KTH

Engineering Skills

Hussam Hassanein, Qi Li
hussamh@kth.se,qi5@kth.se

Experimental Method – Planning

Page of Content

Project Goals

Priority queue has broad implementation in information technology. Different implementation of the priority queue can have a different efficiency. The goal of this experiment is to find out the most efficiency implementation of priority queue between linked list and skew heap.
The experiment will be done by collecting the data and analyze the data through certain graphing tools. The goal of the planning is to give a good understanding of the problem and an overview before the experiment.

Project Deliverables

Simple Preliminary Study

To be able to compare the efficiency between the linked list skew heap implementation. Certain literature study is necessary. The goal of the literature study is to figure out which implementation is most efficient. To be able to properly implement the tree-based structure priority queue the original report from Daniel Sleator and Robert Tarjan will be studied. The report is open sourced online. The course web from KTH course ID1020 Algorithms and Data Structures and the powerpoint about skew heap from Chalmers University of Technology has also been used for preliminary study for skew heap.

Since linked list implementation is quite traditional way to implement the priority queue, revision on the KTH course material for ID1020 Algorithms and Data Structures will be sufficient. The material is available on KTH course web.

To be able to verify and validate the experiment result, the team has decide to use chapter 11 course material from Oregon State University. The reason of using Oregon State University chapter 11 course materia[5]l is because of there are detailed efficiency comparison with graph presented.

The efficiency measurement system will be implement from dedication C library. The system could help us measure the speed and the complexity for each of the structure and then compare them to come up with an answer for the question. In case of system error that caused by wrong implementation,The team will study the "C Library - <time.h> “ article from Tutorials Point[6] before implement the efficiency measurement system.

The experiment method is designed to be as automated as possible. Priority queue experiment is the first time the team engage automotive experiment. The KTH operating system and Engineering skill in ICT course leader has present few automotive method during the lecture. The lecture note that taken from these course by the team member will be used as preliminary study. The team is expecting to engage a lot of problem during the setup of automotive experiment system. To be able to solve these problem, the post on stackexchange website will be used and studied[7]. The team is also expecting to use some existence *.p file with some modification from operating system course for gnuplot[9].

The program is planned to be designed as Multilevel Feedback queue(MLFQ). To be able to do this, Operating system course material ID1206 will be studied. The material is available on KTH course web.Each item in the queue will be treated as a program that are planning to be executed by the computer in the process queue. In this case,a few parameters are involved in such experiments, like the number of task per program in the queue, the number of program that are waiting to be executed in the queue, and also the context of the experiment could give a certain output considering some other factors. Parameters are influenced by each other as well as their influence on the output result, therefore, which parameter to choose should be always thought through when considering values as inputs of experiments.

Identify which types of methods the study will be based on (qualitative, quantitative, inductive, deductive, etc.).

Deductive method means “Methods that are based on logical conclusions [1]”. Our experiment is not logic base but more on set a hypothesis and provide true or false based on observation on experiment target. The method proceed in following pattern are called inductive method. This is also the most common way engineer use during their work. [1]
Inductive method can be categorized into qualitative and quantities. Qualitative method means “the observations and measurements we make can be given a numeric value [1]” and quantitative means “the method that characterized by that the systems observed, and the results obtained cannot easily be assigned numerical values [1]”. To be able to measure the efficiency, ones need to list out multiple input and output data. The result will be based on the comparison and analysis of the output data. Since the experiment observation is highly data dependent, it will be categorized into qualitative method.

Project Schedule

Experiment Plan

Parameter that will affect the queue for the experiment:

Parameter	Description	Experiment need
Item size	the length of decomposition/ complexity of the process in the queue/vertical length of the process tree	Not needed. Set as constant or random.
Queue size	The process that waiting to be dequeue	Needed. This is our X value
The number of N	The Queue size inside each thread/item. /the horizontal length of the tree	Not needed.Set as constant or random.
Hardware Load	Current hardware load and how many background process is currently running	Not needed. This is systematic error. Need to avoid.

