# Materializing Distributed Skyline Queries

Samiha Brahimi[1] and Mohamed-khireddine Kholladi[2]
[1]*Difa Departement, University of Constantine, Constantine, Algeria*
[2]*University of L'oued, El Qued, Algeria*

Keywords: Skyline Queries, Distributed Skyline, Skycube, Structured Peer-to-Peer Systems, Top-sky, Query Optimization.

Abstract: In this paper, we tackle the problem of efficient skycube computation in structured P2P systems. We introduce a top-down algorithm called Distributed-Top-Sky based on the recently introduced Top-Sky. Furthermore, we introduce two types of nodes namely the Scheduling-Node where the network is organized by assigning the computation of each cuboid to a Data-Node and the Data-Node which holds a part of the dataset used to compute the assigned cuboids. In order to evaluate the effectiveness of our approach, we have conducted extensive experiments on three real datasets over a simulated CAN (content addressable network) network.

## 1 INTRODUCTION

Since nowadays data are increasingly stored and managed in a distributed way, processing skyline queries (Börzsönyi et al, 2001) over distributed data has attracted much attention considering the high cost they require. On the other hand, P2P systems are one of the most attractive distributed contexts since they are highly distributed which increases the applicability of their approaches on other distributed systems.

In fact, distributed skyline queries can be optimized by adopting the optimization technique developed for relational queries: *the materialization of views*. The latter aims at saving the results of executing a query in a table in order to avoid recomputing them when needed. The views materialization process has two steps; the first one is done over expressing the relations between queries using AND-OR view graphs, syntactical analysis of the workload, data cube lattice or query rewriting. The aim of the second step is to exploit the results of the first step in order to choose the best views to be materialized using heuristics and meta-heuristics.

The interest of this paper is within the first step of the materialization technique; in particular, the computation of the skycube which has been first inspired from the well-known *data Cube* proposed in (Yuan et al., 2005 and (Pei et al., 2005) and introduced as being the result of computing all possible combinations of a *d*-dimensional set.

In a distributed setting, it is not feasible to build the skycube by computing each cuboid individually using the algorithms proposed for the processing of skyline queries since they produce a high cost. Hence, computing the skycube by deriving the cuboids from each other is required in order to reduce cost.

Consequently, we aim in this work at computing the skycube over a structured *P2P* system by introducing a top-down method called *Distributed-Top-Sky* based on the recently introduced *Top-Sky* algorithm (Brahimi et al. 2013). This choice has been motivated by the following points.

- The only a few works treating the problem of skycube computation over distributed contexts in general and P2P systems in particular.
- The main advantage of structured P2P approach is the simplicity of query routing by applying the rule according to which the data have been distributed in the first place since they are built in a controlled manner and impose a relation between peer content and network topology.
- There is some homogeneity between the *Top-Sky* algorithm and the structured P2P approach; the former relies on the values of some points of the skyline to find the others, while the latter stores each point in the node with the limits of its region including the values of the point considering all the dimensions.

Brahimi S. and Kholladi M..
Materializing Distributed Skyline Queries.
DOI: 10.5220/0004794203720379
In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence* (ICAART-2014), pages 372-379
ISBN: 978-989-758-015-4

To sum up, we believe that this paper has the following contributions:

- *Distributed-Top-Sky:* a method for computing the skycube in a top down manner i.e. it enumerates the skycube starting by the highest level (the full space cuboid) and then it begins the computation of the other cuboids each from one of its ancestors.
- A scheduling method aims to assign the computation of the cuboids to the data nodes based on the data distribution in the network, the availability of the data required for the computation of each cuboid in nodes and the quality of data.
- A system design composed of two types of nodes; scheduling node (SN) and data node (DN).
- An extensive evaluation of the proposed methods on three real data sets.

The remainder of the paper is organized as follows. Section 2 gives background information about the skyline and skycube notions. Section 3 gives a detailed description of the proposed method. Section 4 provides the experimental results. Finally, section 5 concludes the paper.

## 2 BACKGROUND

In this section, we present the most important background concepts related to our work.

### 2.1 Skyline and Distributed Skyline

The skyline operator (Börzsönyi et al, 2001) is an extension of the SQL's SELECT statement by an optional SKYLINE OF.

