Parallel Aggregation

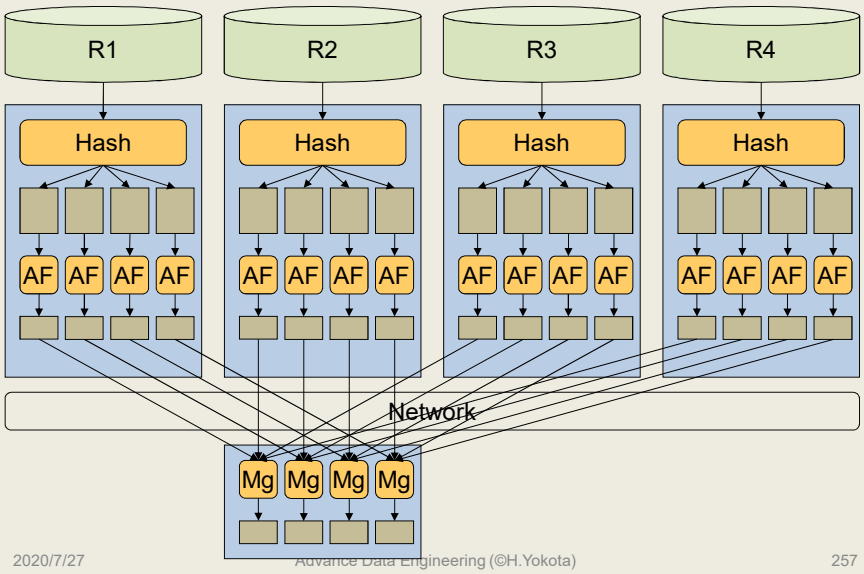
- Based on sequential aggregation algorithm using hash
  - Hashing for a Group-By operation
  - Applying aggregation functions to each group
    - Count, Sum, Average, Max, Min
- Three algorithms for parallel execution
  - Centralized Two Phase Algorithm (C-2P)
  - Two-Phase Algorithm (2P)
  - Repartitioning Algorithm (Rep)

Centralized Two Phase Algorithm (C-2P)

1. Read tuples from each local disk, apply a hash function, and execute aggregate function for hash buckets in each PE
2. Send the results of aggregation to **a node** to merge them

- The centralized node will be a bottleneck when the number of PE increases

Centralized Two Phase Algorithm (C-2P)



Pseudo Code for C-2P

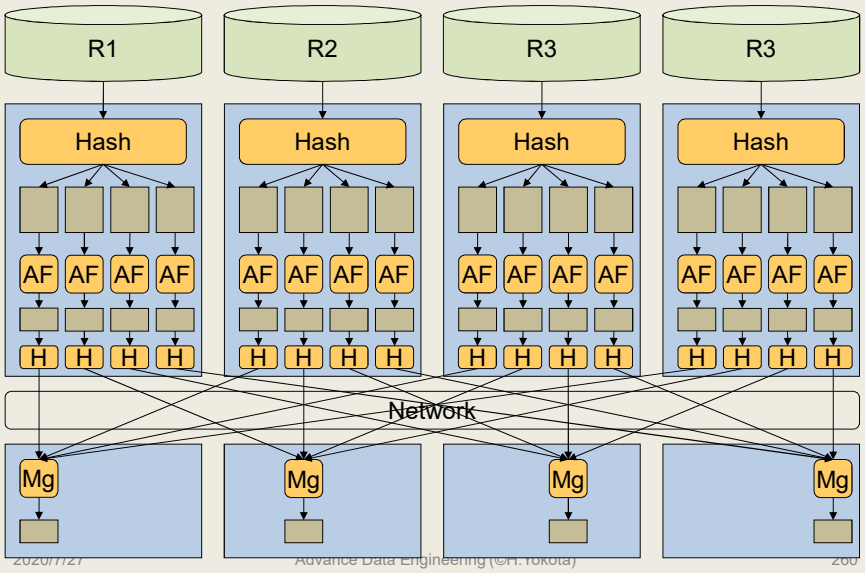
```
In each PE (indicated by x)
 for(i=1; i<={Rx}; i++) {
   get i-th tuple t from Rx;
   derive target attribute value v in t;
   y = h(v);
   keep t in buffer[y] }
  for(i=1; i <= group#; i++) {
   apply aggregate functions for buffer[i]
   send the result to a PE}
In the PE(0)
  merge the results for each group
```

Two-Phase Algorithm (2P)

