Preference Analytics in EXASolution

Stefan Mandl, Oleksandr Kozachuk
EXASOL AG
Neumeyerstraße 24
90411 Nürnberg, Germany
firstname.lastname@exasol.com

Markus Endres, Werner Kießling
University of Augsburg
Universitätsstr. 6a
86159 Augsburg, Germany
lastname@informatik.uni-augsburg.de

Abstract: Skyline queries and the more general concept of preferences are well-known in the database community and there are many academic approaches for the computation of the best-matching objects. Furthermore, data analytics and multi-criteria optimization play an important role in Business Intelligence where it facilitates optimal decision making. *Preference Analytics* is the combination of preferences and data analytics. SKYLINE is EXASOL's implementation of Preference Analytics in its commercial database management system EXASOLUTION. In this paper, we present SKYLINE from a user's perspective, describe algorithmic design decisions, and discuss its implementation in a distributed and parallel environment. The paper closes with an empirical evaluation of the system based on a number of preference queries over the TPC-H dataset using different scale factors.

1 Introduction

The Skyline operator [BKS01] and the more general concept of preference database queries [SKP11, KEW11] has emerged as an important summarization technique for multi-dimensional datasets. The popularity of preferences is mainly due to their applicability for decision making applications [GRB11]. In such applications, the database tuples represent a set of multi-dimensional data objects, and the *Skyline*, also named *Pareto Set*, contains those objects that are the best trade-offs between different attributes, i.e. those objects that are not dominated by any others. An object p having d attributes dominates an object q, if p is strictly better than q in at least one attribute and not worse than q in all other attributes, for a defined comparison function. Preferences enable users to specify a pattern that describes the type of information he is searching for. Since preferences express soft constraints, the most similar data will be returned when no data exactly matches that pattern (Best-Matches Only set). From this point of view, preference database queries are an effective method to reduce very large datasets to a small set of highly interesting results.

EXASolution is a high performance parallel and distributed in-memory engine for data analytics and data warehousing produced by EXASOL AG[1]. It is the first commercial system which incorporates the functionality of Skylines for *Preference Analytics*. Subsequently, we show an example of a preference query using the novel SKYLINE syntax for Preference Analytics in EXASolution (see Section 3.1).

[1]http://www.exasol.com

Example 1. *Consider the task of finding the best funds in the market – a daily problem for financial investors. Assume one wants to invest in the fund that provides the* highest return *and the* lowest risk. *Unfortunately, investment funds do not work that way: the more conservative a fund, typically the lower its risk, but also its return value. Figure 1 presents a sample dataset of funds in the market and the Standard & Poor's 500 (S&P 500), which is a stock market index based on the market capitalizations of 500 large companies. So, there is no right answer as to which fund to select.*

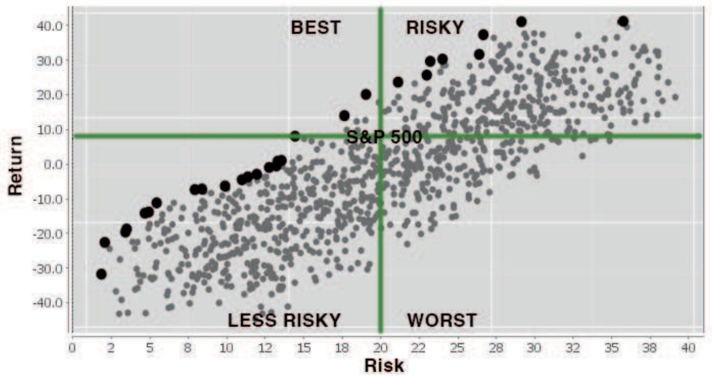

Figure 1: Example of the Fund Universe Comparison graph (*http://www.fundreveal.com*) and the best-matches only set (bold points).

However, using SKYLINE *you can tell the system that this is your preference and it will provide you with a shortlist of candidates – effectively eliminating all objects that are worse in every respect than competitors. In Figure 1 the bold points represent the optimal subset regarding the subsequent preference query:*

```
SELECT * FROM funds
PREFERRING HIGH return PLUS LOW risk;
```

Figure 2: Simple preference query to find the *best* fund.

The **`PREFERRING`** *keyword introduces a preference. The connection of two preferences by* **`PLUS`** *is the Pareto composition and states the equal importance of preferences. The keywords* **`HIGH`** *and* **`LOW`** *define whether an expression should have* high *or* low *values. The query selects all objects which are not dominated by any other objects in all dimensions. From this Skyline set one can now make the final decision, thereby weighing the personal preferences for a more conservative or more risky fund.*

Preference Analytics addresses the fundamental problems of traditional data mining approaches and multi-criteria decision making, where the ever-increasing volumes of data and multiplicity of variables mean that sorting, filtering and weighting lead to sub-optimal analyses. This capability is unique to EXASolution and was developed to help clients with more sophisticated analytic and data mining requirements. Potential application areas include segmentation, next best offer, fraud and portfolio analysis, etc.

The remainder of this paper is organized as follows: In Section 2 we discuss some related work. In Section 3 we present *Preference Analytics* including a parallel and distributed Skyline algorithm used by SKYLINE in EXASolution. Afterward we show elaborate examples of Preference Analytics in Section 4. We conduct a performance evaluation on the TPC-H benchmark dataset in Section 5. Section 6 contains our concluding remarks.

2 Related Work

Preferences in databases—as shown by a recent survey [SKP11]—are a well established framework to create personalized information systems. By using well designed preference models, users can be provided with just the information they need, thereby overcoming the dreaded empty result set and flooding effect [KEW11]. Traditional database engines or query languages do not support preference queries. However, in the last decade some extensions to the SQL language have been proposed.

Börzsönyi et al. [BKS01] introduced the **SKYLINE OF** clause with its basic form

```
SELECT ... FROM ... WHERE ...
  SKYLINE OF [DISTINCT] A1 [min | max | diff], ...,
                        An [min | max | diff];
```

The columns `A1, ..., An` are the attributes over which the preferences apply. Their domains must have a natural total ordering, such as integers. The directives **min** and **max** specify whether one prefers low or high values, respectively. The **diff** states that one wants to retain the best choices with respect to each distinct value of that attribute. Unfortunately, this approach is restricted to **min**, **max** and **diff**, such that complex preference queries are not possible.

