A Bloom Filter Application for Processing Big Datasets through MapReduce Framework

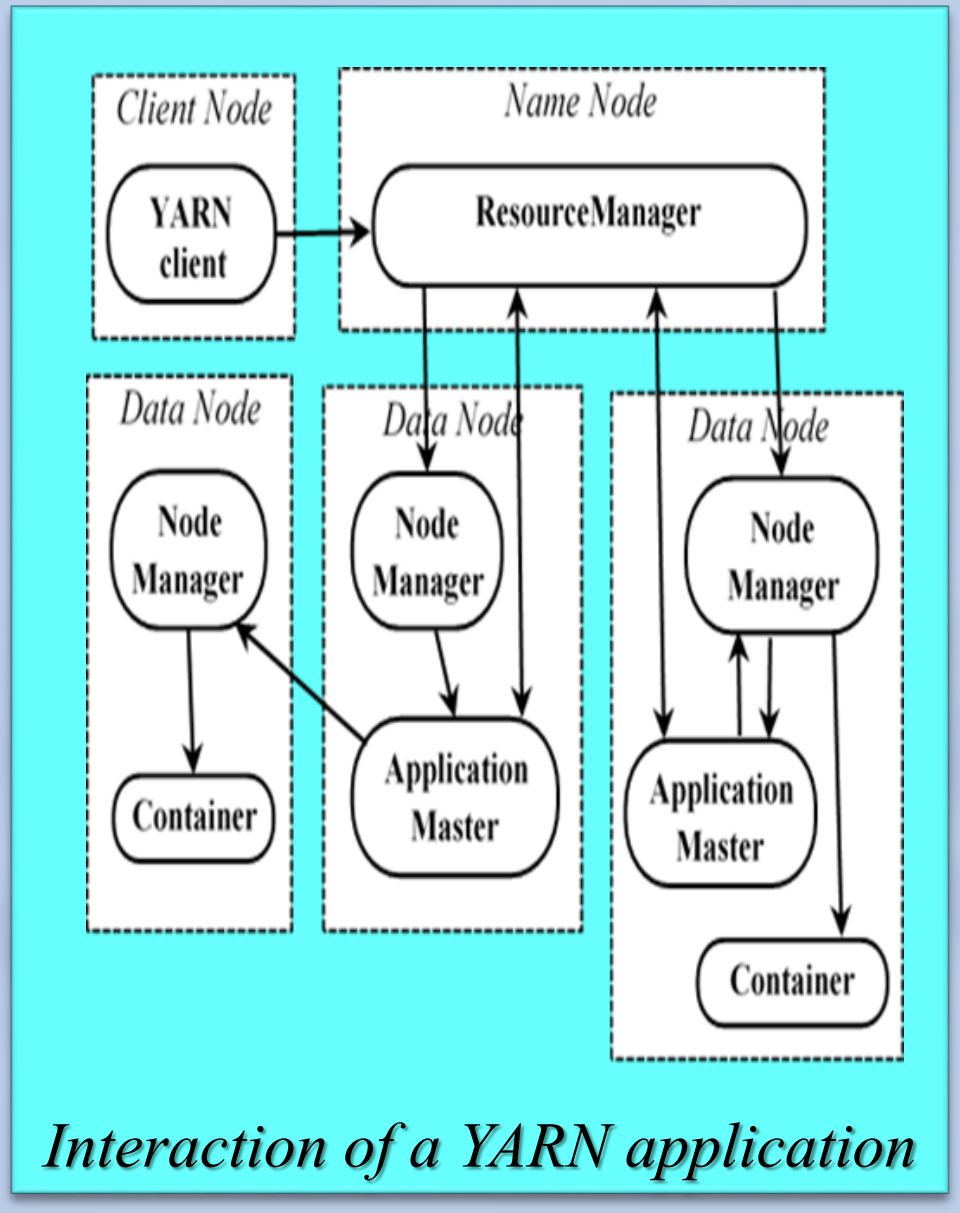
Assoc. Prof. Milko Marinov
Dept. of Computer Systems & Technologies
University of Ruse, Bulgaria

The Outline

- Introduction
- Architecture analysis of Hadoop 2
- Bloom filter general characteristics
- Bloom filter implementation on MapReduce framework
- Bloom filter use cases
- Conclusion

The main objective of the current research is to analyse the architecture and components of Hadoop 2.x with reference to identifying the factors which influence the productivity of MapReduce jobs. Based on this analysis and considering the advantages of probabilistic data structures, the study offers a Bloom filter implementation in a MapReduce framework environment.

