

# Range-Based Bitmap Indexing for High Cardinality Attributes with Skew\*

Kun-Lung Wu and Philip S. Yu  
IBM T. J. Watson Research Center  
Yorktown Heights, NY 10598

## Abstract

*Bitmap indexing, though effective for low cardinality attributes, can be rather costly in storage overhead for high cardinality attributes. Range-based bitmap (RBM) indexing can be used to reduce this storage overhead. The attribute values are partitioned into ranges and a bitmap vector is used to represent a range. With RBM, however, the number of records assigned to different ranges can be highly uneven, resulting in non-uniform search times for different queries. We present and evaluate a dynamic bucket expansion and contraction (DBEC) approach to simultaneously constructing range-based bitmap indexes for multiple high-cardinality attributes. Simulations are conducted to evaluate this DBEC approach. Both synthetic and real data are used in the simulations. The results show that (1) with highly skewed data, DBEC performs quite well compared with a simple approach and (2) DBEC compares favorably with the optimal approach.*

## 1 Introduction

Multidimensional data analysis has become increasingly important for decision support systems as more and more complex queries are posted against ever larger amount of data in a *data/information warehouse* [1, 2, 3]. To efficiently handle multidimensional data analysis, an effective multidimensional index structure is required. Among the many existing multidimensional indexes, such as k-d trees [4], k-d-B trees [5], hB-trees [6, 7], R-trees [8, 9], and grid files [10, 11], bitmap indexing [12, 2, 13] is potentially the easiest to maintain.

To build a bitmap index for an attribute of a table of  $N$  records, a separate bitmap vector of length  $N$  bits is used to represent each distinct attribute value. Each bit in a bitmap vector represents one record, indi-

ating whether or not the record has the matching attribute value represented by the bitmap vector. With bitmap indexes constructed for multiple attributes, it is rather easy to identify the records that satisfy a multi-attribute predicate. All it needs is some bitwise logical AND/OR operations using the corresponding bitmap vectors.

Multidimensional bitmap indexing can be very effective if the indexed attributes are of low cardinality. However, for high cardinality attributes, the storage requirement can be prohibitively large.

One approach to reducing the storage overhead due to high cardinality attributes is to partition the attribute values into a smaller number of ranges<sup>1</sup>. A bitmap vector is then used to represent a range, instead of a distinct value. This way, the number of bitmap vectors for any attribute can be properly controlled.

However, range-based bitmap indexing can result in very uneven bucket sizes and cause access times for different queries to vary substantially. This problem can be aggravated if the attribute values are highly skewed.

In this paper, we present a dynamic bucket expansion and contraction (DBEC) approach to constructing buckets for range-based bitmap indexing for high cardinality attributes with skew. It consists of two phases. In the first phase, data records are sequentially scanned and multiple equally-spaced buckets are used to count the number of records falling into each bucket. This can be done simultaneously for multiple high cardinality attributes<sup>2</sup>. Whenever a bucket has accumulated more than a certain number of records, it is dynamically expanded into more smaller-range buckets. Upon completion of counting, multiple small-range buckets are then contracted into the final buckets such that each

---

<sup>1</sup>Another approach is to use an encoded bitmap [14], where attribute values are encoded and a mapping table is used for decoding.

<sup>2</sup>For simplicity, in this paper we focus on a single attribute even though the same algorithm can be simultaneously applied to multiple attributes.

---

\*This work was supported in part by NASA/CAN grant no. NCC5-101.

bucket contains about the same number of records. We study various heuristics to combine multiple adjacent smaller-range buckets into the final larger-range buckets. In the second phase, data are then sequentially scanned again to actually build the bitmap indexes, with a bitmap vector representing the range of each final bucket.

Simulations are conducted to evaluate the DBEC approach and compare it with a naive partition, a simple expansion and the optimal partition approaches. The results show that (1) the naive partition approach performs poorly; (2) with high skew, DBEC substantially outperforms the simple expansion approach; and (3) using a good heuristic for bucket contraction, DBEC compares favorably with the optimal approach where pre-sorting is needed.

