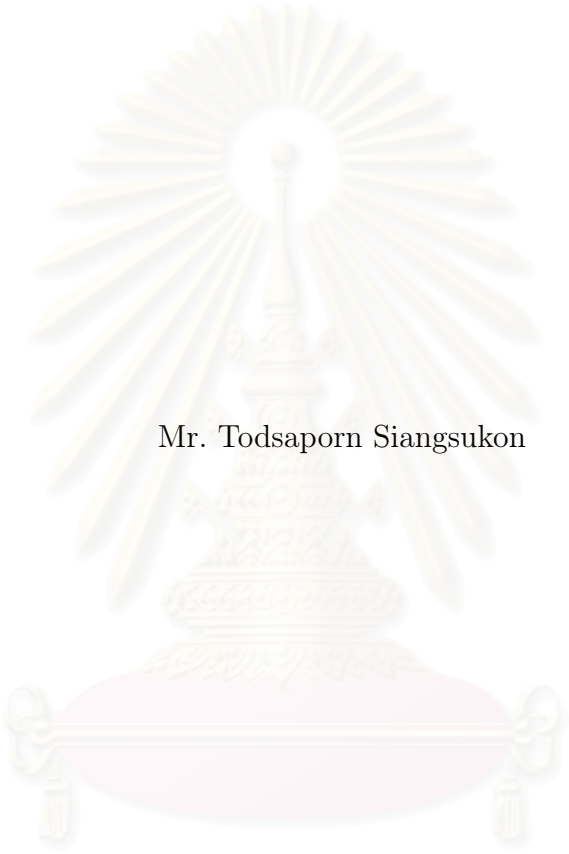
การปรับปรุงแถวคอยปฏิทินสำหรับการจำลองแบบเหตุการณ์ไม่ต่อเนื่อง

นายทศพร เสียงสุคนธ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2546
ISBN: 974-17-4684-9

IMPROVEMENT OF CALENDAR QUEUE FOR DISCRETE EVENT SIMULATION

Mr. Todsaporn Siangsukon



A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
Academic Year 2003

ISBN: 974-17-4684-9

หัวข้อวิทยานิพนธ์	การปรับปรุงแถวคอยปฏิทินสำหรับการจำลองแบบเหตุการณ์ไม่ต่อเนื่อง
โดย	นายทศพร เสียงสุคนธ์
สาขาวิชา	วิศวกรรมไฟฟ้า
อาจารย์ที่ปรึกษา	รองศาสตราจารย์ ดร.ลัญฉกร วุฒิสิทธิกุลกิจ
อาจารย์ที่ปรึกษาร่วม	อาจารย์ ดร.เชาวน์ดิศ อัศวกุล

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

. . . . . คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.ดิเรก ลาวัณย์ศิริ)

คณะกรรมการสอบวิทยานิพนธ์

. . . . . ประธานกรรมการ
(รองศาสตราจารย์ ดร.วาทิต เบญจพลกุล)

. . . . . อาจารย์ที่ปรึกษา
(รองศาสตราจารย์ ดร.ลัญฉกร วุฒิสิทธิกุลกิจ)

. . . . . อาจารย์ที่ปรึกษาร่วม
(อาจารย์ ดร.เชาวน์ดิศ อัศวกุล)

. . . . . กรรมการ
(อาจารย์ ดร.ชัยเชษฐ์ สายวิจิตร)

ทศพร เสียงสุคนธ์ : การปรับปรุงแถวคอยปฏิทินสำหรับการจำลองแบบเหตุการณ์ไม่ต่อเนื่อง (IMPROVEMENT OF CALENDAR QUEUE FOR DISCRETE EVENT SIMULATION) อ.ที่ปรึกษา : รศ. ดร.ลัญฉกร วุฒิสิทธิกุลกิจ, อ.ที่ปรึกษาร่วม : อาจารย์ ดร.เชาวน์ดิศ อัศวกุล, 79 หน้า. ISBN: 974-17-4684-9.

วิทยานิพนธ์ฉบับนี้เสนอแถวคอยแบบมีลำดับความสำคัญที่พัฒนามาจากแถวคอยปฏิทิน (Calendar Queue: CQ) โดยใช้ค่าตัวคูณความกว้างในการกำหนดค่าความกว้างถังแทนการใช้ค่าเฉลี่ยเวลาระหว่างเหตุการณ์ซึ่งใช้ใน CQ แถวคอยแบบมีลำดับความสำคัญที่เสนอในวิทยานิพนธ์ฉบับนี้คือ แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ (Adaptive Bucket width Calendar Queue: ABCQ) ซึ่งถูกตั้งชื่อตามคุณสมบัติการปรับความกว้างถังอัตโนมัติให้สอดคล้องกับค่าสุ่ม (random variate) ของเวลาของเหตุการณ์ที่ใช้ในการจำลอง

วิทยานิพนธ์ฉบับนี้จะใช้ค่าเฉลี่ยของเวลาในการดำเนินการเหตุการณ์คงค่า (hold operation) และการทดสอบกับโปรแกรมจำลองจริงในการเปรียบเทียบประสิทธิภาพของ ABCQ กับ CQ ซึ่งจะมีการทดสอบทั้งกับระบบนิ่ง (stationary) และระบบไม่นิ่ง (non-stationary) โดยใช้การเปลี่ยนค่าเฉลี่ยของค่าสุ่มระหว่างการทดสอบด้วยการดำเนินการเหตุการณ์คงค่า และการเปลี่ยนค่าโหลดในระบบระหว่างการจำลองเป็นตัวแทนของระบบไม่นิ่ง ผลการเปรียบเทียบแสดงให้เห็นว่า ABCQ สามารถลดเวลาในการประมวลผลได้ทั้งในการทดสอบด้วยการดำเนินการเหตุการณ์คงค่า และการทดสอบด้วยโปรแกรมจำลอง ซึ่งเวลาในการประมวลผลที่ลดลงจะเห็นได้อย่างชัดเจนในกรณีทดสอบกับระบบไม่นิ่ง

ภาควิชา	วิศวกรรมไฟฟ้า	**ลายมือชื่อนิสิต** . . . . .
สาขาวิชา	วิศวกรรมไฟฟ้า	**ลายมือชื่ออาจารย์ที่ปรึกษา** . . . . .
ปีการศึกษา	2546	**ลายมือชื่ออาจารย์ที่ปรึกษาร่วม** . . . . .

447 03220 21 : MAJOR ELECTRICAL ENGINEERING

KEY WORD: DISCRETE EVENT SIMULATION / PENDING EVENT SET / PRIORITY QUEUE / CALENDAR QUEUE.

TODSAPORN SIANGSUKON : IMPROVEMENT OF CALENDAR QUEUE FOR DISCRETE EVENT SIMULATION. THESIS ADVISOR: ASSOC. PROF. LUNCHAKORN WUTTISITTIKULKIJ, Ph.D., THESIS COADVISOR: CHAODIT ASWAKUL, Ph.D., 79 pp. ISBN: 974-17-4684-9.

In this thesis, a new priority queue algorithm has been proposed with the basis on the Calendar Queue (CQ). To select an appropriate bucket width, the proposed algorithm uses the width factor (W_f) instead of the average inter-event time being used in CQ. The proposed algorithm is called the Adaptive Bucket width Calendar Queue (ABCQ) to reflect its adaptability to the random variate of incremental time in simulation programs.

To evaluate ABCQ in comparison with CQ, both the conventional hold operation and real simulation scenarios have been adopted as a benchmark framework. In this thesis, the performance evaluation of ABCQ here focuses on both stationary and non-stationary systems. To emulate non-stationary systems, the mean of random variate in the hold operation as well as the system loading in the tested simulation scenarios are made time-dependent. The reported results suggest that ABCQ can decrease the processing time of both hold operation and simulation program especially when the systems are non-stationary.

Department	Electrical Engineering	**Student's signature** . . . . .
Field of study	Electrical Engineering	**Advisor's signature** . . . . .
Academic year	2003	**Co-advisor's signature** . . . . .

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยคำแนะนำและความช่วยเหลืออย่างดียิ่งของอาจารย์ที่ปรึกษาวิทยานิพนธ์ คือ รศ. ดร.ลัญฉกร วุฒิสิทธิกุลกิจ พร้อมทั้งแรงกระตุ้นและคำแนะนำของอาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม คือ ดร.เชาวน์ดิศ อัศวกุล ผู้วิจัยจึงขอกราบขอบพระคุณมา ณ ที่นี้

ขอขอบคุณโครงการเสริมสร้างความเชื่อมโยงระหว่างภาควิชาวิศวกรรมไฟฟ้าและภาคเอกชนทางด้านการวิจัยและพัฒนา (Cooperation Project between Department of Electrical Engineering and Private Sector for Research and Development) ปี พ.ศ. 2545 ที่ช่วยสนับสนุนเงินทุนสำหรับการทำงานวิจัย

ขอขอบคุณพี่ ๆ เพื่อน ๆ น้อง ๆ และคนรอบตัวของผู้วิจัยทุก ๆ คน ไม่ว่าจะเป็นที่อยู่ภายในศูนย์เชี่ยวชาญเทคโนโลยีระบบโทรคมนาคม (Center of Excellence in Telecommunication System) หรือที่ใดก็ตาม สำหรับความช่วยเหลือและกำลังใจ

สุดท้ายผู้วิจัยขอกราบขอบคุณบิดามารดาและครอบครัวที่ให้กำลังใจและการสนับสนุนแก่ผู้วิจัยเสมอมาจนสำเร็จการศึกษา

สารบัญ

สารบัญตาราง

สารบัญรูป

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การศึกษาระบบใด ๆ ก็ตามสามารถแบ่งลักษณะการศึกษาได้เป็น 2 ประเภทคือ การศึกษาระบบจริง (actual system) และการศึกษาระบบเสมือน (visual system) การศึกษาระบบจริงนั้นเราสามารถศึกษาโดยการวัดผลที่เกิดขึ้นจริงกับระบบที่กำลังสนใจ ส่วนการศึกษาระบบเสมือนนั้นจะทำในกรณีที่ไม่สามารถไปวัดผลการทำงานในระบบจริงได้ ดังนั้นจึงไปวัดผลจากแบบจำลองแทน โดยที่แบบจำลองนั้นสามารถแบ่งได้ 2 ประเภท คือ แบบจำลองทางกายภาพ (physical model) และแบบจำลองทางคณิตศาสตร์ (mathematical model) ซึ่งแบบจำลองชนิดที่งานวิจัยนี้นำมาศึกษา คือ แบบจำลองทางคณิตศาสตร์

ในการสร้างแบบจำลองทางคณิตศาสตร์ (ซึ่งต่อไปนี้จะเรียกสั้น ๆ ว่า “แบบจำลอง”) นั้นเราจำเป็นต้องแทนระบบที่เราสนใจให้อยู่ในรูปของตัวแปรคณิตศาสตร์ โดยตัวแปรเหล่านี้มีหน้าที่อธิบายสถานะ และคุณสมบัติของระบบ ซึ่งเราจะเรียกตัวแปรเหล่านี้ว่าเป็น ตัวแปรสถานะ (state variable) โดยระบบหนึ่ง ๆ อาจมีตัวแปรสถานะหลายตัวก็ได้ การสร้างตัวแปรสถานะนี้เอง คือการสร้างแบบจำลองทางคณิตศาสตร์

เป็นที่รู้กันโดยทั่วไปว่า หากสามารถหาความสัมพันธ์ระหว่างตัวแปรสถานะที่ถูกสร้างขึ้นมาได้ ก็จะสามารถใช้การวิเคราะห์ทางคณิตศาสตร์ในการหาผลลัพธ์หรือคำตอบที่ต้องการได้ แต่อย่างไรก็ตามยังคงมีระบบที่ต้องการศึกษาอีกเป็นจำนวนมากที่ไม่สามารถนำการวิเคราะห์ทางคณิตศาสตร์มาใช้เพื่อหาคำตอบได้ หรืออาจสามารถหาได้แต่มีความยุ่งยากในการวิเคราะห์สูงมาก เนื่องจากความสัมพันธ์ระหว่างตัวแปรสถานะของแบบจำลองเหล่านั้นมีความซับซ้อนสูงมาก ทำให้เกิดการพัฒนาเทคนิคในการจำลองด้วยเครื่องคอมพิวเตอร์เพื่อใช้สำหรับแก้ปัญหาระบบที่ไม่สามารถวิเคราะห์ทางคณิตศาสตร์เหล่านั้น

การจำลองด้วยเครื่องคอมพิวเตอร์นั้นจะอาศัยเครื่องคอมพิวเตอร์ในการสร้างเหตุการณ์เสมือนจริงขึ้นมาโดยเหตุการณ์ต่าง ๆ ที่สร้างขึ้นมาจะทำให้ค่าของตัวแปรสถานะเปลี่ยนแปลงไป โดยการเปลี่ยนแปลงที่เกิดขึ้นนี้จะมีลักษณะเช่นเดียวกันกับการเปลี่ยนแปลงที่เกิดขึ้นในระบบจริง

เพื่อควบคุมเหตุการณ์เสมือนจริงที่เกิดขึ้นในโปรแกรมจำลอง ตัวแปรอีกตัวหนึ่งซึ่งเรียกว่า นาฬิกาของโปรแกรมจำลอง (simulation clock) จึงถูกสร้างขึ้นมา โดยที่นาฬิกาจะถูกใช้เป็นตัวกำกับลำดับการเกิดของเหตุการณ์ต่าง ๆ ที่มีในโปรแกรม เนื่องจากเหตุการณ์เหล่านั้นจะถูกกำหนดให้เกิดขึ้นในเวลาที่ต่างกันไป ดังนั้นเราอาจบอกได้ว่า นาฬิกา คือ ตัวควบคุมลำดับการเกิดของเหตุการณ์ให้กับโปรแกรมจำลอง

โดยปกติแล้วนาฬิกาที่ใช้กันในชีวิตประจำวันจะเดินอย่างสม่ำเสมอครั้งละ 1 วินาทีได้เพียงรูปแบบเดียวเท่านั้น แต่ในการเดินของนาฬิกาของโปรแกรมจำลองนั้นมีการเดินด้วยกันทั้งหมด 2 ลักษณะ ซึ่งจะแบ่งตามลักษณะช่วงเวลาของการเดินได้ดังนี้

1. เดินด้วยช่วงเวลาคงที่: การเดินแบบนี้จะมีลักษณะคล้าย กับนาฬิกาที่เราเห็นอยู่ทั่ว ๆ ไป คือจะมีการกำหนดค่าที่แน่นอนไว้ก่อนค่าหนึ่งสำหรับนาฬิกา เช่น 1 วินาที หรือ 1/1000 วินาที เป็นต้น ค่าดังกล่าวนี้จะเรียกว่าค่าช่วงก้าว (time step) เมื่อโปรแกรมจำลองทำงาน นาฬิกาจะเริ่มจากวินาทีที่ 0 และเพิ่มเวลาขึ้นตามค่าช่วงก้าว ในแต่ละครั้งที่เวลาเพิ่มขึ้นโปรแกรมก็จะไปตรวจสอบดูว่าในช่วงเวลานี้มีเหตุการณ์ใดสมควรจะเกิดขึ้นบ้าง หลังจากนั้นจึงสร้างเหตุการณ์ดังกล่าวเหล่านั้นให้เกิดขึ้นจริงในโปรแกรม ในการเลือกใช้นาฬิกาแบบนี้หากต้องการจำลองระบบที่มีความแม่นยำด้านเวลาสูง ก็จำเป็นต้องกำหนดให้ค่าช่วงก้าวมีค่าต่ำมาก ๆ เพื่อให้แต่ละครั้งที่เวลาของนาฬิกามีค่าเพิ่มขึ้นจะไม่ก้าวข้ามเหตุการณ์ที่อาจจะเกิดขึ้นในช่วงเวลานั้น ๆ ไป (ตัวอย่างเช่น มีค่าช่วงก้าวเป็น 1 ขณะนี้นาฬิกาเป็น 0 มีเหตุการณ์หนึ่งเกิดขึ้นที่เวลา 0.2 และเหตุการณ์นี้จะส่งผลให้เกิดเหตุการณ์อีกเหตุการณ์หนึ่งที่เวลา 0.5 แต่โปรแกรมจะไม่รับรู้ว่ามีเหตุการณ์ที่เวลา 0.5 เกิดขึ้นจนกว่าจะเกิดเหตุการณ์ที่เวลา 0.2 ดังนั้นเหตุการณ์ที่เวลา 0.5 จะถูกข้ามไปเพราะตั้งค่าช่วงก้าวสูงเกินไป) แต่หากเรากำหนดให้ค่าช่วงก้าวมีค่าต่ำมากเกินไป จะมีผลให้โปรแกรมต้องเสียเวลาส่วนใหญ่ไปกับการตรวจสอบว่า มีเหตุการณ์เกิดขึ้นในช่วงเวลานั้น ๆ หรือไม่ แต่กลับพบว่าไม่มีเหตุการณ์ใดเกิดขึ้นในช่วงเวลานั้นเลย จึงสามารถสรุปได้ว่าการเลือกค่าช่วงก้าวสำหรับเพิ่มค่านาฬิกานั้นมีความสำคัญเป็นอย่างมาก แต่ในความเป็นจริงเราจะไม่รู้เลยว่าค่าช่วงก้าวที่เหมาะสมควรจะมีค่าเป็นเท่าไร ดังนั้นในการใช้งานจริงจึงมักจะตั้งค่าให้ต่ำไว้ก่อนเพื่อหลีกเลี่ยงการกระโดดข้ามเหตุการณ์ แต่การทำเช่นนี้จะส่งผลให้โปรแกรมต้องสิ้นเปลืองเวลาไปกับการตรวจสอบว่ามีเหตุการณ์เกิดขึ้นในแต่ละช่วงเวลาหรือไม่ ทำให้โปรแกรมทำงานได้ช้าลง การเขียนโปรแกรมจำลองด้วยวิธีนี้อาจถูกเรียกอีกชื่อหนึ่งว่า การจำลองแบบเวลาไม่ต่อเนื่อง (discrete time simulation) เพราะวิธีนี้การเดินของนาฬิกาจะกระโดดข้ามเวลาทีละช่วง (ซึ่งทำให้เวลาของนาฬิกาเดินอย่างไม่ต่อเนื่อง) แต่ละก้าวจะมีค่าเวลาค่าหนึ่งซึ่งเรียกว่า ค่าช่วงก้าวเป็นตัวกำหนด

2. เดินด้วยช่วงเวลาไม่คงที่: การเดินแบบนี้เป็นการเดินที่ถูกพัฒนามาเพื่อแก้ปัญหาที่เกิดขึ้นเนื่องจากการเดินด้วยช่วงเวลาคงที่ ดังได้อธิบายไปแล้วว่าการเดินด้วยช่วงเวลาคงที่จะมีปัญหาอยู่ที่จะต้องเลือกค่าช่วงก้าวที่เหมาะสมไม่เช่นนั้นผลการจำลองก็จะคลาดเลื่อน (เลือกค่าช่วงก้าวมากเกินไป) หรือไม่ก็ใช้เวลาในการประมวลผลนานเกินไป (เลือกค่าช้วงก้าวน้อยเกินไป) ดังนั้นวิธีนี้จึงแก้ปัญหาการเลือกค่าช่วงก้าวโดยการไม่ต้องเลือกค่าช่วงก้าวเสียเลย แต่จะให้นาฬิกามีการเพิ่ม

เวลาแต่ละครั้งด้วยช่วงเวลาไม่คงที่ โดยเวลาที่จะเพิ่มขึ้นนั้นจะดูจากเหตุการณ์ที่กำลังจะเกิด โดยจะเลือกค่าเวลาของเหตุการณ์ที่มีเวลาในการเกิดต่ำที่สุดออกมาแล้วจึงกระโดดไปยังเวลาที่การเกิดเหตุการณ์นั้นทันที ซึ่งจะกระโดดข้ามช่วงเวลาที่ไม่ได้เกิดเหตุการณ์ไป สังเกตุได้ว่าการเดินของนาฬิกาในวิธีนี้จะเดินอย่างไม่ต่อเนื่อง แต่ละก้าวนั้นจะมีเวลาในการเกิดเหตุการณ์เป็นตัวกำหนดช่วงก้าวของนาฬิกา วิธีนี้จึงถูกเรียกว่า การจำลองแบบเหตุการณ์ไม่ต่อเนื่อง (discrete event simulation)

แม้ว่าการจำลองแบบเหตุการณ์ไม่ต่อเนื่องจะสามารถแก้ปัญหาที่เกิดขึ้นในการจำลองแบบเวลาไม่ต่อเนื่องได้ก็จริง แต่สำหรับขั้นตอนในการเลือกหาเหตุการณ์ที่มีเวลาในการเกิดต่ำสุดนั้นยังคงต้องถูกพิจารณาอย่างรอบคอบ เพราะว่าขั้นตอนนี้เป็นปัจจัยสำคัญที่มีผลต่อความเร็วของการจำลองเป็นอย่างมาก เนื่องจากในโปรแกรมจำลองที่เขียนขึ้นมานั้นจะมีเหตุการณ์หลาย ๆ เหตุการณ์กำลังจะเกิดขึ้น ซึ่งเหตุการณ์เหล่านั้นจะถูกเก็บไว้ในรายการเหตุการณ์ (event list) และเหตุการณ์ต่าง ๆ ที่อยู่ในรายการเหตุการณ์นี้เอง คือ เหตุการณ์ที่จำเป็นต้องนำมาตรวจสอบดูเพื่อหาเหตุการณ์ที่มีเวลาในการเกิดต่ำที่สุด ดังนั้นหากเหตุการณ์เหล่านี้ไม่ได้ถูกจัดให้เป็นระเบียบแล้ว วิธีเดียวที่สามารถใช้ได้ คือ การนำเหตุการณ์ทุกเหตุการณ์มาเปรียบเทียบกันเพื่อหาเหตุการณ์ที่มีเวลาในการเกิดต่ำที่สุด แต่หากมีการจัดการที่ดีขึ้น เช่น มีการเรียงเหตุการณ์เหล่านั้นไว้ตามลำดับการเกิดก่อนหลังแล้ว ก็จะสามารถลดเวลาในการหาเหตุการณ์ที่มีเวลาในการเกิดต่ำที่สุดได้ ซึ่งโครงสร้างข้อมูลที่ใช้ในการเก็บและจัดเรียงรายการเหตุการณ์นี้เรียกว่า แถวคอยแบบมีลำดับความสำคัญ (priority queue)

แถวคอยแบบมีลำดับความสำคัญที่ดีนั้นต้องมีคุณสมบัติด้วยกัน 2 ข้อ คือ ความเร็ว และความทนทาน ซึ่งความเร็ว หมายถึง ความเร็วในการค้นหาเหตุการณ์ที่มีเวลาในการเกิดต่ำที่สุดออกจากรายการเหตุการณ์หรือเพิ่มเติมเหตุการณ์เข้าไปในรายการเหตุการณ์ ส่วนความทนทาน หมายถึง สามารถรักษาระดับความเร็วนั้นในสภาวะการทำงานที่หลากหลาย ซึ่งคุณสมบัติทั้ง 2 อย่างนี้ หากขาดอย่างหนึ่งอย่างใดไปอาจจะทำให้ไม่สามารถใช้โปรแกรมจำลองในการศึกษาระบบได้เนื่องจาก

- หากขาดคุณสมบัติข้อแรกไป (ความเร็ว) ก็จะทำให้โปรแกรมจำลองใช้เวลาในการประมวลผลนานมาก ซึ่งอาจนานเกินกว่าที่จะยอมรับได้
- หากขาดคุณสมบัติข้อที่สองไป (ความทนทาน) อาจทำให้เราไม่สามารถจำลองระบบที่มีขนาดใหญ่ได้ (มีเหตุการณ์จำนวนมากอยู่ใน รายการเหตุการณ์) ซึ่งระบบขนาดใหญ่นี้มักจะส่งผลให้เวลาในการทำงานเพิ่มขึ้น หรือสถานการณ์ในการทำงานของระบบที่จะจำลองก็อาจเป็นเหตุให้ แถวคอยแบบมีลำดับความสำคัญ มีการทำงานช้าลงอย่างมาก

ประสิทธิภาพของแถวคอยแบบมีลำดับความสำคัญแต่ละชนิดมักจะถูกเปรียบเทียบกันโดยใช้ค่าเฉลี่ยของเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่า (อธิบายไว้ในหัวข้อที่ 2.2) เพราะการดำเนินการเหตุการณ์คงค่าเป็นการรวมการทำงาน 2 อย่างเอาไว้ด้วยกัน คือ 1) การถอดเหตุการณ์ที่มีเวลาในการเกิดต่ำที่สุดออกจากรายการเหตุการณ์ 2) การใส่เหตุการณ์เข้าไปในรายการเหตุการณ์ และเมื่อทำงาน 2 อย่างนี้พร้อม ๆ กันจะไม่ทำให้จำนวนของเหตุการณ์ในรายการเหตุการณ์เกิดการเปลี่ยนแปลง ซึ่งมีลักษณะคล้ายกับการทำงานในสภาวะอยู่ตัว (steady state) ของโปรแกรมจำลอง โดยที่ผ่านมาแถวคอยแบบมีลำดับความสำคัญได้ถูกนำเสนอขึ้นมามากมายหลายชนิด เช่น แถวคอยแบบมีลำดับความสำคัญชนิดรายการเชิงเส้น (linear list) [1] แถวคอยแบบมีลำดับความสำคัญชนิดสองรายการ (Two List) [2] และแถวคอยแบบมีลำดับความสำคัญชนิดสเปย์ทรี (splay tree) [3] เป็นต้น ซึ่งทั้ง 3 วิธีที่กล่าวมานับได้ว่าแถวคอยแบบมีลำดับความสำคัญชนิดรายการเชิงเส้นมีการทำงานง่ายที่สุดแต่ก็จะมีค่าเฉลี่ยของเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่าสูงที่สุดด้วย ซึ่งค่าเฉลี่ยของเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่าของทั้งสามวิธีสามารถแสดงในรูปของความซับซ้อนเชิงเวลา (time complexity) เป็น $O(N)$, $O(N^{0.5})$ และ $O(\log N)$ ตามลำดับ

นอกเหนือจากแถวคอยแบบมีลำดับความสำคัญทั้ง 3 ชนิดที่กล่าวมาแล้ว ยังมีแถวคอยแบบมีลำดับความสำคัญอีก 3 ชนิดที่มี ความซับซ้อนเชิงเวลาเป็น $O(1)$ โดยในปี พ.ศ. 2520 แถวคอยแบบมีลำดับความสำคัญชนิดตัวชี้สองระดับ (two level pointer) ได้ถูกนำเสนอโดย แฟรนท่า (Franta) และมาลี่ (Maly) ซึ่งในกรณีที่แย่ที่สุดจะมีความซับซ้อนเชิงเวลาเป็น $O(N)$ แต่ข้อมูลจากการทดลองในรายงาน[4], [5] นั้นชี้ให้เห็นว่าจะมีประสิทธิภาพเฉลี่ยในสภาวะการทำงานทั่วไป ใกล้เคียงกับ $O(1)$ แต่อย่างไรก็ตามใน [6] แสดงให้เห็นว่าแถวคอยแบบมีลำดับความสำคัญชนิดตัวชี้สองระดับไม่ได้ทำงานได้ดีตามที่ได้ระบุไว้ใน [4], [5] ซึ่งอาจเกิดจาก มีการทำงานในส่วนหัว (overhead) มากเกินไปหรือมีความผิดพลาดของโปรแกรมที่ใช้ทดลอง

ในปี พ.ศ. 2523 ดาเวย (Davey) และโวเชอร์ (Vaucher) ได้นำเสนอแถวคอยแบบมีลำดับความสำคัญชนิดแบ่งรายการย่อย (partitioned list) [7] ซึ่งได้รับการพิสูจน์ว่ามี $O(1)$ แต่ยังมีปัญหาของรายการล้น (overflow list) โดยแถวคอยแบบมีลำดับความสำคัญชนิดนี้ได้ถูกทดสอบใน [6] เช่นกัน ซึ่งใน [6] แสดงให้เห็นว่าวิธีนี้มีผลการทำงานไม่ดีนักเมื่อมีจำนวนเหตุการณ์ในรายการเหตุการณ์อยู่น้อย นอกจากนั้นอาจจะมีความซับซ้อนเชิงเวลาเป็น $O(N)$ ถ้าหากมีเหตุการณ์จำนวนมากอยู่ในรายการล้น และก็ยังมีปัญหาเหมือนที่พบในแถวคอยแบบมีลำดับความสำคัญชนิดตัวชี้สองระดับ คือ มีการทำงานส่วนหัวมากเกินไป

นอกจากที่กล่าวมาทั้งหมดยังมีแถวคอยแบบมีลำดับความสำคัญอีกแบบหนึ่งถูกเสนอขึ้นในปี พ.ศ.

2531 นั่นก็คือ แถวคอยแบบมีลำดับความสำคัญชนิดแถวคอยปฏิทิน (calendar queue) [8] ซึ่งมีความซับซ้อนเชิงเวลาเป็น $O(1)$ และมีการทำงานส่วนหัวน้อย โดยจะมีการทำงานคล้ายแถวคอยแบบมีลำดับความสำคัญชนิดแบ่งรายการย่อย แต่มีการแก้ปัญหาของรายการล้นที่เกิดขึ้น ทำให้แถวคอยแบบมีลำดับความสำคัญชนิดแถวคอยปฏิทินนี้ได้รับความนิยมเป็นอย่างมากในช่วงหลายปีที่ผ่านมา แต่เนื่องจากในปัจจุบันสถานการณ์ที่ต้องการจำลองมีความหลากหลายมากขึ้น ซึ่งบางสถานการณ์จะส่งผลให้แถวคอยปฏิทินมีความซับซ้อนเชิงเวลาเป็น $O(N)$ ทำให้ต้องพัฒนาแถวคอยแบบมีลำดับความสำคัญให้มีความทนทานในการใช้งานมากขึ้น ค่าความซับซ้อนเชิงเวลาของแถวคอยแบบมีลำดับความสำคัญชนิดต่าง ๆ ที่ใช้กันอยู่ในปัจจุบันได้แสดงไว้ในตารางที่ 1.1

ตารางที่ 1.1 ความซับซ้อนเชิงเวลาของแถวคอยแบบมีลำดับความสำคัญชนิดต่าง ๆ

ชนิดของแถวคอยแบบมีลำดับความสำคัญ	ความซับซ้อนเชิงเวลา	หมายเหตุ
รายการเชิงเส้น (linear list)	$O(N)$	–
สองรายการ (two list)	$O(N^{0.5})$	–
สเปย์ทรี (splay tree)	$O(\log N)$	–
แบ่งรายการย่อย (partitioned list)	$O(1)$	มีปัญหาของรายการล้นซึ่งจะทำให้ความซับซ้อนเชิงเวลาเพิ่มขึ้น
แถวคอยปฏิทิน (calendar queue)	$O(1)$	–

1.2 แนวทางวิทยานิพนธ์

วิทยานิพนธ์นี้จะวิเคราะห์ปัญหาที่เกิดขึ้นกับแถวคอยแบบมีลำดับความสำคัญชนิดแถวคอยปฏิทิน โดยจะพิจารณาในสภาวะการใช้งานที่ค่าพารามิเตอร์ของตัวแปรสุ่มที่ใช้ในการสร้างเหตุการณ์ในโปรแกรมจำลองมีค่าแตกต่างกันในหลายมาตราส่วนของเวลา (time scale) หรือค่าพารามิเตอร์เหล่านี้มีการเปลี่ยนแปลงระหว่างการจำลองเป็นสำคัญ แถวคอยแบบมีลำดับความสำคัญที่ถูกนำเสนอในวิทยานิพนธ์ฉบับนี้จะเป็นการแก้ปัญหาที่เกิดขึ้นกับแถวคอยปฏิทินโดยจะปรับเปลี่ยนการทำงานของแถวคอยปฏิทินเดิมดังนี้

1. แก้ไข วิธีการเลือกค่าความกว้างถังเดิม
2. เพิ่มเติม เงื่อนไขในการกระตุ้นให้เกิดการเปลี่ยนค่าความกว้างถัง

การแก้ไขและเพิ่มเติมขั้นตอนทั้งสองนี้เข้าไปจะมีผลทำให้แถวคอยแบบมีลำดับความสำคัญชนิดแถวคอยปฏิทินมีความทนทานต่อสภาวะการใช้งานที่หลายหลายมากขึ้นโดยยังคงรักษาความสามารถใน

การทำงานในสภาวะการใช้งานเดิมได้

1.3 วัตถุประสงค์ของวิทยานิพนธ์

1. เพื่อนำเสนอวิธีเลือกค่าความกว้างถังของแถวคอยปฏิทินที่มีประสิทธิภาพสูงขึ้น
2. เพื่อนำเสนอวิธีกระตุ้นให้เกิดการเปลี่ยนค่าความกว้างถังของแถวคอยปฏิทินได้แม้โปรแกรมจะทำงานอยู่ในสภาวะอยู่ตัว

1.4 ขอบเขตวิทยานิพนธ์

1. เขียนโปรแกรมจำลองเพื่อทดสอบประสิทธิภาพการทำงานของแถวคอยปฏิทิน
2. ปรับปรุงวิธีการทำงานของแถวคอยปฏิทินตามที่เสนอในแนวทางวิทยานิพนธ์ และทดสอบประสิทธิภาพของวิธีที่เสนอเทียบกับแถวคอยปฏิทินแบบดั้งเดิม

1.5 ขั้นตอนการดำเนินงาน

1. ศึกษาหลักการเขียนโปรแกรมจำลองแบบเหตุการณ์ไม่ต่อเนื่อง
2. ศึกษาหลักการทำงานและทดสอบประสิทธิภาพของแถวคอยปฏิทิน
3. พัฒนาแถวคอยแบบมีลำดับความสำคัญชนิดแถวคอยปฏิทินตามแนวทางที่ได้เสนอไว้ในแนวทางวิทยานิพนธ์
4. วิเคราะห์และเปรียบเทียบผลที่ได้จากการทดสอบประสิทธิภาพของทั้ง 2 วิธี
5. สรุปผลและจัดทำเล่มวิทยานิพนธ์

1.6 ประโยชน์ที่คาดว่าจะได้รับ

1. พัฒนาแถวคอยปฏิทินให้มีความทนทานต่อสถานะการณ์ในการจำลองที่หลากหลายมากขึ้น
2. เพิ่มความเร็วในการทำงานของโปรแกรมจำลองและเป็นแนวทางในการเขียนโปรแกรมจำลองระบบที่มีขนาดใหญ่ต่อไป

1.7 ประมวลวิทยานิพนธ์

วิทยานิพนธ์ฉบับนี้แบ่งเนื้อหาออกเป็น 5 บท คือ

บทที่ 1 บทนำ: ประกอบไปด้วยเนื้อหาที่เกี่ยวกับแนวคิดเบื้องต้นของการจำลองและวิธีการเพิ่มค่านาฟิกาแบบต่าง ๆ พร้อมทั้งแนะนำแถวคอยแบบมีลำดับความสำคัญที่ถูกเสนอขึ้นมาเพื่อนำมาใช้ในโปรแกรมจำลอง

บทที่ 2 หลักการและทฤษฎี: ในบทนี้จะกล่าวถึงความรู้พื้นฐานที่เกี่ยวข้องกับงานวิจัยนี้ โดยจะแบ่งเนื้อหาออกเป็น 3 หัวข้อคือ 1) การเขียนโปรแกรมจำลองแบบเหตุการณ์ไม่ต่อเนื่อง ซึ่งจะกล่าวถึงหลักการเขียนโปรแกรมจำลองแบบเหตุการณ์ไม่ต่อเนื่องทั่ว ๆ ไปและชี้ให้เห็นถึงความจำเป็นที่ต้องนำแถวคอยแบบมีลำดับความสำคัญเข้ามาใช้กับโปรแกรมจำลอง 2) การดำเนินการเหตุการณ์คงค่า เนื้อหาในส่วนนี้จะบอกวิธีที่ใช้วัดประสิทธิภาพของแถวคอยแบบมีลำดับความสำคัญที่ใช้ในวิทยานิพนธ์ฉบับนี้ 3) หลักการทำงานของแถวคอยปฏิทิน ในส่วนนี้จะบอกถึงหลักการทำงานของแถวคอยปฏิทิน และวิเคราะห์ประสิทธิภาพเมื่อพารามิเตอร์ที่เลือกใช้มีค่าต่าง ๆ กัน (ทั้งกรณีที่เลือกได้ดี และกรณีที่เลือกได้ไม่ดี)

