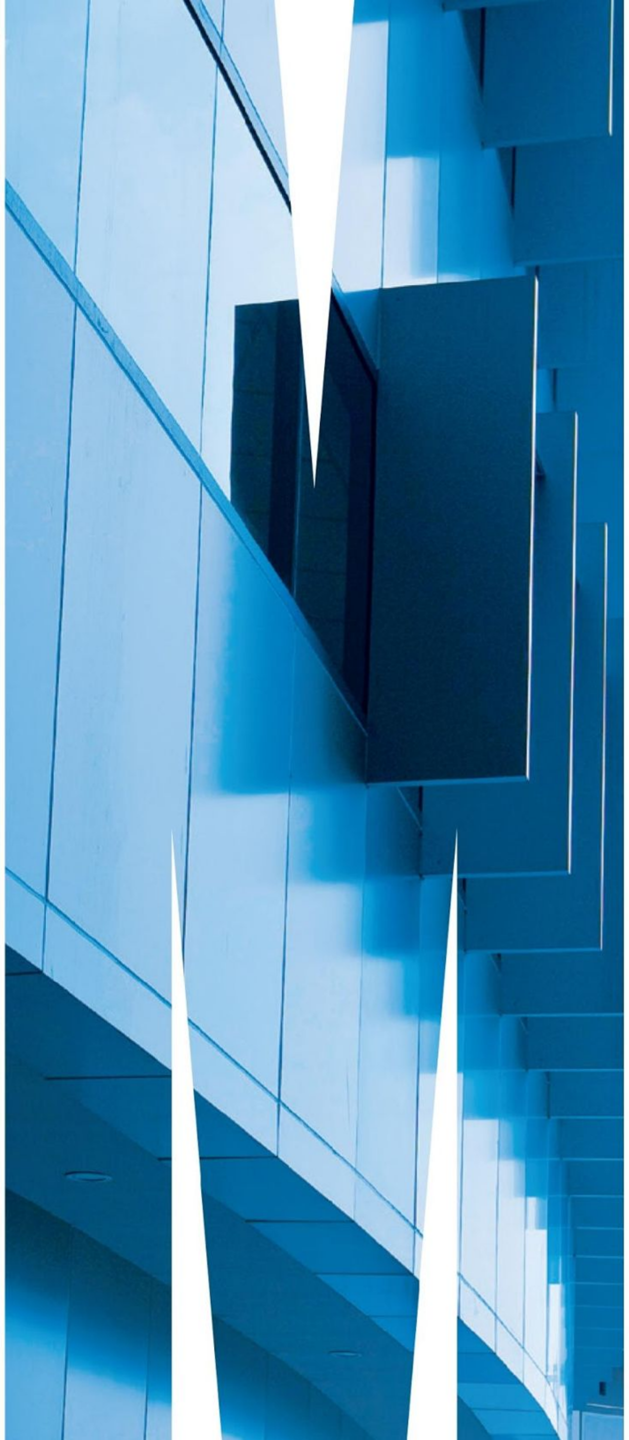
Query - Based Join

A solution for parallel outer joins with highly skewed data in cloud environments [1].



Introduction and Background

Why outer joins?
Outer joins are the most frequent type of query operations in a database [2], prevalent in web domains and major e-commerce websites where customer ids are linked with their corresponding data [3].

Why skewed data?
A typical challenge encountered while handling data is their natural skewness following the Zipf distribution, and the resulting runtime and network load consequences during an outer join query [4].

Why cloud environments?
As data applications continue to expand in scale, cloud environments play a crucial role in enabling the scaling out of large applications through parallelization and expanded storage [1].

Problem Statement

Most outer join algorithms follow these two approaches [3]:

1) Hashing based join
2) Duplication based join

Problem when dealing with skew:

- Hash based joins have hotspots since most of the values get hashed into the same partition, causing hotspots [5].
- Duplication based joins have hefty network costs making it extremely expensive. Especially in a shared-nothing environment [5].
- Some algorithms deal with skew by already knowing the type of skew [1]. Unfortunately, this condition is not always met in the real world and as such we must look into an alternative.

Hypothesis

The hypothesis of the paper is that the proposed query-based algorithm implemented on the Spark framework for outer joins on highly skewed data in a shared-nothing system is more efficient than the conventional approaches (referenced as [6], [3], and [7]) in a cloud-computing environment in terms of runtime and network performance.

