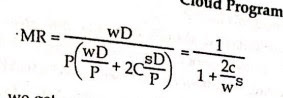
**PARALLEL EFFICIENCY OF MAP-REDUCE**

It is worth trying to figure out what is the parallel efficiency of MapReduce. Now let us assume that the data produced after the map phase is σ times the original data size D (σD), further, we assume that there are P processors which in turn perform map and reduce operations depending on which phase they are used in, so we do not have any wastage of processors. Also, the algorithm itself is assumed to do wD useful work, w is not necessarily a constant it could be D^2 or something like that, but the point is that there is some amount of useful work being done even if you have a single processor and that's wD. Now let us look at the overheads of doing the computation wD using MapReduce. After the map operation, instead of D data items, we have σD data items in P pieces, so each mapper writes data to their local disk. So, P there is some overhead associated with writing this data. Next, this data has to be read by each reducer before it can begin 'reduce operations, so each reducer has to read that one Pth of the data from a particular mapper. Since there are P different reducers, one Pth of the data in a map goes to each reducer and it has to read P of these from P different mappers once again getting us the communication time that a reducer spends getting the data it requires from the different mappers as P

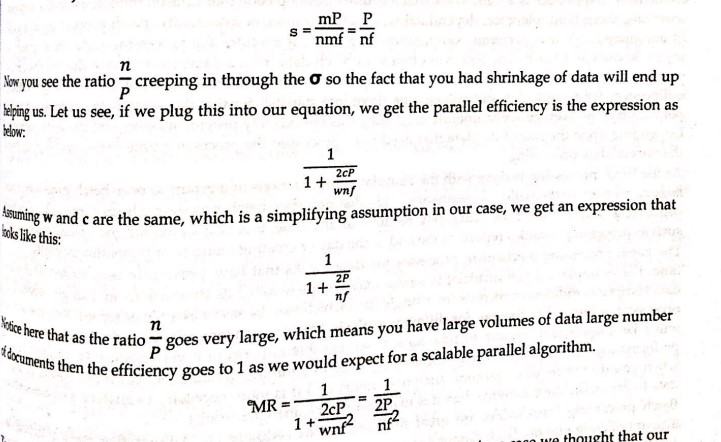
Overheads: σD/P

So, the total overhead is work that would not have to be done if we did not have parallelism, that is writing data to disk and reading data from remote mappers is 2D/P. Now if you look at the efficiency using this overhead, we get the wD is the time it takes on one processor which is the useful work that needs to be donewD/P; if we had P processors but we have this extra overhead of 2σD/P. We have got a constant c which essentially measures how much time it takes to write one data item to disk or to read a data item remotely from a different mapper. Now you assumed both of these to be the same constant.



Simplifying this parallel efficiency, we get an expression that is surprisingly independent of P, dependent on w and σ in a MapReduce important way though. It is nice that it is independent of P because it indicates that the is scalable and you can get large efficiencies even with a large number of processors, on the processors with the efficiency. However, it is dependent on σ and w. In process dependency particular, as long as the amount of useful work that you do per data item grows, the efficiency ends up there is no ing close to 1, which is nice. On the other hand, if the extra data that you produce after the map phase grows then efficiency can suffer and reduce considerably. So, it is very important to use things like Combiners to ensure that too much data does not get blown up after the map phase is done. In particular, if σ up being dependent on P then the statement here that efficiency is independent of P no longer holds so be careful when computing this expression. σ is quite likely to be dependent on P even though the expression hides that dependence.

This is quite insightful and will compute the parallel efficiency of our word-counting problem using this formula. When we were counting words we had n documents, m total possible words, each occurring say, f times per document on average, so the total volume of data to be processed is nmf. Now, after the map phase using combiners, we produce P into m partial counts, that is m different partial counts per mapper and their P mappers. At worst, there will be mP partial counts, some mappers may not produce counts for words that do not occur there, but we do not worry about that. We assume that at worst there will be mP partial counts which give us σ as the data after the map, as compared to the data before it and there you get P/nf.



To summarize, we have seen that MapReduce is indeed scalable in the previous case we thought that our expression was independent of D and P, but we found that he creeps in because of the σ. So the Sigma that our actually, extremely important the expansion or shrinkage factor after the map phase is critical for the efficiency of MapReduce algorithms since we have to make sure that is as smallest possible if you want a scalable implementation. In particular, it should certainly not grow with the number of processors otherwise we will have a blow-up inefficiency. If σ was growing with P, then you would end up having efficiency going down to 0 as a large number of processors were used. It appears that this is indeed the case here but notices that if you use a large number of processors and keep the same problem size you will get low efficiency as you have a lot of processors doing no work. But if you used larger and larger document sets then naturally, you would get better efficiencies which is what a scalable algorithm is supposed to be.