Note that the bucket expansion and contraction discussed here is, to some extent, similar to the bucket partitioning in the *hash-partitioned join* algorithms, such as the GRACE hash-join and Hybrid hash-join algorithms [15, 16]. Both are to partition a relation into multiple buckets based on the values of a particular attribute. However, their constraints and objectives are different. In the hash-partitioned join algorithms, the goal is to hash a relation into disjoint buckets such that each bucket can be fit into main memory to build a hash table for join operations with another relation. The range of a bucket is not important, and thus a tuple can be arbitrarily hashed into any bucket. Bucket tuning can therefore be used to combine any number of small buckets into a larger one so long as the resultant bucket can be fit into the main memory [16]. On the other hand, in the bucket expansion and contraction for bitmap indexing discussed in this paper, the range of a bucket is an important piece of information. Tuples cannot be hashed into any bucket. In order to support range queries, only contiguously ranged small buckets can be combined in bucket contraction. Besides, the bucket size can be flexible.

The rest of the paper is organized as follows. Section 2 describes bucket construction and contraction. Section 3 presents the simulation model and workloads. Section 4 provides the simulation results.

## 2 Bucket construction

Assuming that the minimum ( $V_{min}$ ) and maximum values ( $V_{max}$ ) of an attribute are given, the simplest approach to constructing  $B$  buckets for an attribute is to partition the entire attribute range into  $B$  equal-distance ranges, each with the same width of  $(V_{max} - V_{min})/B$ . This approach is called the **naive partition**. It assumes that the distribution of the attribute values is uniform between  $V_{min}$  and  $V_{max}$ . However,

many real-life data are not uniform. They tend to be skewed to certain ranges. The naive partition approach is therefore likely to construct highly uneven buckets.

### 2.1 Bucket expansion

We propose here a bucket expansion and contraction approach to constructing buckets. It does not require a pre-sorting of the data, but it consists of two phases of sequential I/O scans. In the first phase, the data are scanned into  $B_i$  small-range buckets, where  $B_i > B$ , and  $B$  is the final number of buckets. The initial  $B_i$  buckets can be constructed such that the entire attribute range is partitioned equally into  $B_i$  intervals. These small-range buckets are then contracted into the final  $B$  larger-range buckets. In the second phase, bitmap indexes are constructed for the ranges represented by the final buckets.

There are two different expansion alternatives. One is called **simple expansion** and the other **dynamic expansion**. The simple expansion uses  $B_i$  as the initial buckets and simply collects the data counts in each bucket during the first phase. Namely, the expansion is done initially and uniformly.

On the other hand, the dynamic expansion scheme dynamically expands a growing bucket into more smaller buckets. The criterion for expansion is when a bucket count is greater than a threshold,  $T$ . The checking of threshold can be periodical or whenever a bucket count is updated. Since some of the buckets may be newly expanded, the threshold for triggering a further expansion of a small-range bucket should take into consideration the age of the bucket since its creation. Moreover, the original record counts of all the expanded bucket must be re-distributed into the smaller-range buckets proportional to the respective counts of the smaller-range buckets.

For dynamic bucket expansion, there might be some cases where the same attribute values appear repeatedly for many times. Namely, there are some isolated skew values. In this case, dynamic bucket expansion may result in some buckets with exceptionally large occurrence counts and many other empty buckets. Too many empty buckets are not desirable since they take memory space. To prevent excessive empty buckets from occurring, we can identify these isolated skew values in the process of bucket expansion <sup>3</sup>.