Article contribution

1) Proposing a new join algorithm for efficiently handling skew in parallel outer joins [1].
2) Provide instructions to implement the **query-based join algorithm**.
3) Provide in-depth analysis of state-of-the-art techniques in DBMS: **PRPD**, **DER**, and PRPD + DER and their join performances.

Related work

The query-based algorithm aims to address two key problems:

1) Handle skewed data without prior knowledge.
2) Reduce the network load.

Therefore we need to keep these 2 points in mind when comparing the alternatives.

State-of-the-art methods:

1) PRPD
2) DER
3) PRPD+DER
4) OJSO

Related work (continued)

Method	Pros	Cons
Hash	- Can achieve near-linear speedup [8].	- Greatly sufferers from skew due to hot spots [1].
DER [3]	- Cost of communicate ids is less than tuples. - Can deal with skew for small-large joins	- Suffers from network load when R is large since it has to broadcast it to all nodes [3].
PRPD [6]	- Avoids distributing skewed data. - Only redistributes a small amount of data [1] (less network load).	- Assumes the nature of skew is known prior to the join. - Runtime suffers during high selectivity.
PRPD+ DER [1]	- Can handle data skew. - Uses DER to deal with the high join cardinality problem [1].	- Inherits the same issues of PRPD, requires knowledge of skew.
OJSO [7]	- Deals with skew by redistributing only the non-matched data.	- Does not address attribute skew [1].

MONASH University

Methodology - Approach

1) **R distribution**
2) **Pushing query keys**
   a) Extract the unique keys of S.
   b) Transfer the unique keys to remote nodes.
3) **Local outer join**
   a) Local outer join between Ri and the unique keys.
   b) Results without a match are finalized.
4) **Inner join**
   a) Perform local inner join between the matched results and S.