Implementation plan:

Basic Requirement:

Experiment will start with two implementations of the queue. The program will be designed as Multilevel Feedback queue(MLFQ). Each item that are waiting in the queue will be treated as a process queue that contain 0-N task. There will be a limit set for the “Item size”. When the limit is reached the item /process has reached the end of the tree. The priority of the item that will be enqueue with original item and after each decomposition has been set with the time stamp. This means that the queue will be implement as “first come first serve” style of FIFO structure. After each decomposition, a increment module will be active. This module is used as a random integer generator and the number that it generate will be add into current timestamp for each subitems. The reason that this module exist is for simulating the different timing that each sub item comes into the queue. There will be a timer/benchmark implemented at the main method of the implementation.
During the implementation, the program has been set to output the data that will be analyze later so it can be input into plot software later. After developers have done with the implementation of both queue, developers will start setting up automation test environment

with the implementation programming language or script. The more automation involved during the experiment the less the human mistake that can manipulate the output data.

Example of the implementation:
If Queue size =4

Timer on!	
Main Queue	Operation
Item 1: Queue size:N	→ Pop → Queue that has size N (for example N =3) → calculate increment()→ enqueue subitem that base on current time + increment()
Item 2: Queue size:N	
Item 3: Queue size:N	
Item 4: Queue size:N	
	Recursive the process for item 2,3,and 4 after 1 is done.
Timer off!	

Queue	Operation
Subitem 1	-→ Pop → Queue that has size N (for example N =3) → calculate increment()→ enqueue subitem that base on current time + increment()
Subitem 2	
Subitem 3	
	Recursive the process for item 2 and 3 after 1 is done.

Queue	Operation
Sub Sub Item 1	→ Pop → Queue that has size N (for example N =3) → calculate increment()→ enqueue subitem that base on current time + increment()
Sub Sub Item 2	
Sub Sub Item 3	
Recursive until reach the "Item size"	

→ export output the final data list to .dat file →import it into gnuplot with ">" operation

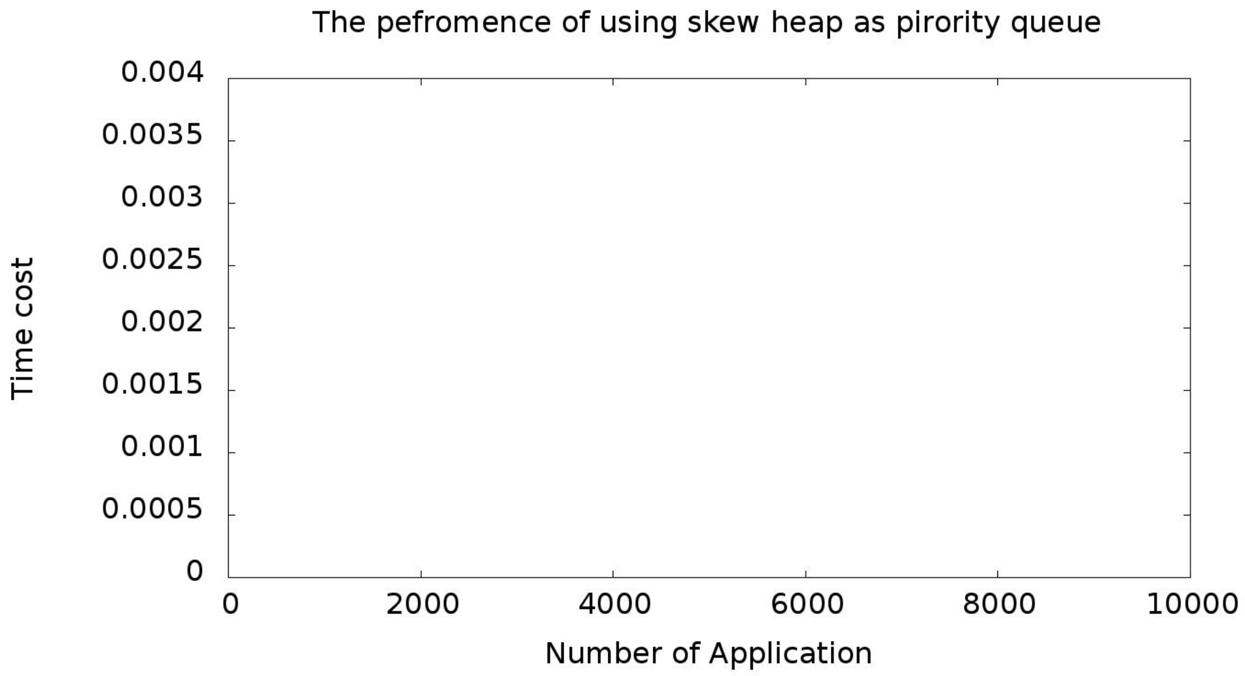
The implementation of the skew heap and linked list has to follow the property of each data structure that ones studied during the literature study. Eventually the final output will compare to the theoretical data for verificat and validate the precision and accuracy of the experiment. The experiment data will never reached the optimal data, so there is not necessary to have exactly the same graph as the theoretical data but they should have the same complexity pattern.

