การวิเคราะห์ประสิทธิภาพของอัลกอริธึมการแทนที่ข้อมูล MRASM สำหรับเว็บแคชเซิร์ฟเวอร์

PERFORMANCE ANALYSIS OF MODIFIED REPLACEMENT ALGORITHM SELECTION MECHANISM FOR WEB CACHE SERVER

โกศล ถิรจิตโต
KOSON THIRAJITTO

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.2548

ISBN 974-15-1822-6

PERFORMANCE ANALYSIS OF MODIFIED REPLACEMENT ALGORITHM SELECTION MECHANISM FOR WEB CACHE SERVER

KOSON THIRAJITTO

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN COMPUTER ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2005

ISBN 974-15-1822-6

หัวข้อวิทยานิพนธ์	การวิเคราะห์ประสิทธิภาพของอัลกอริธึมการแทนที่ข้อมูล MRASM สำหรับเว็บแคชเซิร์ฟเวอร์
นักศึกษา	นายโกศล ถิรจิตโต
รหัสนักศึกษา	43061614
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
พ.ศ.	2548
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.ประทีป บัญญัตินพรัตน์
อาจารย์ผู้ควบคุมวิทยานิพนธ์ร่วม	ผศ.ดร.ศักดิ์ชัย ทิพย์จักษุรัตน์

บทคัดย่อ

วิทยานิพนธ์นี้นำเสนอการวิเคราะห์อัลกอริธึมการแทนที่ข้อมูล (Replacement Algorithm) สำหรับเว็บแคชเซิร์ฟเวอร์ที่ได้ทำการปรับแต่ง ซึ่งอัลกอริธึมการแทนที่ข้อมูลต่างๆ ที่มีใช้กันอยู่ในปัจจุบันมีความเหมาะสมกับขนาดของข้อมูลแต่ละประเภทที่อยู่ภายในเว็บ และได้มีการพัฒนาวิธีการในการเลือกอัลกอริธึมการแทนที่ข้อมูลมาใช้งานโดยแยกตามขนาดของข้อมูลที่เรียกว่า RASM (Replacement Algorithm Selection Mechanism) สำหรับวิทยานิพนธ์นี้ได้นำเอาวิธีการเลือกใช้อัลกอริธึมการแทนที่ข้อมูลแบบ RASM มาทำการปรับแต่งเรียกว่า MRASM (Modified Replacement Algorithm Selection Mechanism) ซึ่ง MRASM จะยอมให้เว็บแคชเซิร์ฟเวอร์สามารถเลือกแทนที่ข้อมูลที่เหมาะสมสำหรับข้อมูลแต่ละช่วงขนาด โดยข้อมูลใหม่จะแทนที่ข้อมูลภายในแคชที่มีขนาดข้อมูลอยู่ในช่วงขนาดเดียวกัน เพื่อเพิ่มค่าอัตราการพบข้อมูลในแคชของเว็บแคชเซิร์ฟเวอร์เมื่อเทียบกับอัลกอริธึมเดิม

Thesis Title	PERFORMANCE ANALYSIS OF MODIFIED REPLACEMENT ALGORITHM SELECTION MECHANISM FOR WEB CACHE SERVER
Student	Koson Thirajitto
Student ID.	43061614
Degree	Master of Engineering
Programme	Computer Engineering
Year	2005
Thesis Advisor	Assoc.Prof. Pratheep Bunyatnoparat
Thesis Co-Advisor	Asst.Prof.Dr. Sakchai Thipchaksurat

ABSTRACT

This thesis proposes the analysis of replacement algorithm of web cache server. We found that each replacement algorithm is suitable for each type of data size in the web pages. RASM (Replacement Algorithm Selection Mechanism) is a selection mechanism of replacement algorithm that has been developed depending on size of data. This thesis presents the modified version of the RASM called MRASM (Modified Replacement Algorithm Selection Mechanism for web cache server). The MRASM allows web cache server to choose the appropriate replacement algorithms, depending on data size. New data will replace the data in cache that have the same range of data size for increasing Hit Rate.

กิตติกรรมประกาศ

ขอขอบพระคุณในพระคุณอันยิ่งใหญ่ของบิดา และมารดาของข้าพเจ้า ที่ให้การสนับสนุน, ให้โอกาสทางด้านการศึกษา, ให้ข้อคิดในการดำเนินชีวิต,คอยห่วงใย และเป็นแรงใจที่ดีแก่ข้าพเจ้าตลอดมา รวมถึงพี่น้องของข้าพเจ้าที่คอยให้กำลังใจในการทำวิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดี

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยพระคุณของ รศ.ประทีป บัญญัตินพรัตน์ อาจารย์ที่ปรึกษา และ ผศ.ดร.ศักดิ์ชัย ทิพย์จักษุรัตน์ อาจารย์ที่ปรึกษาร่วม ที่ให้การช่วยเหลือ, ให้คำแนะนำปรึกษาที่มีประโยชน์ และชี้แนะแนวทางในการทำวิจัยให้สามารถสำเร็จลุล่วงไปได้ด้วยดี ข้าพเจ้าซาบซึ้งในความอนุเคราะห์ของท่านทั้งสอง และขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณอาจารย์ซึ่งเป็นคณะกรรมการคุมสอบวิทยานิพนธ์ฉบับนี้ ที่ให้คำแนะนำเพิ่มเติมในการจัดทำวิทยานิพนธ์ฉบับนี้ให้สมบูรณ์ และขอขอบพระคุณอาจารย์ทุกท่าน ที่ได้มอบวิชาความรู้ให้แก่ข้าพเจ้า รวมทั้งคำสั่งสอนและอบรมให้ข้าพเจ้าในทางที่ดี

ขอขอบคุณบัณฑิตวิทยาลัยที่ได้ให้การสนับสนุนเงินในการดำเนินงานวิจัย

ขอขอบคุณคุณภาวิน สพโชคด้วยความซาบซึ้งเป็นอย่างสูงที่ให้โอกาสแก่ข้าพเจ้าในการสานต่องานวิจัย, ให้คำแนะนำปรึกษา และข้อมูลแก่ข้าพเจ้า

ขอบคุณพี่ต้องสำหรับคำปรึกษาเกี่ยวกับอัลกอริธึม และแนวคิดในการทำวิทยานิพนธ์ ขอบคุณโป้ง, น้องโป้, พี่ฟุกสำหรับความช่วยเหลือเกี่ยวกับโปรแกรม ขอบคุณน้องปอง, เต๋า,น้องบอล,ต้นสนที่ให้คำปรึกษาด้านภาษาอังกฤษ ขอบคุณพี่อ้อย ,น้องโป้สำหรับเครื่องคอมพิวเตอร์ที่ใช้ในการทดลอง ขอบคุณพี่น้อย, น้องส้ม, คำเพชร,น้ำใส สำหรับการช่วยเหลือ และตรวจสอบบทความทางวิชาการ และรูปแบบการทำวิทยานิพนธ์ฉบับนี้

ขอบคุณ วี,อุ้ย,เก่ง,พี่โอ,ป้าหน่อย, พี่เจี๊ยบ,พี่ไหม และเพื่อนๆพี่ๆน้องๆ นักศึกษาปริญญาโท ภาควิชาวิศวกรรมคอมพิวเตอร์ และภาควิชาโทรคมนาคม สำหรับกำลังใจและเป็นแรงใจให้ข้าพเจ้า

สุดท้ายนี้คุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับผู้มีพระคุณทุกท่าน ข้าพเจ้าหวังเป็นอย่างยิ่งว่าวิทยานิพนธ์ฉบับนี้จะเป็นประโยชน์ต่อผู้ศึกษา

โกศล ถิรจิตโต

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญตาราง (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบันการใช้งานเครือข่ายอินเตอร์เน็ต (Internet) ได้รับความนิยมอย่างแพร่หลาย ทำให้ผู้ใช้งานอินเตอร์เน็ตเพิ่มขึ้นอย่างรวดเร็ว เนื่องจากเครือข่ายอินเตอร์เน็ตเป็นแหล่งข้อมูลขนาดใหญ่ที่สามารถเข้าไปค้นข้อมูลหลากหลายประเภทตามที่ต้องการได้ มีความสะดวกรวดเร็วประหยัดเวลา ข้อมูลในอินเตอร์เน็ตนั้นจะถูกเก็บอยู่ในคอมพิวเตอร์ที่เป็นผู้ให้บริการเรียกว่า เว็บเซิร์ฟเวอร์ (Web Server) เมื่อเราเรียกใช้ข้อมูลเหล่านั้นก็จะถูกส่งจากเว็บเซิร์ฟเวอร์มายังเครื่องคอมพิวเตอร์ที่เราใช้งานอยู่ ความต้องการในการเข้าสืบค้นข้อมูลผ่านทางอินเตอร์เน็ตของผู้ใช้ในองค์กรต่างๆที่มีปริมาณมากขึ้นอย่างรวดเร็วทำให้เกิดการคับคั่งของการจราจรของข้อมูลที่อยู่ภายในเครือข่าย ทำให้เกิดความล่าช้าและเสียเวลาในการรอข้อมูลที่ทำการร้องขอหรืออาจจะไม่สามารถที่จะนำข้อมูลที่ต้องการมาได้ ดังนั้นจึงได้มีการเก็บข้อมูล (Cache) ไว้ภายในเครื่องเซิร์ฟเวอร์ภายในเครือข่ายขององค์กรของแต่ละองค์กร เราเรียกเซิร์ฟเวอร์ประเภทนี้ว่าเว็บแคชเซิร์ฟเวอร์ (Web Cache Server) [1][2] เพื่อลดความซ้ำซ้อนในการการร้องขอข้อมูลจากภายนอกเครือข่ายที่เชื่อมต่อไปยังอินเตอร์เน็ต และเพิ่มความรวดเร็วในการได้รับข้อมูล เว็บแคชเซิร์ฟเวอร์ทำหน้าที่เสมือนตัวแทนในการค้นหาข้อมูลจากภายนอกเครือข่าย โดยเว็บแคชเซิร์ฟเวอร์จะรับการร้องขอข้อมูลจากคอมพิวเตอร์ที่ร้องขอบริการ เราเรียกคอมพิวเตอร์ประเภทนี้ว่า ไคลเอนต์ (Client) ถ้าข้อมูลที่ต้องการไม่มีอยู่ในแคชของเว็บแคชเซิร์ฟเวอร์หรือข้อมูลนั้นเป็นข้อมูลเก่าที่หมดอายุแล้ว เว็บแคชเซิร์ฟเวอร์จะทำหน้าที่ร้องขอข้อมูลมาจากเว็บเซิร์ฟเวอร์ที่อยู่ภายในอินเตอร์เน็ต จากนั้นจะส่งข้อมูลที่ได้รับมาให้กับไคลเอนต์ที่ทำการร้องขอภายในเครือข่ายของตน และจะทำการสำเนาข้อมูลที่ได้รับไว้ในแคชของเว็บแคชเซิร์ฟเวอร์เพื่อให้บริการแก่ไคลเอนต์ที่ร้องขอข้อมูลชุดเดิมสามารถนำข้อมูลไปใช้ได้ทันที ทำให้การเชื่อมต่อไปยังอินเตอร์เน็ตเป็นไปอย่างมีประสิทธิภาพ ช่วยลดปริมาณข้อมูลในเครือข่ายและค่าใช้จ่ายในการใช้งานทรัพยากรที่เชื่อมต่อไปยังอินเตอร์เน็ต

ดังนั้นการทำงานของเว็บแคชเซิร์ฟเวอร์จึงมีส่วนสำคัญต่อการเพิ่มประสิทธิภาพการใช้งานเครือข่ายคอมพิวเตอร์ ซึ่งสะท้อนถึงความสามารถในการให้บริการข้อมูลแก่ผู้ใช้ที่อยู่ภายในเครือข่าย เมื่อข้อมูลที่นำมาเก็บในแคชของเว็บแคชเซิร์ฟเวอร์มีมากขึ้นก็ทำให้แคชมีโอกาสที่จะเต็มหรือถึงค่าที่กำหนดขนาดไว้ เว็บแคชเซิร์ฟเวอร์ก็จะเรียกใช้ฟังก์ชันที่ทำการจัดลำดับความสำคัญของข้อมูล เพื่อนำข้อมูลที่สำคัญน้อยที่สุดออกไปจากแคช เพื่อให้สามารถจัดเก็บข้อมูลใหม่ที่กำลังจะนำเข้ามาได้ ฟังก์ชันในการจัดการบริหารแคชของเว็บแคชเซิร์ฟเวอร์นี้เรียกว่า

อัลกอริธึมการแทนที่ข้อมูล (Replacement Algorithm) [3][4][5] ซึ่งมีอยู่หลายอัลกอริธึมด้วยกัน การเลือกใช้อัลกอริธึมการแทนที่ข้อมูลขึ้นอยู่กับความเหมาะสมและลักษณะพฤติกรรมการใช้งานของผู้ใช้ภายในของเครือข่ายนั้นๆ

เราสามารถเพิ่มประสิทธิภาพของอัลกอริธึมการแทนที่ข้อมูลให้สามารถจัดการ, บริหารข้อมูลภายในแคชของเว็บแคชเซิร์ฟเวอร์ให้มีเปอร์เซ็นต์การพบข้อมูลภายในแคชเพิ่มขึ้น เพื่อลดการร้องขอข้อมูลจากภายนอกเครือข่าย เพื่อลดความคับคั่งของข้อมูล และยังช่วยให้ผู้ใช้งานภายในเครือข่ายสามารถได้รับข้อมูลที่รวดเร็วขึ้นกว่าเดิม

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

วิทยานิพนธ์ฉบับนี้ได้มุ่งเน้นเกี่ยวกับการวิเคราะห์ประสิทธิภาพของอัลกอริธึมการแทนที่ข้อมูล โดยพิจารณาถึงตัววัดประสิทธิภาพ 2 ประเภทคือ ค่าอัตราพบข้อมูลภายในแคชเทียบตามจำนวนครั้ง หรือเรียกว่าค่า Hit Rate [6] และค่าอัตราการพบข้อมูลภายในแคชเทียบตามขนาดข้อมูล หรือเรียกว่าค่า Byte Hit Rate [6] ซึ่งในวิทยานิพนธ์นี้มุ่งเน้นเพื่อวิเคราะห์ประสิทธิภาพโดยพิจารณาจากค่า Hit Rate และค่า Byte Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลที่ทำการปรับแต่งขึ้นเปรียบเทียบกับอัลกอริธึมเดิม โดยพิจารณาตามขนาดของข้อมูล และอัลกอริธึมแต่ละแบบมีความเหมาะสมกับรูปแบบข้อมูลที่แตกต่างกัน ซึ่งในวิทยานิพนธ์นี้ได้พิจารณาปรับแต่งอัลกอริธึมการแทนที่ข้อมูลแบบ RASM (Replacement Algorithm Selection Mechanism) [7] ซึ่งเป็นอัลกอริธึมที่มีกลไกในการพิจารณาเลือกใช้อัลกอริธึมการแทนที่ข้อมูลแบบ GDSF และ LFUDA อีกทีหนึ่งซึ่งเรียกอัลกอริธึมที่ได้ทำการปรับแต่งว่าอัลกอริธึมการแทนที่ข้อมูลแบบ MRASM (Modified Replacement Algorithm Selection Mechanism) [8] โดยมีวัตถุประสงค์ในการทำวิทยานิพนธ์ดังนี้

1) ศึกษาอัลกอริธึมการแทนที่ข้อมูลแบบต่างๆ เพื่อนำอัลกอริธึมที่เลือกใช้มาปรับแต่ง เพื่อเพิ่ม และวิเคราะห์ประสิทธิภาพการทำงานของอัลกอริธึมดังกล่าวในส่วนของค่า Hit Rate และค่า Byte Hit Rate

2) สามารถจำลองผลการทดลองของอัลกอริธึมที่ทำการปรับแต่ง

3) สามารถวิเคราะห์ข้อดี ข้อเสียของอัลกอริธึมที่ได้ทำการปรับแต่งเมื่อเทียบกับอัลกอริธึมเดิม

1.3 ขอบเขตของการศึกษา

ในวิทยานิพนธ์ฉบับนี้ได้ทำการศึกษาถึงการทำงานของอัลกอริธึมการแทนที่ข้อมูลแบบ RASM เพื่อนำมาปรับแต่ง และวิเคราะห์ประสิทธิภาพการทำงาน และจำลองการทำงานของอัลกอริธึมที่ได้ทำการปรับแต่งโดยวัดจากค่า Hit Rate และค่า Byte Hit Rate ดังที่ได้กล่าวไว้แล้วข้างต้น

1.4 ขั้นตอนของการศึกษา

- ทำการค้นคว้าหาข้อมูลเกี่ยวกับอัลกอริธึมการแทนที่ข้อมูลแบบต่างๆ รวมทั้งอัลกอริธึมการแทนที่ข้อมูลแบบ RASM ทั้งในวิทยานิพนธ์ของนักศึกษาปริญญาโทที่สำเร็จการศึกษาแล้ว และบทความจากงานประชุมวิชาการต่างๆ เพื่อเป็นพื้นฐานในการปรับปรุงอัลกอริธึมต่อไป
- นำอัลกอริธึมที่ต้องการทำการปรับแต่งมาศึกษาถึงคุณสมบัติการทำงาน ข้อเด่น ข้อด้อย เพื่อหาจุดที่สามารถปรับแต่งเพิ่มเติมจากเดิมได้
- หาข้อมูลการร้องขอหลายๆรูปแบบจากอินเตอร์เน็ตเพื่อนำมาใช้ทดสอบ
- พัฒนาโปรแกรมจำลองการทำงานของอัลกอริธึมที่ทำการปรับแต่ง และเปรียบเทียบกับอัลกอริธึมเดิม โดยนำข้อมูลการร้องขอที่หามาได้ป้อนเข้าไปในโปรแกรมเพื่อใช้ทดสอบการทำงาน
- วิเคราะห์ข้อมูลการร้องขอหลายๆรูปแบบที่นำมาทดสอบ เพื่อพิจารณาถึงผลที่ได้รับจากการจำลองการทำงาน
- สรุปข้อดี ข้อเสียของอัลกอริธึมการแทนที่ข้อมูลที่ทำการปรับแต่งเมื่อเปรียบเทียบกับอัลกอริธึมการแทนที่ข้อมูลเดิม
- พิจารณาแนวทางในการพัฒนาต่อไปในอนาคต

1.5 รายละเอียดในแต่ละบท

วิทยานิพนธ์ฉบับนี้ได้จัดแบ่งรายละเอียดของเนื้อหาออกเป็น 6 บทดังต่อไปนี้

- บทที่ 1 บทนำ กล่าวถึงความเป็นมาและความสำคัญของปัญหา วัตถุประสงค์และขอบเขตของวิทยานิพนธ์ รวมถึงรายละเอียดของเนื้อหาของแต่ละบท
- บทที่ 2 หลักการทำงานของเว็บแคชเซิร์ฟเวอร์ กล่าวถึงระบบการทำงานของเว็บแคชเซิร์ฟเวอร์ ชนิดของเว็บแคชเซิร์ฟเวอร์ ตลอดจนองค์ประกอบที่เกี่ยวข้องกับการทำงานของเว็บแคชเซิร์ฟเวอร์ โดยอ้างอิงด้วย Squid เว็บแคชเซิร์ฟเวอร์ ซึ่งเป็นเว็บแคชเซิร์ฟเวอร์ที่ได้ทำการศึกษาเป็นแนวทางเพื่อใช้ในวิทยานิพนธ์นี้
- บทที่ 3 อัลกอริธึมการแทนที่ข้อมูลที่เกี่ยวข้องกับวิทยานิพนธ์ กล่าวถึงอัลกอริธึมการแทนที่ข้อมูลแบบต่างๆที่ได้รับการยอมรับ และใช้กันอยู่ในปัจจุบัน

- บทที่ 4 การวิเคราะห์ประสิทธิภาพของอัลกอริธึมการแทนที่ข้อมูล อธิบายถึงการปรับแต่งอัลกอริธึมให้มีประสิทธิภาพเพิ่มขึ้น, วิธีการจำลองการทำงาน ผลการทดลอง และการวิเคราะห์ประสิทธิภาพจากผลการจำลองการทำงาน
- บทที่ 5 การวิเคราะห์หาค่า Threshold ที่เหมาะสม กล่าวถึงวิธีการจำลองการทำงานเพื่อนำไปพิจารณาค่า Threshold ที่เหมาะสม
- บทที่ 6 บทสรุป กล่าวถึงการสรุป วิเคราะห์ผลการทดลอง รวมทั้งข้อเสนอแนะ และแนวทางการทำวิจัยต่อ

บทที่ 2
หลักการทำงานของเว็บแคชเซิร์ฟเวอร์

บทนี้จะกล่าวถึงรายละเอียดต่างๆ เกี่ยวกับเว็บแคชเซิร์ฟเวอร์ (Web Cache Server) [9] เริ่มจากหลักการทำงานทั่วไป ส่วนประกอบต่างๆ ประเภทของเว็บแคชเซิร์ฟเวอร์ เว็บแคชเซิร์ฟเวอร์ที่ได้ศึกษา พารามิเตอร์ที่สำคัญในการปรับตั้งการทำงาน และรูปแบบของล็อกไฟล์ที่เก็บอยู่ในเว็บแคชเซิร์ฟเวอร์ เพื่อให้เข้าใจถึงการทำงานเบื้องต้นของเว็บแคชเซิร์ฟเวอร์

2.1 ความหมายและหลักการทำงานของเว็บแคชเซิร์ฟเวอร์

เว็บแคชเซิร์ฟเวอร์ หรืออาจเรียกอีกชื่อว่าเว็บพร็อกซี่เซิร์ฟเวอร์ (Web Proxy Server) [1] คือเครื่องเซิร์ฟเวอร์ที่ติดตั้งภายในเครือข่าย อยู่ระหว่างไคลเอนต์กับเว็บเซิร์ฟเวอร์ภายนอกเครือข่าย ทำหน้าที่ให้บริการแก่ไคลเอนต์ภายในเครือข่าย โดยการเป็นตัวแทนไปดึงข้อมูลที่ไคลเอนต์ต้องการจากอินเตอร์เน็ตตามที่ได้รับการร้องขอ แล้วส่งกลับมาให้กับไคลเอนต์เหล่านั้น และทำการจัดเก็บข้อมูลเหล่านั้นไว้ในหน่วยความจำชั่วระยะเวลาหนึ่ง เราเรียกการทำงานนี้ว่า การแคชข้อมูล (Caching) เมื่อไคลเอนต์ต้องการร้องขอข้อมูลจากภายนอกเครือข่าย จะติดต่อไปยังเว็บแคชเซิร์ฟเวอร์ก่อนเพื่อตรวจสอบดูว่ามีการสำเนาข้อมูลเก็บไว้ในเว็บแคชเซิร์ฟเวอร์หรือไม่ ถ้ามีก็จะส่งข้อมูลนั้นๆ ที่ได้จัดเก็บไว้ในแคชให้กับไคลเอนต์ที่ทำการร้องขอได้เลย โดยไม่ต้องไปดึงมาจากภายนอกอีก แต่ถ้าเว็บแคชเซิร์ฟเวอร์ไม่มีข้อมูลเหล่านั้น เว็บแคชเซิร์ฟเวอร์ก็จะทำหน้าที่ติดต่อไปยังเว็บเซิร์ฟเวอร์ต้นทางเพื่อขอข้อมูลเหล่านั้นนำกลับมาให้กับไคลเอนต์อีกทีหนึ่ง ประโยชน์ของการนำเว็บแคชเซิร์ฟเวอร์มาใช้งานคือ

- Reduce Latency คือช่วยลดระยะเวลาระหว่างการเรียกข้อมูลจนถึงเริ่มส่งข้อมูล เพราะการร้องขอข้อมูลจากแคชของเว็บแคชเซิร์ฟเวอร์ที่อยู่ภายในเครือข่ายจะใช้น้อยกว่าการร้องขอไปยังเครื่องเว็บเซิร์ฟเวอร์ต้นทางที่อยู่ภายนอกเครือข่าย
- Reduce Traffic คือช่วยลดความคับคั่งของระบบเครือข่ายที่เชื่อมต่อไปยังอินเตอร์เน็ต เพราะมีการร้องขอข้อมูลจากเว็บเซิร์ฟเวอร์ต้นทางเพียงครั้งเดียว ไม่จำเป็นต้องร้องขอข้อมูลจากภายนอกซ้ำหลายๆครั้งถ้าไคลเอนต์หลายๆเครื่องต้องการข้อมูลอันเดียวกัน เป็นการช่วยลดการสูญเสียแบนด์วิดธ์ (bandwidth)

2.2 ชนิดของเว็บแคชชิ่ง (Web Caching)

ชนิดของเว็บแคชชิ่ง สามารถแบ่งได้เป็น 2 ประเภทคือ

1. Browser Caches จะพบได้ตามบราวเซอร์(Browser) ไม่ว่าจะเป็น Internet Explorer หรือ Netscape Communicator แคชประเภทนี้จะทำการเก็บข้อมูลเว็บที่เคยเข้าไปดูมาแล้ว และเมื่อมีการกดปุ่ม "Back" เพื่อจะกลับไปยังหน้าต่างก่อนหน้านั้น มันจะดึงข้อมูลจาก Browser Caches แสดงขึ้นมาอย่างรวดเร็ว
2. Proxy Caches เป็น Proxies ที่ให้บริการแก่ผู้ใช้จำนวนมากในเส้นทางเดียวกัน และเพื่อลดจำนวนของ internet bandwidth แคชประเภทนี้ทำการเก็บข้อมูลที่มีการเรียกใช้งานบ่อยๆ มาไว้ที่ proxy caches เพื่อช่วยให้การใช้งานเร็วขึ้น

นอกจากนี้ยังมีคำที่มีความหมายใกล้เคียงหรือสัมพันธ์กับเว็บแคชซิ่ง อยู่อีก 2 คำด้วยกัน คือ Firewall และ Gateway ในปัจจุบันซอฟแวร์ส่วนใหญ่จึงได้นำความสามารถในการทำงานของเซิร์ฟเวอร์เหล่านี้มาผสมผสานกัน ทำให้มีความสามารถและมีความยืดหยุ่นในการทำงานมากขึ้น ในองค์กรขนาดเล็กจึงอาจใช้เว็บแคชซิ่งซึ่งเป็นส่วนหนึ่งของ Proxy Firewall Server โดยให้บริการกับเครื่องคอมพิวเตอร์ภายในองค์กร

2.3 ประเภทและความสัมพันธ์ของเว็บแคชเซิร์ฟเวอร์

เราสามารถแบ่งประเภทของเว็บแคชเซิร์ฟเวอร์ตามลักษณะการจัดวางรูปแบบของแคชตามโทโปโลยี (Topology) ได้เป็นสองประเภท คือ Simple Web-Cache และ Cooperate Web-Cache

2.3.1 การทำงานของ Simple Web-Cache

Simple Web-cache หมายถึงการเชื่อมต่อเว็บแคชเซิร์ฟเวอร์ที่มีการทำงานต่อกันเป็นลำดับชั้น (Hierarchy) เมื่อไม่พบข้อมูลที่แคชของตัวเองก็จะทำการร้องขอข้อมูลดังกล่าวกับเว็บแคชเซิร์ฟเวอร์เครื่องอื่นที่อยู่ในระดับสูงขึ้นไป

Web Browser ส่วนมากจะมีวิธีการติดต่อผ่านเครือข่ายอย่างง่ายๆ โดยระบุ URL ซึ่งประกอบไปด้วยชื่อโฮสต์ (Host) และชื่อไฟล์ที่ต้องการ ซึ่งอยู่บนโฮสต์นั้น มันจะสร้างการติดต่อแบบ TCP ไปยังโฮสต์ที่ระบุนั้นและนำไฟล์ที่ต้องการนั้นมาแสดงผล แต่ถ้าไม่พบ ผู้ใช้จะได้รับ Error Message แทน เนื่องจากบราวเซอร์มีความอิสระในการเข้าถึงข้อมูลที่ตนเองต้องการ จึงอาจมีข้อมูลเดียวกันหลายชุดซ้ำๆกันอยู่ในเครือข่าย ซึ่งเป็นการสูญเสียโดยไม่จำเป็น ดังเช่นเว็บไซต์ (Web Site) ที่เป็นที่นิยมทั้งหลายจะมีผู้เข้าชมในวันหนึ่งเป็นจำนวนมาก ซึ่งอาจจะผ่านเส้นทางเดียวกันในเครือข่ายสื่อสารหลายครั้ง

เว็บแคชจะช่วยลดการสูญเสียในกรณีเช่นนี้ได้ โดยการสกัดกั้นการร้องขอข้อมูลที่ซ้ำๆ กัน เว็บแคชจะสร้างการเชื่อมต่อไปยังโฮสต์ที่อยู่ระยะไกลเพียงครั้งเดียว แล้วนำข้อมูลที่ต้องการมาทำการสำเนาแล้วส่งต่อให้กับบราวเซอร์ที่ร้องขอมาอีกทีหนึ่ง การทำเช่นนี้จะประสบผลสำเร็จก็ต่อเมื่อ

เว็บแคชต้องอยู่ใกล้ชิดกับผู้ใช้งานมากที่สุดเท่าที่เป็นไปได้ และต้องมีผู้ใช้งานจำนวนมากพอที่จะทำให้เกิดการร้องขอข้อมูลที่มีความซ้ำกัน

การติดต่อระหว่างบราวเซอร์บราวเซอร์กับเว็บแคชเหมือนกับการติดต่อระหว่าง บราวเซอร์กับเว็บไซต์ ต่างกันที่การร้องขอไปยังเว็บแคชต้องส่ง URL ไป ไม่ใช่แค่เฉพาะชื่อของไฟล์ข้อมูลนั้น เมื่อเว็บแคชได้รับการร้องขอก็จะนำ URL นั้นเทียบกับการร้องขออื่นๆ ที่ผ่านมา หากเป็น URL ที่เคยมีการดึงข้อมูลมาจากเว็บไซต์ต้นทาง และได้มีการทำสำเนาเก็บไว้แล้ว ก็จะนำส่งให้กับผู้ร้องขอทันที มิเช่นนั้นแล้วเว็บแคชจะส่งการร้องขอต่อไปยังโฮสต์ที่ระบุใน URL หรือ Parent Cache (ซึ่งจะกล่าวถึงในภายหลัง) แล้วนำข้อมูลส่งต่อให้กับบราวเซอร์อีกทีหนึ่ง ถ้าข้อมูลชุดนี้ตรงตามกฎของแคช มันจะมีการสำเนาไว้ทันที เผื่อมีการร้องขอมาอีกในอนาคต การร้องขอข้อมูลของ Simple Web-Cache ผ่านการติดต่อแบบ TCP เช่นเดียวกับบราวเซอร์ ถ้าการติดต่อนั้นไม่สำเร็จก็จะส่งข้อความที่ระบุถึงความผิดพลาดกลับมายังผู้ใช้งาน

แคชสามารถส่งการร้องขอไปยังแคชอื่นๆ ได้ ซึ่งเป็นโครงสร้างแบบ Hierarchy แบ่งได้เป็นหลายระดับ ให้บริการโดยอ้อมกับผู้ใช้งานจำนวนมากมาย แต่อย่างไรก็ตามมีการกำหนดว่าแต่ละไคลเอนต์ (ไม่ว่าจะเป็นบราวเซอร์ หรือแคชเองก็ตาม) สามารถมี Parent Cache ได้จำกัดเพียงหนึ่งเท่านั้น โครงสร้าง Hierarchy จึงมีลักษณะเหมือน Simple Tree โดยมีบราวเซอร์อยู่ที่จุดล่างสุดต่อจาก Local Cache และการร้องขอข้อมูลต้องทำผ่านขึ้นไปยังจุดสูงสุด และต่อไปยังโฮสต์ที่เป็นต้นฉบับ อย่างไรก็ตามบราวเซอร์ก็สามารถติดต่อโดยตรงกับ Cache ที่ระดับบนๆ ได้โดยไม่จำเป็นต้องผ่าน Local Cache

ปัญหาของ Simple Caching

Cache Hierarchy สามารถลดความคับคั่งของเครือข่ายได้ แต่จะเป็นปัญหาเรื่องการเก็บข้อมูลในดิสก์ (Disk) ที่ซ้ำๆ กันในแต่ละแคช เพราะเมื่อมีการร้องขอผ่านในแต่ละชั้นของ Cache Hierarchy ข้อมูลที่ได้จะถูกเก็บไว้ในแต่ละแคชที่ข้อมูลถูกส่งผ่าน ปัญหาที่สำคัญของ Cache Hierarchy อีกปัญหาหนึ่ง คือแคชในระดับเดียวกันไม่สามารถติดต่อถึงกันได้ ดังนั้นการร้องขอจะต้องร้องขอไปยัง Parent Cache ซึ่งเป็นการเพิ่มปริมาณข้อมูลภายในเครือข่าย อาจก่อให้เกิดความคับคั่งภายในเครือข่าย

เนื่องจากแคชมีเนื้อที่สำหรับเก็บข้อมูลจำกัด ดังนั้นจึงจำเป็นต้องลบข้อมูลเก่าทิ้งอยู่เสมอ เพื่อให้มีเนื้อที่เหลือว่างพอสำหรับการร้องขอข้อมูลใหม่ โดยส่วนใหญ่ใช้หลักการลบข้อมูลที่มีการร้องขอมาน้อยที่สุดที่เรียกว่า Least Recently Uses (LRU) และในอนาคตมีโอกาสใช้มันน้อย ถ้าเรามีเนื้อที่ดิสก์ไม่เพียงพอ จะทำให้แคชต้องลบข้อมูลออกจำนวนมาก หากมีการร้องขอมาใหม่แคชจะต้องสร้างการเชื่อมต่อ และนำข้อมูลมาอีกครั้งโดยไม่จำเป็น จำนวนเนื้อที่ดิสก์ที่ต้องการขึ้นอยู่กับจำนวนผู้ใช้งานที่เว็บแคชรองรับ ในทางอุดมคติแคชควรจะมีเนื้อที่เพียงพอสำหรับเก็บ

ข้อมูลทุกๆ ข้อมูลที่ผู้ใช้ร้องขอมามากกว่า 1 ครั้งตลอดช่วงอายุของข้อมูล แต่ในความเป็นจริงแล้ว เราควรเก็บข้อมูลไว้ทั้งหมดเพราะเราไม่สามารถจะทราบได้ว่าในอนาคตข้อมูลใดจะถูกอ่านอีกหรือไม่ ความสัมพันธ์ระหว่างจำนวนผู้ใช้งานและเนื้อที่เก็บข้อมูลแสดงให้เห็นว่า แคชในระดับบนของ Hierarchies มีความต้องการเนื้อที่ดิสก์เก็บข้อมูลมากกว่าแคชในระดับล่าง

สำหรับการทำงานของเว็บแคชเซิร์ฟเวอร์เครื่องหนึ่ง สามารถที่จัดการข้อมูล และรองรับการร้องขอจากบราวเซอร์ที่ขอใช้บริการอย่างเสมอภาค แต่ถ้ามองในแต่ละแคชจะต้องมีเนื้อที่ดิสก์เพียงพอสำหรับเก็บข้อมูลที่ผู้ใช้งานร้องขอทั้งหมด ถ้าไม่มีข้อมูลหรือข้อมูลได้ถูกลบไปแล้ว การร้องขอจะต้องทำต่อไปยัง Parent Cache หรือโฮสต์ต้นทางเท่านั้น ถึงแม้ว่าข้อมูลนั้นจะมีอยู่ในแคชเครื่องอื่นที่อยู่ในระดับเดียวกันก็ตาม

2.3.2 การทำงานของ Cooperate Web-Cache

Cooperate Web-Cache เป็นการทำงานประสานร่วมกันระหว่างบราวเซอร์ และแคช หรือแม้แต่แคชด้วยกันเอง ซึ่งสามารถแบ่งกระจายความรับผิดชอบในการให้ตอบสนองต่อข้อมูลที่ถูกร้องขอจากเซิร์ฟเวอร์ต้นทางที่อยู่ภายใต้กลุ่มของโดเมนเนม (Domain Name) ที่กำหนด เพื่อเป็นการลดความซ้ำซ้อนในการทำลำเนาข้อมูลในแคชของเว็บแคชเซิร์ฟเวอร์

ความฉลาดของไคลเอนต์

บราวเซอร์ หรือเว็บแคชโดยส่วนใหญ่จะมีรายการติดต่อที่แน่นอนอยู่แล้วว่าควรจะติดต่อโดยตรงกับเว็บแคชเครื่องใด ซึ่งการติดต่อโดยตรงจะรวดเร็ว และไม่เสียเนื้อที่เก็บข้อมูลในเว็บแคช การเลือกใช้แคชโดยพิจารณาจาก URL จะช่วยลดการซ้ำซ้อนของข้อมูลในเว็บแคชแต่ละตัว ถ้าแคชมีปัญหาและหยุดรับการร้องขอ ไคลเอนต์จะพยายามค้นหาเส้นทางอื่นที่จะส่งการร้องขอไปถึงโฮสต์ต้นทาง อย่างไรก็ตามบราวเซอร์ส่วนใหญ่ไม่มีลักษณะยืดหยุ่นเช่นนี้ เพราะเว็บแคชจะกำหนดให้ทำหน้าที่เป็นไฟร์วอลล์ (Firewall) ป้องกันการติดต่อโดยตรง ใน Auto-configuration script ยอมให้มีการกำหนดรายการเว็บแคชได้หลายเครื่องตามลำดับ โดยกำหนดเป็นชื่อ IP Address ก็ได้ และหากเว็บแคชทั้งหมดไม่มีการตอบรับก็สามารถติดต่อโดยตรง กลไกนี้กลายเป็นมาตรฐานของ Internet Application อย่างไรก็ตามบราวเซอร์โดยส่วนใหญ่ก็จะเลือกแคชในลำดับต้นๆ บนรายการ DNS เท่านั้น ซึ่ง DNS Server นี้จะรวมเว็บแคชเซิร์ฟเวอร์ไว้เป็นกลุ่มตามลำดับ

การติดต่อกับเว็บแคชเพื่อตรวจสอบว่าเว็บแคชยังคงให้บริการอยู่ จะใช้การติดต่อแบบ TCP เพื่อร้องขอข้อมูล ถ้าการติดต่อไปยังแคชเครื่องแรกถูกปฏิเสธ บราวเซอร์จะพยายามติดต่อกับแคชเครื่องถัดไปในลิสต์ ส่วนทางเลือกอื่น การติดต่อกับแคชในครั้งแรกอาจใช้แพ็กเก็ตแบบ UDP และสร้างการติดต่อแบบ TCP เมื่อมีการตอบรับแพ็กเก็ตแบบ UDP อาจถูกส่งไปยัง Port พิเศษ ซึ่งกำหนดไว้โดย Cache Software หรือส่งไปยัง Echo Port ซึ่งเป็นส่วนหนึ่งของมาตรฐาน TCP/IP การใช้ Echo Port จะเป็นการตรวจสอบอย่างง่ายๆว่า Server ยังคงอยู่แต่จะไม่บ่งบอกแน่

ชัดว่า Cache Software ยังทำงานอยู่ การติดต่อแบบนี้ดูจะเป็นวิธีการที่ด้อยกว่าวิธีการติดต่อแบบ TCP เสียอีกเพราะเป็นการเพิ่ม Packet การติดต่อในกรณีที่แคชยังคงทำงานอยู่อีกด้วย อย่างไรก็ตามถ้าหากแคชหลายๆเครื่องยังคงทำงานอยู่ การติดต่อด้วย Packet UDP ในตอนเริ่มต้นสามารถใช้ในกาหาว่าเครื่องใดให้การตอบรับเร็วที่สุด โดย Packet จะถูกส่งออกไปพร้อมๆกันและจะวัดเวลาในการตอบสนองแคช (Round-trip Time) เครื่องใดมีการตอบสนองเร็วที่สุดจะถูกเลือกให้รับการร้องขอข้อมูลโดยใช้กรติดต่อแบบ TCP เวลาในการตอบสนองของแคชจะขึ้นอยู่กับความเร็วและภาระงานแต่ละแคช รวมไปถึงผลจากระยะทางและความคับคั่งของเครือข่ายด้วย

การส่ง Packet แบบ UDP ก่อนการสร้างการติดต่อแบบ TCP ดูเหมือนจะเป็นการสูญเสียแบนด์วิดธ์ แต่ในความเป็นจริงแล้วการแลกเปลี่ยนข้อมูลแบบ UDP ต้องการเพียง 2 Packet ในแต่ละครั้งของการติดต่อเท่านั้น ซึ่งถ้าเทียบกับการแลกเปลี่ยนข้อมูลแบบ TCP ต้องใช้ถึง 8 Packet เป็นอย่างน้อย เราสามารถเพิ่มข้อมูลการร้องขอลงไปใน UDP Packet โดยไม่เป็นการเพิ่มความคับคั่งของเครือข่าย ส่วนการตอบรับด้วย Packet UDP จากแคชไม่เพียงแต่ระบุสถานะและความเร็วเท่านั้น แต่ยังสามารถบรรจุข้อมูลที่ไคลเอนต์ต้องการได้ ถ้ามีข้อมูลนั้นอยู่ในแคช และข้อมูลนั้นเล็กพอที่จะบรรจุอยู่ใน UDP Packet เพียง Packet เดียว ดังนั้นจึงสามารถลดขั้นตอนการสร้าง TCP Connection ได้ทั้งหมด

โดยสรุปการใช้ความชาญฉลาดของไคลเอนต์สามารถแบ่งได้เป็น 5 ระดับดังตารางที่ 2.1

ตารางที่ 2.1 สรุปการใช้ความชาญฉลาดของไคลเอนต์

0	Simple	ใช้แคชเพียงเครื่องเดียว แต่สามารถติดต่อโดยตรงได้ในบางรูปแบบของ URL
1	Selection	เลือกใช้แคชได้ หรือติดต่อโดยตรงโดยดูจากรูปแบบของ URL เป็นหลัก
2	Resilience	มีการพยายามติดต่อกับแคชเครื่องอื่นๆเมื่อตรวจสอบแล้วพบว่าแคชเครื่องแรกไม่ทำงาน
3	Load Balancing	เลือกใช้แคชโดยดูจากความเร็วในการตอบสนอง
4	Discovery	เลือกใช้แคชซึ่งมีข้อมูลที่ไคลเอนต์ต้องการอยู่ด้วย

ลักษณะการทำงานของ Cooperate Web-Cache

ใน Harvest Project ที่มหาวิทยาลัย Colorado สหรัฐอเมริกา ได้ทำการพัฒนา Internet Cache Protocol (ICP) เพื่อจุดมุ่งหมายในการทำ Load Balancing และ Discovery ซึ่งปัจจุบัน ICP ได้กลายเป็นมาตรฐานของอินเตอร์เน็ตสำหรับติดต่อระหว่างแคช ICP ใช้การส่ง Packet

UDP เพื่อการร้องขอข้อมูลเริ่มต้นเท่านั้น แต่ถ้าต้องการส่งข้อมูล Hyper text Transfer Protocol (HTTP) จะใช้การส่งแบบ TCP จะใช้ ธรรมดา การติดต่อระหว่าง ICP Cache

แต่ละแคชจะมีรายการของแคชเครื่องอื่นๆ ซึ่งรูปแบบอาจกำหนดให้เลือกแต่ละแคช โดยดูจาก URL เป็นหลัก เมื่อได้รับการร้องขอข้อมูล ซึ่งตนเองไม่มีข้อมูลนั้นอยู่ มันจะส่ง ICP Packet ไปยังเครื่องที่เหมาะสมพร้อมๆกัน แล้วรอการตอบรับกลับเพื่อป้องกันกรณีที่ Packet มีการสูญหาย หรือแคชที่ส่งไปนั้นไม่ทำงาน จะมีการกำหนดเวลาที่จะต้องรอไว้ ถ้าแคชได้รับการตอบรับว่า 'hit' จะหมายความว่ามีข้อมูลที่ร้องขอไป ดังนั้นมันจะดึงข้อมูลนั้นโดยการสร้าง HTTP Connection ไปยังเซิร์ฟเวอร์ที่ตอบรับ 'hit' ที่เร็วที่สุด แต่ถ้าการตอบรับมีเพียงแต่ 'miss' เท่านั้น ไคลเอนต์จะพิจารณาเลือกแคชที่ถูกกำหนดไว้ให้เป็น 'parent' เท่านั้น (ไม่ใช่ 'neighbor') โดยจะสร้าง HTTP Connection ไปยัง Parent Cache ที่มีการตอบรับ 'miss' ที่เร็วที่สุด แม้ในขณะนี้ไคลเอนต์จะรู้อยู่แล้วว่า Parent Cache จะต้องส่งต่อการร้องขอต่อไป แต่อย่างน้อยก็เป็นวิธีการที่ดีที่สุดแล้ว ICP ถือว่าเวลาที่ตรวจจับการตอบรับต้องมีความสัมพันธ์กัน แต่ถ้ามีเหตุผลในทางเทคนิคที่ต้องการให้เลือกใช้แคชบางเครื่อง ถึงแม้ว่าจะมีเวลาในการตอบรับช้าก็ตาม สามารถนำ Weighting Factor มาประยุกต์ใช้ได้

Hierarchy ของ Parent และ Neighbor ที่นำมาใช้ใน ICP มีความเป็นอิสระมากกว่าการเป็นโครงสร้างแบบ Rigid Tree แต่จะเป็นไปตามลักษณะเฉพาะของแต่ละแคช ในช่วงการติดต่อเพื่อค้นหาข้อมูลจะไม่มีความแตกต่างระหว่างแคชทั้ง 2 ชนิด จะต่างกันก็เพียงแต่ว่า การค้นหาที่ไม่มีอยู่ในแคชใดๆเลย ไคลเอนต์จะคิดว่า Parent ของมันจะอยู่ใกล้กับแหล่งที่มาของข้อมูลมากกว่า เราสามารถกำหนดให้แคชทำสำเนาข้อมูลซึ่งส่งมาจากแคชเครื่องอื่นๆหรือเพียงแต่ส่งมันต่อไปยังไคลเอนต์ของมันก็ได้ ซึ่งกรณีการส่งต่อจะเป็นการลดการใช้เนื้อที่ดิสก์ในแต่ละแคช แต่จะเป็นการเพิ่มความคับคั่งของเครือข่ายระหว่างแคชด้วยกัน Cache Software โดยส่วนใหญ่สามารถเลือกได้โดยอิสระสำหรับแต่ละ Neighbor หรือ Parent ว่าจะให้ทำการสำเนาข้อมูลหรือเพียงแต่ส่งต่อเท่านั้น ดังนั้นจึงสามารถปรับความสมดุลระหว่างการใช้เนื้อที่ดิสก์ และความคับคั่งของเครือข่ายได้

จะพบว่าไคลเอนต์ที่ทำการร้องขอในแบบ ICP มักจะเป็นแคชด้วยกัน ซึ่งไม่มีเหตุผลที่แน่นอนว่าทำไมจึงไม่ใช้ Protocol นี้ติดต่อระหว่างบราวเซอร์ด้วยกัน อาจเป็นเพราะความยากในการจัดการบราวเซอร์ที่ต่างยี่ห้อและต่างรุ่นกันจำนวนมากมาย และโดยทั่วไปแล้วมี Local Cache เพียงหนึ่งเครื่องที่คอยรับข้อมูลจากภายนอกอยู่แล้ว ดังนั้นจึงไม่จำเป็นต้องมีความสามารถเต็มที่เช่นเดียวกับ Cooperate Web Cache

การทำงานร่วมกับ Simple Web Cache

จากส่วนที่ผ่านมาได้อธิบายถึงการทำงานระหว่าง Cooperate Cache ด้วยกันเท่านั้น แต่ในทางปฏิบัติอาจต้องกำหนดให้ทำงานร่วมกับ Simple Cache ดังนั้นจึงต้องนำหลักการทำงานมาพิจารณาร่วมกัน Simple Cache สามารถอ้างโดย ICP Client ได้ แต่โดยปรกติจะถูกกำหนดให้เป็น Parent เท่านั้น เพราะว่า Simple Client ไม่สามารถรองรับการติดต่อแบบ ICP จะทำได้ก็เพียงการติดต่อไปยัง TCP/IP Echo Port เท่านั้น ถ้า Simple Cache ยังคงทำงานอยู่ก็จะมี Packet ตอบกลับมาเช่นกัน ซึ่งจะถูกแปลความหมายเช่นเดียวกับ 'miss' เสมอ ดังนั้นหากกำหนดให้ Simple Cache เป็น 'neighbor' ก็จะไม่มีการดึงข้อมูลมาจากเครื่องนั้น และถ้าไม่มีการตอบรับ 'hit' จากเครื่องอื่นๆ อาจเป็นไปได้ว่า Simple Cache จะถูกเลือกให้เป็น Parent Cache ที่ตอบรับ 'miss' ที่เร็วที่สุด ถ้า ICP Client มีทั้ง Simple Cache และ Cooperate Cache ที่เป็น Parent จะมีความเป็นไปได้สูงที่ Simple Cache จะถูกเลือกก่อนเสมอ เพราะมันมีงานที่จะต้องทำน้อยกว่าก่อนที่จะตอบรับเพื่อป้องกันปัญหานี้ อาจมีการนำระบบ weighting มาใช้ ให้การเลือกเอนเอียงไปทาง Cooperate Cache มากกว่า แม้ว่า Simple Cache จะมีตอบรับก่อนก็ตาม ในทางกลับกัน หากมีการร้องขอข้อมูลจาก Simple Cache ไป Cooperate Cache การทำงานจะเป็นไปตามปรกติ ซึ่งจะคล้ายกับการที่บราวเซอร์ติดต่อไปยัง Cooperate Cache

ผลกระทบที่เกิดขึ้น

การทำงานของ Cooperate Cache จะไม่มีผลกระทบต่อระบบเครือข่าย เมื่อข้อมูลถูกพบในแคชของบราวเซอร์ก่อน หรือคิดว่า ICP Timeout มีค่าน้อยมากเมื่อเทียบกับเวลาที่ใช้ในการดึงข้อมูลมาจากเครื่องต้นฉบับ ซึ่งข้อมูลไม่ได้มีการเก็บไว้ในแคชเครื่องใดๆที่อยู่ระหว่างบราวเซอร์ถึงเครื่องต้นฉบับ การใช้ Cooperate Web Cache ที่ระดับในระดับหนึ่งของ Hierarchy จะเป็นการเพิ่มจำนวนของแคชที่บราวเซอร์จะต้องติดต่อในทางอ้อม ดังนั้นจึงเป็นการเพิ่มโอกาสในการค้นพบข้อมูลมากขึ้นสำหรับข้อมูลซึ่งมีขนาดไม่เล็กพอที่จะบรรจุอยู่ใน Packet แบบ ICP ที่ตอบรับกลับ จำนวนของความคับคั่งในเครือข่ายจะเพิ่มขึ้นเล็กน้อยจากการใช้ Packet พิเศษซึ่งใช้ UDP Protocol แต่สำหรับข้อมูลขนาดเล็กเป็นไปได้ที่ความคับคั่งของเครือข่ายอาจจะลดลง เนื่องจากไม่มีจำเป็นต้องสร้าง TCP Connection ในการส่งข้อมูล

Cooperate Cache ซึ่งต่ออยู่ในเครือข่ายความเร็วสูงสามารถที่จะกำหนดให้มีการแบ่งปันเนื้อที่ดิสก์ซึ่งเป็นการลดความต้องการในการใช้ดิสก์ขนาดใหญ่ในแต่ละเครื่อง แต่จะเป็นการเพิ่มความคับคั่งของเครือข่ายระหว่างแคช ความสามารถในการทำงานของ Cooperate Cache มีความยืดหยุ่นและสามารถแบ่งเบาภาระได้ ซึ่งจะเป็นผลดีกับผู้ใช้งาน

ความถูกต้องของข้อมูลในแคช

เมื่อไรก็ตามที่มีการสำเนาข้อมูลต้นฉบับก็จะมีปัญหาในการจัดการเกิดขึ้น ถ้าข้อมูลต้นฉบับมีการเปลี่ยนแปลงหลังจากที่ได้มีการทำสำเนาเก็บไว้ มันจะกลายเป็นข้อมูลที่พ้นสมัย (out of date) ในทันที เรียกว่าปัญหา Consistency ความล้าสมัยของสำเนาข้อมูลถูกกำหนดเป็นช่วงเวลาตั้งแต่ข้อมูลต้นฉบับเปลี่ยนไป และมักจะแปรผันไปตามสัดส่วนของข้อมูลมันเป็นไปได้ยากที่จะติดต่อกับแหล่งข้อมูลว่าสำเนาข้อมูลนี้ล้าสมัยแล้วหรือยัง ถ้าให้ระบบเครือข่ายและโฮสต์ที่ให้บริการไม่มีการเปลี่ยนแปลงใดๆ เนื่องจาก Hypertext Transfer Protocol ในรุ่นก่อนๆมีเพียงแต่การร้องขอแบบ 'GET' เท่านั้น ซึ่งจะได้รับข้อมูลมาทั้งหมด เหมือนกับขบวนการร้องขอข้อมูลแล้วนำเปรียบเทียบกับสำเนา

การปรับปรุงสิ่งแรกก็คือเพิ่มการร้องขอแบบ 'HEAD' ซึ่งจะมีการส่งข้อมูลสรุปเกี่ยวกับข้อมูลที่ร้องขอ เช่นเวลาที่เปลี่ยนแปลงครั้งหลังสุด ถ้าข้อมูลนั้นแสดงให้เห็นว่าสำเนามีความล้าสมัย มันจะต้องสร้าง Connection ไปยังเครื่องต้นทางเพิ่มขึ้นอีกเพื่อร้องขอข้อมูลแบบ 'GET' ข้อด้อยนี้ทำให้มีการนำการร้องขอในแบบมีเงื่อนไขมาใช้ ซึ่งเรียกกันว่า 'If-Modified-Since GET' หรือ 'IMS GET' การร้องขอแบบนี้จะมีการนำเวลาของสำเนาข้อมูลมาใส่ไว้ในการร้องขอด้วย แล้วคาดหวังว่าการตอบกลับจะเป็นเพียง 'unchanged' หรือการส่งข้อมูลมาใหม่ ดังนั้นทั้งการตรวจสอบและการแก้ไขข้อมูลสามารถทำได้ในการติดต่อเพียง Connection เดียวเท่านั้น ปัจจุบันเว็บเซิร์ฟเวอร์โดยส่วนใหญ่สนับสนุนการร้องขอแบบ 'IMS GET' แต่การตรวจสอบแบบนี้ยังคงต้องมีการติดต่อไปยังโฮสต์ต้นทาง ดังนั้นอาจจะต้องใช้เวลาบ้างสำหรับ site ที่อยู่ห่างไกล แต่มันจะช่วยลดปริมาณความต้องการใช้ระบบเครือข่ายเมื่อข้อมูลไม่มีการเปลี่ยนแปลง

แต่อย่างไรก็ตามมันยังไม่เป็นที่น่าพอใจ โดยเฉพาะข้อมูลขนาดเล็กที่อยู่ระยะไกล การสร้างการเชื่อมต่อไปยังเครื่องต้นทาง อาจใช้เวลามากกว่าการส่งข้อมูลเสียอีก ถึงแม้ว่าจะมีการตรวจสอบความล้าสมัยด้วยการร้องขอแบบมีเงื่อนไข (IMS GET) มันก็ยังใช้เวลาเกือบจะเท่ากับการที่ไคลเอนต์ร้องขอข้อมูลจากเครื่องต้นทางโดยตรง จากกรณีเช่นนี้จุดประสงค์ของการใช้แคชจะสูญเสียไป ดังนั้นทางที่เป็นไปได้ผู้ใช้งานอาจต้องมีการยอมรับข้อมูลที่มีความล้าสมัยอยู่บ้างเพื่อความรวดเร็วในการตอบสนองในทันทีทันใดจากแคช แต่ถ้าผู้ใช้งานรู้สึกว่าข้อมูลมีความล้าสมัยก็ยังสามารถที่จะบังคับให้แคชทำการดึงข้อมูลจากเครื่องต้นทางมาใหม่ได้เช่นกัน ถ้าแคชไม่ได้มีการตรวจสอบความล้าสมัยในทุกๆการร้องขอข้อมูล จะต้องมีวิธีการที่จะตัดสินว่าเมื่อไรจะต้องมีการตรวจสอบ วิธีที่ง่ายที่สุดคือตรวจสอบเมื่อช่วงระยะเวลาหนึ่งผ่านไป (ในทางปฏิบัติจะตรวจในครั้งแรกที่มีการร้องขอข้อมูลหลังจากได้หลังจากได้ผ่านช่วงเวลาที่กำหนด) ดังนั้นจึงเป็นการยอมให้ผู้บริหารเว็บแคชสามารถกำหนดความล้าสมัยมากที่สุดเป็นเวลาที่แน่นอน ตัวอย่างเช่นกำหนดให้แคชต้องไม่นำข้อมูลที่มีอายุเวลามากเกินกว่า 12 ชั่วโมงมาใช้

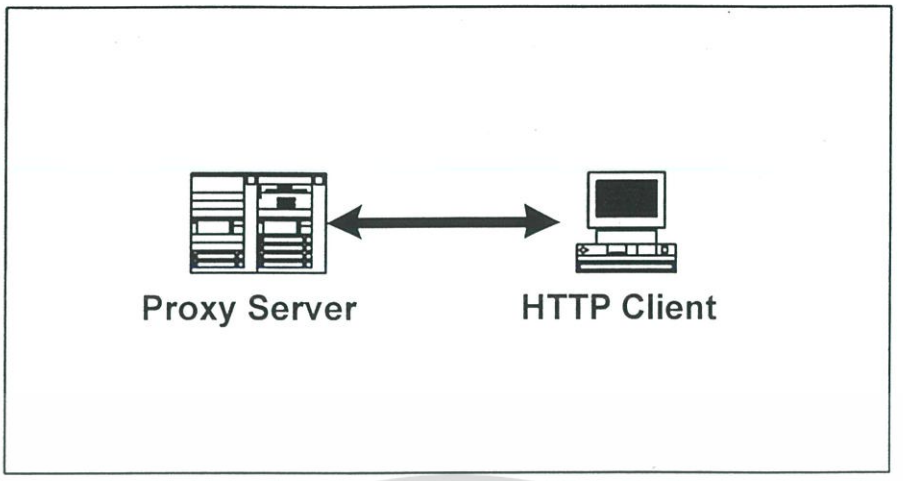
อีกวิธีหนึ่งเป็นวิธีที่พยายามจะสะท้อนถึงความหลากหลายของข้อมูลบนเว็บที่บ้างก็เปลี่ยนเป็นรายชั่วโมง บ้างก็ไม่เคยเปลี่ยนแปลงเลย เมื่อแคชเก็บข้อมูล มันจะกำหนดอายุเวลาของข้อมูลเอาไว้ (time to live หรือ TTL) ข้อมูลนี้จะถูกนำมาใช้โดยมิต้องมีการตรวจสอบกับต้นฉบับจนกระทั่งผ่านพ้นเวลานี้ ใน HTTP Protocol สามารถกำหนด Timestamp Headers ซึ่งนำมาใช้ในการกำหนดค่า TTL ได้ แต่โดยส่วนใหญ่ไม่ได้ถูกนำมาใช้หรือไม่มีอยู่ในเว็บเซิร์ฟเวอร์ทั้งหลาย ดังนั้นสิ่งที่นำมาใช้กันอย่างกว้างขวางก็คือเวลาที่มีการเปลี่ยนแปลงครั้งหลังสุด (Last-Modify) โดยเราจะกำหนด TTL ให้เป็นเปอร์เซ็นของอายุข้อมูลในปัจจุบัน ตัวอย่างเช่นกำหนดให้ TTL มีค่าเป็น 10% ของข้อมูลซึ่งมีการเปลี่ยนแปลงครั้งสุดท้ายเมื่อ 10 วันก่อน ดังนั้นสำเนาข้อมูลจะไม่ถูกตรวจสอบจนกว่าจะผ่าน 1 วันให้หลัง ยังมีอีกสองค่าที่โดยปรกติจะนำมาใช้ในการคำนวณหา TTL ด้วยก็คือค่ามากที่สุดที่จะปกป้องสำเนาข้อมูลให้สามารถนำมาใช้ได้โดยไม่ต้องมีการตรวจสอบ และค่า Default TTL ซึ่งจะใช้สำหรับข้อมูลที่ไม่รู้เวลาของการเปลี่ยนแปลงครั้งหลังสุด