1. Read tuples from each local disk, apply a hash function, and execute aggregate function for hash buckets in each PE
2. Send the results of aggregation to **nodes** corresponded with hash partition in **parallel**

- The merge operations are also executed in parallel

Two-Phase Algorithm (2P)



Pseudo Code for 2P

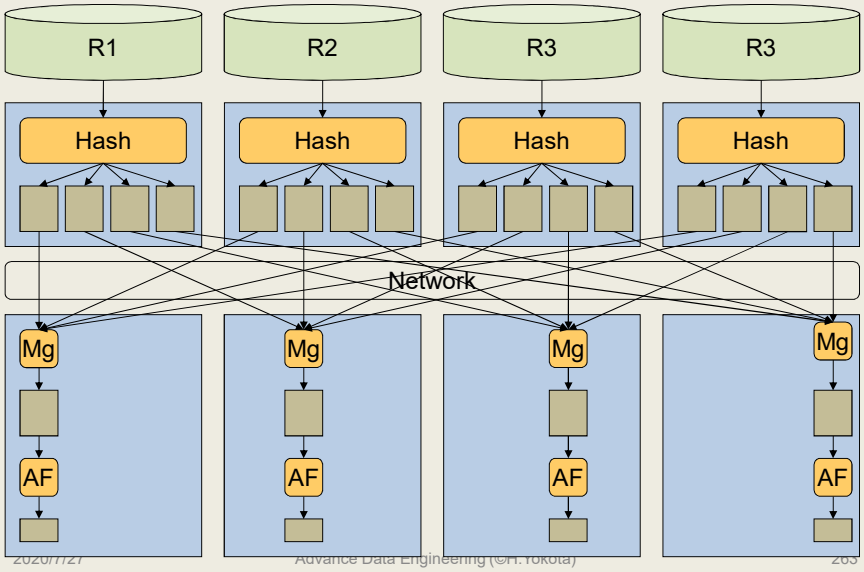
```
In each PE (indicated by x)
  for(i=1; i<={Rx}; i++) {
    get i-th tuple t from Rx;
    derive target attribute value v in t;
    y = h(v);
    keep t in buffer[y] }
  for(i=1; i <= group#; i++) {
    apply aggregate functions for buffer[i]
    z = h2(i);
    send the result to PE(z)}
In each PE
  merge the results for corresponding groups
```


Repartitioning Algorithm (Rep)

1. Read tuples from each local disk, and apply a hash function
2. Send the result to nodes depend on the hash result
3. A node receive the result execute aggregate function in **parallel**

- The number of invocations of aggregate functions can be reduced

Repartitioning Algorithm (Rep)



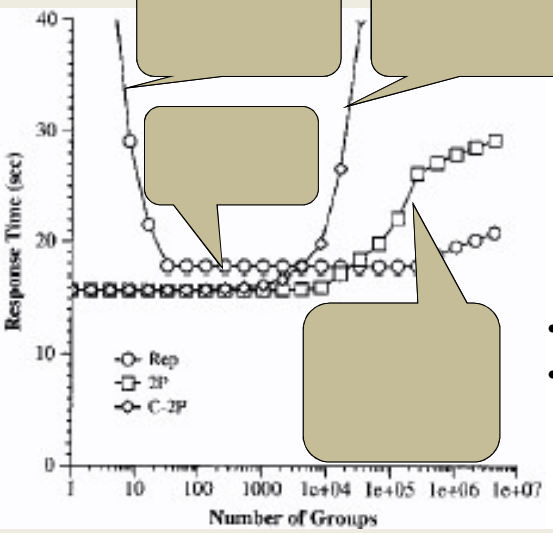
Pseudo Code for Rep (1)

In each PE (indicated by x)
for(i=1; i<={Rx}; i++) {
  get i-th tuple t from Rx;
  derive target attribute value v in t;
  y = h2(v);
  send t to PE(y) }

Pseudo Code for Rep (2)

```
In each PE
 merge tuples;
 n = count tuples;
 for(i = 1, i < n ; i++) {
  get i-th tuple from buffer and derive target attribute
  value v in t;
  z = h(v);
  store t in buffer[z];
 }
 for( j= 1; j < number of buffer in the PE; j++)  {
  apply aggregate functions for buffer[j]
 }
```


Comparison on Parallel Aggregation



From:

- Adaptive Parallel Aggregation Algorithm
- A. Shatdal and J.F. Naufhton,
- Proc. of Int'l Conf. SIGMOD '95, pp.104--114, (fig1)