Best and worst case of each implementation:
Linked list has average complexity O(n) during insertion and O(1) during pop as a priority queue if linked list is always ordered[10] which n is the length of the list. The worst case for linked list is when the item append always have lowest priority. In that case insertion will always goes into the end of list and will cause constant O(n) complexity and this will be the case when it comes to priority queue. Because we want first in first out, which append item will always goes into the end of the list.During the experiment ones have decompose function, decompose function is a special case because of the it will generate its priority base on the item that just popped which is the one has lowest priority. Decompose function will cause random insertion of the linked list. In this case, n is the length of the list that has the highest priority than current item.
The linked list is not the most optimized solution when comes to priority queue. Because of the issues that linked list have, researchers create the heap implementation. Heap is one kind of binary tree. Heap implementation in worst case can create leftist heaps and this will be no much difference than a linked list implementation. To fix the leftist heaps issue, introduce skew heap. Skew heap have all the property that heap have but self- adjusting by swapping constantly. This will provide constant O(log n) complexity for merge, push and pop and create average complexity O(log n). However, for merge, push and pop individually, they can have worst case as O(n) therefore ones can only say the average complexity is O(log n). During normal heap or linked list implementation the best case is when the appending item always have smallest priority but unfortunately this does not fit FIFO first in first out structure since the further the time move the lowest priority ones get. That is also the reason why skew heap is better implementation since skew heap "blindly" swap its node during each pop and push. This can average the appending time and merge time since the tree will be relevantly even.

Data collection and analyze plan:

The experiment will be carried out for 3 different input data patterns. The definition of done for each experiment is when the main queue is cleared, and the output data will be provid into a .dat file. *These data pattern will base on X coordinate = the number of program in queue that initially been scheduled in the queue(Queue size) and Y coordinates = time cost to clean up the queue.* There will be timer implemented in the main method and use as output data in the end of each run from the code.
Each input data pattern will be executing 3 times, then calculate the mean value out of them. This is for minimize the Random errors. Before the analyze part start, developers should have 3 different mean value as output data that calculated from raw data during the 3 trials from different input pattern. These data will be used to plot 3 different graphs for 3 different data input patterns. These graphs will be compared and analyzed. If they have the approximate same pattern, the experiment has a conclusion based on the data and graph. All the recursion experiment step is to minimize the Random errors and Systematic errors. Because computer might under different kind of load during different timing. Launch different data pattern 3 times and use 3 different data pattern can eliminate the systematic errors too. The data and graph generated will be compare will theoretical data[5] that describe in literature study for validate and verify the data correctness. The data and graphs should not

be exactly the same as the theoretical data and optimal data stated but the data and graphs should have relevantly same pattern. The conclusion of the experiment will also compare and consult with the theoretical report [5] for more accurate and precise conclusion.
Expected graph output format:



Implementation Method:

Implementation will be done to fits the requirement of the experiment.

In skew heap implementation, the code has been separated into 7 module. They are heap/node creation,naive merge,swap,add,pop,decompose and random number generator"increment". The reason of separating the code into 7 module is due to the property of the skew heap. To be able to do the following implementation correctly,the item of the heap will be predefined with the type define"typedf structure" method.

