【VLDB2011勉強会】

Session 16: Streams and Events

担当： 小山田 昌史（筑波大学）

Session 16 概要

- 内容
  - Continuous Query 系 (ストリーム処理，CEP) が 2 件
  - RDBMS のクエリ最適化系が 1 件
- 論文一覧

″Lightweight Graphical Models for Selectivity Estimation Without Independence Assumptions″

- *RDBMS* におけるクエリ最適化（選択率の見積もり）の話

″Active Complex Event Processing over Event Streams″

- *CEP* + *Active Rule* という新しい処理モデルの話

″Massive Scale-out of Expensive Continuous Queries″

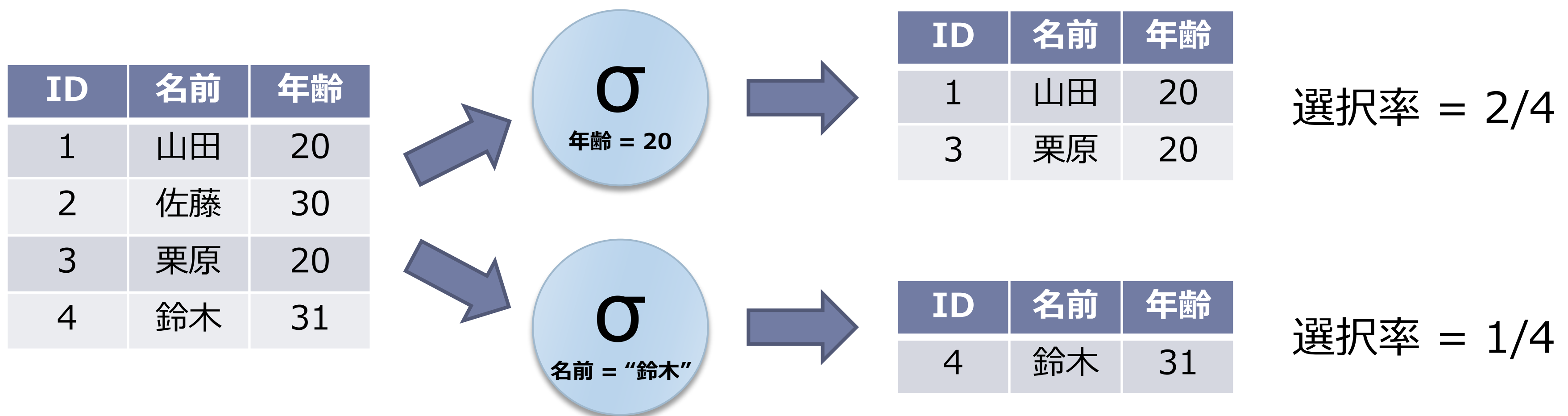
- 並列データストリーム処理の話

Lightweight Graphical Models for Selectivity Estimation Without Independence Assumptions

Kostas Tzoumas (Aalborg University, Denmark), Amol Deshpande (University of Maryland), Christian S. Jensen (Aarhus University)

基礎知識の確認：クエリ最適化

- RDBMS においてクエリは処理木へと変換される
  - 一つのクエリに対していくつかの処理木（候補）が考えられる
  - それぞれ，実行コスト（主に二次記憶へのアクセス量）は異なる
  - 最もコストが小さいものを選びたい
- どうやって，処理木のコストを見積もる？
  - 流れるデータの量を，オペレータの内容とデータベースの統計情報から推測
  - オペレータにデータを流した際，データの量がどれほど変化するか（選択率）
  - オペレータの条件を満たすデータがデータベースにどれだけ存在するか，推測
    - （もちろんデータベースの中身を覗けば「正しい」選択率は求められるが，それでは本末転倒）
- 各属性について，値の確率分布を近似するという話に帰着



