Adaptive Hashing for IP Address Lookup in Computer Networks

Christopher Martinez and Wei-Ming Lin
Department of Electrical and Computer Engineering
The University of Texas at San Antonio
San Antonio, TX 78249-0669, USA

Abstract— **For applications that rely on large databases as the core data structure, the need for a fast search process is essential. Hashing algorithms have widely been adopted as the search algorithm of choice for fast lookups. Hashing algorithms involve the creation of hash values from the target database entries. A hashing algorithm that transforms the database to hash values with a distribution as uniform as possible would lead to a better search performance. When a database is already value-wise uniformly distributed, any regular hashing algorithm, such as bit-extraction, group-XOR, etc., will lead to a statistically perfect hashing result. In almost all known practical applications, the target database rarely demonstrates uniformly distributed characteristic. The use of any known regular hashing algorithm can lead to a performance far less than desirable. This paper aims at designing a hashing algorithm that can deliver a better performance for all practical databases. An analytical preprocess is performed on the original database to extract critical information that would significantly benefit the design of a better hashing algorithm. The process includes sorting database hash bits to provide a priority that would facilitate the decision-making on which bits and how these bits should be combined to generate better hash values. The algorithm follows an ad hoc design that is critical to adapting to real-time situation when there exists a changing database with an irregular non-uniform distribution. The proposed technique clearly outperforms all known regular hashing algorithms by a significant margin.**

I. Introduction

Extensive advancements in the past few decades have changed the design of computer and communication networks significantly. Currently this area has reached high levels of transfer speeds and bandwidth to become the backbone to all industries. The area of computer and communication networks can further advance with a new research that looks beyond the physical advances of transmission speed. A common critical development need for any network components, such as routers, firewalls, intrusion detection, etc. is in fast searching through the local databases. Network components nowadays in general involve the search through databases of IP address space and/or large sets of intrusion detection rules which must meet real-time limitations. In a general form, this problem involves a search process through a large database to find a record (or records) associated with a given value. Such a matching process usually is carried out through a hashing algorithm to reduce the otherwise potentially excessively long search time.

Basically, hashing is a process that allows the search to go through a statistically smaller number of steps than a simple sequential straightforward search would have performed. A hash function, usually a mathematical one, maps a number with a large value range into another number with a smaller range. For example, a simplified one as shown in Figure 1, a database of eight given records are to be matched against for any incoming record. Due to the large size of each record and potentially large number of records in the database under real situations, searching through the whole database one at a time could be merely impractical. One may choose to use

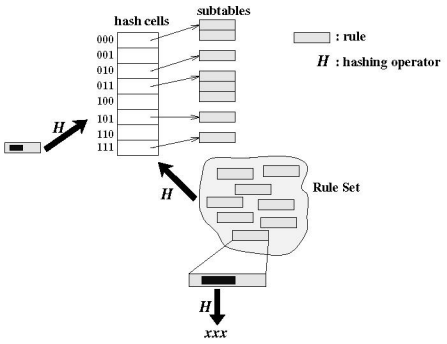

Fig. 1. A Hashing Example

a portion of the record (or its entirety to hash into the target value (a three-bit value as shown in this example) using the hash function (operator) H. Therefore, a database of eight records are now grouped into bins of records according to their corresponding hash results. With this, any incoming record would go through the same hashing to identify the one bin it would need to search through, instead of the whole database. Perfect hashing would guarantee that every bin contains exactly one record, which leads to a search process of exactly one matching to take place. A hash function is considered better than the other if it leads to a smaller expected number of matching steps required. The way to improve the performance of hashing is to minimize the number of database entries which share the same hash value (hash collision). Researchers in the past have developed hashing algorithm based on the premise that all entries in the database are distributed uniformly. The premise of uniform distribution reflects that any *regular* hash function used, e.g. non-overlapped XOR, bit extraction, etc., will create the same expected number of collision for each hash value. For practical applications the premise of uniform distribution usually does not hold true.

