VICTORIA UNIVERSITY of TECHNOLOGY

DEPARTMENT OF COMPUTER AND MATHEMATICAL SCIENCES

Taxonomy of Skew in Parallel Databases

K. Liu, C. H. C. Leung and Y. Jiang

(45 COMP 15)

December, 1994

(AMS : 68P15)

TECHNICAL REPORT

VICTORIA UNIVERSITY OF TECHNOLOGY
(P O BOX 14428) MELBOURNE MAIL CENTRE
MELBOURNE, VICTORIA, 3000
AUSTRALIA

TELEPHONE (03) 688 4249 / 4492
FACSIMILE (03) 688 4050

Footscray Campus

Taxonomy of Skew in Parallel Databases

K. Liu, C. H. C. Leung*, and Y. Jiang

TERABYTE DATABASE GROUP
Department of Computer & Mathematical Sciences
Victoria University of Technology, Melbourne, Australia
**Email: clement@matilda.vut.edu.au*
**Tel: +61 3 688 5020 Fax: +61 3 688 4050*

ABSTRACT

A major obstacle to performance improvement in parallel database processing is the presence of skew, which in extreme cases, can contribute to performance degradation to a level below that of uniprocessing. Here, a new taxonomy of skew for parallel database systems is presented, with attention focused on the relational model operating in a shared-nothing distributed memory environment in SPMD mode. Three types of skews are identified: data skew, load skew, and operations skew. Data skew is an intrinsic property of the attributes related to the naturally occurring replication and clustering of the underlying data values irrespective of how the data are eventually processed. Load skew is related to how the data are allocated to different processing nodes, and is caused by the non-uniform spread of input workload and output across different processors. Operation skew is caused by the combined effects of a number of input relations, which may accumulate and reinforce each another. Load skew may further be distinguished into I/O skew, relational operation skew, and output selectivity skew. These different types of skews are quantitatively analysed and expressions are provided for their evaluation. In addition, performance bounds on best and worst case behaviour are also presented.

1. Introduction

In parallel systems, parallelisation may be applied to either data or programs. It is generally considered that the parallelisation of data offers much greater scope for concurrent operation than the parallelisation of control, since for a large data file, the degree of parallelism through horizontal table fragmentation can be orders of magnitude higher than that for control parallelisation. In addition, the codes for control parallelisation can be significantly more complex than the corresponding serial codes [Thin93]. Due to the relative ease and benefits of data parallelisation, database systems have become important targets for parallel processing. In addition, the demand for increasingly complex and flexible queries contribute to the adoption of high-performance platforms for database processing.

Data parallelism consists of partitioning input relational tables into fragments which are then allocated to different processors to be operated on. Ideally, the partitioning could be done in such a way that the workload of the processors are balanced; that is, the processors are allocated equal sized data fragments, in which case the speedup of query execution can be made to be approximately linearly proportional to the number of processors. However, the linear speedup is hard to achieve because of not only the overheads induced by adding parallel processors, but also the workload skew, since some domain values for a given attribute may occur much more frequently than others. If an unbiased partitioning strategy based on such an attribute is used, the resulting fragments will vary in size, resulting in skewed workloads over the processors. The skewed workload may degrade the speedup since the operation cannot be completed until the heaviest loaded processor finishes. In extreme cases, the heaviest loaded processor is assigned with the entire input relation. The execution time of the operation in this case will be even longer than that of using a single processor because of the additional overheads for coordinating multiple processors.

The problem of load skew in parallel database processing has attracted increasing attention from researchers in recent years. A number of algorithms that take into account the skew factors in relational operations have been proposed [Wolf93a, Kits90, Omie90, Hua91, DeWi92]. Most of these algorithms are designed to improve the existing algorithms by adding some skew handling phases. Moreover, since each of these algorithms is designed to tackle particular skew circumstances, none of them appears to consistently outperform the others.

This paper presents a framework by which various skew handling algorithms may be evaluated and compared. The paper by Walton et al. [Walt91] follows a similar approach. They indicated some issues of data skew, and also classified the data skew effects. However, their approach is less formal and less general than ours. Our work includes three parts: first, the key aspects and issues in skew handling are identified; second, the skew handling approaches are classified and some alternatives are indicated; finally, a model of load skew distribution is presented which may be applied to the evaluation of skew handling effectiveness.

