Proyecto de Grado

Presentado ante la ilustre Universidad de Los Andes como requisito final para obtener el Título de Ingeniero de Sistemas

Sistema de Compresión Automática y Adaptativa para la Transmisión Eficiente de Datos y Uso Adecuado de los Recursos Computacionales

Por

Br. Jhonathan Daniel Abreu Luque

Tutor: José Luis Paredes Quintero, PhD.

Octubre, 2018

Sistema de Compresión Automática y Adaptativa para la Transmisión Eficiente de Datos y Uso Adecuado de los Recursos Computacionales

Br. Jhonathan Daniel Abreu Luque

Proyecto de Grado — Sistemas Computacionales, 63 páginas

Resumen: La compresión de datos es una estrategia muy últil para reducir la cantidad de datos que representan la información siendo transmitida para así mejorar el rendimiento de la comunicación. Desafortunadamente, ningún algoritmo es conveniente en todas las situaciones y pueden incluso degradar el desemepeño de la transmisión, dependiendo de muchos factores del entorno y de los datos. Debido a esto, se propone un mecanismo de compresión automática y adaptativa que seleccione y utilice el mejor algoritmo de compresión en cada oportunidad, o decida no comprimir cuando sea conveniente. Para la evaluación de este mecanismo, se desarrolló un sistema cliente-servidor de transmisión de datos que lo implementa. Los resultados muestran que el mecanismo propuesto logra lidiar con situaciones en la que algunos compresores se hacen subóptimos o más costos que no comprimir y, pese a que toma decisiones equivocadas en algunas situaciones, entender el por qué permite encontrar las limitaciones y nuevos retos a los que se deben enfrentar los mecanismos de compresión adaptativa.

Palabras clave: Compresión de datos, optimización del uso de los recursos computacionales, transmisión de información, compresión automática, compresión adaptativa.

Este trabajo fue procesado en LaTeX.

Índice

Índice de Tablas

Índice de Figuras

Índice de Algoritmos

Capítulo 1

Introducción

"Nos encontramos en medio de la evolución a los procesadores multinúcleo, lo cual ocasiona que la capacidad computacional crezca más rápido que la capacidad de comunicación disponible" (Jägemar et al., 2016). Esto, aunado a la vigencia de la Ley de Moore, trae como consecuencia un aumento en la demanda de mecanismos de comunicación de alto rendimiento.

En aplicaciones modernas, el volumen de información generada y potencialmente transportada es substancial, pudiendo estresar incluso a infraestructuras de comunicación de muy alto desempeño (Wiseman et al., 2004). La información se genera en grandes volúmenes y a tazas muy elevadas, pudiendo provenir de sensores, satélites de observación, fuentes de información especializada o de grandes aplicaciones de negocios. La compresión juega, así, un papel importante en la reducción de volumen de información que debe ser transmitida o almacenada.

Existen diversos métodos y algoritmos de compresión, cada uno con sus ventajas y limitaciones. A modo de ilustración, según Jägemar et al. (2016), el algoritmo *Snappy* (Google, 2018b) ofrece compresión muy rápida y es apropiado para texto, mientras que el *QLZ* (QuickLZ, 2018) ofrece compresión muy rápida y es adecuado sólo para mensajes pequeños. En consecuencia, dependiendo del tipo de datos a comprimir y de los recursos disponibles, un algoritmo en específico puede ser el más apropiado de entre un conjunto dado de algoritmos de compresión.

Factores adicionales, de gran importancia, que se deben tomar en cuenta en el

desarrollo de un sistema de compresión, además de los tipos de datos o contenidos a ser transmitidos, son la carga del CPU y la congestión de la red: un servicio puede compartir recursos con muchos otros y se debe garantizar que estos nunca lleguen a un estado de inanición al utilizar toda la capacidad de cómputo en el proceso de compresión. Además, se debe considerar un equilibrio entre el ancho de banda disponible y la cantidad de información que se transmite.

1.1 Antecedentes

En las últimas décadas, la compresión de datos y, más específicamente, el problema de la compresión automática o adaptativa, han sido investigados desde muchos ángulos y con diversos enfoques. Krintz y Sucu (2006) implementaron un sistema llamado *Adaptive Compression Environment* (ACE), para el cual se generan, de forma *offline*, líneas de regresión que relacionan cada par de algoritmos de compresión disponibles, las cuales son utilizadas por el sistema para predecir el desempeño de cada algoritmo a partir del rendimiento del último algoritmo utilizado y así seleccionar el más adecuado en un instante determinado para transmitir el próximo archivo o paquete.

Peterson y Reiher (2016), en su investigación, refutan a Krintz y Sucu (2006) observando que no se debe generalizar y suponer relaciones lineales entre los distintos algoritmos de compresión. Presentan, entonces, *Datacomp*, un sistema cuyo proceso de toma de decisiones se basa en la afirmación de que un método de compresión tendrá siempre el mismo desempeño dado un conjunto de condiciones del entorno. De forma también *offline*, cuantizan el dominio de las condiciones de los recursos (CPU, ancho de banda, entre otros) y la compresibilidad de los datos para obtener un modelo que utiliza el sistema para determinar el algoritmo más apto en un instante dado, a partir de información de desempeño calculada, en línea, para cada método y para cada clase del modelo.

Las investigaciones previamente mencionadas tienen en común la utilización, en mayor o menor medida, de información generada previo a la inicialización del sistema. Jägemar et al. (2016), por otro lado, realizan la selección automática completamente en línea, dividiendo el flujo de mensajes en rondas de tamaño fijo, cada una con una

etapa inicial en la que se calcula una distribución de probabilidad a partir de los datos de rendimiento de la ronda anterior, asignando la mayor probabilidad de ser utilizado al algoritmo que dio mejores resultados en la ronda previa.

Existen algoritmos que permiten regular el esfuerzo aplicado a la compresión, siendo esto útil en casos en los que se requiera ajustarse a límites de uso del CPU. En este sentido, Zohar y Cassuto (2014) desarrollaron un sistema aplicado al servidor Web Apache (Apache Software Fundation, 2018) que, a partir de mediciones instantáneas de la carga del CPU, ajusta el nivel de compresión del algoritmo *gzip* (Gailly y Adler, 2018a), protegiendo así al *host* de sobrecargas causadas por peticiones en horas pico o ataques del tipo de Denegación de Servicios (DoS) y de comprimir cuando no es necesario.

Es notorio también que, como ya fue mencionado, la compresión no siempre es adecuada y las investigaciones previamente mencionadas manejan este escenario de diferentes maneras, ya sea implementando mecanismos de control de tipo PID para ajustar el tiempo disponible para comprimir (Jägemar et al., 2016), con una clase particular en el modelo cuantizado para información no compresible (Peterson y Reiher, 2016), comparando predicciones de tiempo de transmisión del archivo comprimido o sin comprimir (Krintz y Sucu, 2006) o ajustando el nivel de compresión a 0 para evitar la compresión (Zohar y Cassuto, 2014).

1.2 Planteamiento del problema

La computación enfrenta actualmente un gran reto con respecto a la creciente brecha existente entre el poder de cómputo y la capacidad de comunicación y de acceso a memoria, especialmente con la popularización de la computación paralela y distribuida, los clusters y la computación en malla. La creciente demanda de este y otros tipo de sistemas distribuidos (sistemas Web, P2P, multimedia, etc.) y el volumen de información que en estos se transmite, requieren que se desarrollen sistemas de comunicación más eficientes.

La comunicación, como lo afirman Peterson y Reiher (2016), es una función del tamaño de la información, el cual puede ser reducido utilizando compresión. Sin

embargo, la compresión puede, en ciertos casos, ser innecesaria e incluso degradar el desempeño de la comunicación al incrementar el tiempo efectivo de transmisión o incluso aumentar el tamaño de los datos. Dicho esto, son numerosos los aspectos que se deben tomar en cuenta al considerar la compresión como estrategia para mejorar el proceso de comunicación, entre ellos el formato de la información o la compresibilidad de la misma y los recursos disponibles (CPU, ancho de banda, etc.). Numerosas investigaciones han tratado este problema por muchos años y desde distintos ángulos, aportando, sin embargo, soluciones parciales al mismo (Zohar y Cassuto, 2015), incluyendo en su mayoría soluciones atadas a ciertos sistemas específicos. En ese orden de ideas, es necesario continuar ampliando el espectro de soluciones y mecanismos de simple integración en cuanto al uso de la compresión para mejorar la efectividad de la comunicación en sistemas que así lo requieran.

1.3 Justificación

Los desafíos a los que actualmente hace frente la computación en cuanto al acelerado incremento del poder de cómputo disponible, en contraste con las capacidades de comunicación existentes, aunado al creciente volumen de información siendo generada y transmitida, requieren de métodos innovadores que mejoren y optimicen la utilización deolvio el internet! los recursos de comunicación disponibles. Debido a la cantidad de variables involucradas en la comunicación y transmisión de datos, de las cuales depende su desempeño, no existe aún una solución global al problema de la selección automática del algoritmo más apto en la utilización de la compresión como medio para la mejora de la transmisión. Es por esto que se requiere que se continúen las investigaciones en este campo, de modo que se aporten cada vez más soluciones que conlleven a la optimización de la utilización de los recursos en cuanto a la comunicación y transmisión de datos se refiere y, además, se identifiquen las limitaciones y nuevos retos que estas implican.

1.4 Objetivos

1.4.1 Objetivo General

Desarrollar un sistema de compresión y transmisión de datos que seleccione, de forma automática y cuando sea conveniente, el algoritmo de compresión más apto de entre un conjunto de algoritmos disponibles y se adapte a las condiciones instantáneas de los recursos computacionales y de comunicación disponibles.

1.4.2 Objetivos Específicos

- Seleccionar los tipos característicos de datos a comprimir y generar la base de datos de prueba.
- Seleccionar los algoritmos de compresión a utilizar.
- Analizar la información de rendimiento de cada algoritmo para cada tipo de datos de la base de datos de prueba.
- Seleccionar los recursos computacionales a tomar en cuenta: carga del CPU, ancho de banda disponible, congestionamiento de la red, uso de memoria, entre otros.
- Diseñar la estrategia de selección automática del algoritmo más apto y adaptación a las condiciones de los recursos.
- Implementar el sistema de transmisión y compresión de datos automático y adaptativo.
- Evaluar el rendimiento del sistema implementado mediante las métricas más apropiadas encontradas en la literatura.

1.5 Metodología

El sistema propuesto fue desarrollado siguiendo el modelo iterativo e incremental en todo el ciclo de vida del desarrollo. No obstante, el núcleo del sistema es la estrategia de compresión automática y adaptativa, por lo cual se siguieron los siguientes pasos o actividades como enfoque metodológico para el cumplimiento de los objetivos planteados:

- Se llevó a cabo una revisión bibliográfica exhaustiva sobre los conceptos relacionados a la compresión de datos en general y compresión adaptativa.
- Se diseñó una arquitectura preliminar del sistema de transmisión de datos a partir de un conjunto base de requerimientos, especificando los protocolos y funcionalidades principales requeridas para el sistema.
- Se procedió con el desarrollo del sistema de transmisión, como ya fue mencionado, siguiendo el modelo de desarrollo iterativo e incremental.
- El mecanismo de compresión automática y adaptativa fue diseñado e implementado, integrándolo al sistema de transmisión de datos desarrollado.
- Finalmente, se llevó acabo la evaluación del sistema de compresión automática y adaptativa, tomando en cuenta diferentes escenarios y pruebas comparativas que permitieron evaluar el desempeño de la estrategia de compresión diseñada y el sistema que la implementa.

1.6 Estructura del documento

El resto del presente documento se organiza como sigue.

El capítulo 2 presenta las bases teóricas del presente trabajo. Se reseñan los conceptos básicos de la compresión de datos y sus medidas de rendimiento. Además, se presenta un resumen de los recursos computacionales de los que se disponen y sus potenciales métodos de medición o estimación. Adicionalmente, se describen los árboles

de decisión como algoritmos de clasificación rápida y eficiente y sus aplicaciones, para finalizar con conceptos del modelo de desarrollo iterativo e incremental.

El capítulo 3 presenta el ciclo de vida del desarrollo del sistema de transmisión de datos, con las funcionalidades requeridas para proveer compresión de datos estática y adaptativa.

En el capítulo 4 se presenta el diseño del mecanismo automático y adaptativo de compresión de datos para la transmisión eficiente de información junto con su integración en el sistema de transmisión de datos.

En el capítulo 5, se describen las pruebas de evaluación a las que fue sometido el mecanismo de compresión automática y se discuten los resultados obtenidos.

Finalmente, en el capítulo 6 se presentan las conclusiones del trabajo realizado, así como también las recomendaciones finales y potencial trabajo futuro.

Capítulo 2

Marco teórico

2.1 Compresión de datos

La compresión es el proceso de reducir el tamaño de un archivo de datos, lo cual se logra al reducir el volumen de datos o la cantidad de bits que lo representan. En otras palabras, el resultado de la compresión es una representación compacta de la información, ocupando menos espacio a cambio de tiempo y capacidad de cómputo.

Pu (2004) define a la compresión de datos como la ciencia y arte de representar la información de forma compacta, cuyo enfoque se basa en dos pasos: el *modelado* — que consiste en la construcción de sistemas de conocimiento para la compresión — y la *codificación* — que es el diseño del código que representa la información de forma compacta. Lo anterior tiene sentido, pues la compresión es una especialización de la codificación, con la diferencia que en la primera, el resultado es de menor tamaño.

2.1.1 Tipos de compresión

Según los aspectos de la información tomados en cuenta por los distintos algoritmos de compresión, se deriva una clasificación que depende de si el algoritmo permite regenerar o no la información intacta después de la descompresión.

Compresión sin pérdidas

Los métodos de compresión sin pérdidas son aquellos que permiten que cada bit que representa la información pueda ser recuperado, en el mismo orden, en el proceso de descompresión. Esto significa, en otras palabras, que las señales original y descomprimida, son numéricamente idénticas. Esta categoría aplica principalmente a aquellos algoritmos que se fundamentan en la Teoría de la Información y que aprovechan la redundancia estadística natural de la mayor parte de la información del mundo real (Mahmud, 2012).

En este tipo de técnicas de compresión, la capacidad y la tasa de compresión son funciones de la utilización de los recursos de computación. Estos métodos, por lo general, se basan en *modelos de probabilidad* o en la frecuencia de aparición de los bytes y cadenas de estos — como la codificación de Huffman (Huffman, 1952) — o en *diccionarios*, manteniendo tablas de códigos simples que reemplazan cadenas de símbolos — como la familia de algoritmos Lempel-Ziv (Ziv y Lempel, 1977).

Compresión con pérdidas

La compresión con pérdidas es posible solo en casos en los que existe cierta tolerancia a las pérdidas y la realizan aquellos métodos que no permiten reconstruir la información exactamente. Esto ocurre debido a que estos métodos de compresión reducen el tamaño de los archivos al eliminar de forma permanente información redundante o irrelevante para reducir la cantidad de bits que los representan (Mahmud, 2012).

Este tipo de compresión aplica principalmente a imágenes (JPEG, por ejemplo), audio y video (MPEG-2, por ejemplo), en los que, por lo general, la información se reconstruye con pérdidas que son perceptibles pero tolerables — como lo es en el caso de las videoconferencias, lo cual se refleja en la calidad del audio y video — en cuyo caso podría decirse que la compresión es *subjetivamente con pérdidas*. Si el algoritmo de compresión elimina información redundante y/o irrelevante, haciendo que las pérdidas sean prácticamente imperceptibles, se está frente a un caso de compresión *subjetivamente sin pérdidas* — como en el caso del JPG, donde la pérdida es perceptible solo al acercar (*zoom*), por lo que la falta de información es invisible a simple vista.

En esta clase, los niveles de compresión neta que se alcanzan pueden ser de hasta 80% con degradación de la señal, imagen o video prácticamente imperceptible.

2.1.2 Medidas de rendimiento

Esencialmente, son dos las métricas que se utilizan para medir el desempeño de un algoritmo de compresión: el porcentaje de compresión o *compression ratio* y la tasa de compresión o *compression rate*. Estas métricas refieren a las capacidades de un compresor, tanto en eficacia como en eficiencia.