Chaudhuri et al. show through an implementation in the commercial database product Microsoft SQL Server 2005 (Beta version) that Skyline queries can substantially benefit from cardinality estimation techniques [CDK06, End14]. However, their Skyline implementation never found its way into any product release of MS SQL Server. Eder and Fang [EF09] implemented the original Skyline syntax in PostgreSQL in order to show the integration of the Skyline operator with other relational operators, thus more sophisticated queries can be constructed. This Skyline implementation never became an integral part of the PostgreSQL database system.

A more general approach of Skylines are *preference queries*, which allow a more detailed specification of user wishes. *Preference SQL* is an extension to SQL to specify such arbitrary user preferences [KEW11]. Preferences in this sense are strict partial orders and a preference query returns the maximal elements according to this order, also called the *Best-Matches Only set* (BMO-set) [Kie02]. The basic form of a PreferenceSQL query is as follows:

```
SELECT ... FROM ... WHERE ...
PREFERRING A1 <preference_constructor> [ AND | PRIOR TO | RANK | DUAL ]
           ...                         [ AND | PRIOR TO | RANK | DUAL ]
           An <preference_constructor>;
```

The **PREFERRING** keyword introduces a *preference*. The connection of two preferences by **AND** is the *Pareto composition* and states the equal importance of preferences. If we restrict the attention to LOWEST (**min**) / HIGHEST (**max**) as input preferences, then Pareto preference queries coincide with the traditional *Skyline queries* above.

Arvanitis and Koutrika [AK12] pushed preference query processing closer to the database management system (DBMS) and explained how to integrate the preference operator and its optimization into a DBMS. Levandoski et al. [LEMK13] presented FlexPref, a general framework for extensible preference evaluation. FlexPref is implemented in the query processor of PostgreSQL, and supports various preference evaluation methods. Although FlexPref was integrated into PostgreSQL, it has never been an inherent part of it. Golfarelli et al. [GRB11] followed the basic principles of [Kie02] and developed myOLAP, an approach to express and evaluate OLAP preferences.

Preference query processing received enormous attention in single-database environments, i.e., in a centralized setup, cf. [CCM13], as well as in multi-core environments [PKP+09, SLB10, LVDN14, EK14]. As nowadays data are increasingly stored and processed in a distributed way, Skyline processing over distributed data has attracted much attention recently. EXASolution is a highly distributed database management system and therefore novel algorithms have to be developed for Preference Analytics. We give a short overview on existing approaches on distributed Skyline computation.

Balke et al. [BGZ04] proposed the first distributed Skyline algorithm, by considering the underlying relation is vertically partitioned between local servers, i.e., each server keeps only one attribute of the relation. A vertical partitioning is also used in [TBPY13], where a general solution for arbitrary dimensionality is proposed. In contrast, Wu et al. [WZF+06] assume a constrained horizontal partitioning. They used a CAN overlay to create grid partitions, each assigned to a processor. The main disadvantage of this approach is its low parallelism and that processors exchange the entire Skyline which floods the network. Zhu et al. [YSZ09] studied the distributed Skyline query when the underlying dataset is horizontally partitioned onto a set of local servers. Their algorithm can solve the problem with low bandwidth consumption. In [RJVDN09] the authors propose AGiDS, a framework for efficient Skyline processing over distributed data. Thereby a grid-based data summary is used to reduce the amount of transferred data. The main focus in the related work on distributed Skyline computation is on highly distributed systems, such as P2P systems, where each server stores a fraction of the available data, e.g. [LTL06, WVO+09]. Hose et al. [HLS06] for example focused on Peer Data Management Systems and proposed an algorithm identifying relaxed Skyline results. Skyline queries have also been studied in other distributed environments, such as web information systems [BGZ04, LYLC06], mobile applications [XC10], or parallel shared-nothing architectures [VP10, VDK08]. Also distributed Skylines are relevant with respect to different data types, such as streamed [SHZ+10] or uncertain data [DJ10]. A detailed survey on the state-of-the-art in Skyline processing in distributed environments is given in [HV12].

All mentioned approaches from preference specification to distributed Skyline algorithms have in common that they are of academic nature and none of them has ever been integrated into a product release of a commercial database system. This paper describes the first commercial distributed Skyline environment for *Preference Analytics*.

3 Preference Analytics in EXASolution

Data analytics has become a key priority for many companies to enable a better decision-making process. *Preference Analytics* which integrates preference computations with data analytics enables users to define wishes and to extract the best objects from a given dataset. Preference Analytics is available in EXASolution via SKYLINE which seamlessly integrates with other analytical features. In this section we introduce the basics of SKYLINE; thereby commercializing the ideas in [Kie02, KEW11] by modifying the syntactical terms and introducing expressions and partitions in the preference clause.

3.1 Syntax

SKYLINE is available in EXASolution by an extension of the **SELECT**, **DELETE**, and **UPDATE** statements. In this paper, we only consider the **SELECT** statement because this is the most important use case for Skylines. Furthermore, using **DELETE** and **UPDATE** with Skylines is straightforward.

SELECT statements are extended by an *preferring clause* (Figure 3) which is introduced by the keyword **PREFERRING** (as in [KEW11]), and followed by a *preference term* (Figure 4) and an optional **PARTITION BY** sub-clause.

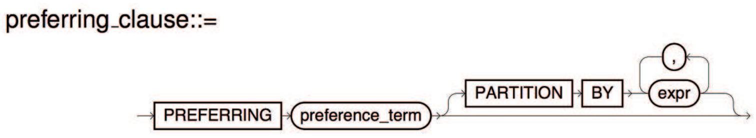

Figure 3: Preference clause in EXASolution.

The **PREFERRING** clause can be defined after the **WHERE** condition of a **SELECT** statement. It can contain the elements depicted in Figure 4 and is evaluated after the **WHERE** clause.

- **PARTITION BY**: If you specify this option, then the preferences are evaluated separately for each partition.
- **HIGH** and **LOW**: Defines whether an expression should have high or low values. Please note that any numerical expressions are expected here.
- *Boolean expressions*: In case of boolean expressions, the elements are preferred where the condition results in TRUE.
- **PLUS**: Via the keyword **PLUS**, multiple expressions of the same importance can be specified, i.e. the expressions form a Pareto preference query.
- **PRIOR TO**: With this clause you can nest two expressions hierarchically. The second term will only be considered if two elements have a similar value for the first term.
- **INVERSE**: By using the keyword **INVERSE**, you can create the dual preference expression. Hence, the expression **LOW** `price` is equivalent to **INVERSE**`(`**HIGH** `price)`.

