March 23, 2015

EFFICIENT SELF-ADJUSTING HASH TABLE

Geoffrey Pike

Justin Lebar

EFFICIENT SELF-ADJUSTING HASH TABLE

TECHNICAL FIELD

[0001] This disclosure generally relates to hash tables.

BACKGROUND

[0002] A hash table may be a data structure used to map keys to objects. A hash table may use one or more hash functions to compute an index into an array of buckets. However, as hash functions may compute the same index for different objects, objects may not necessarily be stored in buckets with indexes that match hashes of the objects.

SUMMARY

[0003] In general, an aspect of the subject matter described may involve a process for finding an object in a hinted hash table. A hinted hash table may be a hash table that includes a hint for each bucket that describes where an object with a key that hashes to that bucket may be found. For example, a hint for a particular bucket may indicate that the particular bucket includes an object with a key that hashes to the index of the particular bucket. In another example, the hint may indicate that a next bucket includes the object with a key that hashes to the index, e.g., the object was stored in the next bucket as the current bucket was already filled when the object was stored. The hint may be used to efficiently find an object.

[0004] Additionally or alternatively, an aspect of the subject matter described may involve a process for defending against hash flooding. Hash flooding may occur when multiple objects with keys that result in the same hash result are inserted in a hash table. A process for defending against hash flooding may include using multiple hash functions for a hash table.

[0005] Additionally or alternatively, an aspect of the subject matter described may involve a process for iterating through elements, stored objects, of the hash table. Iterating though elements of a hash table by iterating through each bucket may be

computationally expensive. For example, a single object may be stored in the last bucket of a hash table. Accordingly, a process may include determining when a hash table has a low density, e.g., when a ratio of the number of elements to a number of buckets is below a pre-determined threshold, and when the hash table has a low density, storing iteration information that may indicate where elements are stored in the hash table.

DESCRIPTION OF DRAWINGS

[0006] FIG. 1 is an illustration of example hinted hash tables.

[0007] FIG. 2 is an illustration of an example hash table using a hash flood defense.

[0008] FIG. 3 is an illustration of an example hash table with iteration information.

DETAILED DESCRIPTION

[0009] FIG. 1 is an illustration 100 of example hinted hash tables. The illustration 100 shows hash tables 110A, 110B, 110C (collectively referred to as 110) as objects are inserted, and a table 102 that describes what hints in the hash tables mean. The objects inserted into the hash table 110 are object A with a key that has a hash function output of 1, object B with a key that has a hash function output of 1, and object C with a key that has a hash function output of 2.

[0010] In the example shown in illustration 100, the hinted hash tables may include four types of hints, where hint type zero indicates that a hash list for bucket [b], the bucket for which the hint is stored, is empty, hint type one indicates that a hash list for a bucket [b] is a single element in bucket [b], hint type two indicates that a hash list for bucket [b] is a single element in bucket [b+1], e.g., the next bucket, and hint type three indicates none of the other hints are applicable.

[0011] A hash list may refer to a set of elements that have keys that hash to the same value. For example, when only object A is stored in the hash table, the hash list

for bucket [1] that corresponds to the hash of the key of object A, includes just object A. Further to the example, when both object A and B are stored in the hash table, the hash list for bucket [1] that corresponds to the hash of both the keys of object A and B, includes object A and object B, even though object B may not be stored in bucket [1].

[0012] Hash table 110A shows that when object A is inserted into an empty hash table, the hash table fills bucket [1] with object A. As the hash list for bucket [1] includes a single element in bucket [1], object A, the hint for bucket [1] may be updated to hint type one, indicating that the hash list for the bucket is a single element in that bucket.

[0013] Hash table 110B shows that when object B is inserted into hash table 110A, because the hash of the key of object B is also 1 and bucket [1] is already filled with object A, object B is stored in the next available bucket, bucket [2]. As the hash list for bucket [1] includes an element in bucket [1] and an element in bucket [2], the hint for bucket [1] may be updated to hint type three, which indicates other.

[0014] Hash table 110C shows that when object C is inserted into hash table 110B, even though no other element has a key with the same hash of two as object C, bucket [2] is already filled with object B because object A already filled bucket [1]. Accordingly, object C may be stored in the next available bucket, bucket [3]. As the hash list for bucket [2] includes a single element which is in bucket [3], object C, the hint for bucket [2] may be updated to hint type two, which indicates that the hash list for that bucket includes a single element in the next bucket, e.g., bucket [3].

