Databases for Developers - Assignment 2

Anders Jacobsen, Dima Karaush

April 9, 2021

1 Task 1 - investigation

Produce a small writeup (around 300 words) answering the following questions.

1. What is point of NoSQL databases?
2. What is the CAP theorem?
3. What are ideal use cases of HBase?

1.1 What is the point of NoSQL databases

The point of NoSQL databases is to fill the holes where SQL databases does not. We handle data in many different ways, so it is just logical that we should have multiple tools for storing that data. Some pros and cons can be found below.

Pros

- Performance
- Scalability
- Flexibility
- Data Models

Cons

- Not mature
- Requires multiple databases
- Huge databases

But the above information depends heavily on witch NoSQL database you choose. There are many!

1.2 What is the CAP theorem

In theoretical computer science, the CAP theorem, also named Brewer's theorem after computer scientist Eric Brewer, states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees [2]:

- Consistency: Every read receives the most recent write or an error
- Availability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write
- Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

1.3 What are ideal use cases of HBase

HBase uses Google's BigTable paper as blueprint, HBase is build on Hadoop and the Hadoop Distributed File System (HDFS) and designed for scaling horizontally on clusters of commodity hardware. HBase makes strong consistency guarantees and features tables with rows and columns - with should make SQL fans feel right at home. Out-of-the-box support for versioning and compression [3].

2 Task 2 - Bloom filters

Bloom filters are used in hbase as an incredible optimization. Solve the following.

1. What is a bloom filter?
2. What is an advantage of bloom filters over hash tables?
3. What is a disadvantage of bloom filters?
4. Using your language of choice, implement a bloom filter with add and check functions. The backing bit-array can simply be a long (64 bit integer)
5. If you are to store one million ASCII strings with an average size of 10 characters in a hash set, what would be the approximate space consumption?
6. The following equation gives the required number of bits of space per inserted key, where E is the false positive rate. $b = 1.44log_2(1/\xi)$
7. How many bits per element are required for a 1% false positive rate?
8. How many bits per element are required for a 5% false positive rate?
9. If you are to store one million ASCII strings with an average size of 10 characters in a bloom filter, what would be the approximate space consumption, given an allowed false positive rate of 5%?.

2.1 What is a bloom filter

A Bloom filter is a data structure designed to tell you, rapidly and memory-efficiently, whether an element is present in a set. The price paid for this efficiency is that a Bloom filter is a probabilistic data structure: it tells us that the element either definitely is not in the set or may be in the set [1].

2.2 What is an advantage of bloom filters over hash tables

Bloom filters is more space efficient. Since it does not store the actual object.

2.3 What is a disadvantage of bloom filters

The bloom filter is a probabilistic data structure, this means that the result has a small risk of returning as a "false positive". This could mean that you get a positive result for an object that does not exist.

2.4 Using your language of choice, implement a bloom filter with add and check functions

We have made a very simple exapmle of a bloom filter in python, it can be found in the folder "bloom_filter".

2.5 If you are to store one million ASCII strings with an average size of 10 characters in a hash set, what would be the approximate space consumption

$$OneASCIIcharacter = 8bits$$

$$Stringlenght10 = 80bits$$

$$Timesamillion = 80.000.000bits$$

$$80.000.000bits = 10Mb(Megabytes)$$

For each entry the HashSet normally takes an empty space of equal size of the object as well as a pointer to the next entry. In Java the standart HashSet implementations memory consumption can be defined as

$$32 * SIZE + 4 * CAPACITY$$

The default Capacity of a HashSet implementation is 16
Therefore

$$32 * 1000000 + 4 * 16 = 32.000.064bytes = 30.5176Mb$$

So the approximate memory consumtion for a HashSet in Java for storing 1.000.000 10 letter string would be 30.5176 Mb

2.6 The following equation gives the required number of bits of space per inserted key, where E is the false positive rate

$$b = 1.44log_2(1/\xi)$$

2.7 How many bits per element are required for a 1% false positive rate

We insert the probability (procent as decimal) in the equation from section 2.6:

$$b = 1.44log_2(1/0.01) = 9.56715291328$$

We can now conclude that we need approx. 9.57 bits of space pr. inserted key, to get a 1% false positive rate.

