Caching and Load Shedding in Semi-Stream Joins for Skewed Big Data

M. Asif Naeem[1], Gerald Weber[2], Gillian Dobbie[2], Christof Lutteroth[2]

[1]Auckland University of Technology, New Zealand
[2]The University of Auckland, New Zealand

February 9, 2016

At a Glance

- **Research issues in Processing of Semi-Stream Data:**
  - Processing of semi-stream data with different arrival rates under limited memory. For example joining of fast stream of source updates with a slow disk-based master data in real-time data warehousing.
  - Processing and load shedding of skewed stream data efficiently.

At a Glance

- **Research issues in Processing of Semi-Stream Data:**
  - Processing of semi-stream data with different arrival rates under limited memory. For example joining of fast stream of source updates with a slow disk-based master data in real-time data warehousing.
  - Processing and load shedding of skewed stream data efficiently.
- **Existing approach:**
  - Mesh Join (MESHJOIN) is a well known approach in this context.
  - MESHJOIN does not consider a very common characteristic (skewed distribution) of the stream data and therefore, both the processing and load shedding of the stream data can be suboptimal.

At a Glance

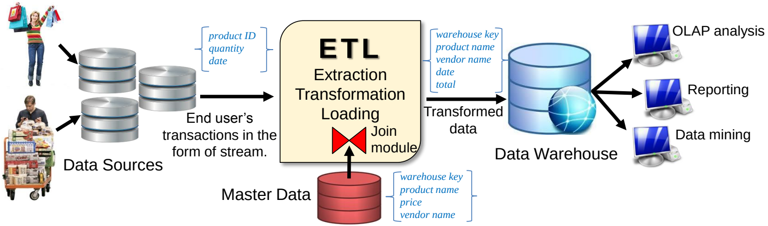
- **Research issues in Processing of Semi-Stream Data:**
  - Processing of semi-stream data with different arrival rates under limited memory. For example joining of fast stream of source updates with a slow disk-based master data in real-time data warehousing.
  - Processing and load shedding of skewed stream data efficiently.
- **Existing approach:**
  - Mesh Join (MESHJOIN) is a well known approach in this context.
  - MESHJOIN does not consider a very common characteristic (skewed distribution) of the stream data and therefore, both the processing and load shedding of the stream data can be suboptimal.
- **Our approach:**
  - We present a generic caching approach that can be used as a front-stage with any semi-stream join algorithm in order to optimize the processing of skewed stream data.
  - we present a novel, selective load shedding technique which sheds the fraction of the stream that is most expensive to join.

Outline

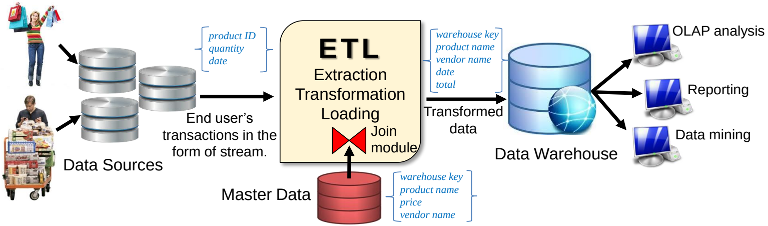
Introduction

- Semi-stream join algorithms join a fast data stream with a disk-based relation. For example in realtime data warehousing where a stream of transactions is joined with master data before loading it into a data warehouse.
- The join is used to enrich the stream data with the master data.
- A common type of join in this scenario is an equijoin which is many-to-one type of join between foreign keys in the stream data and primary key in the master data.
- Usually available memory for the join algorithm is not large enough to hold the whole disk-based master data.
- Stream data is normally a skewed (non-uniform) distribution.