บทที่ 3 แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ: ในบทนี้จะนำเสนอหลักการของแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ ซึ่งเป็นแถวคอยแบบมีลำดับความสำคัญที่นำเสนอในวิทยานิพนธ์ฉบับนี้

บทที่ 4 ผลการทดสอบ: บทนี้จะแสดงผลการทดสอบการประสิทธิภาพการทำงานของแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ เปรียบเทียบกับแถวคอยปฏิทินแบบดั้งเดิม โดยจะมีการเปรียบเทียบประสิทธิภาพด้วยการดำเนินการเหตุการณ์คงค่า และการทำงานในโปรแกรมจำลองจริง

บทที่ 5 บทสรุป: ทำการสรุปผลการวิจัยและคุณประโยชน์ทั้งหมดที่มีในวิทยานิพนธ์ฉบับนี้

บทที่ 2

หลักการและทฤษฎี

ในบทนี้แสดงแนวคิดพื้นฐานของการจำลองแบบเหตุการณ์ไม่ต่อเนื่อง (discrete event simulation) เพื่อแสดงความสำคัญของการเลือกใช้แถวคอยแบบมีลำดับความสำคัญ (priority queue) ในการจำลองแบบเหตุการณ์ไม่ต่อเนื่อง หลังจากนั้นจึงนำเสนอวิธีการวัดประสิทธิภาพของแถวคอยแบบมีลำดับความสำคัญที่นิยมใช้กันอยู่ในปัจจุบัน และในส่วนสุดท้ายจะกล่าวถึงหลักการทำงานของแถวคอยปฏิทิน (calendar queue) ซึ่งเป็นแถวคอยแบบมีลำดับความสำคัญที่จะนำมาใช้ในวิทยานิพนธ์ฉบับนี้

2.1 การจำลองแบบเหตุการณ์ไม่ต่อเนื่อง

การจำลองแบบเหตุการณ์ไม่ต่อเนื่อง (discrete event simulation) [9] เป็นเทคนิคการจำลองด้วยคอมพิวเตอร์ (computer simulation) อีกแบบหนึ่งที่นิยมใช้กันอย่างกว้างขวางในหลายสาขาวิชา แต่ไม่ว่าเทคนิคนี้จะถูกนำไปใช้กับงานด้านใดก็ตาม การเขียนโปรแกรมจำลองนั้นยังคงมีส่วนคล้ายคลึงกันอยู่ ความคล้ายที่กล่าวถึงก็คือ การทำงานของโปรแกรมหลัก หรือหากจะกล่าวอีกนัยหนึ่ง คือ องค์ประกอบต่าง ๆ ที่ประกอบกันขึ้นมาเป็นโปรแกรมจำลองนั้นเหมือนกัน เช่น ตัวแปรที่ใช้ควบคุมการทำงานมีหน้าที่เหมือนกัน หรือโปรแกรมย่อยที่มีวัตถุประสงค์เดียวกัน (แม้ว่าเนื้อหาของโปรแกรมจะแตกต่างกัน แต่มีหน้าที่การทำงานเหมือนกัน) ซึ่งความสัมพันธ์ระหว่างองค์ประกอบแต่ละอย่างที่ได้กล่าวมาข้างต้นสามารถแสดงให้อยู่ในรูปผังการทำงานได้ตามรูปที่ 2.1

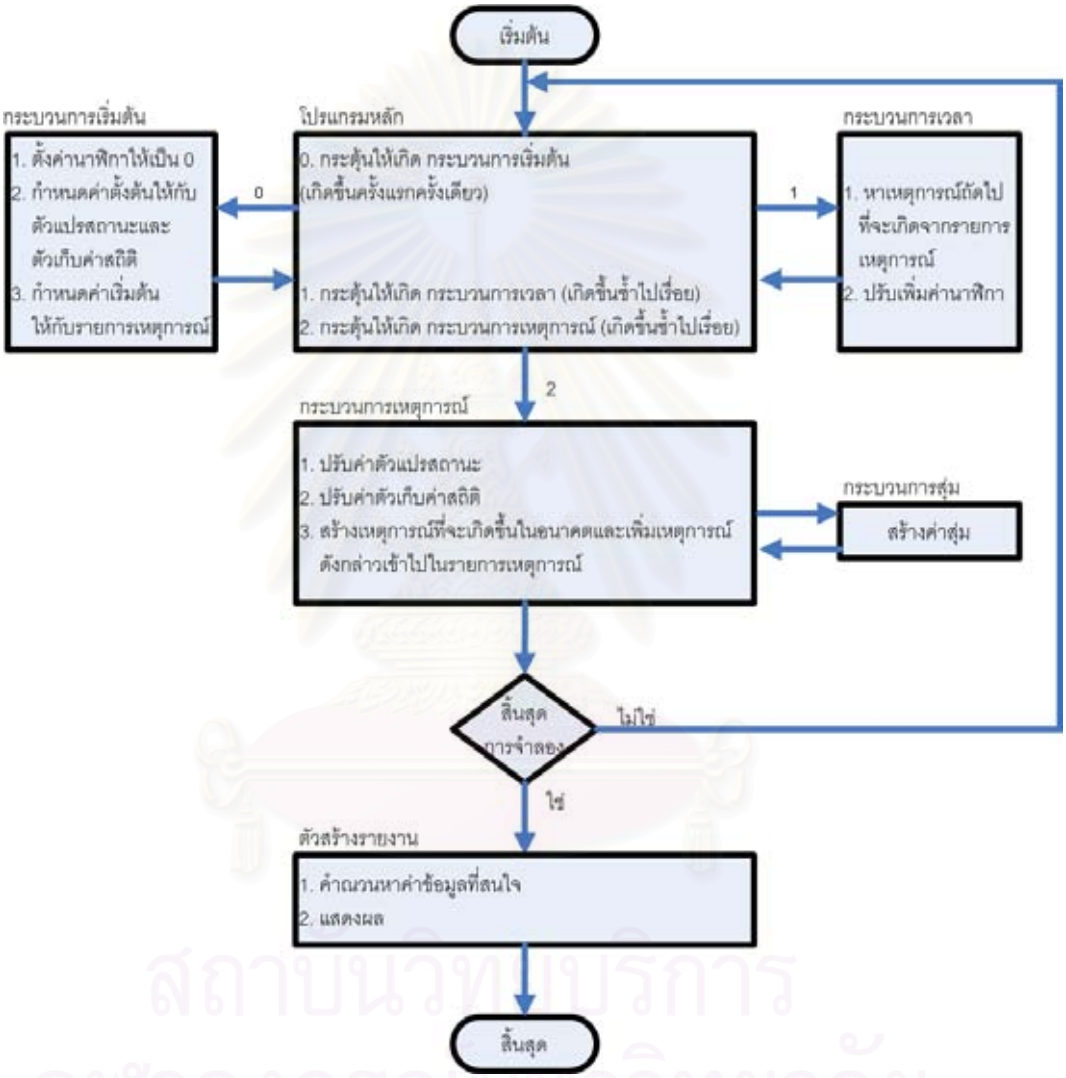
องค์ประกอบพื้นฐานที่จำเป็นต้องใช้ในการเขียนโปรแกรมจำลองมีดังนี้

- สถานะของระบบ (system state): ในโปรแกรมจะแทนสถานะของระบบด้วยกลุ่มของตัวแปรสถานะ (state variable) ที่ใช้ในการอธิบายระบบนั้น ๆ
- นาฬิกาของโปรแกรมจำลอง (simulation clock): ตัวแปรที่เก็บค่าเวลาของโปรแกรมจำลองมีหน้าที่ในการควบคุมลำดับการเกิดของเหตุการณ์ในโปรแกรมจำลอง
- รายการเหตุการณ์ (event list): ตัวแปรที่ใช้เก็บบันทึกเหตุการณ์ ซึ่งในโปรแกรมจำลองจะใช้แถวคอยแบบมีลำดับความสำคัญเป็นโครงสร้างข้อมูลที่ใช้สำหรับเก็บรายการเหตุการณ์
- บันทึกเหตุการณ์ (event notice): ตัวแปรที่นี้มีหน้าที่บันทึกเหตุการณ์ที่จะเกิดขึ้นในโปรแกรม เนื้อหาในบันทึกเหตุการณ์จะถูกแบ่งเป็น 2 ส่วน ส่วนแรกคือ เวลาของเหตุการณ์ (event time)

ใช้เก็บเวลาที่เหตุการณ์นั้นจะเกิดขึ้น ส่วนที่สองคือ รายละเอียดของเหตุการณ์ (event detail) ใช้เก็บข้อมูลที่เป็นรายละเอียดเกี่ยวกับเหตุการณ์

- ตัวนับค่าสถิติ (statistical counter): ตัวแปรที่ใช้เก็บข้อมูลทางสถิติของระบบ ซึ่งจะมีกี่ชนิดกี่ตัวแปรนั้นขึ้นอยู่กับความต้องการเก็บค่าสถิติของผู้ใช้
- กระบวนการเริ่มต้น (initiation routine): โปรแกรมย่อยซึ่งมีหน้าที่กำหนดค่าเริ่มต้นให้กับตัวแปรในโปรแกรมจำลอง กระบวนการนี้จะถูกเรียกขึ้นมาทำงานเพียงครั้งเดียวเท่านั้น คือ เมื่อนาฬิกาของโปรแกรมจำลองมีค่าเป็น 0
- กระบวนการเวลา (timer routine): โปรแกรมย่อยซึ่งมีหน้าที่ปรับค่านาฬิกาของโปรแกรม โดยจะนำค่าเวลาของบันทึกเหตุการณ์ที่มีเวลาในการเกิดต่ำที่สุดจากรายการเหตุการณ์มาใช้ในการปรับค่านาฬิกา
- กระบวนการเหตุการณ์ (event routine): โปรแกรมย่อยซึ่งมีหน้าที่ปรับค่าสถานะของระบบและสร้างบันทึกเหตุการณ์ใหม่ใส่เข้าไปในรายการเหตุการณ์เพื่อตอบสนองต่อเหตุการณ์ที่เกิดขึ้นในโปรแกรม ซึ่งเหตุการณ์แต่ละชนิดก็จะมีการทำงานที่แตกต่างกันออกไป
- กระบวนการสุ่ม (random routine): โปรแกรมย่อยซึ่งมีหน้าที่สร้างค่าสุ่มจากการกระจายแบบต่าง ๆ ที่ใช้ในโปรแกรมจำลอง
- ตัวสร้างรายงาน (report generator): โปรแกรมย่อยซึ่งมีหน้าที่คำนวณค่าที่ต้องการวัด (จากตัวนับค่าสถิติ) เพื่อแสดงให้ผู้ใช้รู้ผลที่ต้องการศึกษาเมื่อการจำลองเสร็จสิ้นแล้ว
- โปรแกรมหลัก (main program): โปรแกรมซึ่งมีหน้าที่จัดการกระบวนการอื่น ๆ ทั้งหมดในโปรแกรม และตรวจสอบการสิ้นสุดการจำลอง

การทำงานของโปรแกรมจำลองจะเริ่มขึ้นตั้งแต่นาฬิกาของโปรแกรมมีค่าเป็น 0 เมื่อโปรแกรมเริ่มทำงานก็จะไปทำงานที่โปรแกรมหลักก่อน ในครั้งแรกที่ทำงานในส่วนของโปรแกรมหลัก กระบวนการเริ่มต้นจะถูกเรียกให้ทำงานเพื่อตั้งค่าเริ่มต้นให้กับตัวแปรทุกตัวในโปรแกรมก่อนจะใช้งานตัวแปรนั้น ๆ โดยนาฬิกาของโปรแกรมถูกตั้งค่าให้เป็น 0 ตัวแปรสถานะถูกกำหนดค่าเริ่มต้นให้สอดคล้องกับสถานการณ์ที่จะจำลอง ตัวนับค่าสถิติมีค่าเป็น 0 นอกเหนือจากการตั้งค่าเริ่มต้นให้กับตัวแปรแล้ว ในกระบวนการนี้ยังต้องสร้างเหตุการณ์เริ่มต้นไว้ในรายการเหตุการณ์ด้วย หลังจากนั้นจึงกลับมาทำงานในขั้นตอนต่อไปของโปรแกรมหลัก ขั้นตอนถัดไปของโปรแกรมหลักคือการเรียกกระบวนการเวลาขึ้นมาทำงาน โดยกระบวนการเวลานั้นจะมีการทำงาน 2 อย่างด้วยกัน อย่างแรกคือการถอดบันทึกเหตุการณ์ที่มี



รูปที่ 2.1 ผังการทำงานของโปรแกรมจำลองแบบเหตุการณ์ไม่ต่อเนื่อง

เวลาในการเกิดต่ำที่สุดออกมาจากรายการเหตุการณ์ หลังจากนั้นจึงปรับค่าเวลาของนาฬิกาไปยังเวลาของบันทึกเหตุการณ์นั้น ซึ่งก็หมายความว่าเหตุการณ์นั้นกำลังจะเกิดขึ้นนั่นเอง เมื่อทำงานทั้งสองอย่างนี้เสร็จแล้วก็จะคืนการควบคุมกลับไปยังโปรแกรมหลักอีกครั้ง ขั้นตอนถัดไป (สุดท้าย) ก็คือเรียกกระบวนการเหตุการณ์ขึ้นมาทำงาน ในกระบวนการนี้จะเป็นการปรับค่าตัวแปรสถานะของระบบ การปรับค่านี้ขึ้นกับชนิดของบันทึกเหตุการณ์ที่ถูกถอดออกมาก่อนหน้านี้ หลังจากที่สถานะของระบบมีการเปลี่ยนไปแล้วก็จะทำการเก็บค่าสถิติต่าง ๆ ที่สนใจเอาไว้แล้วจึงสร้างบันทึกเหตุการณ์ไปเก็บไว้ในรายการเหตุการณ์ โดยบันทึกเหตุการณ์ที่ถูกสร้างขึ้นเป็นบันทึกเหตุการณ์ที่เป็นผลต่อเนื่องมาจากเหตุการณ์ที่เกิดขึ้นนี้เอง ในขั้นตอนนี้อาจจำเป็นต้องใช้ค่าสุ่ม ซึ่งสามารถได้มาโดยการเรียกกระบวนการสุ่มขึ้นมาทำงาน เมื่อทำงานในส่วนของกระบวนการเหตุการณ์เสร็จแล้ว ก็นับได้ว่าโปรแกรมหลักได้ทำงานครบแล้ว จากนั้นจึงไปตรวจสอบดูว่าจะหยุดการจำลองได้แล้วหรือไม่ การตรวจสอบนี้ก็มักจะตรวจสอบโดยใช้เวลาเป็นตัวกำหนด กล่าวคือนาฬิกาของโปรแกรม มีค่ามากเกินกว่าค่าเวลาที่กำหนดค่าหนึ่งแล้วหรือไม่นั่นเอง หากนาฬิกายังมีค่าน้อยกว่าค่าที่กำหนดเอาไว้ก็จะกลับไปทำงานในโปรแกรมหลักใหม่ และจะวนทำงานเช่นนี้ต่อไปจนกว่านาฬิกาของโปรแกรมจะมีค่ามากกว่าค่าเวลาที่กำหนดเอาไว้ หลังจากนั้นจึงนำค่าสถิติที่เก็บเอาไว้ไปคำนวณเป็นข้อมูลที่เราสนใจเพื่อแสดงผล ซึ่งเมื่อโปรแกรมทำงานมาถึงส่วนนี้แล้วก็จะเป็นการสิ้นสุดการจำลอง

ในเนื้อหาที่ผ่านมาอธิบายถึงหลักการทำงานของโปรแกรมโดยรวมไปครบถ้วนแล้ว ในส่วนต่อไปจะเป็นการแสดงตัวอย่างการจำลองเพื่อให้เกิดความเข้าใจในการทำงานแต่ละขั้นตอนของโปรแกรมจำลองได้ดียิ่งขึ้น โดยจะมีการแสดงค่าของตัวแปรที่ใช้ในโปรแกรม ณ ค่านาฬิกาต่าง ๆ ด้วย

ตัวอย่างนี้เป็นการศึกษาการทำงานของตู้โทรศัพท์สาธารณะ ซึ่งมีเงื่อนไขการทำงานดังนี้ ระยะเวลาระหว่างผู้ใช้แต่ละคนที่จะเข้ามาใช้บริการมีการกระจายแบบเลขชี้กำลัง (exponential distribution) $\mu = 60$ วินาที เวลาที่ผู้ใช้ใช้บริการมีการกระจายแบบเลขชี้กำลัง $\mu = 60$ วินาทีเช่นกัน ตู้โทรศัพท์นี้มีที่ให้ผู้ใช้เข้าแถวรอรับบริการได้ 2 คน ในขณะที่มีผู้ใช้รออยู่ในแถว 2 คนแล้ว หากมีผู้ใช้คนถัดไป (คนที่ 3 ที่จะมาเข้าแถว) ต้องการเข้ามาใช้บริการก็จะถือว่าผู้ใช้คนนั้นเห็นว่ามีคนรออยู่ในคิวมากแล้วจึงเลิกล้มความต้องการใช้บริการไป ซึ่งทำให้เกิดการสูญเสียลูกค้า (ผู้ใช้บริการ)

ในการเขียนโปรแกรมจำลองนั้นเราต้องเริ่มต้นจากการสร้างตัวแปรสถานะก่อน โดยที่ตัวแปรสถานะนี้จะต้องสอดคล้องกับค่าสถิติที่เราต้องการศึกษาเพราะบางครั้งเมื่อเราต้องการศึกษาค่าสถิติหลาย ๆ ค่า แต่ตัวแปรสถานะที่เราใช้อธิบายระบบไม่ได้ครอบคลุมถึงรายละเอียดของข้อมูลสถิติเหล่านั้น เป็นเหตุให้ไม่สามารถเก็บค่าสถิติที่ต้องการได้ ในตัวอย่างนี้เราต้องการศึกษาอัตราการสูญเสียผู้ใช้บริการเพียงอย่างเดียว (อัตราการเกิดผู้เข้าแถวคนที่ 3) ดังนั้น ตัวแปรสถานะที่ใช้อธิบายสถานะของระบบจะมีอยู่

ด้วยกันทั้งหมด 2 ตัวแปรด้วยกัน คือ 1) จำนวนผู้ใช้ในแถว 2) สถานะของตู้โทรศัพท์ อย่างที่ชื่อของตัวแปรได้บอกไว้อย่างชัดเจนแล้ว จำนวนผู้ใช้ในแถวจะเก็บค่าที่บอกว่า ณ เวลานั้นมีผู้ใช้อยู่ในแถวเพื่อรอรับการใช้บริการกี่คน ซึ่งในระบบนี้จะมีค่าได้ ตั้งแต่ 0 ถึง 2 ส่วนสถานะของตู้โทรศัพท์นั้นจะมีค่าได้ 2 อย่าง คือ ว่าง และไม่ว่าง โดยที่ ว่าง คือ ตู้โทรศัพท์ไม่มีคนใช้และพร้อมจะให้บริการกับผู้ใช้คนถัดไป ส่วน ไม่ว่าง คือ มีผู้ใช้กำลังใช้ตู้โทรศัพท์อยู่ผู้ใช้คนถัดไปที่เข้ามาต้องไปรอในแถวคอยที่จัดไว้ให้

นอกเหนือจากตัวแปรสถานะที่ถูกสร้างขึ้นแล้วในโปรแกรมยังต้องสร้างค่าสุ่มขึ้นมา โดยค่าสุ่มนี้จะถูกสร้างขึ้นเพื่อใช้เป็นค่า เวลาระหว่างการเข้ามาใช้งานของผู้ใช้แต่ละคน และเวลาของการใช้งานของผู้ใช้แต่ละคนโดยเราสมมติว่าค่าสุ่มเหล่านี้ได้ถูกสร้างขึ้นมาก่อนแล้วดังนี้

A1 = 88, A2 = 30, A3 = 43, A4 = 7, A5 = 15, A6 = 47, . . .
S1 = 12, S2 = 69, S3 = 29, S4 = 13, S5 = 5 , S6 = 18, . . .

โดยที่ A1, A2, A3, ... คือค่าเวลาระหว่างการเข้ามาใช้งานของผู้ใช้แต่ละคน
S1, S2, S3, ... คือค่าเวลาของการใช้งานของผู้ใช้แต่ละคน

ในรูปที่ 2.2 แสดงการทำงานของโปรแกรมจำลองระบบโทรศัพท์เมื่อนาฬิกามีค่าเป็น 0, 88, 100, 118, 161, 168, 183, 187 และ 216 ตามลำดับ ซึ่งในรูปนั้นแบ่งออกเป็น 2 ส่วนคือ 1) ส่วนระบบจริงจะเป็นรูปแสดงการใช้งานตู้โทรศัพท์ ผู้ใช้ที่กำลังใช้ตู้โทรศัพท์อยู่จะยืนหน้าตู้ ในขณะที่ผู้ใช้ที่กำลังคอยรับบริการจะยืนต่อแถวกันอยู่ทางด้านซ้าย 2) ส่วนแบบจำลองจะแสดงค่าของตัวแปรที่คอมพิวเตอร์ต้องใช้ในโปรแกรมจำลอง โดยในกล่องที่มีสีเข้มแสดงว่าค่าของตัวแปรมีการเปลี่ยนแปลง ส่วนกล่องที่เป็นสีขาวแสดงว่าค่าของตัวแปรไม่มีการเปลี่ยนแปลง

ตัวแปรที่ใช้ในโปรแกรม

- ตัวแปรสถานะของตู้โทรศัพท์: จะมีค่าเป็น 0 หรือ 1 เพื่อใช้แทนความหมายว่า ว่าง และไม่ว่าง ตามลำดับ
- ตัวแปรจำนวนผู้ใช้ในแถว: ใช้เก็บค่าจำนวนผู้ใช้ที่คอยอยู่ จะมีค่าเป็น 0, 1 หรือ 2 เท่านั้น หากมีผู้ใช้คนใหม่เข้ามาขณะที่ตัวแปรนี้มีค่าเป็น 2 จะถือว่าระบบสูญเสียผู้ใช้บริการคนนั้นไป
- ตัวแปรนาฬิกา: ใช้เก็บค่าเวลาของโปรแกรมจำลอง
- รายการเหตุการณ์: ใช้เก็บเหตุการณ์ที่เกิดขึ้นในโปรแกรม ซึ่งมีอยู่ด้วยกันทั้งหมด 2 ชนิด คือ มีผู้ใช้คนใหม่เข้ามาในระบบ และผู้ใช้ที่ใช้บริการอยู่ใช้บริการเสร็จเรียบร้อยแล้ว ซึ่งในโปรแกรมจะใช้

รายการเชื่อมโยง (linked list) ที่มีการเรียงลำดับข้อมูลในการเก็บรายการเหตุการณ์ โดยสมาชิกแต่ละตัวจะเรียกว่าบันทึกเหตุการณ์ (event notice) ซึ่งใช้เก็บค่าเวลาของเหตุการณ์ (event time) และรายละเอียดของเหตุการณ์ (event detail)

- ตัวแปรปริมาณการสูญเสียผู้ใช้บริการ: ใช้นับจำนวนผู้ใช้ที่ระบบสูญเสียไปเนื่องจากมีผู้ใช้เข้ามาในระบบขณะที่ตัวแปรจำนวนผู้ใช้ในแถวมีค่าเป็น 2

การทำงานของโปรแกรมจำลอง ณ ค่านาฬิกาต่าง ๆ

- นาฬิกา = 0 กำหนดค่าตั้งต้นให้กับระบบ: เมื่อโปรแกรมเริ่มทำงาน ตัวแปรทุกตัวในโปรแกรมต้องถูกกำหนดค่าตั้งต้น โดยจะเริ่มจากส่วนควบคุมก่อน ส่วนควบคุมนี้ประกอบไปด้วยตัวแปร 2 ตัว คือ นาฬิกา และรายการเหตุการณ์ โดยตัวแปร นาฬิกาจะถูกกำหนดค่าให้เป็น 0 ส่วนรายการเหตุการณ์ก็จะใส่บันทึกเหตุการณ์เข้าไป บันทึกเหตุการณ์ที่ใส่ไป คือ ผู้ใช้เข้ามาในระบบ ณ เวลา 88 ซึ่งค่านี้ได้มาจากค่าสุ่ม A1 ส่วนเหตุการณ์ผู้ใช้ใช้โทรศัพท์เสร็จไม่ได้ถูกใส่เข้าไปด้วยเพราะไม่มีผู้ใช้กำลังใช้บริการอยู่ หลังจากนั้นจึงไปกำหนดค่าให้กับตัวเก็บค่าสถิติให้มีค่าเป็น 0 (ซึ่งก็คือตัวแปรปริมาณการสูญเสียผู้ใช้บริการ) เพราะในตอนเริ่มต้นนั้นยังไม่มีการสูญเสียของผู้ใช้บริการ และสุดท้ายก็จะไปกำหนดค่าให้กับตัวแปรสถานะ (สถานะของตู้โทรศัพท์ และจำนวนผู้ใช้ในแถว) ให้มีค่าเป็น 0 ทั้งคู่เพื่อบอกว่าตู้โทรศัพท์มีสถานะว่างอยู่ และไม่มีผู้ใช้รอคอยใช้ตู้โทรศัพท์

- นาฬิกา = 88 ผู้ใช้คนที่ 1 เข้ามาในระบบ: เมื่อทำการกำหนดค่าตั้งต้นทั้งหมดเรียบร้อยแล้ว หลังจากนั้นโปรแกรมก็จะถอดเอาบันทึกเหตุการณ์ที่มีเวลาเกิดต่ำที่สุดออกมาจากรายการเหตุการณ์ เพื่อเลื่อนค่านาฬิกาไปยังบันทึกเหตุการณ์นั้น ซึ่งบันทึกเหตุการณ์ดังกล่าวก็คือ มีผู้ใช้เข้ามาในระบบที่เวลา 88 นั่นเอง ณ ค่านาฬิกาค่านี้ โปรแกรมจะต้องทำงาน 3 อย่างด้วยกันเพื่อตอบสนองต่อเหตุการณ์ ผู้ใช้คนที่ 1 เข้ามาในระบบ งานอย่างแรกเมื่อผู้ใช้เข้ามาในระบบ คือ โปรแกรมจะต้องไปตรวจสอบดูว่าเครื่องโทรศัพท์ว่างอยู่หรือไม่ ซึ่งจะพบว่าตัวแปรสถานะของตู้โทรศัพท์มีค่าเป็น 0 ดังนั้นเราจะเปลี่ยนค่าตัวแปรนี้ให้เป็น 1 เพื่อบอกว่าผู้ใช้คนที่ 1 ได้เข้าไปใช้บริการตู้โทรศัพท์แล้ว งานอย่างที่ 2 คือ สร้างบันทึกเหตุการณ์การเข้ามาของผู้ใช้บริการคนถัดไปใส่เข้าไปในรายการเหตุการณ์ โดยเวลาของเหตุการณ์ = 88 + A2 = 118 งานอย่างที่ 3 คือ สร้างบันทึกเหตุการณ์ผู้ใช้ใช้โทรศัพท์เสร็จ เวลาของเหตุการณ์ = 88 + S1 = 100

- นาฬิกา = 100 ผู้ใช้คนที่ 1 ใช้บริการเสร็จ: เมื่อผู้ใช้ใช้บริการเสร็จแล้วโปรแกรมจะต้องไปตรวจดูว่ามีผู้ใช้คนอื่นกำลังรอคอยรับบริการอยู่หรือไม่ ซึ่งในตอนนี้พบว่าไม่มีผู้ใช้มารอรับบริการอยู่

เลย โปรแกรมจึงปรับค่าสถานะของตู้โทรศัพท์ให้เป็น 0 ดังนั้นตอนนี้รายการเหตุการณ์จึงมีบันทึกเหตุการณ์อยู่เพียงใบเดียว คือ ผู้ใช้เข้ามาในระบบ ณ เวลา 118

- นาฬิกา = 118 ผู้ใช้คนที่ 2 เข้ามาในระบบ: ครั้งนี้ในรายการเหตุการณ์มีบันทึกเหตุการณ์เพียงใบเดียว คือ มีผู้ใช้เข้ามาในระบบ และตู้โทรศัพท์มีสถานะว่าง ดังนั้นจะปรับค่าตัวแปรต่าง ๆ ดังนี้ สถานะของตู้โทรศัพท์ = 1 เพิ่มบันทึกเหตุการณ์ผู้ใช้เข้ามาในระบบ เวลาของเหตุการณ์ = 118 + A3 = 161 และเพิ่มบันทึกเหตุการณ์ผู้ใช้ใช้โทรศัพท์เสร็จ เวลาของเหตุการณ์ = 118 + S2 = 187

- นาฬิกา = 161 ผู้ใช้คนที่ 3 เข้ามาในระบบ: ครั้งนี้มีผู้ใช้เข้ามาในระบบขณะที่ตู้โทรศัพท์ยังถูกใช้งานอยู่ ผู้ใช้ที่เพิ่งเข้ามาใหม่นี้จะต้องไปรออยู่ในแถวก่อน โดยจะทำการเพิ่มค่าตัวแปรจำนวนผู้ใช้ในแถวขึ้นไปหนึ่งค่า ซึ่งก็คือเปลี่ยนจาก 0 เป็น 1 นั่นเอง แล้วจึงสร้างบันทึกเหตุการณ์ผู้ใช้คนถัดไปจะเข้ามาสู่ระบบ เวลาของเหตุการณ์ = 161 + A4 = 168 ใส่เข้าไปในรายการเหตุการณ์

- นาฬิกา = 168 ผู้ใช้คนที่ 4 เข้ามาในระบบ: เหตุการณ์ที่เกิดขึ้นในครั้งนี้มีลักษณะเหมือนกับเหตุการณ์ เมื่อนาฬิกามีค่า 161 คือ มีผู้ใช้ใช้งานอยู่ ดังนั้นเหตุการณ์นี้ก็จะได้รับการตอบสนองเหมือนกัน โดยจะปรับค่าต่าง ๆ ดังนี้ เพิ่มค่าตัวแปรจำนวนผู้ใช้ในแถวขึ้นไปหนึ่งค่า ซึ่งก็คือเปลี่ยนจาก 1 เป็น 2 สร้างบันทึกเหตุการณ์ผู้ใช้คนถัดไปจะเข้ามาสู่ระบบ เวลาของเหตุการณ์ = 168 + A5 = 183

- นาฬิกา = 183 ผู้ใช้คนที่ 5 เข้ามาในระบบ: เหตุการณ์ที่เกิดขึ้นในครั้งนี้ดูไปแล้วมีลักษณะเหมือนกับเหตุการณ์ เมื่อนาฬิกามีค่า 161 และ 168 คือ มีผู้ใช้ใช้งานอยู่ แต่ในครั้งนี้แถวคอยเต็มแล้ว ดังนั้นเหตุการณ์นี้จึงแตกต่างกับเหตุการณ์เมื่อนาฬิกาเป็น 161 และ 168 ซึ่งสิ่งที่เกิดขึ้นก็ คือ เพิ่มปริมาณการสูญเสียผู้ใช้บริการขึ้นหนึ่งค่า (จาก 0 เป็น 1) และสร้างบันทึกเหตุการณ์ผู้ใช้คนถัดไปจะเข้ามาสู่ระบบ เวลาของเหตุการณ์ = 183 + A6 = 230

- นาฬิกา = 187 ผู้ใช้คนที่ 2 ใช้บริการเสร็จ: ในครั้งนี้เมื่อผู้ใช้บริการเสร็จแล้วแต่ยังมีผู้ใช้คนอื่นกำลังรอคอยรับบริการอยู่จึงจะไม่มีการเปลี่ยนค่าสถานะของตู้โทรศัพท์ (ให้คงค่า 1 ไว้) แต่จะไปลดค่าตัวแปรจำนวนผู้ใช้ในแถวลงไปหนึ่งค่า ซึ่งก็คือเปลี่ยนจาก 2 เป็น 1 แล้วจึงสร้างบันทึกเหตุการณ์ผู้ใช้ (คนที่เพิ่งเข้าไปใช้บริการ) ใช้บริการเสร็จ เวลาของเหตุการณ์ = 187 + S3 = 216

- นาฬิกา = 216 ผู้ใช้คนที่ 3 ใช้บริการเสร็จ: เหตุการณ์ที่เกิดขึ้นในครั้งนี้มีลักษณะเหมือนกับเหตุการณ์ เมื่อนาฬิกามีค่า 187 คือ ผู้ใช้ใช้บริการเสร็จแล้วแต่ยังมีผู้ใช้คนถัดไปรอรับบริการอยู่ ดังนั้นเหตุการณ์นี้ก็จะได้รับการตอบสนองเหมือนกัน โดยจะปรับค่าต่าง ๆ ดังนี้ ลดค่าตัวแปร

จำนวนผู้ใช้ในแถวลงไปหนึ่งค่า ซึ่งก็คือเปลี่ยนจาก 1 เป็น 0 แล้วจึงสร้างบันทึกเหตุการณ์ผู้ใช้ (คนที่เพิ่งเข้าไปใช้บริการ) ใช้บริการเสร็จเวลาของเหตุการณ์ = 216 + S4 = 229

ในตัวอย่างนี้แสดงการทำงานในแต่ละขั้นของโปรแกรมจำลองโดยละเอียด หากพิจารณาดูให้ดีจะพบว่าในทุก ๆ ครั้งที่มีการเปลี่ยนค่านาฬิกาโปรแกรมจะต้องถอดบันทึกเหตุการณ์ที่มีเวลาในการเกิดต่ำที่สุดจากรายการเหตุการณ์เพื่อนำมาดูว่ามีเหตุการณ์ใดเกิดขึ้น และจะมีเหตุการณ์ต่อเนื่องอะไรเกิดขึ้นเนื่องจากเหตุการณ์ที่เกิดขึ้นนี้ แล้วจึงสร้างเหตุการณ์ที่จะเกิดขึ้นใส่เข้าไปในรายการเหตุการณ์ ซึ่งโปรแกรมนี้มีเหตุการณ์อยู่ด้วยกัน 2 ชนิดคือ ผู้ใช้เข้ามาในระบบ และผู้ใช้ใช้บริการเสร็จ

