A Classification of Skew Effects in Parallel Database Systems

Holger Märtens

Universität Leipzig, Institut für Informatik, Postfach 920, D–04009 Leipzig, Germany
maertens@informatik.uni-leipzig.de

1 Introduction

A major performance barrier in parallel database systems (PDBS) are **skew effects**, characterized by an uneven distribution of data and/or workload across the system's resources. Despite numerous proposed **load balancing** strategies, this problem is far from solved, partly because there is no well-structured model of the different types of skew, their causes, consequences, and interdependencies, and the methods to combat them.

This paper aims to help understand how to find the appropriate load balancing methods for different forms of skew. We present a classification of skew effects on the one hand and of load balancing approaches on the other, then match the two to find sensible pairs. This will allow us to state why some previous approaches are less successful than they should be and to propose some required capabilities of future algorithms.

Our study is on a purely qualitative level and makes no architectural assumptions. Instead, we focus on the fundamental relationships of different types of skew, with each other and with the various load balancing techniques, to reach general conclusions independent of numerical parameters. We find that highly dynamic scheduling methods based on observed execution times are superior in both complexity and attainable load balance. We also suggest the tuning of database schemata as a new anti-skew measure.

We first discuss some related research in Sect. 2. Sects. 3 and 4 present our classifications of load balancing methods and skew effects, respectively. These are matched in Sect. 5, and Sect. 6 offers our conclusions and outlook on the future. The reader interested in more detail is referred to an extended version of this article [9].

2 Related Work

Skew effects have been widely studied in the literature, and a large number of skew-aware load balancing algorithms have been developed, some of which are quite successful. However, a systematic analysis and classification of skew does not yet exist.

A taxonomy of data skew in parallel joins [14] includes the aspects of intrinsic and partition skew. It also defines redistribution and join product skew similar to ‘plane skew’ and ‘solid skew’ from [15]. [16] uses the notions of single, double, and ‘messy’ skew that intermingle attribute value skew and correlation (cf. Sect. 4). Attribute value distributions have been modeled in many complex ways [1, 3, 10]. They are used with histograms [10] or sampling methods [12] to predict result sizes and processing costs. Some researchers have studied the correlation between value distributions of different attributes [11].

Load balancing has been classified as ‘static’ and ‘dynamic’ with varying definitions [2, 16]. We have further differentiated dynamic methods [7], while [5] distinguished skew avoidance from skew resolution. ‘Adaptive’ query processing was surveyed for

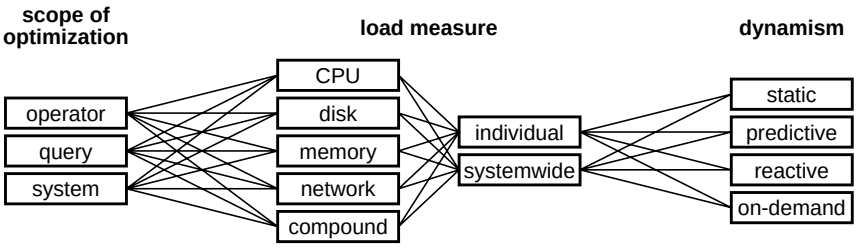

Fig. 1. Overview of load balancing classification

wide-area networks [4]. Load balancing was also classified outside the DBS field [13].

These approaches are commonly limited to subsets of the different skew types and do not capture the complex interactions between them.

3 Classification of Load Balancing Paradigms

We understand load balancing to comprise the following four steps (some of which may be integrated into a single operation):

Load Partitioning. The workload is partitioned into **load units** that have two properties: the **partitioning dimension** reflecting the intended type(s) of parallelism (inter- and intra-transaction, -query, and -operator parallelism), and the **load granule**, i.e., the size of single load units. Thus, load units can range from a single sub-operator to a pipeline of several operators to a batch of multiple transactions. Load partitioning may be predetermined by the **data allocation**, prior processing steps or logical dependencies in the data.

Choice of Degree of Parallelism. The **degree of parallelism (DP)** is primarily determined by the total amount of load and its overall resource demands. Other aspects include the current load situation in the system (both globally and locally) and the relative performance of different resources, e.g., of disks vs. CPUs.

Selection of Processing Nodes. The set of **eligible processors** that can truly process a load unit similarly depends on the load situation, but also on the system architecture (especially for shared-nothing). Ineligible nodes may imply a reduction of the DP.