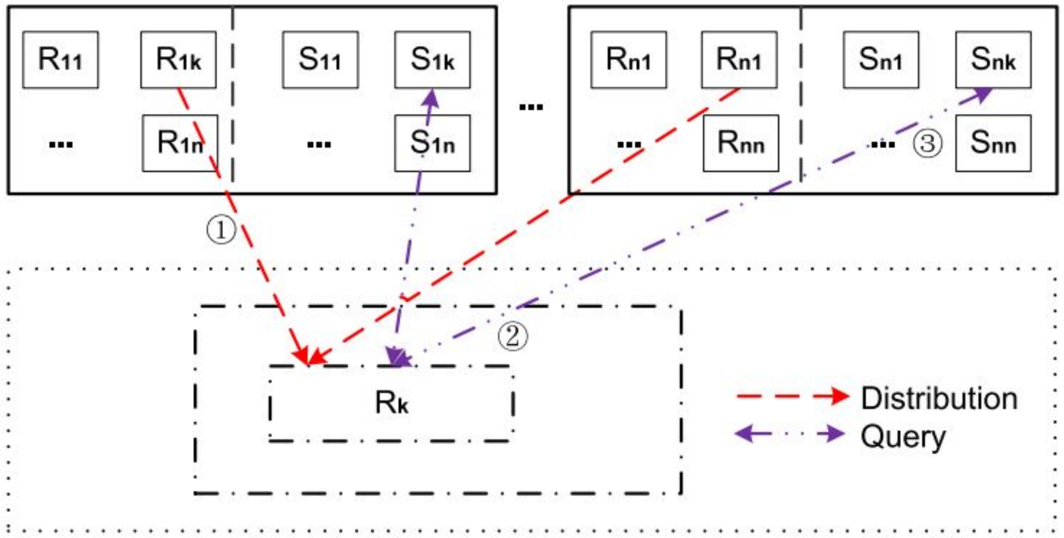
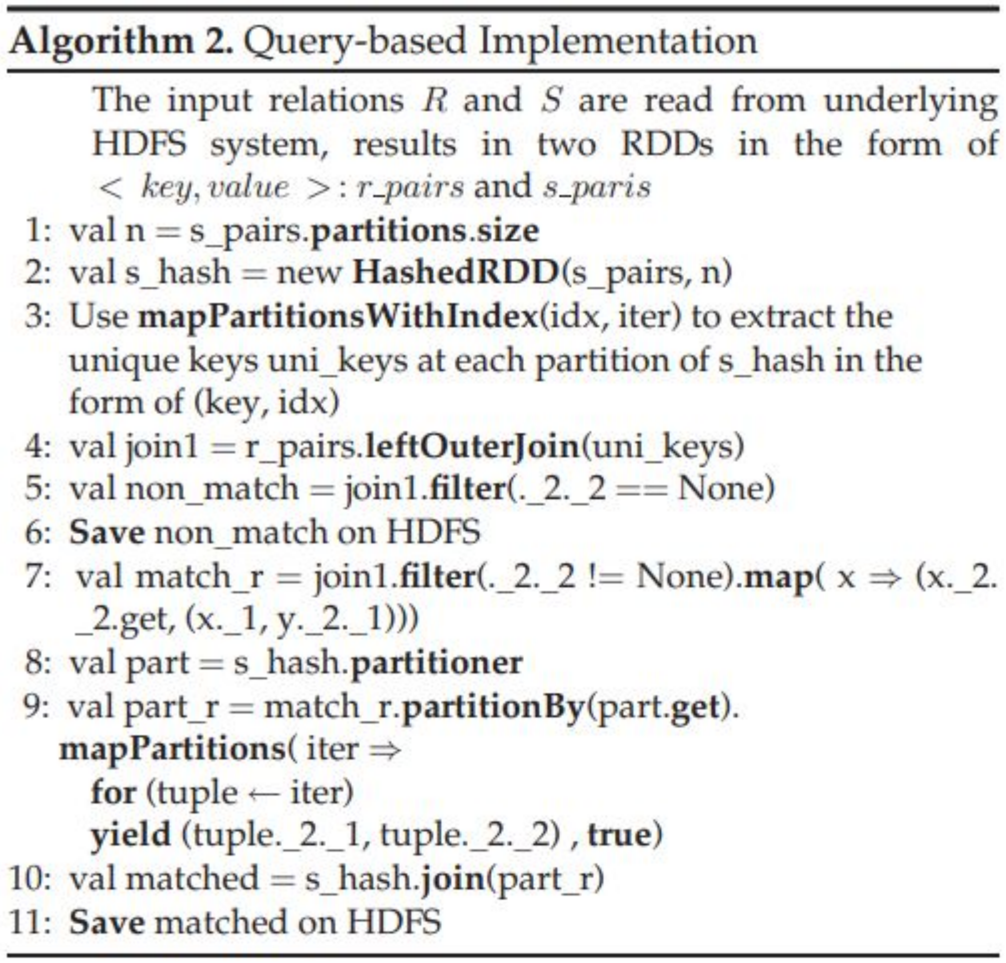


Fig. 1.The query-based approach for outer joins. The dashed square refers to the remote computation nodes and objects [1, Fig.3]

Methodology - Implementation

1) Read R & S from HDFS as RDDs in the form of (Key, Value) pairs.
2) Store S as a HashedRDD (s_hash).
3) Extract unique keys with index
4) Perform left outer join between R and unique keys.
5) Filter the matched and non-matched results
6) Write non-matched to HDFS
7) Perform inner join between matched and s_hash
8) Write result to HDFS



Algorithm 2. Query-based Implementation

```
    The input relations $R$ and $S$ are read from underlying
    HDFS system, results in two RDDs in the form of
    $< key, value >$: $r\_pairs$ and $s\_paris$
 1: val n = s_pairs.partitions.size
 2: val s_hash = new HashedRDD(s_pairs, n)
 3: Use mapPartitionsWithIndex(idx, iter) to extract the
    unique keys uni_keys at each partition of s_hash in the
    form of (key, idx)
 4: val join1 = r_pairs.leftOuterJoin(uni_keys)
 5: val non_match = join1.filter(._2._2 == None)
 6: Save non_match on HDFS
 7: val match_r = join1.filter(._2._2 != None).map( x ⇒ (x._2.
    _2.get, (x._1, y._2._1)))
 8: val part = s_hash.partitioner
 9: val part_r = match_r.partitionBy(part.get).
    mapPartitions( iter ⇒
      for (tuple ← iter)
      yield (tuple._2._1, tuple._2._2) , true)
10: val matched = s_hash.join(part_r)
11: Save matched on HDFS
```

Fig. 2. Query-based algorithm pseudocode [1].

