Ad-hoc star join processing in Clusters of SMP.

1Josep Aguilar-Saborit 1Victor Muntés-Mulero 2Calisto Zuzarte
1Josep-L. Larriba-Pey
1Universitat Politècnica de Catalunya, Computer Architecture Department.
Jordi Girona 1-3. Campus Nord-UPC, Modul D6, E-08034, Barcelona, Spain
{jaguiar,vmuntes,larri}@ac.upc.edu
2IBM Toronto Lab.
8200 Wardem Ave. Markham, ON L6G1C7, Toronto, Canada
{calisto}@ca.ibm.com

Abstract. Data Wharehouse workloads are becoming an important field for the support of On Line Analytical Processing. One important aspect of this field is that the amount of data queried has grown significantly in the recent years, and the tendency shows that a 100% growth per year is feasible in the near future.
The strategy to cope with large queries on such huge amounts of data calls for the use of large parallel computers. The trend today is to use Clusters of Symmetric Multiprocessors (SMP) computers that show a reasonable balance between cost and performance. In such cases, it is necessary to tune the applications in order to obtain a minimum amount of I/O, and communication, such that the global execution time is reduced as much as possible.
In this paper we look into the problem of ad-hoc star join query processing in Clusters of SMPs. We propose a new technique, the Star Hash Join (SHJ), which exploits the use of bit filters in such architectures. SHJ is a generalization of the Pushed Down Bit Filters for Clusters of SMPs. The objectives of the technique are to (i) reduce the amount of data communicated, (ii) reduce the amount of data spilled to disk during the execution of intermediate joins in the query plan, and (iii) reduce the amount of memory used by auxiliary data structures such as bit filters.
We analyze and compare our proposal with different strategies like Multi Hierarchical Clustering, Bitmap Join, Semijoin reduction and Pushed Down Bit Filters. Although each of those techniques focuses on the SMP part of the problem, reducing I/O, or the inter-cluster parallelism, reducing communication, it is difficult to find a technique that gives a benefit in all the cases. As a consequence of our analysis, we propose a novel approach that reduces both inter-cluster communication and intra-cluster I/O in a broader spectrum of cases than the previous techniques.

1 Introduction

The use of Clusters of Symmetric MultiProcessors (SMP) architectures has become a popular solution to implement massive parallelism. The percentage of cluster computers among the top500 list published Nov.2004 (see Figure 1) is
about 60% at present, showing a significant growth in the last few years. Database applications show a high demand on the use of these type of configurations: the amount of data is approaching the petabyte steadily, moving research in Data Warehousing towards many implementation challenges.

In order to balance the load, Clusters of SMPs share large volumes of data in such a way that each cluster keeps a portion of the overall database. An SMP cluster consists of a set of processors that share memory and I/O resources under the control of one copy of the operating system. All the processes access the memory by using high speed buses or advanced cross switching technologies that support point-to-point interconnections between processors. SMP configurations show scalability problems, which are solved by connecting multiple SMP clusters through a network communication system, offering high scalability and cost-effectiveness.

Clusters of SMPs speed up query processing by balancing large volumes of data across all clusters in the system, thus, each cluster can work in parallel with its own part of the overall database. The clustering schema alleviates the problem of I/O processing for large relations. Moreover, SMP processes scan in parallel the data that may be stripped across multiple disks into each cluster. However, data communication becomes important, and makes it even more relevant data management to achieve good performance.

1.1 Clusters of SMP and Data Warehousing.

Data Warehouses databases contain large amount of data from different independent sources for the support of On-Line Analytical Processing (OLAP) type
queries, that allow users to efficiently retrieve data for decision support purposes. Data Warehouse environments are organized according to the multidimensional model as one or more star schemas [10]. Each star schema consists of a very large central Fact table surrounded by multiple dimension tables that are linked to it through primary and foreign key relations. Figure 2.a shows an example of a star schema based on the TPC-H Database [1]. **Lineitem** is the central Fact table and keeps information of every line each order is compounded. The Fact table links the data stored in the four dimension tables: **orders**, **part**, **supplier** and **part_supp**.

On Line Analytical Processing (OLAP) queries are usually complex and ad-hoc with high selectivity factors. Queries are not known in advance and have a multi-table join flavor, joining the Fact table with its respective dimensions. Those queries are also called ad-hoc star join queries in the literature [9]. Different predicates are applied over the dimension tables, reducing the result of the star join that may be grouped for analytical processing. The huge amount of data to be processed from the the Fact table leads the fast processing of star join queries to be a critical issue for the Data Warehousing environments.

![Fig. 2. a.- TPC-H star schema. b.-Data partitioned across k clusters.](image)
The distribution of a database in Clusters of SMPs is done by applying a shipping function that decides the host cluster of each record based on a column or set of columns (called *partitioning key*) from the table to be partitioned [18]. Figure 2.b shows how the data belonging to the star schema in 2.a, are partitioned in an environment consisting of \( k \) cluster computers. When performing a join operation, if the joining key is the same as the partitioning key used for both input tables, then it is said that the join is *collocated*, otherwise it is *non-colllocated*. Collocated joins can be performed locally in each cluster, while non-colllocated joins need for the re-partitioning of data. In this case, it is necessary to either broadcast the smaller relation, or selectively re-partition both relations.