[0015] When an object with a particular key is to be retrieved from the hash table 110C, the hint for the bucket that corresponds to the hash of the particular key may be used to find the object. For example, when object C is to be retrieved from the hash table, a process may determine the hash of object C is two, in response, determine that the hint for bucket [2] states that the hash list for that bucket is a single element in the next bucket, e.g., bucket [3], and provide the information stored in bucket [3]. In another example, when object A is to be retrieved from the hash table, a process may determine

that the hash of object A is one, in response, determine that the hint for bucket [1] does not provide any additional information, and in response determine if the key stored for bucket [1] matches the particular key for the object to be found.

[0016] In some implementations, different hints, fewer hints, or less hints may be used. For example, eight types of hints may be used that indicate different number of elements are included a hash list for a particular bucket. In some implementations, the hints may be stored separately from the hash table.

[0017] FIG. 2 is an illustration 200 of an example hash table using a hash flood defense. A hash list for a bucket may be considered full when a predetermined number of elements are in the hash list. For example, the hash list of bucket [1] in hash table 210A may be full as two elements are in the hash list. When a hash list for a bucket is full, a process for inserting an additional object with a key that hashes to the bucket may instead use another hash function. For example, when adding object D with a key that hashes to one into the hash table 210A, the process may determine that the hash list for the bucket [1] is full, and instead use another hash function that results in an output of zero for the key of object D, and store object D in bucket [0].

[0018] FIG. 3 is an illustration 300 of an example hash table 310 with iteration information 320. As shown, the iteration information 320 may be in the form of an array that identifies the buckets of previous and next elements stored in the hash table 310. Accordingly, the iteration information 320 may be used to quickly iterate through the elements of a hash table 310 that has a low density of elements. This iteration information may be used when a hash table has a low density, discarded when a hash table has a non-high density, and re-created as appropriate.

[0019] More details of implementation are described in the following. In the following K may refer to key type for a map, V may refer to value type for a map or set, payload may refer to a pair<K, V> or V, basic bucketload may refer to the contents of a bucket without the hint, e.g., a pointer to a payload or a T, bucketload may refer to a

hinted bucketload, a basic bucketload plus a hint, B may refer to the current number of buckets, N may refer to the current number of elements, and hash list may refer to a set of elements that have the same hash modulo B.

[0020] Open addressing may be used with hints. Without hints, each bucket may contain an integer known to be a multiple of some integer, m. The hint may be an integer in [0, m]. That may reduce sizeof(bucket). The process for finding an object x with hints may include, computing b = hash(x) mod B, where B is the number of buckets, in bucket b finding a payload plus a hint. Additional actions may include, in some order, (i) checking if the bucket is non-empty and the payload for bucket b is what is being searched for, if yes finish, and (ii) looking at the hint. If the object has not be found, the bucketload may indicate that no payloads hash to bucket b. In that case the process may determine that x isn't present and finish. If not done, the bucketload may hint something about where items that hash to bucket b are located. In general, the hint may provide complete information or no information. A standard fallback strategy such as linear probing can be implemented, for example.

[0021] If m is small, a table may be split in two to gain one bit of storage per bucket while keeping sizeof(bucket) unchanged, or split four ways to gain two bits, or eight ways to gain three bits, etc. Accordingly, m >= 4 cases may be preferable in some implementations. For m=4, hints may include 0 = hash list for this bucket is empty, 1 = hash list for this bucket is a single element in bucket b, 2 = hash list for this bucket is a single element in bucket b+1, 3 = other. “Other” may be handled by a few probes, e.g., b, b+1, b+2, and b+4, and then consulting an auxiliary data structure.

[0022] For m=8, hints may include 0 = hash list for this bucket is empty, 1 = hash list for this bucket is a single element in bucket b, 2 = hash list for this bucket is a single element in bucket b+1, 3 = hash list for this bucket is a single element in bucket b+2, 4 = probe at most 4 elements, with probe order: b, (b+1)%B, (b+3)%B, (b+6)%B, 5 = probe at most 5 elements, with probe order: b, (b+1)%B, (b+3)%B, (b+6)%B, (b+2)%B, 6 = probe at most 5 elements, with probe order: b, (b+1)%B, (b+3)%B, (b+6)%B,

(b+4)%B, other. When probing, "empty" and "deleted" may be used. These may be indicated by, for example, NULL pointers or hints.

