**Yotpo web similarities**

**Possible solutions I thought about:**

1. strait forward solution: map-reduce in 3 phases (**will be explained later on**)
2. represent the URL TAG as a matrix A of all the URLs as rows and all the tags as the columns.

The result will be the multiplication of A and A transpose, and for each URL we need to find the best 10 results in a minimum heap. (**C = A \* At**)

This solution requires to do a distributed matrix multiplication. I such a way we need to execute a pipeline that takes care of the matrix multiplication. in each stage of the pipeline some of the  **C** matrix**, cij = Ai \* Aj-transpose.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Matrix A** | **search** | **videos** | **photos** | **sports** |
| [**www.google.com**](http://www.google.com/) | **1** | **1** | **1** | **1** |
| [**www.youtube.com**](http://www.youtube.com/) | **1** | **1** | **0** | **1** |
| [**www.facebook.com**](http://www.facebook.com/) | **1** | **1** | **1** | **0** |

**This solution seems to be too much more complicated.**

I chose solution number one because of simplicity.

The second solution might be faster in terms of time complexity but It is something that needs to be measured, moreover, it’s more complicated to implement in short time.

**Chosen solution explanation:**

***Phase1:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input** | **map in** | **map out**  (key, value) | **shuffle key**  by the tag ti | **reduce in**  (key, value) | **reduce out** |
| w1,t1  w2,t2  w1,t2  w1,t3  w4,t1  w2, t6  w4, t2 | w1,t1  w2,t2  w1,t2 | (t1, w1)  (t2, w2)  (t2, w1) |  | (t1,w1)  (t1,w4)  t3, w1 | t1: w1 w4  t3: w1 |
| w1,t3  w4,t2 | (t3, w1)  (t2, w4) | (t2, w2)  (t2, w1)  (t2, w4) | t2: w2 w1 w4 |
| w2, t6  w4, t1 | (t6, w2)  (t1, w4) | (t6, w2) | t6: w2 |

***Phase2*:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input** | **map in** | **map out**  (key, value) | **shuffle key**  by the first w URL website | **reduce in**  (key, value) | **reduce out** |
| t1: w1 w4  t3: w1  t2: w2 w1 w4  t6: w2 | t1: w1 w4  t3: w1 | w1, (w4, 1)  w4, (w1, 1) |  | w1, (w4, 1)  w1, (w2, 1)  w1, (w4, 1) | w1, w4, 2  w1, w2, 1 |
| t2: w2 w1 w4 | w2, (w1, 1)  w1, (w2, 1)  w1, (w4, 1)  w4, (w1, 1)  w2, (w4, 1)  w4, (w2, 1) | w2, (w1, 1)  w2, (w4, 1) | w2, w1, 1  w2, w4, 1 |
| t6: w2 |  | w4, (w2, 1)  w4, (w1, 1)  w4, (w1, 1) | w4, w2, 1  w4, w1, 2 |

***Phase3:***

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **map in** | **map out** | **shuffle key**  by the **first** w in the tuple |
| w1, w4, 2  w1, w2, 1  w2, w1, 1  w2, w4, 1  w4, w2, 1  w4, w1, 2 | w1, w4, 2  w1, w2, 1 | (w1, w4, 2) (w4, 2)  (w1, w2, 1) (w2, 1) |  |
| w2, w1, 1 | (w2, w1, 1) (w1, 1) |
| w2, w4, 1  w4, w2, 1  w4, w1, 2 | (w4, w2, 1) (w2, 1)  (w4, w1, 2) (w1, 2)  (w2, w4, 1) (w4, 1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **reduce in**  (key, value) | **Grouping**  by the first wi in key | **Sorting**  by composite key, first wi and then the value | **reduce out**  (key, value)  le’s say **top 1** |
| (w1, w4, 2) (w4, 2)  (w2, w1, 1) (w1, 1)  (w1, w2, 1) (w2, 1)  (w2, w4, 1) (w4, 1) | (w1, w2, 1) (w2, 1)  (w1, w4, 2) (w4, 2)  (w2, w1, 1) (w1, 1)  (w2, w4, 1) (w4, 1) | (w1, w4, 2) (w4, 2)  (w1, w2, 1) (w2, 1)  (w2, w1, 1) (w1, 1)  (w2, w4, 1) (w4, 1) | (w1, w4, 2) (w4, 2)  (w2, w1, 1) (w1, 1) |
|  |  |  |  |
| (w4, w2, 1) (w2, 1)  (w4, w1, 2) (w1, 2) | (w4, w2, 1) (w2, 1)  (w4, w1, 2) (w1, 2) | (w4, w1, 2) (w1, 2)  (w4, w2, 1) (w2, 1) | (w4, w1, 2) (w1, 2) |

**Main issues to think about:**

1. Need to create a configuration file that decided the number of mappers and reducers in the system.
2. I didn’t think yet about memory management and caching of files
3. We might do some unnecessary calculations since any result (w1 w2 counter) we calculated also (w2 w1 counter) and maybe it’s something we can avoid.
4. I didn’t add a combiner to the mapper layer which might improve performance
5. load balancing - since we have some popular tags, if all thous tags are going to the same node, we will harm the parallelismof the map-reduce program. We might think about how to avoid it
6. Since each tag can have many URLs we might think on another direction of aggregate all the tags for each URL and then with a pipeline to calculate all the (wii wj counter) by multiply there tags vector, reminds the second solution I offered at the beginning.
7. We need to think about how to handle the cases of URLs from the same domain for example [www.ebay.uk](http://www.ebay.uk) and [www.ebay.us](http://www.ebay.us) which in my implementation are totally different URLs.