การบีบอัดเอกสารเอชทีเอ็มแอลบนฝั่งเซิร์ฟเวอร์ด้วยการเข้ารหัสแบบฮัฟแมน

โดย

ว่าที่ร้อยตรีบรรพต ดลวิทยากุล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์

ภาควิชาคอมพิวเตอร์

บัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร

ปีการศึกษา 2550

การบีบอัดเอกสารเอชทีเอ็มแอลบนฝั่งเซิร์ฟเวอร์ด้วยการเข้ารหัสแบบฮัฟแมน

โดย
ว่าที่ร้อยตรีบรรพต ดลวิทยากุล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์
ภาควิชาคอมพิวเตอร์
บัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร
ปีการศึกษา 2550

SERVER-SIDED HTML DATA COMPRESSION USING HUFFMAN CODING

By

Banpot Dolwithayakul

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

Department of Computing

Graduate School

SILPAKORN UNIVERSITY

2007

บัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร อนุมัติให้วิทยานิพนธ์เรื่อง “การบีบอัดเอกสารเอชทีเอ็มแอลบนฝั่งเซิร์ฟเวอร์ด้วยการเข้ารหัสแบบฮัฟแมน” เสนอโดยว่าที่ร้อยตรี บรรพต คลวิทยากุล เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์

……..

(รองศาสตราจารย์ ดร.ศิริชัย ชินะตังกูร)

คณบดีบัณฑิตวิทยาลัย

วันที่.....เดือน..... พ.ศ.....

อาจารย์ที่ปรึกษาวิทยานิพนธ์

อาจารย์ ดร.สุนีย์ พงษ์พินิจภิญโญ

คณะกรรมการตรวจสอบวิทยานิพนธ์

..... ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.ปานใจ ธารทัศนวงศ์)

...../...../.....

..... กรรมการ

(อาจารย์ ดร.วสันต์ ภัทรอธิคม)

...../...../.....

..... กรรมการ

(อาจารย์ ดร.สุนีย์ พงษ์พินิจภิญโญ)

...../...../.....

48307305 : สาขาวิชาวิทยาการคอมพิวเตอร์
คำสำคัญ : การบีบอัดข้อมูล/การเข้ารหัสฮัฟแมน/เครือข่ายอินเตอร์เน็ต

บรรพต ดลวิทยากุล : การบีบอัดเอกสารเอชทีเอ็มแอลบนฝั่งเซิร์ฟเวอร์ด้วยการเข้ารหัสแบบฮัฟแมน. อาจารย์ที่ปรึกษาวิทยานิพนธ์ : อ.ดร.สุนีย์ พงษ์พินิจภิญโญ. 64 หน้า.

การบีบอัดข้อมูลเอกสาร HTML ที่เว็บเซิร์ฟเวอร์และถูกส่งมายังเว็บบราวเซอร์นั้นทำเป็นแบบ on-the-fly เนื่องจากสามารถรองรับการทำงานกับเอกสารที่เป็น Dynamic เช่นภาษาสคริปต์และ SSI โดยใช้ GZIP อัลกอริธึมในการบีบอัดข้อมูลซึ่งเป็นการบีบอัดเนื้อหาเอกสาร HTML ทุกครั้งที่มีการร้องขอก่อนส่งข้อมูลมายังบราวเซอร์ วิธีนี้ไม่ได้รับความนิยมเพราะว่าไม่เพียงทำให้เว็บเซิร์ฟเวอร์ทำงานหนักเนื่องจากต้องใช้เวลาส่วนหนึ่งในการบีบอัดข้อมูล แต่จะทำให้ประสิทธิภาพการให้บริการลดลงอย่างมากด้วย

ดังนั้นงานวิจัยนี้จึงได้นำเสนอวิธีการแก้ปัญหาดังกล่าวอย่างมีประสิทธิภาพด้วยวิธีการ Huffman Coding ซึ่งเป็นรูปแบบหนึ่งของ Entropy Encoding เพื่อใช้สำหรับการบีบอัดเอกสาร HTML ที่มีประสิทธิภาพมากกว่า ผลการทดลองเปรียบเทียบแสดงให้เห็นว่าวิธีการ Huffman Coding ใช้อัตราการบีบอัดเอกสาร และใช้เวลาในการบีบอัดที่น้อยกว่าใช้วิธี GZIP อย่างมาก เมื่อใช้ขนาดของไฟล์เอกสาร HTML ที่แตกต่างกัน

ภาควิชาคอมพิวเตอร์ บัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร ปีการศึกษา 2550
ลายมือชื่อนักศึกษา.....
ลายมือชื่ออาจารย์ที่ปรึกษาวิทยานิพนธ์ .....

48307305 : MAJOR : COMPUTER SCIENCE
KEY WORD : HUFFMAN CODING/DATA COMPRESSION/INTERNET
BANPOT DOLWITHAYAKUL : SERVER-SIDED HTML DATA COMPRESSION USING HUFFMAN CODING. THESIS ADVISOR : SUNEE PONGPINITPINYO,Ph.D. 64 pp.

HTML documents are compressed at web servers and sent to web browser by on-the-fly data compression with GZIP algorithms. The HTML document contents are compressed every time whenever there are requested before they are sent to the web browsers. This method is not popular because it will not only make servers over workload due to spending part of time compressing data, but also will give much of decrease efficiency of service. Therefore, in this paper, we propose a method to solve such problems efficiently using Huffman Coding which is a form of Entropy Encoding to be more efficiently used for HTML documents compression.

The experimental comparison results show that the document compression rate and document compression time of Huffman Coding is most less than GZIP when using different HTML document file sizes.

Department of Computing Graduate School, Silpakorn University Academic Year 2007
Student's signature .....
Thesis Advisor's signature .....

กิตติกรรมประกาศ

วิทยานิพนธ์นี้ประสบความสำเร็จเป็นอย่างดีด้วยคำแนะนำของอาจารย์ที่ปรึกษาคือ อ. ดร. สุนีย์ พงษ์พินิจภิญโญ และคำแนะนำของกรรมการวิทานิพนธ์ทั้ง 2 ท่านคือ ผศ. ดร. ปานใจ ธารทัศนวงศ์ และ ดร. วสันต์ ทางผู้วิจัยต้องขอขอบพระคุณกรรมการทั้ง 3 ท่านเป็นอย่างสูง ที่แนะนำตลอดจนมหาวิทยาลัยศรีปทุมที่ได้คัดเลือกวิทยานิพนธ์นี้ไปเข้าร่วมการประชุมในงาน NCSEC 2007

ทางผู้วิจัยขอขอบพระคุณบิดา มารดา เพื่อน และพี่น้อง ที่สนับสนุนและร่วมให้กำลังใจกับผู้วิจัยมาตลอด จนกระทั่งวิทยานิพนธ์นี้เสร็จสมบูรณ์

สารบัญ

สารบัญภาพ

สารบัญแผนภูมิ

สารบัญตาราง

บทที่ 1
บทนำ

1.ที่มาและความสำคัญของปัญหา

ในปัจจุบันระบบเครือข่ายอินเทอร์เน็ต (Internet) ได้พัฒนาไปมาก จนได้มีบริการต่างๆ เกิดขึ้นมากมาย ซึ่งบริการหนึ่งบนเครือข่ายอินเทอร์เน็ตที่มีคนใช้กันมากที่สุดคือบริการเวิร์ลไวด์เว็บ (World Wide Web - WWW) ซึ่งเป็นบริการดูเอกสารต่างๆ และเนื้อหาอื่นๆ เช่นรูปภาพ ซึ่งเอกสารต่างๆ ที่สามารถเปิดดูได้นั้นส่วนมากสร้างมาจากภาษา Hypertext Markup Language (HTML)

เมื่อมีผู้ใช้งานบริการเวิร์ลไวด์เว็บ มากขึ้นเรื่อยๆทำให้เกิดปัญหาในหลายๆ ด้านเช่น อัตราการส่งข้อมูลที่มีอยู่ไม่สามารถรองรับการส่งข้อมูลที่มีการร้องขอพร้อมกันจำนวนมากได้เนื่องจากความเร็วในการส่งข้อมูลบนเครือข่ายไม่เพียงพอ จึงได้มีแนวคิดในการแก้ปัญหาเหล่านี้หลายวิธี แต่วิธีหนึ่งที่นำมาใช้คือการบีบอัดเอกสารให้มีขนาดเล็กลงบนฝั่งเว็บเซิร์ฟเวอร์ก่อนส่งไปยังเบราเซอร์

ในปัจจุบันเว็บเซิร์ฟเวอร์บางตัวเช่น Apache และ IIS นั้นใช้อัลกอริทึม GZIP ในการบีบอัดข้อมูลเอกสาร HTML (Compression) ซึ่งรู้จักกันในชื่อ mod_gzip (Internet Engineering Task Forge 2007)แต่วิธีนี้ทำให้ใช้งานหน่วยประมวลผลทั้งการบีบอัดและคลายการบีบอัดมาก ในเว็บเซิร์ฟเวอร์ใหญ่ๆ ที่มีผู้ใช้ปริมาณมากจึงไม่มีการใช้งานอัลกอริทึมนี้

อัลกอริทึม GZIP นั้นมีการทำงาน 2 ขั้นตอนคือในครั้งแรกตัดข้อมูลออกเป็นชุดย่อยๆ จะมีการค้นหาชุดบิตที่ซ้ำๆกันด้วยอัลกอริทึม LZ77 แล้วจึงมาจัดทำเป็น Dictionary ซึ่งส่วนนี้ใช้หน่วยประมวลผลและหน่วยความจำมาก แต่ในการบีบอัดแบบ Huffman Coding นั้นเป็นการบีบอัดในแบบ Entropy Encoding ซึ่งเป็นวิธีที่ทำให้ไบต์หรือชุดข้อมูลที่มีการใช้งานบ่อยๆจะถูกแทนด้วยบิตจำนวนน้อยๆ โดยวิธีการทำงานนั้นใช้ Binary Tree เป็นหลักและสามารถทำได้ภายในรอบเดียวในการอ่านข้อมูล ทำให้ทำงานได้อย่างรวดเร็ว ดังนั้นการบีบอัดข้อมูลด้วยวิธี Huffman Coding นั้นจึงเหมาะกับไฟล์ ASCII เช่นไฟล์ HTML มากกว่า

วัตถุประสงค์การวิจัย

1. ศึกษาและพัฒนาการบีบอัดข้อมูลเอกสาร HTML และ XML บนฝั่งเว็บเซิร์ฟเวอร์ที่สามารถเข้ากันได้กับ Firefox ซึ่งเป็นเบราเซอร์พัฒนาในรูปแบบ Open Source ที่ใช้กันอยู่ในปัจจุบัน โดยใช้ Huffman Coding เป็นอัลกอริทึมหลักในการศึกษาและทำงานวิจัยนี้

2. นำวิธีการบีบอัดข้อมูลในแบบ GZIP และ Huffman Coding นี้ไปประยุกต์ใช้จริงบนเว็บเซิร์ฟเวอร์ Apache ในรูปแบบ CGI และติดตามผลการทำงาน

3. ประเมินผลของระบบที่พัฒนาขึ้นโดยเปรียบเทียบกับ GZIP

ประโยชน์ที่คาดว่าจะได้รับ

1. ได้ตัวต้นแบบในการบีบอัดเอกสาร HTML บนฝั่งเซิร์ฟเวอร์ที่ทำงานได้รวดเร็วและเข้ากันได้กับเบราเซอร์ Firefox

2. ได้ซอฟต์แวร์ที่ใช้สำหรับบีบอัดเอกสาร HTML บนฝั่งเซิร์ฟเวอร์ และไลบรารี่ในการคลายการบีบอัดเอกสาร HTML ที่มีประสิทธิภาพ

3. ได้ศึกษาซอฟท์แวร์โอเพ็นซอร์สและนำมาประยุกต์ใช้ให้เกิดประโยชน์สูงสุดและเป็นอีกทางเลือกหนึ่งในการเลือกใช้ในอนาคต

ขอบเขตงานวิจัย

1. พัฒนาโปรแกรมที่ทำหน้าที่เป็น CGI บนฝั่งเซิร์ฟเวอร์เพื่อทำงานร่วมกับ Apache โดยสามารถบีบอัดข้อมูลเอกสาร HTML และจับเวลาในการบีบอัดด้วยอัลกอริทึม Huffman Coding, DEFLATE และ GZIP ได้ มีการทำงานในลักษณะ On-the-Fly เมื่อมีการร้องขอข้อมูลจากเว็บเซิร์ฟเวอร์ โดยผู้วิจัยจะวัดเวลาที่ใช้ในการบีบอัดข้อมูลเป็นตัวเปรียบเทียบหลัก (หน่วยเป็น millisecond)

2. เปรียบเทียบ Compression Ratio และเวลาที่ใช้ในการบีบอัดข้อมูลระหว่าง Huffman Coding กับการบีบอัดวิธีอื่นได้แก่ GZIP และ Shannon-Fano

3. พัฒนาเบราเซอร์จำลองเพื่อใช้ดึงข้อมูลจากเว็บเซิร์ฟเวอร์ทดสอบการคลายการบีบอัด ที่สามารถจับเวลาและวัดประสิทธิภาพในการบีบอัดได้

4. พัฒนา Plug In ขนาดเล็กฝังบนเบราเซอร์ Mozilla Firefox โดยใช้ Gecko API ซึ่งเป็น API หลักในการพัฒนาโมดูลย์ของ Firefox เพื่อทดลองการนำไปใช้งานจริง

ตัวอย่างที่ใช้ในการทดลอง

ข้อมูลจากเว็บไซต์ต่างๆ โดยเลือกจากสถิติเว็บไซต์ที่ใช้งานบ่อย 100 อันดับแรกที่บริษัท Netcraft และ TrueHits ได้รวบรวมไว้

เครื่องมือและอุปกรณ์

ฮาร์ดแวร์(ฝั่งเซิร์ฟเวอร์)

- CPU AMD Turion 64 ความเร็ว1.6GHz
- RAM 512 MB
- Hard disk 40 GB
- LAN Card
- CD-ROM

ฮาร์ดแวร์(ฝั่งเบราเซอร์)

- CPU Pentium III ความเร็ว 800Mhz ขึ้นไป
- RAM 256 MB ขึ้นไป
- LAN Card

ซอฟต์แวร์ (ฝั่งเซิร์ฟเวอร์)

- Redhat Fedora Core 6 x86-64 (Linux)
- Apache 2.2 (Web Server)
- Perl (Perl Compiler)
- GCC 4.1.6 (C++ Compiler)

ซอฟต์แวร์ (ฝั่งเบราเซอร์)

- Windows XP หรือ Redhat Linux
- Mozilla Firefox (Browser)

นิยามศัพท์เฉพาะ

1. HTTP ย่อมาจาก Hypertext Transfer Protocol เป็นโปรโตคอลการส่งข้อมูลที่ทำงานอยู่บนโปรโตคอล TCP/IP หน้าที่หลักคือเป็นโปรโคคอลหลักที่ใช้ในการส่งข้อมูลบนของบริการเวิร์ลไวด์เว็บ ข้อมูลที่ส่งนั้นเช่น เอกสาร HTML และ รูปภาพ เป็นต้น

2. HTTP Header คือข้อมูลชุดหนึ่งที่ใช้ส่งบนโปรโตคอล HTTP เพื่อใช้ระบุคุณสมบัติต่างๆของการร้องขอ หรือเนื้อหาข้อมูลที่จะส่ง เช่น MIME Type วันเวลาที่มีการแก้ไข ชื่อไฟล์ การเข้ารหัส รวมทั้งลักษณะการบีบอัดข้อมูล(ถ้ามี) เป็นต้น

3. CGI เป็นวิธีหนึ่งในการติดต่อสื่อสารระหว่างเว็บเซิร์ฟเวอร์และแอปพลิเคชันที่อยู่เบื้องหลัง โดยก่อนที่เว็บเซิร์ฟเวอร์จะส่งเนื้อหาต่างๆไปให้กับเบราเซอร์ ก็จะผ่านตัวโปรแกรมที่ทำหน้าที่เป็น CGI Gateway ก่อนเพื่อทำการประมวลผลแล้วจึงส่งให้กับเบราเซอร์ต่อไป

4. HTML เป็นภาษาหนึ่งซึ่งใช้ในการเผยแพร่เอกสารทางบริการเวิร์ลไวด์เว็บ ซึ่งประกอบด้วยแท็ก เป็นตัวควบคุมการแสดงผลในรูปแบบต่างๆที่คนสร้างเอกสารต้องการ เช่น ตัวหนา สี และส่วนเนื้อหาอื่นๆที่อยู่นอกแท็ก

5. เว็บเซิร์ฟเวอร์ (Web Server) คือเครื่องแม่ข่ายที่ให้บริการเวิร์ลไวด์เว็บ

6. เบราเซอร์ (Browser) คือซอฟต์แวร์ที่ใช้ร้องขอข้อมูลจากเว็บเซิร์ฟเวอร์ และอ่านเอกสาร HTML

7. Turn-Around Time คือเวลาที่ใช้เริ่มตั้งแต่ส่งคำร้องขอไปยังเว็บเซิร์ฟเวอร์ จนกระทั่งเว็บเซิร์ฟเวอร์ส่งข้อมูลกลับมาจนครบถ้วน

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

แนวคิด ทฤษฎี และงานวิจัยที่เกี่ยวข้องกับงานวิจัยนี้ ผู้วิจัยได้แบ่งออกเป็นหัวข้อต่างๆดังนี้

2.1 Huffman Coding

Huffman Coding (Huffman 1952 : 1098-1102) เป็นอัลกอริทึมหนึ่งที่ใช้สำหรับบีบอัดข้อมูลแบบ Entropy Encoding วิธีหนึ่งโดยพัฒนามาจาก Shannon-Fano ซึ่งผลที่ได้จากการบีบอัดด้วยอัลกอริทึมนี้เป็นการบีบอัดข้อมูลแบบ Lossless Compression วิธีการหลักคือใช้ตารางที่บรรจุ Codeword ที่สร้างขึ้นมาแทนที่ Symbol ภายในเอกสาร ซึ่งบางครั้งเราจะเรียกตารางนี้ว่า Dictionary ในการเข้ารหัสเอกสารด้วยการแทนที่ และถอดรหัสเอกสารด้วยการแทนที่กลับด้วยตารางอ้างอิงเดียวกัน

Huffman Coding นั้นสามารถบีบอัดข้อมูลไฟล์ที่เป็น ASCII ได้รวดเร็วมาก และประสิทธิภาพในการบีบอัดนั้นได้ผลดีเนื่องจากเราสามารถทราบ Symbol ทั้งหมดที่เป็นไปได้ (A-Z , a-z , 0-9) ทำให้ไม่ต้องมีการค้นหา Symbol เหมือนกับอัลกอริทึมอื่นๆ ความเร็วในการทำงานนั้นใช้ O(n) = n เท่านั้น

หลักการโดยทั่วไปของ Huffman Coding จะอธิบายแยกออกเป็น 3 ตอนคือ อินพุต เอาท์พุต และเป้าหมายของอัลกอริทึมนี้

2.1.1 อินพุต

ให้ A เป็นเซ็ทของตัวอักขระ ที่ไม่เป็นเซ็ทว่าง ซึ่งมีจำนวนข้อมูลขนาด n ตัว เราสามารถเขียนในรูปแบบสมการได้ดังนี้

$$A = \{a_1, a_2, \ldots, a_n\}$$

และให้ W เป็นเซ็ทของน้ำหนักของตัวอักขระในเซ็ท A แต่ละตัว ตั้งแต่ a_1 จนถึง a_n เราจะเขียนได้ดังนี้

$$W = \{w_1, w_2, \ldots, w_n\}$$

โดยที่ค่าสมาชิกในเซ็ท W (W_i) นั้น ต้องมีค่าเป็นจำนวนจริงบวก และสมาชิกของทั้ง 2 เซ็ทจะต้องมีค่าเป็น n เท่ากัน

$$W_i = \text{weight}(a_i),\ 1 \leq i \leq n$$

2.1.2 เอาท์พุต

เอาท์พุตที่จะได้จากอัลกอริทึมนี้คือ ต้องหาเซ็ท C(A,W) ซึ่งประกอบด้วย c_1 จนถึง c_n ซึ่งเราจะเรียกสมาชิกแต่ละตัวว่า Codeword ซึ่งเขียนได้ดังนี้

$$C(A,W) = \{c_1, c_2, c_3,..,c_n\}$$

โดยที่ $a_i,\ 1 \leq I \leq n$

2.1.3. จุดมุ่งหมายของอัลกอริทึม

เพื่อหาค่า L(C) ที่น้อยที่สุดเท่าที่จะเป็นไปได้ โดย L(C) นั้นคำนวณได้จากผลรวมของน้ำหนักสมาชิกทุกตัวในเซ็ท W คูณด้วยความยาวของ Codeword ในแต่ละตัว ซึ่งจะเขียนได้ตามสมการด้านล่างนี้

$$L(C) = \sum_{i=1}^{n} W_i \text{ x length } (C_i)$$

เช่น ถ้าเรามีข้อมูลชุดหนึ่ง ประกอบด้วยอักขระ 5 ตัว คือ A,B,C,D และ E ซึ่งมีความน่าจะเป็นในการพบอักขระดังกล่าวภายในเอกสาร 10%, 15%, 30%, 16% และ 29% ตามลำดับ เมื่อเราใช้ Huffman Coding ในการบีบอัดเอกสาร เราจะได้ C1 = 000, C2=001, C3=10, C4=01 และ C5=11 ซึ่งเราสามารถเขียนเป็นตารางที่ 1 ได้ดังต่อไปนี้