Note that in contrast to existing approaches as discussed in Section 2 it is possible to use arbitrary mathematical and boolean expressions in a preference term.

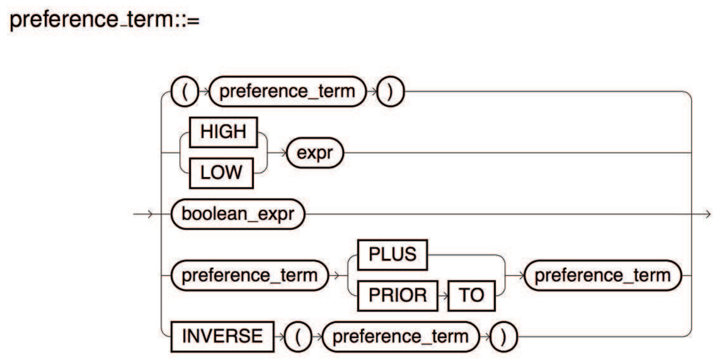

Figure 4: Preference term in EXASolution.

3.2 Evaluation of Preference Queries in EXASolution

EXASolution is a distributed and high-performance parallel database. It typically runs on a number of machines in a cluster. Data is distributed horizontally among all machines. This means that every machine contains parts of all (temporary or permanent) tables and result sets. When answering queries, an important goal of all algorithms in EXASolution is to maximize utilization of the available hardware.

Before we go into the details of the Skyline algorithm in EXASolution, we briefly describe the problem of Skyline computation. In general the Skyline contains those objects of a given dataset D that are not dominated by any others for a defined comparison function. In this paper we define the skyline as the set of best objects of a preference query.

More formally, the Skyline of a dataset D is defined by the maxima in D according to the ordering $<_P$, where $x <_P y$ for d-dimensional tuples $x = (x_1, ..., x_d)$, $y = (y_1, ..., y_d) \in D$ means *"y is better than x"* concerning the preference P. The *SKYline* set is

$$\text{SKY}(D) = \{t \in D | \nexists u \in D : t <_P u\}$$

The basic algorithm of SKYLINE in EXASolution belongs to the block-nested-loop class (BNL) [BKS01] of algorithms that filter the dataset using a compare function. The block-nested-loop algorithm linearly scans over the input dataset D. The idea of this algorithm is to continuously maintain a *window* (or block) of tuples in main memory containing the maximal elements with respect to the data read so far. When a tuple $p \in D$ is read from the input, p is compared to all tuples of the window and, based on this comparison, p is either eliminated, or placed into the window. Three cases can occur: First, p is dominated by a tuple within the window. In this case, p is eliminated and will not be considered in future iterations. Of course, p need not be compared to all tuples of the window in this case. Second, p dominates one or more tuples in the window. In this case, these tuples are eliminated; that is, these tuples are removed from the window and will not be considered in future iterations while p is inserted into the window. And third, p is incomparable with all tuples in the window. In this case p is inserted into the window. At the end of the algorithm the window contains the maximal elements, i.e., *the Skyline*. The best case complexity is of the order $\mathcal{O}(n)$; n being the number of input tuples. In the worst case, the complexity is of the order of $\mathcal{O}(n^2)$.

The major advantage of a BNL-style algorithm is its simplicity and suitability for computing the maxima of arbitrary partial orders.

Given these constraints, the general design goals of the algorithm are straightforward:

- Optimize for interesting Skylines to be computed fast (and trade that for non-interesting ones tending to be slower)
- Compute Skyline in parallel and distributed on all machines

For the first goal we assume that smaller Skylines are more interesting than larger ones. This idea is based on the applications where Skyline is used to create short lists. The second goal is facilitated by the *additivity* [HV12] and the *distributability* of the Skyline operator. Additivity means that given n datasets D_i corresponding to n machines, the Skyline objects are the same if the Skyline operator is evaluated on (i) the union of the n datasets or (ii) first on each set separately and then once more on the union of the result set.

Property 1 (Additivity of the Skyline operator)**.** *Given a dataset D and n datasets D_i such that*

$$D = D_1 \cup \ldots \cup D_n$$

Then the following equation holds:

$$\mathit{SKY}(D_1 \cup \ldots \cup D_n) = \mathit{SKY}\,(\mathit{SKY}(D_1) \cup \ldots \cup \mathit{SKY}(D_n))$$

Distributability means that members of the Skyline can be computed in the cluster with distributed data by testing them on each machine in the cluster independently. If they stand the test on all machines, they are part of the global Skyline. We formally specify this new outcome in the next property.

Property 2 (Distributability of the Skyline operator)**.** *Given a finite dataset D and n datasets D_i such that*

$$D = D_1 \cup \ldots \cup D_n$$

Then for each $d \in D$:

$$d \in \mathit{SKY}(D) \Leftrightarrow d \in (\mathit{SKY}(D_1 \cup \{d\}) \cap \ldots \cap \mathit{SKY}(D_n \cup \{d\}))$$

Proof. "$\Rightarrow$": Let $d \in \text{SKY}(D)$ and $d \notin (\text{SKY}(D_1 \cup \{d\}) \cap \ldots \cap \text{SKY}(D_n \cup \{d\}))$, then for some D_i, $d \notin \text{SKY}(D_i \cup \{d\})$. So there exists some element $d_0 \in D_i$ which dominates d. Since d_0 also is in D, $d \notin \text{SKY}(D)$ which contradicts the assumption.

"$\Leftarrow$": Let $d \in (\text{SKY}(D_1 \cup \{d\}) \cap \ldots \cap \text{SKY}(D_n \cup \{d\}))$ and $d \notin \text{SKY}(D)$. Then there exists $d_0 \in D$ which dominates d. Furthermore, there is some D_i with $d_0 \in D_i$ for which then $d \notin \text{SKY}(D_i \cup \{d\})$, which contradicts the assumption. □

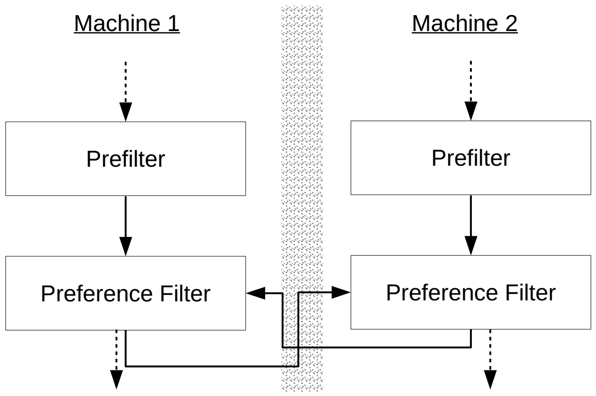

Figure 5: Skyline algorithm on a cluster with two machines

Based on Property 1 and 2, EXASOL's Skyline algorithm (see Figure 5) conceptually works in three steps:

1. Local pre-filtering
2. Local filtering
3. Global filtering

During *local pre-filtering*, subsets of the dataset are filtered independently on every machine fully in parallel, typically yielding a much smaller set of candidates per machine. Each of the datasets comprises a small Skyline but their union does not need to do so. This step corresponds to the inner $\text{SKY}(\cdot)$ operations according to Property 1.

