Characterizing Open Addressing Hash Functions

Wenbin Luo and Gregory L. Heileman
Department of Electrical & Computer Engineering
University of New Mexico
Albuquerque, NM 87131

Abstract
In [1], we showed that different open addressing hash functions perform differently when the data elements are not uniformly distributed. So, it is tempting to attribute their difference to some mechanism governing the behavior of the hash functions. In this paper, a simple method of characterizing open addressing hash functions is presented. We showed that, indeed, the nature of data spreading ability characterizes the behavior of different open addressing hash functions. We measured and analyzed the spreading speed of a cluster of data elements under different open addressing hash functions. Our experimental results and theoretical analysis showed that different hash functions have different abilities of spreading out clustered data elements. The hash function, which spreads out clustered data elements over the whole table space more uniformly and faster, has better performance when it is applied to clustered data. Experimental results are presented to support our claims, which is followed by some theoretic analysis.

1 Hash function families
In this paper we present hash functions in the form of nonlinear dynamical systems. So, first we derive dynamical system expressions for different hashing families.

1.1 Linear and quadratic hashing
The family of linear hash functions can be expressed as

\[ H_L(k, i + 1) = (h(k) + c_1 i + c_1) \mod m, \]  

where \( h \) is an ordinary hash function, and \( i = 0, 1, \ldots \) is the probe number. This technique is known as linear hashing because the argument of the modulus operator is linearly dependent on the probe number. In general, \( c_1 \) needs to be chosen so that it is relatively prime to \( m \) if all slots in the hash table are to be examined by the probe sequence.

In order to construct an equivalent transformation for equation (1), it must be rewritten as a recurrence relation so that it has explicit dependence on previous values of the iteration sequence. Since we have

\[ H_L(k, i + 1) = (h(k) + c_1 i + c_1) \mod m, \]

and using the fact that for \( a, b, m \in \mathbb{R} \),

\[ (a + b) \mod m = (a \mod m + b \mod m) \mod m, \]  

we may rewrite equation (1) as the following linear time-invariant first-order iterator

\[ H_L(k, i + 1) = (H_L(k, i) + (c_1 \mod m)) \mod m \]

\[ H_L(k, 0) = h(k). \]

Note that the dependence on \( k \) is specified in the initial condition \( H(k, 0) \).

Quadratic hashing is a simple extension of linear hashing that makes the probe sequence nonlinearly dependent on the probe number. For any ordinary hash function \( h \), the family of quadratic hash functions is given by

\[ H_Q(k, i + 1) = (h(k) + c_1 i + c_2 i^2) \mod m, \]

where \( c_1 \) and \( c_2 \) are positive constants. Once again, the specific values chosen for the constants are critical to the performance of this method (see [2] for details).

To obtain a recurrence relation solution to equation (3) we note that

\[ H_Q(k, i + 1) = [(h(k) + c_1 i + c_2 i^2) + c_1 + c_2 (2i + 1)] \mod m \]

which leads to the time-varying first-order recurrence relation

\[ H_Q(k, i + 1) = (H_Q(k, i) + c_1 + c_2 (2i + 1)) \mod m \]

\[ H_Q(k, 0) = h(k). \]  

21
1.2 Double hashing family $H_D$

It is well known that linear and quadratic hashing strategies suffer from clustering, because both have probe sequence increments that are independent of the key. Double hashing remedies this problem by introducing a second hash function that is used in the computation of the increment.

Given two hash functions $g$ and $h$, the family $H_D$ of linear double hash functions is given by:

$$H_D(k, i) = (g(k) + ih(k)) \mod m. \quad (5)$$

In this family, the initial probe $H_D(k, 0) = g(k)$, and successive probes are offset from previous probes by multiples of $h(k)$ modulo $m$. Thus the probe sequence depends on $k$ through both $g$ and $h$, and is linear in $g(k)$ and $h(k)$. A widely used member of $H_D$, proposed by Knuth [1973], has $g(k) = k \mod m$ and $h(k) = k \mod (m-2)$, where both $m$ and $m-2$ are prime.

A recurrence relation for the family $H_D$ described by equation (5) is obtained by first noting that

$$H_D(k, i + 1) = [(g(k) + ih(k)) + h(k)] \mod m,$$

which leads to

$$H_D(k, i + 1) = (H_D(k, i) + h(k)) \mod m \quad (6)$$

$$H_D(k, 0) = g(k). \quad (7)$$

Notice that for a given key $k$, $h(k)$ is constant over a single probe sequence, but may change from one probe sequence to the next.