Formally, Given a data space D defined by a set of $d$ dimensions $\{a_1,..., a_d\}$ and a dataset S on D with cardinality $n$, a point $p \in$ S can be represented by: $p=\{p(a_1),\ldots, p(a_d)\}$, where $p(a_i)$ is a value of $p$ on dimension $a_i$.

1) A point $p$ is said to dominate another point $q$ within a subspace $U$, denoted as $p_U \prec q_U$, if (1) on every dimension $a_i \in U$, $p(a_i) \leq q(a_i)$; and (2) on at least one dimension $a_j \in U$, $p(a_j) < q(a_j)$.
2) The skyline is a set of points $SKY(S) \subset$S which are not dominated by any other point.
3) The subspace skyline is a set of points in $S$ which are not dominated by any other point with regard to a subspace dimension set $U$, we denote $SKY_U(S)$.

Computing skyline queries in a distributed setting efficiently is a challenging task since we have to compute the correct and the full skyline from different sources in a minimal execution time.

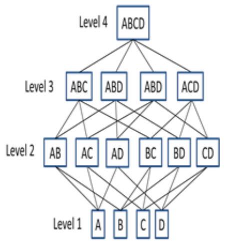
DSL (Wu et al., 2006) is the first work proposed over a structured P2P system where the problem of running parallel constrained skyline queries is dealt with. DSL uses the *CAN* (Ratnasamy et al., 2001) overlay to map data to regions and assign these regions to peers. Later on, (Wang et al., 2007) presented the SSP for distributed processing of skyline queries in BATON networks proposed by (Jagadish et al.,2005). A bit later, (Wang et al., 2009) generalized by SSP skyframe. Subsequently, other works have been conducted over structured P2P systems such as the works in (Chen et al,. 2008; 2009; Li et al,.2006).

### 2.2 t!he Skycube

This section must be in one column. The notion of skycube has been first introduced in (Yuan et al., 2005 and (Pei et al., 2005) where the authors inspired the concept from the well-known data cube. Over a set of dimensions $D$, there are $2^{d-1}$ possible skyline queries on the different dimension sets. The set of all possible skyline query results are the skycube.

| | A | B | C | D |
|---|---|---|---|---|
| p1 | 1 | 4 | 3 | 3 |
| p2 | 4 | 3 | 2 | 5 |
| p3 | 2 | 7 | 3 | 6 |
| p4 | 2 | 2 | 8 | 1 |
| p5 | 1 | 4 | 1 | 4 |
| p6 | 7 | 8 | 3 | 6 |
| p7 | 8 | 3 | 1 | 8 |
| p8 | 7 | 2 | 1 | 8 |

a. Data set



b. The lattice structure of the skycube

| Cuboids | skyline |
|---|---|
| ABCD | p1, p2 ,p4, p5, p8 |
| ABC | p4, p5, p8 |
| ABD | p1, p4, p8 |
| ACD | p2, p4, p5 |
| BCD | p1, p2, p4, p5, p8 |
| AB | p1, p4, p5 |
| AC | p5 |
| AD | p1, p4 |
| BC | p8 |
| BD | p4 |
| CD | p4, p5 |
| A | p1, p5 |
| B | p4, p8 |
| C | p5, p7, p8 |
| D | p4 |

c. The skycube

Figure1: Example of a building a skycube

The computation of the skycube is an active research sub-topic in the field of skyline queries optimization. (Raïssi et al., 2010) proposed an algorithm called *Orion* to compute the skycube in a bottom-up manner. A distributed version of this algorithm has been proposed in in (Veloso et al., 2011). Recently, a top-down algorithm called *top-sky* has been introduced in (Brahimi et al. 2013), where the authors introduced two types of subspaces, *distinct subspaces* computed directly from one of the parents and *non-distinct*, in which a part of their skyline is derived from the parents producing an intermediate skyline used to complete the cuboid by finding similar points to those it includes.

## 3 DISTRIBUTED-TOP-SKY

In this section, we present the system's structure, illustrated in Figure 2. Then, we explain the work of the two types of nodes, composing the system; the Scheduling Node (SN) and the Data Node (DN).

As seen in the figure, our system is composed of one scheduling node responsible of managing the work of the data nodes and *N* data nodes whose role is to compute the cuboids following the orders of the scheduling node.

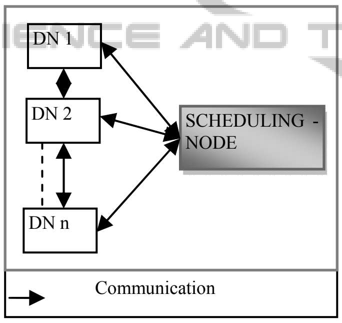


Figure 2: The network structure.