### 2.2 Bucket contraction

After bucket expansion, there are more small-range buckets than the required number of buckets,  $B$ . A

<sup>3</sup>For details on methods to deal with isolated skew values and their performance impacts, readers are referred to [17].

combination of multiple buckets with smaller counts into a bigger one is needed so that the total number of buckets is  $B$  and the resulting bucket sizes are about equal. If attribute values do not have range semantics, e.g., no range queries are allowed against the attribute, then non-adjacent smaller size buckets can be combined. In this case, bucket contraction is relatively easy. We can first sort the small-range buckets based on the bucket count in a non-increasing order. We can then pack the largest of the remaining small-range buckets to one of the large-range  $B$  buckets with the smallest count. This heuristic is referred to in the literature as LPT (longest processing time first) due to [18]. The heuristic optimization algorithm solves the so called minimum makespan or multiprocessor scheduling problem.

Since most attributes of continuous data types do have range semantics and typically allow range queries, we will focus on the case that only adjacent buckets can be combined. In this case, it is more difficult for bucket contraction. The problem is to combine them so that the standard deviation of the sizes of the final  $B$  buckets is minimal. We evaluate 3 different heuristics to bucket contraction in our experiments. For simple exposition, let us call the buckets after the expansion process *small-range buckets* and the final  $B$  buckets *large-range buckets*. Also, assuming that the data structure representing a bucket maintains the ranges of the bucket and its record count, and small-range buckets are represented by  $b[i]$ ,  $i = 0, \dots, E-1$ , while large-range buckets are represented by  $B[j]$ ,  $j = 0, \dots, B-1$ . Here,  $E$  is the total number of small-range buckets after the expansion phase. If the total number of records to be indexed is  $N$ , then the ideal large-range bucket size should be  $N/B$ .

**Iterative one-bar:** The first heuristic uses a bar to guide the decision of whether or not to include the next adjacent small-range bucket into the current large-range bucket. If the total record count of the resulting large-range bucket is less than the bar, then include the next small-range bucket to the current large-range bucket; otherwise, include it to a new large-range bucket. The bar is first set to be  $N/B$ , the ideal mean for a large-range bucket. Then it is raised upward by an increment of 1 in each iteration. In each iteration, the standard deviation of the  $B$  large-range bucket sizes is computed. The iteration stops when the standard deviation stops decreasing.

The iterative one-bar heuristic may cause the first few large-range buckets to be packed with too many small-range buckets such that the last few large-range buckets end up with too few small-range buckets, especially as the bar is raised. Since our objective is to have approximately equal sized large-range buckets, a

second bar probably should be used to limit the first few large-range buckets to be around  $N/B$  while the second bar is raised.

**Iterative two-bar:** The second heuristic uses two bars. The first bar is initially set at  $N/B - \delta$ , while the second bar is initially set at  $N/B$  and raised by 1 at each iteration. Now the next small-range bucket can only be included in the current large-range bucket, if (a) before the inclusion, the large-range bucket count is under the first bar and (b) after the inclusion, the resultant large-range bucket count is also under the second bar. For each first bar, the iteration on the second bar stops when the standard deviation of the  $B$  bucket sizes stops decreasing. Thus, for each first bar, we obtain a set of bucket partition. The process repeats until the first bar value changes from  $N/B - \delta$  to  $N/B + \delta$ . In our experiments,  $\delta = 20\%$  of  $N/B$ , and it is incremented each time by 1. After the final value of first bar is tested, we choose the set of bucket partition with the minimal standard deviation.

**One-pass zigzag:** The third heuristics uses only one pass. The next small-range bucket is included in the current large-range bucket if (i) the resulting large-range bucket count is smaller than  $N/B$  or (ii) the *excess* of the resulting large-range bucket count over  $N/B$  is smaller than the *deficit* of the current large-range bucket count from  $N/B$  if the next small-range bucket is not included. Namely,  $(B[j].count + b[i].count - N/B) < (N/B - B[j].count)$ , where  $B[j]$  is the current large-range bucket and  $b[i]$  is the next small-range bucket. In fact, this heuristic implicitly uses a floating upper bar and lower bar to force the large-range bucket count to zigzag around the ideal  $N/B$  records.