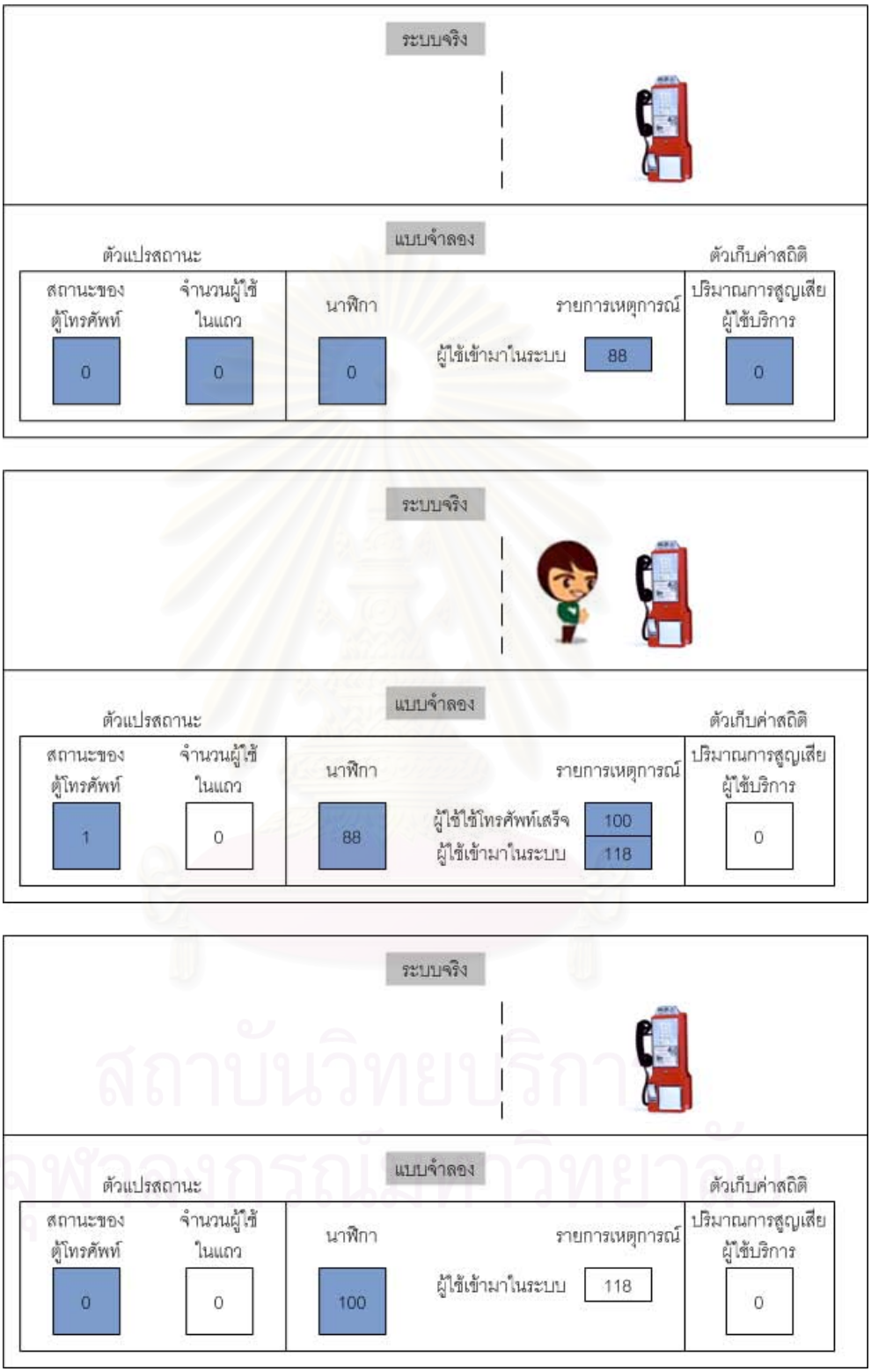
เนื่องจากในตัวอย่างนี้เป็นการจำลองระบบที่มีขนาดเล็ก ดังนั้นจึงสามารถใช้รายการเชื่อมโยงในการเก็บรายการเหตุการณ์ได้ แต่ถ้าขนาดของระบบที่เราจำลองมีขนาดใหญ่ขึ้น เช่น มีตู้โทรศัพท์เพิ่มเป็น 20 ตู้ ปริมาณบันทึกเหตุการณ์ที่ค้างอยู่ในรายการเหตุการณ์จะมีขนาดใหญ่ขึ้นเพราะต้องเก็บบันทึกเหตุการณ์ผู้ใช้ใช้บริการเสร็จของแต่ละตู้ ซึ่งแปลว่าในแต่ละรอบของการทำงานการค้นหาตำแหน่งที่เหมาะสมสำหรับใส่บันทึกเหตุการณ์ลงไปต้องทำงานนานขึ้น (จากมีได้สูงสุด 2 บันทึกเหตุการณ์เป็น 21 บันทึกเหตุการณ์) จากที่กล่าวมานี้แสดงให้เห็นถึงความสำคัญในการเลือกใช้โครงสร้างข้อมูล (data structure) ที่เหมาะสมสำหรับการเก็บรายการเหตุการณ์ในโปรแกรมจำลองเพราะโครงสร้างข้อมูลที่ดีจะทำให้จัดการงานในส่วนนี้ได้อย่างรวดเร็ว

โครงสร้างข้อมูลที่เรียกว่า แถวคอยแบบมีลำดับความสำคัญ (priority queue) มักจะถูกนำมาใช้เพื่อเก็บรายการเหตุการณ์ในโปรแกรมจำลอง เพราะโครงสร้างข้อมูลแบบนี้มีการทำงานสอดคล้องกับลักษณะการใช้งานรายการเหตุการณ์ในโปรแกรมจำลอง กล่าวคือ โครงสร้างข้อมูลแบบมีลำดับความสำคัญจะใช้ในการใส่สมาชิกเข้าไป (enqueue) หรือถอดสมาชิกออกมา (dequeue) จากกลุ่มข้อมูลคล้ายกับโครงสร้างข้อมูลแบบอื่น ๆ แต่ในการถอดสมาชิกออกมานั้นจะถอดสมาชิกออกมาตามลำดับความสำคัญ (priority) สมาชิกที่มีลำดับความสำคัญสูงสุดก็จะถูกดึงออกมาก่อน ซึ่งจะสอดคล้องกับการใช้งานรายการเหตุการณ์ เพราะในการใช้งานรายการเหตุการณ์ก็จะใส่บันทึกเหตุการณ์เข้าไปอย่างไม่สนใจลำดับ แต่ในการถอดบันทึกเหตุการณ์ออกมาจากรายการ จะถอดบันทึกเหตุการณ์ที่มีเวลาต่ำที่สุดออกมาก่อน ดังนั้นถ้าหากมองว่าเวลาก็คือค่าลำดับความสำคัญของแต่ละบันทึกเหตุการณ์ และเวลาที่น้อยกว่าเรียกว่ามีลำดับความสำคัญสูงกว่าก็จะเห็นความคล้ายคลึงกันได้อย่างชัดเจน

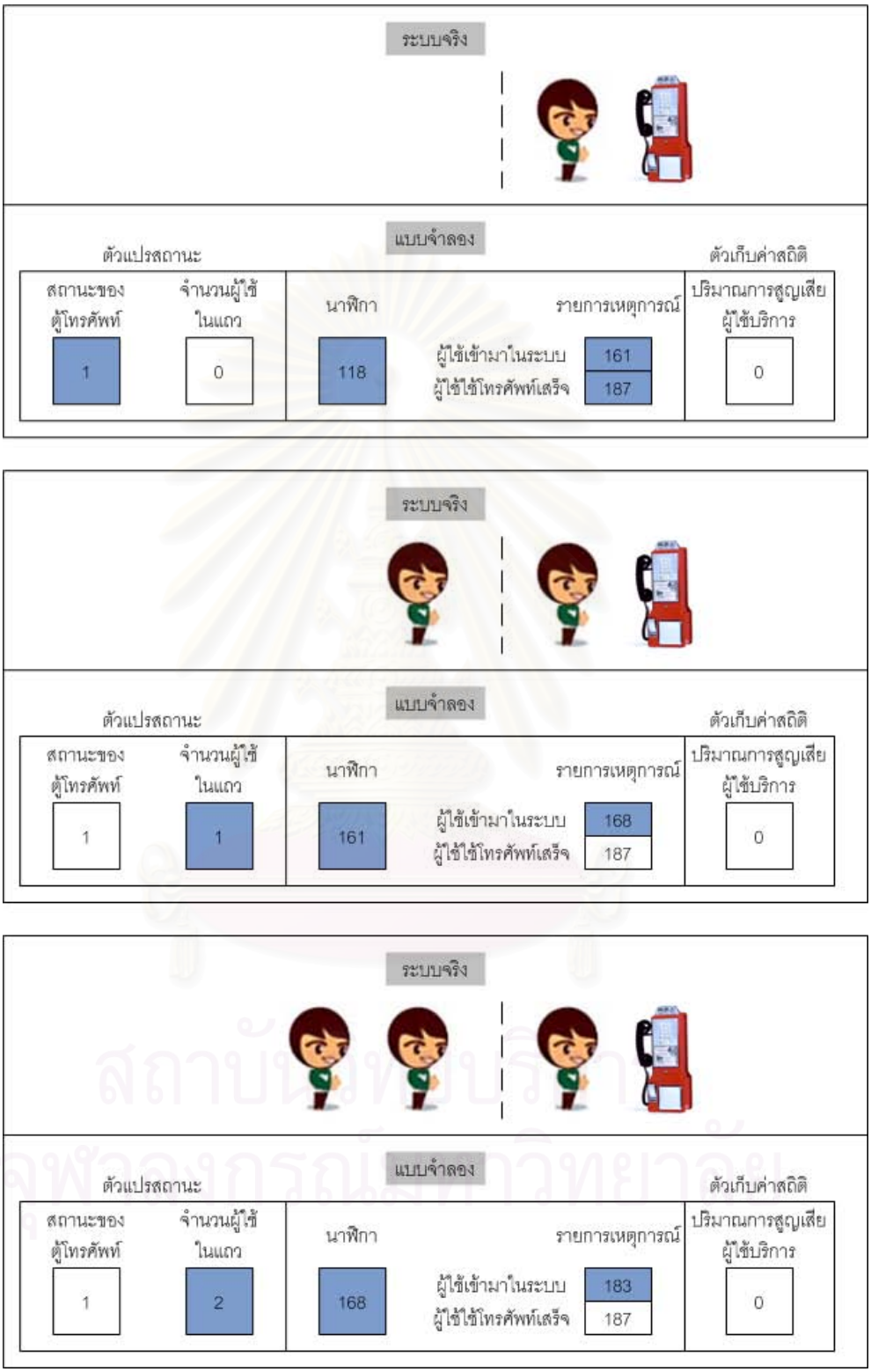
แถวคอยแบบมีลำดับความสำคัญที่มีอยู่ในปัจจุบันนั้นก็มีอยู่ด้วยกันอยู่หลายแบบ ซึ่งปัจจัยสำคัญในการเลือกใช้ แถวคอยแบบมีลำดับความสำคัญแต่ละแบบนั้นอยู่ที่ ความซับซ้อนในการใช้งานวิธีนั้น ๆ (ความยากง่ายในการเขียนโปรแกรม) และประสิทธิภาพ (ความเร็วและความทนทาน) ของวิธีนั้น ๆ

กล่าวคือ บางวิธีอาจสามารถเขียนโปรแกรมได้ง่ายแต่ก็มีประสิทธิภาพต่ำไม่สามารถใช้กับโปรแกรมที่ใช้จำลองระบบขนาดใหญ่ได้ บางวิธีแม้จะมีการใช้งานยุ่งยากกว่าแต่มีประสิทธิภาพดีสามารถนำไปใช้งานได้กับสถานการณ์ที่หลากหลายกว่า เมื่อเป็นเช่นนี้ในบางครั้งผู้ใช้อาจเลือกใช้วิธีที่ง่ายกว่าได้เพราะงานที่กำลังทำอยู่ไม่ได้ต้องการประสิทธิภาพในการทำงานสูงมากนัก แต่บางครั้งอาจจะต้องเลือกวิธีที่มีความยุ่งยากในการใช้งานสูงกว่า เพราะวิธีอื่น ๆ ไม่สามารถใช้งานได้เนื่องจากทำงานช้ากว่าหลายเท่า

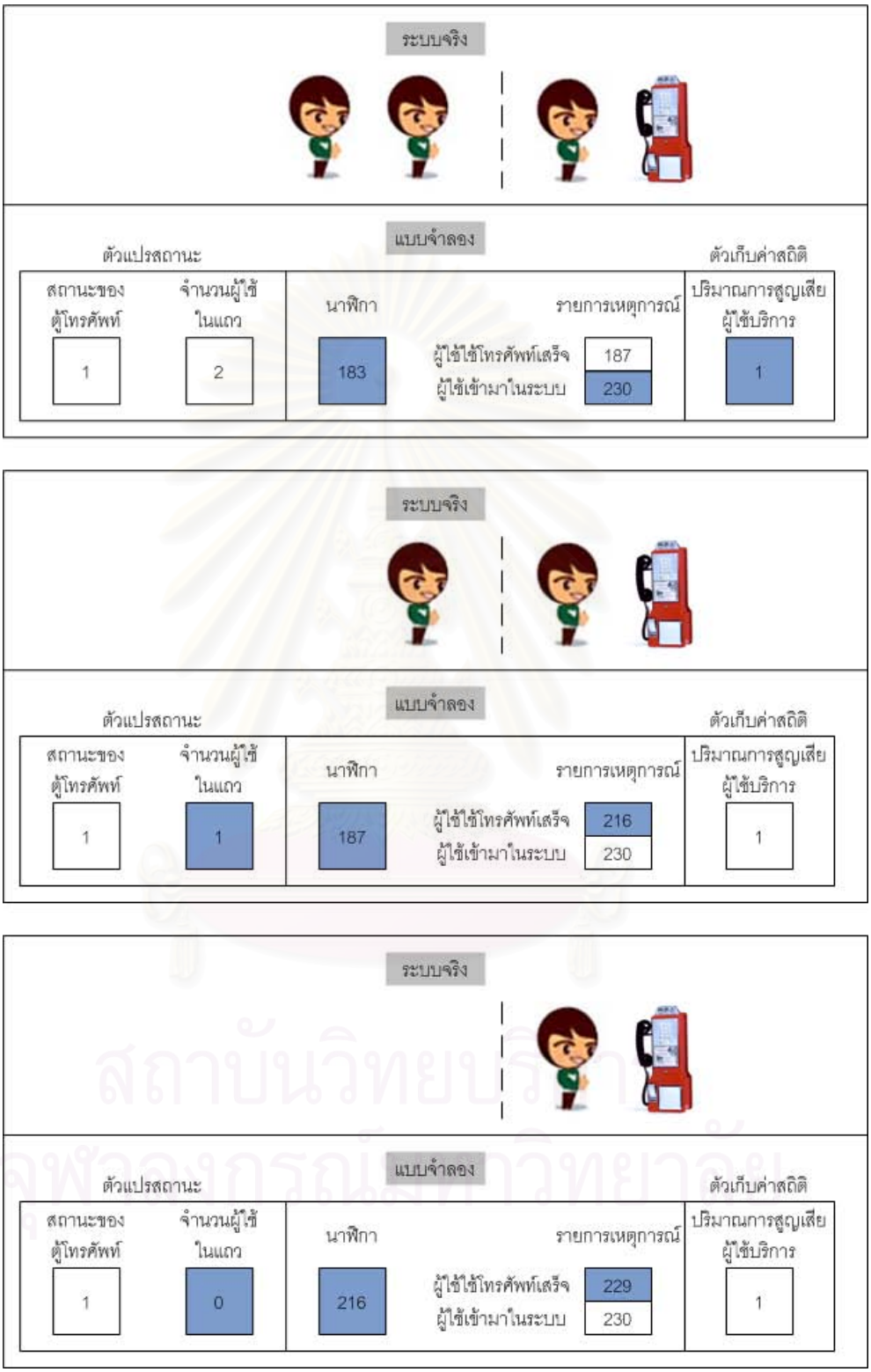
ในปัจจุบันแม้จะมีแถวคอยแบบมีลำดับความสำคัญให้เลือกใช้อย่างมากมาย แต่มีแถวคอยแบบมีลำดับความสำคัญชนิดหนึ่งได้รับความนิยมเป็นอย่างมาก เนื่องจากมีหลักการในการทำงานไม่ซับซ้อนเท่าไรนักและมีประสิทธิภาพในการทำงานสูงกว่าแถวคอยแบบมีลำดับความสำคัญชนิดอื่น แถวคอยแบบมีลำดับความสำคัญที่กล่าวถึงก็คือ แถวคอยแบบมีลำดับความสำคัญชนิดแถวคอยปฏิทิน (calendar queue) ซึ่งจะกล่าวถึงในหัวข้อที่ 2.3



รูปที่ 2.2 ภาพการทำงานของโปรแกรมจำลองระบบโทรศัพท์ ณ ค่านาฬิกาต่าง ๆ



รูปที่ 2.2 ภาพการทำงานของโปรแกรมจำลองระบบโทรศัพท์ ณ ค่านาฬิกาต่าง ๆ(ต่อ)



รูปที่ 2.2 ภาพการทำงานของโปรแกรมจำลองระบบโทรศัพท์ ณ ค่านาฬิกาต่าง ๆ(ต่อ)

2.2 การดำเนินการเหตุการณ์คงค่า

ดังได้อธิบายไปแล้วก่อนหน้านี้ว่าในทุกรอบการทำงานของโปรแกรมจำลองจะต้องเรียกกระบวนการย่อย 2 ชนิดขึ้นมาทำงาน คือ กระบวนการเวลา และกระบวนการเหตุการณ์ ซึ่งกระบวนการทั้ง 2 นี้เป็นกระบวนการที่มีการยุ่งเกี่ยวกับรายการเหตุการณ์ โดยจะมีการถอดบันทึกเหตุการณ์ออกมา (dequeue) ในกระบวนการเวลา และจะมีการใส่บันทึกเหตุการณ์เข้าไป (enqueue) ในกระบวนการเหตุการณ์ เหตุการณ์ที่ใส่เพิ่มเข้าไปนี้จะมีจำนวนกี่ใบขึ้นอยู่กับชนิดของเหตุการณ์ที่เกิดขึ้นด้วย (เหตุการณ์บางชนิดอาจจะไม่ใส่บันทึกเหตุการณ์เพิ่มเลยก็ได้) ซึ่งก็หมายความว่าระหว่างการทำงานของโปรแกรมจำลองจำนวนบันทึกเหตุการณ์ที่อยู่ในรายการเหตุการณ์จะไม่เป็นค่าคงที่ เพราะมีการถอดบันทึกเหตุการณ์ออกและใส่บันทึกเหตุการณ์เข้าไปในรายการอยู่ตลอดเวลา แต่ในโปรแกรมจำลองที่ใช้กันอยู่ทั่วไปเมื่อโปรแกรมทำงานจนอยู่ในสภาวะอยู่ตัว (steady state) จำนวนบันทึกเหตุการณ์ในรายการเหตุการณ์จะไม่มีการเปลี่ยนแปลง ซึ่งเกิดจากการที่จำนวนเหตุการณ์ลดลงเนื่องจากกระบวนการเวลา (ถอดบันทึกเหตุการณ์ออกมา 1 ใบเสมอ) มีค่าเท่ากับ ค่าเฉลี่ยจำนวนเหตุการณ์ที่เพิ่มขึ้นเนื่องจากกระบวนการเหตุการณ์ (มีค่าเป็น 1 เพราะมีจำนวนเหตุการณ์ที่ไม่ทำให้เกิดเหตุการณ์ต่อเนื่องเลย ใกล้เคียงกับจำนวนเหตุการณ์ที่ทำให้เกิดเหตุการณ์ต่อเนื่องหลายใบ) แต่จริง ๆแล้วก็ไม่ใช่ว่าจำนวนบันทึกเหตุการณ์ไม่มีการเปลี่ยนแปลง เพียงแต่การเปลี่ยนแปลงของค่าจำนวนบันทึกเหตุการณ์นั้นจะเกิดขึ้นในช่วงแคบ ๆ ซึ่งคล้ายกับว่าเป็นค่าคงที่เลยทีเดียว

การดำเนินการ เหตุการณ์คงค่า เป็น การ เลียน แบบ ลักษณะ การ ทำงาน ดังกล่าว มา ใช้ เพื่อ วัดประสิทธิภาพการทำงานของแถวคอยแบบมีลำดับความสำคัญ ขั้นตอนการทำงานของการดำเนินการเหตุการณ์คงค่าจะแบ่งเป็น 2 ขั้นตอนดังนี้

1. สร้างแถวคอยที่มีจำนวนบันทึกเหตุการณ์อยู่ N ใบ: ขั้นตอนนี้จะใส่บันทึกเหตุการณ์เข้าไปหรือถอดบันทึกเหตุการณ์ออกมาจากแถวคอยครั้งละ 1 ใบ โดยเราจะการกำหนดค่าความน่าจะเป็นที่มีค่าอยู่ในช่วง $(0.5, 1]$ ขึ้นมาค่าหนึ่ง เพื่อใช้ตัดสินใจเลือกว่าจะใส่หรือถอดบันทึกเหตุการณ์นั้น และเรียกค่าความน่าจะเป็นที่กำหนดขึ้นมานี้ว่า ค่าอัตราการสร้าง (construction rate: p) และถ้าต้องใส่บันทึกเหตุการณ์เข้าไปในรายการเหตุการณ์ จะกำหนดเวลาในการเกิดให้กับบันทึกเหตุการณ์ใหม่ที่ใส่เข้าไปดังสมการที่ (2.1) สังเกตได้ว่า p มีค่ามากกว่า 0.5 แปลว่าต้องมีการใส่เหตุการณ์เข้าไปมากกว่าการถอดเหตุการณ์ออกมา ดังนั้นหากทำกระบวนการนี้วนซ้ำไปเรื่อย ๆ ก็จะมีจำนวนบันทึกเหตุการณ์เป็น N ใบในที่สุด

$$t(e_e) = t(e_d) + \tau_i \tag{2.1}$$

โดยที่ $t(e_e)$ คือเวลาของเหตุการณ์ที่สร้างในกระบวนการเหตุการณ์
$t(e_d)$ คือเวลาของเหตุการณ์ที่ถูกถอดออกมาก่อนหน้านี้
τ_i คือค่าเวลาเพิ่ม (incremental time) ได้มาจากการสร้างค่าสุ่มที่มีการกระจาย f ค่าเฉลี่ย μ

2. ทำการดำเนินการเหตุการณ์คงค่าซ้ำเป็นจำนวน H รอบ: แต่ละรอบการทำงานจะถอดบันทึกเหตุการณ์ที่มีเวลาต่ำที่สุดออกมาจากแถว 1 ใบ และใส่บันทึกเหตุการณ์ใหม่เข้าไปในแถวอีก 1 ใบ (ใช้สมการที่ (2.1) ในการสร้างเวลาของเหตุการณ์เช่นกัน) เพื่อให้จำนวนบันทึกเหตุการณ์มีค่าคงที่เป็น N ใบตลอดเวลา โดยจะทำกระบวนการนี้ไปเป็นจำนวน H รอบ เพื่อจับเวลาที่ใช้ในการประมวลผลทั้ง H รอบ

การวัดประสิทธิภาพใช้ค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่า (ต่อ 1 บันทึกเหตุการณ์) มาเปรียบเทียบกัน (ค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่าหาได้โดย นำค่าเวลาที่ใช้ในการประมวลผลตลอดช่วงการทำการดำเนินการเหตุการณ์คงค่ามาหารด้วย $2H$) แถวคอยแบบมีลำดับความสำคัญชนิดใดมีค่าเฉลี่ยเวลาในการประมวลผลน้อยกว่าหมายความว่าเป็นวิธีที่มีประสิทธิภาพการทำงานดีกว่า

เห็นได้ว่ารายการเหตุการณ์จะมีจำนวนเหตุการณ์ไม่เปลี่ยนแปลงตลอดการทดสอบ (ในช่วงการทำการดำเนินการเหตุการณ์คงค่า) ซึ่งนี่ก็คือเหตุผลที่วิธีนี้เรียกว่า การดำเนินการเหตุการณ์คงค่า (hold เป็นคำภาษาอังกฤษแปลว่าคงค่าไว้ไม่เปลี่ยนแปลง) และการที่จำนวนบันทึกเหตุการณ์ในรายการเหตุการณ์มีค่าไม่เปลี่ยนแปลงนี่เอง คือลักษณะที่คล้ายกับการใช้งานรายการเหตุการณ์ในโปรแกรมจำลองจริง เป็นเหตุให้การดำเนินการเหตุการณ์คงค่าได้รับความนิยมและมีการใช้กันอย่างแพร่หลายในการทดสอบการทำงานของแถวคอยแบบมีลำดับความสำคัญแบบต่าง ๆ เช่นในรายงาน [2], [3], [4], [5], [6], [7], [8], [10]

2.3 แถวคอยปฏิทิน

แถวคอยแบบมีลำดับความสำคัญชนิดแถวคอยปฏิทิน (Calendar Queue: CQ) ได้ถูกเสนอขึ้นในปี พ.ศ. 2531 โดย นายแรนดี้ บราวน์ (Randy Brown) การทำงานของแถวคอยปฏิทิน ได้รับแรงบันดาลใจมาจากการใช้ปฏิทินเพื่อจดบันทึกกำหนดการในชีวิตประจำวัน ในทางกายภาพปฏิทินจะมีการเตรียมเนื้อที่กระดาษไว้สำหรับบันทึกกำหนดการในแต่ละวันอยู่แล้ว โดยปกติกำหนดการที่บันทึกลงในปฏิทิน จะประกอบไปด้วยข้อมูล 2 ส่วนด้วยกันคือ เวลาของกำหนดการ และรายละเอียดของกำหนดการ

รูปที่ 2.3 แสดงตัวอย่างการจดบันทึกกำหนดการลงในปฏิทิน ในรูปแสดงให้เห็นว่าแต่ละวันของปฏิทินมีตัวเลขกำกับไว้อย่างชัดเจน ดังนั้นหากผู้บันทึกต้องการบันทึกเหตุการณ์เพิ่มเข้าไปในปฏิทิน ขอเพียงทราบวันและเวลาของกำหนดการก็สามารถเลือกบันทึกกำหนดการไว้ในส่วนของวันนั้น ๆ ได้โดยสะดวก และในการอ่านกำหนดการก็จะอ่านเฉพาะกำหนดการที่กำลังจะมาถึงเท่านั้น ดังนั้นผู้ใช้ก็จะอ่านเรียงลำดับไปตามวันแต่ละวัน และลบกำหนดการที่ได้ทำไปแล้วออกจากบันทึก

รูปที่ 2.3 ตัวอย่างการบันทึกกำหนดการในหน้าปฏิทิน โดยแสดงเฉพาะวันที่ 1 - 5 เท่านั้น ตัวเลขด้านหน้าแสดงเวลาที่มีกำหนดการนั้น ๆ และ e1 - e8 คือรายละเอียดของกำหนดการ โดยเรียงลำดับจากการบันทึกกำหนดการเข้าไปในปฏิทิน

การบันทึกกำหนดการในรูปที่ 2.3 นั้นเป็นการบันทึกแบบไม่เรียงลำดับเวลา เพราะโดยปกติแล้วการบันทึกกำหนดการผู้บันทึกจะไม่ได้บันทึกตามลำดับเวลาของกำหนดการ แต่จะบันทึกตามลำดับการรับรู้กำหนดการของผู้บันทึกเอง ซึ่งผู้บันทึกมักจะไม่ได้รับรู้กำหนดการตามลำดับเวลา ลำดับ e1 ถึง e8 แสดงลำดับการบันทึกกำหนดการลงในปฏิทิน สังเกตว่าแต่ละวันเหตุการณ์ที่บันทึกไว้อาจไม่ได้เรียงลำดับตามเวลาที่เกิดขึ้น แต่จะเรียงตามลำดับการบันทึกดังได้กล่าวไว้แล้วข้างต้น ในการใช้งานเช่นนี้เป็นไปอย่างสมเหตุสมผลเพราะในแต่ละวันมีกำหนดการอยู่ไม่มากนัก ส่วนในการอ่านบันทึกกำหนดการผู้บันทึกสามารถอ่านกำหนดการทั้งหมดก่อนแล้วจึงเลือกเหตุการณ์ที่เกิดก่อนออกมา ซึ่งการบันทึกเช่นนี้ (ไม่เรียงลำดับ) สามารถทำได้ง่ายและรวดเร็วกว่าการเขียนแทรกกำหนดการลงไปตามลำดับเวลาเพราะไม่จำเป็นต้องเลื่อนกำหนดการเก่าออกไป (โดยการลบและเขียนใหม่) เพื่อให้เกิดที่ว่างในการแทรกกำหนดการใหม่เพิ่มเข้าไป แต่การบันทึกแบบนี้จะเหมาะสมกับการบันทึกในปฏิทินจริงเท่านั้น ส่วนการเขียนโปรแกรมจะมีลักษณะแตกต่างออกไป ดังจะอธิบายในส่วนต่อไป

นอกเหนือจากการใช้งานตามที่กล่าวมาแล้ว ปฏิทินเดียวกันนี้ยังสามารถใช้วางแผนกำหนดการของปีถัดไปได้อีกด้วย โดยการเพิ่มรายละเอียดของเวลาเข้าไป เช่น แทนที่จะเขียนแค่เวลาในการเกิดเหตุการณ์ก็ให้เขียนปีที่จะเกิดเหตุการณ์นั้นเพิ่มเข้าไป ก็จะทำให้สามารถใช้ปฏิทินนี้ซ้ำสำหรับปีถัด ๆ ไปได้ด้วย จาก

วิธีการบันทึกหมายกำหนดการในปฏิทินตามที่กล่าวมาแล้วข้างต้น หากพิจารณาให้ดีจะเห็นว่ามีลักษณะที่สามารถนำไปประยุกต์ใช้กับแถวคอยแบบมีลำดับความสำคัญได้ 2 ข้อคือ 1) การแบ่งบันทึกกำหนดการเป็นวัน ๆ ไป ไม่บันทึกรวมกันไว้ในที่เดียว 2) การใช้ปฏิทินเดิมเพื่อบันทึกกำหนดการของปีถัด ๆ ไป

แม้ว่าการทำงานของแถวคอยปฏิทินจะได้รับแรงบันดาลใจมาจากการใช้งานปฏิทินจริง แต่ว่าค่ากำหนดหลาย ๆ ค่าที่ใช้สำหรับสร้างปฏิทินจริง เช่น จำนวนวันในหนึ่งปี และระยะเวลาของหนึ่งวันนั้นไม่เหมาะสมที่จะนำไปประยุกต์ใช้กับรายการเหตุการณ์ หรือสามารถกล่าวได้อีกแบบหนึ่งว่ามาตราส่วนของเวลา (time scale) ของปฏิทินจริงนั้นไม่สามารถนำมาใช้กับรายการเหตุการณ์ได้ ดังนั้นจึงต้องมีการเปลี่ยนแปลงค่าต่าง ๆ (และชื่อเรียก เพื่อให้สอดคล้องกับลักษณะของค่านั้น ๆ) ที่ใช้ในปฏิทินจริงเพื่อความเหมาะสมในการทำงานของแถวคอยปฏิทิน

2.3.1 แบบจำลองของแถวคอยปฏิทิน

แถวคอยแบบมีลำดับความสำคัญนั้นจะมีการใช้งานอยู่ 2 รูปแบบด้วยกัน คือ การใส่บันทึกเหตุการณ์ (enqueue) และการถอดบันทึกเหตุการณ์ (dequeue) ซึ่งโดยปกติแล้วรายการขนาดเล็กจะสามารถทำงานทั้ง 2 อย่างนี้ได้รวดเร็วกว่ารายการที่มีขนาดใหญ่ ดังนั้นการออกแบบแถวคอยปฏิทินให้สามารถทำงานได้อย่างรวดเร็วแม้ว่ารายการเหตุการณ์จะมีขนาดใหญ่ก็ตาม สามารถทำได้โดยการแบ่งรายการขนาดใหญ่ให้เป็นรายการขนาดเล็กหลาย ๆ รายการ แต่ละรายการที่เก็บบันทึกเหตุการณ์อยู่นี้จะถูกเรียกว่า “ถัง” (bucket) ทำให้การเพิ่มบันทึกเหตุการณ์ หรือการถอดบันทึกเหตุการณ์สามารถทำได้ด้วยความเร็วเดียวกันกับเมื่อรายการมีขนาดเล็ก ซึ่งหากจะเปรียบเทียบกับการบันทึกกำหนดการในปฏิทิน ถังก็คือวันในปฏิทินนั่นเอง และถังนี่เองที่เป็นตัวกำหนดประสิทธิภาพการทำงานของแถวคอยปฏิทิน การออกแบบถังจะมีพารามิเตอร์ที่ใช้กำหนดลักษณะของถังอยู่ด้วยกัน 2 อย่างคือ

1. จำนวนถัง (number of buckets): หากเปรียบเทียบกับปฏิทินแล้ว จำนวนถังก็คือจำนวนวันในหนึ่งปีนั่นเอง จำนวนถังที่ใช้จะบอกถึงความสามารถในการกระจายบันทึกเหตุการณ์ของแถวคอยปฏิทิน ความสามารถในการกระจายบันทึกเหตุการณ์นี้ดูจาก อัตราส่วนของจำนวนบันทึกเหตุการณ์ต่อจำนวนถัง ถ้าอัตราส่วนนี้มีค่ามากหมายความว่าถังแต่ละใบจะต้องรองรับบันทึกเหตุการณ์เป็นจำนวนมากซึ่งอาจเกิดแถวคอยขนาดใหญ่ขึ้นในถังบางใบหรือทุกใบได้ แต่ถ้าอัตราส่วนนี้มีค่าน้อยก็จะหมายความว่าถังแต่ละใบไม่ต้องรองรับบันทึกเหตุการณ์จำนวนมาก ซึ่งก็คือสามารถกระจายรายการขนาดใหญ่เป็นรายการขนาดเล็กนั่นเอง (จริง ๆ แล้วอาจจะเกิดรายการขนาดใหญ่ได้ถ้ากำหนดค่าความกว้างถังไม่ดี ดังจะได้กล่าวต่อไป) ดังนั้นเราจึงต้องการให้ค่าอัตราส่วนของจำนวนบันทึกเหตุการณ์ต่อจำนวนถัง (หรืออีกนัยหนึ่งคือจำนวนบันทึก

เหตุการณ์เฉลี่ยในแต่ละถัง) มีค่าน้อยที่สุดเท่าที่จะเป็นไปได้ ซึ่งก็คือมีจำนวนถังยิ่งมากยิ่งดี แต่ในความเป็นจริงแล้วถังแต่ละใบที่สร้างขึ้นมาในโปรแกรมนั้นต้องใช้หน่วยความจำของเครื่องคอมพิวเตอร์ ดังนั้นเราจึงต้องมีการควบคุมจำนวนถังให้เหมาะสมกับความจำเป็นในการใช้งาน เพื่อไม่ให้หน่วยความจำที่มีอยู่ถูกใช้มากเกินความจำเป็น ใน [8] จะกำหนดให้มีอัตราส่วนของจำนวนบันทึกเหตุการณ์ต่อจำนวนถังอยู่ระหว่าง 0.5 ถึง 2 แต่ในขณะที่เริ่มใช้งาน (ยังไม่มีบันทึกเหตุการณ์อยู่เลย) แถวคอยปฏิทินจะกำหนดให้มีจำนวนถังเป็น 2 แล้วจึงค่อย ๆ มีการปรับเพิ่มขึ้นหรือลดลงตามจำนวนเหตุการณ์ที่อยู่ในแถวคอยปฏิทิน จำนวนถังในแถวคอยปฏิทินจะถูกปรับค่าให้เพิ่มขึ้นก็ต่อเมื่อได้เพิ่มบันทึกเหตุการณ์เข้าไปแล้วมีผลให้อัตราส่วนของจำนวนบันทึกเหตุการณ์ต่อจำนวนถังมีค่าเป็น 2 และในทางกลับกันจะถูกปรับลดจำนวนถังเมื่อถอดบันทึกเหตุการณ์ออกมาแล้วอัตราส่วนของจำนวนบันทึกเหตุการณ์ต่อจำนวนถังมีค่าเป็น 0.5 การปรับเพิ่มหรือลดนี้จะปรับให้แถวคอยปฏิทินมีจำนวนถังเท่ากับจำนวนบันทึกเหตุการณ์ ซึ่งก็คือการทำให้อัตราส่วนของจำนวนบันทึกเหตุการณ์ต่อจำนวนถังมีค่าเป็น 1 นั่นเอง หรือสามารถกล่าวอีกนัยหนึ่งว่าจำนวนถังเพิ่มขึ้นหรือลดลงครั้งละ 2 เท่า และเนื่องจากจำนวนถังเริ่มต้นจาก 2 ดังนั้นจำนวนถังที่เป็นไปได้ก็คือ 2, 4, 8, 16, 32, 64, 128, . . .