### 3.1 Scheduling-node

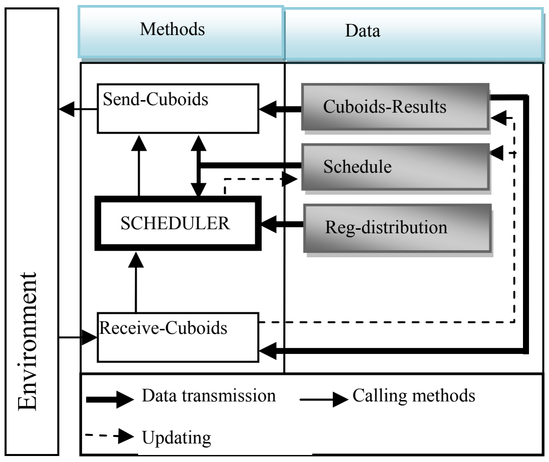


Figure 3: Structure of the Scheduling Node.

The scheduling node (SN) is considered as the heart of the system since there is only one machine which manages the work of the other nodes (data nodes). The SN has three methods exploiting and updating three types of data. Figure 3 depicts the structure of the scheduling node.

#### 3.1.1 Data

**A. Cuboids-Results**

The *Cuboids-Result* is a file where the result of computing each cuboid (the skycube) is stored. At the very beginning the file contains only the scheme illustrating the relations between cuboids (lattice structure of the skycube). Hence, the skyline objects are assigned to each cuboid once they are received from the Data nodes.

**B. Reg-Distribution table**

Since we are working on structured peer-to-peer systems, data are divided on *N* regions assigning each to a specific data node. An example of assigning regions of a bi-dimensional set to a CAN network composed of seven Data nodes is depicted in Figure.4. In the figure, the X-axis represents the first dimension (A) and the Y-axis represents the second (B).

Inspired by the distributed hash table, *Reg_Distribution* is a table having as lines the data nodes and as columns the considered dimensions. The intersection represents the interval [min, max] in which the values of the corresponding dimension in the corresponding node are included. Table 2 gives an example of the *Reg_Distribution* table.

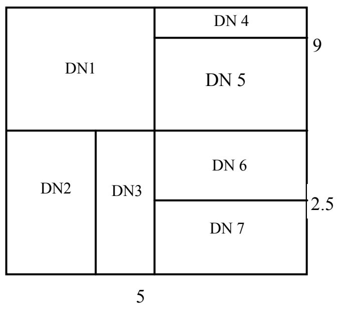


Figure 4: Example of CAN network.

Table 1: An example of Reg-distribution table.

| DATA NODE | A | B |
|---|---|---|
| DN1 | [0,5] | [5,10] |
| DN2 | [0,3] | [0,5] |
| DN3 | [3,5] | [0,5] |
| DN4 | [5,10] | [9,10] |
| DN5 | [5,10] | [5,9] |
| DN6 | [5,10] | [2.5,5] |
| DN7 | [5,10] | [0,2.5] |

### C. SCHEDULE

*Schedule* is a table having as lines the data nodes and as columns all the subspaces (cuboids), the intersection is a binary space obeying the following rules.

- If the space is empty (null), the cuboid is not assigned to the corresponding node ;
- If the space is set to 0, the cuboid is assigned to the corresponding node but not yet computed;
- If the space is set to 1; the cuboid is assigned to the corresponding node and the computation is fulfilled or the data node has received the corresponding cuboid by the Scheduling node in order to perform another computation.

A subspace must be assigned to only one data node, whereas, a data node can receive more than one subspace. It is obvious that *Schedule* is empty at the very beginning since no cuboid is yet assigned.

Table 2 is the schedule built according to the *Reg-Distribution* shown in Table 1, the cuboid AB is assigned to DN1 whereas the computation is accomplished by DN7, the cuboid B is assigned to the node DN3 but not yet computed, and the cuboid A is not yet assigned to any data node. Since AB is required to compute B, it is sent to DN3 by SN, the reason behind setting the intersection [DN3, AB] to 1 even though AB has not been computed by DN3.