2. Taxonomy

In general, several types of skew may be identified, these are *data skew, load skew,* and *operation skew*. Data skew is a property of the attributes without reference to the mode of processing involved. It is concerned with the clustering and replication of the underlying attribute values. Load skew is related to how the data are allocated to different processors, and is linked to the uneven spread of input workload and output across different nodes. Operation skew signifies the combined skew resulting from the skew effects of a number of input relations.

To provide a concrete illustration of the principles, we make use of the following example. Two relations of the database, *customer* and *sales_order*, are shown in

Tables I and II. For these relations, we may have a natural join on the attribute *customer_no*. The result of the join will have nine tuples. We cluster the two relations by *customer_no* and thus may present the join as shown in Table III (where next to the attribute value is the number of times the value occurs).

2.1 Data Skew

The data skew relates to the intrinsic distribution and occurrences of data values of particular attributes, and its presence is unrelated to whether a single processor or multiprocessors are used to process the data. It is caused by the non-uniform distribution and multiple occurrence of attribute values. When a relational operation is applied to such an attribute, the number of result tuples may vary significantly, as compared to a value distribution without data skew. In the above example, data skew is present in attribute *customer_no* of the relation B, where the tuple value 1002 occurs more frequently than that of 1001. The data skew is inherent in the dataset and solely depends on the database applications. The knowledge about the data skew is essential to study the skew behaviour in parallel databases and to develop truly efficient algorithms for database operations. Data skew could be given *a priori* based on certain distributional assumptions, or by carrying out sampling of actual data values. The skewness of the tuples within the relation can be described in terms of coloured balls in an urn, where the colours correspond to the domain values. The question on how to estimate the number of colours present in the urn on the basis of sampling and knowledge of the total number of balls in the urn has been studied in [Good49]. The parent distribution of known size may be subdivided into an unknown number of mutually exclusive classes. In taking a random sample of n elements without replacement from the population, we can estimate the total number of classes. In other words, the number of domain values of the relation can be worked out from sampling and its cardinality. Therefore, data skew can be described in terms of duplicate values of the partitioning attribute.

Customer (hereafter called **relation A**).

Customer_No	Name	Address
1000	Smith A. B.	ABC, Victoria
1001	White B. C.	BCD, N.S.W.
1002	Naugh.C.D.	CDE, Queensland
1003	Young. D.E.	DEF, South Australia
1004	Hua E.F.	EFG, Tasmania
1005	DeWitt. F.G	FGH, N.T.

Table I

Sales_Order (hereafter called **relation B**).

Sales_Order_No	Customer_No	Order_Status	Date
A101	1005	O.K.	1-1-94
A102	1002	O.K.	2-2-94
A103	1002	O.K.	3-3-94
A104	1002	O.K.	3-3-94
A105	1002	O.K.	3-3-94
A106	1000	O.K.	3-3-94
B102	1002	O.K.	3-4-94
B103	1002	O.K.	3-4-94
D456	1002	O.K.	3-8-94
D478	1002	O.K.	3-9-94
E100	1005	O.K.	4-4-94
E200	1002	O.K.	4-4-94

Table II

Customer Relation (A)	*Sales_Order Relation (B)*	*Join Result (# tuples)*
(1000, 1)	(1000, 1)	1
(1001, 1)	(1001, 0)	0
(1002, 1)	(1002, 9)	9
(1003, 1)	(1003, 0)	0
(1004, 1)	(1004, 0)	0
(1005, 1)	(1005, 2)	2

Table III

Denoting by r the number of tuples in the relation R, and n the sample size. We let x_i signify the number of tuples for each domain value in the samples, and y the estimated number of domain values of relation R. Then the data skew factor T_{DS} is given by

$$T_{DS} = \frac{y}{r} \tag{1}$$

where $0 < T_{DS} \leq 1$, and the lower the data skew factor, the higher is the data skew present in the relation for that particular attribute. The estimated number of domain values y is given by

$$y = \sum_{i=1}^{n} A_i x_i \tag{2}$$

where

$$A_i = 1 - (-1)^i \frac{(r-n+i-1)^{(i)}}{n^{(i)}}, \tag{3}$$

with the notation $m^{(i)} = m(m-1)(m-2) \ldots (m-i+1)$, so that

$$T_{DS} = \frac{1}{r}\left\{\sum_{i=1}^{n}\left[1 - (-1)^i \frac{(r-n+i-1)^{(i)}}{n^{(i)}}\right] x_i\right\}. \tag{4}$$

Generally, more samples will result in better accuracy, but the extra cost of sampling will need to be taken into account. Sampling issues can be found in [Sesh92]. In addition, we note that there is no data skew if the partitioning attribute is the primary key of the relation since $y \approx r$ and $T_{DS} \approx 1$.