Motivation

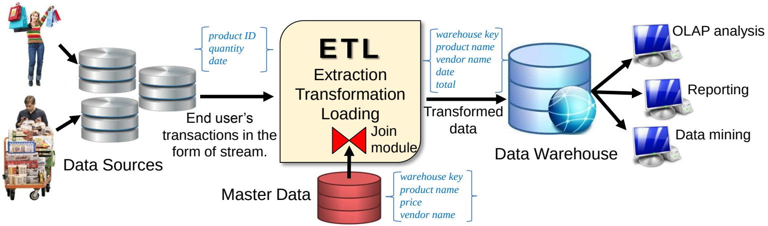

product ID
quantity
date
End user's transactions in the form of stream.
Data Sources
ETL
Extraction
Transformation
Loading
Join module
Master Data
warehouse key
product name
price
vendor name
warehouse key
product name
vendor name
date
total
Transformed data
Data Warehouse
OLAP analysis
Reporting
Data mining

Motivation



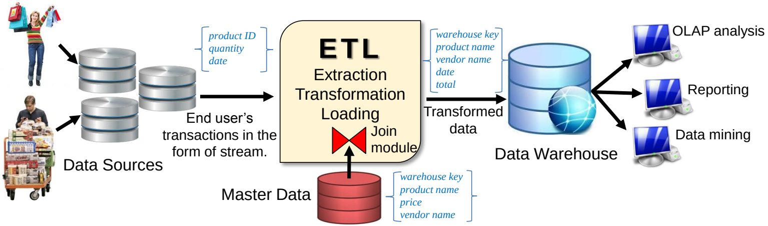
- Data arriving from data sources is huge in volume and fast while access rate of the master data is slow.

Motivation



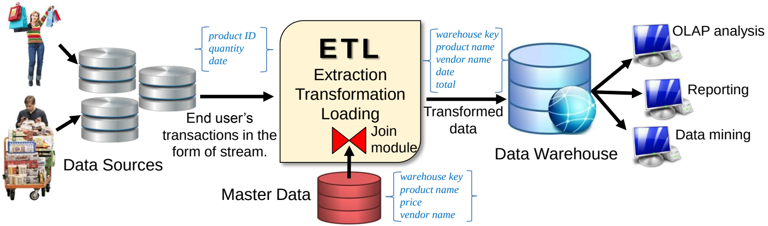
- Data arriving from data sources is huge in volume and fast while access rate of the master data is slow.
- Consequently there is a bottleneck in the transformation layer.

Motivation



- Data arriving from data sources is huge in volume and fast while access rate of the master data is slow.
- Consequently there is a bottleneck in the transformation layer.
- The stream data is skewed.

Motivation



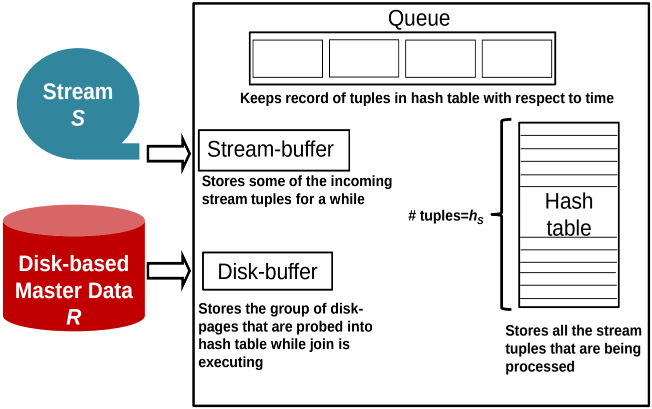
- Data arriving from data sources is huge in volume and fast while access rate of the master data is slow.
- Consequently there is a bottleneck in the transformation layer.
- The stream data is skewed.
- The stream arrival rate can be greater than the service rate of the join algorithm.

MESHJOIN

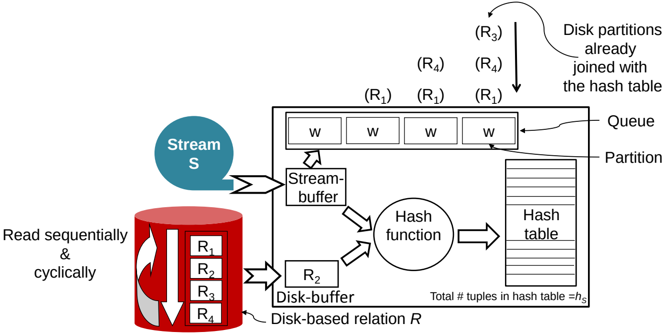
- A wellknown algorithm MESHJOIN[1] has been introduced in this area.
- Implements many-to-one equijoin which is norm in the scenario of data warehousing.
- Designed for joining a stream S with the disk-based master data R.
- Retrieves R sequentially and therefore assumes no index on R.
- Can cope with limited memory.