After pre-filtering stopped on all machines. *Local* and *global filtering* are executed in parallel fashion, therby maximizing hardware utilization. Per node, candidates are compared to all other candidates (quadratic cost in the number of Skyline candidates). Candidates that turn out to be local solutions are sent off to the next machine in the cluster and compared to the local candidates there. Following this process, once a candidate reaches the machine it originated from it is known to be a global solution and written into the result set. Local filtering corresponds to the outer $\text{SKY}(\cdot)$ operations according to Property 1. The test on remote-machines corresponds to the right-hand side in Property 2.

4 Elaborate Examples

In this section we present elaborate examples which show the power of Preference Analytics and its advantage in comparison to standard SQL.

During this section we use the sample data in Table 1, which presents an ***energymap***, an extract of renewable energy sources in Germany. The full dataset is publicly available at *www.energymap.info*. The dataset contains information on solar stations, wind farms, biomass gas facilities, and hydro stations, in total more than 1.5M tuples.

generator_type	**town**	**gpscoord**	**power_output**	**tso**	**...**
Solarstrom	Weismain	POINT (50.05 11.233)	379865	TenneT TSO GmbH	
Solarstrom	Dingolfing	POINT (48.633 12.5)	53582	TenneT TSO GmbH	
Biomasse	Papenburg	POINT (53.067 7.4)	20000	TenneT TSO GmbH	
Biomasse	Emden	POINT (53.367 7.217)	19975	TenneT TSO GmbH	
Biomasse	Neumarkt	POINT (49.28 11.463)	19900	TenneT TSO GmbH	
Solarstrom	Fuchsstadt	POINT (50.1 9.933)	18737	TenneT TSO GmbH	
...	...	...	...	...	...

Table 1: Sample dataset of renewable energy sources in Germany (see *www.energymap.info*; please note that in order to take advantage of EXASolution's GEOMETRY type, the original columns *GPS-Lat* and *GPS-Lon* have been combined into the column *gpscoord* for our example).

Assume that one wants to choose the source of power for his company in Nuremberg. Given the `energymap` sample data in Table 1 the goal is to find suitable power stations. Since there is no power station exactly at the company's location, we have to consider the trade-off between big power stations further away and little power stations nearby.

Example 2 (Energymap – First attempt)**.** *The first attempt* without *Preference Analytics could be to find all interesting power stations using a* **`WHERE`** *clause to select only big power stations which are not too far away and very close power stations which are not too small, cf. Figure 6. We use the* **`ST_DISTANCE`** *function to compute the distance of each power station to the company's location coordinates* `N 49.47 E 11.041`.

```
SELECT (em.power_output/1000000) megawatts,
       ST_DISTANCE('POINT(49.47␣11.041)', em.gpscoord) AS dist,
       em.generator_type, em.town, em.tso AS operator
FROM energymap em
WHERE em.gpscoord IS NOT NULL
-- the power station is nearby
-- (< 1.0 degree straight line distance, roughly 100km)
AND LOCAL.dist < 1.0
-- and it is relatively powerful (> 4 MW)
AND em.power_output > 4000000
ORDER BY em.power_output DESC;
```

Figure 6: Standard SQL query to find big power stations not too far away and very close power stations which are not too small.

One problem to notice here is that this query completely ignores middle sized power stations and those that are in the middle distance. Some of these might be a good solution. So this query is not satisfying and does not produce a usable short list of power stations. In fact, the query above returns no result at all which is an example of the empty result set effect (see Section 1) which is typical for such scenarios.

Example 3 (Energymap cont'd – Second attempt)**.** *The second attempt is to calculate a score based on a function of distance and power.*

```
-- We guess a suitable function of distance and power,
-- calculate a score for each station and consider the first 10 results
SELECT (em.power_output/1000000) megawatts,
       ST_DISTANCE('POINT(49.47␣11.041)', em.gpscoord) AS dist,
       em.generator_type, em.town, em.tso AS operator
FROM energymap em
WHERE em.gpscoord IS NOT NULL
ORDER BY em.power_output/1000 + 100000/(1 + 10 * LOCAL.dist)
DESC LIMIT 10;
```

Figure 7: Calculate scores based on a function of distance and power.

However, there is no universal accepted formula for this task so we have to guess the function. If we are not satisfied with the results we have to change the function, and so forth. This is an iterative and time consuming approach. Furthermore, this process may lead to results which are further away and less powerful than other power stations. There is no protection from getting worse power stations over better ones. Furthermore, what about really powerful power stations further away? And what about really near small power stations?

Example 4 (Energymap cont'd – Preference Analytics)**.** *Finally using* SKYLINE *leads to a short list of Pareto optimal results. All we have to do is to add the* **PREFERRING** *clause for* low distance *and* high power output *to the query, cf. Figure 8.*

```
SELECT (em.power_output/1000000) megawatts,
       ST_DISTANCE('POINT(49.47␣11.041)', em.gpscoord) AS dist,
       em.generator_type, em.town, em.tso AS operator
FROM energymap em WHERE em.gpscoord IS NOT NULL
PREFERRING LOW LOCAL.dist PLUS HIGH em.power_output
ORDER BY em.power_output DESC;
```

Figure 8: Use Preference Analytics to eliminate sub-optimal candidates.