When executing a star join in Clusters of SMPs, the query execution plan is the same for all clusters. The main difference between parallel and sequential environments is data communication. The minimization of inter-cluster data traffic becomes a priority during star join processing because of the large volumes of data managed. Therefore, one common situation in a star schema, is that the partitioning keys would be the primary keys for the dimensions tables, and one of the foreign keys from the Fact table, like the schema shown in figure 2.b. In that example table *lineitem* has been partitioned through \( l.\text{orderkey} \), hence only the join with *orders* (\( l.\text{orderkey} = o.\text{orderkey} \)) is collocated. Any other join between the *lineitem* and another dimension is non-colllocated and needs for the re-partitioning of data. Orders is the largest dimension, thus it makes sense that \( l.\text{orderkey} \) is chosen key to partition the *lineitem*. This way the larger dimension and Fact tables is performed locally into each cluster.

### 1.2 Related Work and Motivation

Ad-hoc star join processing has been the focus of some research in the recent years. Pre-computed aggregate results, the use of indices and clustering schemes are the most relevant techniques to speed up query processing. The use of materialized views [24] in ad-hoc star join queries is very limited as it is not possible to know what are they querying in advance. BitMap Joins, introduced by O’Neil and Graefe in [20] are the basis for other research in the topic and they are based on the use of Bitmap indices [21][8]. The alternative to the use of indices is physical clustering which aims at limiting the number of I/O accesses to the Fact table required to process a query [11][17][19]. In this case, the Fact table may be created by specifying one or more key dimensions that that are used to cluster the table data. Thus, the Fact table is organized in multiple hierarchical dimensions. Then, star joins are turned into multidimensional range queries reducing the number of I/O accesses to the Fact table.

In this paper we propose a technique that saves both intra cluster data processing and inter cluster communication during ad-hoc star join processing. We call our strategy *Star Hash Join* (SHJ) and it is a generalization of Pushed Down Bit Filters (PDBF) [2] for Clusters of SMP computers. PDBF is a technique used in shared everything environments, and consists of the filtering of data in advance by pushing down the bit filters [7] to the lower nodes.
of the query execution plan. SHJ generalizes PDBF for Clusters of SMPs using Semi-join reduction [5][6].

Semi-join reduction is aimed at saving data communication during join processing for distributed configurations. Suppose two relations $R$ and $S$ that are stored in different sites. The size of relations $R$ and $S$ can be reduced by using Semi-joins. A Semi-join from $R$ to $S$ is performed by projecting $R$ on the join attributes, then the resulting projection is sent to the site of $S$ and the join can be performed. The result of this join, usually smaller than $S$, is sent to the site of $R$ to complete the join operation. Bit filters [7] can be used in the execution of Semi-joins [14][16] in order to reduce even more the size of the projection of the joining keys.

We study, simulate, analyze and compare, within a Clusters of SMPs configuration, all the techniques mentioned for ad-hoc star join processing. Our analysis shows the difficulty of finding an optimal approach that reduces both I/O and communication when star join queries are ad-hoc. We conclude in an hybrid solution that borrows ideas from the mentioned techniques and that may have a significant gain for a wide range of ad-hoc star join queries.

Organization of the paper

We start by explaining the Bitmap Join. The Bitmap Join processing sets the basis of star join query processing, and helps to understand techniques appeared afterwards that try to implement the same idea. We follow by explaining the Star Hash Join, our proposal that extends the idea of Pushed Down Bit Filters to Cluster of SMPs. In Section 4 we explaining Multi Hierarchical clustering (MHC). Then, in Section 5 we formulate the mathematical models for each of the techniques exposed in this paper, and used for our analysis. Finally in sections 6, 7 and 8, we explain the setup environment, analyze our proposal, and draw some conclusions.

2 Star Join. The Bitmap Join (BJ)

The Bitmap Join was first introduced by O’Neil and Graefe in [20]. Using Bitmap indices performs the star join through fast bitwise operation ending up in a bit vector that maps the result tuples from the Fact Table. If we did not take into account the additional overhead that indices may cause during the Bitmap Join processing, then, this strategy reduces at maximum either the I/O, or data network traffic communication during ad-hoc star join query processing. In this paper we explain this technique from a theoretical point of view, in order to understand both the path followed by other similar techniques, and which is the maximum benefit they can obtain.

In its simplest form, a Bitmap index on a table T based on a column C consists of a list of records per attribute value, where the list is represented by a Bitmap or bit vector. The size of each bit vector is $\|T\|$, and a bit position is set to one if the associated record is contained in the list represented, otherwise
is set to zero. Usually, the row identifier, known as RID, is the method used in order to map each bit position and the rows indexed. Using this idea, the join between two tables $T$ and $S$ on a common column may be represented through a Bitmap index (i.e. Bitmap Join index) sized in $\|T\| \times \|S\|$ bits; for each record of $T$, a bit vector of size $\|S\|$ is hold indicating which records of $S$ join that record.

Using these indices we may have a Star schema organized as shown in Figure 3:

- a central Fact table $F$ and $n$ dimension tables $D_i, i = 0..n-1$. Each dimension has $c = 0..m-1$ columns, where $m$ may be different for each table.
- a Bitmap index for each column $c$ for each dimension table $D_i$ (i.e. $BI_{i,c}$)
- and a Bitmap Join index for each join between the Fact table and a dimension table $D_i$ (i.e. $BJI_{i}$). All joins between $D_i$ and $F$ are indexed and sized in $\|D_i\| \times \|F\|$ bits each one.