2.2 Load Skew

While data skew is unrelated to the processing mode, load skew is directly related to parallel processing and does not exist in the case of conventional uniprocessor operation. When a relational operation is allocated to more than one processor, skewness may occur because of the uneven load distribution. Again consider the join example given above and assume three processors participating in the join. Let the join domain be Customer_No, and tuples of the two input relations are allocated according to this with two values assigned to each processor. Then three tuples (with values 1000 and 1001) will go to the first processor, eleven tuples (with 1002 and 1003) will go to the second, and four tuples (with 1004 and 1005) will go to the third. All three processors receive different volume of tuples with a consequent effect on processing time. This non-uniform load distribution is referred to as load skew. Clearly, the load skew limits processor utilisation and the speedup of the operations.

Since a relational operation involves several processing steps, the load skew may be further classified as follows based on its physical operations:

- *IO load skew (IOLS)*, which is caused by uneven I/O costs imposed for the processors to read input relations into their local memory. The I/O cost of each processor is mainly determined by the size of the fragment(s) of the input relation(s) that is allocated to it. In addition, the access paths of the relations may also affect the I/O cost. A relation with an index usually

involves less I/O costs than that without an index since only the blocks that contains the required tuples will be retrieved.

- *operation load skew (OLS)*, which is determined by non-uniform CPU costs for searching tuples that belongs to the result relation among the processors. Therefore, OLS depends not only on the partitioning of the input relations, but also on the processing algorithms that are used at different processors. For example, when one of the input relations is small or has index, the nested-loop join involves the smallest cost; otherwise the hash join may perform better than the nested-loop join.

- *result load skew (RLS)*, which refers to the skewed loads for the processors generating the result of the operation. In the case of RLS, some processors may generate a large number of result tuples while others may only have a few, leading to different CPU costs. RLS depends on the selectivity factor of the operation over the fragments allocated to the processors.

Figure 2.1 shows the three different load skews over some common relational operations and associated processing methods. In Figure 2.1, the following assumptions are made.

The input relations are R and S, with $r = |R|$, $s = |S|$, and $r < s$.
The length of predicates and the number of columns to be projected are one;
Sel and Join_Sel are the selectivities for Selection and Joining;
To save memory, hash-based join algorithms build hash table with the smaller input relation;
Memory size is large enough to hold entirely either R or S
For set theoretic operations (Union, Intersection, and Difference), the two input relations (R and S) are compatible with $R \subset S$.

We primarily focus on a shared-nothing distributed memory architecture. In such an architecture, the total load skew may be obtained as follows. We let N denote the number of nodes, where a node includes the I/O as well as processor components. In this study, we assume that there is a single processor to each node. In the general case, a node may consist of a collection of heterogeneous processors operating on a shared memory basis. The following notations are used.

r_i: the number of tuples in the ith node after the partitioning of relation R
$r \geq r_i \geq 0,\ i = 1, ..., N.$

σ_R: selectivity factor of the relation R

σ_i: selectivity factor of the ith fragment of the relation R after partitioning

R_i: total load of the ith node of relation R

R_{Skew}: the maximum imbalanced load associated with relation R

T_{LS}: load skew factor $(0 \leq T_{LS})$.

The total output from an operation is

$$r\sigma_R = \sum_{i=1}^{N} (r_i \sigma_i) = r_1\sigma_1 + r_2\sigma_2 + \ldots + r_N\sigma_N .$$

Now, the workload of each node consists of three components

W_1: loading cost for each tuple (including disk access time and transfer time)

W_2: processing cost for each tuple (mainly comparison and computation time as reading time from RAM should be comparatively small)

W_3: writing cost for each tuple.

