I. Progress done this week:

1. Hypercube Partitioning:

+ Deal with poorly factorized numbers of machines r: looking in the range [r, r - r * 0.1] the number that has most number of prime factors. That is, only tolerate that 10% of the machines is not used.

```
private int findBestR(int r, double tolerate) {
        assert tolerate <= 1 && tolerate > 0.5;

        int bestR = r;
        List<Integer> bestPrimes = Utilities.primeFactors(r);
        for (int i = r - 1; i > r * (1 - tolerate) && i > 0; i--) {
                List<Integer> primeFactors = Utilities.primeFactors(i);
                if (primeFactors.size() > bestPrimes.size()) {
                        bestR = i;
                        bestPrimes = primeFactors;
                }
        }
        return bestR;
    }
```

+ Different cost models to compare the partitions: computation cost only (#tuples each machine), communication cost, both computation and communication costs: CostModel.java

+ Preliminary experiments: d: 2 to 5, r: 100 to 1000 by 100, timeout: 1s, relation sizes: 100, 1K, 10K.

https://docs.google.com/spreadsheets/d/16xMzGU7vIcNzt7Zo7ZemCQ8ZzlizEkPZ6g5B_aaG4qc/edit?usp=sharing

#dims	#joiners	relations	#regions	assignment	runtime(ms)
5	900	[10000, 10000, 1000, 1000, 100]	0	NA	>1s
5	1,000	[100, 100, 100, 100, 100]	0	NA	>1s
5	700	[10000, 10000, 1000, 1000, 100]	640	10-16-2-2-1	968

5	700	[10000, 10000, 10000, 100, 100]	640	10-8-8-1-1	845
5	900	[10000, 100, 100, 100, 100]	896	112-1-2-2-2	679

Integrating Squal and Hypercube:

Writing perform join, select tuple to join, create indexes, update indexes:
https://github.com/khgl/squall/commit/c589b5c745d26e4eab14646e50f437304ec45b2d

Microsoft Azur Environment

Microsoft Azur allows quick deployment of Storm Cluster via their HDInsight service:

- Specify the number of cluster node
- Visual Studio SDK for storm topology deployment
- Can specify the storage information.

Can we configure the version of storm ?
Can we add lib jars file to cluster library ?

Distributed storage use:

- Blobs: to use the existing text-based tables of tpch in squall code
- Table storage
- DocumentDB: NoSQL.

References:

http://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-tables/
http://azure.microsoft.com/en-us/services/documentdb/
http://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-blobs/
http://azure.microsoft.com/en-us/documentation/articles/virtual-machines-linux-tutorial/
https://msdn.microsoft.com/library/azure/dn535788.aspx
http://azure.microsoft.com/en-us/documentation/articles/hdinsight-storm-overview/
http://azure.microsoft.com/en-us/documentation/articles/hdinsight-storm-getting-started/
http://azure.microsoft.com/en-us/documentation/articles/hdinsight-provision-clusters/

http://azure.microsoft.com/en-us/documentation/articles/hdinsight-storm-deploy-monitor-topology/
http://azure.microsoft.com/en-us/documentation/articles/hdinsight-use-blob-storage/

- Distributed file system like Hadoop for Storm ?
- Set up storm in Microsoft trial account.

TPCH benchmark:
http://www.tpc.org/tpch/

The TPC Benchmark™H (TPC-H) is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions.

1) **Performance Metrics**: TPC-H Composite Query-per-Hour Performance Metric (QphH@Size)
- Selected database size against which the queries are executed
- The query processing power when queries are submitted by a single stream
- The query throughput when queries are submitted by multiple concurrent users.
- The TPC-H Price/Performance metric is expressed as $/QphH@Size.

http://www.tpc.org/tpc_documents_current_versions/pdf/tpch2.17.1.pdf

2) **How to use DBGen?**
DBGEN is a database population program for use with the TPC-H benchmark.
It is written in ANSI 'C' for portability, and has been successfully ported to over a dozen different systems. While the TPC-H specification allow an implementor to use any utility to populate the benchmark database, the resultant population must exactly match the output of DBGEN. The source code has been provided to make the process of building a compliant database population as simple as possible.
Like DBGEN, QGEN is controlled by a combination of command line options and environment variables. Command line options are assumed to be single letter flags preceded by a minus sign. They may be followed by an optional argument.