- Using 32 PEs
- The graph indicates:
  - When there are small groups, 2p is better
  - Otherwise, Rep is better

Question (10-1)

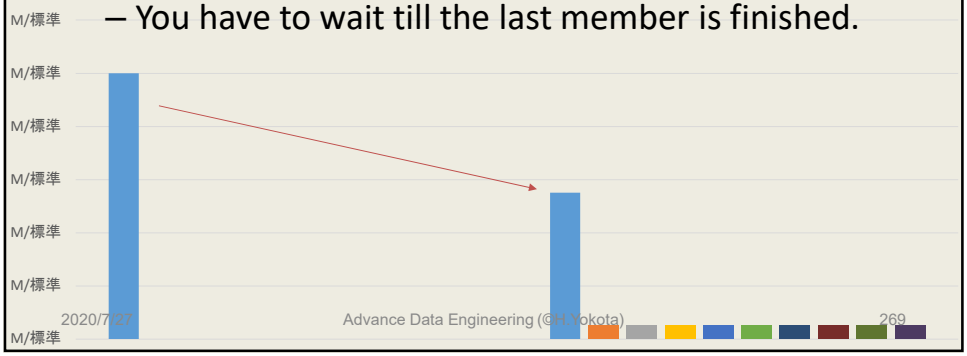
- From the discussion for comparing C-2P, 2P, and Rep, we can find that the problem size (the number of groups) and communication costs are influential factors for the performance of parallel processing.
- Describe the difference of curves in the graph when a lower bandwidth network is used.

Skews in a Join Operation

- If there are skews in parallel processing,
  - We cannot obtain enough scalability
    - Speed-up is restricted by the slowest PE

Image of skews

- Suppose you have 100 jobs and 10 members
- If the jobs are evenly distributed to 9 members (5 jobs for each), but one member takes 55 jobs, then speed up is less than twice.
  - You have to wait till the last member is finished.



Skews in a Join Operation

- If there are skews in parallel processing,
  - We cannot obtain enough scalability
    - Speed-up is restricted by the slowest PE
- Consider GRACE Hash Join
  - There are several reasons for skews
- Assumptions
  - Selection Operations are executed before the Join Operation
  - Results can be output from each PE

Types of Skews in a Join Operation

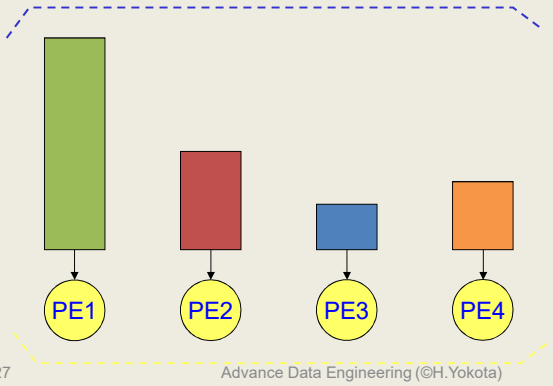
- Tuple Placement Skew
  - Tuple distribution skew before starting the query
- Selectivity Skew
  - Skew in the results of the selection before join
- Redistribution Skew
  - Bucket size skew in distribution phase of join operation
- Join Product Skew
  - Skew in the results of join phase

Handling of Skews in a Join

- Tuple Placement Skew:
  - Adjustment of tuple placement
    - Round-Robin partitioning, Hash partitioning, others
- Selectivity Skew / Redistribution Skew:
  - Fine Bucket Method (will be described soon)
- Join Product Skew:
  - Dynamic bucket allocation / output tuple allocation
- Focus on the Fine Bucket Method

Fine Bucket Method (1)

- If the number of PEs N_p is equal to the number of buckets N_b
  - Skews cannot be removed with any placement strategies



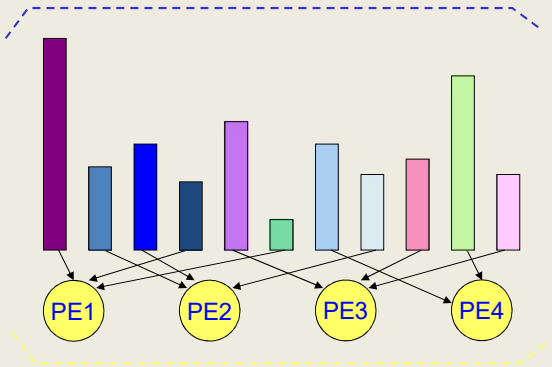
Bucket#: N_b

$N_b = N_p$

PE#: N_p

Fine Bucket Method (2)