選択率の見積もり

古典的な方式 (System-R) は 3 つの仮定を行う

この値を保持しておくだけで良い！

1. データの値は一様分布
   - 各属性について，テーブル中に出現する値の種類数が N のとき
   - 各値 x の出現確率を P(x) = 1 / N と近似できる
2. テーブル内の属性同士には相関がない (Selection)
   - P(R.X = x, R.Y = y) を P(R.X = x) * P(R.Y = y) で近似可能に
3. テーブル間の属性にも相関はない (Join)

これらの仮定は計算量とデータ量を削減するための妥協

- 現実にはデータは一様分布ではなく，属性同士の相関もある
  - 例: 年齢が高いほど，給料も高い
- この方式で見積もられた選択率はあまり良くないものとなってしまう

これまでの研究では

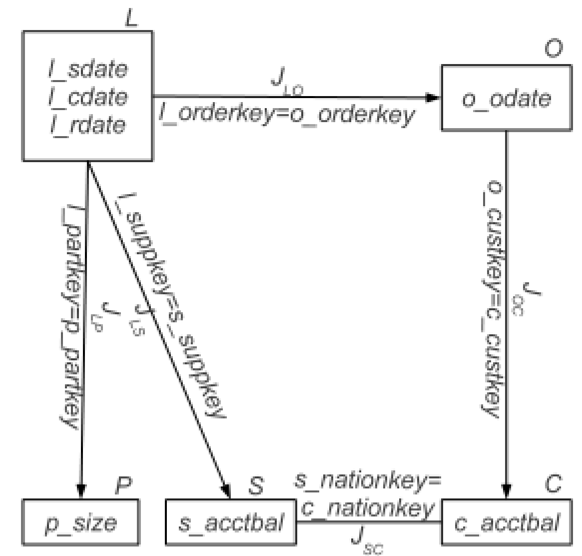
多くの RDBMS で実際に行われている

- 1 に関して
  - 属性ごとに，値の分布をコンパクトに近似する (Attribute-Level Synopsis)
- 2, 3 に関して
  - 独立性を仮定から取り除くと，複数の属性の同時分布を保持する必要が出てくる
  - 高次元の同時分布は，その計算量とデータ量が莫大！

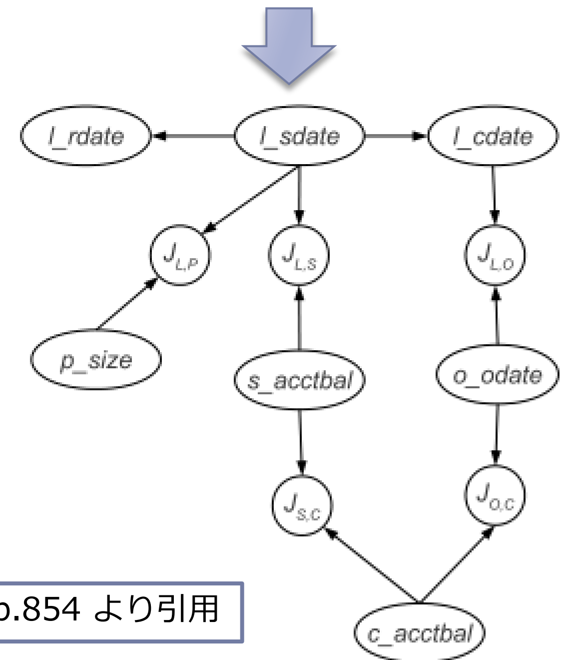
本研究では「**グラフィカルモデル**」を用い，少ない計算量・データ量で同時分布を表現

グラフィカルモデルの利用

- グラフィカルモデルとは？
  - ある確率分布を，それよりも低次の確率分布の積で表現
  - 例: ベイジアンネットワーク
- 選択率見積もりへの応用
  - 先行研究あり [Getoor '01]
    - 高次元の分布を作成する際のオーバーヘッドが大きい
    - キー属性に限られている
- 本研究のアプローチ
  - 多くの依存関係を保持するモデルを作成
    - クエリとスキーマの情報を利用
    - モデルは，ベイジアンネットワーク形式で表現される
      - （実際には Junction Tree という，周辺分布の計算しやすい形式で表現）
    - 非常に少ない確率分布より，様々な確率分布を算出できる
  - ベイジアンネットワークモデルに対して選択率の見積もりを行う
    - 動的計画法で，非常に効率が良い



(a) Schema graph and descriptive attributes.

