/xwiki/bin/view/Main/) / JioMarketPlace (/xwiki/bin/view/Reliance%20Retail%20POS%20Project%20/) / ABTesting (/xwiki/bin/view/Reliance%20Retail%20POS%20Project%20/ABTesting/) / Bucketing Algorithm (/xwiki/bin/view/Reliance%20Retail%20POS%20Project%20/ABTesting/Bucketing%20Algorithm/)

Bucketing Algorithm

Last modified by SonuPrajapati (/xwiki/bin/view/XWiki/SonuPrajapati) on 2023/07/14 13:51

A/B Testing - Deterministic Bucketing

In a simple given A/B test experiment, usually represents the existing control experience and B is the changed treatment variation of the experience. The changes could be relatively small or large. For example: a single word or phrase, a banner or button colour, an image displayed, sorting alphabetically vs most popular, or a design layout change.

To know more about AB Testing System Design (https://wiki.jio.com/xwiki/bin/view/Reliance%20Retail%20POS%20Project%20/ABTesting/System%20Design/)

Basic Functionality

- A/B & Multivariate testing
- Coin toss bucketing
- Total allocation of traffic
- Allocation for each variation (50/50, 90/10, 33,33,34)
- Audience segmentation
- Experiment impression tracking

Let's first discuss the below two scenarios to understand the problem statement.

Scenario 1:

Statement:- A new potential customer Pat enters. We have 2 variant (head and tail). We flip a coin which can randomly land, 50/50, heads / 0 or tails / 1.

```
variation = Math.floor(Math.random() * 2) // Could be 0 or 1

variation = 1 ( tail)
```

Problems:-

- If the customer received a different variant each time they returned, then we wouldn't know which variant was more effective.
- We need to keep track of this coin flip! Store it a cookie, local storage, database, or on a piece of paper for future use.
- We have to find some other solution which can provide a bucket of variant in constant time and without storing it into cookie/DB in sync to remove latency.

Scenario 2:

Statement:-

- There is common understanding, we have to divide users in range of buckets over a scale of 0-100% and in which bucket it falls permanent bucket assignment take place.
- What if we want to run an experiment with 4 variations or only for 10% of the customers.

In this scenario below one is an effective algorithm to distribute the traffic randomly.

```
experimentKey = "unique experiment identier"

userIdentifier = "unique user identifier"

bucketKey =  userIdentifier + "- " + experimentKey.  // combine unique experiment identifier so that bucket_key will be unique for user acc to experiment.

fnv-1a_32_hash = ƒ(e, t) // hash algorithm which is platform independent and provides more scattered hash value of range 32 bit.
```

Generate a unique value for the given visitor & experiment combination using a hashing algorithm. For this example, we'll use the language agnostic **fnv-1a_32_hash**.

Here we are using **bucketKey** as a seed value so we will get same hash value for same bucket key.

```
hashValue = fnv-1a_32_hash(bucketKey).     // hashValue = 2162646926
```

Now for getting the bucket percent range on scale (0-99.9)% range value from **hashValue.**

```
bucket_percent_range = (hashValue % 1000) / 10    // It will give percent value to one decimal
```

Normally, AB test experiments will run on 100% of the total audience population split up 25% Control & 25% for variants respectively, then anyone allocated within

- 0 - 24.9 will be **Control**
- 25.0 - 49.9 will be **Variant A.**
- 50.0 - 74.9 will be **Variant B.**
- 75.0 - 99.9 will be **Variant C.**

Now what if we have to target or segment the audience upto specific percent for the experiment. for e.g. 10% of total audience can be part of experiment.

- In such case we can use the above algorithm but and it will two Target Buckets.
  - (0-9.9) % of **Target_User**
  - (10.0-99.9)% of **Excluded_User**
- Those users who get **Target_User** will be eligible for bucketing according to the configuration
- Except **Target_User** all other users will be a part of **NON_EXPERIMENT**.

Pseudo Code

- **Fnv-1a 32 bit hash algorithm**

Below code snippet written in php but we can utilize the same for diff platforms as well.

```
private const FNV_32_INIT = 2166136261;
private const FNV_32_PRIME = 16777619;
private const PHP_INT_MAX = 2147483647;
private const PHP_INT_MIN = -2147483648;

/**
* Calculates the FNV-1a 32-bit hash value for the given key string.
*
* @param string $data The key string to be hashed.
* @return int The calculated hash value.
*/
function hash32(string $data): int
 {
     $hash = self::FNV_32_INIT;            // Initialize the hash value with a constant FNV_32_INIT
     $len = strlen($data);                 // Get the length of the input data

     for ($i = 0; $i < $len; $i++) {
         $hash ^= ord($data[$i]);          // XOR the hash value with the ASCII value of the current character
         $hash *= self::FNV_32_PRIME;      // Multiply the hash value by a constant FNV_32_PRIME
         $hash &= 0xFFFFFFFF;              // Apply a bitwise AND operation to ensure the hash value stays within 32 bits
      }

      // Check if the hash value exceeds the maximum value of a PHP integer
     // If so, adjust the hash value to be within the range of PHP_INT_MIN to PHP_INT_MAX
     if ($hash > self::PHP_INT_MAX) {
          $hash = (self::PHP_INT_MIN + ($hash - self::PHP_INT_MAX));
      }
   return $hash;
}
```

- **Bucketing**

```
variant_allocations = [
 {id: 'Control', percentage: 25},
 {id: 'Treatment A', percentage: 25},
 {id: 'Treatment B', percentage: 25},  {id: 'Treatment C', percentage: 25}
]

target_audience_percentage = 25;
experimentKey = "unique experiment identifier"
userIdentifier = "unique user identifier"
bucketKey =  userIdentifier + "- " + experimentKey.

function bucket_allocation() {
  assert variant_allocations.map(value -> value.percentage).sum() == 100; ///generate error for not having sum of percentage to 100;
   bucket_percent_range = (fnv-1a_32_hash(bucketKey) %1000)/10;
   if( bucket_percent_range < target_audience_percentage){
           start_range = 0.0;
            for (let allocation of variant_allocations) {
                 if (start_range <= bucket_percent_range && bucket_percent_range < (start_range+allocation.percentage)) {
                   return allocation.id;
                }
               start_range+=allocation.percentage;
           }
   }
   return "NON_EXPERIMENT";
}
```

Tags:

Created by SonuPrajapati (/xwiki/bin/view/XWiki/SonuPrajapati) on 2023/07/13 10:43

No comments for this page