- Make the number of buckets N_b quite larger than the number of PEs N_p



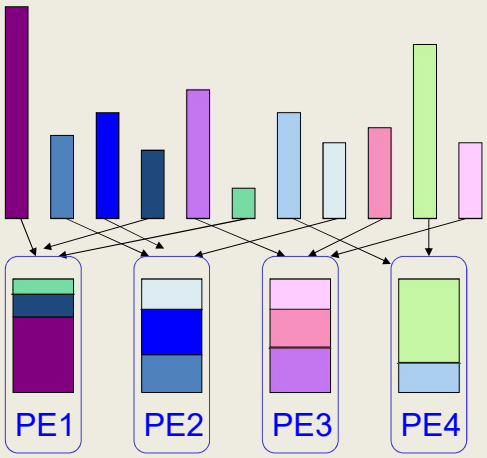
Bucket#: N_b

$N_b >> N_p$

PE#: N_p

Fine Bucket Method (3)

- Goal: task size in each PE becomes equivalent



A Bucket Allocation Strategy

- LPT (Longest Processing Time) First Strategy
  - Heuristics for Minimum Make Span
- Spreading Bucket Method
  - Calculation of bucket size and plan making
  - Distribute buckets to all PEs and make a plan in one of them
  - Merit of Spreading Bucket
    - There is no data concentration in a particular PE + Disk
    - Distribution of fine bucket in each module is similar
      - It is easy to obtain statistics information

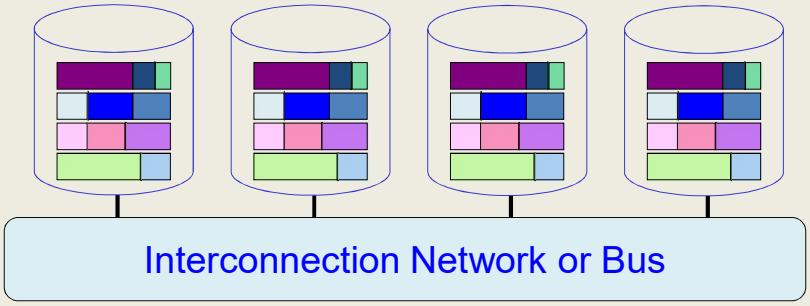
Rotational Bucket Collection (1)

- Routing without congestion during collecting buckets
- Cluster fine buckets into equalized task group by LPT First
  - Distribute N_p subtask group into N_p PEs
- i-th PE PE_i ($1 \leq i \leq N_p$)
  - Read i-th subtask group from

