**Data Partitioning Strategies in Parallel Database Systems**

Partitioning the tables/databases is very important step in parallelizing the database activities. By partitioning the data equally into many different processors’ workload, we can achieve better performance (better parallelism) of the whole system. We have the following types of fragmentation (partitioning) techniques in Parallel database;

**Horizontal Fragmentation** – Partitioning the tables using the conditions specified through WHERE clause of SQL query to distribute bunch of tuples (records). For example, consider the following schema;

STUDENT (Regno, SName, Address, Branch, Phone)

In this table, Branch column is used to store the academic branch in which the student admitted in. suppose, there are various branches like ‘BTech CIVIL’, ‘BTech MECH’, ‘BTech CSE’, and so on, then the following query would be used to partition the table using horizontal partitioning.

SELECT \* FROM student WHERE Branch = branch name;

In the result, there won’t be any changes in the schema of the table; i.e the structure of the table is unaltered. Only, ‘BTech CIVIL’ students fragmented into one partition, ‘BTech CSE’ students fragmented into one partition and so on.

**Vertical Fragmentation** – Partitioning tables using the decomposition rules to break and distribute the tables into multiple partitions vertically (different schemas) is called Vertical fragmentation.

For example, if we would like to break STUDENT into different tables like STUDENT(Regno, SName, Address, Branch) and STU\_PHONE(Regno, Phone), i.e into two different schema on columns is Vertical partitioning.

Among the above discussed fragmentation techniques, partitioning any relation with respect to Parallel databases involves Horizontal Partitioning. Horizontal data partition helps us to distribute the data into several processors to execute queries on them simultaneously.

**Partitioning Strategies:**

There are various partitioning strategies proposed to manage the data distribution into multiple processors evenly.

Let us assume that in our parallel database system we have n processors P0, P1, P2, …, Pn-1 and n disks D0, D1, D2, …, Dn-1 where we partition our data. The value of n is chosen according to the degree of parallelism required. The partitioning strategies are,

**Round-Robin Partitioning**

It is the simplest form of partitioning strategy. Here, the data are distributed into various disks in the fashion, first record into first disk, second record into second disk, and so on. If the number of available disks n is 10, then first record goes to D1 (1 mod 10 = 1), second record goes to D2 (2 mod 10 =2), and so on and 10th record goes to D0 (10 mod 10 = 0), 11th record goes to D1 (11 mod 10 = 1). This scheme distributes data evenly in all the disks.

**Hash Partitioning**

This strategy identifies one or more attributes as partitioning attributes (which is not required in Round-Robin partitioning). We need a hash function which is carefully chosen (careful to avoid data skewness) which takes the identified partitioning attributes as input to hash function.

For example, consider the following table;

**EMPLOYEE(ENo, EName, DeptNo, Salary, Age)**

If we choose *DeptNo* attribute as the partitioning attribute and if we have 10 disks to distribute the data, then the following would be a hash function;

**h(DeptNo) = DeptNo mod 10**

If we have 10 departments, then according to the hash function, all the employees of department 1 will go into disk 1, department 2 to disk 2 and so on.

As another example, if we choose the *EName* of the employees as partitioning attribute, then we could have the following hash function;

**h(EName) = (Sum of ASCII value of every character in the name) mod n**,

where n is the number of disks/partitions needed.

**Range Partitioning**

In Range Partitioning we identify one or more attributes as partitioning attributes. Then we choose a range partition vector to partition the table into n disks. The vector is the values present in the partitioning attribute.

For example, for the EMPLOYEE relation given above, if the partitioning attribute is *Salary*, then the vector would be one as follows;

[5000, 15000, 30000],

where every value means the individual range of salaries. That is, 5000 represents the first range (0 – 5000), 15000 represents the range (5001 – 15000), 30000 represents the third range (15001 – 30000), and it includes the final range which is (30001 – rest). Hence, the vector with 3 values represents 4 disks/partitions.

The above discussed partition strategies must be chosen carefully according to the workload of your parallel database system. The workload may involve many components like, which attribute is frequently used in any queries as filtering condition, the number of records in a table, the size of the database, approximate number of incoming queries in a day, etc.

**Detailed Example:**

Let us start with the following table **Emp\_table**. Emp\_table instance has 14 records and every record stores information about the name of the employee; his/her work grade, and the department name. Assume that we have 3 processors namely P0, P1, P2, and 3 Disks associated with those 3 processors namely D0, D1, D2.

| **Emp\_table** | | |
| --- | --- | --- |
| **ENAME** | **GRADE** | **DNAME** |
| SMITH | 1 | RESEARCH |
| BLAKE | 4 | SALES |
| FORD | 4 | RESEARCH |
| KING | 5 | ACCOUNTING |
| SCOTT | 4 | RESEARCH |
| MILLER | 2 | ACCOUNTING |
| TURNER | 3 | SALES |
| WARD | 2 | SALES |
| MARTIN | 2 | SALES |
| ADAMS | 1 | RESEARCH |
| JONES | 4 | RESEARCH |
| JAMES | 1 | SALES |
| CLARK | 4 | ACCOUNTING |
| ALLEN | 3 | SALES |