### 3 Simulation model

We implemented the naive partition, the simple bucket expansion and contraction, the dynamic bucket expansion and contraction approaches. All three heuristics for bucket contraction were implemented for both simple expansion and dynamic expansion approaches. We conducted 100 experiments and plotted the standard deviations of the sizes of the final  $B$  buckets from each experiment. Each experiment generated 10,000 data points ranging from 100 to 9,900. These 10,000 points were to be partitioned into 20 buckets.

For the low skew case, the histogram of the 10,000 data points is relatively smooth. For the high skew cases, there are five spikes that really contain a lot of data points. For the case of high skew with isolated skew values, same values appeared many times in those spikes. On the other hand, for the case of high skew without isolated skew values, data points representing

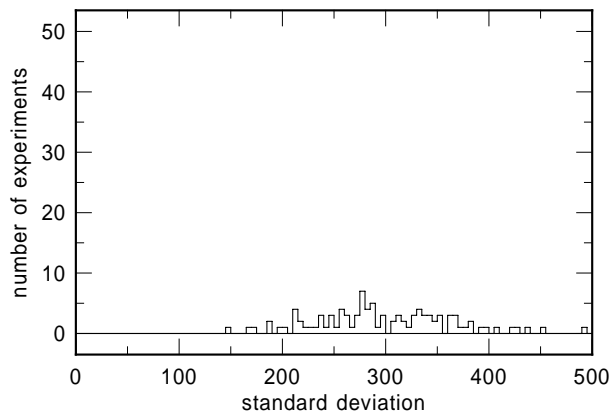
the spikes follow a normal distribution with standard deviation  $\sigma = 10$ . Details on the pseudo code for generating these 10,000 data points and the histograms representing individual cases were provided in [17].

## 4 Simulation results

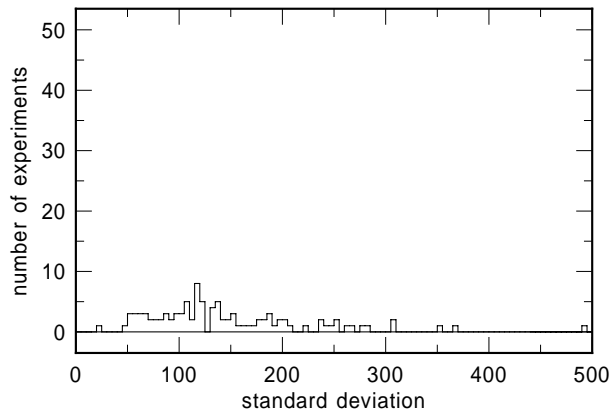
First, we examine the impact of skew on the effectiveness of the expansion schemes assuming that there are no isolated skew values. We plotted the histogram of the standard deviations of the resultant 20 bucket sizes from each experiment. There were 100 experiments in each histogram, each experiment partitioned 10,000 data points into 20 buckets. Therefore, the optimal large-range bucket size was 500 data points. For both expansion schemes the one-pass zigzag heuristic was used for bucket contraction (the impact of the three heuristics will be discussed later on). The initial number of buckets for both simple expansion and dynamic expansion  $B_i$  was set to be  $10B$ , namely 200. Note that for the dynamic expansion scheme, bucket expansions were triggered when bucket counts exceeded a threshold  $T$ . For all the experiments conducted in this paper, we set  $T = 0.2 \times N/B$ . Namely, whenever a bucket grew over 20% of the ideal mean  $N/B$ , the bucket was ready to be expanded into smaller-range buckets. But  $T$  was adjusted according to the age of the bucket since its creation (e.g, if a new bucket was created after 10% of the data was scanned, the threshold for this bucket at the time when 20% of the data was scanned was  $10\% \times 0.2 \times N/B$ ). Bucket sizes were checked to see if an expansion was needed after every 10% of the total data was scanned. In our experiments, every time a bucket was expanded it was expanded into  $s$  smaller buckets. We set  $s$  to be 10 for all the experiments in this paper.