2.8 How many bits per element are required for a 5% false positive rate

We insert the probability (procent as decimal) in the equation from section 2.6:

$$b = 1.44log_2(1/0.05) = 6.22357645664$$

We can now conclude that we need approx. 6.22 bits of space pr. inserted key, to get a 5% false positive rate.

2.9 If you are to store one million ASCII strings with an average size of 10 characters in a bloom filter, what would be the approximate space consumption, given an allowed false positive rate of 5%

We insert the space consumtion from section 2.8:

$$6.22357645664 * 1000000 = 6223576.45664$$

The approx. space consumtion would therefore be 6223576.46 bits that equals approx. 759.71 Kilobytes.

3 Task 3 - Huffman coding

HBase internally uses a compression that is a combination of LZ77 and Huffman Coding.

1. Generate Huffmann Code (and draw the Huffmann Tree) based on the following string: "beebs beepps!!!!! their eerie ears hear pears"
2. How many bits is the compressed string? How many bits is the raw ASCII string?
3. Compress "pete is here" with the Huffmann tree from before.
4. Write your own 10 word sentence. Generate the Huffmann Code (a new Huffmann Tree), and write a new compressed message (ie. in binary). Swap with one of your fellow students, and decompress each other's message.

We have drawn the huffman tree in paint, a png image can be found in `/huffman_coding/extra/HuffmanTree.png`. It's also printed to the console when running the project. An expected output can be found in the readme of the `huffman_coding` folder.

4 Task 4 - Map and Reduce

Solve the following using Javascript, for example in your browser's developer console.

1. Map the list of numbers to a list of their square roots: [1, 9, 16, 100]
2. Map the list of words so each is wrapped in a ¡h1¿ tag: [“Intro”, “Requirements”, “Analysis”, “Implementation”, “Conclusion”, “Discussion”, “References”]
3. Use map to uppercase the words (all letters): [“i'm”, “yelling”, “today”]
4. Use map to transform words into their lengths: [“I”, “have”, “looooooong”, “words”]
5. Get the json file comics.json from the course site. Paste it into your browser's Javascript console. Use map to get all the image urls, and wrap them in img-tags.
6. Use reduce to sum the array of numbers: [1,2,3,4,5]
7. Use reduce to sum the x-value of the objects in the array: [x: 1,x: 2,x: 3]
8. Use reduce to flatten an array of arrays: [[1,2],[3,4],[5,6]]
9. Use reduce to return an array of the positive numbers: [-3, -1, 2, 4, 5]

4.1 Map the list of numbers to a list of their square roots: [1, 9, 16, 100]

To map the list of numbers to their square roots can be completed in Javascript like so:

```
numbers = [1, 9, 16, 100];
squared = numbers.map(item => (item * item));
console.log(squared);

// Expected output:
// Array(4) [ 1, 81, 256, 10000 ]
```

4.2 Map the list of words so each is wrapped in a ¡h1¿ tag: [“Intro”, “Requirements”, “Analysis”, “Implementation”, “Conclusion”, “Discussion”, “References”]

To wrap the words with ¡h1¿ using the map function in Javascript can be done like so:

```
titles = ["Intro", "Re–quirements", "Analysis",
"Implementation", "Conclusion", "Discussion",
"References"];

hOnes = titles.map(item => "<h1>"+item+"</h1>");
console.log(hOnes);

// Expected output:
// Array(7) [ "<h1>Intro</h1>", "<h1>Re–quirements</h1>",
// "<h1>Analysis</h1>", "<h1>Implementation</h1>",
// "<h1>Conclusion</h1>", "<h1>Discussion</h1>",
// "<h1>References</h1>" ]
```

