パブリッククラウド環境を用いた 動的演算資源調整手法の実行性能への影響機序の分析

奥野　晃裕† 　早水　悠登† 　合田　和生† 　喜連川　優†,††

† 東京大学 生産技術研究所 〒 153-8503 東京都目黒区駒場 4-6-1
†† 国立情報学研究所 〒 101-8430 東京都千代田区一ツ橋 2-1-2
E-mail: {okuno,haya,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし　著者らは共有ストレージ型データベースエンジンにおける動的演算資源調整手法の研究に取り組んで．当該手法は共有ストレージ型データベースエンジンにおいて実行中クエリの割当て演算資源の調整を可能とし，著者らはこれまでにパブリッククラウド環境における評価実験によって提案手法の有効性を示した．提案する動的演算資源手法の応用に向けた研究においては当該手法の実行が俊敏に完了することが重要となるが，当該手法の実行性能を決定する要因については十分に明らかになっていない．本論文では，パブリッククラウド環境を用い動的演算資源調整手法の実行時に精緻な観測を行う実験的評価によって，当該手法の実行性能への影響機序を明らかにする．

キーワード　データベースエンジン, 動的演算資源調整, パブリッククラウド

1 はじめに

IT システムの構築においてパブリッククラウドと呼ばれるサービスを計算機環境として利用する形態が広まっている．パブリッククラウドは，ユーザがネットワークを介して仮想化された演算資源やストレージなどを利用するサービスであり，Google や Amazon などの事業者によって提供されている．パブリッククラウドのシェアは 2017 年時点で 25.7%であり，今後は更に利用が拡大すると予測されている [1]．パブリッククラウドは，ユーザの要求に応じて迅速に資源の調達・破棄が可能であるという特徴を有し，当該特徴は資源伸縮性と呼ばれている．パブリッククラウドの資源伸縮性によって，ユーザは随時変動するビジネス要求に即応して必要な資源を確保することが可能となり，パブリッククラウドの資源伸縮性はユーザに広く支持されている．

一方，データベースエンジンは，データの管理・処理を担うソフトウェアとして，多くの IT システムにおいて重要な役割を果たしている．データベースエンジンは一般に多数のユーザから多様な性能要求のクエリを随時受け付けて実行する．パブリッククラウドの資源伸縮性を利用し，データベースエンジンへの資源割当てを変更することによって，随時変動する要求により迅速に応答可能となることが期待されるが，従来のデータベースエンジンはオンプレミス環境における一定の資源を用いて実行することを前提としており，従来のデータベースエンジンにおいてパブリッククラウドの資源伸縮性を利用することは難しい．

著者らは，データベースエンジンにおいて資源伸縮性を利用することを目的とし，共有ストレージ型データベースエンジンにおいて個々のクエリへの演算資源割当てを実行中に調整することを可能とする動的演算資源調整手法を提案してきた [2, 3]．当該手法によって，例えば長時間かかるが低優先度のクエリが全ての演算資源を利用して実行されている状況において，短時間で完了させたい高優先度のクエリの実行が要求された場合には，低優先度のクエリへの演算資源割当てを減らし，当該資源を利用することによって即座に高優先度クエリの実行が可能になるなど，クエリの実行中に随時変動するユーザの要求に柔軟に応えることが可能となる．

著者らが提案する動的演算資源調整手法では，インスタンスを単位としてクエリへの割当て演算資源の調整を行う．演算資源調整を行う際には，追加，削除の対象となるインスタンスにおいて一定の手順に則った処理を行う必要があり，演算資源調整の実行時間は当該処理に要する時間によって決定される．例えば，インスタンスの削除を行う場合，削除対象とするインスタンスにおいて既に処理待ちとなっているタプルの処理が全て完了するまで待つ必要があるため，処理待ちのタプル数が増えるにつれてインスタンス削除の実行時間が延びることとなる．動的演算資源調整に応用に向けた研究においては，調整に要する実行時間がどの程度になるかを把握することが求められるが，演算資源調整手法の実行性能への影響機序ついては十分に明らかになっていない．本論文では，提案する動的演算資源調整手法において，当該手法の実行性能へ影響を与える要因について論じ，当該手法の実行時に精緻な観測を行う評価実験によって，当該手法の実行性能への影響機序を明らかにする．