In the low skew case (the results are not shown due to space limitation), the naive partition scheme resulted in rather uneven bucket sizes while the simple expansion scheme performed comparably with the dynamic expansion scheme. However, in the case of high skew without isolated skew values, the simple expansion scheme can no longer compete with the dynamic expansion scheme. This is illustrated very sharply in Fig. 1. In this figure, the histogram of the standard deviations for the simple expansion scheme is spread between 50 and 300, while that for the dynamic expansion scheme is still concentrated at around 20. In this high skew case, the naive partition approach is clearly not usable as it simply cannot partition the buckets evenly.

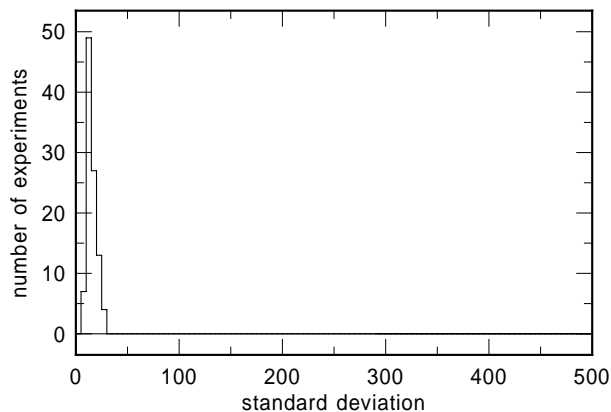
For comparing contraction heuristics, we used the dynamic expansion approach and tested for the high skew without isolated skew values. Fig. 2 shows the



(naive partition, high skew w/o isolated skew values)



(simple expansion, high skew w/o isolated skew values)



(dynamic expansion, high skew w/o isolated skew values)

Figure 1: Comparisons of bucket construction approaches for the case of high skew without isolated skew values.

histograms of standard deviations of the bucket sizes for the three heuristics. The iterative one-bar heuristic performs the worst while the one-pass zigzag the best. Note that for the best scheme, the standard deviations of bucket sizes are mostly less than 20. Since the ideal bucket size is 500, such low standard deviations show that the dynamic bucket expansion and contraction approach compare favorably with the optimal partition approach. Moreover, the dynamic expansion approach does not require costly pre-sorting while the optimal partition approach does.

In addition to synthetic workloads, we also evaluated the different schemes using real stock trading data. The trading data contain some 296 selected stocks traded on the 3 major US stock exchanges on a particular date. We used the trading volumes (highly skewed) and closing prices (less skewed) as input and partitioned them into 5 buckets. The results of the actual bucket sizes are shown in Fig. 3. For the optimal approach, we sorted the data and then did the partition. Clearly indicated in Fig. 3, the naive partition scheme is not effective for both trading volumes and closing prices. The simple partition approach performs acceptably for closing prices but not for trading volumes. The dynamic expansion approach performs a comparable performance with the optimal partition approach.

## 5 Summary

In this paper, we presented a dynamic bucket expansion and contraction approach to constructing range-based multidimensional bitmap indexes for high cardinality attributes with skew. The data are first scanned into the buffer to construct the bucket ranges by counting the data points falling into each bucket. If a bucket grows beyond a threshold, it is expanded into more smaller-range buckets. After the scan, adjacent buckets are combined into the final required number of buckets with approximately balanced count. Bitmap vectors are then built for the contiguous ranges represented by the final buckets. The dynamic expansion scheme was compared with a naive partition, a simple expansion and the optimal partition approach. The results showed that the naive partition approach performs poorly. The simple expansion scheme, though acceptable for low skewed data, cannot handle highly skewed data. However, the dynamic expansion approach performs comparably with the optimal partition approach, especially for highly skewed data.