วิธีการใช้ TTL ดูจะเป็นการเพิ่มความสามารถในการจัดการความล้าสมัยได้ในสองทางด้วยกันคือ ข้อมูลที่ไม่มีการเปลี่ยนแปลงจะไม่ถูกตรวจสอบโดยไม่จำเป็น และข้อมูลที่มีการเปลี่ยนแปลงเสมอจะไม่ล้าสมัยจนเกินไปนัก อย่างไรก็ตามผู้ใช้งานมักจะต้องการให้มีการประกันเวลาสูงที่สุดที่ข้อมูลมีความล้าสมัยด้วย โปรแกรมเว็บแคช โดยส่วนใหญ่ยอมให้มีการกำหนดค่าพารามิเตอร์ของความล้าสมัยที่ต่างกันได้ตามรูปแบบ URL ซึ่งโดยทั่วไปจะมีข้อแตกต่างระหว่างเอกสารที่เป็นข้อความและรูปภาพ เอกสารที่เป็นข้อความมักจะมีขนาดเล็กและมักมีความเปลี่ยนแปลงบ่อย ขณะที่เอกสารที่เป็นรูปภาพมีขนาดใหญ่และเปลี่ยนแปลงน้อยกว่า ความสามารถเช่นนี้สามารถนำไปใช้กับการรองรับ Protocol อื่นๆได้อีกด้วย เช่น FTP และ Gopher ซึ่งจะไม่ปรากฏอายุของข้อมูล

เราสามารถอธิบายความสัมพันธ์ระหว่างเครื่องเว็บแคชเซิร์ฟเวอร์ได้ดังนี้

1 Single Proxy

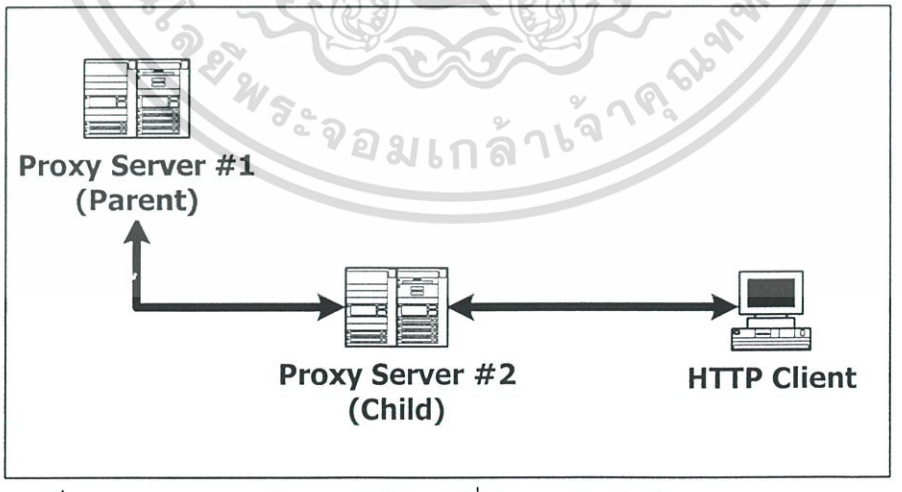
เป็นระบบที่มีเว็บแคชเซิร์ฟเวอร์เพียงตัวเดียวโดยไคลเอนต์ทั้งหมดที่จะติดต่อขอใช้บริการจากเครื่องเซิร์ฟเวอร์นี้เพียงเครื่องเดียว เป็นระบบที่ง่ายต่อการติดตั้งและปรับแต่งค่าทั้งทางฝั่งเซิร์ฟเวอร์ และไคลเอนต์ ตลอดจนการควบคุมสิทธิในการใช้งานของผู้ใช้แต่การที่มีเครื่องให้บริการเพียงเครื่องเดียวอาจจะเสี่ยงแต่ความล้มเหลวของระบบ (Single point of failure) และปัญหาคอขวดที่เครื่องเว็บแคชเซิร์ฟเวอร์มีความสามารถในการให้บริการไม่เพียงพอต่อปริมาณความต้องการของไคลเอนต์ นอกจากนี้การออกแบบระบบให้มีเครื่องเว็บแคชเซิร์ฟเวอร์เพียงเครื่องเดียวนั้นทำให้ต้องใช้เครื่องที่มีประสิทธิภาพในการทำงานและทรัพยากรที่สูงซึ่งหมายความว่าจะต้องใช้งบประมาณที่สูงมากสำหรับเครื่องเซิร์ฟเวอร์นี้



รูปที่ 2.1 แสดงระบบเว็บแคชเซิร์ฟเวอร์แบบ Single Proxy

2 Parent-Child Relation

เป็นระบบที่ประกอบไปด้วย เว็บแคชเซิร์ฟเวอร์ ตั้งแต่สองเครื่องขึ้นไป โดยจะมีเครื่องเว็บแคชเซิร์ฟเวอร์ หลักที่มีขนาดใหญ่ และประสิทธิภาพสูงอยู่ที่ระดับบนสุด (Parent) ทำหน้าที่คอยรับการร้องขอบริการจากเครื่อง Proxy ที่อยู่ระดับต่ำกว่า (Child) ซึ่งโดยมากจะเป็นเครื่องเว็บแคชเซิร์ฟเวอร์ ที่มีขนาดและประสิทธิภาพต่ำกว่าอยู่ในระดับถัดมาเพื่อทำหน้าที่รับการร้องขอบริการจากเครื่อง ไคลเอนต์ ทั้งนี้จำนวนของเครื่องเว็บแคชเซิร์ฟเวอร์ ที่ทำหน้าที่เป็น Child นั้นสามารถมีได้หลายเครื่องเพื่อเป็นการกระจายภาระงานไปยัง เว็บแคชเซิร์ฟเวอร์ หลายตัวและเข้าถึงกลุ่มของ ไคลเอนต์ได้อย่างหลากหลาย



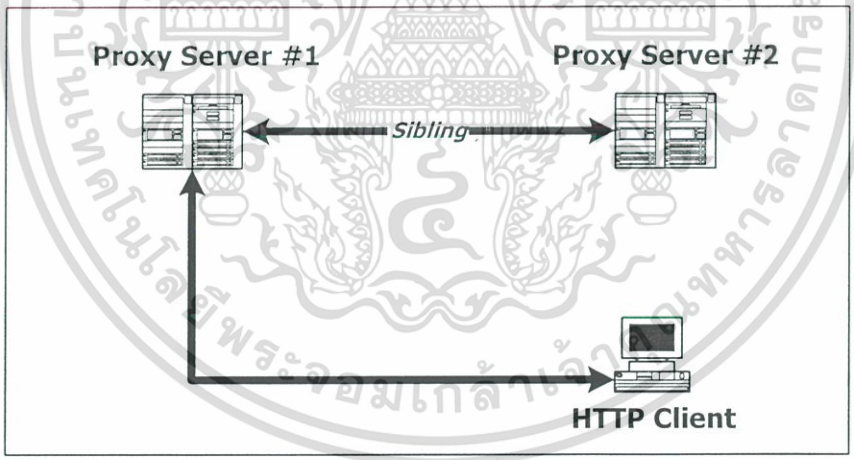
รูปที่ 2.2 แสดงระบบเว็บแคชเซิร์ฟเวอร์ที่มีความสัมพันธ์แบบ Parent-Child

ระบบเว็บแคชเซิร์ฟเวอร์ที่มีความสัมพันธ์แบบนี้เป็นตัวอย่างของระบบแบบ Simple Web-Cache นั้นคือเครื่องเว็บแคชเซิร์ฟเวอร์หมายเลข 2 ในรูปที่ 2.2 จะต้องทำการร้องขอข้อมูลผ่าน

เครื่องเว็บแคชเซิร์ฟเวอร์หมายเลข 1 ทุกครั้งที่ไม่พบข้อมูลในแคชที่ตัวเอง และเมื่อเครื่องหมายเลข 1 นำข้อมูลที่ถูกร้องขอส่งมาให้เครื่องหมายเลข 2 นั้นตัวเครื่องหมายเลข 1 เองก็จะทำการสำเนาข้อมูลดังกล่าวไว้ด้วย ตรงนี้เองที่ทำให้เกิดความซ้ำซ้อนในการเก็บข้อมูล ยิ่งถ้าในกรณีที่ในระบบมีเว็บแคชเซิร์ฟเวอร์หลายเครื่องและมีความสัมพันธ์แบบ Parent-Child ลดหลั่นลงมาเป็น ลำดับชั้น (Hierarchy) ก็จะทำให้เกิดความซ้ำซ้อนในการเก็บสำเนาข้อมูลมาขึ้นตามไปด้วย

3 Sibling Relation

เป็นระบบที่มีเว็บแคชเซิร์ฟเวอร์มากกว่าหนึ่งตัวประกอบกันขึ้นเพื่อรองรับการร้องขอใช้บริการของไคลเอนต์โดยเครื่องเว็บแคชเซิร์ฟเวอร์แต่ละตัวจะทำการร้องถามไปยังเว็บแคชเซิร์ฟเวอร์ที่เป็น Sibling ของมันว่ามีข้อมูลที่ต้องการเก็บไว้หรือไม่พร้อมทั้งตรวจสอบเวลาในการไปร้องขอข้อมูลที่อยู่ที่เครื่องเซิร์ฟเวอร์ต้นทางจากตัวเองเปรียบเทียบกับจากเว็บแคชเซิร์ฟเวอร์ที่เป็น Sibling ของมันเพื่อใช้ในการตัดสินใจ เครื่องเว็บแคชเซิร์ฟเวอร์ที่มีความสัมพันธ์แบบ Sibling นี้จะถือว่ามีความสัมพันธ์อยู่ในระดับ (Level) เดียวกันและสามารถปรับตั้งได้ว่าในกรณีที่เว็บแคชเซิร์ฟเวอร์ตัดสินใจร้องขอข้อมูลจาก Sibling ของมันแล้วจะให้ทำสำเนาหรือไม่ทำสำเนาข้อมูลนั้นเก็บลงในดิสก์แคชของตัวเองก็ได้

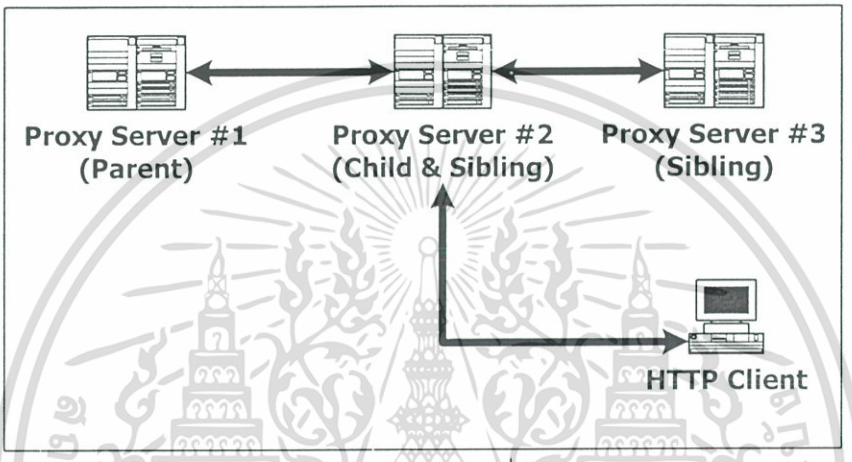


รูปที่ 2.3 แสดงระบบเว็บแคชเซิร์ฟเวอร์ที่มีความสัมพันธ์แบบ Sibling

ระบบเว็บแคชเซิร์ฟเวอร์ที่มีความสัมพันธ์แบบนี้เป็นตัวอย่างของ Cooperate Web-Cache โดยที่เราสามารถปรับตั้งให้เว็บแคชเซิร์ฟเวอร์ที่มีความสัมพันธ์กันแบบ Sibling นี้ไม่ต้องทำการสำเนาข้อมูลที่ร้องขอมาได้จาก Sibling ของตัวเอง (ไม่ใช่ร้องขอมาได้จากเซิร์ฟเวอร์ต้นทาง) ซึ่งจะไม่ทำให้เกิดความซ้ำซ้อนของการจัดเก็บข้อมูล

4 Compound Relation

เป็นระบบที่ประกอบไปด้วย เว็บแคชเซิร์ฟเวอร์ จำนวนมาก ที่มีความสัมพันธ์กันทั้งแบบ Parent-Child และ Sibling ซึ่งสามารถจะรองรับปริมาณการร้องขอใช้บริการของ HTTP Client ได้สูงและมีความอ่อนตัวในการปรับแต่ง Proxy แต่ละตัวเพื่อทำสมดุลภาระงาน (Load-balancing) ซึ่งสามารถเพิ่มจำนวนของเครื่องเว็บแคชเซิร์ฟเวอร์ได้เมื่อมีความต้องการเพิ่มมากขึ้น ตลอดจนสามารถใช้แนวคิดของ Cooperate Web-Cache เพื่อละความซ้ำซ้อนของการเก็บข้อมูลได้อีกด้วย



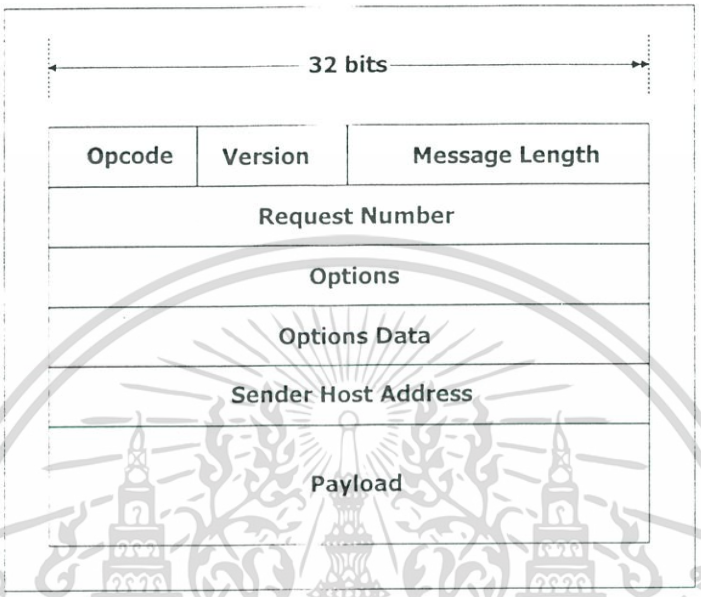
รูปที่ 2.4 แสดงระบบเว็บแคชเซิร์ฟเวอร์ที่มีความสัมพันธ์แบบผสม

ในระบบที่มีเว็บแคชเซิร์ฟเวอร์ทำงานร่วมกันตั้งแต่ 2 เครื่องขึ้นไปจะมีการใช้โพรโตคอลที่ใช้สื่อสารและจัดการระหว่างแคช เพื่อสื่อสารข้อมูลระหว่างแคชเพื่อตรงสอบค่าต่าง ๆ ภายในกลุ่มของเว็บแคช เพื่อเพิ่มประสิทธิภาพ

โพรโตคอล ICP [10] ที่ใช้ในการสื่อสารระหว่างเว็บแคชเซิร์ฟเวอร์ เพื่อแลกเปลี่ยนข้อมูลเกี่ยวกับ URLs ที่มีอยู่ในเว็บแคชเซิร์ฟเวอร์ใกล้เคียงโดยจะเป็นการแลกเปลี่ยน ICP message (ICP-Query & ICP-Reply) เพื่อรวบรวมข้อมูลที่ใช้ในการร้องขอข้อมูลจากเว็บแคชเซิร์ฟเวอร์ที่อยู่ใกล้เคียง โดยเว็บแคชเซิร์ฟเวอร์จะส่ง ICP-Query ไปยังเว็บแคชเซิร์ฟเวอร์ที่อยู่ใกล้เคียง เมื่อได้รับแล้วจึงตอบกลับมาด้วย ICP-Reply เพื่อตอบว่า พบ (HIT) หรือไม่พบ (MISS) ข้อมูลที่ร้องขอนั้น โดยการส่ง ICP-message นี้ จะส่งโดยเลือกใช้บริการจาก TCP หรือ UDP ก็ได้ แต่การใช้บริการจาก UDP จะมีความเหมาะสมมากกว่าเพราะไม่มีการสร้าง Connection จึงทำงานเร็วกว่า TCP หากเกิดการสูญหายของ ICP-message ก็หมายความว่าเกิดความหนาแน่นหรือความเสียหายของเครือข่ายที่ใช้ติดต่อไปยังแคชที่อยู่ใกล้เคียง ในกรณีนี้จึงไม่ควรเลือกเว็บแคชเซิร์ฟเวอร์ตัวนี้ในรับข้อมูลที่ต้องการ นอกจากนี้ TCP ยังมี Overhead ที่ใหญ่กว่า UDP

รูปแบบ ICP message

ICP Message จะประกอบด้วย 2 ส่วนคือ Header ขนาด 20 Octet และ Payload ซึ่งมีขนาดเปลี่ยนแปลงได้ดังรูปที่ 2.5



รูปที่ 2.5 แสดงรายละเอียดรูปแบบของ ICP Message

Opcode: เป็นส่วนที่ใช้บอกประเภทของ ICP-message โดยมีค่า Opcode ที่ใช้ทั่วไปดังนี้คือ ICP_QUERY. ICP_MISS, ICP_HIT เป็นต้น

Version: เป็นส่วนที่ใช้ระบุ version number ของ ICP protocol เพื่อใช้ตรวจสอบ โดยในปัจจุบันมีใช้งาน Version 2 และ 3

Message Length: เป็นส่วนที่ใช้บอกความยาวทั้งหมดของ ICP-message โดยจะมีความยาวไม่เกิน 16384 Octets

Request number: เป็นส่วนที่ใช้ Identify เมื่อมีการตอบ Query ค่านี้จะถูกเก็บลงใน Reply-message ด้วย

Options: มีความยาว 4 Octet เป็น Option flag ที่จะร่วมกับ Opcode เพื่อจุดประสงค์ต่าง ๆ เช่น ICP_FLAG_HIT_OBJ จะใช้ร่วมกับ ICP_QUERY เพื่อยินยอมให้ตอบกลับด้วยข้อมูลเลยหากขนาดของข้อมูลไม่เกินความยาวของคำตอบ (Reply)

Options Data: มีขนาด 4 octet ใช้รองรับการทำงานของ Optional feature ซึ่ง ICP-feature อาจจะต้องใช้ข้อมูลในฟิลด์นี้

Sender Host Address: เป็นส่วนที่ใช้เก็บ IPv4 Address ของ Host ที่ส่ง ICP-Message แต่อย่างไรก็ตามก็มี Address ที่มีพร้อมกับ Socket API อยู่แล้วซึ่งเป็นการซ้ำซ้อน ทำให้ไม่เป็นที่นิยมในการใช้งาน

Payload: เป็นส่วนของเนื้อหาซึ่งมีขนาดไม่แน่นอนโดยจะขึ้นกับ Opcode แต่โดยปรกติแล้วจะเป็นค่า Null-terminated URL String

2.4 Web Cache Software

เนื่องจาก World Wide Web เป็นระบบเปิด (Open System) ที่ยอมให้ระบบและโปรแกรมต่างๆที่มีความแตกต่างกันสามารถทำงานร่วมกันได้ ซึ่ง Web Cache Software ก็มีใช้งานอยู่มากมายในอินเตอร์เน็ต ในส่วนนี้จะอธิบายถึงโปรแกรมเว็บแคช บางโปรแกรมที่นิยมใช้งานกันอยู่

2.4.1 CERN Proxy Server

[11] โปรแกรมนี้ถูกพัฒนาขึ้นโดยใช้โปรแกรมภาษา C และนำมาใช้กับ Unix การทำงานหลักๆของโปรแกรมนี้คือการทำหน้าที่เป็นเว็บเซิร์ฟเวอร์ (Web Server) แต่มีความสามารถในการเป็นเว็บแคช จึงได้รับความนิยม โปรแกรมนี้ทำงานเช่นเดียวกับ Simple Cache ซึ่งสามารถมี Parent ได้เพียง 1 เครื่องเท่านั้น และรองรับโปรโตคอล HTTP, FTP, Gopher และ WAIS ได้ มันไม่สามารถตรวจสอบความล้มเหลวของ Parent และจะส่งข้อผิดพลาดถ้าไม่สามารถติดต่อได้ การตรวจสอบความล้าสมัยของข้อมูลสามารถตรวจสอบได้ทั้งแบบกำหนดเวลาและการใช้ TTL และถ้าต้องการข้อมูลจากเครื่องต้นทางเมื่อข้อมูลมีความล้าสมัยแล้ว จะใช้ IMS GET ในการร้องขอ

ข้อมูลทั้งหมดจะถูกเก็บเป็นไฟล์ (File) ลงในดิสก์ ซึ่งชื่อไฟล์จะถูกกำหนดจาก URL เป็นผลให้โครงสร้างของข้อมูลมี Directory จำนวนมาก การเข้าถึงข้อมูลทำได้ช้า จากผลดังกล่าวทำให้ CERN Server ไม่เหมาะกับงานที่มีปริมาณมากๆ

2.4.2 Apache

[12] โปรแกรมนี้ได้มีการแจกจ่ายให้ใช้ได้ฟรีจึงมีการใช้งานกันอย่างแพร่หลาย เป็นโปรแกรมที่ถูกใช้งานเป็นเว็บเซิร์ฟเวอร์เป็นหลัก โปรแกรมพัฒนาด้วยโปรแกรมภาษา C และสามารถทำงานได้บน Unix ทั้งหมด ในส่วนของ Caching สามารถรองรับการทำงานแบบเลือก Parent Cache โดยดูจากรูปแบบของ URL ได้ ความล้าสมัยสามารถควบคุมได้ทั้งแบบการกำหนดเวลาและการใช้เทคนิค TTL เป็นโปรแกรมที่มีความสามารถในการเป็น Web Caching เช่นเดียวกัน CERN

2.4.3 Netscape Proxy Server

[13] โปรแกรมนี้เป็นโปรแกรมเว็บแคช และพร็อกซี่เซิร์ฟเวอร์โดยเฉพาะ ถูกพัฒนาโดยบุคคลสำคัญที่เป็นผู้สร้าง CERN Proxy ออกจำหน่ายเมื่อปี 1995 และสามารถทำงานได้กับระบบปฏิบัติการหลายๆตัว เช่น Digital Unix, HP-UX, AIX, IRIX, SunOS, Solaris, BSDI, Windows95, Windows NT, Windows XP และสามารถควบคุมผ่านโปรแกรมเว็บบราวเซอร์ (Web Browser) โดยสามารถรองรับการทำงานของโพรโตคอลได้หลายตัว เช่น HTTP, FTP, Gopher, SSL, ICP, CARP (Cache Array Routing Protocol)

ในส่วนของ Caching สามารถรองรับการทำงานแบบเลือก Parent Cache โดยดูจากรูปแบบของ URL และสามารถตรวจสอบความล้มเหลวของ Parent Cache แล้วเลือกเครื่องต่อไปได้ และถ้าทั้งหมดไม่ทำงานจะติดต่อไปยังเครื่องต้นทางที่เป็นต้นฉบับโดยตรง แต่ Netscape Proxy Server ไม่สามารถทำงานในลักษณะ Load Balancing หรือ Discovery ได้ ส่วนความล้าสมัยสามารถควบคุมได้ทั้งแบบกำหนดเวลาและการใช้เทคนิค TTL โปรแกรมนี้สามารถทำงานเป็นไฟร์วอลล์ โดยกรองข้อมูลได้หลายรูปแบบ เช่น จากรูปแบบของ URL, บางลักษณะของ MIME types หรือข้อมูลที่มี Java, JavaScript หรือรูปภาพก็ได้ โปรแกรมนี้สามารถทำงานร่วมกับ SNMP(Simple Network Management Protocol) โดยสามารถกำหนดเงื่อนไขซึ่งจะส่งข้อความเตือนไปยัง SNMP Server ทำให้สามารถตรวจสอบโปรแกรมจากส่วนกลางโดยผ่านอุปกรณ์เครือข่ายอื่นๆได้

2.4.4 Harvest Cache

เป็นส่วนประกอบหนึ่งของ Harvest Project คือข้อมูลแคช และที่นี่เป็นแห่งแรกที่มีการพัฒนาโพรโตคอล ICP (Internet Cache Protocol) โดยตัว Harvest Cache ถูกนำไปพัฒนาต่อโดย Squid และ Netscape โดยสามารถรองรับการทำงานของโพรโตคอล: HTTP, FTP, gopher และ ICP

2.4.5 DeleGate

[14] เป็นเว็บแคชเซิร์ฟเวอร์ที่ทำงานได้บนหลากหลาย Platform ของระบบปฏิบัติการ ไม่ว่าจะเป็น Unix, Windows, OS/2 และสามารถรองรับโพรโตคอลเช่น HTTP, FTP, Gopher, NNTP, POP, SMTP, Telnet, WAIS, X, CU-SeeMe, Socks, SSL และ ICP version 2

2.4.6 Microsoft proxy

[15] ทำงานภายใต้สภาพแวดล้อมของ Windows NT และทำงานร่วมกับโพรโตคอล HTTP และ CARP

2.4.7 Wcol/Catalyst

[16] อาศัยการทำ perfecting เพื่อลดความล่าช้าและเพิ่มแบนด์วิดธ์

2.4.8 Novell BorderManager FastCache

[17] รองรับโพรโตคอล: HTTP, ICP ได้

2.4.9 Cacheflow

[18] ใช้เทคนิค Prefecting ในการลดเวลาแฝงในการร้องขอข้อมูล

2.4.10 Mows

[19] ใช้ภาษาจาวาในการพัฒนาโปรแกรมและสามารถทำงานร่วมกับ โพรโตคอล ICP ได้

นอกจากนี้ยังมีแอพพลิเคชั่นเว็บแคชบางตัวซึ่งไม่ได้กล่าวถึงเพราะว่าในวิทยานิพนธ์นี้มุ่งเน้นศึกษาการทำงานของโปรแกรม Squid เป็นหลัก เนื่องจากเป็นโปรแกรมที่ถูกนิยมนำไปใช้งานในหลาย ๆ ที่ ทั้งสถาบันการศึกษาและภายในองค์การหรือบริษัทต่าง ๆ

2.5 Squid

Internet Research Task Force Research Group on Resource Discovery (IETF-RD) ได้มีการพัฒนาโปรแกรม Harvest เพื่อให้มีการใช้เน็ตเวิร์คของแต่ละเซิร์ฟเวอร์อย่างมีประสิทธิภาพ โปรแกรมถูกออกแบบให้มีการทำงานร่วมกันเองจนกระทั่งสามารถแบ่งภาระกระจายไประหว่างเซิร์ฟเวอร์หลายๆตัว เป็นที่มาของ Internet Cache Protocol (ICP) สำหรับการติดต่อระหว่างเซิร์ฟเวอร์ โปรแกรมสามารถใช้กับ SunOS และ Digital Unix การกำหนดค่า (Configuration) ทำได้โดยการแก้ไข Text File ซึ่งถูกอ่านเมื่อโปรแกรมเริ่มทำงาน โปรแกรมสามารถรองรับโปรโตคอล HTTP, FTP และ Gopher ได้ ค่า Default และ Maximum TTL สามารถกำหนดให้มีค่าต่างกันได้ สำหรับรูปแบบ URL ที่ต่างกัน แต่เมื่อข้อมูลมีความล้าสมัยแล้วก็จะถูกลบไป โดยไม่มีการตรวจสอบโดย IMS GET

Squid [20] คือ Internet Proxy Caching application ซึ่งมีจุดเริ่มต้นมาจากการพัฒนาโปรแกรมของ Harvest Project ซึ่งไม่ได้มีจุดมุ่งหมายหลักในการพัฒนาแคช โดยได้รับเงินทุนสนับสนุนงานวิจัยจาก National Laboratory of Network Research (NLANR) Squid เป็นแอพพลิเคชั่นที่เผยแพร่และเปิดเผย Source Code แจกจ่ายเพื่อให้อาสาสมัครได้มีการทดสอบและปรับปรุงโปรแกรมเพื่อเพิ่มความสามารถในการรองรับการเชื่อมต่อด้วยโปรโตคอลต่างๆ และแก้ไขข้อผิดพลาดที่เกิดขึ้น จากเหตุผลที่กล่าวมาทำให้ Squid เป็นที่นิยมอย่างแพร่หลาย เพราะผู้ที่มีความรู้ทางด้านการเขียนโปรแกรมสามารถจะสามารถปรับปรุงประสิทธิภาพของมันตามความต้องการได้โดย Squid มีความสามารถในการทำ Proxying และ Caching โดยใช้โพรโตคอล

HTTP, FTP และ URL แบบอื่น ๆ เช่น SSL และสามารถจะทำงานแบบ Cache Hierarchies โดยใช้โพรโตคอล ICP, HTCP, CRAP, Cache-digest ในการสื่อสารระหว่างกัน การที่เว็บแคชเซิร์ฟเวอร์จะสามารถทำงานได้อย่างมีประสิทธิภาพได้นั้น จะต้องประกอบด้วยประสิทธิภาพที่ดีของระบบโดยรวม ซึ่งองค์ประกอบที่สำคัญต่อประสิทธิภาพของเว็บแคชเซิร์ฟเวอร์ คือ เวลาที่ใช้ในการเข้าถึงข้อมูลในดิสก์ หน่วยความจำทั้งหมดของระบบ Disk-Throughput และความสามารถในการประมวลผลของ CPU

การที่จะทำให้เว็บแคชเซิร์ฟเวอร์มีเสถียรภาพนั้นผู้ดูแลระบบควรจะมีการเตรียมพร้อมเพื่อรับมือกับปัญหาที่จะเกิดขึ้นไม่ว่าจะเป็นการเตรียมอะไหล่ที่จำเป็นในกรณีฉุกเฉินหรือการเตรียมระบบสำรองที่ติดตั้งระบบปฏิบัติการและ Squid เอาไว้เมื่อเกิดปัญหากับเครื่องเว็บแคชเซิร์ฟเวอร์หลักก็จะสามารถนำเครื่องสำรองมาใช้งานแทนที่ได้ทันที การตัดสินใจเลือกระบบที่จะนำมาใช้ร่วมกับ Squid นั้นมีองค์ประกอบหลาย ๆ อย่างที่ต้องคำนึง ไม่ว่าจะเป็นค่าปริมาณสูงสุดของการร้องขอต่อนาที (Peak number of Request per minute) ซึ่งจะแสดงให้เห็นจำนวนของข้อมูลที่ถูกร้องขอจากไคลเอนต์และสามารถนำมาคำนวณหาภาระงานที่แคชจะต้องแบกรับขณะทำงาน เป็นการยากที่จะคำนวณค่าการร้องขอสูงสุดนี้ ขึ้นอยู่กับลักษณะพฤติกรรมการใช้งานอินเทอร์เน็ตของผู้ใช้ เช่นเดียวกับการตัดสินใจเลือก ฮาร์ดแวร์ ที่เหมาะสม หากไม่มีสถิติการใช้งานอินเทอร์เน็ต เราอาจจะติดตั้งเว็บแคชเซิร์ฟเวอร์ ทดลองเพื่อประเมิน การร้องขอ ที่จะเกิดขึ้นก็ได้ จะเป็นการดี หากเราประเมินฮาร์ดแวร์ ที่เราต้องการมากกว่า ความต้องการใช้งานในปัจจุบัน เพื่อให้สามารถรองรับความต้องการที่จะเพิ่มขึ้นตามมาในอนาคต

Squid สามารถทำงานร่วมกับ ระบบปฏิบัติการ Unix ได้เกือบทุก Platform เนื่องจาก Squid เขียนบน Digital Unix ที่ทำงานกับ GNU C compiler จึงสามารถนำ Squid ไปติดตั้งบน OS ที่มี compiler ดังกล่าว โดยการติดตั้งจะเลือกนำ Source code มา compile หรือนำ Binary Version มาติดตั้ง ซึ่งสามารถกระทำได้ง่ายกว่า แต่อาจจะมีปัญหาเรื่องความปลอดภัยที่เกิดจากการทำงานของ โปรแกรมย่อยต่าง ๆ ที่แอบแฝงอยู่ Squid โดยผู้ไม่หวังดีได้ และในปัจจุบันได้มีการปรับแต่งให้สามารถนำ Squid มาใช้งานได้บน Platform ของ Windows โดยใช้ชื่อว่า SquidNT ซึ่งยังใช้รูปแบบการปรับแต่งตัว Squid เป็นแบบการแก้ไขไฟล์คอนฟิกกุเรชั่น เช่นเดียวกับที่ใช้บน Platform Unix

ส่วนประกอบของ Squid มีส่วนประกอบที่สำคัญโดยทั่วไปดังนี้

1. Client side: เป็นส่วนที่ใช้รับการเชื่อมต่อจาก ไคลเอนต์ แล้วทำการตีความการร้องขอที่ได้รับ และประมวลผลการร้องขอนั้น ๆ ผลของการประมวลผลตรงส่วนนี้ก็จะระบุผลของการร้องขอที่มีเข้ามาด้วยว่ามีผลการทำงานเป็นแบบใด เช่น HIT MISS หรือ

REFRESH เป็นต้น ซึ่งจะมีการเก็บข้อมูลของสถานะของแต่ละการร้องขอที่ประมวลผลเสร็จสิ้นไว้เพื่อใช้งานต่อไป เช่นการเก็บบันทึกลงไฟล์ Log

2. Server side: มีหน้าที่ส่งผ่านการร้องขอจากไคลเอนต์ที่มีผลของการร้องขอว่าไม่พบข้อมูลในแคช (Cache MISS) ไปยังเครื่องเซิร์ฟเวอร์อื่นโดยขึ้นอยู่กับประเภทของโพรโตคอลที่ไคลเอนต์ร้องขอเข้ามาอีกด้วย การส่งผ่านการร้องขอนี้อาจจะส่งผ่านไปยังเว็บเซิร์ฟเวอร์ต้นทางที่ไคลเอนต์ต้องการร้องขอข้อมูลหรืออาจจะเป็นเครื่องเว็บแคชเซิร์ฟเวอร์อื่น ๆ ก็ได้ ทั้งนี้แล้วแต่การปรับตั้งค่าของเว็บแคชเซิร์ฟเวอร์ว่าให้มีการทำงานร่วมกับเว็บแคชเซิร์ฟเวอร์เครื่องอื่น ๆ ด้วยหรือไม่อย่างไร โดยที่ทุก ๆ การร้องขอจากฝั่งเซิร์ฟเวอร์ไปยังเครื่องเซิร์ฟเวอร์อื่น ๆ ดังกล่าวมาแล้วนั้นจะเป็นการส่งผ่านในรูปแบบของโพรโตคอล HTTP เท่านั้น แม้ว่าการร้องขอเริ่มต้นจากทางไคลเอนต์นั้นจะเป็นการร้องขอด้วยโพรโตตอล FTP หรือ Gopher ก็ตาม โพรโตคอล WAIS และ Gopher นั้นไม่ค่อยได้รับความสนใจมากนักเนื่องจากในปัจจุบันมันมีสัดส่วนของปริมาณของข้อมูลและการร้องขอผ่านอินเทอร์เน็ตที่น้อยมากเมื่อเทียบกับ HTTP

3. Storage Manager: ส่วนนี้จะเป็นส่วนเชื่อมประสานระหว่าง client side และ server side โดยจะทำหน้าที่จัดการกับข้อมูลที่เก็บอยู่ในแคช ทั้งการค้นหาข้อมูลจากแคชเมื่อถูกร้องขอ และการจัดการแทนที่ข้อมูลเก่าด้วยข้อมูลที่ใหม่กว่าในกรณีที่ไม่มีเนื้อที่ดิสก์เหลือพอที่จะเก็บข้อมูลใหม่โดยอาศัย อัลกอริธึมการแทนที่ข้อมูล (Replacement Algorithm) แบบต่าง ๆ เช่น Least-Recently-Used (LRU) เป็นต้น

4. Request Forwarding: เป็นส่วนที่จะทำหน้าที่ส่งผ่านการร้องขอจากไคลเอนต์ไปยังเครื่องเว็บเซิร์ฟเวอร์อื่นหรือเว็บแคชเซิร์ฟเวอร์อื่นในบางกรณีที่มีการกำหนดไว้ล่วงหน้าแล้วที่ตัว Squid เช่น ให้ทำการส่งผ่านการร้องขอของไคลเอนต์ที่ต้องการร้องขอข้อมูลจากเว็บเซิร์ฟเวอร์ที่อยู่ภายในโดเมนเดียวกัน ไปยังเว็บเซิร์ฟเวอร์นั้นโดยตรงเลย โดย Squid จะไม่ทำการร้องขอข้อมูลดังกล่าวให้เนื่องจากถือว่าการติดต่อภายในโดเมนเดียวกันนั้นสามารถทำได้อย่างรวดเร็วและมีแบนด์วิดธ์ระหว่างไคลเอนต์และเว็บเซิร์ฟเวอร์ต้นทางที่มากพอ (เช่นอาจจะอยู่บน campus network เดียวกัน)

2.5.1 ขั้นตอนการทำงานของ Squid เว็บแคชเซิร์ฟเวอร์

การทำงานของเว็บแคชเซิร์ฟเวอร์ในที่นี้จะหมายถึงการทำงานที่เกิดจากการสั่งงานของ Squid ซึ่งเป็นแอพพลิเคชั่นที่ทำหน้าที่เป็นพรอกซีอยู่บนตัวเครื่องเว็บแคชเซิร์ฟเวอร์ โดยแบ่งได้เป็นสองช่วง คือ ช่วงเริ่มต้นระบบ (System Initialization Phase) และช่วงให้บริการ (Service Phase)

2.5.1.1 **ช่วงเริ่มต้นระบบ** (System Initialize Phase)

ถ้าเป็นการเริ่มต้นการทำงานครั้งแรกตัวเว็บแคชเซิร์ฟเวอร์จะทำการเตรียมเนื้อที่เก็บข้อมูลบน ดิสก์ โดยการจัดสร้างไดเรกทอรีและไดเรกทอรีย่อยตามที่ถูกตั้งค่าเอาไว้ แล้วจึงเข้าสู่ช่วงการให้บริการ

ถ้าเป็นการเริ่มต้นการทำงานครั้งต่อ ๆ ไปเว็บแคชเซิร์ฟเวอร์จะทำการตรวจสอบความถูกต้องของฐานข้อมูลที่เกี่ยวกับข้อมูลที่เก็บให้ในดิสก์แคชตลอดจนตรวจสอบและลบข้อมูลที่มีอายุ (ที่เก็บไว้ใน ดิสก์แคช) เกินที่ตั้งค่าเอาไว้ จากนั้นจึงเข้าสู่ช่วงการให้บริการ

2.5.1.2 **ช่วงให้บริการ** (Service Phase)

การทำงานจะเริ่มจาก

1. รับการร้องขอ: เมื่อเครื่องเว็บแคชเซิร์ฟเวอร์ได้รับ TCP Connection จากไคลเอนต์ผ่านทางพอร์ตที่กำหนด (ส่วนมากจะเป็น พอร์ต 8080 ยกเว้น Squid ที่ใช้ พอร์ต 3128) โดยไคลเอนต์จะส่งการร้องขอเข้ามายังเว็บแคชเซิร์ฟเวอร์ซึ่งประกอบไปด้วย URL ของข้อมูลและวิธีที่ต้องการกระทำกับข้อมูลนั้น ๆ ตามมาตรฐานของ HTTP/FTP เช่น GET, POST, REFRESH เป็นต้น

2. ค้นหาข้อมูลในเมโมรีแคช: เมื่อได้รับ การร้องขอ เข้ามาเว็บแคชเซิร์ฟเวอร์จะทำการตรวจสอบข้อมูลที่ได้รับการร้องขอนั้นมีอยู่ในเมโมรีแคชหรือไม่ ถ้าพบ ข้อมูล ดังกล่าว (MEM_HIT) ก็จะทำการส่งข้อมูลนั้นให้กับไคลเอนต์แต่ถ้าไม่พบก็จะทำการค้นหาข้อมูลเหล่านั้นใน ดิสก์แคช ต่อไป

3. ค้นหาข้อมูลในดิสก์แคช : ในกรณีที่เว็บแคชเซิร์ฟเวอร์ไม่พบข้อมูลใน เมโมรีแคชก็จะทำการตรวจสอบข้อมูลที่ได้รับการร้องขอนั้นว่ามีเก็บอยู่ในดิสก์แคชของตัวเองหรือไม่

3.1 Request HIT: ถ้าข้อมูลดังกล่าวมีอยู่ในดิสก์แคชและมีอายุไม่เกินที่ตั้งค่าไว้ก็จะทำการอ่านข้อมูลขึ้นมาไว้ในเมโมรีแคชและทำการส่งข้อมูลดังกล่าวไปให้กับไคลเอนต์ที่ได้ร้องขอมา

3.2 Request MISS: ถ้าไม่พบข้อมูลในดิสก์แคช ตัวเว็บแคชเซิร์ฟเวอร์เองจะมีทางเลือกสองทางคือ

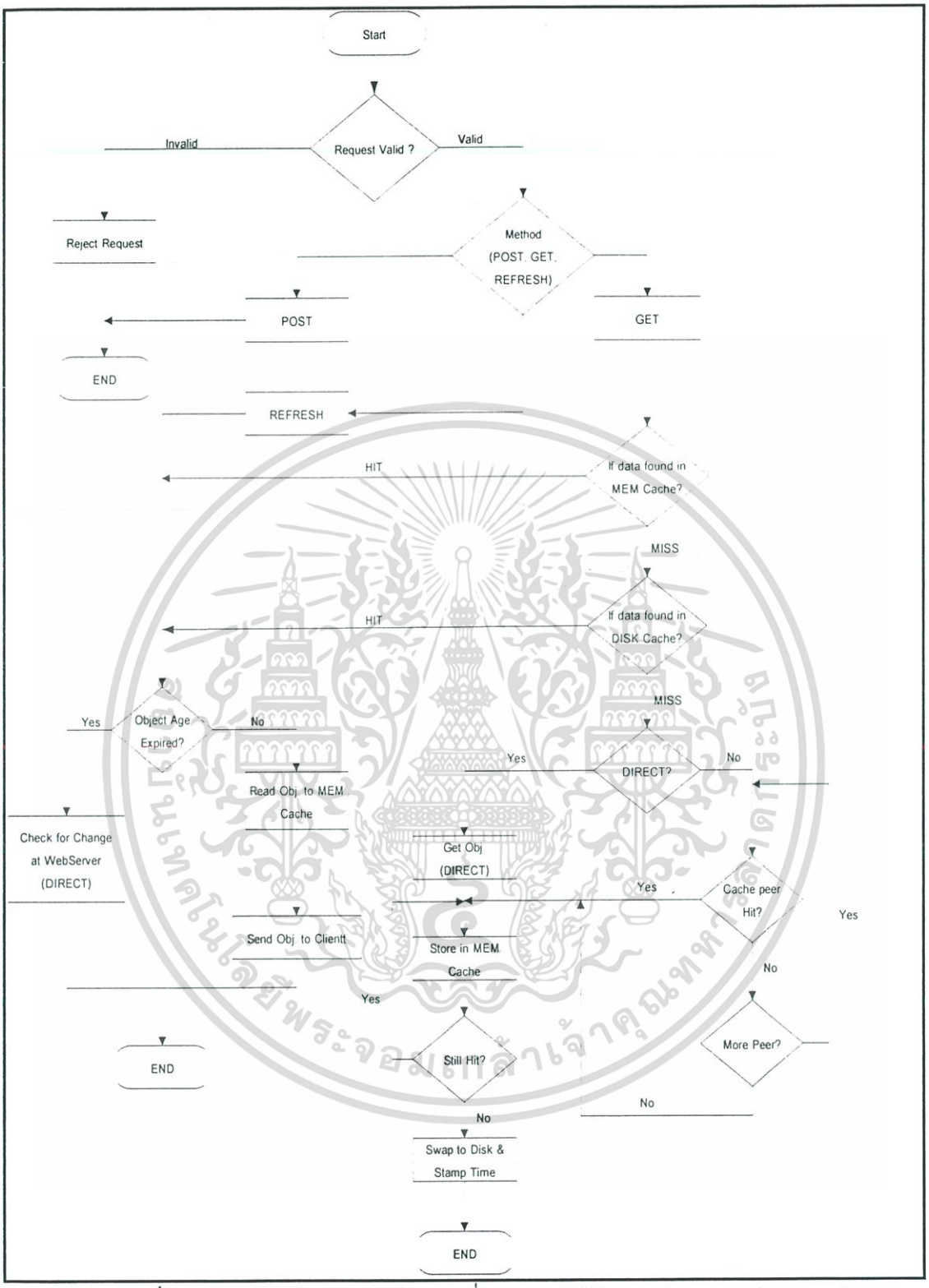
3.2.1 DIRECT: ตัวเว็บแคชเซิร์ฟเวอร์จะทำการติดต่อไปร้องขอ ข้อมูลนั้นจากเว็บเซิร์ฟเวอร์ตาม URL ที่ระบุไว้ในการร้องขอโดยตรง โดยเมื่อได้ข้อมูลดังกล่าวมาแล้วก็จะทำการจัดเก็บไว้ใน เมโมรีแคช และทำการส่งให้กับ ไคลเอนต์ที่ร้องขอมา

3.2.2 CACHE PEER: เว็บแคชเซิร์ฟเวอร์จะสร้างการติดต่อไปยังเว็บแคชเซิร์ฟเวอร์เครื่องอื่นในระบบโดยวิธีการที่จะใช้ขึ้นกับความสัมพันธ์ระหว่างเว็บแคชเซิร์ฟเวอร์คู่นั้น เช่นถ้าทั้งสองเป็น Sibling ซึ่งกันและกัน (อยู่ระดับเดียวกัน) ใน

กรณีของ Squid จะใช้โพรโตคอล ICP ผ่านทาง UDP พอร์ต 3130 แต่ถ้าเว็บแคชเซิร์ฟเวอร์นั้นกับมีความสัมพันธ์แบบ Parent-Child กับเว็บแคชเซิร์ฟเวอร์ที่จะร้องขอบริการเว็บแคชเซิร์ฟเวอร์ก็จะทำหน้าที่เหมือนกับเป็นไคลเอนต์ของเว็บแคชเซิร์ฟเวอร์ ที่มันไปขอใช้บริการ ซึ่งผลของการร้องขอก็จะมีทั้งพบข้อมูล (PARENT_HIT, SIBLING_HIT) และไม่พบ (PARENT_MISS, UDP_MISS) โดยถ้าพบ ข้อมูล ก็จะนำมาเก็บไว้ใน เมโมรีแคช แล้วจึงจัดส่งให้ ไคลเอนต์ ต่อไป แต่ถ้าไม่พบข้อมูลดังกล่าวก็จะทำติดต่อไปยังเว็บแคชเซิร์ฟเวอร์เครื่องต่อไปตามที่ตั้งค่าไว้จนครบ ในกรณีที่ไม่พบข้อมูลที่ต้องการจากเครื่องเว็บแคชเซิร์ฟเวอร์ทุกเครื่องที่ตั้งค่าไว้ก็จะทำการติดต่อไปยังเว็บเซิร์ฟเวอร์ตาม URL ของข้อมูลนั้น

4. เมื่อมีการค้นพบข้อมูล (HIT) ในเมโมรีแคชหรือดิสก์แคชและการสั่ง REFRESH โปรแกรม Squid จะตรวจสอบอายุของข้อมูลว่าอายุเกินกว่าค่าที่กำหนดไว้หรือยัง ในกรณีที่อายุของข้อมูลยังไม่เกินกว่าค่าที่กำหนด เว็บแคชเซิร์ฟเวอร์จะทำการนำข้อมูลไปไว้ที่เมโมรีแคชและส่งข้อมูลไปยังให้ไคลเอนต์ และในกรณีที่อายุของข้อมูลเกินกว่าค่าที่กำหนด เว็บแคชเซิร์ฟเวอร์จะติดต่อไปยังเว็บไซต์ต้นทางของข้อมูลเพื่อตรวจสอบว่าข้อมูลมีการเปลี่ยนแปลงหรือไม่ หากไม่เปลี่ยนแปลงก็จะนำข้อมูลที่มีอยู่ในแคชส่งไปให้ไคลเอนต์พร้อมกับปรับปรุงอายุของข้อมูลที่อยู่ในแคช หากข้อมูลมีการเปลี่ยนแปลงก็จะนำข้อมูลชุดใหม่มา แล้วส่งต่อให้ไคลเอนต์พร้อมกับเก็บลงไปในแคช

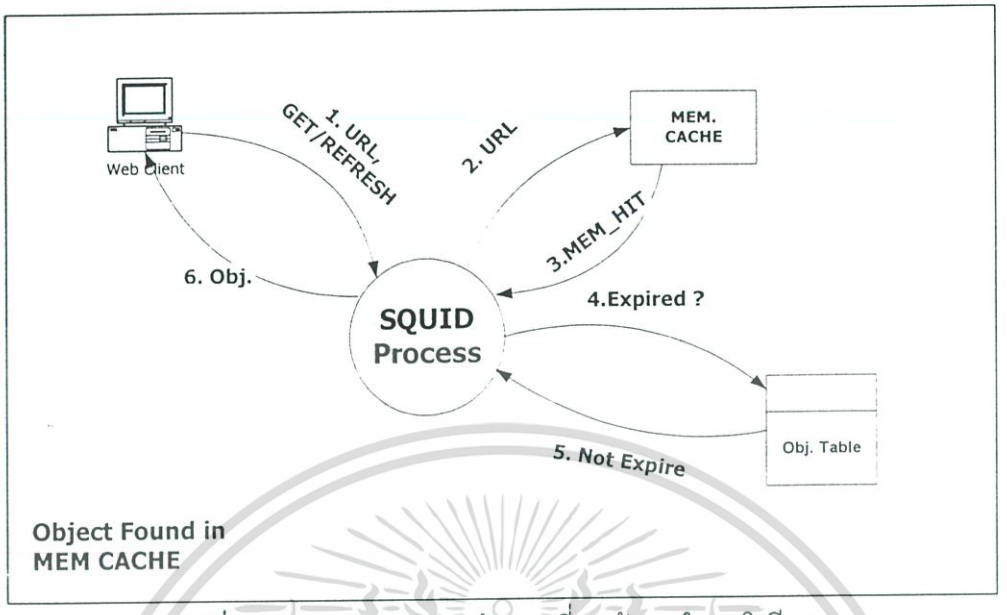
จากขั้นตอนการทำงานที่ได้อธิบายมาข้างต้น สามารถแสดงเป็นโฟลว์ชาร์ตได้ดังรูปที่ 2.6



รูปที่ 2.6 โฟลว์ชาร์ตสรุปการทำงานเมื่อ Squid ได้รับการร้องขอจากไคลเอนต์

จากขั้นตอนการทำงานของ Squid ข้างต้นสามารถนำมาแยกผลของการร้องขอเพื่ออธิบายผลของการร้องขอแต่ละประเภทเพื่อแสดงถึงการไหลของพารามิเตอร์ที่เกี่ยวข้องรวมถึงองค์ประกอบในการทำงานต่าง ๆ ของ Squid โดยอาศัยแผนภาพดังต่อไปนี้

1. กรณีที่พบข้อมูลที่ร้องขอในเมโมรีแคช



รูปที่ 2.7 แสดงผลของการร้องขอที่พบข้อมูลในเมโมรีแคช

ในรูปที่ 2.7การทำงานของ Squid ในกรณีที่พบข้อมูลในเมโมรีแคชจะเริ่มจากขั้นตอนที่1 ถึง 6 ดังนี้

1. มีการร้องขอจากไคลเอนต์เข้ามา โดยไคลเอนต์จะส่ง URL ของข้อมูลและวิธีการที่ต้องการให้เว็บแคชเซิร์ฟเวอร์กระทำต่อข้อมูลนั้น เช่น สั่งให้นำมา (GET) สั่งให้ตรวจสอบการเปลี่ยนแปลง (REFRESH) ส่วนคำสั่งให้ส่งข้อมูลออกไป (POST) นั้นจำ Squid จะทำการส่งออกไปทันทีโดยไม่ได้จัดเก็บข้อมูลดังกล่าวไว้ในดิสก์แคช

2. เมื่อได้ URL มาแล้ว Squid จะทำการตรวจสอบว่ามีข้อมูลที่ระบุนั้นในเมโมรีแคชหรือไม่

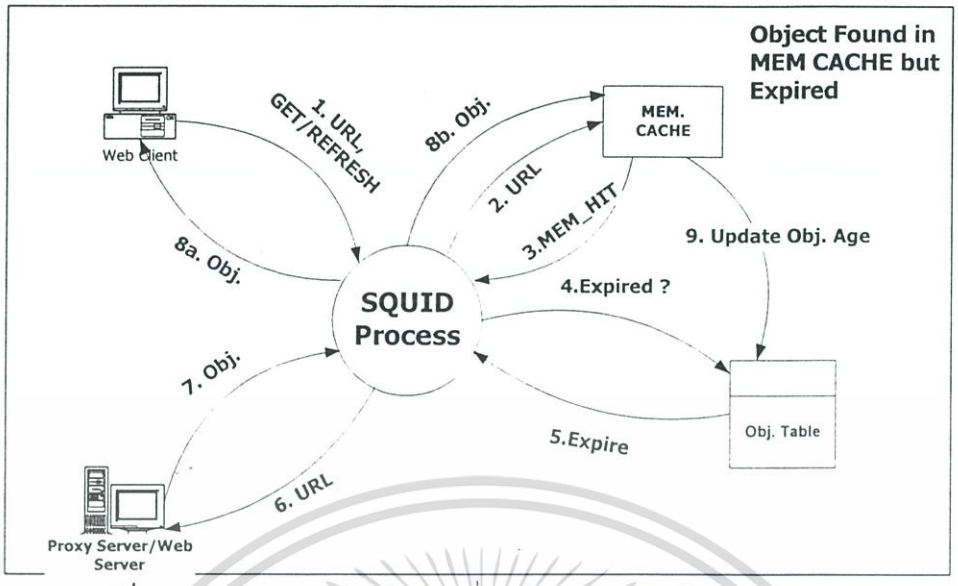
3. ผลของการค้นหาปรากฏว่าพบข้อมูลในเมโมรีแคช

4. ทำการตรวจสอบว่าข้อมูลนั้นมีอายุในการเก็บไว้ในแคชเกินเวลาที่กำหนดไว้หรือไม่

5. ผลของการตรวจสอบพบว่าข้อมูลนั้นมีอายุในแคชยังไม่เกินเวลาที่กำหนด

6. Squid ทำการส่งข้อมูลนั้นไปให้ไคลเอนต์ตามคำร้องขอ

ในกรณีที่ผลของการตรวจสอบว่าข้อมูลที่พบในเมโมรีแคชนั้นมีอายุเกินกว่าที่กำหนดไว้ Squid ก็จะทำการร้องขอข้อมูลนั้นไปยังเซิร์ฟเวอร์ต้นทางตาม URL ของข้อมูลนั้นโดยตรง หรือถ้าหาก Squid ถูกกำหนดค่าให้มีการทำงานร่วมกับเครื่องเว็บแคชเซิร์ฟเวอร์เครื่องอื่นก็จะทำการร้องขอไปยังเว็บแคชเซิร์ฟเวอร์ดังกล่าวตามที่ได้กำหนดค่าเอาไว้ ซึ่งจะมีขั้นตอนเพิ่มตั้งแต่ขั้นตอนที่ 5 ดังแสดงในรูปที่ 2.8



รูปที่ 2.8 แสดงผลของการร้องขอที่พบข้อมูลในเมโมรีแคชแต่หมดอายุ

5. ผลการตรวจสอบปรากฏว่าข้อมูลมีอายุที่เก็บในแคชเกินที่กำหนดไว้

6. Squid จะทำการร้องขอข้อมูลตาม URL ที่ระบุ หรือขอตรวจสอบว่าข้อมูลมีการเปลี่ยนแลงหรือไม่ไปยังเซิร์ฟเวอร์ต้นทางหรือ เว็บแคชเซิร์ฟเวอร์ที่ได้มีการกำหนดค่าไว้

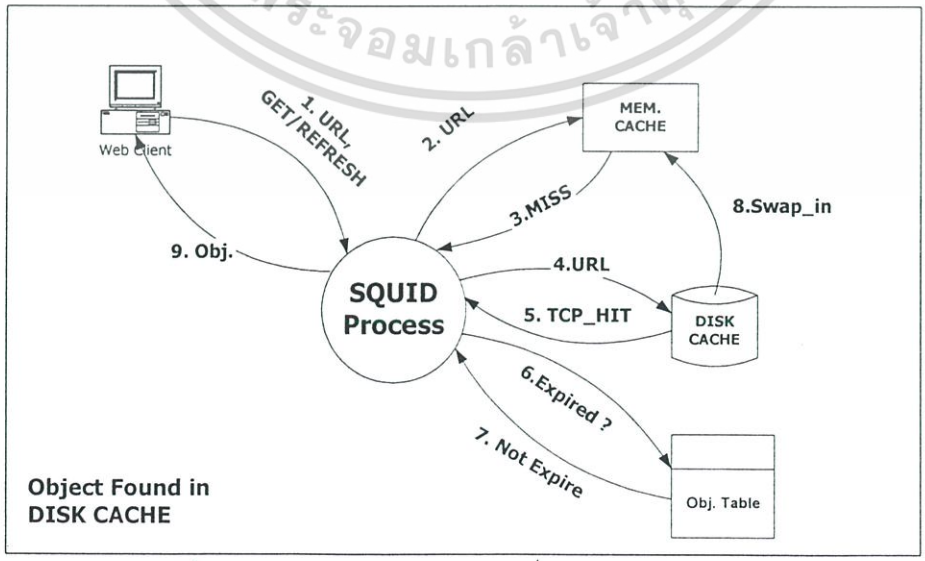
7. ได้รับข้อมูลจากเซิร์ฟเวอร์ที่ร้องขอไปในขั้นตอนที่ 6

8a. Squid จะทำการส่งข้อมูลไปให้กับไคลเอนต์ที่ร้องขอข้อมูลนั้น

8b. Squid ทำการทำสำเนาข้อมูลนั้นไปเก็บไว้ในเมโมรีแคช

9. ทำการปรับปรุง (Update) ค่าอายุของข้อมูลที่เก็บในแคช

2. กรณีที่ไม่พบข้อมูลในเมโมรีแคชแต่พบข้อมูลที่ร้องขอในดิสก์แคช



รูปที่ 2.9 แสดงผลของการร้องขอที่พบข้อมูลในดิสก์แคช

ในรูปที่ 2.9 การทำงานของ Squid ในกรณีที่พบข้อมูลในดิสก์แคชจะเริ่มจากรับการร้องขอจากไคลเอนต์เข้ามาและทำการตรวจสอบว่ามีข้อมูลดังกล่าวในเมโมรีแคชหรือไม่ (ขั้นตอนที่ 1. และ 2.) แต่กรณีนี้ไม่พบข้อมูลที่ต้องการ จึงมีขั้นตอนเพิ่มเติมดังต่อไปนี้