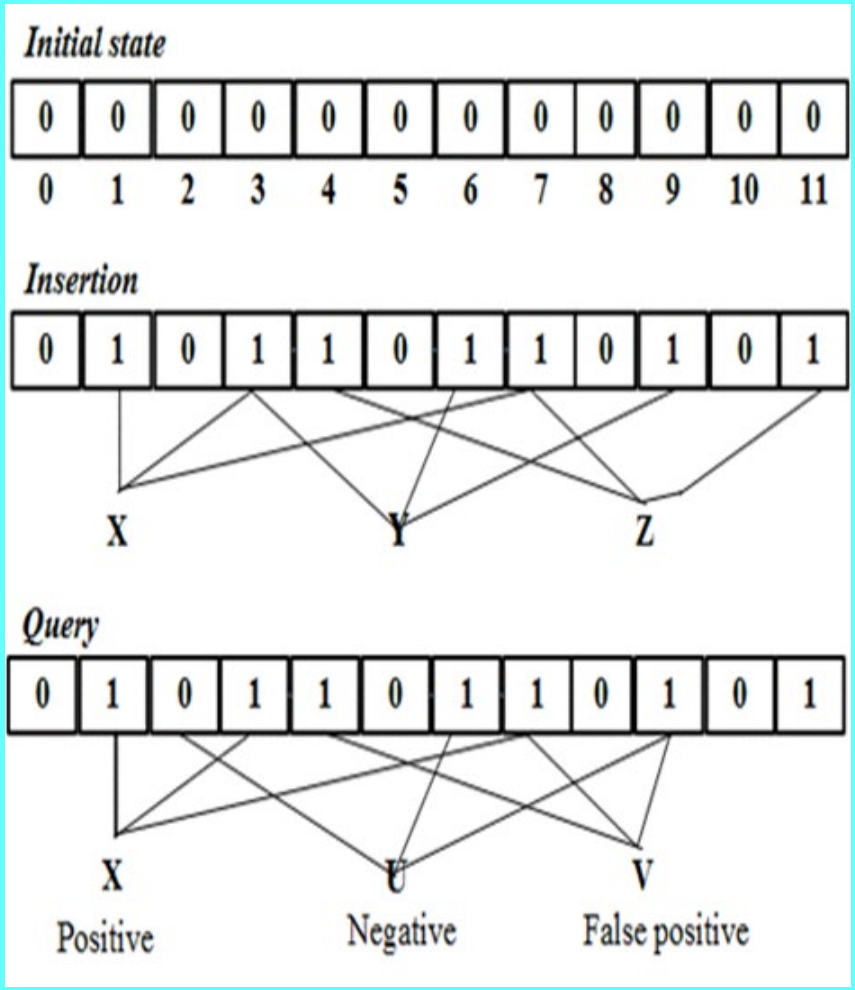
Architecture analysis of Hadoop 2



Interaction of a YARN application

- The main purpose of YARN is to separate resource management from the planning and monitoring of tasks. By separating resource management functions from the program model, YARN performs many tasks related to scheduling the components required for each job. In this way, it is completely removed from the static allocation of resources needed to implement the Map and Reduce functions, considering the resources of the cluster as a continuous space.
- In the second Hadoop version MapReduce is implemented as a YARN application.
- MapReduce is a programming model which runs on a cluster of commodity servers and is based on the shared-nothing architecture.
- The main operation performed by the Map function is to transform input data into intermediary key-value pairs. During the Reducing phase execution, each key-value pair must be checked to find out if the query condition is met.

