US 20040003149A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2004/0003149 A1**

Engel et al. (43) **Pub. Date: Jan. 1, 2004**

(54) **DECIMATION OF FIXED LENGTH QUEUES**

(76) Inventors: **Glenn R. Engel**, Snohomish, WA (US); **Bruce Hamilton**, Menlo Park, CA (US)

Correspondence Address:
AGILENT TECHNOLOGIES, INC.
Legal Department, DL429
Intellectual Property Administration
P.O. Box 7599
Loveland, CO 80537-0599 (US)

(21) Appl. No.: **10/180,801**

(22) Filed: **Jun. 26, 2002**

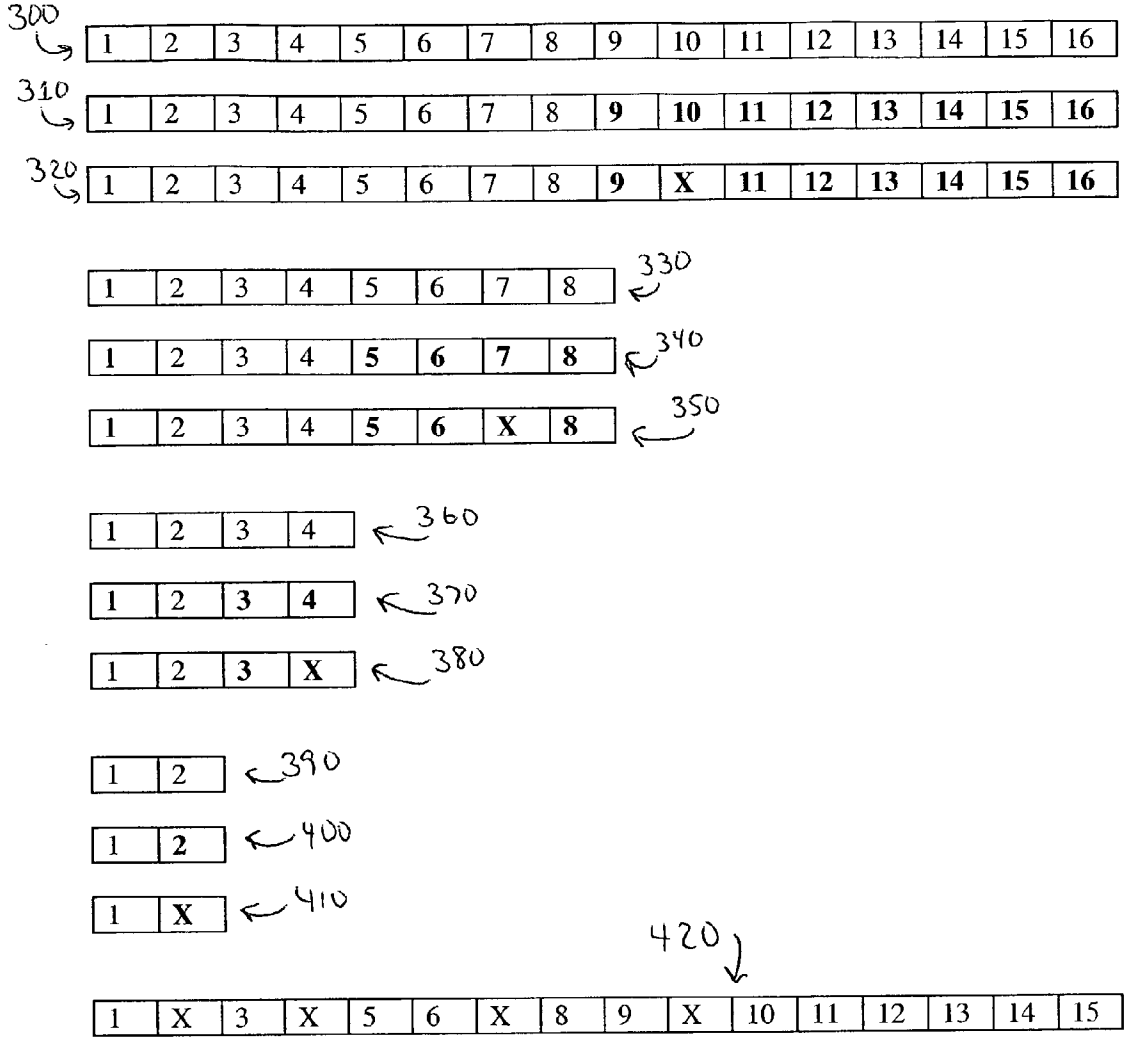
Publication Classification

(51) **Int. Cl.**[7] **G06F 3/00**

(52) **U.S. Cl.** **710/54**

(57) **ABSTRACT**

Decimation of data from a fixed length queue retaining a representative sample of the old data. Exponential decimation removes every nth sample. Dithered exponential decimation offsets the exponential decimation approach by a probabilistic amount. Recursive decimation selects a portion of the queue and removes elements.