## References

- [1] Y. Zhuge *et al.*, “View maintenance in a warehousing environment,” in *Proc. of ACM SIGMOD Int.*

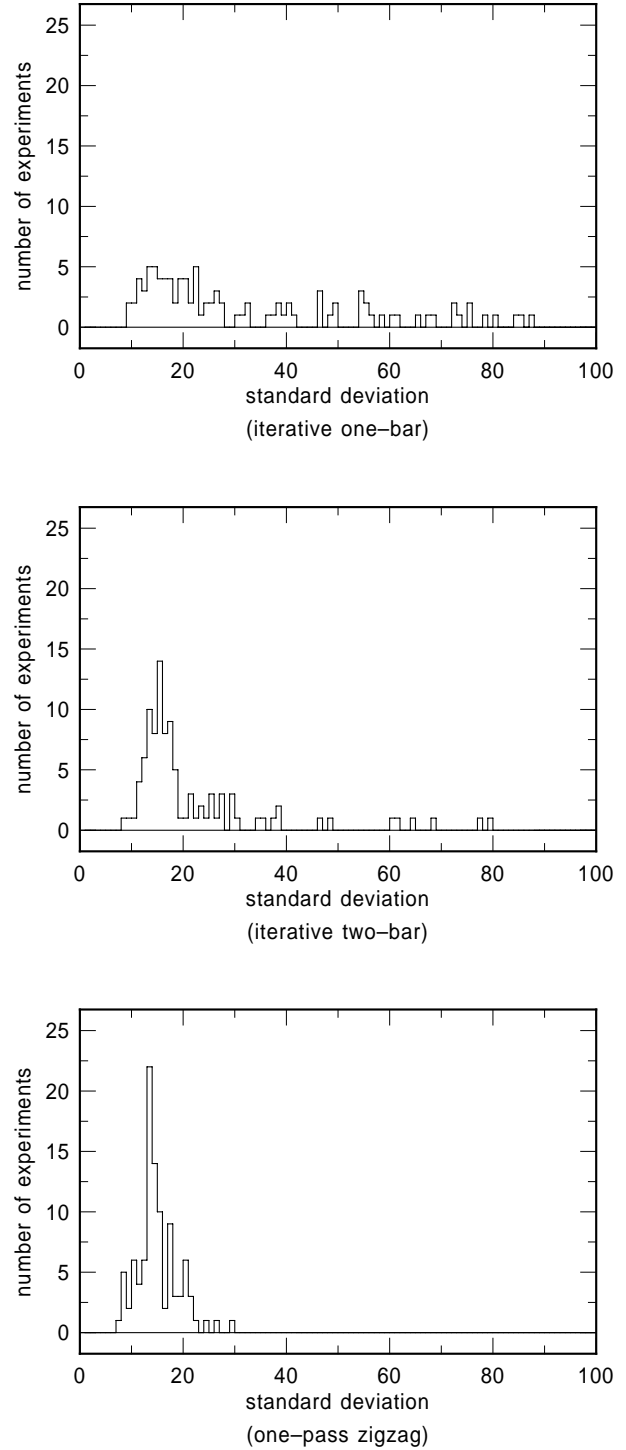


Figure 2: Comparisons of three heuristics for bucket contraction for the case of high skew without isolated skew values.