2. ความกว้างถัง (bucket width): หากเปรียบเทียบกับวันในปฏิทินแล้ว ความกว้างถังก็คือเวลาของแต่ละวัน (วันหนึ่งมี 24 ชั่วโมง) ความกว้างถังนี้เป็นตัวแปรที่มีอิทธิพลอย่างมาก และต้องถูกเลือกอย่างระมัดระวังมากกว่าค่าจำนวนถัง เพราะจำนวนถังอาจจะเลือกให้มีค่ามากไว้ก่อนได้ซึ่งก็จะมีแต่ทำให้สามารถทำงานได้ดีขึ้น (ตราบใดที่หน่วยความจำยังไม่หมด) แต่การเลือกค่าความกว้างถังนี้จะมากไปหรือน้อยไปไม่ได้เลยเพราะจะทำให้ทำงานได้ช้าลง อาจกล่าวได้ว่าแม้แถวคอยปฏิทินจะมีถังอยู่เป็นจำนวนมากก็ตาม แต่หากถังแต่ละใบไม่ได้ถูกใช้งานอย่างมีประสิทธิภาพ แถวคอยปฏิทินก็จะไม่สามารถทำงานได้ดี ความกว้างถังนี่เองที่เป็นตัวควบคุมประสิทธิภาพการใช้งานถังให้กับแถวคอยปฏิทิน และเพื่อให้สอดคล้องกับแนวคิดในการกำหนดจำนวนถังคือให้มีบันทึกเหตุการณ์เฉลี่ยอยู่ระหว่าง 0.5 ถึง 2 เหตุการณ์ต่อหนึ่งถัง ดังนั้นจึงเลือกใช้ค่าเฉลี่ยเวลาระหว่างบันทึกเหตุการณ์มาเป็นค่าความกว้างของถัง เพราะจะทำให้มีบันทึกเหตุการณ์กระจายไปยังทุก ๆ ถังและไม่เกิดการกระจุกตัวของบันทึกเหตุการณ์ในถังใดถังหนึ่ง (ถ้ามีบันทึกเหตุการณ์อยู่ในถังที่ 1 แล้วก็น่าจะมีบันทึกเหตุการณ์อีกใบอยู่ในถังที่ 2 ด้วยเพราะถังแต่ละใบมีความกว้างเท่ากับค่าเฉลี่ยระยะเวลาระหว่างเหตุการณ์) แต่ก็คล้าย ๆ กับการกำหนดค่าจำนวนถัง ในขณะที่เริ่มใช้งาน (ยังไม่มีบันทึกเหตุการณ์อยู่เลย) จะกำหนดให้มีความกว้างถังเป็น 1 แล้วจึงเปลี่ยนค่าตามค่าเฉลี่ยของเวลาระหว่างบันทึกเหตุการณ์ การหาค่าเฉลี่ยนี้ทำได้โดยถอดบันทึกเหตุการณ์ออกมาจากแถวคอยปฏิทิน แล้วนำไปคำนวณหาค่าเฉลี่ยของเวลาระหว่างเหตุการณ์ หลังจากคำนวณเสร็จแล้วก็จะใส่

บันทึกเหตุการณ์ที่ถอดออกมากลับไปในแถวคอยปฏิทินตามเดิม ตามที่กำหนดไว้ใน [8] จำนวนบันทึกเหตุการณ์ที่ใช้ในการหาค่าเฉลี่ยนี้จะอยู่ระหว่าง 5 ถึง 25 ใบ ขึ้นอยู่กับจำนวนบันทึกเหตุการณ์ทั้งหมดที่มีอยู่ในแถวคอยปฏิทิน

ดังได้กล่าวไปแล้วว่าการเพิ่มหรือถอดบันทึกเหตุการณ์ในแถวคอยปฏิทินนั้นจำเป็นต้องทราบก่อนว่าจะเพิ่มบันทึกเหตุการณ์เข้าไปในถังใด (หรือถอดบันทึกเหตุการณ์ออกมาจากถังใด) โดยการกำหนดช่วงเวลาให้กับถังแต่ละใบก่อน และบันทึกเหตุการณ์ที่มีเวลาในการเกิดอยู่ในช่วงเวลาของถังใดก็จะถูกเก็บไว้ในถังนั้น ช่วงเวลาที่กำหนดให้กับถังแต่ละใบนี้เรียกว่าค่าช่วงตอบรับ (admitable range) ซึ่งสามารถหาค่าช่วงตอบรับของถังที่ m จากจำนวนถังและความกว้างถังได้ดังนี้

$$(yM + m)\delta, (yM + m + 1)\delta)$$

โดยที่ y คือรอบการใช้งานปฏิทินซึ่งจะมีค่าเป็น $0, 1, 2, 3, \ldots$

m คือเลขถัง; $m \in \{0, 1, 2, 3, \ldots, M - 1\}$

M จำนวนถัง

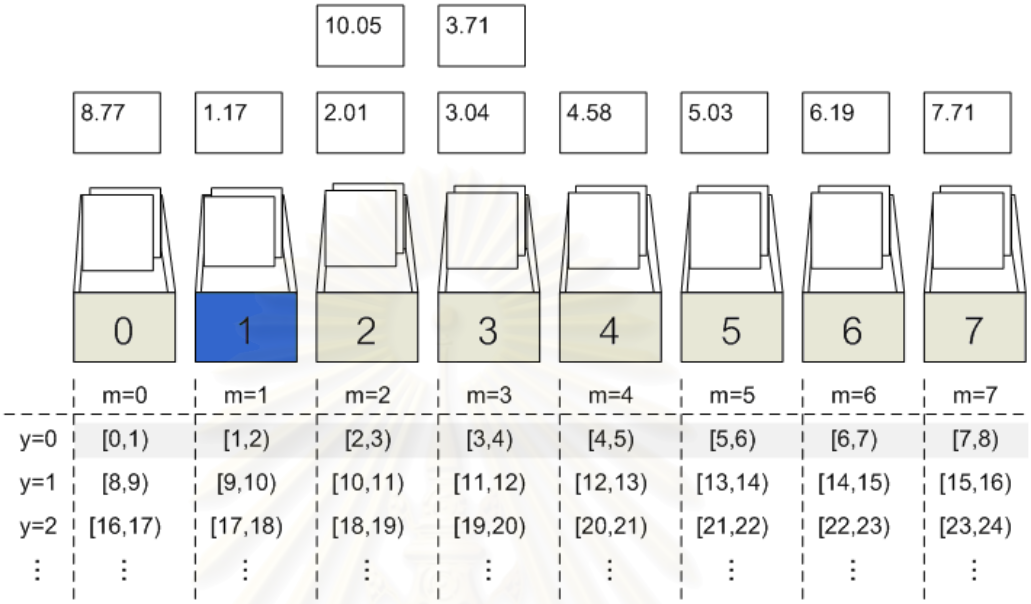
δ ความกว้างถัง

นอกจากนี้ยังมีอีกสิ่งหนึ่งที่ต้องกำหนดให้กับแถวคอยปฏิทิน ซึ่งก็คือแถวคอยแบบมีลำดับความสำคัญที่จะนำมาใช้จัดการรายการย่อยในถังแต่ละใบนั่นเอง แถวคอยแบบมีลำดับความสำคัญที่เลือกมานี้ไม่ได้มีผลกระทบต่อการทำงานของแถวคอยปฏิทินเท่าไรนัก เพราะรายการย่อยนี้มีขนาดเล็ก ดังนั้นวิธีที่สามารถเขียนโปรแกรมได้ง่ายเช่น รายการเชิงเส้น[1] (linear list) หรือรายการไร้ลำดับ[2] (unordered list) จึงมักจะถูกนำมาใช้

รูปที่ 2.4 แสดงแถวคอยปฏิทินที่มีความกว้างถัง 1 หน่วยเวลา มีจำนวนถัง 8 ถัง ใช้รายการเชิงเส้นในการจัดการถังแต่ละใบ และมีบันทึกเหตุการณ์อยู่ในแถวคอยปฏิทินทั้งหมด 10 ใบด้วยกัน ในส่วนบน

[1]มีการเชื่อมต่อบันทึกเหตุการณ์แต่ละใบด้วยโครงสร้างข้อมูลแบบ รายการเชื่อมโยง (linked list) เพื่อความสะดวกในการใส่/ถอดบันทึกเหตุการณ์ ในการใส่บันทึกเหตุการณ์จะแทรกบันทึกเหตุการณ์เข้าไปโดยเรียงตามลำดับเวลาการเกิด ในการถอดบันทึกเหตุการณ์ออกมาจะถอดบันทึกเหตุการณ์ที่อยู่ต้นรายการออกมาเพราะมีการจัดเรียงบันทึกเหตุการณ์ตามเวลาอยู่แล้ว

[2]มีการเชื่อมต่อบันทึกเหตุการณ์แต่ละใบด้วยโครงสร้างข้อมูลแบบ รายการเชื่อมโยง (linked list) เช่นเดียวกับวิธี รายการเชิงเส้น แต่การใส่บันทึกเหตุการณ์นั้นจะแทรกเหตุการณ์ลงไปในตำแหน่งสุดท้ายของรายการ ในการถอดบันทึกเหตุการณ์ออกมาจะต้องค้นหาบันทึกเหตุการณ์ที่มีเวลาในการเกิดต่ำที่สุดจากในรายการก่อนจึงค่อยถอดเหตุการณ์นั้นออกมาจากรายการ ทำให้เสียเวลาในการถอดบันทึกเหตุการณ์มากกว่า แต่เสียเวลาในการใส่บันทึกเหตุการณ์น้อยกว่ารายการเชิงเส้น



รูปที่ 2.4 แบบจำลองของแถวคอยปฏิทิน $\delta = 1,\ M = 8$

ตารางที่ 2.1 ตัวอย่างการคำนวณค่าช่วงตอบรับ

y	m	M	δ	ช่วงตอบรับ
0	0	8	1	$[0, 1)$
0	1	8	1	$[1, 2)$
1	1	8	1	$[9, 10)$
2	4	8	1	$[20, 21)$
2	7	8	1	$[23, 24)$

ของถังจะแสดงบันทึกรายการที่ถูกเก็บอยู่ในแต่ละถังเรียงลำดับเวลาการเกิดจากน้อยไปหามาก (บันทึกเหตุการณ์ที่เกิดก่อนอยู่ด้านล่าง และใบบันทึกเหตุการณ์แต่ละใบที่แสดงให้เห็นนั้นจะมีแต่ส่วนของเวลาในการเกิดเหตุการณ์ โดยไม่แสดงส่วนของรายละเอียดเหตุการณ์เพราะไม่ได้มีความหมายต่อการอธิบายขั้นตอนการทำงานของแถวคอยปฏิทิน) ในส่วนล่างของถังแสดงช่วงตอบรับของถังแต่ละใบ ตัวอย่างการคำนวณค่าช่วงตอบรับได้แสดงไว้ในตารางที่ 2.1

แบบจำลองของแถวคอยปฏิทินในรูปที่ 2.4 นี้แสดงให้เห็นอย่างชัดเจนแล้วว่าแถวคอยปฏิทินจะกระจายบันทึกเหตุการณ์จากรายการขนาดใหญ่ให้กลายเป็นรายการขนาดเล็กซึ่งใช้เวลาประมวลผลน้อยกว่า และนี่เองคือจุดเด่นข้อแรกของปฏิทินที่ถูกนำมาใช้ในแถวคอยปฏิทิน ส่วนจุดเด่นอีกข้อหนึ่งที่ถูกนำมาใช้คือการใช้ปฏิทินซ้ำ ในรูปที่ 2.4 บันทึกเหตุการณ์ 10.5 ถูกเก็บไว้ในถังใบที่ 2 ทั้ง ๆ ที่ไม่ใช่เหตุการณ์ในรอบการทำงานนี้ การที่สามารถทำแบบนี้ได้นั้นเป็นผลมาจากการกำหนดช่วงตอบรับ ให้สามารถใช้ปฏิทินซ้ำได้หลาย ๆ รอบ (มีค่า y หลายค่า)

2.3.2 การใส่บันทึกเหตุการณ์เข้าไปในแถวคอยปฏิทิน

การใส่เหตุการณ์เข้าไปในแถวคอยปฏิทิน (enqueue) นั้นก็สามารถทำได้ โดยการหาถังที่มีช่วงตอบรับสอดคล้องกับเวลาในการเกิดเหตุการณ์ที่ต้องการใส่เข้าไปก่อน แต่ในความเป็นจริงแล้วจะไม่ใช้ช่วงตอบรับในการหาถังที่เหมาะสม เพราะค่าช่วงตอบรับนั้นถูกนำเสนอขึ้นมาในที่นี้เพื่อช่วยให้สามารถอธิบายแบบจำลองของแถวคอยปฏิทินได้ง่ายขึ้นเท่านั้นไม่ได้ถูกนำไปใช้ในการเขียนโปรแกรมแต่อย่างใด ค่าเลขถังที่ใช้เก็บบันทึกเหตุการณ์สามารถหาได้จากสมการที่ (2.2)

$$m(e) = \left\lfloor \frac{t(e)}{\delta} \right\rfloor \mod M \tag{2.2}$$

โดยที่ e คือเหตุการณ์ที่จะนำมาบันทึก
$m(e)$ คือถังที่มีช่วงตอบรับสอดคล้องกับเหตุการณ์ e
$t(e)$ คือเวลาในการเกิดเหตุการณ์ e

2.3.3 การถอดบันทึกเหตุการณ์ออกจากแถวคอยปฏิทิน

ในการถอดบันทึกเหตุการณ์ออกจากแถวคอยปฏิทิน (dequeue) จะใช้ตัวแปรเพื่อควบคุมการทำงานในส่วนนี้ 2 ตัวด้วยกัน คือ

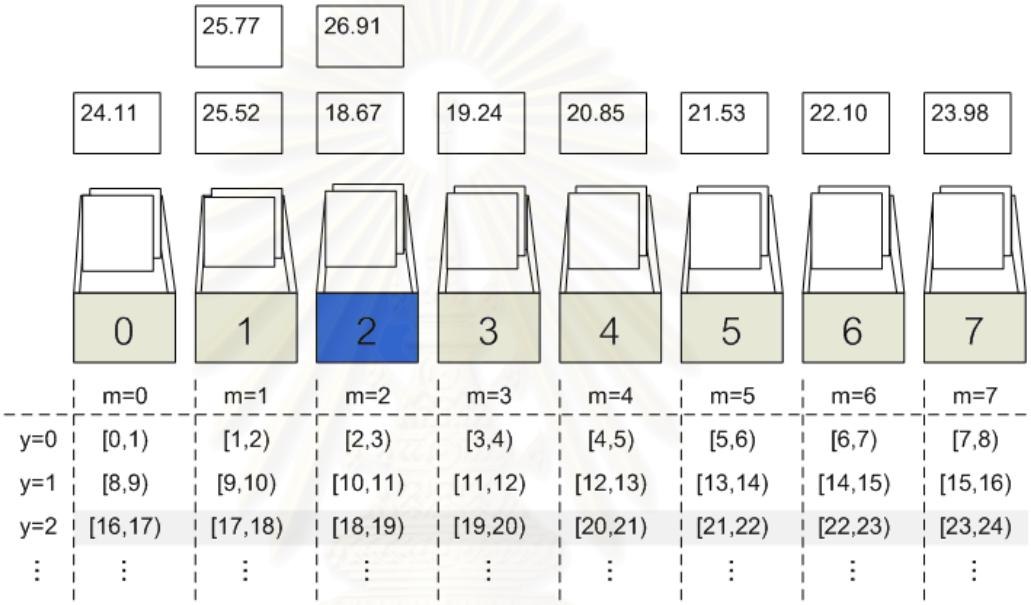
1. ถังสุดท้าย (last bucket) ค่าถังสุดท้ายนี้ไม่ใช่ค่า $M-1$ ซึ่งเป็นเลขถังใบสุดท้ายของแถวคอยปฏิทิน แต่เป็นตัวแปรอีกตัวหนึ่งที่เอาไว้เก็บค่าถังสุดท้ายที่มีการค้นหาบันทึกรายการเกิดขึ้น ตัวแปรนี้จะถูกปรับค่าไปเรื่อย ๆ ในขณะที่ทำการค้นหาบันทึกเหตุการณ์ที่มีเวลาน้อยที่สุด และจะถูกตั้งค่าเป็น 0 (ถังใบแรก) ในตอนเริ่มต้นของโปรแกรม (นาฬิกามีค่าเป็น 0)

2. ขอบบนของถัง (bucket top) เป็นค่าขอบบนของถังสุดท้าย แต่ต้องมีการพิจารณารอบการใช้งานด้วย เช่น (จากรูปที่ 2.4) ค่าขอบบนของถังที่ 3 ในรอบการใช้งานที่ 0 จะมีค่าขอบบนของถังเป็น 4 แม้จะดูคล้ายกับว่าค่าขอบบนของถังนี้ได้มาจากค่าช่วงตอบรับ แต่ในความเป็นจริงแล้วไม่ได้ใช้ค่าช่วงตอบรับในการหาค่าขอบบนของถัง แต่จะค่อย ๆ เพิ่มค่าขอบบนของถังไปพร้อม ๆ กับการเลื่อนถังแต่ละครั้ง เมื่อโปรแกรมเริ่มทำงานตัวแปรนี้จะถูกตั้งค่าเป็น 1 ซึ่งเป็นค่าขอบบนของถังที่ 0 รอบการทำงานที่ 0

การค้นหาบันทึกเหตุการณ์ที่จะถอดออกมาจะเริ่มต้นค้นหาบันทึกเหตุการณ์จากถังสุดท้ายก่อน โดยจะนำเวลาของบันทึกเหตุการณ์แรกในถังสุดท้ายมาเปรียบเทียบกับค่าขอบบนของถังเพื่อทดสอบว่าบันทึกเหตุการณ์ใบนั้น ๆ เป็นบันทึกเหตุการณ์ที่เกิดขึ้นในรอบการทำงานนี้จริง ๆ ไม่ใช่บันทึกเหตุการณ์ของรอบการทำงานถัด ๆ ไป ซึ่งถ้าเวลาของบันทึกเหตุการณ์มีค่าน้อยกว่าค่าขอบบนของถังก็หมายความว่าเป็นบันทึกเหตุการณ์ที่เกิดขึ้นในรอบการทำงานนี้ จึงถอดบันทึกเหตุการณ์นี้ออกมาและกำหนดค่าถังสุดท้ายให้เท่ากับเลขถังใบนี้ แต่ถ้าไม่ใช่หรือว่าไม่มีเหตุการณ์อยู่ในถังนั้น ๆ ก็จะเลื่อนค่าขอบบนของถังไปยังถังถัดไปแล้วนำรายการเหตุการณ์แรกของถังใบนั้นออกมาเปรียบเทียบอีกครั้ง และจะทำเช่นนี้ต่อไป จนกว่าจะพบบันทึกเหตุการณ์ที่จะถอดออกมา

แม้ว่าในกรณีปกติบันทึกเหตุการณ์ในแถวคอยปฏิทินจะมีการกระจายอยู่ทั่วทุกถัง และมีเวลาเฉลี่ยระหว่างเหตุการณ์ประมาณ 1 ถัง ซึ่งทำให้มีโอกาสเจอเหตุการณ์ที่จะถอดออกมาในถังถัดไปสูง แต่ในบางกรณีก็ยังคงมีปัญหาที่อาจเกิดขึ้นได้ นั่นก็คือมีบันทึกเหตุการณ์กระจุกตัวอยู่ในหลายช่วงเวลา หรือเรียกอีกอย่างว่าเหตุการณ์มีการกระโดด ตัวอย่างเช่น รายการเหตุการณ์หนึ่งมีบันทึกเหตุการณ์อยู่ทั้งหมด 1,000 ใบ มี 500 ใบเกิดขึ้นในช่วงเวลา $[101, 200]$ และมีอีก 500 ใบเกิดขึ้นในช่วงเวลา $[1101, 1201]$ ค่าเฉลี่ยของเวลาระหว่างเหตุการณ์ของทั้ง 2 ช่วงนี้มีค่าเป็น 0.2 สถานการณ์นี้จะทำให้เกิดแถวคอยปฏิทินที่มีจำนวนถังเป็น 512 ถัง และมีความกว้างถังเป็น 0.2 ในการค้นหาบันทึกเหตุการณ์ 500 ใบแรกนั้นจะสามารถทำได้อย่างรวดเร็ว แต่หลังจากหมดบันทึกเหตุการณ์ 500 ใบแรกแล้วจะต้องทำการค้นปฏิทินไปอีกประมาณ 9 รอบ (เนื่องจากมีระยะห่างระหว่างเวลาของบันทึกเหตุการณ์เป็น 900 แต่ละรอบของปฏิทินมีเวลา $512 \times 0.2 = 102.4$ นับเป็นระยะห่าง $900/102.4 = 8.79$ รอบ) จึงจะพบบันทึกเหตุการณ์ใบถัดไป ในแต่ละรอบของปฏิทินจะต้องค้นหาทั้งหมด 512 ครั้ง สรุปแล้วต้องทำการเปรียบเทียบทั้งหมด

ประมาณ $9 \times 512 = 4608$ ครั้ง จึงพบบันทึกเหตุการณ์ที่เกิดในช่วงที่สอง ดังนั้นเมื่อทำการค้นหาครบหนึ่งรอบ (ครบทุกถัง) แล้วแต่ยังไม่พบเหตุการณ์ที่จะถอดออกมา จะตั้งสมมติฐานว่ามีการกระโดดของเหตุการณ์เกิดขึ้น จึงแก้ปัญหานี้โดยทำการค้นหาบันทึกเหตุการณ์ที่มีเวลาในการเกิดน้อยที่สุดจากทุก ๆ ถัง แล้วนำมาหาบันทึกที่มีเวลาในการเกิดน้อยที่สุดแล้วถอดเหตุการณ์นั้นออกมาพร้อมทั้งเลื่อนค่าถังล่าสุด และขอบบนของถังให้สอดคล้องกับเวลาของเหตุการณ์นั้น กระบวนการที่เกิดขึ้นนี้เรียกว่าการค้นหาโดยตรง (direct search)



รูปที่ 2.5 แบบจำลองของแถวคอยปฏิทิน $\delta = 1$, $M = 8$

ในส่วนนี้จะแสดงตัวอย่างการถอดบันทึกเหตุการณ์ออกมาจากแถวคอยปฏิทิน โดยจะแสดงทั้งกรณีปกติ และกรณีที่เกิดการค้นหาโดยตรง

- ตัวอย่างการถอดบันทึกเหตุการณ์ออกจากแถวคอยปฏิทิน: จากแถวคอยปฏิทินในรูปที่ 2.4 จะเริ่มต้นจากการหาค่าถังสุดท้ายซึ่งในตอนนี้สมมติว่ายังไม่มีเหตุการณ์ใดถูกถอดออกมาเลย ค่าถังสุดท้ายจึงยังมีค่าเป็น 0 และค่าขอบบนของถังยังมีค่าเป็น 1 อยู่ โปรแกรมจะนำเวลาของบันทึกเหตุการณ์แรกในถังที่ 0 มาเปรียบเทียบกับค่า 1 ผลการเปรียบเทียบพบว่า 8.77 มีค่ามากกว่า 1 จึงเลื่อนค่าขอบบนของถังไปยังถังถัดไปเพื่อค้นหาต่อซึ่งจะเลื่อนค่าขอบบนของถังจาก 1 เป็น $1+\delta = 1+1 = 2$ ในการเปรียบเทียบครั้งนี้จะใช้ 1.17 เปรียบเทียบกับ 2 ซึ่งผลที่ได้คือ 1.17 มีค่าน้อยกว่า 2 เราจึงถอดบันทึกเหตุการณ์นี้ออกมาและปรับค่าถังสุดท้ายให้เป็น 1 ส่วนค่าขอบบนของถังให้มีค่าเป็น 2 ตามเดิม

- ตัวอย่างการถอดบันทึกเหตุการณ์ออกจากแถวคอยปฏิทินแบบใช้การค้นหาโดยตรง: ตัวอย่างนี้แสดงไว้ในรูปที่ 2.5 สมมติว่ายังไม่มีเหตุการณ์ใดถูกถอดออกมาเลย ค่าถังสุดท้ายจึงยังมีค่าเป็น 0 และค่าขอบบนของถังยังมีค่าเป็น 1 อยู่ โปรแกรมจะนำเวลาของบันทึกเหตุการณ์แรกในถังที่ 0 มาเปรียบเทียบกับค่า 1 ผลการเปรียบเทียบพบว่า 24.11 มีค่ามากกว่า 1 จึงเลื่อนค่าขอบบนของถังไปยังถังถัดไปเพื่อค้นหาต่อซึ่งผลการค้นหาก็จะเป็นเหมือนเดิมไปจนกระทั่งค้นหาไปถึงถังที่ 7 แล้วพบว่าบันทึกเหตุการณ์ 23.98 มีค่ามากกว่า 8 ก็จะเลื่อนไปยังถังถัดไปซึ่งก็คือการกลับมาค้นหาที่ถัง 0 อีกครั้ง ซึ่งจะทำให้เกิดการค้นหาโดยตรง โดยจะนำบันทึกเหตุการณ์ใบแรกของทุก ๆ ถังมาเทียบเวลากัน จากการเปรียบเทียบพบว่าบันทึกเหตุการณ์ที่มีเวลาเกิดน้อยที่สุดคือ 18.67 ที่อยู่ในถังที่ 2 ดังนั้นโปรแกรมจะเลื่อนค่าถังสุดท้ายให้เป็น 2 และขอบบนของถังเป็น 19 เพื่อให้สอดคล้องกับบันทึกเหตุการณ์ 18.67 หลังจากนั้นก็กลับไปทำกระบวนการเปรียบเทียบต่อ ในครั้งนี้การเปรียบเทียบจะให้ผลลัพธ์ว่า 18.67 มีค่าน้อยกว่า 19 แล้วจึงถอดเหตุการณ์นั้นออกเป็นการสิ้นสุดกระบวนการ

2.3.4 การเปลี่ยนขนาดของแถวคอยปฏิทิน

การเปลี่ยนขนาดของแถวคอยปฏิทินเป็นกระบวนการย่อยในแถวคอยปฏิทินซึ่งทำหน้าที่ปรับค่าจำนวนถังและความกว้างถังของแถวคอยปฏิทิน โดยจะสร้างปฏิทินใหม่ที่มีจำนวนและความกว้างถังตามที่ต้องการ แล้วจึงย้ายบันทึกเหตุการณ์ทั้งหมดจากปฏิทินเก่าไปไว้ในปฏิทินใหม่ กระบวนการนี้จะเกิดขึ้นเมื่อมีการเพิ่มหรือถอดบันทึกออกจากแถวคอยปฏิทินแล้วทำให้ค่าบันทึกเหตุการณ์เฉลี่ย (ต่อถัง) มีค่าเป็น 0.5 หรือ 2 แถวคอยปฏิทินจะมีค่าบันทึกเหตุการณ์เฉลี่ยเป็น 0.5 ได้เมื่อถูกถอดบันทึกเหตุการณ์ออกไปเท่านั้น และจะมีค่าบันทึกเหตุการณ์เฉลี่ยสูงกว่า 2 เมื่อถูกเพิ่มบันทึกเหตุการณ์เข้าไปเท่านั้น (แต่ส่วนใหญ่การเพิ่มและถอดบันทึกเหตุการณ์จะทำให้ค่าบันทึกเหตุการณ์เฉลี่ยอยู่ระหว่าง 0.5 ถึง 2 ดังนั้นกระบวนการนี้จึงไม่เกิดขึ้น)

กระบวนการนี้มีไว้เพื่อควบคุมจำนวนถังไม่ให้มีค่ามากหรือน้อยจนเกินไป เพราะถ้ามีจำนวนถังมากไปก็จะสิ้นเปลืองหน่วยความจำโดยเปล่าประโยชน์ แต่ถ้าน้อยเกินไปก็จะไม่สามารถกระจายบันทึกเหตุการณ์ได้อย่างที่ต้องการ การเลือกค่าจำนวนถังใหม่จึงต้องเลือกให้สอดคล้องกับสภาพการใช้งานถังในปัจจุบัน คือ ถ้าถังมีมาก (มีค่าบันทึกเหตุการณ์เฉลี่ยเป็น 0.5) ก็จะลดจำนวนถังลงครึ่งหนึ่ง และถ้าถังมีน้อย (มีค่าบันทึกเหตุการณ์เฉลี่ยเป็น 2) ก็จะเพิ่มจำนวนถังขึ้นอีกหนึ่งเท่า ซึ่งทั้งสองเงื่อนไขที่กล่าวมาจะทำให้ปฏิทินใหม่มีจำนวนถังเท่ากับจำนวนบันทึกเหตุการณ์ในแถวคอยปฏิทิน ณ เวลานั้น และนอกจากจะปรับจำนวนถังให้เพิ่มขึ้นหรือลดลงแล้วแถวคอยปฏิทินยังถือโอกาสนี้ปรับค่าความกว้างถังให้เหมาะสมกับลักษณะการกระจายของบันทึกเหตุการณ์ในแถวคอยปฏิทินไปพร้อม ๆ กันเลย โดยปฏิทินใหม่นี้จะ

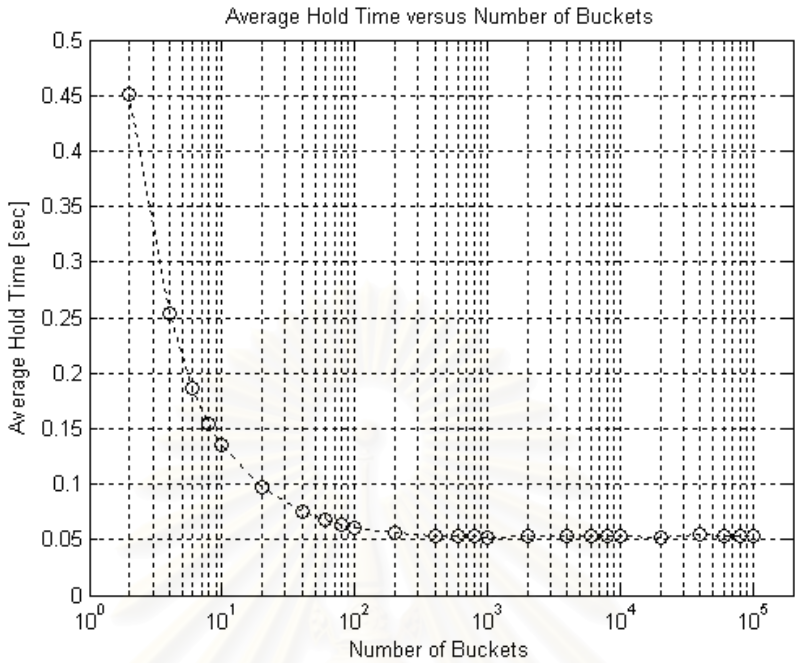
ถอดบันทึกเหตุการณ์ออกมาเพื่อคำนวณค่าเฉลี่ยแล้วจึงใส่บันทึกเหตุการณ์ที่ถอดออกมากลับเข้าไปในแถวคอยปฏิทินแล้วจึงนำค่าเฉลี่ยเวลาระหว่างบันทึกเหตุการณ์ที่คำนวณได้มาเป็นค่าความกว้างของถัง

2.3.5 ประสิทธิภาพของแถวคอยปฏิทิน

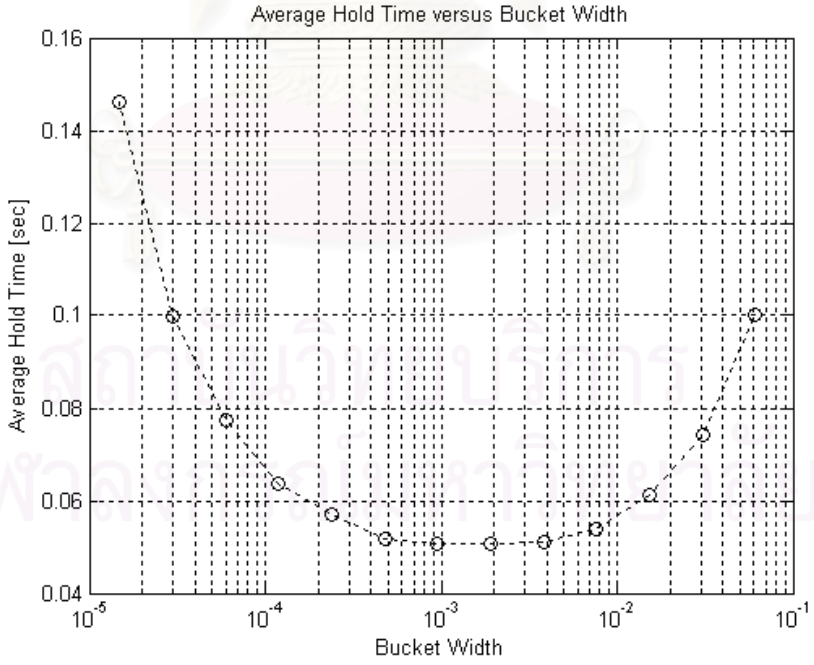
ในหัวข้อนี้จะใช้การดำเนินการเหตุการณ์คงค่า (หัวข้อ 2.2) มาทดสอบการทำงานของแถวคอยปฏิทิน โดยมีค่าอัตราการสร้าง (p) = 0.55 N = 2100 ใบ τ_i มีการกระจายแบบเลขชี้กำลังค่าเฉลี่ย $\mu = 1$ และใช้รายการเชิงเส้นในการจัดเก็บบันทึกเหตุการณ์ในแต่ละถัง เพื่อศึกษาผลกระทบของการเลือกค่าพารามิเตอร์ทั้ง 2 ของแถวคอยปฏิทินที่มีต่อประสิทธิภาพของแถวคอยปฏิทิน โดยจะทำการเปลี่ยนค่าพารามิเตอร์ทั้ง 2 ตัวของแถวคอยปฏิทิน คือ จำนวนถัง และความกว้างถังให้มีค่าต่าง ๆ ในรูปที่ 2.6 แสดงผลการทดสอบประสิทธิภาพของแถวคอยปฏิทินเมื่อจำนวนถัง (M) มีค่าระหว่าง 2 ถึง 100,000 ส่วนในรูปที่ 2.7 แสดงผลการทดสอบประสิทธิภาพของแถวคอยปฏิทินเมื่อความกว้างถัง (δ) มีค่าระหว่าง 0.03125 ถึง 128

เครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบ	
หน่วยประมวลผลกลาง(CPU)	AMD Athlon XP1600+
หน่วยความจำ(RAM)	512 MB
ระบบปฏิบัติการ	Windows XP