0-7803-9746-0/06/$20.00©2006 IEEE

There have been many schemes developed for the IP address lookup problem using a hash function. [9] gives a complete survey and complexity analysis on the IP address lookup problem. Performance comparison among regular hashing functions such as traditional XOR folding, bit extraction, CRC-based functions can be found in [3]. While most regular hash functions, simple XOR folding and bit extraction, are inexpensive to implement the performance of such functions is far from desirable. Other researchers have examined the use of a faster cache memory [5], [6] to improve performance. In software applications the main increase in performance is through the use of efficient data structures [7], [10]. Other hashing algorithms have also been widely adopted to provide for the address look-up process [1], [2], [8], [12]. All hashing algorithms inevitably suffer from unpredictable complexities involving conflicts among the data with the same hash result. It is these hash collisions which result in a long searching time that can exceed the time limitations imposed by design specifications. In this paper a new way of hashing based on bit-XORing operation will be developed to improve search times. Our algorithm will not be restricted to the standard assumption that entries in a database are value-wise uniformly distributed. The proposed algorithm involves an analytical preprocessing step of determining the distribution characteristic of the database and then re-organizes the database for a better manipulation. The algorithm then adopts an ad hoc approach to find the best combination of bits to XOR.

II. Proposed Hashing Algorithm for Arbitrary *pdf*

Regular hashing functions can achieve perfect hashing when applied to databases with a uniform distribution. In reality, uniformly distributed databases may never occur. Database often have irregular non-uniform distribution that cannot be easily modelled by any known mathematical distribution function. In addition, it is uncommon for a database to remain constant during its lifetime. That is, over time the database can expand and/or contract as the application enters new states of execution. The goal of this paper is to develop a hashing algorithm based on an ad hoc design that can adapt to the expanding or changing database. Our proposed hashing algorithm is composed of two parts: 1) preprocessing and reorganizing the database and 2) finding the optimal XORing pattern among the bits.

A. Preprocessing of Database

The proposed hashing algorithm depends entirely on the distribution characteristics of the database. In this paper we choose to use bit-wise distribution along bit vectors in the database. The database is defined as consisting of $M = 2^m$ entries with each entry having n bits in length. In order to render the best (uniform) distribution in the final hashed data set, all the bits in the final hashing function H should demonstrate a distribution as probabilistically random as possible, i.e. evenly distributed between 0's and 1's. An optimal H will have each of its bits demonstrate even distribution of 0's and 1's, and thus leading to the highest probability in reaching the best hashing. The ad hoc parameter used for adaptation in this paper is identified to be the binary distribution of each bit in the n-bit entry value length. For bit position i, d_i is defined as the absolute difference between the number of 0's and 1's in that bit vector across the data set. Once the d value is found for each bit vector as shown in Figure 2(a), they are then sorted into a non-decreasing order as shown in Figure 2(b). Note that, while sorting is performed on the d values, no actual

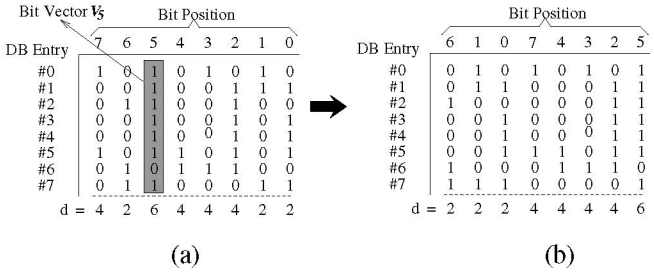

Fig. 2. (a) Database with d Values Found (b) Database Sorted By d

re-arrangement of the database is needed; instead, an array of the sorted bit indices is used as the hash function. Assume the bit sequence before sorting is $(b_{n-1}, b_{n-2}, \ldots, b_1)$ and the bit sequence afterwards is $(s_{n-1}, s_{n-2}, \ldots, s_0)$. A bit vector V_i with $d_i = 0$ indicate that there are even number of 0's and 1's; while, a $d = M$ reflects that all the bits in the vector have the same value. Translated to the distribution from hashing, a bit of $d = 0$ gives an even hashing distribution (i.e. evenly divided) among the entire hash space while a $d = M$ hashing will result in hashing to only one half of the hash space. Intuitively, using the bits with smaller d values for hashing would lead to a probabilistically better hash distribution, i.e. less conflict in the final mapping. Ideally, if one can identify (or through XORing to obtain) m bits with all their d values equal to 0, it should lead to the best potential performance, assuming no correlation among the bit vectors.