**Table 1 – Emp\_table**

**A. Round-Robin Partitioning:**

In this strategy we partition records in a round-robin manner using the function **i mod n**, where i is the record position in the table and n is the number of partitions/disks which is in our case 3. On the application of partitioning technique, first record goes into D1, second record goes into D2, third record goes into D0, fourth record goes into D1, and so on. After distribution of records, we will get the following partitions;

| **Emp\_table\_Partition0** | | |
| --- | --- | --- |
| **ENAME** | **GRADE** | **DNAME** |
| FORD | 4 | RESEARCH |
| MILLER | 2 | ACCOUNTING |
| MARTIN | 2 | SALES |
| JAMES | 1 | SALES |

**Table 2 – Records 3, 6, 9, 12 mod 3**

| **Emp\_table\_Partition1** | | |
| --- | --- | --- |
| **ENAME** | **GRADE** | **DNAME** |
| SMITH | 1 | RESEARCH |
| KING | 5 | ACCOUNTING |
| TURNER | 3 | SALES |
| ADAMS | 1 | RESEARCH |
| CLARK | 4 | ACCOUNTING |

**Table 3 – Records 1, 4, 7, 10, 13 mod 3**

| **Emp\_table\_Partition2** | | |
| --- | --- | --- |
| **ENAME** | **GRADE** | **DNAME** |
| BLAKE | 4 | SALES |
| SCOTT | 4 | RESEARCH |
| WARD | 2 | SALES |
| JONES | 4 | RESEARCH |
| ALLEN | 3 | SALES |

**Table 4 – Records 2, 5, 8, 11, 14 mod 3**

**B. Hash Partitioning:**

Let us take *GRADE* attribute of the Emp\_table to explain Hash partitioning. Let us choose a hash function as follows;

**h(GRADE) = (GRADE mod n)**

where GRADE is the value of *GRADE* attribute of a record and n is number of partitions which is 3 in our case. While applying the hash partitioning on GRADE, we will get the following partitions of Emp\_table. For example, the GRADE of ‘Smith’ is 1 and while hashing the function shows partition 1 (i.e 1 mod 3 = 1). The GRADE of ‘Blake’ is 4, then (4 mod 3) directs to partition 1. The GRADE of ‘King’ is 5 which directs to partition 2 (5 mod 3 = 2).

| **Emp\_table\_Partition0** | | |
| --- | --- | --- |
| **ENAME** | **GRADE** | **DNAME** |
| TURNER | 3 | SALES |
| ALLEN | 3 | SALES |

**Table 5 – GRADEs 3 mod 3**

| **Emp\_table\_Partition1** | | |
| --- | --- | --- |
| **ENAME** | **GRADE** | **DNAME** |
| SMITH | 1 | RESEARCH |
| BLAKE | 4 | SALES |
| FORD | 4 | RESEARCH |
| SCOTT | 4 | RESEARCH |
| ADAMS | 1 | RESEARCH |
| JONES | 4 | RESEARCH |
| JAMES | 1 | SALES |
| CLARK | 4 | ACCOUNTING |

**Table 6 – GRADEs 1, 4 mod 3**

| **Emp\_table\_Partition2** | | |
| --- | --- | --- |
| **ENAME** | **GRADE** | **DNAME** |
| KING | 5 | ACCOUNTING |
| MILLER | 2 | ACCOUNTING |
| WARD | 2 | SALES |
| MARTIN | 2 | SALES |

**Table 7 – GRADEs 2, 5 mod 3**

**C. Range Partitioning:**

Let us consider *GRADE* of Emp\_table to partition under range partitioning. For applying range partition, we need to first identify partitioning vector, [v0, v1, …, vn-2]. Let us choose the following vector as range partitioning vector for our case;

**[2, 4]**

According to the vector, the records having the *GRADE* value 2 and less will go into partition 0, greater than 2 and less than or equal to 4 will go into partition 1, and all the other values (greater than 4) will go into partition 2 as depicted in the following tables.

| **Emp\_table\_Partition0** | | |
| --- | --- | --- |
| **ENAME** | **GRADE** | **DNAME** |
| SMITH | 1 | RESEARCH |
| MILLER | 2 | ACCOUNTING |
| WARD | 2 | SALES |
| MARTIN | 2 | SALES |
| ADAMS | 1 | RESEARCH |
| JAMES | 1 | SALES |

**Table 8 – GRADE values 1 and 2**

| **Emp\_table\_Partition1** | | |
| --- | --- | --- |
| **ENAME** | **GRADE** | **DNAME** |
| BLAKE | 4 | SALES |
| FORD | 4 | RESEARCH |
| SCOTT | 4 | RESEARCH |
| TURNER | 3 | SALES |
| JONES | 4 | RESEARCH |
| CLARK | 4 | ACCOUNTING |
| ALLEN | 3 | SALES |

**Table 9 – GRADE values 3 and 4**

| **Emp\_table\_Partition2** | | |
| --- | --- | --- |
| **ENAME** | **GRADE** | **DNAME** |
| KING | 5 | ACCOUNTING |

**Table 10 – GRADE value 5 and above**