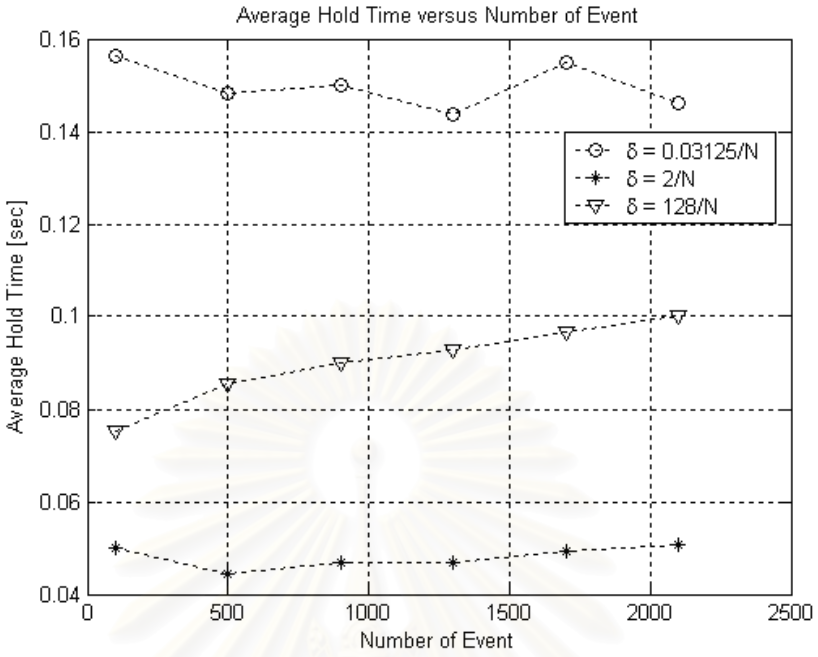
ในรูปที่ 2.6 จะเปลี่ยนจำนวนถัง (M) ให้มีค่าอยู่ระหว่าง 2 ถึง 100,000 แต่จะยังคงการทำงานอื่นของแถวคอยปฏิทินไว้ดังเดิม จะเห็นว่าในช่วงแรก ๆ ที่จำนวนถังมีค่าน้อย ค่าเฉลี่ยของเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่าจะมีค่ามากเนื่องจากจำนวนบันทึกเหตุการณ์ (N) มีค่ามากถึง 2100 ใบ แต่ไม่มีถังที่จะใช้ในการกระจายบันทึกเหตุการณ์ ทำให้แต่ละถังเกิดรายการเชิงเส้นขนาดใหญ่ที่มีความซับซ้อนเชิงเวลาเป็น $O(N)$ และเมื่อเราค่อย ๆ เพิ่มจำนวนถังขึ้นเรื่อย ๆ เวลาที่ใช้ประมวลผลก็จะลดลงเรื่อย ๆ ด้วยเช่นกัน เพราะการเพิ่มจำนวนถังให้แถวคอยปฏิทินมีผลให้บันทึกเหตุการณ์ที่ต้องเก็บไว้ในถังเดียวกันนั้นมีการกระจายการเก็บไว้ในถังหลายใบมากขึ้น ถังแต่ละใบจึงเก็บบันทึกเหตุการณ์น้อยลง แต่อย่างไรก็ตามการเพิ่มจำนวนถังเข้าไปเรื่อย ๆ นั้นไม่ได้ทำให้เกิดผลดีมากขึ้นเท่าไรนัก สังเกตได้จากช่วงแรก (0 ถึง 100) ของการเพิ่มจำนวนถังนั้นจะมีผลให้เวลาที่ใช้ในการประมวลผลการดำเนินการเหตุการณ์คงค่าลดลงอย่างรวดเร็ว แต่ในช่วงท้าย ๆ นั้นการเพิ่มจำนวนถังไม่ได้ทำให้เวลาในการประมวลผลลดลงเท่าไรนักแต่จะค่อย ๆ ลู่เข้าสู่ค่า ๆ หนึ่ง (ในรูปที่ 2.6 ค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่าจะลู่เข้าสู่ค่า 0.05) เป็นเพราะว่าในช่วงแรกการเพิ่มจำนวน



รูปที่ 2.6 ความสัมพันธ์ระหว่างจำนวนถังกับค่าเฉลี่ยของเวลาประมวลผลการดำเนินการเหตุการณ์คงค่า



รูปที่ 2.7 ความ สัมพันธ์ ระหว่าง ความ กว้าง ถัง กับ ค่าเฉลี่ย ของ เวลา ประมวลผล การดำเนินการเหตุการณ์คงค่า



รูปที่ 2.8 ผลจากการเกิดรายการเหตุการณ์ขนาดใหญ่ และการค้นถังอย่างไม่มีประสิทธิภาพ

ถังจะช่วยทำให้สามารถกระจายบันทึกเหตุการณ์ได้ดีขึ้น แต่เมื่อบันทึกเหตุการณ์มีการกระจายอย่างทั่วถึง ไม่ว่าจะเพิ่มจำนวนถังเข้าไปอีกเท่าใดก็ไม่อาจช่วยให้การกระจายของบันทึกเหตุการณ์ดีขึ้นได้ แต่กลับจะทำให้ต้องใช้หน่วยความจำเพิ่มขึ้นเพื่อเก็บตัวแปรของถังแต่ละใบนั่นเอง

ในรูปที่ 2.7 มีการเปลี่ยนค่าความกว้างถัง δ ให้มีค่าอยู่ระหว่าง 1.488×10^{-5} ถึง 6.095×10^{-2} ซึ่งจะเห็นว่าการทำงานของแถวคอยปฏิทินมีประสิทธิภาพไม่เท่ากัน โดยมีเพียงค่าถังที่อยู่ในช่วงใกล้เคียงกับค่า 0.001 เท่านั้นที่ทำให้เวลาที่ใช้ในการประมวลผลการดำเนินการเหตุการณ์คงค่าต่ำที่สุด แต่หากกำหนดให้มีความกว้างถังออกห่างจากค่า 0.001 มาก ๆ จะทำให้ใช้เวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่ามากขึ้น ในกรณีที่กำหนดให้ถังมีความกว้างมากเกินไปนั้นอธิบายได้ว่า ถังที่กว้างเกินไปทำให้บันทึกเหตุการณ์ส่วนใหญ่ถูกเก็บไว้ในถังปัจจุบัน (หรือใกล้เคียง) ลักษณะนี้เองทำให้เกิดรายการเหตุการณ์ขนาดใหญ่ขึ้นที่ถังปัจจุบัน และต้องใช้เวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่ามากขึ้นเพราะต้องใส่บันทึกเหตุการณ์ลงไปในรายการเหตุการณ์ขนาดใหญ่นั่นเอง แต่ในกรณีที่ถังมีความกว้างน้อยเกินไปแถวคอยปฏิทินจะทำงานช้าลงเนื่องจากเหตุการณ์ส่วนใหญ่ที่เก็บอยู่ในถังจะไม่ใช่เหตุการณ์ที่เกิดขึ้นในรอบการทำงานปัจจุบัน ดังนั้นในกระบวนการถอดบันทึกเหตุการณ์ออกมาจากรายการเหตุการณ์จึงต้องค้นบันทึกเหตุการณ์ในแต่ละถังออกมาเพื่อตรวจสอบว่าเป็นเหตุการณ์ในรอบการทำงานปัจจุบันหรือไม่ แต่ส่วนใหญ่ผลที่ได้คือ บันทึกเหตุการณ์ที่ค้นออกมานั้นเป็นบันทึกเหตุการณ์ที่

บันทึกเอาไว้สำหรับรอบการทำงานถัด ๆ ไปจึงต้องค้นรายการเหตุการณ์ในถังถัดไปหลายถังกว่าจะพบบันทึกเหตุการณ์ที่เกิดในรอบการทำงานปัจจุบันครั้งหนึ่ง และหากเราตั้งค่าให้ความกว้างถังให้เล็กลงไปกว่านั้นอีกก็เป็นไปได้ว่าทุก ๆ ครั้งที่จะถอดบันทึกเหตุการณ์ออกมาจากรายการเหตุการณ์ต้องทำการค้นหาโดยตรง ซึ่งจะเกิดขึ้นเมื่อค้นหาบันทึกเหตุการณ์จากทุกถังครบแล้วยังไม่พบบันทึกเหตุการณ์ที่สามารถถอดออกมาได้ ผลการทำงานอย่างนี้จะเรียกว่าการค้นถังอย่างไม่มีประสิทธิภาพ (inefficient search) ผลการเกิดรายการขนาดใหญ่ และการค้นถังอย่างไม่มีประสิทธิภาพ แสดงไว้ในรูปที่ 2.8 โดย เส้น $\delta = 128/N$ เป็นผลจากการเลือกความกว้างถังมากเกินไป (เกิดรายการขนาดใหญ่) เส้น $\delta = 0.03125/N$ เป็นผลจากการเลือกความกว้างถังน้อยเกินไป (เกิดการค้นถังอย่างไม่มีประสิทธิภาพ) ส่วนเส้น $\delta = 2/N$ เป็นการเลือกความกว้างถังได้พอดี (การนำจำนวนเหตุการณ์ (N) มาใช้หาความกว้างถังจะกล่าวต่อไปในบทที่ 3)

ผลการจำลองในรูปที่ 2.6 และ 2.7 สามารถสรุปเป็นภาพรวมของการเลือกค่าพารามิเตอร์ของแถวคอยปฏิทินดังนี้

1. การเลือกค่าจำนวนถังมีเงื่อนไขในการเลือกเพียงอย่างเดียวคือไม่ควรจะเลือกค่าที่น้อยเกินไปจนทำให้แต่ละถังเกิดเป็นรายการขนาดใหญ่ได้ก็พอ และหากต้องการเพิ่มความเร็วให้มากขึ้นก็สามารถทำได้โดยเพิ่มจำนวนถังให้กับแถวคอยปฏิทิน โดยความเร็วที่เพิ่มขึ้นนี้ก็ต้องแลกมาด้วยหน่วยความจำปริมาณมากที่ต้องใช้ในการสร้างถังสำหรับกระจายบันทึกเหตุการณ์เป็นรายการเหตุการณ์ย่อย เพราะแม้การเพิ่มจำนวนถังในช่วงแรก ๆ จะส่งผลให้มีความเร็วเพิ่มขึ้นเป็นอย่างมาก แต่การเพิ่มจำนวนถังไปเรื่อย ๆ พบว่าในช่วงหลัง ๆ การเพิ่มจำนวนถังไม่ได้ช่วยให้มีความเร็วเพิ่มขึ้นเลย หรืออาจจะเพิ่มขึ้นแต่น้อยมาก ใน [8] ได้เสนอให้มีจำนวนวันเท่ากันหรือใกล้เคียงกับจำนวนบันทึกเหตุการณ์ในแถวคอยปฏิทิน ซึ่งค่านี้ก็ยังคงถูกใช้อยู่ในปัจจุบัน

2. การเลือกค่าความกว้างถังนั้นจะต้องเลือกอย่างระมัดระวังเป็นอย่างยิ่ง เพราะการกำหนดค่าความกว้างถังนี้ไม่อาจจะกำหนดให้มีค่าโน้มเอนไปทางใดทางหนึ่งเหมือนกันกับค่าจำนวนถัง (กำหนดค่าไว้มากเกินไปก็ไม่เป็นไร) แต่อย่างไรก็ตามค่าความกว้างถังที่ส่งผลให้แถวคอยปฏิทินมีประสิทธิภาพการทำงานใกล้เคียงกับค่าที่ดีที่สุดนั้นก็เป็นช่วงที่ค่อนข้างกว้าง ดังนั้นถ้าสามารถเลือกค่าความกว้างถังให้อยู่ในช่วงดังกล่าวได้ก็จะสามารถทำงานได้อย่างมีประสิทธิภาพ แต่หากค่าที่เลือกไว้เริ่มออกจากช่วงดังกล่าวนี้แล้ว เวลาที่ใช้ในการประมวลผลก็จะเพิ่มขึ้นอย่างรวดเร็ว ซึ่งใน [8] ได้เสนอให้เลือกค่าความกว้างถังจากค่าเฉลี่ยของเวลาระหว่างเหตุการณ์ในปฏิทิน

2.4 สรุป

ในบทนี้ได้กล่าวถึงหลักการทำงานทั่วไปของการจำลองแบบเหตุการณ์ไม่ต่อเนื่อง เพื่อให้เข้าใจความสัมพันธ์ระหว่างการจำลองแบบเหตุการณ์ไม่ต่อเนื่องกับแถวคอยแบบมีลำดับความสำคัญ หลังจากนั้นจึงได้อธิบายหลักการทำงานของแถวคอยปฏิทิน ซึ่งเป็นแถวคอยแบบมีลำดับความสำคัญที่มีประสิทธิภาพในการทำงานสูง และได้รับความนิยมเป็นอย่างมาก พร้อมทั้งแสดงบทวิเคราะห์การทำงานของแถวคอยปฏิทิน โดยจะเน้นไปที่การวิเคราะห์ผลจากการเลือกค่าพารามิเตอร์ไม่เหมาะสมให้กับแถวคอยปฏิทิน ผลการศึกษาแสดงให้เห็นว่าแถวคอยปฏิทินอาจทำงานได้ช้าลงเป็นอย่างมากหากค่าพารามิเตอร์ที่เลือกมานั้นเป็นค่าที่ไม่เหมาะสม เพราะอาจทำให้เกิดแถวคอยขนาดใหญ่ หรือการค้นถังอย่างไม่มีประสิทธิภาพได้ ซึ่งเหตุการณ์ทั้ง 2 เหตุการณ์ที่กล่าวมานี้จะทำให้แถวคอยปฏิทินมีความซับซ้อนเชิงเวลาสูงขึ้น

บทที่ 3

แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ

เราได้ทราบจากเนื้อหาในหัวข้อที่ 2.3 แล้วว่าแถวคอยปฏิทินจะมีการปรับค่าพารามิเตอร์ก็ต่อเมื่อจำนวนบันทึกเหตุการณ์ในปฏิทินมีการเปลี่ยนแปลงอย่างรุนแรงเท่านั้น แต่ในการใช้งานจริงเมื่อโปรแกรมจำลองทำงานในสภาวะอยู่ตัวแล้วจำนวนบันทึกเหตุการณ์ในแถวคอยปฏิทินจะมีค่าเปลี่ยนแปลงเพียงเล็กน้อยเท่านั้นซึ่งจะไม่ทำให้เกิดการปรับค่าพารามิเตอร์ ดังนั้นหากในขณะที่โปรแกรมทำงานอยู่ในสภาวะนี้แล้ว การกระจายของบันทึกเหตุการณ์เกิดการเปลี่ยนแปลงไป (มีการเปลี่ยนค่าพารามิเตอร์ เช่น อัตราการส่งข้อมูล เป็นต้น) แถวคอยปฏิทินจะไม่สามารถเปลี่ยนแปลงค่าพารามิเตอร์ให้เหมาะสมกับสภาพการทำงานที่เปลี่ยนไปทำให้แถวคอยปฏิทินทำงานช้าลง ซึ่งสถานการณ์ที่กล่าวมานี้อาจจะเกิดขึ้นได้ในโปรแกรมจำลอง เช่น การจำลองระบบ ATM ที่มีแหล่งกำหนดแบบเปิดปิด (on-off source) หรือการใช้งานอินเตอร์เน็ตที่ปริมาณความต้องการแบนด์วิดท์ (bandwidth) ของผู้ใช้แปรผันตามเวลา เป็นต้น นอกจากนั้นวิธีการเลือกค่าพารามิเตอร์ที่ใช้กันอยู่ในปัจจุบันยังไม่ใช่วิธีการเลือกค่าพารามิเตอร์ที่มีประสิทธิภาพดีที่สุด

ในบทนี้จะแสดงผลการศึกษาวิธีการเลือกค่าความกว้างถังสำหรับแถวคอยปฏิทินที่ทำให้แถวคอยปฏิทินมีประสิทธิภาพในการทำงานสูงสุด และจะนำวิธีเลือกค่าความกว้างถังที่ศึกษามาไปใช้ในการกำหนดและควบคุมค่าความกว้างถังเพื่อให้แถวคอยปฏิทินมีความทนทานต่อการใช้งานที่หลากหลายมากยิ่งขึ้น

3.1 การเลือกค่าความกว้างถัง

แม้ว่าพารามิเตอร์ที่ส่งผลต่อประสิทธิภาพการทำงานของแถวคอยปฏิทินจะมีอยู่ด้วยกัน 2 ตัว คือ จำนวนถัง และความกว้างถัง แต่ในหัวข้อที่ 2.3.5 ได้แสดงให้เห็นแล้วว่าไม่สมควรใช้ค่าจำนวนถังในการเพิ่มความเร็วในการทำงานให้แถวคอยปฏิทิน เพราะหากต้องการทำให้แถวคอยปฏิทินมีความเร็วในการทำงานเพิ่มขึ้นด้วยการเพิ่มจำนวนถังแถวคอยปฏิทินจะต้องใช้หน่วยความจำเพิ่มขึ้นเป็นจำนวนมากเพื่อแลกกับความเร็วที่เพิ่มขึ้นเพียงเล็กน้อยเท่านั้น แต่สำหรับค่าความกว้างของถังนั้นไม่ว่าจะกำหนดให้มีค่าเป็นเท่าไรก็ตามก็จะไม่ทำให้ต้องใช้หน่วยความจำเพิ่มขึ้น ซึ่งหมายความว่าถ้าเราเลือกค่าความกว้างถังได้ดีจะสามารถเพิ่มความเร็วให้กับแถวคอยปฏิทินได้โดยที่ไม่ต้องเสียอะไรตอบแทนเลย ดังนั้นในวิทยานิพนธ์ฉบับนี้จึงเลือกศึกษาค่าความกว้างถัง เพื่อหาวิธีเลือกค่าความกว้างถังที่ดีและสามารถจะนำไปใช้กับแถวคอยปฏิทินได้อย่างมีประสิทธิภาพ

แม้ว่าแถวคอยปฏิทินจะถูกนำมาใช้งานเป็นระยะเวลานานแล้วแต่ก็ยังไม่มีการพิสูจน์ยืนยันใด ๆ ว่าการกำหนดค่าให้กับความกว้างถังที่ใช้กันมานั้นเป็นค่าที่เหมาะสมในการใช้งานจริง ๆ จนกระทั้งใน [11] ได้แสดงการวิเคราะห์ความเร็วในการทำงานของแถวคอยปฏิทินที่มีจำนวนถังเป็นอนันต์ในรูปของค่าคาดคะเนของเวลาที่ใช้ประมวลผลเหตุการณ์ (expected time per event) เพื่อหาขนาดของถังที่ทำให้แถวคอยปฏิทินมีประสิทธิภาพการทำงานสูงสุด โดยมีสมมติฐานในการวิเคราะห์ดังนี้

1. แถวคอยปฏิทินที่ใช้มีจำนวนถังเป็นอนันต์ ($M = \infty$) ดังนั้นแต่ละถังจะไม่มีบันทึกเหตุการณ์ที่จะเกิดในรอบการทำงานถัดไปเก็บอยู่ หรืออีกนัยหนึ่งคือไม่มีการใช้ปฏิทินซ้ำนั่นเอง
2. มีความกว้างถัง (δ) เป็นค่าจำนวนจริงบวกใด ๆ
3. ใช้รายการไร้ลำดับ (unordered list) ในการจัดการบันทึกเหตุการณ์ที่อยู่ในถังแต่ละใบ
4. แถวคอยปฏิทินมีจำนวนบันทึกเหตุการณ์ (N) คงที่
5. ในแต่ละรอบการทำงานจะประกอบไปด้วย การตรวจสอบถังปัจจุบันว่ามีบันทึกเหตุการณ์อยู่หรือไม่ ถ้าไม่มีก็จะย้ายไปตรวจสอบถังถัดไป และจะทำเช่นนี้ต่อไป จนพบถังที่มีบันทึกเหตุการณ์อยู่ หลังจากนั้นจึงถอดบันทึกเหตุการณ์ที่มีเวลาในการเกิดต่ำที่สุดออกมา แล้วสร้างบันทึกเหตุการณ์ใหม่ใส่เข้าไปในปฏิทินโดยมีเวลาของเหตุการณ์ใหม่เป็น $t(e) + \tau_i$ โดย $t(e)$ คือ เวลาของบันทึกเหตุการณ์ที่ถูกถอดออกมา และ τ_i เป็นค่าสุ่มซึ่งมีการกระจาย f มีค่าเฉลี่ย μ
6. ค่าเวลาในการประมวลผลถังเปล่า (τ_b) เวลาในการตรวจสอบบันทึกเหตุการณ์หนึ่งใบ (τ_l) และเวลาที่ใช้สำหรับประมวลผลบันทึกเหตุการณ์ (τ_e) เป็นค่าคงที่

ตารางที่ 3.1 พารามิเตอร์ที่ใช้ในการวิเคราะห์ค่าคาดคะเนของเวลาที่ใช้ประมวลผลเหตุการณ์

ตัวย่อ	ความหมาย	วิธีได้รับค่า
N	จำนวนบันทึกเหตุการณ์	รู้ได้เพราะเป็นตัวแปรของแถวคอยปฏิทิน
μ	ค่าเฉลี่ยของค่าเวลาเพิ่ม	ประมาณค่าได้
τ_b	เวลาในการประมวลผลถังเปล่า	รู้ได้โดยการเทียบวัด (calibration)
τ_l	เวลาในการตรวจสอบบันทึกเหตุการณ์หนึ่งใบ	รู้ได้โดยการเทียบวัด (calibration)
τ_e	เวลาในการประมวลผลบันทึกเหตุการณ์	รู้ได้โดยการเทียบวัด (calibration)
δ	ความกว้างถัง	รู้ได้เพราะเป็นตัวแปรของแถวคอยปฏิทิน
M	จำนวนถัง	รู้ได้เพราะเป็นตัวแปรของแถวคอยปฏิทิน

จาก สมมติฐาน ที่ กล่าว มา สามารถ วิเคราะห์ หา ค่า คาดคะเน ของ เวลา ที่ ใช้ ประมวลผล เหตุการณ์ ใน แถวคอยปฏิทินที่ต่ำที่สุด และค่าความกว้างถังที่สอดคล้องกันได้ดังนี้

$$K_N(\delta_{opt}) = \tau_e + \tau_l + \sqrt{2\tau_b/\tau_l} + O(N^{-1}) \tag{3.1}$$

$$\delta_{opt} = \sqrt{2\tau_b/\tau_l}\frac{\mu}{N} + O(N^{-3/2}) \tag{3.2}$$

โดยที่ $K_N(\delta_{opt})$ คือค่าคาดคะเนของเวลาที่ใช้ประมวลผลเหตุการณ์ (expected time per event)

δ_{opt} คือความกว้างถังที่เหมาะสม (optimise width factor)

จากสมการที่ (3.1) และ (3.2) (มีพิสูจน์อยู่ในภาคผนวก D ของเอกสาร [11]) จะเห็นว่าค่าความกว้างถังที่ทำให้ใช้เวลาในการประมวลผลต่ำที่สุดสามารถคำนวณได้หากรู้ค่า τ_b, τ_l, μ, N ซึ่งสามารถหาได้ด้วยวิธีที่บอกไว้ในตารางที่ 3.1

แม้ว่าเราจะสามารถหาค่าความกว้างถังที่เหมาะสม (optimise bucket width) ได้จากสมการที่ (3.2) แต่ในการใช้งานสมการที่ (3.1) และ (3.2) นั้นต้องใช้ด้วยความระมัดระวังเป็นอย่างยิ่ง เนื่องจากสมมติฐานที่ใช้ในการพิสูจน์และเงื่อนไขการใช้งานบางอย่างของสมการที่ (3.1) และ (3.2) นั้นไม่สอดคล้องกับการใช้งานจริง ซึ่งสมมติฐานและเงื่อนไขการใช้งานดังกล่าวมีอยู่ด้วยกัน 4 ข้อดังนี้

1. มีจำนวนถังเป็นอนันต์: การกำหนดให้จำนวนถังเป็นอนันต์ไม่สามารถทำได้ในทางปฏิบัติเพราะไม่มีเครื่องคอมพิวเตอร์ที่มีหน่วยความจำอนันต์

2. ใช้รายการไร้ลำดับ (unordered list) ในการจัดการบันทึกเหตุการณ์ในถังแต่ละใบ: แต่ส่วนใหญ่แล้วแถวคอยปฏิทินมักจะนิยมใช้ รายการเชิงเส้น (linear list) ในการจัดการบันทึกเหตุการณ์ในถังแต่ละใบมากกว่า เนื่องจากมีการประมวลผลที่เร็วกว่า

3. ต้องทำการเทียบวัด (calibration) ค่าคงที่ในการประมวลผล (τ_b, τ_l) ก่อนการใช้งาน: ทำให้ไม่สามารถใช้งานแถวคอยปฏิทินได้สะดวก เนื่องจากต้องคอยเทียบวัดค่าเหล่านี้ก่อนการใช้งานทุกครั้ง

4. ค่าคงที่ในการประมวลผล τ_b, τ_l, τ_e เป็นค่าคงที่: แต่ในความเป็นจริงแล้วค่าเหล่านี้อาจจะเป็นหรือไม่เป็นค่าคงที่ก็ได้ขึ้นอยู่กับปัจจัยหลาย ๆ อย่าง เช่น การกระจายบางชนิดต้องใช้เวลาในการสร้างค่าสุ่มแต่ละครั้งไม่เท่ากัน การประมวลผลเหตุการณ์แต่ละชนิดใช้เวลาไม่เท่ากัน เป็นต้น นอกจากนั้นยังอาจเป็นผลมาจากเครื่องคอมพิวเตอร์ที่ทำการจำลอง เช่น ระบบปฏิบัติการที่ใช้ หน่วยความจำที่มีอยู่เป็นต้น

เนื่องจากสมมติฐานที่ใช้ในการพิสูจน์ และเงื่อนไขการใช้งานบางอย่างของสมการที่ (3.1) และ (3.2) ทำให้ไม่สามารถใช้การหาค่าความกว้างถังด้วยสมการที่ (3.2) ได้ในทางปฏิบัติ วิทยานิพนธ์ฉบับนี้จึงเสนอวิธีการการหาค่าความกว้างถังที่สามารถนำไปใช้ในทางปฏิบัติได้สะดวกขึ้น โดยจะทำการแก้ไขสมการที่ (3.2) เพื่อนำไปใช้ดังนี้

$$\delta_{opt} = W_{fopt}\frac{\mu}{N} \tag{3.3}$$

เนื่องจากงานวิจัยนี้เราสนใจแถวคอยปฏิทินที่มีบันทึกเหตุการณ์จำนวนมาก ทำให้พจน์ $O(N^{-3/2})$ มีค่าน้อยมาก จึงสามารถตัดออกจากสมการได้ ส่วนตัวแปรตัวคูณความกว้างที่เหมาะสม (optimise width factor: $W_{fopt} = \sqrt{2\tau_b/\tau_l}$) ถูกเสนอขึ้นมาเพื่อใช้แทนค่าคงที่ในการประมวลผล τ_b และ τ_l ซึ่งจะช่วยลดพื้นที่ในการค้นหาค่าคงที่ซึ่งใช้ในสมการที่ (3.2) โดยจะลดจากการค้นหาสองมิติ (2 ตัวแปร คือ τ_b และ τ_l) เหลือเพียงมิติเดียว (ตัวแปรเดียวคือ W_{fopt}) ซึ่งเนื้อหาในส่วนถัดไปจะแสดงผลการทดสอบการทำงานของแถวคอยปฏิทินเมื่อเลือกใช้ W_f ค่าต่าง ๆ โดยจะกำหนดค่าความกว้างถัง (δ) ด้วยสมการที่ (3.4) และจะนำผลการทดสอบนี้ไปใช้ในการหาค่าตัวคูณความกว้างที่เหมาะสม (W_{fopt}) ต่อไป

$$\delta = W_f\frac{\mu}{N} \tag{3.4}$$

ในการทดลองนี้จะใช้การดำเนินการเหตุการณ์คงค่า (หัวข้อที่ 2.2) ในการวัดประสิทธิภาพการทำงานของแถวคอยปฏิทินเมื่อเลือกค่า W_f ต่าง ๆ กันมาใช้กำหนดความกว้างถัง โดยมีค่าอัตราการสร้าง (p) = 0.55 จำนวนบันทึกเหตุการณ์ (N) = 100, 500, 900, 1400, 1700 และ2100 ใบ และทดสอบกับการกระจาย (f) ทั้งหมด 5 แบบ ตามที่แสดงไว้ในตารางที่ 3.2

เครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบ	
หน่วยประมวลผลกลาง(CPU)	AMD Athlon XP1600+
หน่วยความจำ(RAM)	512 MB
ระบบปฏิบัติการ	Windows XP

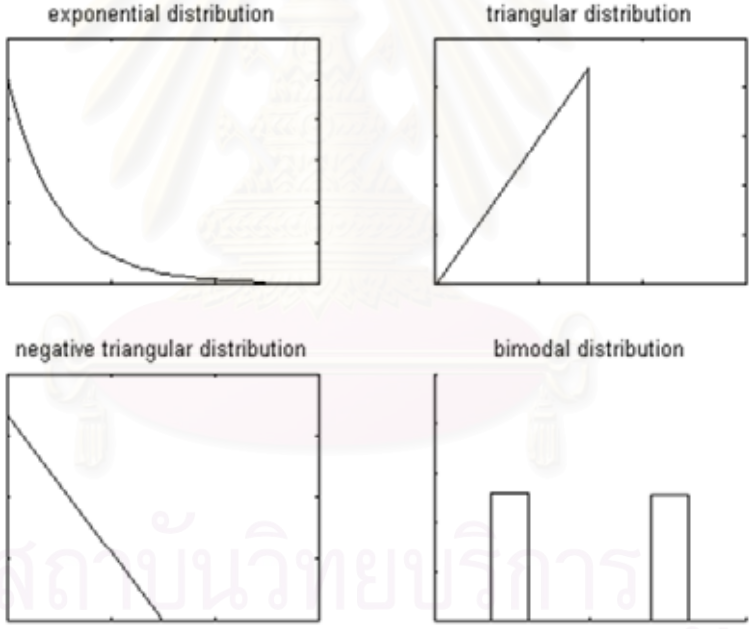
ผลการทดสอบความสัมพันธ์ระหว่างการเลือกค่าตัวคูณความกว้างกับประสิทธิภาพของแถวคอยปฏิทินจะแสดงออกมาในรูปของกราฟแสดงความสัมพันธ์ระหว่าง ค่าเฉลี่ยเวลาที่ใช้ในการประมวลผลการดำเนินการเหตุการณ์คงค่ากับค่าตัวคูณความกว้าง โดยผลทดสอบการประสิทธิภาพการใช้งานเมื่อ τ_i มีการกระจายแบบต่าง ๆ แสดงไว้ในรูปที่ 3.2-3.6

ตารางที่ 3.2 การกระจายของ τ_i

ชนิดของการกระจาย(†)	ฟังก์ชั่นที่ใช้สร้างค่าสุ่ม(‡)
ก) เลขชี้กำลัง (exponential)	$-\ln rand$
ข) สามเหลี่ยม (triangular)	$1.5rand^{0.5}$
ค) สามเหลี่ยมกลับด้าน (negative triangular)	$3((1-rand)^{0.5}+1)$
ง) สองฐานนิยม 1 (bimodal 1)	0.01rand + if $rand > 0.5$ then 0.4975 else 1.4975
จ) สองฐานนิยม 2 (bimodal 2)	0.2rand + if $rand < 0.2$ then 0.45 else 1.45

(†) ค่าเฉลี่ยของ τ_i เป็น 1 ในทุกกรณี

(‡) ฟังก์ชั่น rand จะสร้างค่าสุ่มที่มีการกระจายแบบสม่ำเสมอ (uniform) ระหว่าง 0 และ 1

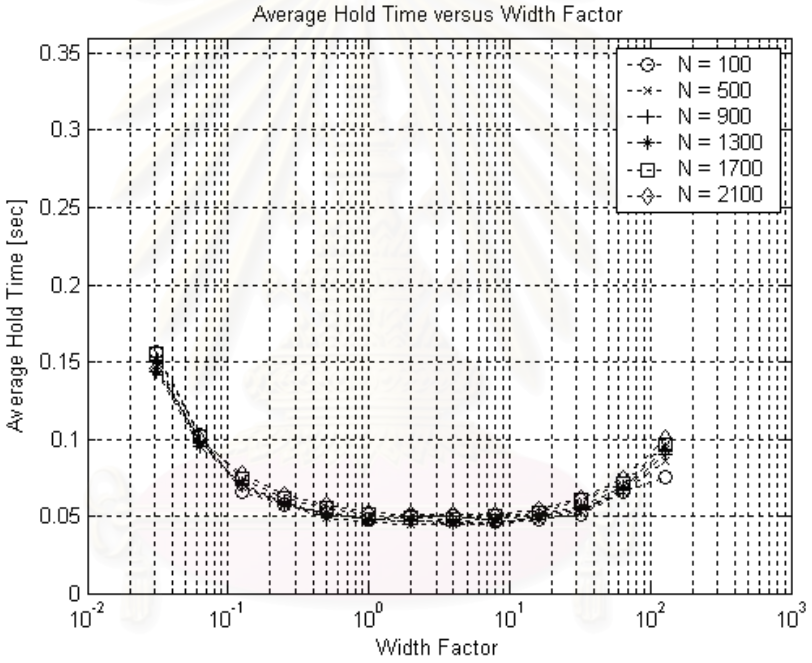


รูปที่ 3.1 การกระจายของ τ_i

ก) การกระจายแบบเลขชี้กำลัง

การกระจายแบบเลขชี้กำลัง (exponential distribution) เป็นการกระจายที่มีคุณสมบัติไร้ความทรงจำ (memory less) ซึ่งมีลักษณะเหมือนกันการกระจายของเหตุการณ์หลายชนิดที่ใช้ในการจำลองจริง ทำให้การกระจายแบบนี้เป็นที่นิยมนำมาใช้ในการจำลองเป็นอย่างยิ่ง ในรูปที่ 3.2 แสดงให้

เห็นถึงผลการทำงานเมื่อเลือกค่า W_f ต่าง ๆ กัน จากรูปจะเห็นได้อย่างชัดเจนว่า W_{fopt} จะมีค่าอยู่ในช่วงระหว่าง 1 ถึง 10 แต่เราไม่สามารถทราบค่าที่แน่นอนของ W_{fopt} ว่ามีค่าเป็นเท่าใดเนื่องจากมีค่า W_f หลายค่าที่มีผลการทำงานใกล้เคียงกัน ซึ่งหมายความว่าค่า W_f ที่ทำให้แถวคอยปฏิทินมีค่าเวลาประมวลผลการดำเนินการเหตุการณ์คงค่าเฉลี่ยใกล้เคียงกันกับตอนที่ $W_f = W_{fopt}$ นั้นมีได้หลายค่า และจะครอบคลุมช่วงของ W_f ที่กว้างมาก แต่ถ้าเลือกค่า W_f ที่อยู่นอกช่วงนี้มาใช้จะทำให้เวลาที่ใช้ในการประมวลผลการดำเนินการเหตุการณ์คงค่าเพิ่มขึ้นอย่างรวดเร็ว โดยการเพิ่มขึ้นทางด้านขวาของกราฟ (W_f มีค่าสูง) เกิดจากรายการเหตุการณ์ขนาดใหญ่ที่เกิดขึ้น ส่วนการเพิ่มขึ้นทางด้านซ้ายของกราฟ (W_f มีค่าต่ำ) เกิดจากการค้นถังอย่างไม่มีประสิทธิภาพ ดังได้อธิบายไปแล้วในหัวข้อ 2.3.5

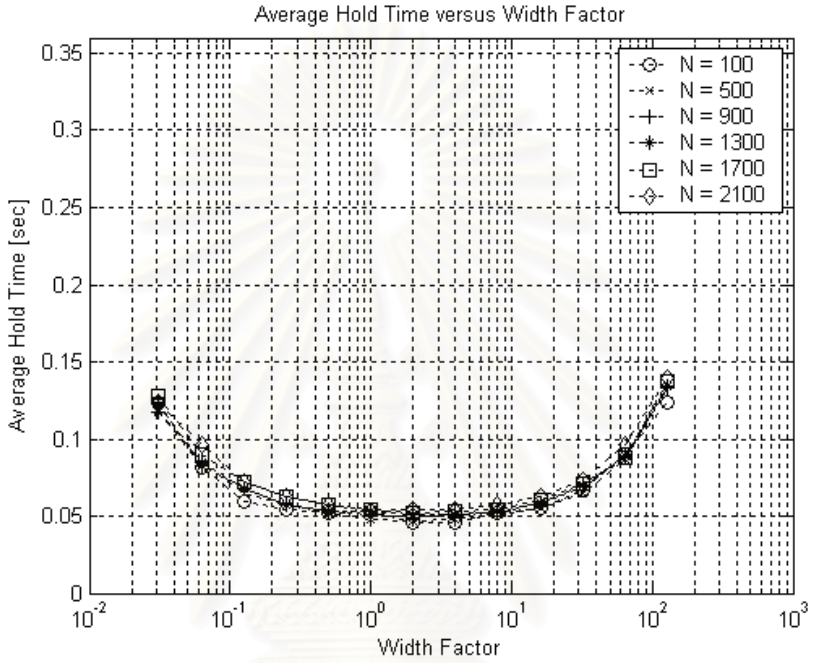


รูปที่ 3.2 ความสัมพันธ์ระหว่างค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่ากับตัวคูณความกว้าง การกระจายแบบเลขชี้กำลัง $\mu = 1, p = 0.55$