本論文の構成は以下の通りである．2 章では，共有ストレージ型データベースエンジンにおける動的演算資源調整を行う手法について述べる．3 章では，提案する動的演算資源調整手法において実行時間へ影響を与える要因について述べる．4 章ではパブリッククラウド環境を用いた評価実験の結果を示す．5 章では関連する研究について記し，6 章では本論文のまとめと今後の展望について述べる．

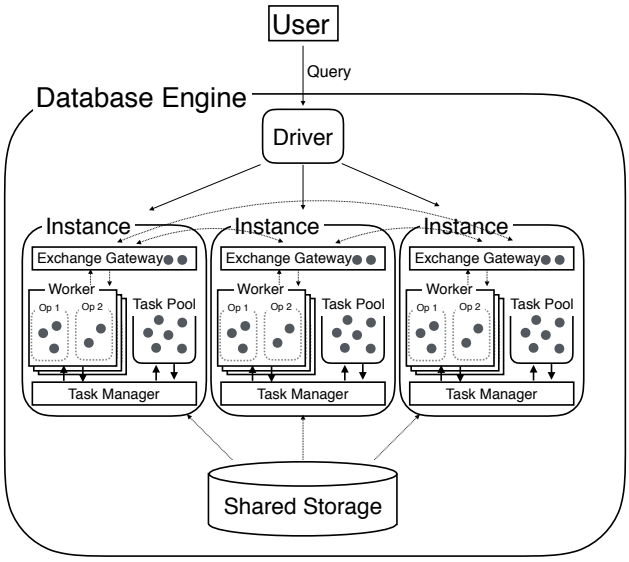


図 1: 動的演算資源調整を可能とする共有ストレージ型データベースエンジンの概要

Fig. 1 Overview of database engine with dynamic resource adjustment mechanism

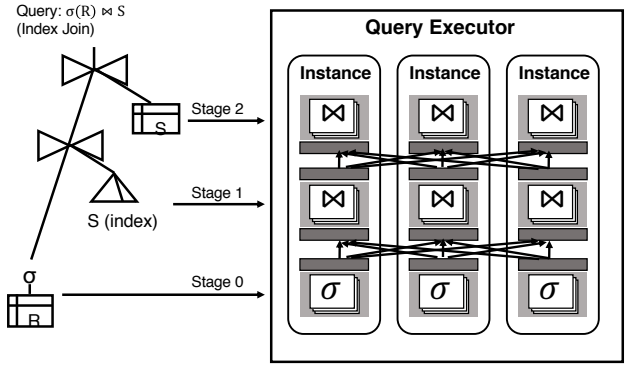


図 2: 提案データベースエンジンにおけるのクエリ実行方法の例

Fig. 2 An example of a query execution in the proposed database engine

2 動的演算資源調整機構を有する共有ストレージ型データベースエンジン

2.1 動的演算資源調整機構を有する共有ストレージ型データベースエンジンの概要

図 1 に動的演算資源調整機構を有するデータベースエンジンの概要を示す．当該データベースエンジンは，共有ストレージ型のアーキテクチャを採用しており，複数の演算資源（インスタンス）が一つのストレージを共有する．当該データベースエンジンにおいてドライバと称するプロセスがユーザからクエリを受け取るとクエリの実行が開始される．図 2 に当該データベースエンジンにおいて $\sigma(R) \bowtie S \bowtie T$ なるクエリを実行する場合のクエリ実行方法の概要図を示す．ドライバがクエリを受け取ると，当該クエリに基いて実行計画を作成し，当該実行計画をステージと称する複数のグループに分割する．ドライバは各ステージを各々のインスタンスへと配布する．各インスタンスはドライバから実行計画を分割したステージを受け取ると，ワーカと称するプロセスを開始し，当該ワーカにおいて受け取った実行計画のステージに基づく演算を実行する．入出力が必要な演算においてはワーカから共有ストレージへとネットワークを介した入出力が行われる．あるステージに基づく演算の実行に，異なるステージの演算の結果を必要とする場合があり，当該状況においては当該演算を実行する複数のワーカ間でのデータの送受信が必要となる．ワーカ間でのデータ送受信はエクスチェンジと称する機構を介して行われる．クエリの実行は複数のエクスチェンジを介した複数のワーカが連なるパイプライン構造となり，当該パイプラインにおいて，ワーカにおける各ステージの演算が並列に実行される．