ตารางที่ 1 ตัวอย่างการทำงานและประสิทธิภาพของ Huffman Coding

อินพุต	อักขระ	A	B	C	D	E	
	ค่าน้ำหนัก	0.10	0.15	0.30	0.16	0.29	= 1.00
เอาท์พุต	Codewords	000	001	10	01	11	
	ความยาวของ Codeword (บิต)	3	3	2	2	2	
	ค่า L(C) = ความยาวCodeword x น้ำหนัก	0.10 x 3	0.15 x 3	0.30 x 2	0.16 x 2	0.29 x 2	= 2.25

จากตัวอย่างจะเห็นว่า เดิมหากมีข้อมูลจำนวน 100 ตัวอักษร ก็จะใช้เนื้อที่ในการเก็บหรือส่งข้อมูล 100 ไบต์ หรือ 800 บิต แต่เมื่อใช้อัลกอริทึม Huffman Coding ในการบีบอัดข้อมูล จะใช้เนื้อที่ในการเก็บดังนี้

$$L(C) = (10 \times 3) + (15 \times 3) + (30 \times 2) + (16 \times 2) + (29x2)$$

$$L(C) = 255$$

ดังนั้นข้อมูลหลังถูกบีบอัดเหลือเพียง 255 บิตจากเดิม 800 บิต คิดเป็น 255/800 x 100 = 31.88% ซึ่งลดขนาดข้อมูลที่จะเก็บหรือส่งลงไปได้อย่างมาก

Huffman Coding นั้นเป็นเพียงแนวคิดที่จะลดค่า L(C) ให้ได้มากที่สุด ซึ่งไม่มีวิธีการทำที่ตายตัว แต่ส่วนมากแล้วนั้นเราจะใช้วิธีการสร้าง Binary Tree เพื่อหาเซ็ท C(A,W) ที่เหมาะสมที่สุด เพราะว่า Binary Tree นั้นทำงานและค้นหาได้อย่างรวดเร็ว

2.2 วิธีการ Linear-Time

วิธีการ Linear-Time (Corman 2001 : 385-392) เป็นเทคนิคในการสร้าง Binary Tree เพื่อใช้งานกับ Huffman Coding ไว้ โดยวิธีที่เป็นที่นิยมมากนั้น เรียกว่าวิธีการ Linear-Time ซึ่งทำงานได้รวดเร็ว มี O(n) เพียง n ในกรณี Best Case คือข้อมูลเรียงมาก่อนแล้ว และ O(n) จะเป็น n log n ในกรณีปกติและ Worst Case การทำงานนั้นได้จาก Queue 2 ตัว โดยมีขั้นตอนดังนี้

1. สร้างโหนดภายใน Tree จำนวนเท่ากับจำนวน Symbol ที่ต้องการ
2. สร้าง Queue ขึ้นมา 1 เซ็ท เพื่อบรรจุโหนดต่างๆไว้
3. ในขณะที่ Queue นั้นมีโหนดมากกว่า 1 ตัว ให้กระทำการดังนี้
   - นำโหนด 2 โหนดที่มีค่าน้ำหนักน้อยที่สุดออกไป
   - สร้างโหนดใหม่ภายใน tree โดยใช้โหนดที่นำออกไปเป็น child ของโหนดนั้น และค่าของโหนดใหม่คือค่าของผลรวมน้ำหนักของทั้ง 2 โหนดที่นำออกไปนั้น
   - เพิ่มโหนดที่สร้างขึ้นมาใหม่นั้นกลับลงไปใน Queue
4. หลังจากนั้นจะเหลือเพียง 1 โหนด ซึ่งโหนดนั้นคือรากของ Tree

2.3 อัลกอริทึม DEFLATE

อัลกอริทึม DEFLATE เป็นอัลกอริทึมในการบีบอัดข้อมูลในรูปแบบ Lossless Data Compression โดยการพัฒนามาจากอัลกอริทึม LZ77(Lempel and Ziv. 1977 : 337-343) และ Huffman Coding รวมกัน โดย Phil Katz ซึ่งอัลกอริทึมนี้จะมีระบุไว้ใน RFC 1951 (Internet Engineering Task Forge, Online) ด้วย

DEFLATE จะตัดสตริงออกเป็นบล็อก บล็อกละ 32kB โดยแต่ละบล็อกนั้นจะใช้พื้นที่ 4 ไบต์ในการเก็บระยะทาง (Distance Code) โดยเป็นระยะทางที่จะถูกแทนที่ และสามารถเก็บ Symbol ที่จะถูกแทนที่ได้ 288 Symbols

หลักการทำงานคือ จะตัดข้อมูลออกเป็นบล็อกย่อยๆ บล็อกละ 32kB ก่อน หลังจากนั้นแต่ละบล็อกจะใช้อัลกอริทึม LZ77 ในการหาบล็อกย่อยๆที่มีบิตซ้ำกัน ในลักษณะที่เรียกว่า Sliding

Window หลังจากนั้นจะสร้างตารางเพื่อเก็บ Symbol ขึ้นมา ซึ่งลำดับของ Symbol แสดงในตารางที่ 2 ดังนี้

ตารางที่ 2 การใช้งานของตาราง Symbol ในอัลกอริทึม DEFLATE

Symbol ตัวที่	การใช้งาน
0-255	Symbol 256 ตัวที่มีการใช้งานทั่วไป
256	เพื่อแสดงว่าจบ Data Stream ให้หยุดการทำงาน
257-285	เป็นการรวมบิตพิเศษ ประกอบด้วยความยาว 3-258 ไบต์
286 และ 287	ไม่มีการใช้งาน (Reserved)

และหลังจากนั้นจะสร้างตาราง Distance Code เพื่อเก็บระยะทางที่จะเข้าไปแทนที่ Symbol ที่ระบุไว้ในตัวข้อมูล

อัลกอริทึม DEFLATE นี้ได้ผลดีมากในการบีบอัดข้อมูลที่เป็นลักษณะไบนารี เช่น ซอฟต์แวร์ต่างๆ จนนำไปพัฒนาต่อมีการใช้งานที่หลากหลายเช่น LZH ซึ่งใช้ในการบีบอัดภาพประเภท GIF แต่วิธีนี้ไม่เหมาะกับการบีบอัดไฟล์ขนาดเล็กและเท็กซ์ไฟล์เนื่องจากทำงานได้ช้ากว่าวิธีแบบ Entropy Encoding เช่น Huffman Coding และต้องเสียพื้นที่ในการสร้าง Header ต่างๆในแต่ละบล็อกมากกว่าวิธีอื่นๆ

2.4 Run Length Encoding (RLE)

RLE (Internet Engineering Task Forge, Online) เป็นวิธีการบีบอัดข้อมูลวิธีหนึ่ง โดยมีหลักการทำงานที่ง่าย โดยการจับข้อมูลหรือกลุ่มของข้อมูลที่ซ้ำกัน และรวบเป็นข้อมูลชุดเดียวกัน โดยมีหมายเลขบอกจำนวนที่ซ้ำ เช่น มีข้อมูลอยู่กลุ่มหนึ่งคือ

WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWW
WWWWWWWWWBWWWWWWWWWWWWW

เราสามารถบีบอัดข้อมูลโดยรวบข้อมูลได้ดังนี้

12WB12W3B24WB14W

โดยตัวเลขนั้นจะบอกจำนวนที่ซ้ำของข้อมูล ซึ่งจากเดิมใช้พื้นที่ในการเก็บข้อมูลจำนวน 67 ไบต์ หลังการบีบอัดจะใช้พื้นที่ในการเก็บข้อมูลเพียง 16 ไบต์เท่านั้น ซึ่งในปัจจุบันมีวิธีเก็บข้อมูลเหล่านี้อยู่ 3 วิธีคือ PackBits, PCX และ ILBM แต่วิธีที่นิยมคือวิธี PackBits ซึ่งคิดค้นโดยบริษัท Apple และปัจจุบันใช้ในการบีบอัดไฟล์ภาพประเภท .TIFF (Tagged Image File Format) ซึ่ง PackBits มีวิธีในการเก็บข้อมูลดังนี้

ในตารางที่ 3 ให้ n เป็นค่าของเฮดเดอร์ในแต่ละส่วนซึ่งในเนื้อที่เก็บข้อมูล 1 ไบต์ ซึ่งมีค่าตั้งแต่ -127 ถึง 127 และแบ่งออกเป็น 2 กรณี แยกตามข้อมูลที่จะตามหลังเฮดเดอร์ดังแสดงในตารางที่ 3

ตารางที่ 3 ค่าเฮดเดอร์ขนาด 1 ไบต์ของการเก็บข้อมูลแบบ PackBits

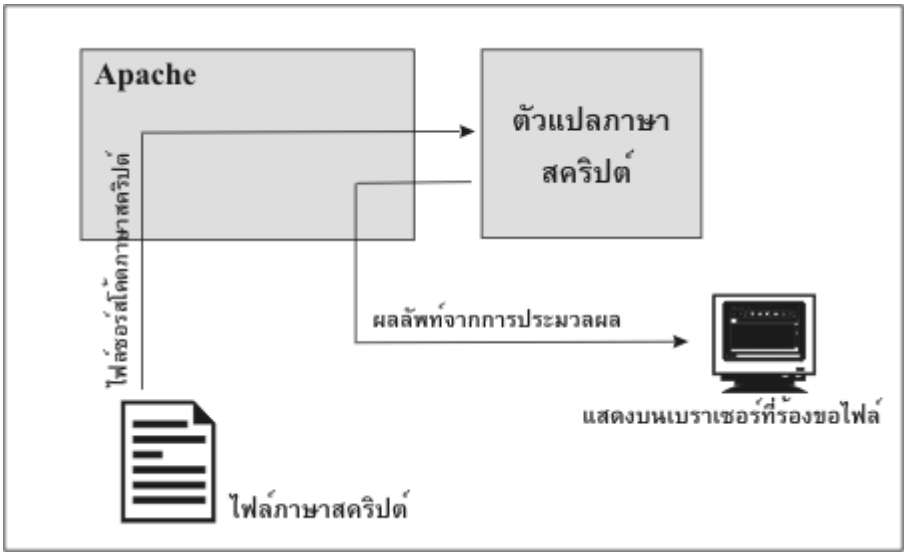
ค่าของเฮดเดอร์	ข้อมูลที่จะตามหลังเฮดเดอร์
0 ถึง 127	ชุดของข้อมูลขนาด 1+n ไบต์
0 ถึง -127	ข้อมูลไบต์เดี่ยว ที่มีการซ้ำกัน (1-n) ครั้ง

หากค่าของเฮดเดอร์เป็น 0 แสดงว่าจะไม่มีการเปลี่ยนแปลงข้อมูลชุดนั้นหลังการบีบอัดข้อมูล แต่วิธีนี้มีข้อเสียคือจะไม่ทราบจุดจบของไฟล์ (EOF) ดังนั้นส่วนหัวของไฟล์ที่ถูกบีบอัดจะต้องเก็บขนาดไฟล์ก่อนที่จะมีการบีบอัดข้อมูลไว้ด้วย

2.5 Common Gateway Interface (CGI)

CGI (W3C, Online) เป็นโปรโตคอลมาตรฐานที่ใช้เพื่อให้เว็บเซิร์ฟเวอร์และซอฟต์แวร์ทั่วไปสามารถติดต่อกันได้ โดยการให้เว็บเซิร์ฟเวอร์ส่งการร้องขอและแลกเปลี่ยนข้อมูลไปยังซอฟต์แวร์ภายนอกได้ โดยระบุพารามิเตอร์ที่โปรแกรมเหล่านั้นต้องการ แล้วจึงส่งผลลัพท์กลับมาให้เว็บเซิร์ฟเวอร์เพื่อให้ส่งข้อมูลไปยังเบราเซอร์ต่อไป โดยขั้นตอนวิธีการทำงานของ CGI จะแสดงในภาพที่ 1

ภาพที่ 1 รูปแบบการทำงานของ CGI บนเว็บเซิร์ฟเวอร์ Apache ในการแปลภาษาสคริปต์



ข้อดีของการใช้ CGI คือง่ายต่อการดูแลเนื่องจากเว็บเซิร์ฟเวอร์และโปรแกรมภายนอกนั้นแยกกันอยู่ หากต้องการเปลี่ยนแปลงตัวใดตัวหนึ่งก็สามารถทำได้ง่าย แต่หากมีการใช้งานปริมาณมากๆจะทำให้เว็บเซิร์ฟเวอร์ต้องเรียกซอฟต์แวร์ขึ้นมาทำงานทุกครั้งทำให้ใช้ทรัพยากรเครื่องมากและเครื่องทำงานไม่ไหว

การทำซอฟต์แวร์เพื่อทำหน้าที่เป็น CGI Gateway ที่ดีนั้นควรเป็นซอฟต์แวร์ที่เขียนจากภาษาที่ทำงานได้รวดเร็ว ใช้เวลาประมวลผลสั้น เช่น C เป็นต้น

บทที่ 3
งานวิจัยที่เกี่ยวข้อง

งานวิจัยทางด้านการบีบอัดข้อมูลซึ่งเกี่ยวข้องกับงานวิจัยนี้ มีดังนี้

3.1 Efficient Huffman Decoding

งานวิจัย Efficient Huffman Decoding(Yu-Chen and Chin-Chen 2001 : 97-99) โดยมีจุดมุ่งหมายเพื่อลดการใช้หน่วยความจำ และหน่วยประมวลผลในการทำ Decompression ของ Huffman Coding ซึ่งเดิมใช้หน่วยความจำอยู่ที่ $O(n^2)$ ต้องการลดให้เหลือ O(n) และการใช้หน่วยประมวลผล O(n) ให้เหลือเพียง O(log n) เมื่อค่า n เป็นจำนวน codeword ทั้งหมดของ Huffman Coding

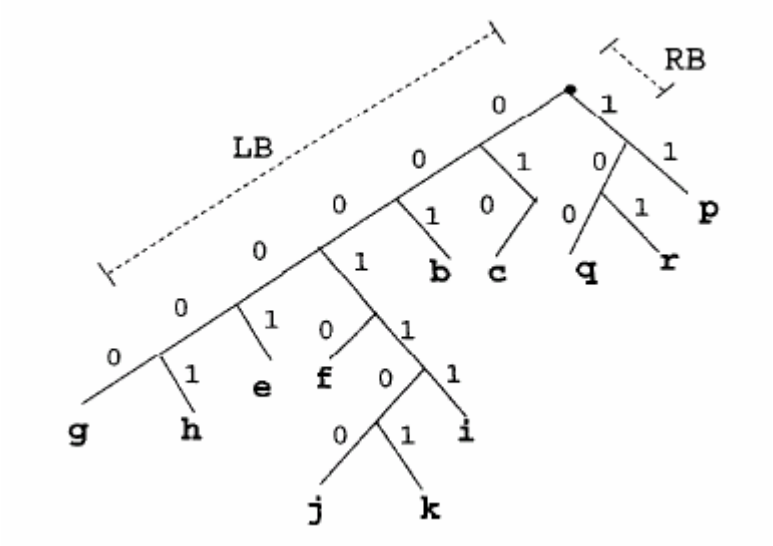
วิธีการลดการประมวลผลและการใช้หน่วยความจำคือ ใช้ตารางซึ่งมีการระบุโค้ดซึ่งเรียกว่า Canonical Code ซึ่งเก็บ codeword เป็นเลขฐาน 2 ของ Huffman Tree ทั้งหมด การ decompression จะแบ่งออกเป็น 2 ช่วงคือ

1. จะแบ่งข้อมูลออกเป็นหลายๆ ส่วนเป็น Partition โดยพิจารณาจากความยาวของ codeword ที่ไม่เท่ากันเป็นหลัก และแต่ละส่วนจะไม่ทับซ้อนกัน ทำให้ค้นหาตำแหน่งของข้อมูลได้ง่ายขึ้นมาก และสร้างตาราง Canonical Table ตามตารางที่ 4

2. ทำการ Decompression ซึ่งจะใช้หน่วยประมวลผลน้อยมาก

ในตารางที่ 4 ในคอลัมน์ Partition จะเก็บหมายเลขของส่วนของ Huffman Tree ย่อยๆ Symbol ก็คือ Canonical name ที่ได้ และ Codeword เก็บไบนารีของ Codeword และ actlen คือความลึกของ Tree เพื่อใช้หา offset ของข้อมูล ส่วน LMBC นั้นเป็นค่าที่ได้จากการคืนค่าของฟังก์ชั่น lmbc() ซึ่งเป็นค่าตำแหน่งบิตที่มีการเปลี่ยนแปลงจากบิตจุดเริ่มต้น ฟังก์ชั่นนี้สามารถทำงานได้ในการประมวลผล cycle เดียวเท่านั้น

Hu Yu-Chen และ Chang Chin-Chen ได้ยกตัวอย่าง Huffman Tree ที่มีข้อมูลดังภาพที่ 2 และสามารถเขียนตารางได้ดังภาพที่ 2 ดังต่อไปนี้



ภาพที่ 2 ตัวอย่าง Huffman Tree

ที่ผู้วิจัยนี้ได้ทดสอบ ตัวอักษรภาษาอังกฤษจะเป็น Canonical name ของตารางที่ 4

ตารางที่ 4 ตารางที่ได้จาก Huffman Tree ในภาพที่ 2

Partition	LMBC	Symbol	Codeword	*actlen*
P_1^L	1	**c**	010	3
P_2^L	2	**b**	001	3
P_3^L	3	**f**	0001000	5
	3	**f**	0001001	5
	3	**f**	0001010	5
	3	**f**	0001011	5
	3	**j**	0001100	7
	3	**k**	0001101	7
	3	**i**	0001110	6
	3	**i**	0001111	6
P_4^L	4	**e**	00001	5
P_5^L	5	**h**	000001	6
P_6^L	≥ 6	**g**	000000	6
P_1^R	1	**q**	100	3
	1	**r**	101	3
P_2^R	≥ 2	**p**	11	2

เมื่อทำตารางเสร็จแล้ว ก็จะมีการทำการ Decompression โดยสามารถเขียนเป็น Pseudocode ได้ตามภาพที่ 3

Partition Step:
if $(fkb(B,1) = 0)$ then
  $Basearray = Leftbase$
  $t_{max} = t^{L}_{max}$
else
  $Basearray = Rightbase$
  $t_{max} = t^{R}_{max}$

$i = lmbc(B)$
if $(i > t_{max})$ then $i = t_{max}$
$L = Basearray[1][i]$
$FC = Basearray[2][i]$
$Oarray = Basearray[3][i]$

Decoding Step:
$C = fkb(B, L)$
$index = C - FC + 1$
$Sym = Oarray[index][1]$
$actlen = Oarray[index][2]$

Next Iteration:
$B = lsh(B, actlen)$
if $(Sym \neq END)$ then
  goto **Partition Step**

ภาพที่ 3 ตารางแสดง Pseudocode แสดงการแบ่งข้อมูล และการ Decompression

ผลการทดลองคือ ในการ Decompression นั้นจะใช้หน่วยประมวลผลน้อยมาก เนื่องจากไม่ต้องค้นหา codeword ที่ต้องการที่อยู่ใน Huffman Tree ทั้งต้น ซึ่งในกรณีที่ดีที่สุด(Best Case) อยู่ที่ O(log n) โดยที่ n เป็นจำนวน Codeword ใน Huffman Tree

แต่อย่างไรก็ดี หน่วยความจำที่ใช้ในการทำ Decompression นั้นยังต้องใช้มากอยู่ เนื่องจากต้องใช้เก็บตาราง Canonical Table แทนในกรณีที่ Codeword จำนวนไม่มากถูกกระจายออกเป็นหลายๆ Branch มากๆ ผู้วิจัยได้ยกตัวอย่างเช่น codeword ที่มีเพียง 2 ตัวคือ {000100 , 00011111} (จะแตกเป็น Complete Binary Tree) จะต้องใช้ตารางสำหรับเก็บข้อมูลถึง 16 แถว