ข) การกระจายแบบสามเหลี่ยม

การกระจายแบบสามเหลี่ยม เป็นตัวแทนของการใช้งานในกรณีที่ค่าเวลาเพิ่มของบันทึกเหตุการณ์เกาะกลุ่มกันอยู่ในช่วงที่มีค่าสูง ในรูปที่ 3.3 แสดงให้เห็นถึงผลการทำงานเมื่อเลือกค่า W_f ต่าง ๆ กัน ซึ่งผลการทำงานในรูปที่ 3.3 นี้มีลักษณะคล้าย ๆ กับกราฟในรูปที่ 3.2 แต่ถูกเลื่อนกราฟไปทางด้านซ้ายเล็กน้อย การที่กราฟเลื่อนออกไปทางซ้ายนั้นมีความหมายว่ามีโอกาสเกิดรายการขนาดใหญ่ มากกว่าเกิด

การค้นถังอย่างไม่มีประสิทธิภาพ ซึ่งการค้นถังอย่างไม่มีประสิทธิภาพนั้นจะเกิดจากการที่ค่าสุ่มที่สร้างขึ้นมามีค่ามากกว่าความกว้างถังมาก ๆ แต่สำหรับการกระจายแบบสามเหลี่ยมนี้ ค่าสุ่มที่สร้างขึ้นมาถูกจำกัดค่าไว้ที่ 1.5 ซึ่งเมื่อค่าสุ่มที่สร้างมาถูกจำกัดค่าเอาไว้แสดงว่าเหตุการณ์ส่วนใหญ่จะเกิดในช่วงเวลาน้อย ๆ เท่านั้น จึงมีโอกาสในการเกิดรายการขนาดใหญ่มากกว่าการทดสอบด้วยการกระจายแบบเลขชี้กำลัง

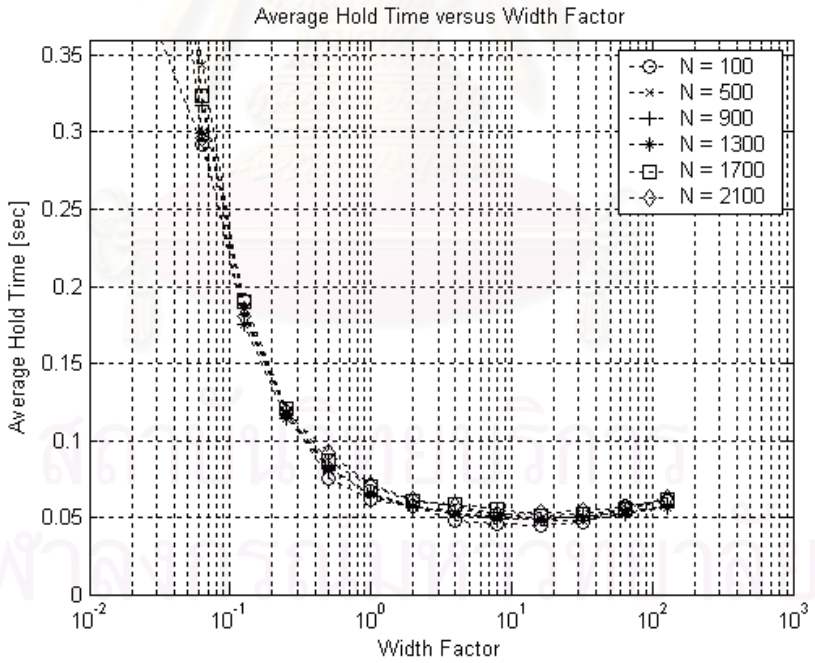


รูปที่ 3.3 ความสัมพันธ์ระหว่างค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่ากับตัวคูณความกว้าง การกระจายแบบสามเหลี่ยม $\mu = 1, p = 0.55$

ค) การกระจายแบบสามเหลี่ยมกลับด้าน

การกระจายแบบสามเหลี่ยม เป็นตัวแทนของการใช้งานในกรณีที่ค่าเวลาเพิ่มของบันทึกเหตุการณ์เกาะกลุ่มกันอยู่ในช่วงที่มีค่าต่ำ ในรูปที่ 3.4 แสดงให้เห็นถึงผลการทำงานเมื่อเลือกค่า W_f ต่าง ๆ กัน ซึ่งผลการทำงานในรูปที่ 3.4 แสดงให้เห็นว่ากราฟนี้มีลักษณะคล้ายกับกราฟในรูปที่ 3.2 และ 3.3 แต่กราฟถูกเลื่อนไปทางขวา เพราะการกระจายแบบนี้มีโอกาสที่จะเกิดการค้นถังอย่างไม่มีประสิทธิภาพสูงกว่าการกระจายทั้ง 2 แบบที่ทดสอบมาก่อนหน้านี้ โดยเมื่อเรานำการกระจายนี้ไปเทียบกับการกระจายแบบเลขชี้กำลังแล้วพบว่า ค่าสุ่มที่ได้จากการกระจายแบบเลขชี้กำลังจะมีค่าสูงกว่าการกระจายแบบสามเหลี่ยมกลับด้าน (เพราะมีค่าได้ตั้งแต่ 0 ถึง อนันต์ แต่การกระจายแบบสามเหลี่ยมกลับด้านมีค่าได้ไม่เกิน 3) แต่การกระจายแบบสามเหลี่ยมกลับด้านมีโอกาสสร้างค่าสุ่มที่มีค่าสูงได้บ่อยครั้งกว่าการ

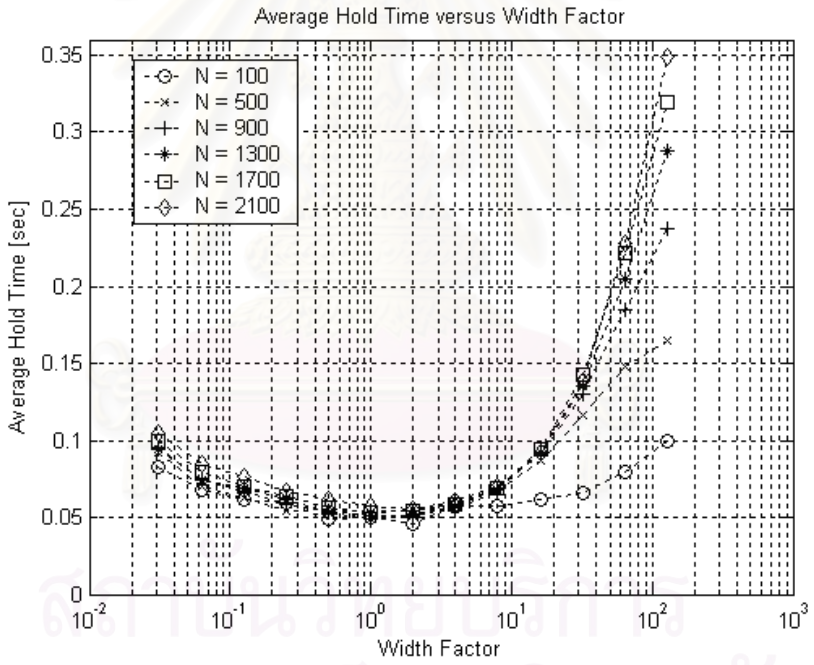
กระจายแบบเลขชี้กำลัง ตัวอย่างเช่นเมื่อเราเลือกค่า $W_f = 1$ ทำให้ปฏิทินของเรามีความกว้างถัง $(\delta) = W_f \times \mu/N = \mu/N$ และหนึ่งปีจะมีเวลาเท่ากับ $M \times \delta$ แต่เนื่องจาก $M \approx N$ ทำให้หนึ่งปีมีความกว้างเป็น $N \times \mu/N = 1$ เมื่อเราหาค่าความน่าจะเป็นที่ค่าสุ่มที่สร้างขึ้นมาจะมีค่ามากกว่า 1 พบว่าการกระจายทั้ง 2 แบบให้ค่าความน่าจะเป็นออกมาดังนี้ การกระจายแบบเลขชี้กำลัง มี $P(\tau_i > 1) = 0.37$ ส่วนการกระจายแบบสามเหลี่ยมกลับด้านมี $P(\tau_i > 1) = 0.44$ ซึ่งแสดงให้เห็นว่าการกระจายแบบสามเหลี่ยมกลับด้านมีโอกาสในการสร้างเหตุการณ์ข้ามปีสูงกว่าการกระจายแบบเลขชี้กำลัง ซึ่งการสร้างเหตุการณ์ข้ามปีนี้เป็นปัจจัยที่ทำให้เกิดการค้นถังอย่างไม่มีประสิทธิภาพ แต่เมื่อเรานำการกระจายแบบสามเหลี่ยมมาหา ความน่าจะเป็นที่ค่าสุ่มที่สร้างขึ้นมาจะมีค่ามากกว่า 1 พบว่ามีค่าความน่าจะเป็นสูงกว่าการกระจายอีก 2 แบบ $(P(\tau_i > 1) = 0.55)$ ซึ่งแสดงว่าการกระจายแบบสามเหลี่ยมมีโอกาสที่จะเกิดการค้นถังอย่างไม่มีประสิทธิภาพสูงกว่าการกระจายอีก 2 แบบด้วย แต่ในความเป็นจริงแล้วปัจจัยนี้ปัจจัยเดียวไม่เพียงพอที่จะใช้บอกได้เพราะการกระจายแบบสามเหลี่ยมนั้นมีค่าจำกัดค่อนข้างต่ำ (มีค่าไม่เกิน 1.5) ดังนั้นแม้ว่าเหตุการณ์ที่สร้างขึ้นส่วนใหญ่จะมีค่าสูงกว่า 1 แต่ก็สูงกว่าเพียงเล็กน้อยเท่านั้นไม่ได้ทำให้มีการค้นถังอย่างไม่มีประสิทธิภาพเกิดขึ้น



รูปที่ 3.4 ความสัมพันธ์ระหว่างค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่ากับตัวคูณความกว้าง การกระจายแบบสามเหลี่ยมกลับด้าน $\mu = 1, p = 0.55$

ง) การกระจายแบบสองฐานนิยม 1

การกระจายแบบสองฐานนิยม 1 เป็นตัวแทนการทำงานในกรณีที่ค่าเวลาเพิ่มของบันทึกเหตุการณ์เกาะกลุ่มกันอย่างหนาแน่นรอบ ๆ ค่าสองค่า กราฟในรูปที่ 3.5 แสดงให้เห็นถึงแนวโน้มในการเกิดรายการเหตุการณ์ขนาดใหญ่ (เห็นการเพิ่มขึ้นของค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่าทางด้านซ้ายอย่างชัดเจน) ของการทดลองด้วยกระจายแบบฐานนิยม 1 ซึ่งเหตุที่การกระจายแบบสองฐานนิยม 1 นี้มีรายการเหตุการณ์ขนาดใหญ่เกิดขึ้นได้ง่ายเพราะลักษณะการกระจายมีการรวมตัวกันอยู่อย่างหนาแน่นในช่วงเวลาแคบ ๆ ดังนั้นการแยกค่าสุ่มที่สร้างขึ้นมา (จากกลุ่มเดียวกัน) จึงทำได้ยากเพราะค่าสุ่มส่วนใหญ่มีเวลาแตกต่างกันเพียงเล็กน้อยเท่านั้น ดังนั้นหากทำให้การแยกค่าสุ่มแต่ละค่าออกจากกันต้องใช้ถังที่แคบมาก ๆ ดังนั้นเมื่อเรานำถังที่มีความกว้างพอดีสำหรับการกระจายแบบอื่นมาใช้ ก็จะเล็กเกินไปสำหรับการกระจายแบบฐานนิยม 1 เป็นเหตุให้มีรายการเหตุการณ์ขนาดใหญ่เกิดขึ้น

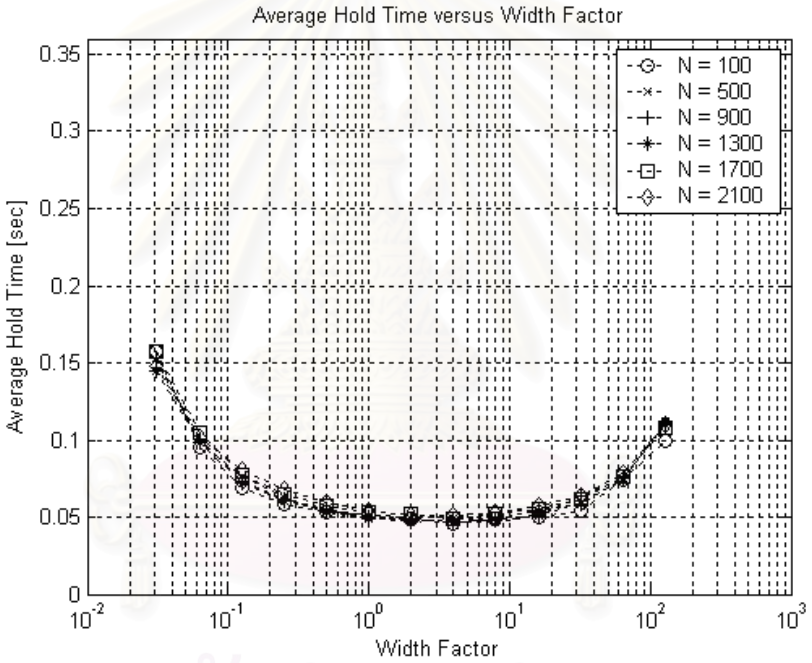


รูปที่ 3.5 ความสัมพันธ์ระหว่างค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่ากับตัวคูณความกว้าง การกระจายแบบสองฐานนิยม 1 $\mu = 1, p = 0.55$

จ) การกระจายแบบสองฐานนิยม 2

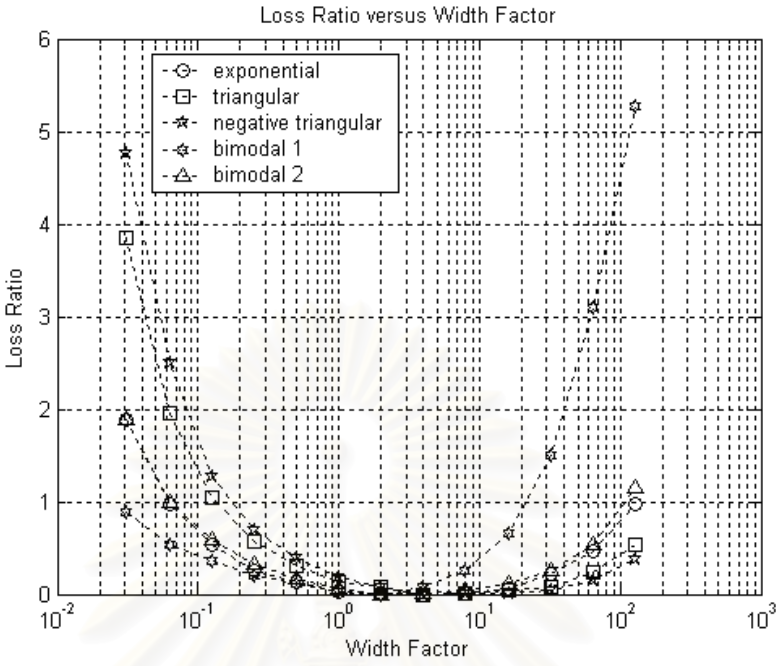
การกระจายแบบสองฐานนิยม 2 เป็นตัวแทนการทำงานในกรณีที่ค่าเวลาเพิ่มของบันทึกเหตุการณ์เกาะกลุ่มกันอย่างหนาแน่นรอบ ๆ ค่าสองค่า แต่จะไม่หนาแน่นเท่ากับการกระจายแบบสองฐานนิยม 1

ทำให้การกระจายมีลักษณะคล้ายกับการกระจายแบบสม่ำเสมอ (uniform) มากขึ้น กราฟในรูปที่ 3.6 แสดงให้เห็นว่าการกระจายแบบสองฐานนิยม 2 ไม่มีแนวโน้มในการเกิดรายการเชิงเส้นขนาดใหญ่ และการค้นถังอย่างไม่มีประสิทธิภาพเลยเมื่อเทียบกับการกระจายแบบอื่น ๆ ที่ได้ทดลองมา การกระจายแบบสองฐานนิยม 2 นี้จะมีค่า W_{fopt} อยู่ระหว่างค่า W_{fopt} ของการกระจายที่มีแนวโน้มในการเกิดการค้นถังอย่างไม่มีประสิทธิภาพ และรายการเหตุการณ์ขนาดใหญ่ ทำให้ค่า W_f ของการกระจายแบบนี้สามารถนำไปใช้เป็นค่า W_f ของการกระจายแบบอื่น ๆ ได้เพราะไม่ทำให้ต้องใช้เวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่าเพิ่มขึ้นเท่าไรนักเมื่อนำไปใช้การกระจายแบบอื่น ซึ่งการเลือกค่า W_f ที่สามารถนำไปใช้ได้กับการกระจายทุกแบบจะกล่าวถึงในส่วนถัดไป



รูปที่ 3.6 ความสัมพันธ์ระหว่างค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่ากับตัวคูณความกว้าง การกระจายแบบสองฐานนิยม 2 $\mu = 1$, $p = 0.55$

ผลการทดลองที่ผ่านมาแสดงให้เห็นว่า แม้จะเลือก τ_i ที่มีการกระจายแบบใดมาทดสอบก็ตาม ความสัมพันธ์ระหว่างเวลาที่ใช้ในการประมวลผลการดำเนินการเหตุการณ์คงค่ากับตัวคูณความกว้างก็ยังคงมีลักษณะเหมือนเดิม คือ เป็นรูปถ้วยที่มีก้นถ้วยกว้าง แต่จะแตกต่างกันที่ตำแหน่งของก้นถ้วยว่าจะเลื่อนไปอยู่ด้านซ้ายหรือด้านขวาของรูปเท่านั้น การที่กราฟเลื่อนไปซ้ายหรือขวานี้ก็คือการเลื่อนค่า W_{fopt} นั่นเอง ในรูปที่ 3.7 แสดงการเปรียบเทียบค่าอัตราส่วนการสูญเสีย (loss ratio (r_l), ใช้สมการที่ (3.5) ใน



รูปที่ 3.7 อัตราส่วนการสูญเสียของการกระจายแบบต่าง ๆ

การคำนวณ) ของการกระจายทุกแบบที่ทดลองมา เมื่อ $N = 2100$ ในรูปที่ 3.7 แสดงให้เห็นว่าการกระจายต่าง ๆ ที่นำมาทดสอบจะมีค่า W_{fopt} แตกต่างกันไป ดังนั้นถ้าเรารู้ค่า W_{fopt} ที่เหมาะสมสำหรับการกระจายแต่ละแบบ และสามารถเลือกใช้ได้อย่างเหมาะสมก็จะสามารถทำให้แถวคอยปฏิทินมีอัตราส่วนการสูญเสียต่ำที่สุดได้ แต่ในความเป็นจริงแล้วเมื่อโปรแกรมจำลองกำลังทำงานอยู่ เราไม่สามารถรู้ได้เลยว่าขณะนั้นการกระจายที่ใช้เป็นการกระจายแบบใด ทำให้เราไม่สามารถเลือกค่า W_{fopt} ที่เหมาะสมมาใช้งานได้ ดังนั้นจึงทำได้แค่พยายามเลือกค่า W_f ที่ส่งผลให้แถวคอยปฏิทินมีค่าอัตราส่วนการสูญเสียต่ำที่สุดมาใช้ แต่เนื่องจากค่า W_f ที่ส่งผลให้แถวคอยปฏิทินมีค่าอัตราส่วนการสูญเสียใกล้เคียง 0 นั้นมีอยู่หลายค่าด้วยกัน ทำให้เกิดเป็นช่วงของค่า W_f ที่สามารถนำมาใช้งานได้ (จะเรียกว่าตัวคูณความกว้างเกือบเหมาะสม (near-optimum width factor W_{fn-opt})) และช่วงของ W_{fn-opt} นี้เป็นช่วงที่กว้างมาก (อัตราส่วนระหว่าง W_{fn-opt} ที่ต่ำที่สุดและสูงที่สุดที่สามารถเลือกมาใช้งานได้มีค่าเป็นจำนวนเท่าของ 10) อีกทั้งการกระจายทุกรูปแบบที่นำมาทดสอบยังมีช่วงของ W_{fn-opt} คาบเกี่ยวกันอยู่ ดังนั้นจึงใช้วิธีเลือกค่า W_{fn-opt} ค่าหนึ่ง ซึ่งสามารถใช้ได้กับทุกการกระจายมาใช้ ซึ่งก็เพียงพอแล้วที่จะทำให้แถวคอยปฏิทินมีการทำงานด้วยความเร็วใกล้เคียงกับความเร็วสูงสุด ซึ่งในวิทยานิพนธ์ฉบับนี้เสนอให้เลือกใช้ค่า $W_f = 3$ ในการกำหนดความกว้างถัง (เลือกจากจุดที่มีอัตราส่วนการสูญเสียต่ำกว่า 0.1 ในรูปที่ 3.7)

$$r_l = \frac{(h - h_{min})}{h_{min}} \tag{3.5}$$

โดยที่ r_l คืออัตราส่วนการสูญเสีย

h คือเวลาประมวลผลการดำเนินการเหตุการณ์คงค่า

h_{min} คือเวลาประมวลผลการดำเนินการเหตุการณ์คงค่า

3.2 แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ

แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ (Adaptive Bucket width Calendar Queue: ABCQ) เป็นการปรับปรุงการทำงานของแถวคอยปฏิทินแบบดั้งเดิมโดยการแก้ไขวิธีหาค่าความกว้างถังจากการใช้ค่าเฉลี่ยของเวลาระหว่างบันทึกเหตุการณ์แต่ละใบ มาเป็นการใช้ค่าตัวคูณความกว้างในการกำหนดความกว้างถัง นอกจากการปรับปรุงวิธีการเลือกค่าความกว้างถังแล้ว แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติยังเพิ่มการทำงานอัตโนมัติเข้าไปอีกส่วนหนึ่ง คือการกระตุ้นให้เกิดการปรับความกว้างถังเมื่อพารามิเตอร์ของการกระจายเวลาเพิ่มมีค่าเปลี่ยนแปลงไป โดยที่ไม่ต้องรอให้จำนวนบันทึกเหตุการณ์มีการเปลี่ยนแปลง (โปรแกรมจำลองทำงานในสภาวะอยู่ตัว)

แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัตินี้จะนำสมการที่ (3.4) มาใช้หาค่าความกว้างถัง โดยใช้ $W_f = 3$, $N =$ จำนวนบันทึกเหตุการณ์ในถัง และ $\mu =$ ค่าเฉลี่ยของค่าเวลาเพิ่ม ซึ่งการหาค่าเฉลี่ยที่ใช้สำหรับประมาณค่า μ นี่จะต้องคำนวณทุกครั้งที่มีการใส่บันทึกเหตุการณ์ (enqueue) เข้าไปในแถวคอยปฏิทิน แต่จะถูกนำไปใช้เมื่อเกิดกระบวนการเปลี่ยนขนาดของแถวคอยปฏิทิน โดยเราสามารถใช้โปรแกรมที่เขียนสำหรับแถวคอยปฏิทินแบบดั้งเดิมมาเปลี่ยนในส่วนการกำหนดความกว้างถังโดยเปลี่ยนจาก การหาค่าเฉลี่ยของเวลาระหว่างบันทึกเหตุการณ์ มาเป็นการใช้สมการที่ (3.4) คำนวณแทน และเพิ่มการคำนวณค่าเฉลี่ยเวลาเพิ่ม (สำหรับประมาณค่า μ) เข้าไปในขั้นตอนการใส่บันทึกเหตุการณ์เท่านั้น ซึ่งรหัสเทียม (psudo code) ของแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติได้แสดงไว้ในภาคผนวก ก

ดังได้กล่าวไปแล้วว่าแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ จะเปลี่ยนแปลงการทำงานของแถวคอยปฏิทินแบบดั้งเดิม 2 อย่าง อย่างแรก คือ เปลี่ยนวิธีหาค่าความกว้างถังซึ่งได้กล่าวไปแล้ว อีกส่วนหนึ่งคือกระบวนการอัตโนมัติที่มีเพื่อกระตุ้นให้เกิดการเปลี่ยนค่าความกว้างถังเมื่อสภาวะการจำลองเปลี่ยนแปลงไป ซึ่งการทำงานอัตโนมัติส่วนนี้ก็คือ การควบคุมค่า W_f ให้อยู่ในช่วงของ W_{fn-opt} โดยเราจะคำนวณค่า W_f ด้วยสมการที่ (3.4) โดยต้องจัดรูปสมการที่ (3.4) ใหม่เพื่อใช้ในการหา W_f

($W_f = \delta N/\mu$) โดยหลังจากที่เราเพิ่งปรับค่าความกว้างถังไปใหม่ ๆ นั้น W_f จะมีค่าเป็น 3 เสมอ เพราะยังไม่มีตัวแปรใดเปลี่ยนแปลงไปเลย แต่หลังจากที่ทำงานไปได้ซักครู่หนึ่งค่าเฉลี่ยเวลาเพิ่มจะมีค่าเปลี่ยนแปลงไป (ไม่มากก็น้อย) ทำให้ W_f ที่คำนวณได้มีค่าไม่เท่ากับ 3 เพราะยังใช้ความกว้างเดิมอยู่ ดังนั้นจึงต้องทำการปรับค่าความกว้างถังเพื่อควบคุมให้ $W_f = 3$ อยู่ตลอดเวลา ดังนั้นเราจึงใช้ค่า W_f นี้เป็นเงื่อนไขในการกระตุ้นให้เกิดการเปลี่ยนความกว้างถัง กล่าวคือเมื่อค่า W_f เปลี่ยนแปลงไปก็จะทำการเปลี่ยนค่าความกว้างถังเพื่อให้ W_f มีค่าเท่าเดิมตลอดเวลา แต่เนื่องจากค่า W_f ที่คำนวณได้มีค่าเปลี่ยนแปลงตลอดเวลา การที่จะเปลี่ยนความกว้างถังอยู่ตลอดเวลาเพื่อให้ W_f มีค่าเท่าเดิมกลับทำให้แถวคอยปฏิทินทำงานได้ช้าลง เพราะต้องเสียเวลาในการสร้างปฏิทินใหม่และย้ายบันทึกเหตุการณ์ไปใส่ไว้ในปฏิทินใหม่ ทำให้การควบคุมให้ค่า $W_f = 3$ อยู่ตลอดเวลานั้นเป็นการกระทำที่ไม่สมเหตุผล แต่จากการศึกษาในหัวข้อที่ 3.1 แสดงให้เห็นว่าการเลือกค่า W_f ผิดไปจาก W_{fopt} เพียงเล็กน้อยไม่ทำให้เกิดผลเสียหายเท่าไร (มีค่าอัตราส่วนการสูญเสียต่ำ) ดังนั้นเราจึงใช้วิธีกำหนดช่วงควบคุมสำหรับ W_f แทนการกำหนดค่าควบคุมสำหรับ W_f โดยจะไม่กระตุ้นให้เกิดการเปลี่ยนความกว้างถังหาก W_f ที่คำนวณได้ยังมีค่าอยู่ในช่วงนี้ ในวิทยานิพนธ์ฉบับนี้เสนอให้เลือกใช้ช่วง $[0.5, 10]$ เป็นช่วงควบคุมสำหรับค่า W_f (เนื่องจากมีค่าอัตราส่วนการสูญเสีย < 0.4) การเขียนโปรแกรมส่วนนี้เพิ่มเข้าไปนั้นจะไปเพิ่มหลังจากที่เราคำนวณค่าเฉลี่ยเวลาเพิ่ม เพราะค่าเฉลี่ยเวลาเพิ่มนี้เองเป็นตัวแปรที่ใช้แสดงการเปลี่ยนแปลงของการกระจายเวลาเพิ่ม ซึ่งเป็นสาเหตุที่ทำให้ต้องมีการปรับค่าความกว้างถัง

3.3 สรุป

บทนี้ได้นำเสนอวิธีการเลือกค่าความกว้างถังที่เหมาะสมและสามารถนำไปใช้ได้จริงในทางปฏิบัติ เพื่อนำไปใช้ในแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัตินี้ที่สามารถทนทานต่อสภาพการจำลองที่หลากหลายได้อย่างมีประสิทธิภาพ โดยมีพื้นฐานการทำงานอยู่ที่การควบคุมค่าตัวคูณความกว้างให้อยู่ในช่วงทำงานที่กำหนดเอาไว้ อีกทั้งยังมีเทคนิคการกระตุ้นให้เกิดการปรับค่าความกว้างถังโดยที่จำนวนเหตุการณ์ไม่จำเป็นต้องมีค่าเปลี่ยนแปลงไป โดยในการเขียนโปรแกรมนั้นสามารถนำโปรแกรมของแถวคอยปฏิทินที่มีอยู่แล้วมาปรับเปลี่ยนอีกเล็กน้อยก็จะกลายเป็นโปรแกรมของแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติได้ เพราะมีการเพิ่มตัวแปร และการทำงานเพียงเล็กน้อยเท่านั้น ซึ่งรหัสเทียมของแถวคอยปฏิทินทั้ง 2 วิธีได้แสดงไว้ใน ภาคผนวก ก

บทที่ 4

ผลการทดสอบ

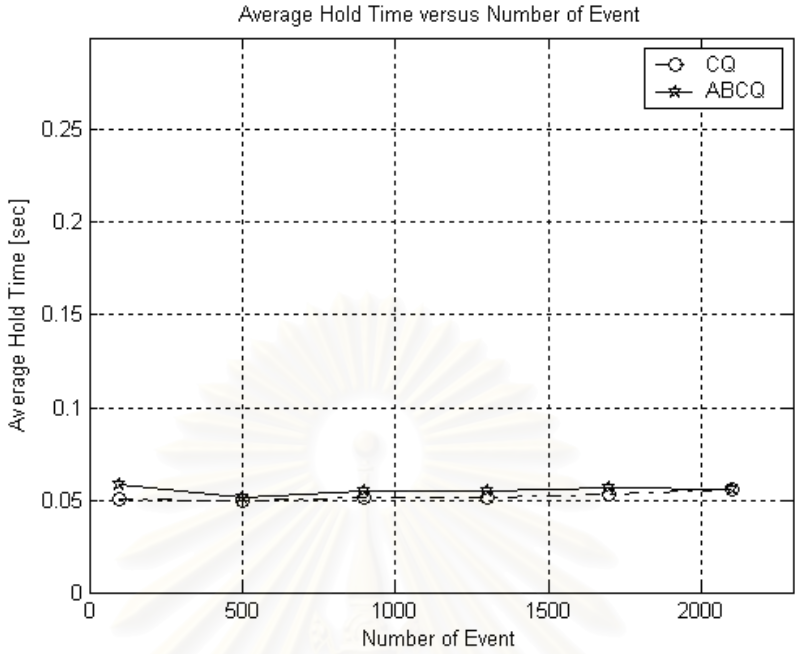
บทนี้จะแสดงผลการทดสอบประสิทธิภาพการทำงานของแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติเปรียบเทียบกับแถวคอยปฏิทินแบบดั้งเดิม โดยจะพิจารณาทั้งในกรณีที่ค่าพารามิเตอร์ของการกระจายเป็นค่าคงที่ตลอดการจำลอง (ระบบที่เป็น stationary) และกรณีที่ค่าพารามิเตอร์ของการกระจายมีค่าเปลี่ยนแปลงไประหว่างการจำลอง (ระบบที่เป็น non-stationary) โดยจะใช้การดำเนินการเหตุการณ์คงค่าในการเปรียบประสิทธิภาพการทำงานของแถวคอยปฏิทินทั้ง 2 ชนิด นอกจากนั้นยังแสดงผลการทดสอบประสิทธิภาพการทำงานของแถวคอยทั้ง 2 ชนิด เมื่อนำไปใช้งานกับโปรแกรมจำลองจริง

4.1 การทดสอบด้วยการดำเนินการเหตุการณ์คงค่า

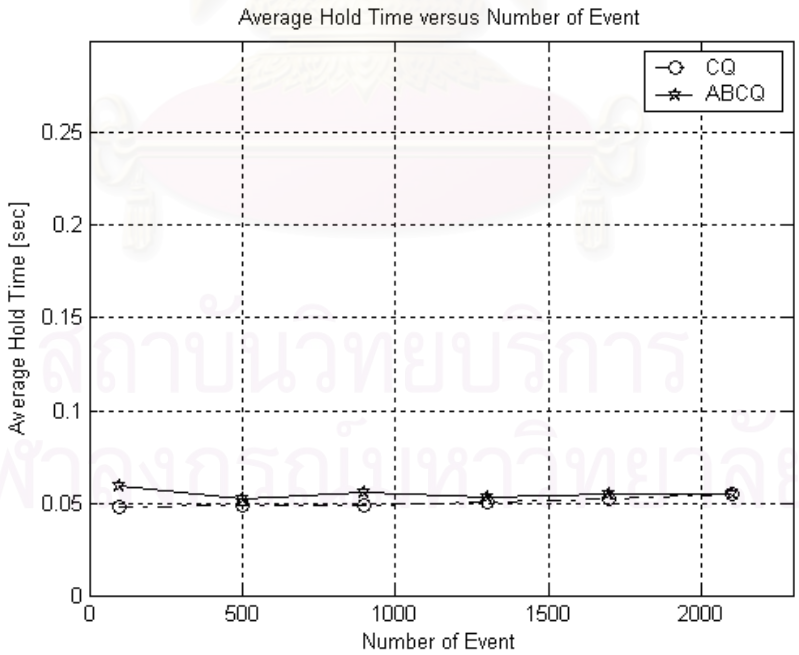
ในการทดลองนี้จะใช้การดำเนินการเหตุการณ์คงค่า (หัวข้อที่ 2.2) ในการวัดประสิทธิภาพการทำงาน โดยมี ค่าอัตราการสร้าง (p) = 0.55 จำนวนบันทึกเหตุการณ์ (N) = 100, 500, 900, 1400, 1700 และ 2100 ใบ และค่าเวลาเพิ่ม (τ_i) มีการกระจายแบบ ก. ข. ค. ง. จ. (เหมือนที่ใช้ในบทที่ 3) เพื่อทดสอบการทำงานของแถวคอยปฏิทินในกรณีที่พารามิเตอร์ของการกระจายเวลาเพิ่มมีค่าคงที่ตลอดการจำลอง และเพิ่มการกระจายสำหรับทดสอบเข้าไปอีก 4 แบบคือ การกระจายแบบเลขชี้กำลังที่มีค่าพารามิเตอร์ (ค่าเฉลี่ย) เปลี่ยนแปลงไประหว่างขั้นตอนการสร้างแถวคอย และขั้นตอนการประมวลผลการดำเนินการเหตุการณ์คงค่า โดยจะเลือกเปลี่ยนค่าเฉลี่ย 4 รูปแบบดังนี้ เปลี่ยนจาก 1 เป็น 10, เปลี่ยนจาก 1 เป็น 100, เปลี่ยนจาก 10 เป็น 1 และเปลี่ยนจาก 100 เป็น 1 การทดลอง 4 แบบนี้สร้างขึ้นเพื่อทดสอบการทำงานในกรณีที่ค่าพารามิเตอร์ของการกระจายเวลาเพิ่มเปลี่ยนแปลงไประหว่างการจำลอง

เครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบ	
หน่วยประมวลผลกลาง(CPU)	AMD Athlon XP1600+
หน่วยความจำ(RAM)	512 MB
ระบบปฏิบัติการ	Windows XP