4.3 Use map to uppercase the words (all letters): [“i’m”, “yelling”, “today”]

To map the words to uppercase using the map function in Javascript can be completed like so:

```
lower = ["i'm", "yelling", "today"];
capital = lower.map(item => item.toUpperCase());
console.log(capital);

// Expected output:
// Array(3) ["I'M", "YELLING", "TODAY"]
```

4.4 Use map to transform words into their lengths: [“I”, “have”, “looooooong”, “words”]

To map the words their respective lengths using the map function in Javascript can be done like so:

```
words =  ["I", "have", "looooooong","words"];
leng = words.map(item => item.length);
console.log(leng);

// Expected output:
// Array(4) [ 1, 4, 10, 5 ]
```

4.5 Get the json file comics.json from the course site. Paste it into your browser's Javascript console. Use map to get all the image urls, and wrap them in img-tags

To wrap the image tags from the json file using the map function in Javascript can be done like so:

```
// Read the json from comics.json and
// when you have the json object in a
// variable in Javascript, you can
// map the image-tags around the image
// keys in the json object as below

imgs = json.map((item) => '<img src="' + item.img + '">');
console.log(imgs);

// Expected output:
// [
//     '<img src="https://imgs.xkcd.com/comics/red_spiders_2.jpg">',
//     '<img src="https://imgs.xkcd.com/comics/love.jpg">',
//     '<img src="https://imgs.xkcd.com/comics/schrodinger.jpg">',
//     '<img src="https://imgs.xkcd.com/comics/secrets.jpg">',
//     '<img src="https://imgs.xkcd.com/comics/counter-red-spiders.jpg">',
//     '<img src="https://imgs.xkcd.com/comics/found.jpg">',
//     '<img src="https://imgs.xkcd.com/comics/want.jpg">',
//     '<img src="https://imgs.xkcd.com/comics/penny_arcade.jpg">'
//  ]
```

4.6 Use reduce to sum the array of numbers: [1,2,3,4,5]

To reduce the list of numbers to their sum, can be completed in Javascript like so:

```
const numbers = [1,2,3,4,5];

const reduced = numbers.reduce((total, num) => {
    return total + num
}, 0);
console.log(reduced);

// Expected output:
// 15
```

4.7 Use reduce to sum the x-value of the objects in the array: [x: 1,x: 2,x: 3]

To map the list of numbers to their square roots can be completed in Javascript like so:

```
// reduce the x values in the objects to their sum
const objects = [{x: 1},{x: 2},{x: 3}];

const reducedX = objects.reduce((total, object) => {
    return total + object.x
}, 0);

console.log(reducedX);

// Expected output:
// 6
```

4.8 Use reduce to flatten an array of arrays: [[1,2],[3,4],[5,6]]

To reduce the array to a flattened array, the following procedure can be followed in Javascript:

```
const arrayToFlatten = [[1,2],[3,4],[5,6]];

const flattenedArray = arrayToFlatten.reduce((arr, inner) => {
    arr.push(inner[0])
    arr.push(inner[1])
    return arr
}, []);

console.log(flattenedArray);

// Expected output:
// [ 1, 2, 3, 4, 5, 6 ]
```

4.9 Use reduce to return an array of the positive numbers: [-3, -1, 2, 4, 5]

To return only the positive numbers in an array using reduce, the following procedure can be followed in Javascript:

```
const reduceToPositiveNumbers = [-3, -1, 2, 4, 5]

const positiveNumberArray = reduceToPositiveNumbers.reduce((arr, number) =
    if(number >= 0){
        arr.push(number)
```

```
    }
    return arr
}, [])

console.log(positiveNumberArray);

// Expected output:
// [ 2, 4, 5 ]
```

References

[1] Bill Mill. Bloom filters by example.

[2] the free encyclopedia Wikipedia. Cap theorem.

[3] Luc Perkins with Eric Redmond and Jim R. Wilson. *Seven Databases in Seven Weeks*. Andy-Hunt, 2018.