Examine Dynamic Data Structures and Complex Algorithms

Yiqiao Yin
Ph.D. Student

Abstract

This report examines the famous file sharing algorithms used in distribution systems. We investigate the architecture of these algorithms and compares how they are deployed. In addition, this report also exposes some of the potential issues with the algorithms. Examples and applications are presented to illustrate the survey of the algorithms in this report.

1 Introduction

The search on similarity of two documents is a well established and import subject in computer science (Wang et al., 2014; Enbody and Du, 1988; Chi and Zhu, 2017; Gui et al., 2017). Amongst many searching algorithms, hashing becomes famous and rise to popularity because of the level of simplicity at deployment and the low cost in regards of time and space analysis. Throughout the years, hashing techniques have been evolved overtime and many different upgrades have been developed (Chi and Zhu, 2017). The hashing techniques are desired to retrieve information from large volume of documents and texts for post process analysis and it is widely used in day-to-day work in computer science. When deep learning was developed in the late 1980s, hashing algorithms have also been used to encourage promising work in machine learning since its design provide crucial guideline for scholars to understand the how searching works from programming perspective. In large-scale data analysis and today's BIGDATA concept, hashing provides the most fundamental understanding of information representation and mapping system between data.

Clustering technique is another important field of practice in computer science (Saxena et al., 2017; Rai and Singh, 2010). In this report, different clustering techniques are discussed and a variety of grouping techniques are covered. There is a clear motivation in today's computer science field work to understand and also develop the advancement of different clustering for a variety of different purposes in mathematics, engineering, computer science, statistics, and so on. Taking a hospital scenario as an example. Patients with different diseases are, evidently, treated differently. This is because different diseases have different symptoms and hence requires different medicine. A good clustering technique can help us navigate the unsupervised world with orders instead of chaos.

2 Description of Different Architectures in Hashing

This section describes the different architectures of hashing techniques.

The first fundamental technique to introduce is the hashing function. It is a mathematical function that maps information and text to a list of dictionary of keys. To express the idea better, we will address the input of the hash functions as keys and the output of the hash functions as hash values. In other words, an abstract formula can be expressed using a hash function $h(\cdot)$

$$h : \text{keys} \to \text{hash values} \tag{1}$$

An important property for this setup is that it utilizes statistical concept of how functions interact with each other. In addition, the key functionality of a hash function is the capability of searching for keys and values even when the information is scrambled because the hash function ensures that the resulting values from the function remains uniformly distributed when generating the output. This is to avoid collision. A collision in a hash function refers to a scenario where two different keys are mapped to one single hash value. Hence, the important properties of a good hash function can be summarized below

1. the hash function is efficient and easy to compute;
2. the hash function is capable of minimizing duplicated scenarios.

2.1 Properties of Hashing

From earlier section, it is discussed that a good hashing function has expected inputs to be spread over the output range as evenly as possible. This is the uniformity property. Like the uniform distribution phenomenon in statistics, every hash value ideally has the same probability. A uniform probability distribution states that a random variable x can take values from a range $[a, b]$ while each value has the same probability. In other words, we can state this formally in the following. Assume that there is support of a random variable x to be $\mathbb{R}_x = [a, b]$. The probability density function of a uniform random variable takes the following form

$$f_X(x) = \frac{1}{b-a} \text{ if } x \in \mathbb{R}_x \text{ and } f_X(x) = 0 \text{ elsewhere} \tag{2}$$

This property can be tested and the test for uniformity is called Chi-square test (Rao, 1972; Inglot and Janic-Wróblewska, 2003). The goal is to produce a goodness-of-fit measure using Chi-square test. The key here is to measure whether the actual distribution is close to the expected distribution. The closer they are, the higher confidence computer scientists have to claim that the actual distribution is uniform. To conduct this test, the following formula is used

$$\frac{n/2m}{n+2m-1} \sum_{j=0}^{m-1} (b_j)(b_j+1)/2 \tag{3}$$

where n is the number of keys, m is the number of buckets, b_j is the number of items in bucket j. Commonly it is desirable to have the test value to be 0.95-1.00 so that it gives computer scientists confidence that the actual distribution is uniform (Castro et al., 2005).