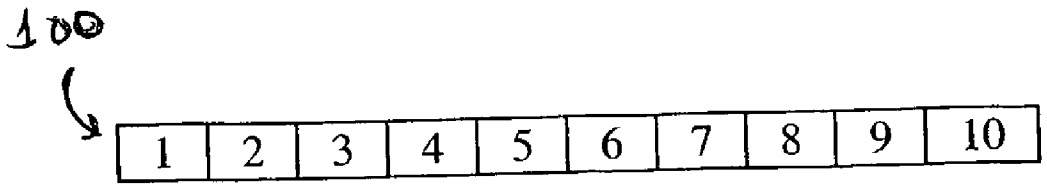

100
1
2
3
4
5
6
7
8
9
10

Fig. 1

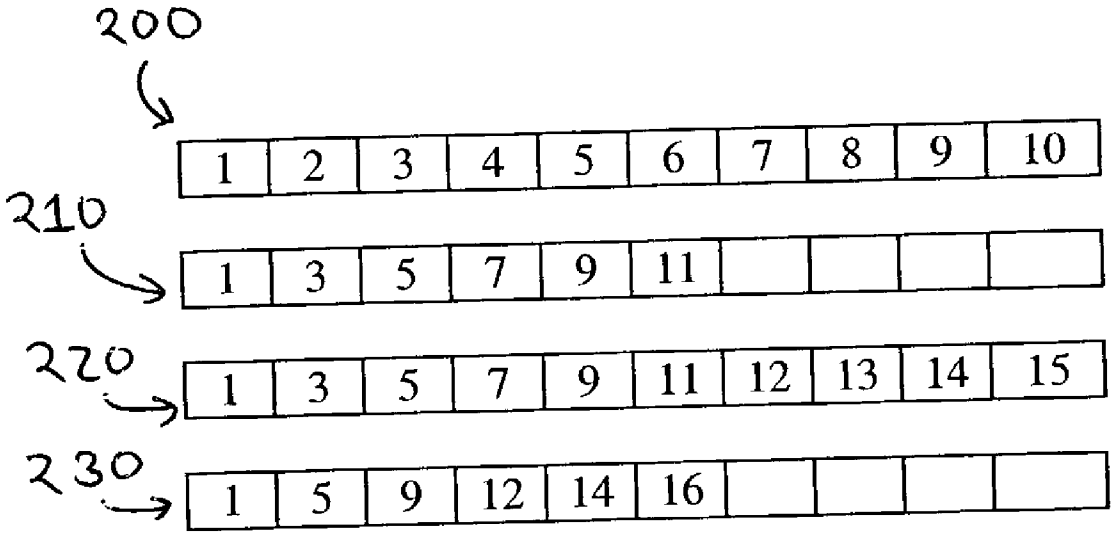

200
1
2
3
4
5
6
7
8
9
10
210
1
3
5
7
9
11
220
1
3
5
7
9
11
12
13
14
15
230
1
5
9
12
14
16