3.2 Level Compressed Huffman Decoding

งานวิจัย Level Compressed Huffman Decoding (Chung and Wu 1999 : 1455-1457) นี้จัดทำโดย และ โดยมีจุดประสงค์เพื่อศึกษาเวลาที่ใช้ในการ Decompression ของ Huffman Coding โดยวิธี breadth-first search ซึ่งเดิมการค้นหา codeword บน Huffman Tree ส่วนใหญ่ที่ใช้นั้นเป็นในรูปแบบ depth-first search โดยการใช้หน่วยความจำจะอยู่ที่ 5n-4 และเวลาที่ใช้ในการประมวลผลอยู่ที่ O(d) โดยที่ d นั้นเป็นความลึกของ Huffman Tree และ n เป็นจำนวนโหนดบน Huffman Tree

จากการทดลองคร่าวๆของ Kuo-Liang Chung และ Jung-Gen Wu พบว่าการทำงานแบบ breadth-first search นั้นจะใช้หน่วยความจำได้อย่างมีประสิทธิภาพเพิ่มขึ้น 34-39% และใช้เวลาในการประมวลผลลดลง 50-76% โดยเปรียบเทียบกับอัลกอริทึมที่มีอยู่เดิมคือ depth-first search ปกติและแบบที่ใช้ canonical table

Kuo-Liang Chung และ Jung-Gen Wu ได้ทำการทดลองโดยเตรียมข้อมูลภาพถ่ายขาวดำแบบ 256 gray levels จำนวน 4 ภาพขนาด 512x512 พิกเซล โดยตั้งชื่อภาพว่า Lena, Baboon, Pepper และ F16 ตามลำดับและใช้ Huffman Coding เพื่อทำการบีบอัด และ Decompress โดยใช้อัลกอริทึมที่แสดงใน Pseudocode ในภาพที่ 3.3 โดยทำบนเครื่องคอมพิวเตอร์ที่มีหน่วยประมวลผล Pentium II ความเร็ว 233Mhz


```
code_ptr ← d'
array_ptr ← decimal value of Huf_array [0, 1, ···, d' − 1]
while (CH_array [array_ptr] is not a symbol)
array_ptr ← array_ptr + CH_array [array_ptr]
    /* point to the left son */
If (Huf_array [code_ptr] ≠ 0) then array_ptr ← array_ptr + 1
    /* point to the right son if the code bit is "1" */
code_ptr ← code_ptr + 1
    /* point to the next code bit */
end while
output CH_array [array_ptr].
```

ภาพที่ 4 : Pseucode แสดงอัลกอรึทึมคลายการบีบอัด Huffman Coding โดยใช้ Breadth-first search

ส่วนผลการทดลองนั้นได้ตามภาพที่ 4 โดยที่ M เป็นขนาดหน่วยความจำที่ใช้เป็นบิตในการ decompression โดยวิธี breadth-first search ส่วน M[2] และ M[3] เป็นการ decompression วิธีเดิมซึ่งแสดงเป็นจำนวนบิตเช่นเดียวกัน ส่วน R1 และ R2 เป็นสัดส่วนของหน่วยความจำที่ใช้น้อยลงเป็นเปอร์เซ็นต์ของทั้ง 2 วิธีเดิม โดยคำนวณได้จากสมการดังนี้

$$R1 = \frac{(M[2] - M) \times 100}{M[2]} \quad \text{และ} \quad R2 = \frac{(M[3] - M) \times 100}{M[3]}$$

ส่วน T[2] และ T[3] นั้นคือเวลาที่ใช้ในอัลกอริทึมแบบเดิมทั้ง 2 แบบ และ T คือเวลาที่ใช้กับอัลกอริทึมแบบ breadth-first search และ R1 กับ R2 คือสัดส่วนของเวลาที่ใช้น้อยลงเป็นเปอร์เซ็นต์ซึ่งได้จากสมการดังนี้

$$R1 = \frac{(T[2] - T) \times 100}{T[2]} \quad \text{และ} \quad R2 = \frac{(T[3] - T) \times 100}{T[3]}$$

เมื่อทำการทดลองและบันทึกผล ได้ผลการทดลองตามตารางที่ 5

ตารางที่ 5 ผลการทดลอง

ภาพ	M[2]	M[3]	M	R1	R2	T Level Compressed Huffman Decoding2]	T[3]	T'	T	R3	R4
Lena	6092	4221	3087	49.3	26.9	1.24	0.85	0.87	0.73	41.1	14.1
Baboon	6066	4203	3069	49.4	27.0	1.24	0.92	0.86	0.73	41.1	20.7
Pepper	6144	4257	4131	32.8	3.0	1.16	0.80	0.81	0.74	36.2	7.5
F16	6066	4203	3933	35.2	6.4	1.23	0.84	0.86	0.81	34.1	3.6

จากผลการทดลองในตารางที่ 5 Kuo-Liang Chung และ Jung-Gen Wu ได้สรุปว่า การ Decompression แบบ breadth-first search นั้นเป็นวิธีที่ง่ายและใช้หน่วยความจำดีขึ้นกว่าวิธีเดิม

ประมาณ 1.1% ถึง 4.1% และใช้เวลาในการทำงานน้อยลง 6.5% ถึง 18.2% ซึ่งดีขึ้นกว่าเดิมเล็กน้อย แต่งานวิจัยนี้ก็ได้ทำกับตัวอย่างที่มีขนาดเล็ก และเป็นกลุ่มเล็กๆ จึงต้องมีการทดลองในข้อมูลตัวอย่างอื่นๆด้วย

3.3 A data compression scheme for Chinese text files using Huffman coding and a two-level dictionary

งานวิจัย A data compression scheme for Chinese text files using Huffman coding and a two-level dictionary (Ongshell and Huang 1995 : 85-99) นี้จัดทำโดยมีจุดประสงค์เพื่อบีบอัดอักขระอักษรจีน ซึ่งปกติแล้วไฟล์จะมีขนาดใหญ่ เนื่องจากอักขระจีนใช้พื้นที่ในการเก็บข้อมูลหลายไบต์ และแต่ละตัวอักษรบางตัวจะมีการใช้งานซ้ำกัน การทำงานจะใช้ 2 วิธีคือ การรวบรวมทางสถิติเกี่ยวกับอักขระจีนที่มีการใช้งานบ่อย และทำเป็น Dictionary และแทนค่าใน dictionary และหลังจากนั้นจึงบีบอัดด้วย Huffman Coding

ในอักขระภาษาอังกฤษและภาษาส่วนใหญ่นั้นจะใช้ 1 ไบต์แทนที่อักขระ 1 ตัว แต่ในอักษรจีนตามมาตรฐานการเข้ารหัส GB2312-80 นั้นอักขระ 1 ตัวจะถูกแทนด้วยจำนวนไบต์ถึง 7 ไบต์ การใช้ Huffman Coding จึงเป็นวิธีที่ดีมากในการบีบอัดข้อมูลเหล่านี้ เนื่องจากอักขระที่ใช้งานบ่อยจะถูกแทนที่ด้วยบิตจำนวนน้อยมากๆ ทางผู้วิจัยได้เตรียมไฟล์ทดสอบจำนวน 4 ไฟล์ตามตารางที่ 6 ดังนี้

ตารางที่ 6 ตารางแสดงจำนวนตัวอักษร และจำนวนตัวอักษรที่แตกต่างกันในข้อมูลตัวอย่าง

ไฟล์ที่	จำนวนตัวอักษรทั้งหมด	จำนวนตัวอักษรที่แตกต่างกัน	จำนวนตัวอักษรที่แตกต่างกัน (%)
1	11,666	1,257	10.8
2	14,671	1,694	11.5
3	15,423	1,348	8.7
4	41,760	2,252	5.4

หลังจากนั้น Ghim Hwee Ong และ Shell Ying Huang ได้ศึกษาการกระจายของข้อมูล และสร้างเป็น Dictionary ตามความถี่ของตัวอักษรจีนที่พบ ซึ่ง dictionary นั้นเป็น dictionary แบบ 2 ชั้น

เพื่อความรวดเร็วในการค้นหา และหลังจากนั้นก็ทำการบีบอัดไฟล์ ซึ่งผู้วิจัยได้ใช้ 4 อัลกอริทึมในการบีบอัดมีดังนี้

1. Huffman Coding ร่วมกับ Dictionary
2. 2 pass Huffman method
3. Adaptive Huffman Method
4. Adaptive LZW method

ขนาดของไฟล์ที่ถูกบีบอัดและอัตราส่วนได้ถูกแสดงในตารางที่ 7 ดังนี้

ตารางที่ 7 แสดงขนาดของไฟล์ที่ถูกบีบอัด (ไบต์) และอัตราส่วนที่บีบอัดได้ (%)

วิธีการบีบอัด	ขนาดของไฟล์ที่ถูกบีบอัด(ไบต์) และอัตราส่วน (%)			
	ไฟล์ที่ 1	ไฟล์ที่ 2	ไฟล์ที่ 3	ไฟล์ที่ 4
Huffman Coding ร่วมกับ Dictionary	16121(39.8)	20135(40.8)	23109(38.7)	59524(39.6)
2 pass Huffman method	21026(21.5)	26083(23.3)	30057(20.3)	76814(22.0)
Adaptive Huffman Method	20796(22.4)	25856(23.9)	29656(21.3)	76428(22.4)
Adaptive LZW method	18989(29.1)	24337(28.4)	25535(32.3)	67443(31.5)

จากผลการทดลอง Ghim Hwee Ong และ Shell Ying Huang ได้สรุปว่า วิธีแรกคือการใช้ Huffman Coding คู่กับ Dictionary นั้นจะบีบอัดได้อัตราส่วนที่ดีกว่าวิธีอื่นๆโดยเฉลี่ย 21.8 ถึง 22.5 เปอร์เซ็นต์

3.4 A new lossless compression scheme based on Huffman coding scheme for image compression

งานวิจัย A new lossless compression scheme based on Huffman coding scheme for image compression(Hu and Chin-Chen 2001 : 97-99) นี้จัดทำเพื่อหาวิธีที่เหมาะสมในการทำบีบอัดภาพถ่ายในลักษณะ Lossless compression คือเมื่อคลายการบีบอัดแล้วก็จะได้ภาพที่คงเดิมไม่สูญเสียรายละเอียด ซึ่งในปัจจุบันมีการใช้งาน 2 รูปแบบคือ Lossless JPEG 2 เวอร์ชั่น คือบีบอัดด้วย Huffman Coding และบีบอัดโดยทางฟังก์ชั่นคณิตศาสตร์

สำหรับเวอร์ชั่น Lossless JPEG ที่เป็น Huffman Coding ในปัจจุบัน จะพิจารณาการบีบอัดจากวิธีการที่เรียกว่า Most-likely magnitudeหรือ MLM ซึ่งหาได้จาก Histogram ของภาพ

สำหรับงานวิจัยนี้สร้างการบีบอัดวิธีใหม่ เรียกว่าวิธีการ enhanced MLM(EMLM) โดย EMLM ที่ปรับปรุงวิธีการทำงาน ผู้วิจัยได้ทดลองบนเครื่อง SUN SPARC 10 และใช้วิธีการ DPCM (Hu 2000 : 367-372) คู่กับ MLM และ EMLM นอกจากนั้นมีการเปรียบเทียบกับอัลกอริทึม GZIP ด้วย การทดลองนั้นจะบีบอัดภาพขาวดำแบบ 256 scale ขนาด 512x512 โดยผลการทดลองจะเปรียบเทียบผลจากอัตราส่วนที่บีบอัดได้ โดยคำนวณได้จากสมการดังนี้

$$\text{อัตราส่วนที่บีบอัดได้} = \frac{\text{ขนาดของไฟล์ต้นฉบับ}}{\text{ขนาดของไฟล์หลังการบีบอัด}}$$

Yu-Chen Hu และ Chin-Chen Chang ได้เตรียมภาพตัวอย่าง 6 ภาพตั้งชื่อว่า Airplane, Family, Boat, Lenna, Tiffany และ Toys ซึ่งผลที่ได้จากการบีบอัดจะแสดงตามตารางที่ 8 ดังนี้

ตารางที่ 8 แสดงอัตราส่วนการบีบอัดของการบีบอัดภาพตัวอย่างทั้ง 4 วิธี

ชื่อภาพ	อัตราส่วนการบีบอัด			
	GZIP	DPCM/Huffman	DPCM/MLM	DPCM/EMLM
Airplane	1.2284	1.4011	1.8351	1.8901
Boat	1.1373	1.2057	1.6360	1.6829
Family	1.0469	1.1391	1.7058	1.7582
Lenna	1.0733	1.2119	1.7712	1.8358
Tiffany	1.2491	1.3604	1.8038	1.8651
Toys	1.1736	1.2943	1.7152	1.7736

จากผลการทดลอง Yu-Chen Hu และ Chin-Chen Chang สรุปว่าวิธีการใหม่คือ EMLM นั้นดีกว่า DPCM/Huffman และ DPCM/MLM อย่างเห็นได้ชัด นอกจากนั้นวิธีการ EMLM ยังช่วยลดการสร้างโค้ดที่ไม่จำเป็น โดยจำนวนที่สร้างขึ้นมาจะเป็นจำนวนแบบ non-zero เท่านั้น

บทที่ 4
วิธีการดำเนินการวิจัย

งานวิจัยนี้ได้นำอัลกอริทึม Huffman Coding มาทดลองใช้ในการบีบอัดข้อมูลของเอกสาร HTML ซึ่งในการดำเนินงานวิจัย ผู้วิจัยได้ใช้ GCC (GNU C Compiler) ในการพัฒนาแอปพลิเคชันสำหรับการบีบอัดข้อมูลบนฝั่งเว็บเซิร์ฟเวอร์ โดยใช้ Apache 2 บนระบบปฏิบัติการ Linux เป็นตัวทดสอบ และ Perl ในการพัฒนาเบราเซอร์จำลองและโมดูลย์สำหรับการคลายการบีบอัดข้อมูล

ตารางที่ 9 ขั้นตอนและระยะเวลาการดำเนินการวิจัย

ที่	ขั้นตอนดำเนินการวิจัย	เดือนที่ 1	เดือนที่ 2	เดือนที่ 3	เดือนที่ 4	เดือนที่ 5	เดือนที่ 6
1	จัดเตรียมข้อมูล	←	→				
2	ศึกษาอัลกอริทึมและวิธีการที่เหมาะสม	←	→	→			
3	พัฒนาไลบรารี่ในการบีบอัดข้อมูล		←	→	→		
4	พัฒนาแอปพลิเคชันที่ทำหน้าที่เป็น CGI บนฝั่งเซิร์ฟเวอร์			←	→	→	
5	พัฒนาเบราเซอร์จำลอง และโมดูลย์ที่ใช้สำหรับทำ Decompression			←	→	→	
6	บันทึกและวัดผล				←	→	→
7	นำไปใช้					←	→
8	สรุปผลการวิจัยและจัดทำรายงานวิทยานิพนธ์					←	→

จากขั้นตอนการดำเนินการวิจัยตามตารางที่ 9 สามารถอธิบายได้ดังนี้

4.1 การจัดเตรียมข้อมูล

ผู้วิจัยได้จัดเตรียมข้อมูลเอกสาร HTML จากเว็บไซต์ต่างๆ โดยแบ่งข้อมูลออกเป็นส่วนๆ ดังนี้

ตารางที่ 10 จำแนกปริมาณข้อมูลตัวอย่างที่จัดเตรียมโดยประมาณ

ที่	ประเภทข้อมูล	จำนวนโดยประมาณ
1	เอกสาร HTML ทั่วไปจากเว็บไซต์ในประเทศ	50
2	เอกสาร HTML ทั่วไปจากเว็บไซต์ต่างประเทศ	50

โดยข้อมูลที่ถูกจัดเตรียมนั้นจะเก็บอยู่ในรูปเอกสาร HTML บนเว็บเซิร์ฟเวอร์ Apache เมื่อได้ข้อมูลมาครบ ผู้วิจัยพิจารณาแบ่งข้อมูลออกเป็นกลุ่มย่อยๆ ตามขนาดของไฟล์ เนื่องจากขนาดของไฟล์มีผลโดยตรงกับเวลาที่ใช้ในการบีบอัด โดยผู้วิจัยแบ่งกลุ่มโดยพิจารณาจากการกระจายตัวของข้อมูลเป็นหลัก

เนื่องจากขนาดของข้อมูลที่ได้มีขนาดแตกต่างกันมาก จึงได้หาค่าเฉลี่ยของขนาดของไฟล์ โดยมีค่าเฉลี่ยอยู่ที่ 50.03 กิโลไบต์ และส่วนเบี่ยงเบนมาตรฐาน(Standard Deviation) อยู่ที่ 44.24 ทางผู้วิจัยจึงได้จำแนกข้อมูลออกเป็น 3 กลุ่มตามขนาดของไฟล์ โดยจำแนกตามการกระจายของข้อมูลคือ

- ไฟล์ข้อมูลขนาดใหญ่ (ขนาดไฟล์มากกว่า 113.3 กิโลไบต์)
- ไฟล์ข้อมูลขนาดกลาง (ขนาดไฟล์อยู่ระหว่าง 19.91 ถึง 113.3 กิโลไบต์)
- ไฟล์ข้อมูลขนาดเล็ก (ขนาดไฟล์น้อยกว่า 19.91 กิโลไบต์)

4.2 อัลกอริทึมและวิธีการที่เหมาะสม

ในงานวิจัยนี้ผู้วิจัยศึกษาอัลกอริทึม Huffman Coding และการสร้าง Huffman Tree โดยการใช้วิธี Linear Timing จากข้อมูลที่ได้มาจากการจัดเตรียมข้อมูลเพื่อหาวิธีการที่เหมาะสมที่สุดคือใช้เวลาในการทำงานและใช้หน่วยประมวลผลน้อยที่สุดในการออกแบบและพัฒนาระบบที่ใช้ในการทดลอง

4.3 พัฒนาไลบรารี่ในการบีบอัดข้อมูล

ขั้นตอนนี้เป็นการพัฒนาไลบรารี่สำหรับบีบอัดข้อมูลทั้ง 3 อัลกอริทึมคือ Huffman Coding, Shannon-Fano และ GZIP ด้วยภาษา GCC บนระบบปฏิบัติการ Linux และคอมไพล์เป็นเฮดเดอร์ (*.h) เพื่อให้สะดวกแก่การนำไปใช้บีบอัดและคลายการบีบอัด ในการเปรียบเทียบประสิทธิภาพการทำงานของอัลกอริทึมในขั้นตอนที่ 4.6.2

4.4. การพัฒนาแอปพลิเคชันที่ทำหน้าที่เป็น CGI บนฝั่งเซิร์ฟเวอร์

แอปพลิเคชันที่ทำงานในขั้นตอนนี้แบ่งออกเป็น 3 ขั้นตอนย่อยคือ

4.4.1 รับพารามิเตอร์มาจากเว็บเซิร์ฟเวอร์ เปิดเอกสาร HTML

ขั้นตอนนี้เป็นการเปิดไฟล์เอกสารจากพารามิเตอร์ที่เว็บเซิร์ฟเวอร์ส่งให้ โดยการใช้ Standard I/O ของระบบปฏิบัติการ Linux และเปิดไฟล์เอกสารตามตำแหน่งที่ระบุไว้ในพารามิเตอร์ เมื่อเปิดไฟล์ได้แล้ว จะนำข้อมูลต่างๆภายในไฟล์เก็บไว้ในบัฟเฟอร์บนหน่วยความจำหลักบนเครื่องเว็บเซิร์ฟเวอร์เพื่อบีบอัดข้อมูลในขั้นตอนถัดไป

4.4.2 บีบอัดข้อมูล

ในขั้นตอนนี้จะใช้โปรแกรมที่ทำหน้าที่เป็น CGI บีบอัดข้อมูลโดยใช้ฟังก์ชั่นต่างๆบนไลบรารี่ที่พัฒนาและคอมไพล์ในขั้นตอนที่ 4.3 เพื่อบีบอัดข้อมูล และมีการจับเวลา บันทึกจำนวนไบต์ก่อนและหลังบีบอัดข้อมูลในแต่ละไฟล์

4.4.3 เปลี่ยนข้อมูลในเฮดเดอร์ของโปรโตคอล HTTP

ขั้นตอนนี้จะเป็นการเปลี่ยนเฮดเดอร์ของโปรโตคอล HTTP เพื่อให้เบราเซอร์ทราบว่าข้อมูลที่ถูกส่งมาจากเว็บเซิร์ฟเวอร์นั้นมีการบีบอัดแบบ Huffman Coding และต้องใช้ไลบรารี่ในการคลายการบีบอัดที่สร้างขึ้น โดยทางผู้วิจัยเลือกใช้เฮดเดอร์ที่มีชื่อว่า Accept-Encoding ซึ่งเป็นชื่อเฮดเดอร์