[0023] "Hash flooding" may refer to triggering the quadratic time worst case by adversarial action. For example, for several widely used hash functions, it may be known in theory, and perhaps in practice, how to compute a set of strings that will all hash to one or a few values. To defend against hash flooding, different user specified hash functions may be used. For example, a user wanting anti-hash flooding functionality may provide a second hash function in the hasher. For keys such as "string," the second hash function may be present by default, e.g., in hash<string>. The second hash function may be one that takes a 128-bit seed. For hash<string>, the seeded hash function may be defaulted to "Apply SHA3 to the concatenation of the given string and the given 128-bit seed."

[0024] Where an item, x, is in an overfull bucket, the following could be used. Let b = hash(x) % B as usual and start at bucket b. If x is not there, note that the hint for bucket b is "other." Accordingly, search near b for x, e.g., in buckets b+2, b+3, and b+4. If x is not there, consult auxiliary data structure. If the auxiliary data structure has no information about bucket b, then x is not present and finish. Otherwise, the auxiliary data structure may say how many items have hash(item) % B == b, where they are, and what 128-bit seed to use with the secondary hash function. Use the secondary hash function to determine a bucket, b'. Use quadratic probing to search for x starting from there and ignore hints.

[0025] In cases where a density of a hash table is low, a double linked list of the N elements may be used. The list may be an extra array of length B that encodes the prev/next data. The type of "prev" and "next" may be "bucket number." When the number of elements grows beyond some threshold, the extra array may be discarded. Similarly, if deletions cause N/B to get tiny again, the extra array may be reconstructed. For an overfull hash list of a bucket, a data structure that represents the set of bucket numbers that are relevant to the hash list may be used.

ABSTRACT OF THE DISCLOSURE

Methods, systems, and apparatus, including computer programs encoded on a computer storage medium, for an efficient self-adjusting hash table. An aspect may include hash tables that include hints that describe where objects may be found in the hash table. Additionally or alternatively, an aspect may include, hash flooding of the hash tables by using multiple hash functions. Additionally or alternatively, an aspect may include iterating through elements of a hash table using iteration information that may be stored when a hash table has a low density.

100

102

Hint	Meaning
0	Hash list for bucket b is empty
1	Hash list for bucket b is single element in bucket b
2	Hash list for bucket b is single element in bucket b+1
3	other

110A

OBJECT A WITH KEY A INSERTED, HASH (A) = 1

Bucket	Hint	Key	Data
0	0		
1	1	A	Data_A
2	0		
3	0		
...	...	...	...
N	0		

110B

OBJECT B WITH KEY B INSERTED, HASH (B) = 1

Bucket	Hint	Key	Data
0	0		
1	3	A	Data_A
2	0	B	Data_B
3	0		
...	...	...	...
N	0		

110C

OBJECT C WITH KEY C INSERTED, HASH (C) = 2

Bucket	Hint	Key	Data
0	0		
1	3	A	Data_A
2	2	B	Data_B
3	0	C	Data_C
...	...	...	...
N	0		

FIG. 1

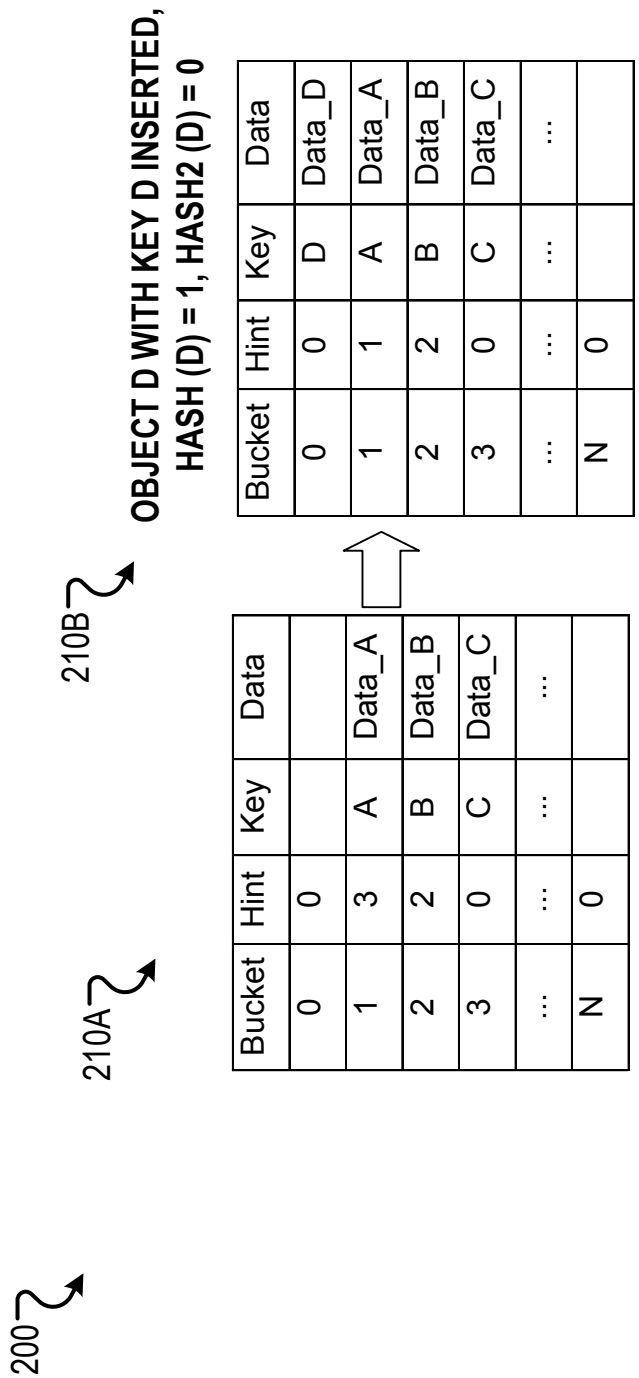
200
210A
210B
OBJECT D WITH KEY D INSERTED,
HASH (D) = 1, HASH2 (D) = 0
Bucket
Hint
Key
Data
0
0
3
A
Data_A
1
2
2
B
Data_B
2
3
0
C
Data_C
3
...
...
...
...
N
0
Bucket
Hint
Key
Data
0
0
D
Data_D
1
1
A
Data_A
2
2
B
Data_B
3
0
C
Data_C
...
...
...
...
N
0