2.2 Discussion of Architecture, Coding, and Implementation

This section provides some demonstration of the architecture of hashing. Since hashing function is a technique to solve a particular data structure question, there is no universal hashing algorithm to solve all problems. Hence, this subsection will use example-based approach to explain the architecture of the hashing algorithm, the coding style, and the implementation.

The first example is to search if one list of items is inside another list of items. It is one of the most common data structure problems that need to be solved. Suppose there are two lists and each of them has certain number of digits. Denote them list 1 and list 2. One simple goal is to check if one list has all of its items inside the other. This type of task is called "sub-list search" and it is a common task abstracted to solve different kinds of computer programming problems.

A sample syntax is presented below. The syntax creates a function called "is_this_subset" and the function takes two inputs. The two inputs can be two lists or two arrays. The algorithm presents a nested for loop or double for loop. The first for loop searches through each item in the second array while the nested loop or the inside loop searches for each item in the first array. Each of the inside loop checks if the second array is inside the first and breaks if the condition does trigger. Then the algorithm checks if the running index j is the same with the length of first array. The algorithm returns negative or “0” value if the condition triggers. In the end, if everything checks, the algorithm returns positive or “1”.

```
def is_this_subset(the_first_array, the_second_array):
    i = 0
    j = 0

    l1 = len(the_first_array)
    l2 = len(the_second_array)

    for i in range(l2):
        for j in range(l1):
            print(i, j)
            if(the_second_array[i] == the_first_array[j]):
                print(i, j, the_second_array[i], the_first_array[j
                    ↪ ])
                break

        # the above nested loop can break
        # when the condition triggers
        # else it runs the following
        # then returns 0 if j is the
        # same as l1
        if (j == l1):
            return 0

    # when the nested for loops
    # finishes above
    # we can wrap things up here
    return 1

the_first_array = [11, 4, 452, 6, 213, 3]
the_second_array = [11, 213, 4, 3]
```

```
print(is_this_subset(the_first_array, the_second_array))
```

3 Clustering Techniques

This section covers the fundamentals of clustering techniques. The basic purpose for clustering techniques is to divide data into different groups based on similarity. For each group, there are certain number of items from the original data and these items for a cluster. Inside of this cluster, the items are similar to each other more than they are to the rest of the items. There can be many building blocks inside of the clustering algorithm and many places require their own tuning parameters range. The key idea for most clustering techniques is to use a distance function to measure how far one data point is from the other. If the distance function produces a value that is high, then it implies that the two data points are far away. This distance value can be used as a comparison purposes amongst all data points. The most basic distance function is the Euclidean distance and it tkaes the following form

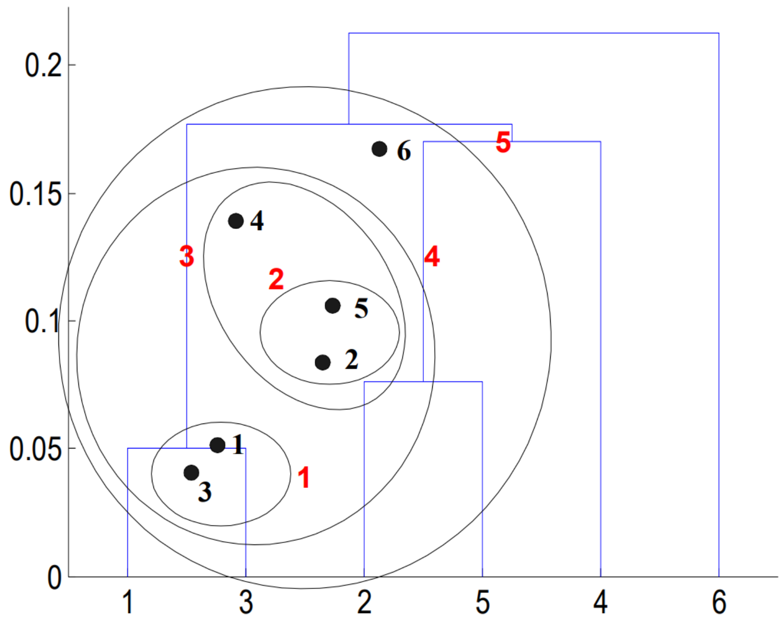
$$d(x, y) = \sqrt{\sum_i (x_i - y_i)^2} \tag{4}$$

where x and y are two attributes in the data and the running index i tracks each instance (or sample) in the data. There are a few more famous distance functions desired for most clustering techniques and we list them below

$$\begin{array}{ccccc} \text{Euclidean:} & d(x,y) & = & \sqrt{\sum_i (x_i - y_i)^2} \\ \text{Squared Euclidean:} & d(x,y) & = & \sum_i (x_i - y_i)^2 \\ \text{Manhattan:} & d(x,y) & = & \sum_i |x_i - y_i| \end{array} \tag{5}$$