The heap/node creation will create a typedef Tree object with all the property initialized. The property of the typedef structure is initially planned as 6 property. They are the address of parent,left element and the right element, the content or priority of the queue, the length of the architecture and the address current node.

The add method will add a node inside the skew heap and reorder it as the skew heap required. To be to add item inside the skew heap without breaking the regular of the skew heap, the add function need to have naive merage method and swap method. Naive merge will merge an item or a skew heap with current skew heap but this will cause rightest tree. Swap method will solve the problem by swap each node from the tail of the heap and eventually make a even binary tree. The pop method will return and detach the head of the tree and reorder the tree like the add method does which is naive merge and swap. During each pop the heap will generate certain amount of subtask. This is the property of MLFQ. To be able to do subtask generation, the team has implement the decompose method. The decompose method will generate the task with a timestep which base on the timestep that the item just pop and plus a random generate number. The random number generator is a easy math function rand() divide and obtain the remainder of the limitation that maximum number allowed to generate. Time step use the same system as efficiency measurement system which is the dedicate c library “time.h”. Time.h function will obtain current CPU time and return a floating point number. Time stamp will take this number and put it inside the node.

The linked list implementation has same structure but different implementation than skew heap inside each module.The linked list have insert(add) add value into the list,free_list to free the malloc space from heap,pop to take out the value from the list,decompose to generate 0-N subtask and eventually increment for generate random number. To be able to fits the basic requirement of the linked list, each item will be type define “typedef structure” as integer data to store the priority/content of the item and a pointer to the next item address inside the list.
Insert function will try to append a item in the start of the priority queue by compare each item from head of the list. When the current append item is smaller than next item to be compared, the program will detach the link with the next compared item. After detach the link with next compared item, the program will save the address of the next compared item before attach to the new item. Eventually, the newly append item will also attach to the next compared item. The pop function is built base on the list is always organized as a FIFO like priority queue structure. So the pop function will pop the item that is in the top of the list which should be the same as the head of the priority queue. Decompose function will act as same as skew heap’s decompose function with 0-N sub item generated after each pop. For increment function, because of the random number generator does not depends on any data structure, the linked list will import the increment function from skew heap header file or just simple copy and paste the increment function from skew heap c code file.

After one has done with all the implementation of the skew heap and linked list priority queue, one should start implement the efficiency measurement system. This will still be done by “time.h” function in the dedicate c library. Program will obtain the CPU time at the beginning of the code. Than obtain the CPU time again in the end of the code and use the floating point value program just get in the end of the code minus the floating point value from the beginning of the code. To convert the CPU time to normal clock second, one should use (CPU TIME)/CLOCKS_PER_SEC which CLOCKS_PER_SEC is a macro expands to an expression representing the number of clock ticks per second[8].
During the implementation, the team member has to be aware of the difference between a pointer and a object inside the pointing address.

The automation part of the experiment will be implement with bash script and Makefile. The most ideal situation is with ./*.sh the program should built itself ,execute the code,port the result into a *.dat file and eventually gnuplot the *.p file that base on *.dat. Inside the script, each implementation should run three times and port the data into different folder but with same filename.dat. Than paste them into one filename.dat and take the average value for each role before plot them in gnuplot. The average value procedure can be done in the *.p file. The random error should be eliminated from taking the average value for plotting.

In the end phase of encoding the program for experiment, certain procedure will ensure the quality of the code does not cause systematic error and affect the experiment result. To be able to ensure the quality of the code, once will use the Continuous Integration (CI) tool to constant doing unit testing and integration testing throughout the code. All the issues that mention in the unit testing tool should be fix and eventually the build process from integration testing tool should pass.