ตัวเดียวกับที่ mod_gzip ของเว็บเซิร์ฟเวอร์ Apache ใช้ในการบ่งบอกชนิดของการบีบอัดกับเบราเซอร์ แต่เปลี่ยนค่าจาก compress, gzip เป็น compress, huffman แทน ตามตัวอย่างด้านล่างนี้

HTTP/1.1 200 OK

Date: Mon, 08 Apr 2006 23:35:42 GMT

Server: Apache/2.0.51 (Linux)

Last-Modified: Thu, 14 Mar 1996 23:00:00 GMT

Accept-Ranges: bytes

Content-Length: 25494

Connection: close

Accept-Encoding: compress, huffman

Content-Type: text/plain

เมื่อเปลี่ยนเฮดเดอร์ของโปรโตคอล HTTP แล้ว จะส่งข้อมูลที่ผ่านการบีบอัดแล้วให้กับเบราเซอร์ผ่านทางเว็บเซิร์ฟเวอร์ Apache ต่อไป

4.5 การพัฒนาเว็บเบราเซอร์จำลอง

เบราเซอร์จำลองจะถูกเขียนขึ้นด้วยภาษา Perl เนื่องจากเป็นภาษาที่จัดการด้านสตริงได้ดี และสามารถพัฒนาเว็บเบราเซอร์จำลองได้บนระบบปฏิบัติการทุกระบบปฏิบัติการ สำหรับในงานวิจัยนี้ใช้ภาษา Perl พัฒนาบนระบบปฏิบัติการ Linux เนื่องจากใช้ทรัพยากรและหน่วยประมวลผลน้อยกว่าบนระบบปฏิบัติการอื่น พร้อมทั้งมีโมดูลย์และฟังก์ชันที่ใช้ด้านเครือข่ายครบถ้วนเช่น Socket::IO ในการติดต่อสื่อสาร รวมทั้งการรับส่งข้อมูลจากเครือข่าย

ในที่นี้ผู้วิจัยจะใช้โมดูลย์ในภาษา Perl ชื่อ Socket::IO ในการดึงข้อมูลจากเว็บเซิร์ฟเวอร์ Apache มาเก็บไว้ในบัฟเฟอร์ (Buffer) พร้อมทั้งแสดงผลข้อมูลเอกสาร HTML ได้ ตามความต้องการข้อผู้วิจัย นอกจากนั้นยังมีคุณสมบัติเพิ่มเติมดังนี้

- สามารถคลายการบีบอัดในแบบ GZIP, Huffman Coding และ Shannon-Fano ได้
- สามารถจับเวลาตั้งแต่เริ่มส่งคำขอ (request) ไปยังเว็บเซิร์ฟเวอร์จนกระทั่งข้อมูลถูกส่งกลับได้
- สามารถเปิดเปิด Web Cache ได้
- สามารถส่งคำขอเป็นจำนวนครั้งตามที่ผู้วิจัยต้องการและจับเวลารวมได้

4.6 การวัดและบันทึกผล

ผู้วิจัยได้แบ่งการทดลองออกเป็น 4 ส่วนคือ

4.6.1 การวัดอัตราส่วนการบีบอัดในแต่ละอัลกอริทึม

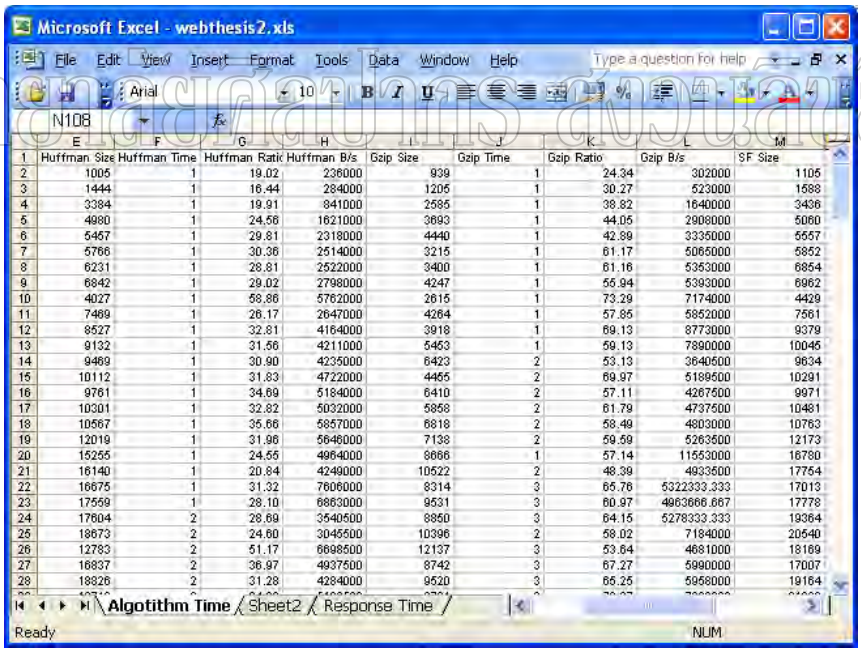
4.6.2 การเปรียบเทียบประสิทธิภาพอัลกอริธึมบนเว็บเซิร์ฟเวอร์

4.6.3 การเปรียบเทียบ Turn Around Time

4.6.4 เปรียบเทียบเวลาทั้งหมดของเซิร์ฟเวอร์ที่มีการใช้งานจริง

4.6.5 การวัด Turn Around Time ของเซิร์ฟเวอร์ที่มีการใช้งานจริง

ซึ่งเมื่อทำการทดลองเสร็จในแต่ละส่วน ผู้วิจัยทำการบันทึกข้อมูลลงในเอกสารในลักษณะตารางเพื่อคำนวณและเรียงข้อมูลดิบเพื่อให้พร้อมทำการประมวลผลใน Microsoft Excel ดังที่แสดงในภาพที่ 5 ต่อไป



	E	F	G	H	I	J	K	L	M
1	Huffman Size	Huffman Time	Huffman Ratio	Huffman B/s	Gzip Size	Gzip Time	Gzip Ratio	Gzip B/s	SF Size
2	1005	1	19.02	236000	939	1	24.34	302000	1105
3	1444	1	16.44	284000	1205	1	30.27	523000	1588
4	3384	1	19.91	841000	2585	1	38.82	1640000	3436
5	4980	1	24.56	1621000	3693	1	44.05	2908000	5060
6	5457	1	29.81	2318000	4440	1	42.89	3335000	5557
7	5766	1	30.36	2514000	3215	1	61.17	5065000	5852
8	6231	1	28.81	2522000	3400	1	61.16	5353000	6854
9	6842	1	29.02	2798000	4247	1	55.94	5393000	6962
10	4027	1	58.86	5762000	2615	1	73.29	7174000	4429
11	7489	1	26.17	2647000	4264	1	57.85	5852000	7561
12	8527	1	32.81	4164000	3918	1	69.13	8773000	9379
13	9132	1	31.56	4211000	5463	1	59.13	7890000	10045
14	9469	1	30.90	4235000	6423	2	53.13	3640500	9634
15	10112	1	31.83	4722000	4455	2	69.97	5189500	10291
16	9761	1	34.69	5184000	6410	2	57.11	4267500	9971
17	10301	1	32.82	5032000	5858	2	61.79	4737500	10481
18	10567	1	35.66	5857000	6818	2	58.49	4803000	10763
19	12019	1	31.96	5646000	7138	2	59.59	5263500	12173
20	15255	1	24.55	4964000	8666	1	57.14	11553000	16780
21	16140	1	20.84	4249000	10522	2	48.39	4933500	17754
22	16675	1	31.32	7606000	8314	3	65.76	5322333.333	17013
23	17559	1	28.10	6863000	9531	3	60.97	4963666.667	17778
24	17604	2	28.69	3540500	8850	3	64.15	5278333.333	19364
25	18673	2	24.60	3045500	10396	2	58.02	7184000	20540
26	12783	2	51.17	6698500	12137	3	53.64	4681000	18169
27	16837	2	36.97	4937500	8742	3	67.27	5990000	17007
28	18826	2	31.28	4284000	9520	3	66.25	5958000	19164

ภาพที่ 5 แสดงการเก็บข้อมูลดิบใน Microsoft Excel

4.6.1 การวัดอัตราส่วนการบีบอัดในแต่ละอัลกอริทึม

ในส่วนนี้เป็นการจับเวลาอัตราส่วนข้อมูลที่บีบอัดได้ในแต่ละอัลกอริทึมคือ GZIP, Shanon Fano และ Huffman Coding โดยอัตราส่วนการบีบอัดข้อมูลนั้นจะได้จากส่วนต่างระหว่างข้อมูลเก่าก่อนที่จะบีบอัดและข้อมูลใหม่หลังการบีบอัดแล้วเป็นเปอร์เซ็นต์ โดยประสิทธิภาพในการบีบอัดข้อมูล สามารถเขียนได้ตามสมการได้ดังต่อไปนี้

$$\text{อัตราส่วนในการบีบอัดข้อมูล (\%)} = \frac{\text{ข้อมูลเก่าก่อนการบีบอัด} - \text{ข้อมูลหลังการบีบอัด}}{\text{ข้อมูลก่อนการบีบอัด}} \times 100$$

จากสมการ อัตราส่วนการบีบอัดข้อมูล ถ้ายิ่งน้อย แสดงว่าการบีบอัดนั้นมีประสิทธิภาพในการย่อขนาดไฟล์มาก แต่อย่างไรก็ดีประสิทธิภาพการบีบอัดนั้นขึ้นกับเวลาที่ใช้ในการบีบอัดข้อมูลด้วย ซึ่งจะแสดงวิธีการเปรียบเทียบในขั้นตอนที่ 4.6.2

4.6.2 การเปรียบเทียบประสิทธิภาพของอัลกอริธึม

เป็นการทดลองเพื่อวัดประสิทธิภาพของอัลกอริธึมบนฝั่งเว็บเซิร์ฟเวอร์ โดยผู้วิจัยได้พัฒนาโปรแกรมจากภาษา C (GNU C Compiler) บนระบบปฏิบัติการ Linux ทำการทดลองเปรียบเทียบบีบอัดเว็บเพจตัวอย่างภาษาไทยและภาษาอังกฤษ จำนวน 100 เว็บเพจ

ผู้วิจัยได้ทดลองบีบอัดข้อมูลเว็บเหล่านี้ด้วยอัลกอริทึม GZIP และ Huffman Coding และเปรียบเทียบจาก Throughput ที่วัดได้

การวัดประสิทธิภาพของผู้วิจัยได้วัดจาก Throughput ซึ่งหาได้จากจำนวนไบต์ที่หายไปโดยการบีบอัดต่อ 1 วินาที โดยคำนวณจากสมการดังนี้

$$\text{Throughput(ไบต์/วินาที)} = \frac{\text{ขนาดก่อนบีบอัด(ไบต์)} - \text{ขนาดหลังบีบอัด(ไบต์)}}{\text{เวลาที่ใช้บีบอัด (วินาที)}}$$

4.6.3 การเปรียบเทียบ Turn Arround Time

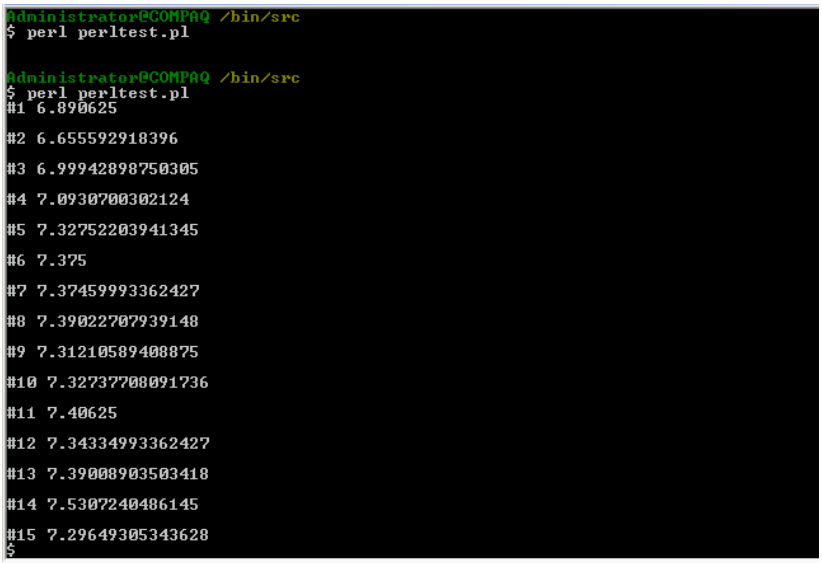
ผู้วิจัยได้ทดลองเขียนโปรแกรมที่ทำงานร่วมกับเว็บเซิร์ฟเวอร์ Apache ในลักษณะ CGI (Common Gateway Interface) โดยใช้ภาษา C เพื่อบีบอัดข้อมูลเอกสาร HTML ด้วย GZIP และ

อัลกอริธึม Huffman Coding ขณะให้บริการแบบ on-the-fly บนฝั่งเว็บเซิร์ฟเวอร์ก่อนที่จะส่งข้อมูลเว็บเพจมายังเว็บบราวเซอร์จำลองซึ่งเขียนด้วยภาษา Perl ที่สามารถส่งคำร้องขอไปยังเว็บเซิร์ฟเวอร์ผ่านโปรโตคอล HTTP ในขณะที่ปิดการทำงานของ Web Cache ทั้ง 2 ฝั่ง และจับเวลารวมตั้งแต่เว็บบราวเซอร์ส่งคำร้องขอมายังเว็บเซิร์ฟเวอร์ จนกระทั่งข้อมูลจากเว็บเซิร์ฟเวอร์ส่งมายังเบราเซอร์จำลองเสร็จจำนวน 100 ครั้ง และเปรียบเทียบเวลารวมของ GZIP และHuffman Coding โดยเฮดเดอร์ที่ส่งไปยังเว็บเซิร์ฟเวอร์ผ่านภาษา Perl มีดังนี้

```
$header = "GET /cgi-bin2/$i.htm HTTP/1.0\n";          (1)
$header .= "Accept-Encoding: compress/huffman\n";     (2)
$header .= "Cache-Control: no-cache\n";               (3)
$header .= "User-Agent: close\n\n";                   (4)
```

ในบรรทัดแรก คือชื่อไฟล์ที่ต้องการร้องขอ บรรทัดที่ 2 คือการเข้ารหัสหรือบีบอัดข้อมูลที่เว็บเบราเซอร์จะรองรับ ในบรรทัดที่ 3 คือไม่ให้มีการทำ cache คือต้องการให้เว็บเซิร์ฟเวอร์มีการบีบอัดข้อมูลทุกครั้งเมื่อมีการร้องขอ เพื่อให้ได้ผลการทดลองที่ถูกต้อง และในบรรทัดที่ 4 คือให้มีการตัดการเชื่อมต่อทุกครั้งเมื่อจบการร้องขอข้อมูล ไม่ให้เว็บเซิร์ฟเวอร์ทำการ Keep-Alive ซึ่งเป็นวิธีการที่เว็บเซิร์ฟเวอร์ให้บริการข้อมูลครั้งละหลายๆไฟล์ในการเชื่อมต่อแต่ละครั้ง ซึ่งอาจทำให้ผลการทดลองคลาดเคลื่อนได้

โดยผลการทดลองนั้น เมื่อใช้เบราเซอร์ที่พัฒนาขึ้นจะแสดงได้ตามภาพที่ 6 โดยเมื่อใช้คำสั่ง perl perltest.pl แล้ว ตัวคอมไพเลอรภาษา Perl จะทำการแปลไฟล์ซอร์สโค้ดที่ชื่อว่า perltest.pl และส่งคำร้องขอไฟล์หมายเลข 1 ถึง 100 ไฟล์ละ 100 ครั้ง และแสดงผลออกทางหน้าจอ โดย #n คือหมายเลขไฟล์ที่ทำการร้องขอ ส่วนตัวเลขด้านหลังคือเวลาที่ใช้ทั้งหมด หน่วยเป็นวินาที



ภาพที่ 6 แสดงผลการทำงานของเบราเซอร์จำลอง

4.6.4 เปรียบเทียบเวลาที่ใช้รวมทั้งหมดจากเว็บที่มีการใช้งานจริง

ผู้วิจัยได้เลือกเว็บไซต์ Yahoo.com (www.yahoo.com) เพื่อใช้ในการทดลองเนื่องจากเป็นเว็บไซต์ที่มีผู้เข้าชมเป็นจำนวนมาก และมีเนื้อหาเอกสารไฟล์ HTML เป็นจำนวนมาก และทำการเซฟหน้าเว็บเพจทั้งหมดในความลึก 2 ระดับชั้น และทำการร้องขอไฟล์ HTML ที่เซฟได้มาทั้งหมดในแต่ละอัลกอริทึม และทำการเปรียบเทียบในลักษณะแผนภูมิแท่งดังปรากฏในแผนภูมิที่ 7 ของบทที่ 5 (ผลการเปรียบเทียบ Turn-Around Time ของทั้ง 3 อัลกอริทึมบนเว็บเซิร์ฟเวอร์จริง)

4.6.5 Turn Around Time บนเว็บเซิร์ฟเวอร์ที่มีการใช้งานจริง

ผู้วิจัยได้ทดลองเขียนโปรแกรมที่ทำงานร่วมกับเว็บเซิร์ฟเวอร์ Apache บนเว็บเซิร์ฟเวอร์ที่มีการใช้งานจริง ซึ่งมีการส่งคำขอจากภายนอกตลอดเวลา โดยจับเวลาตั้งแต่ส่งคำขอจากเว็บเบราเซอร์จนกระทั่งข้อมูลทั้งหมดจากเว็บเซิร์ฟเวอร์ส่งกลับมายังเว็บเบราเซอร์จำนวน 100 ครั้ง

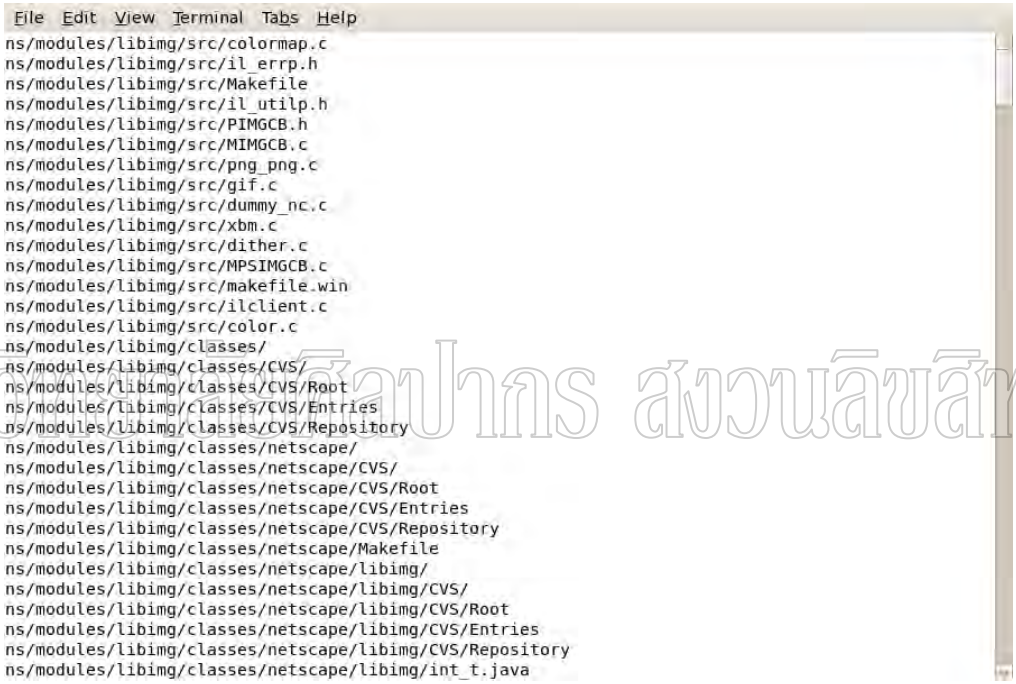
4.7 การพัฒนา Plug-In บนเว็บเบราเซอร์จริง

เมื่อจัดทำทดลองในเซิร์ฟเวอร์และเบราเซอร์จำลองแล้ว ผู้วิจัยสร้าง Plug-In สำหรับเบราเซอร์เพื่อใช้งานจริง โดยผู้วิจัยเลือกทำบนเบราเซอร์ Mozilla Firefox เนื่องจากเป็นเบราเซอร์ที่เป็น