2.2 共有ストレージ型データベースエンジンにおける動的演算資源調整手法

Algorithm 1 ステージ i のワーカを追加する方法

Require: i: 起動するワーカのステージ
Require: num: 起動するワーカ数
1: **function** AddWorkers(i, num)
2: $targets \leftarrow$ choose num from available instances
3: $prevWorkers \leftarrow$ workers of stage $i-1$
4: $nextWorkers \leftarrow$ workers of stage $i+1$
5: **for** $t \leftarrow targets$ **do**
6: start worker of stage i on t
7: open connections from worker on t to $nextWorkers$
8: **end for**
9: $newWorkers \leftarrow$ workers on $targets$
10: **for** $p \leftarrow prevWorkers$ **do**
11: open connections from p to $newWorkers$
12: start to send tuples from p to $newWorkers$
13: **end for**
14: **end function**

Algorithm 2 ステージ i のワーカを停止する方法

Require: i: 停止するワーカのステージ
Require: num: 停止するワーカ数
1: **function** RemoveWorkers(i, num)
2: $targets \leftarrow$ choose num from workers of stage i
3: $prevWorkers \leftarrow$ workers of stage $i-1$
4: $nextWorkers \leftarrow$ workers of stage $i+1$
5: **for** $p \leftarrow prevWorkers$ **do**
6: stop sending tuples from p to $targets$
7: close connections from p to $targets$
8: **end for**
9: **for** $t \leftarrow targets$ **do**
10: receive all tuples from $prevInstances$ on t
11: process all tuples on t
12: close connections from t to $nextWorkers$
13: stop worker on t
14: **end for**
15: **end function**

図 2 に示したクエリ実行方法において，新たなインスタンスでワーカを起動することによって実行中クエリへの演算資源の追加が可能になる．また，ワーカを実行中のインスタンスにおいてワーカを停止することによって，実行中クエリからの演算資源の削除が可能になる．各ワーカには前ステージから随時タプルが送信されるため，クエリの実行結果を正しく保ちながら演算資源の追加・削除を行うためには，ワーカの起動・停止中に当該ワーカに送信された全てのタプルについて，実行計画に基づいて正しい演算を行い，次のステージへと送信することが必要となる．

アルゴリズム 1 に演算資源の追加，即ち新たなインスタンスにおいてワーカを起動するアルゴリズムを示す．演算資源の追加においては，まずワーカを起動するとともに，当該ワーカと次ステージのワーカ間のエクスチェンジのためにコネクションを確立し，新規ワーカでタプルを受け取り処理する準備を整える．その後に，前ステージのワーカとのコネクションを確立し，前ステージのワーカからタプルの処理を開始することによって，新規ワーカにおいて全てのタプルが実行計画に基づいて正しく処理されることを可能としている．

アルゴリズム 2 に演算資源の削除，即ちワーカを実行中のインスタンスにおいてワーカを停止するためのアルゴリズムを示す．ワーカの停止においては，まず前ステージのワーカから停止するワーカへのタプル送信を停止する．その後に，停止ワーカにおいて，前ステージからのタプルを全て受信し，受信した全てのタプルが実行計画に基づいて処理されて次ステージのワーカに送信されるまで待つ．停止ワーカにおいて次ステージのワーカへのタプル送信が完了した後に，当該ワーカの停止を行う．

