Find K most frequent word in a really huge file

Problem: Given a very huge file (one that may not fit into several computer's hard disk), find the top K most frequent words in that file.

Solution (Case 1): If the file was a regular one, following steps would have given the top K most frequent words:

```
/***************************************************************************
1) Load the given file into RAM.
2) Split the big string into words using white-space and punctuation-marks as separator.
3) Put each word into a hash-map where word is key and frequency is value.
4) Find the top K most frequent words by using a min-heap, sorted on frequency.
   (Note: A min-heap is required here, not max-heap)
   Heap usage will give N*logK performance.
***************************************************************************/
```

File not fitting in RAM

If the file is extremely large such that it cannot be loaded into RAM of one machine, then multiple threads may be launched to load small parts of file into memory and then apply the above algorithm.
Assumption here is that the entire dictionary of words in the file is small enough to be loaded into RAM.

Dictionary not fitting in RAM

Now if the dictionary is also so huge that it cannot fit one machine's RAM, then this problem cannot be solved with one machine.

Multiple machines are required which will count the word frequency after loading pieces of file.
But if the file is equally divided among different machines, then results of individual machines will need to be merged.
This is so because same word may be present in different pieces of the file.

Plus, the full dictionary would be required on each machine which is again not possible due to huge size of dictionary.

This can be solved by using distributed hashing.
Use ***word.hashCode() % numProcessingMachines*** to get the machine number which will process that word.
This approach will evenly distribute all the words on all the processing machines and can even be done in parallel on chunks of file.
Each machine will roughly receive an equal number of words and will also contain only a chunk of the dictionary.

After this step, we need to find K most frequent words whose frequencies are already present on the processing machines.
Take 2 machines at a time and find their K most frequent words by using a max-heap.
numProcessingMachines/2 tasks can do this in parallel.
Then again merge these K heaps, 2 at a time until only one heap is left.

You have the K most frequent words of this extremely huge file and a comparably huge dictionary.