Methodology - Justification

Overall the methodology addresses the two main issues identified earlier since:

1) Skew **does not** negatively **impact the runtime** and efficiency of the outer join since we use the **unique keys.**
2) By only distributing the unique keys we significantly **reduce the network load**, which is an important factor in a shared-nothing architecture.

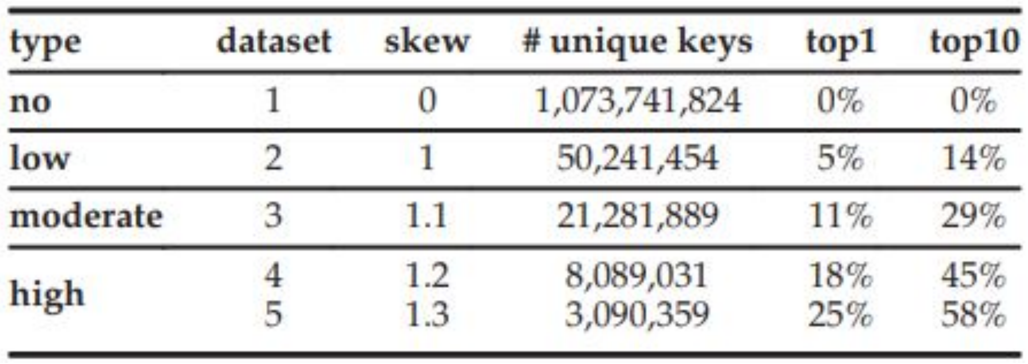
Analysis of Results - Runtime

The testing system has nodes with 12 cores (two six-core Intel Xeon CPU X5660 processors) running at 2.80GHz, 48GB RAM, one 128GB SSD local disk, and InfiniBand connectivity. The software stack is Spark 1.2.1, Hadoop 1.2.1, Scala 2.10.4, and Java 1.7.0_25.

The authors conduct the tests using different levels of skewed datasets to analyze:

- Runtime
- Load balancing
- Network load
- Scalability

Details of the Skewed Datasets



type	dataset	skew	# unique keys	top1	top10
no	1	0	1,073,741,824	0%	0%
low	2	1	50,241,454	5%	14%
moderate	3	1.1	21,281,889	11%	29%
high	4	1.2	8,089,031	18%	45%
	5	1.3	3,090,359	25%	58%

Network Load

- Without skew, hash, PRPD+Dup, and PRPD+DER have identical shuffle reads due to even data distribution, optimizing workload distribution.
- The query algorithm exhibits reduced network communication with higher levels of skew.
- The query algorithm transfers only unique keys and retrieves matched tuples in R without moving any tuples in S. This approach results in smaller redistribution compared to other algorithms.

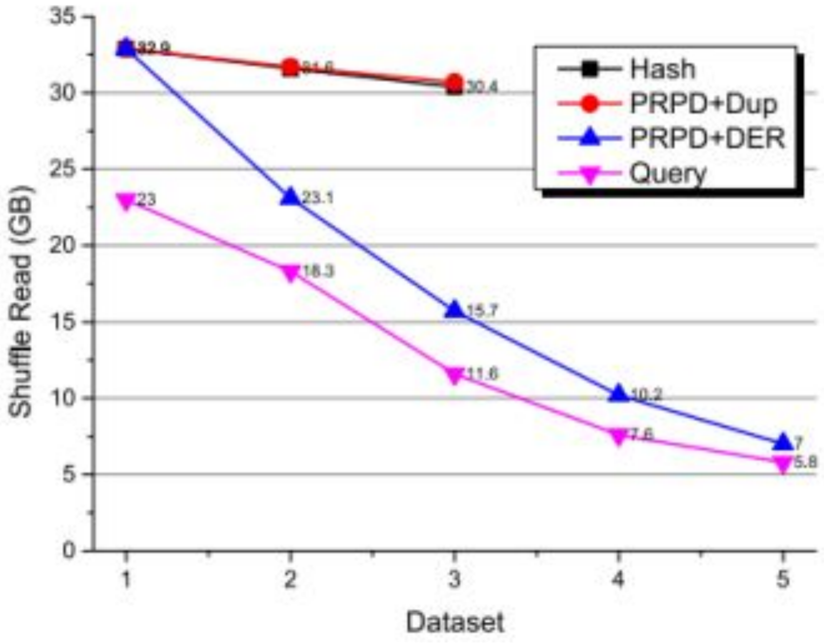


Fig. 5. Intermachine communication with selectivity factor 50%[1, Fig. 7].

