June 03, 2016

# MULTIPLE TIER LOW OVERHEAD MEMORY LEAK DETECTOR

Ben Cheng

Simon Que

# MULTIPLE TIER LOW OVERHEAD MEMORY LEAK DETECTOR

## ABSTRACT

A memory leak detector system can be used to detect memory leaks, which is when a computer program fails to release unneeded memory allocations, in a computer that executes multiple programs. The system utilizes a multi tier methodology to detect memory leaks. In a first tier, the system collects a histogram representing allocation counts for different allocation sizes of memory at the computer. If the system detects an above-a-threshold increase in the number of allocations for one or more of the allocation sizes, the system marks the one or more allocation sizes as suspected leaks and proceeds to a second tier of the multiple tier method.

In the second tier, the system collects a histogram based on call stacks that led to each above-a-threshold increase in allocation sizes detected in the first tier. The system marks the call stacks with an above-a-threshold increase in call stack traces as prospective leaks and proceeds to a third tier of the multiple tier leak detection method.

In the third tier, the system records the allocation times of each memory allocation that fits the suspected leak profile, including leak sizes found in the first tier and call stacks found in the second tier. If the oldest allocations are not being freed and persist over a period of time, then the system marks the allocation(s), the allocation size(s), and the originating call stack(s) as a probable memory leak.

## PROBLEM STATEMENT

Memory leaks occur when a software program fails to release memory that is no longer needed. Memory leaks waste Random Access Memory (RAM) and thus reduce computer performance. Often, small or slow memory leaks occurring in a program go unnoticed for an extended period of time until they aggregate and then become noticeable. At this point, though, it is harder to locate and rectify the sources of the leaks. Thus, it is helpful to detect memory leaks before they reduce the performance of computers.

There are various methods used for addressing memory leaks which provide feedback to software programmers about the origin of leaks; however, such methods often carry significant overhead that slows down the overall execution speed and hence spoil the user experience. Moreover, existing methods sometimes require a special allocator and a process to terminate. Thus, there are opportunities to develop alternate methods to detect memory leaks.

## DETAILED DESCRIPTION

The systems and techniques described in this disclosure relate to a memory leak detector system that detects memory leaks using a multiple tier method. The system can be implemented for use in an Internet, an intranet, or another client and server environment. The system can be implemented locally on a client device or implemented across a client device and server environment. The client device can be any electronic device such as a computer, a laptop, a mobile device, a smartphone, a tablet, etc.

Fig. 1 illustrates an example multiple tier method 100 that can be used to detect memory leaks. The memory leak detector system utilizes multiple layers of analysis to detect memory leaks in a computer. In an example scenario, the system utilizes three tiers to detect memory leaks. In a first tier of the multiple tier method, as shown in Fig. 1a, the system initializes 102 a counter K =1 and collects 104 a histogram representing allocation counts for different allocation sizes of memory. Memory allocation includes assigning specific memory to programs and/or services as per their requirements when they are executed. After the programs and/or services complete their operation or are idle, the processor releases the memory and allocates the memory to another program or merges the memory within a primary memory of the computer. For every memory allocation event and memory free event, the system increments or decrements the allocation count for the particular allocation size.

The system detects 106 if there is an above-a-threshold increase in the number of allocations for one or more of the memory allocation sizes. Threshold determination will be described below with reference to Fig. 4. If yes, the system increments 108 the counter K by 1, i.e., K=K+1. For example, Fig. 2 illustrates histograms 2a, 2b, and 2c of allocation sizes (in bytes) at times t1=0, t1=1 and t1=2. The histograms depict that allocation size 30 has an increase in allocations from t1=0 to t1=3 as represented by 210, 220 and 230.

If the system does not detect 106 a notable increase in the number of allocations, the system again initializes 102 the counter K to 1 and flows through the subsequent steps continuing with step 104.

When the system detects an increase above a static or dynamic threshold in the number of allocations for a particular allocation size for at least a certain number of consecutive times (like K=5), that allocation size becomes an indicator of a suspected memory leak. Thus, the system checks 110 if the value of the counter K is greater than 5. If yes, the system marks 112 the one or more increasing allocation sizes as reflecting suspected leaks in the memory and then the system proceeds to the next tier of the multiple tier method.