Requirements:

MoSCoW: Must have, Should have, Could have, and Won't have[2]

Requirements number	Type of requirement	Designation Source	Description of the requirement / what must be met	Completed / Not met / Partially met
1	Must have	Queue structure [3]	Queue must be FIFO structure. To verify it the first item that goes into the queue must be the first one be popped.[3]	

2	Must have	Multiple input data pattern. [4]	The experiment must be carry out at least 3 different input data patterns. Each data pattern must be recursively done 3 times. This is to minimize the random error of the data.	
3	Should have	Use Usual implementation [3]	The implementation of the queue is preferable in usual implementation than naïve implementation as Wikipedia stated. [3]	
4.	Should have	Automation experiment step (experiment requirement)	The experiment is preferable to be implemented as automotive as possible. This is to eliminate Random error and systematic error.	

5	Should have	Pass the Unit testing and Integration testing	Eliminate the systematic error from the implementation of the code. This can be verified when SonarQube start showing passing the Quality gate and TravisCI finished with build success.	

Supporting Plans

Experimental setup

Following error will affect the experiment:
Systematic error: CPU and Hardware load
Implementation issue during unappropriated code
Random error: Human mistake during experiment phase.

The experiment will be proceeding under following equipment setup to minimize the uncertainties:
Operating System: Linux (Programming under Mac and Linux)
Programming language: C,script
Compiler: gcc
IDE: Atom
Plotting tools:gnuplots
Unit testing: SonarQube
Integration testing: TravisCI
Version control: Github
CPU and hardware load can cause systematic error during the experiment. Operating system has loaded with different background applications. At different timing the CPU and hardware load are different. This will cause uncertainties for the data. C language is the language that closest to the hardware. Use C to programming and compile the experiment can ideally eliminate systematic error from CPU and other hardware load as low as possible. Window and mac are having so many background applications running which will affect the output data. Linux has relevantly ideally environment for this experiment since it only loads with what the experiment needed.
Operating mistake by human can cause Random errors for the experiment. Random error can be eliminated by repeating the experiment. It can also been avoided by make the experiment step automotive. Since computer is good at doing stuff that have certain pattern and repeat itself. That is why the experiment will be as automotive as possible. To be able to reach automotive goal, the experiment equipment should be able to use port or tunnel operation in the script. Gnuplot can be used as port ">" operation from c code directly. This can be used in automation of the experiment to eliminate the random errors.
To be able to further eliminate the Random errors, as Experiment Plan stated the experiment for each data pattern will be carry out 3 trials and there will be 3 different data patterns. Each data pattern will be calculated the mean value out of 3 trials. The graph will be compared to make sure the conclusion ones have is correct.
Implementation the code in an unappropriated way can cause Systematic error.This kind of error cannot be eliminating from repeating the experiment and will create a fix amount of uncertainty in the output data. To avoid that, the develop phase will be carry out with Continuous integration and deployment tools. Implementation phase will be carry out with

SonarQube for unit testing and TravisCi for integration testing to eliminate the systematic error that caused during the develop phase. The standard for verification of this part is to pass the Quality gate from SonarQube and successfully build from TravisCI.

Plan for Verifying / Validating / Testing the Experiment Setup?

The output that we are expecting before final analyze phase are (after raw data)

1. Three different data that well processed from different data pattern(three *.dat file)
2. 1 data that well processed 3*3 experiment (mean value,best fit and etc)(*.dat file)
3. A well processed graph with 3 different data pattern from the data in 1(three gnuplot graph)
4. A well processed graph with the data that from data in 2(gnuplot graph)

To be able to verify and validate the data is correct, ones will compare both graph with the theoretical output and optimal output from the original report from Daniel Sleator and Robert Tarjan ,the ID1020 course web and the report from Oregon State University [5]. The graph should be approximately having the same shape and other property.
As mentioned in previous chapter, the implementation verification and validation for eliminate systematic error will depends on the Unit testing tools and integration testing tool. It will be considered a correct implementation code without systematic error when most of the Sonar issue has been solved, pass the “sonar way” quality gate and successfully build from TravisCI.
The hardware and CPU load is hard to consider since we are using personal computers. In real lab environment, most ideal setup is probably a Linux computer that just been reseted and only load what the module that computer need to properly run and what the lab required. In this case is hard to eliminate the system error from personal computers. Since we can not format whole computer just for this experiment. In this case, we will try to run a small loop benchmark and compare it to the optimal output on internet. Than we will try to find out the systematic error with “the time that our loop run - the optimal time”. Maybe do this three times and find out the mean systematic error.