B. Ad Hoc Pivot Hashing

The goal of the proposed *ad hoc pivot* hashing algorithm is to identify which bit positions and in what combination should be used for XORing to minimize the d values. XOR operator has been widely used for hashing and known to be an excellent operator in enhancing randomness in distribution. It also possesses a nice characteristic allowing for analytical performance analysis and thus better algorithm design. XOR-folding is a commonly used hashing technique by simply folding the n-bit key into m-bit hash result through a simple process XORing every $\frac{n}{m}$ key bits into a final hash bit. To precisely quantify the benefit in using two bits for XORing with their d values being d_i and d_j, a formula can be derived to find the expected resultant d for the new bit after XORing, denoted as $d_{[d_i \oplus d_j]}$:

$$d_{[d_i \oplus d_j]} = \sum_{k=0}^{x_j} |M - 2\Delta - 4k| \cdot \frac{C_{x_j-k}^{x_i} \cdot C_k^{M-x_i}}{C_{x_j}^{M}}$$

where $x_i = \frac{M - d_i}{2}$,$x_j = \frac{M - d_j}{2}$,$\Delta = x_i - x_j$

Note that the term $\frac{C_{x_j-k}^{x_i} \cdot C_k^{M-x_i}}{C_{x_j}^M}$ indicates the probability for the resultant vector to have a d value equal to $|M-2*\Delta-4k|$. A complete spectrum of $d_{[d_i \oplus d_j]}$ for all possible integral values of d_i and d_j for $M = 32$ is given in Figure 3. It is important

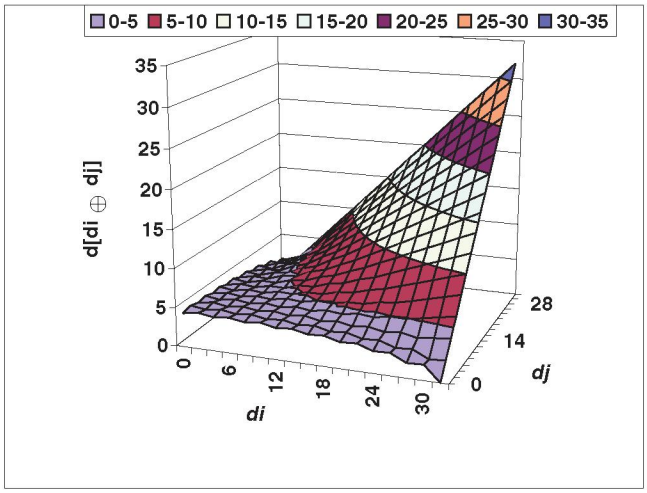

Fig. 3. Spectrum of $d_{[d_i \oplus d_j]}$ Values

to note that for some combinations of d_i and d_j, the resultant d value will be greater than the smaller of d_i and d_j. Thus, in order to reach the best solution, a hashing algorithm should avoid XORing any two bits that would lead to a larger d value. Maximizing the percentage of reduction in d value should be the main deciding factor in which bits to XOR. The percentage of reduction, denoted as p_{ij}, between d_i and d_j is determined by the following formula:

$$p_{ij} = \frac{min(d_i, d_j) - d_{[d_i \oplus d_j]}}{min(d_i, d_j)}$$

. Figure 4 shows the percentage of reduction when XORing d_i and d_j for $M = 64$. From the plot one can see a range of values that result in a negative change. This region of d_i and d_j will be absolutely avoided in XORing to maintain a non-destructive property in hashing. The plot also shows a pyramid shape where the topmost point indicates the greatest decrease in d, which we in turn define as the *pivot point*. This pivot point will give information to the algorithm on which bits should be XORed to maximize the overall performance.