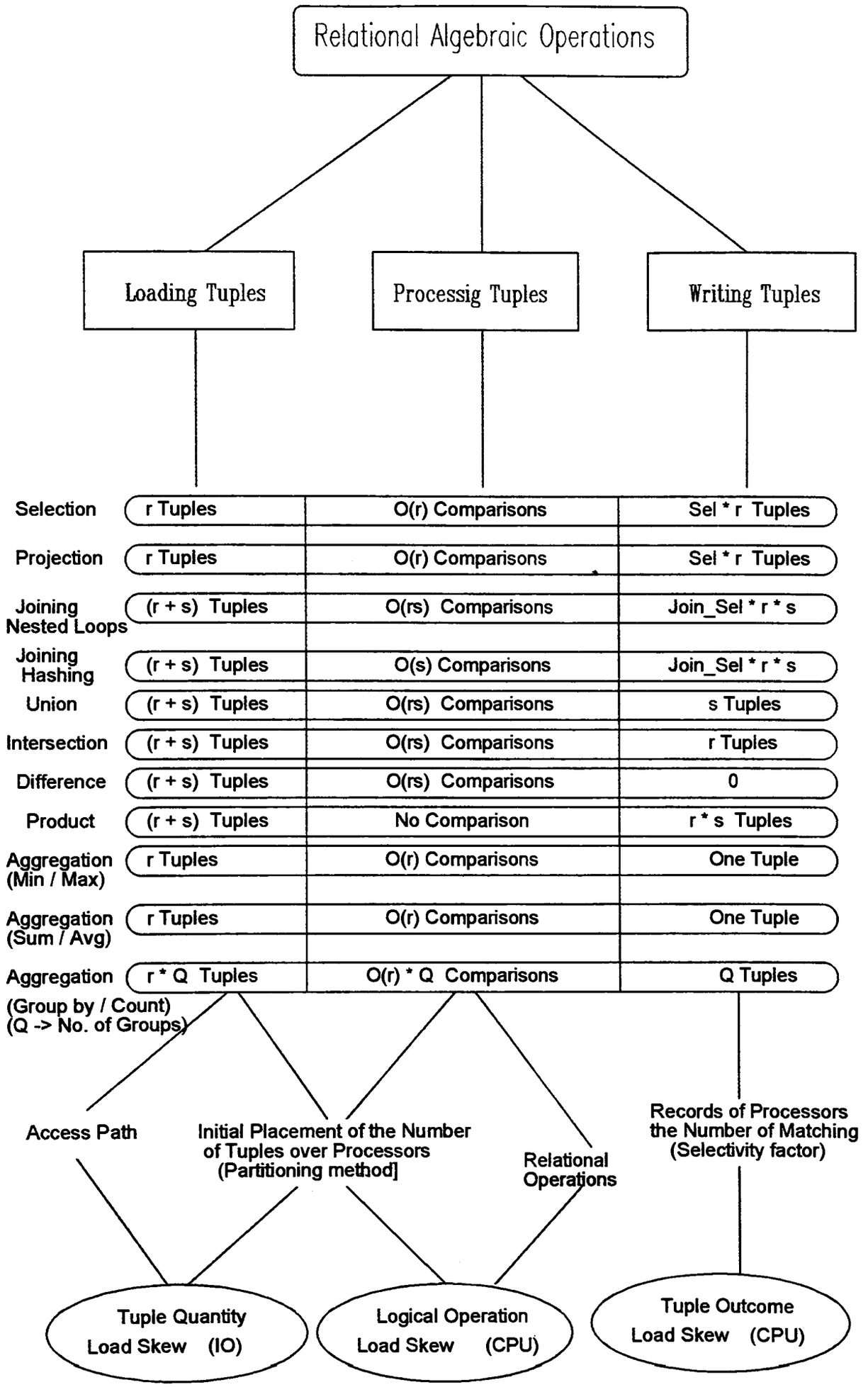
Relational Algebraic Operations
Loading Tuples
Processig Tuples
Writing Tuples
Selection
r Tuples
O(r) Comparisons
Sel * r Tuples
Projection
r Tuples
O(r) Comparisons
Sel * r Tuples
Joining Nested Loops
(r + s) Tuples
O(rs) Comparisons
Join_Sel * r * s
Joining Hashing
(r + s) Tuples
O(s) Comparisons
Join_Sel * r * s
Union
(r + s) Tuples
O(rs) Comparisons
s Tuples
Intersection
(r + s) Tuples
O(rs) Comparisons
r Tuples
Difference
(r + s) Tuples
O(rs) Comparisons
0
Product
(r + s) Tuples
No Comparison
r * s Tuples
Aggregation (Min / Max)
r Tuples
O(r) Comparisons
One Tuple
Aggregation (Sum / Avg)
r Tuples
O(r) Comparisons
One Tuple
Aggregation (Group by / Count) (Q -> No. of Groups)
r * Q Tuples
O(r) * Q Comparisons
Q Tuples
Access Path
Initial Placement of the Number of Tuples over Processors (Partitioning method]
Relational Operations
Records of Processors the Number of Matching (Selectivity factor)
Tuple Quantity Load Skew (IO)
Logical Operation Load Skew (CPU)
Tuple Outcome Load Skew (CPU)