With this Star schema, a star join query between the $n$ dimension table and the Fact table, it may be performed through fast bitwise operations. Its mechanism is shown in figure 4 and can be explained as follows:

1. Perform the selection through the dimension tables. Each $D_i$ involved in the query may have a set of one or more restrictions on each column. Bitwise operations through $BI_{i,c}$ indices are performed to obtain a bit vector $BV_i$ of size $\|D_i\|$ indicating those records of $D_i$ that qualify.
2. Each bit in $BV_i$ represents a record from $D_i$, and it has associated a bit vector of size $\|F\|$ in the $BJI_i$ index. Then, for those bits from $BV_i$ set to one, ORing is applied to the respective bit vectors contained in $BJI_i$. For each join between a dimension table $D_i$ and the Fact table, the result is a bit vector $BV_{join_i}$ of size $\|F\|$ that indicates which records from $F$ join the qualified tuples of $D_i$. 

![Fig. 3. Star schema using Bitmap Indices.](image-url)
3. Finally ANDing is performed between all $BV_{join_i}$. The result is a single bit vector that indicates the tuples from the Fact table that satisfy the whole query.

Thus, the conceptual idea of the Bitmap Join processing is to reduce the data to be processed from the Fact table, to the set of records that already satisfy the star join query. This way, the mechanism ends up creating a bit vector indexed by RID, where each single bit maps one record of the Fact table: a bit position set to one indicates that the associated record must be processed, otherwise it can be skipped. The star join is completed by joining the reduced Fact table with the dimension tables involved in the final output. The use of Join Indices [21] has been proposed in order to speed up this last step, thus either the scan over the Fact table or the join with the dimension tables are not necessary at all.

Fact tables may have billions of records of magnitude, hence the large size of the indices needed by the Bitmap Join processing difficulties the use of the technique. However, in an hypothetical situation where those indices would not suppose any overhead, then the Bitmap Join processing would reduce at maximum either the amount of data to be transmitted, or the data to be read from the Fact table. In the following sections we will explain the Star Hash Join and Multi Hierarchical clustering, techniques that try to reach the same goal in a more affordable way.

3 Star Hash Join (SHJ)

In this section we explain our proposal, the Star Hash Join. We will go through some basic concepts before introducing the algorithm.
3.1 The Hybrid Hash Join.

The Hybrid Hash Join [12] is a hash based join algorithm commonly used to perform hash join operations in DBMSs such as DB2, SQL-Server or Oracle. It consists of two phases, (1) a build phase where a hash table is built with tuples from the smaller input relation, and (2) a probe phase where tuples from the remaining larger relation are probed in the hash table looking for matches. If the hash table created during the build phase does not fit in memory, then hashing [7] is used to partition both input relations in the same way such that each partition from the smaller relation can fit in memory, and the join can be performed on each pair of corresponding partitions.

The Hybrid Hash Join may use bit filters [7] to speed up its execution. Bit filters are created during the build phase: by applying hashing, the value of the joining key for each record from the build relation is mapped into the bit filter, and its respective bit is set to 1. Then, the bit filter is checked using the same hashing function for the joining key of each record processed during the probe phase. If the corresponding bit was not set during the build phase, then the tuple can be filtered out. The main goal of this structure is to avoid spilling tuples to disk during the probe phase, saving I/O.

When performing a parallel non-collocated Hybrid Hash Join either the build or probe relations may be re-partitioned. One common situation is either to selectively repartition the probe relation, or to broadcast the build relation. We focus in the former case.

3.2 Star Hash Join Schema

A star join query with \( n \) dimension tables and a Fact table, may be performed through \( n \) Hybrid Hash Join operations as a left-deep tree-shaped query plan. Figure 5 shows the shape of a Star Hash Join schema for a system configuration with \( k \) clusters. We use the following definitions:

- \( D_{ij} \) is the part of the \( i^{th} \) dimension table stored in cluster \( j \), and \( BF_{ij} \) is the bit filter created during its build phase.
- \( Fact_j \) is the portion of the Fact table stored in the cluster \( j \).
- RO is the Re-partitioning Operator responsible for selectively sending data to the rest of computational clusters, as well as receiving data from each of those clusters.

Typically in a star schema the partitioning keys are the primary keys for the dimension tables, and one of the foreign keys for the Fact table. Hence, one join between one dimension table and the Fact table may be collocated. The rest of the joins will be non-collocated, and will need for the presence of a RO during the probe phase of the left deep tree.

3.3 The algorithm

Our approach generalizes the use of Pushed Down Bit Filters (PDBF) for a faster execution of ad-hoc star join queries in Clusters of SMPs. The Star Hash Join algorithm makes use of Semijoin reduction [6][5] to do such generalization.
The use of Semijoin reduction in Clusters of SMPs has been called Remote Bit Filters Broadcasted ($RBF_B$) in the literature [3]. $RBF_B$ broadcasts the bit filters created during the build phase locally, to all the clusters involved in the processing of a non-collocated Hybrid Hash join. Hence, before sending a record to a remote cluster during the probe phase, the record is tested against the bit filter of the target cluster, and it is discarded before sending it if possible. $RBF_B$ is graphically explained in Figure 6.a. For a given cluster $j$, a hash join between one dimension ($D_j$) and the Fact table ($Fact_j$), will keep bit filters $BF_0, BF_1, \ldots, BF_{n-1}$ in its local memory, one for each of the $n$ dimension tables.

PDBF is aimed at saving intra data processing within shared everything environments [2]. PDBF uses the bit filters generated in the upper nodes of the query plan to filter out tuples in the leaf nodes. This saves the processing of data and I/O of the intermediate results, that otherwise would be processed by the nodes in between. PDBF is graphically shown in Figure 6.b. Each record scanned from the Fact table is tested against each of the $n$ bit filters. If only one of the bit filters has the associated entry of the current record set to 0, then the given record can be discarded. PDBF cannot be used in Clusters of SMPs, because it is not possible to filter out data that has to be transmitted to remote clusters, using only the local bit filters.