The ad hoc pivot hashing algorithm operates by taking all n entry bits, after being sorted, as the input and in each intermediate step two bits are selected for XORing until either (a) m final hash bits are left or (b) any more XORing would lead to destructive result on the d values. Note that, the first m bits in any intermediate step are the ones with m smallest d values. That is, XORing of bit i and bit j would render the bit sequence resorted to reflect the new order with the new d value inserted accordingly in the sequence. In each intermediate step, the selection of the two bits for XORing is determined by

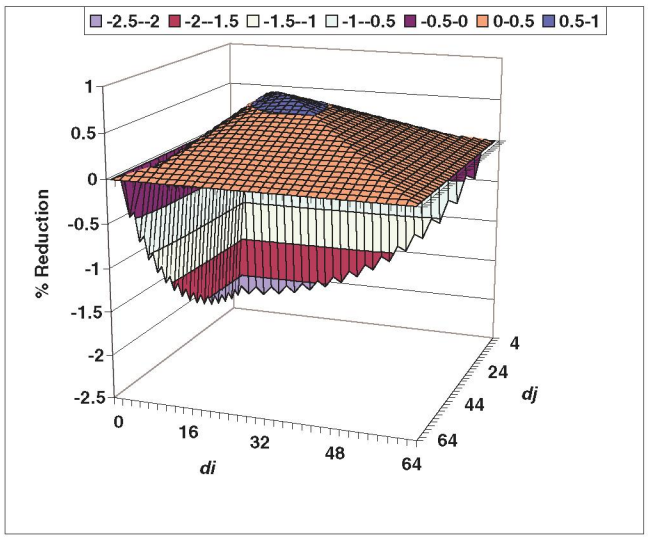

Fig. 4. Distribution of Percentage of Reduction p_{ij}

examining two different bit locations. The first location to be examined is in the position close to the pivot point, named as the pivot point location, to find the largest reduction in d. The other location examined is the $(m+1)$-th bit location, named as "$m+1$" location. Examination of this second location is needed to avoid having to include bits of large d into the first m-bit group at the end. This scenario happens when the pivot point is already within the first m bits and thus by XORing two bits around the pivot point may instead increase the overall d values within the first m bits since the original $(m+1)$-th bit would be included in. When XORing bits the resultant d value after XORing will be rounded to the nearest even integer so that the percentage of reduction can be pre-calculated and accessed from a lookup table.

When examining the pivot point location and "$m+1$" location to determine which bits should be XORed, each target bit is paired with its neighbors to look for the greatest reduction in d. The reason that only neighbors are examined is from an insight learned from Figure 4. For example, examining the "$m+1$" location refers to the checking of pairings between m-th and $(m+1)$-th bits and between $(m+1)$-th and $(m+2)$-th bits. Examining the pivot point location involves checking three different pairings. Figure 5 shows an example of the pairings to be checked where $m = 5$ and pivot point is found to be 12. Deciding if the pivot point location or the "$m+1$" location should be used for XORing in the current stage is based upon the average d value among the first m values (hash values) thus resulted. In each intermediate step, let μ_{pre} denote this average d value from the previous step. The corresponding resulted average d value produced by XORing the pivot point location pairing and the "$m+1$" location pairing are denoted as, respectively, μ_{pivot} and μ_{m+1}. If $\mu_{pivot} > \mu_{pre}$ and $\mu_{m+1} > \mu_{pre}$, no XORing will occur and the algorithm ends. Otherwise, the smaller one between the μ_{pivot} and μ_{m+1} is chosen. For the example in Figure 5, the pivot point location produces reductions of 48% and 51% and the use the latter one

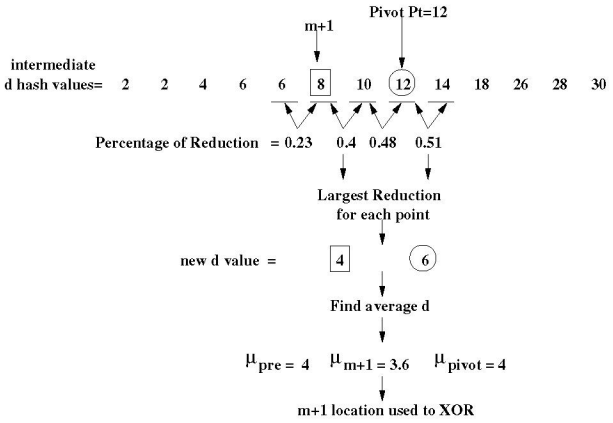

Fig. 5. Ad Hoc Pivot Hashing Checking Pairings from "Pivot Point" Location and from the "$m+1$" Location

leads to a new average d value of 4, while the "$m+1$" location produces reductions of 23% and 40% and the use of latter one leads to a new average of 3.6. Therefore, the XOR pairing from the "$m+1$" location is selected. The complete ad hoc pivot hashing algorithm is given in Figure 6. As aforementioned, the algorithm continues to search paring of bits to XOR until only m bits remain in the intermediate hash value array or until no pairing can be found for productive XORing. In the latter case, the first m bits (and thus with the smallest d values) and their associated XORing group patterns are used as the final hash function.