DBGEN built:

1. Download TPC-H Data Generator (dbgen)
2. Unzip and cd dbgen, *cp makefile.suite* file in *tpch* directory and change some parameters in copied file.(103~112 lines)
    - CC

- DATABASE
- MACHINE
- WORKLOAD

set the parameter according to your machine and database;

For example in mac:

1)config.h:

add definition:

```
#ifdef MAC

#define _POSIX_C_SOURCE 200112L

#define _POSIX_SOURCE

…...//the same with other system

#endif /* MAC */
```

2)dss.h:

add: #define PR_HUGE_LAST(f, str) dbg_print(DT_HUGE, f, (void *)str, 0, 0)

3)rnd.c

modify “#ifdef LINUX” to “#if (defined(LINUX)||defined(_POSIX_SOURCE))”

4)varsub.c

add: #include "config.h"

Or referenced to https://github.com/electrum/tpch-dbgen. It is for Mac OS but the TPC-H is not the newest one.

3.Run it to produce the data files (***.tbl*** files) and queries.

DBGEN usage:
Reference to readme file in dbgen.
There are Command Line Options for DBGEN and QGEN,Sample DBGEN executions and etc.
And there are more specific explanation and examples in TPC-H specification.
For the usage of qgen:
if following error occurs: Open failed for ./1.sql at qgen.c:170, do: cp -f queries/*.sql ./

Queries Analysis -- 22 TPCH benchmark queries:

http://www.tpc.org/tpc_documents_current_versions/pdf/tpch2.17.1.pdf

Q1: Pricing Summary Report Query

=> Can be run in parallel. Only access information from one table.

Q2: Minimum Cost Supplier Query

This query contains a nested select statement.

Q3: Shipping Priority Query

```
select
        l_orderkey,
        sum(l_extendedprice*(1-l_discount)) as revenue, o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = '[SEGMENT]' and c_custkey = o_custkey
        and l_orderkey = o_orderkey and o_orderdate < date '[DATE]' and l_shipdate > date '[DATE]'
group by l_orderkey,
        o_orderdate,
        o_shippriority order by
        revenue desc, o_orderdate;
```

=> This query should be able to parallelize

How to perform multiple GroupBy ?

Q4: Order Priority Checking Query

Nested select query. Won't be able to run parallel

Q5: Local Supplier Volume Query

The query join 6 relations

Q6: Forecasting Revenue Change Query

Yes, this query should be easily parallelized. It retrieves data from only a single relation.

Q7: Volume Shipping Query

Complicate join

GroupBy multiple column

SortBy multiple column

Q8: National Market Share Query

Nested select query. Won't be able to run parallel

Q9: Product Type Profit Measure Query

Nested select query. Won't be able to run parallel

Q10: Returned Item Reporting Query

Should be able to parallelize. A lot of groupBy columns

Q11: Important Stock Identification Query

Nested select query. Won't be able to run parallel

Q12:Shipping Modes and Order Priority Query

This query can be able to parallelize, similar to Q3

Q13:Customer Distribution Query

Nested select query. Won't be able to run parallel

Q14: Promotion Effect Query

This query can be able to parallelize, just join

Q15: Top Supplier Query

Need to create a view first. Nest query as well. Won't be able to run parallel

Q16: Parts/Supplier Relationship Query

Nested select query.

Q17: Small-Quantity-Order Revenue Query

Nested select query.

Q18: Large Volume Customer Query

Nested select query.

Q19: Discounted Revenue Query

This query can be able to parallelize, many comparable attributes

Q20: Potential Part Promotion Query

Complicate nested query.