Porcentaje de compresión

El porcentaje de compresión o *compression ratio* (PC) es una medida de cuantificación de la reducción del tamaño de un archivo o señal por parte de un algoritmo de compresión. Formalmente, se define como la razón o proporción entre el tamaño del archivo original y el tamaño del archivo comprimido (Ecuación 2.1) y depende de las propiedades de los datos y del algoritmo.

$$PC = \frac{T_o}{T_c} \tag{2.1}$$

donde:

T_o = Tamaño original
T_c = Tamaño comprimido

Del mismo modo, puede utilizarse el concepto de ahorro de espacio o *space savings* (AE), que representa la reducción del tamaño relativo al tamaño del archivo descomprimido — es decir, la proporción o porcentaje de reducción de espacio — y se define como en la ecuación 2.2.

$$AE = 1 - \frac{1}{PC} = 1 - \frac{T_c}{T_o} \tag{2.2}$$

Dicho esto, una compresión que lleva un archivo de 20MB a una representación compacta de 5MB, tiene un PC de 4 (20/5 = 4) y un AE de 0,75 (1 - 5/20 = 0,75),

lo que significa que se comprime el archivo a un cuarto de su tamaño original y que se ahorra un 75% del espacio requerido para almacenarlo, respectivamente. Lo ideal es, entonces, diseñar compresores con alta capacidad de compresión, pues de esta manera se consiguen representaciones más compactas de los archivos — permitiendo así su transmisión en forma más rápida o ahorros en recursos de almacenamiento — no sin tomar en cuenta que altas capacidades de compresión requieren tiempo y esfuerzo de computación.

Tasa de compresión

La tasa o velocidad de compresión (*compression rate* o *compression speed*), TC, depende de los recursos computacionales (y su estado) y se define como la tasa o velocidad a la cual un algoritmo reduce la cantidad de bits que representan un determinado archivo. Se mide en unidades de información por unidad de tiempo y, considerando recursos computacionales y condiciones constantes, puede depender únicamente del formato de los datos y del algoritmo mismo. La Ecuación 2.3 define formalmente la forma de calcular la velocidad de compresión.

$$TC = \frac{T_o}{t_c} \tag{2.3}$$

donde:

t_c = Tiempo de compresión

Así mismo, este concepto aplica a la velocidad de descompresión o *decompression speed* (TD), para medir el rendimiento del proceso de descompresión (Ecuación 2.4).

$$TD = \frac{T_o}{t_d} \tag{2.4}$$

donde:

t_d = Tiempo de descompresión

Los algoritmos de compresión, por lo general, intercambian capacidad de compresión por velocidad (y viceversa), dependiendo de los fines para los que son

diseñados. Por ejemplo, según Jägemar et al. (2016), los algoritmos LZFX (Collette, 2018) y LZO (Oberhumer, 2018) ofrecen compresión rápida pero con bajo PC, mientras que el algoritmo LZMA (Pavlov, 2018) ofrece compresión lenta con alto PC, existiendo también puntos intermedios, como en el caso de LZW (Welch, 1984) y BZIP2 (Seward, 2018). Este fenómeno es ilustrado en el Capítulo 4.

2.2 Recursos computacionales

Todo componente, sea físico o virtual, con capacidades limitadas en un sistema computacional, se considera un recurso. Un dispositivo conectado a un sistema computacional y cualquier componente dentro del mismo se considera un recurso. Los recursos virtuales incluyen los descriptores de archivos, sockets de red, áreas de memoria, entre otros.

Algunos recursos pueden ser regulados por el kernel, ocasionando que el usuario se vea limitado o ralentizado proporcionalmente a dicha regulación. Específicamente — y tomando en cuenta aquellos que pueden afectar el rendimiento de la compresión de datos — los recursos de importancia para la presente investigación son el tiempo de CPU, el ancho de banda y la memoria.

2.2.1 Tiempo de CPU

El tiempo total que una unidad central de procesamiento (CPU) es utilizado para procesar instrucciones de un programa de computadora, en lugar de, por ejemplo, esperar por operaciones de entrada/salida (E/S), es denominado tiempo de CPU y es asignado y medido en unidades discretas denominadas *clocks* o *ticks* del reloj del sistema, que corresponde, usualmente, a 1/100 segundos en la mayoría de las arquitecturas (Kerrisk, 2018).

Clasificación del tiempo de CPU

El tiempo de CPU es generalmente clasificado dependiendo de su estado o las tareas que este estuvo ejecutando en el último intervalo de tiempo. En las páginas del manual

de Linux (Kerrisk, 2018), específicamente en la sección `proc`, se describe la siguiente clasificación del tiempo del CPU para los valores reportados por el sistema operativo:

- **Tiempo de usuario ($t_{cpu\_usuario}$):** corresponde al tiempo que el CPU estuvo ocupado ejecutando instrucciones de procesos en modo usuario.
- **Tiempo del sistema ($t_{cpu\_sistema}$):** corresponde al tiempo que el CPU fue utilizado ejecutando instrucciones en modo kernel, es decir, instrucciones del núcleo del sistema operativo o en nombre del usuario, como en el caso de las llamadas al sistema.
- **Tiempo ocioso o *idle* (t_{cpu_idle}):** tiempo que el CPU estuvo desocupado y mide la capacidad no utilizada del CPU.
- **Tiempo de espera de entrada/salida (t_{cpu_es}):** tiempo que el CPU estuvo desocupado en espera por operaciones de entrada salida (E/S o *I/O*, por sus siglas en inglés).

Adicionalmente, el **tiempo total del CPU** (t_{cpu_total}), *elapsed time* o *wall time*, es el tiempo total transcurrido desde el inicio del sistema hasta el instante de su consulta, mientras que el **tiempo total de uso del CPU** ($t_{cpu\_total\_uso}$) es el tiempo durante el cual el CPU fue efectivamente utilizado.

Carga del CPU

El principal uso del tiempo del CPU es la medición de la carga del CPU, que se define como el porcentaje del tiempo del CPU que fue efectivamente utilizado en un intervalo de tiempo — lo cual también puede interpretarse como una medida de qué tan cargado se encuentra el CPU — y puede ser calculado con la Ecuación 2.5 (Zohar y Cassuto, 2014).

$$Uso\ del\ CPU = \frac{t_{cpu\_total\_uso}}{t_{cpu\_total}} = \frac{t_{cpu\_total} - t_{cpu\_idle} - t_{cpu\_es}}{t_{cpu\_total}} \tag{2.5}$$

El tiempo total del CPU se calcula de manera intuitiva como la suma de cada tiempo individual según la clasificación presentada anteriormente de la siguiente manera:

$$t_{cpu\_total} = t_{cpu\_usuario} + t_{cpu\_sistema} + t_{cpu\_idle} + t_{cpu\_es}$$

De este modo, la Ecuación 2.5 para el cálculo del uso o carga del CPU puede escribirse como en la Ecuación 2.6.

$$\begin{aligned} Uso\ del\ CPU &= \frac{(t_{cpu\_usuario} + t_{cpu\_sistema} + t_{cpu\_idle} + t_{cpu\_es}) - t_{cpu\_idle} - t_{cpu\_es}}{t_{cpu\_usuario} + t_{cpu\_sistema} + t_{cpu\_idle} + t_{cpu\_es}} \\ &= \frac{t_{cpu\_usuario} + t_{cpu\_sistema}}{t_{cpu\_usuario} + t_{cpu\_sistema} + t_{cpu\_idle} + t_{cpu\_es}} \end{aligned} \tag{2.6}$$

2.2.2 Ancho de banda

En este caso aplicado a la red, el ancho de banda es la tasa de transmisión de datos o la cantidad de bits que pueden ser transmitidos por unidad de tiempo. Se mide en bits por segundo (bit/s) y depende en gran medida del ruido del canal de comunicación. Pueden extraerse dos conceptos distintos en el caso de los sistemas de comunicación, siendo estos la **capacidad de ancho de banda** y el **consumo de ancho de banda** (Forouzan, 2006).

Capacidad de ancho de banda

La capacidad de ancho de banda o ancho de banda disponible, representa la capacidad neta del canal de comunicación o su tasa máxima de transmisión y tiene un límite teórico definido por el teorema de Shannon-Hartley (Shannon, 1949).

Consumo de ancho de banda

La tasa promedio efectiva de transmisión exitosa a través de un canal de comunicación se denomina consumo de ancho de banda. Esto difiere del concepto

de capacidad de ancho de banda debido a que, en aplicaciones reales, los protocolos, el cifrado y otros factores, agregan un *overhead* considerable que no permite alcanzar efectivamente el ancho de banda disponible desde el punto de vista del usuario.

2.2.3 Memoria

Específicamente la memoria de acceso aleatorio (RAM), es un recurso que permite almacenar los datos y código ejecutable actualmente en uso. Este tipo de memoria es volátil, normalmente costosa y es utilizada como medio de almacenamiento y espacio de trabajo para el sistema operativo y otras aplicaciones. Una de las principales restricciones de este tipo de memoria es la diferencia de velocidades con el CPU, pues se encuentra fuera del chip y las capacidades de comunicación y ancho de banda entre estos recursos es relativamente limitado.

2.3 Árboles de decisión

El proceso de toma de decisiones para seleccionar el mejor algoritmo de compresión en un instante determinado, dado un conjunto de variables que definen el estado del entorno y las propiedades de los datos, puede verse desde la perspectiva del aprendizaje automatizado, específicamente como un problema de clasificación, donde los atributos corresponden a las características actuales del entorno, a saber, carga del CPU, estado de la red, tipos de datos, entre otros.

El problema de clasificar o etiquetar observaciones o individuos de cierto fenómeno o población en un conjunto finito de clases, es uno de los campos de acción del aprendizaje automatizado o *machine learning* y de la ciencia de datos en general y se encuentra presente en muchos ámbitos del mundo real. Los árboles de decisión son una de las herramientas más básicas del aprendizaje automatizado para la tarea de **clasificación** y pertenecen al subconjunto de algoritmos de **aprendizaje supervisado**, los cuales utilizan datos de entrada o entrenamiento, cuyas clases son conocidas, para aprender — ya sea según patrones encontrados en los datos, reglas o funciones matemáticas — cómo clasificar individuos no presentes en el conjunto de datos de entrenamiento.

2.3.1 Descripción

Los árboles de decisión clasifican individuos u observaciones planteando una serie de preguntas acerca de las características de dichos individuos. Estas preguntas se plantean de forma jerárquica en forma de árbol, donde cada nodo interno almacena una pregunta y tiene tantos hijos como posibles respuestas tenga la pregunta. Los nodos hojas — aquellos que no tienen ningún hijo — no almacenan preguntas sino clases. La clasificación en un árbol de decisión se lleva a cabo encontrando el único camino que lleva, para una observación dada, desde la raíz hacia una hoja, la cual contiene la clase a la que pertenece dicha observación, de acuerdo a las respuestas asociadas a las características de la misma. En algunas variaciones, los nodos hoja no almacenan una clase específica sino un arreglo de probabilidades o una distribución que estima la probabilidad de que un individuo que haya alcanzado dicha hoja pertenezca a una clase particular (Kingsford y Salzberg, 2008).

Las preguntas almacenadas en cada nodo pueden tener numerosas respuestas y pueden ser tan complicadas como sea necesario, mientras puedan ser computadas de forma eficiente. Sin embargo, en su forma más sencilla, las preguntas son binarias (si o no) y se codifican en un árbol binario.

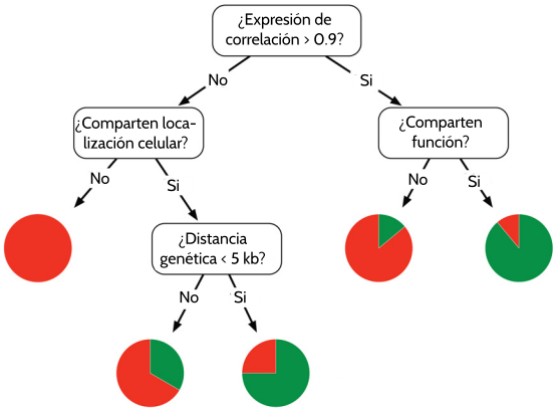
Por ejemplo, dado un conjunto de observaciones conocidas sobre diferentes pares de genes (Figura 2.1(a)) con características que definen si este par de genes interactúa. Un árbol de decisión puede construirse para clasificar pares de genes que no se encuentren presentes en el conjunto de datos (Figura 2.1(b)). En este ejemplo, las preguntas en cada nodo son binarias y los nodos hoja contienen la probabilidad de interacción entre el par de genes que se clasifica (representado por los gráficos de tortas), de modo que se predice que un par de genes interactúa si su clasificación lleva a un nodo hoja predominantemente verde.

2.3.2 Ventajas y desventajas

Algunas de las ventajas y desventajas principales de los árboles de decisión, según Kingsford y Salzberg (2008) y la documentación de la biblioteca de aprendizaje automático de Python, `scikit-learn` (Pedregosa et al., 2011), se presentan a

¿Interactúan?	Correlación de expresión	¿Comparten localización?	¿Comparten función?	Distancia genética
Si	0.77	Si	No	1 kb
Si	0.91	Si	Si	10 kb
No	0.1	No	No	1 Mb
⋮				

(a) Datos de entrenamiento.



(b) Árbol de decisión resultante.

Figura 2.1: Ejemplo hipotético de cómo un un árbol de decisión podría predecir interacciones entre genes (adaptado de Kingsford y Salzberg (2008)).

continuación.

Ventajas

- Son mas sencillos de entender e interpretar que otros algoritmos de clasificación, pues pueden ser visualizados gráficamente.
- El costo de utilizarlos, una vez construidos, es logarítmico con respecto al número de observaciones utilizadas para entrenarlos.
- Requieren poca preparación de datos, pues manejan naturalmente tanto datos numéricos como categóricos, así como también clasificación multiclase.
- Su modelo es caja blanca, pues cualquier situación u observación es fácilmente explicable mediante lógica booleana.

Desventajas

- Pueden ser inestables, debido a que pequeños cambios en los datos de entrada puede resultar en la generación de un árbol completamente diferente.
- Se pueden generar árboles sesgados si una clase predomina en el conjunto de datos de entrenamiento.

- Se pueden generar árboles demasiado complejos (sobreentrenamiento) cuando no es posible generalizar muy bien los datos.

2.4 Desarrollo de software iterativo e incremental

El modelo iterativo, como una implementación del ciclo de vida del desarrollo de software, se enfoca en una implementación inicial y simplificada — generada basándose en un conjunto de requerimientos iniciales razonablemente bien definidos — que progresivamente crece en complejidad, con un conjunto de características y funcionalidades cada vez mas amplio, hasta que el sistema final esté completo. Es común que los términos *iterativo* e *incremental* se utilicen liberalmente y de forma intercambiable. El término incremental, entonces, describe las alteraciones incrementales que se llevan a cabo durante el diseño e implementación de cada nueva iteración.

Cada iteración corresponde a una secuencia de actividades (ciclo de vida) que generan un incremento entregable del producto. El primer incremento es denominado *producto núcleo* y aborda los requerimientos básicos, dejando otras funcionalidades complementarias para posteriores incrementos. El usuario final o cliente usa y evalúa cada incremento y, con su retroalimentación, se genera el plan para el siguiente incremento. Este proceso se lleva a cabo al finalizar cada incremento y hasta que se obtenga el producto final (Pressman, 2015).

2.4.1 El proceso de desarrollo de software

El proceso que se sigue en el desarrollo de software bajo el modelo iterativo e incremental sigue un proceso cíclico que, después de una fase inicial de planeación, repite una serie de fases una y otra vez, generando un incremento al final de cada ciclo o iteración que mejora el software o le añade nuevas funcionalidades. El proceso se ilustra en la Figura 2.2 y se describe en detalle a continuación.

Planeación y Requerimientos: La fase inicial, incluso antes de comenzar las iteraciones, pasa por la elaboración de un *plan* inicial para generar un conjunto

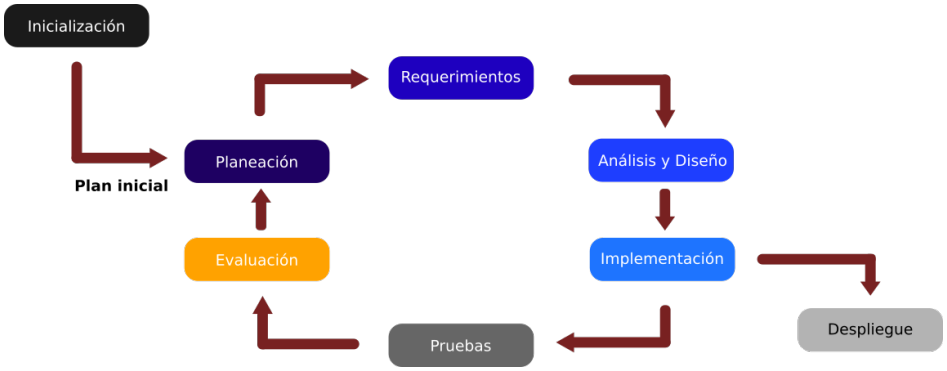


Figura 2.2: Fases detalladas y flujo de trabajo del modelo incremental.

de *requerimientos* base y preparar el proceso para las fases venideras del ciclo.