FIG. 2

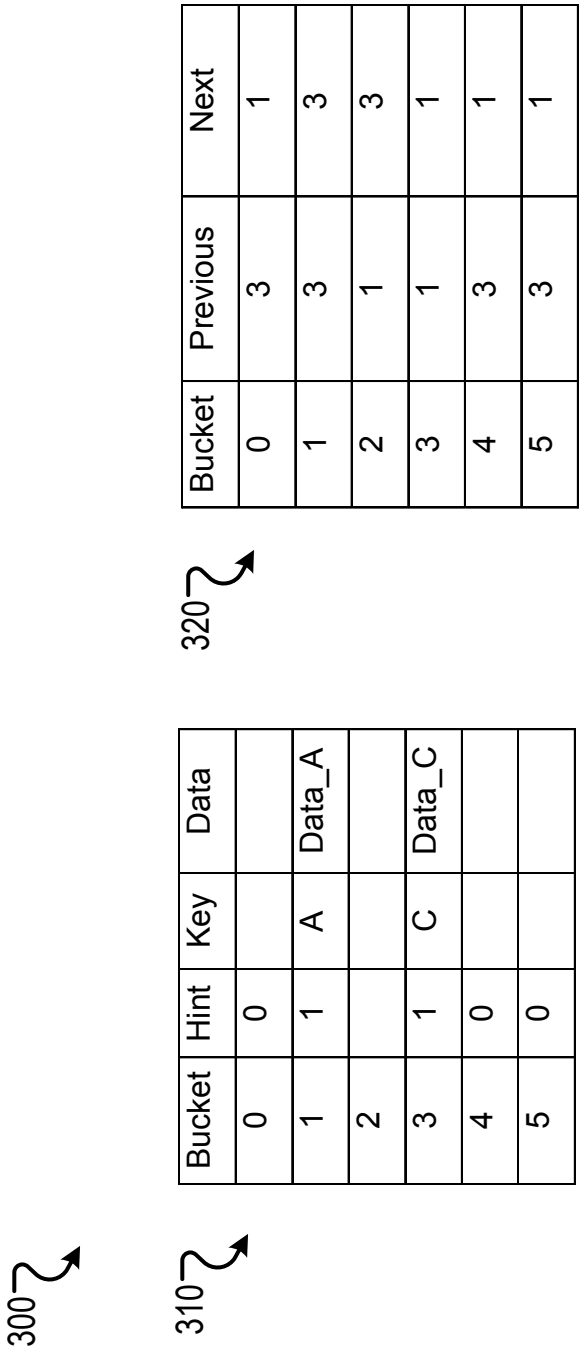
300
310
Bucket
Hint
Key
Data
0
0
1
1
A
Data_A
2
3
1
C
Data_C
4
0
5
0
320
Bucket
Previous
Next
0
3
1
1
3
3
2
1
3
3
1
1
4
3
1
5
3
1

FIG. 3