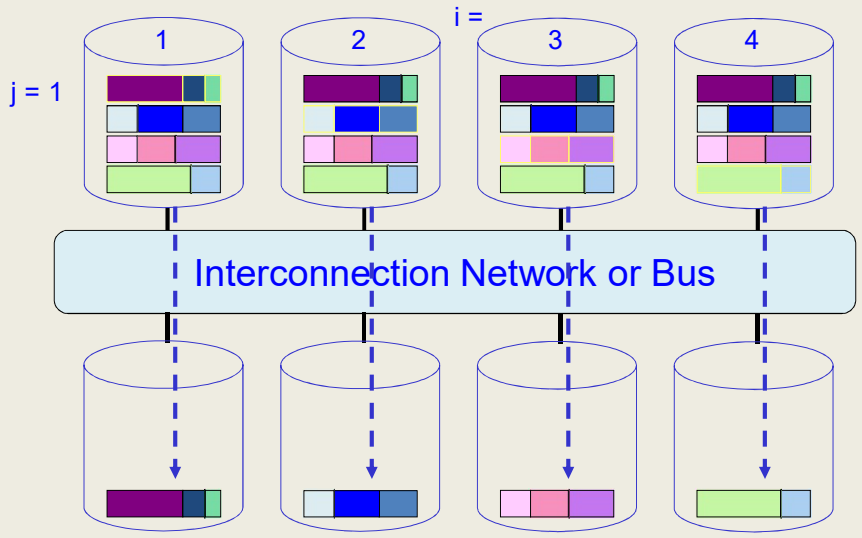
    $(((i + j) - 2) \mod N_p) + 1$

    module in j-th step

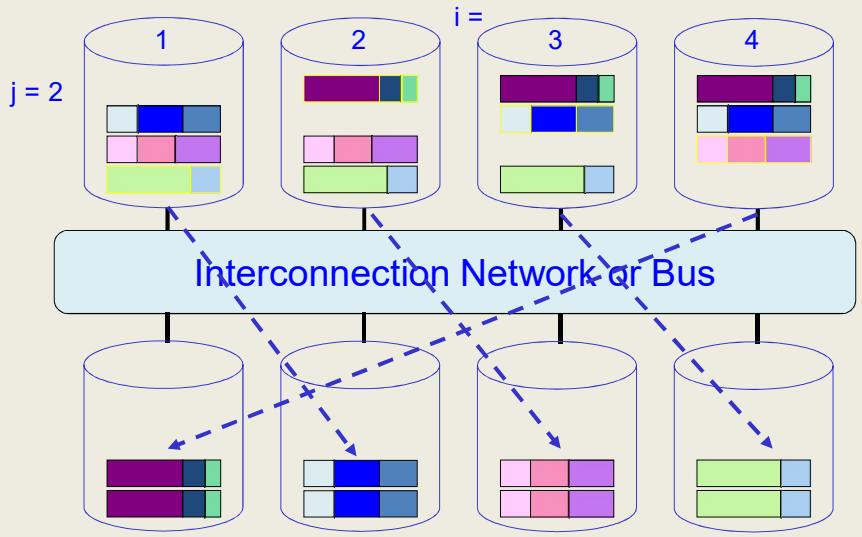
Rotational Bucket Collection (2)