III. Simulation

Our simulation is devoted to determining the performance of the proposed ad hoc pivot hashing on practical performance indicators. This paper looks at two important performance indicators: maximal search length (MSL) and average search length (ASL). Figure 7 shows a graphical example of the performance indicators. The indicator MSL denotes the maximum number of hash collisions which in turn indicates the maximal number of search steps required to search through the collision. ASL reflects the average number of lookups needed to find an item in the database. A good hashing algorithm will be able to minimize both performance indicators.

The simulation also compares the performance of the proposed ad hoc pivot hashing with other well-known hashing algorithms. Three hashing algorithms are used for performance comparison. The first is the standard group XORing which take a database of n-bit address and group-XORs every n/m bits together to create an m-bit hash value. The second one is an advanced bit extraction algorithm based on the proposed preprocessed database. In this bit extraction algorithm, the database is sorted with the d value, and then the smallest d values are extracted to be used as the m-bit hash value. The third hashing algorithm is another ad hoc hashing algorithm discussed in , referred to as the "ad hoc algorithm", which uses a hybrid of bit extraction and group XORing.

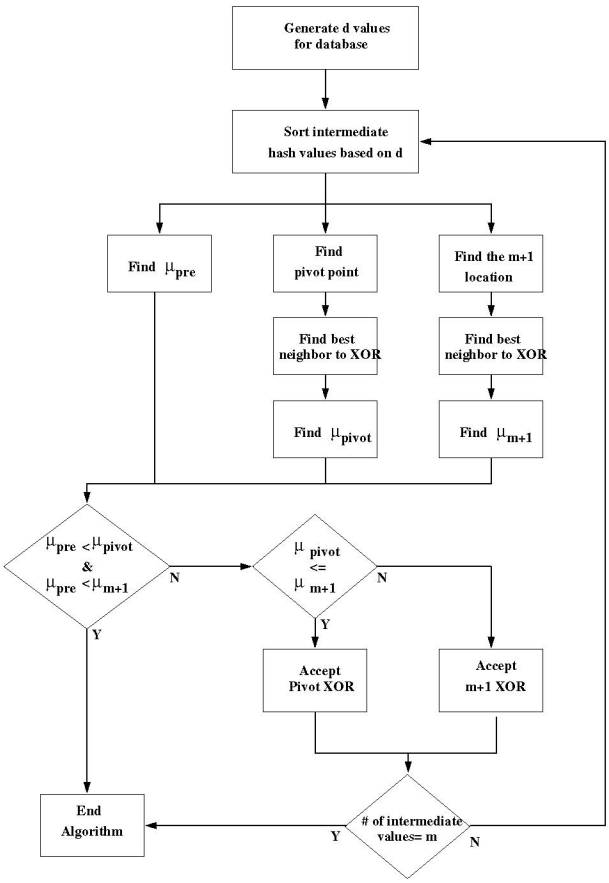

Fig. 6. Ad Hoc Pivot Hashing Algorithm

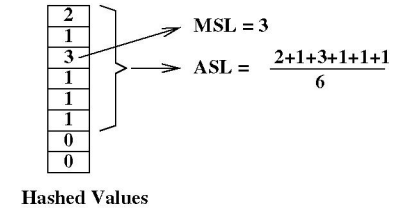

Fig. 7. Hashing Performance Indicators

A. Databases with Real IP Addresses

Simulation is first performed on a collection of real IP addresses gathered from a network trace from a local network router. For the simulation 2^m IP addresses were randomly taken from the trace and then used as a database to perform the hashing. Figures 8 and 9 show the MSL and ASL results for all four different hashing algorithms. The two ad hoc algorithms show much better performance than the two regular hashing algorithms. Figures 10 and 11 show the MSL performance gain for the two ad hoc hashing algorithms compared to group XORing and sorted bit extraction respectively. While the two ad hoc algorithms produce very similar performance in this comparison, the proposed ad hoc pivot algorithm requires much less computation time than the previously proposed ad hoc technique in [4].