Load Assignment (Scheduling). Finally (and perhaps most importantly), load units are assigned to processing nodes, determining the final load distribution and balancing. As above, it may be predetermined by data allocation, system architecture, previous processing, or data dependencies. Scheduling primarily equalizes CPU and main memory load but also affects disk and network utilization, e.g., by selecting the order of data access [8].

3.1 Classification

Our classification, summarized in Fig. 1, is tailored to the purpose of matching algorithms with the types of skew they are capable of resolving. It has three main criteria:

The **scope** of optimization is the portion of load that the algorithm can simultaneously oversee, ranging from one operator to one query to (rarely) the entire system. It reflects the types of parallelism and the load partitioning dimensions given above.

The **measure** of system load can refer either to a single type of resource (CPU, memory, disks, network) or to a compound load measure. Furthermore, the load situation can

be regarded either for individual resources or for the overall system.

The **dynamism** of an algorithm denotes the time when load balancing decisions are taken. **Static** methods use, e.g., constant load granules and DPs with random or round-robin load assignment. **Predictive** techniques assign all load in advance for their scope and then strictly execute the resulting schedule [2]. **Reactive** methods use a predictive schedule that is later adapted as needed [5, 6, 15]. **On-demand** algorithms avoid advance planning and assign one load unit at a time as execution proceeds [7, 16]. The latter two are called **runtime** techniques; runtime and predictive methods are labeled **dynamic**.

These criteria are not totally independent of each other. For instance, a static algorithm will not have a complex load measure at runtime, and a 'high' ranking in all categories would pose a prohibitively complex optimization problem. Tuning the schema definition and data allocation may be understood as static actions.

4 Classification of Skew Effects

Our classification is depicted in Fig. 2. Intrinsic, query, and partition skew are summarized as **data skew (DS)** [14], as they all relate to the distribution of (values within) data. Capacity and execution skew refer to the processing performance of the system.

4.1 Intrinsic Skew (IS)

Intrinsic skew broadly denotes an uneven distribution of attribute values within the data:

Attribute value skew (AVS) refers to a single attribute of a single relation.

Correlation skew (CoS) depends on the logical correlation of value distributions from more than one attribute. CoS may span several attributes and can be either **intra-relational** or **inter-relational** (concerning attributes of one or several relations).

Intrinsic skew can occur in both base relations and intermediate results. It does not depend on the storage or processing methods applied and is caused only by the properties of the world modeled and by the schema definition that maps them into the database.

4.2 Query Skew (QS)

This term denotes the bias in query predicates, which normally tend to select certain relations, attributes, or values more frequently than others. Like intrinsic skew, it derives from the logical view of the data independent of the storage or processing approach. Though query skew is not strictly a property of the data, it qualifies as a type of data skew because the queries asked by a user depend of the contents and semantics of the data. Like IS, QS may also partially depend on the definition of the database schema.

4.3 Partition Skew (PS)

This is the type of skew most widely studied in the literature. It is characterized by an uneven distribution of data across physical resources (processors, disks, main memory) and/or load units. We generalize the categorization by Walton et al. [14] as follows:

Tuple placement skew (TPS) concerns the initial distribution of raw input data (i.e., the base relations) across the disks and processing nodes in the system.

Selectivity skew (SS) is defined by varying selectivity rates for sub-scans on different partitions of data.

Redistribution skew (RS) occurs due to different amounts of data being transferred between different processors.

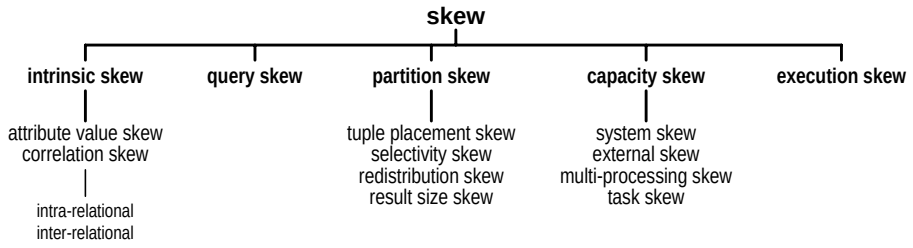

Fig. 2. Overview of skew classification

Result size skew (RSS) denotes disparate result sizes for concurrent load units.