Análisis y Diseño: Una vez se tiene un plan y un conjunto de requerimientos, se realiza un procedimiento de *análisis* para determinar la lógica de negocios requerida. En esta fase de *diseñan* los modelos de bases de datos y se establecen los aspectos técnicos — como los lenguajes, capas, servicios, protocolos, etc. — requeridos para cumplir las necesidades de la fase de *análisis*.

Implementación: En esta fase, se codifican e *implementan* todos los requerimientos obtenidos en las fases anteriores.

Pruebas: Se llevan a cabo una serie de procedimiento de *prueba* para identificar cualquier potencial problema o *bug* en la *implementación* o *diseño*.

Evaluación: Finalmente, el producto de todas las etapas anteriores es *evaluado* para determinar el estado del proyecto e identificar las necesidades y posibles cambios que deban ser llevados a cabo.

Ya finalizadas todas las etapas de una iteración, es cuando la esencia de este modelo se pone en práctica: la retroalimentación obtenida de la etapa de evaluación permite generar un nuevo plan de trabajo y un nuevo conjunto de requerimientos, lo cual da inicio a una nueva iteración que va a dar como resultado un incremento del software en desarrollo. Este ciclo se repite hasta alcanzar un nivel de refinamiento predeterminado o aceptable y se obtenga un producto final (lo que corresponde en el recuadro “Despliegue” en la Figura 2.2).

Capítulo 3

Desarrollo del sistema de transmisión de datos

Un mecanismo de compresión adaptativa debe abstraerse dentro de un sistema o aplicación que lo implemente para acelerar el proceso de transmisión de datos. En el presente capítulo se describe el proceso de desarrollo del sistema de transmisión de datos que, posteriormente, podrá habilitar compresión automática y adaptativa con la finalidad de aprovechar de forma eficiente los recursos computacionales y de comunicación subyacentes y mejorar el desempeño de la comunicación.

3.1 Requerimientos del sistema

El proceso de desarrollo, bajo el modelo iterativo e incremental, comienza con una serie de requerimientos base bien definidos para la obtención del producto núcleo. En este proyecto, el proceso de recaudación de requerimientos, planeación y evaluación de las iteraciones semanales se dio en reuniones con el tutor, quien fungía, al igual que el autor, como usuario y miembro del equipo de desarrollo. No obstante, el diseño e implementación fue llevado a cabo en su totalidad por el autor. Los requerimientos, listados a continuación, evolucionaron de una iteración a otra, como es de esperarse con este tipo de metodologías de desarrollo.

3.1.1 Requerimientos funcionales

Los siguientes son los requisitos funcionales del sistema en desarrollo, es decir, aquellos que definen la funcionalidad del sistema y sus componentes:

- El sistema debe funcionar como una herramienta de transmisión de datos cliente-servidor (como *scp* o *rsync*[1]): del lado del servidor, un servicio o demonio espera por solicitudes de transmisiones que se generan del lado del cliente.
- El sistema debe poder transmitir tanto archivos individuales como directorios completos.
- Debe permitirse escoger, en tiempo de ejecución, entre diversas estrategias de compresión disponibles, ya sean estáticas o adaptativas, según opciones emitidas por el usuario, incluyendo la opción de no compresión.
- Se debe habilitar la compresión y transmisión simultáneas: los archivos se deben leer en trozos o *chunks* que, una vez comprimidos, se envían mientras los siguientes trozos se comprimen. De este modo, se evita que la interfaz de red esté ociosa mientras se lleva a cabo la compresión, con lo cual se espera una mejora del desempeño general.
- Como una prueba de concepto, el mecanismo "automático y adaptativo", en una primera iteración, comprimirá cada trozo en modalidad *round-robin* — esto es, de forma secuencial y ordenada, comenzando con el primer compresor de la lista (circular) hasta llegar al último, para comenzar de nuevo con el primero de la lista.

3.1.2 Requerimientos no funcionales

A continuación, se listan los requerimientos no funcionales del sistema, que describen sus restricciones y criterios para juzgar su operabilidad:

[1] *scp* es un programa que implementa el protocolo SCP (*Secure Copy Protocol*), basado en el protocolo SSH (*Secure Shell*), para transferir archivos entre dos — posiblemente remotos — terminales o *hosts*. *rsync* es una herramienta que permite la transferencia de archivos incremental y sincronización de directorios entre dos *hosts*, permitiendo compresión y encriptación.

- Debe ser implementado en el lenguaje C++, utilizando el *framework* Google Test (Google, 2018a) para la implementación de las pruebas unitarias.
- Debe funcionar y ser probado en sistemas operativos Linux.
- No debe agregar ningún tipo de *overhead* adicional al proceso de transmisión de datos.

3.2 Módulo de compresión de datos

Con respecto a la compresión de datos, el requisito más importante plantea que el sistema debe poder cambiar, en tiempo de ejecución, el mecanismo de compresión, según el estado actual del mismo (carga del CPU, ancho de banda disponible, etc.). Además, debe permitir que el usuario escoja manualmente el método de compresión a utilizar. El principal objetivo de esto es permitir compresión estática (con un algoritmo particular) para efectos de realización de pruebas. En este caso, el patrón de diseño por estrategias provee una solución útil para este problema.

Según Pressman (2015), un patrón de diseño es una abstracción que proporciona una receta para un problema de diseño en un contexto particular. Específicamente, el **patrón de diseño por estrategias** provee una solución que encapsula un conjunto de algoritmos y los hace intercambiables en tiempo de ejecución en un contexto dado. La utilidad es clara en el contexto del presente trabajo, pues se tiene un conjunto de algoritmos de compresión, estáticos y adaptativos, que deben ser intercambiables de acuerdo a las decisiones tomadas por el mecanismo adaptativo o a las opciones emitidas por el usuario.

3.2.1 Compresión adaptativa

El diseño del módulo de compresión, como diagrama de clases UML, se muestra en la Figura 3.1, donde es notable que el servidor únicamente interactúa con la clase abstracta `AdaptiveCompressionStrategy`, la cual debe ser implementada por las clases que modelan diferentes mecanismos de compresión automática y adaptativa. De esta manera, el servidor únicamente configura el objeto `adaptiveCompressionStrategy`

para utilizar el indicado por el usuario, con la lógica de cada uno abstraída en el método `compress()`. Hasta el momento de la finalización del desarrollo del sistema, solo el mecanismo "automático y adaptativo" `RoundRobinCompressor` estaba disponible, como una prueba de concepto, el cual comprime cada trozo de datos con uno de los algoritmos disponibles, de manera equitativa y secuencial. El mecanismo de compresión adaptativa propuesto en este proyecto se agregó, como una implementación de la clase abstracta `AdaptiveCompressionStrategy`, una vez que su diseño fue completado.

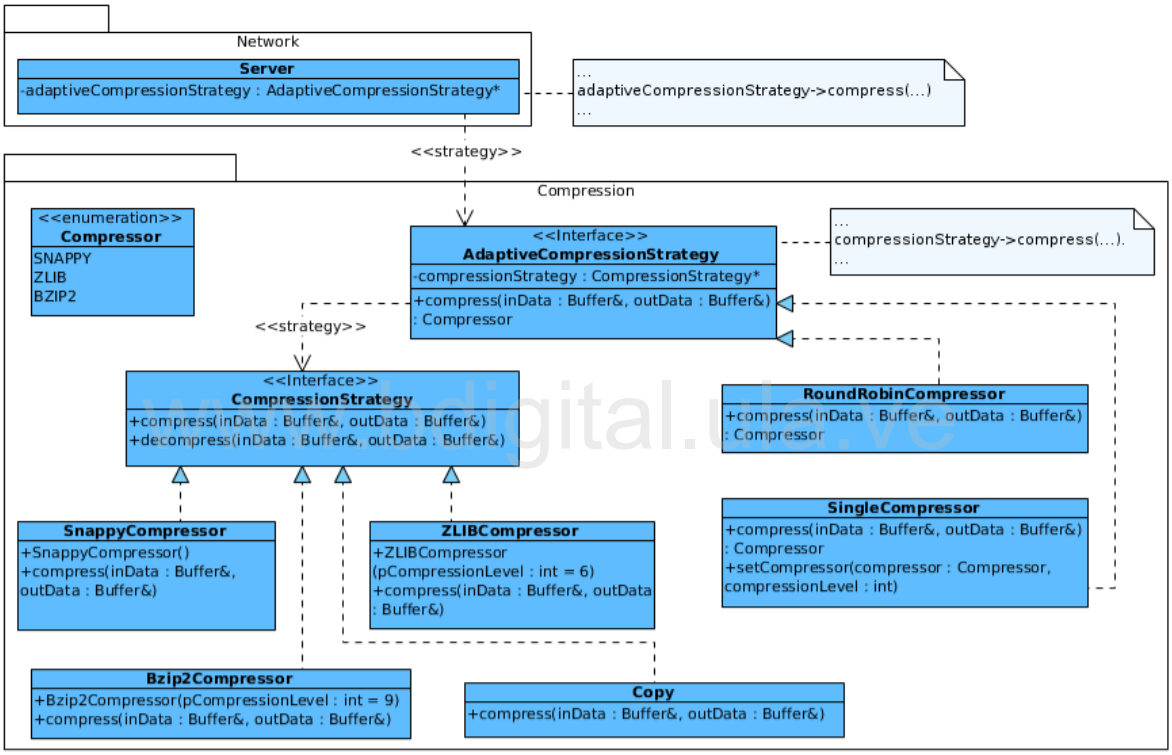


Figura 3.1: Diseño del módulo de compresión del sistema siguiendo el patrón de diseño por estrategias.

También se remarca que el diseño por estrategias fue utilizado de forma "anidada", pues la lógica de un algoritmo de compresión particular se abstrae en el método `compress()` de la clase abstracta `CompressionStrategy`, cuyo comportamiento (comprimir con uno de los compresores en particular) puede ser seleccionado, en tiempo de ejecución, por las clases que implementan los mecanismos de compresión adaptativa. Nótese que, para no modificar el comportamiento del servidor, el no comprimir o "copia" también es considerado un algoritmo o estrategia de compresión (clase `Copy`).

3.2.2 Compresión estática

Para efectos de la evaluación del mecanismo de compresión adaptativa propuesto, surgió un nuevo requerimiento: un mecanismo "adaptativo" que comprima, de forma estática, con el algoritmo especificado por el usuario. Para este fin, se agregó la clase `SingleCompressor` (Figura 3.1), que no paga ningún costo de adaptación y comprime cada trozo con un mismo compresor. Aunque no es un mecanismo adaptativo, se implementa bajo esta estrategia para no proveer una interfaz distinta al servidor para que solicite compresión estática de los datos.

3.3 Capa de transmisión de datos

La capa superior del sistema es una aplicación cliente-servidor, en la cual, un demonio del lado del servidor se encuentra bloqueado hasta que recibe una petición de transmisión de un cliente. La petición requiere la ruta del archivo o directorio que se solicita, así como también un parámetro opcional que indica el tipo de compresor a utilizar; de no recibirse este último parámetro, el sistema utiliza el mecanismo de compresión adaptativa propuesto en este trabajo.

Para soportar compresión y transmisión simultáneas, el servidor utiliza dos hilos para comprimir y transmitir, comunicados a través de una cola en la cual el hilo compresor almacena los trozos de datos a ser consumidos y enviados por el hilo transmisor. El mismo concepto aplica al cliente, en el que la recepción y la descompresión son simultáneas.

3.3.1 Protocolo de transmisión

El proceso de transmisión utiliza los mensajes de la Tabla 3.1 y se da de la siguiente manera:

1. El cliente solicita la transmisión de un archivo o directorio mediante una instancia del mensaje `FileRequestMessage`. El cliente puede especificar el modo, compresor y nivel de compresión a utilizar.

Tabla 3.1: Mensajes utilizados en el protocolo de transmisión de archivos.

Mensaje	Campo	Opcional	Tamaño máximo (bytes)
`FileRequest`	*path*	No	3 + tamaño(*path*)
	mode	Si	
	compressor	Si	
	compressionLevel	Si	
`FileHeader`	*filename*	No	17 + tamaño(*filename*)
	fileSize	No	
	chunkSize	No	
	lastFile	Si	
`ChunkHeader`	*compressor*	No	2
	lastChunk	Si	

2. El servidor recibe la solicitud y, para cada archivo (uno solo si no es un directorio), envía un mensaje de tipo `FileInitialMessage` con el tamaño del archivo y de cada trozo. El último archivo es marcado como `lastFile`.

   2.1. Para cada trozo o *chunk* de un archivo, envía antes una cabecera (`ChunkHeader`). Del mismo modo, el último trozo de cada archivo es marcado como `lastChunk`.

3.3.2 Cliente

El diagrama de actividades UML mostrado en la Figura 3.2, presenta el diseño del programa cliente para el sistema de transmisión de datos desarrollado. Se representan gráficamente, en cada sección del diagrama, la transmisión y descompresión simultáneas.

3.3.3 Servidor

Del mismo modo que para el cliente, el diagrama de actividades de la Figura 3.3 muestra el diseño del programa servidor, en el cual se remarca tanto el proceso de compresión y transmisión simultáneas, como la posibilidad de transmitir una serie de archivos (un directorio) con una misma petición.

Nótese, además, la acción "Comprimir trozo", la cual abstrae cualquier tipo de decisiones que se tomen en el proceso de compresión. Cabe resaltar también que el diagrama no muestra el proceso inherente de configuración del objeto `adaptiveCompressionStrategy`.

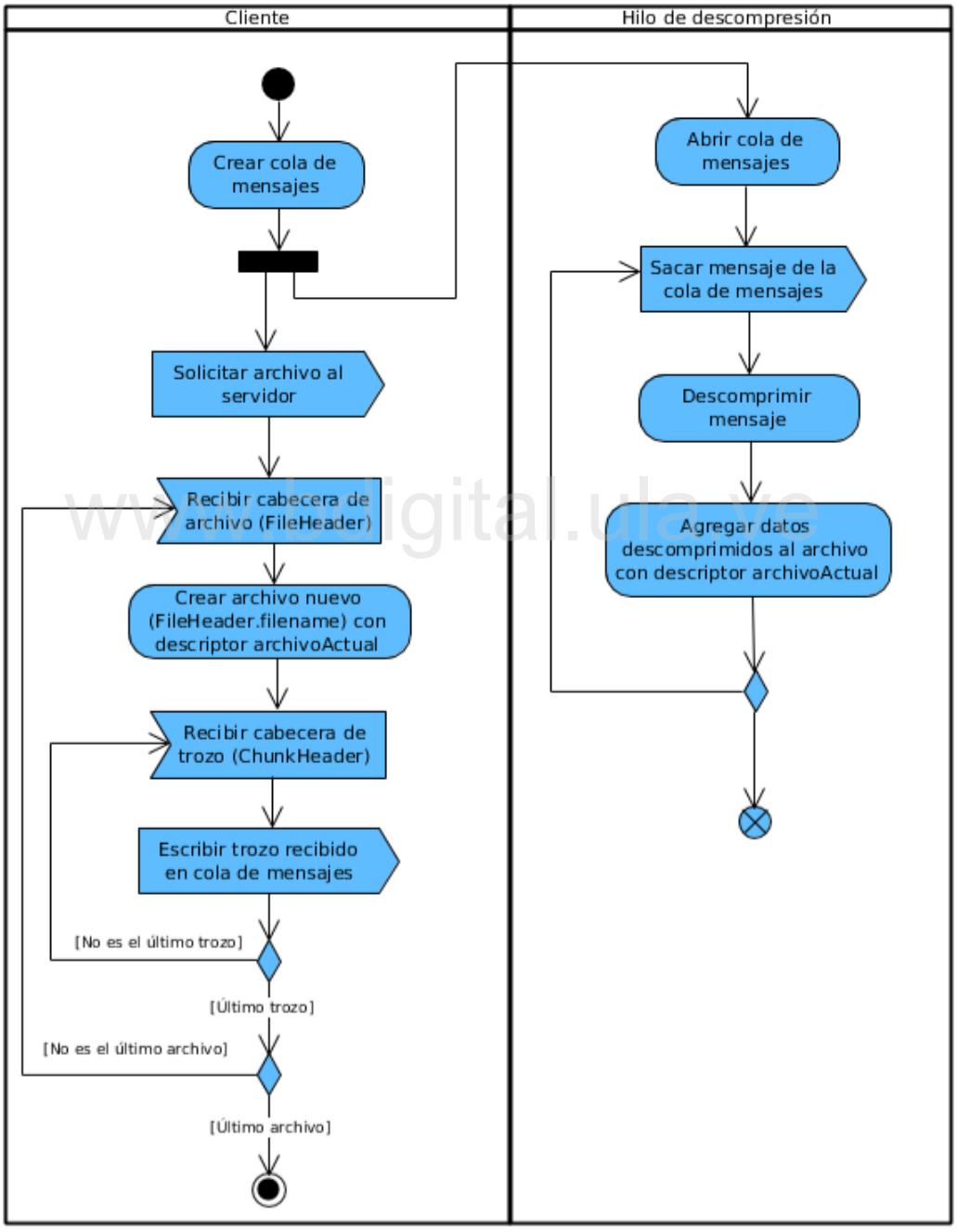


Figura 3.2: Diagrama de actividades para el programa cliente.