Fig. 2

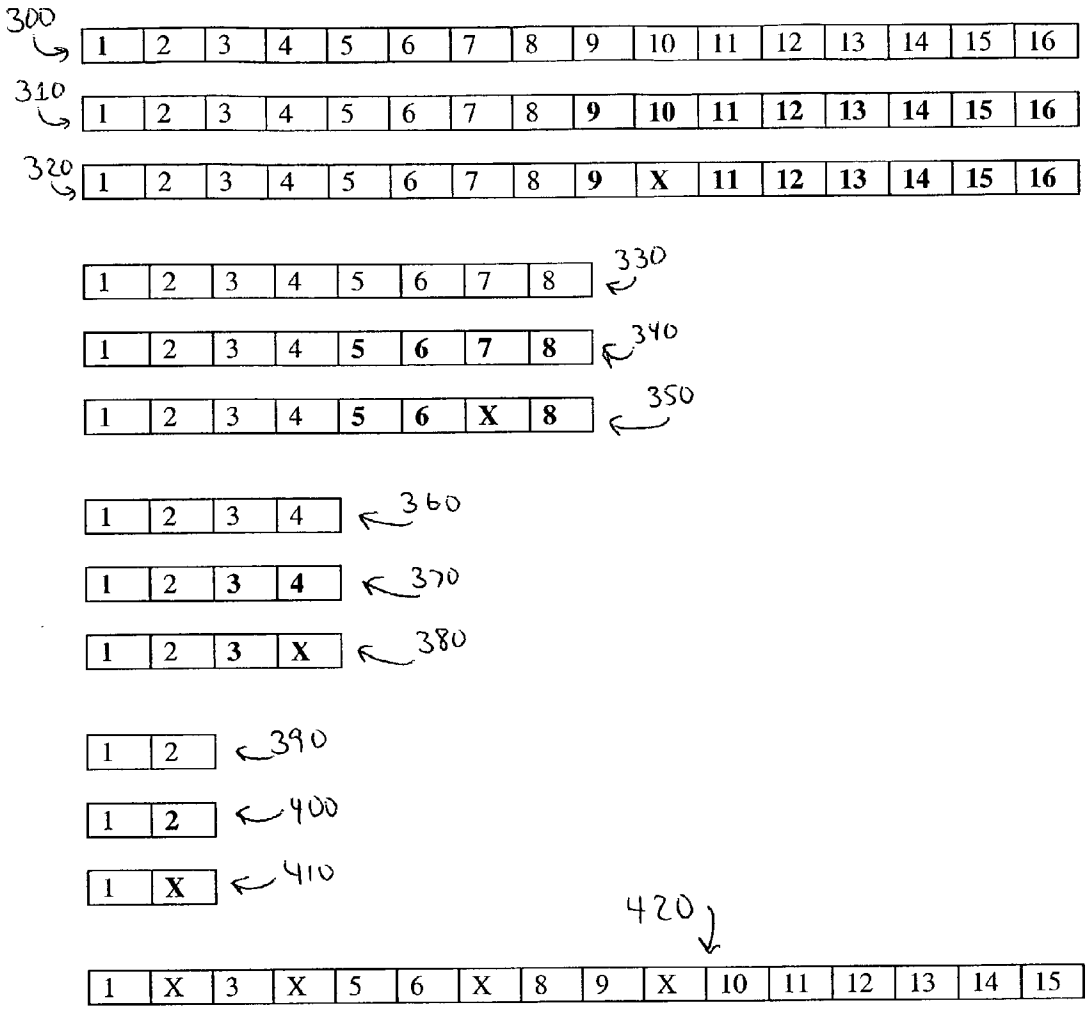

300
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
310
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
320
1 2 3 4 5 6 7 8 9 X 11 12 13 14 15 16
1 2 3 4 5 6 7 8
330
1 2 3 4 5 6 7 8
340
1 2 3 4 5 6 X 8
350
1 2 3 4
360
1 2 3 4
370
1 2 3 X
380
1 2
390
1 2
400
1 X
410
420
1 X 3 X 5 6 X 8 9 X 10 11 12 13 14 15

Fig. 3

DECIMATION OF FIXED LENGTH QUEUES

BACKGROUND OF THE INVENTION

[0001] 1. Field of the Invention

[0002] The present invention deals with fixed-length queues in computer hardware and software. More specifically, the present invention deals with techniques for managing fixed-length queues when they become full.

[0003] 2. Art Background

[0004] Many systems use fixed-length queues to buffer data between a data producer and a data consumer. If the consumer is not able to consume data as fast as the producer places it in the queue, or is unable for some reason to take data from the queue or empty the queue, the queue can become full. The queue management system, hardware or software, must have policies in place for dealing with queue overflow situations.

[0005] Traditional approaches to queue overflow include techniques such as overwriting the last item, dropping the oldest item, or discarding the newest item. The choice is usually made based on the needs of the application and the presumed importance of new data versus old data.