[1] Polyzotis et al., IEEE Transaction on Knowledge and Data Engineering, July, 2008.

Components of MESHJOIN



MESHJOIN Operation



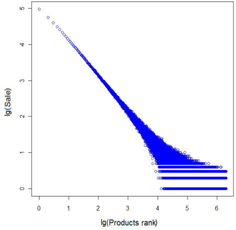
Size of disk-buffer = one disk partition

Iterations required to bring all of *R* into memory = *k* (in this example *k*=4)

tuples in stream-buffer = # pointers in one queue partition= $w = \frac{h_S}{k}$

Issues with MESHJOIN

- Skewed distribution is a common characteristic of real world data e.g. in many markets some products are bought with higher frequency[1].
- MESHJOIN makes no assumptions about data distribution or the organization of the master data.
- Experiments of MESHJOIN have shown that the algorithm performs worse with skewed data than with uniform data.
- Load shedding approach used in MESHJOIN is suboptimal.



[1]Anderson, C., The Long Tail: Why the Future of Business is Selling Less of More., Hyperion, 2006.

Our Optimization Approaches

As a solution we present two optimization approaches.

1. A novel caching approach that works as a generic front-stage for existing semi-stream join algorithms.
   - The new front-stage uses a tuple-level rather than a page-level cache.
   - The front stage significantly improves join service rate for skewed data.
   - We tested our front-stage with three different well known semi-stream join algorithms.
   - We provide experimental results to demonstrate the benefits of the approach.

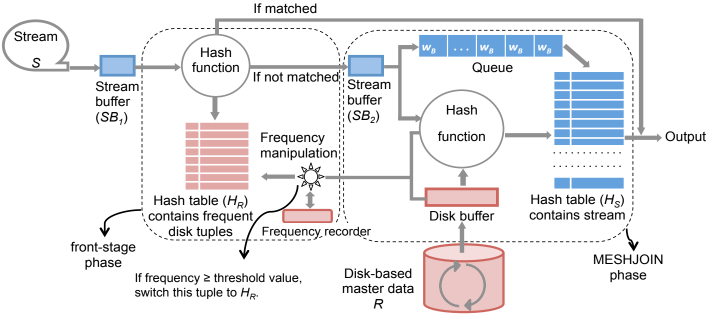
Our Optimization Approaches

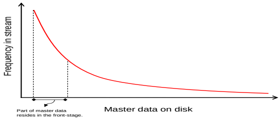
As a solution we present two optimization approaches.

1. A novel caching approach that works as a generic front-stage for existing semi-stream join algorithms.
    - The new front-stage uses a tuple-level rather than a page-level cache.
    - The front stage significantly improves join service rate for skewed data.
    - We tested our front-stage with three different well known semi-stream join algorithms.
    - We provide experimental results to demonstrate the benefits of the approach.
2. A novel load shedding approach.
    - Contrary to the existing approach our approach sheds the tuples which are expensive to join.
    - In our approach load shedding increase the service rate which is not the case in the existing approach.
    - We provide experimental data with significant improvement in service rate.

Caching Approach

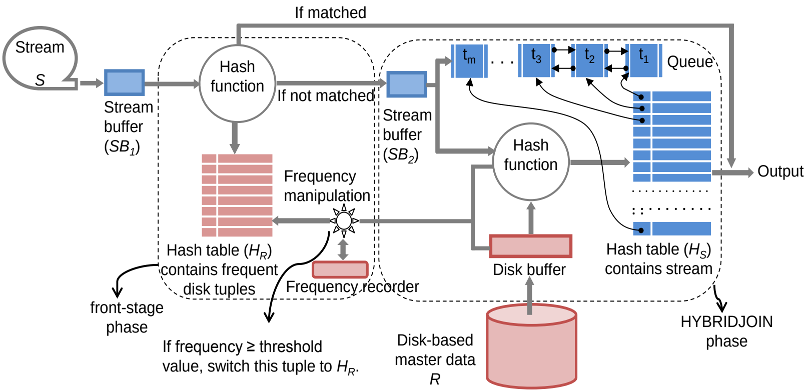
Applied on MESHJOIN (Mesh Join) – renamed with CMESHJOIN (Cached Mesh Join)