**Star Hash Join (SHJ).** The strategy followed by SHJ is graphically explained in Figure 7.a. $RBF_B$ is applied in order to keep copies of all the bit filters for the star join query in every single cluster. On the other hand, PDBF allows the lower of the query execution plan to have access to all the bit filters of the query. This way, every record of the Fact table is checked against the bit filters of its target cluster, and thus, it is only transmitted if it has a potential joining record in the remote destination cluster. Thus, SHJ extends the use of PDBF to Cluster of SMPs. Moreover, it gets a significant major reduction in terms of data communication than just the Semi-join reduction executed alone.
A. Semijoin Reduction. Remote Bit Filters Broadcasted

B. Pushed Down Bit Filters.

Fig. 6. Semijoin Reduction (\(\text{RBF}_B\)), and \(\text{PDBF}\).

Fig. 7. Star Hash Join. \(\text{PDBF} + \text{RBF}_B\).

We show the algorithmic version of SHJ in Figure 7.b. Given cluster \(j\) and \(n\) joins between dimension tables \((D_i, j, i = 0..n-1)\) and the Fact table \((\text{Fact}_j)\), we name the joining keys \(\text{JK}_i\). Thus, each record being scanned from table \(\text{Fact}_j\) is processed as follows:

1. For each join in the query, we apply the shipping function used to distribute data, to the joining key \((\text{where}_\text{to}_\text{ship} (\text{JK}_i))\). This way, we figure out the target cluster of the current record. Joining keys in a star join are usually primary keys in the dimension tables, which are also the partitioning keys. Hence, the joining key \(\text{JK}_i\) stored in each record from the Fact table, is the only information needed to know where it has to be shipped.

2. Then, the record being processed is tested against the bit filter of the target node \(\text{BF}_{\text{target}}\). If it returns zero, this means that the record can be discarded, otherwise it has to be processed.
Multi Hierarchical Clustering (MHC)

Clusters of SMPs may cut big problems into small pieces, hence each piece of work can be processed in parallel by each cluster. For the specific case of star join query processing, the Fact table is clustered across the system, thus the heavy cost that supposes to process such huge amount of data is shared amongst all clusters in the system. Moreover, news paths of the Fact table organization within one physical machine have emerged lately [11][17][19]. The portion of the Fact table belonging to each cluster, may be stored by specifying one or more dimension keys which to cluster table’s data, thus the Fact table is organized respect multiple hierarchical dimensions. In Figure 8 we show an example of MHC for the star schema proposed in Figure 2.a and based on the TPC-H database. For any cluster, the Fact table, lineitem is organized through three dimension hierarchies, in this example by o_orderdate, p_brand and s_nationkey.

![Diagram of MHC](image)

Fig. 8. Multi Hierarchical Clustering.

The benefits of hierarchical clustering for star join queries was first observed in [11], where the multidimensional query space is uniformly divided into chunks with lower granularity than query level caching. In [17] surrogate keys are used to efficiently encode dimension hierarchies, and the UB-Tree multidimensional index [4], an access method for multidimensional point data, is used as primary organization of the Fact table. Then, star joins are turned into multidimensional range queries reducing the number of I/O accesses to the Fact table. In [19] a chunk-based file system specific to OLAP cubes is proposed. Again star join
queries are transformed to range queries in the multidimensional and multi-level
data space of a cube, which its data-access is provided by the storage manager.
In [13] they present a detailed and complete abstract processing plan for the
processing of ad-hoc star join queries over hierarchically clustered Fact tables.
Under a de-normalized schema they hierarchically encode the dimension tables
by applying surrogates, hence each surrogate dimension key is stored in the
Fact table. When performing a star join query, ranges are created over each
dimension’s surrogate keys that have may have a restriction. These ranges define
one or more hype-rectangles in the multidimensional space of the Fact table,
hence the number of accesses to the Fact table is highly reduced.
In order to efficiently access to the hierarchically encoded Fact table, dif-
f erent clustering schemes are proposed in the above mentioned previous works
[11][17][19]. DB2 Universal Database has its own data layout schema, and it is
based in a block oriented organization of the clustered tables [22][23]. An unique
combination of dimension values is physically organized as block of pages, being
a block a set of consecutive pages on disk. Block indices are created to access
these blocks, and access methods are extended for a fast retrieving of the data.
Multi Hierarchical Clustering schemes are good for star joins queries when the
selectivity of the whole query is relatively small, thus the cost of the query resides
in reading the Fact table. However it has one stunning block: the attributes
used to multi-dimensionally cluster the Fact table have to be the same as the
attributes that restrict the dimension tables in the ad-hoc star join query. Thus
there is a high probability that only few dimensions of the Fact table coincide
with the restriction attributes.