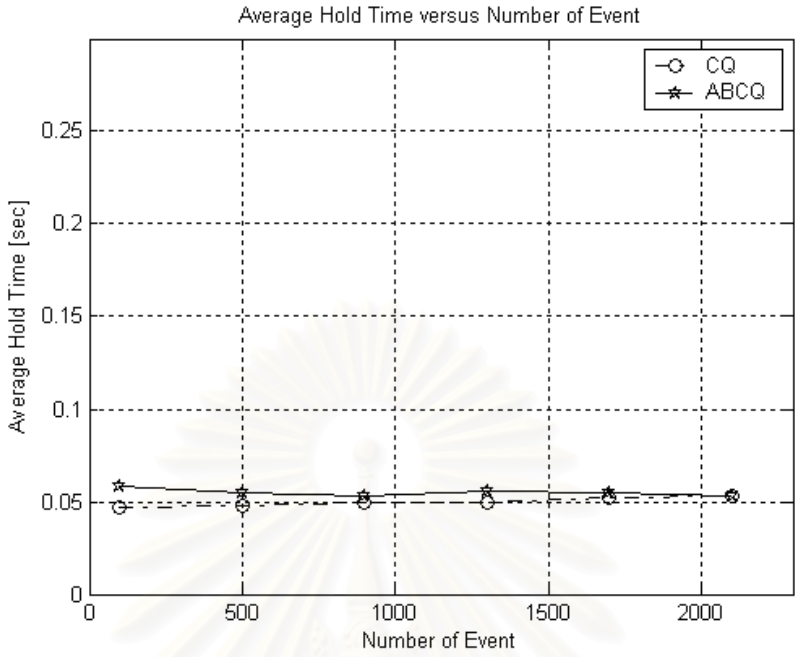
รูปที่ 4.1, 4.2, 4.3 แสดงค่าเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่าเฉลี่ย ในกรณีที่ค่าพารามิเตอร์ของการกระจายมีค่าไม่เปลี่ยนแปลงระหว่างการจำลอง ผลการทดลองได้แสดงให้เห็นอย่างชัดเจนว่า เวลาที่ใช้ประมวลผลการดำเนินการเหตุการณ์คงค่าของแถวคอยปฏิทินทั้ง 2 วิธีนั้นไม่มีความแตกต่างกันเท่าไร เพราะความกว้างถังที่คำนวนได้จากวิธีหาค่าเฉลี่ย (แถวคอยปฏิทินแบบดั้งเดิม)



รูปที่ 4.1 ค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่า เมื่อทดสอบด้วยการกระจายแบบเลขชี้กำลัง $\mu = 1, p = 0.55$



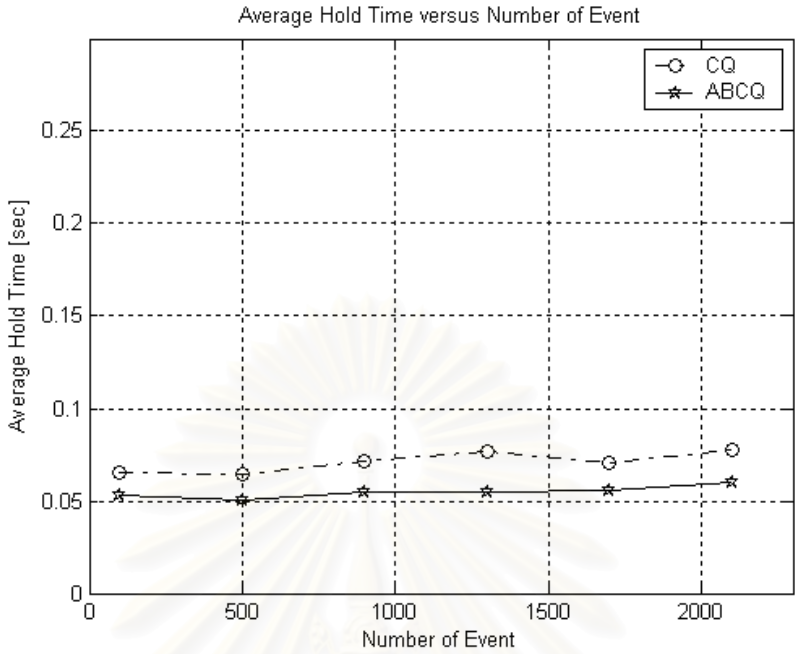
รูปที่ 4.2 ค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่า เมื่อทดสอบด้วยการกระจายแบบสามเหลี่ยม $\mu = 1, p = 0.55$



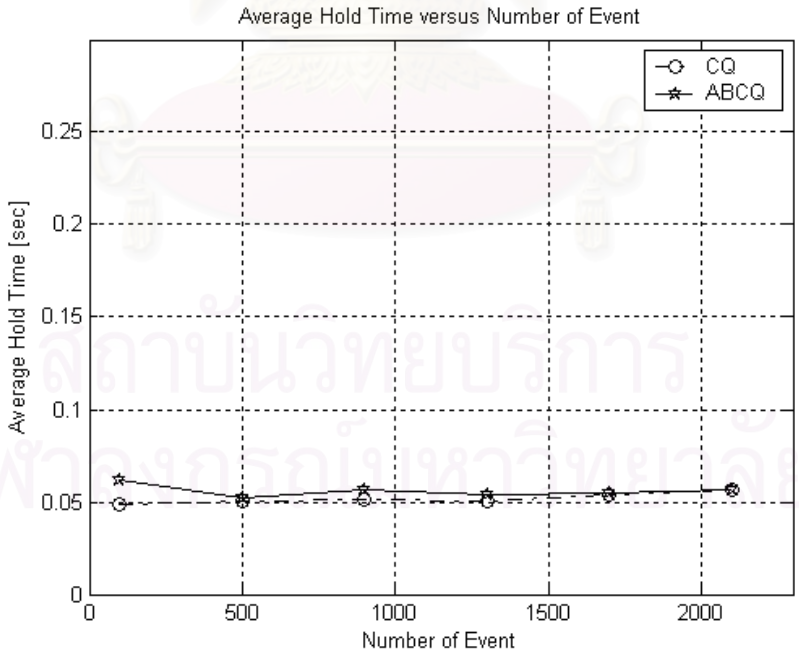
รูปที่ 4.3 ค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่า เมื่อทดสอบด้วยการกระจายแบบสามเหลี่ยมกลับด้าน $\mu = 1, p = 0.55$

กับความกว้างถังที่คำนวณจากค่าตัวคูณความกว้าง (แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ) มีค่าใกล้เคียงกัน แต่ถ้าสังเกตให้ดีจะพบว่าแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติจะใช้เวลามากกว่าแถวคอยปฏิทินแบบดั้งเดิมอยู่เล็กน้อย เนื่องจากแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติจะเพิ่มการทำงานในส่วนของการควบคุมค่าตัวคูณความกว้างเข้าไปทำให้ต้องเสียเวลาในการคำนวณ และเปรียบเทียบค่าตัวคูณความกว้าง แต่อย่างไรก็ตามความสูญเสียนี้นับได้ว่าน้อยมากเมื่อเปรียบเทียบกับความทนทานที่เพิ่มขึ้นในกรณีที่ค่าพารามิเตอร์เปลี่ยนแปลงไประหว่างการจำลอง (รูปที่ 4.6, 4.7, 4.8, 4.9)

รูปที่ 4.4 และ 4.5 เป็นการศึกษาเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่าเฉลี่ยในกรณีที่ค่าพารามิเตอร์ของการกระจายมีค่าไม่เปลี่ยนแปลง และการกระจายที่ใช้เป็นการกระจายแบบสองฐานนิยม เนื่องจากการใช้งานจริงในโปรแกรมจำลอง การกระจายที่ใช้สร้างเวลาเพิ่มของบันทึกเหตุการณ์เป็นการกระจายที่เกิดขึ้นจากเหตุการณ์หลายประเภท ซึ่งเหตุการณ์แต่ละประเภทก็มีชนิดและค่าพารามิเตอร์ของการกระจายแตกต่างกันไป จึงใช้การกระจายแบบสองฐานนิยมมาเป็นตัวแทนการทำงานในสถานการณ์ดังกล่าว ผลการทดลองที่แสดงในรูปที่ 4.4 เป็นกรณีที่มีค่าความแปรปรวนน้อย ส่วนในรูปที่ 4.5 มีความแปรปรวนสูงกว่า จะเห็นว่าในกรณีที่การกระจายมีความแปรปรวนน้อย ทั้งแถวคอยปฏิทิน

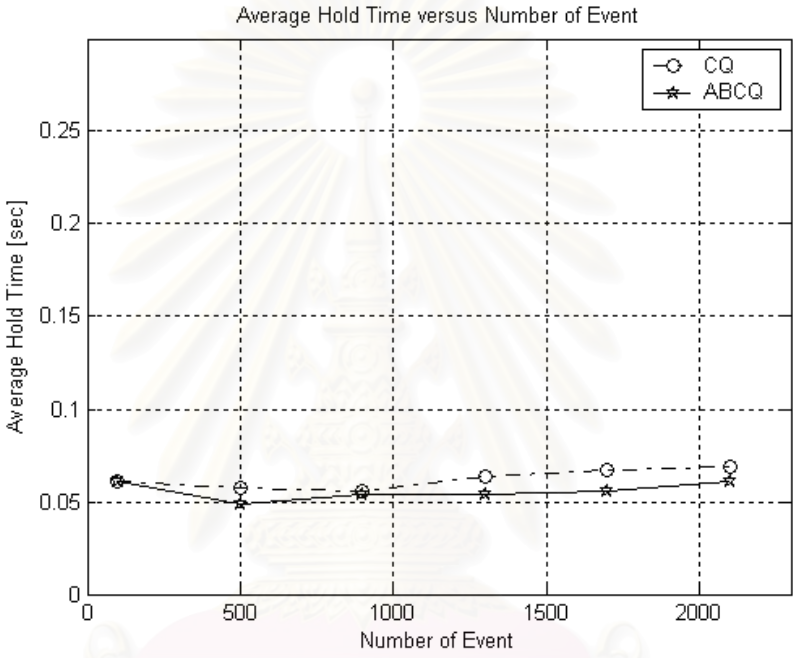


รูปที่ 4.4 ค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่า เมื่อทดสอบด้วยการกระจายแบบสองฐานนิยม 1 $\mu = 1, p = 0.55$



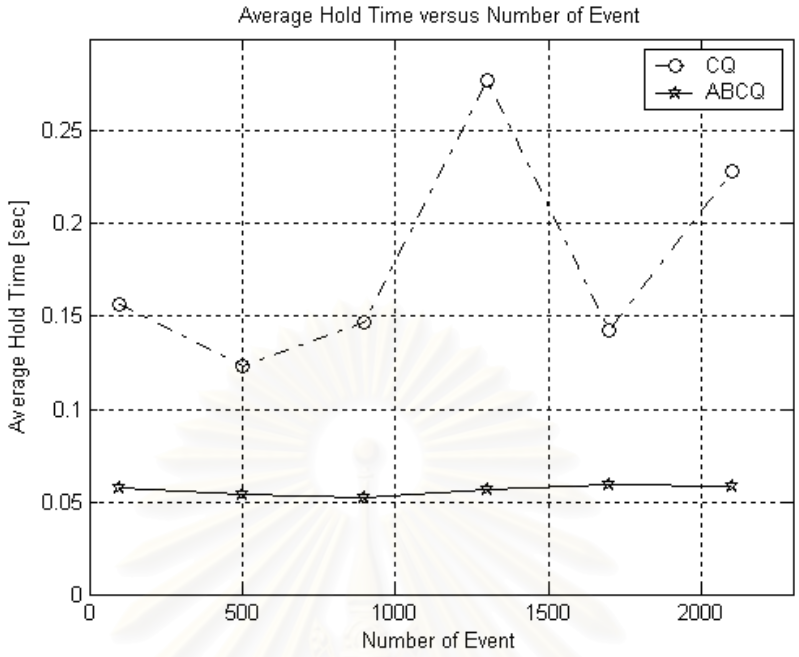
รูปที่ 4.5 ค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่า เมื่อทดสอบด้วยการกระจายแบบสองฐานนิยม 2 $\mu = 1, p = 0.55$

แบบดั้งเดิมและแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติจะยังรักษาระดับความเร็วในการทำงานไว้ได้เท่าเดิม (เปรียบเทียบกับรูปที่ 4.1) เพราะแถวคอยปฏิทินทั้ง 2 ชนิดยังสามารถเลือกความกว้างถังที่เหมาะสมมาใช้ได้ แต่เมื่อการกระจายมีค่าความแปรปรวนสูงขึ้น แถวคอยปฏิทินแบบดั้งเดิมจะมีการทำงานที่ช้าลง เพราะแถวคอยปฏิทินแบบดั้งเดิมไม่สามารถคำนวณค่าความกว้างได้อย่างเหมาะสม แตกต่างกับแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติที่สามารถคำนวณค่าความกว้างถังได้ดีกว่า ทำให้สามารถรักษาระดับความเร็วในการทำงานไว้ได้



รูปที่ 4.6 ค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่า เมื่อทดสอบด้วยการกระจายแบบเลขชี้กำลัง μ เปลี่ยนจาก 1 เป็น 10, $p = 0.55$

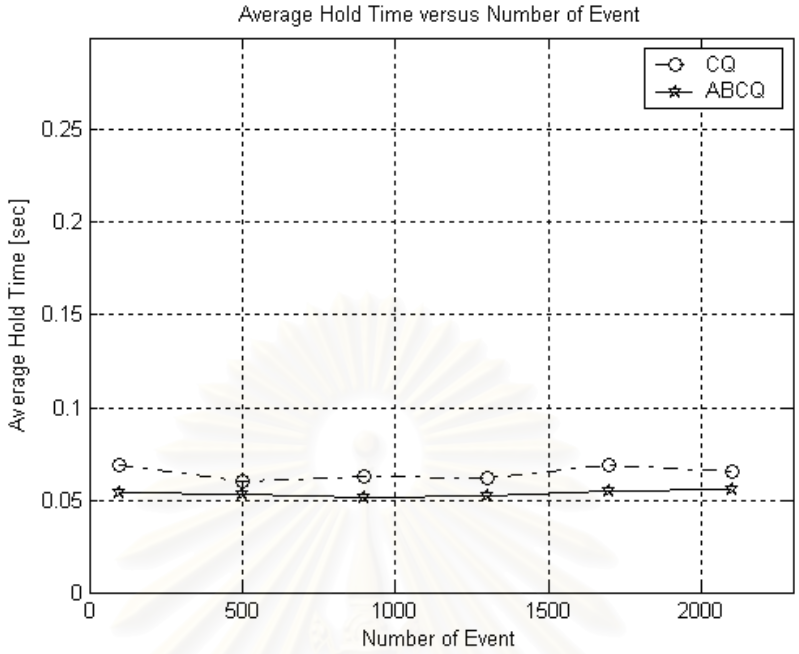
รูปที่ 4.6 และ 4.7 เป็นการศึกษาเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่าเฉลี่ย ในกรณีที่ค่าพารามิเตอร์ของการกระจายมีค่าเปลี่ยนแปลงไประหว่างการจำลอง การกระจายที่เลือกใช้คือการกระจายแบบเลขชี้กำลัง ซึ่งเปลี่ยนค่าเฉลี่ยที่ใช้ในช่วงการสร้างแถวคอย และช่วงการประมวลผลการดำเนินการเหตุการณ์คงค่า โดยจะเปลี่ยนจาก 1 เป็น 10 และจาก 1 เป็น 100 ตามลำดับ การทดสอบนี้สร้างขึ้นเพื่อดูผลกระทบของการเปลี่ยนค่าพารามิเตอร์ให้เพิ่มขึ้น ซึ่งการเพิ่มขึ้นของค่าพารามิเตอร์ระหว่างการจำลองนั้นอาจเกิดขึ้นได้ เช่น การจำลองระบบเอทีเอ็มที่ใช้แหล่งกำเนิดสัญญาณแบบเปิดปิด(เปลี่ยนจากสถานะปิดเป็นเปิด) หรือการจำลองการใช้งานระบบโครงข่ายที่มีปริมาณการใช้งานในแต่ละช่วงเวลาไม่เท่ากัน (เปลี่ยนจากน้อยเป็นมาก) ผลการทดลองแสดงให้เห็นว่าแถวคอยปฏิทินชนิดปรับความกว้าง



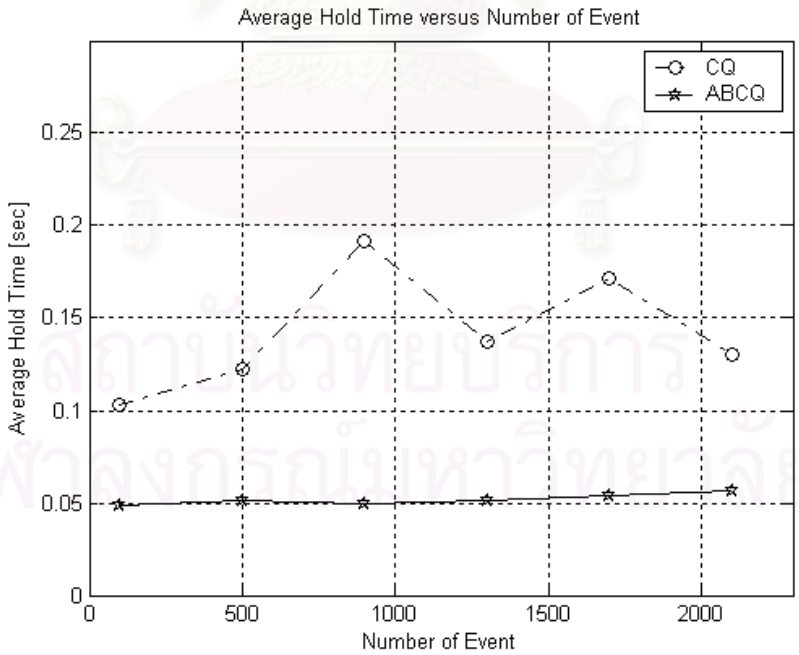
รูปที่ 4.7 ค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่า เมื่อทดสอบด้วยการกระจายแบบเลขชี้กำลัง μ เปลี่ยนจาก 1 เป็น 100, $p = 0.55$

ถังอัตโนมัติจะรักษาระดับความเร็วของการทำงานเอาไว้ได้เท่าเดิมในการทดลองทั้งสองกรณี (เทียบกับรูปที่ 4.1) เนื่องจากแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติสามารถเลือกความกว้างถังได้ดีกว่าและสามารถปรับค่าความกว้างถังได้ถ้าความกว้างถังที่ใช้อยู่มีค่าไม่เหมาะสม ($W_f \notin [0.5, 10]$) ส่วนแถวคอยปฏิทินแบบดั้งเดิมนั้นจะใช้เวลาในการทำงานเพิ่มขึ้น เนื่องจากไม่สามารถเปลี่ยนความกว้างถังให้เหมาะสมกับค่าพารามิเตอร์ที่เปลี่ยนแปลงไปได้

รูปที่ 4.8 และ 4.9 เป็นการศึกษาเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่าเฉลี่ย ในกรณีที่ค่าพารามิเตอร์ของการกระจายมีค่าเปลี่ยนแปลงไประหว่างการจำลอง โดยการกระจายที่เลือกใช้คือการกระจายแบบเลขชี้กำลัง ซึ่งมีการเปลี่ยนค่าเฉลี่ยที่ใช้ในช่วงการสร้างแถวคอย และช่วงการประมวลผลการดำเนินการเหตุการณ์คงค่าจาก 10 เป็น 1 และจาก 100 เป็น 1 ตามลำดับ การทดสอบนี้สร้างขึ้นเพื่อดูผลกระทบของการเปลี่ยนค่าพารามิเตอร์ให้ลดลง ซึ่งการลดลงของค่าพารามิเตอร์ระหว่างการจำลองนั้นอาจเกิดขึ้นได้ เช่น การจำลองระบบเอทีเอ็มที่ใช้แหล่งกำเนิดสัญญาณแบบเปิดปิด (เปลี่ยนจากสถานะเปิดเป็นปิด) หรือการจำลองการใช้งานระบบโครงข่ายที่มีปริมาณการใช้งานในแต่ละช่วงเวลาไม่เท่ากัน (เปลี่ยนจากมากเป็นน้อย) ซึ่งเป็นสถานการณ์ตรงกันข้ามกับการทดลองที่ 4.6 และ 4.7



รูปที่ 4.8 ค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่า เมื่อทดสอบด้วยการกระจายแบบเลขชี้กำลัง μ เปลี่ยนจาก 10 เป็น 1, $p = 0.55$



รูปที่ 4.9 ค่าเฉลี่ยเวลาในการประมวลผลการดำเนินการเหตุการณ์คงค่า เมื่อทดสอบด้วยการกระจายแบบเลขชี้กำลัง μ เปลี่ยนจาก 100 เป็น 1, $p = 0.55$

แม้ว่าการทดลองที่ 4.8 และ 4.9 จะเป็นการทดลองในสถานการณ์ตรงกันข้ามกับการทดลองที่ 4.6 และ 4.7 แต่จากรูปที่ 4.8 และ 4.9 แสดงให้เห็นแล้วว่า ผลการทำงานของแถวคอยปฏิทินทั้ง 2 ชนิด มีลักษณะเหมือนกันกับการรูปที่ 4.6 และ 4.7 คือ แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติยังคงสามารถรักษาระดับความเร็วของการทำงานเอาไว้ได้ในการทดลองทั้งสองกรณี แต่แถวคอยปฏิทินแบบดั้งเดิมจะต้องใช้เวลาในการทำงานเพิ่มขึ้น ซึ่งก็เกิดจากสาเหตุเดียวกันกับการทดลองที่ 4.6 และ 4.7

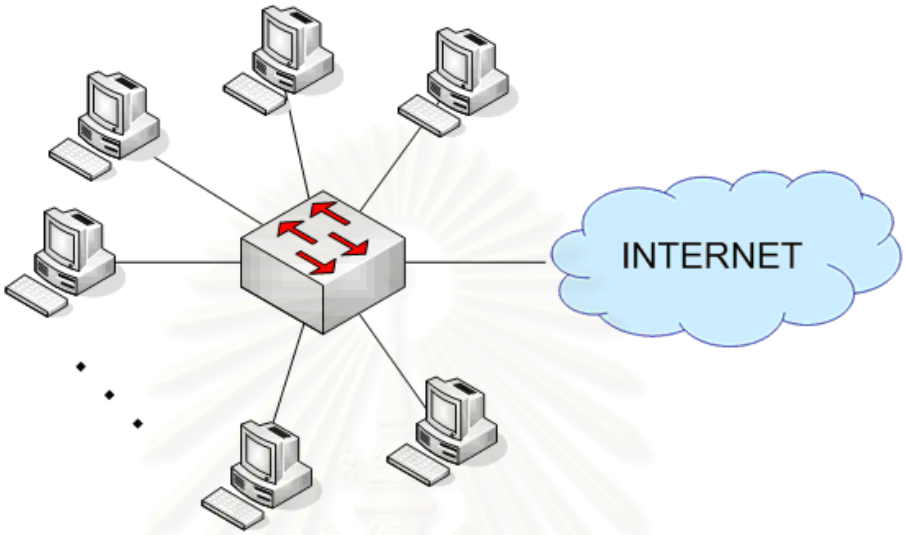
เมื่อนำผลจากการทดลองที่ 4.6, 4.7, 4.8, 4.9 มาพิจารณาร่วมกันเพื่อเปรียบเทียบการทำงานของแถวคอยปฏิทินทั้ง 2 ชนิดในกรณีที่พารามิเตอร์ของการกระจายมีค่าเปลี่ยนแปลงไประหว่างการจำลองพบว่า ในกรณีที่ค่าพารามิเตอร์มีการเปลี่ยนแปลงไม่มาก (จาก 1 เป็น 10 และจาก 10 เป็น 1) แถวคอยปฏิทินแบบดั้งเดิมจะใช้เวลาเพิ่มขึ้นประมาณ 10-20% (เทียบกับรูปที่ 4.1) ส่วนกรณีที่ค่าพารามิเตอร์มีการเปลี่ยนแปลงอย่างรุนแรง (จาก 1 เป็น 100 และจาก 100 เป็น 1) แถวคอยปฏิทินแบบดั้งเดิมจะใช้เวลาเพิ่มขึ้นประมาณ 100-450% (เทียบกับรูปที่ 4.1) แต่แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติจะสามารถรักษาระดับความเร็วในการทำงานเอาไว้ได้ในทุก ๆ กรณี

จาก การ ทดสอบ ประสิทธิภาพ การ ทำงาน ของ แถวคอย ปฏิทิน ชนิด ปรับ ความ กว้าง ถัง อัตโนมัติ เปรียบเทียบกับแถวคอยปฏิทินชนิดดั้งเดิม ด้วยการดำเนินการเหตุการณ์คงค่าพบว่า แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติจะมีประสิทธิภาพการทำงานด้อยกว่าแถวคอยปฏิทินแบบดั้งเดิมอยู่เล็กน้อย เนื่องจากต้องคำนวณค่าตัวคูณความกว้างในทุก ๆ รอบการทำงาน แต่แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติจะสามารถรักษาระดับความเร็วนี้ไว้ได้ในทุก ๆ กรณีที่ทดสอบ ซึ่งต่างกับแถวคอยปฏิทินแบบดั้งเดิมที่จะสามารถรักษาระดับความเร็วไว้ได้เมื่อทดสอบในกรณีที่การกระจายมีฐานนิยมเดียวและค่าพารามิเตอร์ไม่เปลี่ยนแปลงระหว่างการจำลองเท่านั้น แต่เมื่อทดสอบกับกรณีที่การกระจายเป็นแบบสองฐานนิยม หรือการกระจายมีค่าพารามิเตอร์เปลี่ยนแปลงไประหว่างการจำลองแล้ว จะเห็นว่าใช้เวลาในการประมวลผลเพิ่มขึ้นอย่างชัดเจน ดังนั้นแม้ว่าความเร็วของแถวคอยปฏิทินชนิดปรับความกว้างถังจะด้อยกว่าแถวคอยปฏิทินแบบดั้งเดิมอยู่เล็กน้อย (น้อยมากจนเกือบจะเท่ากัน) แต่เมื่อเปรียบเทียบกับความทนทานที่เพิ่มขึ้นแล้ว ถือว่าแถวคอยปฏิทินชนิดปรับความกว้างถังมีประสิทธิภาพในการใช้งานสูงกว่าแถวคอยปฏิทินแบบดั้งเดิม

4.2 การทดสอบด้วยโปรแกรมจำลอง

จาก การ ทดสอบ ด้วย การดำเนินการ เหตุการณ์คงค่า ใน หัวข้อ ที่ผ่าน มา แสดง ให้ เห็น แล้ว ว่า แถวคอย ปฏิทิน ชนิด ปรับ ความ กว้าง ถัง อัตโนมัติ มี ความ ทนทาน ต่อ การ กระจาย สูง กว่า แถวคอย ปฏิทิน แบบ ดั้ง เดิม แต่ เพื่อ ยืนยัน ว่า แถวคอย ปฏิทิน ชนิด เพิ่ม ความ กว้าง ถัง อัตโนมัติ สามารถ ทำงาน ได้ ดี ใน การ ใช้ งาน จริง

ด้วย จึงนำแถวคอยปฏิทินแบบดั้งเดิม และแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติมาทดสอบประสิทธิภาพด้วยโปรแกรมจำลองจริง โดยจะจำลองการใช้งานอินเทอร์เนตของผู้ใช้ 100 คนผ่านสวิตช์ตัวหนึ่ง (ดังแสดงในรูปที่ 4.10)



รูปที่ 4.10 แบบจำลองการใช้อินเทอร์เนต

ตารางที่ 4.1 ผลการทดสอบด้วยโปรแกรมจำลอง

สถานการณ์จำลอง	เวลาในการประมวลผลเหตุการณ์เฉลี่ย		GAIN [%]
	$CQ(ht_{eCQ})$ $[\mu sec.]$	$ABCQ(ht_{eABCQ})$ $[\mu sec.]$	$100 \times (ht_{eCQ} - ht_{eABCQ})/ht_{eCQ}$
1	4.806	4.806	0
2	5.324	4.954	6.9497
3	5.968	5.346	10.4223
4	5.330	5.360	-0.5629
5	5.971	5.451	8.7088

ในการทดลองเราจะแบ่งสถานการณ์การจำลองออกเป็น 5 รูปแบบดังนี้

1. ผู้ใช้ทั้ง 100 คนส่งข้อมูลด้วยอัตราการส่ง 5 แพ็กเกตต่อวินาที สวิตช์มีบัฟเฟอร์ 24000 บิต และสามารถรองรับข้อมูลได้ 1 Mbps

2. ผู้ใช้ทั้ง 100 คนส่งข้อมูลด้วยอัตราการส่ง 5 แพ็กเกตต่อวินาที และมีการปรับเปลี่ยนอัตราการใช้

งานเป็น 50 แพ็กเกตต่อวินาที ทุก ๆ 50 วินาที สวิตช์มีบัฟเฟอร์ 24000 บิต และสามารถรองรับข้อมูลได้ 1 Mbps

3. ผู้ใช้ทั้ง 100 คนส่งข้อมูลด้วยอัตราการส่ง 5 แพ็กเกตต่อวินาที และมีการปรับเปลี่ยนอัตราการใช้งานเป็น 500 แพ็กเกตต่อวินาที ทุก ๆ 50 วินาที สวิตช์มีบัฟเฟอร์ 24000 บิต และสามารถรองรับข้อมูลได้ 1 Mbps

4. แบ่งผู้ใช้เป็นสามกลุ่ม กลุ่มแรกมี 30 คนส่งข้อมูลด้วยอัตราการส่ง 5 แพ็กเกตต่อวินาที ผู้ใช้กลุ่มที่ 2 มี 30 คนส่งข้อมูลด้วยอัตราการส่ง 50 แพ็กเกตต่อวินาที ผู้ใช้กลุ่มที่ 3 มี 40 คนส่งข้อมูลด้วยอัตราการส่ง 500 แพ็กเกตต่อวินาที สวิตช์มีบัฟเฟอร์ 24000 บิต และสามารถรองรับข้อมูลได้ 1 Mbps

5. แบ่งผู้ใช้เป็นสามกลุ่ม กลุ่มแรกมี 30 คนส่งข้อมูลด้วยอัตราการส่ง 5 แพ็กเกตต่อวินาทีและมีการปรับเปลี่ยนอัตราการใช้งานเป็น 500 แพ็กเกตต่อวินาที ทุก ๆ 50 วินาที ผู้ใช้กลุ่มที่ 2 มี 30 คนส่งข้อมูลด้วยอัตราการส่ง 50 แพ็กเกตต่อวินาทีและมีการปรับเปลี่ยนอัตราการใช้งานเป็น 500 แพ็กเกตต่อวินาที ทุก ๆ 50 วินาที ผู้ใช้กลุ่มที่ 3 มี 40 คนส่งข้อมูลด้วยอัตราการส่ง 500 แพ็กเกตต่อวินาที สวิตช์มีบัฟเฟอร์ 24000 บิต และสามารถประมวลผลข้อมูลได้ 1 Mbps

ผลการทดสอบด้วยโปรแกรมจำลองในสถานการณ์จำลองทั้ง 5 สถานการณ์ได้แสดงไว้ในตารางที่ 4.1 จะเห็นว่าในสถานการณ์ที่ 1 แถวคอยปฏิทินทั้งสองแบบสามารถทำงานได้ที่ความเร็วใกล้เคียงกัน เนื่องจากทั้ง 2 สถานการณ์นี้เป็นกรณีที่ไม่มีการเปลี่ยนแปลงค่าพารามิเตอร์ระหว่างการจำลองซึ่งแถวคอยปฏิทินทั้ง 2 แบบสามารถทำงานได้ด้วยความเร็วใกล้เคียงกันเหมือนกับรูปที่ 4.1, 4.2, 4.3 ส่วนกรณีที่ 4 นั้นเป็นการทดสอบในกรณีที่ค่าพารามิเตอร์ที่ใช้มีหลายฐานนิยม คล้ายกับรูปที่ 4.4 และ 4.5 แต่เนื่องจากค่าพารามิเตอร์ของการกระจายไม่ได้แยกกันอย่างชัดเจนจึงมีแนวโน้มจะคล้ายกับรูปที่ 4.5 มากกว่า ซึ่งกรณีนี้ก็จะมีประสิทธิภาพในการทำงานใกล้เคียงกัน แต่ในสถานการณ์ที่ 2, 3 และ 5 ซึ่งมีการเปลี่ยนแปลงค่าพารามิเตอร์ระหว่างการจำลองนั้นจะเห็นได้อย่างชัดเจนว่าการเลือกใช้แถวคอยชนิดปรับความกว้างถังอัตโนมัติสามารถลดระยะเวลาในการประมวลผลของโปรแกรมจำลองลงได้ประมาณ 6-10% ซึ่งก็สอดคล้องกับผลการทดลองที่ได้อธิบายไปแล้วในส่วนการทดสอบด้วยการดำเนินการเหตุการณ์คงค่า (รูปที่ 4.6, 4.7, 4.8, 4.9)

4.3 สรุป

ในบทนี้ได้แสดงผลการทดสอบประสิทธิภาพของแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติและแถวคอยปฏิทินแบบดั้งเดิม โดยใช้กระบวนการการดำเนินการเหตุการณ์คงค่า ผลการทดสอบแสดงให้เห็นอย่างชัดเจนว่า แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติมีความทนทานในการใช้งานมากกว่าแถวคอยปฏิทินแบบดั้งเดิม เนื่องจากแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติมีการเลือกค่าความกว้างถังที่ดีกว่า และยังสามารถปรับค่าความกว้างถังให้เหมาะสมกับการใช้งานได้แม้ปฏิทินยังมีขนาดเท่าเดิม นอกจากนั้นเมื่อนำแถวคอยปฏิทินทั้งสองชนิดไปใช้กับโปรแกรมจำลองจริง พบว่าแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติสามารถทำงานได้ด้วยความเร็วที่เทียบเท่าหรือมากกว่าการจำลองด้วยแถวคอยปฏิทินแบบดั้งเดิมในทุก ๆ สถานการณ์

บทที่ 5

บทสรุป

5.1 บทสรุป

วิทยานิพนธ์ฉบับนี้ได้ศึกษาและปรับปรุงประสิทธิภาพการทำงานของแถวคอยปฏิทิน (calendar queue, CQ) ให้มีความเร็วในการประมวลเพิ่มขึ้น โดยแนวความคิดหลักที่ใช้สำหรับปรับปรุงประสิทธิภาพของแถวคอยปฏิทินมีอยู่ 2 อย่างด้วยกันคือ

1. การเลือกค่าความกว้างถังด้วยค่าตัวคูณความกว้าง (Width factor: W_f) ซึ่งสามารถเลือกค่าความกว้างถังได้ดีกว่าการหาค่าเฉลี่ยเวลาระหว่างเหตุการณ์ที่ใช้อยู่ในแถวคอยปฏิทินแบบดั้งเดิม

2. การกระตุ้นให้มีการเปลี่ยนค่าความกว้างถังโดยไม่จำเป็นต้องรอให้จำนวนบันทึกเหตุการณ์มีการเปลี่ยนแปลง ซึ่งทำให้แถวคอยปฏิทินสามารถปรับค่าความกว้างถังให้เหมาะสมกับสภาวะการทำงานที่เปลี่ยนไปแม้โปรแกรมจำลองจะกำลังทำงานอยู่ในสภาวะอยู่ตัว (steady state)

จากการศึกษาผลกระทบของการเลือกค่าตัวคูณความกว้างที่มีต่อเวลาในการประมวลผลรายการเหตุการณ์พบว่า ค่าตัวคูณความกว้างที่เหมาะสม (W_{fopt}) นั้นหาได้ยากมาก และยังมีค่าเปลี่ยนแปลงไปเมื่อทดสอบกับการกระจายที่แตกต่างกัน ซึ่งในการใช้งานจริงนั้นผู้ใช้ไม่สามารถทราบได้ว่าการกระจายที่เกิดขึ้นในโปรแกรมจำลองเป็นการกระจายแบบใด ทำให้ไม่สามารถเลือกค่า W_{fopt} ที่เหมาะสมได้ แต่จากกราฟในรูปที่ 3.2-3.6 แสดงให้เห็นว่าแม้ผู้ใช้จะไม่ได้เลือกค่า W_{fopt} มาใช้งานก็สามารถทำให้แถวคอยปฏิทินมีความเร็วในการทำงานใกล้เคียงกับความเร็วสูงสุดได้ โดยผู้ใช้เพียงเลือกค่า W_f ให้ใกล้เคียงกับค่า W_{fopt} เท่านั้น

วิทยานิพนธ์ฉบับนี้เสนอแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ (Adaptive Bucket width Calendar Queue: ABCQ) ขึ้นมาโดยมีพื้นฐานการทำงานอยู่ที่การควบคุมค่า W_f ให้อยู่ในช่วงที่กำหนด ซึ่งจะทำให้แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ มีความสามารถในการเลือกค่าความกว้างถังได้ดีขึ้น อีกทั้งยังมีการปรับค่าความกว้างให้เหมาะสมกับสภาวะการทำงานอยู่ตลอดเวลาด้วย

จากการทดสอบพบว่า การเลือกค่าความกว้างถังในแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติมีผลการทำงานดีกว่าการเลือกค่าความกว้างถังที่ใช้ในแถวคอยปฏิทินดั้งเดิม โดยจะแบ่งการทดสอบเป็น 2 ส่วนคือ

1. การทดสอบด้วยการดำเนินการเหตุการณ์คงค่า ซึ่งจะใช้เพื่อทดสอบเฉพาะในส่วนการทำงานของแถวคอยปฏิทินเท่านั้น ผลการปรียบเทียบประสิทธิภาพพบว่า แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติสามารถลดเวลาในการประมวลผลลงได้สูงสุดถึง 75% ในกรณีทดสอบด้วยการกระจายแบบเลขชี้กำลังที่มีการเปลี่ยนค่าเฉลี่ยระหว่างการทดสอบจาก 1 เป็น 100

2. การทดสอบกับโปรแกรมจำลองจริงซึ่งเวลาที่ใช้ประมวลผลจะได้รับผลกระทบจากการทำงานส่วนอื่น ๆ ของโปรแกรมด้วย เช่น การกระจายที่ใช้จริงไม่ใช่การกระจายเดียวกันกับที่ใช้ในการทดสอบด้วยการดำเนินการเหตุการณ์คงค่า การจับเวลาจะต้องจับเวลารวมที่ใช้ในการประมวลผลโปรแกรมทั้งหมดซึ่งจะรวมเวลาในการประมวลผลงานส่วนอื่นนอกจากส่วนการจัดการรายการเหตุการณ์เข้าไปด้วย ผลการเปรียบเทียบประสิทธิภาพในหลาย ๆ สถานการณ์จำลองพบว่า แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติสามารถลดเวลาในการประมวลผลลงได้สูงสุดถึง 10% ในกรณีจำลองระบบการใช้งานอินเทอร์เนตที่ผู้ใช้มีการเปลี่ยนค่าอัตราการส่งข้อมูลจาก 5 แพ็กเกตต่อวินาทีเป็น 500 แพ็กเกตต่อวินาที ทุก ๆ 50 วินาที

จากการทดสอบทั้งสองส่วนสามารถยืนยันได้เป็นอย่างดีว่า แถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ มีความทนทาน สูงกว่าแถวคอยปฏิทินแบบดั้งเดิม ซึ่งสามารถสังเกตได้จากการที่แถวคอยปฏิทินแบบปรับความกว้างถังอัตโนมัติสามารถรักษาระดับความเร็วในการทำงานไว้ได้เท่าเดิมในทุกกรณีที่ทำการทดสอบ ส่วนแถวคอยปฏิทินแบบดั้งเดิมจะทำงานได้ช้าลงเมื่อค่าพารามิเตอร์ของการกระจายมีการเปลี่ยนค่าไประหว่างการจำลอง

5.2 ข้อเสนอแนะ

ในจากงานวิจัยนี้ได้เสนอแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติขึ้นมาเพื่อใช้งานกับโปรแกรมจำลองแบบเหตุการณ์ไม่ต่อเนื่อง ซึ่งแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติที่เสนอขึ้นมานั้นจะจัดการกับบันทึกเหตุการณ์ทั้งหมดในรายการเหตุการณ์อย่างเท่าเทียมกัน แต่จากการศึกษาการเขียนโปรแกรมจำลองพบว่ายังมีบันทึกเหตุการณ์อีกกลุ่มหนึ่งซึ่งใช้ในการควบคุมกระบวนการต่าง ๆ ที่เกิดขึ้นในโปรแกรมจำลอง ซึ่งบันทึกเหตุการณ์เหล่านี้มักจะมีค่าเวลาเพิ่มสูงมากเมื่อเทียบกับบันทึกเหตุการณ์อื่น ๆ ดังนั้นในงานวิจัยต่อ ๆ ไปควรจะมีการศึกษาแถวคอยแบบมีลำดับความสำคัญที่มีการจัดการบันทึกเหตุการณ์เหล่านี้เป็นพิเศษ

รายการอ้างอิง

[1] D. E. Knuth. The Art of Computer Programming. vol.1 3rd edition., Fundamental Algorithms. Addison-Wesley, 1997. chapter 2.