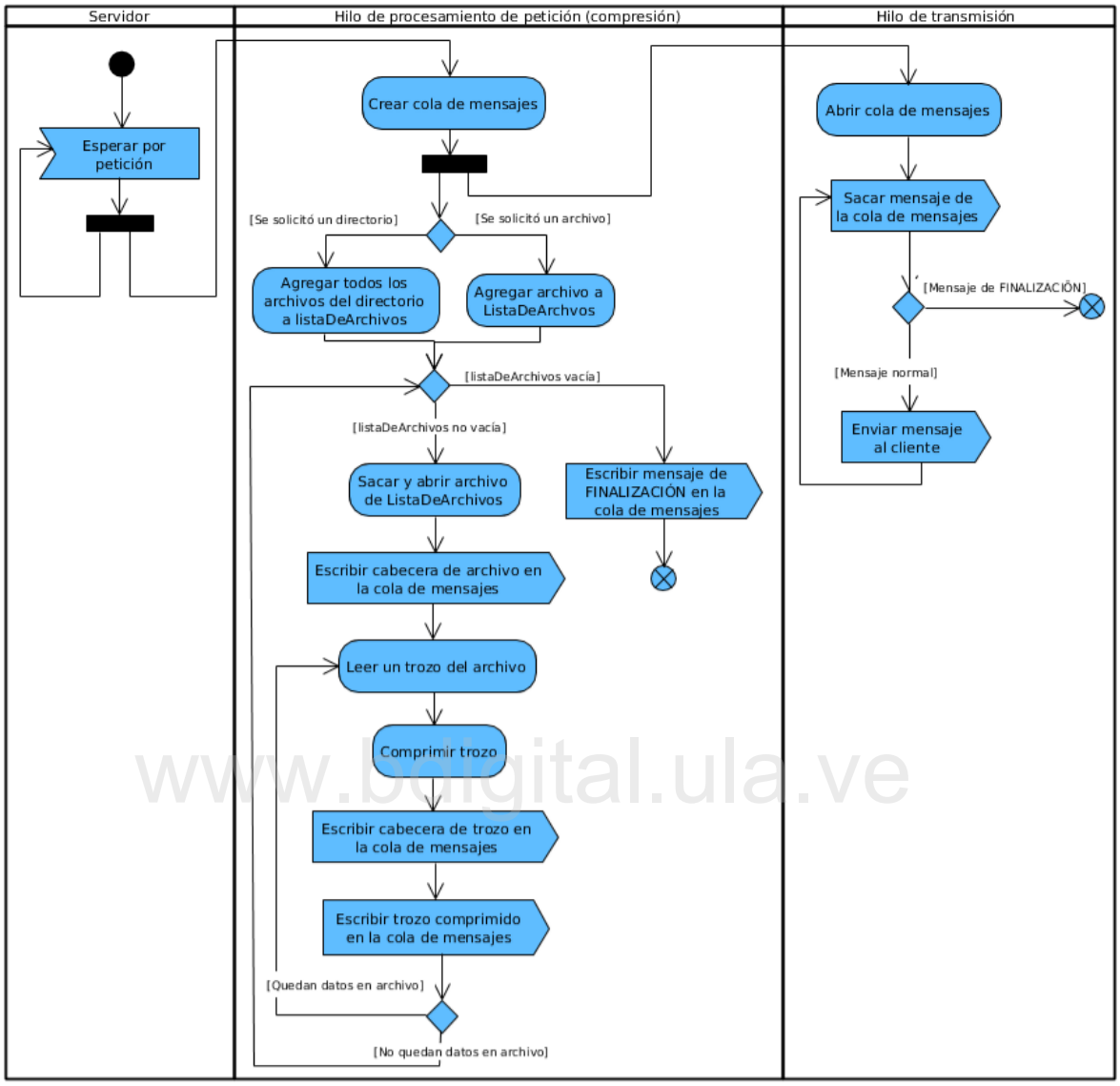


Figura 3.3: Diagrama de actividades para el programa servidor.

Capítulo 4

Mecanismo de compresión automática y adaptativa

En la búsqueda de la optimización de la transmisión de datos, se debe minimizar o maximizar una función objetivo o una medida de rendimiento, tomando en cuenta los factores que afectan el proceso de cómputo y comunicación. El objetivo de una estrategia de compresión automática y adaptativa es, entonces, minimizar o maximizar una función objetivo para así mejorar el rendimiento de la transmisión de información.

En este capítulo se presenta el mecanismo propuesto y la función objetivo a optimizar, así como también los procedimientos para la medición o estimación de los aspectos del entorno que afectan el proceso de compresión y transmisión para su posterior utilización en el proceso de toma de decisiones.

4.1 Compresión adaptativa

La compresión, al ser el proceso de disminuir el número de bits que representan cierta información, es una de las estrategias estudiadas como mecanismo para mejorar el rendimiento de la transmisión de datos, permitiendo disminuir la cantidad de datos siendo transmitidos y, por consiguiente, aumentando el ancho de banda percibido. Sin embargo, el uso de la compresión sin ninguna consideración puede ser perjudicial para las aplicaciones de que sirven de ella.

Muchos sistemas y aplicaciones no utilizan la compresión por el inherente riesgo de degradar el desempeño o la utilizan únicamente cuando se tiene completa certeza de que el cómputo adicional de la compresión mejorará el desempeño de la comunicación o al menos no lo degradará. Otros sistemas se basan en configuración manual para lidiar con el dinamismo de las condiciones de transmisión (normalmente con intervención humana), como es el caso del servidor Web Apache (Apache Software Fundation, 2018); no obstante, es imposible para un humano establecer la configuración óptima lo suficientemente frecuente. Adicionalmente, los algoritmos de compresión se desempeñan de forma distinta dependiendo tanto del estado los recursos subyacentes como de la estructura de los datos siendo comprimidos. A esto se añaden numerosos factores que aumentan el riesgo de utilizar la compresión (o no) sin ningún tipo de consideración.

4.1.1 Aspectos de adaptación

El proceso de adaptación (utilizar el mejor compresor para cierto tipo de datos, ajustar el nivel de compresión o incluso deshabilitar la compresión) debe tomar en cuenta diversos aspectos, entre los cuales se mencionan los siguientes:

- **El tiempo de CPU** disponible en el sistema en un momento determinado, debido a que la compresión requiere poder computacional. El rendimiento de un algoritmo de compresión puede verse degradado cuando el tiempo de CPU disponible es muy limitado (CPU cargado), pudiendo disminuir la tasa de transmisión si la tasa de compresión es menor que la primera. La carga del CPU varía con el tiempo y con la cantidad de procesos en ejecución, por lo cual es un aspecto de adaptación significativo que se debe tomar en cuenta.

- **El estado actual de la red** o el ancho de banda disponible. El dinamismo de este aspecto se encuentra en que, por lo general, la red se comparte con otros usuarios, por lo cual su capacidad varía con el tiempo. En este caso, si la red es muy rápida, podría no haber tiempo suficiente para comprimir; por el otro lado, si la red es muy lenta, podría aprovecharse los recursos de cómputo para reducir la cantidad de datos que se deben transmitir.

- **El tipo de datos** que se transmiten, pues la capacidad de compresión de un algoritmo en particular depende de los datos que procesa. Por ejemplo, el texto (ASCII) es más compresible — y se comprime más rápido — que los datos binarios, principalmente debido a la cantidad máxima de bytes utilizados en su estructura.

- **Los algoritmos de compresión disponibles**, los cuales poseen características que los diferencian y que determinan su desempeño. Algunos compresores sacrifican velocidad para generar salidas muy compactas, mientras que otros están optimizados para comprimir muy rápidamente a costa de tener razones de compresión pobres. Esto hace que ningún compresor sea óptimo en todas las condiciones posibles.

4.1.2 Abstracción de los sistemas de compresión adaptativa

Generalmente, los sistemas de compresión adaptativa buscan optimizar el rendimiento de la transmisión de datos seleccionando, de forma dinámica, el algoritmo de compresión disponible que mejor se desempeñe en cada conjunto de condiciones, lo cual es una tarea altamente retadora debido a los aspectos expuestos en la Sección 4.1.1. Esta similitud entre los sistemas de compresión adaptativa es abstraída por Peterson y Reiher (2016) en cuatro componentes principales: *métodos*, *monitores*, *modelos* y *mecanismos*.

Métodos

Un método se define como un algoritmo de compresión particular del cual se sirve un sistema de compresión adaptativa. Existen diversos algoritmos de compresión y muchos de estos — como el caso de gzip (Gailly y Adler, 2018a) — proveen múltiples métodos o niveles que modulan el poder computacional aplicado a la compresión.

Monitores

Los monitores son los módulos encargados de medir, estimar o predecir información necesaria para el proceso de toma de decisiones, como lo son las propiedades de los

datos y de los recursos subyacentes.

Modelos

Podría decirse que los modelos son el núcleo de los sistemas de compresión adaptativa, pues son los encargados de tomar decisiones a partir de los valores generados u obtenidos por los *monitores*.

Mecanismos

Los mecanismos conglomeran los componentes antes mencionados y definen la capa de abstracción en la que actúa el sistema de compresión, ya sea como una biblioteca de usuario, proxys remotos, kernel, etc.

4.2 Algoritmos de compresión

Como se menciona en la Sección 4.1.2, los algoritmos de compresión conforman el primer componente de un sistema de compresión adaptativa. El objetivo es seleccionar algoritmos con características bien diferenciadas que lo conviertan en el potencial mejor método en ciertas oportunidades. En las siguientes subsecciones, se describen los algoritmos de compresión sin pérdidas utilizados por el mecanismo de compresión adaptativa que se propone en el presente trabajo.

4.2.1 Snappy

Snappy (Google, 2018b) es una biblioteca de compresión sin pérdidas de código abierto desarrollada por Google que apunta a obtener compresión decente a muy altas velocidades. No existe una descripción formal del algoritmo, pero según las notas distribuidas junto con el código fuente, se basa en el algoritmo LZ77 (Ziv y Lempel, 1977) y, al ser comparado con zlib en su nivel más alto, Snappy es una orden de magnitud más rápido, generando, sin embargo, salidas con tamaños que van de 20% a 100% más grandes que los de zlib.

4.2.2 zlib

zlib (Gailly y Adler, 2018b) es una biblioteca de compresión sin pérdidas de código abierto que implementa el algoritmo gzip (Gailly y Adler, 2018a), el cual se basa en el algoritmo DEFLATE, que a su vez es una variación de LZ77 y de la codificación de Huffman (Huffman, 1952). Es ampliamente utilizado — por ejemplo, en el kernel de Linux, el servidor Apache, Git, entre otros — y provee 9 niveles de compresión que permiten ajustar el poder de cómputo requerido y, por consiguiente, la velocidad de compresión, a expensas de capacidad de compresión. Este algoritmo es considerado bien balanceado (Peterson y Reiher, 2016), pues encuentra un equilibro entre su porcentaje de compresión y la tasa a la que procesa los datos.

4.2.3 bzip2

bzip2 (Seward, 2018) es una biblioteca y un programa de compresión sin pérdidas de código abierto que utiliza la transformación de Burrows-Wheeler (Burrows y Wheeler, 1994) y codificación de Huffman. Al igual que zlib, bzip2 provee 9 niveles que regulan la cantidad de memoria que utiliza y, por consiguiente, su capacidad de compresión. A diferencia de Snappy y zlib, bzip2 es más poderoso, con porcentajes de compresión mucho mayores pero a tasas muy reducidas.

4.2.4 Estudio comparativo

Los métodos antes mencionados (snappy, zlib y bzip2) fueron seleccionados debido a las características que los diferencian unos de otros, lo cual puede convertir a cada uno en el mejor compresor en diferentes oportunidades. Snappy es un compresor que optimiza la velocidad, con porcentajes de compresión decentes; bzip2 permite obtener porcentajes de compresión muy altos a tasas muy bajas; y zlib, por otro lado, es un punto medio entre snappy y bzip2. La Figura 4.1 muestra un diagrama de dispersión del porcentaje de compresión o *compression ratio* contra el tiempo de compresión para un conjunto de archivos tomados de los corpus de compresión de Calgary (Bell et al., 1989) y Canterbury (Universidad de Canterbury, 2018). El diagrama muestra cómo las nubes de puntos tienden a trasladarse más hacia los infinitos de ambos ejes (más

tiempo de compresión, más ganancia) al pasar de compresor a compresor según sus características ya mencionadas.

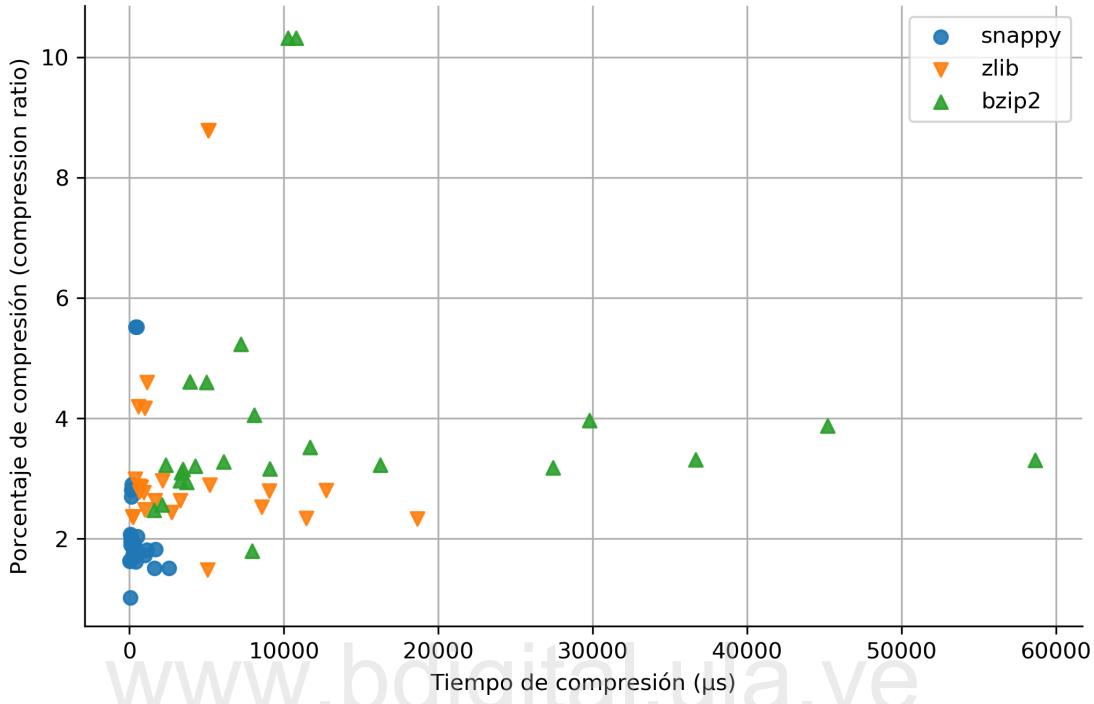


Figura 4.1: Estudio comparativo de las capacidades de los métodos de compresión seleccionados.

4.3 Monitores

En la Sección 4.1.1 se trataron solo algunos de los aspectos que se deben considerar para la toma de decisiones en sistemas de compresión adaptativa. Desafortunadamente, es imposible monitorear o estimar todos los aspectos posibles que afectan el proceso de transmisión y compresión, pues se correría el riesgo de degradar el desempeño solo en la etapa de monitoreo. Por lo tanto, es necesario elegir un conjunto de factores que permitan modelar las propiedades del entorno y de los datos en un instante determinado. En este trabajo, se va a tratar con tres aspectos que se consideran de gran importancia y que podrían modelar de forma eficiente el entorno: la carga del CPU de la máquina que transmite, el estado de la red y el tipo de datos siendo transmitidos.