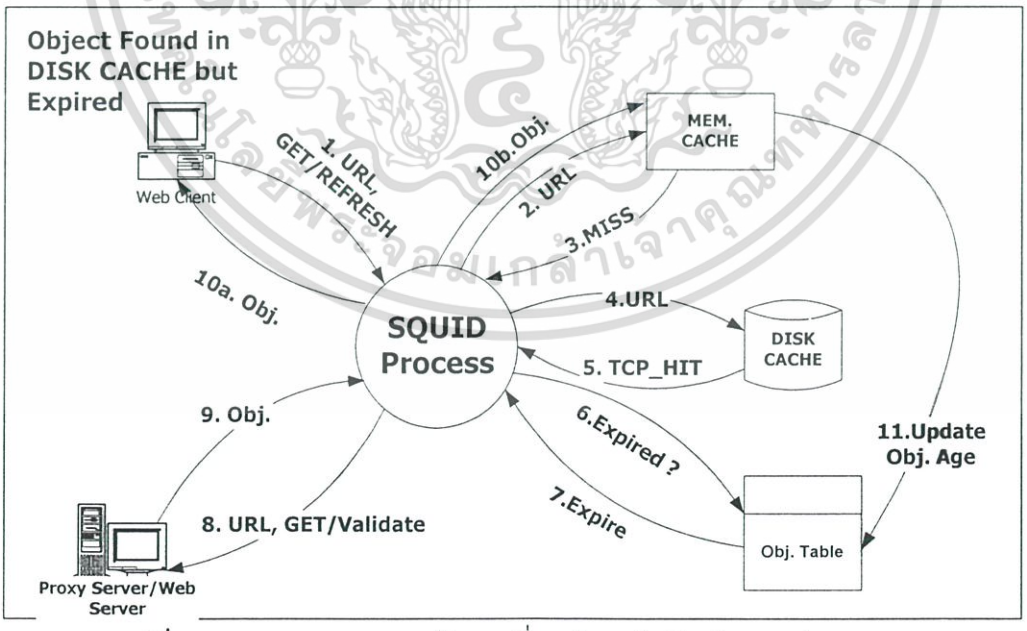
3. ผลการตรวจสอบปรากฏว่าไม่พบข้อมูลในเมโมรีแคช

4. Squid จะทำการตรวจสอบว่ามีข้อมูลตาม URL ในดิสก์แคชหรือไม่

5. ผลการตรวจสอบปรากฏว่าพบข้อมูลในดิสก์แคช

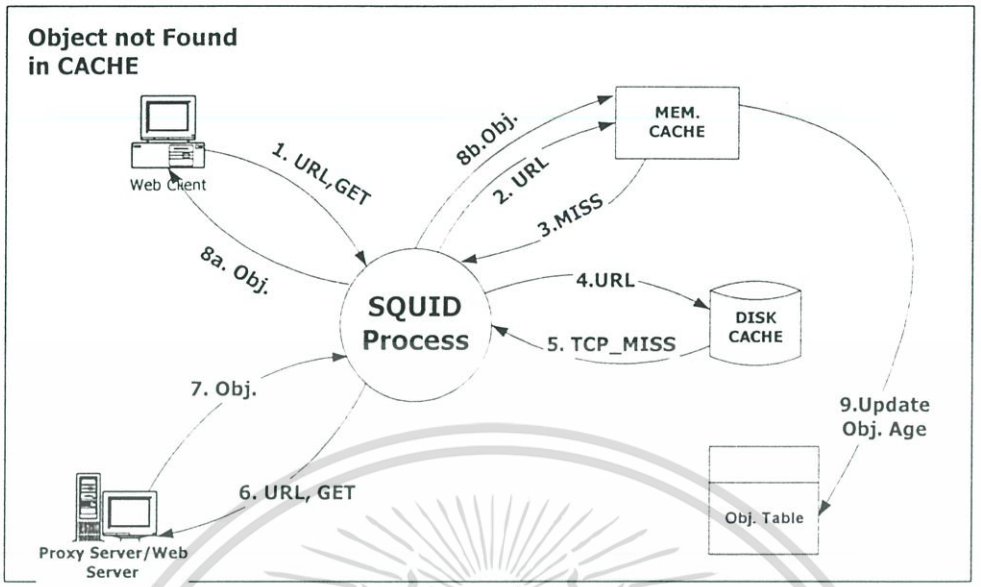
จากนั้น Squid ก็จะทำการตรวจสอบอายุของข้อมูลที่เก็บไว้ในแคชว่าเกินค่ากำหนดที่ตั้งไว้หรือไม่ ถ้าผลของการตรวจสอบปรากฏว่าข้อมูลยังมีอายุไม่เกินค่าที่กำหนด ก็จะทำการสำเนาข้อมูลนั้นไปเก็บไว้ในเมโมรีแคช (ขั้นตอนที่ 8) และส่งข้อมูลนั้นให้กับไคลเอนต์ที่ร้องขอเข้ามา (ขั้นตอนที่ 9)

ในกรณีที่พบข้อมูลในดิสก์แคชแต่ปรากฏว่าข้อมูลนั้นมีอายุที่เก็บไว้ในแคชนานเกินกำหนดที่ตั้งไว้ ก็จะต้องเพิ่มขั้นตอนในการไปร้องขอข้อมูล (GET) หรือขอตรวจสอบว่าข้อมูลดังกล่าวมีการเปลี่ยนแปลงหรือไม่ (REFRESH) ดังกล่าว ดังแสดงไว้ในขั้นตอนที่ 8 และ 9 ในรูปที่ 2.5 ซึ่งเมื่อ Squid ได้รับการตอบรับกลับมาแล้วก็จะทำการส่งข้อมูลนั้นให้กับไคลเอนต์ที่ร้องขอเข้ามาและทำการปรับปรุงค่าอายุของข้อมูลนั้นที่เก็บไว้ในแคชใหม่



รูปที่ 2.10 แสดงผลของการร้องขอที่พบข้อมูลในดิสก์แคชแต่หมดอายุ

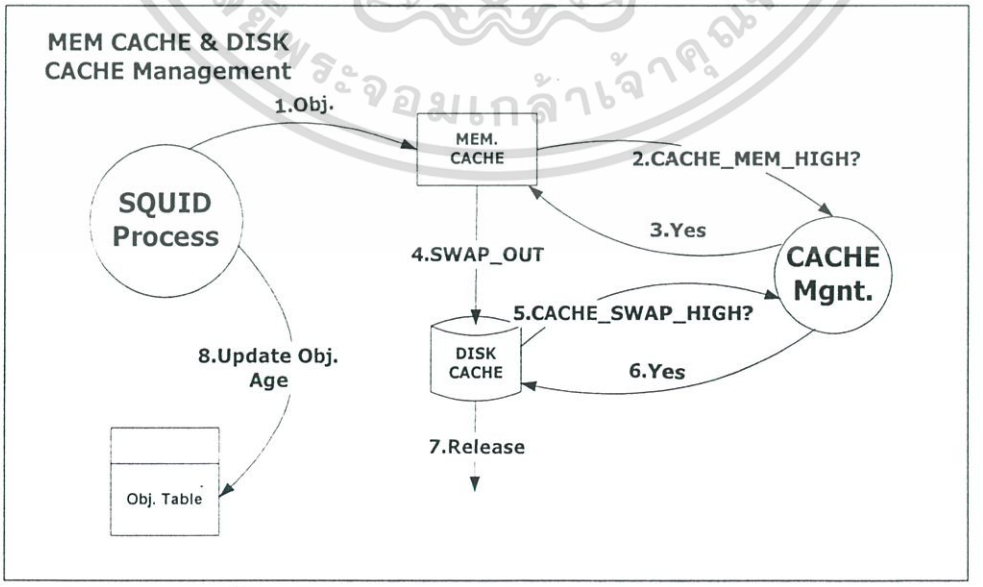
3. กรณีที่ไม่พบข้อมูลในเมโมรีแคชและดิสก์แคช



รูปที่ 2.11 แสดงผลของการร้องขอที่ไม่พบข้อมูลในแคชของ Squid

ในกรณีที่ข้อมูลที่ถูกร้องขอเข้ามานั้นไม่พบอยู่ทั้งในเมโมรีแคชและดิสก์แคชนั้น Squid ก็จะทำการร้องขอข้อมูลนั้นไปยังเซิร์ฟเวอร์ต้นทางหรือเว็บแคชเซิร์ฟเวอร์ที่ได้มีการตั้งค่าเอาไว้ ซึ่งการทำงานก็จะคล้ายกับกรณีที่ Squid เจอข้อมูลที่ถูกร้องขอในแคชของตัวเองแต่ปรากฏว่าข้อมูลนั้นหมดอายุ เพียงแต่ในขั้นตอนที่ 6 จากรูปที่ 2.11 จะเป็นการร้องขอแบบขอข้อมูล (GET) ไม่ใช่การตรวจสอบว่าข้อมูลนั้นมีการเปลี่ยนแปลงหรือไม่ (REFRESH)

ในการทำงานของ Squid จะมีการเก็บข้อมูลใหม่ที่ถูกร้องขอเข้ามาไว้ในแคชโดยจะมีลำดับขั้นตอนการทำงานและส่วนที่เกี่ยวข้องดังแสดงไว้ในรูปที่ 2.12



รูปที่ 2.12 แสดงการจัดเก็บข้อมูลในแคชของ Squid

1. เมื่อ Squid ได้รับข้อมูลที่ถูกร้องขออันใหม่เข้ามาก็นำมาจัดเก็บไว้ในส่วนของเมโมรีแคชเป็นอันดับแรก

2. ถึง 4. เมื่อเนื้อที่ที่ใช้ในการจัดเก็บข้อมูลของเมโมรีแคชมีค่ามากถึงค่าระดับที่ตั้งไว้ก็จะมีการย้ายข้อมูลจำนวนหนึ่งจากเมโมรีแคชลงไปเก็บไว้ในดิสก์แคช

5. ถึง 7. ในทำนองเดียวกันเมื่อเนื้อที่ที่ใช้จัดเก็บข้อมูลของดิสก์แคชมีค่าถึงระดับที่ตั้งไว้ก็จะมีการลบข้อมูลจำนวนหนึ่งทิ้งไปเพื่อให้มีเนื้อที่เหลือพอสำหรับจัดเก็บข้อมูลที่จะเข้ามาใหม่

8. เมื่อมีการนำข้อมูลใหม่เข้ามาเก็บก็จะมีการเก็บค่าอายุที่ข้อมูลนั้นถูกเก็บไว้ในแคชด้วย

ในกรณีที่แคชเก็บข้อมูลจนถึงระดับที่ตั้งไว้ก็จะต้องมีกระบวนการเพื่อใช้ในการตัดสินใจที่จะทำการแทนที่ข้อมูลเก่าซึ่งมีอัลกอริธึมในการแทนที่ข้อมูลอยู่หลายวิธีด้วยกัน ซึ่งจะได้อธิบายอย่างละเอียดภายในบทที่ 3

2.5.2 **ไฟล์ Squid.conf กับพารามิเตอร์ที่มีผลต่อการทำงานของเว็บแคชเซิร์ฟเวอร์**

ไฟล์ Squid.conf เป็นไฟล์สำคัญที่ใช้เก็บค่าพารามิเตอร์ต่าง ๆ ที่ Squid ต้องใช้ในการทำงาน มีลักษณะเป็นไฟล์ตัวอักษรที่สามารถเข้าไปแก้ไขได้ด้วยโปรแกรมอิดิเตอร์ทั่วไป โดยมีพารามิเตอร์ที่สำคัญและมีผลต่อการทำงานในส่วนที่เกี่ยวข้องกับวิทยานิพนธ์ดังต่อไปนี้

http_port เป็นการระบุหมายเลขพอร์ตให้ Squid ใช้ในการรอรับการร้องขอจาก HTTP ไคลเอนต์ ซึ่งการระบุนั้นสามารถทำได้ 3 แบบคือ 1 ระบุหมายเลขพอร์ตอย่างเดียว 2 ระบุชื่อเครื่องกับพอร์ต และ 3 ระบุ IP Address ของเครื่องกับพอร์ต ดังแสดงในตัวอย่างตามลำดับ

```
http_port 3218
http_port proxy.ce.kmitl.ac.th:3128
http_port 161.246.5.101:3128
```

icp_port เป็นการระบุหมายเลขพอร์ตให้ Squid ใช้ในการรับ-ส่ง ICP-Query กับเครื่องเว็บแคชเซิร์ฟเวอร์อื่น โดยสามารถทำการระบุได้ทั้ง 3 แบบเช่นเดียวกับ http_port ที่ได้แสดงไว้ข้างต้น

htcp_port เป็นการระบุหมายเลขพอร์ตให้ Squid สำหรับใช้ในการติด HTCP-Query กับเครื่องเว็บแคชเซิร์ฟเวอร์อื่น

cache_peer เป็นการระบุรายชื่อของพร็อกซีเซอร์เวอร์เครื่องอื่นที่จะกำหนดให้ทำงานร่วมกับ Squid โดยในการตั้งค่า cache_peer นี้จะมีพารามิเตอร์อื่นนอกจากชื่อของพร็อกซีเซอร์อีกดังรูปแบบต่อไปนี้

```
cache_peer hostname type http_port proxy_port options
```

โดยที่

Hostname หมายถึงชื่อเครื่องเว็บแคชเซิร์ฟเวอร์ที่กำหนด

Type หมายถึงรูปแบบความสัมพันธ์ระหว่าง Squid กับเว็บแคชเซิร์ฟเวอร์ที่กำหนด โดยแบ่งเป็นความสัมพันธ์แบบ Parent หรือ Child

proxy_port หมายถึง หมายเลขพอร์ตของเว็บแคชเซิร์ฟเวอร์ที่กำหนดใช้รอรับการร้องขอ ซึ่งจะต้องเป็นหมายเลขเดียวกับ http_port ที่กำหนดไว้บนเว็บแคชเซิร์ฟเวอร์ที่กำหนดด้วย

icp_port หมายถึง หมายเลขพอร์ตของเว็บแคชเซิร์ฟเวอร์ที่กำหนดจะใช้รอรับการร้องขอ ICP-Query ซึ่งจะต้องเป็นหมายเลขเดียวกับ icp_port ที่กำหนดไว้บนเว็บแคชเซิร์ฟเวอร์ที่กำหนดด้วย

options จะเป็นค่าที่กำหนดรายละเอียดปลีกย่อยของการทำงานร่วมกับเว็บแคชเซิร์ฟเวอร์ที่กำหนด ซึ่งมีรายละเอียดดังนี้

proxy-only เป็นการกำหนดให้มีการร้องขอข้อมูลที่ต้องการไปยังเว็บแคชเซิร์ฟเวอร์ที่กำหนดไว้แต่ไม่ต้องทำสำเนาข้อมูลที่ได้รับกลับมาไว้ที่ตัวเอง

weight=n โดย n มีค่าเป็นเลขจำนวนเต็ม การกำหนดค่านี้ให้กับแต่ละเซิร์ฟเวอร์จะเป็นการบอกถึงว่าต้องการในการใช้งานเซิร์ฟเวอร์ว่ามากน้อยเพียงใด โดยที่ Squid จะเลือกใช้งานเซิร์ฟเวอร์ที่มีค่านี้มากที่สุดก่อน ตามปรกติถ้าไม่มีการกำหนดเป็นอย่างอื่นค่า weight จะมีค่าเท่ากับ 1

no-query เป็นการกำหนดว่าไม่ต้องทำการส่ง ICP-Query กับเว็บแคชเซิร์ฟเวอร์นี้

default เป็นการกำหนดให้เว็บแคชเซิร์ฟเวอร์นี้เป็นเครื่องที่ Squid จะส่งการร้องขอข้อมูลไปเมื่อไม่พบข้อมูลที่แคชของตัวเอง

round-robin เป็นการกำหนดให้เว็บแคชเซิร์ฟเวอร์นี้อยู่ในรายชื่อของกลุ่มเครื่องที่จะมีการแบ่งภาระงานแบบ round-robin

closest-only เป็นการกำหนดให้ใช้งานพรอกซีนี้เมื่อไม่มีเซิร์ฟเวอร์เครื่องอื่นตอบกลับมาจาก ICP ว่าไม่พบข้อมูลที่ร้องขอไป

login=user:password ใช้ในการส่ง Username และ Password ไปยังเว็บแคชเซิร์ฟเวอร์ที่กำหนด

ตัวอย่างการกำหนดค่า cache_peer ให้เครื่อง proxy1.ce.kmitl.ac.th ทำหน้าที่เป็น parent โดยใช้ proxy_port เท่ากับ 8080 และ icp_port เท่ากับ 3130 โดยให้มี weight เท่ากับ 20 สามารถเขียนแสดงได้ดังนี้

cache_peer proxy1.ce.kmitl.ac.th parent 8080 3130 weight=20

cache_mem เป็นการกำหนดขนาดของหน่วยความจำหลักที่จะนำมาใช้เป็นเมโมรีแคช มีหน่วยเป็นเมกะไบต์ (Mbytes)

cache_dir เป็นการระบุตำแหน่งบนดิสก์และขนาดที่จะให้ Squid นำมาใช้เป็นดิสก์แคช โดยมีรูปแบบในการระบุค่าดังนี้คือ cache_dir type path size L1 L2 โดยที่

type คือประเภทของระบบดิสก์ที่ใช้ ซึ่งถ้าเป็นระบบปฏิบัติการ Unix ก็จะมีค่าเป็น ufs แต่ถ้าระบบดิสก์ที่ใช้เป็นแบบ Asynchronous I/O ก็จะมีค่าเป็น asyncufs

path คือ ตำแหน่งบนดิสก์ที่จะให้ Squid นำมาใช้ทำดิสก์แคช

size คือขนาดของดิสก์แคชมีหน่วยเป็นเมกะไบต์

L1 และ L2 คือจำนวนของไดเรกทอรีย่อยในลำดับที่ 1 และ 2 นับจาก path ลงไป

ตัวอย่างการกำหนดค่า cache_dir โดยให้ Squid ทำดิสก์แคช ที่ไดเรกทอรี /squid/cache โดยแบ่งไดเรกทอรีย่อยระดับที่ 1 และ 2 เท่ากับ 64 และ 128 ไดเรกทอรีย่อยตามลำดับและกำหนดขนาดของดิสก์แคชไว้ที่ 1 กิกะไบต์ (GigaBytes)

cache_dir ufs /squid/cache 1024 64 128

cache_swap_low และ cache_swap_high เป็นการกำหนดขอบเขตในการลบข้อมูลออกจากเนื้อที่เก็บข้อมูลของแคช โดยค่าทั้งสองจะมีหน่วยเป็นเปอร์เซ็นต์ โดย cache_swap_high จะใหม่ถึงขอบเขตที่เมื่อใดก็ตามที่มีข้อมูลที่จัดเก็บในแคชคิดเป็นเปอร์เซ็นต์เท่ากับค่า cache_swap_high Squid ก็จะทำการลบข้อมูลบางส่วนออกด้วยกระบวนการในการแทนที่ข้อมูลจนข้อมูลที่เหลืออยู่มีการใช้เนื้อที่ในการเก็บคิดเป็นเปอร์เซ็นต์ในแคชต่ำกว่าค่า cache_swap_in

maximum_object_size คือการกำหนดขนาดสูงสุดของข้อมูลที่อนุญาตให้เก็บไว้ในดิสก์แคชได้

cache_access_log คือการกำหนดไดเรกทอรีสำหรับใช้ในการเก็บล็อกไฟล์ (access.log) ซึ่งเป็นผลมากจากการทำงานของ Squid

reference_age คือค่าอายุอ้างอิงที่ Squid จะใช้ในการเปรียบเทียบว่าข้อมูลแต่ละข้อมูลนั้นหมดอายุหรือยังเพื่อที่จะทำการลบข้อมูลนั้นออกจากเนื้อที่ของดิสก์แคชในกรณีที่มีความต้องการใช้เนื้อที่นั้นในการเก็บข้อมูลใหม่ที่เข้ามา

replacement_policy เป็นการกำหนดให้ Squid ใช้กระบวนการในการแทนที่ข้อมูลใหม่ลงในแคชโดยจะมีวิธีเลือกลบข้อมูลออกไปที่แตกต่างกัน เช่น Least-Recently Used (LRU), Greedy-Dual Size Frequency (GDSF) และ Least Frequently Used with Dynamic Aging (LFUDA)

2.5.3 ไฟล์ Log ของ Squid เว็บแคชเซิร์ฟเวอร์

เป็นไฟล์ตัวอักษร (Text file) ที่เก็บรายละเอียดในการทำงานของ Squid ให้บริการการร้องขอโดยจะเก็บหนึ่งบรรทัดต่อหนึ่งการร้องขอที่มีเข้ามาไม่ว่าจะมีผลของการทำงานต่อการร้องขอนั้นเป็นอย่างไร โดย ใน 1 บรรทัดนั้นจะแบ่งออกเป็น 10 ฟิลด์ [6] ดังแสดงในตารางที่ 2.1

ตารางที่ 2.2 แสดงฟิลด์ต่าง ๆ ของไฟล์ Log ที่เกิดจากการประมวลผลการร้องขอของ Squid

Timestamp	Elapsed	Client-address	Log-Tag/ HTTP-Code	Size	Request Method	URL	rfc931	Hierarchy Data / Hostname	Content-Type

1. Timestamp

เป็นเวลาในรูปแบบของระบบปฏิบัติการแบบ Unix โดยเริ่มนับเวลาทีละ 1 วินาทีตั้งแต่วันที่ 1 มกราคม ค.ศ.1970 (Julians Time) โดยการบันทึกเวลาลงไฟล์ Log จะกระทำก็ต่อเมื่อการเชื่อมต่อกับไคลเอนต์ถูกปิดลง

2. Elapsed Time

คือเวลาที่ใช้ในการประมวลผลการร้องขอ ตั้งแต่เริ่มการเชื่อมต่อจนสิ้นสุดการเชื่อมต่อ มีหน่วยเป็นมิลลิเซกคั่น (Millisecond, ms)

3. Client Address

คือ IP Address ของไคลเอนต์ที่ร้องขอข้อมูล ในกรณีที่เปิดการใช้งาน log_fqdn ใน file squid.conf ก็จะเก็บ Full Qualify Domain Name แทน

4. Log Tag / HTTP Code

เป็นส่วนที่ใช้อธิบายว่าการร้องขอที่เข้ามามีผลจากการทำงานเป็นอย่างไร เช่น พบ หรือ ไม่พบข้อมูลที่ร้องขอ ซึ่งจะอธิบายต่อไป ส่วน HTTP code คือข้อมูลในส่วนของ HTTP header ที่ตอบกลับมาจากเซิร์ฟเวอร์ที่พรอกซีไปร้องขอข้อมูลมา ซึ่งจะได้อธิบายแยกต่อไปด้านล่าง

5. Size

ขนาดของข้อมูลที่ถูกร้องขอ มีหน่วยเป็นไบต์

6. Request Method

จะเก็บวิธีการกระทำต่อข้อมูลที่ร้องขอเข้ามา เช่น GET (นำมา) POST (ส่งออกไป) หรือ ICP_Query

7. URL

เก็บ URL ของข้อมูลที่ถูกร้องขอ

8. Ident

ในกรณีที่มีการใช้ความสามารถในการทำ Ident (โดยปรับตั้งค่า Ident_lookup ที่ไฟล์ squid.conf)

9. Hierarchy Data / Hostname

แสดงรายละเอียดว่า ข้อมูลนั้นถูกนำมาจากไหนและอย่างไร เช่น นำมาโดยตรงจากเซิร์ฟเวอร์ต้นทาง หรือ พบที่ Parent ของตัวมัน ซึ่งจะได้อธิบายแยกต่อไปด้านล่าง

10. Content Type

เก็บข้อมูลประเภทของข้อมูลที่ได้รับกลับมาจากเซิร์ฟเวอร์ต้นทาง เช่น เป็นข้อมูล Text, JPG, GIF เป็นต้น

2.5.3.1 **รายละเอียดของ Log Tag**

Log tag ที่ขึ้นต้นด้วย TCP_ จะหมายถึงการเชื่อมต่อด้วยโพรโตคอล HTTP จากไคลเอนต์เข้ามายัง Squid (โดยเข้ามาที่ พอร์ต 3128 ถ้าไม่มีการกำหนดเป็นอย่างอื่นในค่า tcp_port ในไฟล์ squid.conf)

TCP_HIT หมายถึงข้อมูลที่ไคลเอนต์ร้องขอพบอยู่ในแคชและสามารถส่งให้กับไคลเอนต์ได้

TCP_MEM_HIT หมายถึงพบข้อมูลที่ร้องขออยู่ใน เมโมรีแคช

TCP_MISS หมายถึงข้อมูลที่ไคลเอนต์ร้องขอไม่พบอยู่ในแคช

TCP_REFRESH_HIT หมายถึงข้อมูลที่ไคลเอนต์ร้องขอพบอยู่ในแคชแต่หมดอายุ จากนั้น Squid ได้ทำการส่งการร้องขอแบบ If-Modify-Since ไปยังเซิร์ฟเวอร์ต้นทางและได้รับคำตอบกลับมาทาง HTTP reply ว่า "304 Not modified" ซึ่งหมายถึงว่าข้อมูลนั้นยังไม่มีการเปลี่ยนแปลง (ขนาดและ/หรือวันที่สร้าง) จากนั้น squid จึงส่งข้อมูลที่เก็บอยู่ในแคชไปให้ไคลเอนต์พร้อมทั้งเริ่มนับเวลาที่ข้อมูลถูกเก็บในแคชใหม่

TCP_REF_FAIL_HIT หมายถึงข้อมูลที่ไคลเอนต์ร้องขอพบอยู่ในแคชแต่หมดอายุ จากนั้น Squid ได้ทำการส่งการร้องขอแบบ If-Modify-Since ไปยังเซิร์ฟเวอร์ต้นทางแต่ไม่ได้รับการตอบกลับมา Squid จึงทำการส่งข้อมูลที่พบในแคชแต่หมดอายุนั้นไปให้ไคลเอนต์

TCP_REFRESH_MISS หมายถึงข้อมูลที่ไคลเอนต์ร้องขอพบอยู่ในแคชแต่หมดอายุ จากนั้น Squid ได้ทำการส่งการร้องขอแบบ If-Modify-Since ไปยังเซิร์ฟเวอร์ต้นทางและได้รับข้อมูลใหม่กลับมา

TCP_CLIENT_REFRESH หมายถึงไคลเอนต์ระบุการร้องขอแบบไม่ใช้แคช (คือขอให้ Squid ไปร้องขอข้อมูลนั้นโดยตรงจากเซิร์ฟเวอร์ต้นทาง)

TCP_IMS_HIT หมายถึงไคลเอนต์ส่งการร้องขอแบบ If-Modified-Since เข้ามาแต่ปรากฏว่าพบข้อมูลในแคชและข้อมูลยังไม่หมดอายุ Squid จึงทำการส่งข้อมูลที่พบในแคชนั้นให้กับไคลเอนต์

TCP_IMS_MISS หมายถึงไคลเอนต์ส่งการร้องขอแบบ If-Modified-Since เข้ามาและปรากฏว่าพบข้อมูลในแคชแต่ข้อมูลหมดอายุไปแล้ว Squid จึงต้องทำการร้องขอข้อมูลนั้นใหม่

TCP_SWAPFAIL หมายถึงข้อมูลที่ร้องขอถูกลงทะเบียนไว้ในรายชื่อของข้อมูลที่ถูกจัดเก็บอยู่ในดิสก์แคช แต่ไม่สามารถเข้าถึงได้ กรณีนี้อาจจะเกิดจากการหยุดการทำงานอย่างผิดปรกติของ Squid process ทำให้เกิดความผิดพลาดในการจัดเก็บข้อมูลในดิสก์แคช

TCP_DENIED หมายถึงการร้องขอจากไคลเอนต์นี้ถูกปฏิเสธ ซึ่งเป็นผลมาจากการทำงานในส่วนของ Access Control List

Log tag ที่ขึ้นต้นด้วย UDP_ จะหมายถึงการเชื่อมต่อด้วยโพรโตคอล ICP จากไคลเอนต์เข้ามายัง Squid (โดยเข้ามาที่ พอร์ต 3130 ถ้าไม่มีการกำหนดเป็นอย่างอื่นในค่า icp_port ในไฟล์ squid.conf)

UDP_HIT หมายถึงข้อมูลที่ไคลเอนต์ร้องขอพบอยู่ในแคชของพรอกซีปลายทาง (ที่ Squid ส่ง ICP-Query ไปร้องขอ) และสามารถส่งให้กับไคลเอนต์ได้

UDP_HIT_OBJ หมายถึงข้อมูลที่ไคลเอนต์ร้องขอถูกพบเหมือนกับกรณีของ UDP_HIT แต่ตัวข้อมูลที่ต้องการนั้นมีขนาดเล็กพอที่จะส่งกลับมาพร้อมกับคำตอบว่าพบข้อมูลนั้นเลย (ส่งกลับมาในส่วนของ TCP-Reply)

UDP_MISS หมายถึงข้อมูลที่ไคลเอนต์ร้องขอไม่พบอยู่ในแคชของพร็อกซีปลายทาง (ที่ Squid ส่ง ICP-Query ไปร้องขอ)

UDP_DENIED หมายถึงการร้องขอจากไคลเอนต์นี้ถูกปฏิเสธ ซึ่งเป็นผลมาจากการทำงานในส่วนของ Access Control List

UDP_INVALID หมายถึงการร้องขอจากไคลเอนต์ที่ไม่ถูกต้อง

UDP_RELOADING หมายถึงการร้องขอถูกปฏิเสธเนื่องจาก Squid กำลังเริ่มต้นการทำงานของตัวอยู่และยังไม่เสร็จสิ้นกระบวนการเรียกข้อมูลขึ้นมาเพื่อใช้ในการทำงาน

Log tag ที่ขึ้นต้นด้วย "ERR_" หมายถึงการร้องขอแบบ HTTP ที่ผิดพลาดไม่สามารถตีความหรือทำงานได้อย่างถูกต้อง

2.5.3.2 **รายละเอียดของ** Hierarchy Data

DIRECT หมายความว่า Squid นำข้อมูลที่ถูกร้องขอนั้นมาจากเซิร์ฟเวอร์ต้นทางตามที่ระบุไว้ใน URL โดยตรง

FIREWALL_IP_DIRECT หมายความว่า Squid นำข้อมูลที่ถูกร้องขอนั้นมาจากเซิร์ฟเวอร์ต้นทางตามที่ระบุไว้ใน URL โดยตรงเนื่องจากเซิร์ฟเวอร์ต้นทางนั้นอยู่ภายในเครือข่ายหลังไฟลวอลล์ตัวเดียวกับ Squid

FIRST_PARENT_MISS หมายความว่า Squid นำข้อมูลที่ถูกร้องขอนั้นมาจาก Parent ที่ติดต่อได้เร็วที่สุด

FIRST_UP_PARENT หมายความว่า Squid นำข้อมูลที่ถูกร้องขอนั้นมาจาก Parent ที่ว่างในรายชื่อของ parent ทั้งหมดที่มี

LOCAL_IP_DIRECT หมายความว่า Squid นำข้อมูลที่ถูกร้องขอนั้นมาจากเซิร์ฟเวอร์ต้นทางตามที่ระบุไว้ใน URL โดยตรงเนื่องจากเซิร์ฟเวอร์ต้นทางนั้นอยู่ในรายชื่อของโฮสต์ที่ระบุไว้ใน local_ip ในไฟล์ squid.conf

SIBLING_HIT หมายความว่า Squid นำข้อมูลที่ถูกร้องขอนั้นมาจากเว็บแคชเซิร์ฟเวอร์ที่อยู่ในระดับเดียวกัน (Sibling)

NO_DIRECT_FAIL หมายความว่า Squid ไม่สามารถนำข้อมูลที่ถูกร้องขอมาได้เนื่องจากไม่สามารถผ่านไฟลวอลล์หรือไม่มีเว็บแคชเซิร์ฟเวอร์ที่เป็น Parent ว่างในการรับการร้องขอจาก Squid

NO_PARENT_DIRECT หมายความว่า Squid นำข้อมูลที่ถูกร้องขอนั้นมาจากเซิร์ฟเวอร์ต้นทางอันเนื่องมากจากไม่มีรายชื่อของเว็บแคชเซิร์ฟเวอร์ที่เป็น Parent ที่กำหนดไว้สำหรับข้อมูลที่มีการร้องขอเข้ามา

PARENT_HIT หมายความว่า Squid นำข้อมูลที่ถูกร้องขอนั้นมาจากเว็บแคชเซิร์ฟเวอร์ที่เป็น Parent

SINGLE_PARENT หมายความว่าข้อมูลนี้เป็น URL ที่กำหนดไว้ว่าต้องถูกร้องขอจาก Parent ที่ระบุไว้เท่านั้น

SOURCE_FASTEST หมายความว่า Squid นำข้อมูลที่ถูกร้องขอนั้นมาจากเซิร์ฟเวอร์ต้นทางเนื่องมากจากเครื่องเซิร์ฟเวอร์ต้นทางตอบกลับมาก่อนพรอกซีเซิร์ฟเวอร์ตัวอื่น

PARENT_UDP_HIT_OBJ หมายความว่าข้อมูลที่ถูกร้องขอนั้นได้รับกลับมาในการตอบรับ UDP_HIT_OBJ reply จาก Parent เลย

SIBLING_UDP_HIT_OBJ หมายความว่าข้อมูลที่ถูกร้องขอนั้นได้รับกลับมาในการตอบรับ UDP_HIT_OBJ reply จาก Sibling เลย

PASSTHROUGH_PARENT หมายความว่าข้อมูลถูกร้องขอผ่านเว็บแคชเซิร์ฟเวอร์ที่ระบุไว้ใน passthrough_proxy ในไฟล์ squid.conf

SSL_PARENT_MISS หมายความว่าข้อมูลถูกร้องขอผ่านเว็บแคชเซิร์ฟเวอร์ที่ระบุไว้ใน ssl_proxy ในไฟล์ squid.conf.

DEFAULT_PARENT หมายความว่าข้อมูลถูกร้องขอผ่านเว็บแคชเซิร์ฟเวอร์ที่กำหนดไว้เป็น default ใน cache_peer ในไฟล์ squid.conf โดยที่ไม่ได้มีการส่ง ICP-Message ออกไปเพื่อหาพรอกซีที่มีการตอบสนองเร็วที่สุด

ROUNDROBIN_PARENT หมายความว่าข้อมูลถูกร้องขอผ่านเว็บแคชเซิร์ฟเวอร์ที่กำหนดไว้เป็น default ใน cache_peer ในไฟล์ squid.conf และเป็นเว็บแคชเซิร์ฟเวอร์ที่มีการใช้งานน้อยที่สุดจากกระบวนการแบ่งภาระงานแบบ Round-robinโดยที่ไม่ได้มีการส่ง ICP-Message ออกไปเพื่อหาพรอกซีที่มีการตอบสนองเร็วที่สุด

CLOSEST_PARENT_MISS หมายความว่าข้อมูลถูกร้องขอผ่านเว็บแคชเซิร์ฟเวอร์ที่ทีค่า Round-trip time ไปยังเซิร์ฟเวอร์ต้นทางต่ำที่สุด ซึ่งกรณีนี้จะเกิดขึ้นก็ต่อเมื่อมีการเปิดใช้งาน query_icmp ในไฟล์ squid.conf

CLOSEST_DIRECT หมายความว่าข้อมูลถูกร้องขอไปยังเซิร์ฟเวอร์ต้นทางโดยตรง เนื่องจากตัว Squid เองเป็นผู้ที่มีค่า Round-trip time ต่ำที่สุด

บทที่ 3
อัลกอริธึมการแทนที่ข้อมูลที่เกี่ยวข้องกับวิทยานิพนธ์

ในขั้นตอนการทำงานของเว็บแคชเซิร์ฟเวอร์ ช่วงที่เวลาที่เว็บแคชเซิร์ฟเวอร์จะทำการบันทึกข้อมูลที่นำมาจากเว็บไซต์ต้นทางของเว็บเพ็จลงไปในแคชซึ่งเป็นพื้นที่ของฮาร์ดดิสก์หรือแรมของเว็บแคชเซิร์ฟเวอร์ เว็บแคชเซิร์ฟเวอร์จะตรวจสอบพื้นที่ว่างภายในแคชสำหรับการเก็บข้อมูล หากแคชของเว็บแคชเซิร์ฟเวอร์มีเนื้อที่สำหรับเก็บข้อมูลไม่เกินค่าที่กำหนดสำหรับการเก็บข้อมูลภายในแคช เว็บแคชเซิร์ฟเวอร์จะบันทึกข้อมูลที่ได้มาลงไปในแคช ถ้าแคชของเว็บแคชเซิร์ฟเวอร์มีเนื้อที่สำหรับเก็บข้อมูลเกินกว่าค่าที่กำหนดสำหรับการเก็บข้อมูลภายในแคช เว็บแคชเซิร์ฟเวอร์จะเลือกข้อมูลที่จะย้ายจากเมมโมรี่แคช (แคชของเว็บแคชเซิร์ฟเวอร์ในแรม) ไปยังดิสก์แคช (แคชของเว็บแคชเซิร์ฟเวอร์ในฮาร์ดดิสก์) หรือนำออกไปจากดิสก์แคช เพื่อให้พื้นที่ว่างภายในแคชมีเพียงพอสำหรับเก็บข้อมูล โดยเรียกใช้ฟังก์ชั่นที่ชื่อว่า อัลกอริธึมการแทนที่ข้อมูล

3.1 ชนิดของอัลกอริธึมการแทนที่ข้อมูล

อัลกอริธึมการแทนที่ข้อมูล [3][4][5] เป็นฟังก์ชั่นหลักที่ต้องสร้างไว้ภายในเว็บแคชเซิร์ฟเวอร์เพื่อทำหน้าที่ในการจัดการข้อมูลภายในแคชของเว็บแคชเซิร์ฟเวอร์ ในอดีตจนถึงปัจจุบันอัลกอริธึมการแทนที่ข้อมูล ถูกนำเสนอและถูกพัฒนาขึ้นมาหลายชนิด ซึ่งอัลกอริธึมการแทนที่ข้อมูลแต่ละชนิด จะมีรูปแบบการทำงานและตัวแปรที่นำมาพิจารณา แตกต่างกันไป อัลกอริธึมการแทนที่ข้อมูลที่ถูกนิยมนำมาประยุกต์ใช้งานในเว็บแคชเซิร์ฟเวอร์แสดงได้ดังนี้

- Least-Recently Used (LRU) [4][5] จะทำการเลือกแทนที่ข้อมูลออกจากแคชของเว็บแคชเซิร์ฟเวอร์โดยพิจารณาจากอายุของข้อมูล ซึ่งเริ่มนับตั้งแต่เวลาที่ข้อมูลถูกเก็บไว้ในแคชหรือข้อมูลถูกเรียกใช้จากแคชไปจนถึงเวลาปัจจุบัน

- Least-Frequently-Used (LFU) [5] จะพิจารณาเลือกแทนที่ข้อมูลที่มีความถี่ในการร้องขอจากไคลเอนต์น้อยที่สุดออกจากแคชของเว็บแคชเซิร์ฟเวอร์เป็นอันดับแรก

- Size [21] จะทำการเลือกแทนที่ข้อมูลที่มีขนาดใหญ่ที่สุดออกจากแคชของเว็บแคชเซิร์ฟเวอร์เป็นอันดับแรก

- LRU-Threshold [22] จะใช้หลักในการเลือกแทนที่ข้อมูลในแคชเหมือนกับวิธีแบบ LRU แต่มีเกณฑ์ของขนาดสูงสุดของข้อมูลที่จะเก็บไว้ในแคชกำกับไว้ด้วย โดยถ้าข้อมูลที่จะทำการเก็บมีขนาดใหญ่กว่าเกณฑ์ที่กำหนดก็จะไม่เก็บข้อมูลนั้นลงในแคช

- Log (Size) + LRU [22] จะใช้หลักในการเลือกแทนที่ข้อมูลในแคชเหมือนกับวิธีแบบ LRU แต่จะทำการเปรียบเทียบเฉพาะกับข้อมูลที่มีขนาดใกล้เคียงกับข้อมูลที่จะนำมาแทนที่เท่านั้น

- Hyper-G [21] จะใช้หลักในการเลือกแทนที่ข้อมูลในแคชเหมือนกับวิธีแบบ LFU แต่จะนำ เวลาในการเรียกใช้ข้อมูลครั้งล่าสุดและขนาดของข้อมูลมาพิจารณาประกอบ

- Pitkow/Rocker [21] จะใช้หลักในการเลือกแทนที่ข้อมูลในแคชเหมือนกับวิธีแบบ LRU ยกเว้นในกรณีที่ข้อมูลทั้งหมดถูกร้องขอภายในวันเดียวกันทั้งหมด (ไม่มีข้อมูลใดไม่ถูกร้องขอนานเกินกว่า 1 วัน) จะเปลี่ยนมาพิจารณาแทนที่ข้อมูลที่มีขนาดใหญ่ที่สุด

- Lowest-Latency-First [23] อัลกอริธึมนี้แคชจะพยายามลดค่าเวลาแฝง (Latency time) ในการเรียกข้อมูลจากเว็บเซิร์ฟเวอร์ต้นทาง โดยจะทำการแทนที่ข้อมูลที่มีเวลาแฝงในการเรียกข้อมูลน้อยที่สุดก่อนเป็นอันดับแรก

- Least Frequently Used with Dynamic Aging (LFUDA) [24][25] อัลกอริธึมนี้จะทำงานคล้ายกับ LFU แต่จะเพิ่มการพิจารณาอายุของข้อมูลที่ถูกเรียกใช้เข้ากับกับจำนวนครั้งที่ข้อมูลนี้ถูกเรียกใช้ด้วย และจะเลือกข้อมูลที่มีค่าจำนวนการถูกเรียกใช้ซึ่งบวกกับค่าฟังก์ชันอายุของมันแล้วมีค่าน้อยที่สุดออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์

- Greedy-Dual-Size Frequency (GDSF)[24][26] อัลกอริธึมนี้จะนำขนาดของข้อมูลมาคำนวณร่วมกับจำนวนการเรียกใช้งาน และค่าฟังก์ชันอายุของข้อมูลในแคช โดยจะเป็นการนำอัลกอริธึมการแทนที่ข้อมูลแบบ GD-Size มาเพิ่มการพิจารณาถึงความถี่ในการร้องขอข้อมูลด้วย และจะเลือกข้อมูลที่มีค่าการพิจารณาน้อยที่สุดออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์

นอกเหนือจากที่กล่าวมา ในงานวิจัยอื่น ๆ ได้มีการนำเสนออัลกอริธึมการแทนที่ข้อมูลของเว็บแคชเซิร์ฟเวอร์อีกหลาย ๆ ชนิด [5] และในวิทยานิพนธ์นี้ได้ศึกษาและปรับปรุงอัลกอริธึมการแทนที่ข้อมูลแบบ RASM (Replacement Algorithm Selection Mechanism) ซึ่งได้นำเอาอัลกอริธึมที่ได้รับความนิยมคืออัลกอริธึมการแทนที่ข้อมูลแบบ GDSF และ อัลกอริธึมการแทนที่ข้อมูลแบบ LFUDA มาประยุกต์ใช้ด้วยกัน เนื่องจากอัลกอริธึมการแทนที่ข้อมูลทั้ง 2 ชนิด เกี่ยวข้องกับตัวแปรที่เหมาะสมสำหรับการพิจารณาการนำข้อมูลที่อยู่ในแคชของเว็บแคชเซิร์ฟเวอร์ออกไป

3.2 การพิจาณาอัลกอริธึมการแทนที่ข้อมูลแบบ Least Frequently Used with Dynamic Aging (LFUDA)

อัลกอริธึมการแทนที่ข้อมูลแบบ LFUDA เป็นอัลกอริธึมการแทนที่ข้อมูลที่ถูกพัฒนามาจากอัลกอริธึมการแทนที่ข้อมูลแบบ Least Frequency Used (LFU) และอัลกอริธึมการแทนที่ข้อมูลแบบ Least Frequency Used-Aging (LFU-Aging) ตามลำดับ ซึ่งอัลกอริธึมการแทนที่ข้อมูลแบบ LFU และแบบ LFU-Aging จะมีรูปแบบการทำงานดังนี้

3.2.1 **อัลกอริธึมการแทนที่ข้อมูลแบบ** Least Frequency Used (LFU)

อัลกอริธึมการแทนที่ข้อมูลแบบ LFU เป็นอัลกอริธึมการแทนที่ข้อมูลที่จะพิจารณาข้อมูลที่จะนำออกจากแคชของเว็บแคชเซิร์ฟเวอร์โดยใช้จำนวนครั้งหรือความถี่การร้องขอข้อมูลที่อยู่ภายในแคช ซึ่งข้อมูลใดที่มีจำนวนครั้งหรือความถี่ของการร้องขอน้อยครั้งที่สุด จะถูกนำออกไปจากแคชเป็นข้อมูลกลุ่มแรก

3.2.2 **อัลกอริธึมการแทนที่ข้อมูลแบบ** Least Frequency Used-Aging (LFU-Aging)

อัลกอริธึมการแทนที่ข้อมูลแบบ LFU-Aging [5] เป็นอัลกอริธึมการแทนที่ข้อมูลที่ถูกพัฒนามาจากอัลกอริธึมการแทนที่ข้อมูลแบบ LFU เนื่องจากในรูปแบบการแทนที่ข้อมูลแบบ LFU ข้อมูลที่มีความนิยมมากระหว่างช่วงระยะเวลาหนึ่งจะสามารถอยู่ภายในแคชของเว็บแคชเซิร์ฟเวอร์ แม้จะไม่ได้รับการร้องขอเป็นเวลานาน เนื่องจากผลรวมของจำนวนครั้งหรือความถี่ของการร้องขอที่มีค่าสูงของข้อมูลนั้น เพื่อหลีกเลี่ยงปัญหาของแคชตรงนี้ การวิเคราะห์ผลกระทบทางด้านอายุจึงได้ถูกนำเสนอขึ้นมา

การทำงานของอัลกอริธึมการแทนที่ข้อมูลแบบ LFU-Aging จะใช้ค่า Threshold เข้ามาเป็นตัวพิจารณาสำหรับการเลือกข้อมูลที่จะถูกนำออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์ โดยมีเงื่อนไขการพิจารณาดังนี้ หากค่าเฉลี่ยของการนับจำนวนครั้งหรือความถี่การร้องขอทั้งหมดมีค่ามากกว่าค่า Threshold ที่กำหนด จำนวนครั้งหรือความถี่การร้องขอทั้งหมดจะถูกหารด้วย 2 และอัลกอริธึมการแทนที่ข้อมูลแบบนี้ยังใช้ค่าที่มากที่สุดสำหรับการนับจำนวนครั้งหรือความถี่การร้องขอ

3.2.3 **อัลกอริธึมการแทนที่ข้อมูลแบบ** Least Frequently Used with Dynamic Aging (LFUDA)

จากงานวิจัยและการพัฒนาอัลกอริธึมการแทนที่ข้อมูลแบบ LFU และ LFU-Aging จะพบว่า อัลกอริธึมการแทนที่ข้อมูลแบบ LFU จะพิจารณาเฉพาะความถี่ของการร้องขอข้อมูลเท่านั้น และอัลกอริธึมการแทนที่ข้อมูลแบบ LFU-Aging จะขึ้นอยู่กับพารามิเตอร์อย่างมาก (ค่า Threshold และ ข้อมูลที่มีความถี่สูง) อัลกอริธึมการแทนที่ข้อมูลแบบ LFUDA พยายามแก้ไขปัญหาที่เกิดขึ้น โดยได้นำอัลกอริธึมการแทนที่ข้อมูลแบบ LFU-Aging มาพิจารณาและแทนที่ฟังก์ชั่นในส่วนของอายุข้อมูล ด้วยฟังก์ชั่นที่มีความเป็นไดนามิกส์เข้าไป ซึ่งฟังก์ชั่นที่เป็นไดนามิกส์ที่นำเข้าไปแทนที่จะเรียกว่า ฟังก์ชั่น Running Age (L) โดย L จะมีค่าเริ่มต้นที่ 0 และจะมีการปรับเปลี่ยนค่าเมื่อมีการนำข้อมูล f ออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์: $L = K_f$ และค่า L ที่มีการปรับเปลี่ยนจะถูกนำไปใส่ให้กับค่าคีย์ (K) ของข้อมูลใหม่ที่เข้ามาแทนที่

จากการพัฒนาอัลกอริธึมการแทนที่ข้อมูลแบบ LFUDA ทำให้เมื่อเว็บแคชเซิร์ฟเวอร์ต้องการพิจารณาข้อมูลที่จะนำออกไปจากแคช สามารถพิจารณาโดยคำนวณค่าคีย์ (K) ของข้อมูลแต่ละตัวภายในแคชของเว็บแคชเซิร์ฟเวอร์ โดยมีสมการสำหรับการคำนวณค่าคีย์แสดงได้ดังสมการที่ 3.1

$$K_i = (C_i * F_i) + L \tag{3.1}$$

โดย K_i คือค่าคีย์สำหรับนำมาพิจารณาข้อมูลที่จะนำออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์

C_i คือค่าคงที่ซึ่งนำมารวมเมื่อนำข้อมูลเข้ามาในแคช

F_i คือจำนวนครั้งหรือความถี่ของการร้องขอข้อมูลในแคชของเว็บแคชเซิร์ฟเวอร์

L คือฟังก์ชั่น Running Age [5][6] ซึ่งจะมีค่าเริ่มต้นที่ 0 และจะมีการปรับเปลี่ยนค่าเมื่อมีการนำข้อมูล f ออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์: $L = K_f$ และค่า L ที่มีการปรับเปลี่ยนจะถูกนำไปใส่ให้กับค่าคีย์ (K) ของข้อมูลใหม่ที่เข้ามาแทนที่

3.3 การพิจารณาอัลกอริธึมการแทนที่ข้อมูลแบบ Greedy-Dual-Size Frequency (GDSF)

อัลกอริธึมการแทนที่ข้อมูลแบบ (GDSF) ได้รับการนำเสนอโดย Cherkasova [26] ซึ่งเป็นอัลกอริธึมการแทนที่ข้อมูลที่ได้รับการพัฒนามาจากอัลกอริธึมการแทนที่ข้อมูลแบบ Greedy-Dual (GD) ที่ได้รับการนำเสนอโดย Young [27] และอัลกอริธึมการแทนที่ข้อมูลแบบ Greedy-Dual-Size (GD-Size) ที่ได้รับการนำเสนอโดย Cao และ Irani [28] ซึ่งอัลกอริธึมการแทนที่ข้อมูลแบบ GD และ GD-Size จะมีรูปแบบการทำงานดังนี้

3.3.1 อัลกอริธึมการแทนที่ข้อมูลแบบ Greedy-Dual (GD)

อัลกอริธึมการแทนที่ข้อมูลแบบ GD ได้รับการนำเสนอโดย Young ซึ่งจะพิจารณาเกี่ยวข้องกับกรณีที่เมื่อข้อมูลภายในแคชของเว็บแคชเซิร์ฟเวอร์มีขนาดเดียวกัน แต่มีค่าที่จะพิจารณาข้อมูลแตกต่างกันเพื่อนำข้อมูลออกมาจากแคช

อัลกอริธึมการแทนที่ข้อมูลนี้สัมพันธ์กับค่า H กับข้อมูลเพ็จ p แต่ละเพ็จที่ถูกเก็บไว้ในแคช ในเบื้องต้น เมื่อข้อมูลเพ็จถูกนำมาเก็บไว้ในแคช ค่า H ถูกกำหนดให้มีค่าที่จะนำข้อมูลเพ็จไปเก็บไว้ในแคชของเว็บแคชเซิร์ฟเวอร์ (ให้สังเกตว่าค่าจะไม่เป็นค่าลบ) เมื่อมีการทำการแทนที่ข้อมูล ข้อมูลเพ็จที่มีค่า H น้อยที่สุด ($\min_H$) จะถูกแทนที่และข้อมูลเพ็จทั้งหมดจะถูกลดค่า H

ของข้อมูลเพ็จด้วยค่า $\min_H$ ถ้าข้อมูลเพ็จ p ถูกร้องขออีกครั้งหนึ่งค่า H ในปัจจุบันของข้อมูลเพ็จนั้นจะถูกจัดเก็บใหม่ ให้กับค่าเริ่มต้นของการนำข้อมูลเพ็จนี้เข้าสู่แคชของเว็บแคชเซิร์ฟเวอร์

ด้วยวิธีการนี้ค่า H ของข้อมูลเพ็จที่มีการร้องขอเมื่อไม่นานนี้ จะยังคงมีค่าที่มากกว่าของค่าเริ่มต้นเมื่อนำไปเปรียบเทียบกับข้อมูลเพ็จที่ไม่ได้รับการร้องขอเป็นเวลานาน อัลกอริธึมการแทนที่ข้อมูลแบบ GD จะใช้ข้อมูลเพ็จที่มีค่า H น้อยที่สุดเพื่อใช้สำหรับการแทนที่ข้อมูลในครั้งแรก ซึ่งค่านี้จะเป็นค่าที่ "แพงน้อยที่สุด" ที่จะนำไปเก็บไว้ในแคชของเว็บแคชเซิร์ฟเวอร์หรือเป็นข้อมูลเพ็จที่ไม่ได้รับการร้องขอเป็นเวลานาน

เพื่อพัฒนาและเพิ่มประสิทธิภาพการทำงานของอัลกอริธึมการแทนที่ข้อมูลแบบ GD โดยการใช้ลำดับความสำคัญและใช้ค่าชดเชย (offset value) สำหรับการปรับแต่งค่า H จึงได้มีการนำเสนออัลกอริธึมการแทนที่ข้อมูลแบบ Greedy-Dual-Size ขึ้นมา

3.3.2 อัลกอริธึมการแทนที่ข้อมูลแบบ Greedy-Dual-Size (GD-Size)

เนื่องจากเว็บแคชเซิร์ฟเวอร์จะเป็นเซิร์ฟเวอร์ที่ทำหน้าที่เกี่ยวข้องกับการเก็บข้อมูลขนาดต่าง ๆ Cao และ Irani ได้พัฒนาอัลกอริธึมแบบ GD ให้มีการพิจารณาเกี่ยวข้องกับตัวแปรขนาดของข้อมูล โดยปรับค่า H ไปเป็น $cost/size$ โดย $cost$ คือค่าของการนำเข้าของข้อมูล และ $size$ คือขนาดของข้อมูลในหน่วยไบต์ จากกระบวนการนี้ทำให้เรียกอัลกอริธึมการแทนที่ข้อมูลที่ได้พัฒนาใหม่ว่า อัลกอริธึมการแทนที่ข้อมูลแบบ Greedy-Dual-Size (GD-Size)

ผลการจำลองการทำงานที่แสดงให้เห็นโดย Cao และ Irani แสดงให้เห็นว่าอัลกอริธึมการแทนที่ข้อมูลที่นำเสนอทำงานได้ดีกว่าอัลกอริธึมการแทนที่ข้อมูลตัวอื่นที่รู้จักกันในปัจจุบัน สำหรับเมตริกซ์ประสิทธิภาพที่แตกต่างกันด้วยฟังก์ชั่น $cost$ ที่ถูกเลือก
เมตริกซ์ธรรมดา 2 แบบที่ถูกนำมาใช้ประเมินประสิทธิภาพของเว็บแคชเซิร์ฟเวอร์คือ

- Hit ratio คือ จำนวนของการร้องขอที่พบข้อมูลที่ร้องขอภายในแคชของเว็บแคชเซิร์ฟเวอร์เป็นเปอร์เซ็นต์ของการร้องขอทั้งหมด
- Byte-Hit ratio คือจำนวนไบต์ของขนาดข้อมูลที่มีการร้องขอและพบข้อมูลที่ร้องขอภายในแคชของเว็บแคชเซิร์ฟเวอร์เป็นเปอร์เซ็นต์ของจำนวนการร้องขอทั้งหมดเป็นไบต์

เพื่อให้ได้ค่า Hit ratio ที่ดีที่สุด ฟังก์ชั่น $cost$ สำหรับข้อมูลแต่ละตัวจะตั้งค่าเป็น 1 โดยวิธีนี้ ข้อมูลที่มีขนาดใหญ่กว่าจะมีค่าคีย์ความสำคัญที่เล็กกว่าข้อมูลที่มีขนาดเล็กกว่า และมีแนวโน้มที่จะถูกนำออกไปจากแคชหากไม่ได้รับการร้องขออีกในอนาคต การขยายค่า Hit ratio จะมีประโยชน์มากกว่าที่จะแทนที่ข้อมูลที่มีขนาดใหญ่เพียงข้อมูลเดียว (และพลาดข้อมูลนี้หากข้อมูลนี้ได้รับการร้องขออีกครั้ง) มากกว่าที่จะนำข้อมูลที่มีขนาดเล็กจำนวนมากออกไปจากแคช เพื่อนำพื้นที่ของแคชที่ใช้ขนาดเดียวกัน (และพลาดข้อมูลเหล่านี้เป็นจำนวนมากเมื่อข้อมูลเหล่านี้ถูกร้องขออีกครั้ง

หนึ่ง) ค่า $cost$ ที่เป็น 1 ข้อมูลเล็ก ๆ ที่ได้รับความนิยมและเข้าแทนที่ข้อมูลที่มีขนาดใหญ่ โดยเฉพาะสิ่งที่อ้างอิงที่หายากแหละนี้ ทำให้เรียกรูปแบบนี้ว่า GD-Size ดังนั้น GD-Size จะมีค่าของ Hit ratio ที่ดีที่สุด แต่ค่า Hit ratio ที่สูงสำหรับ GD-Size จะให้ค่าของ Byte-Hit ratio ที่ต่ำกว่า

รูปแบบ GD-Size(packets) ตั้งฟังก์ชั่น $cost$ สำหรับข้อมูลเป็น $2 + size / 536$ ซึ่งเป็นค่าประมาณของแพ็กเก็ตในเครือข่ายที่มีการส่งและรับในกรณีที่เว็บแคชเซิร์ฟเวอร์ไม่พบข้อมูลที่ถูกร้องขอ รูปแบบ GD-Size(packets) จะมีค่า Hit ratio และค่า Byte-Hit ratio ที่สูง ฟังก์ชั่น $cost$ นี้จะให้ค่าคีย์ที่มีขนาดใหญ่กว่าสำหรับข้อมูลที่มีขนาดใหญ่มากกว่าข้อมูลที่มีขนาดเล็ก ฟังก์ชั่นนี้จะยอมให้ข้อมูลที่มีขนาดเล็กแทนที่ได้มากกว่าข้อมูลที่มีขนาดใหญ่ (โดยเฉพาะถ้าเอกสารที่มีขนาดใหญ่ถูกร้องขอบ่อย) ถ้าเอกสารที่มีขนาดใหญ่ไม่ถูกอ้างอิงอีกเลย ก็จะถูกแทนที่โดยใช้ตัวแปรทางด้านอายุ

ดังนั้น GD-Size จึงมีจุดมุ่งหมายที่จะทำให้ Miss ratio ลดน้อยลง ขณะที่ GD-Size(packets) พยายามที่จะทำให้ความคับคั่งของเครือข่ายที่มีผลกระทบจากการ Miss ข้อมูลลดลง ที่จริงแล้ว ดังที่ได้พิจารณาเมตริกซ์ทั้ง 2 จะมีความขัดแย้งกัน และมีความยากมากที่รูปแบบใดรูปแบบหนึ่งจะให้ผลการทำงานของเมตริกซ์ทั้ง 2 ที่ดี โดยทั่วไปค่า Hit ratio ที่สูงเป็นทางเลือกที่ดีกว่าเพราะเป็นการแสดงว่าไคลเอนต์เป็นจำนวนมากได้ค้นพบข้อมูลที่ร้องขอจากเว็บแคชเซิร์ฟเวอร์และลดค่าเวลาแฝงเฉลี่ยของการร้องขอ แต่การลดความคับคั่งของเครือข่ายก็ยังเป็นที่ต้องการมากกว่า รูปแบบที่มีค่า Byte-Hit ratio สูงจึงถูกนำมาใช้ อัลกอริธึมการแทนที่ข้อมูลแบบ GD-Size ที่มี $cost$ ที่เหมาะสมทำได้เหนือกว่าอัลกอริธึมการแทนที่ข้อมูลที่เป็นที่รู้จักกันในปัจจุบันเกี่ยวกับจำนวนของเมตริกซ์ของประสิทธิภาพรวมทั้งค่า Hit ratio และค่า Byte-Hit ratio