Figure 2.1

A high IOLS does not necessarily mean high total load skew as processing and writing cost (OLS and RLS) need to be taken into account. The total load is the weighted arithmetic mean

$$R_i = \frac{W_1 r_i + W_2 r_i + W_3 r_i \sigma_i}{W_1 + W_2 + W_3} \quad , \tag{5}$$

and the total load imbalance as represented by the deviation from the perfectly balanced situation is given by

$$R_{Skew} = \max(R_1, \ldots, R_N) - \frac{1}{N}\left(\sum_{i=1}^{N} R_i\right) \quad . \tag{6}$$

The load skew factor is given by

$$T_{LS} = \frac{\max(R_1, \ldots, R_N)}{\frac{1}{N}\left(\sum_{i=1}^{N} R_i\right)} \quad , \tag{7}$$

where $1 \leq T_{LS} \leq N$, and the larger this factor, the higher the load skew occurs over the nodes. For a shared-nothing architecture, since load skew is caused by non-uniform placement of data over processors, the methods which partition the input relation(s) onto the nodes will significantly contribute to the load skew. However, a partitioning method may not be the sole cause of all the above load skews. For example, the round-robin partitioning for a unary operation such as selection, does not involve IOLS and OLS since the tuples of the input relation(s) are evenly spread over the processors. However, hash join partitioning would involve all types of load skews. In summary, the IOLS would be caused by allocating the tuples with the identical join attribute values to the same processors in the presence of data skew. The various relational algebraic operations and the existence of the data skew create

OLS. In comparison, the RLS should only occur when the number of matching records of processors are different, and depends on selectivity factor.

2.3 Operation Skew

Given a relational operation, skew may also be classified into no operation skew (NOS), single operation skew (SOS), and double operation skew (DOS) depending on how many input relations are skewed (i.e. the *skew dimension*) after partitioning. Multiple operations can always be regarded as a series of binary operations. If the tuples of both operands are evenly allocated over the processors, it is NOS. The SOS indicates that one input relation has load skew in the attribute(s) related to the operation, whereas the DOS relates to the load skew on both input relations for binary operations, such as join. Evidently, unary relational operations, such as selection and projection, may have either SOS or NOS, while binary operations are likely to have any one of them. In our example, there is only a SOS (due to relation *B*) in the join operation described above.

It is worth noting that DOS is complex but *normally* only appears in the operations which use the intermediate results of other operations. For the base relations, it is pointed out in [Elma94] that when a binary 1:1 or 1:*N* relationship type is involved, a single join operation is usually needed; For a binary *M*:*N* relationship type, two join operations are needed. For example, joining Sales_Order and Product is actually doing two joins, Sales_Order and Order_Product (join attribute is Sales_Order_No), and Order-Product and Product (join attribute is Product_No). Therefore, it is not a join of DOS but two joins of SOS. Furthermore, duplicate attributes (except foreign keys) are avoided in the database design because they cause anomalies (deletion, insertion, and update). Therefore, the operations on the normalised database relations does not often involve DOS.

If we adopt the following notations

s_i: the total load of the ith processor after partitioning relation S

u_R: the average load associated with relation R over N nodes

$$u_R = \frac{1}{N}\left(\sum_{i=1}^{N} R_i\right)$$

u_S: the average load associated with relation S over N nodes

$$u_S = \frac{1}{N}\left(\sum_{i=1}^{N} S_i\right)$$

S_{Skew}: the maximum imbalanced load associated with relation S

T_{OS}: operation skew factor,

then for unary operations such as selection (relation R), we have

NOS: 1 for each processor

SOS: $T_{OS} = T_{LS}$ (same as load skew);

and for binary operations such as Nested-Loops Join (relation R and S), we have

NOS: 1 for each processor

SOS: $T_{LS} \times 1 = T_{LS}$

DOS: $\max(R_1 S_1, \ldots, R_N S_N)/(u_R\ u_S)$.

That is,

$$T_{OS} = \begin{cases} \dfrac{\max(R_i)}{u_R}, & \text{unary operations} \\ \dfrac{\max(R_i S_i)}{u_R u_S}, & \text{binary operations} \end{cases} \qquad (8)$$

where $T_{OS} \geq 1$, and the larger this factor, the higher is the operation skew over the nodes.