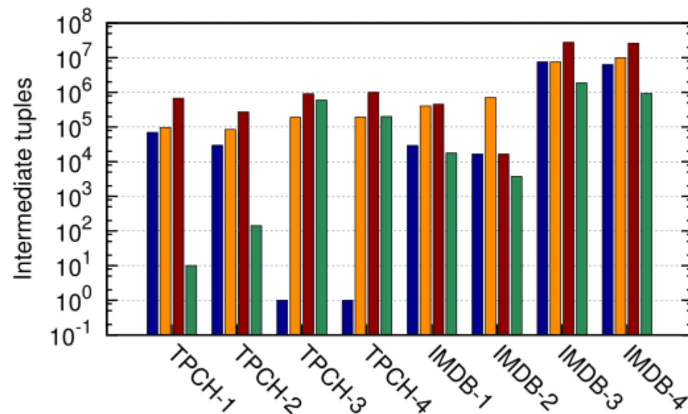


p.854 より引用

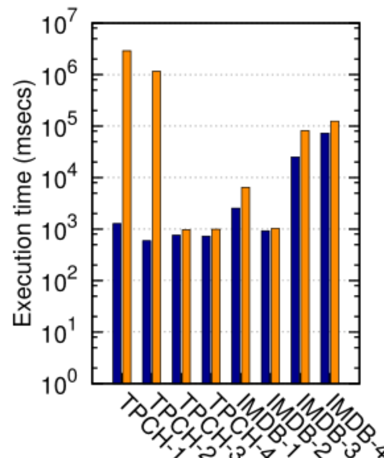
(b) Bayesian network.

評価実験

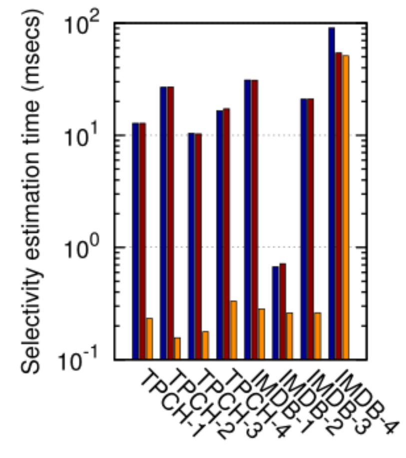
- PostgreSQL 上に，グラフィカルモデルを用いた選択率見積もり方式を実装
  - 実際の DBMS に実装した例はこれが初めてだろうとの主張
- グラフィカルモデルの構築は DBMS の外で（Java による実装）
  - PostgreSQL 内のデータは，クエリを投げて取得
  - TPC-H (scale = 1.0) や IMDB データセットなら一時間程度
- 選択率の見積もりは PostgreSQL 内で
  - 1 ~ 3 msec 程度で計算は行える



(a) Optimizer cost, as number of intermediate tuples generated. Estimated and actual costs are shown.



(b) Execution time.



(c) Time for selectivity estimation.

p.859 より引用

Active Complex Event Processing over Event Streams

Di Wang (Worcester Polytechnic Institut), Elke Rundensteiner (Worcester Polytechnic Institute), Richard Ellison III (University of Massachusetts Medical School)

Active Complex Event Processing

CEP: 到着するイベント列から特定のパターンを検出

- 例: 「下駄箱に靴が入る」 「ドアが開く」→ 人が部屋に入ってきた
- 「パターンを検出した際に行われる処理」に対して目が向けられてこなかった

Active CEP (提案する処理モデル): CEP + Active Rule

- Active Rule という「パターンを検出した際に行われる処理」が記述可能
  - その処理の中で「システムの状態」を変更できる
- 例:
  - Q1: （電灯が消えているとき） 「下駄箱に靴が入る」 「ドアが開く」→ 「電灯をつける」
  - Q2: （電灯がついているとき） 「ドアが開く」 「下駄箱から靴が取り出される」→ 「電灯を消す」

ACEP クエリ

- ACEP クエリ: CEP クエリ + Active Rule

- **CEP クエリ**
  - SEQ によるパターン指定
  - WHERE による状態指定
    - DB の状態を読む

```
CREATE QUERY Q1 ON estream
PATTERN SEQ(EXIT, !SANITIZE, ENTER)
WHERE [HCW-ID] AND
EXIT.location != ENTER.location AND
'safe'=(SELECT status FROM workerStatus
   WHERE workerID=ENTER.HCW-ID)
WITHIN 45 sec
RETURN ENTER.HCW-ID, ENTER.location
```

- **Active Rule**
  - CEP クエリのアクション
  - DB に対する更新処理が可能

```
CREATE RULE R1
ON OUTPUT Q1
REFERENCING NEW AS newEvent
FOR EACH EVENT
BEGIN
   UPDATE workerStatus SET status = 'warning'
   WHERE workerID = newEvent.HCW-ID
END
```

p.637 より引用

- 問題
  - 複数の ACEP クエリが登録されているとき，これらは並行にシステムの状態を書き換える
  - クエリの実行順序によっては，本来検出すべきパターンが検出できないことがある

正しい実行順序を定式化し，実行スケジュール方式を提案

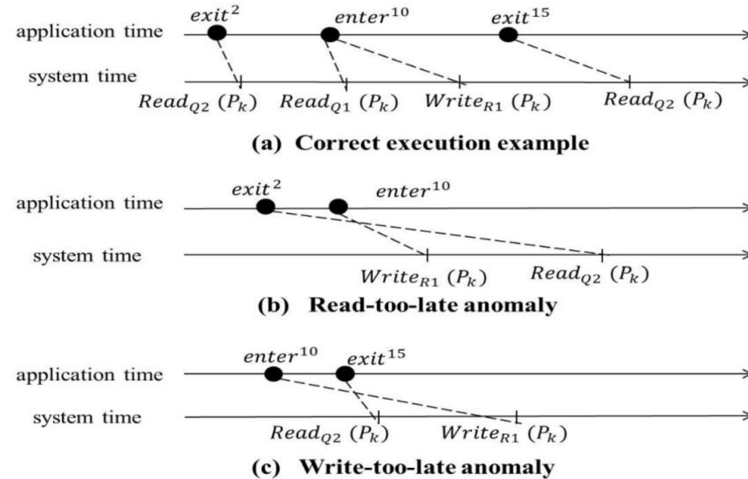
実行スケジュール / ストリームトランザクション

ACEP クエリの「正しい」実行スケジュール

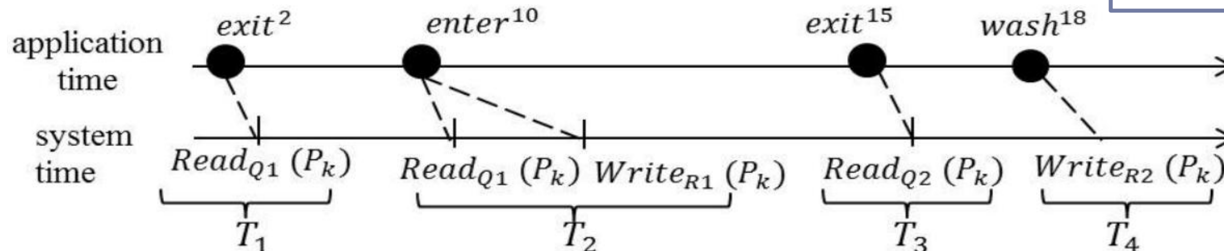
- 読み込み・書き込み操作が，トリガとなるイベントの**アプリケーション時刻**順に行われる

ストリームトランザクション (S-Txn)

- あるイベントをトリガとし，システムの状態に対して行われる読み込み・書き込み操作の列
- 読み込み:
  - ACEP クエリの条件式 (WHERE 節) に出現
  - パターンを検出している間，イベントが来るたびに読み込みが生じる（毎回 WHERE 条件を満たしているか確認されるため）
- 書き込み:
  - ACEP クエリの Active Rule (パターンを検出した際の処理) に出現



アプリケーション時刻: イベントの発生した時刻
システム時刻: イベントが CEP システムに到着した時刻



ストリームトランザクションの同時実行制御

- 3 つの同時実行制御方式に言及
  - ※イベントがアプリケーション時間の順番に到着するという仮定を行なっている

1. ナイーブな方式
   - イベントが一つ届くたび，そのイベントによってトリガされる Active Rule を全て処理し，それらが終わると次のイベントの処理に移る
   - **欠点:** コンフリクトしない S-txn の実行も遅らされてしまい，効率が悪い
2. Strict 2-Phase Lock (S2PL)
   - トランザクション処理でよく用いられる S2PL を適用する
   - **欠点:** Lock-Compatibility が厳密すぎる
3. Low-Water Mark (LWM)
   - データ（システムの状態）のバージョニング + ロックの利用
   - Lock-Compatibility を緩める

S2PLのLock Compatibility (Held = 行, Request = 列)

Held \ Request	R	W
R	O	X
W	X	X

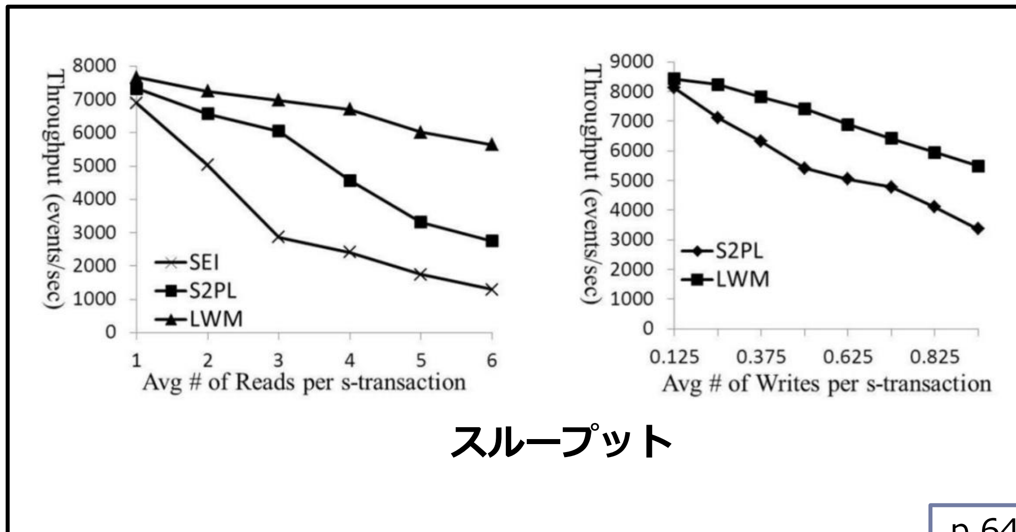
ここを O にすると R が後に読もうとする値が W により上書きされてしまうので，バージョニングにより R が後で読もうとするデータを保持しておくことで対応

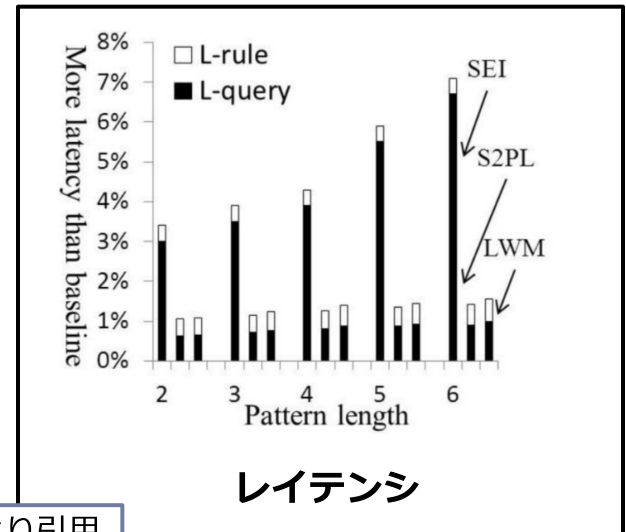
R が読み込みを行おうとしたアプリケーション時刻のほうが，実際に書き込み W が行われるアプリケーション時刻よりも早い場合は，ロックがかかっていようと読み込んで良い

LWMのLock Compatibility (Held = 行, Request = 列)

Held \ Request	R	W
R	O	O
W	△	X

評価実験





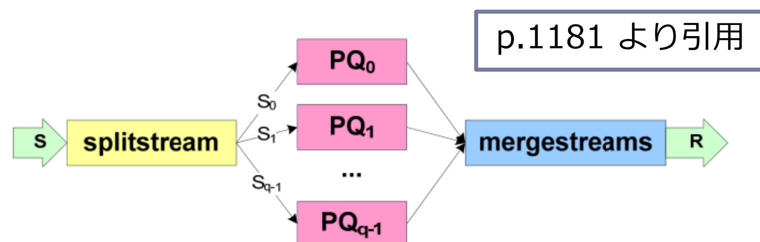
p.640 より引用

- 環境
  - HP lab. の CHAOS CEP engine [9] に ACEP と各同時実行制御手法を実装
- スループット
  - R / W 共に Low-Water-Mark が他よりも高いスループットを示す
  - SEI (ナイーブな方式) は全 Txn を直列に実行するため，スループットは最も悪い
- レイテンシ
  - LWM はバージョニングなどにリソースを消費するため，オーバーヘッドが LWM よりやや大きくなる

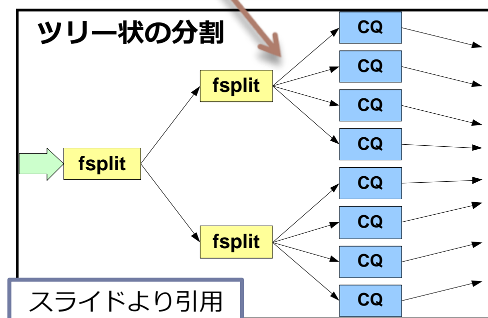
Massive Scale-out of Expensive Continuous Queries

Erik Zeitler (Uppsala University), Tore Risch (Uppsala University)

ストリーム処理のスケールアウト



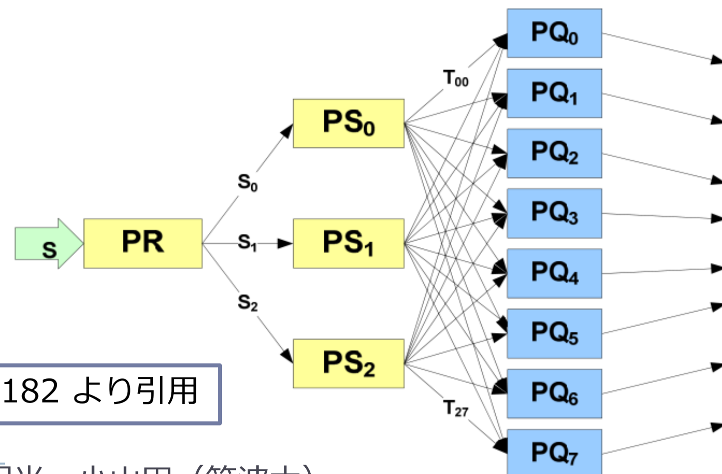
- 背景
  - 大量・頻繁にストリームデータが流れてくる
  - Continuous Query の内容も重たいことがある
  - 単一のノードでは処理が追いつかない
  - スケールアウト（データ並列）を行う場合ストリームデータを分割する必要あり
- 問題（先行研究での対応）
  - 問題1: どのようにストリームデータを分割するか
    - アプリケーションによって分割のポリシーは異なる（ルーティング，ブロードキャスト）
    - 対応: **ユーザが分割の仕方を指定できるように，クエリ言語を拡張**
      - ルーティング関数(tuple, N): tuple の行き先ノード 0 ~ N-1 を返す
      - ブロードキャスト関数(tuple): tuple をブロードキャストして欲しい場合 true を返す
  - 問題2: 分割をする処理自体がボトルネックとなりうる
    - 分割数を増やしていった場合の影響が顕著
    - 対応: **ツリー状の分割 (maxtree)**
      - 問題: ルーティング関数，ブロードキャスト関数の処理が重いと パフォーマンスが下がっていってしまう



- 本研究
  - Parasplit という分割方式を提案
    - ルーティング関数，ブロードキャスト関数のコストに関係なく，良い性能を示す

Parasplit

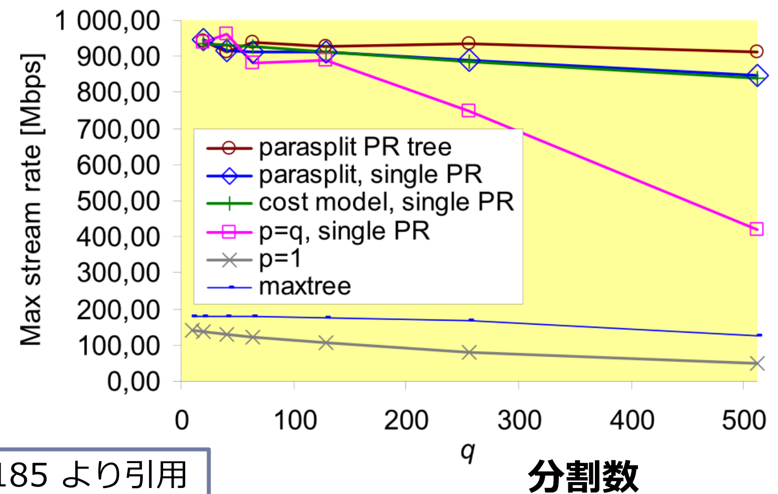
- ツリー状の分割の問題
  - ルーティング関数，ブロードキャスト関数がスケールアウトされていない
- 提案方式: Parasplit
  - ルーティング関数，ブロードキャスト関数までスケールアウトさせる
  - **手順**
    - PR:
      - 入力ストリームを一定の時間ごとにウィンドウへと区切る (Physical Window)
      - 区切られたウィンドウがはPS (Window Splitter) へと一様かつランダムに分配される
    - PS (Window Splitter):
      - 各 PS は，ウィンドウ中のタプルを PQ (Query Processor) へと送信
      - ここで，ユーザの指定したルーティング関数，ブロードキャスト関数が使われる
  - 必要に応じて PR を多段にすることも可能 (Parasplit PR Tree)



p.1182 より引用

評価実験

- 環境
  - 70 台のマシンからなるクラスタ
  - プロセッサ: Intel Xeon E5520 @2.27GHz (L2 キャッシュ 8MB)
  - クラスタ内のノード間通信は TCP/IP （ギガビット・イーサネット上）
- 内容
  - Linear Road Benchmark (LRB)
    - ストリーム処理のベンチマーク仕様として有名なものの一つ
    - 研究グループによる scsq-plr という LRB 実装を利用
- 結果
  - Parasplit が CPU バウンドでなく，ネットワークの帯域バウンドであることが確かめられた



p.1185 より引用