แต่อัลกอริธึมการแทนที่ข้อมูลแบบ GD-Size ยังมีข้อบกพร่องอีกประการคือ ไม่ได้มีการนับจำนวนครั้งของการร้องขอในอดีตเข้าไปในคีย์สำหรับการพิจารณาการนำข้อมูลออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์ ดังนั้นจึงได้มีการพัฒนาอัลกอริธึมการแทนที่ข้อมูลแบบ GD-Size ให้มีตัวแปรที่เกี่ยวของกับจำนวนครั้งของการร้องขอข้อมูล มาเป็น อัลกอริธึมการแทนที่ข้อมูลแบบ Greedy-Dual-Size Frequency โดยปรับค่า H ให้เป็น

$$H = frequency \times \frac{cost}{size} \tag{3.2}$$

3.3.3 อัลกอริธึมการแทนที่ข้อมูลแบบ Greedy-Dual-Size Frequency (GDSF)

จากการพัฒนาปรับปรุงอัลกอริธึมการแทนที่ข้อมูลแบบ GD และ แบบ GD-Size ให้เป็นอัลกอริธึมการแทนที่ข้อมูลแบบ GDSF จะสามารถอธิบายการทำงานของอัลกอริธึมการแทนที่ข้อมูลแบบ GDSF ได้ดังนี้

- ให้ขนาดแคชของเว็บแคชเซิร์ฟเวอร์เป็น *Total*

- ให้ $Used$ เป็นจำนวนพื้นที่ของแคชที่ใช้สำหรับจัดเก็บข้อมูลถูกนำมาเก็บไว้ในเว็บแคชเซิร์ฟเวอร์
- เริ่มต้นให้ $Used = 0$
- เมื่อข้อมูลแต่ละตัว f ถูกเก็บไว้ในแคช และได้ทำการนับความถี่ $Fr(f)$: จำนวนครั้งของการร้องขอ ข้อมูล f ที่ไม่อยู่ในแคชแต่กำลังจะถูกนำมาเก็บไว้ในแคชจะมีค่าของความถี่เป็น 1: $Fr(f) = 1$

เพื่อระบุว่าข้อมูลใดจะถูกแทนที่เมื่อขนาดความจุของแคชเกินกว่าค่าที่กำหนด เราสามารถกำหนดลำดับความสำคัญของข้อมูลได้ดังนี้

ข้อมูล f จะถูกใส่เข้าไปในลำดับความสำคัญด้วยค่าคีย์ความสำคัญ $\Pr(f)$ ที่คำนวณได้ดังสมการ 3.3

$$\Pr(f) = Clock + Fr(f) \times \frac{Cost(f)}{Size(f)} \qquad (3.3)$$

โดย $Clock$ คือ ฟังก์ชัน Running Age [5][24][26] ที่มีค่าเริ่มต้นที่ 0 มีการปรับเปลี่ยนค่าสำหรับแต่ละการแทนที่ข้อมูล $f_{evicted}$ ให้กับคีย์ความสำคัญของข้อมูลในลำดับความสำคัญ: $\Pr(f_{evicted})$

$Fr(f)$ คือการนับความถี่การร้องขอข้อมูล f ถ้าข้อมูล f มีการ Hit (ข้อมูลถูกพบในแคชของเว็บแคชเซิร์ฟเวอร์) ค่า $Fr(f)$ จะถูกเพิ่มขึ้นมา 1 ค่า: $Fr(f) = Fr(f) + 1$ ถ้าข้อมูล f มีการ Miss (ข้อมูลไม่ถูกพบในแคชของเว็บแคชเซิร์ฟเวอร์) ค่า $Fr(f)$ จะถูกกำหนดค่าให้มีค่าเป็น 1: $Fr(f) = 1$

$Size(f)$ คือขนาดของข้อมูลในหน่วยไบต์

$Cost(f)$ คือค่าที่สัมพันธ์กับข้อมูล f เพื่อที่จะใช้นำข้อมูลมาไว้ในแคช ซึ่งได้อธิบายไว้ในเรื่องอัลกอริธึมการแทนที่ข้อมูลแบบ GD-Size

3.3.3.1 ขั้นตอนการทำงานของอัลกอริธึมการแทนที่ข้อมูลแบบ GDSF

จะสามารถแบ่งขั้นตอนการทำงานออกเป็น 2 กรณีได้ดังนี้

1. ถ้าข้อมูล f ที่ถูกร้องขอมีค่าเป็น Hit ในแคช ดังนั้นข้อมูลที่ร้องขอจึงถูกนำออกไปจากแคช และ
   - จำนวนของแคช $Used$ จะไม่มีการเปลี่ยนแปลง
   - การนับความถี่ของข้อมูล $Fr(f)$ จะเพิ่มขึ้นมา 1
   - คีย์ความสำคัญ $\Pr(f)$ จะถูกคำนวณใหม่อีกครั้งตามสมการที่ 3.3
   - ข้อมูล f ที่มีการคำนวณคีย์ความสำคัญ $\Pr(f)$ จะถูกนำไปใส่ในลำดับความสำคัญอีกครั้งเพื่อใช้สำหรับการทำงานครั้งใหม่

สิ้นสุดการทำงานของกรณีนี้

2. ถ้าข้อมูล f ที่ถูกร้องขอมีค่าเป็น Miss ภายในแคชดังนั้นข้อมูลที่ถูกร้องขอจะต้องถูกนำมาจากเซิร์ฟเวอร์ต้นทางและจะถูกเว็บแคชเซิร์ฟเวอร์สำเนาเก็บไว้ในแคชด้วย

- การนับความถี่ของข้อมูล $Fr(f)$ จะถูกตั้งค่าให้เป็น 1
- คีย์ความสำคัญ $\Pr(f)$ จะถูกคำนวณโดยใช้สมการที่ 3.3
- ข้อมูล f จะถูกนำไปเก็บไว้ในลำดับความสำคัญด้วยคีย์ความสำคัญ $\Pr(f)$ ที่คำนวณได้
- จำนวนของแคช $Used$ จะถูกคำนวณใหม่ดังสมการที่ 3.4

$$Used = Used + Size(f) \tag{3.4}$$

หลังจากนั้นจะเกิดเหตุการณ์ใดเหตุการณ์หนึ่งใน 2 เหตุการณ์นี้จะเกิดขึ้น

- ถ้า $Used \leq Total$ หมายความว่ามีเนื้อที่เพียงพอที่จะจัดเก็บข้อมูล f และไม่มีข้อมูลที่ต้องถูกแทนที่ ข้อมูล f จะถูกจัดเก็บลงในแคชของเว็บแคชเซิร์ฟเวอร์ และเสร็จสิ้นการทำงาน
- ถ้า $Used > Total$ หมายความว่า ไม่มีเนื้อที่เพียงพอที่จะจัดเก็บข้อมูล f และข้อมูลภายในแคชของเว็บแคชเซิร์ฟเวอร์บางส่วนจะต้องมีการถูกแทนที่

  เริ่มต้น เราระบุกลุ่มของข้อมูลจำนวนน้อยที่จะถูกลบออกไปจากแคชโดยใช้ขั้นตอนต่อไปนี้: ข้อมูล k ข้อมูล (k อาจมีค่าเป็น 1) ที่มีความสำคัญต่ำสุดในลำดับความสำคัญจะถูกเลือก $f_1, f_2, \ldots, f_k$ เพื่อให้เนื้อที่ $UsedEstimate \leq Total$ โดย

$$UsedEstimate = Used - \sum_{i=1}^{k} Size(f_i) \tag{3.5}$$

(a) ถ้าข้อมูลต้นฉบับ f ที่เราจะทำการเก็บไว้ ไม่อยู่ในกลุ่มข้อมูล $f_1, f_2, \ldots, f_k$ จะเกิดเหตุการณ์ดังนี้

i. ค่า $Clock$ (Running Age ของลำดับความสำคัญ) จะถูกคำนวณใหม่ด้วยสมการที่ 3.6

$$Clock = \max_{i=1}^{k} \Pr(f_i) = \Pr(f_k) \tag{3.6}$$

ii จำนวนของแคช $Used$ ได้รับการปรับค่าใหม่เป็น

$$Used = Used - \sum_{i=1}^{k} Size(f_i) \tag{3.7}$$

iii ข้อมูล f_1, f_2, ..., f_k ถูกนำออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์

iv ข้อมูล f ถูกนำไปเก็บไว้ในแคชของเว็บแคชเซิร์ฟเวอร์

(b) ถ้าข้อมูลต้นฉบับ f ที่เราจะทำการเก็บไว้อยู่ระหว่างข้อมูล f_1, f_2, ..., f_k ซึ่งต้องถูกนำออกไปเพื่อจัดเก็บข้อมูล f ดังนั้น

i ข้อมูล f จะไม่ถูกนำไปเก็บไว้ในแคชของเว็บแคชเซิร์ฟเวอร์และค่าคีย์ความสำคัญ $\Pr(f)$ จะถูกนำออกไปจากลำดับความสำคัญ

ii ไม่มีข้อมูลในแคชของเว็บแคชเซิร์ฟเวอร์ถูกลบออก

สิ้นสุดการทำงานของกรณีนี้

จุดหลักการเปลี่ยนแปลงที่นำเสนอในส่วนนี้คือการนำเสนอการนำความถี่ของการร้องขอ $Fr(f)$ ที่มีความสัมพันธ์กับข้อมูล f แต่ละตัว ที่ถูกนำไปเก็บไว้ในแคชของเว็บแคชเซิร์ฟเวอร์รวมกันในสมการที่ 3.3 สิ่งนี้จะสะท้อนให้เห็นรูปแบบการเข้าถึงข้อมูลและนำเสนอการปรับปรุงอัลกอริธึมการแทนที่ข้อมูลแบบ GD-Size เข้ากับเมตริกซ์ของประสิทธิภาพแบบต่าง ๆ พร้อมด้วยการเชื่อมการทำงานเข้ากับฟังก์ชั่น $\cos t$

ส่วนของอัลกอริธึมที่เกี่ยวข้องกับการลบข้อมูลออกจากแคชของเว็บแคชเซิร์ฟเวอร์ มีส่วนที่น่าสนใจสัมพันธ์กับกรณีที่เมื่อข้อมูลไม่ถูกเก็บไว้ในแคชเนื่องจากค่าคีย์มีค่าต่ำมากซึ่งได้ใส่ข้อมูลนี้ (กรณีที่ถูกเก็บไว้ในแคช) ไว้เป็นข้อมูลแรกสำหรับการแทนที่ โดยทั่วไปแล้วมันสามารถเกิดขึ้นได้เมื่อขนาดของไฟล์ใหญ่มาก กระบวนการที่นำเสนอจะห้ามไม่ให้มีกรณีเหล่านี้โดยอัตโนมัติเมื่อข้อมูลถูกเก็บลงในแคช

3.3.3.2 คุณสมบัติของอัลกอริธึมการแทนที่ข้อมูลแบบ GDSF

เมื่อพิจารณาคุณสมบัติของการแทนที่ข้อมูลแบบ GDSF จะพบว่าในกรณีนี้คีย์ความสำคัญสำหรับข้อมูล f จะถูกคำนวณได้ดังสมการ 3.8

$$\Pr(f) = Clock + (Fr(f) \times \frac{1}{Size(f)}) \tag{3.8}$$

ดังนั้น ข้อมูลที่มีการนับความถี่มากกว่าจะมีค่าของคีย์ความสำคัญที่มากกว่า และมีโอกาสที่จะอยู่ภายในแคชของเว็บแคชเซิร์ฟเวอร์มากกว่า เมื่อเปรียบเทียบกับข้อมูลที่ไม่ค่อยมีการร้องขอ GD-Size-Frequency กำหนดให้คีย์ความสำคัญที่มีค่าสูงให้กับข้อมูลที่มีขนาดเล็กเมื่อเปรียบเทียบกับข้อมูลที่ขนาดใหญ่ โดยมีจุดมุ่งหมายให้ค่า Hit ratio สูงสุดและค่า Miss ratio น้อยสุด เมื่อข้อมูลมีการถูกแทนที่

พารามิเตอร์ *Clock* มีค่าที่ค่อย ๆ เพิ่มขึ้น (มันจะถูกทำให้เพิ่มขึ้นเมื่อข้อมูลถูกแทนที่ออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์) ข้อมูลที่ไม่ได้รับการร้องขอมาเป็นระยะเวลานาน จะไม่มี

การเปลี่ยนค่าคีย์ภายในลำดับความสำคัญ เมื่อถึงจุดหนึ่ง ค่าของ *Clock* มีค่าสูงเพียงพอที่ข้อมูลใหม่ใด ๆ จะถูกนำเข้ามาเก็บหลังข้อมูล "เป็นระยะเวลานานที่ไม่ได้รับการร้องขอ" เหล่านี้ โดยวิธีนี้ข้อมูลที่ขนาดเล็กและข้อมูลที่มีการนับความถี่การร้องขอมากจะถูกทำการแทนที่ ถ้าข้อมูลเหล่านี้ไม่ได้รับการร้องขออีก วิธีการ Aging นี้จะป้องกันเว็บแคชเซิร์ฟเวอร์จากปัญหาที่กล่าวมา

อัลกอริธึมการแทนที่ข้อมูลที่มีคุณสมบัติที่คล้ายคลึงกันคืออัลกอริธึมการแทนที่ข้อมูล GD-Size-Frequency(packets) โดยมีจุดมุ่งหมายที่จะได้ค่าHit ratio และค่า Byte-Hit ratio มีค่าสูง โดยมีความแตกต่างเมื่อเทียบกับอัลกอริธึมการแทนที่ข้อมูลแบบ GD-Size-Frequency เพียงอย่างเดียวคือ GD-Size-Frequency(packets) จะไม่แบ่งแยกและปฏิเสธข้อมูลที่มีขนาดใหญ่

3.4 การพิจารณาอัลกอริธึมการแทนที่ข้อมูลแบบ Replacement Algorithm Selection Mechanism (RASM)

อัลกอริธึมการแทนที่ข้อมูลแบบ RASM มุ่งศึกษาถึงการเพิ่มค่าของ Byte-Hit Rate ของเว็บแคชเซิร์ฟเวอร์ ซึ่งค่าของ Byte-Hit Rate จะมีค่าสูงเมื่อมีการพบข้อมูลขนาดใหญ่ภายในแคชของเว็บแคชเซิร์ฟเวอร์เป็นจำนวนมาก จากคุณสมบัติของอัลกอริธึมการแทนที่ข้อมูลแบบ GDSF และ LFUDA ดังนี้

1. อัลกอริธึมการแทนที่ข้อมูลแบบ GDSF จะมีตัวแปรที่มีผลกับค่าการเลือกข้อมูลออกประกอบด้วย ขนาดของข้อมูล, จำนวนครั้งของการเรียกใช้ข้อมูลในแคช และค่าฟังก์ชัน Running Age ของข้อมูลที่ถูกเก็บไว้ในแคชดังสมการที่ 3.3 ซึ่งพบว่าในกรณีของข้อมูลที่มีค่า Running Age ของข้อมูลที่เก็บไว้ในแคชเท่ากัน ขนาดของข้อมูลจะเป็นตัวแปรสำคัญของค่าการเลือกข้อมูลออกไปจากแคชเพราะ เมื่อข้อมูลมีขนาดใหญ่ค่าการเลือกข้อมูลออกจะมีค่าน้อยกว่าเมื่อเทียบกับข้อมูลที่มีขนาดเล็ก ดังนั้นข้อมูลที่มีขนาดใหญ่จึงถูกนำออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์เป็นลำดับแรก

2. อัลกอริธึมการแทนที่ข้อมูลแบบ LFUDA จะมีตัวแปรที่มีผลกับค่าการเลือกข้อมูลออกประกอบด้วย จำนวนครั้งของการเรียกใช้ข้อมูลในแคช และค่า Running Age ของข้อมูลที่ถูกเก็บไว้ในแคชดังสมการที่ 3.1 ซึ่งพบว่าในกรณีของข้อมูลที่มีค่า Running Age ของข้อมูลที่เก็บไว้ในแคชเท่ากัน จำนวนครั้งของการเรียกใช้ข้อมูลในแคชจะเป็นตัวแปรสำคัญของการเลือกข้อมูลออกไปจากแคชเพราะ ข้อมูลที่ถูกเก็บไว้ในแคชและมีการเรียกใช้งานบ่อย จะมีค่าการเลือกข้อมูลออกสูงกว่าข้อมูลที่ถูกเก็บไว้ในแคชและมีการเรียกใช้งานน้อย ดังนั้นข้อมูลที่ถูกเรียกใช้งานน้อยจึงถูกนำออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์เป็นลำดับแรก

จึงมีการปรับปรุงอัลกอริธึมการแทนที่ข้อมูล ดังนี้

1. กำหนดค่า Threshold ซึ่งเป็นค่าที่ใช้สำหรับเปรียบเทียบกับขนาดของข้อมูลที่เว็บแคชเซิร์ฟเวอร์ได้รับมาจากเซิร์ฟเวอร์ต้นทางของเว็บไซต์

2. ข้อมูลที่มีขนาดเท่ากับหรือมากกว่าค่า Threshold ขึ้นไปจะพิจารณาการนำข้อมูลออกจากแคชโดยใช้อัลกอริธึมการแทนที่ข้อมูลแบบ LFUDA เนื่องจากข้อมูลที่มีขนาดใหญ่และมีการเรียกใช้งานบ่อย ๆ ควรจะถูกนำมาเก็บไว้ในแคชและข้อมูลขนาดใหญ่ที่ไม่มีการเรียกใช้งานบ่อย ๆ ก็ควรนำออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์

3. ข้อมูลที่มีขนาดน้อยกว่าค่า Threshold ลงมาจะถูกพิจารณาการนำข้อมูลออกจากแคชโดยการใช้อัลกอริธึมการแทนที่ข้อมูลแบบ GDSF เนื่องจากการแทนที่ข้อมูลแบบ GDSF จะพิจารณาถึงข้อมูลที่มีขนาดเล็กและมีการเรียกใช้งานบ่อย ๆ เป็นหลัก

จากขั้นตอนการปรับแต่งอัลกอริธึมการแทนที่ข้อมูลที่นำเสนอ สามารถนำมาเขียนสมการของของค่า คีย์ความสำคัญ ของข้อมูลแต่ละตัวได้ดังสมการที่ 3.9

$$K_i = \begin{cases} \left(\frac{F_i * C}{S_i}\right) + L & \text{เมื่อ } S_i < \text{Threshold} \\ (F_i * C) + L & \text{เมื่อ } S_i \geq \text{Threshold} \end{cases} \qquad (3.9)$$

ในการใช้งานอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งจะทำการกำหนดให้ค่าของ C มีค่าเป็น 1 เพื่อให้ความถี่ของการร้องขอและอายุของข้อมูลที่ถูกเก็บไว้ในแคชของเว็บแคชเซิร์ฟเวอร์มีความสำคัญที่เท่าเทียมกันดังนั้นจากสมการที่ 3.9 จะสามารถเขียนได้ใหม่เป็นดังสมการที่ 3.10

เมื่อวิเคราะห์สมการที่ 3.9 พบว่า ค่าของ F_i ของอัลกอริธึมการแทนที่ข้อมูลแบบ GDSF และอัลกอริธึมการแทนที่ข้อมูลแบบ LFUDA จะเริ่มต้นจากค่า 1 และเพิ่มขึ้นเรื่อย ๆ เมื่อมีการร้องขอซ้ำจากไคลเอนต์

เมื่อนำมาใช้กับอัลกอริธึมการแทนที่ข้อมูลที่ทำการปรับแต่งพบว่า $1 \leq S_i < \alpha$ ดังนั้นเมื่อพิจารณากรณีที่ $S_i <$ Threshold ในกรณีที่เว็บแคชเซิร์ฟเวอร์ทำการเก็บข้อมูลลงไปในแคชครั้งแรกจะทำให้ $0 < F_i/S_i \leq 1$ ซึ่งเมื่อนำมาเปรียบเทียบกับค่า F_i ในกรณีที่ $S_i \geq$ Threshold จะพบว่าค่าของ F_i จะมีค่ามากกว่าค่าของ F_i/S_i เสมอ ดังนั้นเมื่อนำมารวมกับค่า L จะทำให้ค่าของ K_i ในกรณี ที่ $S_i \geq$ Threshold มีค่ามากกว่า ค่า K_i ในกรณีที่ $S_i <$ Threshold เสมอ ทำให้โอกาสที่ข้อมูลที่มีขนาดมากกว่าค่า Threshold จะถูกนำออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์มีค่าเป็นศูนย์

ดังนั้นเพื่อแก้ไขปัญหาที่เกิดขึ้น ในกรณี $S_i \geq$ Threshold จึงได้ทำการปรับค่าความถี่ของการนับข้อมูลที่ถูกร้องขอจากค่าเดิมที่จะเริ่มนับตั้งแต่ค่า 1 เปลี่ยนให้เริ่มนับจากค่า 0 แทน

เพราะว่าเมื่อค่าความถี่ของการนับข้อมูลที่ถูกร้องขอมีค่าเริ่มตั้งแต่ 0 จะทำให้ค่า K_i ที่ได้เมื่อนำข้อมูลมาเก็บไว้ในแคชครั้งแรกจะมีค่าเป็นค่า L ทำให้ข้อมูลที่มีขนาดมากกว่าหรือเท่ากับค่า Threshold มีโอกาสที่จะถูกนำออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์ และในอัลกอริธึมการแทนที่ข้อมูลแบบ GDSF ค่าของคีย์ความสำคัญจะเกี่ยวข้องกับค่าขนาดของข้อมูลดังนั้นหากให้ค่าความถี่ของการร้องขอเป็น 0 จะเป็นผลให้ค่าคีย์ความสำคัญเริ่มต้นมีค่าเป็น 0 จำนวนมากเป็นผลให้ค่าของฟังก์ชั่น L ไม่มีการเปลี่ยนค่า เพื่อแก้ปัญหาที่เกิดขึ้นจึงกำหนดให้ค่าความถี่ของการนับข้อมูลสำหรับกรณี S_i < Threshold เริ่มนับตั้งแต่ 1 และในการใช้งานอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งจะทำการกำหนดให้ค่าของ C มีค่าเป็น 1 เพื่อให้ความถี่ของการร้องขอและค่าฟังก์ชันอายุของข้อมูลที่ถูกเก็บไว้ในแคชของเว็บแคชเซิร์ฟเวอร์มีความสำคัญที่เท่าเทียมกันดังนั้นจากสมการที่ 3.9 จะสามารถเขียนได้ใหม่เป็นดังสมการที่ 3.10

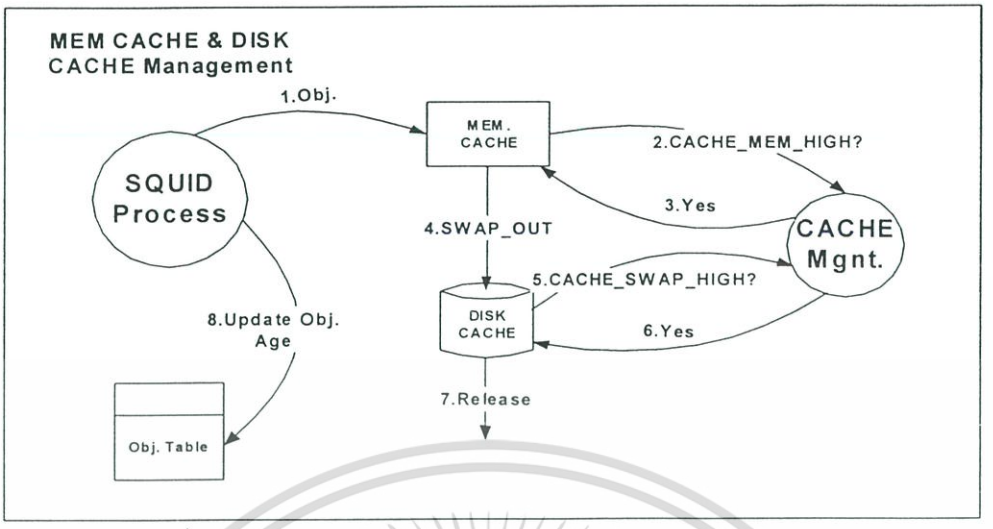
$$K_i = \begin{cases} \left(\frac{F_i}{S_i}\right) + L & \text{เมื่อ } S_i < \text{Threshold} \\ F_i + L & \text{เมื่อ } S_i \geq \text{Threshold} \end{cases} \qquad (3.10)$$

โดย K_i คือค่าคีย์สำหรับนำมาพิจารณาข้อมูลที่จะนำออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์

F_i คือจำนวนครั้งหรือความถี่ของการร้องขอข้อมูลในแคชของเว็บแคชเซิร์ฟเวอร์ โดยจะมีค่าเริ่มตั้งแต่ 0 และเพิ่มขึ้นครั้งละ 1 เมื่อมีการร้องขอซ้ำ สำหรับกรณี $S_i \geq$ Threshold และมีค่าเริ่มตั้งแต่ 1 และเพิ่มขึ้นครั้งละ 1 เมื่อมีการร้องขอซ้ำ สำหรับกรณี $S_i <$ Threshold

L คือฟังก์ชั่น Running Age [5][6][24][26] ซึ่งจะมีค่าเริ่มต้นที่ 0 และจะมีการปรับเปลี่ยนค่าเมื่อมีการนำข้อมูล f ออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์: $L = K_f$ และค่า L ที่มีการปรับเปลี่ยนจะถูกนำไปใส่ให้กับค่าคีย์ (K) ของข้อมูลใหม่ที่เข้ามาแทนที่

จากรูปที่ 3.1 เราจะพิจารณาถึงขั้นตอนการทำงานของเว็บแคชเซิร์ฟเวอร์ในขั้นตอนการจัดเก็บข้อมูลลงในแคชของเว็บแคชเซิร์ฟเวอร์ โดยจะมีขั้นตอนการทำงานใหญ่ 2 ขั้นตอนคือขั้นตอนการจัดเก็บข้อมูลลงในแคชและขั้นตอนการเรียกใช้งานอัลกอริธึมการแทนที่ข้อมูลดังแสดงได้ในขั้นตอนที่ 1 จนถึงขั้นตอนที่ 8 ของรูปที่ 3.1



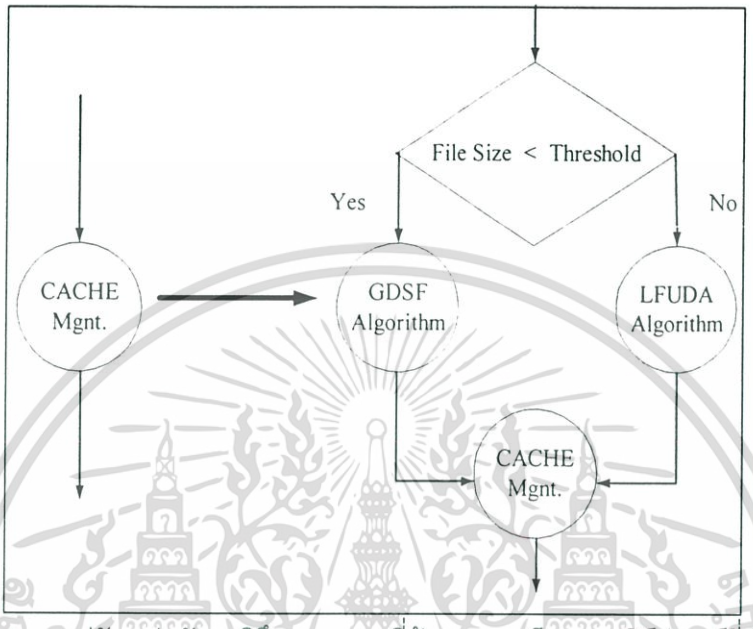
รูปที่ 3.1 แสดงการจัดเก็บข้อมูลในแคชของเว็บแคชเซิร์ฟเวอร์

เมื่อพิจารณาเฉพาะในส่วนของการเรียกใช้อัลกอริธึมการแทนที่ข้อมูล (ฟังก์ชั่น CACHE Mgnt. ภายในรูปที่ 3.1) จะแสดงได้ในขั้นตอนที่ 2 จนถึงขั้นตอนที่ 7 โดยจะแบ่งขั้นตอนการทำงานออกเป็น 2 ส่วนคือส่วนของ Memory Cache โดย Memory Cache เป็นแคชของเว็บแคชเซิร์ฟเวอร์ที่สร้างจากแรมของเครื่องเซิร์ฟเวอร์และ ส่วนของDisk Cache โดย Disk Cache เป็นแคชของเว็บแคชเซิร์ฟเวอร์ที่สร้างจากฮาร์ดดิสก์ของเครื่องเซิร์ฟเวอร์

1. Memory Cache เป็นการเรียกใช้อัลกอริธึมการแทนที่ข้อมูลสำหรับการเลือกข้อมูลที่จะถูกย้ายจาก Memory Cache ลงไปเก็บไว้ใน Disk Cache ของเว็บแคชเซิร์ฟเวอร์โดยจะมีการทำงานตั้งแต่ขั้นตอนที่ 2 จนถึงขั้นตอนที่ 4 ของรูปที่ 3.1 ซึ่งในขั้นตอนที่ 2 เว็บแคชเซิร์ฟเวอร์จะตรวจสอบขนาดของข้อมูลที่อยู่ใน Memory Cache มีค่าเกินกว่าค่าที่กำหนดหรือไม่ หากมีค่าเกินจะเรียกใช้อัลกอริธึมการแทนที่ข้อมูล และนำข้อมูลที่ถูกเลือกออกไปเก็บไว้ใน Disk Cache ในขั้นตอนที่ 4 จากนั้นจะย้อนกลับไปทำงานในขั้นตอนที่ 2 ถึงขั้นตอนที่ 4 จนกระทั่งค่าของขนาดข้อมูลใน Memory Cache มีค่าน้อยกว่าค่าที่กำหนด

2. Disk Cache เป็นการเรียกใช้อัลกอริธึมการแทนที่ข้อมูลสำหรับการเลือกข้อมูลที่จะถูกนำออกไปจากDisk Cache ของเว็บแคชเซิร์ฟเวอร์โดยจะมีขั้นตอนการทำงานตั้งแต่ขั้นตอนที่ 5 จนถึงขั้นตอนที่ 7 ของรูปที่ 3.1 ซึ่งในขั้นตอนที่ 5 เว็บแคชเซิร์ฟเวอร์จะตรวจสอบขนาดของข้อมูลที่อยู่ใน Disk Cache มีค่าเกินกว่าค่าที่กำหนดหรือไม่ หากมีค่าเกิน จะเรียกใช้อัลกอริธึมการแทนที่ข้อมูล และนำข้อมูลที่ถูกเลือกออกไปจาก Disk Cache ในขั้นตอนที่ 7 จากนั้นจะย้อนกลับไปทำงานในขั้นตอนที่ 5 ถึงขั้นตอนที่ 7 จนกระทั่งค่าของขนาดข้อมูลใน Disk Cache มีค่าน้อยกว่าค่าที่กำหนด

ในการปรับแต่งอัลกอริธึมการแทนที่ข้อมูลของการทดสอบจะเข้าไปปรับแต่งในส่วนฟังก์ชั่นที่ชื่อว่า CACHE Mgnt. ที่แสดงในรูปที่ 3.1 ซึ่งเป็นฟังก์ชั่นที่ทำหน้าที่เป็นอัลกอริธึมการแทนที่ข้อมูลของเว็บแคชเซิร์ฟเวอร์โดยการปรับแต่งแสดงได้ดังรูปที่ 3.2



รูปที่ 3.2 การปรับแต่งอัลกอริธึมการแทนที่ข้อมูลของเว็บแคชเซิร์ฟเวอร์ที่นำเสนอ

บทที่ 4
การวิเคราะห์ประสิทธิภาพของอัลกอริธึมการแทนที่ข้อมูล

การวิเคราะห์ประสิทธิภาพของอัลกอริธึมการแทนที่ข้อมูลของเว็บแคชเซิร์ฟเวอร์ที่ได้ทำการปรับแต่งเริ่มตั้งแต่การวิเคราะห์อัลกอริธึมการแทนที่ข้อมูลที่นำมาพิจารณา, การปรับแต่งอัลกอริธึมการแทนที่ข้อมูลของเว็บแคชเซิร์ฟเวอร์, การวัดประสิทธิภาพของอัลกอริธึมการแทนที่ข้อมูลที่ทำการปรับแต่งเปรียบเทียบกับอัลกอริธึมเดิม, การจำลองการทำงานของอัลกอริธึมการแทนที่ข้อมูล, ผลการทดสอบ และการวิเคราะห์การทำงานของอัลกอริธึมการแทนที่ข้อมูลที่ถูกปรับแต่ง และการวิเคราะห์หาค่า threshold ที่เหมาะสมสำหรับการปรับแต่งอัลกอริธึมการแทนที่ข้อมูล

4.1 การวิเคราะห์อัลกอริธึมการแทนที่ข้อมูลที่นำมาพิจารณา

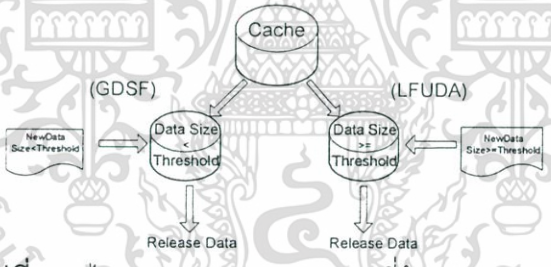
ในหัวข้อที่ 3.4 ของวิทยานิพนธ์นี้ได้พิจารณาถึงการทำงานของอัลกอริธึมการแทนที่ข้อมูลแบบ RASM ซึ่งมีการกำหนดค่า threshold ขึ้นมาเพื่อทำการเลือกใช้อัลกอริธึม 2 ชนิดคือ อัลกอริธึมการแทนที่ข้อมูลแบบ GDSF และ อัลกอริธึมการแทนที่ข้อมูลแบบ LFUDA ดังสมการที่ 3.10 โดยเมื่อนำอัลกอริธึมทั้ง 2 ชนิดมาใช้งานร่วมกันจึงได้มีการปรับเปลี่ยน และกำหนดค่าเริ่มต้นในการทำงานของอัลกอริธึมเพื่อให้เกิดความเหมาะสมมากยิ่งขึ้น โดยพิจารณาที่ค่าความถี่ F_i เนื่องจากถ้าค่า F_i ของทั้งกรณีที่ S_i < Threshold และกรณีที่ $S_i \geq$ Threshold เริ่มต้นที่ 1 เท่ากันจะทำให้โอกาสที่ข้อมูลที่มีขนาดมากกว่าค่า threshold จะถูกนำออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์มีค่าเป็นศูนย์ เพราะค่าของ K_i ในกรณีที่ $S_i \geq$ Threshold มีค่ามากกว่า ค่า K_i ในกรณีที่ S_i < Threshold เสมอ จึงได้ทำการปรับค่าความถี่ F_i ของการนับข้อมูลที่ถูกร้องขอจากค่าเดิมที่จะเริ่มนับตั้งแต่ค่า 1 เปลี่ยนให้เริ่มนับจากค่า 0 แทน เพราะว่าเมื่อค่าความถี่ของการนับข้อมูลที่ถูกร้องขอมีค่าเริ่มตั้งแต่ 0 จะทำให้ค่า K_i ที่ได้เมื่อนำข้อมูลมาเก็บไว้ในแคชครั้งแรกจะมีค่าเป็นค่า L ทำให้ข้อมูลที่มีขนาดมากกว่าหรือเท่ากับค่า threshold มีโอกาสที่จะถูกนำออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์

นอกจากนี้การพิจารณาถึงคุณสมบัติของอัลกอริธึมการแทนที่ข้อมูลแบบ RASM เมื่อแคชเก็บข้อมูลเต็ม และมีข้อมูลใหม่เข้ามา จะมีการแทนที่ข้อมูลในแคชของเว็บแคชเซิร์ฟเวอร์ โดยพิจารณาจากข้อมูลที่มีค่า priority key (K_i) ต่ำสุดจะถูกนำออกไปจากแคชเรื่อยๆจนกว่าขนาดพื้นที่ว่างภายในแคชเหลือพอที่จะใส่ข้อมูลใหม่ลงไปได้ ซึ่งการแทนที่ข้อมูลภายในแคชนี้จะไม่สนใจว่าข้อมูลที่ถูกเอาออกไปเป็นข้อมูลในกลุ่มที่มีขนาด (S_i) มากกว่าหรือน้อยกว่าค่า threshold

ตัวอย่างเช่น ข้อมูลใหม่ที่จะนำเข้ามามีขนาดมากกว่า threshold อาจจะเข้ามาแทนที่ข้อมูลภายในแคชที่มีขนาดน้อยกว่า threshold จำนวนหลายๆไฟล์ ซึ่งการแทนที่ข้อมูลในลักษณะนี้อาจดูไม่ยุติธรรมถ้าการหาค่า priority key (K_i) ของข้อมูลระหว่างกรณีที่ S_i < Threshold (สมการ3.10 บน) และกรณีที่ $S_i \geq$ Threshold (สมการ 3.10 ล่าง) อยู่ในสัดส่วนที่ไม่เหมาะสมกัน โดยค่า K_i ของสมการ 3.10 บนอาจจะน้อยกว่าค่า K_i ที่ได้จากสมการล่างมาก ทั้งๆที่ข้อมูลในสมการบน (S_i < threshold) อาจถูกเรียกใช้บ่อยครั้งกว่าข้อมูลในสมการล่าง ($S_i \geq$ threshold) ทำให้ข้อมูลที่มีขนาดน้อยกว่า threshold (S_i < threshold) ถูกแทนที่ด้วยข้อมูลใหม่ที่มากกว่า threshold ได้

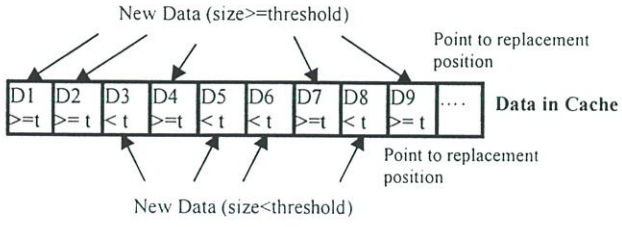
4.2 การปรับแต่งอัลกอริธึมการแทนที่ข้อมูลที่นำมาพิจารณา

จากการพิจารณาคุณสมบัติของอัลกอริธึมการแทนที่ข้อมูล RASM เราได้ทำการปรับเปลี่ยนอัลกอริธึม RASM และเรียกอัลกอริธึมที่ทำการปรับแต่งใหม่นี้ว่า MRASM ดังหัวข้อ 1.2 ที่ได้กล่าวไปแล้ว โดยข้อมูลที่เข้ามาถ้า $S_i >$ threshold ก็นำไปแทนที่ข้อมูลภายในแคชในกลุ่มข้อมูลที่มีขนาดมากกว่า threshold เหมือนกัน ส่วนข้อมูลที่เข้ามามีขนาดน้อยกว่า threshold ก็นำไปแทนที่ข้อมูลภายในแคชในกลุ่มข้อมูลที่มีขนาดน้อยกว่า threshold ดังแสดงในรูปที่ 4.1



รูปที่ 4.1 ไดอะแกรมแสดงการแทนที่ข้อมูลของอัลกอริธึม MRASM

เราสามารถแสดงลักษณะการแทนที่ข้อมูลจากรูปที่ 4.1 โดยแสดงถึงข้อมูลใหม่ที่เข้ามาในแคชจะถูกนำไปแทนที่ข้อมูลที่อยู่ภายในแคชที่มีค่าความสำคัญของข้อมูลต่ำที่สุดและมีขนาดข้อมูลอยู่ในกลุ่มขนาดเดียวกันดังแสดงในรูปที่ 4.2



รูปที่ 4.2 แสดงการแทนที่ข้อมูลในแคชของอัลกอริธึม MRASM

ซึ่งสามารถเขียนเป็นสมการของอัลกอริธึมการแทนที่ข้อมูล MRASM ได้ดังสมการที่ 4.1 ดังนี้

$$K_i = \begin{cases} \left(\frac{F_i}{S_i}\right) + L & \text{If } S_i < \text{Threshold} \\ F_i + L & \text{If } S_i \geq \text{Threshold} \end{cases} \qquad (4.1)$$

โดย K_i คือค่าคีย์ในการพิจารณาข้อมูลที่จะนำออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์

S_i คือขนาดของข้อมูล (ไบต์)

F_i คือจำนวนครั้งหรือความถี่ของการร้องขอข้อมูลในแคชของเว็บแคชเซิร์ฟเวอร์

L คือฟังก์ชั่น Running Age [5][6][24][26] ซึ่งจะมีค่าเริ่มต้นที่ 0 และจะมีการปรับเปลี่ยนค่าเมื่อมีการนำข้อมูล f ออกไปจากแคชของเว็บแคชเซิร์ฟเวอร์: $L = K_i$ และค่า L ที่มีการปรับเปลี่ยนจะถูกนำไปใส่ให้กับค่าคีย์ (K) ของข้อมูลใหม่ที่เข้ามาแทนที่

โดยมีเงื่อนไขการแทนที่ข้อมูลดังนี้

$$f_{i\_new} \text{ ®> } f_{i\_cache}$$

โดยที่

$[(S_{i\_new} \geq \text{Threshold}) \text{ and } (S_{i\_cache} \geq \text{Threshold})] \text{ or } [(S_{i\_new} < \text{Threshold}) \text{ and } (S_{i\_cache} < \text{Threshold})]$

เมื่อ ®> แทนสัญลักษณ์การแทนที่ข้อมูลโดยนำข้อมูลด้านซ้ายไปแทนที่ข้อมูลด้านขวา

f_{i_new} คือข้อมูลใหม่ที่เข้ามาในแคช

f_{i_cache} คือข้อมูลในแคชที่จะถูกนำออกไป

S_{i_new} คือขนาดข้อมูลใหม่ที่เข้ามาในแคช (ไบต์)

S_{i_cache} คือขนาดข้อมูลที่จะถูกนำออกไป (ไบต์)

4.3 การวัดประสิทธิภาพของเว็บแคชเซิร์ฟเวอร์

เว็บแคชเซิร์ฟเวอร์เป็นระบบที่ทำหน้าที่รับการร้องขอข้อมูลจากไคลเอนต์ และจะทำการตรวจสอบข้อมูลที่อยู่ภายในแคชของเว็บแคชเซิร์ฟเวอร์ ถ้าหากข้อมูลที่ไคลเอนต์ร้องขอ ถูกพบภายในแคช และข้อมูลยังไม่หมดอายุ (ตามช่วงอายุของข้อมูลที่ได้ตั้งค่าเอาไว้) ก็จะทำการส่งข้อมูลกลับไปยังให้ไคลเอนต์ แต่หากข้อมูลที่ไคลเอนต์ร้องขอไม่ถูกพบในแคชหรือข้อมูลถูกพบในแคชแต่หมดอายุ เว็บแคชเซิร์ฟเวอร์จะติดต่อไปยังเซิร์ฟเวอร์ต้นทางของเว็บไซต์ที่ไคลเอนต์ร้องขอเพื่อให้เซิร์ฟเวอร์ต้นทางส่งข้อมูลกลับมา เมื่อเว็บแคชเซิร์ฟเวอร์ได้รับข้อมูล ก็จะส่งข้อมูลต่อไปยังไคลเอนต์ที่ร้องขอข้อมูลพร้อมกับทำการสำเนาข้อมูลลงไปในแคชของเว็บแคชเซิร์ฟเวอร์ด้วย จาก

หลักการทำงานของเว็บแคชเซิร์ฟเวอร์ที่กล่าวมา จะแสดงให้เห็นถึงค่า 2 ค่าที่ถูกนิยมใช้ในการพิจารณาถึงประสิทธิภาพการทำงานของเว็บแคชเซิร์ฟเวอร์คือ

1. การพบข้อมูลของเว็บแคชเซิร์ฟเวอร์ (Hit) คือกรณีข้อมูลที่ไคลเอนต์ร้องขอมีการค้นพบภายในแคชของเว็บแคชเซิร์ฟเวอร์ โดยพิจารณาจากข้อมูลภายในไฟล์ Log ของเว็บแคชเซิร์ฟเวอร์ในส่วนของฟิลด์ Log Tag/HTTP Code ซึ่งค่าภายในฟิลด์ Log Tag/HTTP Code ที่นำมาพิจารณาว่าเป็นการพบข้อมูลของเว็บแคชเซิร์ฟเวอร์ (Hit) ประกอบด้วยค่า TCP_HIT, TCP_MEM_HIT, TCP_REFRESH_HIT, TCP_REF_FAIL_HIT และ TCP_IMS_HIT

2. การไม่พบข้อมูลของเว็บแคชเซิร์ฟเวอร์ (Miss) คือกรณีข้อมูลที่ไคลเอนต์ร้องขอไม่ถูกค้นพบภายในแคชของเว็บแคชเซิร์ฟเวอร์หรือข้อมูลที่อยู่ในแคชของเว็บแคชเซิร์ฟเวอร์หมดอายุแล้วทำให้เว็บแคชเซิร์ฟเวอร์ต้องทำการติดต่อไปยังเซิร์ฟเวอร์ของเว็บไซต์ต้นทางของเว็บไซต์เพื่อนำข้อมูลมาใหม่อีกครั้งหนึ่ง ค่าภายในฟิลด์ Log Tag/HTTP Code ที่นำมาพิจารณาว่าเป็นการไม่พบข้อมูลของเว็บแคชเซิร์ฟเวอร์ (Miss) ประกอบด้วยค่า TCP_MISS, TCP_REFRESH_MISS, TCP_CLIENT_REFRESH , TCP_IMS_MISSม TCP_SWAPFAIL และ TCP_DENIED

ในการพิจารณาประสิทธิภาพของเว็บแคชเซิร์ฟเวอร์ สิ่งที่ได้รับความนิยมในการนำมาวัดประสิทธิภาพของเว็บแคชเซิร์ฟเวอร์คือการพบข้อมูลภายในแคชของเว็บแคชเซิร์ฟเวอร์ ซึ่งจากการค้นพบข้อมูลภายในแคชของเว็บแคชเซิร์ฟเวอร์สามารถแสดงเป็นค่าสำหรับการวิเคราะห์จำนวน 2 ค่าคือ

1.อัตราการพบข้อมูล (Hit Rate) คืออัตราส่วนของจำนวนการร้องขอที่พบข้อมูลในแคชเปรียบเทียบกับการร้องขอทั้งหมดที่เว็บแคชเซิร์ฟเวอร์ได้รับจากไคลเอนต์ และค่าอัตราการพบข้อมูลเมื่อนำไปคูณกับค่า 100 จะถูกเรียกว่า เปอร์เซ็นต์การพบข้อมูล (Hit Rate)

2. อัตราการพบข้อมูลแบบไบต์ (Byte-Hit Rate) คืออัตราส่วนปริมาณข้อมูลแบบไบต์ของจำนวนการร้องขอที่พบข้อมูลในแคชเปรียบเทียบกับปริมาณข้อมูลแบบไบต์ของการร้องขอทั้งหมดที่เว็บแคชเซิร์ฟเวอร์ได้รับจากไคลเอนต์ และค่าอัตราการพบข้อมูลแบบไบต์เมื่อนำไปคูณกับค่า 100 จะถูกเรียกว่าเปอร์เซ็นต์การพบข้อมูลแบบไบต์ (Byte-Hit Rate)

4.4 การจำลองการทำงานของเว็บแคชเซิร์ฟเวอร์

การจำลองการทำงานของเว็บแคชเซิร์ฟเวอร์ในฟังก์ชั่นของอัลกอริธึมการแทนที่ข้อมูลเป็นกระบวนการทดสอบ การทำงานของอัลกอริธึมการแทนที่ข้อมูลแบบ MRASM เปรียบเทียบกับอัลกอริธึมการแทนที่ข้อมูล RASM, GDSF และ LFUDA และเพื่อหลีกเลี่ยงการได้รับผลกระทบจากตัวแปรที่ไม่เกี่ยวข้องและมีส่วนที่ทำให้ค่าของ Hit Rate และ Byte-Hit Rate ผิดพลาดไปจากที่

ต้องการ เช่น ค่า latency ของระบบเครือข่าย, การได้รับการร้องขอที่นอกเหนือจากที่กำหนด และการล่าช้าในการประมวลผลของเว็บแคชเซิร์ฟเวอร์

4.4.1 เครื่องมือสำหรับการจำลองการทำงานของเว็บแคชเซิร์ฟเวอร์

เครื่องมือใช้สำหรับการจำลองการทำงานของอัลกอริธึมการแทนที่ข้อมูลจะมีส่วนประกอบดังนี้

1. เครื่องไมโครคอมพิวเตอร์ ใช้ซีพียู Pentium4 3.0 GHz หน่วยความจำ 512 MB ฮาร์ดดิสก์ความจุ 80 GB ระบบปฏิบัติการที่ใช้คือ Microsoft Windows XP Professional

2. โปรแกรม Gawk ซึ่งนำมาใช้งานสำหรับแปลงข้อมูลที่นำมาทำการทดสอบให้อยู่ในรูปแบบที่เหมาะสมสำหรับการจำลองการทำงานของเว็บแคชเซิร์ฟเวอร์ในฟังก์ชั่นของอัลกอริธึมการแทนที่ข้อมูล

3. โปรแกรม Matlab เวอร์ชั่น 6.5 นำมาใช้ในการจำลองผลการทำงานของอัลกอริธึมการแทนที่ข้อมูลที่ได้ทำการปรับปรุง และเปรียบเทียบกับอัลกอริธึมเดิม พร้อมทั้งแสดงผลออกมาในรูปแบบกราฟเปรียบเทียบ

4. โปรแกรม Microsoft Excel XP2003 นำมาใช้ในการแสดงผลเปรียบเทียบในรูปแบบกราฟ

4.4.2 ข้อมูลที่นำมาทดสอบ

ข้อมูลที่นำมาทดสอบกับการทดสอบการทำงานของเว็บแคชเซิร์ฟเวอร์ ประกอบด้วยข้อมูลจำนวน 2 ชุดข้อมูลโดยมีรายละเอียดดังนี้

1. ข้อมูลไฟล์ Log ที่ได้จากการรวบรวมภายในระยะเวลา 2 สัปดาห์ของเว็บเซิร์ฟเวอร์ของ Clarknet (ClarkNet WWW server) ประกอบด้วยการร้องขอข้อมูลทั้งหมด 1,465,049 ครั้ง ซึ่งการร้องขอทั้งหมดจะเป็นข้อมูลที่เป็น unique จำนวน 35,356 ชุด โดยข้อมูลไฟล์ Log จากเว็บเซิร์ฟเวอร์ของ Clarknet สามารถดาวน์โหลดจากเว็บไซต์ http://ita.ee.lbl.gov/html/contrib/Clarknet-HTTP.html [29]

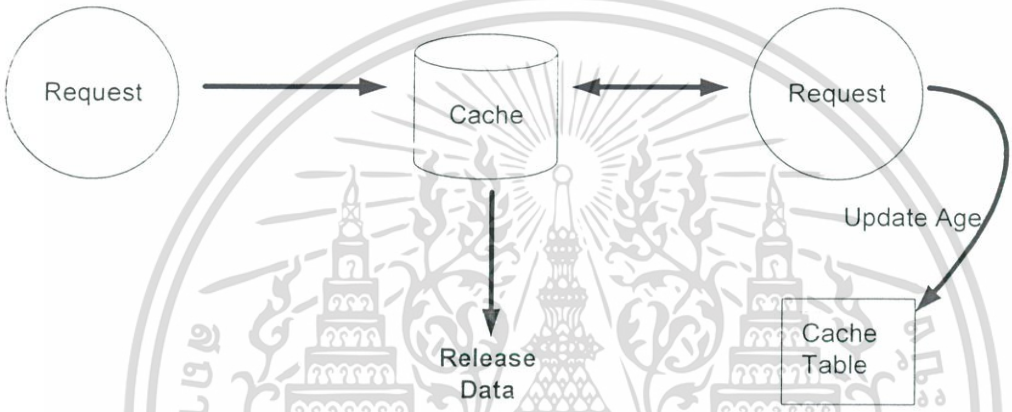
2. ข้อมูลไฟล์ Log ที่ได้จากการรวบจากเว็บเซิร์ฟเวอร์ของ Nasa (NASA Kennedy Space Center WWW server) ประกอบด้วยการร้องขอข้อมูลทั้งหมด 1,387,398 ครั้ง ซึ่งการร้องขอทั้งหมดจะเป็นข้อมูลที่เป็น unique จำนวน 22,531ชุด โดยข้อมูลไฟล์ Log จากเว็บเซิร์ฟเวอร์ของ Nasa สามารถดาวน์โหลดจากเว็บไซต์ http://ita.ee.lbl.gov/html/contrib/Nasa-HTTP.html [29]

3. ข้อมูลไฟล์ Log ที่ได้จากการรวบจากเว็บเซิร์ฟเวอร์ของ Calgory (University of Calgary's Department of Computer Science WWW server) ประกอบด้วยการร้องขอข้อมูล

ทั้งหมด 567,132 ครั้ง ซึ่งการร้องขอทั้งหมดจะเป็นข้อมูลที่เป็น unique จำนวน 30,822 ชุด โดยข้อมูลไฟล์ Log จากเว็บเซิร์ฟเวอร์ของ Calgory สามารถดาวน์โหลดจากเว็บไซต์ http://ita.ee.lbl.gov/html/contrib/Calgary-HTTP.html [29]

4.4.3 โมเดลของการจำลองการทำงานของเว็บแคชเซิร์ฟเวอร์ในฟังก์ชั่นของอัลกอริธึมการแทนที่ข้อมูล

โมเดลของการจำลองการทำงานของเว็บแคชเซิร์ฟเวอร์ในฟังก์ชั่นของอัลกอริธึมการแทนที่ข้อมูล ดังแสดงในรูปที่ 4.3



รูปที่ 4.3 โมเดลของการจำลองการทำงานของเว็บแคชเซิร์ฟเวอร์ในฟังก์ชั่นของอัลกอริธึมการแทนที่ข้อมูล

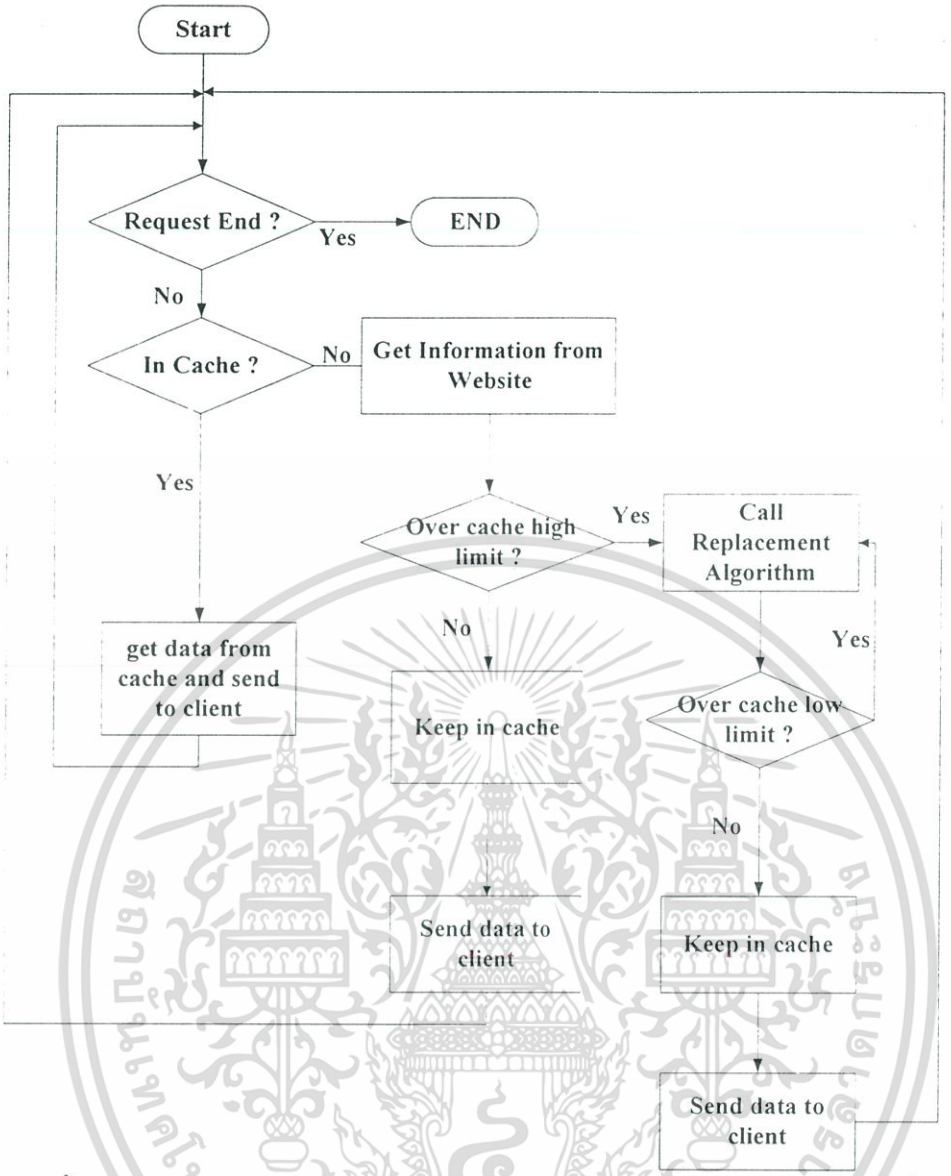
จากรูปที่ 4.3 การทำงานจะเริ่มจาก

- กำหนดแคชของเว็บแคชเซิร์ฟเวอร์ โดยมีตัวแปรที่เกี่ยวข้องคือ ขนาดของแคช, ชนิดของอัลกอริธึมการแทนที่ข้อมูลที่ใช้, ค่า Higher limit ของแคช, ค่า Lower limit ของแคช และค่า Threshold สำหรับอัลกอริธึมการแทนที่ข้อมูลที่นำเสนอ

- ทำการส่งการร้องขอที่ได้จากการข้อมูลทดสอบแต่ละชุดเข้าที่ละการร้องขอ

-ตรวจสอบข้อมูลในแคช หากพบจะทำการอัพเดทค่าคีย์ความสำคัญ และรับการร้องขอต่อไป หากไม่พบจะทำการตรวจสอบขนาดของแคช ถ้าเกินค่าที่กำหนดจะเรียกใช้อัลกอริธึมการแทนที่ข้อมูลเพื่อนำข้อมูลออกไปจากแคชและทำการอัพเดทค่าอายุข้อมูลจนกระทั่งขนาดของแคชลดลงถึงค่าที่กำหนดจะหยุดเรียกใช้อัลกอริธึมการแทนที่ข้อมูลพร้อมกับเก็บข้อมูลลงในแคช และรับการร้องขอต่อไป

ขั้นตอนการทำงานของการจำลองการทำงานของเว็บแคชเซิร์ฟเวอร์ในฟังก์ชั่นของอัลกอริธึมการแทนที่ข้อมูลจะแสดงได้ดังรูปที่ 4.4