The algorithm for Skyline computation in EXASolution is very efficient and runs distributed and parallel on all the machines of an EXASolution cluster. On a small test cluster of two rather dated servers, the result is computed in less than 1 second on the energymap dataset having more than 1.5M tuples. Using Preference Analytics one can compute the optimal trade-off of distance and power, creating a short list of highly interesting objects, ranging from a massive solar power station more than 400km away to tiny stations next door, cp. Table 2. From this set the user can make the final decision, thereby weighing the personal preferences for the best object.

megawatts	**dist**	**generator_type**	**town**	**operator**
1.468280	2.343107338557071	Solarstrom	Herrenberg	TransnetBW GmbH
0.626608	0.484542051838642	Solarstrom	Schirradorf	TenneT TSO GmbH
0.019900	0.462800172860813	Biomasse	Neumarkt	TenneT TSO GmbH
0.009200	0.374518357360488	Solarstrom	Pleinfeld	TenneT TSO GmbH
0.008213	0.277218325512582	Solarstrom	Heilsbronn	TenneT TSO GmbH
0.006125	0.093171884171138	Solarstrom	Zirndorf	TenneT TSO GmbH
0.006025	0.080993826925266	Biomasse	Sandreuth	TenneT TSO GmbH
0.001999	0.074060785845142	Biomasse	Fürth	TenneT TSO GmbH
0.000856	0.069570108523705	Solarstrom	Fürth	TenneT TSO GmbH
0.000451	0.034828149534534	Solarstrom	Nürnberg	TenneT TSO GmbH

Table 2: Result of the Preference Analytics query in Figure 8.

Example 5 (Energymap – Standard SQL)**.** *Since Skyline queries are not outside the expressive power of standard SQL [BKS01] we want to present the query in Example 4 in SQL using an inelegant* **`WHERE NOT EXISTS`** *clause. Doing so, this is cumbersome and not practical and leads to a much longer runtime of several minutes, especially for large datasets and many dimensions.*

```
SELECT (emo.power_output/1000000) megawatts,
       ST_DISTANCE('POINT(49.47␣11.041)', emo.gpscoord) AS disto,
       emo.generator_type, emo.town, emo.tso AS operator
FROM energymap emo
WHERE emo.gpscoord IS NOT NULL
AND NOT EXISTS (
  SELECT (emi.power_output/1000000) megawatts,
         ST_DISTANCE('POINT(49.47␣11.041)', emi.gpscoord) AS disti
  FROM energymap emi
  WHERE emi.gpscoord IS NOT NULL
  AND (
   -- no closer stations that are at least as powerful
   (LOCAL.disti < LOCAL.disto AND emi.power_output >= emo.power_output) OR
   -- no more powerful stations that are at least as close
   (LOCAL.disti <= LOCAL.disto AND emi.power_output > emo.power_putput)
  )
);
```

Figure 9: Preference Analytics query in standard SQL.

In our next example we would like to show that, as indicated in Figure 4, arbitrary expressions can be used in EXASolution's Preference Analytics.

Example 6. *If for some reason the user in Example 4 had an affinity to the location* Pleinfeld*, he could use a preference like in Figure 10. Here the standard SQL expression* ***LIKE*** *is used to find all towns which have the term* `PLEINFELD` *in their name. Then all towns are preferred where the condition results to* TRUE.

```
SELECT (em.power_output/1000000) megawatts,
       ST_DISTANCE('POINT(49.47␣11.041)', em.gpscoord) AS dist,
       em.generator_type, em.town, em.tso AS operator
FROM energymap em
WHERE em.gpscoord IS NOT NULL
PREFERRING UPPER(em.town) LIKE '%PLEINFELD%'
           PLUS LOW LOCAL.dist PLUS HIGH em.power_output
ORDER BY em.power_output DESC;
```

Figure 10: Preference with Boolean expression.

5 Experiments

5.1 Test Environment

All experiments for Skyline processing have been performed on a powerful EXASolution cluster [KMZG13]. EXASolution is a parallelized relational DBMS which runs on a cluster of standard hardware servers. Following the *single program, multiple data* (SPMD) model, on each node the identical code is executed simultaneously. The data is stored in a column-oriented way and proprietary in-memory compression methods are used. Furthermore, EXASolution is designed to run in-memory, although data is persistently stored on disc following the ACID rules. EXASolution supports the SQL Standard 2003 and can be integrated via standard interfaces like ODBC, JDBC or ADO.NET. EXASolution runs on EXASOL's own Cluster Operating System (EXACluster OS). It is based on standard Linux and provides functionality for parallel programs. In our tests we used a cluster of 25 Dell PowerEdge R720xd servers which supports 1000 simultaneously executing threads.

5.2 Dataset

In many research papers on traditional Skylines the well-known real-world datasets NBA[2], Household[3], or Zillow[4] are used to examine the performance of preference query process-

[2]www.nba.com
[3]www.ipums.org
[4]www.zillow.com

ing [YSZ09, HV12, TBPY13, EK14]. The NBA dataset is a small 5-dimensional dataset containing 17264 tuples, where each entry records performance statistics for a NBA player. Following [CJT^{+}06], NBA is fairly correlated and therefore is not a very challenging case for Skyline computation. Household is a larger 6-dimensional dataset having 127931 points. The Zillow dataset contains more than 2M entries about real estates in the United States. In a distributed environment synthetic datasets with less than 10M entries are often used to examine the performance of parallel Skyline processing. However, such a small input size is not really a challenge for a high-performance database management system such as EXASolution.

Currently there exists no real world benchmark for Skyline queries which supports large datasets. In order to gain access to realistic data sets with large volumes, we use the dataset which is used in the TPC-H benchmark[5] for analytical databases. The TPC-H benchmark 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.

By using the TPC-H dataset to evaluate Skyline queries, we gain access to data at nearly arbitrary volumes. Data volumes in TPC-H are defined by *scale factors*, where a scale factor of 1 refers to 1 GB of data.

5.3 Queries

In this section we present queries for Preference Analytics based on queries in the TPC-H benchmark but expanded by **PREFERRING** clauses. We used queries having simple preferences (Q0) and complex queries using Pareto or Prioritization as well as partitioning (Q14). All queries are given below.

```
Q0: SELECT p_partkey, p_size, ps_supplycost
    FROM part, partsupp
    WHERE p_partkey = ps_partkey
    PREFERRING LOW ps_supplycost;
```

```
Q1: SELECT p_partkey, p_size, ps_supplycost
    FROM part, partsupp
    WHERE p_partkey = ps_partkey
    PREFERRING LOW ps_supplycost PRIOR TO HIGH p_size;
```

```
Q2: SELECT p_partkey, p_size, ps_supplycost
    FROM part, partsupp
    WHERE p_partkey = ps_partkey
    PREFERRING LOW ps_supplycost PLUS HIGH p_size;
```