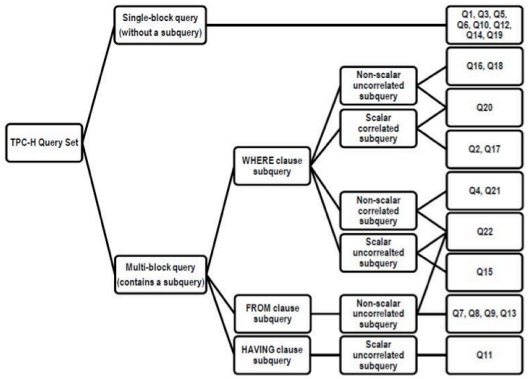
Q21: Suppliers Who Kept Orders Waiting Query

Complicate nested query.

Q22: Global Sales Opportunity Query

Complicate nested query.

The TPC-H queries can be separated into single-block query having no subquery, and the multi-block query having a subquery in FROM, WHERE or HAVING clauses. For multi-block query, it is categorized into scalar and non-scalar subqueries with correlated and uncorrelated data. A scalar subquery returns exactly one value while the non-scalar subquery returns a table. The correlated subquery has its inner block referred to tables in the outer block while the inner block of the uncorrelated subquery does not refer to tables the outer block.



II. Discussions in the meeting:

1. Hypercube partitioning

a. Tolerance of #machines: 50% (e.g. 1999 -> 1000)

b. Efficient implementation from Zhang paper (section 5.1):

Step 1: solve the following equations:

1. r_d1 x r_d2 x … r_dn = r (the number of reducers)
2. S_d1 x r_d2 x r_d3 x … r_dn = S_d2 x r_d1 x r_d3 x … r_dn= ....

Example 1: matrix partition: R1 x R2, and a pre-defined number of reducers r

We have the equations:

r_1 x r_2 = r

|R1| x r_2 = |R2| x r_1

⇒ r_1^2 = r x |R1| / |R2|

Example 2: cube partition: R1 x R2 x R3 and a pre-defined number of reducers: r

We have the equation:

r_1 x r_2 x r_3 = r

|R1| x r_2 x r_3 = |R2| x r_1 x r_3 = |R3| x r_1 x r_2

⇒ r_1^3 = r x |R1|^2 / (|R2| x |R_3|)

Step 2: Round up fraction numbers such that r_d1 x r_d2 x … r_dn \in [r/2,r].

For example, r_d1 = 4.08 try 4 and 5, r_d2 = 1.55 try 1 and 2.

Step 3: **Take the best assignment** with computation cost (p_d1 x p_d2 x p_dn) and communication cost (p_d1 + p_d2 + p_dn), where p is the largest partition (region).

2. Integrate Squall and Hypercube: check the Java interface

3. Local join index:

a. Discussed the concepts: which attribute is index, index type (hash, btree), comparison predicate, index key and rowID, value to index.

b. Important implementation points:

+ The join predicate is already generalized for multi-way
+ When a tuple from a relation comes (in stream), only the index of that relation is updated. The other indexes are just used for looking up.

+ Most challenging point is finding which relation corresponds to what inde

4. Microsoft Azure:

a. Check HDSight service configuration: storm version, file system (hdfs, local, etc.)
b. Check one-click installation solutions: storm-deploy, clj, jzmq

5. TPCH Queries:

a. Checkout existing query plans: TPCH7Plan.java. Ignore orderBy (because of data stream). groupBy is supported: GroupBy(ValueExpression), ColRef<ColRef,<ColRef>>
b. Try query plan for simple queries
c. Re-check nested select query and answer the questions:

+ Why/Why not parallelizable? (e.g. the result of one node depends on that of other node)
+ Can we communicate between machines? How much? How many machines involve?
+ Can we adjust the partitioning accordingly?
+ Can we replicate some parts of the information?

III. Plan for next week:

Hypercube partitioning: new implementation
Integrate into Squall: align interfaces
Local join index: tentative changes
Microsoft Azure:

- Find more information on HDInsight - Which storm version? how much control we have over the cluster?
- Store data in the cluster using Microsoft Storage Account - Blob Storage for text files - Development work needed for this ?

TPCH Queries:

- Take 3 examples to do detail analysis:
    - Whether it makes sense to use hypercube partition
    - With nested queries, what is the communication involved? is there any changes needed