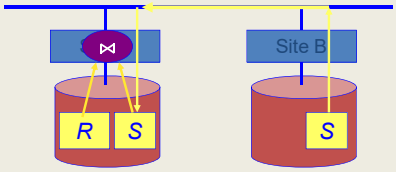
## Distributed Join

- When *R* and *S* are placed on the different sites
- Shall whole tuples of *S* (or *R*) be send to the other site?




- We can reduce the amount of data to be transferred.



## Approaches

- To reduce the amount of transferred data
- Type1: naïve (send whole tuples)
- Type2: Semi-Join
- Type3: 2-Way Semi-Join
- Type4: Hashed Bit Vector (Bloom Filter)
- Compare the costs for these approaches



## Cost Estimation

- Denotation
  - $C_R(R)$ denotes cost for reading relation *R* from a disk
  - $C_W(R)$ denotes cost for writing relation *R* to a disk
  - $C_D(R)$ (= $C_R(R) = C_W(R)$) denotes cost for disk I/O for *R*
  - $C_C(R)$ denotes cost for sending relation *R* via a network
  - $C_E(Op)$ denotes cost for executing operation Op
- Here, we assume $C_D(R)$ and $C_C(R)$ are proportional to the amount of data
  - If $\alpha = |S'|/|S|$ (and $\beta = |R'|/|R|$), then $C_{D/C}(S') = \alpha\, C_{D/C}(S)$
  - $C_{D/C}(S)$ can be replaced by $S \times C_{D/C}$

## Cost for Naïve Distributed Join

- Type1 (naïve: send whole tuples)
  - send whole tuples of S to site A and do join on site A
    - $C_{naive} = C_R(S) + C_C(S) + C_W(S) + C_E(R \text{ join } S)$
  - If we adopt GRACE hash join
    - $C_E(R \text{ join } S) \approx 3(C_D(R) + C_D(S))$
  - Thus
    - $C_{naive} \approx 5C_D(S) + 3C_D(R) + C_C(S)$
      $= (5S + 3R) \times C_D + S \times C_C$

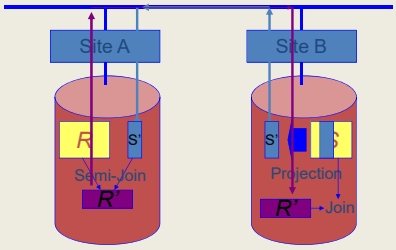


## Semi-Join Optimization

1. Do Projection on *S* for the join attributes
2. Send the results of the Projection from site B to A
3. Do Semi-Join *R* with the received attribute
4. Send the Join results relation from A to B
5. Do Join *S* and the received tuples



## Semi-Join Flow



## Cost for Semi-Join Approach

- Type2 (Semi-Join)
  - $C_{SJ}$
    $= C_R(S)\ [+ C_E(\pi S)] + C_W(S')$ : Projection
    $[+ C_R(S')\ ] + C_C(S') + C_W(S')$ : Send the result
    $+ C_E(R \text{ join } S') + C_W(R')$ : Do Semi-Join
    $[+ C_R(R')\ ] + C_C(R') + C_W(R')$ : Send back
    $+ C_E(R' \text{ join } S)$ : Do Join
    $\approx 4C_D(S) + 5C_D(S') + 3C_D(R) + 5C_D(R') + C_C(S') + C_C(R')$
    [projection can be overlapped on I/O]
    $= (4S + 5\alpha S + 3R + 5\beta R) \times C_D + (\alpha S + \beta R) \times C_C$

2019/8/5 Advance Data Engineering (©H.Yokota) 293

## Cost Comparison (1)

- For $C_{naive} > C_{SJ}$
  - $(S - \alpha S - \beta R)\ C_C > (5\alpha S + 5\beta R - S)\ C_D$
- If the selectivity of Projection ($\alpha$) and Join ($\beta$) becomes low (generate small sets),
  - Left side of the condition becomes large
  - Right side becomes small
  - It means that $C_{SJ}$ easily become effective
- On the other hand, if the communication cost $C_C$ is small compared with disk I/O cost $C_D$,
  - $C_{SJ}$ is not so effective