[5]http://www.tpc.org/tpch/spec/tpch2.17.0.pdf

Q3:
```
SELECT p_partkey, p_size, ps_supplycost, s_acctbal
FROM part, partsupp, supplier
WHERE p_partkey = ps_partkey AND ps_suppkey = s_suppkey
PREFERRING HIGH p_size PRIOR TO LOW s_acctbal;
```

Q4:
```
SELECT p_partkey, p_size, ps_supplycost, s_acctbal
FROM part, partsupp, supplier
WHERE p_partkey = ps_partkey AND ps_suppkey = s_suppkey
PREFERRING HIGH p_size PLUS LOW s_acctbal;
```

Q5:
```
SELECT p_partkey, p_size, ps_supplycost, s_acctbal
FROM part, partsupp, supplier
WHERE p_partkey = ps_partkey AND ps_suppkey = s_suppkey
PREFERRING (LOW ABS(ps_supplycost - 500) PLUS LOW ABS(s_acctbal))
           PRIOR TO LOW ABS(p_size - 12);
```

Q6:
```
SELECT p_partkey, p_size, ps_supplycost, s_acctbal
FROM part, partsupp, supplier
WHERE p_partkey = ps_partkey AND ps_suppkey = s_suppkey
PREFERRING HIGH p_size PLUS LOW ps_supplycost PLUS LOW s_acctbal;
```

Q7:
```
SELECT p_partkey, p_size, ps_supplycost, s_acctbal, l_orderkey
FROM part, partsupp, supplier, lineitem
WHERE p_partkey = ps_partkey AND ps_suppkey = s_suppkey
        AND l_suppkey = s_suppkey AND l_partkey = p_partkey
PREFERRING (LOW ps_supplycost PLUS LOW s_acctbal PLUS HIGH p_size)
           PRIOR TO HIGH l_quantity;
```

Q8:
```
SELECT p_partkey, p_size, ps_supplycost, s_acctbal,
       l_orderkey, l_quantity
FROM part, partsupp, supplier, lineitem
WHERE p_partkey = ps_partkey and ps_suppkey = s_suppkey
      AND l_suppkey = s_suppkey AND l_partkey = p_partkey
PREFERRING LOW ABS(ps_supplycost - 300) PLUS LOW ABS(s_acctbal)
   PLUS LOW CASE WHEN p_size <= 10 THEN 0
                 ELSE p_size - 10 END
   PLUS LOW CASE WHEN l_quantity >= 20 AND l_quantity <= 30 THEN 0
                 ELSE LEAST(ABS(l_quantity-20),
                            ABS(l_quantity-30)) END;
```

Q9:
```
SELECT p_partkey, p_size, ps_supplycost, s_acctbal, l_orderkey
FROM part, partsupp, supplier, lineitem
WHERE p_partkey = ps_partkey AND ps_suppkey = s_suppkey
      AND l_suppkey = s_suppkey AND l_partkey = p_partkey
PREFERRING LOW ps_supplycost PLUS LOW s_acctbal PLUS
           HIGH p_size PLUS HIGH l_quantity;
```

Q10:
```
SELECT ps_suppkey, p_partkey, p_size, ps_supplycost
FROM part, partsupp where p_partkey = ps_partkey
PREFERRING LOW ps_supplycost PRIOR TO HIGH p_size
   PARTITION BY ps_suppkey;
```

Q11:
```
SELECT p_partkey, p_size, ps_supplycost, s_acctbal
FROM part, partsupp, supplier
WHERE p_partkey = ps_partkey AND ps_suppkey = s_suppkey
PREFERRING (LOW ABS(ps_supplycost - 500) PLUS LOW ABS(s_acctbal))
           PRIOR TO LOW ABS(p_size - 12)
    PARTITION BY s_suppkey;
```

Q12:
```
SELECT ps_suppkey, p_partkey, p_size, ps_supplycost, s_acctbal
FROM part, partsupp, supplier
WHERE p_partkey = ps_partkey AND ps_suppkey = s_suppkey
PREFERRING HIGH p_size PLUS LOW s_acctbal
   PARTITION BY ps_suppkey;
```

Q13:
```
SELECT p_partkey, p_size, ps_supplycost, s_acctbal,
       l_orderkey, l_quantity
FROM part, partsupp, supplier, lineitem
WHERE p_partkey = ps_partkey AND ps_suppkey = s_suppkey
      AND l_suppkey = s_suppkey AND l_partkey = p_partkey
PREFERRING LOW ABS(ps_supplycost - 300) PLUS LOW ABS(s_acctbal)
   PLUS LOW CASE WHEN p_size <= 10 THEN 0 ELSE p_size - 10 END
   PLUS LOW CASE WHEN l_quantity >= 20 AND l_quantity <= 30 THEN 0
            ELSE LEAST( ABS(l_quantity-20), ABS(l_quantity-30)) END
   PARTITION BY s_suppkey;
```

Q14:
```
SELECT ps_suppkey, p_partkey, p_size,
       ps_supplycost, s_acctbal, l_orderkey
FROM part, partsupp, supplier, lineitem
WHERE p_partkey = ps_partkey AND ps_suppkey = s_suppkey
      AND l_suppkey = s_suppkey AND l_partkey = p_partkey
PREFERRING LOW ps_supplycost PLUS LOW s_acctbal
           PLUS HIGH p_size PLUS HIGH l_quantity
    PARTITION BY ps_suppkey;
```

5.4 Results

In Tables 3 - 6 we used the TPC-H dataset with a scaling factor of 1 to 1000, i.e. up to 1000 GB of data. The column titled *join size (rows)* contains the number of rows after the join in the SQL query. The *result size (rows)* denotes the Skyline size, i.e. the number of objects after the preference selection. The last column *elapsed time* represents the runtime of the query in seconds.

Since there is no other commercial implementation of Skyline and other approaches for Preference Analytics (cp. Section 2) are of academic nature, this is the first benchmark of Skylines on large datasets.

Query	**join size (rows)**	**result size (rows)**	**elapsed time (seconds)**
Q0	800 000	6	0.094
Q1	800 000	1	0.049
Q2	800 000	1	0.049
Q3	800 000	3	0.058
Q4	800 000	3	0.059
Q5	800 000	12	0.066
Q6	800 000	33	0.077
Q7	6 001 215	35	0.184
Q8	6 001 215	51	0.197
Q9	6 001 215	73	0.206
Q10	800 000	10 000	1.109
Q11	800 000	10 001	0.695
Q12	800 000	19 646	0.886
Q13	6 001 215	58 119	3.098
Q14	6 001 215	101 997	3.610

Table 3: Benchmark results for TPC-H scale factor 1.