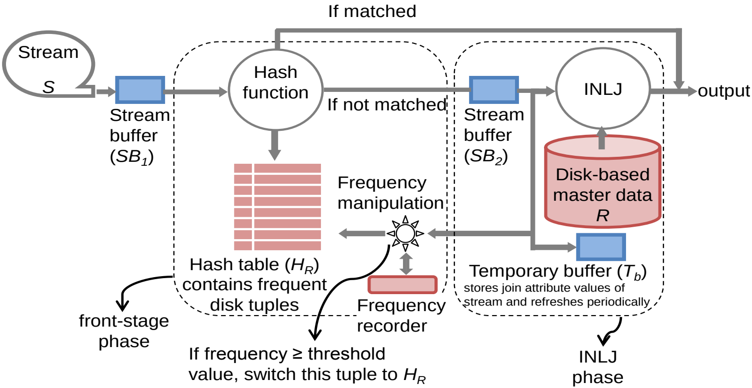
Caching Approach

Applied on HYBRIDJOIN (Hybrid Join) – renamed with CHYBRIDJOIN (Cached Hybrid Join)



Caching Approach

Applied on INLJ (Index Nested Loop Join) – renamed with CINLJ (Cached Index Nested Loop Join)



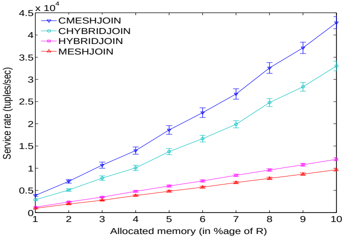
Experimental Evaluation

Setup

- Hardware specifications: Core i5, 8GB main memory, 500GB hard drive
- Allocated memory: 1% of R (0.11GB) to 10% of R (1.12GB)
- Data set:
  - Disk-based data
    - Size of R, 100 million tuples (11.2GB)
    - Size of each tuple, 120 bytes (Similar to MESHJOIN)
  - Stream data
    - Size of each tuple, 20 bytes (Similar to MESHJOIN)
    - Size of each pointer in queue, 4 bytes in CMESHJOIN while 12 bytes in CHYBRIDJOIN
    - Based on Zipf's Law with skew value from 0 to 1
- Evaluation metrics: We calculate confidence interval by considering 95% accuracy rate.

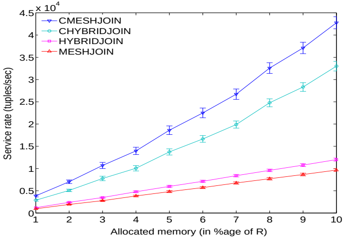
Experimental Evaluation

Service rate vs memory (R≈11.2G, Skew=1)

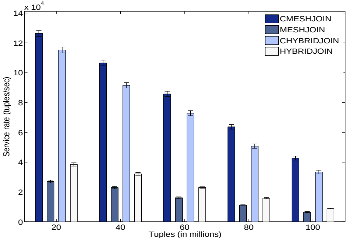


Experimental Evaluation

Service rate vs memory (R≈11.2G, Skew=1)

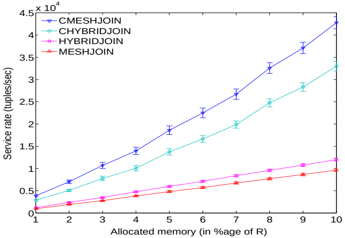


Service rate vs size of *R* (M≈1.12G, Skew=1)

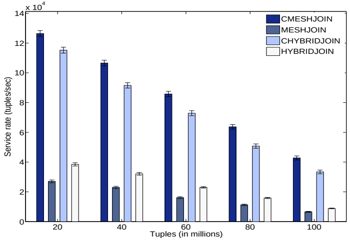


Experimental Evaluation

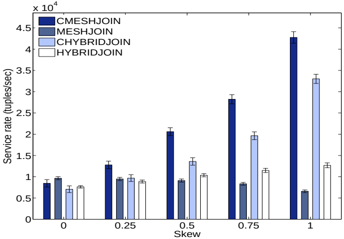
Service rate vs memory (R≈11.2G, Skew=1)