4.3.1 Carga del CPU en el transmisor

La carga del CPU es una medida que determina indirectamente el tiempo de CPU disponible para operar en un instante determinado. Si la carga es alta en un intervalo de tiempo determinado, significa que el CPU ha estado siendo utilizado intensivamente por otros procesos. La carga del CPU es calculada utilizando el método mostrado en la Sección 2.2.1 como el porcentaje de tiempo que el CPU ha sido utilizado efectivamente en en último intervalo de tiempo, de tal manera que la Ecuación 2.5 se reescribe como en la Ecuación 4.1.

$$Uso\ del\ CPU = \frac{\Delta t_{cpu\_total\_uso}}{\Delta t_{cpu\_total}} \tag{4.1}$$

El monitor del CPU es un proceso independiente que reporta el valor en un segmento de memoria compartida. Los valores necesarios para el cálculo de la carga de CPU son leídos del sistema de archivos `/proc` (sistema de archivos de procesos) de los sistemas operativos Unix, específicamente del archivo `/proc/stat`, en el cual se encuentran, en tiempo real, estadísticas y métricas del sistema desde que se inició, incluyendo los tiempos que el CPU estuvo realizando diferentes tipos de tareas. Se debe seleccionar un intervalo lo suficientemente corto para seguir el ritmo al dinamismo de las condiciones pero tomando en cuenta que el cálculo requerido para monitorear la carga del CPU puede cargarlo en sí mismo si se calcula con una frecuencia muy alta. En este trabajo se seleccionó un intervalo de 0,5 segundos para este fin.

4.3.2 Estado de la red

Se decidió utilizar el ancho de banda disponible (ABD) como métrica para determinar el estado del canal de transmisión. Para esto, se estima el ABD midiendo la tasa a la cual el kernel envía los datos del socket. El Algoritmo 4.1 muestra, en pseudocódigo, el procedimiento para la estimación del ABD.

La técnica que se utilizó para estimar el ABD (que se muestra en el Algoritmo

Algoritmo 4.1 Estimación del ancho de banda disponible

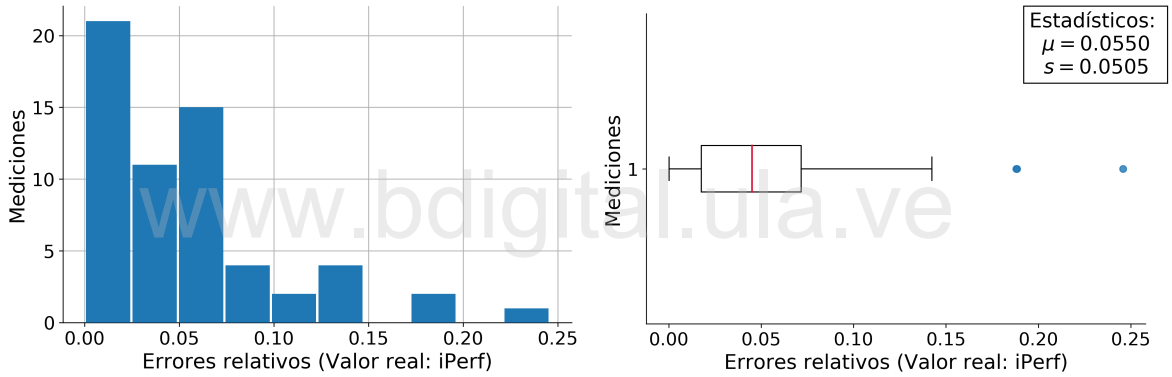
```
1: procedimiento EstimarABD(socket)
2:     t_i ← obtener_tiempo_actual()
3:     bytes_en_buffer_en_t_i ← 0
4:     mientras true hacer
5:         datos ← esperar_datos_para_enviar()
6:         t_f = obtener_tiempo_actual()
7:         Δt ← t_f − t_i
8:         si Δt ≥ UMBRAL entonces
9:             bytes_en_buffer_en_t_f ← obtener_bytes_en_buffer(socket)
10:            ABD ← (bytes_en_buffer_en_t_f − bytes_en_buffer_en_t_i) / Δt
11:            escribir_datos_a_enviar(datos, socket)
12:            bytes_en_buffer_en_t_i ← obtener_bytes_en_buffer(socket) + tamaño(datos)
13:            t_i ← obtener_tiempo_actual()
14:        si no
15:            escribir_datos_a_enviar(datos, socket);
16:            bytes_en_buffer_en_t_i ← bytes_en_buffer_en_t_i + tamaño(datos)
17:        fin si
18:    fin mientras
19: fin procedimiento
```


4.1) consiste en rastrear las adiciones al buffer de escritura del socket y los tiempos correspondientes para posteriormente utilizar la llamada al sistema `ioctl` (UNIX) con el parámetro `SIOCOUTQ` para obtener la cantidad de bytes almacenados en el buffer de escritura en un instante determinado y así estimar el ABD. En el algoritmo, la funciones `obtener_bytes_en_buffer()` y `escribir_datos_a_enviar()` corresponden a la llamada al sistema `ioctl` y a la función `send()` de los sockets, respectivamente. En cuanto a la constante `UMBRAL`, es definida como el intervalo de tiempo mínimo entre estimaciones del ancho de banda, pues se debe tener cierta cantidad de datos acumulados en el buffer para obtener una medición válida del ABD; este umbral fue definido en 10 ms, con la suposición de tener un flujo de datos continuo dado que se desea mejorar la eficiencia de la transmisión.

Validación

Con la intención de validar el algoritmo de estimación del ABD, se realizó la siguiente prueba: utilizando el sistema de transmisión de datos desarrollado en modo copia, se midió el ancho de banda para 60 transmisiones de 60 segundos de duración

cada una — tomando el promedio de los valores reportados por el sistema en cada transmisión — con diversos límites de ancho de banda. Para contrastar, se realizaron las mismas pruebas, en igualdad de condiciones, con el software *iPerf* (iPerf, 2018) que mide el ABD entre dos interfaces de red. El error relativo entre las mediciones fue calculado, tomando como valores reales los reportados por *iPerf*, resultados que pueden observarse en la Figura 4.2. En la Figura 4.2(a) se puede observar que, para la mayoría de las mediciones, el error relativo es muy cercano al 0, lo cual se confirma en el digrama de caja de la Figura 4.2(b), el cual muestra solo 3 valores atípicos, con una media del 5.5%, desviación estándar de 5.05% y el 50% de las observaciones entre el 1.8% y 6.8%. Con esta información, es posible aceptar las mediciones del método propuesto como válidas.



(a) Histograma de frecuencia de aparición de errores relativos.

(b) Diagrama de caja del error relativo de las mediciones.

Figura 4.2: Pruebas de validación del método de estimación del ancho de banda disponible.

4.3.3 Tipo de datos

Existen diversos métodos para estimar las propiedades o la compresibilidad de los datos: se podría utilizar el contexto para este fin, como el formato o extensión de los archivos, mas sin embargo esto ataría al sistema al concepto de archivo que, adicionalmente, debería tener extensión, lo cual no es un escenario completamente realista; se puede realizar un muestro de los datos a transmitir y comprimirlos con un algoritmo lo suficientemente rápido y observar el porcentaje de compresión,

pero esto conllevaría pérdidas cuando la compresión realizada sea descartada; Harnik et al. (2013) proponen heurísticas, tanto para pre-compresión como para compresión en línea, basadas en entropía y la cantidad de bytes únicos que representan los datos; no obstante, Peterson y Reiher (2016) proponen una técnica muy sencilla pero igualmente eficiente y poderosa para estimar no las propiedades de los datos, sino su compresibilidad, la cual se describe a continuación.

Bytecounting

La técnica propuesta por Peterson y Reiher (2016) (la cual es utilizada en el presente trabajo para estimar la compresibilidad), se denomina *bytecounting* (BC) o "conteo de bytes" (aunque su traducción literal al español no refleja su esencia) y estudia la distribución de bytes únicos en el archivo a comprimir. El BC es un número entero que indica la cantidad de bytes en la entrada que aparecen al menos N veces, donde N es un umbral que depende del tamaño de los datos de entrada y se define como $N = tamaño(datos)/256$. En otras palabras, el BC indica el número de bytes que aparecen al menos la cantidad de veces que deberían aparecer si todos los bytes posibles ($2^8 = 256$) estuvieran uniformemente distribuidos. El BC, entonces, permite medir la uniformidad de los bytes que representan la entrada. El principio del cálculo del BC es presentado en el Algoritmo 4.2.

Algoritmo 4.2 Cálculo del *bytecounting*

```
1: función BYTECOUNTING(datos)
2:     ocurrencias[256] ← {0, 0, ..., 0}
3:     BC ← 0
4:     umbral ← tamaño(datos) / 256
5:     para byte ∈ datos hacer
6:         ocurrencias[byte] ← ocurrencias[byte] + 1
7:     fin para
8:     para ocurrence ∈ ocurrences hacer
9:         si ocurrence ≥ umbral entonces
10:            BC ← BC + 1
11:        fin si
12:    fin para
13:    retornar BC
14: fin función
```

Mientras menor sea el BC, menor la cantidad de caracteres que relativamente representan los datos de entrada; valores de BC en el extremo superior indican que la entrada está compuesta por la mayoría de todos los bytes posibles. Para ejemplificar, un BC de 1 indica que la entrada esta representada virtualmente por un único byte, lo cual la hace altamente compresible. Un valor de BC de 127 indica que la mitad de los bytes posibles aparecen frecuentemente, lo cual, al igual que valores de BC mayores, da indicios de que los bytes se encuentran distribuidos uniformemente, la cual es una propiedad típica de datos incompresibles.

Validación del *bytecounting*

Para mostrar y validar lo que el BC representa, se llevó a cabo una prueba en la que se comprimieron los mismos archivos utilizados en el estudio comparativo de los algoritmos de compresión en la Sección 4.2.4, a los cuales se añadieron 5 archivos de 1 MB cada uno con datos aleatorios extraídos del dispositivo `/dev/urandom` (Linux), una serie de archivos multimedia personales del autor y 1 archivo de 1 MB con un solo byte repetido (extraído del dispositivo `/dev/zero`). La Figura 4.3 muestra un gráfico de la relación entre el porcentaje de compresión y el BC para cada uno de los archivos, donde se observa que, efectivamente, el porcentaje de compresión tiende a 1 (cero ganancia) cuando el BC aumenta; de hecho, la ganancia de compresión es nula para valores de BC entre 100 y 256. Es decir, si la distribución de los bytes es prácticamente uniforme, no se obtiene ganancia alguna al realizar la compresión.

4.4 Modelo adaptativo

El modelo propuesto se basa en dos principios propuestos por Peterson y Reiher (2016):

1. El concepto de **oportunidad de compresión**, que definen como el conjunto de condiciones (propiedades de los datos y estado del entorno) que determinan el desempeño de un método de compresión en un instante determinado.

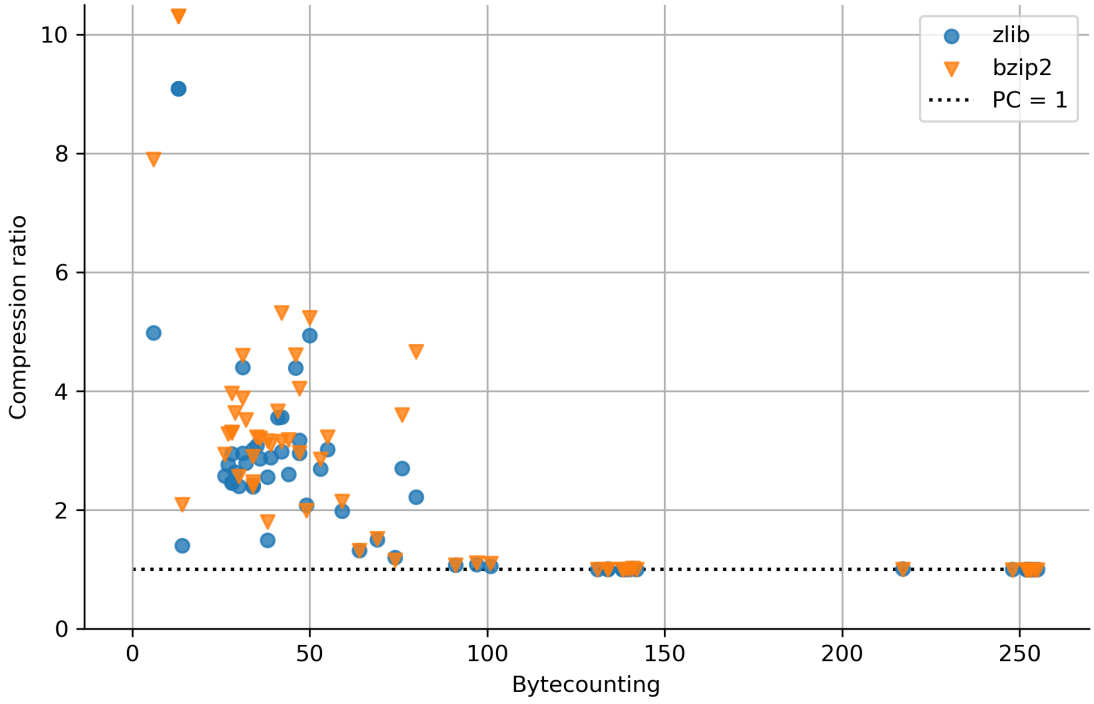


Figura 4.3: Relación entre el porcentaje de compresión (PC) y el valor del *bytecounting*.

2. El supuesto de que el desempeño de un algoritmo de compresión en una oportunidad dada es constante, debido al comportamiento determinista de los mismos.

Si se puede determinar el espacio de oportunidades y el conjunto de aspectos que definen una oportunidad, es viable entonces conocer el mejor compresor en cada oportunidad posible.

4.4.1 Función objetivo

El mecanismo de compresión adaptativa propuesto tiene el objetivo de seleccionar el mejor algoritmo de compresión en una oportunidad dada. El reto no es solo saber cuál es el mejor compresor, sino por qué lo es y como decidirlo.

El ancho de banda disponible determina la tasa a la cual se transmiten los datos a través de una canal de comunicación. No obstante, la tasa a la cual se transmite la **información** puede ser mayor que el ABD. Cuando se habilita la compresión, es posible

transmitir la misma información representada con menos datos, lo que aumentaría el ABD percibido. Esto se conoce como Tasa Efectiva de Transmisión (TET) (Peterson y Reiher, 2016) y se modela con la Ecuación 4.2.

$$TET = mín(ABD, TC) * PC \tag{4.2}$$

donde:

ABD = Ancho de banda disponible
TC = Tasa o velocidad de compresión
PC = Porcentaje de compresión o *compression ratio*

En la Ecuación 4.2, si se comprime a velocidades menores al ABD, la TET es definida por la velocidad de compresión, mientras que si se comprime más rápido de lo que se pueden transmitir los datos, la TET es definida por el ABD. De igual manera, el factor PC indica la cantidad de información (no de datos) que se transmiten, sin importar la cantidad de bytes que la representan — es decir, si se obtiene un PC de 2, se transmite la misma cantidad de información pero con la mitad de los bytes que ciertamente se requieren para su representación original. De este modo, el objetivo del mecanismo propuesto se convierte en encontrar y seleccionar, en una oportunidad dada, el algoritmo de compresión que provea la mayor TET.

4.4.2 Cuantización del espacio de oportunidades

Determinar el espacio de oportunidades (ABD, carga del CPU y BC) a alta resolución es abrumador y, debido a la continuidad de algunos de ellos, sería prácticamente imposible determinar una oportunidad de manera exacta. Por lo tanto, se propone cuantizar o discretizar el espacio de oportunidades, como lo proponen Peterson y Reiher (2016), pero a mayor resolución, debido al tipo de modelo de toma de decisiones que se utiliza en este trabajo.

La Tabla 4.1 muestra los niveles de cuantización seleccionados. La carga del CPU se dividió en intervalos de 10%; el ABD se dividió en intervalos de 5 Mbits/s para valores menores a 100 Mbits/s, mientras que para valores entre 100 Mbits/s y 1 Gbit/s los niveles se dividen en intervalos de 100 Mbit/s; el BC es un caso especial, dividido en

Tabla 4.1: Niveles de cuantización del espacio de oportunidades.