Within queries, TPS and SS refer to load units that scan base relations, whereas RS and RSS can occur only in operations that use intermediate results. This distinction is important because the allocation of base data cannot be changed at query runtime. The different types of PS are related with each other and with intrinsic skew in various ways:

- Tuple placement skew can occur if data allocation is based on attribute values and if the partitioning attribute is burdened with AVS.
- For scans, selectivity skew is likely if a selection involves a partitioning attribute.
- For data redistribution, the distribution key may again refer to the selection attribute from the previous step, or to one having AVS on it, leading to redistribution skew.
- RS alone may lead to result size skew simply because a load unit's output depends on the size of its input. But even with balanced input sizes, result sizes may vary strongly if the redistribution key is referenced by the operation.

If several of these effects occur, they may either amplify or assuage each other. In the last three steps, query skew is also involved through selection and join predicates. All points are also valid for correlated attributes instead of selection or redistribution keys.

4.4 Capacity Skew (CS)

Capacity skew refers to the amount of work a processor is capable of performing and thus comprises those effects whose cause lies outside the data distribution:

System skew (SyS) consists in heterogeneities of the hardware and the operating system, such as different processors, memory sizes, or OS versions.

External skew (ExtS) denotes skew effects due to workloads outside the database system, especially application programs on non-dedicated servers.

Multi-processing skew (MPS) comprises the effects caused within the DBS, but outside the current scope of load balancing. Like ExtS, it can denote a skew in the availability of resources to the load units under consideration.

Task skew (TS) is present when two load units work on equal amounts of data but have different tasks to perform (e.g., a scan and a sort operation, or a simple and a complex query), causing disparate resource consumption.

Both multi-processing and task skew may stem from query skew, as the type and order of queries submitted to the system determine the basic sequence of tasks required.

A query optimizer can develop execution plans that may or may not cause capacity skew. For instance, partial parallelism – though often beneficial to reduce the total workload for small queries – is far more likely than full parallelism to lead to multi-processing skew. MPS will turn into PS for load balancing methods with system scope.

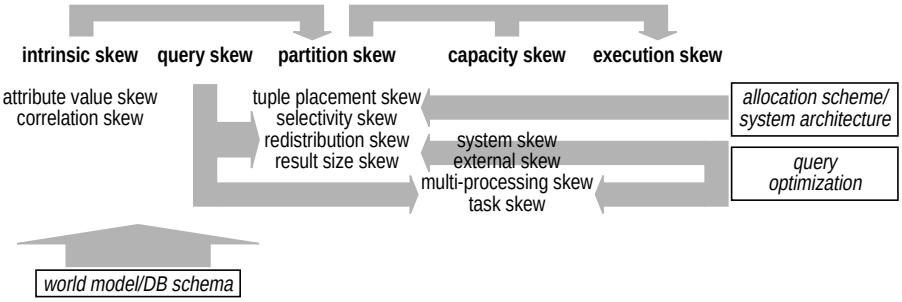

Fig. 3. Overview of skew causes

4.5 Execution Skew (ES)

Execution skew denotes the disparate execution times of concurrent tasks within the database system. Strictly speaking, ES is the only type of skew that is truly a problem for a PDBS. It exists purely as a consequence of other skew types since both the amount of data processed (PS) and the availability of resources (CS) determine the execution speed of a given load unit. The effects of intrinsic and query skew on execution times are indirect.

4.6 Analysis of Causal Relationships

The network of dependencies is illustrated in Fig. 3, which is based on Fig. 2 but enhanced with the causal relationships between the various skew types. The graph allows us to identify three categories of skew types that seem to be the cause of all others:

- intrinsic skew of all shades, due to the properties of the world modeled in the DBS;
- query skew, based on the users' demands;
- system and external skew, which are caused outside the DBMS.

The figure also contains the database and allocation schemata as well as the query optimizer, portrayed as system-inherent components that influence the likelihood of skew at the time of processing. It is these elements in which load balancing can be enacted to combat skew. The system architecture is also shown but must be assumed as fixed.

5 Matching the Classifications

We now compare the two classifications of skew effects and load balancing techniques, proceeding along the categories of skew as given above, in order to analyze which methods are generally capable of alleviating which types of skew. This naturally leads to a list of characteristics that we deem sensible for future load balancing algorithms.

5.1 Intrinsic and Query Skew

As intrinsic and query skew are both caused by the world model itself, they can be treated only by manipulating the database schema. This type of manipulation would be classified as static, with a systemwide scope of optimization. Since it would not be concerned with system load at runtime, the category of measure is irrelevant here.