3 動的演算資源調整手法の実行性能への影響

前節で示したワーカ起動・停止のアルゴリズムの実行に要する時間が，動的演算資源調整の実行時間となる．ワーカの起動アルゴリズムは，主にコネクションの確立から構成されている．一方，ワーカの停止においては，前ステージから当該停止ワーカへと送信された全てのタプルの処理が完了するまで待つ必要がある．ワーカ停止において処理待ちタプルが増した場合，全ての処理待ちタプルに対する処理の完了に要する時間は長くなるため，演算資源削除の実行時間は延びるものと考えられる．処理待ちタプルは，ワーカにおいて処理を待つためのキュー内に存在する他，エクスチェンジにおいてワーカ間でタプルを送受信するためのコネクションの経路，即ちコネクションのソケットバッファや，NIC のバッファなどにも存在する．当該コネクションはクエリ実行に割当てられている全てのインスタンス間で確立されるため，コネクションの経路に存在する処理待ちタプルについては，クエリ実行への割当てインスタンス数が増えるにつれて，確立されたコネクションも増えて処理待ちタプルは増加する．そのため，ワーカ停止においては，クエリ実行への割当てインスタンス数に応じて，演算資源削除の実行時間が長くなるものと考えられる．エクスチェンジ経路のコネクションに存在する処理待ちタプルは，コネクションにおけるソケットバッファのサイズを小さくすることによってを削減することが可能であると考えられる．即ち，コネクションのバッファサイズ調整によって演算資源調整の実行時間を短縮することが可能であると考えられる．しかしながら，一般にコネクションのバッファサイズが過度に小さい場合にはネットワークのスループットが低下することが知られており，コネクションのバッファサイズの調整によって動的演算資源調整の実行時間を短縮を行う場合には，クエリ実行に十分なネットワークのスループットが得られる範囲においてバッファサイズの調整を行うことが必要となる．

表 1: AWS（N. Virginia region）実験環境諸元
Table 1 Specifications of AWS (N. Virginia region) environments

EC2 c4.8xlarge	**64 Instances**
CPU	36 vCPU
Memory	60 GiB
OS	Amazon Linux 64bit (hvm)
Hardware	Shared (Default Tenancy)
Network Bandwidth	10Gbps
Network Location	Colocated (Placement Group)

```
select p_brand,
       sum(l_quantity)
       sum(l_extendedprice * (1 - l_discount) * (1 + l_tax))
from part
join lineitem on part.p_partkey = lineitem.l_partkey
where [part selection condition]
group by p_brand
```

図 3: 評価実験に用いたクエリ: Q1
Fig. 3 Query for evaluation experiments: Q1

4 パブリッククラウド環境を用いた評価実験

4.1 動的演算資源調整機構を有した共有ストレージ型データベースエンジンの試作

2 節に示した共有ストレージ型データベースエンジンの試作実装を行った．当該試作においてドライバプロセスにインスタンス追加・削除のコマンドを設けて，ドライバを介して実行中クエリへのインスタンス割当ての変更を行えるようにした．ドライバに対して実行中クエリを指定してインスタンス追加コマンド実行すると新たなインスタンスにおいて当該クエリに基づく実行計画の全ステージを実行するワーカが起動され，実行中クエリを指定してインスタンス削除コマンドを実行すると，当該クエリに割当てられたインスタンスにおいて全てのワーカが停止される．また，当該試作においてエクスチェンジの送受信タプルを観測することによって，処理待ちタプル数の観測を可能とした．

4.2 実 験 環 境

試作実装を用いて，パブリッククラウド環境における評価実験を行った．パブリッククラウド環境は Amazon Web Service（AWS) を用い，演算資源として EC2 を，共有ストレージと

```
select o_orderpriority,
       sum(l_quantity)
       sum(l_extendedprice * (1 - l_discount) * (1 + l_tax))
from customer
join orders on customer.c_custkey = orders.o_custkey
join lineitem on o_orderkey = l_orderkey
where [customer selection condition]
group by o_orderpriority
```

図 4: 評価実験に用いたクエリ: Q2
Fig. 4 Query for evaluation experiments: Q2

```
select n_name,
       sum(ps_supplycost),
       sum(l_quantity)
       sum(l_extendedprice * (1 - l_discount) * (1 + l_tax))
from customer
join orders on customer.c_custkey = orders.o_custkey
join lineitem on o_orderkey = l_orderkey
join partsupp on l_partkey = ps_partkey and l_suppkey = ps_suppkey
join supplier on s_suppkey = ps_suppkey
join nation on s_nationkey = n_nationkey
where [customer selection condition]
group by n_name
```

図 5: 評価実験に用いたクエリ: Q3
Fig. 5 Query for evaluation experiments: Q3

して DynamoDB をそれぞれ用いた．表 1 に実験環境の諸元を示す．

データセットは，TPC-H ベンチマーク [4] の dbgen を用い，$ScaleFactor = 1000$ で生成したデータを DynamoDB のテーブルとしてロードした．また，各テーブルには TPC-H の仕様に定められた二次索引を作成した．