Table 2: Example of a schedule.

| | AB | B | A |
|---|---|---|---|
| DN 1 | 0 | | |
| DN 2 | | | |
| DN 3 | 1 | 0 | |
| DN 4 | | | |
| DN 5 | | | |
| DN 6 | | | |
| DN 7 | 1 | | |

### 3.1.2 Methods

The scheduling node has three methods; each one is deactivated automatically when no data is available. It is activated by another method once the latter detects the availability of the required data.

### A. SCHEDULER

*Scheduler* is the method responsible of assigning each cuboid to the appropriate Data node. The assigning is based on three priority principles sorted as follows.

1. The first priority is for the less loaded node; if the Data node has no work, its coefficient DN-coef (see Algorithm 1) is incremented by 4.
2. The second priority is given to the Data node that holds one of the parent cuboids; so, we save the communication cost of sending it. The coefficient of such node is incremented by 2.
3. The third priority is for the node containing the best points i.e. the best values in each dimension. DN2 is the best node in our running example. The coefficient of such a node is incremented by 1.

*Sheduler* is the start point of the system; it is automatically deactivated when no parent is available. Also, when new parents are computed, *Sheduler* is activated By *Receive-Cuboids*. Algorithm 1 is a pseudo code of the method *Scheduler*.

Algorithm 1. SCHEDULER pseudo code

```
    Input: schedule, Reg-distribution, Current cuboid;
    Begin
  SCHEDULER-state←on;
  For (C from current_cuboid to the first cuboid) do
  If(C is not the full cuboid) then
    If(no computed parent cuboid ) then
      SCHEDULER-state←off;
      Exit;
    Else //at least one parent cuboid is available
        For (each Data node DN) do
            DN-coef←0;
            If(DN has the best region) then
              DN-coef←DN-coef+1;
            If(DN owns C's parent cuboid)then
              DN-coef←DN-coef+2;
            If (DN is not active) then // not loaded
              DN-coef←DN-coef+4;
        End for;
        Assign C to the node with the max coef ;
          If (schedule [DNMAX, parent]=0) then
            schedule [DNMAX, parent]←1;
  Else //C is the full cuboid
    Assign C to the best node;
  C←next cuboid;
  Current_cuboid ← C;
  If (Send-cuboid_state=off) then
    Send-cuboid ();// activate Send-cuboids
End for;
End.
```

### B. SEND-CUBOIDS

*Send-Cuboids* is a method responsible for sending messages to the outside (data nodes). When is called by *Sheduler*, this method sends each cuboid to the corresponding Data node in the schedule.

The sent message can include the cuboid to be computed and a parent cuboid if required. *Send-Cuboids* needs sending one of the parent cuboids if the data node does not hold any of them.

Algorithm 2. SEND-CUBOIDS pseudo code

```
  Input: schedule
         Current cuboid;
  Begin
Send-Cuboid_state←on;
For (C, from current_cuboid to the first cuboid) do
      DN ← the node in which its intersection
   with C is equal to 0;
   If (!exist DN) do
     Send-Cuboid_state←off;
     Exit;
   Else;
     If ((C =full) or (DN holds a parent)) do
       Send(C, DN);
     Else
       Send(C, DN, parent(C));
     C← the next cuboid;
     Current cuboid←C;
End for;
  End.
```

### C. RECEIVE-CUBOIDS

*Receive-Cuboids* is a simple method having the role of receiving computed cuboids from the data nodes and saving the result in the file *Cuboids-Results*. In addition, it updates the *Schedule* indicating that the cuboid is computed and if *Scheduler* is deactivated, *Receive-Cuboids* calls it in order to consider the new cuboid. Algorithm 3 below gives the pseudo code of the method *Receive-Cuboids*.

Algorithm 3. Receive-Cuboids pseudo code

```
Input: C: received cuboid
Begin
  Save (C); // in Result-Cuboid
 C ∩ DN←1; //in the schedule
  If (SCHEDULER_state=off) then
   SCHEDULER;
End.
```

## 3.2 Data Node

In this section, we provide a detailed description of the data node which plays a double role; in addition to holding its assigned data region, DN accomplishes the tasks assigned to it by the scheduling node or by the other data nodes. These tasks are achieved by means of four methods exploiting and updating three kinds of data.