[2] J. H. Blackstone, G. L. Hogg and D. T. Phillips. “A two-list synchronization procedure for discrete event simulation.” Communications of the ACM. Vol. 24, Issue 12, December 1981, pp. 825-829.

[3] M. R. Brown. “Implementation and analysis of binomial queue algorithms.” SIAM Journal on Computing. Vol. 7, No. 3, August 1978, pp. 298-319.

[4] W. R. Franta and K. Maly. “An efficient data structure for the simulation event set.” Communications of the ACM. Vol. 20, Issue 8, August 1977, pp. 596-602.

[5] W. R. Franta and K. Maly. “A comparison of heaps and the TL structure for the simulation event set.” Communications of the ACM. Vol. 21, Issue 10, October 1978, pp. 873-875.

[6] W. M. McCormack, and R. G. Sargent. “Analysis of future event-set algorithm for discrete event simulation.” Communications of the ACM. Vol. 24, Issue 12, December 1981, pp. 801-812.

[7] D. Davey. and J. Vaucher. “Self-optimizing partitioned sequencing sets for discrete event simulation.” INFOR. Vol. 18, No. 1 February 1980, pp. 41-61.

[8] R. Brown. “Calendar queue: A fast O(1) Priority queue implementation for the simulation event set problem.” Communications of the ACM. Vol. 31, Issue 10, October 1988, pp. 1220-1227.

[9] A. M. Law and W. D. Kelton. Simulation modeling and Analysis. 3nd edition. Singapore: McGraw-Hill, 2000.

[10] D. W. Jones. “An empirical comparison of priority-queue and event-set implementations.” Communications of the ACM. Vol. 29, Issue 4, April 1986, pp. 300-311.

[11] K. B. Erickson, R. E. Ladner and A. LaMarca. “Optimizing Static Calendar Queue.” 35th Annual IEEE symposium on Foundations of Computer Science, November 1994, pp. 732-743.

ภาคผนวก

ภาคผนวก ก

รหัสเทียม

ก.1 รหัสเทียมของแถวคอยปฏิทินแบบดั้งเดิม

แถวคอยปฏิทินนั้นสามารถแบ่งการทำงานออกเป็นส่วน ๆ ได้ 5 ส่วนดังนี้

1. การใส่บันทึกเหตุการณ์ (enqueue)
2. การถอดบันทึกเหตุการณ์ (dequeue)
3. การกำหนดค่าตั้งต้น (initialisation)
4. การเปลี่ยนขนาด (resize)
5. การคำนวณค่าความกว้างถัง (newwidth)

ก.1.1 การใส่บันทึกเหตุการณ์

```
enqueue(entry, priority)
double priority;
struct eventtype *entry;
/* This adds one entry to the queue. */
{
int i;
/* Calculate the number of the bucket in which to place the new entry. */
i = priority/width; / * Find virtual bucket. */
i = i%nbuckets; /* Find actual bucket. */
Insert entry into bucket i in sorted list;
++qsize; / * Update record of queue size. */
/* Double the calendar size if needed. */
if (qsize>top_threshold) resize(2*nbuckets);
}
```

ก.1.2 การถอดบันทึกเหตุการณ์

```
struct eventtype *dequeue( )
/* This removes the lowest priority event from the
queue and returns a pointer to the node containing it. */
{
int i;
if (qsize == 0) return(NULL);
for (i = lastbucket; ; ) /* Search buckets */
{
/* Check bucket i */
if (bucket[i] != NULL && bucket[i]->prio < buckettop)
{
/* Item to dequeue has been found. */
Remove item from list;
/ * Update position on calendar. */
lastbucket = i; lastprio = priority of item removed;
−−qsize;
/* Halve calendar size if needed. */
if (qsize < bot_threshold) resize(nbuckets/2);
return item found:
}
else
{
/* Prepare to check next bucket or else go to a direct search. */
++i; if (i == nbuckets) i = 0;
buckettop += width;
if (i == lastbucket) break; /* Go to direct search */
}
/* Directly search for minimum priority event. */
Find lowest priority by examining first event of each bucket;
```

```
Set lastbucket, lastprio, and buckettop for this event;
return(dequeue( ); / * Resume search at minnode. */
}
```

ก.1.3 การกำหนดค่าตั้งต้น

```
localinit(qbase, nbuck, bwidth, startprio)
{
int qbase, nbuck;
double bwidth, startprio;
/* This initializes a bucket array within the array a[]. Bucket width is set equal to
bwidth.
Bucket[0] is made equal to a[qbase]; and the number of buckets, is nbuck.
Startprio is the priority at which dequeueing begins.
All external variables except resizeenabled are initialized. */
int i;
long int n;
/* Set position and size of new queue. */
firstsub = qbase;
bucket = pointer to start of bucket[] in a[];
width = bwidth;
nbuckets = nbuck;
Calculate bit mask for modulo nbuckets operation;
/* Initialize as empty. */
qsize = 0;
for(i = 0; i < nbuckets; ++i) bucket[i] = NULL;
/* Set up initial position in queue. */
Lastprio = startprio;
n = startprio/width; /* Virtual bucket */
lastbucket = n%nbuckets;
buckettop = (n+1)*width+0.5*width;
```

```
/* Set up queue size change thresholds. */
bot_threshold = nbuckets/2-2;
top_threshold = 2*nbuckets;
}/* end */
```

การสร้างปฏิทินเปล่าจะสร้างโดยใช้ฟังก์ชั่น initqueue( )

```
initqueue( )
{
/* This initializes an empty queue. */
localinit(0, 2, 1.0, 0.0);
resizeenabled = TRUE;
}
```

ก.1.4 การเปลี่ยนขนาด

```
resize(newsize)
int newsize;
/* This copies the queue onto a calendar with newsize buckets.
The new bucket array is on the opposite end of the array a[QSPACE] from the
original. */
{
double bwidth;
int i;
int oldnbuckets;
struct eventtype ** oldbucket;
if (!resizeenabled) return;
bwidth = newwidth( ); /* Find new bucket width. */
/* Save location and size of old calendar for use when copying calendar. */
bldbucket = bucket; oldnbuckets = nbuckets;
/* Initialize new calendar. */
if (firstsub == 0)
localinit(QSPACE-newsize, newsize, bwidth, lastprio);
```

```
else
localinit(0, newsize, bwidth, lastprio);
/* Copy queue elements to new calendar. */
for (i = oldnbuckets-1; i >= 0; −−i)
Transfer elements from bucket i to new calendar
by enqueueing them;
} /* end */
```

ก.1.5 การคำนวณค่าความกว้างถัง

```
double newwidth( )
{
/* This calculates the width to use for buckets. */
int nsamples;
/* Decide how many queue elements to sample. */
if (qsize < 2) return(1.0);
if (qsize <= 5)
nsamples = qsize;
else
nsamples = 5+qsize/10;
if (nsamples > 25) nsamples = 25;
Record lastprio, lastbucket, buckettop;
Dequeue nsample events from the queue and record
their priorities with resizeenabled equal to FALSE;
Restore the sampled events to the queue using enqueue( );
Restore lastprio, lastbucket, and buckettop;
Calculate average separation of sampled events;
Recalculate average using only separations smaller than twice the original average;
return(3.0 times final average);
} /* end */
```

ก.2 รหัสเทียมของแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติ

เนื่องจากแถวคอยปฏิทินชนิดปรับความกว้างถังอัตโนมัติมีการทำงานส่วนใหญ่คล้ายกับแถวคอยปฏิทินแบบดั้งเดิม แต่จะไม่มีขั้นตอนการคำนวณค่าความกว้างถังเนื่องจากใช้ W_f ในการกำหนดค่าความกว้างถัง และมีขั้นตอนการถอดบันทึกเหตุการณ์เหมือนเดิม ในหัวข้อนี้จะแสดงเฉพาะส่วนที่มีการทำงานเปลี่ยนแปลงไปตามรายการดังนี้ (สิ่งที่เพิ่มหรือแก้ไขจากรหัสเทียมของแถวคอยปฏิทินแบบดั้งเดิมจะแสดงเป็นตัวเอียงขีดเส้นใต้)

1. การใส่บันทึกเหตุการณ์ (enqueue)
2. การกำหนดค่าตั้งต้น (initialisation)
3. การเปลี่ยนขนาด (resize)

ก.2.1 การใส่บันทึกเหตุการณ์

```
enqueue(entry, priority)
double priority;
struct eventtype *entry;
/* This adds one entry to the queue. */
{
int i;
/* Calculate the number of the bucket in which to place the new entry. */
i = priority/width; / * Find virtual bucket. */
i = i % nbuckets; /* Find actual bucket. */
Insert entry into bucket i in sorted list;
++qsize; / * Update record of queue size. */
Calculate average incremental time from entry's incremental time;
Calculate current width factor;
if (current width factor < 0.5 || current width factor > 2) resize(nbuckets);
/* Double the calendar size if needed. */
if (qsize > top_threshold) resize(2*nbuckets);
}
```

ก.2.2 การกำหนดค่าตั้งต้น

```
localinit(qbase, nbuck, bwidth, startprio)
{
int qbase, nbuck;
double bwidth, startprio;
/* This initializes a bucket array within the array a[]. Bucket width is set equal to
bwidth.
Bucket[0] is made equal to a[qbase]; and the number of buckets, is nbuck. Startprio is
the
priority at which dequeueing begins. All external variables except resizeenabled are
initialized. */
Initial average incremental time to 0;
int i;
long int n;
/* Set position and size of new queue. */
firstsub = qbase;
bucket = pointer to start of bucket[] in a[];
width = bwidth;
nbuckets = nbuck;
Calculate bit mask for modulo nbuckets operation;
/* Initialize as empty. */
qsize = 0;
for(i = 0; i < nbuckets; ++i) bucket[i] = NULL;
/* Set up initial position in queue. */
lastprio = startprio;
n = startprio/width; /* Virtual bucket */
lastbucket = n % nbuckets;
buckettop = (n+1)*width+0.5*width;
/* Set up queue size change thresholds. */
bot_threshold = nbuckets/2-2;
```

```
top_threshold = 2*nbuckets;
}/* end */
```

ก.2.3 การเปลี่ยนขนาด

```
resize(newsize)
int newsize;
/* This copies the queue onto a calendar with newsize buckets.
The new bucket array is on the opposite end of the array a[QSPACE] from the
original. */
{
double bwidth;
int i;
int oldnbuckets;
struct eventtype ** oldbucket;
if (!resizeenabled) return;
bwidth = 2* average incremental time/qsize; /* Calculate new bucket width. */
/* Save location and size of old calendar for use when copying calendar. */
bldbucket = bucket; oldnbuckets = nbucltets;
/* Initialize new calendar. */
if (firstsub == 0)
localinit(QSPACE-newsize, newsize, bwidth, lastprio);
else
localinit(0, newsize, bwidth, lastprio);
/* Copy queue elements to new calendar. */
for (i = oldnbuckets-1; i >= 0; −−i)
Transfer elements from bucket i to new calendar
by enqueueing them;
} /* end */
```

บทความทางวิชาการที่ได้รับการเผยแพร่แล้ว

[1] T. Siangsukone, C. Aswakul and L. Wuttisittikulkij. “Study of Optimised bucket widths in Calendar Queue for Discrete Event Simulator.” Electrical Engineering Conference. Vol. 26, No. 3, November 2003, pp. 981-985.

Study of Optimised bucket widths in Calendar Queue for Discrete Event Simulator

T.Siangsukone C.Aswakul L.Wuttisittikulkij
Center of Excellence in Telecommunication Technology Department of Electrical Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, 10200, Thailand. Phone +662 218-6908 Fax +662 218-6912
E-mail todsaporn.s@student.chula.ac.th chaodit.a@chula.ac.th lunch@ee.eng.chula.ac.th

Abstract

In this paper, a study of optimised bucket widths in calendar queue for discrete event simulator is presented. It is shown that the performance of calendar queue can vary noticeably depending on the chosen width factor(W_f). Existing algorithms appears to perform reasonably well in general situations. However, under certain scenarios these algorithms may deteriorate due to the highly skewed probability distribution of incremental event time. In such scenarios, the calendar queue with optimum width factor W_{fopt} has been found to accelerate up to twice as fast as existing algorithms.

Keywords : Discrete Event Simulation, Calendar Queue

1 Introduction

Discrete Event Simulation (DES) is a widely developed simulation method where the simulation clock is driven by events. An event is here defined as the situation that can happen in the considered system and change the system state variables. Events in DES are scheduled in a list (named Pending Event Set, PES) to occur in any future time. Typically, this list of events is implemented as a priority queue where the event priority is associated with the occurrence time of event, $t(e)$, and the event with the smallest priority is executed first.

In any DES, there are 3 main routines: (i) to find the next event from the list to be executed, (ii) to execute the event and (iii) to insert new events, being invoked from the execution, to the list. Since a DES needs to perform at least one event operation (dequeue or enqueue) in every program cycle, that is, in routines (i) and (iii), respectively, it becomes necessary that the priority queue algorithm selected for PES implementation must be able to handle all the event operations in the most efficient way.

Various priority queue algorithms (e.g. [1]-[5]) have been proposed with the aim of reducing the time complexity required for a large PES. Among these algorithms, calendar queue [5] has gained the most popularity due to its $O(1)$ time complexity, given that the calendar parameters have been properly set. In Section 2, we will illustrate the structure of calendar queue. Section 3 discusses about the effect of improperly selected calendar queue parameters. Section 4 presents the analysis of optimal bucket width for calendar queue. Experimental results of calendar queue performance at different bucket width selections are then given in Section 5. Finally, Section 6 gives a conclusion of findings and suggests some directions worthy a future investigation.

2 Calendar Queue [5]

A calendar queue is defined as an array of lists, each of which contains future events. To show the advantage of calendar queue algorithm, let us compare it with a linear list [10].

For time $t > 0$, let $N(t) \geq 0$ denote the number of active events being scheduled to occur in our PES. When the simulation is running in its steady state, the number $N(t)$ may be approximately constant, hence dropping the temporal argument t. In this case, if we schedule events by using a linear list, then $O(N)$ bound on the time taken by event operations follows because, in an enqueue operation, we may have to search all the N events until we find the proper position to place the enqueued event. If N is too large (e.g. when a big system is simulated), then our DES will not finish its task within an acceptable time. It is generally agreed that any algorithms that require the time complexity greater than $O(1)$ may become impractical in any large DES.

To avoid unnecessarily large time complexity, the principle of calendar queue is to partition the large list of N events into M shorter lists and each list is called a bucket. Bucket is a list with a specified range of admission times. Only events that occur in this range are allowed to be scheduled in the bucket. Without loss of generality, let every bucket have an equal width of δ. Any event with the occurrence time $t(e)$ will be associated with the m-th bucket in year y ($y = 0, 1, 2, \ldots$) if and only if $t(e) \in [((yM + m)\delta), (yM + m + 1)\delta)$.

Since the number of events in each of the M lists is typically small, any priority queue algorithms with low overheads can be conveniently utilised for the bucket discipline operations without any significant effect on the order of calendar queue time complexity.

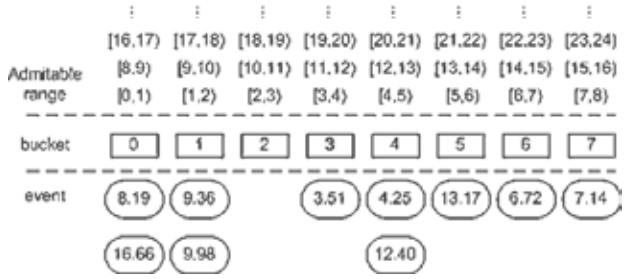


Figure 1: Calendar queue structure ($M = 8, N = 10, \delta = 1$).

Recall that there are two main operations, enqueue and dequeue, in PES management. To find the bucket number $m(e)$ to enqueue an event e that occurs at time $t(e)$, we may apply

$$m(e) = \left\lfloor \frac{t(e)}{\delta} \right\rfloor \mod M \tag{1}$$

which results in the $O(1)$ time complexity. For a dequeue operation, an index variable may be introduced to identify the most recently dequeued bucket. From this bucket onwards, the algorithm can then search for the next event to be dequeued. An example of calendar queue is illustrated in Figure 1.

In Figure 1, the most recently dequeued bucket is the third bucket. Each bucket is implemented as a linear list. Thus, a dequeue operation can be easily performed by taking out the event on top of the current bucket's list. To enqueue an event e into the calendar, the target bucket is easily obtained by (1) and the event e can then be inserted into the list of the target bucket. Ideally, the calendar queue requires the time complexity of $O(1)$ for both enqueue and dequeue operations if the optimal calendar parameters are selected. An analytical approach for the optimal tuning of calendar has been proposed in [9]. However, for the sake of analysability, it is resorted in [9] to a few assumptions (to be further clarified in Section 4) which may not be justified in practice. This paper is aimed at investigating the effects of such injustification and proposing an alternative way of modifying the obtained formula in [9] for practical usage of calendar.

3 Calendar Queue Parameters

There are two control parameters in the calendar queue. The first parameter is the number of buckets M. On one hand, if M is too small, then the partitioned list in each bucket may become large. This can result in an increase in the time required for the bucket operations, especially when the number of events N is huge. On the other hand, setting M to a too large value will gain nothing of substantial advantage in terms of computational complexity. The reason is that a calendar queue with too many buckets -though decreasing the time complexity- will unavoidably increase the memory requirement. Based on rigorous empirical studies (e.g. [5] - [8]), it has been found that a plausible value of M should be $M = 2N$ (whose value will also be utilised in this paper).

The second parameter is the bucket width (δ). This parameter affects how a calendar queue algorithm performs in both dequeue and enqueue operations. For dequeue operations, if δ is too small, then most of the events in each bucket's list will become the events being scheduled to happen in the next years. This causes the calendar to search unnecessarily many buckets before reaching the bucket that contains the next dequeue event. However, for enqueue operations, if δ is too large, then most of the events will be placed in the current bucket (or only few next buckets), which causes a long list to occur within each bucket and leaves many other buckets mostly empty. The optimal values for bucket width will be the focus of investigations in this paper.

4 Calendar Queue Analysis

The analytical study of calendar queue parameters has been carried out [9] under the condition that the number of active events in the calendar is constant (e.g. at its steady state). Further, given that a dequeue event at time t generates a subsequent event at time $t + \tau_i$, the incremental time τ_i is defined as an independent and identically distributed random variable with mean μ. The analysis in [9] relies on the following assumptions.

[A1] The calendar queue contains an infinite number of buckets, i.e., $M = \infty$.

[A2] A direct search algorithm is applied for all dequeue operations within each bucket's list.

[A3] The processor times needed to process an empty bucket (τ_b) and to process a list element (τ_l) are all constant.

Based on assumptions [A1] - [A3], it is possible to derive for the optimal value of bucket width, δ_{opt}, which minimises the the expected time to process an event [9]:

$$\delta_{opt} = \sqrt{2b/c}\frac{\mu}{N} + O(N^{-3/2}) \tag{2}$$

However, the formula (2) should only be used cautiously. Due to the limited memory resource in most computing facilities, the number of buckets (M) cannot be infinitely increased; hence, the invalidation of [A1]. Furthermore, instread of the direct search [A2], the linear list may be employed to achieve a better performance of the calendar queue. Regarding assumption [A3], the processing time may be more reasonably assumed dependent on the amount of memory usage and the computational requirement for generat-

Table 1: Distribution of τ_i

Distribution[(a)]	Expression to compute random variate[(b)]
exponential	$-\ln rand$
triangular	$1.5 rand^{0.5}$
negative Triangular	$3((1-rand)^{0.5}+1)$
bimodal a	0.0025rand + if $rand$ ¡ 0.5 then 0.94875 else 1.04875
bimodal b	0.8rand + if $rand$ ¡ 0.5 then 0.1 else 1.1
(a)Mean of τ_i is normalised to 1 in every case. (b)Function rand returns a value uniformly distributed between 0 and 1.	

ing random event times according to different probability distributions.

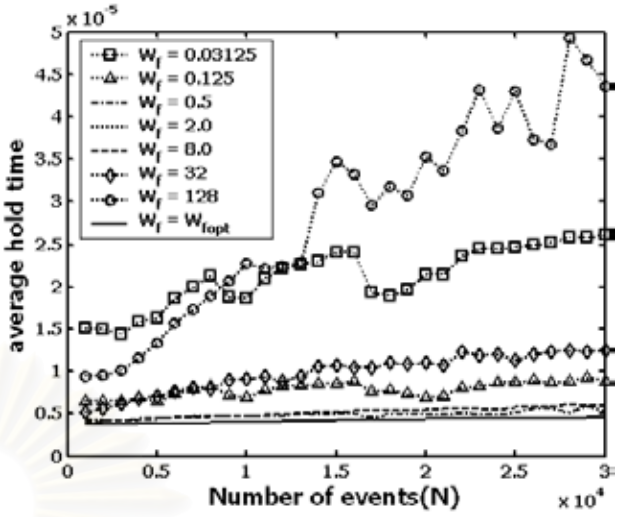


Figure 2: average time per hold operation.

5 Study of Optimal Bucket Width

In doubt of the justification for [A1] - [A3] , this paper proposes an alternative practical study for the optimal bucket width. Because of our study focuses on a large simulation system, the number of active events (N) becomes unavoidably large, which lets the term $O(N^{-3/2})$ be approximately insignificant. Thus, formula (2) can be rewritten as

$$\delta_{opt} = W_f \frac{\mu}{N} \tag{3}$$

where a new parameter, called width factor (W_f), is here introduced to summarise all the processor time parameters. [1]

In our experiments, the widely adopted hold model (e.g. [1]-[6]) is employed to evaluate the performance of calendar queue over a wide range of W_f. The distributions of τ_i as detailed in Table 1 are here studied. All the experiments are performed on an INTEL P4 2.0GHz with 512Mb RAM and Microsoft Windows XP operating system.

Figure 2 plots the resultant average time per hold operation when the bucket width of calendar queue is computed from 3 with $W_f = 2^i$ $(i = -5, -4, \ldots, 7)$ at various values of N when the employed distribution for τ_i is bimodal a. The figure shows that the optimal width factor W_{fopt}, which gives the minimum time to perform a hold operation, should be set to 2. Choosing W_f too far from its optimum W_{fopt} results in the complexity of calendar queue much greater than $O(1)$.

A sensitivity study to the optimality of bucket width is studied in Figure 3. It is clear that an attempt to achieve a near optimal range for W_f is not too hard. The reason is that, although W_{fopt} is not employed, the time to process a hold operation is still in an acceptable range of complexity for an order-of-magnitude wide range of W_f around W_{fopt}.

Figure 4 plots the average hold time versus W_f for a large $N = 30{,}000$ and five different distributions of τ_i. Each distribution gives the average hold time with a parabola-alike curve and reflects a different value of W_{fopt}. However, the obtained results from all the distributions give a wide near-optimal range of W_f, which always includes the value of $W_f = 2$. This finding is consistent with all other experiments not reported herein. Thus, the results suggest a practical approximation for $W_f = 2$, by which the time complexity would increase by less than 15% (based on our experiences so far) from its minimum.

Figure 5 compares the performance of calendar queue with W_{fopt} and another three alternative implementations of calendar in the literature, namely,

(i) the conventional calendar queue with its fixed value of bucket width computed from the average inter-event time on the head of queue [5],

(ii) the dynamic calendar queue with its bucket width dynamically computed from the average inter-event time on the part of queue with the highest event density [7] and

(iii) SNOOPY algorithm whose implementation is based on (ii) with an addition adaptive mechanism to trace for a near-optimal bucket width [8].

Figure 5 suggests that, if one could set W_f to its optimum, then we can improve the speed of existing implementations (i) - (iii) by making the optimal calendar queue algorithm upto twice as fast.

[1]Since τ_b and τ_l are not known a priori, replacing both of them by a single parameter, $W_f = \sqrt{\frac{2\tau_b}{\tau_l}}$, reduces the cardinality of solution space from 2 to 1 dimension when searching for an optimal bucket width.

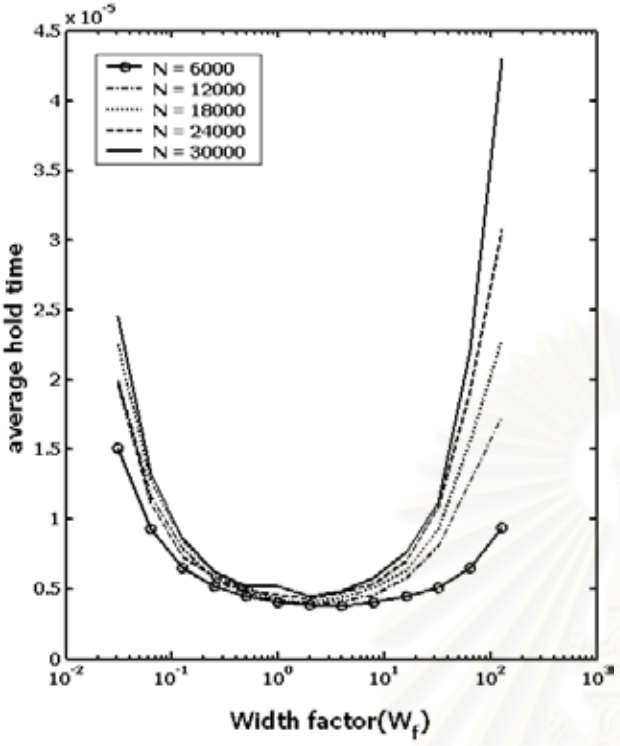


Figure 3: average time per hold operation.

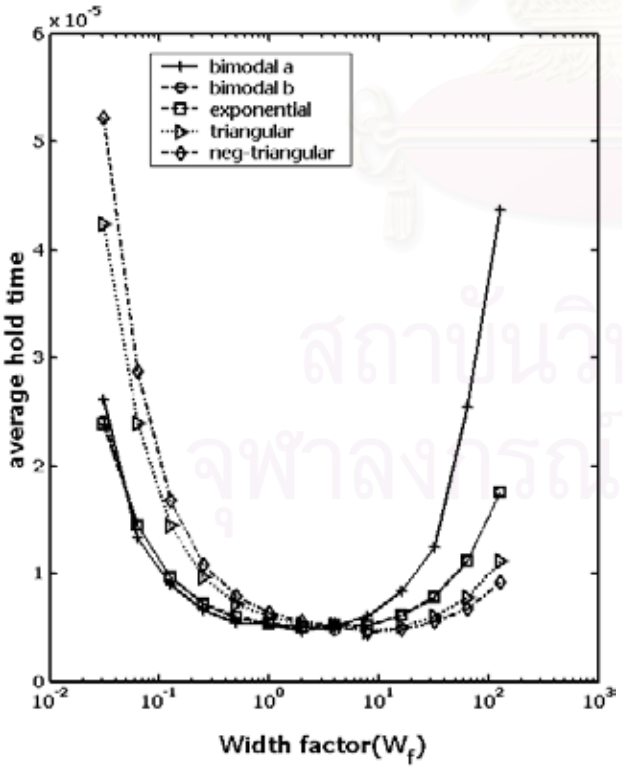


Figure 4: average time per hold operation.

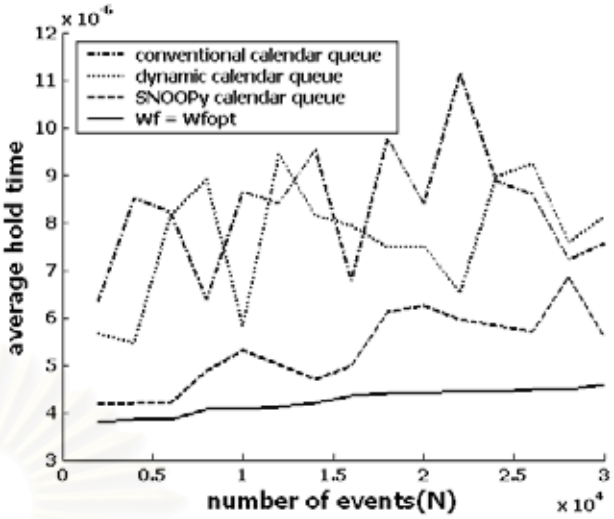


Figure 5: comparison of existing algorithm and W_{fopt} algorithm.

6 Conclusion

Because the simulation time of DES is mostly spent on the event management operations, an efficient data structure can explicitly speed up the time needed to simulate a huge system. For this reason, the calendar queue, and other calendar-based algorithms, are the popular data structure adopted for DES implementation for their $O(1)$ time complexity.

Our experiments show that the performance of existing calendar-based algorithms may vary and can much deteriorate when the event incremental time (τ_i) has a highly skewed probability distribution. However, it is here found that the calendar queue with its width factor W_f set to its optimum W_{fopt} can speed up the running time up to twice as fast as other existing calendar-based algorithms.

Although the selection of W_{fopt} can result in the optimal speed of calendar, it is nontrivial how one may a priori set this value. This is because W_{fopt} is dependent on many unknown variables (e.g. distribution of τ_i). Incorporating an adaptive mechanism that can change W_{fopt} dynamically and optimally in the calendar is here believed a rewarding research. Results in this direction will be reported in our sequel paper.

References

[1] J.H. Blackstone, C.L. Hogg, and D.T. Phillips, "A two-list synchronization procedure for discrete event simulation," *CACM*, Vol. 24, No.12, pp. 625-629, Dec. 1981.

[2] D. Davey, and J. Vaucher, "Self-optimizing partitioned sequencing sets for discrete event simulation" *infor*, Vol. 18, No. 1, pp. 41-61, Dec. 1981.

[3] W.R. Franta, and K. Maly, "An efficient data structure for the simulation event set," *CACM*, Vol. 20, No. 8, pp. 596-602, Aug. 1977.

[4] W.R. Franta, and K. Maly, "A comparison of heaps and the TL structure for the simulation event set," *CACM*, Vol. 21, No. 10, pp. 873-875, Aug. 1977.

[5] R. Brown, "Calendar Queues: A Fast 0(1) Priority Queue Implementation for the Simulation Event Set Problem," *CACM*, Vol. 31, No. 10, pp. 1220-1227, Oct. 1988.

[6] D.W. Jones, "An Empirical Comparison of Priority-queue and Event-set Implementations," *CACM*, Vol. 29, No. 4, pp. 300-311, Oct. 1986.

[7] S. Oh, and J. Ahn, "Dynamic Calendar Queue" *Inproceeding of the 32nd Annual Simulation Symposium*, 1999

[8] K.H. Tan, and L.J. Thng, "SNOOPY CALENDAR QUEUE," *Inproceeding of the 2000 Winter Simulation Conference*

[9] K.B. Erickson, R.E. Ladner, and A. LaMarca, "Optimizing Static Calendar Queue," Annual IEEE Symposium on Foundations of Computer Science, Vol. 35, pp. 732-743, 1994.

[10] D.E. Knuth, "The Art of Computer Programming. Vol.1 3rd edition., Fundamental Algorithms," Addison-Wesley, Reading, Mass., 1997. chapter 2.

ประวัติผู้เขียนวิทยานิพนธ์

นายทศพร เสียงสุคนธ์ เกิดเมื่อวันที่ 15 กันยายน พ.ศ. 2523 ที่จังหวัดกรุงเทพมหานคร เข้าศึกษาในหลักสูตรวิศวกรรมศาสตรบัณฑิต คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์ ในปีการศึกษา 2540 และสำเร็จการศึกษาในปี พ.ศ. 2544 หลังจากนั้นได้เข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต ณ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2544