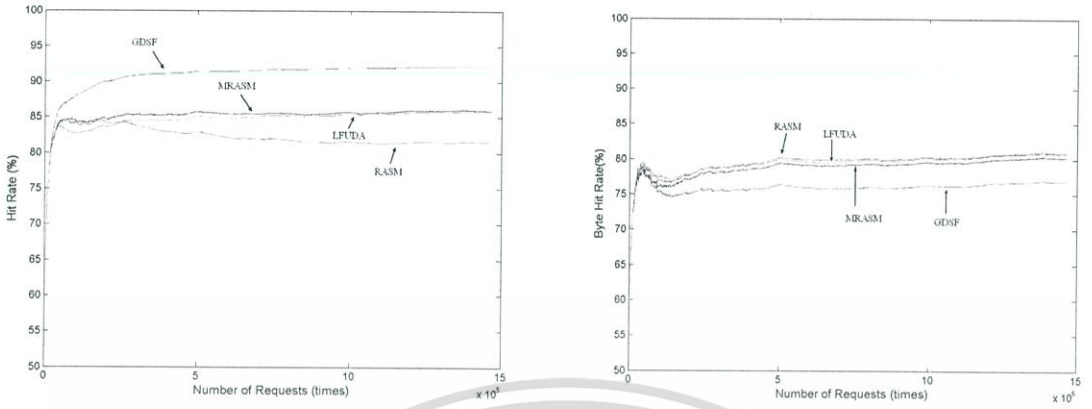


รูปที่ 4.4 ขั้นตอนการทำงานของการจำลองการทำงานของเว็บแคชเซิร์ฟเวอร์ในฟังก์ชั่นอัลกอริธึมการแทนที่ข้อมูล

4.5 ผลการทดสอบการทำงานของอัลกอริธึมการแทนที่ข้อมูลที่ถูกปรับแต่ง

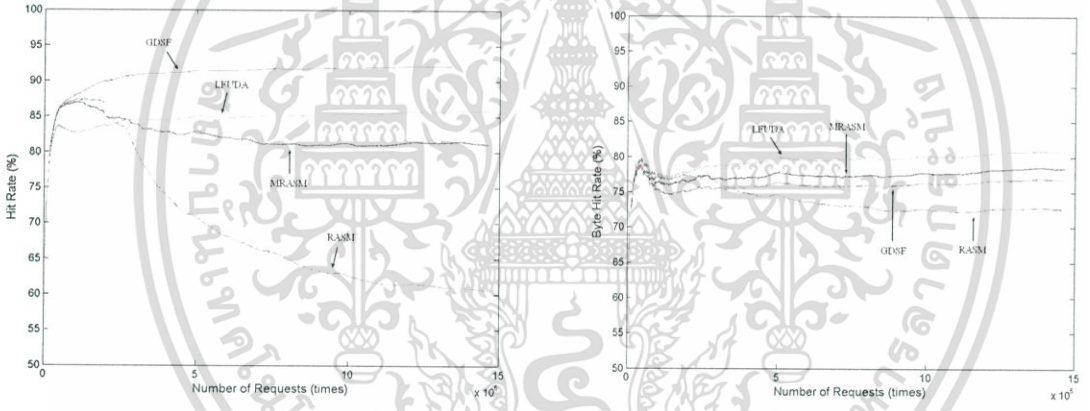
ในการจำลองการทำงานของอัลกอริธึมการแทนที่ข้อมูลที่ถูกปรับแต่ง MRASM เปรียบเทียบกับอัลกอริธึม RASM, GDSF และ LFUDA จะทำการทดสอบโดยทำการปรับเปลี่ยนค่าของ Threshold จำนวน 3 ค่าคือ 1Kbytes,10Kbytes และ 100 Kbytes โดยใช้ข้อมูลชุดเดียวกันในการทดสอบจำนวน 3 ชุดคือข้อมูลที่ได้จาก Clarknet, Nasa และ Calgory แล้วนำมาทำการวิเคราะห์ โดยทำการวิเคราะห์ในส่วนของ Hit Rate และ Byte-Hit Rate ที่ได้จากการวิเคราะห์ค่า Threshold ที่มีการปรับเปลี่ยนค่า และได้ทำการปรับเปลี่ยนขนาดแคชของเว็บแคชเซิร์ฟเวอร์ให้มีขนาด 32, 64, 128, 256 และ 512 Mbytes เพื่อพิจารณาผลกระทบที่เกิดจากขนาดของแคชที่เปลี่ยนไป ซึ่งจะได้ผลการทดสอบเปรียบเทียบดังนี้

4.5.1 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 32 Mbytes



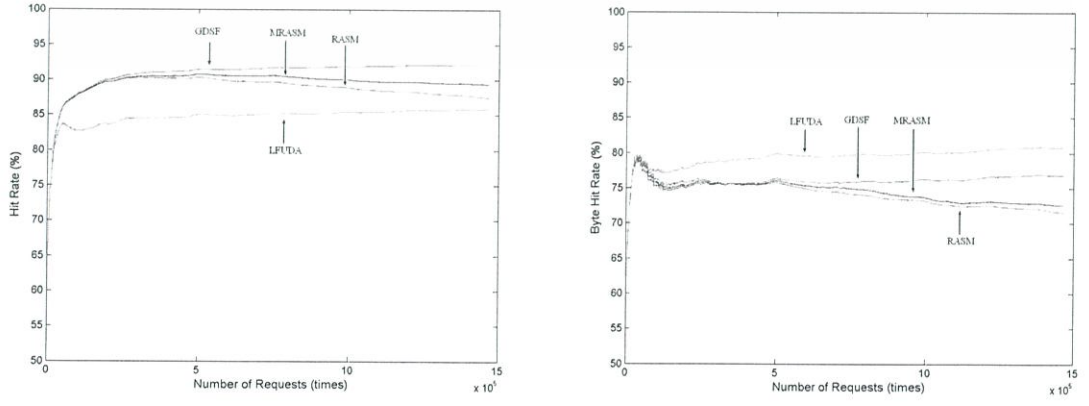
รูปที่ 4.5 **และ** 4.6 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.2 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 32 Mbytes



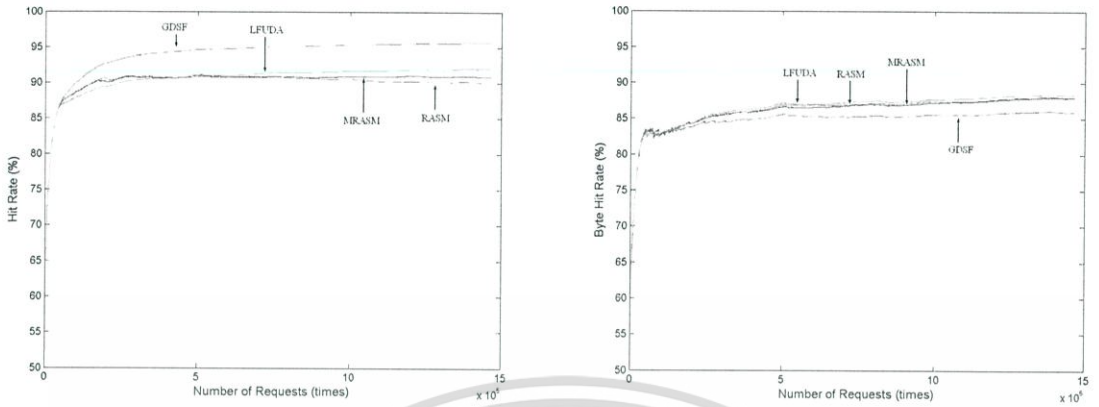
รูปที่ 4.7 **และ** 4.8 กราฟแสดงค่า Hit Rate และ Bytehit Rate ตามลำดับ

4.5.3 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 32 Mbytes



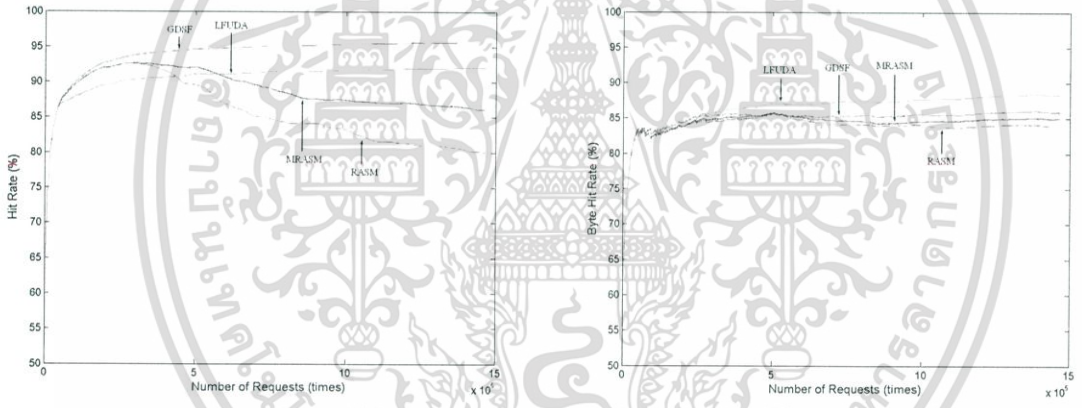
รูปที่ 4.9 **และ** 4.10 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.4 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 64 Mbytes



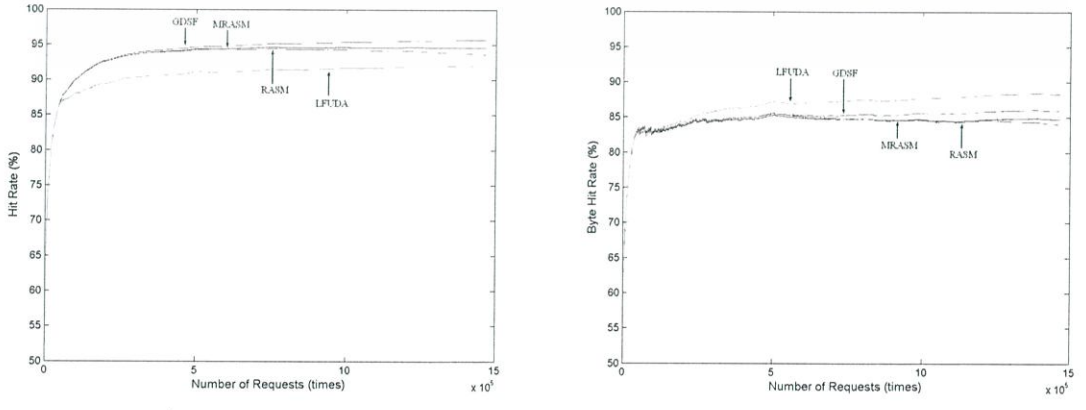
รูปที่ 4.11 และ 4.12 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.5 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 64 Mbytes



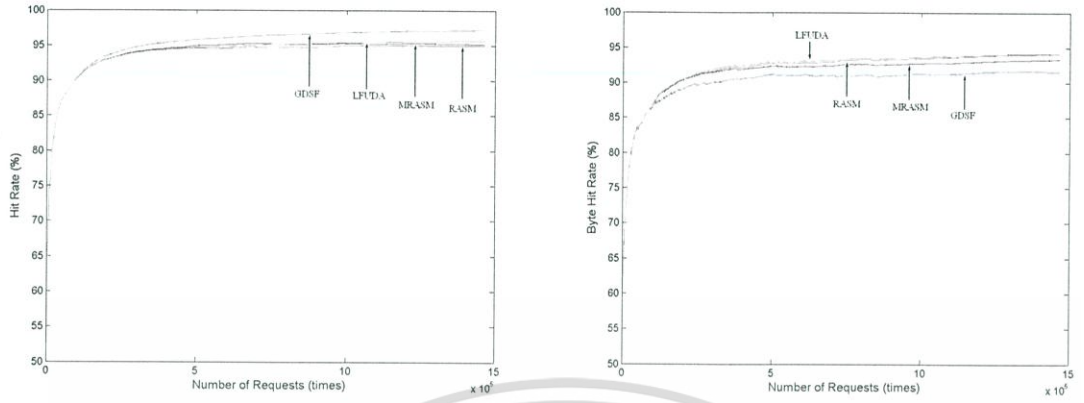
รูปที่ 4.13 และ 4.14 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.6 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 64 Mbytes



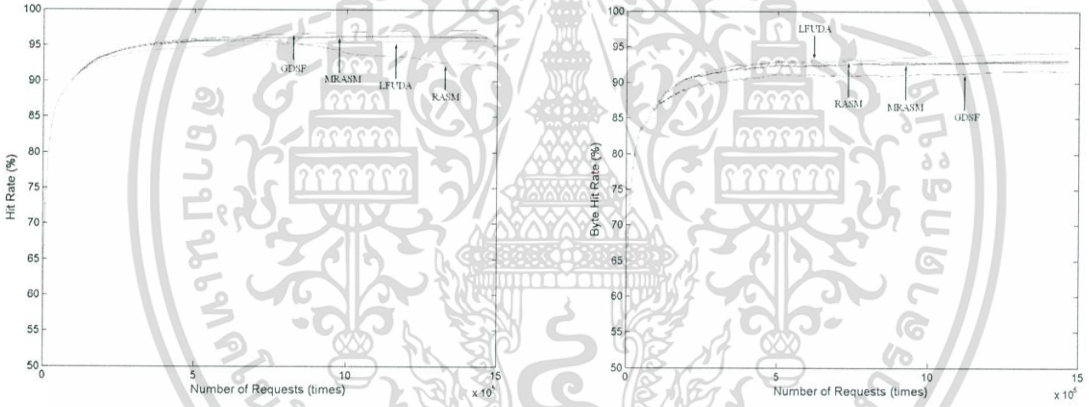
รูปที่ 4.15 และ 4.16 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.7 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 128 Mbytes



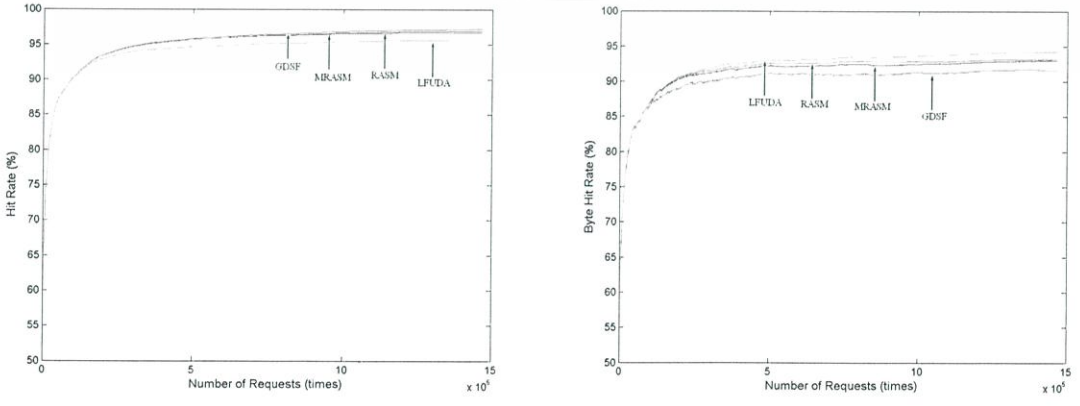
รูปที่ 4.17 **และ** 4.18 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.8 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 128 Mbytes



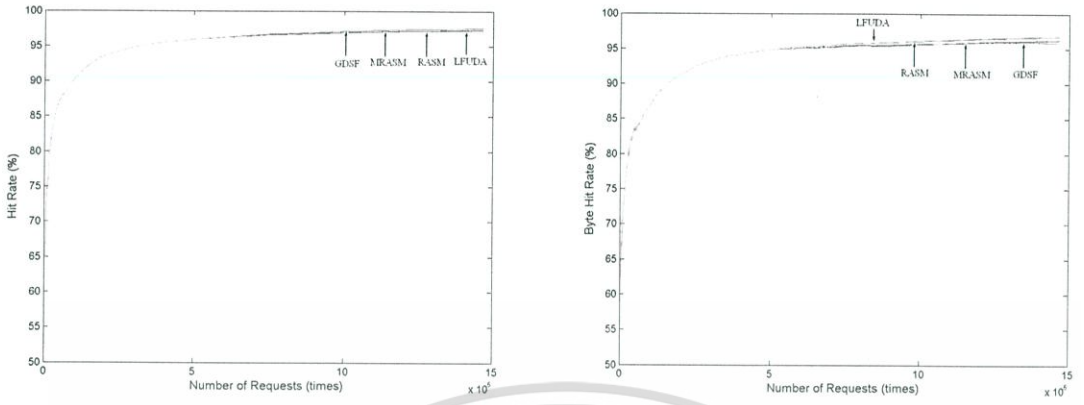
รูปที่ 4.19 **และ** 4.20 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.9 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 128 Mbytes



รูปที่ 4.21 **และ** 4.22 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.10 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 256 Mbytes

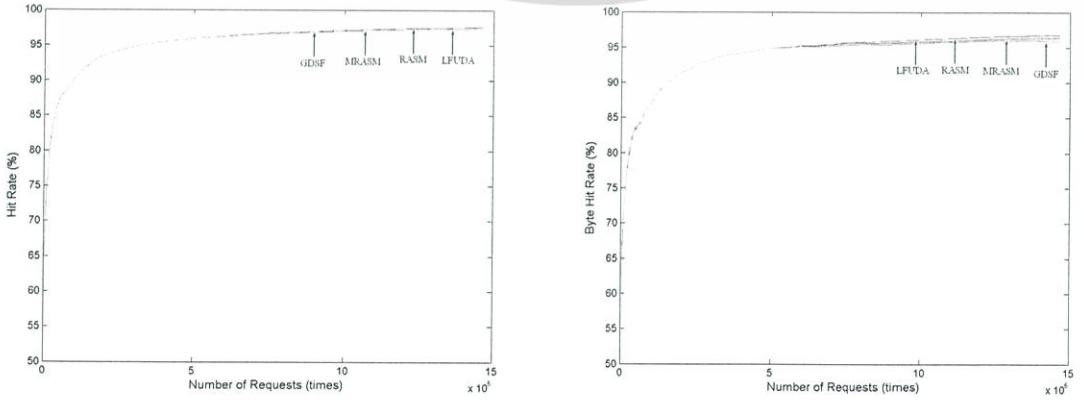


รูปที่ 4.23 **และ** 4.24 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.11 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 256 Mbytes

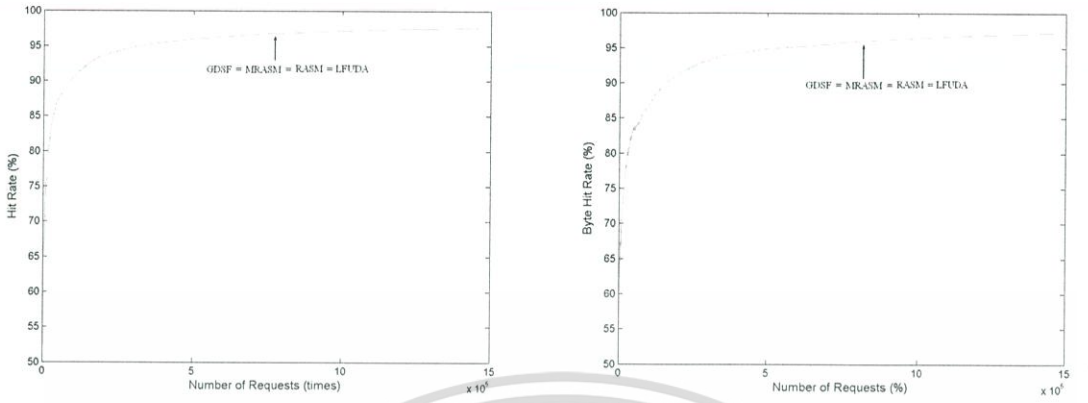
รูปที่ 4.25 **และ** 4.26 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.12 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 256 Mbytes



รูปที่ 4.27 **และ** 4.28 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.13 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 512 Mbytes

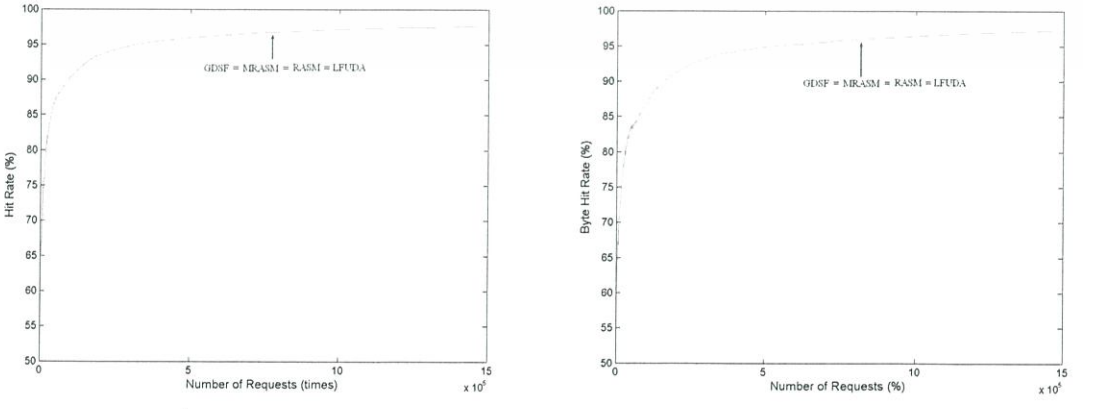


รูปที่ 4.29 **และ** 4.30 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.14 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 512 Mbytes

รูปที่ 4.31 **และ** 4.32 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.15 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก Clarknet โดยแคชขนาด 512 Mbytes



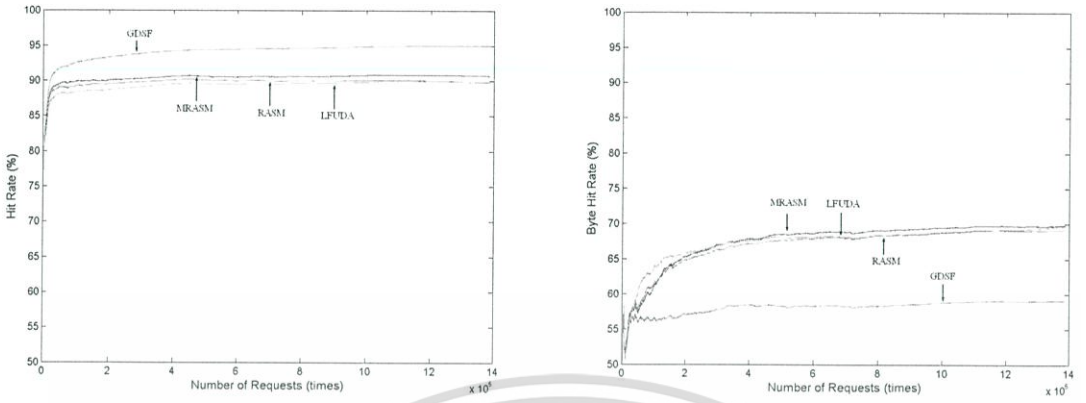
รูปที่ 4.33 **และ** 4.34 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

จากกราฟที่ได้แสดงผลการทดลองของ Clarknet สามารถแสดงในรูปตัวเลขได้ดังตารางที่ 4.1

ตารางที่ 4.1 แสดงผลการจำลองการทำงานเชิงตัวเลขโดยใช้ข้อมูลของ Clarknet

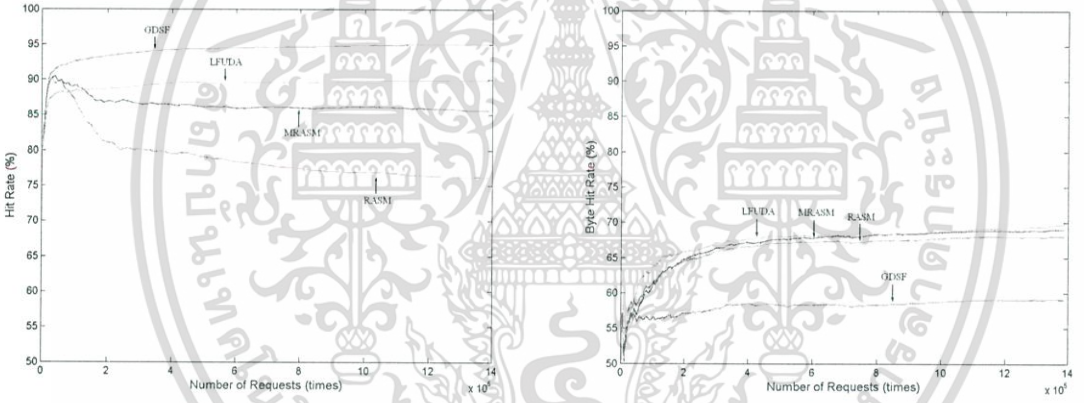
Clarknet								
threshold	Cache 32 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	92.208	85.825	81.539	85.929	76.838	80.763	80.927	80.233
10 Kbytes			60.554	81.153			72.634	78.389
100 Kbytes			87.476	89.389			71.545	72.576
1 Mbytes			92.143	91.120			77.014	74.674
	Cache 64 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	95.682	92.028	90.110	90.873	85.946	88.334	88.017	87.965
10 Kbytes			80.015	86.089			83.931	85.001
100 Kbytes			93.616	94.551			84.014	84.758
1 Mbytes			95.631	95.416			86.087	85.693
	Cache 128 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	97.230	95.692	94.984	95.195	91.589	94.246	94.130	93.350
10 Kbytes			92.250	96.109			92.813	93.085
100 Kbytes			96.671	96.926			93.180	93.008
1 Mbytes			97.158	97.161			93.117	92.817
	Cache 256 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	97.590	97.371	97.390	97.298	95.873	96.846	96.732	96.250
10 Kbytes			97.525	97.525			96.787	96.427
100 Kbytes			97.570	97.563			96.760	96.411
1 Mbytes			97.588	97.584			96.527	96.432
	Cache 512 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	97.618	97.618	97.618	97.618	97.209	97.209	97.209	97.209
10 Kbytes			97.618	97.618			97.209	97.209
100 Kbytes			97.618	97.618			97.209	97.209
1 Mbytes			97.618	97.618			97.209	97.209

4.5.16 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 32 Mbytes



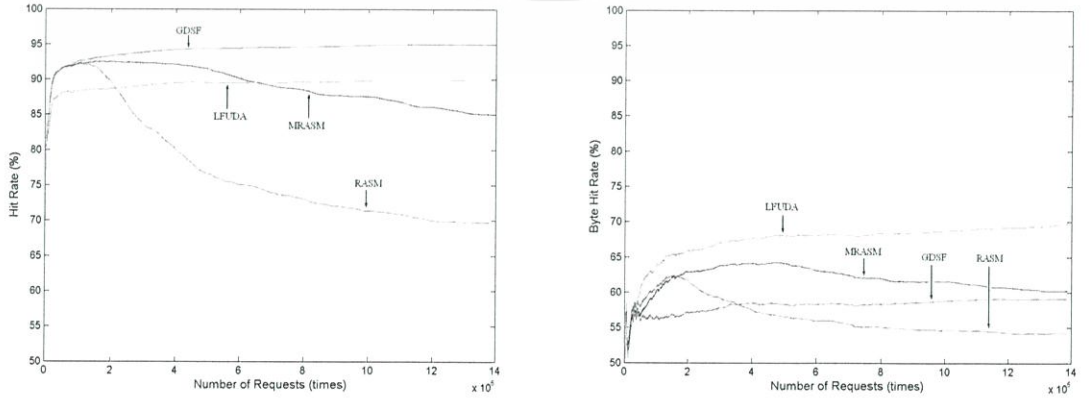
รูปที่ 4.35 **และ** 4.36 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.17 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 32 Mbytes



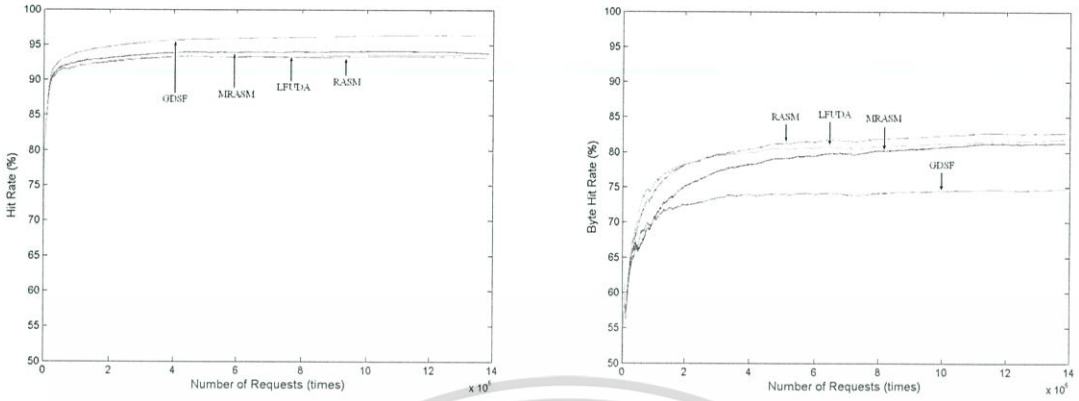
รูปที่ 4.37 **และ** 4.38 กราฟแสดงค่า Hit Rate และ Bytehit Rate ตามลำดับ

4.5.18 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 32 Mbytes



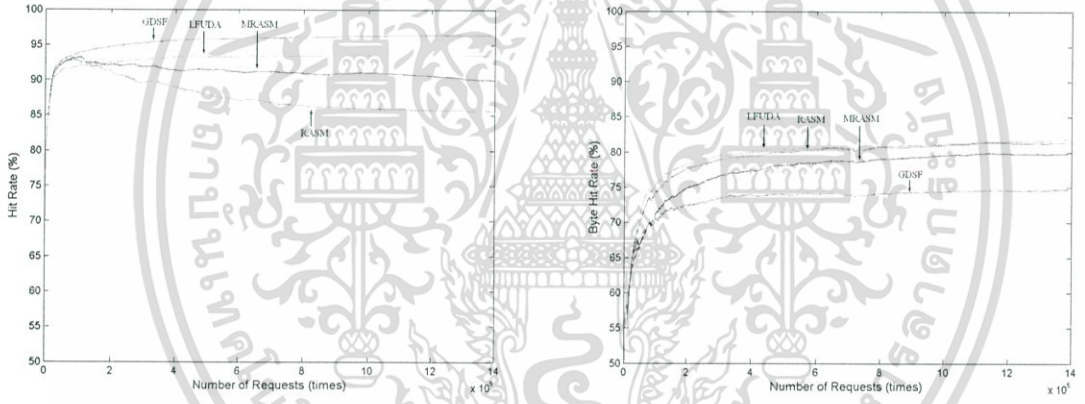
รูปที่ 4.39 **และ** 4.40 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.19 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 64 Mbytes



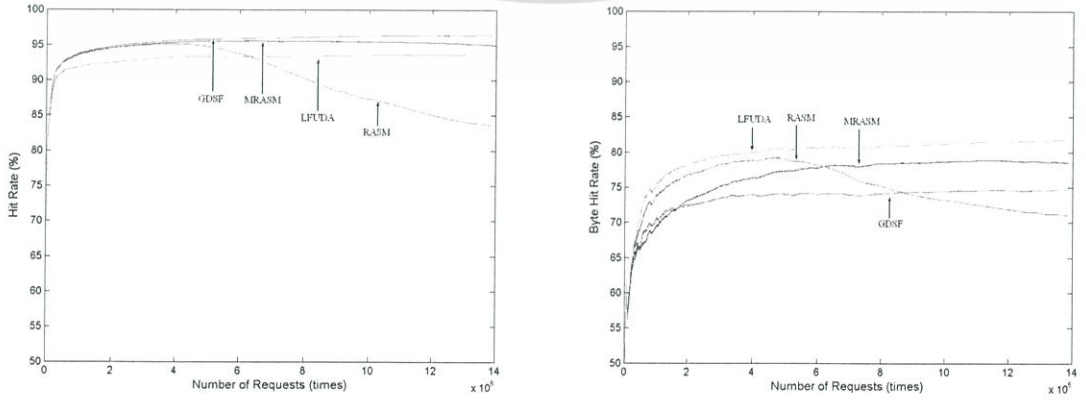
รูปที่ 4.41 และ 4.42 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.20 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 64 Mbytes



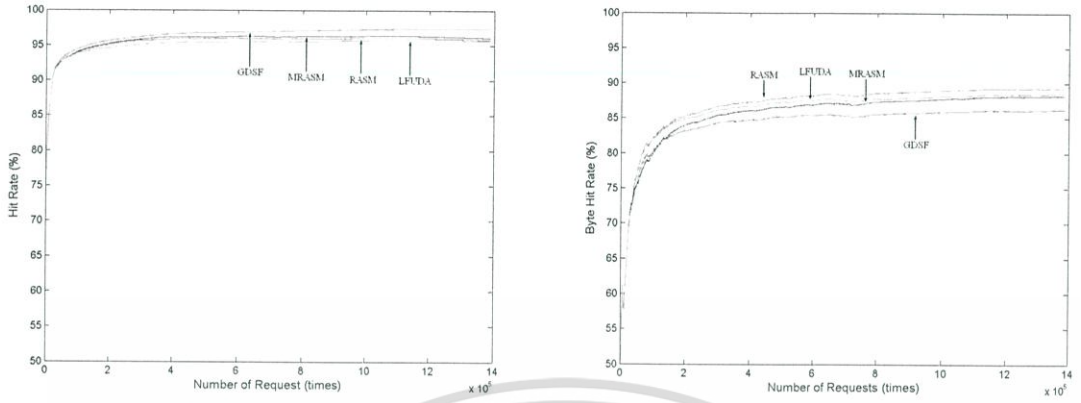
รูปที่ 4.43 และ 4.44 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.21 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 64 Mbytes



รูปที่ 4.45 และ 4.46 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.22 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 128 Mbytes

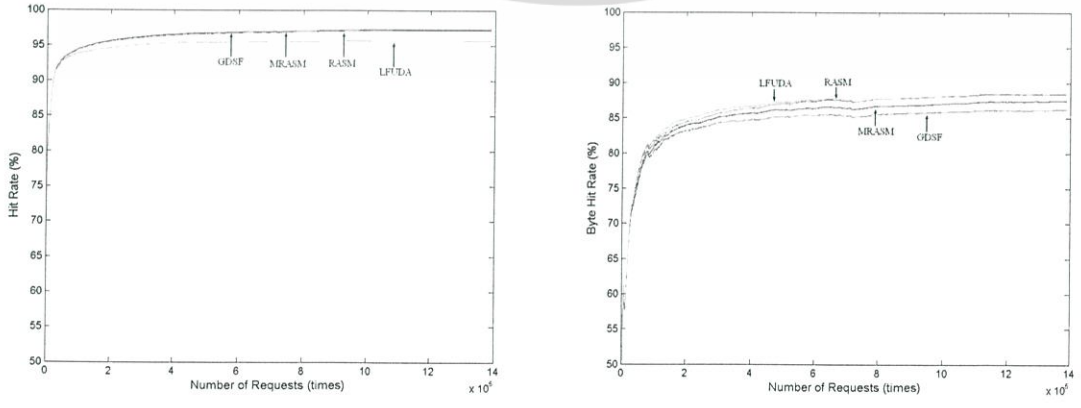


รูปที่ 4.47 และ 4.48 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.23 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 128 Mbytes

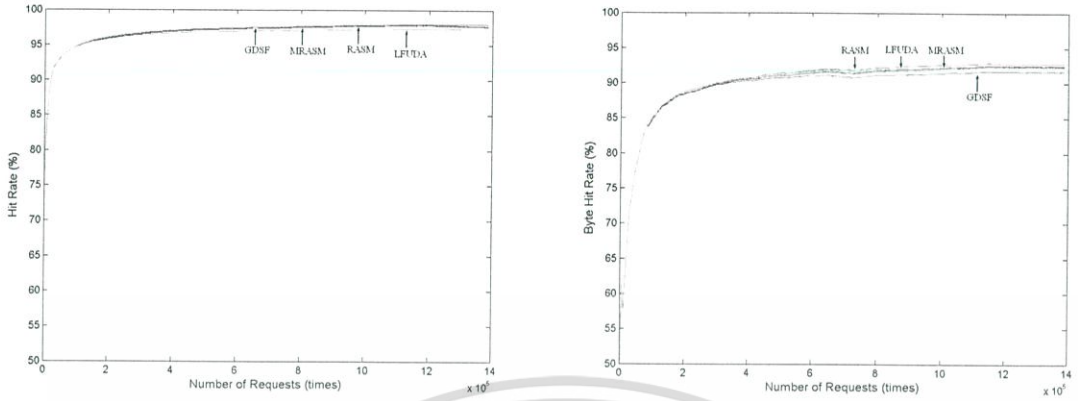
รูปที่ 4.49 และ 4.50 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.24 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 128 Mbytes



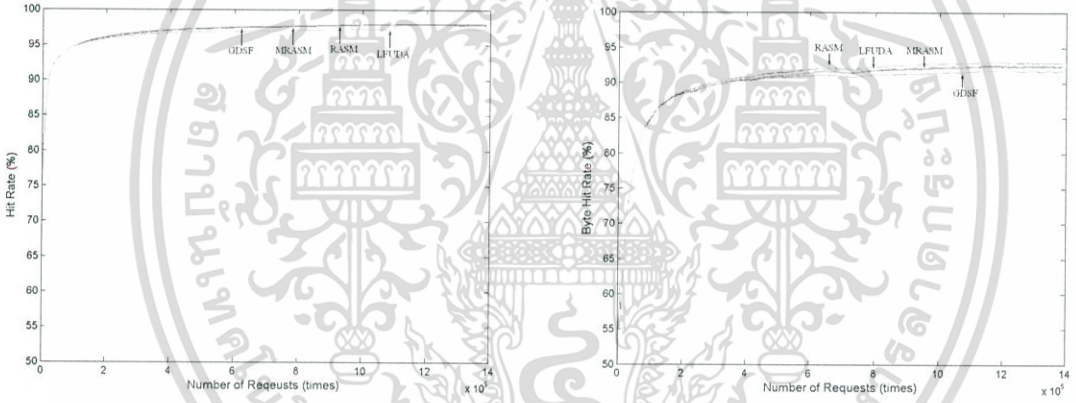
รูปที่ 4.51 และ 4.52 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.25 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 256 Mbytes



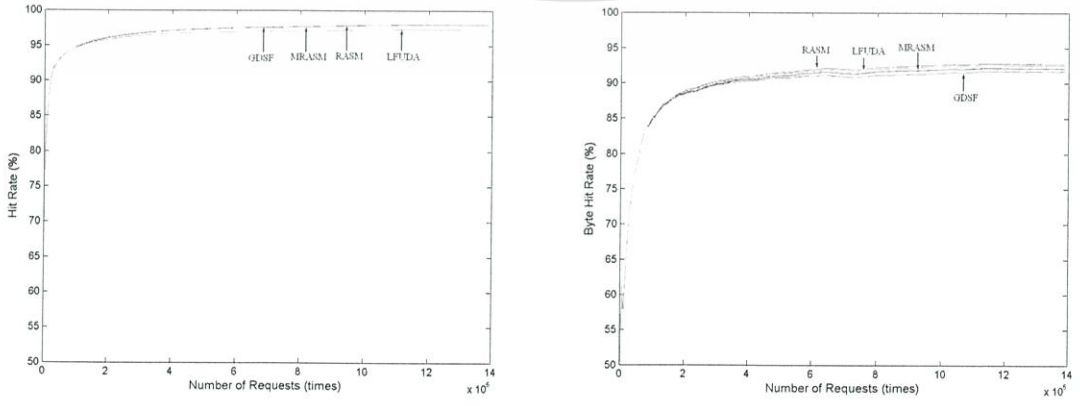
รูปที่ 4.53 **และ** 4.54 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.26 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 256 Mbytes



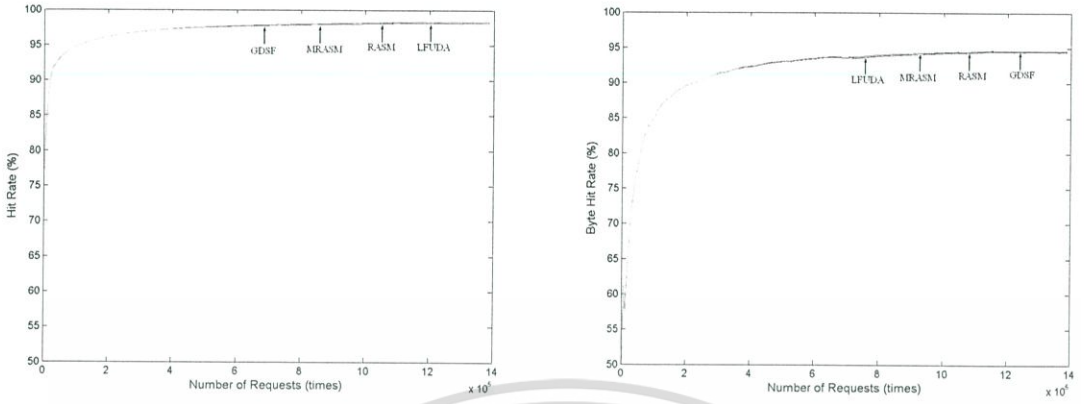
รูปที่ 4.55 **และ** 4.56 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.27 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 256 Mbytes



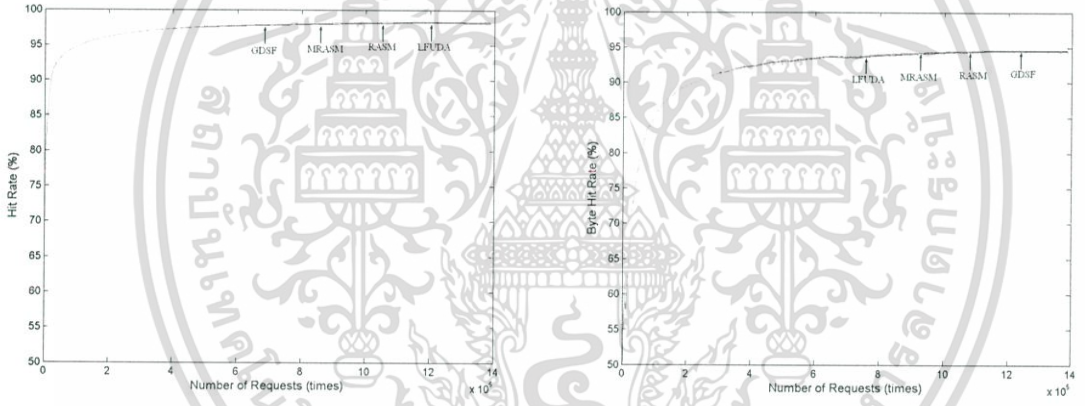
รูปที่ 4.57 **และ** 4.58 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.28 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 512 Mbytes



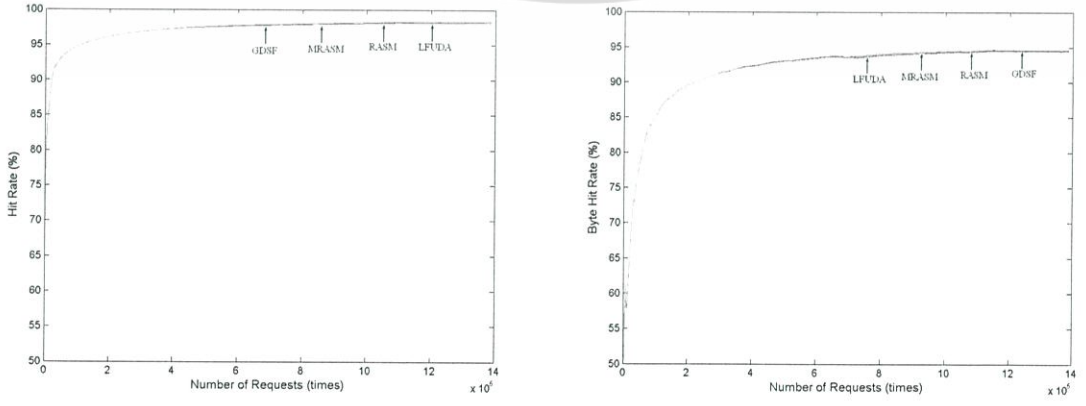
รูปที่ 4.59 **และ** 4.60 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.29 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 512 Mbytes



รูปที่ 4.61 **และ** 4.62 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.30 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก NASA โดยแคชขนาด 512 Mbytes



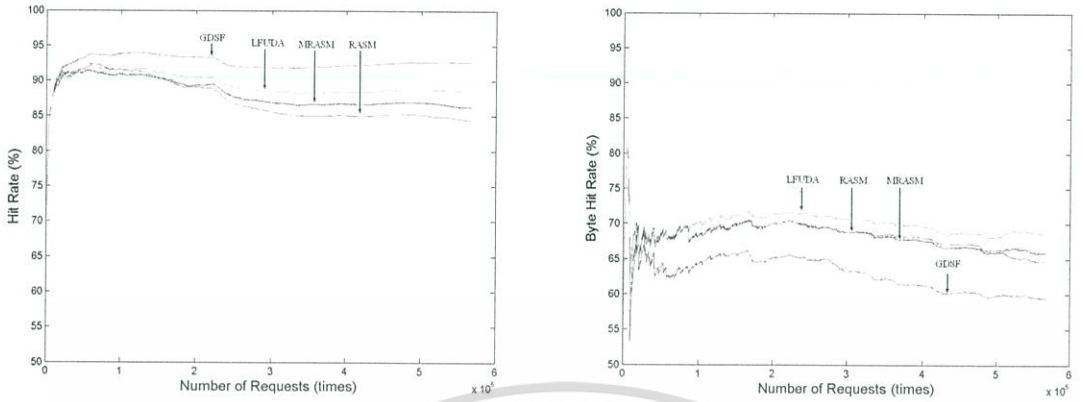
รูปที่ 4.63 **และ** 4.64 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

จากกราฟที่ได้แสดงผลการทดลองของ NASA สามารถแสดงผลอยู่ในรูปตัวเลขได้ดังตารางที่ 4.2

ตารางที่ 4.2 แสดงผลการจำลองการทำงานเชิงตัวเลขโดยใช้ข้อมูลของ NASA

NASA								
Threshold	Cache 32 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	95.001	89.853	89.769	90.707	59.069	69.722	69.125	69.796
10 Kbytes			76.110	85.619			68.014	69.015
100 Kbytes			69.664	85.048			54.236	60.234
1 Mbytes			94.891	94.783			64.350	63.415
	Cache 64 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	96.383	93.530	93.149	93.824	74.752	81.797	82.707	81.287
10 Kbytes			85.632	90.059			81.340	79.779
100 Kbytes			83.573	95.004			71.096	78.541
1 Mbytes			96.376	96.348			76.582	76.113
	Cache 128 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	97.364	95.659	95.796	96.075	86.241	88.509	89.301	88.221
10 Kbytes			92.779	94.910			88.754	87.699
100 Kbytes			97.130	97.238			88.452	87.501
1 Mbytes			97.373	97.358			86.746	86.525
	Cache 256 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	97.963	97.231	97.588	97.699	91.673	92.508	92.880	92.406
10 Kbytes			97.678	97.879			92.891	92.346
100 Kbytes			97.976	97.964			92.756	92.110
1 Mbytes			97.973	97.963			92.101	91.714
	Cache 512 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	98.287	98.068	98.192	98.196	94.554	94.649	94.663	94.472
10 Kbytes			98.231	98.230			94.703	94.506
100 Kbytes			98.285	98.281			94.694	94.523
1 Mbytes			98.295	98.287			94.739	94.685

4.5.31 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 32 Mbytes

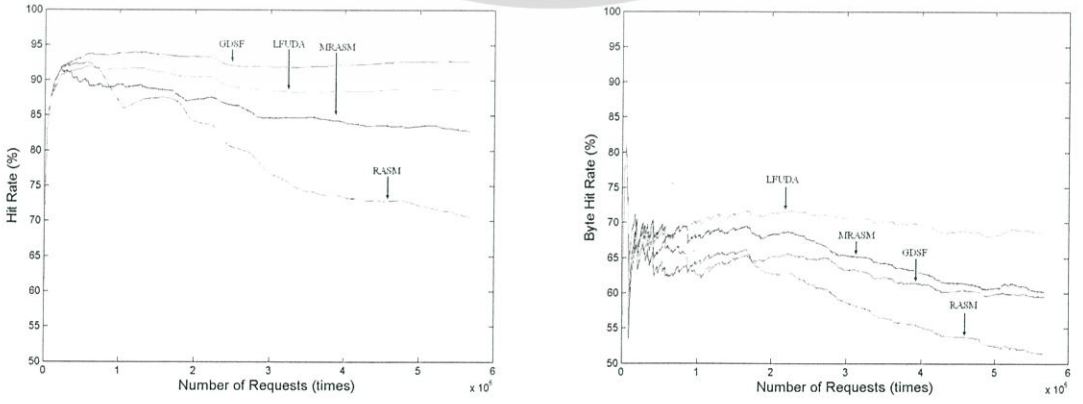


รูปที่ 4.65 **และ** 4.66 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.32 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 32 Mbytes

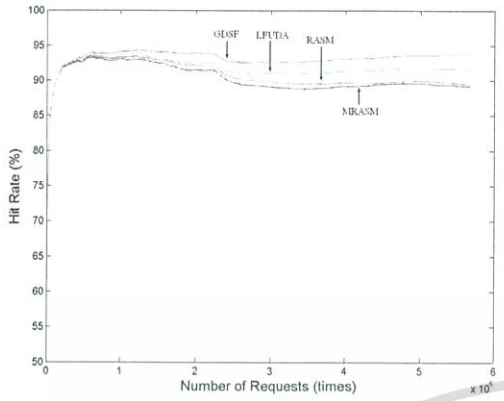
รูปที่ 4.67 **และ** 4.68 กราฟแสดงค่า Hit Rate และ Bytehit Rate ตามลำดับ

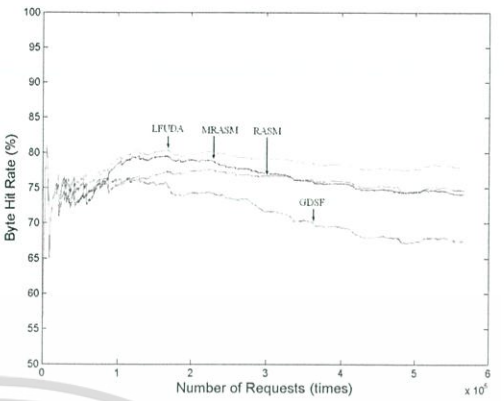
4.5.33 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 32 Mbytes



รูปที่ 4.69 **และ** 4.70 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

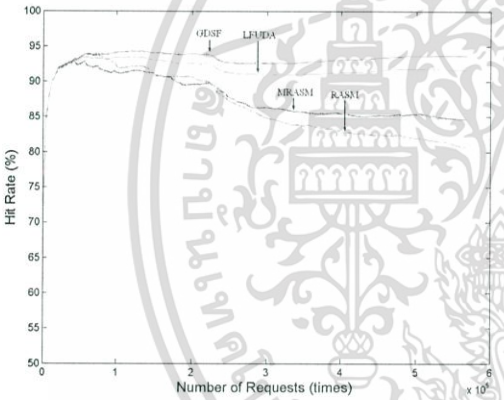
4.5.34 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 64 Mbytes

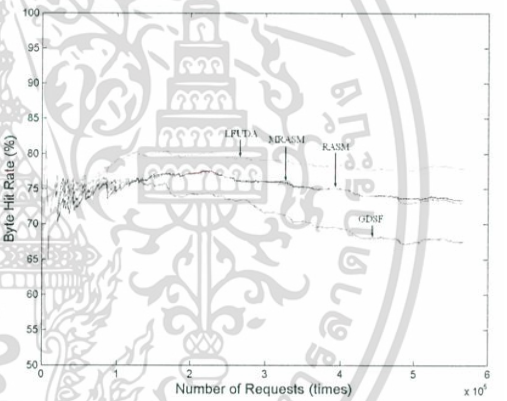




รูปที่ 4.71 **และ** 4.72 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

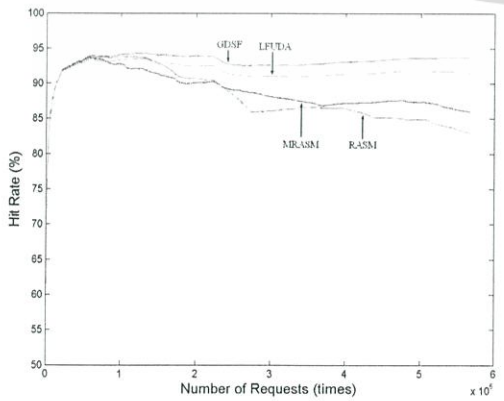
4.5.35 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 64 Mbytes

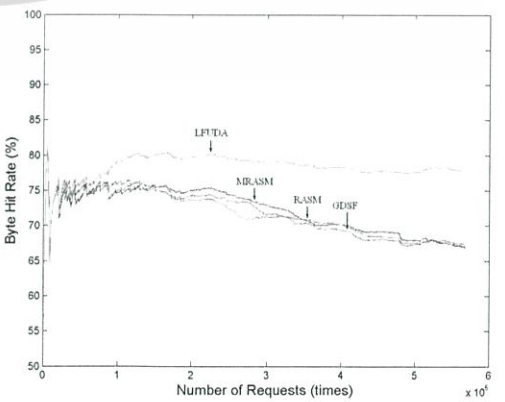




รูปที่ 4.73 **และ** 4.74 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

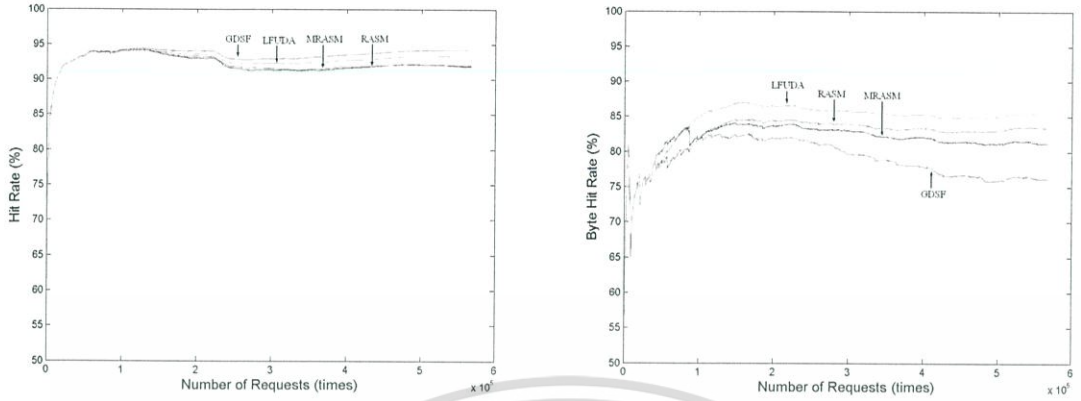
4.5.36 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 64 Mbytes





รูปที่ 4.75 **และ** 4.76 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.37 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 128 Mbytes

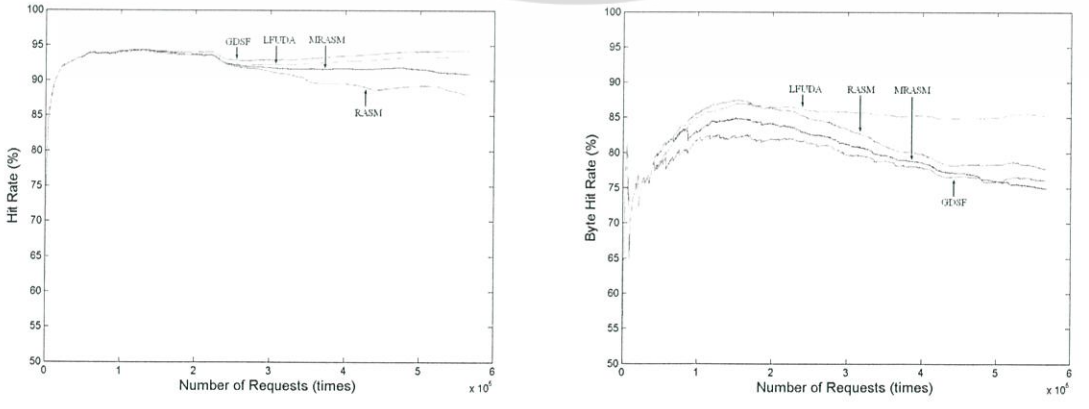


รูปที่ 4.77 และ 4.78 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.38 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 128 Mbytes

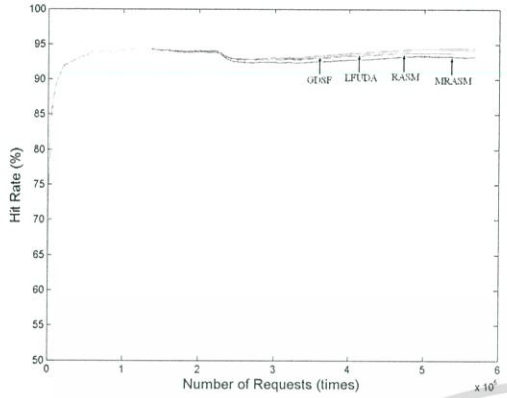
รูปที่ 4.79 และ 4.80 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

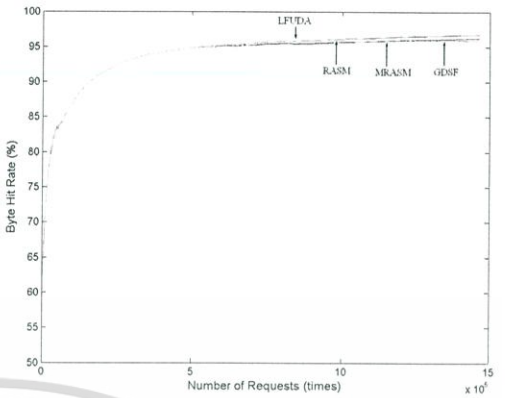
4.5.39 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 128 Mbytes



รูปที่ 4.81 และ 4.82 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

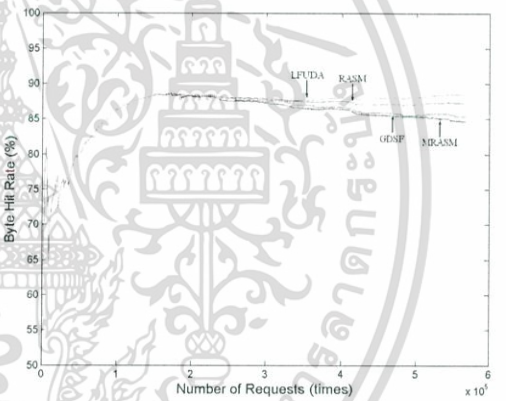
4.5.40 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 256 Mbytes





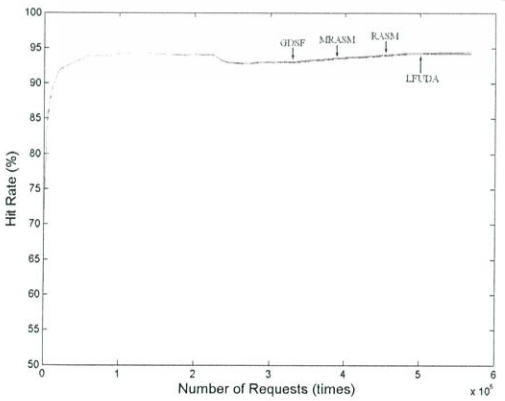
รูปที่ 4.83 **และ** 4.84 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

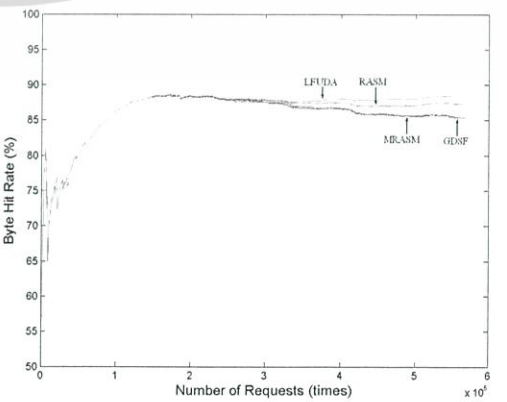
4.5.41 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 256 Mbytes