SUMMARY OF THE INVENTION

[0006] Techniques for decimation, the removal of old data, from a fixed length queue retain a representative sample of the data Exponential decimation removes every nth sample. Dithered exponential decimation applies an offset to the exponential decimation approach. Recursive random decimation selects a portion of the queue and removes elements, and recurses on the remaining portion of the queue.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] The present invention is described with respect to particular exemplary embodiments thereof and reference is made to the drawings in which:

[0008] **FIG. 1** shows a fixed length queue,

[0009] **FIG. 2** shows exponential decimation in a fixed length queue, and

[0010] **FIG. 3** shows recursive decimation in a fixed length queue.

DETAILED DESCRIPTION

[0011] **FIG. 1** shows a fixed length queue **100** of 10 elements, where each item in the queue is represented by its sample number. Element **1** is the oldest element in the queue, element **2** the next oldest, and so on through element **10** which is the newest. Queue **100** is full. When a new item arrives, either the new item must be discarded, or room must be made for it in the queue. Prior art solutions to adding a new item to a full queue include discarding the new item, and overwriting the most recently added item. The approach used depends on the needs of the application, and the presumed importance of old data versus new data.

[0012] Decimation as taught by the present invention trades reduced accuracy for increased apparent size of the queue. For example, if a 60 element queue contains samples taken every second, the queue when full holds samples spanning a minute, with one sample for every second in that minute. After many rounds of applying the decimation techniques according to the present invention, the same 60 element queue holds data covering a time span equivalent to that of a queue many times that size. However, the queue no longer contains a sample for each second of that time span.

[0013] The embodiments of the present invention may be implemented in a wide range of software, ranging from microcode or very low-level implementations to high-level language implementations. Embodiments of the present invention may also be implemented directly in hardware.

[0014] It should be understood that truly random numbers are very difficult to generate, and that the term random in this context is understood to be a shorthand for pseudorandom numbers. The generation of pseudorandom numbers is well understood in the art, described at length for example in Chapter 3 of *The Art of Computer Programming* by Donald E. Knuth.

[0015] Exponential decimation removes samples from the queue in such a way that old data is removed at the expense of new data, while still maintaining a representative sampling of the old data. An example of exponential decimation is shown in **FIG. 2**. Fixed length queue **200** is full. Exponential decimation by n=2 removes every second sample before adding a new item, removing items **2**, **4**, **6**, **8**, and **10** from queue **200** to produce queue **210** New samples are added until the queue once again is full, shown in **220**. Decimation is repeated and a new sample added, removing every second item, namely items **3**, **7**, **11**, **13**, and **15**, producing queue **230**. As decimation continues, the distribution of the data becomes exponential in nature.

[0016] Exponential decimation can also be applied with divisors other than n=2 and can begin with any item in the queue, effectively adjusting the exponential rate of decay of old data in the queue. While exponential decimation may be applied to a queue removing multiple elements at one time, as shown in **FIG. 2**, it may also be practiced removing one element at a time. This requires that the decimation process retain state between invocations. As an example, consider the case of a 10 element queue and divisor n=2. The first time the decimation process is called, the item in position **2** of the queue is removed. The next time the decimation process is called, the item in position **4** of the queue is removed, then the item in position **6**, then the item in position **10**, and then the item in position **2** once again. Applying the decimation process gradually in this manner essentially allows the queue to remain full at all times once it has initially been filled, eliminating old items only when necessary

[0017] Exponential decimation may also be dithered, probabilistically adding (or subtracting) a dither offset m to the sample position to be removed. At each position a probability of offsetting is calculated. As an example with the case of exponential decimation with a divisor of n=2 and an offset of m=1, samples at positions **2**, **5**, **7**, and **8** in the queue are removed, rather than positions **2**, **4**, **6**, and **8**. Dithered exponential decimation gives the same emphasis to old data, but is less susceptible to sample bias. In the general case of dithered exponential decimation where the divisor is n and the dither value is ±m, the distribution function should ideally be uniform with a zero mean, but any distribution will do.

[0018] Another method of removing data from a full queue according to the present invention is recursive deci-

mation. This is shown in **FIG. 3**. **300** shows a full queue of 16 items. Recursive decimation begins by dividing the queue in half. If the queue size is not an integer power of 2, some method can be used to make it a power of two in all rounds but the first. For example, assume the queue size is s and let $m=\lfloor \log_2(s) \rfloor$. Then the older "half" of the queue contains the oldest 2^m elements and the newer "half" contains the rest. Select the newer half of the queue, shown as **310** with items **9-16** in bold, and delete a point at random, shown in **320** with item **10** replaced by an X.