Load Balancing

- There is significant variation in the amount of data read from remote executors for Hash and PRPD+Dup, indicating poor load balancing under skew.
- On the other hand, PRPD+DER shows improvement under skewed conditions. However, the query-based algorithm still outperforms it.

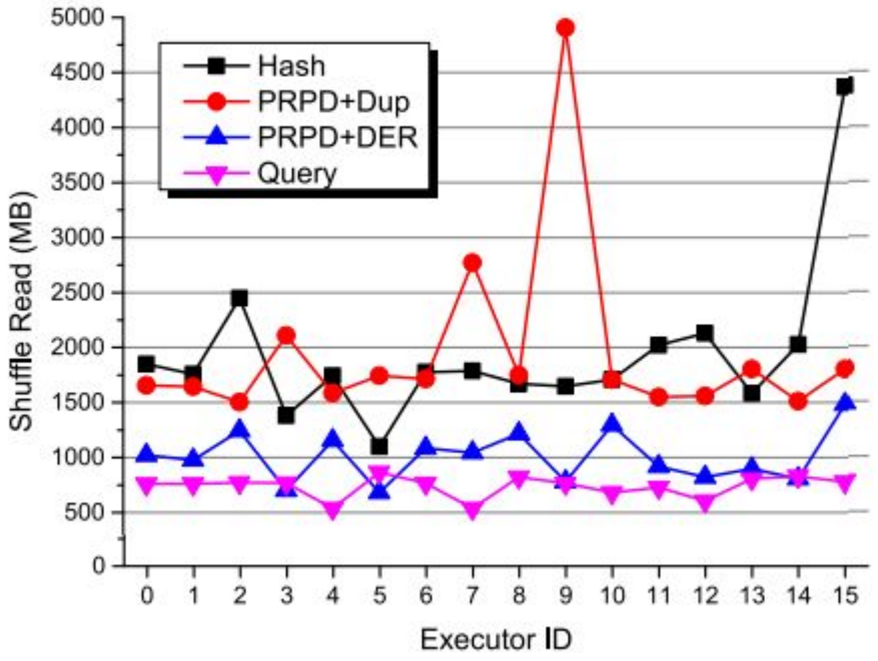


Fig. 6. Retrieved data for each executor at selectivity factor 50% and skewness of 1.1[1, Fig.8].

Scalability

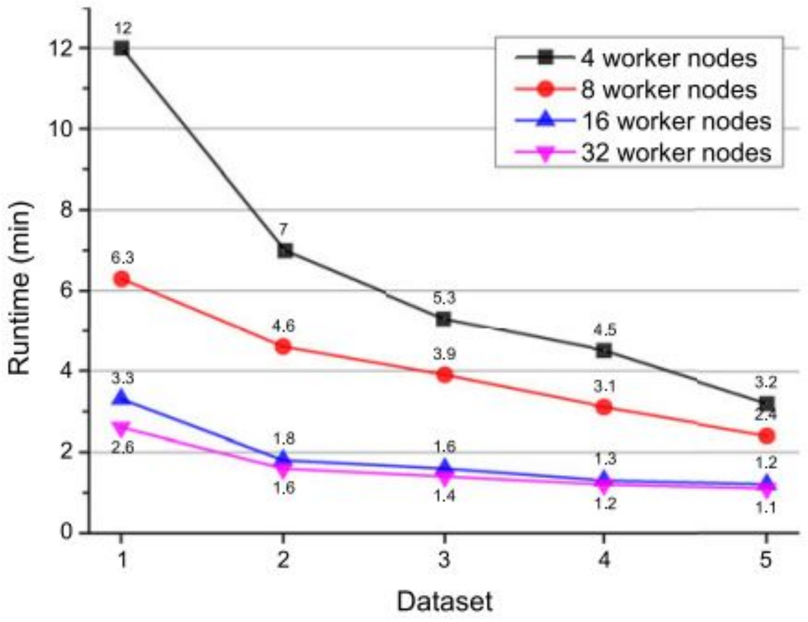


Fig. 7. Runtime of query-based join with different numbers of nodes & selectivity of 50%[1, Fig.9].