Service rate vs size of *R* (M≈1.12G, Skew=1)

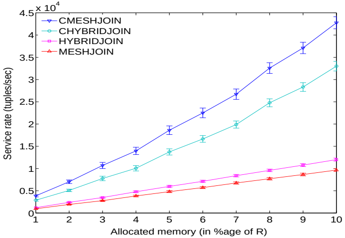


Service rate vs skew (R≈11.2G, M≈1.12G)

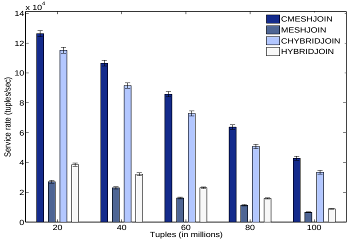


Experimental Evaluation

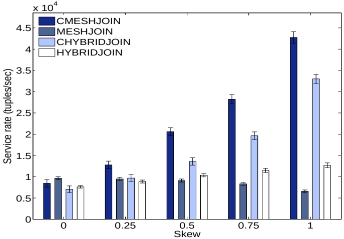
Service rate vs memory (R≈11.2G, Skew=1)



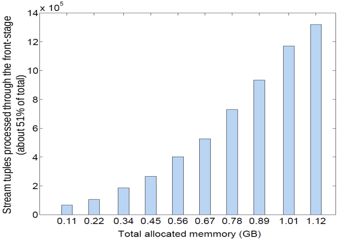
Service rate vs size of *R* (M≈1.12G, Skew=1)



Service rate vs skew (R≈11.2G, M≈1.12G)

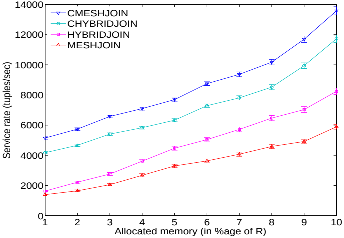


Role of the front-stage (R≈11.2G, Skew=1)



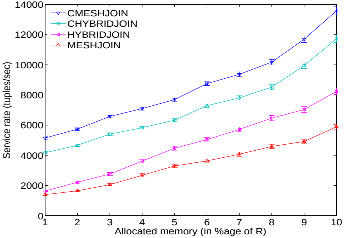
Experimental Evaluation

TPCH dataset (R≈20 million tuples)

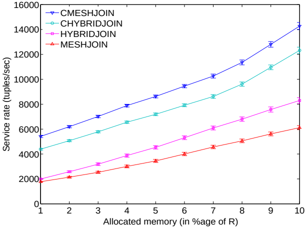


Experimental Evaluation

TPCH dataset (R≈20 million tuples)

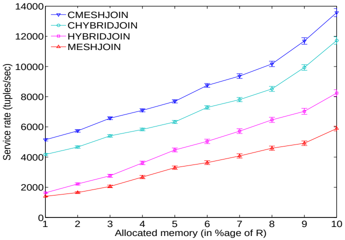


Real-life dataset (R≈20 million tuples)

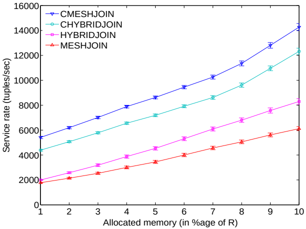


Experimental Evaluation

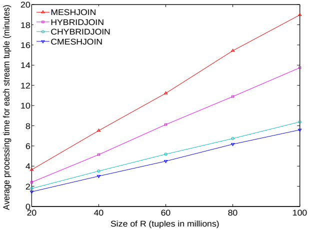
TPCH dataset (R≈20 million tuples)



Real-life dataset (R≈20 million tuples)