3. Performance Degradation

Data skew, load skew, and operation skew are not isolated but closely related. Their relationships are presented in Figure 3.1. Data skew cannot usually be changed,

whereas load skew and operation skew could be avoided using various skew management approaches (see Section 4).

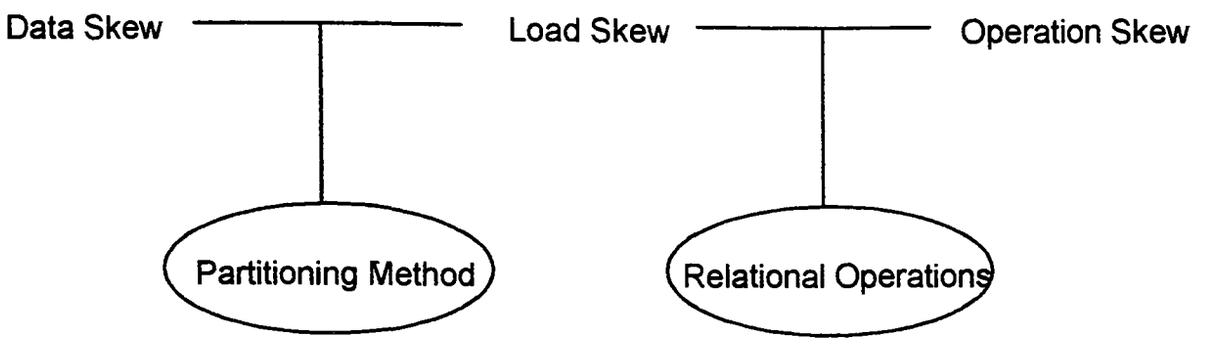

Figure 3.1

In equation (5), the parameter r_i is the tuple allocation over processors, and clearly data skew has significant influence on r_i. Supposing there is *no data skew,* load skew can still occur because of uneven partitioning and variation of the selectivity factor. Assuming a perfect hash function is employed in a situation where data skew is absent, we obtain the following bounds

Lower Bound:

$$\sigma_i r_i = \frac{r\sigma_R}{N}$$

$$R_{Skew} = 0$$

Upper Bound:

$$\sigma_i r_i = r_i$$

$$R_{Skew} = \frac{r_i(W_1 + W_2 + W_3\sigma_i)}{W_1 + W_2 + W_3} - \frac{1}{N}\left(\sum_{i=1}^{N} R_i\right)$$

$$= \frac{r}{N} - \frac{1}{N}\left(\sum_{i=1}^{N} R_i\right),$$

where the last equality is obtained by noting that $r_i = r/N$. The question arise is "If there is *data skew,* what effect does data skew have on load skew?". Our answer is data skew affects load skew through partitioning method. Round-robin, range partitioning and hashing are the three most common partitioning strategies. Among

them, in terms of load balancing, round-robin is the best policy as it destroys the effect of data skew entirely if it exists. Unfortunately, round-robin policy can only be feasibly adopted with unary operations, if significant inter-node communication is to be avoided. On the other hand, data skew has influence on range partitioning and hashing, and in most cases, data skew exacerbates load skew except where proper splitting function is employed in which case data skew can offset the impact of load skew.

Skew degrades the system performance in a number of ways. It causes bucket overflow in parallel processors with shared-nothing or shared-disk architecture. If a highly skewed relation is partitioned into N parts without any load balancing mechanism, it is possible for one processor get all the work while all other processors have nothing in the extreme case. The individual bucket (memory such as RAM) of the processors may not have enough space to store the entire relation, resulting in bucket overflow, in which case it may be necessary to re-partition the relation.

Skew also causes load imbalance by directing different tuple volumes to different processors. The most lightly loaded processor has to wait till the heaviest loaded one finishes. Thus scaleup and speedup cannot achieve the expected linear results, and it is possible that system performance may be even worse than that of the uniprocessor case since the use of multiple processors involves extra overheads.

Skew may also cause problem on the network and connecting processors. If one processor is over utilised, the corresponding I/O operations and the CPU time for the processor outweigh others. Clearly, disk I/O is a particular problem in database systems because of its slow access time, typically 10,000 times of main memory access time. In addition, various processors must exchange information such as synchronisation data, concurrency control messages and some of the intermediate

results. Eventually, the heavily loaded processor becomes the bottleneck (hot spot) of the system and can even cause congestion of the communication network.

4. Skew Management

Skew management involves two basic steps, skew estimation and skew handling. Skew estimation attempts to identify the existence of the skew among parallel processors and, if exist, estimate the degree of the skew. Based on the knowledge of the skew, the skew handling procedure then attempts to avoid or solve skewed load distribution using various approaches.