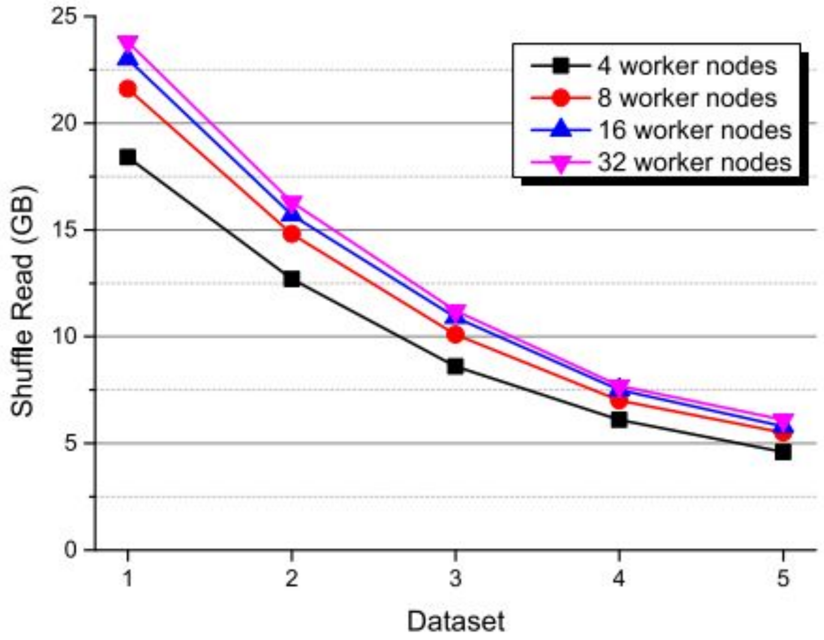


Fig. 8. Intermachine communication of query-based join with different numbers of nodes & selectivity of 50%[1, Fig.10].

In Fig. 7, the query-based algorithm exhibits faster runtime as the number of nodes increases, but the speedup begins to stagnate beyond 16 nodes. Meanwhile, in Fig. 8, the intermachine communication increases as the number of nodes increases. However, the authors argue that this may be due to platform overhead.

Runtime

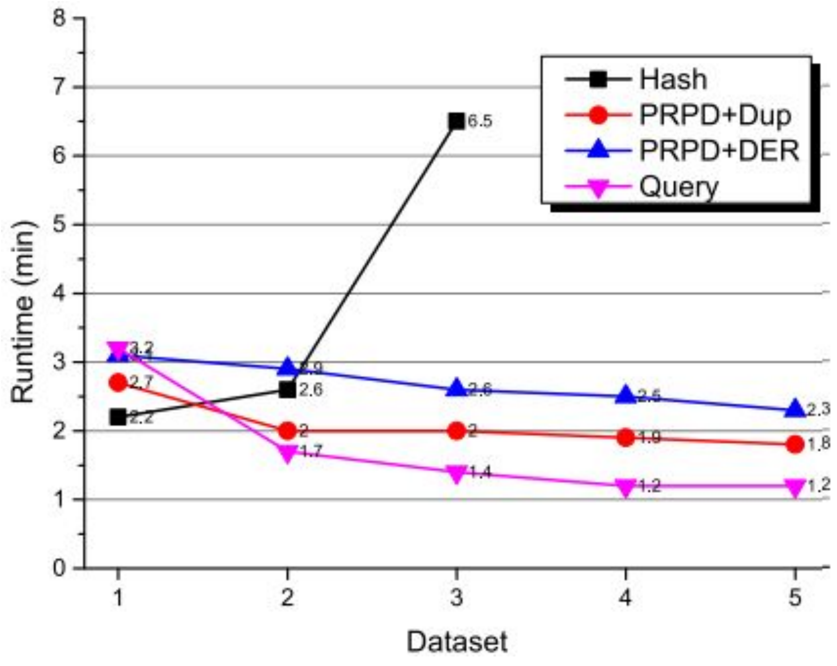


Fig. 3. Runtime when selectivity factor is 0%[1, Fig.6].