1.3 Exponential hashing family $H_E$

Exponential hashing [3] uses two ordinary hash functions $g$ and $h$ to compute a probe sequence according to

$$H_E(k, i) = (g(k) + h(k)^i) \mod m. \quad (8)$$

To obtain the desired transformation for the family $H_E$ described in equation (8), we write

$$H_E(k, i + 1) = (h(k)h(k)^i + g(k)) \mod m$$

$$= [((h(k)^i + g(k))h(k)) \mod m] \mod m$$

$$+((1 - h(k))g(k)) \mod m \mod m$$

and the transformation we obtain for $H_E$ is

$$H_E(k, i + 1) = (H_E(k, i)h(k) + (1 - h(k))g(k)) \mod m$$

$$H_E(k, 0) = g(k) + 1. \quad (9)$$

1.4 Our new hash function $H_N$

We proposed a new hash function recently [4], that computes a probe sequence according to

$$H_N(k, i) = g(k) + a'h(k) \mod m. \quad (10)$$

where $k$ is the key, $g(k)$ and $k(k)$ are ordinary hash functions which return values in the range $[0, m-1]$, $i$ is the probe index, $m$ is the table size, and $a$ is a constant. $H_N(k, 0) = g(k)$. The probe sequence is not of length $m$ for all values of $a$. But as we proved in [4], if $a$ is a primitive root of the prime $m$, the probe sequence will be full length.

To obtain the desired transformation for the family $H_N$ described in equation (10), we write

$$H_N(k, i + 1) = a(g(k) + h(k)a^i)$$

$$+g(k)(1-a) \mod m$$

and the transformation we obtain for $H_N$ is

$$H_N(k, i + 1) = aH_N(k, i) + g(k)(1-a) \mod m$$

$$H_N(k, 0) = g(k). \quad (11)$$

2 Experimental results

In general, better a hash function performs, faster it mixes up a cluster of data elements (keys) over the whole table space. So, the speed of convergence of the clustered data to the uniform distribution characterizes the behavior of hash functions. Based on the above observation, the approach we will take to characterize different hash functions is based on the speed of convergence to uniform distribution under different hash functions.

In thermodynamics, when a system of gas molecules is in equilibrium state, the gas molecules are uniformly distributed over the whole space, i.e., the average distance between all molecules is maximized. So, in order to measure when a cluster of initial keys approaches the uniform distributions under different hash functions, we may measure how the average distance among them changes with the number $n$ of iterations. Faster the average distance gets close to the maximal value, stronger is the ability of hash functions to spread out the clustered keys. For a good hash function, we expect that a cluster of initial keys will be spread over the whole table space with large average distance between all mapped slots after some steps of iterations in order to avoid clustering problem.

Suppose that we have a cluster of initial keys $k_1^0, k_2^0, \ldots, k_T^0$. After the first iteration, they are
is 5087, and a few (cluster of keys over the whole table space after only initial hashing and our new hash function spread out a
age distance among all the keys corresponding to our the clustered keys. Also, we noticed that the aver-
ber of iterations (which shows the average distance (

The number of iterations
Average distance
Quadratic hashing    
Double hashing       
Exponential hashing  
Our new hash function
Uniform distribution

3. Analysis

Next, we analyze the spreading speed of a pair of adjacent initial keys for quadratic hashing, double hashing, exponential hashing and our new hash function. Please note that all the following calculations and conclusions hold up to the mod m operation.

3.1 Quadratic hashing family \( H_Q \)

From equation (4), we see that for a key \( k \in \mathbb{Z} \), after \( n \) iterations, we have,

\[ H_Q(k, n) = H_Q(k, 0) + n(n-1)c_2 + n(c_2 + c_1) \] (12)

From equation (12), it is easy to see that for two different keys \( k_1 \) and \( k_2 \), \( k_1 \neq k_2 \), the distance \( \Delta Y_n \) between \( H_Q(k_1, n) \) and \( H_Q(k_2, n) \) is,

\[ \Delta Y_n = H_Q(k_1, n) - H_Q(k_2, n) = h(k_1) - h(k_2) \] (13)

From equation (13), we see that \( \Delta Y_n \) is fixed, which means that the distance between two probe sequences generated by two different keys \( k_1 \) and \( k_2 \), is constant no matter how \( n \) is changed. So, for a cluster of initial keys, the average distance among all the keys is constant, independent of the number of iterations \( n \). The experimental result in Fig. 1 verifies our analysis.

3.2 Double hashing family \( H_D \)

From equations (7), we see that for a key \( k \in \mathbb{Z} \), after \( n \) iterations, we have,

\[ H_D(k, n) = H_D(k, 0) + n \cdot h(k) \mod m \] (14)

From equation (14), it is easy to see that for two different keys \( k_1 \) and \( k_2 \), \( k_1 \neq k_2 \), the distance \( \Delta Y_n \) between \( H_D(k_1, n) \) and \( H_D(k_2, n) \) is,

\[ \Delta Y_n = H_D(k_1, n) - H_D(k_2, n) = n \cdot (h(k_1) - h(k_2)) + g(k_1) - g(k_2) \] (15)

From equation (15), we see that \( \Delta Y_n \) is linear in \( n \), which means that the distance between the two probe sequences generated by two different keys \( k_1 \) and \( k_2 \), is linear with the number of iterations. So, for a cluster of initial keys, the average distance among all the keys is changed linearly during the first several iterations. The experimental result in Fig. 1 shows that for the first 90 iterations, the average distance among all the keys is linear to the number of iterations \( n \). After that, the average distance approaches the maximal distance.