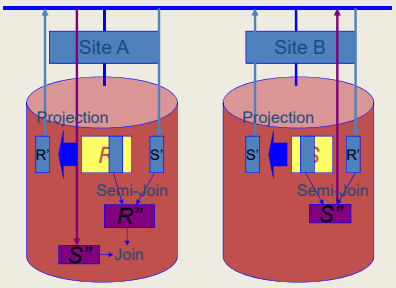
2019/8/5 Advance Data Engineering (©H.Yokota) 294

## 2-way Semi-Join Optimization

1. Do Projection on *S* and *R* for the join attributes simultaneously
2. Send the results of the Projection of *S* and *R* from site B to A and A to B, respectively
3. Do Semi-Join in each site
4. Send smaller results relation
5. Do Join

2019/8/5 Advance Data Engineering (©H.Yokota) 295

## 2-Way Semi-Join Flow






## Cost for 2-Way Semi-Join

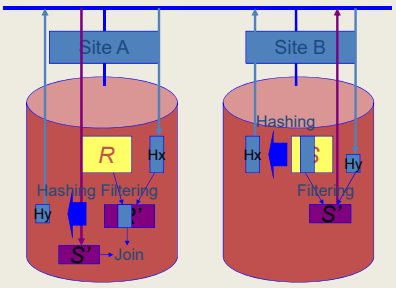
- Type3 (2-Way Semi-Join)
  - $C_{2WSJ} \approx$
    $max(\ C_R(S) + C_E(\pi S) + C_W(S'),\ \underline{C_R(R) + C_E(\pi R) + C_W(R')})$
    $+\ max(\ C_C(S') + C_W(S'),\ \underline{C_C(R') + C_W(R')})$
    $+\ max(\ \underline{C_E(R \text{ join } S') + C_W(R'')},\ C_E(S \text{ join } R') + C_W(S''))$
    $+\ \min(\ C_C(R'') + C_W(R''),\ \underline{C_C(S'') + C_W(S'')})$
    $+\ or(\ C_E(R'' \text{ join } S''),\ \underline{C_E(S'' \text{ join } R'')})$
  - If $S < R$ and $\alpha S < \beta R$,
    $C_{2WSJ} \approx 4C_D(R) + 5C_D(R') + 3C_D(S') + C_D(R'') + 4C_D(S'') + C_C(R') + C_C(S'') = (4R + 5\beta R + 3\alpha S + \beta\gamma R + 4\alpha\gamma S) \times C_D + (\beta R + \alpha\gamma S) \times C_C$



## Hashed Bit Vector Approach (Bloom Filter)

1. Make a Hashed Bit Vector *X* from *S* in site B
2. Send the Vector *X* to site A
3. Filtering *R* using the received Vector *X*
4. Make the other Hashed Bit Vector *Y* by the result of filtering
5. Send the Vector *Y* to site B
6. Filtering *S* using the received Vector *Y*
7. Send the result of filtering to site A
8. Do Join both results of filtering on site A

## Hashed Bit Vector Flow






## Cost of Hashed Bit Vector (1)

- $C_{HBV}$
  $=C_R(S)$ : Read $S$ from disk on site B
  + $C_E(Hash)$: Apply hash for $S$ to make a Vector $H_X$
    – This can be overlap on the previous read operation
  + $C_C(H_X)$ : Send the Vector $H_X$ from site B to site A
  + $C_R(R)$: Read $R$ from disk on site A
  + $C_E(Filter)$ : Filter $R$ by the $H_X$
    – This can also be overlap on the previous read operation
  + $C_W(R')$ : Write the filtered $R'$ into disk on site A
  + $C_E(Hash)$: Apply hash for $R'$ to make a Vector $H_Y$
    – This can also be overlap on the previous write operation



## Cost of Hashed Bit Vector (2)

  + $C_C(H_Y)$ : Send the Vector $H_Y$ from site A to site B
  + $C_R(S)$ : Read $S$ from disk on site B
  + $C_E(Filter)$ : Filter $S$ by the $H_Y$
    – This can also be overlap on the previous read operation
  + $C_W(S')$ : Write the filtered $S'$ into disk on site B
  + $C_C(S')$ : Send the filtered $S'$ to site A
  + $C_W(S')$ : Write the received $S'$ into disk on site A
  + $C_E(R' Join S')$ : Do GRACE hash join on site A
- $C_{HBV} \approx (2 + 5\alpha)C_D(S) + (1 + 4\beta)C_D(R) + C_C(H_X) + C_C(H_Y) + \alpha C_C(S)$