There are many algorithms surveyed in clustering techniques (Rai and Singh, 2010). They come down to two original ideas. The first one is nearest neighbor method and the second one is the hierarchical method. Both methods can be demonstrated in the Figure 1. The nearest neighbor method uses distance functions to measure how far a data point is from the rest of the data points. The data points with closer distances are grouped together. The second method is hierarchical method. A famous hierarchical method is dendrogram. The algorithm investigates the pair-wise distances between data points and assign them to different branches accordingly. The branch accelerates in levels when the distances get pass certain thresholds where these thresholds are numerical tuning parameter.

Figure 1: **Clustering Techniques**. Famous clustering techniques can be seen in this diagram. By using two attributes, the data points can be grouped into different clusters using nearest neighbors or hierarchical methods. The graph is an adaption from Figure 4 of this research (Rai and Singh, 2010).



4 Deployment

This section briefly discusses the crucial steps required for deploying a hashing algorithm in practice. The stages and levels of deployment depends on the existing system and pipeline in place.

First, it is likely that the existing system in place consists of some of hashing techniques and the system works smoothly and efficiently to a certain level. It is up to the computer scientists to evaluate the system and the business requirement to decide whether the current system needs to be replaced or upgraded.

Second, the scenario can be that there is no existing system in place. In this case, there is more freedom of what hashing algorithm to be put in place. A new algorithm will be proposed and certain measurements need to be taken.

In regards to measurements, our previous work investigated the Big-O analysis on different algorithms. This can be taken as the major measurement to assess and evaluate the performance of different hashing algorithms. It can certainly be used to provide crucial guidance whether the current system needs to be replaced. It is recommended and certainly desired to investigate the time and space consumption using Big-O notation before any upgrade or novel proposal of new hashing algorithm.

5 Potential Issues

Potential issues can occur when the current or the proposed hashing algorithms do not satisfy the business moat. Every business has an economic moat and the range of such moat expands as the business scales up in size. Different consumer groups will arise and the business will have to adjust its operation activities accordingly. This is where data type and data structure changes.

Whenever there is a change on data type or structure, it is absolutely necessary to evaluate the time and space consumption of the current hashing algorithms. If the current benchmark does not meet the requirements anymore, it would have to be replaced or upgraded.

References

Castro, J. C. H., Sierra, J. M., Seznec, A., Izquierdo, A., and Ribagorda, A. (2005). The strict avalanche criterion randomness test. *Mathematics and Computers in Simulation*, 68(1):1–7.

Chi, L. and Zhu, X. (2017). Hashing techniques: A survey and taxonomy. *ACM Computing Surveys (CSUR)*, 50(1):1–36.

Enbody, R. J. and Du, H.-C. (1988). Dynamic hashing schemes. *ACM Computing Surveys (CSUR)*, 20(2):850–113.

Gui, J., Liu, T., Sun, Z., Tao, D., and Tan, T. (2017). Fast supervised discrete hashing. *IEEE transactions on pattern analysis and machine intelligence*, 40(2):490–496.

Inglot, T. and Janic-Wróblewska, A. (2003). Data driven chi-square test for uniformity with unequal cells. *Journal of Statistical Computation and Simulation*, 73(8):545–561.

Rai, P. and Singh, S. (2010). A survey of clustering techniques. *International Journal of Computer Applications*, 7(12):1–5.

Rao, J. (1972). Some variants of chi-square for testing uniformity on the circle. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 22(1):33–44.

Saxena, A., Prasad, M., Gupta, A., Bharill, N., Patel, O. P., Tiwari, A., Er, M. J., Ding, W., and Lin, C.-T. (2017). A review of clustering techniques and developments. *Neurocomputing*, 267:664–681.

Wang, Q., Si, L., Zhang, Z., and Zhang, N. (2014). Active hashing with joint data example and tag selection. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 405–414.