Open Source โดยพัฒนาบน API(Application Programming Interface) ที่มีชื่อว่า Gecko ซึ่งเป็น API สำหรับพัฒนา Plug-In บนเบราเซอร์ Mozilla

โดยผู้วิจัยได้แบ่งการลำดับออกเป็น 3 ส่วนคือ

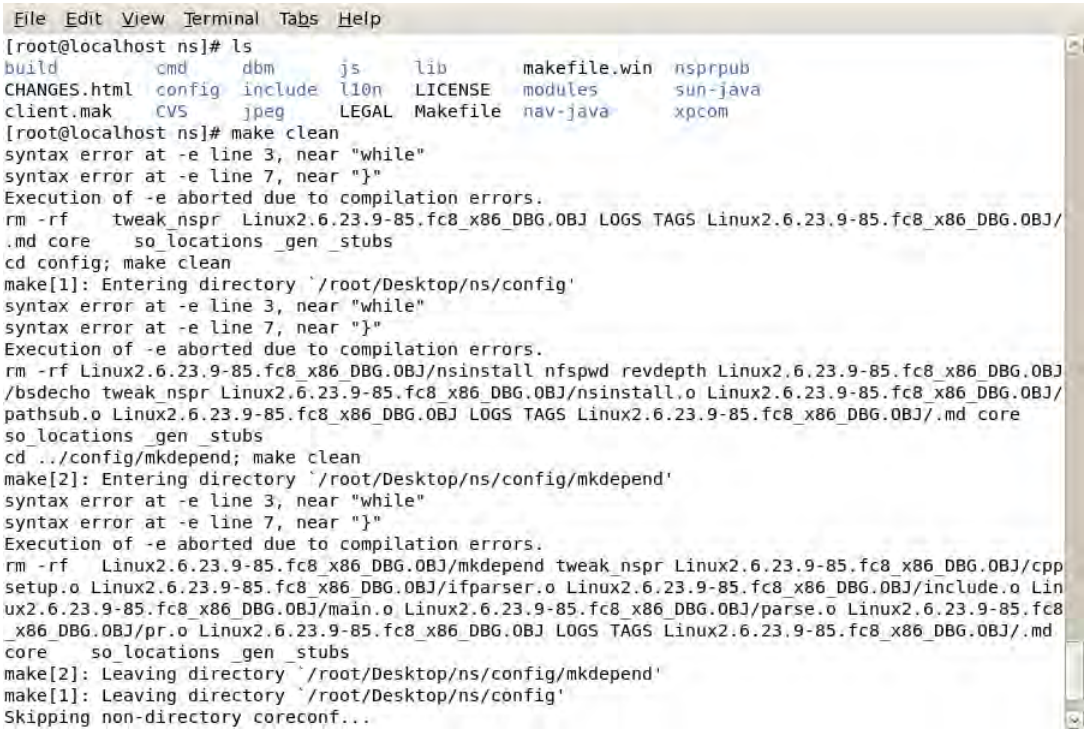
4.7.1 ดาวน์โหลดและ Decompression ซอร์สโค้ด ในขั้นตอนนี้ผู้วิจัยดาวน์โหลดซอร์สโค้ดของเว็บเบราเซอร์ Firefox จากเว็บไซต์ผู้พัฒนามาคอมไพล์ด้วยตัวเองมายังเครื่องคอมพิวเตอร์ที่ติดตั้งระบบปฏิบัติการ Linux โดยเมื่อดาวน์โหลดแล้วก็จะทำการแตกไฟล์ลงไดเร็กทอรี่ชั่วคราว ดังที่แสดงในภาพที่ 7



ภาพที่ 7 ภาพขณะแตกไฟล์ซอร์สโค้ดที่ได้จากเว็บไซต์ผู้พัฒนา

4.7.2 สร้างซอร์สโค้ดตัว Plug-In โดยทำเป็นไฟล์ ที่มีส่วนขยายเป็น *.m4 ภายในไฟล์จะเป็นฟังก์ชั่นในการ Decompression ไฟล์ที่ถูกบีบอัดด้วย Huffman Coding และทำการ include ไฟล์ไลบรารี่ ของซอร์สโค้ด Firefox ของผู้พัฒนา Firefox ที่ให้มาด้วย นำไฟล์ที่ได้ใส่ไว้ในไดเรกทอรี modules และทำการ configure ซอร์สโค้ด ผ่านคำสั่ง configure

4.7.3 ทำการคอมไพล์ซอร์สโค้ดที่ได้มาผ่านคำสั่ง make และติดตั้งโปรแกรมที่คอมไพล์ได้บนเครื่องผ่านคำสั่ง make install โดยขณะคอมไพล์จะแสดงได้ตามภาพที่ 8



ภาพที่ 8 ภาพขณะกำลังคอมไพล์ซอร์สโค้ด

เมื่อคอมไพล์ซอร์สโค้ดและติดตั้งแล้ว ผู้วิจัยจึงทำการทดสอบการร้องขอจากเว็บเซิร์ฟเวอร์และบันทึกผลการทดสอบต่อไป

บทที่ 5
ผลการดำเนินงานวิจัย

หลังจากได้ศึกษาอัลกอริทึมต่างๆ มาแล้วนั้น ผู้วิจัยได้ดำเนินการทดลองและรวบรวมผลการทดลองและรายงานผลการปฏิบัติงานออกเป็นส่วนๆ ตามขั้นตอนการดำเนินงานดังนี้

5.1 ข้อมูลที่รวบรวมได้และผลการจัดเตรียมข้อมูล

ทางผู้วิจัยได้รวบรวมเว็บไซต์ภาษาไทยและภาษาอังกฤษที่มีคนใช้งานกันมากๆ โดยเลือกจาก 100 อันดับแรกของการจัดอันดับบนเว็บไซต์ Netcraft และ TrueHit ซึ่งเป็นเว็บไซต์ที่รวบรวมสถิติการเข้าถึงเว็บไซต์ทั่วโลกไว้ และจากเว็บไซต์ TrueHits.net ซึ่งเป็นเว็บไซต์สำหรับเก็บสถิติการเข้าชมของแต่ละเว็บไซต์ในประเทศไทย ผู้วิจัยได้ตัดเว็บเพจที่เป็นหน้าเหมือนกันแต่ภาษาที่ใช้ต่างกันทิ้งไป เช่นgoogle.com กับ google.fr ซึ่งเป็นเว็บเพจที่หน้าตาเหมือนกัน แต่มีการใช้งานภาษาบนเว็บที่ต่างกันออก เพื่อให้เกิดข้อแตกต่างระหว่างตัวอย่าง ซึ่งรายละเอียดข้อมูลที่นำมาทดลองได้แสดงในตารางที่ 14 ถึงตารางที่ 16 ในภาคผนวก ก

5.2 ผลการพัฒนาไลบรารีในการบีบอัดข้อมูล

ผู้วิจัยได้พัฒนาไลบรารี่ในการบีบอัดข้อมูลทั้งแบบ GZIP และ Huffman Coding โดยใช้ภาษา GCC และคอมไพล์เป็นไฟล์ header ของภาษา C โดยมีฟังก์ชั่นต่างๆดังนี้

1. HuffmanCompress(*input , *output , int inputsize)

เป็นฟังก์ชั่นสำหรับบีบอัดข้อมูลด้วยวิธี Huffman Coding โดย *input คือตำแหน่งพอยเตอร์ของข้อมูลอินพุต(ก่อนการบีบอัด) และ *output คือตำแหน่งพอยเตอร์ของข้อมูลเอาท์พุต(หลังการบีบอัด) และ inputsize คือขนาดของข้อมูลก่อนการบีบอัด

2. HuffmanDecompress(*input , *output)

เป็นฟังก์ชั่นสำหรับคลายการบีบอัดข้อมูลโดย Huffman Coding โดย *input คือตำแหน่งพอยเตอร์ของข้อมูลอินพุต(ข้อมูลที่ถูกบีบอัดอยู่) และ *output คือตำแหน่งพอยเตอร์ที่ให้เขียนข้อมูลเอาท์พุต (หลังคลายการบีบอัด)

3. GZIPCompress(*input , *output , int inputsize , *buffer)

เป็นฟังก์ชั่นสำหรับบีบอัดข้อมูลด้วยวิธี GZIP โดย *input คือตำแหน่งพอยเตอร์ของข้อมูลอินพุต(ก่อนการบีบอัด) และ *output คือตำแหน่งพอยเตอร์ของข้อมูลเอาท์พุต(หลังการบีบอัด) inputsize คือขนาดของข้อมูลก่อนการบีบอัด และ *buffer คือตำแหน่งพอยเตอร์สำหรับพื้นที่ในการทำงาน(บัพเฟอร์)

4. GZIPDecompress(*input , *output , *buffer)

เป็นฟังก์ชั่นสำหรับคลายการบีบอัดข้อมูลโดย Huffman Coding โดย *input คือตำแหน่งพอยเตอร์ของข้อมูลอินพุต(ข้อมูลที่ถูกบีบอัดอยู่) และ *output คือตำแหน่งพอยเตอร์ที่ให้เขียนข้อมูลเอาท์พุต (หลังคลายการบีบอัด) และ *buffer คือตำแหน่งพอยเตอร์สำหรับพื้นที่ในการทำงาน(บัพเฟอร์)

5.3 ผลการพัฒนาแอปพลิเคชันที่ทำหน้าที่เป็น CGI Application บนฝั่งเซิร์ฟเวอร์

ผู้วิจัยได้พัฒนาแอบพลิเคชั่นที่ทำหน้าที่เป็น CGI Application ที่ทำหน้าที่บีบอัดข้อมูลโดย GZIP, Huffman Coding และ Shannon-Fano ทางผู้วิจัยได้ปรับให้ทำงานเข้ากับเว็บเซิร์ฟเวอร์ Apache โดยทำงานคู่กับ mod_cgi ของ Apache ซึ่งเป็นโมดูลย์หลักในการควบคุมการทำงานกับโปรแกรม CGI ภายนอก Apache

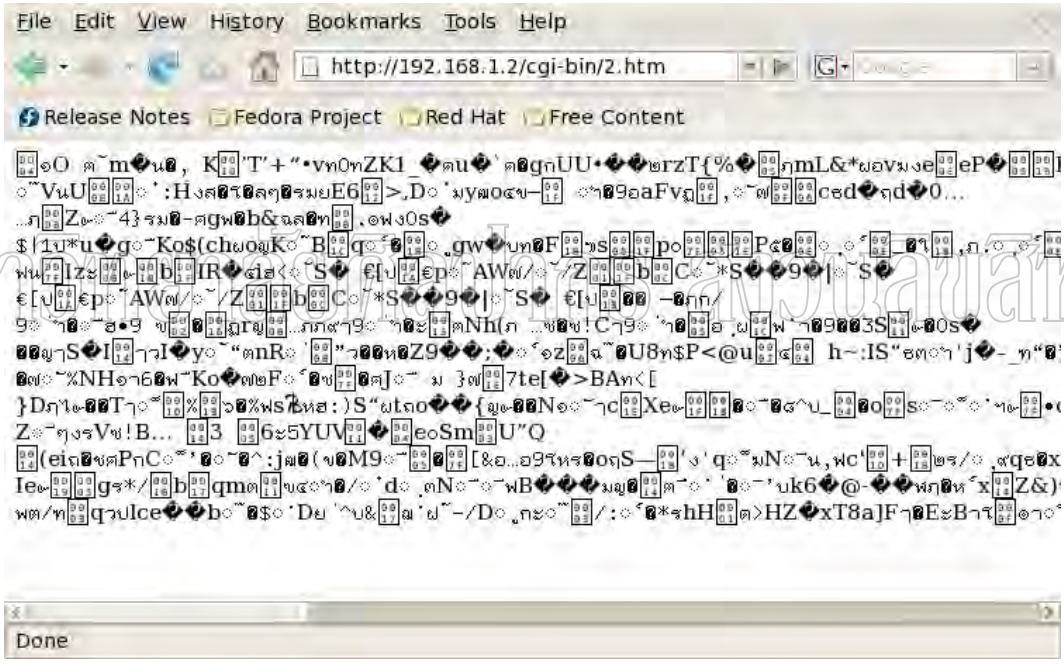
เมื่อมีการร้องขอข้อมูลจากฝั่งเบราเซอร์ Apache จะส่งไฟล์ HTML ที่มีการร้องขอไปยัง CGI ที่ได้พัฒนาไว้ และทำการประมวลผลบีบอัดข้อมูล พร้อมทั้งจับเวลาในการบีบอัด หลังจากนั้นจึงส่งข้อมูลกลับไปยัง Apache เพื่อให้ส่งข้อมูลที่ประมวลผลแล้วกลับไปยังเว็บเบราเซอร์ต่อไป

5.4 ผลการพัฒนาเบราเซอร์จำลอง และสร้างปลั๊กอินบน Firefox

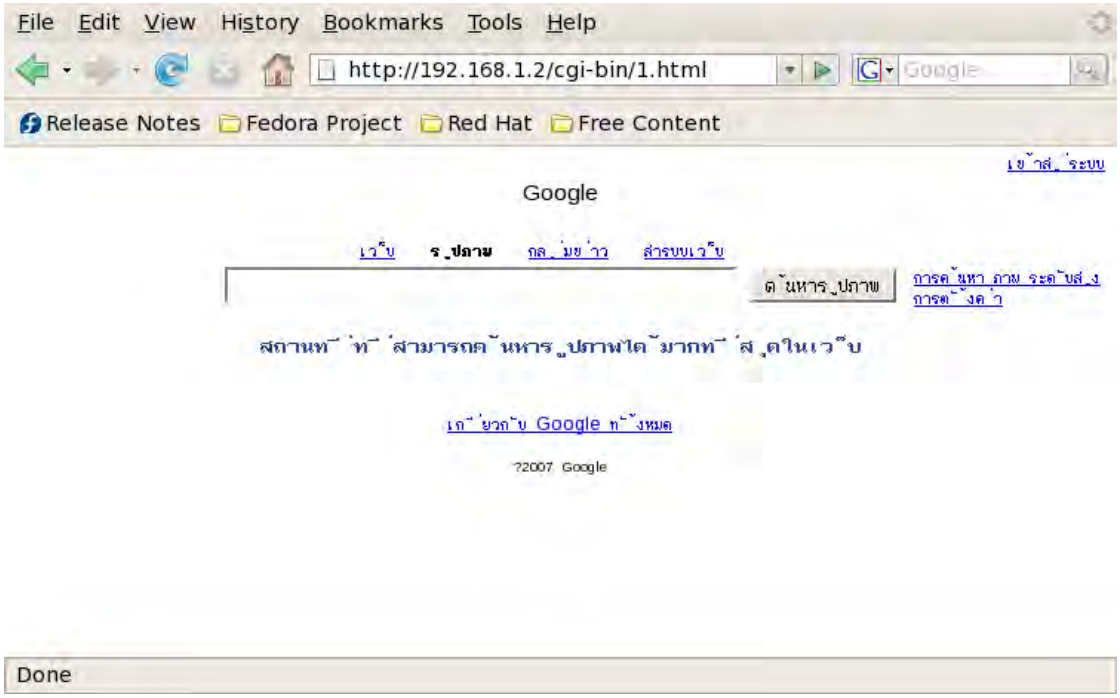
ผู้วิจัยได้เลือกพัฒนาปลั๊กอินบนเบราเซอร์ Firefox ซึ่งเป็นเบราเซอร์ที่มีการพัฒนา Open Source ซึ่งสามารถแก้ไขและพัฒนา Source Code ได้ง่ายและไม่เสียค่าใช้จ่ายในการพัฒนา

ผู้วิจัยได้พัฒนาปลั๊กอินบนเบราเซอร์นี้ที่มีหน้าที่คลายการบีบอัดด้วยอัลกอริทึม Huffman Coding จากเว็บเซิร์ฟเวอร์ Apache โดยมีการสร้าง Source Tree ใหม่คอมไพล์รวมกับ Firefox บนระบบปฏิบัติการ Linux โดยใช้ GCC เป็นตัวคอมไพเลอร์ ทำให้สามารถใช้งาน Firefox ได้อย่างรวดเร็ว และใช้ทรัพยากรของเครื่องคอมพิวเตอร์ที่ติดตั้งเบราเซอร์ตัวนี้น้อยมาก การทำงานของเบราเซอร์จำลองนั้นสามารถทำงานได้ดีไม่เกิดปัญหาในการใช้งาน

โดยผู้วิจัยได้แสดงภาพก่อนและหลังการเพิ่มปลั๊กอินเข้าไปบน Firefox แสดงได้ดังภาพที่ 9 และภาพที่ 10 ตามลำดับดังนี้



ภาพที่ 9 ภาพการแสดงผลของเบราเซอร์ Firefox ก่อนการเพิ่มปลั๊กอิน



ภาพที่ 10 ภาพการแสดงผลของเบราเซอร์ Firefox หลังจากเพิ่มปลั๊กอินเข้าไปแล้ว

จากภาพที่ 9 เป็นภาพเมื่อยังไม่ได้เพิ่มปลั๊กอิน เมื่อมีการร้องขอเว็บเพจที่มีการบีบอัดจากเว็บเซิร์ฟเวอร์ ข้อมูลที่แสดงออกมาจะเป็นอักขระที่อ่านไม่ออกเนื่องจากถูกบีบอัดอยู่ แต่ในภาพที่ 10 เป็นภาพเมื่อเบราเซอร์ติดตั้งปลั๊กอินคลายการบีบอัดแล้ว เมื่อเบราเซอร์สามารถคลายการบีบอัดได้สำเร็จ ก็จะแสดงผลแท็ก และ element ต่างๆ ของ HTML ได้อย่างถูกต้อง

5.5 ผลการทดลองในการทำงานของอัลกอริทึม

ทางผู้วิจัยได้วัดอัตราส่วนการบีบอัดข้อมูล (Compression Ratio) โดยใช้อัตราส่วนจากสมการดังนี้

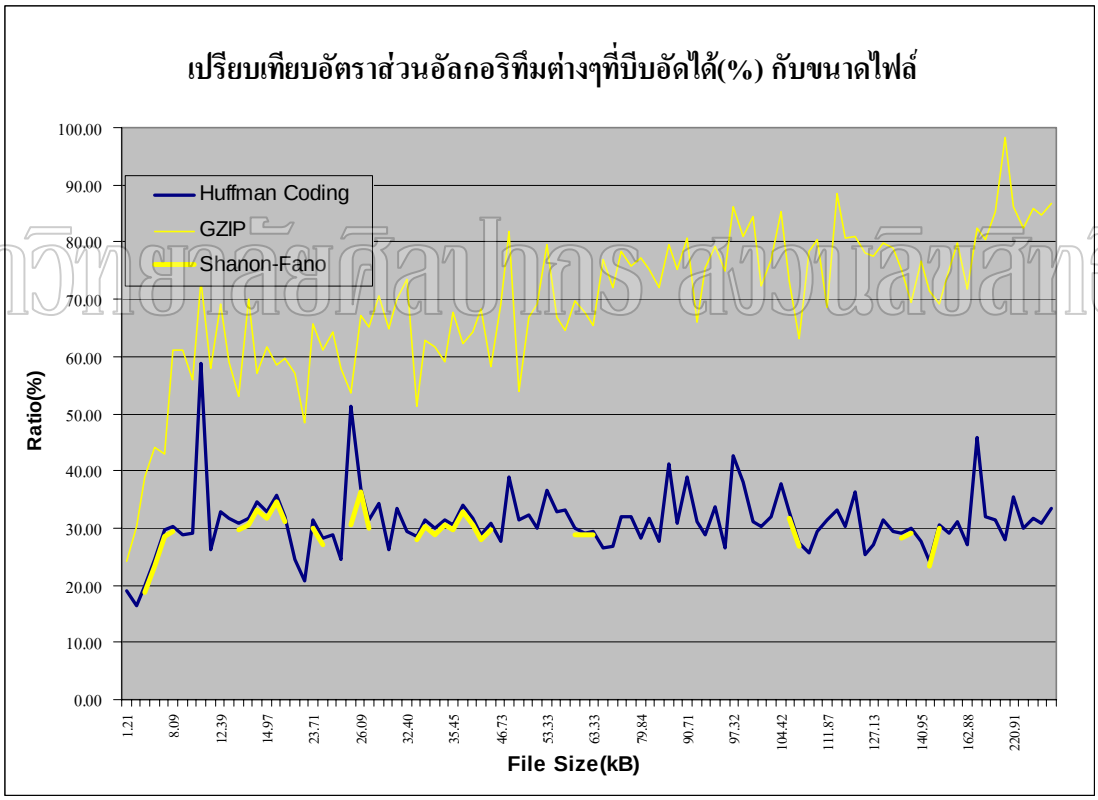
$$\text{สัดส่วนในการบีบอัดข้อมูล} = \frac{\text{ขนาดไฟล์ก่อนบีบอัด} - \text{ขนาดไฟล์หลังบีบอัด}}{\text{ขนาดไฟล์ก่อนบีบอัด}}$$