Nivel	Carga del CPU	ABD	BC
0	[0%, 10%)	(0 Mbit/s, 5 Mbit/s)	[1, 10)
1	[10%, 20%)	[5 Mbit/s, 10 Mbit/s)	[10, 20)
⋮	⋮	⋮	⋮
9	[90%, 100%)	[45 Mbit/s, 50 Mbit/s)	[90, 100)
10	100%	[50 Mbit/s, 55 Mbit/s)	–
⋮	–	⋮	
19		[95 Mbit/s, 100 Mbit/s)	
20		[100 Mbit/s, 200 Mbit/s)	
21		[200 Mbit/s, 300 Mbit/s)	
⋮		⋮	
28		[900 Mbit/s, 1 Gbit/s]	

intervalos de 10, únicamente para valores menores a 100. Por los resultados encontrados y mostrados en la Sección 4.3.3, el mecanismo de compresión ignora las entradas con BC mayor o igual a 100 y los envía sin comprimir.

4.4.3 Proceso de toma de decisiones

La decisión de seleccionar el mejor algoritmo de compresión en una oportunidad dada, puede verse como una tarea de clasificación, tal y como se le conoce en el ámbito del aprendizaje automatizado. Para esto, es necesario conocer, en la instalación del sistema, la información del desempeño de cada compresor en cada oportunidad posible.

Conjunto de datos de entrenamiento

Para recaudar los datos de entrenamiento, se realizaron transmisiones de un conjunto de archivos de prueba extraídos de los corpus de Calgary, Canterbury y Silesia (Universidad Politécnica de Silesia, 2018), a los cuales se añadieron cinco archivos de 1 MB, cada uno, de datos aleatorios, cinco archivos de 1 MB, cada uno, con un único byte repetido y un conjunto de archivos multimedia propios del autor, totalizando 146 MB.

En total, se realizaron transmisiones de los archivos con cada compresor (incluyendo compresión "nula" o copia), bajo 16 límites de ancho de banda — modulado con el

Tabla 4.2: Porcentaje de clasificación correcta de los clasificadores estudiados.

	Porcentaje de clasificación correcta	
Clasificador	**Entrenamiento**	**Prueba**
SVM	92,9%	85,4%
Árbol de decisión	91,4%	84,7%
AdaBoost	100%	85,4%
Bayesiano ingenuo	69,3%	68,8%

software *Wondershaper* (Hubert et al., 2018) — y 5 niveles de carga artificial al CPU, para un total de 320 transmisiones, las cuales se llevaron a cabo 10 veces, cada una, para tener datos lo suficientemente correctos. En cada oportunidad, fueron registrados la TET y los datos que definen la oportunidad en ese instante para posteriormente extraer, para cada oportunidad, el compresor con la mayor TET promedio.

El clasificador

Cuatro tipos de clasificadores fueron estudiados, siendo estos las máquinas de vectores de soporte (SVM, por sus siglas en inglés), árboles de decisión, AdaBoost (con árboles de decisión como clasificadores débiles) y un bayesiano ingenuo. Las exactitudes de prueba y entrenamiento, obtenidos dividiendo los datos disponibles en datos de entrenamiento y prueba en proporción 9:1, se muestran en la Tabla 4.2.

La diferencia de resultados en la Tabla 4.2 entre el bayesiano ingenuo y los demás clasificadores es suficiente como para descartarlo. Los mejores resultados se obtuvieron con el SVM y el AdaBoost, mas sin embargo, con el árbol de decisión se obtuvieron resultados prácticamente idénticos a los del SVM. La Figura 4.4 muestra las matrices de confusión para los clasificadores. De nuevo, se reafirma la decisión de prescindir del bayesiano ingenuo. Los mejores resultados se ven con el AdaBoost, con porcentajes de exactitud sobre el 80% en todas las clases. Sin embargo, por la diferencia de complejidad de éstos y de los SVM multi-clase con respecto al árbol de decisión, tanto en funcionamiento como en implementación, se decidió utilizar este último como clasificador para este problema. Se debe recordar que es necesario un proceso de toma de decisiones lo suficientemente rápido y sencillo para no agregar *overhead* innecesario a la transmisión de datos.

Figura 4.4: Matrices de confusión de los clasificadores estudiados.

4.4.4 Algoritmo de compresión automática y adaptativa

Es momento de definir el algoritmo de compresión automática propuesto, el cual se muestra en el Algoritmo 4.3, sobre el cual se resaltan algunas características:

- Cuando se solicita la compresión de un trozo de datos incompresibles ($BC \geq 100$), el algoritmo envía los siguientes 512 KB sin costo de adaptación.
- Cuando el porcentaje de ocupación del buffer de salida del socket es menor al 5%, lo cual ocurre principalmente al comienzo de la transmisión, se comprime con zlib. Esto se decidió debido a que al comienzo de la transmisión no se tiene información para el clasificador, por lo tanto se debe utilizar un compresor equilibrado para no reducir la ganancia ni degradar el desempeño a bajos y altos ABD, respectivamente.

- Se debe recordar que el clasificador puede decidir utilizar "copia" como el mejor compresor, por lo cual el método `comprimir` encapsula la abstracción del compresor "copia" y retorna los datos originales inmediatamente.

Algoritmo 4.3 Algoritmo de compresión adaptativa propuesto

```
1: función ComprimirAdaptativamente(datos)
2:     bytesAEnviarSinComprimir ← 0                                  ▷ Variable estática
3:     porcentajeDatosEnBuffer ← 0                                   ▷ Variable estática
4:     si bytesAEnviarSinComprimir > 0 entonces
5:         bytesAEnviarSinComprimir ← bytesAEnviarSinComprimir − tamaño(datos)
6:         retornar datos                                            ▷ No comprimir
7:     fin si
8:     BCActual ← Bytecounting(datos)
9:     si BCActual ≥ 100 entonces
10:        bytesAEnviarSinComprimir ← 512 ∗ 1024
11:        retornar datos                                            ▷ No comprimir
12:    fin si
13:    porcentajeDatosEnBuffer ← obtener_porcentaje_ocupación_bufer_socket()
14:    si porcentajeDatosEnBuffer < 5% entonces
15:        datosComprimidos ← comprimir(datos, ZLIB)
16:        retornar datosComprimidos
17:    fin si
18:    cargaCPU ← leer_carga_cpu()
19:    ABD ← leer_abd()
20:    compresor ← clasificar(cargaCPU, ABD, BCActual)
21:    datosComprimidos ← comprimir(datos, compresor)
22:    retornar datosComprimidos
23: fin función
```

Capítulo 5

Evaluación del mecanismo de compresión

Con el sistema de transmisión desarrollado y el mecanismo de compresión adaptativa diseñado, implementado e integrado al sistema de transmisión, se presentan en este capítulo las pruebas realizadas para evaluar el rendimiento y las capacidades adaptativas del sistema en cuestión.

5.1 Prueba de adaptabilidad

El mecanismo de compresión propuesto debe ser adaptativo, por lo cual, para probar si efectivamente se adapta a cambios en el entorno, se llevaron a cabo dos pruebas sencillas que muestran la adaptabilidad del mecanismo ante cambios en el estado de la red.

5.1.1 Diseño del experimento

Para la prueba de adaptabilidad, se utilizó un conjunto de 1000 archivos de 1 MB, cada uno, totalizando 1 GB, compuestos de un único byte repetido. Los experimentos se describen a continuación:

1. Los datos fueron transmitidos a un límite de 1 Gbit/s y, en cierto punto de la transferencia, se limitó el ancho de banda a 15 Mbits/s para,poco después,

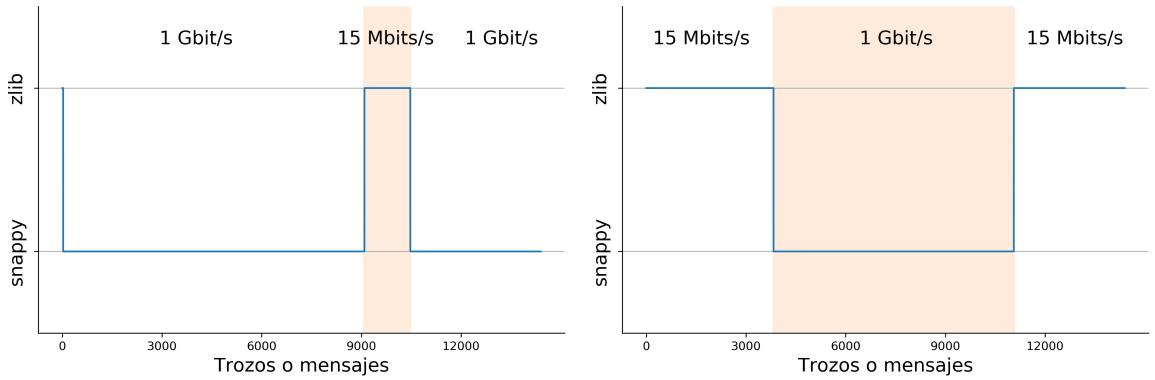
restablecerlo a 1 Gbit/s. En este caso, se espera que la compresión pase de realizarse mediante el compresor más rápido (snappy) o sin comprimir en absoluto cuando el ancho de banda es superior, a realizarse a través un método computacionalmente intensivo con altos porcentajes de compresión como bzip2 o zlib cuando el ancho de banda es menor.

2. El segundo experimento fue similar: la transmisión comenzó a 15 Mbits/s, en cierto punto se moduló a 1 Gbit/s y se restableció a 15 Mbits/s. El resultado esperado es, ciertamente, el reverso del que se espera en el primer experimento.

5.1.2 Resultados

La Figura 5.1 muestra los resultados de los experimentos realizados. Según la Figura 5.1(a), el mecanismo de compresión adaptativa, en efecto, reacciona ante una variación de ancho de banda de alto a bajo seleccionando mayoritariamente un método con mejores porcentajes de compresión, siendo, en este caso, zlib; en esta situación, el uso de zlib en unos cuantos mensajes al comienzo de la transmisión se explica en la Sección 4.4.4. Del mismo modo, en la Figura 5.1(b) se observa cómo, ante variaciones del ancho de banda de bajo a alto, el mecanismo selecciona un método más ligero (snappy). En ambos casos se observa el mismo fenómeno: para altos anchos de banda, el sistema reacciona disminuyendo el costo computacional al usar un método más ligero, mientras que para anchos de banda bajos, se utiliza intensivamente el CPU en tareas de compresión para reducir el volumen de información transmitida.

Un aspecto muy importante que se extrae de los resultados de estas pruebas, es el hecho de que, para el nivel más alto de ancho de banda disponible, el sistema no decide no comprimir, sino que utiliza snappy, lo cual da indicios de que la rapidez de este algoritmo es tal, que para que su utilización se convierta en un cuello de botella, se debe contar con anchos de banda mayores a 1 Gbit/s.



(a) Variación de ABD de alto a bajo.

(b) Variación de ABD de bajo a alto.

Figura 5.1: Pruebas de adaptabilidad del mecanismo de compresión propuesto ante cambios en el ancho de banda.

5.2 Pruebas de rendimiento

El principal motivo de un mecanismo de compresión adaptativa es mejorar el rendimiento de la transmisión de información mediante la compresión, cuando esta sea beneficiosa, de acuerdo al tipo de información que se transmite y al estado de los recursos computacionales y de comunicación subyacentes. La mayoría de los sistemas de transmisión de datos no utiliza compresión o la utiliza de manera estática, es decir, utilizando un mismo algoritmo durante toda la transmisión (Peterson y Reiher, 2016). Debido a esto, se decidió estudiar el rendimiento del mecanismo propuesto ante compresión estática (un algoritmo fijo o no comprimir en absoluto).

5.2.1 Descripción del entorno

Las pruebas fueron realizadas en una LAN gigabit privada, con el transmisor ejecutándose en un procesador Intel i7-5820K y el receptor en un Intel i5-4690, con 6 y 4 núcleos de CPU disponibles, respectivamente. Para modular el ancho de banda disponible se utilizó el software *Wondershaper* (Hubert et al., 2018). La carga del CPU, por otro lado, fue modulada lanzando procesos que, de forma continua, leen 32 KB de datos del archivo especial `/dev/urandom` (Linux), los comprimen con zlib y escriben el resultado en `/dev/null`; se realizaron pruebas con 0, 2, 3 y 5 procesos de carga.

5.2.2 Datos de prueba

Para representar un amplio rango de tipos de información con características variadas, fueron recolectados 5 tipos de datos con variados porcentajes de compresión esperados: (*a*) datos tipo *código*, conformados por una recopilación de código fuente de varias bibliotecas y *frameworks* libres y de código abierto, totalizando 74 MB; (*b*) datos tipo *latex*, conformados por un conjunto de códigos LaTeX de proyectos (tesis, papers, etc.), para un total de 71 MB; (*c*) datos tipo *multimedia*, los cuales consisten en una serie de 76 MB de archivos multimedia (imágenes, audio, videos, archivos pdf, etc.) propios del autor; y por último, los datos tipo (*d*) *cero* (60 MB) y (*e*) *aleatorio* (60 MB), los cuales consisten en 60 archivos de 1 MB, cada uno, de un byte repetido (tomado de `/dev/zero`) y datos aleatorios (tomados de `/dev/urandom`), respectivamente.

Los datos tipo *cero* y *aleatorio* representan los extremos en cuanto a ganancia de compresión, siendo el mejor y peor caso, respectivamente. Los datos tipo *código* se componen, en su mayoría, de archivos de texto plano (código fuente), que es altamente compresible. Por otro lado, los datos tipo *multimedia* son muy poco compresibles debido a que, la mayoría, se almacena en formatos pre-comprimidos (mp3, jpg, etc.). Los datos tipo *latex*, por su parte, representan el punto intermedio, con una relación entre imágenes (poco compresible) y texto (muy compresible) relativamente equilibrada.

5.2.3 Diseño del experimento

El experimento consistió en una serie de transmisiones, para cada conjunto de datos descritos en la Sección 5.2.2, en diferentes condiciones (límite de ancho de banda y carga del CPU) y utilizando el mecanismo de compresión adaptativa propuesto, las cuales fueron cronometradas. Adicionalmente, y en igualdad de condiciones, las mismas transmisiones fueron realizadas utilizando compresión estática — utilizando snappy, zlib y bzip2 — y sin comprimir (copia). Los experimentos fueron realizados un total de 10 veces para tener datos lo suficientemente representativos.

Cada método de compresión (los estáticos y el adaptativo) fue comparado con

el caso por defecto (copia), para medir la ganancia de cada uno en las diferentes oportunidades. Para esto, se calculó, para cada método, la ganancia, expresada en porcentaje, con respecto a la transmisión sin compresión, con la Ecuación 5.1.

$$GP = \frac{TT_c - TT_m}{TT_c} \times 100 \tag{5.1}$$

donde:

GP = Ganancia porcentual respecto a la transmisión sin compresión
TT_c = Tiempo de transmisión sin compresión (o copia)
TT_m = Tiempo de transmisión con el método m

En este sentido, un valor de GP de 50% significa que el método en cuestión llevó el tiempo de transmisión a la mitad, por lo cual un GP de 100% es imposible, debido a que significaría que el tiempo de transmisión fue de 0. Del mismo modo, un GP de -50% significa que con el método en cuestión, la transmisión de realizó en 1,5 veces el tiempo que llevó la transmisión sin compresión, es decir, que el tiempo de transmisión aumentó en la mitad, mientras que un GP de -100% indica que el tiempo de transmisión aumento el doble.