รูปที่ 4.85 **และ** 4.86 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

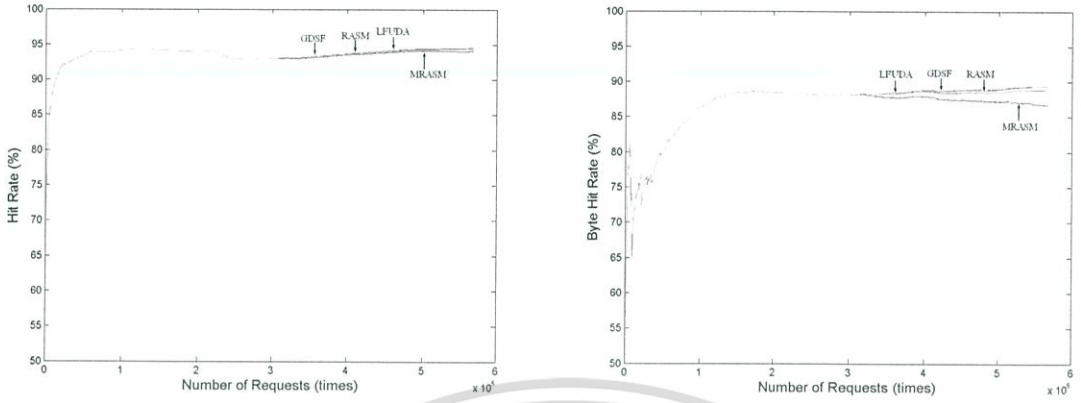
4.5.42 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 256 Mbytes





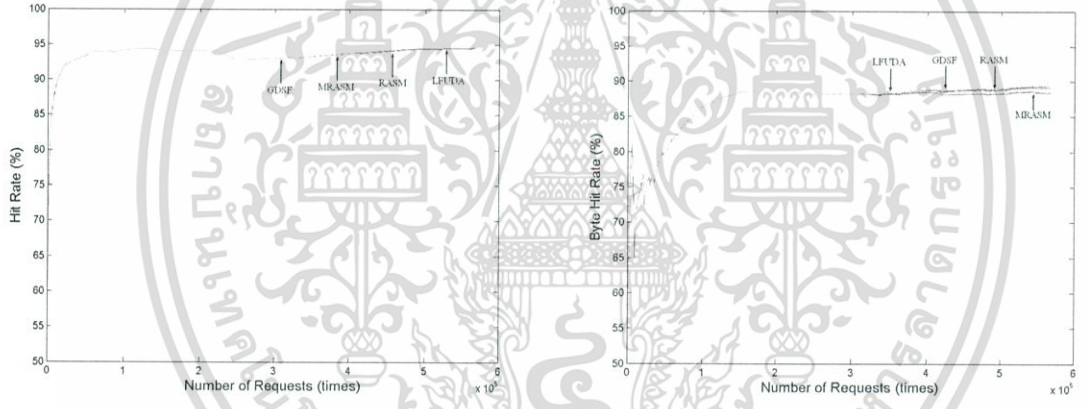
รูปที่ 4.87 **และ** 4.88 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.43 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 1 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 512 Mbytes



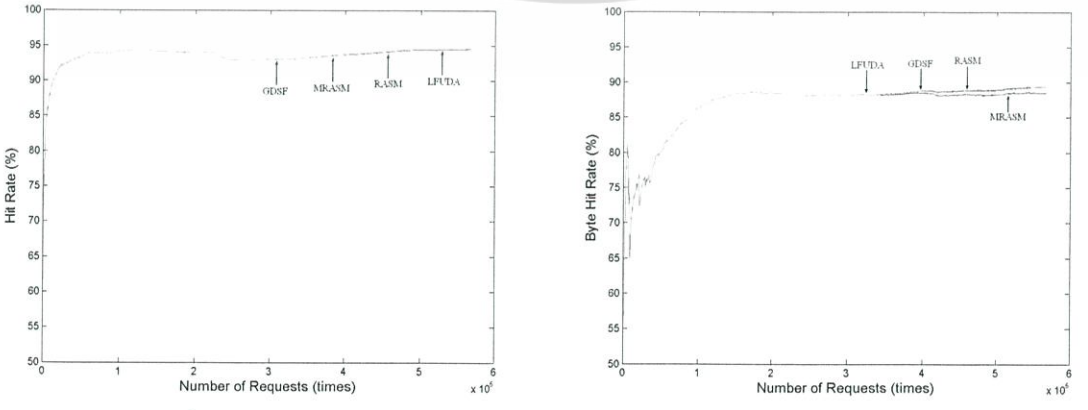
รูปที่ 4.89 **และ** 4.90 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.44 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 10 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 512 Mbytes



รูปที่ 4.91 **และ** 4.92 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

4.5.45 การเปรียบเทียบอัลกอริธึมการแทนที่ข้อมูลที่ปรับแต่งให้ค่า Threshold มีค่า 100 Kbytes ซึ่งใช้ข้อมูลจาก Calgory โดยแคชขนาด 512 Mbytes



รูปที่ 4.93 **และ** 4.94 กราฟแสดงค่า Hit Rate และค่า Bytehit Rate ตามลำดับ

จากกราฟที่ได้แสดงผลการทดลองของ Calgory สามารถแสดงในรูปตัวเลขได้ดังตารางที่ 4.3

ตารางที่ 4.3 แสดงผลการจำลองการทำงานเชิงตัวเลขโดยใช้ข้อมูลของ Calgory

Calgory								
Threshold	Cache 32 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	92.687	88.504	84.444	86.282	59.547	68.644	64.762	65.914
10 Kbytes			71.843	81.786			61.819	65.904
100 Kbytes			70.568	82.800			51.395	60.213
1 Mbytes			89.345	87.609			58.017	57.087
	Cache 64 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	93.693	91.604	89.447	89.199	67.508	77.898	74.671	74.120
10 Kbytes			80.628	84.685			73.011	73.443
100 Kbytes			82.987	86.014			66.913	67.110
1 Mbytes			91.336	90.451			66.791	62.456
	Cache 128 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	94.257	93.275	91.842	91.941	76.197	85.372	83.449	81.237
10 Kbytes			88.179	89.112			82.252	80.142
100 Kbytes			87.824	90.929			77.717	75.009
1 Mbytes			93.502	93.377			77.485	75.057
	Cache 256 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	94.500	94.188	93.708	93.183	85.374	88.528	87.516	84.665
10 Kbytes			93.691	93.841			87.229	84.710
100 Kbytes			94.326	94.317			87.282	85.406
1 Mbytes			94.428	94.425			86.437	85.591
	Cache 512 Mbytes							
	Average Hit Rate (%)				Average ByteHit Rate (%)			
	GDSF	LFUDA	RASM	MRASM	GDSF	LFUDA	RASM	MRASM
1 Kbytes	94.564	94.504	94.396	94.068	89.403	89.408	88.889	86.761
10 Kbytes			94.532	94.480			89.197	88.414
100 Kbytes			94.560	94.552			89.417	88.509
1 Mbytes			94.563	94.558			89.245	88.424

4.6 การวิเคราะห์ผลการทดสอบ

จากผลการทดสอบเปรียบเทียบของอัลกอริธึมการแทนที่ข้อมูล MRASM เมื่อเปรียบเทียบกับอัลกอริธึม RASM, GDSF และ LFUDA โดยใช้ข้อมูลจาก Clarknet, NASA และ Calgory สามารถวิเคราะห์ได้ดังนี้

4.6.1 การวิเคราะห์ผลการจำลองการทำงานโดยข้อมูลของ Clarknet

การวิเคราะห์ผลการจำลองการทำงานของอัลกอริธึมการแทนที่ข้อมูล MRASM เมื่อเปรียบเทียบกับอัลกอริธึม RASM, GDSF และ LFUDA เราจะทำการวิเคราะห์ใน 2 ลักษณะคือ วิเคราะห์ในเชิงปรับเปลี่ยนขนาดของแคช และวิเคราะห์ในเชิงปรับเปลี่ยนค่า threshold จากกราฟรูปที่ 4.5 ถึง 4.34 และตารางที่ 4.1 ซึ่งแสดงผลการจำลองการทำงานเชิงตัวเลข

4.6.1.1 การวิเคราะห์ผลการปรับเปลี่ยนขนาดของแคช

จากการปรับเปลี่ยนขนาดแคชของเว็บแคชเซิร์ฟเวอร์โดยตั้งค่าขนาดของแคชเป็น 32, 64, 128, 256 และ 512 Mbytes

- ขนาดของแคชที่ 32 Mbytes เมื่อใช้ข้อมูลจาก Clarknet จะให้ค่า Hit Rate ของอัลกอริธึมการแทนที่ข้อมูล MRASM สูงกว่าอัลกอริธึม RASM ในช่วง threshold ที่มีค่าต่ำถึงปานกลาง แต่ในช่วงค่า threshold สูงๆ ค่า Hit Rate ของอัลกอริธึม MRASM จะต่ำกว่าของอัลกอริธึม RASM และค่า Hit Rate ของอัลกอริธึม MRASM จะอยู่ระหว่างอัลกอริธึม GDSF และ LFUDA โดยจะอยู่ต่ำกว่าอัลกอริธึม GDSF และสูงกว่าอัลกอริธึม LFUDA ซึ่งเป็นลักษณะที่เหมาะสมของอัลกอริธึม MRASM และ RASM [9]

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM มีค่าสูงกว่าอัลกอริธึม RASM ในช่วง threshold กลางๆ ส่วนที่ threshold ที่ต่ำและสูงมากๆจะได้ค่าต่ำกว่าอัลกอริธึม RASM

- ขนาดของแคชที่ 64 Mbytes จะให้ค่า Hit Rate ของอัลกอริธึม MRASM สูงกว่าอัลกอริธึม RASM ในช่วง threshold ที่มีค่าต่ำถึงปานกลาง แต่ในช่วงค่า threshold สูงๆ จะได้ค่า Hit Rate ใกล้เคียงกัน และค่า Hit Rate ของอัลกอริธึม MRASM จะอยู่ระหว่างอัลกอริธึม GDSF และ LFUDA โดยจะอยู่ต่ำกว่าอัลกอริธึม GDSF และสูงกว่าอัลกอริธึม LFUDA ในช่วง threshold ที่สูง

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM มีค่าสูงกว่าอัลกอริธึม RASM ในช่วง threshold กลางๆ ส่วนที่ threshold ที่ต่ำและสูงมากๆจะได้ค่าต่ำกว่าอัลกอริธึม RASM

- ขนาดของแคชที่ 128 Mbytes จะให้ค่า Hit Rate ของอัลกอริธึม MRASM สูงกว่าอัลกอริธึม RASM ในช่วง threshold ที่มีค่าต่ำถึงปานกลาง แต่ในช่วงค่า threshold สูงๆ จะได้ค่า Hit Rate ใกล้เคียงกัน เหมือนขนาดแคช 64 Mbytes และค่า Hit Rate ของอัลกอริธึม MRASM จะ

อยู่ระหว่างอัลกอริธึม GDSF และ LFUDA โดยจะอยู่ต่ำกว่าอัลกอริธึม GDSF และสูงกว่าอัลกอริธึม LFUDA

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM มีค่าสูงกว่าอัลกอริธึม RASM ในช่วง threshold กลางๆ ส่วนที่ threshold ที่ต่ำและสูงมากๆจะได้ค่าต่ำกว่าอัลกอริธึม RASM ไม่มาก

- ขนาดของแคชที่ 256 Mbytes จะให้ค่า Hit Rate ของอัลกอริธึม MRASMและ RASM ใกล้เคียงกัน และค่าอยู่ระหว่างอัลกอริธึม GDSF และ LFUDA โดยจะอยู่ต่ำกว่าอัลกอริธึม GDSF และสูงกว่าอัลกอริธึม LFUDA

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASMและ RASM ก็ใกล้เคียงกัน และอยู่ระหว่างอัลกอริธึม GDSF และ LFUDA โดยจะอยู่ต่ำกว่าอัลกอริธึม LFUDA และสูงกว่าอัลกอริธึม GDSF [9]

- ขนาดของแคชที่ 512 Mbytes จะให้ค่า Hit Rate ของอัลกอริธึม MRASM, RASM, GDSF และ LFUDA เท่ากัน

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM, RASM, GDSF และ LFUDA มีค่าเท่ากันเช่นกัน

4.6.1.2 การวิเคราะห์ผลการปรับเปลี่ยนค่า threshold

จากตารางที่ 4.1 เมื่อทำการปรับเปลี่ยนค่า threshold จาก 1,10,100 Kbytes จะได้ค่า Hit Rate ของ MRASM จะมีค่าลดลงในช่วง threshold ค่าน้อยถึงปานกลาง และค่า Hit Rate จะเพิ่มขึ้นเมื่อเพิ่มขนาด threshold ในระดับสูง ส่วนค่า Byte Hit Rate จะมีค่าต่ำลงเมื่อเพิ่มขนาด threshold สูงขึ้น

เนื่องจากเมื่อ threshold มีค่าต่ำๆทำให้อัลกอริธึม MRASM และ RASM มีการพิจารณาข้อมูลส่วนใหญ่มีขนาดมากกว่า threshold ทำให้สมการการหาค่าความสำคัญของข้อมูลเป็นแบบ LFUDA เป็นผลทำให้ค่า Hit Rate ในช่วง threshold ต่ำๆนี้มีค่าต่ำกว่าค่า Hit Rate ในช่วง threshold ที่สูงขึ้น ซึ่งเมื่อ threshold มีค่าสูงมากๆข้อมูลส่วนใหญ่ก็จะมีขนาดน้อยกว่า threshold ทำให้สมการการหาค่าความสำคัญของข้อมูลเป็นแบบ GDSF จึงทำให้ค่า Hit Rate มีค่าสูงขึ้น [9]

ส่วนค่า Byte Hit Rate เมื่อ threshold มีค่าต่ำๆทำให้อัลกอริธึม MRASM และ RASM มีการพิจารณาข้อมูลส่วนใหญ่มีขนาดมากกว่า threshold ทำให้สมการการหาค่าความสำคัญของข้อมูลเป็นแบบ LFUDA เป็นผลทำให้ค่า Byte Hit Rate ในช่วง threshold ต่ำๆนี้มีค่าสูง และจะลดลงเมื่อค่า threshold มีค่าสูงขึ้น ซึ่งสลับกับของค่า Hit Rate [9]

4.6.2 การวิเคราะห์ผลการจำลองการทำงานโดยข้อมูลของ NASA

การวิเคราะห์ผลการจำลองการทำงานของอัลกอริธึมการแทนที่ข้อมูล MRASM เมื่อเปรียบเทียบกับอัลกอริธึม RASM, GDSF และ LFUDA เราจะทำการวิเคราะห์ใน 2 ลักษณะคือ วิเคราะห์ในเชิงปรับเปลี่ยนขนาดของแคช และวิเคราะห์ในเชิงปรับเปลี่ยนค่า threshold เหมือนกับข้อมูลจาก Clarknet โดยกราฟรูปที่ 4.35 ถึง 4.64 และตารางที่ 4.2 ซึ่งแสดงผลการจำลองการทำงานเชิงตัวเลข

4.6.2.1 การวิเคราะห์ผลการปรับเปลี่ยนขนาดของแคช

จากการปรับเปลี่ยนขนาดแคชของเว็บแคชเซิร์ฟเวอร์โดยตั้งค่าขนาดของแคชเป็น 32, 64, 128, 256 และ 512 Mbytes

- ขนาดของแคชที่ 32 Mbytes เมื่อใช้ข้อมูลจาก NASA จะให้ค่า Hit Rate ของอัลกอริธึมการแทนที่ข้อมูล MRASM สูงกว่าอัลกอริธึม RASM ในช่วง threshold ที่มีค่าต่ำถึงปานกลาง แต่ในช่วงค่า threshold สูงๆ ค่า Hit Rate ของอัลกอริธึม MRASM และของอัลกอริธึม RASM จะใกล้เคียงกัน และค่า Hit Rate ของอัลกอริธึม MRASM จะอยู่ระหว่างอัลกอริธึม GDSF และ LFUDA โดยจะอยู่ต่ำกว่าอัลกอริธึม GDSF และสูงกว่าอัลกอริธึม LFUDA ซึ่งเป็นลักษณะที่เหมาะสมของอัลกอริธึม MRASM และ RASM [9]

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM มีค่าสูงกว่าอัลกอริธึม RASM ในช่วง threshold ที่มีค่าต่ำถึงปานกลาง ส่วนที่ threshold ที่สูงจะได้ค่าต่ำกว่าอัลกอริธึม RASM

- ขนาดของแคชที่ 64 Mbytes จะให้ค่า Hit Rate ของอัลกอริธึม MRASM สูงกว่าอัลกอริธึม RASM ในช่วง threshold ที่มีค่าต่ำถึงปานกลาง แต่ในช่วงค่า threshold สูงๆ จะได้ค่า Hit Rate ใกล้เคียงกัน และค่า Hit Rate ของอัลกอริธึม MRASM จะอยู่ระหว่างอัลกอริธึม GDSF และ LFUDA โดยจะอยู่ต่ำกว่าอัลกอริธึม GDSF และสูงกว่าอัลกอริธึม LFUDA ในช่วง threshold ที่สูง

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM มีค่าสูงกว่าอัลกอริธึม RASM ในช่วง threshold กลางๆ ส่วนที่ threshold ที่ต่ำและสูงมากๆจะได้ค่าต่ำกว่าอัลกอริธึม RASM

- ขนาดของแคชที่ 128 Mbytes จะให้ค่า Hit Rate ของอัลกอริธึม MRASM สูงกว่าอัลกอริธึม RASM ในช่วง threshold ที่มีค่าต่ำถึงปานกลาง แต่ในช่วงค่า threshold สูงๆ จะได้ค่า Hit Rate ใกล้เคียงกัน เหมือนขนาดแคช 64 Mbytes และค่า Hit Rate ของอัลกอริธึม MRASM จะอยู่ระหว่างอัลกอริธึม GDSF และ LFUDA โดยจะอยู่ต่ำกว่าอัลกอริธึม GDSF และสูงกว่าอัลกอริธึม LFUDA

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM มีค่าสูงกว่าอัลกอริธึม RASM ไม่มาก ในช่วง threshold กลางๆ ส่วนที่ threshold ที่ต่ำและสูงมากๆจะได้ค่าต่ำกว่าอัลกอริธึม RASM ไม่มาก โดยรวมแล้วใกล้เคียงกัน

- ขนาดของแคชที่ 256 Mbytes จะให้ค่า Hit Rate ของอัลกอริธึม MRASMและ RASM ใกล้เคียงกัน และมีค่าใกล้เคียงกับของอัลกอริธึม GDSF และ LFUDA โดยจะอยู่ต่ำกว่าอัลกอริธึม GDSF และสูงกว่าอัลกอริธึม LFUDA

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASMและ RASM ก็ใกล้เคียงกัน และมีค่าใกล้เคียงกับของอัลกอริธึม GDSF และ LFUDA โดยจะอยู่ต่ำกว่าอัลกอริธึม LFUDA และสูงกว่าอัลกอริธึม GDSF [9]

- ขนาดของแคชที่ 512 Mbytes จะให้ค่า Hit Rate ของอัลกอริธึม MRASMและ RASM ใกล้เคียงกัน และมีค่าใกล้เคียงกับของอัลกอริธึม GDSF และ LFUDA โดยจะอยู่ต่ำกว่าอัลกอริธึม GDSF และสูงกว่าอัลกอริธึม LFUDA

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM, RASM, GDSF และ LFUDA มีค่าใกล้เคียงกัน

4.6.2.2 การวิเคราะห์ผลการปรับเปลี่ยนค่า threshold

จากตารางที่ 4.2 เมื่อทำการปรับเปลี่ยนค่า threshold จาก 1,10,100 Kbytes จะได้ค่า Hit Rate ของ MRASM จะมีค่าลดลงในช่วง threshold ค่าน้อยถึงปานกลาง และค่า Hit Rate จะเพิ่มขึ้นเมื่อเพิ่มขนาด threshold ในระดับสูง ส่วนค่า Byte Hit Rate จะมีค่าต่ำลงเมื่อเพิ่มขนาด threshold สูงขึ้น

เนื่องจากเมื่อ threshold มีค่าต่ำๆทำให้อัลกอริธึม MRASM และ RASM มีการพิจารณาข้อมูลส่วนใหญ่มีขนาดมากกว่า threshold ทำให้สมการการหาค่าความสำคัญของข้อมูลเป็นแบบ LFUDA เป็นผลทำให้ค่า Hit Rate ในช่วง threshold ต่ำๆนี้มีค่าต่ำกว่าค่า Hit Rate ในช่วง threshold ที่สูงขึ้น ซึ่งเมื่อ threshold มีค่าสูงมากๆข้อมูลส่วนใหญ่ก็จะมีขนาดน้อยกว่า threshold ทำให้สมการการหาค่าความสำคัญของข้อมูลเป็นแบบ GDSF จึงทำให้ค่า Hit Rate มีค่าสูงขึ้น [9]

ส่วนค่า Byte Hit Rate เมื่อ threshold มีค่าต่ำๆทำให้อัลกอริธึม MRASM และ RASM มีการพิจารณาข้อมูลส่วนใหญ่มีขนาดมากกว่า threshold ทำให้สมการการหาค่าความสำคัญของข้อมูลเป็นแบบ LFUDA เป็นผลทำให้ค่า Byte Hit Rate ในช่วง threshold ต่ำๆนี้มีค่าสูง และจะลดลงเมื่อค่า threshold มีค่าสูงขึ้น ซึ่งสลับกับของค่า Hit Rate [9]

4.6.3 การวิเคราะห์ผลการจำลองการทำงานโดยข้อมูลของ Calgory

การวิเคราะห์ผลการจำลองการทำงานของอัลกอริธึมการแทนที่ข้อมูล MRASM เมื่อเปรียบเทียบกับอัลกอริธึม RASM, GDSF และ LFUDA เราจะทำการวิเคราะห์ใน 2 ลักษณะคือ วิเคราะห์ในเชิงปรับเปลี่ยนขนาดของแคช และวิเคราะห์ในเชิงปรับเปลี่ยนค่า threshold จากกราฟรูปที่ 4.65 ถึง 4.94 และตารางที่ 4.3 ซึ่งแสดงผลการจำลองการทำงานเชิงตัวเลข

4.6.3.1 การวิเคราะห์ผลการปรับเปลี่ยนขนาดของแคช

จากการปรับเปลี่ยนขนาดแคชของเว็บแคชเซิร์ฟเวอร์โดยตั้งค่าขนาดของแคชเป็น 32, 64, 128, 256 และ 512 Mbytes

- ขนาดของแคชที่ 32 Mbytes เมื่อใช้ข้อมูลจาก Calgory จะให้ค่า Hit Rate ของอัลกอริธึมการแทนที่ข้อมูล MRASM สูงกว่าอัลกอริธึม RASM ในช่วง Threshold ที่มีค่าต่ำถึงปานกลาง แต่ในช่วงค่า threshold สูงๆ ค่า Hit Rate ของอัลกอริธึม MRASM จะต่ำกว่าของอัลกอริธึม RASM แต่ค่า Hit Rate ของอัลกอริธึม MRASM และอัลกอริธึม RASM มีค่าต่ำกว่าอัลกอริธึม GDSF และ LFUDA ซึ่งแตกต่างจากผลการทดลองของข้อมูลจาก Clarknet และ NASA ซึ่งปริมาณการร้องขอข้อมูลของ Calgory มีปริมาณน้อยกว่าของ Clarknet และ NASA มาก ซึ่งจะแสดงรายละเอียดในหัวข้อ 5.4 ของบทที่ 5

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM มีค่าสูงกว่าอัลกอริธึม RASM ในช่วง threshold ที่ต่ำและปานกลาง ส่วนที่ threshold สูงมากๆจะได้ค่าต่ำกว่าอัลกอริธึม RASM

- ขนาดของแคชที่ 64 Mbytes จะให้ค่า Hit Rate ของอัลกอริธึม MRASM สูงกว่าอัลกอริธึม RASM ในช่วง threshold ที่มีค่าปานกลาง แต่ในช่วงค่า Threshold ต่ำและสูงมากๆ จะได้ค่า Hit Rate ใกล้เคียงกัน แต่ค่า Hit Rate ของอัลกอริธึม MRASM และอัลกอริธึม RASM มีค่าต่ำกว่าอัลกอริธึม GDSF และ LFUDA

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM มีค่าสูงกว่าอัลกอริธึม RASM ในช่วง threshold กลางๆ ส่วนที่ threshold ที่ต่ำและสูงมากๆจะได้ค่าต่ำกว่าอัลกอริธึม RASM

- ขนาดของแคชที่ 128 Mbytes จะให้ค่า Hit Rate ของอัลกอริธึม MRASM สูงกว่าอัลกอริธึม RASM ในช่วง threshold ที่มีค่าปานกลาง แต่ในช่วงค่า Threshold ต่ำและสูงมากๆ จะได้ค่า Hit Rate ใกล้เคียงกัน เหมือนขนาดแคช 64 Mbytes

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM มีค่าต่ำกว่าอัลกอริธึม RASM

- ขนาดของแคชที่ 256 Mbytes จะให้ค่า Hit Rate ของอัลกอริธึม MRASMและ RASM ใกล้เคียงกัน และมีค่าใกล้เคียงกับอัลกอริธึม GDSF และ LFUDA

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM มีค่าต่ำกว่าอัลกอริธึม RASM

- ขนาดของแคชที่ 512 Mbytes จะให้ค่า Hit Rate ของอัลกอริธึม MRASM, RASM, GDSF และ LFUDA ใกล้เคียงกัน

ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM มีค่าต่ำกว่าอัลกอริธึม RASM

4.6.3.2 **การวิเคราะห์ผลการปรับเปลี่ยนค่า threshold**

จากตารางที่ 4.3 เมื่อทำการปรับเปลี่ยนค่า threshold จาก 1,10,100 Kbytes และ 1 Mbytes จะได้ค่า Hit Rate ของ MRASM จะมีค่าลดลงในช่วง threshold ค่าน้อยถึงปานกลาง และค่า Hit Rate จะเพิ่มขึ้นเมื่อเพิ่มขนาด threshold ในระดับสูง ส่วนค่า Byte Hit Rate จะมีค่าต่ำลงเมื่อเพิ่มขนาด threshold สูงขึ้น

เนื่องจากเมื่อ threshold มีค่าต่ำๆทำให้อัลกอริธึม MRASM และ RASM มีการพิจารณาข้อมูลส่วนใหญ่มีขนาดมากกว่า threshold ทำให้สมการการหาค่าความสำคัญของข้อมูลเป็นแบบ LFUDA เป็นผลทำให้ค่า Hit Rate ในช่วง threshold ต่ำๆนี้มีค่าต่ำกว่าค่า Hit Rate ในช่วง threshold ที่สูงขึ้น ซึ่งเมื่อ threshold มีค่าสูงมากๆข้อมูลส่วนใหญ่ก็จะมีขนาดน้อยกว่า threshold ทำให้สมการการหาค่าความสำคัญของข้อมูลเป็นแบบ GDSF จึงทำให้ค่า Hit Rate มีค่าสูงขึ้น

ส่วนค่า Byte Hit Rate เมื่อ threshold มีค่าต่ำๆทำให้อัลกอริธึม MRASM และ RASM มีการพิจารณาข้อมูลส่วนใหญ่มีขนาดมากกว่า threshold ทำให้สมการการหาค่าความสำคัญของข้อมูลเป็นแบบ LFUDA เป็นผลทำให้ค่า Byte Hit Rate ในช่วง threshold ต่ำๆนี้มีค่าสูง และจะลดลงเมื่อค่า threshold มีค่าสูงขึ้น ซึ่งสลับกับของค่า Hit Rate

4.6.4 **การวิเคราะห์ลักษณะของอัลกอริธึม MRASM กับ RASM**

จากการทดสอบการทำงานของอัลกอริธึม MRASM เปรียบเทียบกับ RASM โดยใช้ข้อมูลทดสอบ 3 ชุดคือจาก Clarknet, NASA และ Calgory จะพบว่าค่า Hit Rate ของอัลกอริธึม MRASM และ RASM ในช่วงค่า threshold จากค่าต่ำจนถึงปานกลางจะมีค่าลดลง แต่ช่วงค่า threshold จากปานกลางถึงค่าสูง จะได้ค่า Hit Rate สูงขึ้น โดยค่า Hit Rate ของอัลกอริธึมการแทนที่ข้อมูล MRASM สูงกว่าอัลกอริธึม RASM ในช่วง threshold ที่มีค่าต่ำถึงปานกลาง แต่ในช่วงค่า threshold สูงๆ ค่า Hit Rate ของอัลกอริธึม MRASM และของอัลกอริธึม RASM จะใกล้เคียงกัน โดยขนาดแคชไม่ใหญ่มาก ซึ่งถ้าขนาดแคชใหญ่จะได้ค่า Hit Rate ที่ใกล้เคียงกัน ส่วนค่า Byte Hit Rate ของอัลกอริธึม MRASM ส่วนใหญ่จะสูงกว่าของ RASM ในช่วง threshold กลางๆ และขนาดแคชไม่ใหญ่มาก แต่ถ้าขนาดแคชใหญ่มากจะได้ค่า Byte Hit Rate ใกล้เคียงกัน ยกเว้นข้อมูลของ Calgory จะได้ค่า Byte Hit Rate ของอัลกอริธึม MRASM ต่ำกว่าของ RASM ในช่วงแคชขนาดใหญ่

บทที่ 5
การวิเคราะห์หาค่า Threshold ที่เหมาะสม

5.1 แนวคิดในการหาค่า Threshold ที่เหมาะสม

จากผลการทดสอบของอัลกอริธึมการแทนที่ข้อมูลที่นำเสนอค่าของ Hit Rate และ Byte-Hit Rate ที่ได้จากการปรับค่า threshold เป็นค่า 10 Kbytes, 100 Kbytes, 1 Mbytes และ 10 Mbytes จะพบว่าค่าของ Hit Rate และ Byte-Hit Rate ไม่สามารถที่จะแสดงออกมาในรูปของสมการทางคณิตศาสตร์ได้ จึงไม่สามารถที่จะหาค่า threshold ที่เหมาะสมได้ด้วยสมการทางคณิตศาสตร์ได้ เพราะขึ้นกับลักษณะพฤติกรรมการร้องขอข้อมูล, ขนาดข้อมูล และความถี่ในการร้องขอข้อมูล ดังนั้นการวิเคราะห์หาค่า threshold ที่เหมาะสมจะกระทำได้ด้วยการปรับเปลี่ยนค่า threshold ไปเรื่อยๆ [30] และทำการพิจารณาถึงค่าของ Hit Rate และ Byte-Hit Rate ที่ได้จากการปรับเปลี่ยนค่า threshold ไปเรื่อยๆ โดยการพิจารณาจะถูกพิจารณาตามเงื่อนไข 2 ข้อนี้

1. ค่า Hit Rate ของอัลกอริธึม MRASM ต้องเพิ่มขึ้นเมื่อเปรียบเทียบกับค่า Hit Rate ของอัลกอริธึม RASM ซึ่งสามารถแทนได้ด้วยค่า $\overline{\Delta Hit}$ (ค่า $\overline{\Delta Hit}$ ต้องเป็นบวก) สามารถแสดงสมการได้ดังนี้

$$\Delta Hit = \frac{HitRate_{mrasm} - HitRate_{rasm}}{HitRate_{rasm}} \quad (5.1)$$

$$\overline{\Delta Hit} = \frac{\sum_{i=1}^{N} \Delta Hit_i}{N} \quad (5.2)$$

โดยค่า $HitRate_{mrasm}$ คือ ค่า Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลแบบ MRASM

$HitRate_{rasm}$ คือ ค่า Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลแบบ RASM

ΔHit_i คือ ค่าผลต่างระหว่าง Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลแบบ MRASM กับ Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลที่นำเสนอเทียบกับ Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลแบบ RASM ในการร้องขอครั้งที่ i

$\overline{\Delta Hit}$ คือ ค่าผลต่างเฉลี่ยระหว่าง Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลแบบ MRASM กับ Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลที่นำเสนอเทียบกับ Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลแบบ RASM

2. ค่า Byte Hit Rate ของอัลกอริธึม MRASM ต้องเพิ่มขึ้นเมื่อเปรียบเทียบกับค่า Byte Hit Rate ของอัลกอริธึม RASM ซึ่งสามารถแทนได้ด้วยค่า $\overline{\Delta Byte}$ (ค่า $\overline{\Delta Byte}$ ต้องเป็นบวก) สามารถแสดงสมการได้ดังนี้

$$\Delta Byte = \frac{ByteHitRate_{mrasm} - ByteHitRate_{rasm}}{ByteHitRate_{rasm}} \quad (5.3)$$

$$\overline{\Delta Byte} = \frac{\sum_{i=1}^{N} \Delta Byte_i}{N} \quad (5.4)$$

โดยค่า

$BytehitRate_{mrasm}$ คือ ค่า Byte Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลแบบ MRASM

$BytehitRate_{rasm}$ คือ ค่า Byte Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลแบบ RASM

$\Delta Byte_i$ คือ ค่าผลต่างระหว่าง Byte Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลแบบ MRASM กับ Byte Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลแบบ RASM ในการร้องขอครั้งที่ i

$\overline{\Delta Byte}$ คือ ค่าผลต่างเฉลี่ยระหว่าง Byte Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลแบบ MRASM กับ Byte Hit Rate ของอัลกอริธึมการแทนที่ข้อมูลแบบ RASM

N คือ จำนวนการร้องขอทั้งหมด

ในการวิเคราะห์หาค่า Threshold ที่เหมาะสม ได้ทำการเปลี่ยนค่า Threshold ตั้งแต่ 1 Kbytes ไปจนถึง 1 Mbytes โดยการเพิ่มของค่า Threshold จะเพิ่มในอัตราส่วนแบบ Semi-log และหากค่าผลต่างเฉลี่ยที่ได้มีค่าเป็นบวกแสดงว่าค่า Threshold นั้นให้ผลที่ดีกว่าอัลกอริธึมเดิม แต่หากค่าผลต่างเฉลี่ยที่ได้มีค่าติดลบจะถือว่าค่า Threshold นั้นให้ผลที่ต่ำกว่าอัลกอริธึมเดิม และจากเงื่อนไขที่กำหนด 2 ข้อข้างบน เมื่อนำมาพิจารณากับค่าเฉลี่ยของอัลกอริธึมการแทนที่ข้อมูลแต่ละแบบ จะได้คุณลักษณะของค่า Threshold ที่เหมาะสมเมื่อเปรียบเทียบกับอัลกอริธึมการแทนที่ข้อมูลแบบ RASM เมื่อพิจารณาจากเงื่อนไขข้อที่ 1 และ 2 หากนำค่าเฉลี่ยของผลต่างของ Hit Rate ($\overline{\Delta Hit}$ ซึ่งต่อไปจะใช้สัญลักษณ์ $\overline{\Delta H_i}$ แทน) และของ Byte Hit Rate ($\overline{\Delta Byte}$ ซึ่งต่อไปจะใช้ $\overline{\Delta H_b}$ แทน) มาบวกกันแล้วให้ค่ามากที่สุด (ΔH_{i+b} (max)) จะแสดงถึงค่า Threshold ที่ให้ประสิทธิภาพมากที่สุด

$$\Delta H_{i+b} = \overline{\Delta Hit} + \overline{\Delta Byte} \quad (5.5)$$

ในการวิเคราะห์หาค่า Threshold ที่เหมาะสมจะใช้ข้อมูลจาก 3 แหล่งดังนี้

1. Clarknet โดยกำหนดขนาดของแคชไว้ที่ 32, 64 และ 128 Mbytes
2. Nasa โดยกำหนดขนาดของแคชไว้ที่ 32, 64 และ 128 Mbytes

3. Calgory โดยกำหนดขนาดของแคชไว้ที่ 32, 64 และ 128 Mbytes

5.2 การทดสอบข้อมูลจาก Clarknet

ข้อมูลไฟล์ Log ที่ได้จากการรวบรวมของเว็บเซิร์ฟเวอร์ของ Clarknet มีปริมาณการร้องขอ โดยแสดงในตารางต่อไปนี้

ตารางที่ 5.1 แสดงจำนวนการร้องขอข้อมูล

ข้อมูลที่ใช้ทดสอบ	จำนวน (ครั้ง)
จำนวนครั้งในการร้องขอ	1,465,049
จำนวนครั้งในการร้องขอที่ไม่ซ้ำกัน	35,365

ซึ่งสามารถแบ่งเปอร์เซ็นต์การร้องขอตามขนาดของข้อมูลตามตารางข้างล่างนี้

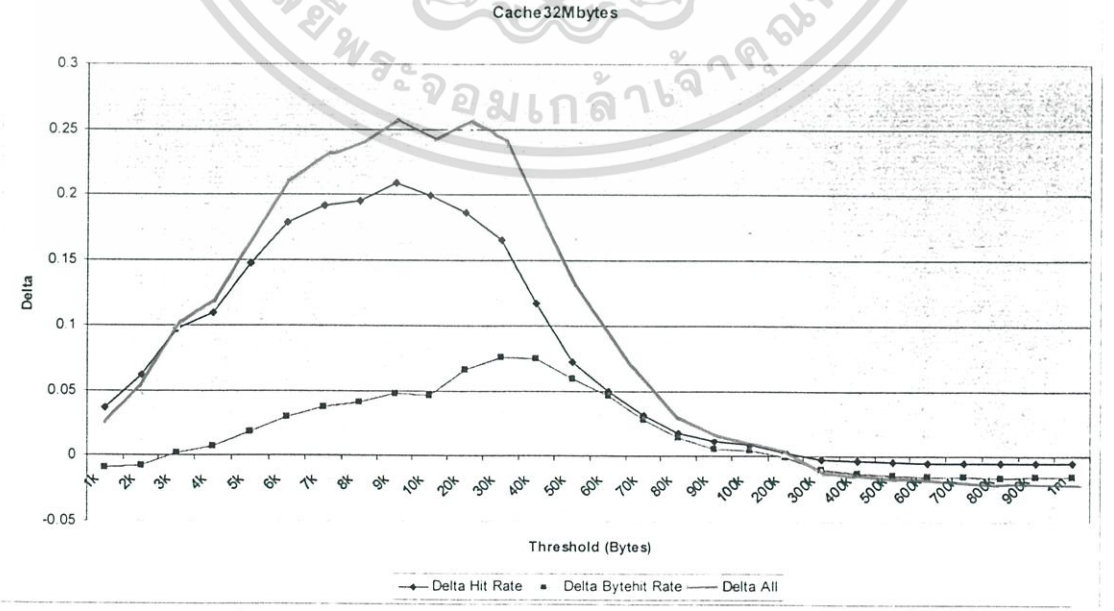
ตารางที่ 5.2 แสดงเปอร์เซ็นต์การร้องขอแบ่งตามช่วงขนาดของข้อมูล

ช่วงขนาดข้อมูล (ไบต์)	จำนวนการร้องขอ (ครั้ง)	จำนวนการร้องขอ (%)	จำนวนการร้องขอเทียบตามขนาดข้อมูล (ไบต์)	จำนวนการร้องขอเทียบตามขนาดข้อมูล (%)
0 - 1K	324,835	22.17	209,158,113	1.45
1K - 10K	799,278	54.56	3,544,644,990	24.56
10K - 100K	334,330	22.82	9,126,466,719	63.24
100K - 1M	6,473	0.44	1,141,927,683	7.91
Over 1M	133	0.01	408,939,965	2.83
Summary	1,465,049	100.00	14,431,137,470	100.00

โดยได้จำลองการทำงาน ในการวิเคราะห์หาค่า threshold ที่เหมาะสม ได้ทำการเปลี่ยนค่า threshold ตั้งแต่ 1 Kbytes ไปจนถึง 1 Mbytes โดยการเพิ่มของค่า threshold จะเพิ่มในอัตราส่วนแบบ Semi-log เป็นจำนวนทั้งสิ้น 28 ระดับ และปรับเปลี่ยนแคชที่ 32, 64 และ 128 Mbytes ซึ่งผลการจำลองการทำได้สามารถแสดงในตาราง และกราฟต่อไปนี้

ตารางที่ 5.3 แสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ของข้อมูลจาก Clarknet โดยขนาดแคชเท่ากับ 32 Mbytes

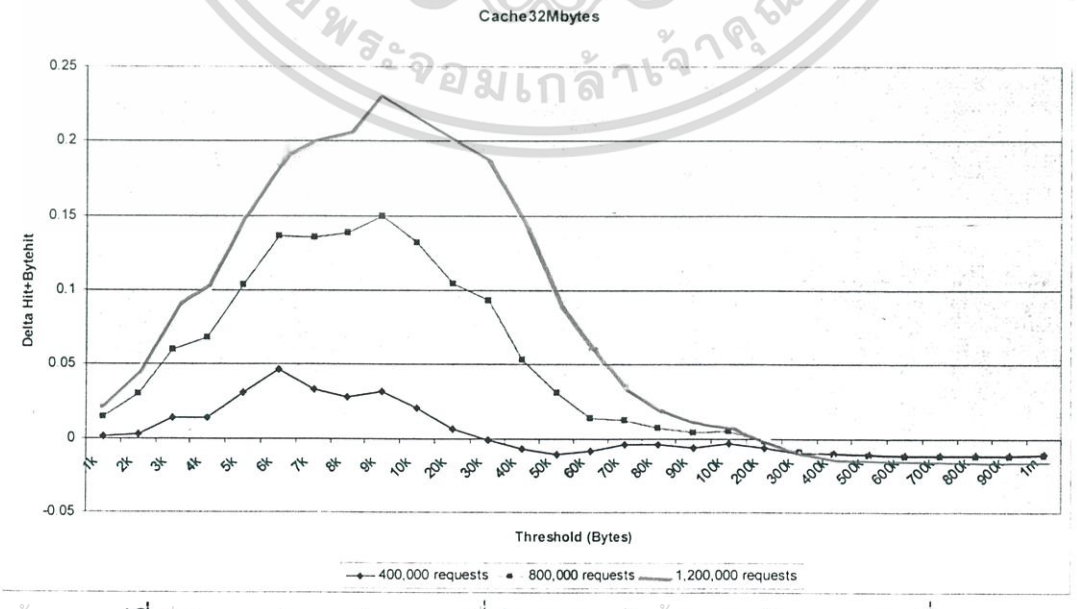
Threshold	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}
1Kbytes	0.036644	-0.00892	0.027726
2Kbytes	0.061663	-0.00715	0.05451
3Kbytes	0.097506	0.00183	0.099336
4Kbytes	0.10903	0.006898	0.115928
5Kbytes	0.14763	0.018601	0.166231
6Kbytes	0.17889	0.029931	0.208821
7Kbytes	0.19166	0.037432	0.229092
8Kbytes	0.195	0.040558	0.235558
9Kbytes	0.20919	0.047896	0.257086
10Kbytes	0.19984	0.046542	0.246382
20Kbytes	0.18616	0.066403	0.252563
30Kbytes	0.16565	0.075295	0.240945
40Kbytes	0.11686	0.074865	0.191725
50Kbytes	0.072516	0.059221	0.131737
60Kbytes	0.049299	0.045985	0.095284
70Kbytes	0.031602	0.027687	0.059289
80Kbytes	0.017394	0.013737	0.031131
90Kbytes	0.011466	0.005122	0.016588
100Kbytes	0.008988	0.004688	0.013676
200Kbytes	0.002832	-0.00029	0.002543
300Kbytes	-0.00195	-0.00994	-0.01189
400Kbytes	-0.00357	-0.01263	-0.0162
500Kbytes	-0.00438	-0.01409	-0.01847
600Kbytes	-0.005	-0.01559	-0.0206
700Kbytes	-0.00501	-0.01563	-0.02064
800Kbytes	-0.00529	-0.01624	-0.02152
900Kbytes	-0.00502	-0.0157	-0.02072
1Mbytes	-0.00518	-0.01564	-0.02082



รูปที่ 5.1 กราฟแสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ของตารางที่ 5.3

ตารางที่ 5.4 แสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ซึ่งมีปริมาณการร้องขอ400,000 ครั้ง, 800,000 ครั้ง และ 1,200,000 ครั้งของข้อมูลจาก Clarknet โดยขนาดแคชเท่ากับ 32 Mbytes

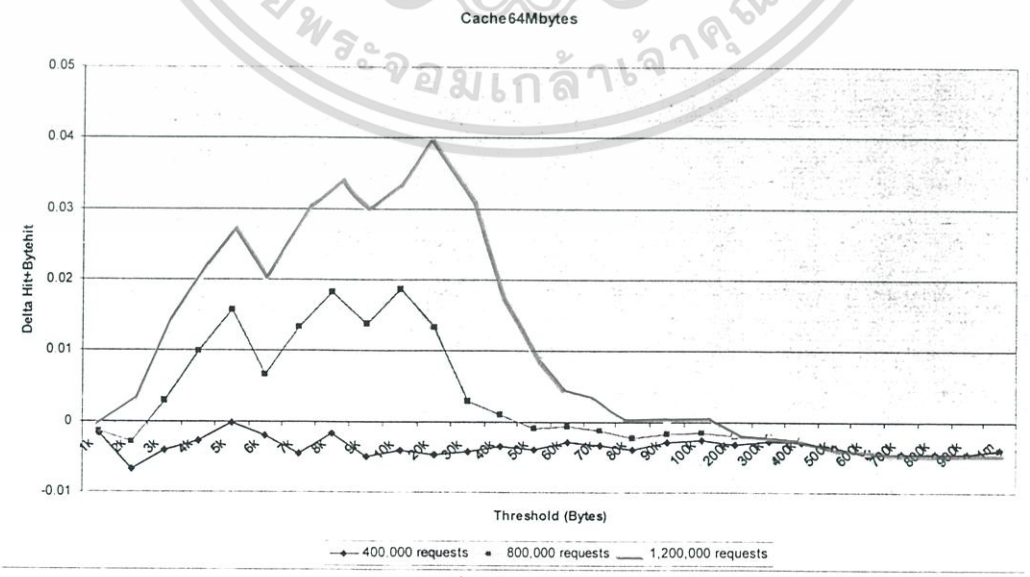
	400,000 requests			800,000 requests			1,200,000 requests		
Threshold	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}
1Kbytes	0.009231	-0.00806	0.001166	0.02387	-0.00899	0.014877	0.032756	-0.00898	0.023779
2Kbytes	0.011978	-0.00926	0.002713	0.038754	-0.00836	0.030399	0.054694	-0.00757	0.047122
3Kbytes	0.018302	-0.00463	0.01367	0.061331	-0.0014	0.059927	0.086404	0.000845	0.087249
4Kbytes	0.017384	-0.00322	0.014166	0.065479	0.002585	0.068064	0.095486	0.005634	0.10112
5Kbytes	0.030863	0.000073	0.030936	0.093293	0.010752	0.104045	0.13088	0.016339	0.147219
6Kbytes	0.042728	0.004114	0.046842	0.11694	0.019328	0.136268	0.16009	0.026857	0.186947
7Kbytes	0.030889	0.002312	0.033201	0.11387	0.021784	0.135654	0.1672	0.032719	0.199919
8Kbytes	0.02571	0.001966	0.027676	0.11468	0.023988	0.138668	0.16975	0.035599	0.205349
9Kbytes	0.028828	0.002486	0.031314	0.12193	0.027718	0.149648	0.18188	0.041796	0.223676
10Kbytes	0.020859	-0.00064	0.020219	0.10819	0.024052	0.132242	0.17046	0.039679	0.210139
20Kbytes	0.010137	-0.00377	0.006366	0.079904	0.024853	0.104757	0.14853	0.051917	0.200447
30Kbytes	0.007194	-0.00854	-0.00134	0.067066	0.02588	0.092946	0.13071	0.058014	0.188724
40Kbytes	0.002847	-0.00976	-0.00691	0.037195	0.016328	0.053523	0.088912	0.054326	0.143238
50Kbytes	0.001861	-0.01258	-0.01072	0.022781	0.008346	0.031127	0.053506	0.039765	0.093271
60Kbytes	0.002029	-0.01054	-0.00851	0.013027	0.001029	0.014056	0.032171	0.024104	0.056275
70Kbytes	0.00285	-0.0065	-0.00365	0.011031	0.001248	0.012279	0.022077	0.014139	0.036216
80Kbytes	0.001615	-0.00529	-0.00367	0.006463	0.000552	0.007015	0.012289	0.007324	0.019613
90Kbytes	0.00067	-0.0066	-0.00593	0.005121	-0.00064	0.004478	0.008473	0.002018	0.01049
100Kbytes	0.000406	-0.00344	-0.00303	0.003862	0.001228	0.005089	0.006879	0.003396	0.010275
200Kbytes	-0.00059	-0.00577	-0.00635	0.000783	-0.0023	-0.00152	0.00198	-0.00043	0.001552
300Kbytes	-0.00176	-0.00809	-0.00984	-0.00164	-0.00749	-0.00912	-0.00185	-0.00878	-0.01062
400Kbytes	-0.00206	-0.008	-0.01006	-0.00226	-0.00818	-0.01043	-0.00303	-0.01051	-0.01353
500Kbytes	-0.00214	-0.00854	-0.01069	-0.00248	-0.00872	-0.0112	-0.0036	-0.01163	-0.01523
600Kbytes	-0.00213	-0.00913	-0.01126	-0.00254	-0.00936	-0.0119	-0.00397	-0.01279	-0.01676
700Kbytes	-0.00214	-0.00918	-0.01132	-0.00255	-0.0094	-0.01195	-0.00398	-0.01283	-0.01681
800Kbytes	-0.00215	-0.00922	-0.01137	-0.00256	-0.00949	-0.01205	-0.00408	-0.01308	-0.01716
900Kbytes	-0.00215	-0.00923	-0.01137	-0.00256	-0.00947	-0.01203	-0.00399	-0.0129	-0.01689
1Mbytes	-0.00216	-0.00873	-0.01088	-0.00256	-0.00917	-0.01173	-0.00404	-0.01272	-0.01676



รูปที่ 5.2 กราฟแสดงค่า ΔH_{i+b} ที่ปริมาณการร้องขอ 3 ระดับของตารางที่ 5.4

ตารางที่ 5.5 แสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ซึ่งมีปริมาณการร้องขอ400,000 ครั้ง, 800,000 ครั้ง และ 1,200,000 ครั้งของข้อมูลจาก Clarknet โดยขนาดแคชเท่ากับ 64 Mbytes

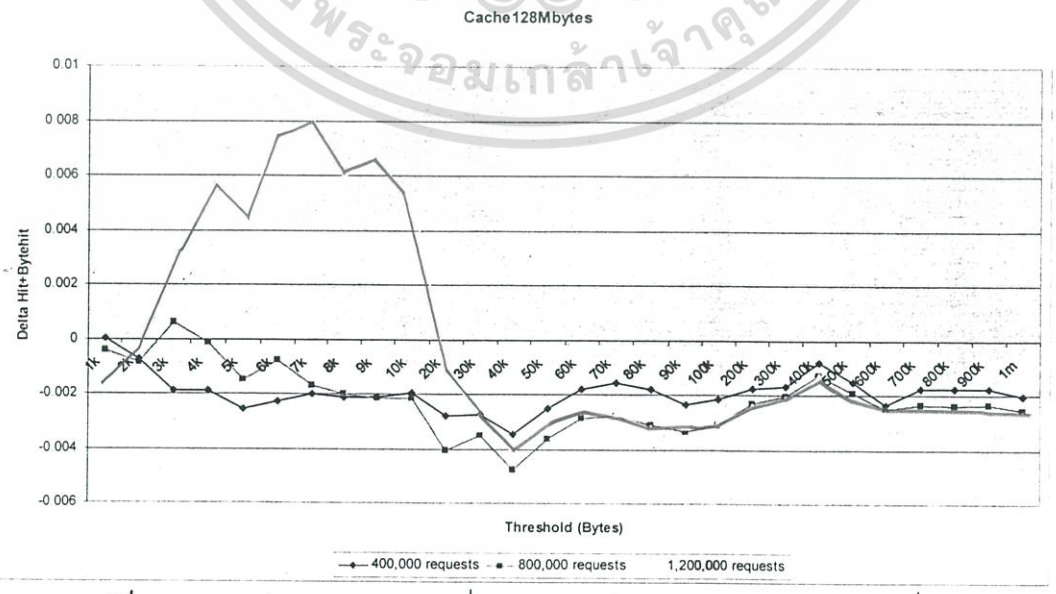
	400,000 requests			800,000 requests			1,200,000 requests		
Threshold	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}
1Kbytes	-0.00118	-0.00044	-0.00162	-2.4E-06	-0.00144	-0.00144	0.001615	-0.00154	7.93E-05
2Kbytes	-0.00159	-0.00522	-0.00681	0.002207	-0.0051	-0.00289	0.007891	-0.00472	0.003171
3Kbytes	-0.00139	-0.00268	-0.00407	0.005399	-0.00249	0.002905	0.014254	-0.00189	0.012364
4Kbytes	0.001007	-0.00373	-0.00273	0.012201	-0.00228	0.009923	0.021726	-0.00122	0.020506
5Kbytes	0.002012	-0.00225	-0.00024	0.015702	3.05E-05	0.015733	0.025124	0.001688	0.026812
6Kbytes	0.001895	-0.00392	-0.00202	0.009493	-0.00293	0.00656	0.020454	-0.00068	0.019771
7Kbytes	-0.00043	-0.00406	-0.00449	0.014623	-0.00135	0.013272	0.027808	0.001055	0.028863
8Kbytes	0.002531	-0.0042	-0.00166	0.018894	-0.00065	0.018244	0.030467	0.001789	0.032256
9Kbytes	-2.9E-05	-0.00487	-0.0049	0.01538	-0.00166	0.013725	0.029176	0.001113	0.030289
10Kbytes	0.000782	-0.0049	-0.00412	0.019402	-0.00076	0.01864	0.031113	0.001898	0.033011
20Kbytes	8.65E-05	-0.00478	-0.00469	0.012882	0.000454	0.013336	0.031255	0.008218	0.039473
30Kbytes	-0.00076	-0.0034	-0.00417	0.004352	-0.00144	0.00291	0.022397	0.010128	0.032525
40Kbytes	-0.00012	-0.00331	-0.00342	0.003062	-0.00214	0.000921	0.013188	0.006102	0.01929
50Kbytes	-1.5E-05	-0.00389	-0.00391	0.002473	-0.00342	-0.00094	0.00736	0.001189	0.008549
60Kbytes	-5.2E-05	-0.00281	-0.00287	0.00169	-0.0023	-0.00061	0.004334	0.000717	0.00505
70Kbytes	-0.00012	-0.00317	-0.00329	0.001353	-0.00253	-0.00117	0.003488	0.000382	0.003871
80Kbytes	-0.00017	-0.00368	-0.00385	0.000921	-0.00319	-0.00227	0.002166	-0.00146	0.000702
90Kbytes	-0.00016	-0.00267	-0.00284	0.000619	-0.00236	-0.00174	0.001555	-0.00117	0.00039
100Kbytes	-2.7E-05	-0.0026	-0.00263	0.000807	-0.0023	-0.00149	0.001802	-0.00115	0.000656
200Kbytes	-0.00039	-0.00276	-0.00315	-8.5E-06	-0.00199	-0.002	3.62E-06	-0.00152	-0.00152
300Kbytes	-0.00051	-0.00224	-0.00276	-0.00037	-0.00178	-0.00215	-0.00039	-0.00155	-0.00195
400Kbytes	-0.00068	-0.00221	-0.00289	-0.00073	-0.00179	-0.00251	-0.00079	-0.00154	-0.00233
500Kbytes	-0.00061	-0.00305	-0.00367	-0.00078	-0.00282	-0.00359	-0.00107	-0.00296	-0.00403
600Kbytes	-0.00064	-0.00397	-0.00461	-0.00074	-0.00341	-0.00415	-0.00094	-0.00319	-0.00414
700Kbytes	-0.00075	-0.00375	-0.0045	-0.00096	-0.00367	-0.00463	-0.00121	-0.00375	-0.00496
800Kbytes	-0.00075	-0.00375	-0.0045	-0.00097	-0.00368	-0.00465	-0.00122	-0.00377	-0.00499
900Kbytes	-0.00075	-0.00375	-0.0045	-0.00099	-0.00367	-0.00466	-0.00132	-0.00379	-0.00511
1Mbytes	-0.00072	-0.00314	-0.00386	-0.00095	-0.00316	-0.00411	-0.00127	-0.00347	-0.00474



รูปที่ 5.3 กราฟแสดงค่า ΔH_{i+b} ที่ปริมาณการร้องขอ 3 ระดับของตารางที่ 5.5

ตารางที่ 5.6 แสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ซึ่งมีปริมาณการร้องขอ400,000 ครั้ง, 800,000 ครั้ง และ 1,200,000 ครั้งของข้อมูลจาก Clarknet ขนาดแคชเท่ากับ 128 Mbytes

	400,000 requests			800,000 requests			1,200,000 requests		
Threshold	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}
1Kbytes	0.00039	-0.00038	6.19E-06	0.002283	-0.0027	-0.00042	0.002767	-0.00438	-0.00161
2Kbytes	-0.00023	-0.00052	-0.00075	0.001869	-0.0027	-0.00084	0.003968	-0.00431	-0.00035
3Kbytes	4.92E-05	-0.00191	-0.00186	0.003805	-0.00318	0.000626	0.006885	-0.00419	0.002693
4Kbytes	2.31E-06	-0.00189	-0.00188	0.002842	-0.00297	-0.00013	0.00843	-0.00322	0.005205
5Kbytes	-3.9E-05	-0.00251	-0.00255	0.002337	-0.00381	-0.00147	0.008433	-0.00401	0.004423
6Kbytes	8.63E-06	-0.0023	-0.00229	0.00275	-0.00354	-0.00079	0.010843	-0.00316	0.007681
7Kbytes	8.55E-07	-0.002	-0.00199	0.001764	-0.00344	-0.00167	0.010665	-0.00273	0.007933
8Kbytes	4.07E-06	-0.00213	-0.00212	0.001555	-0.00355	-0.002	0.009035	-0.00299	0.006046
9Kbytes	2.65E-05	-0.00216	-0.00213	0.001487	-0.00366	-0.00217	0.009357	-0.00297	0.006391
10Kbytes	1.94E-05	-0.00197	-0.00195	0.001272	-0.00343	-0.00216	0.007947	-0.00267	0.005274
20Kbytes	-0.00024	-0.00256	-0.0028	0.000254	-0.0043	-0.00405	0.003205	-0.00437	-0.00116
30Kbytes	-0.00011	-0.00263	-0.00274	0.000244	-0.00376	-0.00352	0.001473	-0.00421	-0.00273
40Kbytes	-7.8E-05	-0.00338	-0.00346	0.000406	-0.00517	-0.00476	0.001452	-0.00557	-0.00412
50Kbytes	-9.3E-05	-0.00242	-0.00251	0.000172	-0.00379	-0.00362	0.001047	-0.00397	-0.00292
60Kbytes	-8.1E-05	-0.00174	-0.00182	0.000123	-0.00301	-0.00288	0.000734	-0.00326	-0.00252
70Kbytes	-9.5E-05	-0.00145	-0.00155	6.36E-05	-0.00285	-0.00279	0.000592	-0.00331	-0.00272
80Kbytes	-7.7E-05	-0.00174	-0.00182	3.21E-05	-0.00316	-0.00313	0.000469	-0.00368	-0.00321
90Kbytes	-9E-05	-0.00226	-0.00234	6.34E-05	-0.0034	-0.00333	0.000482	-0.00363	-0.00315
100Kbytes	-7.8E-05	-0.0021	-0.00217	3.8E-05	-0.00314	-0.00311	0.000449	-0.00346	-0.00301
200Kbytes	-7E-05	-0.00168	-0.00175	-8.4E-05	-0.00222	-0.0023	7.05E-05	-0.00251	-0.00244
300Kbytes	-7E-05	-0.00162	-0.00169	-0.0001	-0.00195	-0.00205	-5.5E-05	-0.00212	-0.00217
400Kbytes	-5.6E-05	-0.00074	-0.0008	-9.6E-05	-0.00118	-0.00128	-7.3E-05	-0.00124	-0.00131
500Kbytes	-7E-05	-0.00146	-0.00153	-0.00013	-0.00179	-0.00192	-8.4E-05	-0.00217	-0.00225
600Kbytes	-5.5E-05	-0.00229	-0.00235	-3.6E-05	-0.00248	-0.00252	-5.3E-07	-0.00265	-0.00265
700Kbytes	-6.5E-05	-0.00169	-0.00176	-6.8E-05	-0.00229	-0.00235	-4.4E-05	-0.00261	-0.00266
800Kbytes	-6.4E-05	-0.00169	-0.00175	-6.2E-05	-0.00234	-0.0024	-2.1E-05	-0.00268	-0.0027
900Kbytes	-6.5E-05	-0.00169	-0.00176	-6.8E-05	-0.0023	-0.00237	-4.3E-05	-0.00263	-0.00268
1Mbytes	-7.1E-05	-0.00197	-0.00204	-8.2E-05	-0.00247	-0.00255	-6.2E-05	-0.00267	-0.00273