[**0019**] The process is repeated recursively with the remaining half of the queue, shown in **330**. The newer half is selected, items **5-8** in **340**. An element is deleted at random, item **7** replaced by an X in **350**. Recursive decimation continues in the same fashion with **360-380**.

[**0020**] **390-410** represent the end of the recursive process. When the queue size being examined is equal to two, one of the elements is deleted at random and the recursive process terminated.

[**0021**] The overall result of this example of random recursive decimation is shown as **420**. As with exponential decimation, recursive decimation may be applied over the entire queue, recursively decimating successively smaller portions of the queue, or it may be applied one recursive round at a time, maintaining state between rounds. Again, applying the decimation process gradually in this manner essentially allows the queue to remain full at all times once it has initially been filled, eliminating old items only when necessary

[**0022**] As stated, certain aspects of the computation are simplified if the queue length in recursive decimation is an integer power of 2. While a random number may be generated each time an element is to be deleted, if the queue size is indeed an integer power of 2, a single randomly generated number may suffice, since in a sufficiently random number all bits in a binary representation will be random.

[**0023**] As an example, consider a queue containing 64 elements. In the first recursion, a random position spanning items **33** to **64** must be selected, requiring a random number in the range of 0-31. A random number is generated and five consecutive bits (either right most or leftmost) are selected to span the range 0-31. In the next round of recursion, the range needed is 0-15, so the next 4 bits of the random number are used. The next round uses 3 bits for a range of 0-7, the following round uses 2 bits for 0-3, and the final round uses 1 bit. In total then, 5+4+3+2+1=15 bits are needed in total. In general, the number of bits k needed for a queue of size n is:

$$k = \frac{\log(n)^2 - \log(n)}{2}$$

[**0024**] This approach generates a single random number and does not reuse bits. While the possibility of introducing sample bias is increased, an alternate approach is to generate a single random number with at least the number of bits required for the first round of recursion, and reuse that random number in succeeding rounds, selecting fewer bits for each round.

[**0025**] If the size of the queue is not an integer power of 2, random numbers may be generated individually for each round of recursive decimation, or a single random number may be generated and reused in successive stages, for example by taking the random number modulo the queue size at issue in each round.

[**0026**] The foregoing detailed description of the present invention is provided for the purpose of illustration and is not intended to be exhaustive or to limit the invention to the precise embodiments disclosed. Accordingly the scope of the present invention is defined by the appended claims.

We claim:

1. The method of removing elements from a fixed-length queue, the method comprising decimation of elements from the queue.

2. The method of claim 1 where the decimation method comprises exponential decimation.

3. The method of claim 2 where exponential decimation is applied to remove multiple elements from the queue.

4. The method of claim 2 where exponential decimation is applied to remove a single element from the queue.

5. The method of claim 1 where the decimation method comprises exponential decimation with dithering.

6. The method of claim 1 where the decimation method comprises recursive decimation, recursively selecting a portion of the queue and removing a randomly chosen element.

7. The method of claim 6 where separate random numbers are used for each removal of an element from the selected portion of the queue.

8. The method of claim 6 where a single random number is generated for the recursive decimation process, each step of the process using only as many bits of the random number as needed.

9. The method of claim 6 where recursive decimation is applied to remove multiple elements from the queue.

10. The method of claim 6 where recursive decimation is applied to remove a single element from the queue.

11. An article of manufacture for performing decimation of a fixed length queue, the article of manufacture comprising:

at least one computer readable medium;

processor instructions contained on the computer readable medium, the instructions configured to be readable by at least one processor and to cause the processor to remove elements from a fixed length queue by decimation.

12. The article of claim 11, where the decimation method comprises exponential decimation.

13. The article of claim 11 where the decimation method comprises exponential decimation with dithering.

14. The article of claim 11 where the decimation method comprises recursive decimation, recursively selecting a portion of the queue and removing a randomly chosen element.

15. The article of claim 11 where the decimation method removes multiple elements from the queue.

16. The article of claim 11 where the decimation method removes a single element from the queue.

17. The article of claim 14 where separate random numbers are used for each removal of an element from the selected portion of the queue.

18. The article of claim 14 where a single random number is generated for the recursive decimation process, each step of the process using only as many bits of the random number as needed.

* * * * *