## Cost Comparison

- For $C_{naive} > C_{HBV}$
  - $(S - \alpha S - H_X - H_Y)C_C > (5\alpha S + 4\beta R - 3S - 2R)C_D$
- Cf. $C_{naive} > C_{SJ}$
  - $(S - \alpha S - \beta R)\ C_C > (5\alpha S + 5\beta R - S)\ C_D$
- Right side of $C_{HBV}$ condition easily becomes small
  - It meas that Hashed Bit Vector easier become effective than Semi-Join approach



## Summary of Cost Estimation

- Type1: naïve (send whole tuples)
  - $C_{naive} \approx (5S + 3R) \times C_D + S \times C_C$
- Type2: Semi-Join
  - $C_{SJ} \approx (4S + 5\alpha S + 3R + 5\beta R) \times C_D + (\alpha S + \beta R) \times C_C$
- Type3: 2-Way Semi-Join
  - $C_{2WSJ} \approx (4R + 5\beta R + 3\alpha S + 2\beta\gamma R + 2\alpha\gamma S) \times C_D + (\beta R + \alpha\gamma S) \times C_C$
- Type4: Hashed Bit Vector (Bloom Filter)
  - $C_{HBV} \approx (2S + 5\alpha S + R + 4\beta R) \times C_D + (H_X + H_Y + \alpha S) \times C_C$



## Assignment 13

- Roughly estimate the execution time for the 4 distributed join algorithms with the following assumptions:
  - Cardinality of a relation R: 1,000,000, S: 500,000
  - Total length of a tuple: R:1,000B, S:2,000B
  - Disk transfer bandwidth: 10MB/s
  - Network bandwidth: 5MB/s
  - Selectivity: $\alpha$ =10%, $\beta$ =10%, $\gamma$=10%
  - Hash Bit Vector: use 1 bit for each tuple

## Current Trend

- The Internet
  - Its bandwidth becomes wide (Mbps, Gbps)
  - Its latency is large
    - Overhead for establishing connections
    - Optical speed (for long distance)
  - The communication frequency is more dominant than amount of transferred data
  - Database Migration (Proposed a group in Osaka U. )
- Backup Sites using the Internet
  - iSCSI (SCSI on IP)
    - Disk control protocol on the IP network
    - Replication by the disk access level

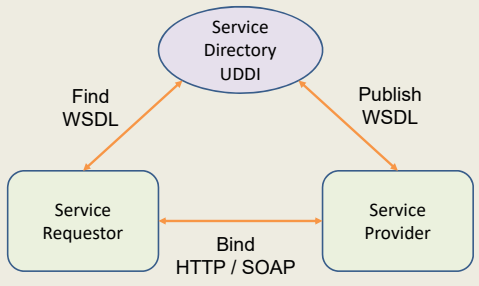


## Other Trends

- B2B : Business-to-Business
  - Supply Chain Management
- C2B/B2C : Customer-to-Business
  - Travel Arrangement
  - Amazon.com
- P2P: Pear-to-Pear
  - File Sharing
- Web Service
- Semantic Web



## Web Service



## Web Service

- WSDL: Web Services Description Language
  - An XML format for describing network services
- SOAP: Simple Object Access Protocol
  - A protocol for accessing web services
- UDDI: Universal Description, Discovery and Integration
  - A directory for storing information about web services



## Semantic Web

- The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries
  - The term was coined by Tim Berners-Lee for a web of data that can be processed by machines
  - Tim Berners-Lee: best known as the inventor of the World Wide Web



## Important Components

- XML: Extensible Markup Language
- RDF: Resource Description Framework
- RDFS: RDF Schema
  - provides a data-modelling vocabulary for RDF data
- SPARQL: SPARQL Protocol and RDF Query Language
- OWL: Web ontology language
- URI: Uniform Resource Identifier