対象クエリとして用いたクエリを図 3，図 4，図 5 に示す．Part，Lineitem の 2 表の結合演算を行うクエリ Q1 と，Customer，Orders，Lineitem の 3 表の結合演算を行うクエリ Q2，Customer，Orders，Lineitem，Partsupp，Supplier の 5 表の結合演算を行う Q3 の 3 つのクエリを用い，それぞれのクエリの選択率は 1%となるように調整した．また，結合演算の方式は二次索引を利用したネステッドループ結合であるインデックス結合を用いた．

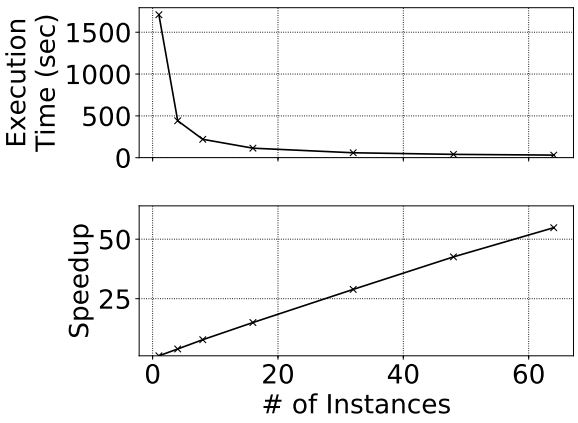
エクスチェンジにおけるコネクションのバッファサイズのデフォルト値は 1MiB とした．

4.3 演算資源に対するスケーラビリティ

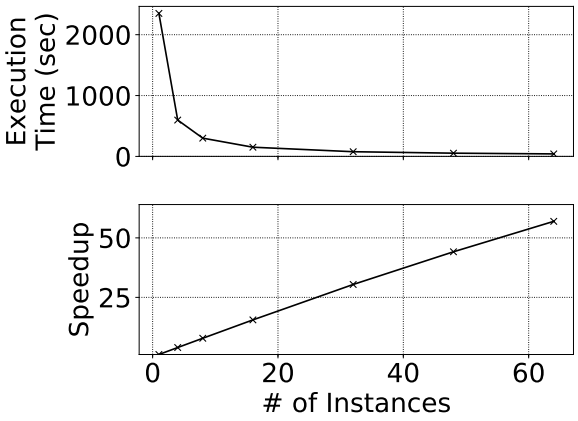
まず，演算資源に応じてクエリの実行速度の向上が得られることを示すために，クエリ実行に割当てたインスタンス数に対する実行時間を計測した．Q1，Q2，Q3 のそれぞれについて，実行時間ならびにインスタンス数 1 の時の実行時間を 1 とした性能向上率を図 6 に示す．インスタンス数 64 の時にインスタンス数 1 の時と比べて，Q1 では 54.8 倍，Q2 では 56.9 倍，Q3 では 58.3 倍の性能向上が得られた．本結果から，本論文で示したデータベースエンジンでは，利用するインスタンス数に応じた性能向上が得られることを明らかにした．

4.4 演算資源削除手法の実行性能への影響

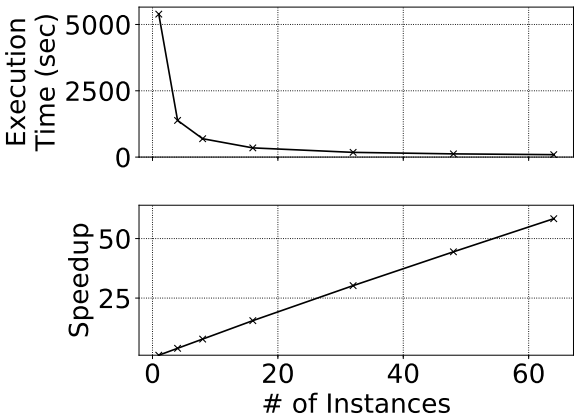
次に，演算資源削除手法が実行性能へ及ぼす影響を明らかにするために，実行開始時の割当てインスタンス数を変えて Q1