However, if the system checks 110 that the value of K is less than 5, the system again collects 104 a histogram representing allocation counts for different allocation sizes of memory and flows through the subsequent steps continuing with step 106.

After certain allocation sizes are detected to have above-a-threshold increase in memory allocations, at least, for example, five times, the system initiates a second level of analysis (second tier). In the second tier, the system initializes 114 another counter L=1, as shown in Fig. 1b. The system collects 116 histograms of originating call stack traces for the memory allocation sizes that were marked 112 as suspected leaks in the first tier. A call stack is a data structure that stores information about active subroutines of a program. Subroutines include a sequence of program instructions that perform a specific task.

In a manner similar to tier 1, the system detects 118 if the call stack traces have an above-a-threshold increase for one or more suspected leak allocation sizes. Threshold determination will be described below with reference to Fig. 4. If yes, the system increments 120 the counter L by 1, i.e., L=L+1. For example, Fig. 3 illustrates histograms 3a, 3b and 3c of call stack traces at times t2=0, t2=1 and t2=2. The histograms depict that one particular call stack trace for an allocation size (e.g., allocation size 30 from Fig. 2) has above-a-threshold increase as shown from from t2=0 to t2=3 as represented by 310, 320 and 330.

When the system detects an increase above a static or dynamic threshold in the number of call stack traces for a particular allocation size for at least a certain number of consecutive times (like L=5), that corresponding call stack becomes an indicator of a suspected memory leak. Thus, the system checks 122 if the value of L is greater than 5. If yes, then the system marks 124 the one or more originating call stack traces as suspected call stacks. The system then proceeds to the third tier of the method.

However, if the system checks 122 that the value of L is less than 5, the system again collects 116 a histogram of originating call stack traces for the memory allocation sizes that were marked as suspected leaks in 112 and flows through the subsequent steps continuing with step 118.

In the third tier, the system records the allocation times of each allocation that fits the suspected leak profile, including suspected leak sizes found in tier 1 and originating call stack traces found in tier 2. As shown in Fig. 1b, the system checks 126 if the allocation size(s) and the originating call stack(s) persist as suspected leaks over a period of time. If yes, then the system marks 126 the allocation, the allocation size(s), and the originating call stack(s) as a probable memory leak. The system then reports 128 the probable memory leak along with the allocation, the allocation size(s), and the originating call stack(s) to a server.

If the allocation size(s) and the originating call stack(s) do not persist as suspected leaks over a period of time, then the system loops back to the first tier and flows through the subsequent steps continuing with step 102, as shown in Fig. 1a and Fig. 1b.

The multiple tier methodology can be implemented in any software. Fig. 4 illustrates an example table that the system generates while implementing the multiple tier based method 100 on a chrome browser application. For example, the Fig. 4 depicts memory allocation counts 410 at times t=0, t=1, t=2, and t=3 for various top allocation sizes like 40, 24, etc. The system calculates delta values for memory allocations by subtracting allocations at a time period 430 (t=2) from a previous time period 420 (t=1). Fig. 4 shows delta values in parentheses, an example is illustrated by 450. The system can generate similar table in the second tier for call stack traces counts.

Further in the table, the system, in the implementation shown, ranks the top allocation sizes by their delta values, and identifies a first major drop in deltas. A major drop may be defined as a drop of at least 50%. The system identifies the drop such that the system can locate notably (above-a-threshold) increasing entries. For example, in the data for t=1 (420) and t=2 (430), the counts for all eight allocation sizes are growing rapidly, so it is difficult to distinguish notably increasing entries from other increasing entries. However, in the data for t=3 (440), there is a drop of 50% from delta=498 to delta=169, the delta entries 40, 16, 56, and 32 (450, 460, 470, and 480) bytes are the ones that occur before this drop, and are suspected as fast growers and thus become notably increasing entries.