รูปที่ 5.4 กราฟแสดงค่า ΔH_{i+b} ที่ปริมาณการร้องขอ 3 ระดับของตารางที่ 5.6

จากตารางที่ 5.3 และรูปที่ 5.1 หน้า 86 แสดงถึงค่าผลต่างเฉลี่ย $\overline{\Delta H_t}$, $\overline{\Delta H_h}$ และ ΔH_{t+h} ของอัลกอริธึม MRASM กับอัลกอริธึม RASM จากข้อมูลของ Clarknet หากค่าผลต่างเฉลี่ยที่ได้มีค่าเป็นบวกแสดงว่าค่า threshold นั้นให้ผลที่ดีกว่าอัลกอริธึมเดิม แต่หากค่าผลต่างเฉลี่ยที่ได้มีค่าติดลบจะถือว่าค่า threshold นั้นให้ผลที่ต่ำกว่าอัลกอริธึมเดิม เมื่อนำมาพิจารณากับค่าเฉลี่ยของอัลกอริธึมการแทนที่ข้อมูลแต่ละแบบ จะได้คุณลักษณะของค่า threshold ที่เหมาะสม ณ จุดที่ผลรวมของค่าผลต่างเฉลี่ยของ Hit Rate ($\overline{\Delta H_t}$) และค่าผลต่างเฉลี่ยของ Byte Hit Rate ($\overline{\Delta H_h}$) รวมกันแล้วให้ค่ามากที่สุด (ΔH_{t+h} (max)) คือค่า threshold ที่ 9Kbytes

จากตารางที่ 5.4 และรูปที่ 5.2 หน้า 87 แสดงถึงค่าผลต่างเฉลี่ย $\overline{\Delta H_t}$, $\overline{\Delta H_h}$ และ ΔH_{t+h} ของอัลกอริธึม MRASM กับอัลกอริธึม RASM ของแคชขนาด 32Mbytes ซึ่งแบ่งปริมาณการร้องขอข้อมูลเป็น 3 ระดับคือปริมาณการร้องขอข้อมูลน้อย (400,000 requests) ,ปานกลาง (800,000 requests) และมาก (1,200,000 requests) จะเห็นได้ว่าในปริมาณการร้องขอที่น้อยจะได้ค่าผลต่างเฉลี่ยที่น้อย แต่ยังคงเป็นค่าบวกแสดงว่าประสิทธิภาพของอัลกอริธึมที่ปรับปรุงใหม่ดีกว่าของอัลกอริธึมเดิม และเมื่อเพิ่มปริมาณการร้องขอเป็นระดับปานกลาง และมาก จะให้ค่าผลต่างเฉลี่ยที่มากขึ้นตามลำดับ แสดงว่าอัลกอริธึมที่ได้ทำการปรับปรุงใหม่จะให้ผลดีขึ้นในปริมาณการร้องขอที่มากขึ้น

เช่นเดียวกับแคชขนาด 64 Mbytes จากตารางที่ 5.5 และรูปที่ 5.3 หน้า 88 จะเห็นว่าในปริมาณการร้องขอที่น้อยจะได้ค่าผลต่างเฉลี่ยที่น้อย และมีค่าติดลบ โดยค่าผลต่างเฉลี่ยของ Hit Rate ($\overline{\Delta H_t}$) จากตารางที่ 5.3 ยังเป็นบวก แต่ค่าผลต่างเฉลี่ยของ Bytehit Rate($\overline{\Delta H_h}$) เป็นลบแสดงว่าค่า Hit Rate เพิ่มขึ้นจากอัลกอริธึมเดิม แต่ค่า Bytehit Rate ลดลง และเมื่อเพิ่มปริมาณการร้องขอเป็นระดับปานกลาง และมาก จะให้ค่าผลต่างเฉลี่ยที่มากขึ้นตามลำดับจากระดับกราฟที่สูงขึ้นและมีค่าเป็นบวก แสดงว่าอัลกอริธึมที่ได้ทำการปรับปรุงใหม่จะให้ผลดีขึ้นในปริมาณการร้องขอที่มากขึ้น

และในแคชขนาด 128 Mbytes จากตารางที่ 5.6 และรูปที่ 5.4 หน้า 89 จะเห็นว่าในปริมาณการร้องขอน้อยๆจะให้ค่าผลต่างเฉลี่ยที่ได้จะติดลบ โดยค่า Hit Rate เพิ่มขึ้นแต่ค่า Bytehit Rate ลดลงเหมือนในแคชขนาด64 Mbytes และในปริมาณการร้องขอมากๆ ณ จุด threshold ที่ต่ำจะให้ค่าผลต่างเฉลี่ยเป็นบวก แสดงว่าในแคชที่ใหญ่ แต่ปริมาณการร้องขอน้อยประสิทธิภาพของอัลกอริธึมที่ปรับปรุงใหม่จะให้ค่าที่ด้อยกว่าอัลกอริธึมเดิม แม้ว่าจะให้ค่า Hit Rate ที่เพิ่มขึ้นแต่ค่า Bytehit Rate กลับได้ค่าลดลง

ต่อไปเป็นการทดสอบเพื่อวิเคราะห์หาค่า threshold ที่เหมาะสมโดยใช้ข้อมูลการร้องขอจาก Nasa โดยทำการทดสอบด้วยการจำลองขนาดของแคช 32 Mbytes,64 Mbytes และ 128 Mbytes ตามลำดับเหมือนเดิม ดังแสดงในตาราง และกราฟถัดไป

5.3 การทดสอบข้อมูลจาก Nasa

ข้อมูลไฟล์ Log ที่ได้จากการรวบรวมของเว็บเซิร์ฟเวอร์ของ Nasa มีปริมาณการร้องขอ โดยแสดงในตารางต่อไปนี้

ตารางที่ 5.7 แสดงจำนวนการร้องขอข้อมูล

ข้อมูลที่ใช้ทดสอบ	จำนวน (ครั้ง)
จำนวนครั้งในการร้องขอ	1,387,398
จำนวนครั้งในการร้องขอที่ไม่ซ้ำกัน	22,531

ซึ่งสามารถแบ่งเปอร์เซ็นต์การร้องขอตามขนาดของข้อมูลตามตารางข้างล่างนี้

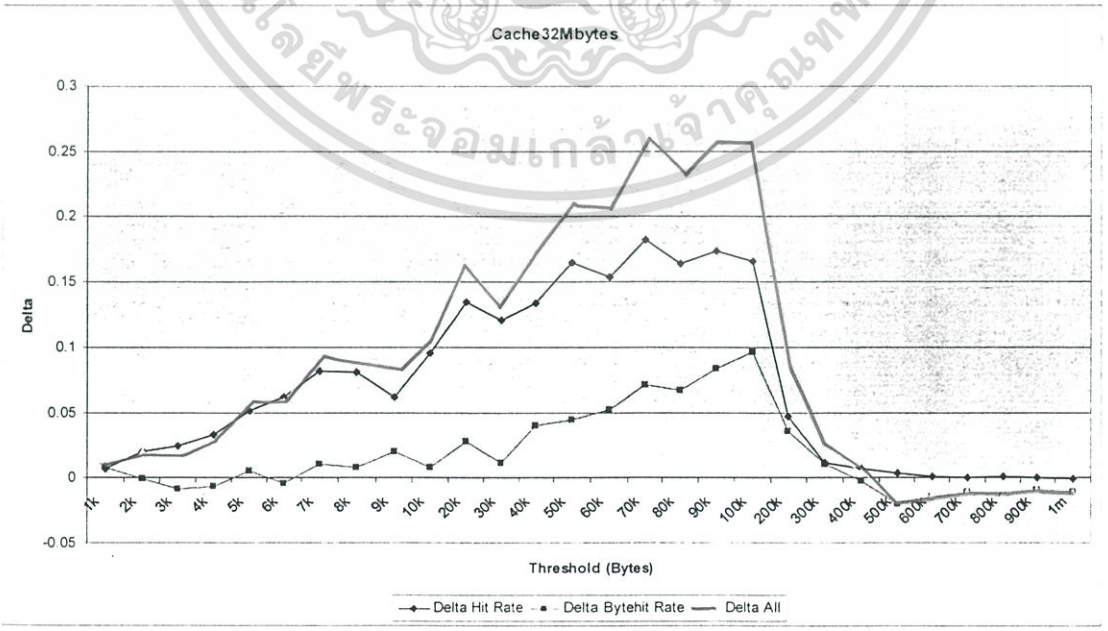
ตารางที่ 5.8 แสดงเปอร์เซ็นต์การร้องขอแบ่งตามช่วงขนาดของข้อมูล

ช่วงขนาดข้อมูล (ไบต์)	จำนวนการร้องขอ (ครั้ง)	จำนวนการร้องขอ (%)	จำนวนการร้องขอเทียบตามขนาดข้อมูล (ไบต์)	จำนวนการร้องขอเทียบตามขนาดข้อมูล (%)
0 - 1K	369,919	26.66	189,190,080	0.71
1K - 10K	653,598	47.11	2,759,967,061	10.37
10K - 100K	316,737	22.83	10,407,074,727	39.11
100K - 1M	45,117	3.25	10,887,322,794	40.92
Over 1M	2,027	0.15	2,364,172,850	8.89
Summary	1,387,398	100.00	26,607,727,512	100.00

โดยได้จำลองการทำงาน ในการวิเคราะห์หาค่า threshold ที่เหมาะสม ได้ทำการเปลี่ยนค่า threshold ตั้งแต่ 1 Kbytes ไปจนถึง 1 Mbytes โดยการเพิ่มของค่า threshold จะเพิ่มในอัตราส่วนแบบ Semi-log เป็นจำนวนทั้งสิ้น 28 ระดับ และปรับเปลี่ยนแคชที่ 32, 64 และ 128 Mbytes ซึ่งผลการจำลองการทำได้สามารถแสดงในตาราง และกราฟต่อไปนี้

ตารางที่ 5.9 แสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ของข้อมูลจากNasa โดยขนาดแคชเท่ากับ32 Mbytes

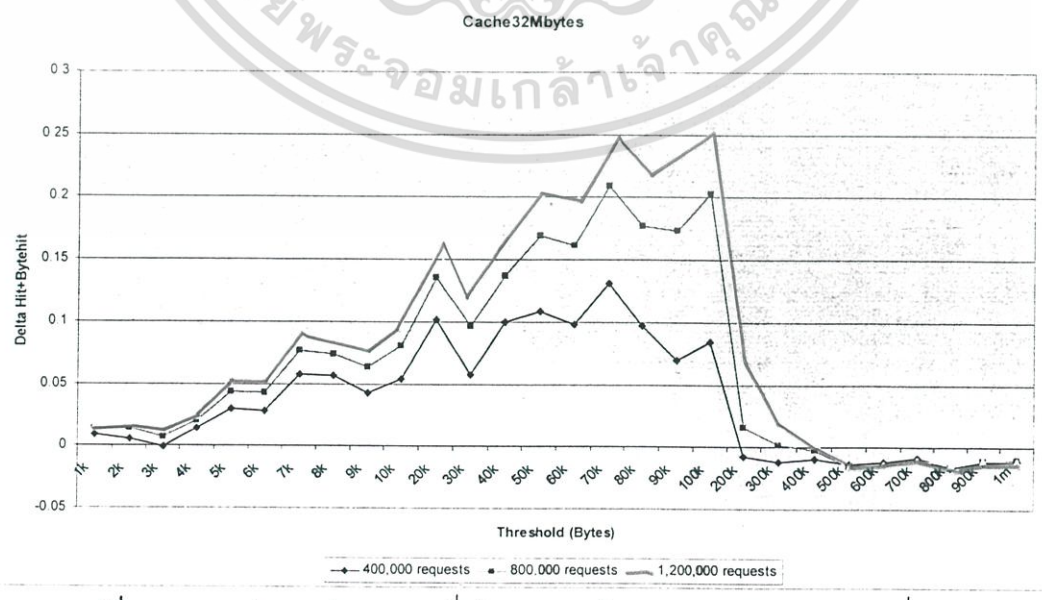
Threshold	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}
1Kbytes	0.006933	0.008449	0.015382
2Kbytes	0.020358	-0.00022	0.020135
3Kbytes	0.024106	-0.00862	0.015486
4Kbytes	0.032953	-0.00678	0.026171
5Kbytes	0.051046	0.005511	0.056557
6Kbytes	0.062053	-0.00405	0.058007
7Kbytes	0.081563	0.010546	0.092109
8Kbytes	0.080908	0.008045	0.088953
9Kbytes	0.062171	0.020546	0.082717
10Kbytes	0.095369	0.007898	0.103267
20Kbytes	0.13431	0.02796	0.16227
30Kbytes	0.12028	0.011832	0.132112
40Kbytes	0.13383	0.040053	0.173883
50Kbytes	0.1646	0.044245	0.208845
60Kbytes	0.1535	0.05208	0.20558
70Kbytes	0.18256	0.071548	0.254108
80Kbytes	0.16409	0.067159	0.231249
90Kbytes	0.17338	0.083289	0.256669
100Kbytes	0.16576	0.096387	0.262147
200Kbytes	0.047425	0.036028	0.083453
300Kbytes	0.011263	0.010329	0.021592
400Kbytes	0.007167	-0.00247	0.004702
500Kbytes	0.003582	-0.01962	-0.01604
600Kbytes	0.001534	-0.01458	-0.01304
700Kbytes	-0.00012	-0.01083	-0.01095
800Kbytes	0.000919	-0.01337	-0.01246
900Kbytes	-4.6E-05	-0.00934	-0.00939
1Mbytes	-0.00085	-0.01099	-0.01184



รูปที่ 5.5 กราฟแสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ของตารางที่ 5.9

ตารางที่ 5.10 แสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ซึ่งมีปริมาณการร้องขอ400,000 ครั้ง, 800,000 ครั้ง และ 1,200,000 ครั้งของข้อมูลจาก Nasa ขนาดแคชเท่ากับ 32 Mbytes

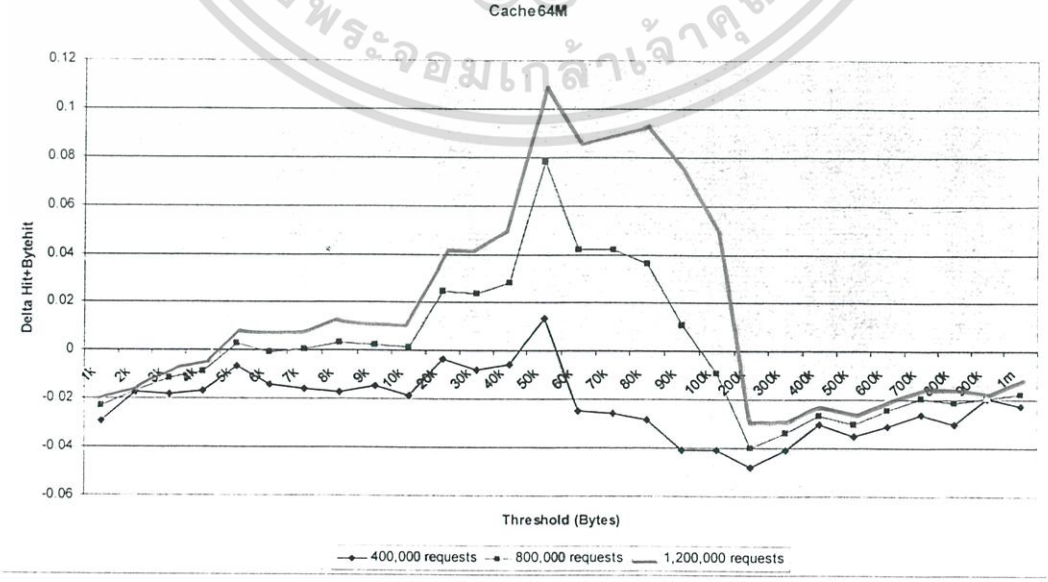
	400,000 requests			800,000 requests			1,200,000 requests		
Threshold	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}
1Kbytes	0.005645	0.003623	0.009268	0.005839	0.007626	0.013465	0.006506	0.008331	0.014837
2Kbytes	0.012529	-0.00725	0.00528	0.016303	-0.00247	0.013829	0.019386	-0.00068	0.018704
3Kbytes	0.012435	-0.01264	-0.00021	0.017545	-0.01059	0.006956	0.022283	-0.00911	0.013177
4Kbytes	0.018723	-0.00427	0.014454	0.025881	-0.00578	0.020103	0.031089	-0.00649	0.024599
5Kbytes	0.028371	0.001347	0.029718	0.039783	0.004069	0.043852	0.048428	0.005247	0.053675
6Kbytes	0.034134	-0.0064	0.027736	0.04822	-0.00527	0.042946	0.058732	-0.00436	0.054371
7Kbytes	0.049033	0.008725	0.057758	0.067154	0.009072	0.076226	0.077858	0.009948	0.087806
8Kbytes	0.053144	0.00371	0.056854	0.068659	0.005533	0.074192	0.077875	0.007216	0.085091
9Kbytes	0.031108	0.011429	0.042537	0.046853	0.016856	0.063709	0.058338	0.019507	0.077845
10Kbytes	0.055309	-0.00097	0.054341	0.076451	0.004238	0.080689	0.09078	0.00701	0.09779
20Kbytes	0.08598	0.015581	0.101561	0.11323	0.022574	0.135804	0.12939	0.02674	0.15613
30Kbytes	0.061249	-0.0039	0.05735	0.092952	0.003425	0.096377	0.11398	0.009802	0.123782
40Kbytes	0.078265	0.021665	0.09993	0.10632	0.031038	0.137358	0.12721	0.037685	0.164895
50Kbytes	0.086078	0.022852	0.10893	0.1334	0.03558	0.16898	0.15829	0.042298	0.200588
60Kbytes	0.070848	0.02718	0.098028	0.11908	0.042178	0.161258	0.14574	0.049861	0.195601
70Kbytes	0.084511	0.046922	0.131433	0.14581	0.063537	0.209347	0.1752	0.070196	0.245396
80Kbytes	0.067506	0.02973	0.097236	0.12417	0.052918	0.177088	0.15516	0.064054	0.219214
90Kbytes	0.048695	0.021066	0.069761	0.1139	0.059416	0.173316	0.15854	0.078134	0.236674
100Kbytes	0.051813	0.032318	0.084131	0.12376	0.079599	0.203359	0.15654	0.093718	0.250258
200Kbytes	0.003547	-0.01137	-0.00782	0.013567	0.002028	0.015595	0.037477	0.026373	0.06385
300Kbytes	0.002075	-0.01377	-0.01169	0.006004	-0.00445	0.001559	0.009539	0.006252	0.015791
400Kbytes	0.001846	-0.01124	-0.00939	0.004313	-0.00656	-0.00225	0.006294	-0.00381	0.002481
500Kbytes	0.001071	-0.01469	-0.01362	0.00252	-0.0177	-0.01518	0.003297	-0.01924	-0.01594
600Kbytes	-0.00033	-0.01187	-0.0122	0.000502	-0.01478	-0.01428	0.001223	-0.01496	-0.01373
700Kbytes	-0.00053	-0.00831	-0.00884	-0.00035	-0.00988	-0.01023	-0.00018	-0.01034	-0.01052
800Kbytes	0.000126	-0.01675	-0.01662	0.000616	-0.01645	-0.01583	0.000866	-0.01417	-0.0133
900Kbytes	-0.00023	-0.01127	-0.0115	-7.3E-05	-0.01092	-0.01099	-3E-05	-0.00969	-0.00972
1Mbytes	-0.00066	-0.01012	-0.01078	-0.00072	-0.0096	-0.01032	-0.00081	-0.01053	-0.01135



รูปที่ 5.6 กราฟแสดงค่า ΔH_{i+b} ที่ปริมาณการร้องขอ 3 ระดับของตารางที่ 5.10

ตารางที่ 5.11 แสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ซึ่งมีปริมาณการร้องขอ400,000 ครั้ง, 800,000 ครั้ง และ 1,200,000 ครั้งของข้อมูลจาก Nasa ขนาดแคชเท่ากับ 64 Mbytes

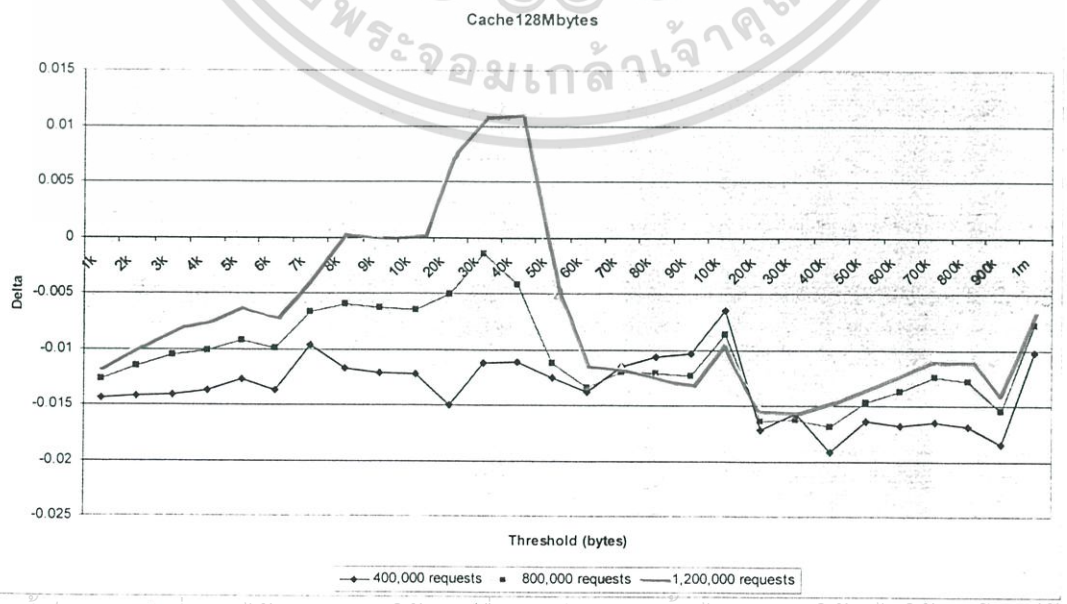
	400,000 requests			800,000 requests			1,200,000 requests		
Threshold	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}
1Kbytes	0.005379	-0.03446	-0.02908	0.006323	-0.02934	-0.02301	0.006887	-0.02605	-0.01916
2Kbytes	0.00609	-0.02327	-0.01718	0.007826	-0.02475	-0.01693	0.009525	-0.024	-0.01448
3Kbytes	0.009871	-0.02782	-0.01795	0.014085	-0.02574	-0.01166	0.01712	-0.02337	-0.00625
4Kbytes	0.014121	-0.03074	-0.01662	0.019947	-0.02861	-0.00866	0.023815	-0.02696	-0.00315
5Kbytes	0.019231	-0.02592	-0.00669	0.027483	-0.02469	0.002794	0.032704	-0.02344	0.009264
6Kbytes	0.01437	-0.02853	-0.01416	0.023387	-0.02408	-0.00069	0.030039	-0.02111	0.008932
7Kbytes	0.016764	-0.03254	-0.01578	0.029881	-0.0292	0.000679	0.037188	-0.02713	0.010063
8Kbytes	0.018295	-0.03565	-0.01736	0.032499	-0.02916	0.003338	0.040519	-0.02479	0.015734
9Kbytes	0.015002	-0.02974	-0.01474	0.028934	-0.02658	0.002355	0.037601	-0.02389	0.013716
10Kbytes	0.012806	-0.03126	-0.01845	0.028371	-0.02683	0.00154	0.037834	-0.02407	0.013763
20Kbytes	0.028044	-0.03125	-0.00321	0.049357	-0.02449	0.024871	0.062024	-0.02033	0.041697
30Kbytes	0.020518	-0.0283	-0.00778	0.044357	-0.02072	0.023635	0.058226	-0.01611	0.042119
40Kbytes	0.027431	-0.03322	-0.00579	0.055335	-0.02693	0.028404	0.070449	-0.02175	0.048702
50Kbytes	0.035295	-0.02189	0.013406	0.077325	0.000701	0.078026	0.097131	0.009699	0.10683
60Kbytes	0.011107	-0.03579	-0.02468	0.050185	-0.00817	0.042016	0.076735	0.006749	0.083484
70Kbytes	0.004869	-0.03057	-0.0257	0.042131	1.76E-06	0.042133	0.070067	0.01842	0.088487
80Kbytes	0.003302	-0.03158	-0.02827	0.037363	-0.00096	0.036407	0.070184	0.023253	0.093437
90Kbytes	0.002981	-0.04394	-0.04095	0.025358	-0.01457	0.010785	0.058085	0.017754	0.075839
100Kbytes	0.000333	-0.04124	-0.04091	0.011787	-0.02081	-0.00903	0.038423	0.010574	0.048997
200Kbytes	0.000305	-0.04845	-0.04815	0.002039	-0.04191	-0.03987	0.003926	-0.03293	-0.02901
300Kbytes	-0.00021	-0.04051	-0.04072	0.000549	-0.0344	-0.03385	0.001256	-0.02913	-0.02788
400Kbytes	0.000274	-0.03036	-0.03008	0.000887	-0.02727	-0.02638	0.001363	-0.0237	-0.02234
500Kbytes	2.39E-05	-0.03511	-0.03508	0.000625	-0.03055	-0.02993	0.001087	-0.02676	-0.02568
600Kbytes	-0.00026	-0.03067	-0.03094	-2E-06	-0.02425	-0.02425	0.000114	-0.02003	-0.01992
700Kbytes	-0.00036	-0.02572	-0.02608	-0.0002	-0.01913	-0.01933	-6.5E-05	-0.01571	-0.01578
800Kbytes	-0.00017	-0.02998	-0.03015	-3.8E-05	-0.02137	-0.02141	6.62E-05	-0.01683	-0.01676
900Kbytes	-0.00018	-0.01941	-0.01959	-0.00011	-0.01945	-0.01956	-1.1E-05	-0.01867	-0.01868
1Mbytes	-0.00042	-0.022	-0.02243	-0.00042	-0.01717	-0.01759	-0.0004	-0.01427	-0.01467



รูปที่ 5.7 กราฟแสดงค่า ΔH_{i+b} ที่ปริมาณการร้องขอ 3 ระดับของตารางที่ 5.11

ตารางที่ 5.12 แสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ซึ่งมีปริมาณการร้องขอ400,000 ครั้ง, 800,000 ครั้ง และ 1,200,000 ครั้งของข้อมูลจาก Nasa ขนาดแคชเท่ากับ 128 Mbytes

	400,000 requests			800,000 requests			1,200,000 requests		
Threshold	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}
1Kbytes	0.001592	-0.01599	-0.0144	0.002367	-0.01513	-0.01276	0.002691	-0.01452	-0.01183
2Kbytes	0.001245	-0.01544	-0.01419	0.003096	-0.01467	-0.01157	0.004384	-0.01418	-0.00979
3Kbytes	0.00117	-0.01526	-0.01409	0.003054	-0.01364	-0.01059	0.005091	-0.01282	-0.00773
4Kbytes	0.001314	-0.01505	-0.01373	0.003552	-0.01373	-0.01018	0.00597	-0.01306	-0.00709
5Kbytes	0.001485	-0.01421	-0.01272	0.00398	-0.01327	-0.00929	0.006693	-0.01269	-0.00599
6Kbytes	0.000699	-0.01441	-0.01371	0.003916	-0.01385	-0.00993	0.006911	-0.01318	-0.00627
7Kbytes	0.000666	-0.01031	-0.00964	0.003952	-0.01068	-0.00673	0.00633	-0.01076	-0.00443
8Kbytes	0.000453	-0.01223	-0.01178	0.005235	-0.01127	-0.00604	0.010151	-0.01039	-0.00024
9Kbytes	0.000445	-0.01255	-0.0121	0.005688	-0.01199	-0.0063	0.011316	-0.01129	2.8E-05
10Kbytes	0.00029	-0.01249	-0.0122	0.005518	-0.01199	-0.00647	0.011308	-0.01097	0.000342
20Kbytes	-6.1E-05	-0.01497	-0.01503	0.006795	-0.01193	-0.00514	0.015332	-0.00917	0.006164
30Kbytes	-0.00014	-0.01113	-0.01126	0.007188	-0.00864	-0.00145	0.016205	-0.00553	0.010678
40Kbytes	-6.1E-05	-0.01108	-0.01114	0.004702	-0.00886	-0.00416	0.01497	-0.00395	0.011021
50Kbytes	0.000126	-0.01267	-0.01255	0.000586	-0.01177	-0.01119	0.004263	-0.00889	-0.00463
60Kbytes	-0.00013	-0.01367	-0.0138	0.000125	-0.01355	-0.01342	0.00121	-0.0127	-0.01149
70Kbytes	-0.0003	-0.01123	-0.01153	-8E-05	-0.01193	-0.01201	0.000373	-0.0119	-0.01153
80Kbytes	-0.00032	-0.01033	-0.01065	-0.0002	-0.01195	-0.01215	3.36E-05	-0.01258	-0.01255
90Kbytes	-0.00048	-0.00989	-0.01037	-0.00033	-0.01199	-0.01232	-9.9E-05	-0.01286	-0.01296
100Kbytes	-0.00035	-0.00618	-0.00652	-0.00026	-0.00839	-0.00865	-6.8E-05	-0.00956	-0.00963
200Kbytes	-0.00015	-0.01704	-0.01719	3.11E-05	-0.01644	-0.01641	0.000182	-0.01544	-0.01526
300Kbytes	-0.00033	-0.01536	-0.01569	-0.00017	-0.01607	-0.01624	-2E-05	-0.01537	-0.01539
400Kbytes	-0.0003	-0.01889	-0.0192	-0.00019	-0.0167	-0.01688	-9E-05	-0.01476	-0.01485
500Kbytes	-0.00025	-0.01616	-0.01641	-9.9E-05	-0.0146	-0.0147	2.68E-05	-0.01323	-0.0132
600Kbytes	-0.00033	-0.01646	-0.01679	-0.00021	-0.01351	-0.01372	-0.00012	-0.01189	-0.012
700Kbytes	-0.00027	-0.01619	-0.01645	-0.00017	-0.01228	-0.01245	-0.00012	-0.01033	-0.01045
800Kbytes	-0.00032	-0.01652	-0.01684	-0.00025	-0.01256	-0.01281	-0.00021	-0.01053	-0.01074
900Kbytes	-0.00034	-0.01809	-0.01843	-0.00026	-0.01523	-0.01549	-0.0002	-0.01355	-0.01375
1Mbytes	-0.00025	-0.00998	-0.01023	-0.00021	-0.00757	-0.00779	-0.00019	-0.00619	-0.00638



รูปที่ 5.8 กราฟแสดงค่า ΔH_{i+b} ที่ปริมาณการร้องขอ 3 ระดับของตารางที่ 5.12

จากตารางที่ 5.9 และรูปที่ 5.5 หน้า 92 แสดงถึงค่าผลต่างเฉลี่ย $\overline{\Delta H_r}$, $\overline{\Delta H_b}$ และ ΔH_{r+b} ของอัลกอริธึม MRASM กับอัลกอริธึม RASM จากข้อมูลของ Nasa หากค่าผลต่างเฉลี่ยที่ได้มีค่าเป็นบวกแสดงว่าค่า threshold นั้นให้ผลที่ดีกว่าอัลกอริธึมเดิม แต่หากค่าผลต่างเฉลี่ยที่ได้มีค่าติดลบจะถือว่าค่า threshold นั้นให้ผลที่ต่ำกว่าอัลกอริธึมเดิม เมื่อนำมาพิจารณากับค่าเฉลี่ยของอัลกอริธึมการแทนที่ข้อมูล จะได้คุณลักษณะของค่า threshold ที่เหมาะสม ณ จุดที่ผลรวมของค่าผลต่างเฉลี่ยของ Hit Rate ($\overline{\Delta H_r}$) และค่าผลต่างเฉลี่ยของ Byte Hit Rate ($\overline{\Delta H_b}$) รวมกันแล้วให้ค่ามากที่สุด (ΔH_{r+b} (max)) คือค่า threshold ที่ 100Kbytes

จากตารางที่ 5.10 และรูปที่ 5.6 หน้า 93 แสดงถึงค่าผลต่างเฉลี่ย $\overline{\Delta H_r}$, $\overline{\Delta H_b}$ และ ΔH_{r+b} ของอัลกอริธึม MRASM กับอัลกอริธึม RASM ของแคชขนาด 32Mbytes ซึ่งแบ่งปริมาณการร้องขอข้อมูลเป็น 3 ระดับคือปริมาณการร้องขอข้อมูลน้อย (400,000 requests) ,ปานกลาง (800,000 requests) และมาก (1,200,000 requests) จะเห็นได้ว่าในปริมาณการร้องขอที่น้อยจะได้ค่าผลต่างเฉลี่ยที่น้อย แต่ยังคงเป็นค่าบวกแสดงว่าประสิทธิภาพของอัลกอริธึมที่ปรับปรุงใหม่ดีกว่าของอัลกอริธึมเดิม และเมื่อเพิ่มปริมาณการร้องขอเป็นระดับปานกลาง และมาก จะให้ค่าผลต่างเฉลี่ยที่มากขึ้นตามลำดับ แสดงว่าอัลกอริธึมที่ได้ทำการปรับปรุงใหม่จะให้ผลดีขึ้นในปริมาณการร้องขอที่มากขึ้น

เช่นเดียวกับแคชขนาด 64 Mbytes จากตารางที่ 5.11 และรูปที่ 5.7 หน้า 94 จะเห็นว่าในปริมาณการร้องขอที่น้อยจะได้ค่าผลต่างเฉลี่ยที่น้อย และมีค่าติดลบ โดยค่าผลต่างเฉลี่ยของ Hit Rate ($\overline{\Delta H_r}$) จากตารางที่ 5.7 ยังเป็นบวก แต่ค่าผลต่างเฉลี่ยของ Bytehit Rate($\overline{\Delta H_b}$) เป็นลบ แสดงว่าค่า Hit Rate เพิ่มขึ้นจากอัลกอริธึมเดิม แต่ค่า Bytehit Rate ลดลง และเมื่อเพิ่มปริมาณการร้องขอเป็นระดับปานกลาง และมาก จะให้ค่าผลต่างเฉลี่ยที่มากขึ้นตามลำดับจากระดับกราฟที่สูงขึ้น และมีค่าเป็นบวก แสดงว่าอัลกอริธึมที่ได้ทำการปรับปรุงใหม่จะให้ผลดีขึ้นในปริมาณการร้องขอที่มากขึ้น

และในแคชขนาด 128 Mbytes จากตารางที่ 5.12 และรูปที่ 5.8 หน้า 95 จะเห็นว่าในปริมาณการร้องขอน้อยๆจะให้ค่าผลต่างเฉลี่ยที่ได้จะติดลบ โดยค่า Hit Rate เพิ่มขึ้นแต่ค่า Bytehit Rate ลดลงเหมือนในแคชขนาด64 Mbytes และในปริมาณการร้องขอมากๆ ณ จุด threshold ที่ต่ำจะให้ค่าผลต่างเฉลี่ยเป็นบวก แสดงว่าในแคชที่ใหญ่แต่ปริมาณการร้องขอน้อยประสิทธิภาพของอัลกอริธึมที่ปรับปรุงใหม่จะให้ค่าที่ด้อยกว่าอัลกอริธึมเดิม แม้ว่าจะให้ค่า Hit Rate ที่เพิ่มขึ้นแต่ค่า Bytehit Rate กลับได้ค่าลดลง

ต่อไปเป็นการทดสอบเพื่อวิเคราะห์หาค่า threshold ที่เหมาะสมโดยใช้ข้อมูลการร้องขอจาก Calgory โดยทำการทดสอบด้วยการจำลองขนาดของแคช 32Mbytes,64 Mbytes และ 128 Mbytes ตามลำดับเหมือนเดิม ดังแสดงในตาราง และกราฟถัดไป

5.4 การทดสอบข้อมูลจาก Calgory

ข้อมูลไฟล์ Log ที่ได้จากการรวบรวมของเว็บเซิร์ฟเวอร์ของ Calgory มีปริมาณการร้องขอ โดยแสดงในตารางต่อไปนี้

ตารางที่ 5.13 แสดงจำนวนการร้องขอข้อมูล

ข้อมูลที่ใช้ทดสอบ	จำนวน (ครั้ง)
จำนวนครั้งในการร้องขอ	567,132
จำนวนครั้งในการร้องขอที่ไม่ซ้ำกัน	30,822

ซึ่งสามารถแบ่งเปอร์เซ็นต์การร้องขอตามขนาดของข้อมูลตามตารางข้างล่างนี้

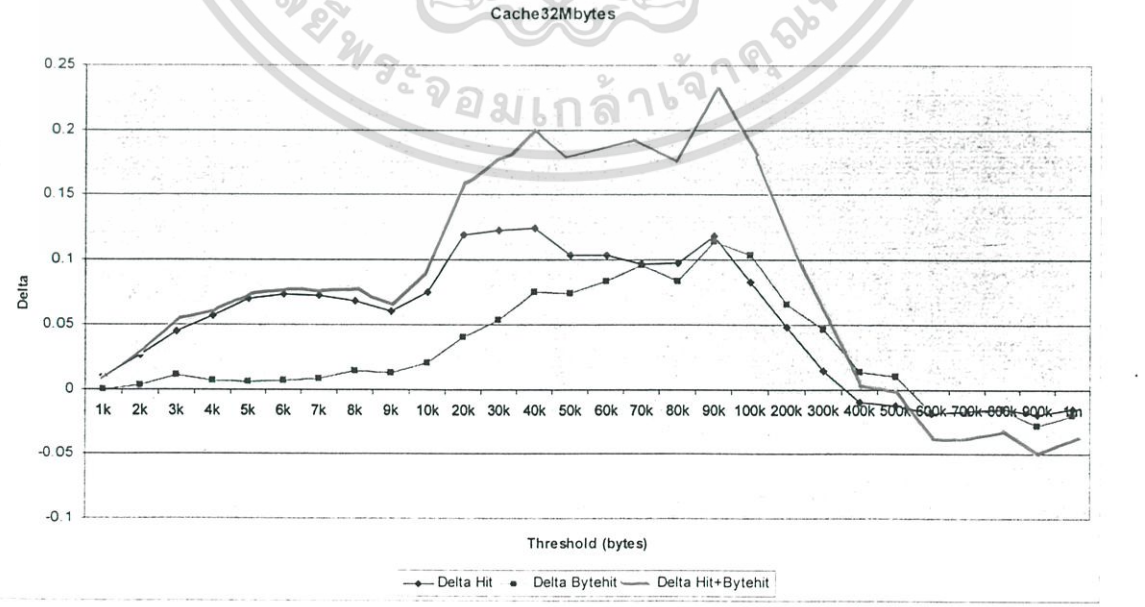
ตารางที่ 5.14 แสดงเปอร์เซ็นต์การร้องขอแบ่งตามช่วงขนาดของข้อมูล

ช่วงขนาดข้อมูล (ไบต์)	จำนวนการร้องขอ (ครั้ง)	จำนวนการร้องขอ (%)	จำนวนการร้องขอเทียบตามขนาดข้อมูล (ไบต์)	จำนวนการร้องขอเทียบตามขนาดข้อมูล (%)
0 - 1K	128,290	22.62	68,863,501	0.87
1K - 10K	318,973	56.24	1,100,693,834	13.88
10K - 100K	111,649	19.69	3,059,032,895	38.56
100K - 1M	7,511	1.32	1,961,095,222	24.72
Over 1M	709	0.13	1,742,961,290	21.97
Summary	567,132	100.00	7,932,646,742	100.00

โดยได้จำลองการทำงาน ในการวิเคราะห์หาค่า threshold ที่เหมาะสม ได้ทำการเปลี่ยนค่า threshold ตั้งแต่ 1 Kbytes ไปจนถึง 1 Mbytes โดยการเพิ่มของค่า threshold จะเพิ่มในอัตราส่วนแบบ Semi-log เป็นจำนวนทั้งสิ้น 28 ระดับ และปรับเปลี่ยนแคชที่ 32, 64 และ 128 Mbytes ซึ่งผลการจำลองการทำได้สามารถแสดงในตาราง และกราฟต่อไปนี้

ตารางที่ 5.15 แสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ของข้อมูลจาก Calgory ขนาดแคชเท่ากับ 32Mbytes

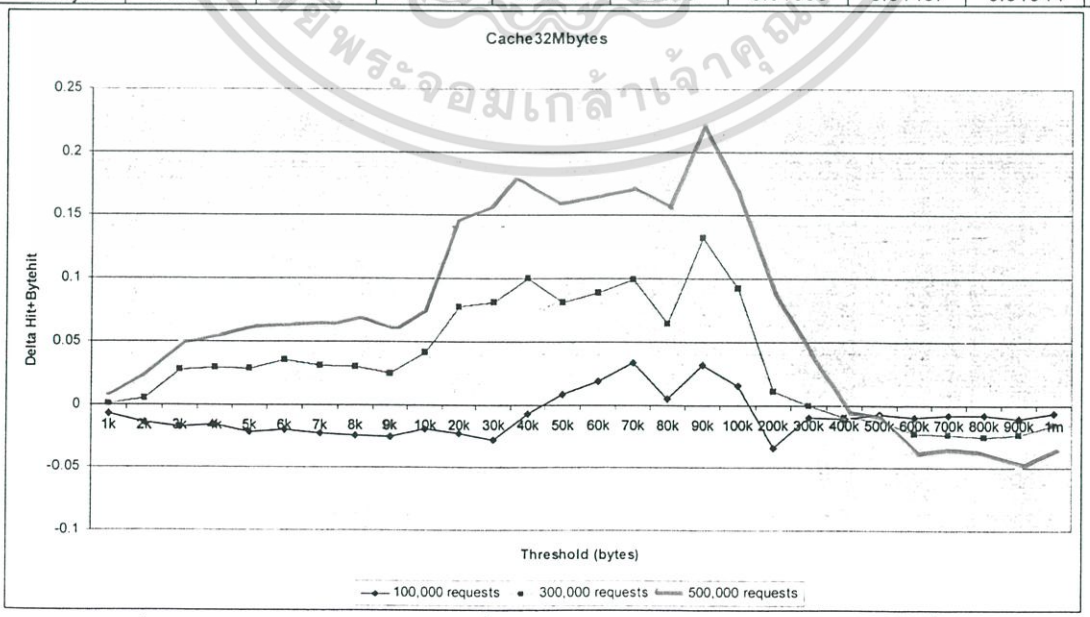
Threshold	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}
1Kbytes	0.010132	-0.0005	0.009628
2Kbytes	0.026535	0.002897	0.029432
3Kbytes	0.04451	0.010613	0.055123
4Kbytes	0.056704	0.006741	0.063445
5Kbytes	0.069391	0.005747	0.075138
6Kbytes	0.072933	0.006195	0.079128
7Kbytes	0.072301	0.008412	0.080713
8Kbytes	0.067934	0.014437	0.082371
9Kbytes	0.060364	0.012922	0.073286
10Kbytes	0.075393	0.020268	0.095661
20Kbytes	0.11955	0.0405	0.16005
30Kbytes	0.12256	0.05292	0.17548
40Kbytes	0.12403	0.07542	0.19945
50Kbytes	0.10363	0.074415	0.178045
60Kbytes	0.10377	0.083261	0.187031
70Kbytes	0.097065	0.095423	0.192488
80Kbytes	0.097701	0.083543	0.181244
90Kbytes	0.11857	0.11406	0.23263
100Kbytes	0.082528	0.10394	0.186468
200Kbytes	0.04831	0.065756	0.114066
300Kbytes	0.014521	0.046514	0.061035
400Kbytes	-0.0094989	0.013126	0.003627
500Kbytes	-0.012421	0.009911	-0.00251
600Kbytes	-0.01837	-0.01755	-0.03592
700Kbytes	-0.016996	-0.01844	-0.03544
800Kbytes	-0.015253	-0.0162	-0.03145
900Kbytes	-0.019587	-0.02822	-0.04781
1Mbytes	-0.015043	-0.02024	-0.03528



รูปที่ 5.9 กราฟแสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ของตารางที่ 5.15

ตารางที่ 5.16 แสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ซึ่งมีปริมาณการร้องขอ400,000 ครั้ง, 800,000 ครั้ง และ 1,200,000 ครั้งของข้อมูลจาก Calgory ขนาดแคชเท่ากับ 32 Mbytes

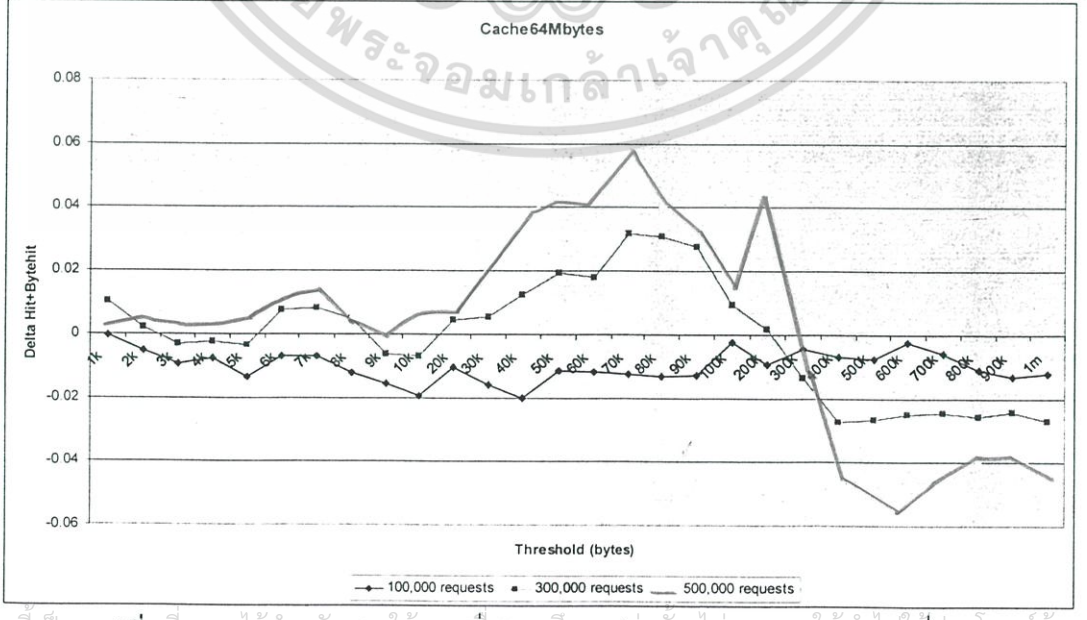
	100,000 requests			300,000 requests			500,000 requests		
Threshold	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}
1Kbytes	-0.00348	-0.00383	-0.00731	0.001771	-0.001	0.000771	0.008662	-0.00252	0.006141
2Kbytes	-0.00976	-0.00468	-0.01444	0.00557	-0.00084	0.004726	0.022962	0.000227	0.023189
3Kbytes	-0.01291	-0.00465	-0.01756	0.020325	0.006631	0.026956	0.040724	0.007866	0.04859
4Kbytes	-0.01301	-0.00257	-0.01558	0.02619	0.002873	0.029063	0.051684	0.003991	0.055675
5Kbytes	-0.01168	-0.01059	-0.02227	0.030171	-0.00172	0.028453	0.063557	0.002136	0.065693
6Kbytes	-0.0102	-0.00974	-0.01995	0.034355	0.000409	0.034764	0.066863	0.002985	0.069848
7Kbytes	-0.01493	-0.00809	-0.02302	0.030145	0.001092	0.031237	0.065406	0.004598	0.070004
8Kbytes	-0.01988	-0.00517	-0.02505	0.024564	0.005803	0.030367	0.060957	0.010599	0.071556
9Kbytes	-0.01929	-0.00605	-0.02534	0.019632	0.005192	0.024824	0.053454	0.009985	0.063439
10Kbytes	-0.01513	-0.00461	-0.01974	0.032069	0.008959	0.041028	0.067863	0.015028	0.082891
20Kbytes	-0.01659	-0.00595	-0.02254	0.058214	0.01973	0.077944	0.11049	0.034423	0.144913
30Kbytes	-0.01875	-0.00975	-0.0285	0.054902	0.026276	0.081178	0.11078	0.045744	0.156524
40Kbytes	-0.0031	-0.00462	-0.00772	0.05921	0.040708	0.099918	0.11299	0.066477	0.179467
50Kbytes	-0.00041	0.008807	0.008396	0.040242	0.040609	0.080851	0.092674	0.065755	0.158429
60Kbytes	0.002492	0.016478	0.01897	0.042427	0.046796	0.089223	0.094135	0.074624	0.168759
70Kbytes	-2.82E-05	0.03311	0.033082	0.038554	0.061093	0.099647	0.087793	0.087033	0.174826
80Kbytes	-0.00861	0.013699	0.005088	0.024481	0.040295	0.064776	0.083843	0.071908	0.155751
90Kbytes	-0.00426	0.03559	0.031328	0.054413	0.077476	0.131889	0.10842	0.10647	0.21489
100Kbytes	-0.01108	0.026455	0.015372	0.025556	0.066412	0.091968	0.071332	0.095088	0.16642
200Kbytes	-0.01626	-0.01767	-0.03393	-0.00219	0.013102	0.010912	0.037931	0.051721	0.089652
300Kbytes	-0.00739	-0.00274	-0.01014	-0.01158	0.01099	-0.00059	0.007777	0.035764	0.043541
400Kbytes	-0.00418	-0.00641	-0.01058	-0.01143	0.001825	-0.0096	-0.01159	0.008107	-0.00348
500Kbytes	-0.00295	-0.0044	-0.00735	-0.01195	-0.00017	-0.01212	-0.01429	0.005051	-0.00923
600Kbytes	-0.00265	-0.00725	-0.00991	-0.01039	-0.01265	-0.02304	-0.01995	-0.02162	-0.04157
700Kbytes	-0.0023	-0.0059	-0.0082	-0.01465	-0.00949	-0.02414	-0.01711	-0.01736	-0.03448
800Kbytes	-0.00226	-0.00585	-0.00811	-0.01466	-0.0105	-0.02516	-0.01699	-0.01791	-0.0349
900Kbytes	-0.00191	-0.00852	-0.01043	-0.00868	-0.01533	-0.024	-0.0189	-0.02885	-0.04775
1Mbytes	-0.00181	-0.0048	-0.00661	-0.00764	-0.0083	-0.01595	-0.01407	-0.01944	-0.03351



รูปที่ 5.10 กราฟแสดงค่า ΔH_{i+b} ที่ปริมาณการร้องขอ 3 ระดับของตารางที่ 5.16

ตารางที่ 5.17 แสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ซึ่งมีปริมาณการร้องขอ400,000 ครั้ง, 800,000 ครั้ง และ 1,200,000 ครั้งของข้อมูลจาก Calgory ขนาดแคชเท่ากับ 64 Mbytes

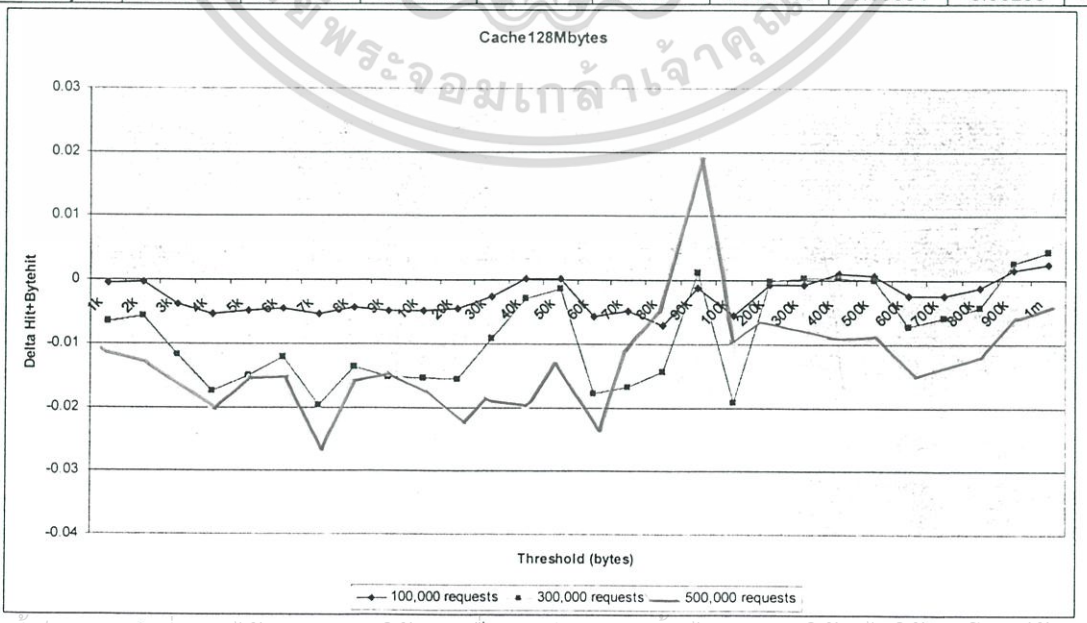
	100,000 requests			300,000 requests			500,000 requests		
Threshold	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}
1Kbytes	-0.00151	0.000974	-0.00054	-0.00454	0.015035	0.010495	-0.0049	0.00785	0.00295
2Kbytes	-0.00198	-0.00333	-0.00531	-0.00124	0.003321	0.002077	0.004679	0.000856	0.005535
3Kbytes	-0.0028	-0.00649	-0.00929	-0.00107	-0.00206	-0.00314	0.007278	-0.00412	0.003162
4Kbytes	-0.00241	-0.00534	-0.00774	0.005167	-0.00755	-0.00239	0.011324	-0.00819	0.003133
5Kbytes	-0.00217	-0.01148	-0.01365	0.01407	-0.01742	-0.00335	0.022031	-0.01547	0.006557
6Kbytes	-0.00501	-0.00201	-0.00703	-0.00091	0.008619	0.007705	0.010587	0.002468	0.013055
7Kbytes	-0.00519	-0.00188	-0.00707	-0.00083	0.008933	0.008105	0.013377	0.003606	0.016983
8Kbytes	-0.00508	-0.00704	-0.01212	-0.00616	0.010923	0.004765	-0.00064	0.006284	0.005645
9Kbytes	-0.00614	-0.00941	-0.01555	-0.00251	-0.00371	-0.00622	0.002946	-0.00376	-0.00081
10Kbytes	-0.00865	-0.01068	-0.01933	-0.00346	-0.00353	-0.00698	0.009278	-0.00169	0.007592
20Kbytes	-0.00709	-0.00336	-0.01045	-0.00058	0.004961	0.004384	0.004056	0.004868	0.008924
30Kbytes	-0.00635	-0.00976	-0.01611	0.001129	0.004519	0.005648	0.01168	0.009823	0.021503
40Kbytes	-0.0057	-0.01462	-0.02031	0.008492	0.00385	0.012341	0.025277	0.012984	0.038261
50Kbytes	-0.00471	-0.0068	-0.01151	0.00932	0.010055	0.019375	0.024711	0.017954	0.042665
60Kbytes	-0.00444	-0.00755	-0.01199	0.009177	0.008874	0.018051	0.025872	0.01607	0.041942
70Kbytes	-0.00426	-0.00843	-0.01269	0.016143	0.015857	0.032	0.030523	0.025879	0.056402
80Kbytes	-0.00401	-0.00922	-0.01323	0.013702	0.016944	0.030646	0.021583	0.019306	0.040889
90Kbytes	-0.00385	-0.00902	-0.01286	0.010282	0.017502	0.027784	0.016719	0.017706	0.034425
100Kbytes	-0.00284	0.000389	-0.00245	-0.00169	0.011054	0.009367	0.005941	0.008899	0.014839
200Kbytes	-0.00119	-0.0083	-0.00948	0.000226	0.001472	0.001699	0.019737	0.021634	0.041371
300Kbytes	-0.00074	-0.00377	-0.00451	-0.00639	-0.00714	-0.01352	-0.0028	-0.00326	-0.00606
400Kbytes	-0.00066	-0.00635	-0.00701	-0.01064	-0.01673	-0.02737	-0.0133	-0.02937	-0.04266
500Kbytes	-0.00054	-0.00704	-0.00758	-0.00781	-0.0189	-0.02671	-0.01371	-0.0342	-0.04791
600Kbytes	-0.00057	-0.00196	-0.00253	-0.0072	-0.01785	-0.02505	-0.01476	-0.03872	-0.05348
700Kbytes	-0.00055	-0.00544	-0.00599	-0.00519	-0.01955	-0.02474	-0.00819	-0.03544	-0.04363
800Kbytes	-0.00044	-0.01055	-0.01099	-0.00345	-0.02217	-0.02561	-0.00514	-0.03424	-0.03938
900Kbytes	-0.00042	-0.01281	-0.01323	-0.00288	-0.02144	-0.02432	-0.00431	-0.03305	-0.03736
1Mbytes	-0.00053	-0.01176	-0.01229	-0.00383	-0.02341	-0.02725	-0.00596	-0.03732	-0.04328



รูปที่ 5.11 กราฟแสดงค่า ΔH_{i+b} ที่ปริมาณการร้องขอ 3 ระดับของตารางที่ 5.17

ตารางที่ 5.18 แสดงค่า $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ซึ่งมีปริมาณการร้องขอ400,000 ครั้ง, 800,000 ครั้ง และ 1,200,000 ครั้งของข้อมูลจาก Calgory ขนาดแคชเท่ากับ 128 Mbytes

