The groupby command in Spark and how we can optimize it

The groupby command in spark allows to group data according to a key and do some operation on them. Typical example would be

Location	*Product*	*sale*
NewYork	*Apple*	20
NewYork	*Banana*	30
London	*Apple*	10

we can group data according to product or location and do for example the sum of the entries.

The point is that the groupby operation can be over a very large number of entries. To achieve high performance, the standard technique for databases is to do a partition of the data by rows with each computational entry having access to the whole set of columns.

How the partition is done varies depending on the context. The database could be sorted but more often than not, it is not sorted and instead it is partitioned by hash of a subset of the columns.

For a standard computation like sum the computation would go along the following lines :

1. If the keys used for the groupby are the ones of the partitioning, no shuffle is needed. Otherwise, yes it is.
2. Do the shuffle if needed.
3. Do the sum of the the entries in question and create a new table.

That is the basic scheme but there are a number of ways to improve it :

1. If for some reason the table is sorted and the sorting keys correspond to the ones used by groupby, then no shuffling is needed.
2. Before doing the shuffling, we can do the sum of the entries locally and then sends the keys and their partial sums in the shuffling. We gain a lot here.
3. The big danger of groupby is when a specific key occurs a lot of time (e.g. "Country = USA") which slows everything down. The trick then is to further partitioning by introducing additional keys (e.g. "Postcode" or "State") which then allows to regain a good partitioning of the data.

Sums are easy, but a whole set of functionalities are covered by this : **mean**, **std**, **skew**, **kurtosis**, etc. For those instead of broadcasting a single scalar, we broadcast 2 or more entries. Some functionality are definitely harder to implement like **median** but there are ways to achieve parallelism there.

It is a little bit unclear to me what strategies Spark is using and what else can be done. Here are some ideas on how to get better performance :

1. Spark is written in Scala and runs on the JVM which is not necessarily as highly performing as native. But there has been improvement of the JVM implementation.
2. The hashing function is something to consider. I know that at some point after I left Bodo registered increases of performance in all categories that in my opinion could only be explained by better hash functions.
3. I do not know which hashmaps are being used internally in Spark. But there is room for improvement in this respect. By using the tessil hashmaps I had some improvement in
4. Mean and std are your bread and butter. But other functions like **percentile_approx** (which generalized the median) are more complex which presents more opportunity for improvement.
5. The operations processed by Spark are in its own language of
6. Other operations like **collect_list** makes Spark disable some optimization due to the large size of what is being

PS : I learned this kind of techniques at Bodo and used this later to great effect in mathematical computations : I computed a canonical form for the mathematical object being considered, which allow a hash to be computed and to partition the search space. I also used special hash maps for this work. See :

1. The lattice packing problem in dimension 9 by Voronoi's algorithm
2. Polyhedral: A framework for polyhedral computations

sed that to partition the object space

Mathieu Dutour Sikirić