4.1 Skew Estimation

Parametric estimation vs. non-parametric estimation

Both parametric and non-parametric estimation apply statistical methods to estimate the skew. When the distribution of the skew is known or somehow predictable, the parametric method can be used to determine which processors would be heavily loaded or lightly loaded. Zipf distribution has been widely adapted for skew estimation [Wolf93a, Kell91]. However, if the skew distribution is not known, non-parametric method which relies on profiles of the data collected experimentally may be used. The histogram is the most known example of this method [Mura88]. In the histogram method, the frequency distribution of domain values are stored in the database and are used to determine the upper bound and lower bound of the workload over the processors. Other estimation methods that falls in these two categories may be found in [Seli79, Sun93, Ioan93].

Sampling

Sampling is another general estimation method. It has been shown to be efficient in estimating query result size, and has been applied to multi-processor systems [Sesh92]. By taking samples from the relations/fragments to be allocated, the skewness of the loads may also be calculated by various standard sampling methods

such as random sampling and stratified sampling, from which the heaviest, the average, and the most lightly loaded processors can be determined. As compared to the other estimation methods, sampling is usually more accurate, especially for complex applications, but could involve significant costs.

4.2 Skew Handling

Static skew handling

Static skew handling aims at avoiding non-uniform load distribution of an operation over processors before the operation begins. Depending on the degree of skewness, the algorithms are mainly concerned with handling either high skew or low skew. The high skew handling algorithms often involve a scheduling phase after the input relation(s) is partitioned into fragments. If some fragments are larger than the others due to skew, more processors will be assigned to them such that the execution time of the processors remains uniform [Leun93, Omie91, Wolf93a, Wolf93b]. In contrast, the low skew handling algorithms often involve a tuning of bucket sizes before the buckets are spread to the processors equally [Kits90, Hua91]. The high skew and low skew handling algorithms may be combined to form hybrid algorithms. The idea behind hybrid algorithms is that no single algorithm proposed always outperforms the others, and therefore selection of the proper algorithm to handle a certain degree of the skew would be beneficial. In a hybrid algorithm proposed by DeWitt *et al.*, a sampling phase is first applied to estimate the skewness, followed by selection of the best join methods from five alternatives, i.e. hybrid hash, simple range partitioning, weighted range partitioning, virtual processor partitioning and round-robin [DeWi92].

Dynamic skew handling

Unlike static skew handling, the dynamic skew handling attempts to either allocate workload dynamically during operation execution or re-allocate the workload from heavily loaded processors to lightly loaded ones [Lu92, Hua91]. In other words, it is

waiting for skew occurrence and then striving to solve skew during execution. In dynamic load allocation, the number of buckets used for relation partitioning is much larger than the number of processors. Each processor is assigned with one bucket at a time for execution, with the rest of the buckets put in a waiting queue. Once a processor finishes, it takes another bucket from the queue for execution until all buckets on the queue have been processed. In dynamic load re-allocation, in contrast, the workload is first partitioned and allocated to the processors. The skew is then monitored during the execution. If the skew is found to rise above a certain threshold, the processor with the heaviest load will split its load and migrate some load to other processors.

The static skew handling method approaches the problem before dividing the total workload. It is normally feasible for simple operations such as unary operation and binary join. However, it will become very complicated as the number of processors increases and in situations where multiple queries with multiple operations are involved. The static handling method will need to perform estimation for every input relations and intermediate result relations. As it is not always possible to estimate precisely the host of parameters, it often leads to unrealistic bounds on the results.

The dynamic skew handling approaches the problem at run time by waiting for skew occurrence and resolving the skew. Generally, it does not require preprocessing and relies on no distribution assumptions. However, taking overheads into account, the extent of performance gain of this approach is not always certain with the result that few systems will take the risk of adding in extra complexity and cost to the existing system through the deployment of these algorithms.

Most of the existing skew handling algorithms can not dealing with no skew case properly with adding in significant overheads. Another problem is skews stochastic

feature. Our suggestion goes to skew estimation. Therefore, we believe skew estimation should be emphasised along with skew handling. Then, viewing the system as a whole, the workload can be further balanced and the system performance can be further improved.