และทางผู้วิจัยได้จับเวลาการทำงานของอัลกอริทึมทั้ง 2 ชนิดเทียบกันคือ GZIP และ Huffman Coding โดยจับเวลาที่ใช้ในการบีบอัดข้อมูล (Compression Time) และจับเวลาที่ใช้การ

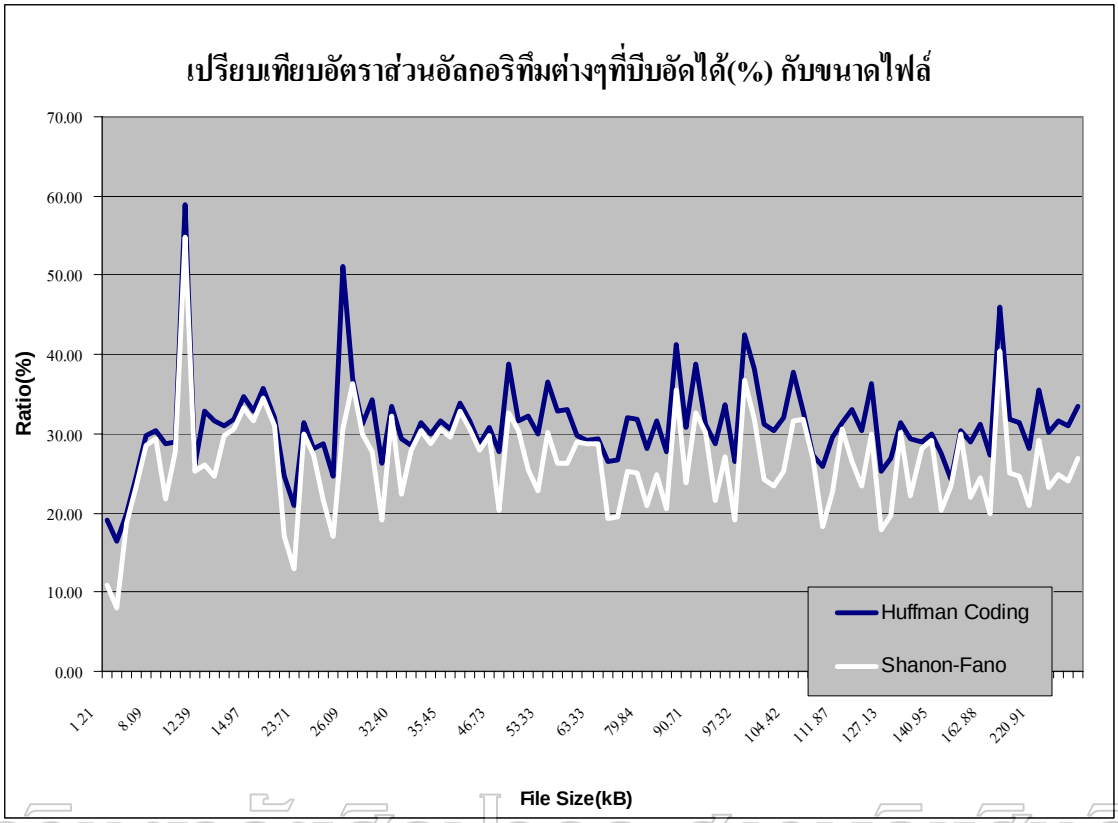
ตอบสนองจากเซิร์ฟเวอร์ (Response Time) คือเป็นเวลาที่เริ่มตั้งแต่เซิร์ฟเวอร์ใช้บีบอัดข้อมูลและส่งข้อมูลมายังเบราเซอร์จนสำเร็จ ซึ่งผลการทดลองออกแสดงในข้อ 5.5.1, 5.5.2 และ 5.5.3

5.5.1 อัตราส่วนการบีบอัดข้อมูล (Compression Ratio)

จากผลการทดลองทั้ง 3 อัลกอริทึม สามารถทำอัตราส่วนการบีบอัด มาทำเป็นกราฟเส้นเทียบระหว่างขนาดไฟล์ และอัตราส่วนที่บีบอัดได้ของทั้ง 3 อัลกอริทึม โดยแกน X แทนขนาดของไฟล์ต่างๆ เป็นกิโลไบต์ และแกน Y แทนอัตราส่วนที่สามารถบีบอัดได้ (Compression Ratio) ซึ่งแสดงในแผนภูมิที่ 1 และแผนภูมิที่ 2 ได้ดังนี้



แผนภูมิที่ 1 แสดงการเปรียบเทียบประสิทธิภาพการบีบอัดไฟล์ HTML ทั้ง 3 อัลกอริทึม



แผนภูมิที่ 2 แสดงการเปรียบเทียบประสิทธิภาพการบีบอัดไฟล์ HTML (ปรับสเกล)

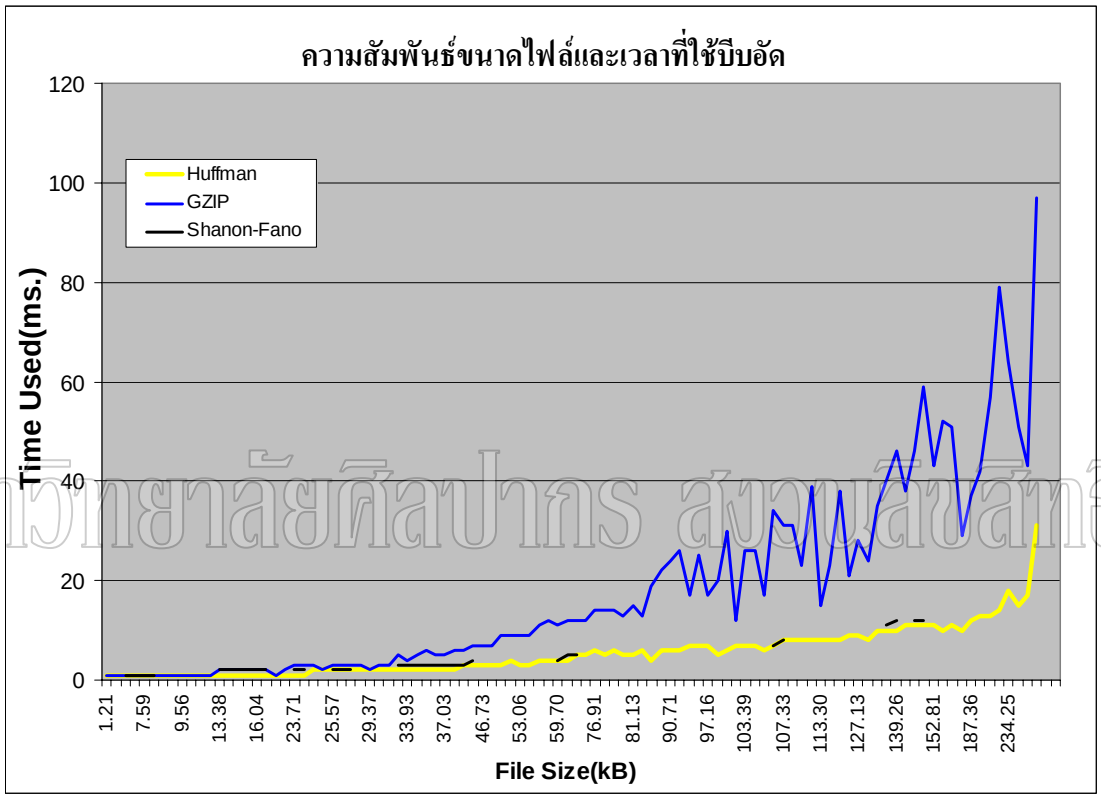
จากแผนภูมิที่ 1 และแผนภูมิที่ 2 จะเห็นว่าขนาดไฟล์นั้นไม่มีผลต่ออัตราส่วนการบีบอัดข้อมูลของอัลกอริทึม Shanon-Fano และ Huffman Coding ส่วน GZIP นั้นเมื่อไฟล์ขนาดใหญ่ขึ้นอัตราส่วนการบีบอัดจะมีแนวโน้มเพิ่มขึ้นเล็กน้อย และ GZIP นั้นมีอัตราการบีบอัดที่สูงกว่า Huffman Coding และ Shannon-Fano มาก เพราะ GZIP ใช้การบีบอัดข้อมูลในลักษณะ Dictionary-Based Compression ซึ่งจะมีอัตราการบีบอัดที่มากกว่า Huffman Coding และ Shannon-Fano มาก เนื่องจากสามารถการบีบอัดแบบ Dictionary-Based นั้นใช้อัลกอริทึม LZ77 เป็นหลักเพื่อหา Symbol ที่ซ้ำๆ กัน ทำให้วิธีนี้จะมีอัตราในการบีบอัดดีกว่าแบบ Entropy Encoding

5.5.2 การจับเวลาการบีบอัดข้อมูล (Compression Time)

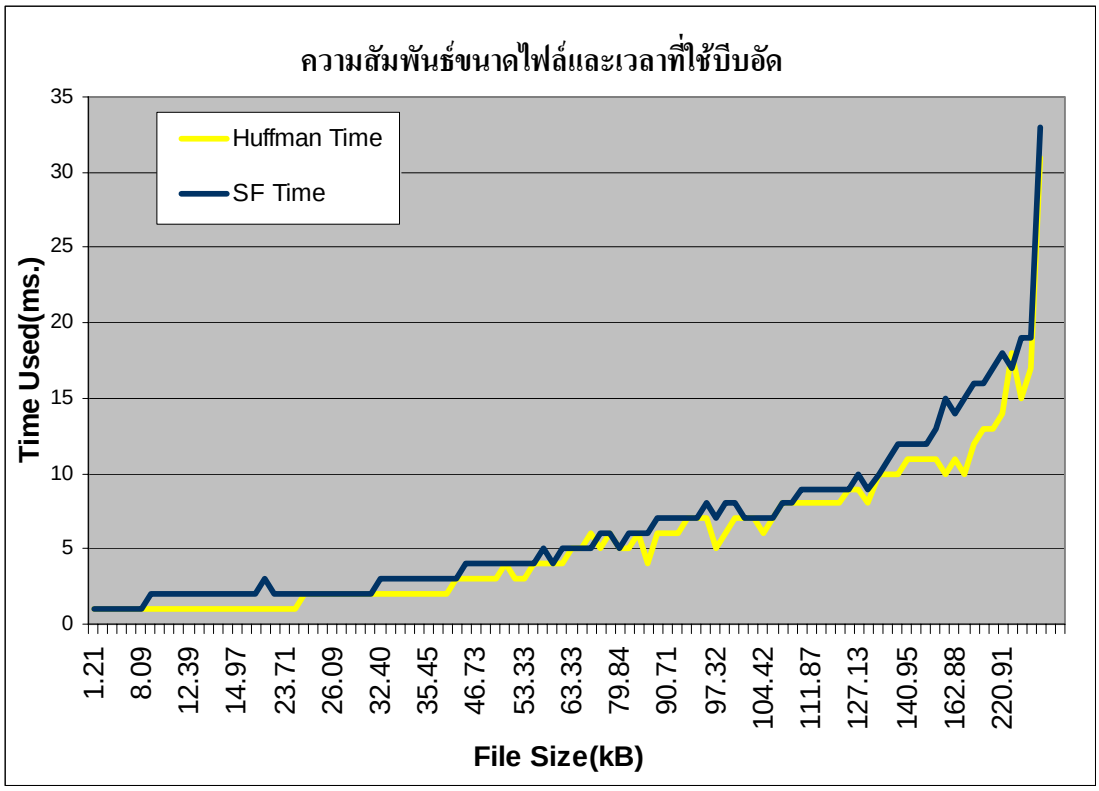
การจับเวลาในการบีบอัดข้อมูลนั้น ผู้วิจัยได้เขียนโปรแกรมขึ้นมาโดยใช้ภาษา GCC บนระบบปฏิบัติการ Linux ในลักษณะ Standalone และนำไปทดลองกับเว็บเพจตัวอย่างต่างๆ โดยการจับเวลานี้จะใช้เวลาเฉลี่ยของแต่ละอัลกอริทึมที่ทดลองได้ (Average Case) ผลการทดลองนั้น

เป็นไปตามสมมุติฐานที่ว่าการบีบอัดแบบ Entropy Encoding นั้นใช้เวลาน้อยกว่าในแบบ Dictionary-Based Encoding

เมื่อนำผลการทดลองมาเขียนเป็นแผนภูมิเส้นแสดงความสัมพันธ์ระหว่างขนาดของไฟล์ที่ถูกบีบอัดด้วยอัลกอริทึมทั้ง 3 ชนิด และเวลาที่ใช้ในการบีบอัดข้อมูล จะแสดงได้ดังแผนภูมิที่ 3 โดยแกน X จะแทนขนาดของไฟล์เป็นกิโลไบต์และแกน Y จะแทนเวลาในหน่วย millisecond



แผนภูมิที่ 3 แสดงความสัมพันธ์ระหว่างขนาดของไฟล์ที่ถูกบีบอัดกับเวลาที่ใช้บีบอัด



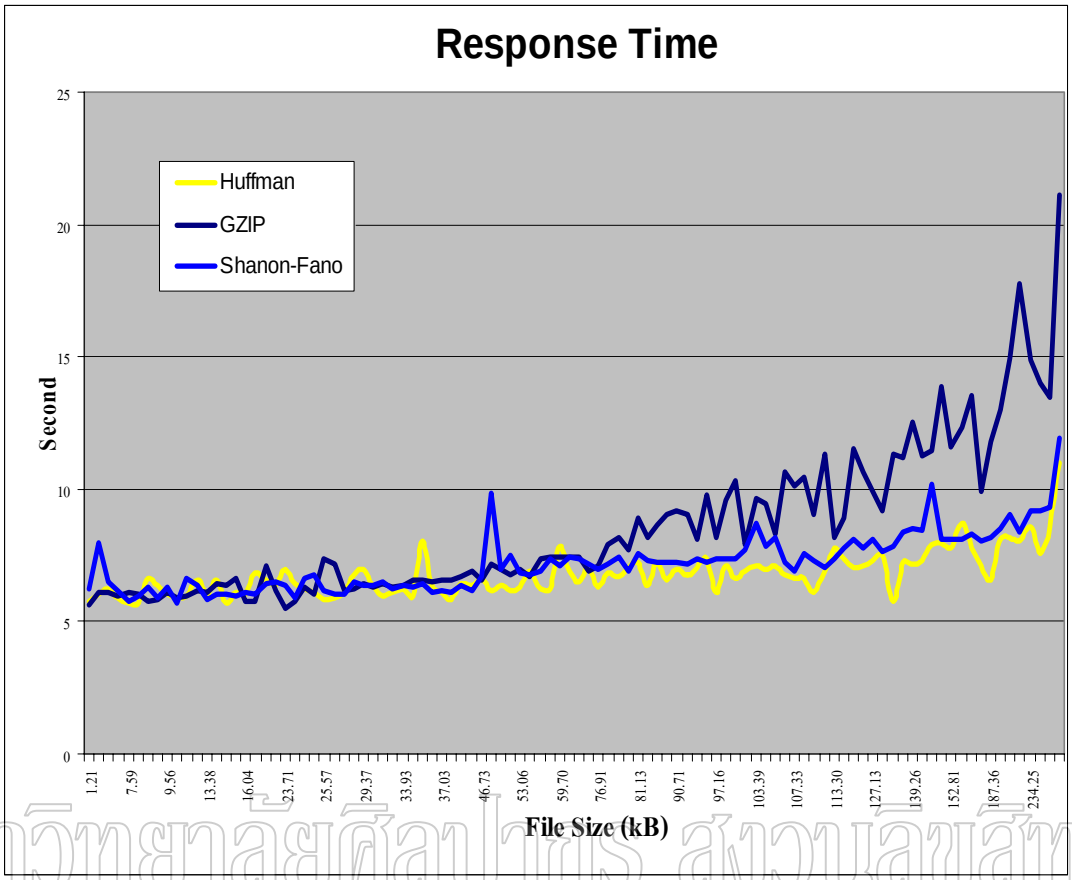
แผนภูมิที่ 4 แสดงความสัมพันธ์โดยปรับสเกลจากแผนภูมิที่ 2

จากแผนภูมิที่ 3 และ 4 จะสรุปได้ว่า เมื่อไฟล์มีขนาดเล็ก (ต่ำกว่า 30kB) การใช้เวลาในการบีบอัดจะใกล้เคียงกันมาก โดยที่การบีบอัดแบบ GZIP จะใช้เวลามากกว่าเล็กน้อย แต่เมื่อบีบอัดไฟล์ขนาดกลางและขนาดใหญ่ การบีบอัดแบบ GZIP นั้นจะใช้เวลามากกว่า Huffman Coding มาก

5.5.3 การจับเวลาตอบสนองจากเซิร์ฟเวอร์ (Response Time)

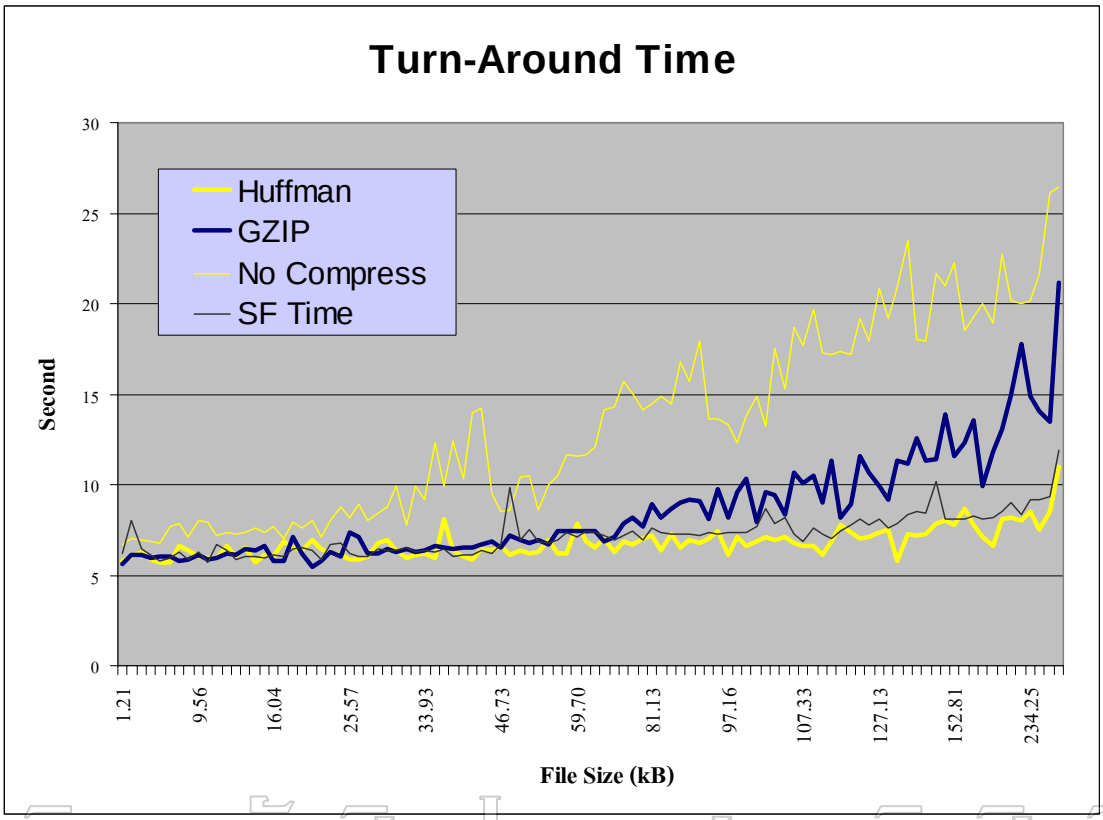
การจับเวลาตอบสนองจากเซิร์ฟเวอร์ ผู้วิจัยได้พัฒนา CGI-Application ฝังบนเว็บเซิร์ฟเวอร์ Apache และใช้เบราเซอร์จำลองซึ่งเขียนจากภาษา Perl บนระบบปฏิบัติการ Linux ทำการร้องขอเว็บเพจจาก Apache โดยเบราเซอร์สามารถจับเวลาที่ใช้ตั้งแต่มีการร้องขอข้อมูลจากเว็บเซิร์ฟเวอร์จนกระทั่งข้อมูลถูกส่งมายังเบราเซอร์จำลอง