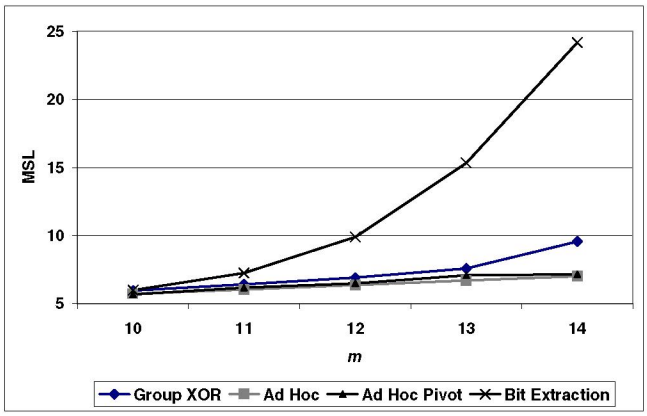

Fig. 8. Performance Comparison with MSL on Real IP Addresses

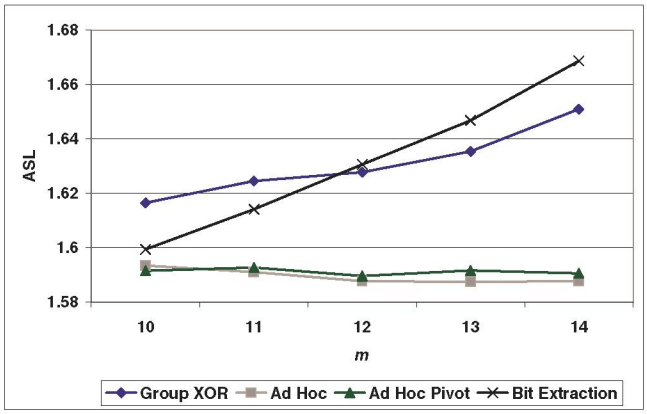

Fig. 9. Performance Comparison with ASL on Real IP Addresses

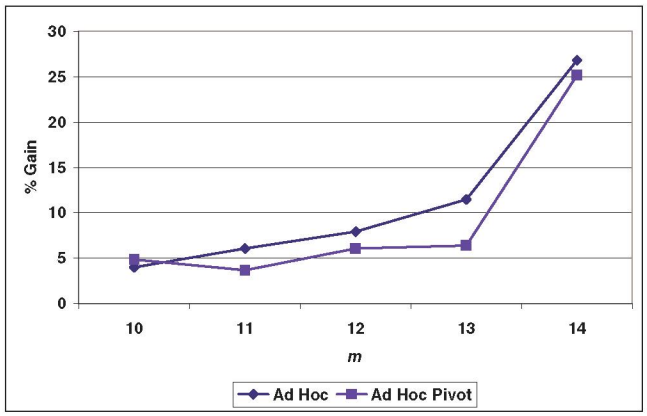

Fig. 10. Performance Gain over the Standard Group XOR on Real IP Addresses

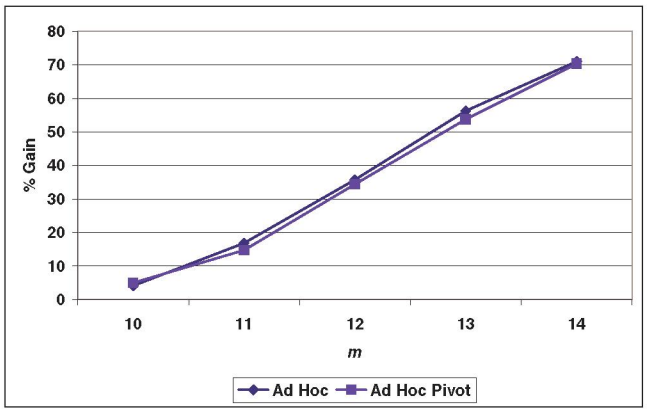

Fig. 11. Performance Gain over the Bit Extraction on Real IP Addresses

B. Randomly Generated Databases

An artificially generated database is created to provide a non-uniform while constantly changing database for simulation. For each simulation run, a database of 2^m 32-bit entries are generated such that the d value for each bit position is uniformly distributed. That is, a random d value is generated for each V_i and then each bit in the vector is randomly chosen. Figures 12 and 13 show the MSL and ASL results for all four different hashing algorithms. Performance gain in MSL from the two ad hoc techniques compared to group