The subject matter described herein can be implemented in software and/or hardware (for example, computers, circuits, or processors). The subject matter can be implemented on a single device or across multiple devices (for example, a client device and a server device). Devices implementing the subject matter can be connected through a wired and/or wireless network. Such devices can receive inputs from a user (for example, from a mouse, keyboard, or touchscreen) and produce an output to a user (for example, through a display and/or a speaker). Specific examples disclosed are provided for illustrative purposes and do not limit the scope of the disclosure.

## DRAWINGS

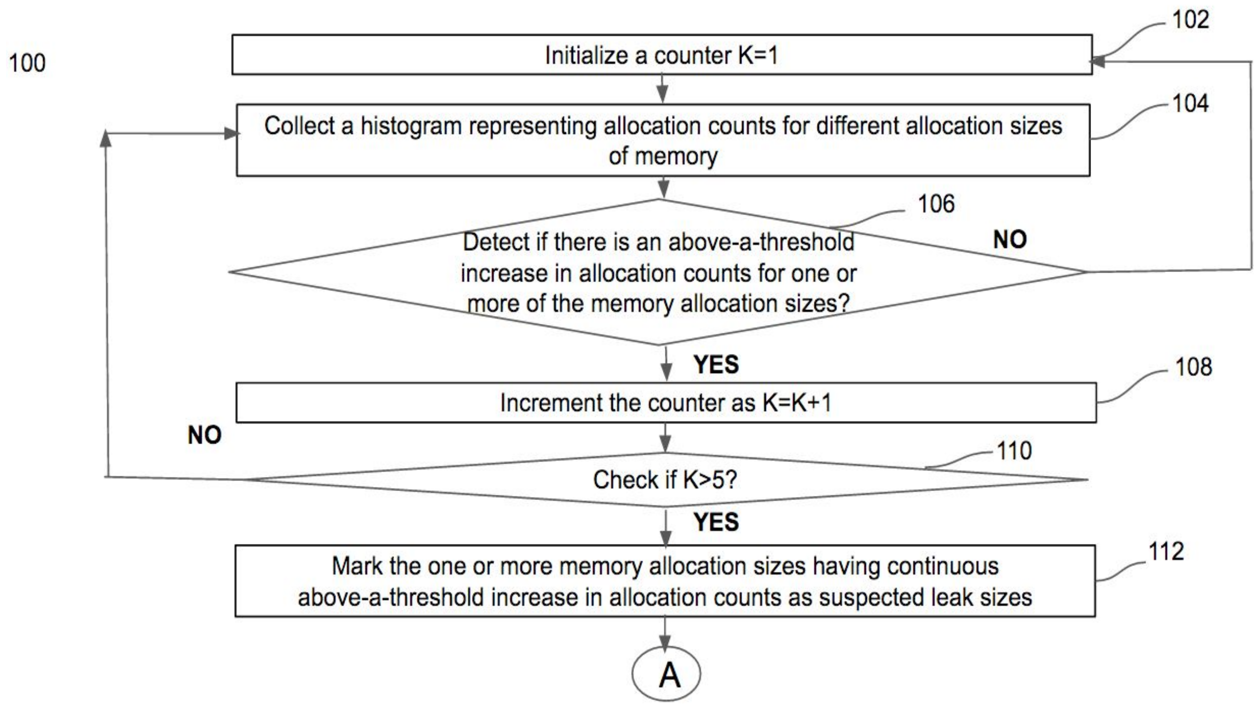

100
102
Initialize a counter K=1
104
Collect a histogram representing allocation counts for different allocation sizes of memory
106
Detect if there is an above-a-threshold increase in allocation counts for one or more of the memory allocation sizes?
NO
YES
108
Increment the counter as K=K+1
NO
110
Check if K>5?
YES
112
Mark the one or more memory allocation sizes having continuous above-a-threshold increase in allocation counts as suspected leak sizes
A