(a) クエリ 1



(b) クエリ 2



(c) クエリ 3

図 6: 各クエリの実行時間と性能向上率
Fig. 6 Plan trees of queries for evaluation experiments

の実行を開始し，当該クエリの実行中に 1 インスタンスを削除した時のインスタンス削除に要した時間を計測した．実行開始時の割当てインスタンス数は 8，16，32，48，64 とした．結果を図 7 に示す．実行開始時の割当てインスタンス数が増えるにつれて，インスタンス削除に要する時間が増加した．これは，

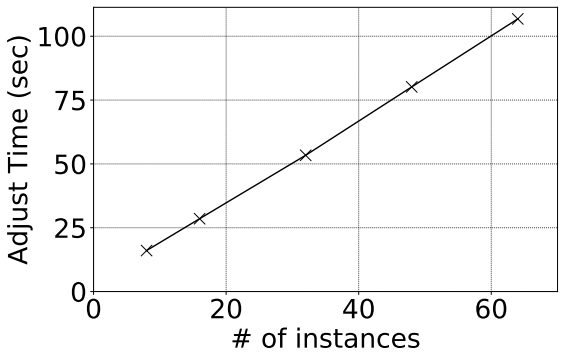


図 7: クエリ Q1 に対する演算資源削除の実行時間
Fig. 7 Resource deletion time for q1

割当てインスタンスの増加に併せて，削除対象となったインスタンスにおけるコネクションが増加し，インスタンス削除の実行時に処理を待つ必要があるタプル数が増加したためと考えられる．

次に，インスタンス削除の実行時における処理待ちタプルの影響を明らかにするために，Q1 を 64 インスタンスを割当てて開始し，15 秒後に 56 インスタンスを削除し，割当てインスタンス数が 8 になるようにした．実行中に各ステージに対応するワーカが実行されているインスタンス数，各ステージの処理での IOPS，各ステージの処理待ちタプル数を 0.1 秒ごとに計測した．Q1 は全部で 3 つのステージで分割して実行されるため，インスタンス数，IOPS については 3 ステージ全てを，処理待ちタプル数については処理の起点となるステージ 0 では処理待ちのタプルが存在しないため，ステージ 0 を除く 2 ステージを対象として計測を行った．得られた結果を 8 に示す．ステージ 0，ステージ 2 についてはインスタンス削除操作の開始直後に，全ての削除対象インスタンスにおいてワーカが速やかに停止したが，ステージ 1 については凡そ 100 秒後にワーカ停止が完了した．各ステージの IOPS についても，ワーカが速やかに停止したステージ 2 では，削除したインスタンス数に応じて IOPS が減少した．処理待ちタプル数の結果を見ると，ステージ 1 の処理待ちタプル数はクエリ開始 5 秒程度で 1.9M まで増加した後に減少に転じているが，インスタンス削除開始後には減少速度が低下し，処理待ちタプルがなくなったのは 115 秒時点であった．一方，ステージ 2 の処理待ちタプル，インスタンス削除開始前には殆どなかったが，開始後に大きく増加した．これは，インスタンス削除操作開始後にステージ 2 のワーカが速やかに停止し，ステージ 2 全体の処理速度が低下したため，全てのワーカが起動中のステージ 1 から送信されるタプルの流量に対して，ステージ 2 における処理が追いつかなくなり，処理待ちのタプル数が増加したものと考えられる．本実験により，演算資源削除の実行開始前に処理待ちタプルが多く発生しているステージにおけるワーカの停止に長時間を要し，当該ステージにおけるワーカの停止時間によって演算資源削除全体の実行時間が律速されることを明らかにした．

次に，ソケットバッファサイズの調整によって，演算資源削除実行時の処理待ちタプル数を削減し演算資源削除の実行時間を短縮可能であることを示すために，エクスチェンジにおけるソケットバッファのサイズを変えて Q1 の実行を開始し，1 インスタンスを削除した時のインスタンス削除に要した時間を計測した．計測に用いたソケットバッファのサイズは 256B，1KiB，4KiB，16KiB，64KiB，256KiB，1MiB とした．結果を図 9 に示す．バッファサイズ 1MiB ではインスタンス削除の実行に 104.8 秒を要したが，バッファサイズを小さくした場合には，バッファサイズ 16KiB からインスタンス削除の実行時間が短くなり，バッファサイズ 1KiB ではインスタンス削除の実行時間は 4.2 秒となった．本実験により，バッファサイズの調整によってインスタンス削除の実行時間を短縮可能であることを示した．