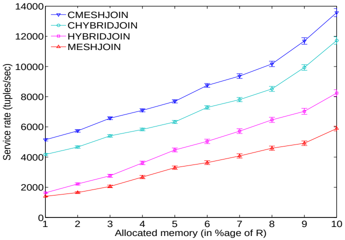


Processing time (M≈1.12G, Skew=1)

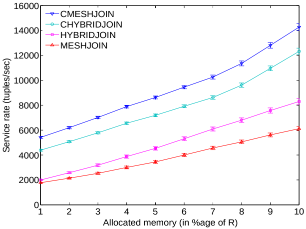


Experimental Evaluation

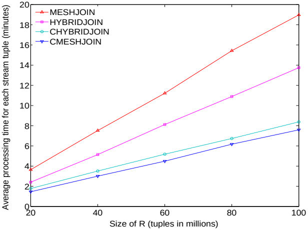
TPCH dataset (R≈20 million tuples)



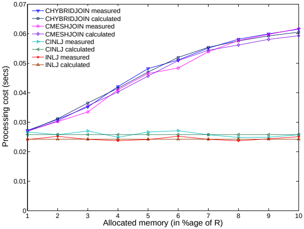
Real-life dataset (R≈20 million tuples)



Processing time (M≈1.12G, Skew=1)

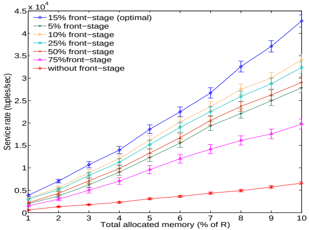


Cost validation (R≈11.2G, Skew=1)



