Power Iterations

MAHOUT-796

November 25, 2011

1 Formulae

1.1 Modified approach to power iterations formula

Original formula

$$\mathbf{Q} = \mathrm{qr}\left[\left(\mathbf{A}\mathbf{A}^\top\right)^q \mathbf{A}\boldsymbol{\Omega}\right].\mathbf{Q},$$

$$\mathbf{B} = \mathbf{Q}^\top \mathbf{A}.$$

Modified version

$$\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega},$$

$$\mathbf{B}_0 = \left[\mathrm{qr}\left(\mathbf{Y}\right).\mathbf{Q}\right]^\top \mathbf{A},$$

$$\mathbf{B}_i = \left[\mathrm{qr}\left(\mathbf{A}\mathbf{B}_{i-1}^\top\right).\mathbf{Q}\right]^\top \mathbf{A},\ i \in [1..q].$$

Notation $\mathrm{qr}\left(\cdot\right).\mathbf{Q}$ means "compute **QR** decomposition of the argument and retain **Q** as a result".

Current combination of QJob and BtJob is essentially producing $\mathbf{B}_0^\top = \mathbf{A}^\top \mathrm{qr}\left(\mathbf{A}\boldsymbol{\Omega}\right).\mathbf{Q}$. Intermediate QJob results are QR blocks, not a final Q, so QJob is not terribly meaningful without BtJob.

See also section §2 for quick recap of $\mathbf{B}_0$ pipeline details. B pipeline takes 2 tasks with only one global sorter (map + map + combiner + shuffle-sort + reducer).

It seems that the task boils down to figuring out alternative pipeline modifications necessary to produce $\mathbf{B}_i^\top$. After that, algorithm proceeds as before with assumption of $\mathbf{B} \equiv \mathbf{B}_q$.

the existing processing will be equivalent to $q = 0$.

(Ted Dunning points out: Note that the **QR** decomposition is not strictly necessary with $q = 0$ (see https://issues.apache.org/jira/browse/MAHOUT-792). Instead, the Cholesky decomposition can be applied to $\mathbf{Y}'\mathbf{Y}$ and the resulting triangular matrix can be used to expose chunks of **Q** as required).

1.2 $\mathbf{B}_i$ pipeline (some new code)

$\mathbf{B}_i$ pipeline produces $\mathbf{B}_i^\top = \mathbf{A}^\top \mathrm{qr}\left(\mathbf{A}\mathbf{B}^\top\right).\mathbf{Q}$.

This is very similar to $\mathbf{B}_0$ pipeline with specfics being full multiplication of $\mathbf{A}\mathbf{B}^\top$ in the first job and first pass qr pushdown to the reducer of the first job:

- **map1:** $\mathbf{A}\mathbf{B}^\top$vertical blocks of outer products are produced
- **combiner1:** presumming vertical blocks of outer products of $\mathbf{A}\mathbf{B}^\top$.
- **reducer1:** finalizes summing up outer products of $\mathbf{A}\mathbf{B}^\top$ and starts qrFirstPass $\left(\mathbf{A}\mathbf{B}^\top\right) \rightarrow$ qr blocks.

- **mapper2, combiner2, reducer2** proceed exactly as mapper2, combiner2, reducer2 in $\mathbf{B}_0$ pipeline]] and output final $\mathbf{B}_i^\top$ with blocks corresponding to initial splits of $\mathbf{A}$ input.

Thus, this pipeline is 2 MR jobs with 2 sorts (map1 + combiner1 + shuffle and sort + reducer1 + map2 + combiner2 + shuffle and sort + reducer2).

1.3 Integration of Cholesky trick route for computing power iterations*

Also note that whatever route is chosen to calculate $\mathbf{B}_i = g(\mathbf{Y}_i)$, mathematically it should still be valid $\forall i \in [0..q]$ for as long as we assume

$$\mathbf{Y}_i = \begin{cases} \mathbf{A}\boldsymbol{\Omega}, & i = 0; \\ \mathbf{A}\mathbf{B}_{i-1}^\top, & i > 0. \end{cases}$$

So, if Cholesky trick allows to produce $\mathbf{B}_0$ efficiently, I see no reason why it could not be applied to producing the $\mathbf{B}_i$.

2 Quick recap of the $\mathbf{B}_0$ pipeline

2.1 QR

QR takes more than one *map-only* steps, but current implementation requires only 2 map jobs and that is thought to scale to a few billion rows assuming k+p=500 and with upper limit on RAM in mappers at 1G.

First QR step may also be pushed down to reducers of previous job (something i have plans to have as an option in $\mathbf{B}_0$ pipeline as well). Last QR step may be optimized to be first part of further execution tree in a MR job thus last Q product output may not be required (and it is not in $\mathbf{B}_0$ pipeline).

*Will cover in MAHOUT-797

2 QR steps do output intermediate blocks of Q and R, but this from I/O standpoint is basically as efficient as 1 MR multiplication job, if not even better, because full MR outputs map results too and sends them to reducers, but it also incurs sorts, which 2 map-only jobs do not incur.

The only I/O deficiency of 2 map-only jobs vs. 1 MR job is that intermediate output would be replicated thus increasing I/O but that can be addressed by forcefully reducing replication factor on the output of intermediate QR blocks.

2.2 Multiplications

- $\mathbf{Q}^\top\mathbf{A}$. This is a generic multiplication that requires full MR jobs with significant pressure on combiners that combine intermediate outer products.
- There are exceptions that do not require combiners when accumulator of the result can be fit in memory and accumulated in mappers in line (These are $\mathbf{B}\mathbf{B}^\top$ and potentially $\mathbf{Y}^\top\mathbf{Y}$ which require only upper-triangular accumulator of $(k+p)\times(k+p)$ geometry).
- The product $\mathbf{A}\boldsymbol{\Omega}$ is of course much simpler since the entire matrix $\boldsymbol{\Omega}$ is always available without taking any memory, so this is can be done inline in either mapper or reducer.

2.3 Execution plan of the $\mathbf{B}_0$ pipeline

This, execution plan of the $\mathbf{B}_0$ pipeline is:

Map1: qrFirstPass $(\mathbf{A}\boldsymbol{\Omega}) \rightarrow$ intermediate qr blocks

Map2: $[\text{qrLastPass}\,(\text{qr blocks})\,.\mathbf{Q}]^\top\mathbf{A} \rightarrow$ vertical blocks of the outer products of $\mathbf{B}_0^\top$

Combiner2: summing up intermediate outer products of $\mathbf{B}_0^\top$.

Reducer2: final sums of all outer products $\rightarrow \mathbf{B}_0^\top$.

Thus, currently it takes 1 map-only job + 1 MR job to get to $\mathbf{B}_0$. (In other words, 2 jobs + 1 sorter).

The caveat is that this process is not currently terribly performant while handling supersparse matrices as **Q** blocks are treated as dense.