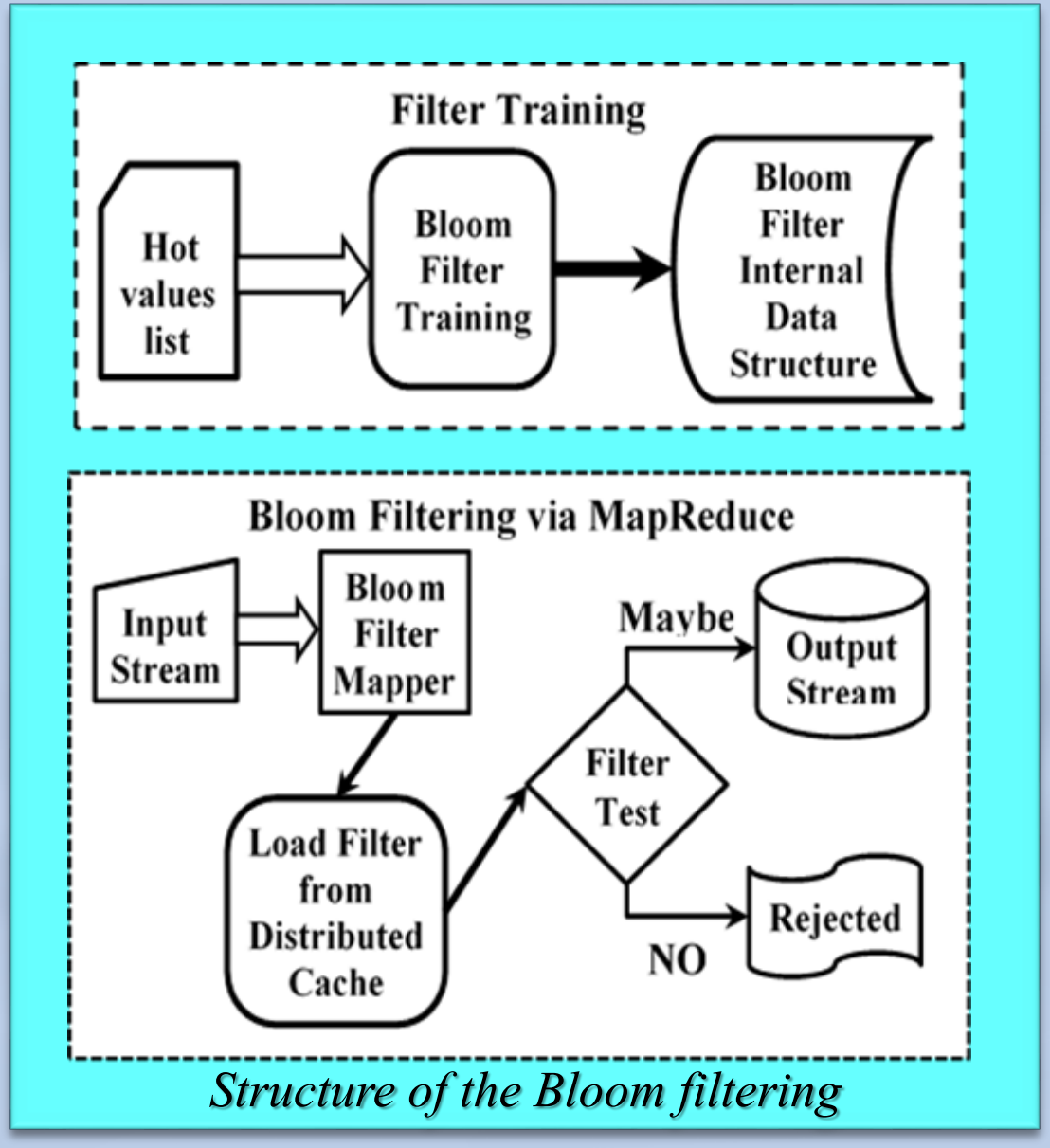
Bloom filter general characteristics



An example of the Bloom filteringter with m=12 bits and k=3 hash functions

- **The Bloom filter is a probabilistic data structure that requires very little space to represent a set of elements. It also supports a mechanism that verifies whether an element belongs to the set.**
- **The Bloom filter supports two operations, namely inserting an element and telling if an element is present in a set.**
- **The size of the filter (m), the total number of elements inserted in the filter (n) and the number of hash functions (k) used in the filter determine its accuracy.**
- **The probability of false positive results (p) depends on the number of added elements, i.e., when the number of elements added to the Bloom filter increases, the probability of obtaining false positive results also increases.**
- **When an element is inserted to the filter, it is fed to each of the k hash functions and an array consisting of k elements is formed.**
- **When a request for an element is sent (checking if the element is a member of the set), it is passed to each of the k hash functions.**

