2004.

[64] D. Eager, M. Vernon, and J. Zahorjan, "Optimal and efficient merging schedules for video-on-demand servers," in *7th ACM International Multimedia Conference*, Orlando, Florida, November 1999, pp. 199–202.

[65] Jr. E. G. Coffman, P. Jelenkovic, and P. Momcilovic, "The diadyc stream merging algorithm," *Journal of Algorithms*, vol. 43, pp. 120–137, 2002.

[66] A.O. Allen, *Probability, Statistics and Queueing Theory: with Computer Sciences Applications*, Academic Press, 2nd edition, 1990.

[67] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," *Multimedia Systems*, vol. 4, pp. 112–121, June 1996.

[68] E.L. Abram-Profeta and K. Shin, "Scheduling video programs in near video-on-demand systems," in *ACM International Conference on Multimedia*, November 1997, pp. 359–371.

[69] S.-H.G. Chan and F.A. Tobagi, "On achieving profit in providing near video-on-demand services," in *IEEE International Conference on Communications (ICC '99)*, June 1999, pp. 988–993.

[70] S.-H.G. Chan and F.A. Tobagi, "Tradeoff between system profit and user delay/loss in providing near video-on-demand service," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 8, pp. 916–927, August 2001.

[71] M. Gupta and M. Ammar, "A novel multicast scheduling scheme for multimedia servers with variable access patterns," in *IEEE International Conference on Communications (ICC '03)*, May 2003, pp. 875–879.

[72] W.-F. Poon, K.-T. Lo, and J. Feng, "Determination of efficient transmission scheme for video-on-demand (vod) services," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 2, pp. 188–192, 2003.