เมื่อนำผลการทดลองของทั้ง 3 อัลกอริทึมมาเขียนเป็นแผนภูมิเส้นแสดงความสัมพันธ์ระหว่างขนาดไฟล์และเวลาที่ใช้ จะได้ผลดังแผนภูมิที่ 5



แผนภูมิที่ 5 ผลการเปรียบเทียบ Response Time ของทั้ง 3 อัลกอริทึม

นอกจากนั้นผู้วิจัยยังได้ทดลองกับเว็บเซิร์ฟเวอร์ที่มีการใช้งานจริงๆ ซึ่งมีการร้องขอจากที่อื่นตลอดเวลา โดยได้จับเวลาในการร้องขอแต่ละไฟล์ 100 ครั้งจะได้ผลการทดลองตามแผนภูมิที่ 6 ซึ่งแกน X จะแทนเวลาทั้งหมดที่ใช้หน่วยเป็นวินาที และแกน Y เป็นขนาดของไฟล์



แผนภูมิที่ 6 ผลการเปรียบเทียบ Turn-Around Time ของทั้ง 3 อัลกอริทึมบนเว็บเซิร์ฟเวอร์จริง

จากแผนภูมิที่ 6 จะเห็นได้ว่า ผลการทดลองนั้นสอดคล้องกับผลการทดลองก่อนหน้านี้ (แผนภูมิที่ 4 และ 5) คือการบีบอัดแบบ Dictionary-Based นั้นใช้เวลาเพิ่มขึ้นมากเมื่อขนาดไฟล์เพิ่มขึ้น ทำให้เมื่อมีการรับส่งไฟล์จำนวนหลายๆครั้ง เว็บเซิร์ฟเวอร์จะให้บริการเว็บเพจได้ช้ากว่าการบีบอัดแบบ Entropy Encoding

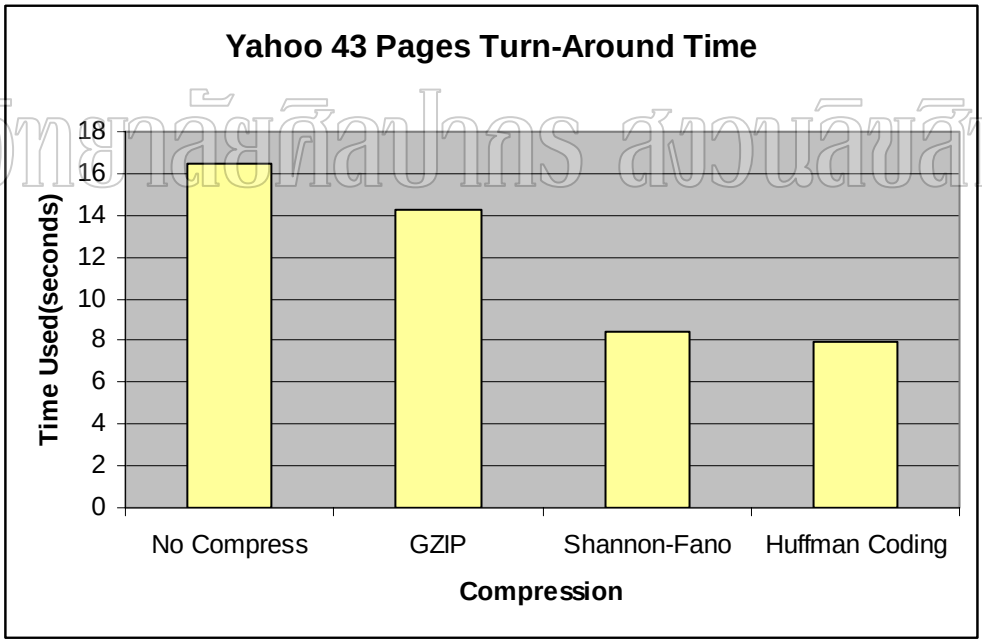
5.5.4 เปรียบเทียบเวลาทั้งหมดของเซิร์ฟเวอร์ที่มีการใช้งานจริง

เมื่อผู้วิจัยได้รวมรวมเว็บเพจทั้งหมดของเว็บไซต์ Yahoo โดยได้เว็บเพจจำนวนทั้งหมด 43 เพจ และนำไปไว้บนอินเตอร์เน็ต และใช้เบราเซอร์ที่เขียนขึ้นจากภาษา Perl ส่งคำร้องขอและจับเวลาจนได้รับข้อมูลครบถ้วน ผลการทดลองจะได้ตามตารางที่ 11

ตารางที่ 11 เปรียบเทียบเวลาในการร้องขอข้อมูลผ่านอินเตอร์เน็ตของทั้ง 3 อัลกอริทึม

การบีบอัด	ครั้งที่ 1 (วินาที)	ครั้งที่ 2 (วินาที)	ครั้งที่ 3 (วินาที)	เวลาเฉลี่ย (วินาที)
ไม่มีการบีบอัด	15.2	17.1	17.2	16.5
GZIP	13.0	14.7	15.1	14.3
Shanon-Fano	8.7	8.7	7.7	8.4
Huffman Coding	7.7	8.0	7.9	7.9

จากตารางที่ 11 ผู้วิจัยจึงได้นำผลการทดลองไปสร้างเป็นแผนภูมิแท่งแสดงการเปรียบเทียบเวลาที่ใช้ในแผนภูมิที่ 7



แผนภูมิที่ 7 เปรียบเทียบ Turn-Around Time ที่ใช้ในการร้องขอ Yahoo จำนวน 43 Page

จากตารางที่ 11 และแผนภูมิที่ 7 จะเห็นได้ว่าเมื่อใช้อัลกอริทึม Huffman Coding ในการบีบอัดเอกสาร HTML จากอินเตอร์เน็ต ผู้วิจัยพบว่า Huffman Coding นั้นสามารถลดเวลาในการเรียกดูเอกสาร HTML ผ่านทางอินเตอร์เน็ตได้ถึง 52.12% และวิธีนี้ใช้เวลาน้อยกว่า GZIP และ Shanon-Fano คิดเป็น 44.75% และ 5.9% ตามลำดับ

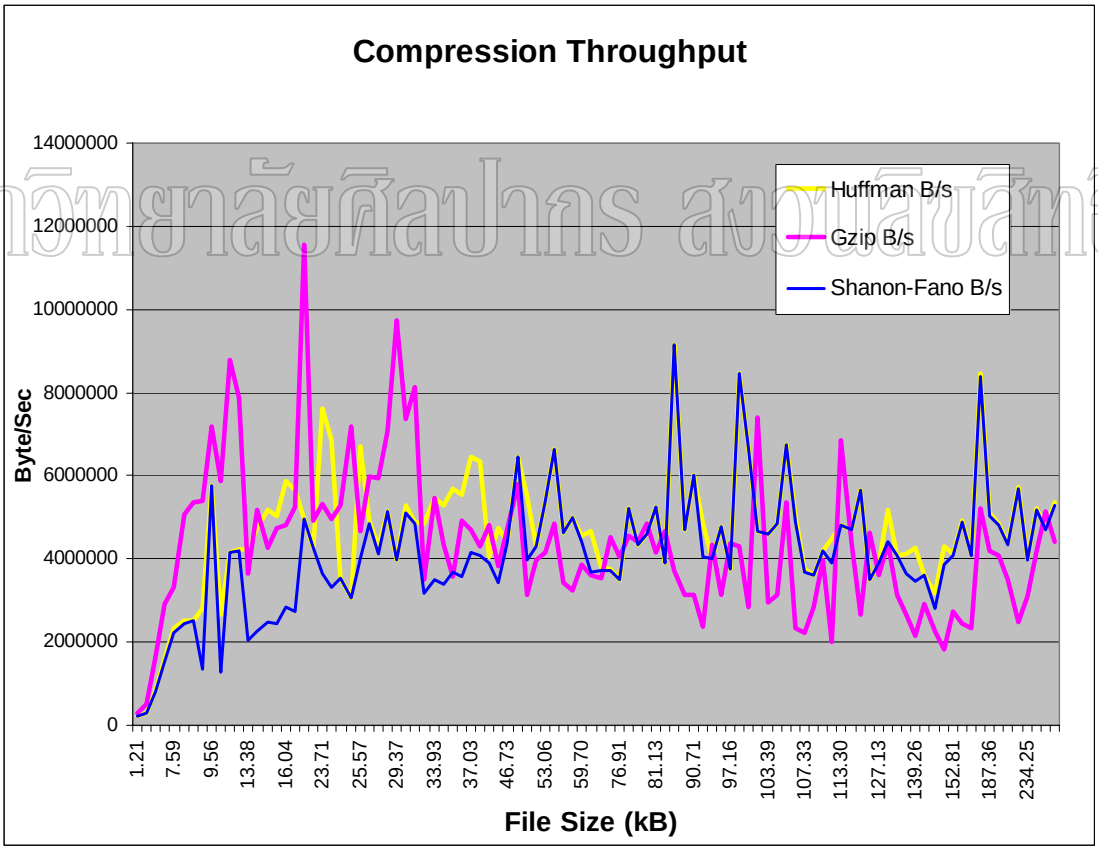
5.6 ประสิทธิภาพในการบีบอัดข้อมูล

5.6.1 ประสิทธิภาพของอัลกอริทึม

ประสิทธิภาพของอัลกอริทึมผู้วิจัยเลือกวัดจาก Throughput ซึ่งได้จากจำนวนไบต์ที่หายไปจากการถูกบีบอัดใน 1 วินาที ซึ่งหาได้ตามสมการต่อไปนี้

$$\text{Throughput(B/s)} = \frac{\text{ขนาดข้อมูลก่อนบีบอัด} - \text{ขนาดข้อมูลหลังบีบอัด}}{\text{เวลาที่ใช้บีบอัด (ms.)}} \times 1000$$

ซึ่งผลการคำนวณเมื่อนำมาทำเป็นแผนภูมิเส้นจะได้ผลดังแผนภูมิที่ 8



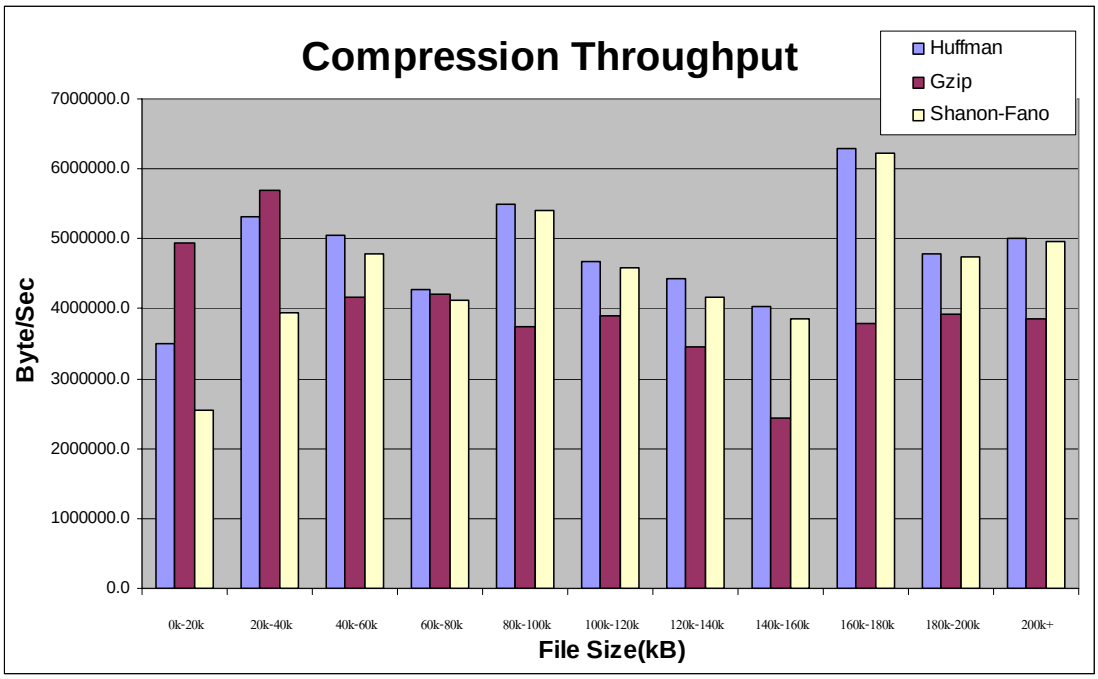
แผนภูมิที่ 8 แสดงการเปรียบเทียบ Throughput ของทั้ง 3 อัลกอริทึม

เนื่องจากผลการทดลองการบีบอัดนั้นมีการกระจายของข้อมูลมาก ผู้ทดลองจึงแบ่งข้อมูลเป็นช่วงชั้นเป็นช่วงชั้นละ 20 กิโลไบต์ และใช้ค่าเฉลี่ยในแต่ละช่วงชั้น ตามตารางที่ 12

ตารางที่ 12 Throughput ในการบีบอัดข้อมูล จำแนกตามช่วงชั้นของข้อมูล

ช่วงชั้น	Huffman (B/s)	GZIP (B/s)	Shanon-Fano (B/s)
0k-20k	3490350.0	4929800.0	2538625.0
20k-40k	5324611.1	5692614.8	3951157.4
40k-60k	5061704.5	4165614.8	4777280.3
60k-80k	4265245.2	4216222.4	4111809.5
80k-100k	5488256.4	3735954.7	5405327.9
100k-120k	4677770.8	3907205.5	4592667.9
120k-140k	4430847.9	3454175.8	4174802.4
140k-160k	4030125.5	2431961.8	3854340.6
160k-180k	6284936.4	3788075.4	6234936.4
180k-200k	4780970.1	3918469.1	4730970.1
200k+	5011905.7	3854621.9	4961905.7

จากตารางที่ 12 ผลที่ได้ออกมาจะเขียนเป็นแผนภูมิแท่งเชิงเปรียบเทียบได้ตามแผนภูมิที่ 9 โดยแกน X แทนขนาดของไฟล์เป็นช่วงชั้นและแกน Y แทน Throughput ของการบีบอัด มีหน่วยเป็นไบต์ต่อวินาที ดังนี้



แผนภูมิที่ 9 เปรียบเทียบ Throughput ของอัลกอริทึมทั้ง 3 เป็นช่วงชั้นละ 20 กิโลไบต์

จากแผนภูมิที่ 9 จะเห็นว่า ไฟล์ขนาดต่ำกว่า 40 กิโลไบต์นั้น การบีบอัดข้อมูลด้วยอัลกอริทึม GZIP จะให้ Throughput ที่ดีกว่าเนื่องจากมีอัตราการบีบอัดที่สูงกว่า Huffman มาก แต่ใช้เวลาในการบีบอัดเกือบจะเท่ากัน แต่เมื่อไฟล์มีขนาดมากกว่า 40 กิโลไบต์นั้นการบีบอัดแบบ Entropy Encoding คือ Huffman Coding จะให้ Thoughput เฉลี่ยที่สูงกว่าการบีบอัดในรูปแบบอื่นๆ เนื่องจากใช้เวลาในการบีบอัดน้อยมาก และอัตราบีบอัดนั้นค่อนข้างคงที่

จากผลการทดลองทั้งหมดที่ผ่านมา สามารถนำผลการทดลองสรุปเป็นตารางที่ 13 เพื่อเปรียบเทียบจุดเด่นจุดด้อยในแต่ละอัลกอริทึม โดยให้หมายเลข 1 คือประสิทธิภาพด้านนั้นดีที่สุด หมายเลข 2 คือมีประสิทธิภาพด้านนั้นระดับปานกลาง เมื่อเปรียบเทียบทั้ง 3 อัลกอริทึม หมายเลข 3 คือประสิทธิภาพต่ำที่สุดเมื่อเปรียบเทียบทั้ง 3 อัลกอริทึม ดังตารางที่ 13

ตารางที่ 13 ตารางเปรียบเทียบจุดเด่นจุดด้อยในแต่ละอัลกอริทึม

การเปรียบเทียบ	Huffman Coding	GZIP	Shannon-Fano
อัตราส่วนการบีบอัด	2	1	3
ความเร็วในการหา symbol	1	3	2
การใช้หน่วยความจำ	2	3	1
อัตราการเพิ่มเวลาในการบีบอัดเมื่อไฟล์มีขนาดใหญ่ขึ้น	3	1	2
ความเร็วในการบีบอัดโดยรวม	1	3	2

จากตารางที่ 13 จะสรุปได้ว่า ความ Huffman Coding นั้น ใช้เวลาในการบีบอัดโดยรวมน้อยที่สุดเมื่อเทียบกับ GZIP และ Shanon-Fano โดย Shanon-Fano ถึงแม้ว่าจะเป็นการบีบอัดในแบบ Entropy Encoding เหมือนกับ Huffman Coding แต่ก็ให้ประสิทธิภาพการทำงานต่ำกว่า นอกจากนี้ Huffman Coding นั้นยังมีข้อได้เปรียบ GZIP ในด้านการใช้หน่วยความจำซึ่งใช้น้อยกว่า แต่อย่างไรก็ดี อัตราส่วนการบีบอัดของ Huffman Coding นั้นน้อยกว่า GZIP แต่เมื่อเทียบกับประสิทธิภาพในการบีบอัดแล้ว ด้าน Huffman Coding โดยรวมแล้วก็ยังสูงกว่า

บทที่ 6
สรุป วิเคราะห์ผลและข้อเสนอแนะ

6.1 สรุปและวิเคราะห์ผลการทดลอง

การบีบอัดเอกสาร HTML แบบ on-the-fly นั้นมีความจำเป็นเนื่องจากในปัจจุบัน เนื้อหาเว็บเพจมักจะเป็นภาษาสคริปต์ที่มีการดึงข้อมูลจากฐานข้อมูล และมีการเปลี่ยนข้อมูลเว็บเพจตลอดเวลา การทำการบีบอัด pre-compression ที่มีการบีบอัดเนื้อหาเว็บเพจทั้งหมดก่อนแล้วจึงให้บริการจึงไม่สามารถใช้งานได้จริงในปัจจุบัน

จากการทดลองนั้นจะเห็นว่าการบีบอัดแบบ Entropy Encoding คือ Huffman Coding และ Shanon-Fano นั้นสามารถบีบอัดข้อมูลได้ในอัตราส่วนที่คงที่คือประมาณ 30% (จากแผนภูมิที่ 1) ไม่ว่าข้อมูลจะขนาดใหญ่เท่าใดก็ตาม แต่ในขณะที่ GZIP นั้นมีอัตราส่วนในการบีบอัดข้อมูลสูงกว่าการบีบอัดในรูปแบบ Entropy Encoding และอัตราส่วนนั้นไม่แน่นอน โดยจะอยู่ในช่วง 60-90% ซึ่งทั้งนี้ขึ้นอยู่กับการเรียงบิตของไฟล์ที่แสดงไว้ในแผนภูมิที่ 1 ในบทที่ 5 แต่สิ่งที่ Entropy Encoding นั้นดีกว่าคือเวลาที่ใช้ในการบีบอัดนั้นใช้น้อยกว่า GZIP มากเมื่อไฟล์มีขนาดใหญ่

จากการทดลองจะเห็นว่าเมื่อไฟล์เอกสารมีขนาดน้อยกว่า 40 กิโลไบต์นั้น เวลาที่ใช้และประสิทธิภาพในการบีบอัดของ GZIP นั้นจะสูงกว่าเนื่องจากอัตราการบีบอัดของ GZIP นั้นมากกว่ากว่าการบีบอัดในรูปแบบ Entropy Encoding มาก แต่ GZIP กลับใช้เวลาในการบีบอัดใกล้เคียงกัน แต่เมื่อขนาดไฟล์ใหญ่เกิน 40kB ประสิทธิภาพของ Entropy Encoding นั้นจะสูงกว่าเสมอเนื่องจากใช้เวลาในการบีบอัดน้อยกว่า GZIP อย่างมาก ซึ่งสามารถดูได้จากกราฟการจำแนกค่าเฉลี่ยประสิทธิภาพในแต่ละช่วงชั้นในแผนภูมิที่ 9 (เปรียบเทียบ Throughput ของอัลกอริทึมทั้ง 3 เป็นช่วงชั้นละ 20 กิโลไบต์) ในบทที่ 5 ทำให้เว็บเซิร์ฟเวอร์การบีบอัดในแบบ Entropy Encoding นั้นสามารถบริการข้อมูลได้รวดเร็วกว่า

เมื่อผู้วิจัยได้ทำการทดลองบนเว็บเซิร์ฟเวอร์จริงผ่านทางเครือข่ายอินเตอร์เน็ต ผลที่ได้นั้นสอดคล้องกับผลการทดลองข้างต้น คือ การใช้ Huffman Coding นั้นทำให้เวลาเฉลี่ยในการให้บริการข้อมูลนั้นน้อยกว่าการบีบอัดด้วยอัลกอริทึมแบบอื่นและแบบไม่บีบอัด