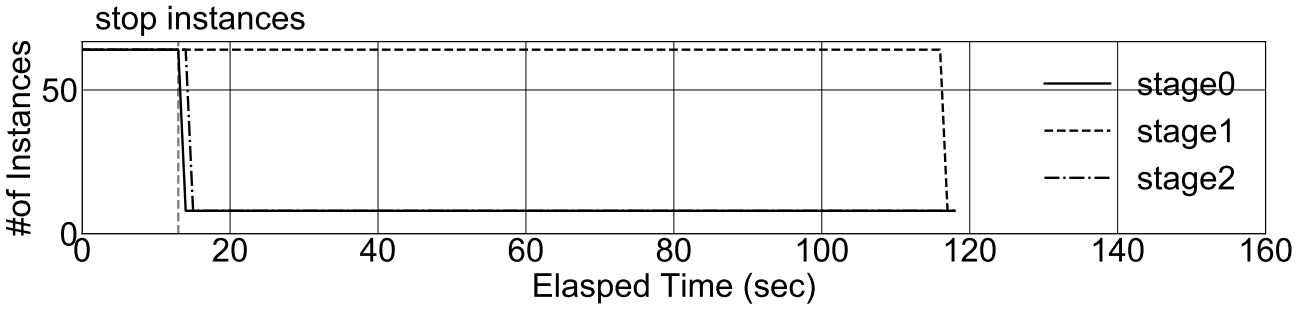
4.5 実行中クエリに対する動的演算資源調整

ソケットバッファサイズの調整によって動的演算資源調整による実行中クエリの割当て演算資源の調整を俊敏に完了することが可能となることを示すために，ソケットバッファサイズを 16KiB として Q1 の実行中に割当てインスタンスの追加・削除を行い，割当てられているインスタンス数，およびクエリ全体でのストレージへの毎秒リクエスト回数（IOPS）を 1 秒ごとに計測した．割当てるインスタンス数はクエリの実行を開始時には 8 とし，インスタンス数が 32，64 になるように，インスタンス追加を 2 回行った．その後，インスタンス数が 32，8 になるようにインスタンス削除を 2 回行った．得られた結果を図 10 に示す．動的演算資源調整を用いてクエリの割当てインスタンスを変更することができた．インスタンス追加操作の完了に要した時間は 2 秒程度，インスタンス削除の完了に要した時間は，64 インスタンスから 32 インスタンスへと削除した時には 7 秒，32 インスタンスから 8 インスタンスへと削除した時には 5 秒程度であった．また，割当てインスタンス数の調整に応じてクエリの IOPS を増加・減少することができ，割当てインスタンス数 64 の時に，最大で 2.1M IOPS が得られた．