- Conf. on Management of Data*, pp. 316–327, 1995.
- [2] C. D. French, ““One size fits all” database architectures do not work for DSS,” in *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pp. 449–450, 1995.
- [3] G. Hallmark, “The oracle warehouse,” in *Proc. of Very Large Data Bases*, pp. 707–709, 1995.
- [4] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, Sept. 1975.
- [5] J. Robinson, “The K-D-B-tree: A search structure for large multidimensional dynamic indexes,” in *Proc. of 1981 ACM SIGMOD*, pp. 10–18, 1981.
- [6] D. B. Lomet and B. Salzberg, “The hB-tree: A multiattribute indexing method with good guaranteed performance,” *ACM Trans. on Database Systems*, vol. 15, no. 4, pp. 625–658, 1990.
- [7] G. Evangelidis, D. Lomet, and B. Salzberg, “The hB<sup>II</sup>-tree: A modified hB-tree supporting concurrency, recovery and node consolidation,” in *Proc. of Very Large Data Bases*, pp. 551–561, 1995.
- [8] A. Guttman, “R-trees: A dynamic index structure for spatial searching,” in *Proc. of 1984 ACM SIGMOD*, pp. 47–57, 1984.
- [9] T. Sellis, N. Roussopoulos, and C. Faloutsos, “The R<sup>+</sup>-tree: A dynamic index for multi-dimensional objects,” in *Proc. of 13th VLDB Conference*, pp. 507–518, 1987.
- [10] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, “The grid file: An adaptable, symmetric multikey file structure,” *ACM Trans. on Database Systems*, vol. 9, no. 1, pp. 38–71, 1984.
- [11] M. Freeston, “The BANG file: a new kind of grid file,” in *Proc. of 1987 ACM SIGMOD*, pp. 260–269, 1987.
- [12] D. E. Knuth, *The Art of Computer Programming, Vol 3: Sorting and Searching*. Addison-Wesley, 1973.
- [13] P. O’Neil and D. Quass, “Improved query performance with variant indexes,” in *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pp. 38–49, 1997.
- [14] M.-C. Wu and A. P. Buchmann, “Encoded bitmap indexing for data warehouses,” in *Proc. of Int. Conf. on Data Engineering*, pp. 220–230, 1998.
- [15] D. J. DeWitt and R. Gerber, “Multiprocessor hash-based join algorithms,” in *Proc. of Very Large Data Bases*, pp. 151–164, 1985.
- [16] M. Kitsuregawa, M. Nakayama, and M. Takagi, “The effect of bucket size tuning in the dynamic hybrid GRACE hash join method,” in *Proc. of Very Large Data Bases*, pp. 257–266, 1990.
- [17] K.-L. Wu and P. S. Yu, “Range-based bitmap indexing for high cardinality attributes with skew,” tech. rep., IBM Research Report, RC 20449, 1996.
- [18] R. Graham, “Bounds on multiprocessing timing anomalies,” *SIAM Journal of Computing*, vol. 17, pp. 416–429, 1969.

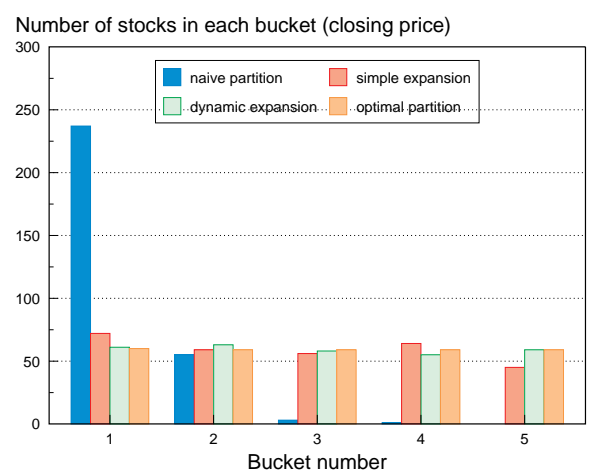
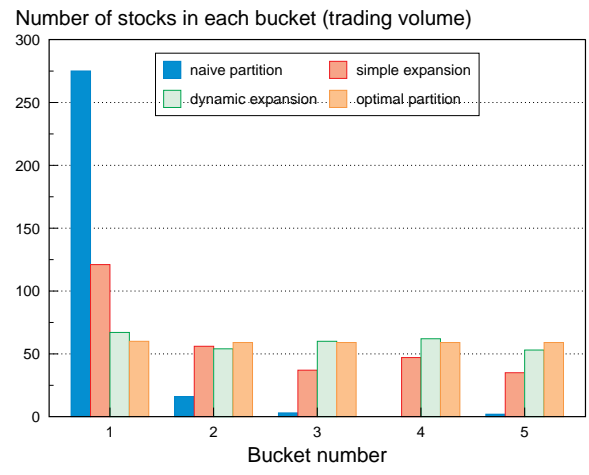


Figure 3: Actual bucket sizes for different bucket construction schemes.