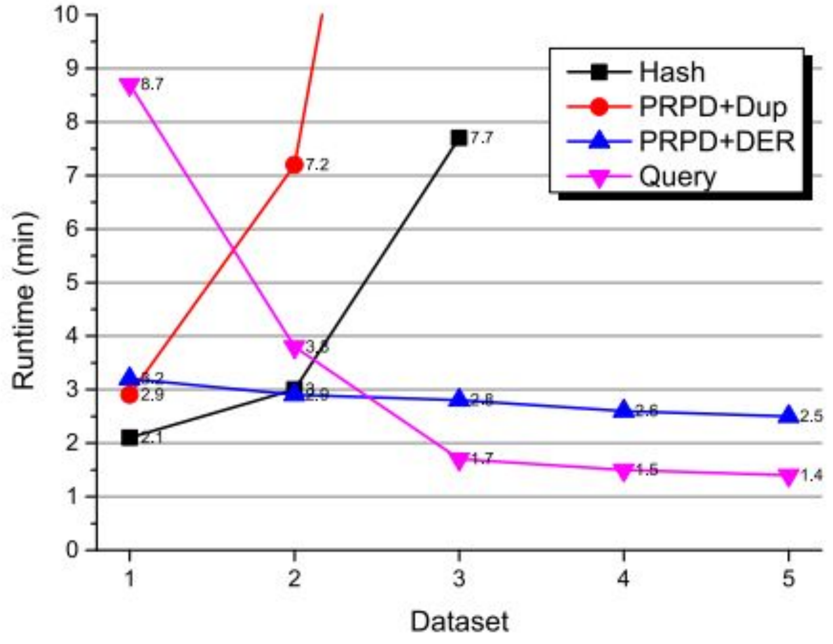


Fig. 4. Runtime when selectivity factor is 100%[1, Fig.5]

Figures 3 and 4 show that regardless of the join selectivity and skewness the runtime of the query-based algorithm will decrease as the skewness increases. However, when the join selectivity is high, the query-based algorithm will suffer significantly under no skew, this is because the algorithm has to perform an inner join with the entire table since all of the tuples will get matched in phase 3, causing an extremely large inner join input.

Analysis of Results - Conclusion

Overall, the tests performed by the researchers show that the tests do indeed support the hypothesis made by showing that:

1) The algorithm does not degrade in runtime performance as the skewness of the dataset S increases.
2) The algorithm does not inflict a heavy load on the network.

References

[1] L. Cheng and S. Kotoulas, "Efficient Skew Handling for Outer Joins in a Cloud Computing Environment," in IEEE Transactions on Cloud Computing, vol. 6, no. 2, pp. 558-571, 1 April-June 2018, doi: 10.1109/TCC.2015.2487965.

[2] M. Atre, “Left bit right: For SPARQL join queries with OPTIONAL patterns (Left-outer-joins),” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2015, pp. 1793–1808.

[3] Y. Xu and P. Kostamaa, “A new algorithm for small-large table outer joins in parallel DBMS,” in Proc. IEEE 26th Int. Conf. Data Eng., 2010, pp. 1018–1024.

[4] S. Kotoulas, E. Oren, and F. van Harmelen, “Mind the data skew: Distributed inferencing by speeddating in elastic regions,” in Proc. 19th Int. Conf. World Wide Web, 2010, pp. 531–540.

[5] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, “Parallel data processing with MapReduce: A survey,” SIGMOD Rec., vol. 40, no. 4, pp. 11–20, Jan. 2012.

[6] Y. Xu, P. Kostamaa, X. Zhou, and L. Chen, “Handling data skew in parallel joins in Shared-nothing systems,” in Proc. ACM SIG MOD Int. Conf. Manage. Data, 2008, pp. 1043– 1052.

[7] Y. Xu and P. Kostamaa, “Efficient outer join data skew handling in parallel DBMS,” Proc. VLDB Endowment, vol. 2, no. 2, pp. 1390– 1396, Aug. 2009.

[8] D. DeWitt and J. Gray, “Parallel database systems: The future of high performance database systems,” Commun. ACM, vol. 35, no. 6, pp. 85–98, Jun. 1992.