The chances of tuning the database schema to avoid IS and QS are limited by the inherent biases of the world itself and by the requirements of a ‘natural’ view allowing convenient querying. Still, routine modeling steps such as **(de)normalization** and

materialized view selection might be extended to account for skew effects. We are not aware of any existing solution of this kind.

5.2 Partition Skew

The variety of subclasses of PS can be tackled in different ways.

Tuple placement skew is treated statically during data allocation, using either system or query scope to optimize throughput or response times, respectively. Like in Sect. 5.1, the load measure is irrelevant. Simple approaches try to decluster the raw data in equally sized disk partitions, others aim to balance the I/O expected at runtime. The latter require some knowledge of query skew (e.g., through traces of past activity) and a load measure reflecting individual disks.

Selectivity skew is also remedied by data allocation, with the same classification of algorithms, usually by random or hash-based declustering. Range declustering, on the other hand, *intentionally causes* SS in order to restrict range queries to a subset of disks, avoiding full parallelism that can be inefficient.

Redistribution skew and **result size skew** are often inevitable due to intrinsic skew and thus call for dynamic techniques that may be predictive, reactive, or on-demand. System scope has been implemented only for limited workloads, and most solutions work rather well with query or even operator scope. The load measure should comprise all potential bottlenecks, but actual implementations are often limited to CPU load or use the amount of data as representative for overall resource consumption. In all cases, estimates of future load are notoriously error-prone despite the efforts described in Sect. 2, threatening the success of load balancing [6, 16].

5.3 Capacity Skew

By definition, system, external, and multi-processing skew cannot be *avoided* by any load balancing algorithm. Still, they should be *accounted for* by monitoring resource capacity and routing workloads accordingly. This is easier for SyS than for fluctuating ExtS and MPS. In contrast, task skew can be treated by the optimizer through performance cost models, although inaccuracies will multiply with the errors in size estimates noted in Sect. 5.2. On the whole, capacity skew is far more difficult to handle than partition skew and has mostly been passed over in the literature, with most studies addressing single-user mode only and neglecting the problems of multi-user processing.

In any case, a wide scope of optimization and a comprehensive load measure will benefit the treatment of capacity skew. Most importantly, however, load balancing must be highly dynamic to respond to unforeseen changes in processor capacity.

5.4 Execution Skew

Being ‘only’ the consequence of PS and CS, execution skew might not be considered a problem in its own right, but treating it directly instead of its underlying causes can have several advantages. Specifically, a runtime algorithm (re)assigning single load units based on the progress of execution (cf. Sect. 3.1) could largely do without predictions of data and capacity skew because load balancing would depend on actual resource consumption instead. This would reduce the complexity of load balancing itself by eliminating the work of obtaining cost estimates. More importantly, the workload can be balanced better because in contrast to vague cost estimates, observed execution times are accurate by definition. The benefit increases with more complex and irregular load.

Table 1. Overview of load balancing requirements for different skew types

skew type		*time of correction*	*scope of optimization*	*load measure*	*dynamism*
intrinsic	attribute value	schema design	system	—	static
	correlation				
query		schema design	system	—	static
partition	tuple placement	data allocation	system	individual disks	static
	selectivity				
	redistribution	load balancing (steps 1 – 4)	operator/query/ system	CPU, memory	dynamic
	result size				
capacity	system	—	—	—	—
	external				
	multi-processing				
	task	load balancing (2 – 4)	query	CPU	dynamic
execution		load balancing (2 – 4)	operator/query/ system	compound	runtime

A load balancing technique addressing execution skew is always of the runtime variety. Observed execution times constitute a compound measure including CPU load as well as, for instance, delays caused by disk contention and memory shortage. A large optimization scope may be beneficial by reducing interference between load units.

5.5 Analysis

Our summary of requirements in Table 1 shows a clear analogy between the causal order of skew types and the temporal order of the load balancing steps where they can be amended. The latter in turn mirrors the dynamism needed for the associated anti-skew techniques, revealing that only highly dynamic methods can successfully treat skew in complex environments. This seemingly simple perception challenges the majority of existing load balancing methods. We think most studies have achieved too optimistic results by neglecting aspects such as correlation and capacity skew or multi-user mode.

Surprisingly, dynamic load balancing need not be excessively complex. While static methods require system scope to account for all possible future load, dynamic algorithms can work well on the query or even operator level. Runtime techniques also work with a single, compound, easily gathered load measure (i.e., actual execution times) while others are enhanced with ever more complex measures and estimates.

