Viewing email #994bc086d42e1c33c6088698551d7d107e10066b... (and replies):

Click to view as flat thread, sort by date

View Source Permalink Reply

From: Sergio Esteves <s...@gmail.com>
To: de...@tajo.incubator.apache.org
Subject: Hybrid Hash Join
Date: 2013/06/18 06:17:49
List: dev@tajo.apache.org

Hi,

I have been thinking of the partition phase in hybrid hash join and would like to discuss it with the rest of the community.

During the partition of data into buckets, a very important issue is how to deal with skewed buckets (many tuples with a certain same join key) that may cause hash table overflow, i.e., having partitions that do not fit in memory.

It is hard to divide the hash space into buckets with different sizes without knowing the join key distribution beforehand. Therefore, one possible way is to partition the overflow buckets 1 level further (this process may go on onto deeper levels). Other way that is sometimes followed, is to do very small partitions of the data and then combine them in accordance with their sizes (which can also fail if the number of duplicates inside a bucket is too high).

I have been discussing with my GSoC mentor, and we think the best and more efficient solution would be to have an histogram of the distribution of the join keys, so that we can partition more evenly the hash space through the buckets. Such histogram can be mantained for every table that can be used with a join (i.e., that have a foreign key or that can be used as foreign key to other table). Hence, it is necessary to update the histograms every time data is updated/deleted from the respective tables. Other way we were discussing was to build the histogram on the first time hybrid hash join physical operator is called with a certain join query (i.e., during query execution). On the second time the same query is submitted, or a similar query containing the same inner relation, the histogram was already there and thus hybrid hash join could be executed more efficiently. This latter approach has the drawback of ignoring changes between consecutive join calls. On the other hand, maintaining a histogram for every table and updating it every time changes occur can be costly and introduce some overhead.

What do you guys think?

Thanks.

View Source Permalink Reply

From: Jihoon Son <g...@gmail.com>
Subject: Re: Hybrid Hash Join
Date: 2013/06/18 07:54:01
List: dev@tajo.apache.org

Hi,

it's a good idea,
but i think that histograms can be used to decide the optimal physical execution plan.
In other words, the physical planner can selects the optimal plan based on the data distribution.

For example,
if a relation is sorted by a join key, the physical planner might decide to use the sorted merge join.
If the join key distribution is well balanced, the hash join can be used.

Making histograms requires scan of the whole relation.
This causes the high overheads to the system.

To reduce this overhead, Tajo can piggyback making histograms into the query processing.
Given a query, the global planner first checks whether there is a histogram of the relation relevant to the query.
If the histogram does not exist, the global planner builds a query plan including making a histogram of the relation.
After the histogram is made, the global planner and physical planner can decide the better query plan based on the data distribution.

Histograms should be maintained by the Tajo master, because it can be also used to build distributed plans.
I think that changes of a small portion of a relation is not common,
especially for large-scale data storage.
(As you know, for large data, the bulk load of whole data might be faster than updates of a small portion of the data.)

If you want to work about the physical query optimization,
I recommend that you separately proceed the work with the GSoC project.
If you do, I am prepared to give you any advice.

Thanks,
Jihoon

2013/6/18 Sergio Esteves <sr...@gmail.com>

> Hi,
>
> I have been thinking of the partition phase in hybrid hash join and would like to discuss it with the rest of the community.
>
> During the partition of data into buckets, a very important issue is how to deal with skewed buckets (many tuples with a certain same join key) that may cause hash table overflow, i.e., having partitions that do not fit in memory.
>
> It is hard to divide the hash space into buckets with different sizes without knowing the join key distribution beforehand. Therefore, one possible way is to partition the overflow buckets 1 level further (this process may go on onto deeper levels). Other way that is sometimes followed, is to do very small partitions of the data and then combine them in accordance with their sizes (which can also fail if the number of duplicates inside a bucket is too high).
>
> I have been discussing with my GSoC mentor, and we think the best and more efficient solution would be to have an histogram of the distribution of the join keys, so that we can partition more evenly the hash space through the buckets. Such histogram can be mantained for every table that can be used with a join (i.e., that have a foreign key or that can be used as foreign key to other table). Hence, it is necessary to update the histograms every time data is updated/deleted from the respective tables. Other way we were discussing was to build the histogram on the first time hybrid hash join physical operator is called with a certain join query (i.e., during query execution). On the second time the same query is submitted, or a similar query containing the same inner relation, the histogram was already there and thus hybrid hash join could be executed more efficiently. This latter approach has the drawback of ignoring changes between consecutive join calls. On the other hand, maintaining a histogram for every table and updating it every time changes occur can be costly and introduce some overhead.
>
> What do you guys think?
>
> Thanks.