3.3 Exponential hashing family \( H_E \)

From equations (9), we see that for a key \( k \in \mathbb{Z} \), after \( n \) iterations, we have,

\[ H_E(k, n) = h^n(k) \cdot (g(k) + 1) + \frac{h^n(k) - 1}{h(k) - 1} \cdot g(k) \cdot (1 - h(k)) \] (16)
From equation (16), for two different keys \( k_1 \) and \( k_2, k_1 \neq k_2 \), the distance \( \Delta Y_n \) between \( H_E(k_1, n) \) and \( H_E(k_2, n) \) is,

\[
\Delta Y_n = H_E(k_1, n) - H_E(k_2, n)
\]

\[
= h^n(k_1) - h^n(k_2) + g(k_1) - g(k_2)
\]

\[
= h^n(k_1) - (h(k_1) + h(k_2) - h(k_1))^n
+ g(k_1) - g(k_2)
\]

\[
= h^n(k_1) - h^n(k_1) \cdot \left( 1 + \frac{h(k_2) - h(k_1)}{h(k_1)} \right)^n
+ g(k_1) - g(k_2)
\]

or

\[
\approx h^n(k_1) - h^n(k_1) \cdot \left( 1 + n \cdot \frac{h(k_2) - h(k_1)}{h(k_1)} \right)
+ g(k_1) - g(k_2)
\]

\[
= n \cdot \left( h(k_1) - h(k_2) \right) \cdot h^{n-1}(k_1)
+ g(k_1) - g(k_2)
\]

\[\text{(19)}\]

Equation (18) holds when \( (h(k_2) - h(k_1)) \ll h(k_1) \). That is, for two very close initial keys, the distance between them will change according to equation (19).

Comparing equation (19) with equation (15), we see that due to the exponential term \( h^{n-1}(k_1) \), the distance \( \Delta Y_n \) between two very close initial keys of exponential hashing spreads out much faster than that of double hashing. The experimental result in Fig. 1 shows that after only a few (<5) iterations, the average distance among all the keys approaches the maximal distance.

### 3.4 Our new hash function \( H_N \)

From equations (11), we see that for a key \( k \in \mathbb{Z} \), the dynamical system \( H_N(k, i + 1) = a H_N(k, i) + g(k)(1 - a) \mod m \) generates the probe sequence for the key with \( H_N(k, 0) = g(k) \).

Using similar calculations as in the subsections above, for two different keys \( k_1 \) and \( k_2, k_1 \neq k_2 \), the distance \( \Delta Y_n \) between \( H_N(k_1, n) \) and \( H_N(k_2, n) \) is,

\[
\Delta Y_n = H_N(k_1, n) - H_N(k_2, n)
\]

\[
= a^n (h(k_1) - h(k_2)) + g(k_1) - g(k_2)
\]

\[\text{(20)}\]

Similar to exponential hashing, due to the exponential term \( a^n \), the distance \( \Delta Y_n \) between two very close initial keys spreads out much faster than that of double hashing. The experimental result in Fig. 1 shows that after only a few (<5) iterations, the average distance among all the keys approaches the maximal distance.

In summary, the rates of separation of nearby probe sequences are as follows for quadratic, double, exponential and our new hash functions, respectively.

\[
\Delta Y_n = h(k_1) - h(k_2)
\]

\[
\Delta Y_n = n \cdot (h(k_1) - h(k_2)) + g(k_1) - g(k_2)
\]

\[
\Delta Y_n = n \cdot (h(k_1) - h(k_2)) \cdot h^{n-1}(k_1) + g(k_1) - g(k_2)
\]

\[
\Delta Y_n = a^n (h(k_1) - h(k_2)) + g(k_1) - g(k_2)
\]

Where \( \Delta Y_n \) denotes the distance between two initially close points after \( n \) iterations. The formulas above explain very well the experimental results in Fig. 1, and characterize different hashing families.

### 4 Conclusions

In this paper, we characterized the performance of open addressing hash functions. We studied the speed of spreading out clustered keys by different hash functions. Both experimental results and theoretic analysis showed that the performance of hash functions is closely related to their capability of spreading out clustered keys. In general, faster a hash function spreads out clustered keys, better it performs. This result points us another way to study, analyze, and compare the performance of different hash functions.

### References