Query	**join size (rows)**	**result size (rows)**	**elapsed time (seconds)**
Q0	8 000 000	91	0.077
Q1	8 000 000	5	0.053
Q2	8 000 000	5	0.056
Q3	8 000 000	1	0.086
Q4	8 000 000	4	0.086
Q5	8 000 000	7	0.093
Q6	8 000 000	53	0.110
Q7	59 986 052	56	0.445
Q8	59 986 052	66	0.405
Q9	59 986 052	140	0.483
Q10	8 000 000	100 001	6.236
Q11	8 000 000	100 001	5.622
Q12	8 000 000	199 263	7.752
Q13	59 986 052	580 554	34.033
Q14	59 986 052	1 000 383	38.765

Table 4: Benchmark results for TPC-H scale factor 10.

Query	join size (rows)	result size (rows)	elapsed time (seconds)
Q0	80 000 000	816	0.216
Q1	80 000 000	20	0.085
Q2	80 000 000	20	0.094
Q3	80 000 000	4	0.364
Q4	80 000 000	4	0.618
Q5	80 000 000	10	0.357
Q6	80 000 000	48	0.377
Q7	600 037 902	49	4.595
Q8	600 037 902	41	5.746
Q9	600 037 902	125	4.856
Q10	80 000 000	1 000 007	55.782
Q11	80 000 000	1 000 024	61.296
Q12	80 000 000	2 000 115	79.933
Q13	600 037 902	5 496 641	350.081
Q14	600 037 902	9 268 943	406.202

Table 5: Benchmark result for TPC-H scale factor 100.

Query	join size (rows)	result size (rows)	elapsed time (seconds)
Q0	800 000 000	8029	1.178
Q1	800 000 000	138	0.342
Q2	800 000 000	138	0.349
Q3	800 000 000	15	2.785
Q4	800 000 000	15	2.618
Q5	800 000 000	2	2.627
Q6	800 000 000	34	2.755
Q7	5 999 989 709	104	52.394
Q8	5 999 989 709	64	57.664
Q9	5 999 989 709	346	64.447
Q10	800 000 000	10 000 079	649.118
Q11	800 000 000	10 000 225	719.784
Q12	800 000 000	20 020 772	924.670
Q13	5 999 989 709	141 220 077	6 899.464
Q14	5 999 989 709	243 220 024	9 620.591

Table 6: Benchmark result for TPC-H scale factor 1000.

Figure 11 presents the runtime for all queries in logarithmic scale for scale factors 1, 10, 100, and 1000 on the TPC-H benchmark dataset.

By examining the results, there is a clear tendency for queries with smaller result sizes to run fast and for queries with larger result sizes (at the same input size) to be slower. A tendency which corresponds to the trade-off described in Section 3.2.

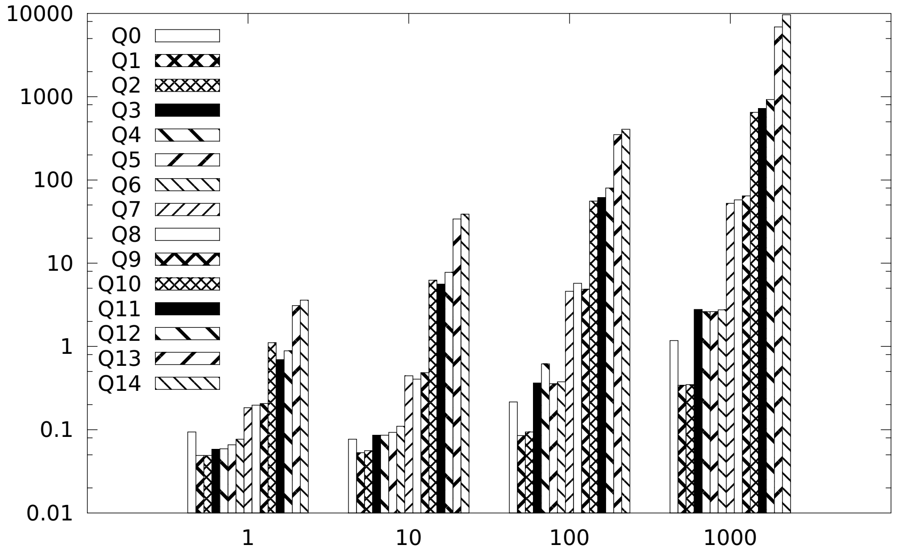

Figure 11: Runtime for Queries 0 to 14 in seconds (logarithmic scale) for scale factors 1, 10, 100, and 1000 on the TPC-H dataset.

6 Conclusion

During the last decade, data are increasingly stored in a distributed way, hence distributed query processing has become an important and challenging problem. On the other hand, the popularity of preference database queries due to their ability to identify sets of highly interesting objects in large databases makes it necessary to unify these two approaches. To tackle the problem of high performance Skyline processing in a distributed environment we integrated preference queries into EXASolution.

This paper provides insight into how Preference Analytics works in EXASolution, a high performance parallel and distributed engine for data analytics and data warehousing. It is the first (and by the time of writing also the only) commercial database that features Skylines natively and at scale. We present novel and expressive operations for preference construction and describe a distributed Skyline algorithm. Furthermore, a quantitative performance analysis is provided. Providing Skylines as an in-database analytical tool enables *Preference Analytics*, which facilitates focusing the results of advanced analytics on best-matching objects only, thereby enabling better and more agile decision-making.

References

[AK12] A. Arvanitis and G. Koutrika. Towards Preference-Aware Relational Databases. In *Proceedings of ICDE '12*, Washington, DC, USA, April 2012.

[BGZ04] W.-T. Balke, U. Güntzer, and J. X. Zheng. Efficient Distributed Skylining for Web Information Systems. In *EDBT '04: Proceedings of the 9th International Conference on Extending Database Technology*, volume 2992 of *LNCS*, pages 256–273, 2004.

[BKS01] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *ICDE '01: Proceedings of the 17th International Conference on Data Engineering*, pages 421–430, Washington, DC, USA, 2001. IEEE Computer Society.

[CCM13] J. Chomicki, P. Ciaccia, and N. Meneghetti. Skyline Queries, Front and Back. *SIGMOD Rec.*, 42(3):6–18, October 2013.

[CDK06] S. Chaudhuri, N. Dalvi, and R. Kaushik. Robust Cardinality and Cost Estimation for Skyline Operator. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 64, Washington, DC, USA, 2006. IEEE Computer Society.