XORing and sorted bit extraction is given in Figures 14 and 15 respectively. In this randomly generated database the advantage in performance in ad hoc pivot hashing is much more significant. The previously proposed ad hoc technique loses its advantage when m becomes large, while the newly proposed ad hoc pivot technique consistently outperforms the two regular algorithms by a very sizable margin. A gain of up to $50 - 60\%$ is observed when $m = 14$.

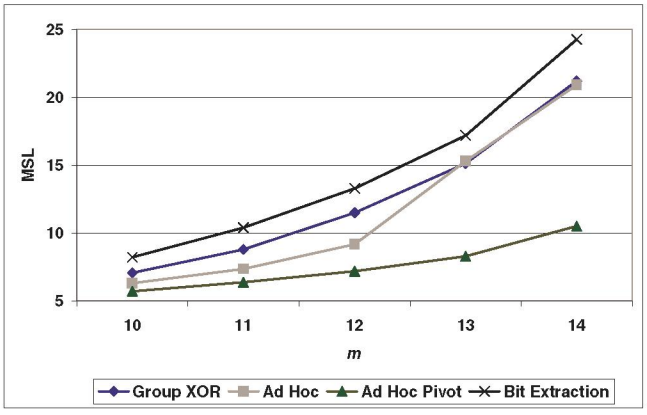

Fig. 12. Performance Comparison with MSL on Randomly Generated Databases

IV. Conclusion

The distribution of data in a wide range of applications requires new research in hashing. Many applications such as IP address lookup, intrusion detection systems, general

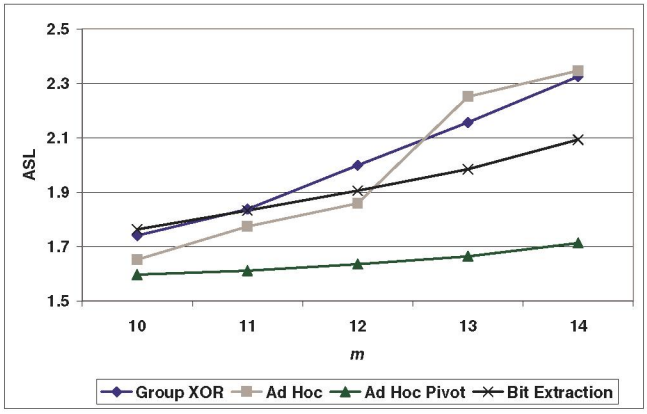

Fig. 13. Performance Comparison with ASL on Randomly Generated Databases

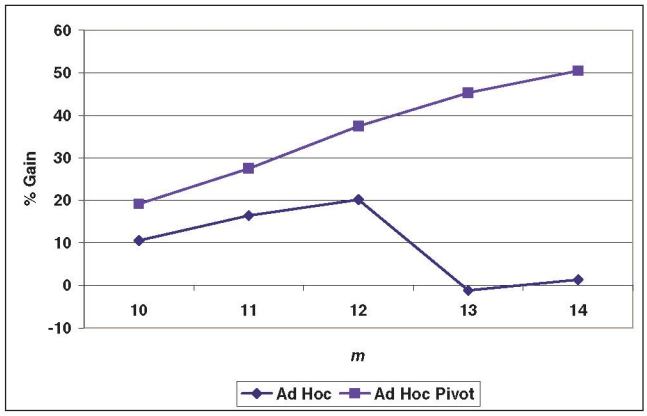

Fig. 14. Performance Gain over the Standard Group XOR on Randomly Generated Databases