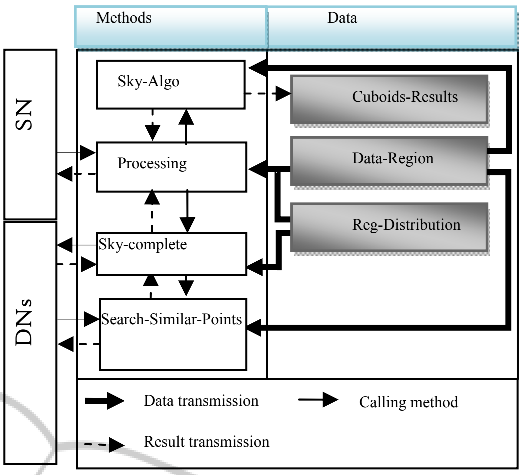


Figure 5: Structure of the data node.

### 3.2.1 Data

### A. DATA-REGION

*Data-Region* is the file containing the data region assigned to the node. Based on the distribution example explained in Figure 4 and Table 1, Figure 6 illustrates the data region of each data node using the data set of our running example.

| DN1 | DN2 |
|---|---|
| p3 | p1, p4, p5 |
| DN3 | DN4 |
| p2 | |
| DN5 | DN6 |
| p6 | p7 |
| DN7 | |
| p8 | |

Figure 6: Data-Region files.

### B. REG-DISTRIBUTION

A part of the *Reg-Distribution* table of the scheduling node is available on each data node, this table has the same structure but it considers only the data node's neighborhood.

According to CAN network, for instance, two nodes are neighbors in a *d*-dimensional space if their regions overlap along *d*-1 dimensions and abut along one dimension. For example, the neighbours of DN3 are DN1, DN2, DN6 and DN7.

### C. CUBOIDS-RESULTS

The data node holds a summarized version of the *Cuboids-Results* file included in the scheduling node. It has only the cuboids computed by the DN or sent to it by the SN during the computation of another cuboid.

### 3.2.2 Methods

As illustrated in figure5, the data node has four methods; *Processing, Sky_Algo, Sky-Complete* and *Search -Similar-Points.*

#### A. PROCESSING

*Processing* is the manager of the Data node; it is responsible for computing each cuboid using the suitable methods considering the nature of the cuboid.

If the cuboid is of the full space and DN is the first queried node, the method *Processing* calls the method *Sky-Algo* which has as input the *Data-Region* and output the skyline points and the filter points (F-P) (the filter points may be the skyline points themselves), the latters are forwarded to another DN chosen using the *Reg-Distribution* and the network's routing system. If the query is forwarded with the filtering point(s) from another DN, a pruning test is conducted, if the region of the node in hand is pruned by the filtering point(s), the query is forwarded to another DN with the received filtering point(s) (R-F-P) without performing any computation; otherwise, *Sky-Algo* is executed using the data region and the received filtering point(s) producing the local skyline points and new filter point(s) (N-F-P). If there are more nodes to be queried, the query is forwarded with the new filtering points, if not, the final skyline is returned back to the SN.

In the case a subspace cuboid, the method *Processing* calls *Sky-Algo* providing it with a parent cuboid. When getting back the results from *Sky-Algo*, *Processing* either returns them directly to SN, in case of a distinct subspace, or provides them to *Sky-Complete* in order to complete the skyline points. The results of *Sky- Complete* are directly returned to the scheduling node as a final cuboid.

#### B. SKY-ALGO

*Sky-Algo* is the method which computes the skyline locally using either the data available on the node (data region) if the cuboid is a full or the data provided by the method *Processing* which gets them either from the SN or from the cuboids already computed by the Data node in hand.

Besides, the method uses the well-known skyline algorithm BNL to compute the skyline points, these points are sometimes filtered using the received filter points.

#### C. SKY-COMPLETE

*Sky-Complete* relies only on the computation of the subspace, it has as input the intermediate skyline computed by the *Sky-Algo* method.

Algorithm 4. Sky-Complete pseudo code

```
Input: Reg-Distribution table
       S: set of points
Define: P: S.first
        DN: data node
Begin
  For (S) do
    DN← first data node;
    For (each node in Reg-Distribution) do
      If (P ϵ DN) then
        If exist(cluster(DN)) then
          Cluster(DN)← Cluster(DN) ∪ {P};
        Else
          Create (cluster (DN));
          Cluster(DN)← Cluster(DN) ∪ {P};
        Reg-Distribution.end;
      Else
        Reg-Distribution.next;
      S.next;
    End for;
  End for;
  For (all clusters) do
    Msg← search-similar-points(actual cluster);
    Send(Msg, cluster.DN);
  End for;
End.
```

In fact, the *Reg-Distribution* table is used in order to determine which data node has to be queried to find the similar points of each skyline point.