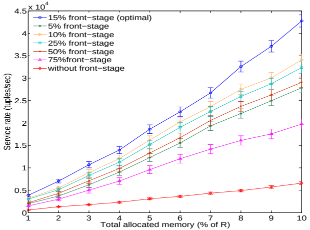
Sensitivity Analysis

Front-stage analysis

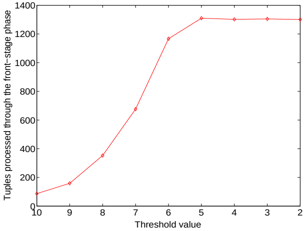


Sensitivity Analysis

Front-stage analysis

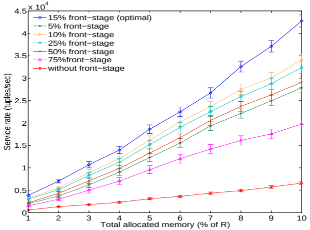


Threshold sensitivity analysis

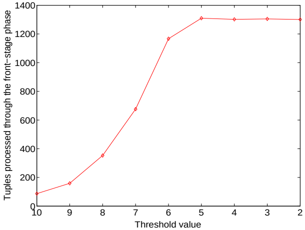


Sensitivity Analysis

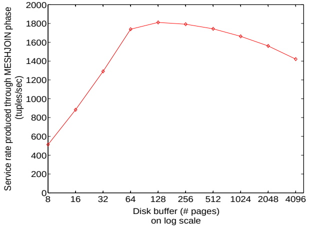
Front-stage analysis



Threshold sensitivity analysis

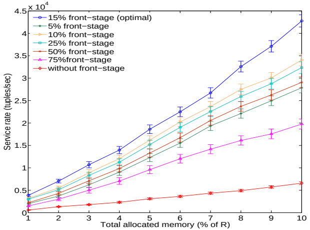


Disk buffer size analysis

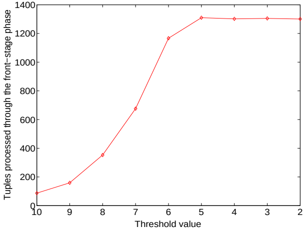


Sensitivity Analysis

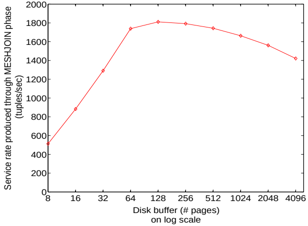
Front-stage analysis



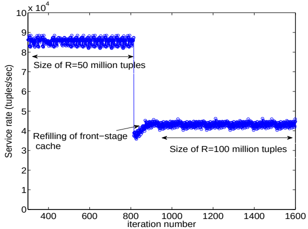
Threshold sensitivity analysis



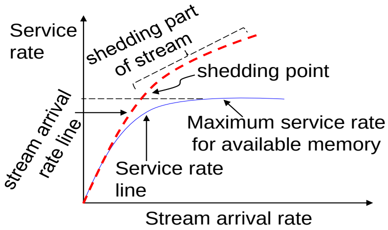
Disk buffer size analysis



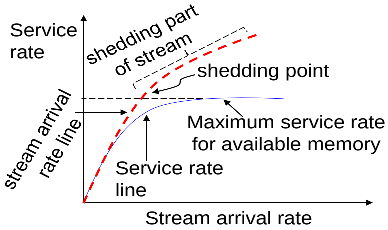
Changing the size of R online



Load Shedding Approach

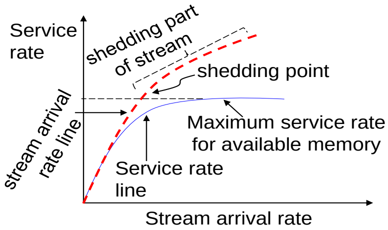


Load Shedding Approach



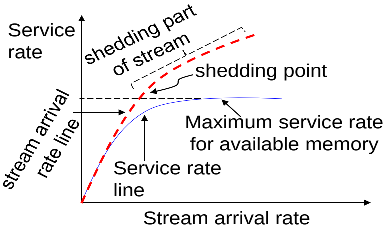
- A novel and efficient load shedding technique particularly for skewed semi-stream data.

Load Shedding Approach



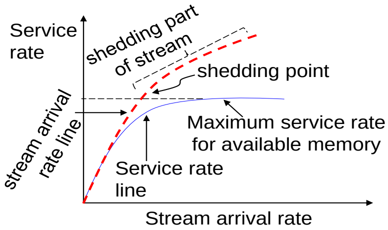
- A novel and efficient load shedding technique particularly for skewed semi-stream data.
- Contrary to the existing approaches, only sheds those tuples which are expensive to join.

Load Shedding Approach



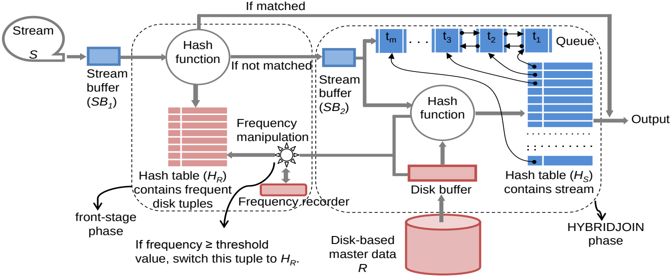
- A novel and efficient load shedding technique particularly for skewed semi-stream data.
- Contrary to the existing approaches, only sheds those tuples which are expensive to join.
- Uses the existing architecture with minimum overhead.

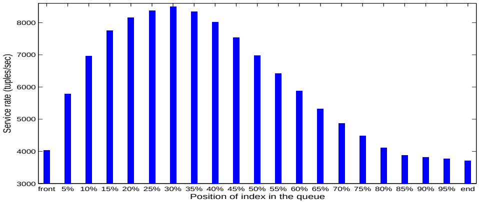
Load Shedding Approach



- A novel and efficient load shedding technique particularly for skewed semi-stream data.
- Contrary to the existing approaches, only sheds those tuples which are expensive to join.
- Uses the existing architecture with minimum overhead.
- As a consequence the service rate improves significantly.