Fig. 1a

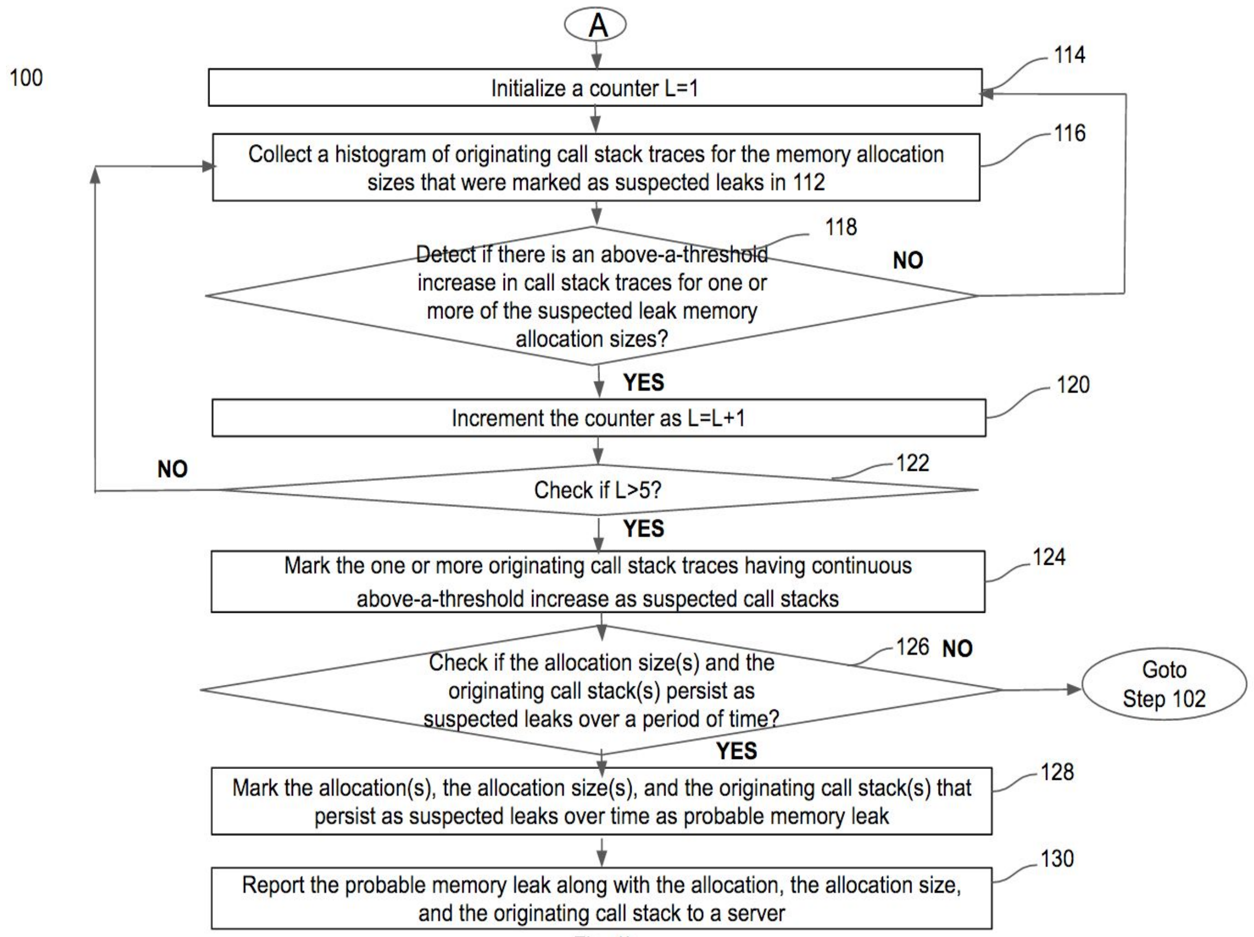

100
A
Initialize a counter L=1
114
Collect a histogram of originating call stack traces for the memory allocation sizes that were marked as suspected leaks in 112
116
Detect if there is an above-a-threshold increase in call stack traces for one or more of the suspected leak memory allocation sizes?
118
NO
YES
Increment the counter as L=L+1
120
NO
Check if L>5?
122
YES
Mark the one or more originating call stack traces having continuous above-a-threshold increase as suspected call stacks
124
Check if the allocation size(s) and the originating call stack(s) persist as suspected leaks over a period of time?
126
NO
Goto Step 102
YES
Mark the allocation(s), the allocation size(s), and the originating call stack(s) that persist as suspected leaks over time as probable memory leak
128
Report the probable memory leak along with the allocation, the allocation size, and the originating call stack to a server
130