In order to avoid querying the same node many times, *Sky-Complete* classifies the intermediate skyline according to the node regions having at most *N* clusters (*N* is the number of Data nodes), then each cluster is sent to the corresponding node after omitting the repeated points (points which have the same values in all the dimensions of the subspace). Since the *Reg-Distribution* table in the Data node contains only the information of the neighborhood, the remaining points are sent to the neighbors in order to get clustered.

#### D. SEARCH-SIMILAR-POINTS

*Search-Similar-Points* relies only on computing the subspace cuboids. It receives the order from *Sky-Complete* run whether in the same node or in the outside (another Data node) providing it with a cluster of data points.

*Search-Similar-Points* scans the Data-Region of its node to find the similar points to those received from the querying node and return them back to it. The process is exhibited in Algorithm 5 below.

```
Algorithm 5. Search-Similar-Points
Input: cluster: received data points
       DRP: data region points
Output: SP: similar points
Define: P1: cluster.first
        P2: DRP.first
Begin
   For (cluster) do
      For (DRP) do
         If (P1(U)==P2(U)) then
            SP← SP ∪ {P2};
         cluster.next;
      End for;
     RDP.next;
   End for;
End.
```

## 4 EXPERIMENTAL RESULTS

In this section, we present an extensive performance evaluation of our algorithms. Since there is no work in the literature treating the computation of the skycube in P2P systems, we compare our algorithms with computing the cuboids individually using the DSL (Wu et al., 2006).

### 4.1. Experimental Settings

The proposed algorithms and the algorithm DSL have been encoded in java language and implemented in a simulated CAN network of 11 nodes, 10 data nodes and a Scheduling Node on a laptop with an intel (R) Core (TM) i 3 CPU and 4GB of main memory machine running the Windows 7 operating system.

In all experiments we use three real datasets; NBA players' season statistics from 1946 to 2009 (http://basketballreference.com), Shuttle Landing Control dataset and a part from El Nino dataset which contains oceanographic and surface meteorological readings taken from a series of buoys positioned throughout the equatorial Pacific http://archive.ics.uci.edu/ml/datasets.html).

### 4.2. Effect of Dimensionality

First, we study the effect of dimensionality on the performance of the proposed method over the three datasets. We notice, as seen in Figure 7, that *Distributed-Top-Sky* outperforms DSL within all datasets

In fact, the gain differs from a dataset to another for instance, in the case of NBA dataset, *Distributed-Top-Sky* is only two times faster. It is 10 times faster within Shuttle and more than 100 times faster in the case of El Nino dataset. Moreover, the DSL approach could not complete the computation for high dimensions within NBA and El Nino datasets.

### 4.3. Rate of Transferred Objects

Figure 8 shows the rate of transferred objects per cuboid. We see that the number of transferred objects in DSL is always higher than *Distributed-Top-Sky*. As well, this difference is very large when El Nino dataset is considered which explains the great performance of *Distributed-Top-Sky* in such a case.

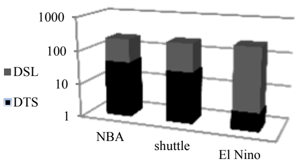


Figure 8: Rate of transferred objects.

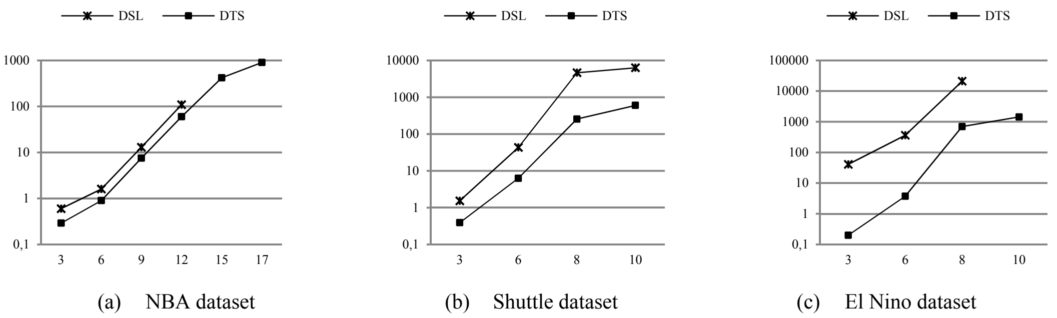


Figure 7: Scalability w.r.t dimensionality.