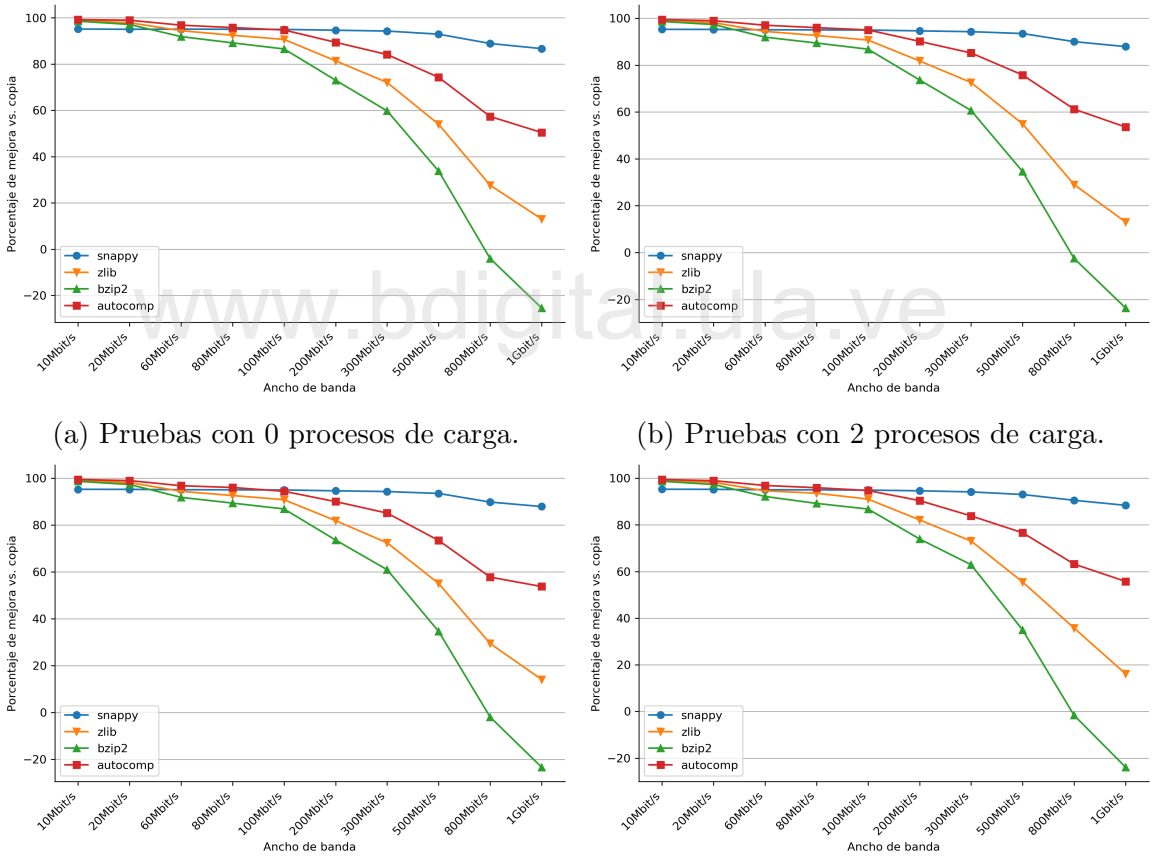
5.2.4 Resultados

Las siguientes gráficas muestran los resultados del experimento diseñado, en las que las líneas correspondientes a `snappy`, `zlib` y `bzip2` denotan la mejora obtenida por dichos métodos estáticos, mientras que `autocomp` denota la mejora obtenida por el mecanismo de compresión automática y adaptativa propuesto en este trabajo.

Datos tipo *cero*

Los datos tipo *cero* (aquellos consistentes de un único byte repetido) corresponden al mejor caso y es donde se deberían obtener los resultados de mejor ganancia a través de la compresión. La Figura 5.2 muestra los resultados para los datos tipo *cero*, donde se puede observar que el método adaptativo propuesto supera el desempeño de los tres métodos estáticos para anchos de banda de hasta 100 Mbits/s, disminuyendo el tiempo de transmisión incluso en 99% a 10 Mbits/s. No obstante, snappy supera a `autocomp`

para anchos de banda sobre los 100 Mbits/s. El que `autocomp` supere tanto a zlib como a bzip2 pero no a snappy a altos anchos de banda, indica que (*a*) se está usando snappy intensivamente, (*b*) posiblemente se esté intercalando con el método "copia" y/o (*c*) se está pagando costosamente el *bytecounting* y la clasificación. Según los resultados de las pruebas de adaptabilidad (Sección 5.1.2), la respuesta al anterior cuestionamiento se encuentra en los items *a* y *c*: sí se está tomando la decisión acertada, pero se está pagando costosamente el *bytecounting* y la clasificación. Debido a estos resultados, se espera que, probablemente, snappy supere a `autocomp` mientras los datos sean lo suficientemente compresibles a altos anchos de banda.



(a) Pruebas con 0 procesos de carga.

(b) Pruebas con 2 procesos de carga.

(c) Pruebas con 3 procesos de carga.

(d) Pruebas con 5 procesos de carga.

Figura 5.2: Porcentaje de mejora de cada método relativo a no comprimir para datos tipo *cero*.

Datos tipo *aleatorio*

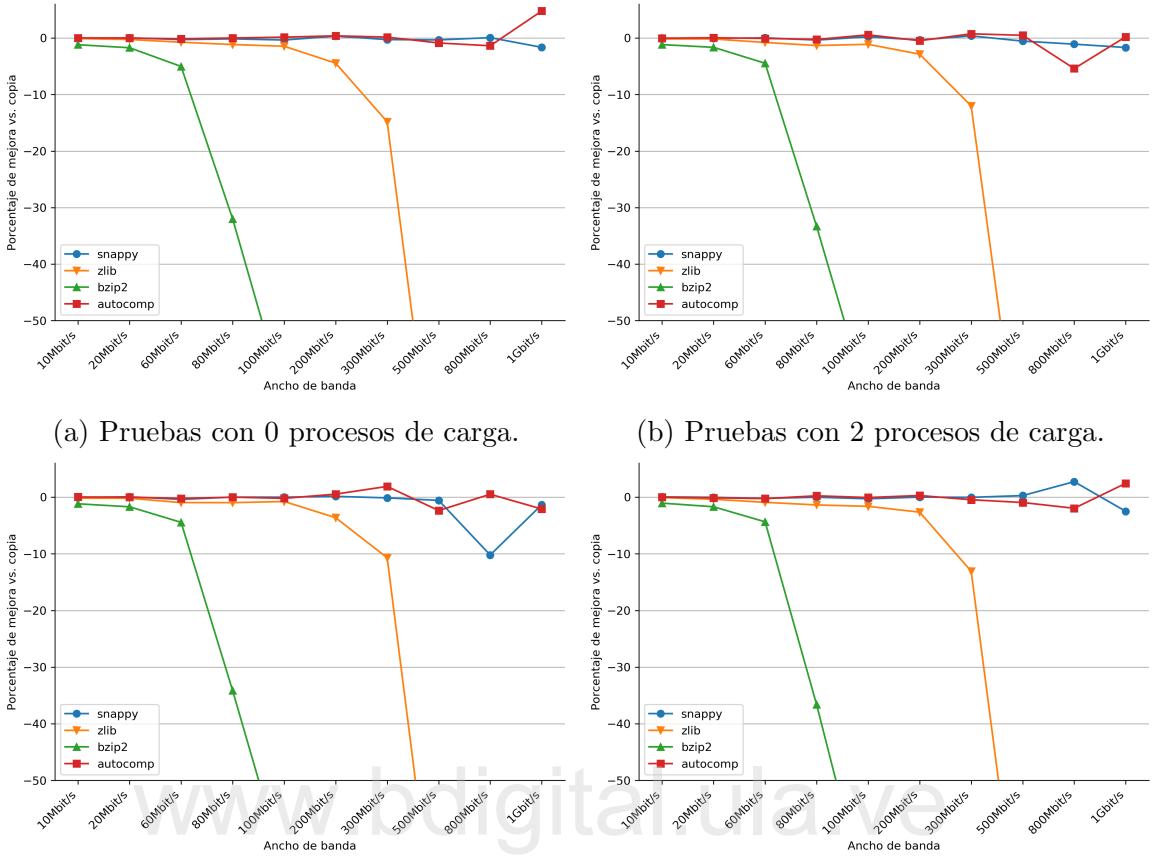
El peor escenario se trata de la trasmisión de datos aleatorios, los cuales son incompresibles y se espera que todos los métodos degraden el desempeño o, a lo sumo, lo igualen al de no comprimir. Los resultados para este tipo de datos se muestran en la Figura 5.3. `autocomp` logra mantener su rendimiento cercano al de la copia, al igual que snappy, mientras que zlib comienza a degradar el desempeño visiblemente a los 60 Mbits/s y bzip2 trabaja a pérdida en todos los casos, llegando incluso a obtener pérdidas de ∼1300% a 1 Gbit/s; lo anterior se debe al tiempo computacional intensiva e innecesariamente utilizado por bzip2 y zlib. No obstante, todos los métodos degradan el desempeño incluso a anchos de banda bajos: a 10 Mbits/s, el porcentaje de degradación para `autocomp`, snappy, zlib y bzip2 es de $\sim 0,002\%$, $\sim 0,006\%$, $\sim 0,1\%$y $\sim 1,2\%$, respectivamente, debido al trabajo de compresión innecesario realizado por los métodos estáticos y al costo de adaptación, como el *bytecounting*, en el caso del método adaptativo (pues este no comprime datos aleatorios).

En el caso de los datos aleatorios, `autocomp` siempre pagará el costo de adaptación, pero superando a los métodos estáticos incluso a altos anchos de banda, como a 1 Gbit/s, donde degrada el desempeño cerca de ∼2% (únicamente con 3 procesos de carga), mientras que snappy y zlib lo degradan, en promedio, $\sim 1,6\%$ y ∼227%, respectivamente.

Datos tipo *código*

Para los datos de tipo *código*, la Figura 5.4 muestra los resultados, donde se observa que a 10 Mbits/s, todos los métodos mejoran el desempeño entre 64% y 78%. `autocomp` lo hace ∼31% más lento que bzip2 a 20 Mbits/s, donde este último comienza a disminuir su porcentaje de mejora muy rápidamente, llegando incluso a degradarlo a partir de anchos de banda de 80 Mbits/s (alcanzando ∼950% de degradación a 1 Gbit/s). Del mismo modo, `autocomp` pierde frente a zlib hasta los 100 Mbits/s, donde este último comienza a disminuir su ganancia pero más lentamente que bzip2, llegando a degradar el desempeño a anchos de banda a partir de los 300 Mbits/s.

Se debe recordar que la compresión pierde su ganancia cuando el ancho de banda se



(a) Pruebas con 0 procesos de carga.

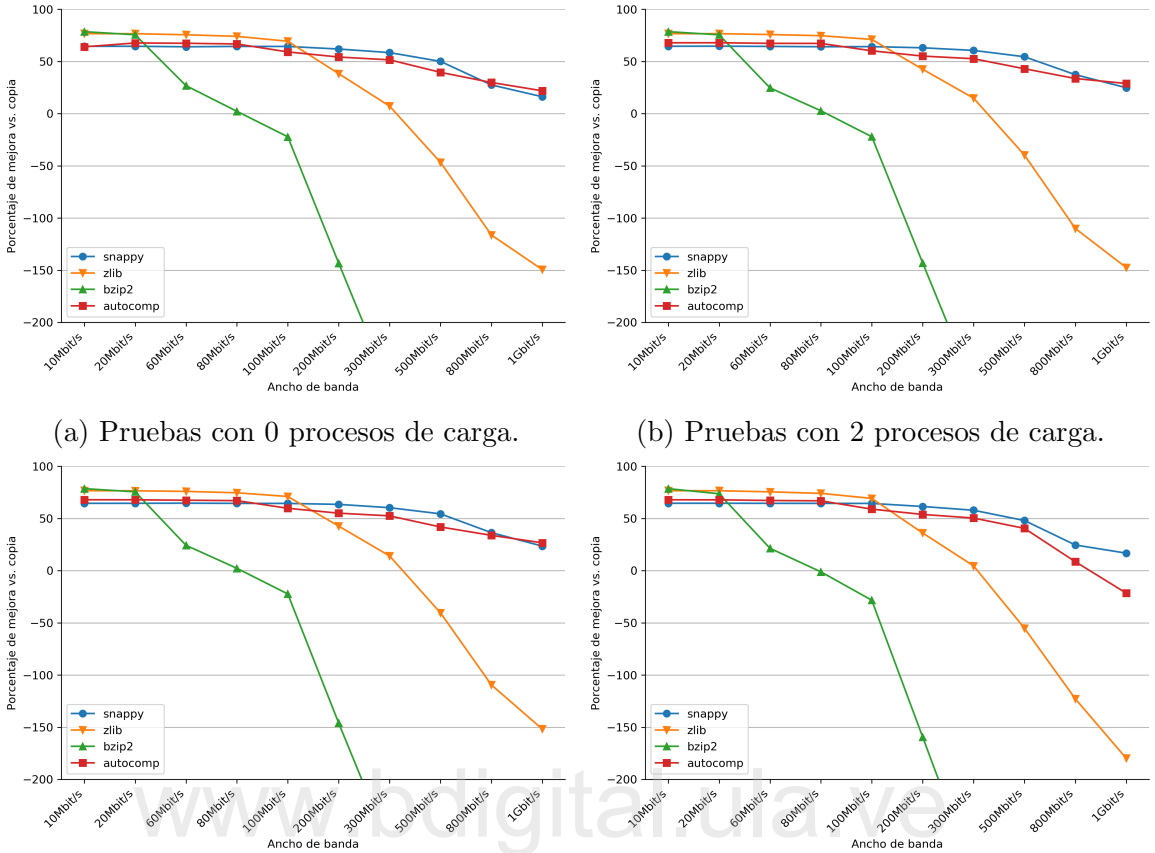
(b) Pruebas con 2 procesos de carga.

(c) Pruebas con 3 procesos de carga.

(d) Pruebas con 5 procesos de carga.

Figura 5.3: Porcentaje de mejora de cada método relativo a no comprimir para datos tipo *aleatorio*.

hace lo suficientemente alto (según la Ecuación 4.2), lo cual se ve reflejado para bzip2 y zlib a partir de anchos de banda de 80 Mbits/s y 300 Mbits/s, respectivamente, en este caso. `autocomp`, por otro lado, obtiene ganancia de la compresión en la mayoría de los casos, debido a las decisiones que puede estar tomando de comprimir con snappy o no comprimir en absoluto (la excepción es la degradación vista a 1 Gbit/s con 5 procesos de carga que, en el entorno de prueba, cargan el CPU al 100%, lo cual puede deberse al sobreúso del CPU en compresión, adaptación y toma de decisiones innecesarias a altas velocidades de transmisión). Con respecto a snappy, una vez más `autocomp` pierde terreno ante este compresor a altos anchos de banda. Sin embargo, la tendencia es que `autocomp` obtiene un aproximado de 7% de mejora más que snappy a 800 Mbits/s y 1 Gbit/s, posiblemente debido a que, a altos anchos de banda, se esté decidiendo



(a) Pruebas con 0 procesos de carga.

(b) Pruebas con 2 procesos de carga.

(c) Pruebas con 3 procesos de carga.

(d) Pruebas con 5 procesos de carga.

Figura 5.4: Porcentaje de mejora de cada método relativo a no comprimir para datos tipo *código*.

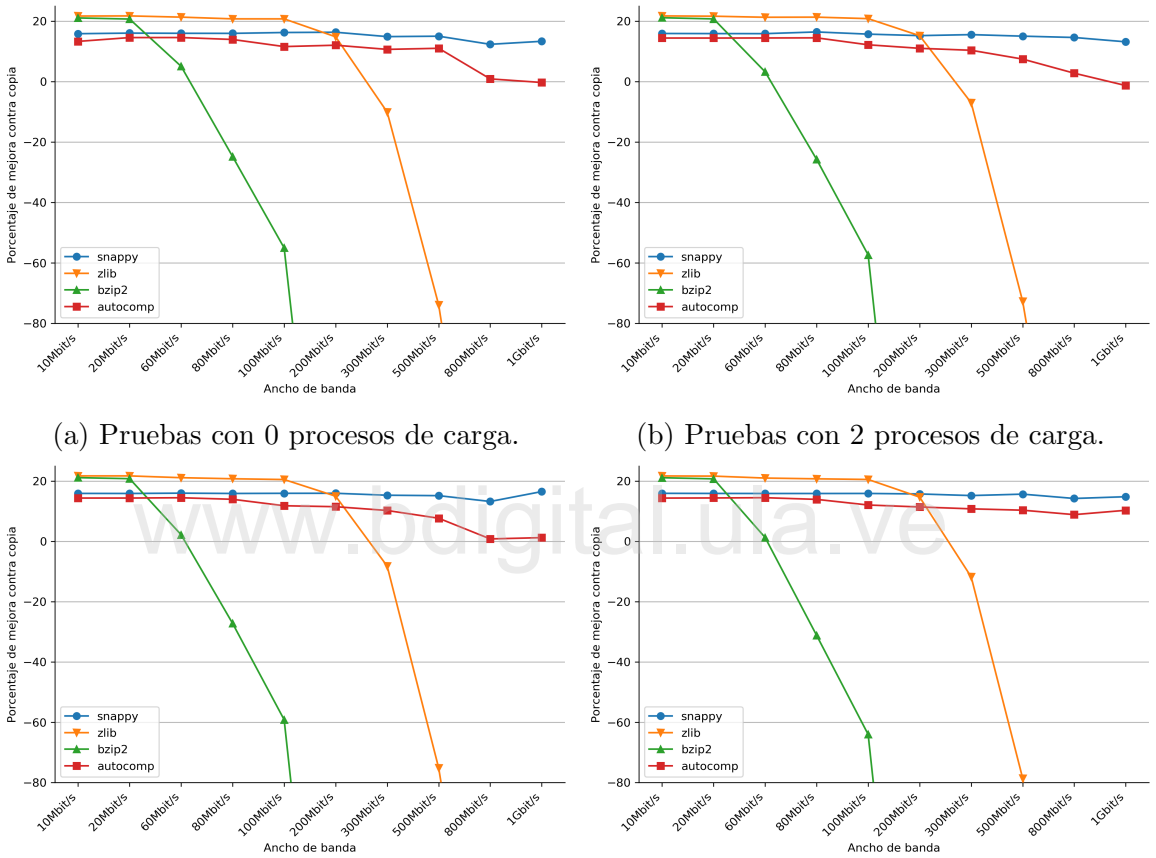
no comprimir algunos mensajes cuando comprimir no es producente (de nuevo, la excepción se da a 1 Gbits/s con 5 procesos de carga, donde se observa que incluso snappy disminuye su porcentaje de mejora a cerca de 10%).