ส่วนอัลกอริทึม Shanon-Fano นั้นเป็นการบีบอัดในแบบ Entropy Encoding เหมือนกับ Huffman Coding และอัตราในการบีบอัดจะใกล้เคียงกับ Huffman Coding มาก แต่ใช้เวลามากกว่า Huffman Coding เล็กน้อย จึงทำให้ประสิทธิภาพของ Shanon-Fano นั้นต่ำกว่า Huffman Coding

ผู้วิจัยจึงสามารถสรุปได้ว่าในการบีบอัดเอกสาร HTML ขนาดเกิน 40kB แบบ on-the-fly ที่เน้นประสิทธิภาพเป็นหลักบนเว็บเซิร์ฟเวอร์นั้น การใช้ Huffman Coding จึงเป็นทางเลือกที่ดีที่สุด เมื่อเปรียบเทียบกับ 2 อัลกอริทึมที่เหลือคือ GZIP และ Shanon-Fano ส่วนอัลกอริทึม GZIP นั้นที่มีประสิทธิภาพในการบีบอัดสูงแต่ใช้เวลามากนั้นเหมาะกับการบีบอัดไฟล์ที่ไม่สนใจด้านเวลา หรือเวลามีความสำคัญน้อย เช่นบีบอัดไฟล์ต่างๆบนเครื่องเดสก์ท็อป เป็นต้น

6.2 งานวิจัยในอนาคต

การบีบอัดเอกสาร HTML โดยวิธี Huffman Coding นี้แสดงให้เห็นว่ามีประสิทธิภาพสูงกว่า GZIP ซึ่งทำให้ข้อมูลถูกส่งไปยังบราวเซอร์ได้เร็วขึ้น ในเครือข่ายที่มีความเร็วสูงมากอาจไม่ค่อยเห็นผลชัดเจนมากนัก แต่ในเครือข่ายความเร็วต่ำอาทิเช่นโทรศัพท์มือถือและPocket Digital Assistant (PDA) หากนำไปประยุกต์ใช้จะเห็นผลชัดเจนยิ่งขึ้น ผู้วิจัยจึงมีความคิดที่จะทดลองนำวิธีการบีบอัดแบบ Huffman Coding ไปประยุกต์ใช้กับเครือข่ายประเภทนี้ต่อไป

แต่อย่างไรก็ดี GZIP นั้นได้ผลดีกับการบีบอัด HTML ขนาดเล็กกว่า 40kB ผู้วิจัยจึงมีแผนในการพัฒนาระบบบีบอัดข้อมูลซึ่งมีความสามารถในการเลือกอัลกอริธึมที่เหมาะสมตามขนาดไฟล์ที่มีการบีบอัดได้เอง และจะพัฒนาให้เป็นโมดูลย์เสริมเชื่อมกับเว็บเซิร์ฟเวอร์ Apache ได้โดยตรงผ่าน Apache Programming Interface (API) เพื่อให้ประสิทธิภาพในการทำงานดีขึ้น รวมทั้งติดตั้งและถอนการติดตั้งได้สะดวกกว่าการทำงานในรูปแบบ CGI

บรรณานุกรม

ภาษาต่างประเทศ

Chung, Kuo-Liang and Jung-Gen Wu. “Level-Compressed Huffman Decoding.” IEEE Transaction on Comunication (1999) : 1455-1457.

Cormen, Thomas H. “Introduction to Algorithms Second Edition.” The MIT Press (2001) : 385-392.

Encyclopedia of Graphics File Formats. PCX File Format Summary [Online]. Accessed 15 December 2007. Available from http://www.fileformat.info/format/pcx/.

Hu ,Yu-Chen. “A new lossless compression scheme based on Huffman coding scheme for image compression.” Chiayi : Signal Processing: Image Communication (2000) : 367-372.

Hu, Yu-Chen and Chang Chin-Chen. “Efficient Huffman Decoding.” Information Processing Letters archive (2001) : 97-99.

Huffman, David A. “A Method for the Construction of Minimum-Redundancy Codes.” Proceedings of the I.R.E (1952) : 1098-1102.

Internet Engineering Task Forge (IETF). DEFLATE Compressed Data Format Specification version 1.3 [Online]. Accessed 15 December 2007. Available from http://www.ietf.org/rfc/rfc1951.txt.

Internet Engineering Task Forge (IETF). RFC 1952: GZIP Compressed Data Format Specification version 1.3 [Online]. Accessed 15 December 2007. Available from http://www.ietf.org/rfc/rfc1952.txt.

Lempel, Abraham and Jacob Ziv. “A Universal Algorithm for Sequential Data Compression.” IEEE Transactions on Information Theory (1977) : 337-343.

Netcraft Ltd. Most Vitited Web Sites[Online]. Accessed 15 December 2007. Available from http://toolbar.netcraft.com/stats/topsites.

OngShell, Ghim Hwee. “A data compression scheme for Chinese text files using Huffman coding and a two-level dictionary.” New York:Information Sciences (1995) : 85-99.

True Hits Com. TrueHits.net Web Statistic[Online]. Accessed 15 December 2007. Available from http://www.truehit.net/award2006/.

W3C. Common Gateway Interface Standard [Online]. Accessed 15 December 2007. Available from http://www.w3.org/CGI/.

ภาคผนวก

ภาคผนวก ก

ตัวอย่างข้อมูลและผลการทดลอง

ตารางที่ 14 แสดงรายชื่อเว็บไซต์ตัวอย่างและขนาดไฟล์ของกลุ่มไฟล์ขนาดเล็ก

ลำดับ	ชื่อเว็บไซต์	URL	ขนาด (kB)
1	TeeNee.com	www.teenee.com	1.21
2	The Krungthep turakij web site	www.bangkokbiznews.com	1.69
3	Google ค้นหารูปภาพ	images.google.com	4.13
4	Google	www.google.com	6.45
5	Live Search	search.live.com	7.59
6	eBay	www.ebay.com	8.09
7	มติชนกรุ๊ป - Matichon Group	www.matichon.co.th	8.55
8	MSN Groups - More Useful Everyday	groups.msn.com	9.41
9	True World-Welcome	www.true.co.th	9.56
10	ลงชื่อเข้าใช้	secure.ebay.com	9.88
11	Asiasoft Corporation :: No.1 Online Game Service Provider	www.asiasoft.co.th	12.39
12	SIAMPHONE.COM : โทรศัพท์มือถือ มือถือ	www.siamphone.com	13.03
13	Yahoo! GeoCities: Get a free web site with easy-to-use site building tools	geocities.yahoo.com	13.38
14	Casale Media	www.casalemedia.com	14.49
15	Microsoft Windows Update	www.microsoft.com/update	14.59
16	Sign In: Sell Your Item	secure.ebay.com	14.97
17	Wells Fargo Sign On	www.wellfargo.com	16.04
18	Welcome to Gmail	www.gmail.com	17.25
19	[Thai|Mail] Thailand's Largest Email Service	www.thaimail.com	19.75
20	More Than Diary, StoryThai	www.storythai.com	19.91

ตารางที่ 15 แสดงรายชื่อเว็บไซต์ตัวอย่างและขนาดไฟล์ของกลุ่มไฟล์ขนาดกลาง

ลำดับ	ชื่อเว็บไซต์	URL	ขนาด (kB)
1	Welcome - PayPal	www.paypal.com	23.71
2	Sign Up - My Yahoo!	signup.yahoo.com	23.85
3	MusicATM : สื่อกลางในการแลกเปลี่ยน และรวบรวมเพลง	www.musicatm.com	24.11
4	www.Uploadtoday.com:บริการรับฝากไฟล์ทุกชนิดฟรี	www.uploadtoday.com	24.18
5	Sign in to Yahoo!	signin.yahoo.com	25.57
6	MapQuest.Com: Maps, Directions and More	www.mapquest.com	26.09
7	Symantec Corp.	www.symantec.com	26.75
8	The Stock Exchange of Thailand	www.set.or.th	29.27
9	Architectural Presentation, Animation and Web Design	www.creationstudio.co.uk	29.37
10	eBay Search: Find Item	search.ebay.com	30.78
11	Thai Board Games Online : Thai Chess, Thai Checkers	www.thaibg.com	32.40
12	DRUDGE REPORT 2007	www.drudgereport.com	33.29
13	My eBay Summary	my.ebay.com	33.93
14	Consumer Electronics Co.	www.ce.org	34.45
15	Hotmail	www.hotmail.com	35.31
16	Microsoft Corporation	www.microsoft.com	35.45
17	HOME - Comcast.net	www.comcast.net	37.03
18	The Internet Movie Database (IMDb)	www.imdb.com	39.17
19	MySpace	www.myspace.com	41.14
20	AOL.com - Welcome to AOL	www.aol.com	45.06
21	Pramool.com Thailand Auction and Classified for Thai	www.pramool.com	46.73

ตารางที่ 15 (ต่อ)

22	Doo-DD : Indy In SOUL	www.doo-dd.com	48.63
23	Google Maps	maps.google.com	51.24
24	Clubic : Informatique et Multimedia	www.clubic.com	52.00
25	My Space Signup	signup.myspace.com	53.06
26	Special Force	www.sf.in.th	53.33
27	Bangkok's Independent Newspaper	www.nationmultimedia.com	54.95
28	Main Page - Wikipedia	www.wikipedia.org	58.88
29	Best Buy	www.bestbuy.com	59.70
30	FOXNews.com	www.foxnews.com	62.38
31	Yahoo! Finance - Get stock quotes	www.yahoo.com	63.33
32	วิชาการ.คอม - คลังความรู้ ปัญญาไทย	www.vcharkarn.com	76.91
33	PostJung.com - ฟังเพลง เกมส์ ดูดวง	www.postjung.com	79.37
34	Thai Educational Portal : ประตูสู่โลกการศึกษา	www.eduzones.com	79.64
35	Jabchai.com [จับฉ่าย ดอท คอม]	www.jabchai.com	79.84
36	Meemodel : Main	www.meemodel.com	81.13
37	THAIWARE.COM	www.thaiware.com	81.98
38	Siam Zone : The Biggest Entertainment Zone	www.siamzone.com	86.57
39	BBC NEWS | News Front Page	www.bbc.co.uk	89.40
40	MCOT : MCOT Public Company Limited	www.mcot.or.th	90.71
41	ซาบซ่าส์ดอทคอม l เกมส์ ดูดวง ฟังเพลง	www.zabza.com	91.36
42	The New York Times - Breaking News	www.nytimes.com	95.66
43	Narak.com : Cute Variety Online	www.narak.com	96.60
44	PantipMarket.com	www.pantipmarket.com	97.16
45	BlogGang.com :: Weblog for You and Your Gang	www.bloggang.com	97.32
46	Gmember	www.gmember.com	102.39
47	Thaitop.net ทางเลือกใหม่ของวัยไซเบอร์	www.thaitop.net	102.79

ตารางที่ 15 (ต่อ)

48	MarketAtHome.com	www.marketathome.com	103.39
49	01net. : Accueil - Toute l'informati	www.01.net	103.86
50	Sanook.com	www.sanook.com	104.42
51	Planet - Planet Internet Homepage	www.planet.nl	104.89
52	CNN.com - Breaking News	www.cnn.com	107.33
53	Yahoo Search	search.yahoo.com	109.49
54	Doctor San	www.doctorsan.com	110.51
55	หมูหิน.คอม	www.moohin.com	111.87
56	Le Monde.fr : A la une	www.lemonde.fr	113.30

ตารางที่ 16 แสดงรายชื่อเว็บไซต์ตัวอย่างและขนาดไฟล์ของกลุ่มไฟล์ขนาดใหญ่

ลำดับ	ชื่อเว็บไซต์	URL	ขนาด (kB)
1	ดูดวงเนื้อคู่ ความรัก	www.meesuk.com	121.42
2	Hunsa	www.hunsa.com	121.77
3	Siam Power.net	www.siampower.net	121.97
4	365jukebox.com	www.siamjukebox.com	127.13
5	รถมือสอง One2car	www.one2car.com	128.77
6	Google Signin	www.google.com/account/	136.13
7	TARAD.com	www.tarad.com	139.15
8	Amazon.com: Online Shopping	www.amazon.com	139.26
9	Mayville-online	www.mayvilleonline.com	140.95
10	Yenta4.com เพื่อนตัวโตบนโลกอินเตอร์เน็ท	www.yenta4.com	141.33
11	Yahoo Search	search.yahoo.com/search	152.00
12	FOX Sports on MSN	www.foxsports.com	152.81
13	กระปุกดอทคอม	www.kapook.com	154.26
14	Channel 7	www.ch7.com	162.88

ตารางที่ 16 (ต่อ)

15	Mtha!.Com	www.mthai.com	179.53
16	SIAMSPORT OFFICIAL WEBSITE	www.siamsport.com	187.36
17	TTTonline	www.tttonline.com	196.16
18	โอ้โหซ่าส์ (Next Step)	www.ohoza.com	198.51
19	Dek-D.com : มหานครวัยรุ่นออนไลน์	www.dek-d.com	220.91
20	SiamZa.Com >> ซาบซ่าทุกอารมณ์	www.siamza.com	234.25
21	Manager Online	www.manager.co.th	242.26
22	Net Guide Thailand, Madoo.com	www.madoo.com	254.68
23	PostJung.com	www.postjung.com	482.36

ตารางที่ 17 เวลาที่ใช้ในการบีบอัดในแต่ละอัลกอริทึมของไฟล์ตัวอย่างขนาดเล็ก

ลำดับ	Huffman Coding (ms.)	GZIP (ms.)	Shanon-Fano (ms.)
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	1	1	1
7	1	1	1
8	1	1	1
9	1	1	2
10	1	1	2
11	1	1	2
12	1	1	2
13	1	2	2
14	1	2	2
15	1	2	2

ตารางที่ 17 (ต่อ)

16	1	2	2
17	1	1	2
18	1	2	2
19	1	2	2
20	2	2	2

ตารางที่ 18 เวลาที่ใช้ในการบีบอัดในแต่ละอัลกอริทึมของไฟล์ตัวอย่างขนาดกลาง

ลำดับ	Huffman Coding (ms.)	GZIP (ms.)	Shanon-Fano (ms.)
1	1	2	2
2	1	2	2
3	2	2	2
4	2	2	2
5	2	1	2
6	2	2	2
7	2	3	2
8	2	3	2
9	2	3	2
10	2	2	2
11	2	3	2
12	2	3	3
13	2	3	3
14	2	3	3
15	2	2	3
16	2	3	3
17	2	3	3
18	2	5	3
19	3	4	3

ตารางที่ 18 (ต่อ)

20	3	5	4
21	3	6	4
22	3	5	4
23	3	5	4
24	4	6	4
25	3	6	4
26	3	7	4
27	4	7	4
28	4	7	4
29	4	9	4
30	4	9	5
31	5	9	5
32	5	9	7
33	6	11	6
34	5	12	5
35	6	11	5
36	5	12	6
37	6	12	6
38	4	12	7
39	6	14	6
40	6	14	6
41	6	14	6
42	7	13	7
43	7	15	7
44	7	13	7
45	5	19	7
46	6	22	7
47	7	24	7

ตารางที่ 18 (ต่อ)

48	7	26	7
49	7	17	7
50	6	25	7
51	7	17	7
52	8	20	7
53	8	30	8
54	8	12	8
55	8	26	8
56	8	26	9

ตารางที่ 19 เวลาที่ใช้ในการบีบอัดในแต่ละอัลกอริทึมของไฟล์ตัวอย่างขนาดใหญ่

ลำดับ	Huffman Coding (ms.)	GZIP (ms.)	Shanon-Fano (ms.)
1	8	17	9
2	8	34	9
3	9	31	9
4	9	31	9
5	8	23	9
6	10	39	9
7	10	15	9
8	10	23	11
9	11	38	12
10	11	21	12
11	11	28	12
12	11	24	12
13	10	35	12
14	11	40	13
15	10	46	13

ตารางที่ 19 (ต่อ)

16	12	38	13
17	13	49	12
18	13	59	14
19	14	43	14
20	18	52	19
21	15	51	16
22	17	29	16
23	31	37	35

ตารางที่ 20 ขนาดไฟล์หลังการบีบอัดในแต่ละอัลกอริทึมของไฟล์ตัวอย่างขนาดเล็ก

ลำดับ	Huffman Coding (B)	GZIP (B)	Shanon-Fano (B)
1	1005	939	1105
2	1444	1205	1588
3	3384	2585	3436
4	4980	3693	5060
5	5457	4440	5557
6	5766	3215	5852
7	6231	3400	6854
8	6842	4247	6962
9	4027	2615	4429
10	7469	4264	7561
11	8527	3918	9379
12	9132	5453	10045
13	9469	6423	9634
14	10112	4455	10291
15	9761	6410	9971
16	10301	5858	10481

ตารางที่ 20 (ต่อ)

17	10567	6818	10763
18	12019	7138	12173
19	15255	8666	16780
20	16140	10522	17754

ตารางที่ 21 ขนาดไฟล์หลังการบีบอัดในแต่ละอัลกอริทึมของไฟล์ตัวอย่างขนาดกลาง

ลำดับ	Huffman Coding (B)	GZIP (B)	Shanon-Fano (B)
1	16675	8314	17013
2	17559	9531	17778
3	17604	8850	19364
4	18673	10396	20540
5	12783	12137	18169
6	16837	8742	17007
7	18826	9520	19164
8	19718	8791	21689
9	22147	10589	24361
10	20959	9417	21343
11	23438	8827	25781
12	24377	16586	24588
13	23807	12903	24209
14	24726	13551	25137
15	24757	14751	25101
16	25207	11728	25541
17	25041	14324	25455
18	27419	14331	27870
19	29991	13363	30387
20	31921	19278	32426

ตารางที่ 21 (ต่อ)

21	34630	14771	38093
22	30460	9097	33506
23	35914	24226	36517
24	36056	17585	39661
25	38102	16785	41912
26	34704	11114	38174
27	37756	18655	41531
28	40365	21347	44401
29	42879	18480	43469
30	45249	20662	45557
31	45850	22482	46229
32	51696	16244	56865
33	57687	22009	63455
34	55232	17567	60755
35	55537	19778	61090
36	58703	18638	64573
37	56837	20544	62520
38	60618	23352	66679
39	52034	18084	57237
40	63364	22569	69700
41	56856	17914	62541
42	64412	31707	65297
43	69820	24004	76802
44	65521	20500	72073
45	73105	24851	80415
46	57284	13653	63012
47	64918	19931	71409
48	72463	16327	79709

ตารางที่ 21 (ต่อ)

49	73750	29186	81125
50	72334	24635	79567
51	66461	15705	73107
52	72091	28660	73278
53	79805	40606	80308
54	83224	24363	91546
55	79748	22121	87722
56	78593	35835	79602

ตารางที่ 22 ขนาดไฟล์หลังการบีบอัดในแต่ละอัลกอริทึมของไฟล์ตัวอย่างขนาดใหญ่

ลำดับ	Huffman Coding (B)	GZIP (B)	Shanon-Fano (B)
1	77573	13377	85330
2	86583	23856	95241
3	79389	23748	87327
4	93375	27468	102712
5	95054	29120	104559
6	90481	26629	92033
7	98537	29343	108390
8	101171	35442	102345
9	99918	43516	101147
10	104577	33531	115034
11	109782	41157	110835
12	108317	47944	109120
13	111124	38625	122236
14	108627	31803	119489
15	121403	46871	133543
16	99403	32321	109343

ตารางที่ 22 (ต่อ)

17	130734	37378	143807
18	137686	29714	151454
19	146218	31608	160839
20	145847	42294	160431
21	167679	34931	184446
22	169724	40096	186696
23	328275	66085	361102

ประวัติผู้วิจัย

ชื่อ-สกุล ว่าที่ร้อยตรีบรรพต ดลวิทยากุล
ที่อยู่ 82 ถ.เกษตรสิน ต.ลำพยา อ.เมือง จ.นครปฐม 73000
ที่ทำงาน ศูนย์คอมพิวเตอร์ มหาวิทยาลัยศิลปากร อ.เมือง จ.นครปฐม

ประวัติการศึกษา

พ.ศ. 2546 สำเร็จการศึกษาปริญญาวิทยาศาสตรบัณฑิต สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร อ.เมือง จ.นครปฐม

พ.ศ. 2548 ศึกษาต่อระดับปริญญามหาบัณฑิต คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

ประวัติการทำงาน

พ.ศ. 2546-ปัจจุบัน ศูนย์คอมพิวเตอร์ มหาวิทยาลัยศิลปากร