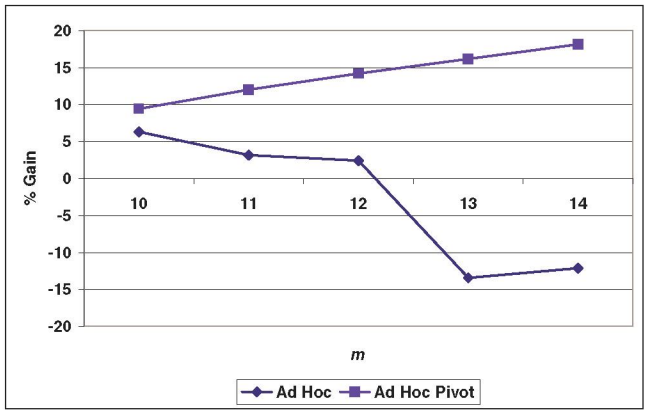

Fig. 15. Performance Gain over the Bit Extraction on Randomly Generated Databases

database query and string matching can benefit from hashing algorithms designed for an arbitrary distributed database. The proposed methodology of ad hoc pivot hashing demonstrates that improvement in overall performance can be achieved by carefully adapting to the distribution of the application. The ad hoc pivot hashing delivers several critical insights into new areas of hashing research. A potential expansion to hashing includes further exploring the database by investigating correlation among bit vectors for even better decision on how and which bits to be XORed. Other extensions the authors have undertaken include finding a non-exclusive XORing hashing in which bits are reused to further improve the search performance.

REFERENCES

[1] A. Broder and M. Mitzenmacher, "Using Multiple Hash Functions to Improve IP Lookups", *IEEE INFOCOM*, 2001.

[2] Sang-Hun Chung, J. Sungkee, Hyunsoo Yoon, Jung-Wan Cho, "A Fast and Updatable IP Address Lookup Scheme", *International Conference on Computer Networks and Mobile Computing*, 2001.

[3] Raj Jain, "A Comparison of Hashing Schemes for Address Lookup in Computer Networks," *IEEE Transactions on Communications,*, Vol. 40, No. 10, Oct 1992.

[4] Christopher Martinez and Wei-Ming Lin, "An Ad Hoc Adaptive Hashing Technique for Non-Uniformly Distributed IP Address Lookup in Computer Networks," *The 3rd International Conference on Cybernetics and Information Technologies, Systems and Applications,* July 2006, Orlando, FL.

[5] Andreas Moestedt, Peter Sjodin, "IP Address Lookup in Hardware for High-speed Routing", *Proc. IEEE Hot Interconnects 6 symposium*, Stanford, California, pp.31-39, August 1998.

[6] Xiaojun Nie, David J. Wilson, Jerome Cornet, Gerard Damm, Yiqiang Zhoa, "P Address Lookup Using A dynamic Hash Functio", *IEEE Electrical and Computer Engineering, Canadian Conference*, Page(s) 1646 - 1651, May 1-4, 2005.

[7] Stefan Nilsson and Gunnar Karlsson, "IP Address Lookup Using LC-Tries", *IEEE Journal on Selected Areas in Communications*, pp. 1083-1092, June 1999.

[8] D. Pao, C. Liu, L. Yeung and K.S. Chan, "Efficient Hardware Architecture for Fast IP Address Lookup", *IEEE INFOCOM*, 2002.

[9] M.A. Ruiz-Sanchez, E.W. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms", *IEEE Network*, Vol.15, pp.8-23, Mar./Apr.2001.

[10] V. Srinivasan and G. Varghese, "Faster Ip Lookups Using Controlled Prefix Expansion", *Proceedings of SIGMETRICS 98*, pp. 1-10, Madison, 1998.

[11] M. Waldvogel, G. Varghese, J. Turner and B. Plattner, "Scalable High Speed Ip Routing Lookups", in *Proc. ACM SIGCOMM'97*, pp. 25-35, Sept. 1999.

[12] P.A. Yilmaz, A. Belenkiy, N. Uzun, N. Gogate and M. Toy, "A Trie-based Algorithm for IP Lookup Problem", *Global Telecommunications Conference (GLOBECOM)* 2000.

[13] Daxiao Yu, Brandon Smith, and Belle Wei, "Forwarding Engine for Fast Routing Lookups and Updates," *Global Telecommunications Conference,*, Globecom '99, p.1556-p.1564.