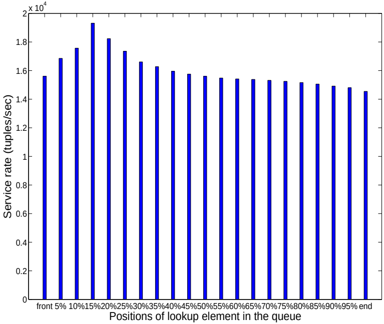
Reconsidering of CHYBRIDJOIN





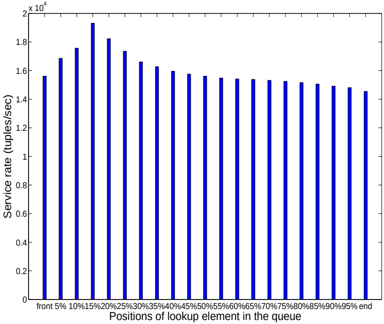
Experimental Evaluation

Queue optimization for load shedding

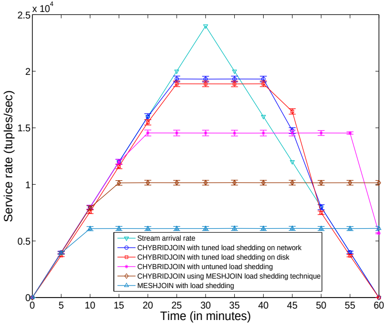


Experimental Evaluation

Queue optimization for load shedding



Service rate analysis under load shedding



Related Work

1. Index Nested Loop Join (INLJ)[1]
2. MESHJOIN [2]
3. Partition-based approach[3]
4. R-MESHJOIN [4]
5. HYBRIDJOIN [5]
6. Semi-Streamed Index Join [6]

[1] R. Ramakrishnan et al., Database Management System, McGraw-Hill, Inc., 1999.
[2] N. Polyzotis et al., IEEE TKDE, 2008.
[3] A. Chakraborty et al., IPDPS, 2009.
[4] M. A. Naeem et al., DOLAP, 2010.
[5] M. A. Naeem et al., IJDWM, 2011.
[6] Mihaela A. Bornea et al., ICDE, 2011.

Conclusions

- Existing algorithm MESHJOIN is suboptimal for processing of the skewed semi-stream data.
- The load shedding technique presented in MESHJOIN is inefficient.

Conclusions

- Existing algorithm MESHJOIN is suboptimal for processing of the skewed semi-stream data.
- The load shedding technique presented in MESHJOIN is inefficient.
- We presented the following two major contributions:
  - Caching
    - Presented a generic cache module called front-stage.
    - Uses a tuple level cache.
    - Tested it with three wellknown algorithms.
    - Improved the service rate of the existing algorithms.

Conclusions

- Existing algorithm MESHJOIN is suboptimal for processing of the skewed semi-stream data.
- The load shedding technique presented in MESHJOIN is inefficient.
- We presented the following two major contributions:
  - Caching
    - Presented a generic cache module called front-stage.
    - Uses a tuple level cache.
    - Tested it with three wellknown algorithms.
    - Improved the service rate of the existing algorithms.
  - Load shedding
    - Presented an intelligent load shedding technique.
    - The new technique exploits the skew in stream data.
    - Contrary to the existing MESHJOIN our load shedding approach improved the service rate.

Conclusions

- Existing algorithm MESHJOIN is suboptimal for processing of the skewed semi-stream data.
- The load shedding technique presented in MESHJOIN is inefficient.
- We presented the following two major contributions:
  - Caching
    - Presented a generic cache module called front-stage.
    - Uses a tuple level cache.
    - Tested it with three wellknown algorithms.
    - Improved the service rate of the existing algorithms.
  - Load shedding
    - Presented an intelligent load shedding technique.
    - The new technique exploits the skew in stream data.
    - Contrary to the existing MESHJOIN our load shedding approach improved the service rate.
- We performed a sensitivity analysis with respect to various internal parameters.
- We validated our cost model empirically.

Conclusions

- Existing algorithm MESHJOIN is suboptimal for processing of the skewed semi-stream data.
- The load shedding technique presented in MESHJOIN is inefficient.
- We presented the following two major contributions:
  - Caching
    - Presented a generic cache module called front-stage.
    - Uses a tuple level cache.
    - Tested it with three wellknown algorithms.
    - Improved the service rate of the existing algorithms.
  - Load shedding
    - Presented an intelligent load shedding technique.
    - The new technique exploits the skew in stream data.
    - Contrary to the existing MESHJOIN our load shedding approach improved the service rate.
- We performed a sensitivity analysis with respect to various internal parameters.
- We validated our cost model empirically.
- Source download: `www.cs.auckland.ac.nz/research/groups/serg/j/tkde/`

Thanks