--
Jihoon Son

Database & Information Systems Group,
Prof. Yon Dohn Chung Lab.
Dept. of Computer Science & Engineering,
Korea University
1, 5-ga, Anam-dong, Seongbuk-gu,
Seoul, 136-713, Republic of Korea

Tel : +82-2-3290-3580
E-mail : jihoonson@korea.ac.kr

View Source Permalink Reply

From: Hyunsik Choi <h...@apache.org>
Subject: Re: Hybrid Hash Join
Date: 2013/06/18 09:00:00
List: dev@tajo.apache.org

Sergio,

That's good idea. That is necessary work. However, now it would be better to adopt some of your approaches to the distributed plan part which has a global view of relations and running tasks.

Actually, I've been also concerned with hash partition on intermediate data between execution blocks using histogram. I believe that Tajo can handle most of skewed problems if the intermediate are evenly distributed across cluster nodes.

Probably, the straightforward solution is as follows. In the current implementation, a running subquery collects some statistics from running query units (tasks). The statistics of each task includes hash keys on specified columns and the number of bytes for each key. By using the statistics, Tajo could build evenly distributed sets of hash keys. They could be used to create join tasks with evenly distributed loads; the creation of even load tasks is not implemented yet and is the future plan. Like one of your approaches, we could store the statistics of joining tables into Tajo catalog. The statistics also would be reused for later query processing.

Best regards,
Hyunsik

On Tue, Jun 18, 2013 at 1:17 PM, Sergio Esteves <sr...@gmail.com>wrote:

> [...]

View Source Permalink Reply

From: Sergio Esteves <s...@gmail.com>
Subject: Re: Hybrid Hash Join
Date: 2013/06/28 03:38:07
List: dev@tajo.apache.org

OK, since there are two different problems that can be separated, I decided for now to put my efforts on the hybrid hash join operator.

Doing so, I started a draft with some design choices (mostly for the partition phase of the algorithm, since it is the most challenging part): https://www.dropbox.com/s/menm6vb2hv3jv0n/HybridHashJoin_v0.pdf

I checked some related work and did not find any better solution to deal with overflowed buckets caused by duplicated join keys.

Also, would it be possible to get histograms (like the ones described in the document), value bounded or with hash ranges, through the stats retrieved by the running subquery?

Feedback regarding the solution described in Section 2.1 is appreciated.

Thanks/Regards.

> [...]

View Source Permalink Reply

From: Jihoon Son <g...@gmail.com>
Subject: Re: Hybrid Hash Join
Date: 2013/07/02 04:05:52
List: dev@tajo.apache.org

Hi, Sergio

I wonder the performance of your idea.

Do you have any experiment plans?

Thanks,
Jihoon

2013/6/28 Sergio Esteves <sr...@gmail.com>

> [...]

--
Jihoon Son

Database & Information Systems Group,
Prof. Yon Dohn Chung Lab.
Dept. of Computer Science & Engineering,
Korea University
1, 5-ga, Anam-dong, Seongbuk-gu,
Seoul, 136-713, Republic of Korea

Tel : +82-2-3290-3580
E-mail : jihoonson@korea.ac.kr

View Source Permalink Reply

From: Sergio Esteves <s...@gmail.com>
Subject: Re: Hybrid Hash Join
Date: 2013/07/02 06:14:18
List: dev@tajo.apache.org

Hi Jihoon,

HHJ would perform optimally if there are no overflowing buckets.

After the implementation, I was thinking of taking some measures (e.g., processing time, number of I/O operations) and comparing the performance of HHJ with other join operators already implemented in TAJO.

If bucket overflow happens in the solution I described, the performance of HHJ could be worse than the performance of other join algorithms due to the extra I/O cost of rescanning. However, we could dispatch the overflowing join tuples to another join operator. Some papers in the literature follow such approach, e.g., the original HHJ paper suggests to attack overflowing buckets (due to duplicated keys) with nested loops join.

Thanks.

> [...]