5. Summary and Conclusion

We have provided a new taxonomy of skew for parallel database systems, with attention focused on the relational model operating in a shared-nothing distributed memory architecture in SPMD mode. Three types of skews are identified: data skew, load skew, and operations skew. Data skew is an intrinsic property of the attributes related to the naturally occurring replication and clustering of the underlying data values irrespective of how the data are processed. Load skew is related to how the data are allocated to different processing nodes, and is caused by the non-uniform spread of input workload and output across different processors. Operation skew is caused by the combined effects of a number of input relations, which may reinforce or neutralise each another. Load skew may further be distinguished into I/O skew, relational operation skew, and output selectivity skew. These different types of skews have been quantitatively analysed and expressions are provided for their evaluation. In addition, performance bounds on best and worst case behaviour are also derived.

References

[DeWi90] DeWitt D. J., Ghandeharizadeh S., Schneider D. A., Bricker A., Hsiao H. I., and Rasmussen R., The Gamma Database Machine Project, *IEEE Transactions On Knowledge and Data Engineering*, Vol. 2, No. 1, March 1990.

[DeWi92] DeWitt D.J., Naughton J.F., Schneider D.A., Seshadri S., Practical Skew Handling in Parallel Joins, *Proceedings of the eighteenth International Conference on Very Large Data Bases*, Vancouver, British Columbia, Canada 1992.

[Elma94] Elmasri R., Navathe S.B., *Fundamentals of Database Systems.* Benjamin/Cummings, 1994.

[Good49] Goodman B. L. A., On the Estimating of the Number of Classes in a Population, *Annals of Mathematical Statistics*, Volume XX, 1949, pp 572-579

[Hua91] Hua K. A., Lee C., Handling Data Skew in Multiprocessor Database Computers Using Partition Tuning, In *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, September, 1991, pp 525-535

[Ioan93] Ioannidis Y. E., Christodoulakis S., Optimal Histograms for Limiting Worst-Case Error Propagation in the Size of Join Results, *ACM Transactions on Database Systems*, No.4, Dec 1993.

[Kell91] Keller A. M., Roy S., Adaptive Parallel Hash Join in Main-Memory Databases, *Proceedings of the first International Conference on Parallel and Distributed Information Systems*, Dec 1991.

[Kits90] Kitsuregawa M., Ogawa Y., A New Parallel Hash Join Method with Robustness for Data Skew in Super Database Computer (SDC), *Proceedings of the sixteenth International Conference on Very Large Data Bases*, 1990, pp. 210-221.

[Leun93] Leung, C. H. C. and H. T. Ghogomu A high-performance parallel database architecture, *Proc. 7th ACM International Conference on Supercomputing*, Tokyo, July 1993, pp. 377-386.

[Mura88] Muralikrishna M., DeWitt D. J., Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries, *Proceedings SIGMOD International Conference on Management of Data*, Chicago Illinois, June 1988.

[Omie91] Omiecinski E., Performance Analysis of a Local Balancing relational hash-join algorithm for a main-memory databases, *Proceedings of the first International Conference on Parallel and Distributed Information Systems*, Dec 1991, pp. 58-67.

[Seli79] Selinger P.G., Astrashan M.M., Chamberlin D.D., Lorie R.A., Price T.G., Access path selection in a relational database management system,

Proceedings of the ACM SIGMOD International Conference on the Management of Data, Boston, Mass., June, 1979, pp. 23-34.

[Sesh92] Seshadri S., Naughton J.F., Sampling Issues in Parallel Database Systems, *Advances in Database Technology – EDBT'92, 3rd International Conference on Extending Database Technology*, Vienna, Austria, March, 1992 Proceedings, Springer-Verlag.

[Sun93] Sun W., Ling Y. B., Rishe N., and Deng Y., An Instant and Accurate Size Estimation Method for Joins and Selection in a Retrieval-Intensive Environment, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washinton D. C., May 1993.

[Thin93] Thinking Machines Corporation, *Connection Machine CM-5 Technical Summary*. 1993.

[Walt91] Walton B., Dale A., and Jenevein R., A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins. In *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, September, 1991, pp. 537-548.

[Wolf93a] Wolf J. L., Dias D. M., and Yu P. S., A Parallel Sort-Merge Join Algorithm for Managing Data Skew, *IEEE Transactions On Parallel And Distributed Systems*, Vol. 4, No. 1, January 1993.

[Wolf93b] Wolf J. L., Yu P. S., Turek J. and Dias D. M., A Parallel Hash Join Algorithm for Managing Data Skew, *IEEE Transactions On Parallel and Distributed Systems*, Vol.4, No. 12, December 1993.