5 Mathematical model

We formulate a mathematical model exclusively aimed to analyze the three tech-
niques explained across the paper: Bitmap Join (BJ), Star Hash Join (SHJ)
and Multi Hierarchical Clustering (MHC). We do not model Semi-join reduction
(RBF), as we can find a detailed analytical model of this technique for Cluster
computers in [3]. On the other hand PDBF is not a technique we can compare
with the other as it is use in not aimed for Cluster configurations, moreover its
use is included in the SHJ algorithm.
The analytical model is formulated under the conditions of a Star Hash Join
schema as explained in section 3. We do it that way for simplicity, and conclusions
could be easily generalized to any kind of query execution plan for star join
query processing. The mathematical model we present assumes uniform data ;
no correlation between values of different attributes; and joining keys are unique
values in their respective dimension tables.
As seen in section 4 there are several ways to organize clustered data. The
model also assumes that the data layout schema used by MHC to multi-dimensionally
organize the Fact table is optimal and has a negligible overhead over I/O and
memory resources. For the Bitmap Join, we just model I/O and network com-
unication without taking into account the overhead the indices would cause.
We do that because with the BJ technique we just want to give the maximum benefit we could achieve during star join query processing. Indices used in the Bitmap Join, as explained in section 2, could be larger than the database itself.

We will start explaining the basic mathematical concepts related to bit filters. Then we will follow modelling data traffic and I/O, finishing with the analysis of memory resources.

**Bit Filters**

Concepts and formulae we explain in this section resorts on the study performed by B.Bloom in [7]. A bit filter is a Bitmap structure defined by the following parameters:

- $M_{bf}$ size in bytes of the bit filter.
- $n$ number of distinct values mapped into the bit filter. In our case, the number of distinct values coming up from the build relation.
- $d$ number of distinct bits set to one for each upcoming value. By default we take $d=1$.
- $F_p$ fraction of false positives given by the bit filter. $F_p$ depends on the previous parameters and is defined as follows:

$$F_p = (1 - (1 - \frac{1}{M_{bf}})^{dn})^d$$  \hspace{1cm} (1)

We define as $S_{bf}$, the selectivity of a bit filter. By $S_{bf}$ we understand the fraction of records from the probe relation that are not filtered out by the bit filter. In our case, values coming up from the build relations, and that will be mapped into the bit filter are assumed to be unique. Thus $S_{bf}$ is directly related to $F_p$ and to the selectivity of the build relation $S_R$ used to create the bit filter. Therefore we define the selectivity of a bit filter as:

$$S_{bf} = S_R + (1 - S_R)F_p$$  \hspace{1cm} (2)

The perfect value for $S_{bf}$ would be the one that match with $S_R$ meaning that there are no false positives. This situation only happens in the case of the Bitmap Join indices explained in 2 due to the one to one mapping between records and bit positions. Nevertheless, when using bit filters, due to the presence of false positives, we have to add to $S_R$ the fraction of records from the probe relation that cross the bit filter and won’t join with any record of the build relation : $(1 - S_R)F_p$.

Given the assumptions of the mathematical model, bit filters created during the SHJ processing, in any cluster $j$ from values of the $i^{th}$ dimension table : $D_i$, say $BF_{i_1}, BF_{i_2}, \ldots, BF_{i_{m-1}}$, they have the same selectivity factor ($S_{bf}$) each one $S_{bf_{i_1}} = S_{bf_{i_2}} = \ldots = S_{bf_{i_{m-2}}} = S_{bf_{i_{m-1}}}$.

**Data traffic.**

In order to model data network traffic communication and I/O, we need to figure out the output cardinalities of each affected node in the query execution plan.
after applying any of the techniques. As a consequence of its data reduction effect over the Fact table, each technique will vary in a different way the output cardinalities from the join nodes and the scan node from the Fact table. We give the following definitions:

- $OC_{ij}$ original output cardinality of the $i^{th}$ join between a dimension and the Fact table in a cluster $j$: $F_j \bowtie D_{ij}$.
- $S_{D_{ij}}$ selectivity of the $i^{th}$ dimension table in any cluster $j$.
- $NOC_{ij}$ new output cardinality for the join $D_{ij} \bowtie F_j$.
- $NOC_{Fj}$ new output cardinality from the scan of the Fact table.

We calculate $NOC_{ij}$ and $NOC_{Fj}$ differently for each technique:

**Bitmap Join** The Bitmap Join only needs information related to the selectivity of the dimension tables. Based on that we calculate its output cardinalities as follows:

\[
NOC_{Fj} = \|F_j\| \prod_{i=0..n-1} S_{D_{ij}} \\
NOC_{ij} = OC_{ij} \times \prod_{j=i..n-1} S_{D_{ij}}
\]  

**MHC** We define by $R$ the set of dimensions which respective restriction attributes coincide with the hierarchical attributes applied to multi-dimensionally cluster the Fact table into each cluster. Then we define new output cardinalities for MHC as follows:

\[
NOC_{Fj} = \|F_j\| \prod_{i \in R} S_{D_{ij}} \\
NOC_{ij} = OC_{ij} \times \prod_{j=i..n-1} S_{D_{ij}}
\]  

**SHJ** This technique depends on the the selectivity factor of the bit filter created from each dimension table in any cluster $j$. We formulae the new output cardinalities as follows:

\[
NOC_{Fj} = \|F_j\| \prod_{i=0..n-1} S_{bf_{ij}} \\
NOC_{ij} = OC_{ij} \times \prod_{j=i..n-1} S_{bf_{jj}}
\]
Network and I/O.

We start by giving some definitions:

- $M$ memory pages available for a hash join.
- $k$ number of Clusters in the system.
- $\text{rec}_F$ size in bytes of a record stored in the Fact table $\frac{|F|}{|D_i|}$.
- $\text{rec}_\text{scan}$ size in bytes of a record projected from the Fact table scan.
- $\text{rec}_\text{join}$ size in bytes of a record result of the join $F_j \bowtie D_i$.
- $\text{SMP}_p$ SMP degree in each cluster.
- $\lambda$ fudge factor added in order to take into account the extra room needed to manage data from the build relation.
- $\nu$ fudge factor added in order to take into account the extra room needed to manage a single record.

We quantify in bytes the data traffic network communication produced by each technique as follows:

$$\sum_{i=1..n-1} NOC_{ij} \times \frac{k-1}{k} \times \text{rec}_\text{join} \nu \quad (9)$$

Besides the join with the first dimension, all hash joins add data network communication. The ratio of the data being communicated coming up from its output cardinality is: $\frac{k-1}{k}$.

When calculating the I/O saved by each technique, we might separate the I/O saved from the Fact table scan, and the I/O saved during join processing.

- Fact table. $\text{SHJ}$ does not affect to the I/O when scanning the Fact table, data are filtered out once it is read. On the other hand MHC and the Bitmap Join do save I/O processing over the Fact table. The number of records read by both techniques is the same as the $NOC_F$ calculated before. Thus we calculate in bytes the amount of I/O produced by any cluster $j$ when reading the Fact table as:

$$I/O_{\text{Fact}} = NOC_F \times \text{rec}_F \nu \quad (10)$$

Taking into account that Clusters have an SMP configuration, and assuming that data are stripped across several parallel disks, then the scan is parallelized according to the SMP degree: $\frac{I/O_{\text{Fact}}}{\text{SMP}_p}$.

- Join processing. The I/O saved during hash join processing depends on the amount of data from the build relation spilled to disk. We name $\mu_i$ the fraction of the $i^{\text{th}}$ dimension table being spilled to disk during the hash join processing. $\mu_i$ is calculated as:

$$\left\{ \begin{array}{ll} 1 - \frac{|M| - \text{SMP}_p}{|D_i| \times S D_i \lambda}, & |D_i \times S D_i \lambda| > |M|; \\ 0, & \text{otherwise}; \end{array} \right.$$  

Taking the model of the parallel Hybrid Hash Join proposed in [15], each SMP process writes in a different page of memory to avoid contention:
\(|M| - SMP_p\). Then, the pages of memory occupied by the selected data from the dimension table are calculated as: \(|D_{ij} \times S_{D_j}| \lambda\). Then we calculate the amount of bytes read and written by each cluster during join processing as:

\[
\text{I/O join} = 2 \times |D_{j0}| \mu_0 + \text{NOC}_{F_j}\mu_0(\text{rec.size}_\text{scan}v) + \\
\sum_{i=1..n-1} |D_{ij}| \mu_i + (\text{NOC}_{ij}\mu_i) \times (\text{rec.size}_\text{join}v)
\] (11)

\(|D_{ij}| \mu_i\) for \(i = 0..n-1\) is the I/O produced from the spilled portion of the dimension table for each hash join. \(\text{NOC}_{F_j}\mu_0\) is the number of records from the Fact table spilled to disk during the first collocated hash join processing. \(\text{NOC}_{ij}\mu_i\) is the number of records from the output joins cardinalities spilled to disk. As happened with the I/O from the Fact table, having an SMP configuration for each cluster, then processes can write and read in parallel partitions spilled to disk. Thus the I/O is parallelized into each cluster: \(\text{I/O join}_{SMP_p}\).

**Memory resources for SHJ.**

All non-collocated hash joins need to store the bit filters of the remote computational nodes involved in the parallel join. Those executed locally, they just need room for the bit filter originally created by the hash join. Being \(Mbf_i\) the amount of memory needed by the \(i^{th}\) hash join for any cluster \(j\), then the amount of memory needed by the whole system can be calculated as:

\[
M = Mbf_0 + \left[ \sum_{i=1..n-1} (k - 1) \times (Mbf_{ji}) \right]
\] (12)

As shown in equation 1, the size of the bit filters \(Mbf\) is highly dependant to the factor of false positives. We can see this relation if we isolate \(Mbf\) from equation 1:

\[
Mbf = \frac{1}{(1 - ((1 - F_p^{1/d_j})^{1/nd}))}
\] (13)

Figure 9 illustrates the relation between the fraction of false positives, and the memory and selectivity of a bit filter. We can appreciate that while \(Sbf\) is directly proportional to \(F_p\), \(Mbf\) gets highly modified for small fractions of false positives, ie. with \(F_p = 0.1\), in comparison with the starting value 0.05, we are increasing the selectivity in a 10% and reducing the amount of memory needed for the bit filter in a 50%. Thus, in case of memory starvation we could highly reduce memory resources used by SHJ at a low cost of false positives.

6 Evaluation set up

We perform our evaluation analyzing a similar environment as those used for the TPC-H benchmark [1]. We simulate and analyze the execution of one TPC-H
like query over five Clusters of SMPs, each server with a 32 processors sharing 256GB of main memory and 512 disks. The same configuration has been used by IBM in a 10TB TPC-H benchmark [1].

```sql
select nation, o_year, sum(amount) as sum_profit from
(select s_name as nation,
year(o_orderdate) as o_year,
l_extendedprice * (1 - l_discount) - l_supplycost*
l_quantity as amount
from tpcd.part, tpcd.supplier,
       tpcd.lineitem, tpcd.partsupp,
       tpcd.orders, tpcd.nation
where s_suppkey = l_suppkey
and ps_suppkey = l_suppkey
and p_partkey = l_partkey
and l_orderkey = l_orderkey
and ps_partkey = ps_suppkey
and p_partkey = l_partkey
and n_nationkey > y
and o_orderpriority = 'a'
and ps_availqty > w ) as profit
group by nation, o_year)
order by nation, o_year desc)
```

<table>
<thead>
<tr>
<th>Attr</th>
<th>MHC-4D</th>
<th>MHC-1D</th>
<th>MHC-2D</th>
</tr>
</thead>
<tbody>
<tr>
<td>o_orderpriority</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>p_name</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>s_nationkey</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Fig. 9. Bit filters. Relation between Mbf, Sbf and \( F_p \).

Fig. 10. Query. a) Execution plan. b) Set of selectivities being used. c) MHC by different dimension attributes.
We assume a 10TB TPC-H populated database, partitioned across the system with hash partitioning. For our analysis we have used a query based on TPC-H query 9 (see Figure 10). Figure 10 shows the execution plan of the part of the query that executes the star join, which has the shape of the Star Hash Join schema explained in this paper.

We assume a 10TB TPC-H populated database, partitioned across the system with hash partitioning. For our analysis we have used a query based on TPC-H query 9 (see Figure 10). This query, is a model of a star join query where the Fact table, `lineitem` is joined with 4 dimension, `partsupp`, `part`, `supplier` and `orders`. The dimension table `supplier` is also joined with `nation`. The original query had only restriction in `part`, thus, we have modified it adding three conditions in `nation`, `orders` and `partsupp` in order to restrict the query to a smaller set of data.

Figure 10.a shows the execution plan of the part of the query that executes the star join, which has the shape of the Star Hash Join schema explained in this paper. Following the example shown in Figure 2, dimension tables have been partitioned through the primary key, and `lineitem` through `lorderkey`. Thus, only the join between `lineitem` and `orders` is collocated (hash join 0). Hash Join 1, 2 and 3 need for repartitioning of data in order to be executed. The memory available to each hash join is 1.3 GBytes, and the SMP degree is 32, meaning that the 32 processors will work in parallel sharing resources within each cluster.

The selectivity of the dimension tables will vary depending on the values $x, y, z$ and $w$. Figure 10.b shows the different sets of selectivities applied to the dimension tables that we use in order to analyze a wide range of situations for the different techniques being compared. Bit filters are used in all hash joins no matter the selectivity of the build relation. The default fraction of false positives [7]($F_p$) for any bit filter is set to $F_p = 0.05$. TPC-H fulfills the assumptions of the mathematical model, thus, original and new output cardinalities ($OC_i$ and $NOC_i$) are all calculated through the cardinalities of a 10TB TPC-H benchmark and the selectivities of the dimension tables.

7 Analysis

We will perform a detailed analysis of all techniques explained through the paper: Bitmap Join (BJ), Multi-hierarchical clustering (MHC), Semi-join reduction (RBF_B), Star Hash Join (SHJ). In the case of MHC, we will analyze its behavior depending on the attributes used to multi-dimensionally cluster the Fact table table. Figure 10.c shows the three different cases we analyze in this paper. MHC-4D, MHC-1D, MHC-2D, combined with the 6 sets of selectivities shown in Figure 10.b, cover a wide range of possibilities that fulfill our analysis.

Following, we show results of the benefit obtained by the techniques compared to the baseline strategy unless otherwise specified. Also, the plots show in their horizontal axis results for sets 1 to 6 shown in Figure 10.b.
Communication and I/O-join processing

Figure 11 shows the percentage of data communication reduction and the I/O reduction during the join processing by each technique. Trends in both plots are quite similar:

- Bitmap Join (BJ) and MHC-4D always gets the best results, showing benefits over 90% for data traffic, and over 40% in I/O during join processing, when selectivities placed on non collocated dimensions are low, like in sets 1 and 3. MHC-4D will always get the same results as BJ as far as the attributes that restrict the dimensions are the same as the ones used to multi-dimensionally cluster the Fact table.

- SHJ gets almost the same results as the Bitmap Join and MHC-4D techniques. The slightly difference comes from the fraction of false positives provoked by the use of bit filters. Also it is important to see that SHJ always reduces more inter-cluster communication than Semi-join reduction ($RBFB$): 65% more in the best case and 25% in the worst case.

- When the Fact table has not been multi-dimensionally clustered through the attributes used in the star join query to restrict dimensions, then MHC does not have so much benefit and strongly depends on these attributes. For instance, in the case of MHC-D1, as only coincides the attribute from the dimension that is collocated, then, no inter-cluster communication is saved. Also, MHC-D2, in sets 1, 4 and 5, gets small percentages of gain either in I/O during join processing or data network communication. In these sets, the low selectivities are placed on dimensions attributes from relations $suppliers$ and $ps$,$p$,$sup$, which have not been used in MHC-2D to cluster the Fact table. Thus, as most of the reduction comes from restrictions on these dimensions, MHC-2D gets low benefit.

- Best improvements come when selectivities are low in the upper non-collocate dimensions, like in 1, 3 and 5 sets of selectivities. That’s normal as we are reducing in advance more data from the Fact table. On the other hand in set 6 we get the less improvement for any technique: this happens due to
the fact that the lowest selectivity is placed only in the collocated dimension table, orders. Hence, most of the records from the Fact table are purged at the very beginning, being the star join processing less costly even in the baseline execution.

Fact table I/O processing

Figure 12 shows the amount of I/O reduced when scanning the Fact table. Bitmap Join and MHC-4D reduce the maximum possible. Also an important fact is that MHC-1D and MHC-2D get a good average reduction specially when the selectivity of the first and second dimensions are low, like in sets 2, 3, 4 and 6. On the other hand, SHJ and $RBF_B$ purge data from the Fact table once it has been scanned, hence they do not affect the I/O from the Fact table.

![I/O Fact Table reduction](image)

**Fig. 12.** I/O reduction from the Fact table.

Trade off. Communication vs. I/O

Figure 13 shows in TBytes, the amount of communication and I/O during the star join processing. We do not show results for Semi-join reduction ($RBF_B$): the technique is already included in the Star Hash Join (SHJ), and our analysis has clearly shown that $RBF_B$ is always outperformed by SHJ. The most important points we can extract from plots in Figure 13 are:

- In Clusters of SMP, communication does not scale in the same way as I/O. In all the cases we get 20 times more network communication than I/O.
Figure 14 shows, for the set of selectivities number 1, how the I/O scales as more clusters of the same type are added. While I/O is highly reduced, communication, being the network one single resource shared by all clusters will remain the same.

- If the Bitmap Join is not feasible due to the vast use of indices, then SHJ is the technique to choose when communication is a problem. SHJ gets quite the same results as the Bitmap Join. On the other hand, MHC only gets good results when the Fact table has been multi-dimensionally clustered through the attributes that restrict dimension tables in the star join query (MHC-4D). Thus, in terms of communication savings for add-hoc processing, MHC is not the best choice.

- For I/O processing it does not happen the same as we have seen with the communication. The heavy part from I/O processing comes from reading the Fact table, hence although SHJ reduces I/O during join processing, the overall I/O processing we get with SHJ is high when compared with the Bitmap Join and MHC. Thus, the alternative to the Bitmap Join would be in this case the MHC technique. Which as we saw in Figure 12, reduces I/O in a wide spectrum of possible queries.

**Memory resources**

Figure 15.a shows the amount of memory in MBytes required by SHJ for each set of selectivities. We can see that SHJ needs, at most, 1.8 GBytes of main memory to keep all bit filters of the system with a fraction of false positives $F_p = 0.05$. If SHJ was limited by the amount of memory, and as shown in Figure 9 (section 5), then a larger fraction of false positives would allow to fit the necessary bit filters in the local memory, at the cost of slightly increasing the number of false positives in a controlled manner. Set number 3 is always the worst case because orders and ps_partsupp, which are the larger dimension tables, have high selectivities, hence the size of the bit filters created during the
respective build phases, are large due the presence of a high number of incoming distinct values.

If we compared SHJ with the Bitmap Join method, we have to realize that the amount of memory needed is far smaller. For a 10TBytes TPC-H database, the minimum amount of memory that the BJ would need, a bit vector of size the cardinality of the Fact table, is 7Gbytes of main memory. And the smaller Bitmap Join index, that is the one between lineitem and suppliers, is 70TBytes in size. Seven times larger than the database itself.
8 Conclusions and hybrid solution

In Table 1 we briefly describe the main characteristics of all techniques analyzed across the paper. The Bitmap Join is the best theoretical way to implement star joins, however it requires a massive use of indices, which are costly to maintain and large in size, leading to the use of other techniques that try to implement the same idea in a different way.

We have proposed the Star Hash Join approach (SHJ), that consists of the combination of Semi-join reduction ($RBF_B$) and Push Down Bit Filters (PDBF). Our analysis shows that SHJ is the best alternative to the BJ approach to avoid network data communication for ad-hoc star join query processing. Moreover is not aggressive in the use of memory, and it does not require the use of indices at all. On the other hand, Multi Hierarchical Clustering (MHC), which does not take good results in terms of communication for ad-hoc star join processing, it has a significant impact in the amount of I/O saved in a wide range of queries.

Therefore, we end up having SHJ, which is good for communication, and MHC, which is good for I/O. Hence, we conclude by proposing an hybrid solution that would guide us to a better solution having a good balance between I/O and communication savings. We show the behavior of the hybrid solution between Star Hash Join and Multi Hierarchical Clustering in Figure 16. In Figure 16.a we see that the problem shown by MHC for ad-hoc processing in terms of communication, it is solved in all cases thanks to the use SHJ. Also, in Figure 16.b, I/O we can appreciate that is highly reduced in a 50% of cases in our evaluation.

<table>
<thead>
<tr>
<th>Clusters?</th>
<th>BJ</th>
<th>MHC</th>
<th>SHJ</th>
<th>PDBF</th>
<th>$RBF_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comm. reduction</td>
<td>high</td>
<td>low</td>
<td>high</td>
<td>none</td>
<td>med.</td>
</tr>
<tr>
<td>I/O Fact Table reduction</td>
<td>high</td>
<td>high</td>
<td>low</td>
<td>low</td>
<td>none</td>
</tr>
<tr>
<td>I/O join processing reduction</td>
<td>high</td>
<td>low</td>
<td>high</td>
<td>high</td>
<td>none</td>
</tr>
<tr>
<td>Use of indices</td>
<td>high</td>
<td>high</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>Mem. usage</td>
<td>high</td>
<td>low</td>
<td>med.</td>
<td>low</td>
<td>med.</td>
</tr>
</tbody>
</table>

Table 1. Main characteristics of each technique.
Inter-Cluster network data communication - 10 TB TPC-H

Selectivity sets

Data Traffic (TBytes)

SHJ + MHC-1D
SHJ + MHC-2D
BJ
Baseline

I/O per SMP process - 10 TB TPC-H

Selectivity sets

I/O (TBytes)

SHJ + MHC-1D
SHJ + MHC-2D
BJ
Baseline

Fig. 16. a) IO hybrid use. b) Network hybrid use.

References