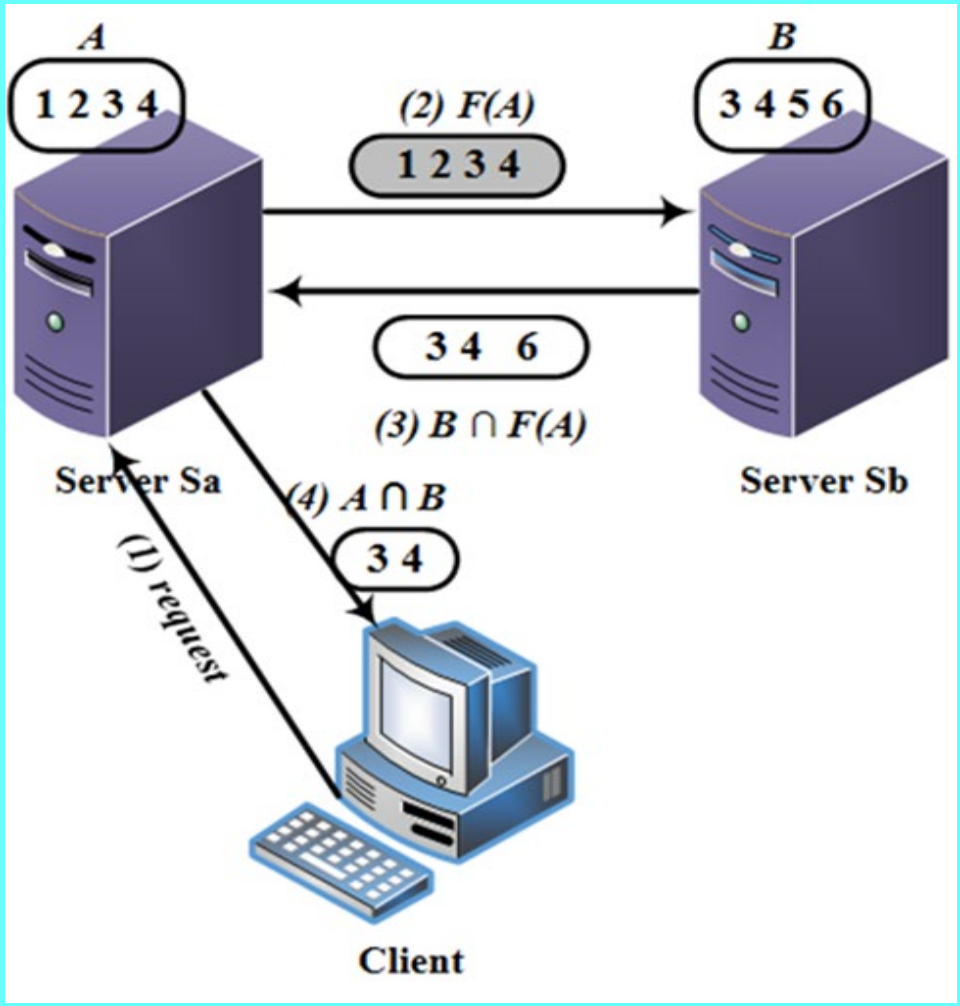
Bloom filter implementation on MapReduce framework



Structure of the Bloom filtering

- The process is divided into two steps.
- First, the filter must be trained from the set of "hot values". This is done by loading the data from where it is stored and adding each item to the Bloom filter. The resulting data object is stored in HDFS.
- Second, the actual filtering must be done. The Bloom filter is loaded from the distributed cache when the Map task is started. Then, in the Map function, an iteration is undertaken through the records and the Bloom filter checks for membership in the "hot values" list. The recording is saved or rejected depending on the Bloom filter affiliation test. The record is forwarded or not based on the Bloom filter membership test.
- A Reduce task is not executed because the records are analysed one by one and aggregation is not performed.

Bloom filter implementation on MapReduce framework



Bloom filter application in defining a remote intersection of two datasets

- A Bloom filter can be used to reduce a large number of unnecessary operations in any application where it is necessary to run a check on the presence or absence of an item before an expensive operation.
- One of the main applications of the Bloom filter is to present very large datasets in different applications. A dataset with millions of elements can take up gigabytes of memory. In addition, expensive input-output operations must be performed to transfer data from the disk. The Bloom filter can drastically reduce the number of bytes needed to represent datasets, allowing them to fit in the memory and reduce the time required for their reading.
- Reducing the number of database queries that return a lot of empty or negative results is a common application of the Bloom filters. By performing an initial Bloom filter test, the application can skip plenty of negative results even before sending the query to the database.

Conclusion

- The outlined suggestion for implementing a Bloom filter in the MapReduce framework environment provides good productivity and does not require considerable costs because the Bloom filter is loaded from the distributed cache. The value checking operation in the Bloom filter is also relatively inexpensive as each test is performed for a constant period of time. The Bloom filter is an ideal way to replace the presented value lists.
- Apart from this fact, however, the impact of false positive results must be carefully assessed before it is used for a specific application. When the Bloom filter is used in a distributed MapReduce framework, it is difficult to train actively the filter (very often, constantly) as it is done in a database. Once the Bloom filter is trained, separated in parts, and sent to HDFS, it can be easily read and used by other applications