## 5 CONCLUSIONS

This paper has dealt with the computation of the skycube in a structured peer-to-peer system by introducing a method called *Distributed-Top-Sky*).

For validating the proposed approach, we have conducted an experimental evaluation on various datasets. The efficiency of the proposed method has been proven; it outperformed individual computation of the cuboids using the DSL approach. These results can be optimized in the future by distributing the scheduling task.

Another possible direction for future work is to investigate the second step of the materialized views selection problem for skyline queries under various constraints since no previous work treats the distributed version of this problem with skyline queries and there are a few works for relational queries. The works presented in (Bauer et al., 2003; Chaves et al., 2009) can be the start point of such an investigation.

## REFERENCES

Basccketball refrence. http://basketballreference.com (accessed january 8th, 2013).

Bauer and Lehner, W. (2009) “On solving the view selection problem in distributed data warehouse architectures.” In SSDBM, pages 43–, 2003.

Börzsönyi, S., Kossmann, D. and Stocker, K. (2001) “The skyline operator.” In Proceedings of International Conference on Data Engineering (ICDE).

Brahimi, S. Kholladi, M.K. and hamerelain, A. (2013) “top-Sky: top-down algorithm for computing the skycube” in international symposium of programming and systems” (isps 2013).

Chaves, L.W.F. Buchmann, E.Hueske, F. and B¨ohm ,K. “Towards materialized view selection for distributed databases”. In EDBT, pages 1088–1099, New York, NY, USA, 2009. ACM.

Chen, B. and Liang, W. (2009) “ Progressive skyline query processing in wireless sensor networks” In: International Conference on Mobile Ad-hoc and Sensor Networks(MSN), pp. 17{24 (2009).

Chen, L., Cui, B., Lu, H., Xu, L. and Xu, Q. (2008) “iSky: Efficient and progressive skyline computing in a structured P2P network.” In: Proceedings of the International Conference on Distributed Computing Systems , 2008.

Jagadish, H., Ooi, B. and Vu, Q.(2005) “BATON: a balanced tree structure for peer-to-peer networks.” In Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 661{672 (2005).

Li, H., Tan, Q., Lee, W.: Efficient progressive processing of skyline queries in peer-to-peer systems. In: Proceedings of the International Conference on Scalable Information Systems(Infoscale), p. 26 (2006).

Machine learning repository. http://archive.ics.uci.edu/ ml/datasets.html (accessed may 25th, 2013).

Pei, J., W. Jin, M. Ester, et Y. Tao (2005). Catching the best views of skyline : A semantic approach based on decisive subspaces. In VLDB, pp. 253–264.

Raïssi, C. Pei, J. and Kister, T. “Computing closed skycubes,”PVLDB, vol. 3, pp. 838–847, 2010.

Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Schenker, S. (2001) “A scalable content-addressable network”. In: Proceedings Conference on Applications, technologies, architectures, and protocols for computer communications.

Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H. (2001) Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of Conference on Applications, technologies, architectures, and protocols for com computer communications (SIGCOMM), pp. 149{160 (2001).

Veloso R. R., Cerf .L, Raıssi .C and W. Meira Jr, “Distributed Skycube Computation with Anthill”, In ISCAHPC 2011. IEEE Computer Society.

Wang, S., Ooi, B., Tung, A., Xu, L. (2007) » Efficient skyline query processing on peer-to-peer networks”. In: Proceedings of International Conference on Data Engineering (ICDE).

Wang, S., Vu, Q.H., Ooi, B.C., Tung, A.K. and Xu, L. (2009) “Skyframe: a framework for skyline query processing in peer-to-peer systems.” In The VLDB Journal 18(1), 345-362 (2009).

Wu, P., Zhang, C., Feng, Y., Zhao, B., Agrawal, D. and Abbadi, A (2006). “Parallelizing skyline queries for scalable distribution.” In: Proceedings of International Conference on Extending Database Technology (EDBT), pp. 112{130 (2006).

Yuan, Y., X. Lin, Q. Liu,W.Wang, J. X. Yu, et Q. Zhang (2005). Efficient computation of the skyline cube. In VLDB, pp. 241–252.