	100,000 requests			300,000 requests			500,000 requests		
Threshold	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}	$\overline{\Delta H_i}$	$\overline{\Delta H_b}$	ΔH_{i+b}
1Kbytes	-0.00022	-0.00031	-0.00052	-0.00145	-0.00507	-0.00652	-0.00129	-0.00953	-0.01082
2Kbytes	-0.00013	-0.00025	-0.00038	-0.00115	-0.00452	-0.00567	-0.00076	-0.01138	-0.01214
3Kbytes	-0.0002	-0.00358	-0.00378	0.001012	-0.01272	-0.01171	0.002264	-0.01789	-0.01562
4Kbytes	-0.00021	-0.0052	-0.00541	0.001435	-0.01884	-0.01741	0.002893	-0.02247	-0.01958
5Kbytes	-8.8E-05	-0.00475	-0.00484	0.002537	-0.0176	-0.01506	0.005116	-0.02052	-0.01541
6Kbytes	-8.3E-05	-0.00432	-0.0044	0.001506	-0.01356	-0.01206	0.003547	-0.01769	-0.01414
7Kbytes	-0.0001	-0.00523	-0.00533	0.001358	-0.02106	-0.01971	0.00162	-0.02932	-0.0277
8Kbytes	-6.2E-05	-0.00431	-0.00437	0.00165	-0.01535	-0.0137	0.003883	-0.02012	-0.01624
9Kbytes	-4.1E-05	-0.00484	-0.00488	0.002398	-0.01758	-0.01518	0.006555	-0.02109	-0.01454
10Kbytes	-4.4E-05	-0.00479	-0.00484	0.002036	-0.01737	-0.01534	0.004563	-0.02119	-0.01663
20Kbytes	-8.3E-05	-0.00445	-0.00454	0.002327	-0.0179	-0.01558	0.004841	-0.02701	-0.02217
30Kbytes	-5E-05	-0.0025	-0.00255	0.004266	-0.01347	-0.00921	0.004986	-0.02394	-0.01895
40Kbytes	-3.2E-05	0.000287	0.000255	0.003018	-0.00597	-0.00296	0.001357	-0.021	-0.01964
50Kbytes	-3.4E-05	0.00026	0.000226	0.003641	-0.00498	-0.00134	0.005636	-0.01883	-0.01319
60Kbytes	-3.6E-05	-0.00569	-0.00573	0.002375	-0.02018	-0.0178	0.006684	-0.03039	-0.02371
70Kbytes	-2.1E-05	-0.00487	-0.00489	0.001372	-0.01817	-0.01679	0.010998	-0.02214	-0.01114
80Kbytes	-3.4E-05	-0.00712	-0.00716	0.004618	-0.01896	-0.01435	0.015608	-0.0207	-0.00509
90Kbytes	-2.5E-05	-0.0011	-0.00112	0.004235	-0.00304	0.001191	0.019913	-0.00145	0.018468
100Kbytes	-2.5E-05	-0.00553	-0.00556	0.000628	-0.01989	-0.01927	0.009582	-0.0193	-0.00971
200Kbytes	-1.3E-05	-0.00063	-0.00064	-0.00022	6.71E-06	-0.00021	0.001224	-0.00634	-0.00511
300Kbytes	-1.3E-05	-0.00062	-0.00064	-0.0002	0.000656	0.000457	0.000213	-0.00703	-0.00681
400Kbytes	-2.8E-06	0.000999	0.000996	-0.00046	0.000857	0.000398	-0.00083	-0.00809	-0.00891
500Kbytes	-1.2E-05	0.000745	0.000734	-0.00039	0.000392	-5.1E-07	-0.00051	-0.00821	-0.00872
600Kbytes	-3.9E-05	-0.00239	-0.00243	-0.0003	-0.00704	-0.00734	-0.00043	-0.0154	-0.01583
700Kbytes	-3.9E-05	-0.00239	-0.00243	-0.00037	-0.00544	-0.0058	-0.00061	-0.0133	-0.01391
800Kbytes	-2.9E-05	-0.00111	-0.00114	-0.00034	-0.00393	-0.00427	-0.00057	-0.01214	-0.01271
900Kbytes	-2.1E-06	0.001566	0.001564	-0.0003	0.002887	0.002584	-0.00054	-0.00557	-0.00611
1Mbytes	-5.5E-07	0.002367	0.002367	-0.0003	0.00471	0.004406	-0.00054	-0.00293	-0.00347



รูปที่ 5.12 กราฟแสดงค่า ΔH_{i+b} ที่ปริมาณการร้องขอ 3 ระดับของตารางที่ 5.18

จากตารางที่ 5.15 และรูปที่ 5.9 หน้า 98 แสดงถึงค่าผลต่างเฉลี่ย $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ของอัลกอริธึม MRASM กับอัลกอริธึม RASM จากข้อมูลของ Calgory หากค่าผลต่างเฉลี่ยที่ได้มีค่าเป็นบวกแสดงว่าค่า threshold นั้นให้ผลที่ดีกว่าอัลกอริธึมเดิม แต่หากค่าผลต่างเฉลี่ยที่ได้มีค่าติดลบจะถือว่าค่า threshold นั้นให้ผลที่ต่ำกว่าอัลกอริธึมเดิม เมื่อนำมาพิจารณากับค่าเฉลี่ยของอัลกอริธึมการแทนที่ข้อมูล จะได้คุณลักษณะของค่า threshold ที่เหมาะสม ณ จุดที่ผลรวมของค่าผลต่างเฉลี่ยของ Hit Rate ($\overline{\Delta H_i}$) และค่าผลต่างเฉลี่ยของ Byte Hit Rate ($\overline{\Delta H_b}$) รวมกันแล้วให้ค่ามากที่สุด (ΔH_{i+b} (max)) คือค่า threshold ที่ 90Kbytes

จากตารางที่ 5.16 และรูปที่ 5.10 หน้า 99 แสดงถึงค่าผลต่างเฉลี่ย $\overline{\Delta H_i}$, $\overline{\Delta H_b}$ และ ΔH_{i+b} ของอัลกอริธึม MRASM กับอัลกอริธึม RASM ของแคชขนาด 32Mbytes ซึ่งแบ่งปริมาณการร้องขอข้อมูลเป็น 3 ระดับคือปริมาณการร้องขอข้อมูลน้อย (100,000 requests) ,ปานกลาง (300,000 requests) และมาก (500,000 requests) จะเห็นได้ว่าในปริมาณการร้องขอที่น้อยจะได้ค่าผลต่างเฉลี่ยที่น้อย และค่าส่วนใหญ่เป็นค่าลบแสดงว่าประสิทธิภาพของอัลกอริธึมที่ปรับปรุงใหม่ด้อยกว่าของอัลกอริธึมเดิม และเมื่อเพิ่มปริมาณการร้องขอเป็นระดับปานกลาง และมาก จะให้ค่าผลต่างเฉลี่ยที่มากขึ้นตามลำดับ แสดงว่าอัลกอริธึมที่ได้ทำการปรับปรุงใหม่จะให้ผลดีขึ้นในปริมาณการร้องขอที่มากขึ้น

เช่นเดียวกับแคชขนาด 64 Mbytes จากตารางที่ 5.17 และรูปที่ 5.11 หน้า 100 จะเห็นว่าในปริมาณการร้องขอที่น้อยจะได้ค่าผลต่างเฉลี่ยที่น้อย และมีค่าติดลบ โดยค่าผลต่างเฉลี่ยของ Hit Rate ($\overline{\Delta H_i}$) และค่าผลต่างเฉลี่ยของ Bytehit Rate($\overline{\Delta H_b}$) จากตารางที่ 5.11 เป็นลบทั้งคู่ แสดงว่าอัลกอริธึมที่ได้ทำการปรับปรุงใหม่จะให้ผลด้อยกว่าในปริมาณการร้องขอน้อย และเมื่อเพิ่มปริมาณการร้องขอเป็นระดับปานกลาง และมาก จะให้ค่าผลต่างเฉลี่ยที่มากขึ้นตามลำดับจากระดับกราฟที่สูงขึ้น และมีค่าเป็นบวก แสดงว่าอัลกอริธึมที่ได้ทำการปรับปรุงใหม่จะให้ผลดีขึ้นในปริมาณการร้องขอที่มากขึ้น

และในแคชขนาด 128 Mbytes จากตารางที่ 5.18 และรูปที่ 5.12 หน้า 101 จะเห็นว่าในปริมาณการร้องขอน้อยๆจะให้ค่าผลต่างเฉลี่ยที่ได้จะติดลบ โดยค่า Hit Rate($\overline{\Delta H_i}$) เพิ่มขึ้นแต่ค่า Bytehit Rate($\overline{\Delta H_b}$) และในปริมาณการร้องขอมากๆ ณ บางจุด threshold จะให้ค่าผลต่างเฉลี่ยโดยรวมเป็นบวก แต่ส่วนใหญ่เป็นลบ แสดงว่าในแคชที่ใหญ่ประสิทธิภาพของอัลกอริธึมที่ปรับปรุงใหม่จะให้ค่าที่ด้อยกว่าอัลกอริธึมเดิม แม้ว่าจะให้ค่า Hit Rate ที่เพิ่มขึ้น แต่ค่า Bytehit Rate กลับได้ค่าลดลง

5.5 สรุปการวิเคราะห์ผลการหา Threshold ที่เหมาะสม

จากข้อมูลการร้องขอทั้ง 3 ชุดที่ได้ทำการทดสอบเพื่อทำการหาจุด threshold ที่เหมาะสมที่ทำให้ประสิทธิภาพของอัลกอริธึมที่ปรับปรุงใหม่ดีที่สุดจะพบว่าค่า threshold ที่เหมาะสมของแต่ละชุดข้อมูลจะได้ค่าไม่เหมือนกัน ขึ้นกับลักษณะพฤติกรรมการร้องขอข้อมูล, ขนาดข้อมูล และความถี่ในการร้องขอข้อมูล โดยข้อมูลจาก Clarknet จะได้ค่า threshold ที่เหมาะสมที่ 9Kbytes ซึ่งเราสามารถจะพิจารณาได้จากตารางที่ 5.2 เมื่อทำการเปรียบเทียบจำนวนการร้องขอเทียบตามขนาดของข้อมูลเป็นเปอร์เซ็นต์จะเห็นว่าช่วงขนาดข้อมูล 1-10Kbytes และ 10-100Kbytes เป็นช่วงที่จำนวนการร้องขอเทียบตามขนาดของข้อมูลหนาแน่นที่สุด ซึ่งเป็นผลทำให้ค่า ΔH_{i+b} ของกราฟในรูปที่ 5.1 มีค่าสูงในช่วงที่จุด threshold ระหว่างนั้นคือประมาณ 6Kbytes ถึง 30Kbytes ซึ่งจุดที่ให้ค่า ΔH_{i+b} สูงที่สุดคือ threshold ที่ 9Kbytes ดังนั้นจึงเป็นค่า threshold ที่เหมาะสมที่สุด

ส่วนข้อมูลจาก Nasa จะได้ค่า threshold ที่เหมาะสมที่ 100Kbytes ซึ่งเราสามารถจะพิจารณาได้จากตารางที่ 5.8 เมื่อทำการเปรียบเทียบจำนวนการร้องขอเทียบตามขนาดของข้อมูลเป็นเปอร์เซ็นต์จะเห็นว่าช่วงขนาดข้อมูล 10-100Kbytes และ 100Kbytes-1Mbytes เป็นช่วงที่จำนวนการร้องขอเทียบตามขนาดของข้อมูลหนาแน่นที่สุด ซึ่งเป็นผลทำให้ค่า ΔH_{i+b} ของกราฟในรูปที่ 5.5 มีค่าสูงในช่วงที่จุด threshold ระหว่างนั้นคือประมาณ 50Kbytes ถึง 100Kbytes ซึ่งจุดที่ให้ค่า ΔH_{i+b} สูงที่สุดคือ threshold ที่ 100Kbytes ดังนั้นจึงเป็นค่า threshold ที่เหมาะสมที่สุด

และข้อมูลจาก Calgory จะได้ค่า threshold ที่เหมาะสมที่ 90Kbytes ซึ่งเราสามารถจะพิจารณาได้จากตารางที่ 5.14 เมื่อทำการเปรียบเทียบจำนวนการร้องขอเทียบตามขนาดของข้อมูลเป็นเปอร์เซ็นต์จะเห็นว่าช่วงขนาดข้อมูล 10-100Kbytes เป็นช่วงที่จำนวนการร้องขอเทียบตามขนาดของข้อมูลหนาแน่นที่สุด ซึ่งเป็นผลทำให้ค่า ΔH_{i+b} ของกราฟในรูปที่ 5.9 มีค่าสูงในช่วงที่จุด threshold ระหว่างนั้นคือประมาณ 40Kbytes ถึง 100Kbytes ซึ่งจุดที่ให้ค่า ΔH_{i+b} สูงที่สุดคือ threshold ที่ 90Kbytes ดังนั้นจึงเป็นค่า threshold ที่เหมาะสมที่สุด นอกจากนี้จากการสังเกตจะเห็นว่าจำนวนการร้องขอเทียบตามขนาดของข้อมูลมีการการกระจายอยู่ในช่วงต่างๆ ทำให้กราฟในรูปที่ 5.9 ได้ค่าค่อนข้างกระจายเช่นกัน

เมื่อพิจารณาถึงขนาดของแคช จากการทดสอบจะพบว่าที่ขนาดแคช 32 Mbytes ในปริมาณการร้องขอที่น้อยจะได้ค่าผลต่างเฉลี่ยที่น้อย แม้ว่าค่าเฉลี่ยผลต่างของ Hit Rate($\overline{\Delta H_i}$) ส่วนใหญ่เป็นบวก แต่ค่าเฉลี่ยผลต่างของ Bytehit Rate($\overline{\Delta H_b}$) กลับเป็นลบ แต่ค่าผลต่างเฉลี่ยโดยรวมส่วนใหญ่ยังคงเป็นค่าบวกแสดงว่าประสิทธิภาพของอัลกอริธึมที่ปรับปรุงใหม่ดีกว่าของอัลกอริธึมเดิมไม่มาก และเมื่อเพิ่มปริมาณการร้องขอเป็นระดับปานกลาง และมาก จะให้ค่าผลต่างเฉลี่ยที่มากขึ้นตามลำดับ แสดงว่าอัลกอริธึมที่ได้ทำการปรับปรุงใหม่จะให้ผลดีขึ้นในปริมาณการร้องขอที่มากขึ้น

ส่วนในแคชขนาด 64 Mbytes ก็มีลักษณะคล้ายแคชขนาด 32 Mbytesคือในปริมาณการร้องขอที่น้อยแม้ว่าค่า $\overline{\Delta H_i}$ ส่วนใหญ่เป็นบวก แต่ค่า $\overline{\Delta H_b}$ กลับเป็นลบ ซึ่งค่าผลต่างเฉลี่ยโดยรวมส่วนใหญ่ยังคงเป็นค่าบวกแสดงว่าประสิทธิภาพของอัลกอริธึมที่ปรับปรุงใหม่ดีกว่าของอัลกอริธึมเดิมนิดหน่อย ไม่แตกต่างจากของอัลกอริธึมเดิมมากนัก และเมื่อเพิ่มปริมาณการร้องขอเป็นระดับปานกลาง และมาก จะให้ค่าผลต่างเฉลี่ยที่มากขึ้นตามลำดับ แสดงว่าอัลกอริธึมที่ได้ทำการปรับปรุงใหม่จะให้ผลดีขึ้นในปริมาณการร้องขอที่มากขึ้น

ส่วนในแคชขนาด 128 Mbytes ในปริมาณการร้องขอตั้งแต่ระดับน้อย, ปานกลาง และมาก จะมีแนวโน้มของค่าเฉลี่ยผลต่างโดยรวม (ΔH_{i+b}) เพิ่มขึ้นตามลำดับ แต่ค่าส่วนใหญ่เป็นค่าติดลบ มีเพียงบางจุด threshold ที่ให้ค่าเป็นบวกซึ่งเป็นจุดที่ threshold ต่ำๆ แสดงว่าอัลกอริธึมที่ได้ทำการปรับปรุงใหม่ให้ผลโดยรวมด้อยกว่าอัลกอริธึมเดิม ในแคชขนาดใหญ่ แม้ว่าค่าเฉลี่ยผลต่างของ Hit Rate($\overline{\Delta H_i}$) ส่วนใหญ่ได้ค่าดีขึ้น (ได้ค่าเป็นบวก) แต่เนื่องจากค่าเฉลี่ยผลต่างของ Bytehit Rate($\overline{\Delta H_b}$) กลับเป็นลบเพิ่มขึ้น จึงทำให้ค่าเฉลี่ยผลต่างโดยรวม (ΔH_{i+b}) ไม่ดีขึ้น

ดังนั้นสามารถสรุปได้ว่าที่แคชขนาดไม่ใหญ่มากอัลกอริธึมที่ทำการปรับปรุงจะให้ประสิทธิภาพที่ดีขึ้นเมื่อเปรียบเทียบกับอัลกอริธึมเดิม โดยเฉพาะที่ปริมาณข้อมูลการร้องขอมากๆ แต่ที่แคชขนาดใหญ่จะให้ประสิทธิภาพด้อยลง โดยเฉพาะเมื่อมีปริมาณการร้องขอน้อยๆ แม้ว่าจะได้ค่า Hit Rate ส่วนใหญ่เพิ่มขึ้น แต่ก็ได้ค่า Bytehit Rate ลดลงมากขึ้น

บทที่ 6
บทสรุป

6.1 สรุปผล

วิทยานิพนธ์ฉบับนี้ได้นำเสนอการพัฒนาเว็บแคชเซิร์ฟเวอร์ โดยมุ่งเน้นศึกษาถึงการเพิ่มและวิเคราะห์ประสิทธิภาพของอัลกอริธึมการแทนที่ข้อมูล ซึ่งเป็นฟังก์ชันในการบริหารแคชของเว็บแคชเซิร์ฟเวอร์ เพื่อพิจารณานำข้อมูลออกจากแคช โดยพิจารณาถึงตัววัดประสิทธิภาพ 2 ประเภท คือ ค่าอัตราการพบข้อมูลภายในแคชเทียบตามจำนวนครั้งหรือค่า Hit Rate และอัตราการพบข้อมูลภายในแคชเทียบตามขนาดข้อมูลหรือค่า Byte Hit Rate

โดยวิทยานิพนธ์นี้ได้เลือกพิจารณาปรับแต่งอัลกอริธึมการแทนที่ข้อมูลแบบ RASM ซึ่งเป็นอัลกอริธึมที่มีกลไกในการพิจารณาเลือกใช้อัลกอริธึมการแทนที่ข้อมูลแบบ GDSF และ LFUDA อีกทีหนึ่งซึ่งเรียกอัลกอริธึมที่ได้ทำการปรับแต่งว่าอัลกอริธึมการแทนที่ข้อมูลแบบ MRASM โดยพิจารณาว่าการที่ไฟล์ขนาดใหญ่มาแทนที่ไฟล์ขนาดเล็กที่มีการเรียกใช้งานบ่อยครั้ง จะทำให้ค่าอัตรา Hit Rate ต่ำลง จึงได้ทำการแยกการแทนที่ข้อมูลที่มีขนาดใหญ่ และข้อมูลที่มีขนาดเล็กกว่าค่ากำหนดหรือค่า threshold ออกจากกัน เพื่อให้ข้อมูลที่มีขนาดใหญ่กว่าค่า threshold ไม่มาแทนที่ข้อมูลที่มีขนาดเล็กกว่าค่า threshold เพื่อทำให้อัตรา Hit Rate มีค่าสูงขึ้น ซึ่งพบว่าจากการใช้ข้อมูลทดสอบหลายชุดจะได้ค่าอัตรา Hit Rate ของอัลกอริธึม MRASM สูงขึ้นกว่าอัลกอริธึม RASM ในช่วง threshold ค่าต่ำถึงปานกลาง ส่วนในช่วง threshold สูงจะให้ค่า Hit Rate ใกล้เคียงกัน

จากผลการทดลองเพื่อหาค่า threshold ที่เหมาะสมสำหรับข้อมูลของ Clarknet, Nasa และ Calgory จะสามารถสรุปได้ผลการทดลองได้ดังนี้ จากการจำลองการทำงานโดยการเปรียบเทียบประสิทธิภาพของอัลกอริธึม MRASM กับอัลกอริธึม RASM โดยวัดจุด threshold ที่ให้ค่า ΔH_{i+b} สูงสุดจะได้ว่าข้อมูลจาก Clarknet จะได้ค่า threshold ที่เหมาะสมที่ 9Kbytes , ข้อมูลจาก Nasa จะได้ค่า threshold ที่เหมาะสมที่ 100Kbytes , ข้อมูลจาก Calgory จะได้ค่า threshold ที่เหมาะสมที่ 90Kbytes ซึ่งสอดคล้องตามจำนวนการร้องขอที่เทียบตามขนาดข้อมูล ซึ่งช่วงที่มีการร้องขอมากๆ ก็จะได้ค่า ΔH_{i+b} ที่สูงตามไปด้วย

6.2 ปัญหาและอุปสรรค

ปัญหาที่พบในส่วนการจำลองการทำงานคือข้อมูลที่นำมาใช้ในการร้องขอจะเป็นข้อมูลในล็อกไฟล์ซึ่งเป็นไฟล์ขนาดเล็ก ซึ่งยังไม่สามารถทำการหาล็อกไฟล์ของเว็บแคชเซิร์ฟเวอร์ที่ได้มาตรฐานเพื่อนำมาใช้เป็นไฟล์ทดสอบการทำงานของอัลกอริธึมการแทนที่ข้อมูล ดังนั้นการจำลอง

การทดสอบอัลกอริธึมการแทนที่ข้อมูลนั้นได้กำหนดขนาดของแคชที่ใช้เก็บข้อมูลในเว็บแคชเซิร์ฟเวอร์ให้มีขนาดไม่ใหญ่มากนักเพื่อให้สามารถเห็นผลการทดลองได้ชัดเจนมากยิ่งขึ้น ปัจจุบันขนาดของข้อมูลที่เก็บในแคชมีขนาดสูงขึ้น และจะเห็นว่าขนาดแคชที่ใช้เก็บข้อมูลก็สูงขึ้นตามไปด้วย

6.3 แนวทางการพัฒนาต่อ

ในวิทยานิพนธ์ฉบับนี้ได้นำเสนอการปรับแต่งอัลกอริธึมการแทนที่ข้อมูล RASM โดยได้ทำการปรับเปลี่ยนการจัดการเนื้อที่สำหรับใช้งานแคชของเว็บแคชเซิร์ฟเวอร์ โดยการแยกการแทนที่ข้อมูลให้อยู่ในกลุ่มที่มีช่วงขนาดข้อมูลเดียวกันคือช่วงขนาดข้อมูลที่มากกว่าค่า threshold กับช่วงขนาดข้อมูลที่น้อยกว่า threshold ซึ่งจากผลการทดลองจะได้ค่า Hit Rate เพิ่มขึ้นกว่าเดิม แต่ค่า Byte Hit Rate เพียงบางส่วนเท่านั้นที่เพิ่มขึ้นจากเดิม ซึ่งเราสามารถนำวิธีการบริหารจัดการเนื้อที่ดิสก์ด้วยวิธีการต่างๆเช่น วิธีการแบบ Partitioned Caches โดยการแบ่งเนื้อที่แคชออกเป็นส่วนๆ โดยสามารถกำหนดสัดส่วนเนื้อที่ของแต่ละพาร์ทิชั่น และอัลกอริธึมการแทนที่ข้อมูลที่เหมาะสมสำหรับแต่ละพาร์ทิชั่นได้ เพื่อให้สามารถปรับปรุงประสิทธิภาพของเว็บแคชเซิร์ฟเวอร์ให้มีค่า Hit Rate และ Byte Hit Rate เพิ่มขึ้นได้

เอกสารอ้างอิง

[1] Luotonen, A. Web Proxy Servers. New Jerry : Prentice-Hall, Inc. 1998

[2] Arlitt M. and Williamson C. "Internet Web Servers: Workload Characterization and Performance Implications." IEEE/ACM Transactions on Networking, vol. 5, no. 5, October 1997. pp. 631-645.

[3] Dilley J., Arlitt M. and Perret S. "Enhancement and Validation of Squid's Cache Replacement Policy." Technical Report HPL-1999-69, Hewlett-Packard Laboratories, May 1999.

[4] Balamash A. and Krunz M. "An Overview of Web Caching Replacement Algorithm" IEEE Communications Surveys & Tutorials, vol.6, no. 2, Second Quarter 2004, pp. 44-56.

[5] Podlipnig S. and Boszormenyi L. "A Survey of Web Cache Replacement Strategies." ACM Computing Surveys, vol. 35, no. 4, December 2003, pp. 374-398

[6] Dilley J. "Improving Proxy Cache Performance: Analysis of Three Replacement Policies" IEEE Internet Computing., November-December 1999. pp. 44-50.

[7] Sontisiri T., Sopechoke P., Thipchaksurat S., and Varakulsiripunth R. 2004 "Replacement Algorithm Selection Mechanism Considering File Size for Web Cache Server." in ICCAS2004. Bangkok Thailand : The Shangri-La Hotel.

[8] Thirajitto K., Thipchaksurat S., and Bunyatnopparat P. 2005 "The Improvement of Replacement Algorithm Selection Mechanism for Web Cache Server." In ECTI-Con2005. Pattaya Thailand : Asia Pattaya Hotel.

[9] Sopechoke P. "Replacement Algorithm for Web Cache Server." Master Thesis of Electrical Engineering, King Mongkut's Institute of Technology Ladkrabang. 2004

[10] Wessels D. and Claffy K. "Internet Cache Protocol (icp) version 2." Network Working Group RFC 2186, [Online]. Available : http://ds.internic.net/rfc/rfc2186.txt. September 1997.

[11] CERN/W3C HTTP Daemon Software. [Online]. Available : http://www.w3.org/.

[12] Apache HTTP Daemon Software. [Online]. Available : http://www.apache.org/.

[13] Netscape HTTP Daemon Software. [Online]. Available : http://www.netscape.com/.

[14] Delegate HTTP Daemon Software. [Online]. Available : http://www.delegate.org/.

[15] Microsoft Proxy Software. [Online]. Available : http://www.microsoft.com/.

[16] Wcol/Catalyst Proxy Software. [Online]. Available : http://shika.aistnara.ac.jp/products/wcol/wcol.html/.

[17] Novell BorderManager Proxy Software. [Online]. Available : http://www.novell.com/.

[18] Cacheflow Proxy Software. [Online]. Available : http://www.cacheflow.com/.

[19] MOWS HTTP Daemon and Proxy Software. [Online]. Available : http://mows.rz.unimannheim.de/mows/.

[20] Squid Internet Object Cache Software. [Online]. Available : http://www.squidcache.org/.

[21] Williams S., Abrams M., Stanbridge C., Abdulla G. and Fox E. "Removal Policies in Network Caches for World-Wide Web Documents." Proc. ACM Sigcomm96, vol. 26, August 1996. pp. 293-305.

[22] Abrams M., Stanbridge C., Abdulla G., Williams S. and Fox E. "Caching Proxies: Limitation and Potentials." WWW-4, December 1995.

[23] Wooster R. and Abrams M. "Proxy Caching the estimates Page Load Delays." Proc. 6th International World Wide Web Conference, 1997.

[24] Jin S. and Bestavros A. "Popularity-aware greedy dual-size Web proxy caching algorithms." Proc. 20th International Conference on Distributed Computing Systems, April 2000. pp. 254-261.

[25] Arlitt M., Cherkasova L., Dilley J., Friedrich R. and Jin T. "Evaluating Content Management Techniques for Web Proxy Caches." Technical Report HPL-98-173, Hewlett-Packard Laboratories, April 1999.

[26] Cherkasova L. "Improving WWW proxies Performance with Greedy-Dual-Size-Frequency Caching Policy." Technical Report HPL-1998-69R1, Hewlett-Packard Laboratories, November 1998.

[27] Young N. "On-line caching as cache size varies." 2nd Annual ACM-SIAM Symposium on Discrete Algorithm, 1991. pp. 241-250.

[28] Cao P. and Irani S. "Cost Aware WWW Proxy Caching Algorithms." Proc. USENIX Symposium on Internet Technologies and Systems (USITS), December 1997, pp. 193-206.

[29] Internet Traffic Archive. [Online]. Available : http://ita.ee.lbl.gov/.

[30] โกศล ถิรจิตโต, ศักดิ์ชัย ทิพย์จักษุรัตน์, ประทีป บัญญัตินพรัตน์. "การปรับปรุงวิธีการเลือกใช้อัลกอริทึมการแทนที่ข้อมูลสำหรับเว็บแคชเซิร์ฟเวอร์" วิศวกรรมสารลาดกระบัง, ปีที่ 21, ฉบับที่ 3, กันยายน 2547. หน้า 56-61.

ภาคผนวก ก.
ผลงานวิจัยที่ได้รับการตีพิมพ์

[1] โกศล ถิรจิตโต, ศักดิ์ชัย ทิพย์จักษุรัตน์, ประทีป บัญญัตินพรัตน์. "การปรับปรุงวิธีการเลือกใช้อัลกอริทึมการแทนที่ข้อมูลสำหรับเว็บแคชเซิร์ฟเวอร์" วิศวสารลาดกระบัง,ปีที่ 21,ฉบับที่ 3,กันยายน 2547.หน้า 56-61

[2] Thirajitto K., Thipchaksurat S., and Bunyatnopparat P. "The Improvement of Replacement Algorithm Selection Mechanism for Web Cache Server." *In The 2005 Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology (ECTI-Con2005)*. Pattaya, Thailand, May 2005. pp.835-838.

ISSN 0125-1724

วิศวสาร

ลาดกระบัง

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

LADKRABANG ENGINEERING JOURNAL

ปีที่ 21 ฉบับที่ 3 กันยายน 2547

การปรับปรุงวิธีการเลือกใช้อัลกอริทึมการแทนที่ข้อมูล สำหรับเว็บแคชเซิร์ฟเวอร์

The Improvement of Selection Mechanism for Replacement Algorithms for Web Cache Server

โกศล นิรจิตโต ศักดิ์ชัย ทิพย์จักษุรัตน์ ประทีป บัญญัตินพรัตน์
ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

บทคัดย่อ

งานวิจัยนี้นำเสนอการปรับปรุงการทำงานของเว็บแคชเซิร์ฟเวอร์ (Web Cache Server) โดยพิจารณาในส่วนของอัลกอริทึมการแทนที่ข้อมูล (Replacement Algorithms) ที่เครื่องไกลเอนต์เรียกใช้ อัลกอริทึมการแทนที่ข้อมูลต่างๆ ที่มีการใช้กันอยู่จะเหมาะสมกับขนาดของข้อมูลแต่ละประเภทที่อยู่ภายในเว็บ และได้มีการพัฒนาวิธีการในการเลือกอัลกอริทึมการแทนที่ข้อมูลมาใช้งานโดยแยกตามขนาดของข้อมูลที่เรียกว่า RASM (Replacement Algorithm Selection Mechanism) สำหรับงานวิจัยนี้ได้นำเอาวิธีการเลือกใช้อัลกอริทึมการแทนที่ข้อมูลแบบ RASM มาทำการปรับปรุงที่เรียกว่า ISMRA (Improvement of Selection Mechanism for Replacement Algorithm) ซึ่ง ISMRA จะยอมให้เว็บแคชเซิร์ฟเวอร์สามารถเลือกอัลกอริทึมการแทนที่ข้อมูลที่เหมาะสมสำหรับข้อมูลแต่ละขนาด โดยข้อมูลใหม่จะแทนที่ข้อมูลภายในแคชที่มีขนาดข้อมูลอยู่ในช่วงขนาดเดียวกัน ผลการทดลองที่ได้จะได้ค่าอัตราการพบข้อมูลในแคชของเว็บแคชเซิร์ฟเวอร์เพิ่มขึ้นเมื่อเทียบกับอัลกอริทึมเดิม

Abstract

This paper describes the improvement of web cache server by studying in replacement algorithm of data which is requested from the clients. We found that each replacement algorithm is suitable for each type of data size in the web pages. The selection mechanism of replacement algorithm has been developed depended on size of data; call RASM (Replacement Algorithm Selection Mechanism). For our research we try to improve the RASM, call the Improvement of Selection Mechanism for Replacement Algorithms (ISMRA) for web cache server. The ISMRA allows web cache server to choose the appropriate replacement algorithms, depended on data size. New data will replace the data in cache that have the same range of data size. The result by means of simulation showed that ISMRA gave the higher hit rate than RASM.

Ladkrabang Engineering Journal, Vol. 21, No. 3, September 2004 57

1. บทนำ

ปัจจุบันนี้โลกอินเตอร์เน็ตได้เติบโตอย่างรวดเร็ว ทำให้เกิดการแออัดของระบบเน็ตเวิร์ค ดังนั้นเพื่อแก้ปัญหาในส่วนนี้เว็บแคชเซิร์ฟเวอร์จึงได้ถูกพัฒนาขึ้นมา โดยจะอยู่ระหว่างเครื่องไคลเอนต์กับเครื่องเว็บเซิร์ฟเวอร์ เครื่องไคลเอนต์ที่ต้องการเว็บเพจจะต้องร้องขอไปยังเว็บเซิร์ฟเวอร์ จากนั้นเว็บเซิร์ฟเวอร์ก็จะส่งเว็บเพจนั้นกลับมาให้กับเครื่องไคลเอนต์ ดังนั้นข้อมูลทุกอย่างจะถูกเก็บไว้ในเว็บแคชเซิร์ฟเวอร์ เมื่อแคชในเว็บแคชเซิร์ฟเวอร์ใกล้เต็ม อัลกอริทึมการแทนที่ข้อมูลจะถูกเรียกใช้เพื่อเลือกข้อมูลที่จะลบออกจากแคช

ในส่วนที่เหลือของเอกสารงานวิจัยมีดังนี้ ส่วนที่ 2 อ้างถึงเนื้อหาที่เกี่ยวข้อง ส่วนที่ 3 อธิบายการปรับปรุงวิธีการเลือกใช้อัลกอริทึมการแทนที่ข้อมูล (ISMRA) ส่วนที่ 4 แสดงผลการจำลองการทำงานของอัลกอริทึม ISMRA ส่วนที่ 5 แสดงการวิเคราะห์ผลการจำลองการทำงาน ส่วนที่ 6 แสดงการหาค่า threshold ที่เหมาะสม สุดท้ายส่วนที่ 7 เป็นการสรุปงานวิจัย และแนวทางในการวิจัยต่อในอนาคต

2. ส่วนเนื้อหาที่เกี่ยวข้อง

ปัจจุบันมีอัลกอริทึมในการแทนที่ข้อมูลหลายชนิดเพื่อเพิ่มประสิทธิภาพของเว็บแคชเซิร์ฟเวอร์ ซึ่งอัลกอริทึมแต่ละชนิดจะพิจารณาถึงตัวแปรที่เกี่ยวข้องเพื่อทำการเลือกข้อมูลที่จะเอาออกจากแคชเพื่อที่จะเพิ่มค่า Hit Rate และ Byte-Hit Rate ซึ่งพบว่ามี 2 ตัวแปรที่สำคัญจากอัลกอริทึมการแทนที่ข้อมูลเพื่อเลือกข้อมูลเก็บไว้ในแคช นั่นคือความถี่ (frequency) ของการร้องขอและอายุของข้อมูล (age) ที่เก็บไว้ในแคช อัลกอริทึมการแทนที่ข้อมูลที่พิจารณาถึงตัวแปรทั้ง 2 ตัวแปรนี้คือ Greedy-Dual Size with Frequency (GDSF) [1] และ Least Frequency Used with Dynamic Aging (LFUDA) [1] ได้นำค่า threshold เข้ามาใช้เลือกอัลกอริทึมการแทนที่ข้อมูลของเว็บแคชเซิร์ฟเวอร์ เพื่อนำมาหาค่าคีย์ความสำคัญ (priority key) ของข้อมูลในแคชของเว็บแคชเซิร์ฟเวอร์ เพื่อพิจารณาในการเอาข้อมูลออกจากแคชของเว็บแคชเซิร์ฟเวอร์ โดยข้อมูลที่มีค่าคีย์ความสำคัญต่ำที่สุดจะถูกนำออกจากแคชก่อนข้อมูลอื่น

2.1 อัลกอริทึม Greedy-Dual-Size with Frequency (GDSF)

อัลกอริทึม GDSF ถูกพัฒนาขึ้นโดย Cao และ Irani บนพื้นฐานของอัลกอริทึม GD-Size [2] โดยนำค่าความถี่ (frequency) ซึ่งแทนด้วยสัญลักษณ์ F_i มาใช้ในการคำนวณหาค่าคีย์ความสำคัญ (priority key) ซึ่งแทนด้วยสัญลักษณ์ K_i ดังแสดงในสมการที่ 1

$$K_i = \frac{F_i * C_i}{S_i} + L \qquad (1)$$

เมื่อ C_i เป็นค่าเกี่ยวกับการนำข้อมูล i เข้ามาเก็บในแคช S_i เป็นขนาดของข้อมูล และ L เป็นปัจจัยของเวลา (running age factor) ที่เริ่มต้นที่ 0 และถูกอัพเดทจากค่า priority key ของข้อมูลที่ถูกแทนที่นั่นคือ $L = K_j$ โดยค่า K_j คือค่า priority key ของข้อมูลตัวล่าสุดที่ถูกนำออกไปจากแคช ซึ่งอัลกอริทึมเหล่านี้จะให้ค่า hit rate ที่สูงสุดเมื่อ $C_i = 1$

2.2 อัลกอริทึม Least Frequency Used with Dynamic Aging (LFUDA)

อัลกอริทึม LFUDA ถูกพัฒนาขึ้นจากอัลกอริทึม LFU และ LFU-Aging [3] สามารถให้ค่า Byte-Hit Rate สูงสุด อัลกอริทึมนี้คำนึงถึงข้อมูลที่เรียกใช้บ่อยโดยไม่สนใจขนาดของข้อมูล และได้ใช้วิธีการให้อายุของข้อมูลแบบปรับได้ ซึ่งคำนวณค่า Priority Key (K_i) สำหรับข้อมูล i โดยใช้สมการที่ 2 และให้ค่า $C_i = 1$

$$K_i = (C_i * F_i) + L \qquad (2)$$

2.3 อัลกอริทึม Replacement Algorithm Selection Mechanism (RASM)

ตามลักษณะของอัลกอริทึมการแทนที่ข้อมูลแบบ GDSF และ LFUDA ได้มีการแนะนำวิธีการปรับปรุงเลือกใช้อัลกอริทึมการแทนที่ข้อมูลสำหรับเว็บแคชเซิร์ฟเวอร์เรียกว่า RASM [4] โดยใช้ค่า threshold ใช้เปรียบเทียบกับขนาดของข้อมูล มันถูกนำมาใช้ในการเลือกใช้อัลกอริทึมการแทนที่ข้อมูลแบบ GDSF และ LFUDA ยิ่งกว่านั้นค่าของพารามิเตอร์ใน priority key

(K_i) ของข้อมูลแต่ละอัลกอริทึมถูกปรับเปลี่ยนให้สามารถพิจารณาร่วมกันดังนี้

$$K_i = \begin{cases} \left(\frac{F_i}{S_i}\right) + L & \text{If } S_i < \text{Threshold} \quad (3.1) \\ F_i + L & \text{If } S_i \geq \text{Threshold} \quad (3.2) \end{cases}$$

เมื่อ S_i เป็นขนาดข้อมูล , L เป็นค่าอายุข้อมูล (running age) ที่เริ่มต้นจาก 0 และถูกอัพเดทจากค่า priority key ของข้อมูลที่ถูกเอาออกไป นั่นคือ $L = K_i$ และ F_i เป็นความถี่ของข้อมูล i แต่ละตัว โดยค่าความถี่เริ่มจาก 1 เมื่อ $S_i <$ threshold แต่ถ้า $S_i \geq$ Threshold ค่าความถี่จะเริ่มต้นจาก 0

3. อัลกอริทึม Improvement of Selection Mechanism for Replacement Algorithm (ISMRA)

จากอัลกอริทึม RASM เมื่อแคชเก็บข้อมูลเต็ม และมีข้อมูลใหม่เข้ามา จะมีการแทนที่ข้อมูลในแคช โดยข้อมูลที่มีค่า priority key ต่ำที่สุดจะถูกนำออกไปจากแคช โดยไม่สนใจว่าข้อมูลที่ถูกเอาออกไปเป็นข้อมูลในกลุ่มที่มีขนาดมากกว่าหรือน้อยกว่าค่า threshold ตัวอย่างเช่น ข้อมูลใหม่ที่จะนำเข้ามามีขนาดมากกว่า threshold อาจจะเข้ามาแทนที่ข้อมูลภายในแคชที่มีขนาดน้อยกว่า threshold จำนวนหลายๆไฟล์ ซึ่งการแทนที่ข้อมูลในลักษณะนี้อาจดูไม่ยุติธรรมถ้าการหาค่า priority key ของข้อมูลระหว่างสมการบน (3.1) และสมการล่าง (3.2) อยู่ในสัดส่วนที่ไม่เหมาะสมกัน โดยค่า K_i ของสมการบน (3.1) อาจจะน้อยกว่าค่า K_i ที่ได้จากสมการล่างมาก ทำให้ข้อมูลในสมการบน ($S_i <$ threshold) อาจถูกเรียกใช้บ่อยครั้งกว่าข้อมูลในสมการล่าง ($S_i \geq$ threshold) ทำให้ข้อมูลที่มีขนาดน้อยกว่า threshold ถูกแทนที่ด้วยข้อมูลใหม่ที่มากกว่า threshold ได้

ดังนั้นเพื่อแก้ปัญหาตรงจุดนี้จึงได้ปรับอัลกอริทึม RASM โดยข้อมูลที่เข้ามามีขนาดมากกว่า threshold ก็นำไปแทนที่ข้อมูลภายในแคชในกลุ่มข้อมูลที่มีขนาดมากกว่า threshold เหมือนกัน ส่วนข้อมูลที่เข้ามามีขนาดน้อยกว่า threshold ก็นำไปแทนที่ข้อมูลภายในแคชในกลุ่มข้อมูลที่มีขนาดน้อยกว่า threshold ดังแสดงในสมการที่ (4) และรูปที่ 1 ตามลำดับ

$$K_i = \begin{cases} \left(\frac{F_i}{S_i}\right) + L & \text{If } S_i < \text{Threshold} \\ F_i + L & \text{If } S_i \geq \text{Threshold} \end{cases} \quad (4)$$

โดยมีเงื่อนไขการแทนที่ข้อมูลดังนี้

$f_{i\ new}(R) > f_{i\ cache}$ โดยที่

$$\left[(S_{i\ new} < \text{Threshold}) \text{ and } (S_{i\ cache} < \text{Threshold})\right]$$

$$\text{or } \left[(S_{i\ new} \geq \text{Threshold}) \text{ and } (S_{i\ cache} \geq \text{Threshold})\right]$$

เมื่อ (R=>) แทนสัญลักษณ์การแทนที่ข้อมูลโดยนำข้อมูลด้านซ้ายไปแทนที่ข้อมูลด้านขวา

$f_{i\ new}$ คือข้อมูลใหม่ที่เข้ามาในแคช

$f_{i\ cache}$ คือข้อมูลในแคชที่จะถูกนำออกไป

$S_{i\ new}$ คือขนาดข้อมูลใหม่ที่เข้ามาในแคช (ไบต์)

$S_{i\ cache}$ คือขนาดข้อมูลที่จะถูกนำออกไป (ไบต์)

รูปที่ 1 โฟลว์แสดงการแทนที่ข้อมูลของอัลกอริทึม ISMRA

4. ผลการจำลองการทำงาน

เราได้จำลองการทำงานของอัลกอริทึม ISMRA โดยการเปรียบเทียบกับอัลกอริทึม RASM ข้อมูลที่นำมาใช้ในการจำลองการทำงานของ ISMRA เก็บมาจาก Log file ของเว็บแคชเซิร์ฟเวอร์ของ Clarknet (http://ita.ee.lbl.gov/html/contrib-Calgary-HTTP.html) ซึ่งข้อมูลแต่ละชุดใน Log file เป็นข้อมูลการร้องขอของเครื่องไคลเอนต์ในแต่ละครั้งที่รับมาเก็บไว้ในเว็บแคชเซิร์ฟเวอร์ โดยรายละเอียดของข้อมูลมีดังนี้

ข้อมูลที่ใช้ทดสอบ	จำนวน (ครั้ง)
จำนวนครั้งในการร้องขอ	1,465,049
จำนวนครั้งในการร้องขอที่ไม่ซ้ำกัน	35,365

ตารางที่ 1 แสดงจำนวนการร้องขอข้อมูล

1. ค่า Hit Rate ของอัลกอริทึม ISMRA ต้องเพิ่มขึ้นเมื่อเปรียบเทียบกับค่า Hit Rate ของอัลกอริทึม RASM ซึ่งสามารถแทนได้ด้วยค่า $\overline{\Delta H_i}$ (ค่า $\overline{\Delta H_i}$ ต้องเป็นบวก) สามารถแสดงสมการได้ดังนี้

$$\Delta H_i = \frac{HitRate_{ismra} - HitRate_{rasm}}{HitRate_{rasm}} \qquad (5.1)$$

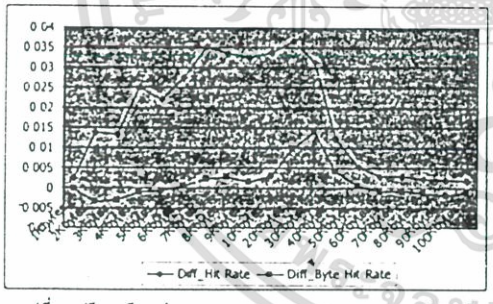
$$\overline{\Delta H_i} = \frac{\sum_{n=1}^{N} (\Delta H_i)_n}{N} \qquad (5.2)$$

2. ค่า Byte Hit Rate ของอัลกอริทึม ISMRA ต้องเพิ่มขึ้นเมื่อเปรียบเทียบกับค่า Byte Hit Rateของอัลกอริทึม RASM ซึ่งสามารถแทนได้ด้วยค่า $\overline{\Delta H_b}$ (ค่า $\overline{\Delta H_b}$ ต้องเป็นบวก) สามารถแสดงสมการได้ดังนี้

$$\Delta H_b = \frac{ByteHitRate_{ismra} - ByteHitRate_{rasm}}{ByteHitRate_{rasm}} \qquad (5.3)$$

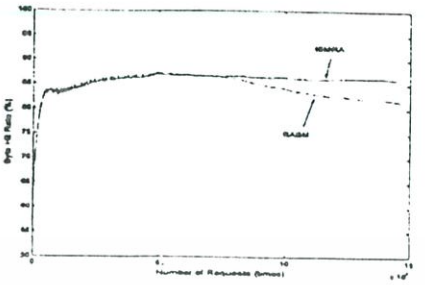
$$\overline{\Delta H_b} = \frac{\sum_{n=1}^{N} (\Delta H_b)_n}{N} \qquad (5.4)$$

3. ค่า Threshold ที่เหมาะสมคือค่าที่ทำให้ผลรวมของ $\overline{\Delta H_i}$ และ $\overline{\Delta H_b}$ มีค่ามากที่สุด ซึ่งจะได้ว่าที่ค่า threshold 30Kbytes จะได้ค่า Hit Rate และ Byte Hit Rate เหมาะสมที่สุด แสดงโดยกราฟที่ 4, 5, และ 6



กราฟที่ 4 เปรียบเทียบค่า Hit Rate และ Byte Hit Rate ของอัลกอริทึม ISMRA และ RASMที่ค่า threshold ต่างๆ

กราฟที่ 5 แสดงค่า Hit Rate ที่ threshold = 30 Kbytes



กราฟที่ 6แสดงค่าByte-Hit Rate ที่ threshold= 30 Kbytes

7. สรุปผลและแนวทางงานวิจัยในอนาคต

ในงานวิจัยนี้ได้ทำการปรับปรุงอัลกอริทึมการแทนที่ข้อมูล (Replacement Algorithm Selection Mechanism (RASM)) โดยเพิ่มส่วนของการแทนที่ข้อมูลโดยให้ข้อมูลใหม่แทนที่ข้อมูลภายในแคชที่มีขนาดข้อมูลอยู่ในช่วงขนาดเดียวกัน เรียกว่า ISMRA (Improvement of Selection Mechanism for Replacement Algorithm) ซึ่งผลการทดลองจะพบว่าอัลกอริทึม ISMRA จะให้ค่า Hit Rate ที่สูงกว่าอัลกอริทึมแบบเดิม (RASM) ซึ่งในแนวทางในอนาคตจะนำข้อมูลที่มีการกระจายในรูปแบบต่างมาทดสอบ

8. เอกสารอ้างอิง

[1] S. Jin and A. Bestavros "Popularity-aware greedy dual-size Web proxy caching algorithms." Proc. 20th International Conference on Distributed Computing Systems, April 2000. pp. 254-261.

[2] P. Cao and S. Irani "Cost Aware WWW Proxy Caching Algorithms." Proc. USENIX Symposium on Internet Technologies and Systems (USITS), December 1997, pp. 193-206.

[3] S. Podlipnig and L. Boszormenyi "A Survey of Web Cache Replacement Strategies." ACM Computing Surveys,vol.35,no.4, December 2003, pp. 374-398.

[4] T. Sontisiri, P. Sopechoke, S. Thipchaksurat, and Varakulsiripunth R. "Replacement Algorithm Selection Mechanism Considering File Size for Web Cache Server." Proc. ICCA2004 Conference,August 2004, pp. 122-127

Where C_i is the cost associated with bringing data i into the cache, S_i is the data size and L is a running age factor that starts at 0 and is updated for each replaced data f to the priority key of this data in the priority queue, i.e., $L = K_f$. This algorithm achieves the best hit rate when $C_i = 1$.

2.2 Least Frequency Used with Dynamic Aging (LFUDA)

The LFUDA algorithm was developed from LFU and LFU-Aging algorithm [4] that can achieve the highest of Byte Hit Rate. These algorithms accomplish by retaining the most popular data, regardless of data size. LFUDA calculates the priority key value (K_i) for data i using Eq. (2) by setting C_i to 1.

$$K_i = (C_i * f_i) + L \qquad (2)$$

2.3 Replacement Algorithm Selection Mechanism (RASM)

According to the characteristics of GDSF replacement algorithm and LFUDA replacement algorithm, there is a selection method for replacement algorithm for web cache server called Replacement Algorithm Selection Mechanism (RASM) [5]. RASM calculates K_i value in Eq.(3) shown below.

$$K_i = \begin{cases} \left(\frac{f_i}{S_i}\right) * L & \text{If } S_i < \text{Threshold} \quad (3.1) \\ f_i + L & \text{If } S_i > \text{Threshold} \quad (3.2) \end{cases}$$

Where S_i is the data size, L is a running age factor that starts from 0 and is updated for each replaced data f to the priority key of this data in the priority queue; i.e., $L = K_f$ and f_i is frequency of each i that starts from 1 when S_i threshold, otherwise, f_i will start from 0.

3. IMPROVEMENT OF REPLACEMENT ALGORITHM SELECTION MECHANISM (IRASM)

According to the RASM Algorithm, when cache is almost full and has new data, the data replacement will occur by removing data that has lowest priority key value (lowest K_i) out of web cache server until there is enough space to keep new data in cache, regardless of replaced data that has data size greater or lower than threshold. Example, the new data file which has size greater than threshold may replace to many data files in cache which size is lower than threshold. So it looks unfair on the ratio of calculating K_i between Eq.(3.1) and Eq.(3.2) because Eq.(3.1) is divided by size of data. Consequently,

- K_i of Eq.(3.1) is very lower than K_i of Eq.(3.2)
- Data that has size lower than threshold often be released from the cache and the hit rate is decreased

In this paper, we introduce a method for improving the RASM by considering the priority key for each group of data which is separated by depending of the threshold. By replacing new data which has size greater than threshold to data in cache which has size greater than threshold too. And new data which has size lower than threshold replace to data in cache which has size lower than threshold too as shown in Eq.(4) and Fig.1, respectively.

$$K_i = \begin{cases} \left(\frac{f_i}{S_i}\right) * L & \text{If } S_i < \text{Threshold} \\ f_i + L & \text{If } S_i > \text{Threshold} \end{cases} \qquad (4)$$

by this condition,

$f_{i,new}$ (R) > $f_{i,cache}$ which

[($S_{i,new}$ > Threshold) and($S_{i,cache}$ > Threshold)] or

[($S_{i,new}$ > Threshold) and($S_{i,cache}$ > Threshold)]

where $R_{i,i}$ is replacement from data in left side to data in right side

$f_{i,new}$ is new data that enter to cache

$f_{i,cache}$ is data that will be removed from cache

$S_{i,new}$ is size of new data (bytes)

$S_{i,cache}$ is size of data that will be removed from cache (bytes)

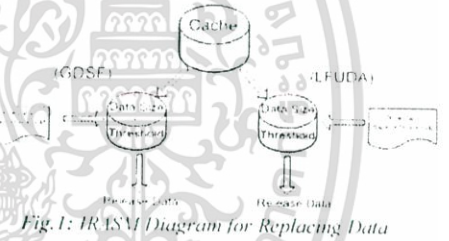


Fig.1: IRASM Diagram for Replacing Data

4. SIMULATION MODEL AND CONFIGURATION

We evaluate the performance of RASM by modeling the simulation model using matlab6.5. The simulation process can be explained as follows:

1. Define parameters of web cache server such as size of cache, type of replacement algorithm, high limit, low limit of cache, and threshold as follows:
   - The Threshold is set to 1, 10 and 100 Kbytes, respectively.
   - The size of cache is set to 32, 64 and 128 MBytes, respectively.
   - The high limit is equal to 90%.
2. Send requests of test data to web cache server one by one. The test data is collected from log file of web cache server of Clarknet (http://ita.ee.lbl.gov/html/contrib/Calgary-HTTP.html) by this detail
   - The number of requests is 1,465,049 times
   - The number of requests that is unique is 35,356 times

-Percent of requests which is separated by the period of data size is shown in Table 1

Table1: Percent of Requests along with Data Size

Size of Data (Bytes)	% of requests
0 - 1K	22.17
1K - 10K	54.56
10K - 100K	22.82
100K - 1M	0.44
Over 1M	0.01

3. Check files in the cache. If a request file is a hit, its priority key will be updated and wait for a next request. Otherwise, the process will check the size of cache. If it is higher than a high limit, the replacement algorithm will start to remove files from a cache until the size of cache decreases below a high limit. Then the replacement algorithm stops and the new file is kept in cache and waits for a next request.

5. RESULT OF IRASM ALGORITHMS

In this section, we evaluate the performance of IRASM in terms of Hit Rate and Byte Hit rate by comparing with RASM[8] as show in Fig.3 to Fig.8, respectively.

- For the cache size of 32 Mbytes, Hit Rate of IRASM is higher than Hit Rate of RASM at threshold 1Kbytes and 10 Kbytes and Byte Hit Rate is very close RASM at threshold 1 Kbytes and 10 Kbytes because IRASM replaces new data to data in cache in same size group. Big size data replace to big size data in cache not replace to small size and high access times data in cache. So Hit Rate of IRASM is very high.
- For the cache size of 64 Mbytes, the result is the same as 32 Mb.
- For the cache size of 128 Mbytes, the result is not different and does not show in this paper and for the higher size of cache the result is not different between IRASM and RASM algorithm too.

Fig.3: Hit Rate Threshold =1KB., CacheSize= 32MB

Fig.4: BytesHitRate Threshold =1KB,CacheSize= 32MB

Fig.5: Hit Rate Threshold =10KB ,CacheSize= 32MB

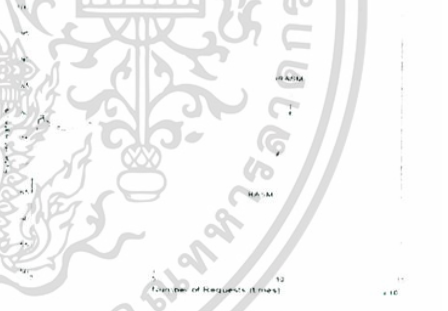


Fig.6:BytesHitRate Threshold =10KB,CacheSize= 32MB

Fig.7: Hit Rate Threshold =10KB., CacheSize= 64MB

Fig.8: BytesHitRate Threshold 10KB Cache Size 64MB

Fig.10: Hit Rate Threshold 30KB Cache Size 64MB

6. OPTIMUM THRESHOLD

Since "oHit and "oByte-Hit of RASM cannot be proved into mathematical equation, they depend on the environment of requests, the size of data, and the frequency of requests in each data. In order to consider the suitable threshold, we do a simulating by changing the threshold from 1 Kbytes to 100 Kbytes. We can explained as follows

1. Hit Rate of IRASM must increase from RASM denoted as MH_n (MH_n should be positive) in Eq (5.1) and Eq (5.2)

$$MH_n = HitRate_{(min,t)} - HitRate_{(t,\infty)} \quad (5.1)$$

$$HitRate_{(min,\infty)}$$

$$MH_n = \frac{\sum_{n=1}^{N} (MH_n)_n}{N} \quad (5.2)$$

2. Byte Hit Rate of IRASM must increase from RASM denoted as MH_n (should be positive) in Eq (5.3) and (5.4)

$$MH_n = ByteHitRate_{(min,t)} - ByteHitRate_{(t,\infty)} \quad (5.3)$$

$$ByteHitRate_{(min,\infty)}$$

$$MH_n = \frac{\sum_{n=1}^{N} (MH_n)_n}{N} \quad (5.4)$$

3. Optimum Threshold is the maximum value of sum MH_n and MH_n . So the threshold at 30Kbytes get best Hit Rate and Byte Hit Rate as shown in Fig.9 to Fig11.

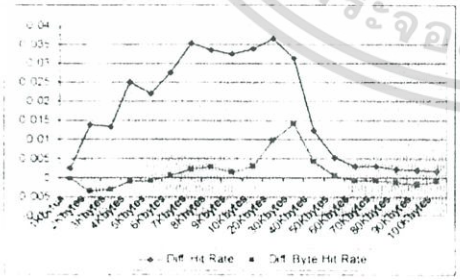


Fig.9: Comparison of Hit Rate and Byte Hit Rate of IRASM and RASM at each threshold

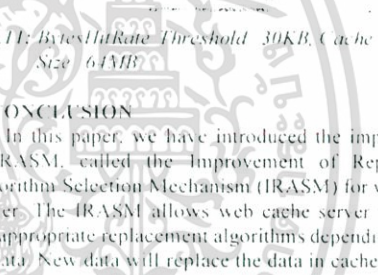


Fig.11: BytesHitRate Threshold 30KB Cache Size 64MB

7. CONCLUSION

In this paper, we have introduced the improvement of RASM, called the Improvement of Replacement Algorithm Selection Mechanism (IRASM) for web cache server. The IRASM allows web cache server to choose the appropriate replacement algorithms depending on size of data. New data will replace the data in cache that have the same range of data size. In the small cache size of 32 and 64 Mbytes, IRASM provides the higher Hit Rate than RASM.

8. REFERENCES

[1] J. Dilley, M. Arlitt and S. Perret, "Enhancement and Validation of Squid's Cache Replacement Policy," Technical Report HPL-1999-69, Hewlett-Packard Laboratories, May 1999.

[2] S. Jin and A. Bestavros, "Popularity-aware greedy dual-size Web proxy caching algorithms," Proc. 20th International Conference on Distributed Computing Systems, pp. 254-261, April 2000.

[3] P. Cao and S. Irani, "Cost Aware WWW Proxy Caching Algorithms," Proc. USENIX Symposium on Internet Technology and Systems (USITS), pp. 193-206, December 1997.

[4] S. Podlipnig and L. Boszormenyi, "A Survey of Web Cache Replacement Strategies," ACM Computing Surveys, vol. 35, no. 4, pp. 374-398, December 2003.

[5] T. Sontisiri, P. Sopeehoke, S. Thipchaksurat, and R. Varakulsiripunth, "Replacement Algorithm Selection Mechanism Considering File Size for Web Cache Server," Proc. ICCAS2004 Conference, August 2004, pp. 122-127.

ประวัติผู้เขียน

ชื่อ-นามสกุล	นายโกศล ถิรจิตโต
วันเดือนปีเกิด	วันที่ 4 พฤษภาคม 2521 ที่จังหวัดนครศรีธรรมราช
ที่อยู่	141 ถนนผดุงราษฎร์ ตำบลปากแพรก อำเภอทุ่งสง จังหวัดนครศรีธรรมราช
ประวัติการศึกษา	2543 จบการศึกษาวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง