ICTイノベーションフォーラム2012
2012年10月2日，幕張メッセ・国際会議場

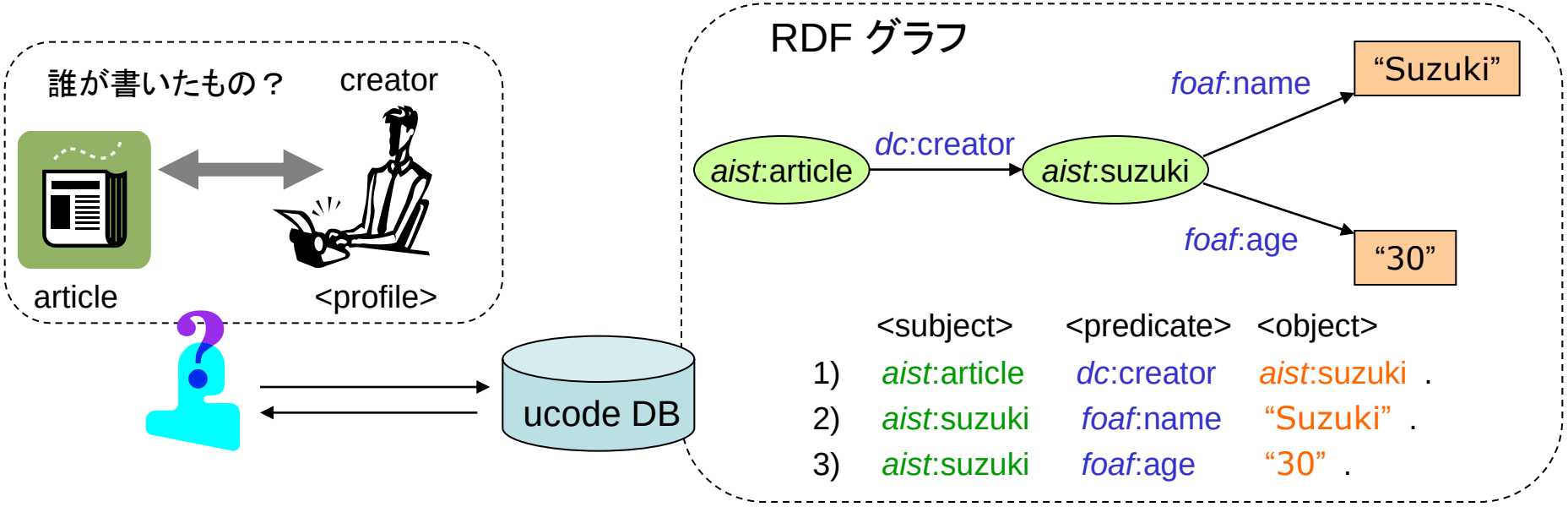
次世代大規模分散・並列環境における高度メタデータ管理・解析システム技術の研究開発

谷村勇輔，Steven Lynden，的野晃整

独立行政法人 産業技術総合研究所

研究背景

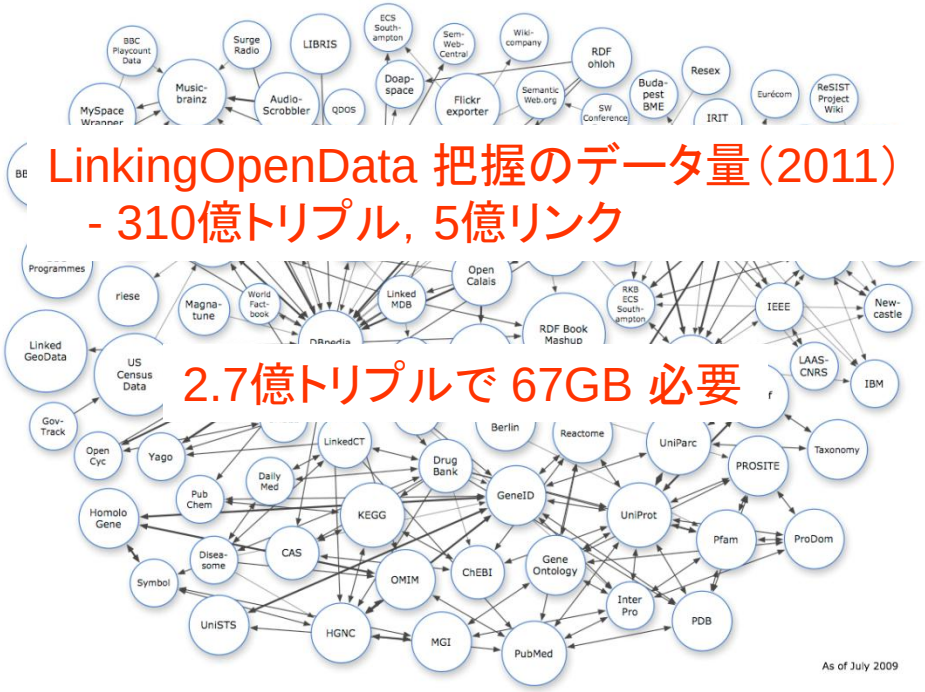
- 様々なコンテンツやコンテキストを説明するには**「メタデータ」**が不可欠
  - 高度な知識ベースの構築
  - ユビキタスコンピューティングの世界
    - あらゆる物や場所の全てに “ucode” と呼ばれる ID を付ける



研究背景(続き)

- RDF (Resource Description Framework)
  - W3C において標準化
  - メタデータの表現方法として有望
- RDF の課題
  - データ規模の増大
    - トリプル数
    - サイズ
    - メタデータ／データ混合
  - 高い処理コスト
    - 部分グラフ検索
    - 経路探索

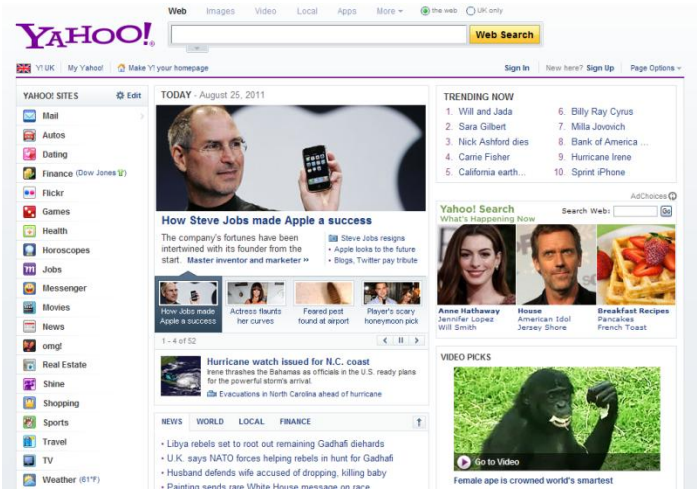
    など



研究背景(続き)

- “Big Data” 処理プラットフォームの進歩と普及
  - MapReduce ＋ 分散ファイルシステム
    - 2000年代前半の Google の論文発表
  - オープンソースのソフトウェア “Hadoop”
    - Yahoo による貢献，活用事例
      - ペタバイト規模のデータ
        - » Web インデックス
        - » 広告，ニュース，ショッピング
        - » 個人メール，写真，動画

        など
      - 億単位のトランザクション
        - » 顧客管理，個人広告
        - » スパムフィルタリング
        - » ログ解析



研究目的

- 新しい分散・並列処理基盤（Big Data Platform）をもとに大規模 RDF データ処理を実現する

- 目標実現のための要素技術開発
  - 方針
    - Pig/Hadoop を基盤に用いる.
    - RDF データの特徴を調べ，それを最大限に活用する.
  - サブ課題（各要素技術）

    A）データ格納方式と並列データ処理の高速化手法

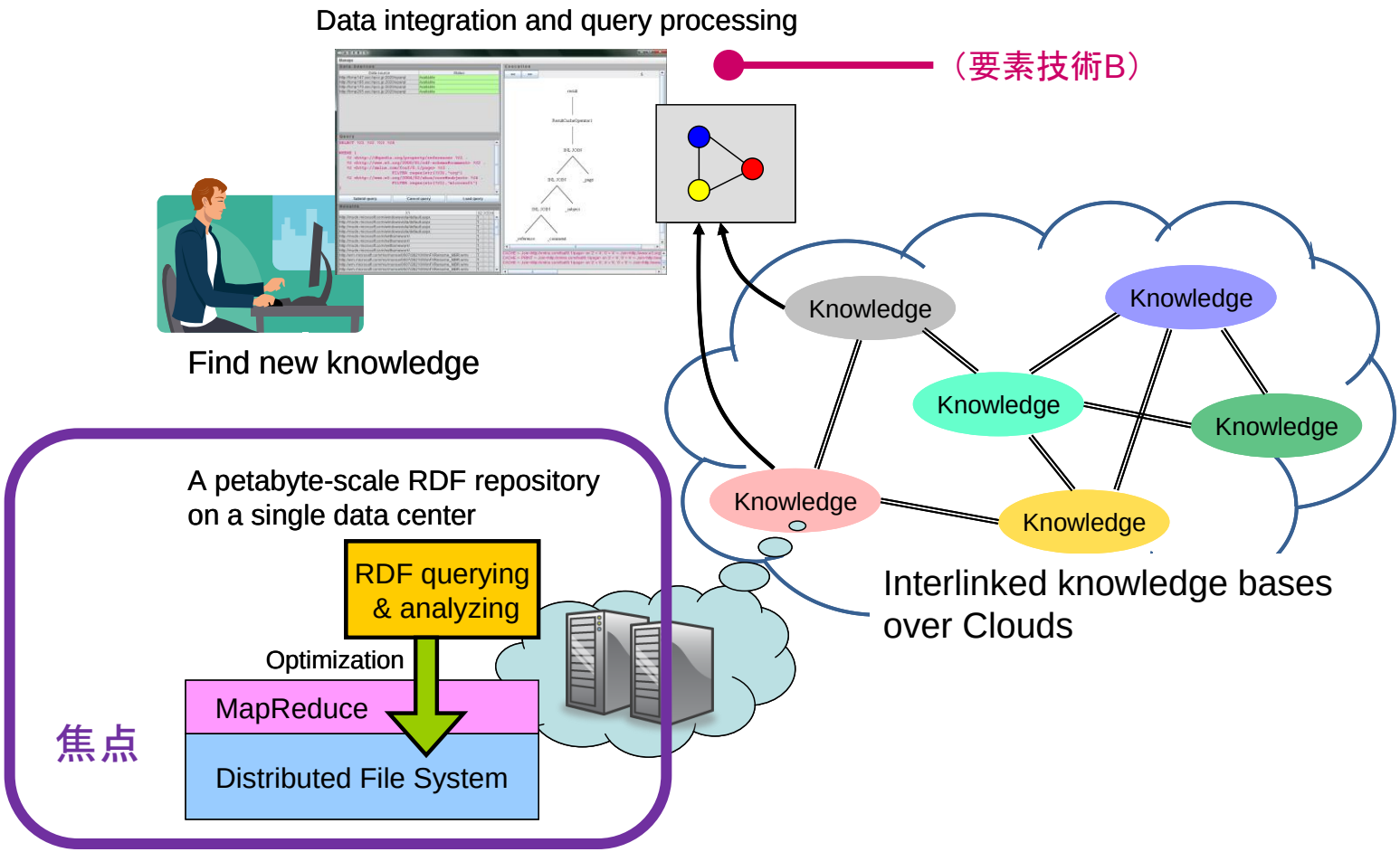
    B）動的な問合せ・解析処理の最適化手法

    C）効率的な中間結果の構築手法

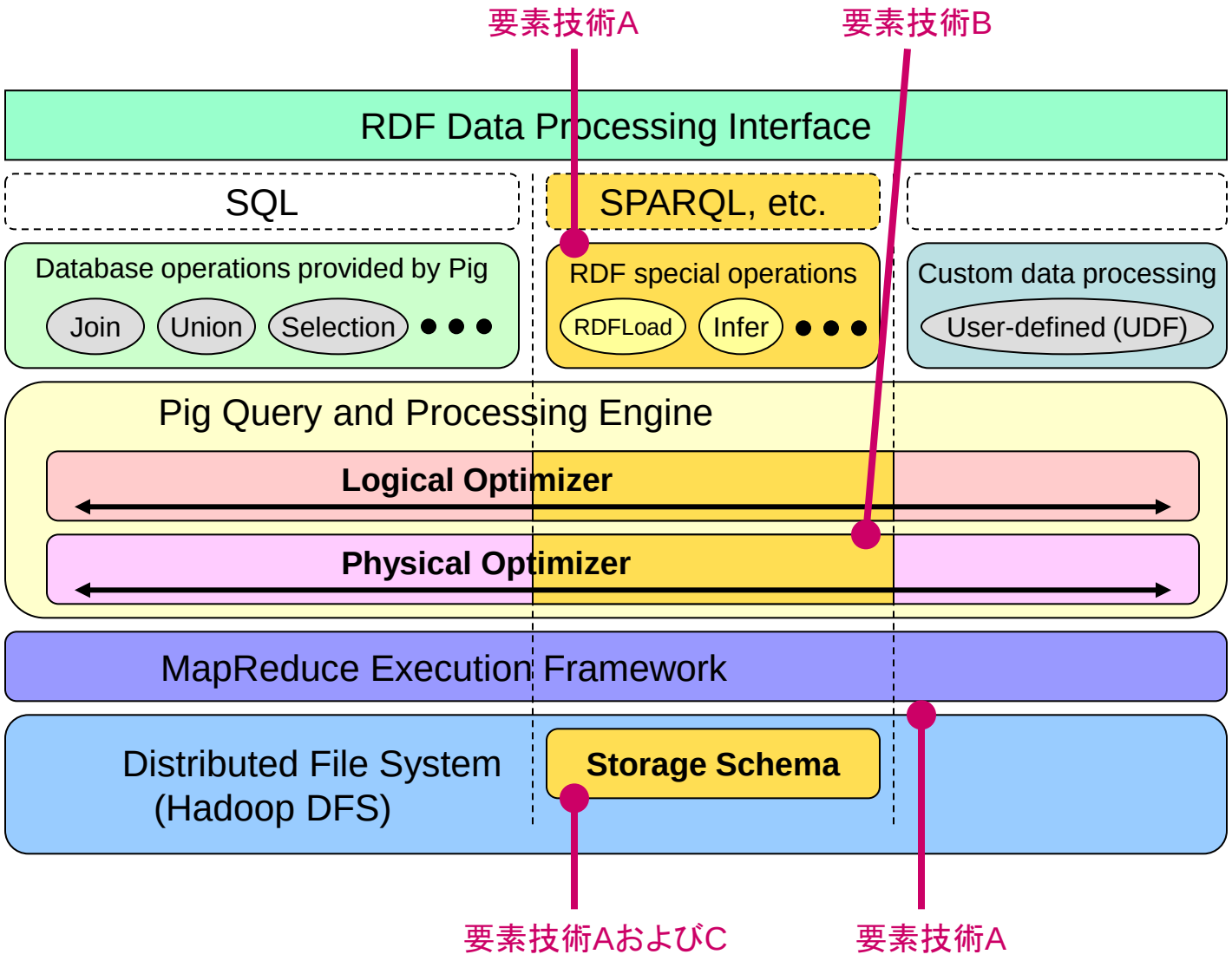
    およびこれらの統合利用の手法

想定システムアーキテクチャ

- 高度知識ベース: 複数の RDF リポジトリの連合



想定システムアーキテクチャ（続き）



A-1）データ格納方式

- VP-MapFile の開発
  - Vertical Partitioning (VP)
    - Predicate 毎に HDFS 上にファイルを用意
  - MapReduce で扱いやすいバイナリ形式（MapFile）
    - Index を持ち，Key-value pair の高速検索，圧縮が可能

	サイズ [MB]	サイズ比 [%]	0.17MB 抽出時間 [sec]	56MB 抽出時間 [sec]
N-Triplesフォーマットの元データ	318.2	100	—	—
N-Triples に基づくタブ区切りのテキスト	317.9	99.89	57.1 (5p: 25.6)	67.2 (5p: 27.2)
VPフォーマット：テキスト，非圧縮	219.5	68.97	13.0	22.7
VPフォーマット：MapFile	236.8	74.41	12.7	32.0
VPフォーマット：MapFile，LZO圧縮	36.9	11.59	12.5	24.8

A-1）データ格納方式（続き）

- VP-MapFile に対応した Pig 向けの RDFLoader モジュール（built-in function) の開発

```
a = LOAD ‘DBpedia’ USING RDFLoader(‘?predicate = http://www.w3.org/ ....’);
```

  - SPARQL Filter 節により，不要データを読み飛ばせる.

- VP-MapFile へのフォーマット変換ツール
  - OpenRDF Sesame を利用し，MapReduce により処理
  - 2500万トリプル（N-Triples形式で4GB）を 524 [sec] で VP-MapFile（圧縮）に変換可能
    - 96 CPU コアの Hadoop 0.20 クラスタを利用した場合

“ 提案システム内の HDFS へのインポート → VP-MapFile へ変換 → Pig による処理 ” の一連の操作の高速化を行った.

A-2）データ処理の高速化

- SP²Bench ベンチマークを用いた RDF データ処理性能の検証（96 CPUコアの Hadoop クラスタ）
  - SPQRQL の SP²Bench を Pig Latin に移植
    - RDF Loader モジュールによるチューニングも実施

	2500万トリプルに対するクエリの実行時間 [sec]		
	Query 5a	Query 5b	Query 6
プロトタイプシステム（圧縮なし）	447	368	842
プロトタイプシステム（圧縮あり）	301	240	350
ARQ v2.2 / Jena 2.5.5	Timeout (30分)	Timeout (30分)	Timeout (30分)
Sesame v2.2beta2 (in-memory)	Memory Exhaustion	Timeout (30分)	Memory Exhaustion
Sesame v2.2beta2 (Mulgara SAIL v1.3beta1)	Timeout (30分)	30分以内	Timeout (30分)
Virtuoso v5.0.6	Loading Failure	Loading Failure	Loading Failure

- Reasoning（Transitive Closure）処理の高速化
  - Join アルゴリズムや MR パラメータの受動的最適化

B）動的な問合せ・解析処理の最適化手法

- Adaptive RDF Distributed Query Processing
  - 複数の RDF データリポジトリに対する問合せ
  - 手法を実装したソフトウェア（ADERIS）は公開
    - http://code.google.com/p/sparql-aderis
- Dynamic data re-partitioning for RDF joins
  - 提案アーキテクチャにおける問合せに関して，受動的最適化の適用による効率化を検討
    - 要素技術 A）との連携に関する研究
    - 提案：MapReduce における多段 Join の高速化

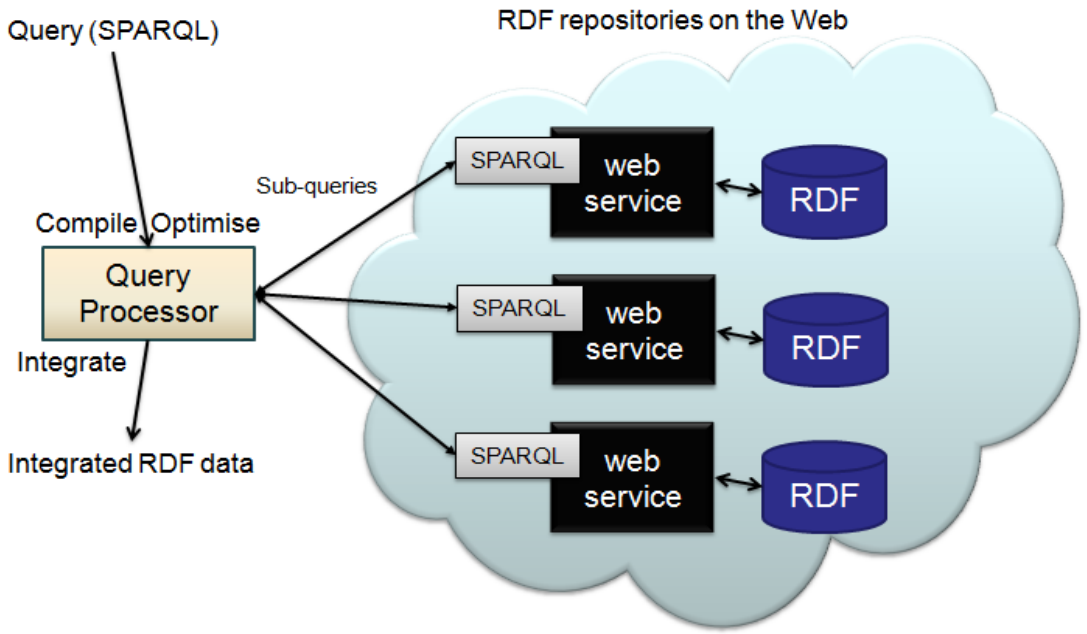
Distributed query processing over RDF data

Aim

- Provide a way of integrating RDF data, relying on the SPARQL query language, protocol and Linked Data.

Approach

- Input = SPARQL query
- The system retrieves the data required from multiple SPARQL endpoints and integrates the data
- the entire process is automatically executed
  - Declaratively
  - Adaptively



Adaptive RDF Distributed Query Processing

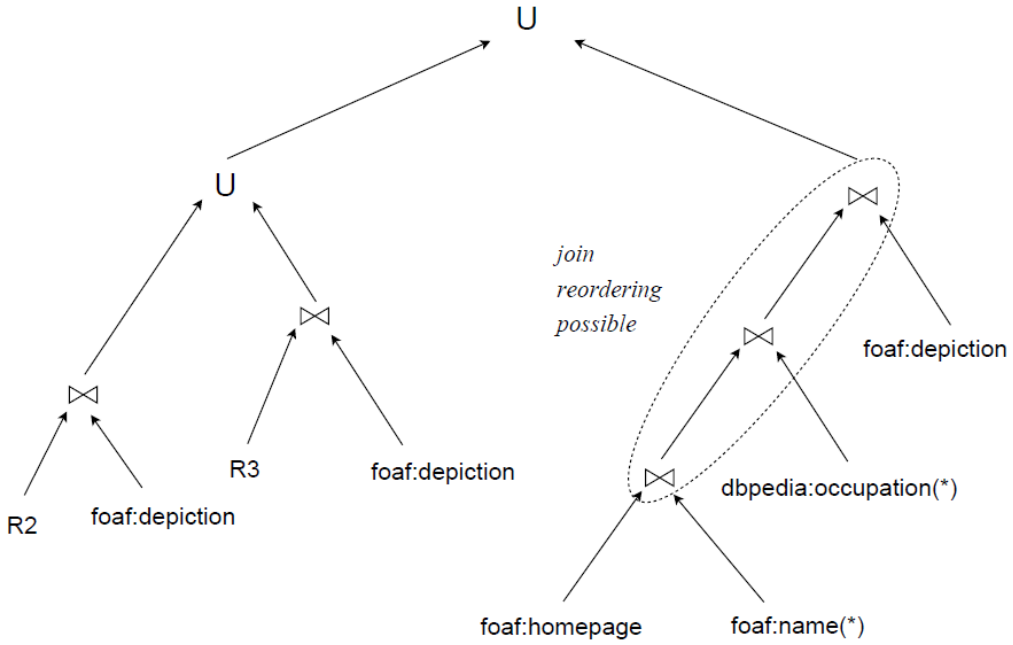
Dynamically retrieve RDF data from distributed databases

- Continually issue sub-queries to individual databases; configure the query plan at run-time depending on the results.

First adaptive query processing technique for RDF data

- Recent work on switching join order in pipelined relational query plans provides a way to switch join order without throwing away results.

Example: dynamically joining RDF data from multiple sources - Adaptive query processing is used to switch join order and reduce query processing time.



Example Results

```
dbo: <http://dbpedia.org/ontology/>
dbp: <http://dbpedia.org/property/>
owl: <http://www.w3.org/2002/07/owl#>
rdf:  <http://www.w3.org/2000/01/rdf-schema#>
skos: <http://www.w3.org/2004/02/skos/core#>
foaf: <http://xmlns.com/foaf/0.1/>
```

Query 1 (Result size = 150):

```
select * where {
 ?x dbp:reference ?ref .        777,679
 ?x rdf:comment ?comment .      10,000
 ?x skos:subject ?subj .        9971
 ?x foaf:page ?page .           10,000
 ?x rdf:type ?type .            800,000
 FILTER ( regex(str(?subj),"building") )
}
```

Query 2 (Result size = 8):

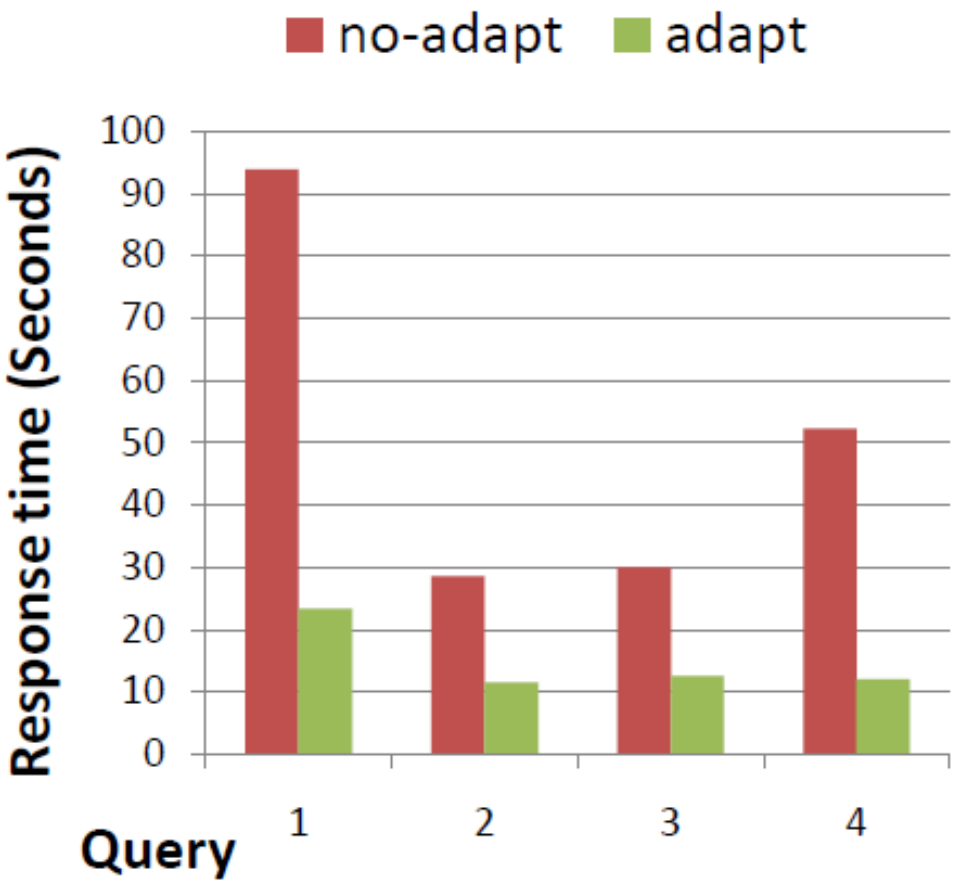
```
select * where {
 ?x dbp:reference ?ref .        777,679
 ?x rdf:comment ?comment .      10,000
 ?x skos:subject ?subj .        9971
 ?x foaf:page ?page .           10,000
 ?x rdf:type dbo:book           3105
}
```

Query 3 (Result size = 8):

```
select * where {
 ?x dbp:reference ?ref .        777,679
 ?x rdf:comment ?comment .      10,000
 ?x skos:subject ?subj .        9971
 ?x foaf:page ?page .           10,000
 ?x rdf:type dbo:book           3105
 ?x dbo:releaseDate ?date       (DBP) 126,737
}
```

Query 4 (Result size = 13):

```
select * {
 ?book rdf:type dbo:Book .      3105
 ?book foaf:page ?p .           10,000
 ?book owl:sameAs ?link         (DBP) 10,121,699
}
```

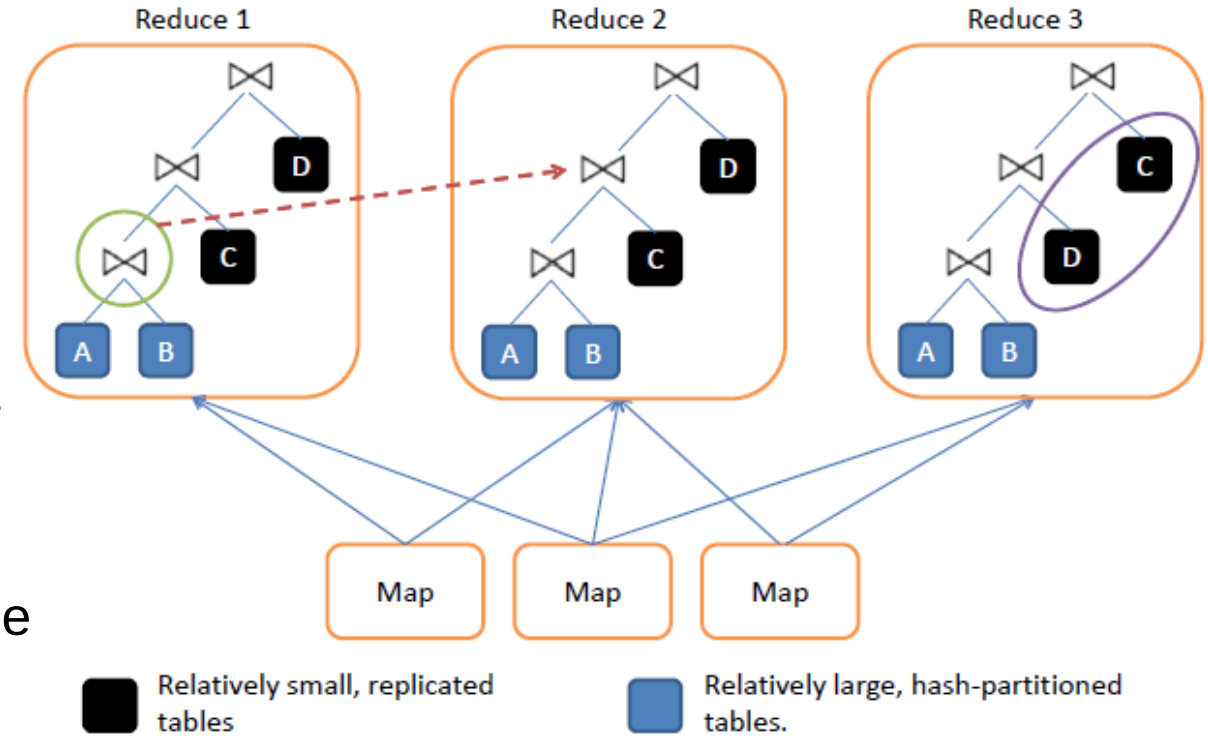


Dynamic data re-partitioning for RDF joins

- **MapReduce is increasingly being used to analyse large datasets**
  - Easy to program, Parallelism, Fault tolerance, etc.
- **Joins**
  - Fundamental operation in many applications, therefore optimisation is important

Dynamic re-distribution

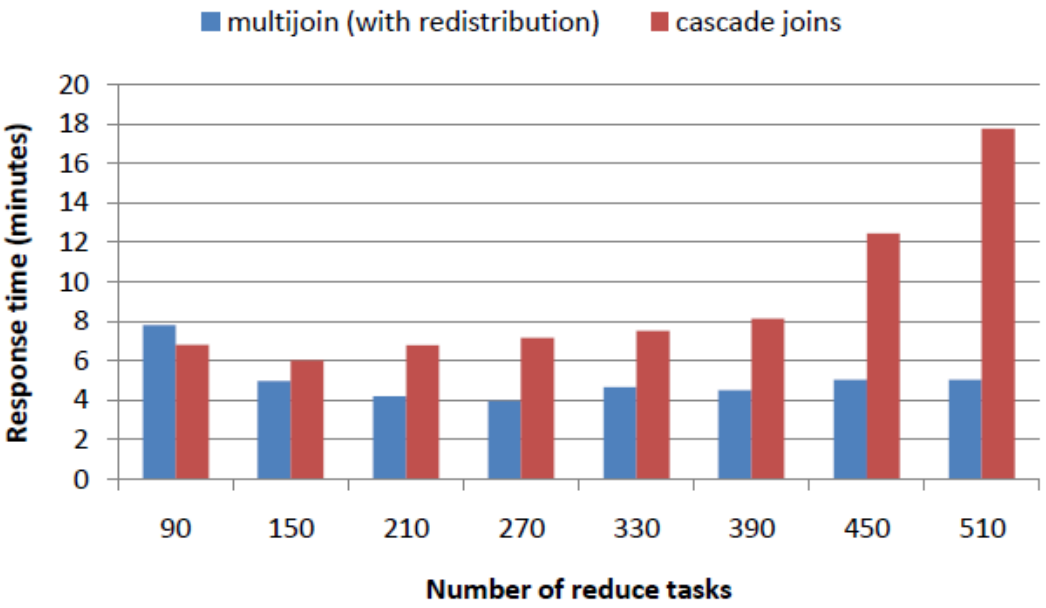
Use Adaptive Query Processing-based monitoring (e.g. join selectivity) to trigger re-distribution of data in RDF joins. Data from slow reduce tasks is sent to faster ones to speed up the overall processing time of a sequence of joins.

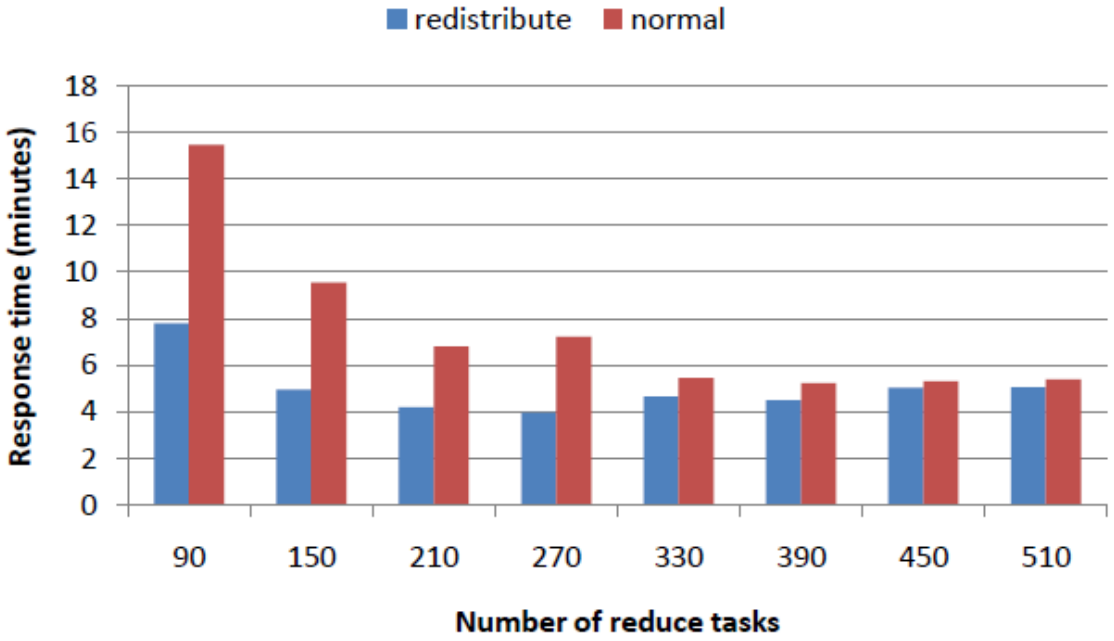


Example Results

Separate vs Multiple-joins

Using the proposed technique to join multiple RDF tables was faster on average than the “cascade” (sequences of separate joins) approach in most cases.





Static vs Dynamic

Re-distributing data between reduce tasks showed improvements over normal Hadoop scheduling in our experiments.

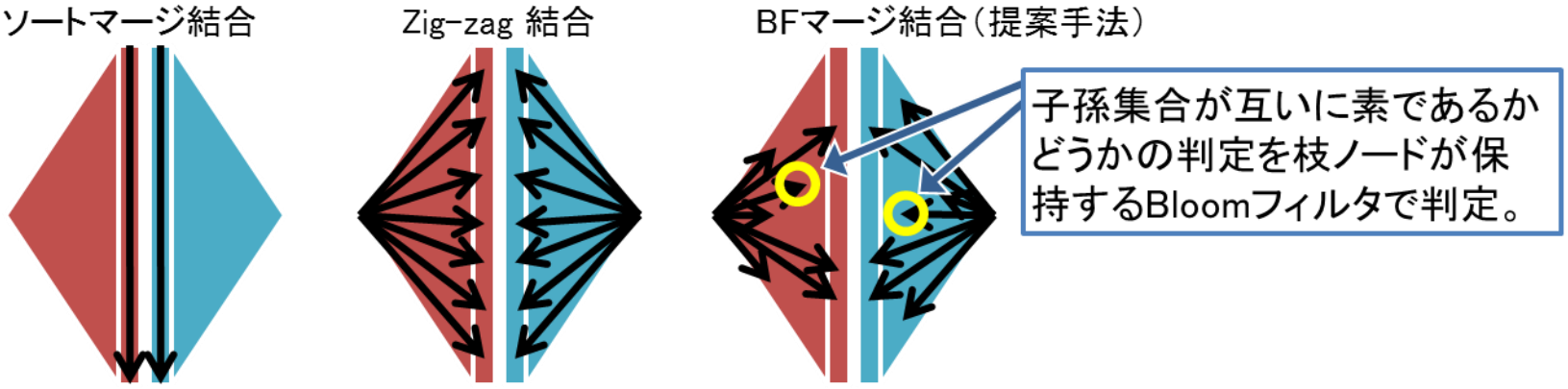
C）効率的な中間結果の構築法

- “低選択な結合”のための結合アルゴリズム
  - 問題：RDF では入力データが膨大であり，結合結果が少ない結合（低選択な結合）が頻出する
  - 解決案：結果に含まれないデータを予測し，読み飛ばすことで I/O を減らす

- RDF 文書の構造に着目した格納法
  - 問題：従来法では，RDF データを細切れにして格納し，検索時にそれらを結合するため，結合が頻出する
  - 解決案：あらかじめ結合して格納することで，結合回数を減少させる
    - どこを結合するかの決定に RDF 文書の構造を利用

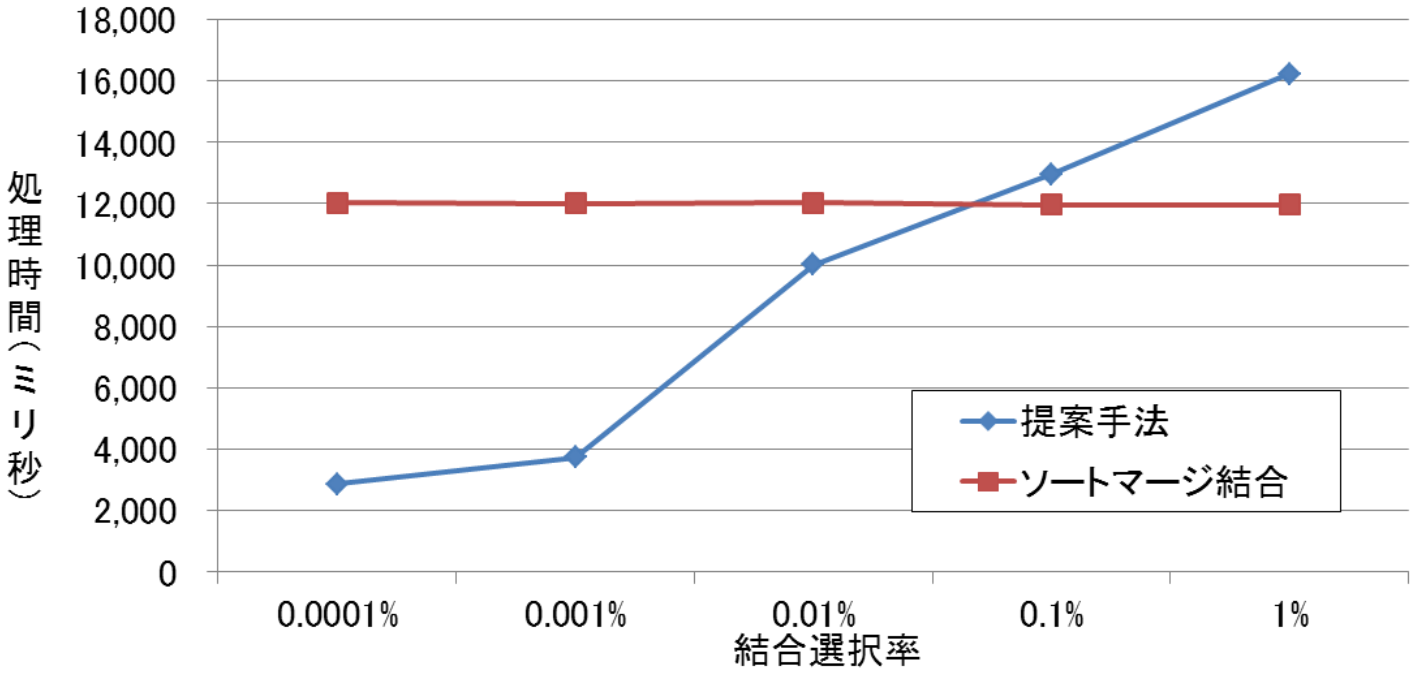
低選択な結合のための結合アルゴリズム

- 前提：2つの B+ 木を突き合わせ，共通集合を求める結合
  - ソートマージ結合（典型的な結合手法）：読み飛ばしなし
    - 全ての葉ノードを1度スキャンする
  - Zig-zag 結合：不要な葉ノードの読み飛ばしが可能
    - 葉ノード同士を比較して，結合可能かどうかを判断
  - BF マージ結合（提案手法）：不要な枝ノード以下の全ノードの読み飛ばしが可能
    - 枝ノード同士を比較して，結合可能かどうかを判断
    - 各ノードに Bloom フィルタを持つように拡張した B+ 木を使用



提案手法と従来法の比較

- 実験データ
  - ✓ 重複のない100万要素のデータを疑似乱数を用いて，2つ作成
  - ✓ 解である共通集合データを別途作成し，上記2つに混ぜる（実験ではこれらを結合）
    - – 結合選択率が 1% の場合，結果の共通集合は1万要素となる



結合選択率が低い場合に提案手法は効率的である.
最大4.2倍の高速化を達成した.

RDF文書の構造に着目した格納法

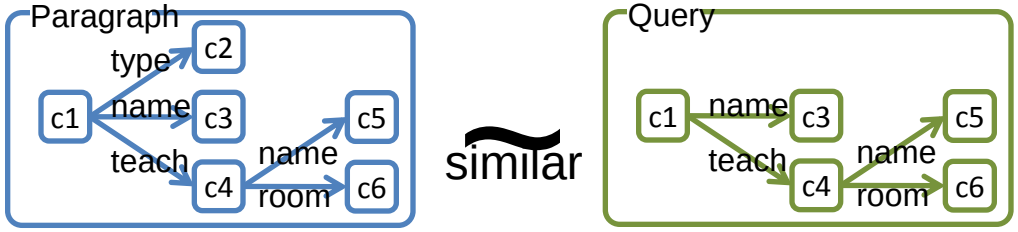
RDF documents

```
<rdf:RDF ...>
  <FullProfessor rdf:about="id:codd">
    <name>Ted</name>
    <teach rdf:resource="id:course1">
      <name>AAA</name>
      <room>0123</room>
    </teach>
    <teach rdf:resource="id:cource2">
      <name>BBB</name>
      <room>0124</room>
    </teach>
    <email>ted@...</email>
  </FullProfessor>
  <AssistantProfessor rdf:about="id:jim">
    :
  </AssistantProfessor>
  :
  <Paper rdf:about="id:xxx">
    <name>XXX</name>
    <author rdf:about="id:codd"/>
  </Paper>
  <Book rdf:about="id:yyy">
    <name>YYY</name>
    <author rdf:about="id:codd"/>
    <author rdf:about="id:jim"/>
  </Book>
</rdf:RDF>
```

RDF paragraphs

- RDF 文書
  - 構造化言語で記述されている
  - 幾つかの塊（RDF paragraph と呼ぶ）で記述されている
  - 人の可読性の考慮して決められた
- RDF 問合せ
  - 問合せ言語は構造的な記述である
  - RDF 問合せは人が発行することが多い

仮説：RDF paragraph と RDF query の構造は類似

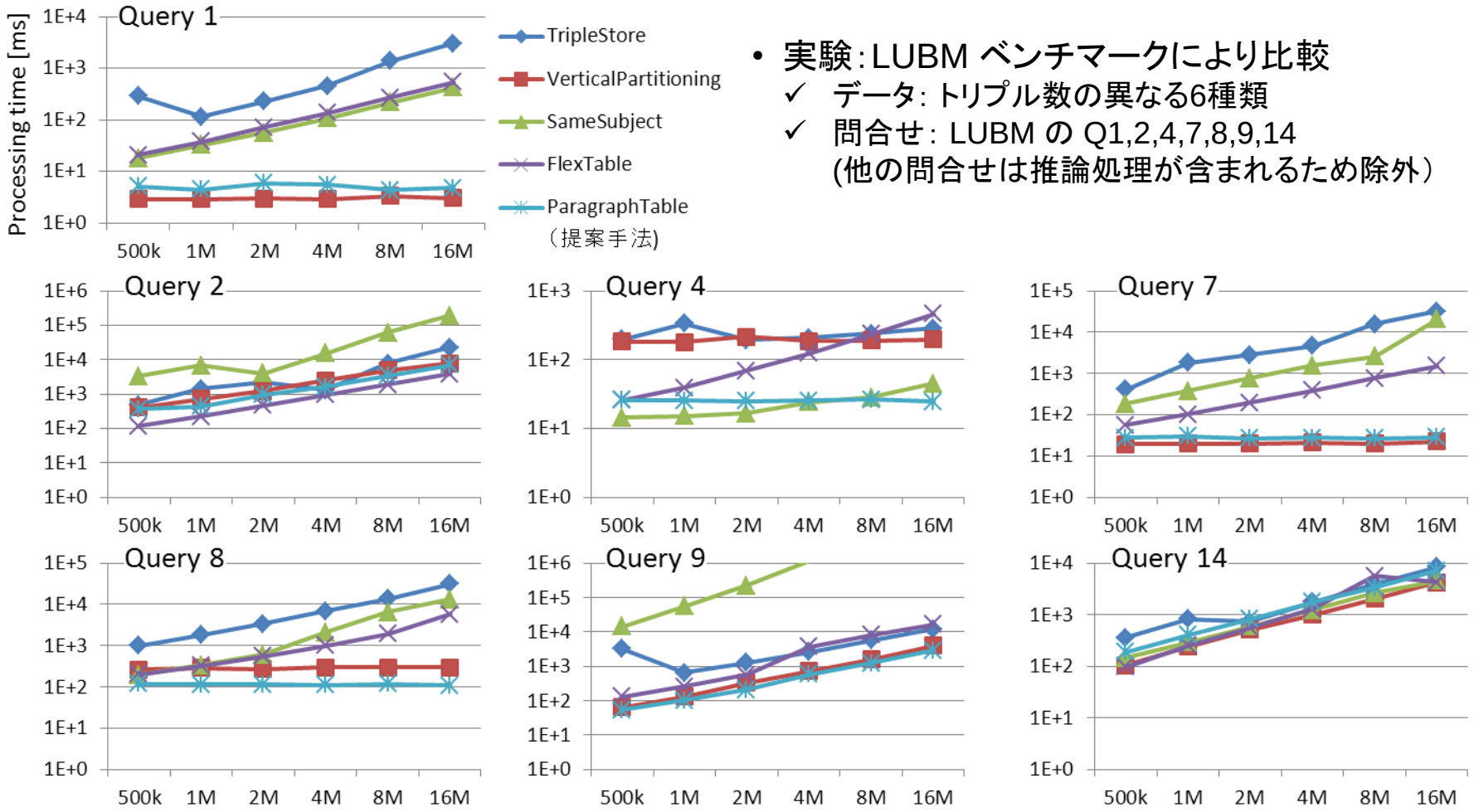


c1	c2	c3	c4	c5	c6	c7
id:codd	FullProfessor	Ted	id:course1	AAA	0123	ted@...
id:codd	FullProfessor	Ted	id:course2	BBB	0124	ted@...
id:jim	Assistant Professor					

c1	c2	c3	c4
id:xxx	Paper	XXX	id:codd
id:yyy	Book	YYY	id:codd
id:yyy	Book	YYY	id:jim

RDF paragraph をそのまま格納する
→ 検索される構造がすでに格納済み
→ 検索時の結合が減少
→ 検索性能が向上

提案手法と従来法（4種類）の比較



- 実験：LUBM ベンチマークにより比較
  - ✓ データ：トリプル数の異なる6種類
  - ✓ 問合せ：LUBM の Q1,2,4,7,8,9,14
    (他の問合せは推論処理が含まれるため除外）

問合せ Q1,4,7,8 において，提案手法はトリプル数に対するスケーラビリティがある．
ほとんどの問合せにおいて，提案手法は1番か2番目に高速である．

まとめ

- 大規模 RDF データ処理の基盤を提示し，その実現に必要な要素技術の開発を行った.
  - A）データ格納方式と並列データ処理の高速化手法
  - B）動的な問合せ・解析処理の最適化手法
    - → Pig/Hadoop での実装，実際的な評価
      - 従来困難であった規模のデータ処理を可能にした
      - 分散ストレージと MapReduce を用いた先駆的な成果
  - C）効率的な中間結果の構築手法
    - → 基礎的な研究成果
      - RDF データ処理手法としての学術的な新規性が高い
      - 今後提案システムへの適用を検討

今後に向けて

- RDF データの応用分野の開拓・普及
  - 期待：扱えるデータ規模の拡大による効果
    - より意味のある問合せ，解析が可能になる
    - これまで想像できなかったような知見が得られる
  - 課題：「メタデータを誰がどのように生成するか？」という問題の克服
- システムの実運用の課題
  - すべての提案手法の組み込み
  - Pig/Hadoop のアップデートへの追従
  - クラウドサービスとしての運用