En este tipo de datos se observa claramente el hecho de que la selección del mejor compresor (de forma estática) depende tanto del ancho de banda como del tipo de datos, pues ningún método es siempre el mejor y algunas decisiones pueden llegar a ser muy costosas.

Datos tipo *latex*

El comportamiento para los datos tipo *latex*, mostrado en la Figura 5.5, es similar al de los datos tipo *código*, pero, como es de esperarse por la naturaleza de los datos,

con porcentajes de mejora mucho menores (entre 13% y 22% para todos los métodos a 10Mbits/s). En este caso `autocomp` es superado por snappy — mostrando de nuevo el alto costo que se está pagando en la toma de decisiones — mas sin embargo `autocomp` logra superar a zlib y bzip2 cuando el ancho de banda es lo suficientemente alto para que estos compresores trabajen a pérdida.



(a) Pruebas con 0 procesos de carga.

(b) Pruebas con 2 procesos de carga.

(c) Pruebas con 3 procesos de carga.

(d) Pruebas con 5 procesos de carga.

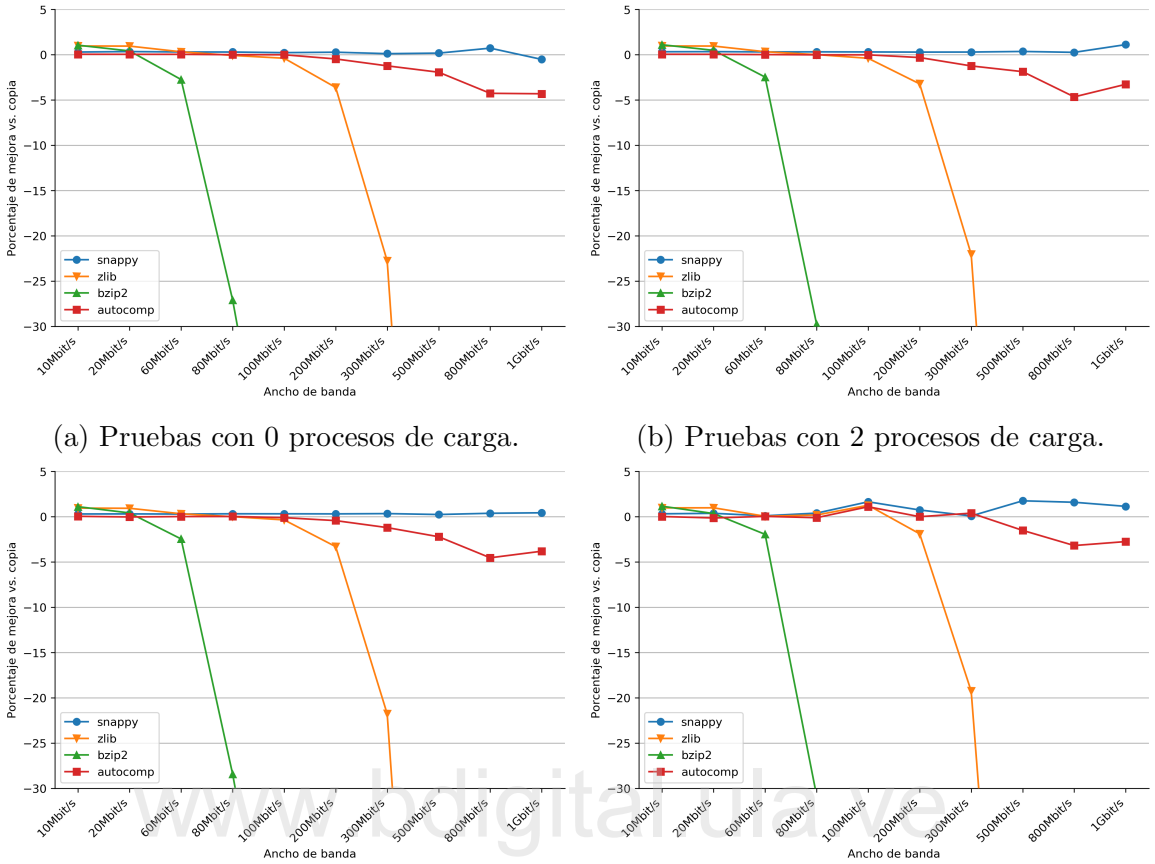
Figura 5.5: Porcentaje de mejora de cada método relativo a no comprimir para datos tipo *latex*.

El efecto de la pérdida de ganancia para los compresores computacionalmente más intensivos (bzip2 y zlib) se presenta de igual manera, con porcentajes de degradación mucho mayores — una vez más, como se espera para este tipo de datos. Sin embargo, prácticamente no hay degradación por parte de `autocomp` para este tipo de datos, siendo el peor caso la condición de "empate" (degradación de ∼2%) frente a la transmisión sin compresión a 1 Gbit/s con 2 procesos de carga, posiblemente debido a

que el mecanismo adaptativo propuesto ignora todos los trozos de este tipo de datos para los cuales la compresión no es conveniente.

Datos tipo *multimedia*

Este caso es muy similar al del tipo de datos *aleatorio* pues los datos son prácticamente incompresibles, como se muestra en la Figura 5.6. De nuevo se observa cómo la compresión pierde valor para bzip2 y zlib cuando el ancho de banda es lo suficientemente alto, incluso para `autocomp`, que degrada el desempeño un máximo de 5% a los niveles más altos de ancho de banda disponibles, principalmente debido a los costos de adaptación. Snappy y `autocomp`, sin embargo, manejan muy bien el caso de datos prácticamente incompresibles. En el caso de snappy, se demuestra que el compresor sacrifica en gran medida su capacidad de compresión por velocidad, para obtener ganancias relativamente decentes; para `autocomp`, por otro lado, se debe a que puede decidir no comprimir en absoluto la mayor parte de la información transmitida, pues en este caso comprimir sería mayoritariamente contraproducente.



(a) Pruebas con 0 procesos de carga.

(b) Pruebas con 2 procesos de carga.

(c) Pruebas con 3 procesos de carga.

(d) Pruebas con 5 procesos de carga.

Figura 5.6: Porcentaje de mejora de cada método relativo a no comprimir para datos tipo *multimedia*.

Capítulo 6

Conclusiones

Se propuso un mecanismo o estrategia de compresión automática y adaptativa, el cual fue implementado e integrado a un sistema de transmisión de datos desarrollado también como parte de este proyecto, para su evaluación y prueba de concepto. El mecanismo se basa en el supuesto presentado por Peterson y Reiher (2016) de que el desempeño de un algoritmo de compresión es constante en una oportunidad determinada. Basándose en lo anterior, el mecanismo cuantiza o discretiza el espacio de oportunidades e intenta seleccionar el algoritmo más apto en cada oportunidad, es decir, aquel que provea la mejor TET, mediante un árbol de decisión como herramienta de clasificación, debido a su simplicidad de funcionamiento e implementación y a que su desempeño es similar al de los demás clasificadores — más complejos — estudiados.

Los resultados muestran que, en el escenario de mayor ganancia esperada, el mecanismo adaptativo se comporta de manera similar a los mejores algoritmos a 10 Mbits/s, superándolos cuando el ancho de banda se hace lo suficientemente alto para superar la velocidad de estos compresores. No obstante, la compresión siempre se convertirá en un cuello de botella cuando el ancho de banda sea lo suficientemente alto (Peterson y Reiher, 2016). Del mismo modo, la suma de la adaptación, monitoreo y potencial compresión, se convertirá en un cuello de botella más rápidamente cuando el ancho de banda alcanza niveles lo suficientemente altos, por lo cual es imperativo que el método de clasificación sea lo suficientemente rápido y que los módulos de monitoreo y adaptación pasen por una fase intensiva de optimización para mitigar este efecto.

Esto ocurre, en este caso, cuando el ancho de banda supera los 100 Mbits/s, donde un algoritmo tan rápido como snappy logra superar por mucho al mecanismo propuesto — aunque este último esté utilizando intensivamente también a snappy — debido a que se paga el costo de adaptación y clasificación. Esto podría solucionarse agregando una especie de "*bypass*" para evitar la clasificación a altos anchos de banda y utilizar siempre snappy como mejor elección; sin embargo, esto se pagaría costosamente cuando los datos sean incompresibles. Otra solución sería que, a medida que el ancho de banda crece, se calcule el *bytecounting* para un trozo o *chunk* y se asuma este mismo para los trozos subsiguiente y, de esta manera, mitigar el costo de adaptación. En los peores escenarios, donde los datos son virtualmente incompresibles, el mecanismo propuesto logra mantener su desempeño cercano al de la transmisión sin compresión, pagando únicamente el costo de adaptación, con un porcentaje de degradación máximo cercano al 5% a 800 Mbits/s.

Los resultados además muestran que un único algoritmo de compresión nunca es óptimo, lo cual se observa mayormente en los algoritmos computacionalmente más costosos (zlib y bzip2), que reducen su porcentaje de ganancia de manera significativa cuando el ancho de banda crece (se hacen subóptimos), incluso degradando el desempeño cuando el canal de comunicación es lo suficientemente rápido.

En resumen, para un variado tipo de datos, aun cuando el mecanismo de compresión propuesto toma decisiones muy costosas — muy probablemente debido a la característica de los árboles de decisión que los hace muy sensibles antes pequeñas variaciones en los datos de entrada, como se menciona en la Sección 2.3.2 — este logra adaptarse a las condiciones del entorno y a las propiedades de los datos, evitando los costos de compresión y el consecuente deterioro del desempeño de la transmisión cuando los datos no son compresibles y obteniendo altos porcentajes de mejora a velocidades sobre los 100 Mbits/s donde, según Peterson y Reiher (2016), la compresión está, por lo general, deshabilitada en los sistemas de transmisión para evitar el riesgo de usar la compresión cuando no hay tiempo para ello.

6.1 Recomendaciones

El uso de un árbol de decisión como clasificador fue una elección realizada de una forma que podría decirse directa o inocente (*naive*), debido a su simplicidad de funcionamiento e implementación. No obstante, es muy probable este clasificador esté tomando decisiones erróneas ante cambios sutiles en los datos de entrada. Valdría la pena realizar un estudio completo y detallado de este y otros clasificadores que puedan ser integrados al sistema, mientras sean lo suficientemente rápidos, y realizar pruebas de adaptabilidad y rendimiento más exhaustivas con cada uno, así como también un perfilado temporal para determinar el porcentaje de tiempo, en cada oportunidad, efectivamente utilizado en la toma de decisiones.

Para lidiar con el poco tiempo disponible para compresión a anchos de banda sobre los 100 Mbits/s — si es que hubiere — es posible que se deba deshabilitar todo proceso de adaptación (*bytecounting*) y toma de decisiones (clasificación) y evitar la compresión. Sin embargo, es notorio que snappy es un compresor lo suficientemente rápido como para obtener ganancia incluso a 1 Gbit/s, por lo cual también se debería estudiar alguna estrategia para reducir la cantidad de veces que se calcula el *bytecounting* y utilizarlo únicamente para detectar flujos de datos con poco o ningún potencial de compresión.

Adicionalmente, se puede utilizar una técnica estadística de muestreo para seleccionar, para cada trozo en una oportunidad dada, sub-trozos para los cuales se calcule el *bytecounting* y se utilice alguna métrica o estadístico para obtener el *bytecounting* general del trozo en cuestión. Esto responde a la necesidad de reducir el costo de adaptación, principalmente a altos anchos de banda, que se genera por el cálculo del *bytecounting*.

Bibliografía

Apache Software Fundation (2018). Apache http server project. Versión 2.4.29. https://httpd.apache.org/.

Bell, T., Witten, I. H., y Cleary, J. G. (1989). Modeling for text compression. *ACM Comput. Surv.*, 21(4):557–591.

Burrows, M. y Wheeler, D. J. (1994). A block-sorting lossless data compression algorithm. Reporte técnico.

Collette, A. (2018). LZFX Data Compression Library. https://code.google.com/archive/p/lzfx/.

Forouzan, A. B. (2006). *Data communications & networking (sie)*. Tata McGraw-Hill Education.

Gailly, J.-l. y Adler, M. (2018a). GNU Gzip. Versión 1.9. https://www.gnu.org/software/gzip/.

Gailly, J.-l. y Adler, M. (2018b). zlib, sitio Web. https://www.gnu.org/software/gzip/. Fecha de consulta 26/09/2018.

Google (2018a). Google Test. https://github.com/google/googletest. Fecha de consulta 30/09/2018.

Google (2018b). Snappy. Versión 1.1.7. http://google.github.io/snappy/.

Harnik, D., Kat, R. I., Margalit, O., Sotnikov, D., y Traeger, A. (2013). To zip or not to zip: effective resource usage for real-time compression. En *FAST*, p. 229–242.

Hubert, B., Geul, J., y Séhier, S. (2018). Wondershaper. https://github.com/magnific0/wondershaper. Fecha de consulta 29/09/2018.

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

iPerf (2018). Sitio Web. https://iperf.fr/. Fecha de consulta 27/09/2018.

Jägemar, M., Eldh, S., Ermedahl, A., y Lisper, B. (2016). Automatic message compression with overload protection. *Journal of Systems and Software*, 121(C):209–222.

Kerrisk, M. (2018). Projecto man-pages de linux. http://man7.org/. Fecha de consulta 01/10/2018.

Kingsford, C. y Salzberg, S. L. (2008). What are decision trees? *Nature biotechnology*, 26(9):1011.

Krintz, C. y Sucu, S. (2006). Adaptive on-the-fly compression. *IEEE Transactions on Parallel and Distributed Systems*, 17(1):15–24.

Larman, C. y Basili, V. R. (2003). Iterative and incremental developments. a brief history. *Computer*, 36(6):47–56.

Mahmud, S. (2012). An improved data compression method for general data. *International Journal of Scientific & Engineering Research*, 3(3):2.

Oberhumer, M. (2018). LZO (Lempel-Ziv-Oberhumer) Data Compression Library. Versión 2.10. http://www.oberhumer.com/opensource/lzo/.

Pavlov, I. (2018). LZMA Software Development Kit. Versión 18.05. https://www.7-zip.org/sdk.html.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., y Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Peterson, P. A. y Reiher, P. L. (2016). Datacomp: Locally independent adaptive compression for real-world systems. En *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, p. 211–220. IEEE.

Pressman, R. S. (2015). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, USA, 8 edición.

Pu, I. (2004). *Data Compression*. University of London. Consultado en https://london.ac.uk/courses/data-compression-co3325. Fecha de consulta: 12 de Junio de 2018.

QuickLZ (2018). QuickLZ. Versión 1.5.0. http://www.quicklz.com/.

Seward, J. (2018). BZIP2, a program and library for data compression. Versión 1.0.6. http://www.bzip.org/.

Shannon, C. E. (1949). Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21.

Universidad de Canterbury, N. Z. (2018). Corpus de Canterbury, sitio Web. http://corpus.canterbury.ac.nz/. Fecha de consulta 27/09/2018.

Universidad Politécnica de Silesia, P. (2018). Sitio Web. http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia. Fecha de consulta 29/09/2018.

Welch, T. A. (1984). A technique for high-performance data compression. *Computer*, 17(6):8–19.

Wiseman, Y., Schwan, K., y Widener, P. (2004). Efficient end to end data exchange using configurable compression. En *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, p. 228–235.

Ziv, J. y Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343.

Zohar, E. y Cassuto, Y. (2014). Automatic and dynamic configuration of data compression for web servers. En *28th Large Installation System Administration Conference (LISA14)*, p. 106–117, Seattle, WA. USENIX Association. http://www.eyalzo.com/projects/ecomp. Fecha de consulta 01/10/2018.

Zohar, E. y Cassuto, Y. (2015). Data compression cost optimization. En *2015 Data Compression Conference*, p. 393–402.