Fig. 1b

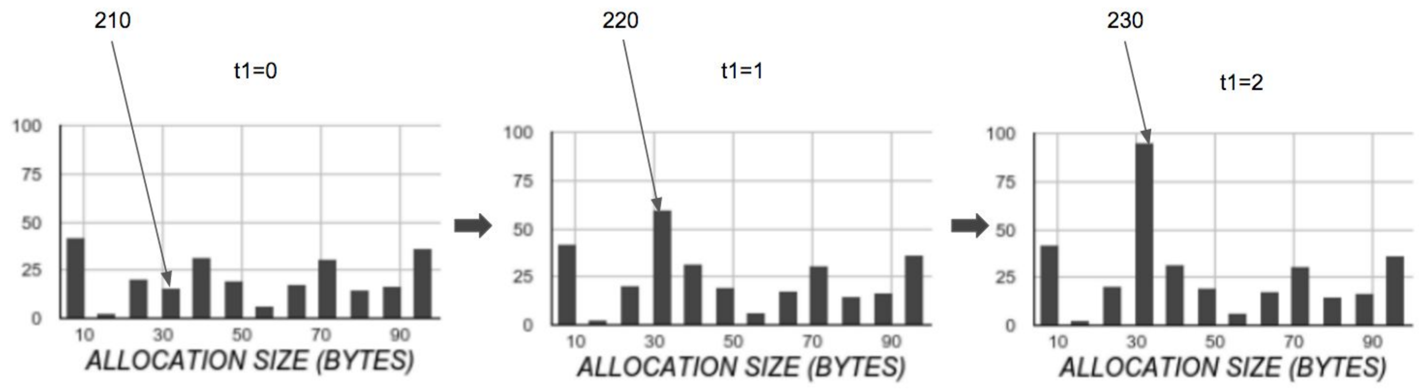


Fig. 2a

Fig. 2b

Fig. 2c

Fig. 2

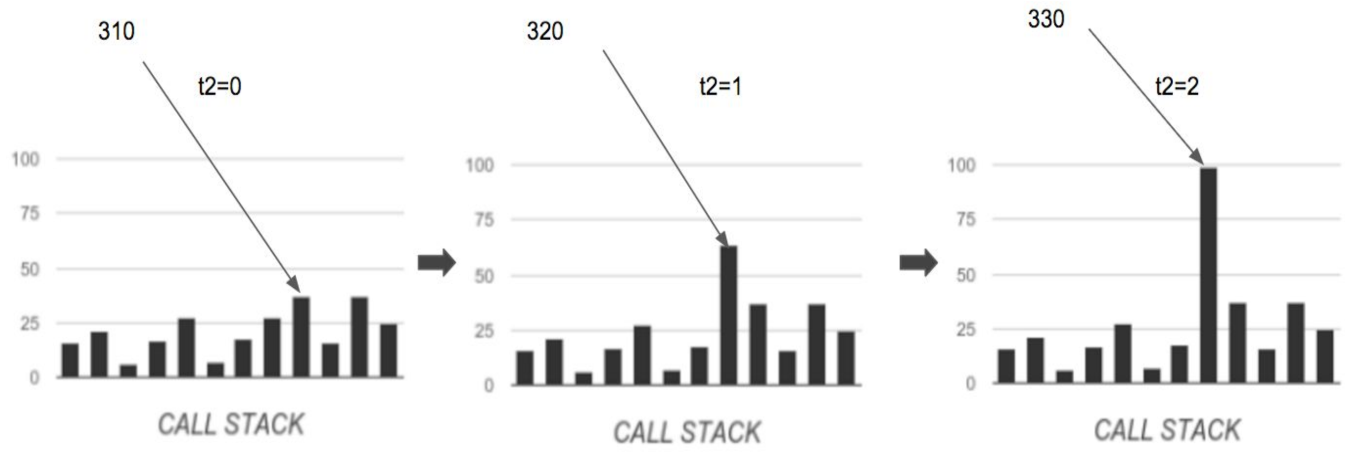
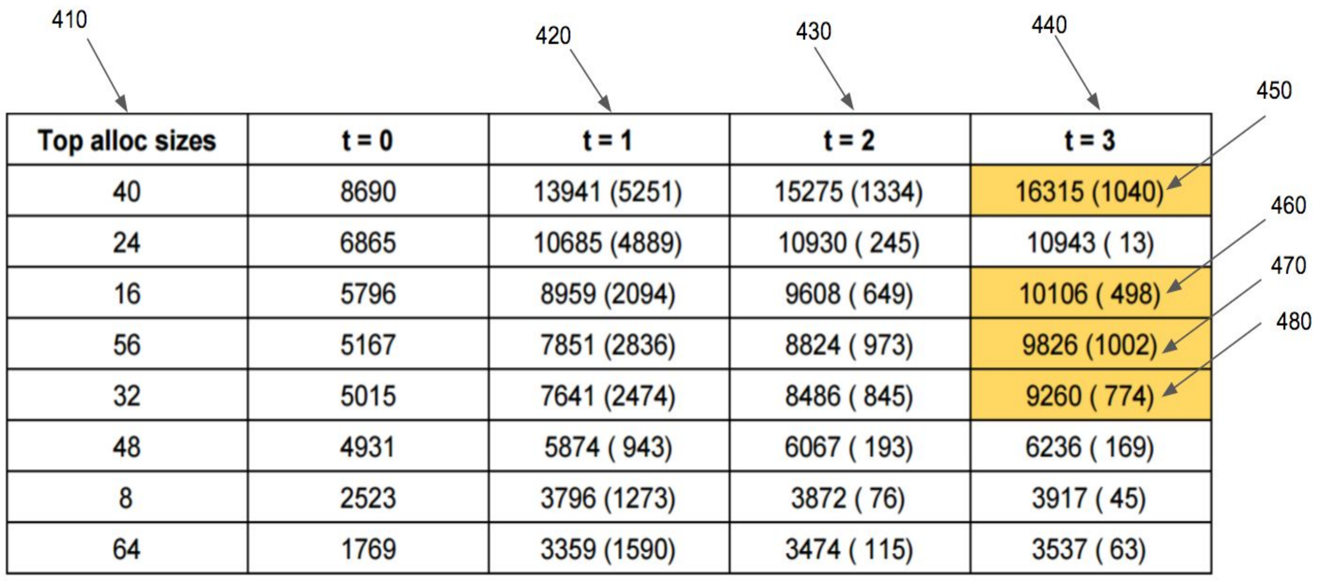


Fig. 3a

Fig. 3b

Fig. 3c

Fig. 3


410
420
430
440
450
460
470
480

| Top alloc sizes | t = 0 | t = 1 | t = 2 | t = 3 |
|---|---|---|---|---|
| 40 | 8690 | 13941 (5251) | 15275 (1334) | 16315 (1040) |
| 24 | 6865 | 10685 (4889) | 10930 ( 245) | 10943 ( 13) |
| 16 | 5796 | 8959 (2094) | 9608 ( 649) | 10106 ( 498) |
| 56 | 5167 | 7851 (2836) | 8824 ( 973) | 9826 (1002) |
| 32 | 5015 | 7641 (2474) | 8486 ( 845) | 9260 ( 774) |
| 48 | 4931 | 5874 ( 943) | 6067 ( 193) | 6236 ( 169) |
| 8 | 2523 | 3796 (1273) | 3872 ( 76) | 3917 ( 45) |
| 64 | 1769 | 3359 (1590) | 3474 ( 115) | 3537 ( 63) |

Fig. 4