[CJT$^+$06] C. Y. Chang, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On High Dimensional Skylines. In *EDBT '06: Proceedings of the 10th International Conference on Extending Database Technology*, volume 3896 of *LNCS*, pages 478–495, 2006.

[DJ10] X. Ding and H. Jin. Efficient and Progressive Algorithms for Distributed Skyline Queries over Uncertain Data. In *2010 IEEE 30th International Conference on Distributed Computing Systems*, pages 149–158. IEEE, 2010.

[EF09] H. Eder and W. Fang. Evaluation of Skyline Algorithms in PostgreSQL. In *IDEAS '09: Proceedings of the 2009 International Database Engineering & Applications Symposium*, pages 334–337, New York, NY, USA, 2009. ACM.

[EK14] M. Endres and W. Kießling. High Parallel Skyline Computation over Low-Cardinality Domains. In *ADBIS '14: Advances in Databases and Information Systems*, pages 97–111. Springer, 2014.

[End14] M. Endres. A Survey on Selectivity Estimation for Preference Database Queries. In T. Robal H.-M. Haav, A. Kalja, editor, *Frontiers in Artificial Intelligence and Applications (FAIA)*, volume 270, pages 159–172. IOS Press, 2014.

[GRB11] M. Golfarelli, S. Rizzi, and P. Biondi. myOLAP: An Approach to Express and Evaluate OLAP Preferences. *TKDE*, 23(7):1050–1064, 2011.

[HLS06] K. Hose, C. Lemke, and K.-U. Sattler. Processing Relaxed Skylines in PDMS Using Distributed Data Summaries. In *Proceedings of CIKM '06*, pages 425–434, New York, NY, USA, 2006. ACM.

[HV12] K. Hose and A. Vlachou. A Survey of Skyline Processing in Highly Distributed Environments. *The VLDB Journal*, 21(3):359–384, June 2012.

[KEW11] W. Kießling, M. Endres, and F. Wenzel. The Preference SQL System - An Overview. *Bulletin of the Technical Commitee on Data Engineering, IEEE*, 34(2):11–18, 2011.

[Kie02] W. Kießling. Foundations of Preferences in Database Systems. In *Proceedings of VLDB '02*, pages 311–322, Hong Kong, China, 2002. VLDB Endowment.

[KMZG13] S. Klenk, S. Mandl, S. Zentgraf, and M. Golombek. Advanced Analytics mit der analytischen In-Memory Datenbank EXASolution. In *GI-Jahrestagung*, 2013.

[LEMK13] J. J. Levandoski, A. Eldawy, M. F. Mokbel, and M. E. Khalefa. Flexible and Extensible Preference Evaluation in Database Systems. *ACM Trans. Database Syst.*, 38(3):17:1–17:43, September 2013.

[LTL06] H. Li, Q. Tan, and W.-C. Lee. Efficient Progressive Processing of Skyline Queries in Peer-to-Peer Systems. In *Proceedings of the 1st International Conference on Scalable Information Systems*, pages 26–es, New York, NY USA, May 2006. ACM.

[LVDN14] S. Liknes, A. Vlachou, C. Doulkeridis, and K. Nørvåg. APSkyline: Improved Skyline Computation for Multicore Architectures. In *Proc. of DASFAA '14*, LNCS, 2014.

[LYLC06] E. Lo, K. Y. Yip, K.-I. Lin, and D. W. Cheung. Progressive Skylining Over Web-accessible Databases. *IEEE TKDE*, 57(2):122–147, May 2006.

[PKP$^+$09] S. Park, T. Kim, J. Park, J. Kim, and H. Im. Parallel Skyline Computation on Multicore Architectures. In *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 760–771, Washington, DC, USA, 2009. IEEE.

[RJVDN09] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. AGiDS: A Grid-Based Strategy for Distributed Skyline Query Processing. In *Globe '09: Proceedings of the 2nd International Conference on Data Management in Grid and Peer-to-Peer Systems*, Globe '09, pages 12–23, Berlin, Heidelberg, 2009. Springer-Verlag.

[SHZ$^+$10] S. Sun, Z. Huang, H. Zhong, D. Dai, H. Liu, and J. Li. Efficient Monitoring of Skyline Queries over Distributed Data Streams. *Knowl. Inf. Syst.*, 25(3):575–606, 2010.

[SKP11] K. Stefanidis, G. Koutrika, and E. Pitoura. A Survey on Representation, Composition and Application of Preferences in Database Systems. *ACM TODS*, 36(4), 2011.

[SLB10] J. Selke, C. Lofi, and W.-T. Balke. Highly Scalable Multiprocessing Algorithms for Preference-Based Database Retrieval. In *DASFAA '10: 15th International Conference on Database Systems for Advanced Applications*, volume 5982 of *LNCS*, pages 246–260, Tsukuba, Japan, 04/2010 2010. Springer.

[TBPY13] G. Trimponias, I. Bartolini, D. Papadias, and Y. Yang. Skyline Processing on Distributed Vertical Decompositions. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):850–862, 2013.

[VDK08] A. Vlachou, C. Doulkeridis, and Y. Kotidis. Angle-based Space Partitioning for Efficient Parallel Skyline Computation. In *SIGMOD '08: Proceedings of the 34th SIGMOD International Conference on Management of data*, SIGMOD '08, pages 227–238, New York, NY, USA, 2008. ACM.

[VP10] G Valkanas and A N Papadopoulos. Efficient and Adaptive Distributed Skyline Computation. *Lecture Notes in Computer Science*, pages 1–18, April 2010.

[WVO$^+$09] S. Wang, Q. H. Vu, B. C. Ooi, A. K. H. Tung, and L. Xu. Skyframe: A Framework for Skyline Query Processing in Peer-to-Peer Systems. *VLDB J.*, 18(1):345–362, 2009.

[WZF$^+$06] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. E. Abbadi. Parallelizing Skyline Queries for Scalable Distribution. In *EDBT '06: Proceedings of the 10th International Conference on Extending Database Technology*, volume 3896 of *LNCS*, pages 112–130. Springer, 2006.

[XC10] Y.-Y. Xiao and Y.-G. Chen. Efficient Distributed Skyline Queries for Mobile Applications. *Journal of Computer Science and Technology*, 25(3):523–536, 2010.

[YSZ09] L. Zhu Y., S., and Zhou. Distributed Skyline Retrieval with Low Bandwith Consumption. *TKDE*, 2009.