There will be 6 part that need to be implement for this experiment.

Implementation type	**Description**
Random Integer Generator	Generate random integer for decomposition between 0-N item. Also for the increment() part.
Queue in tree architecture	Test subject 1. Queue that implement with tree structure. This part is most about integration. Following FIFO architecture.
Queue in Link List architecture	Test subject 2. Queue that implement with Linked List structure. This part is most about integration. Following FIFO architecture.
Benchmark	A timer that calculate the total usage of the time when the queue is empty. Output the data for data analyze and gnuplot after it returns.
Experiment Automation	Automate the experiment with script code. Eliminate the Random error.
Threads/Item in the Queue	Each item in the queue are treated as a thread/program. Each of them has a small queue inside for the subtask that the program need to executed. This is implemented as MLFQ

Meeting the experiment's goals

The experiment is considered to be done only when both methods are executed and run through different numbers under the same requirements, and that results in having a similar pattern for these different inputs. The end goal is to determine which method is more efficient and that could be decided by running different algorithm to measure the speed/ effectivity unit for skew heap and linked list implementation as priority queue. The comparison is to be done under the same requirements to have some sort of a module for each of the methods and be able to validate and test the results.
All the requirements have to be fulfilled/met in the experiment to be able to reach the goal of the experiment. The input values can varies but not the main condition that the experiments are based on.

Bibliography

[1]On experimental methodology. (2017). *On experimental methodology*. [online] Available at: https://www.kth.se/social/files/59f735ca56be5b2058fad8b5/On%20experimental%20methodology.pdf [Accessed 6 Dec. 2017].
[2] En.wikipedia.org. (2017). *MoSCoW method*. [online] Available at: https://en.wikipedia.org/wiki/MoSCoW_method [Accessed 6 Dec. 2017].
[3] En.wikipedia.org. (2017). *Priority queue*. [online] Available at: https://en.wikipedia.org/wiki/Priority_queue [Accessed 6 Dec. 2017].
[4] UNIVERSITY OF MARYLAND. (2017). *Random vs Systematic Error*. [online] Available at: https://www.physics.umd.edu/courses/Phys276/Hill/Information/Notes/ErrorAnalysis.html [Accessed 6 Dec. 2017].
[5]"Chapter 11: Priority Queues and Heaps", Web.engr.oregonstate.edu, 2017. [Online]. Available: http://web.engr.oregonstate.edu/~sinisa/courses/OSU/CS261/CS261_Textbook/Chapter11.pdf. [Accessed: 19- Dec- 2017].
[6]"C Library - <time.h>", www.tutorialspoint.com, 2017. [Online]. Available: https://www.tutorialspoint.com/c_standard_library/time_h.htm. [Accessed: 19- Dec- 2017].
[7]"Hot Questions - Stack Exchange", Stackexchange.com, 2017. [Online]. Available: https://stackexchange.com/. [Accessed: 19- Dec- 2017].
[8]Cplusplus.com. (2017). *CLOCKS_PER_SEC - C++ Reference*. [online] Available at: http://www.cplusplus.com/reference/ctime/CLOCKS_PER_SEC/ [Accessed 20 Dec. 2017].
[9]Gits-15.sys.kth.se. (2017). *Lecture material for ID1200/06*. [online] Available at: https://gits-15.sys.kth.se/johanmon/ID1206 [Accessed 20 Dec. 2017].
[10]Eecs.wsu.edu. (2017). *Priority Queues (Heaps)*. [online] Available at: http://www.eecs.wsu.edu/~ananth/CptS223/Lectures/heaps.pdf [Accessed 20 Dec. 2017].
[11]Courses.cs.washington.edu. (2017). *Priority Queues (Today: Skew Heaps & Binomial Queues) Chapter 6 in Weiss*. [online] Available at: https://courses.cs.washington.edu/courses/cse326/06au/lectures/lect07.pdf [Accessed 20 Dec. 2017].