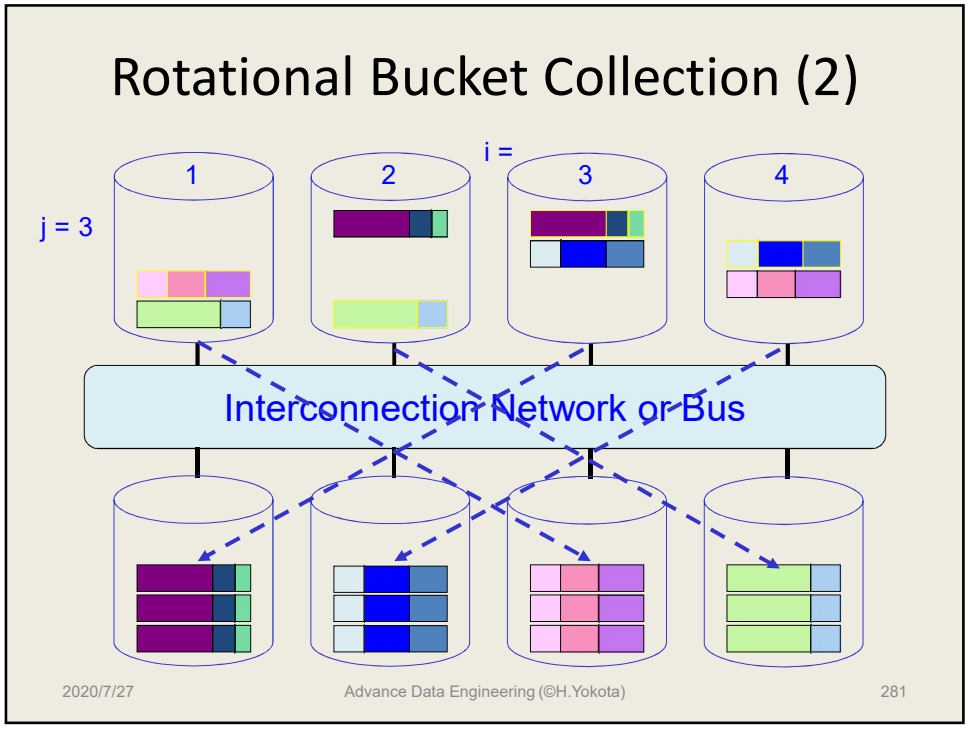


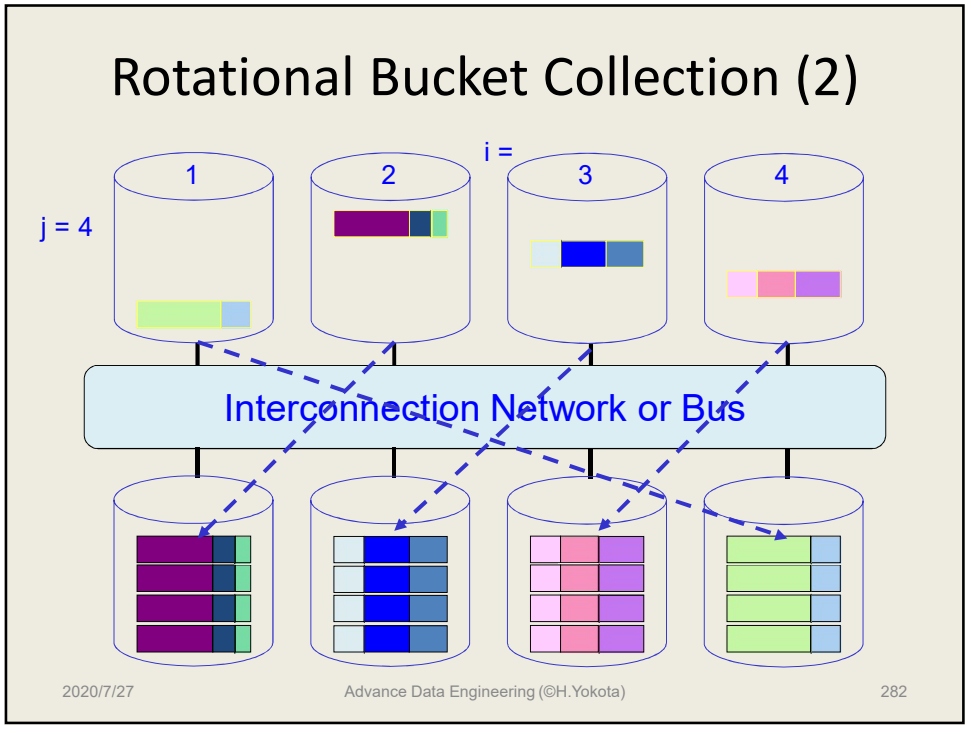
Rotational Bucket Collection (2)



Rotational Bucket Collection (2)




Rotational Bucket Collection (2)
i =
1
2
3
4
j = 3
Interconnection Network or Bus
2020/7/27
Advance Data Engineering (©H.Yokota)
281


Rotational Bucket Collection (2)
i =
1
2
3
4
j = 4
Interconnection Network or Bus
2020/7/27
Advance Data Engineering (©H.Yokota)
282

Process Flow of Fine Buckets

1. All tuples are hashed into N_b buckets, where $N_p << N_b$
2. Applying the Spreading-Bucket Method
3. Make task groups by the LPT First Scheduling
4. Applying the Rotational Bucket Collection
5. Do Join operation in each node

Costs of Fine Buckets

1. I/O for 1 PE during Hash:
   $2 \times (|R| + |S|)/ N_p$
2. The Spreading-Bucket Method can be done on-the-fly
3. Data collection for scheduling can be overlap on the I/O
4. I/O for collecting task groups:
   $2 \times (|R| + |S|)/ N_p$
5. I/O for Join operation:
   $(|R| + |S|)/ N_p$

Comparison on costs of Fine Buckets

- Let the maximum skew α%
  - When N_p is infinity, α% for sequential execution time
    - Thus, the execution time: α /100 ×3 × (|R| + |S|)
- When we adopt the Fine Bucket Method with the Spreading Bucket Method
  - Total I/O Cost: 5 ×(|R| + |S|)/ N_p
- A Rough Comparison
  - If α /100 ×3 × (|R| + |S|) > 5 ×(|R| + |S|)/ N_p , then Fine Bucket is effective
  - For N_p=10, if α > 17 %, then Fine Bucket is effective
  - For N_p=100, if α > 1.7%, then Fine Bucket is effective

Question (10-2)

- In actual situations, it is hard to make the load distribution completely even by the LPT First Strategy.
  a. Consider the condition of α for the case in which the Fine Bucket Method with the Spreading Bucket Method is effective for N_p = 100, when we assume that the maximum skew remains β% (difference between the longest and shortest execution time is β% of the sequential execution time) after applying the Fine Bucket Method.
  b. Consider approaches to make β smaller.

Combination of Methods

- Tuple placement Method
  - For Tuple Placement Skews
- Fine Bucket Method
  - For Selectivity / Redistribution Skews
- Dynamic bucket allocation / output tuple allocation Method
  - Join Product Skews
- Each method is independent
  - Combine these methods

Parallelize Hybrid Hash Join

- Prepare a corresponding Hash Table in each PE
  - It is difficult to build the Hash Table during read disk, because data is fragmented to all disks
  - Build the Hash Table while writing data into the disk in the Phase 1
    - It can reduce time of Phase 2
- However, we can not apply the fine bucket method