Since on-demand schemes cannot analyze the exact cause of processing delays, they should be complemented to observe, for instance, certain memory and I/O limits, possibly based on rough load estimates far less complex than in predictive algorithms. Delays can then be assumed as CPU-based and scheduling can proceed accordingly.

Recommendation. For the development of future load balancing algorithms, we suggest the following twofold strategy that combines static and dynamic aspects:

1. Use static methods to prepare the data in such as way as to limit data skew occurring at the time of query processing. This exploits known techniques of data allocation but should also include some skew-aware schema tuning as outlined in Sect. 5.1.
2. In query processing, employ an on-demand load balancing scheme. Use its reduced complexity either to keep the algorithms lean or to allow for a greater optimization scope. Supplement this with aspects like memory restrictions and disk contention.

6 Conclusions and Future Work

In this paper, we developed classifications for both skew effects and load balancing techniques, then compared the two to achieve a conceptual framework for the assessment of existing processing approaches and for the development of new algorithms.

We found that highly dynamic techniques have great advantages with respect to their own complexity as well as to the expected success of load balancing because they rely on observed execution times rather than inaccurate cost estimates. We particularly favor on-demand scheduling methods treating execution skew and strongly recommend to pursue this approach further. In addition, we noted an unexplored potential for skew treatment in the design of database schemata and materialized views. We consider this an interesting line of research for efficient parallel query processing.

Our own future work will primarily concern the development of new, on-demand load balancing methods. In addition, we will continue our quest for suitable data allocation schemes and proceed with our investigations into the nature of skew effects.

References

1. C. M. Chen, N. Roussopoulos: *Adaptive Selectivity Estimation Using Query Feedback*. Proc. ACM SIGMOD Conf., Minneapolis, 1994.
2. H.M.Dewan, M.Hernández, K.W.Mok, S.J.Stolfo: *Predictive Dynamic Load Balancing of Parallel Hash-Joins Over Heterogeneous Processors in the Presence of Data Skew*. Proc. 3rd PDIS Conf., Austin, 1994.
3. C. Faloutsos, Y. Matias, A. Silberschatz: *Modeling skewed distributions using multifractals and the '80-20 law'*. Proc. 22nd VLDB Conf., Bombay, 1996.
4. J. M. Hellerstein, M. J. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, M. A. Shah: *Adaptive Query Processing: Technology in Evolution*. Data Eng. Bulletin 23 (2), 2000.
5. K. A. Hua, C. Lee: *Handling Data Skew in Multiprocessor Database Computers Using Partition Tuning*. Proc. 17th VLDB Conf., Barcelona, 1991.
6. N. Kabra, D. J. DeWitt: *Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans*. Proc. ACM SIGMOD Conf., Seattle, 1998.
7. H. Märtens: *Skew-Insensitive Join Processing in Shared-Disk Database Systems*. Proc. IDPT Conf., Berlin, 1998.
8. H. Märtens: *On Disk Allocation of Intermediate Query Results in Parallel Database Systems*. Proc. Euro-Par Conf., Toulouse, 1999.
9. H. Märtens: *A Classification of Skew Effects in Parallel Database Systems*. Technical report, University of Leipzig, to appear. (Will be available at: `http://dbs.uni-leipzig.de/~maertens`)
10. Y. Matias, J. S. Vitter, M. Wang: *Wavelet-Based Histograms for Selectivity Estimation*. Proc. ACM SIGMOD Conf., Seattle, 1998.
11. V. Poosala, Y. E. Ioannidis: *Selectivity Estimation Without the Attribute Value Independence Assumption*. Proc. 23rd VLDB Conf., Athens, 1997.
12. S. Seshadri, J. F. Naughton: *Sampling Issues in Parallel Database Systems*. Proc. 3rd EDBT Conf., Vienna, 1992.
13. T. Schnekenburger, G. Stellner (eds.): *Dynamic Load Distribution for Parallel Applications*. Teubner, Leipzig, 1997.
14. C. B. Walton, A. G. Dale, R. M. Jenevein: *A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins*. Proc. 17th VLDB Conf., Barcelona, 1991.
15. X. Zhou, M. E. Orlowska: *A Dynamic Approach for Handling Data Skew Problems in Parallel Hash Join Computation*. Proc. IEEE TENCON Conf., Beijing, 1993.
16. X. Zhou, M. E. Orlowska: *Handling Data Skew in Parallel Hash Join Computation Using Two-Phase Scheduling*. Proc. ICA^{3}PP Conf., Brisbane, 1995.