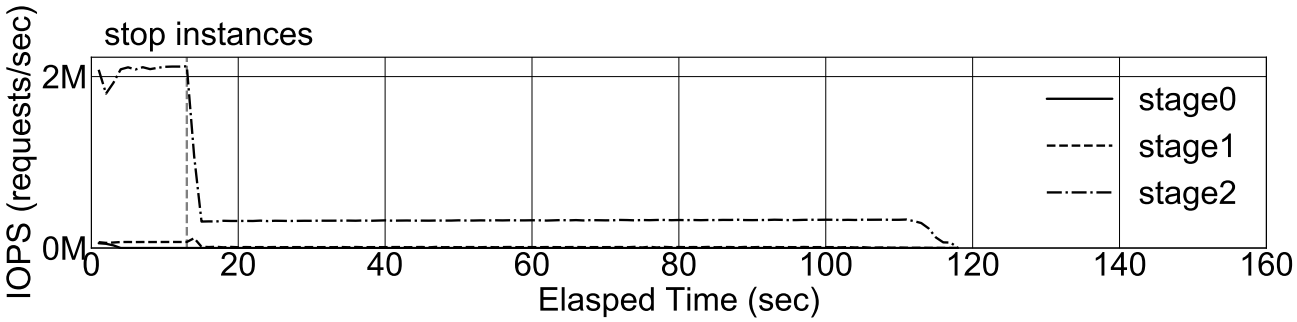
5 関 連 研 究

パブリッククラウド環境の資源伸縮性を利用するデータベースエンジンは，AWS の Aurora[5] や Google の BigQuery[6] などがパブリッククラウド事業者によって提供されている．学術界においても，ワークロードやユーザのサービス要求に応じてクエリへの演算資源の割当てを調整する手法 [7–9] などが提案されている．Snowflake[10] では，パブリッククラウド環境の資源伸縮性の利用として，データベースエンジンの再起動をせずに，データベースエンジンから利用可能な資源量の調整を提案している．著者らの提案する動的演算資源調整はデータベースエンジンにおいて実行中クエリに割当てられた資源の調整を可能とするものであり，これに類する研究は他に見当たらない．[11][12]

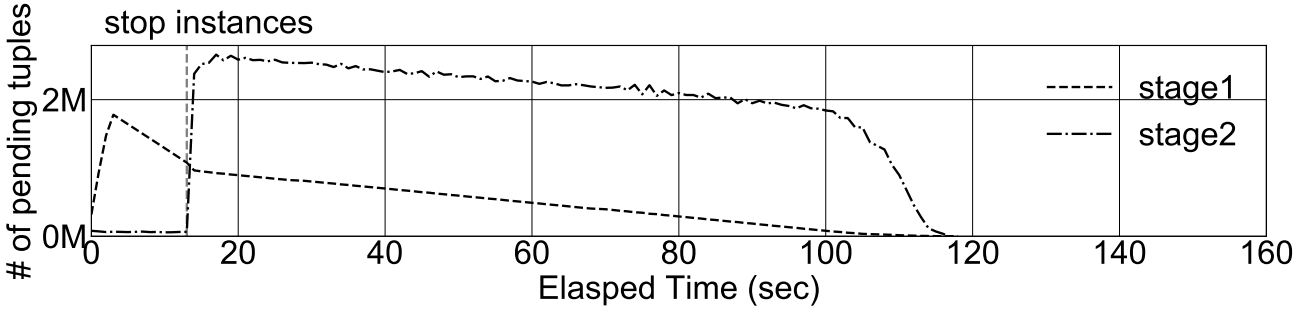
本論文ではデータベースエンジンのアーキテクチャとして，全てのプロセッサがネットワークを介して一つのストレージを



(a) インスタンス数



(b) IOPS



(c) 処理待ちタプル数

図 8: 演算資源削除実行時のステージごとのインスタンス数，IOPS，処理待ちタプル数

Fig. 8　Plan trees of queries for evaluation experiments

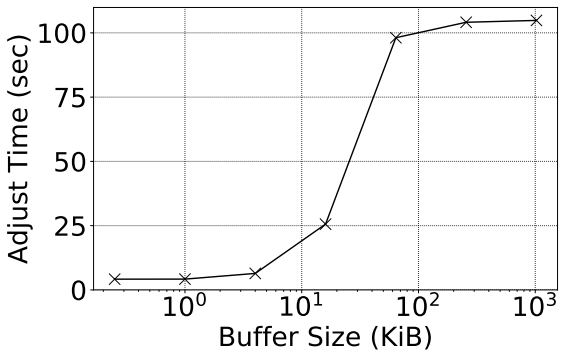


図 9: ソケットバッファサイズを変えた場合の演算資源削除の実行時間

Fig. 9　Resource deletion time with various socket buffer size

共有する共有ストレージ型を用いているが，無共有型のアーキテクチャも広く用いられている．無共有型ではプロセッサがストレージを共有せずに，プロセッサごとにデータを分割して配置する [11]．無共有型ではプロセッサ間でのストレージ入出力の競合をなくすことによって高いスケーラビリティが得られるが，データの配置の影響をうけてクエリの実行効率が低下する可能性があるため，クエリの実行履歴を利用して各プロセッサ内で処理が完結するようにデータの配置を調整する手法が提案されている [13–15]. 無共有型における動的演算資源調整については今後の課題としたい．

本論文で示した共有ストレージ型データベースエンジンの試作実装では AWS が提供するキーバリューストアの DynamoDB をストレージとして用いた．パブリッククラウドが提供するキーバリューストアやオブジェクトストアなどのストレージを共有ストレージとして用いる共有ストレージ型データベースエンジンの研究が近年行われており [5, 16–19]，共有ストレージ

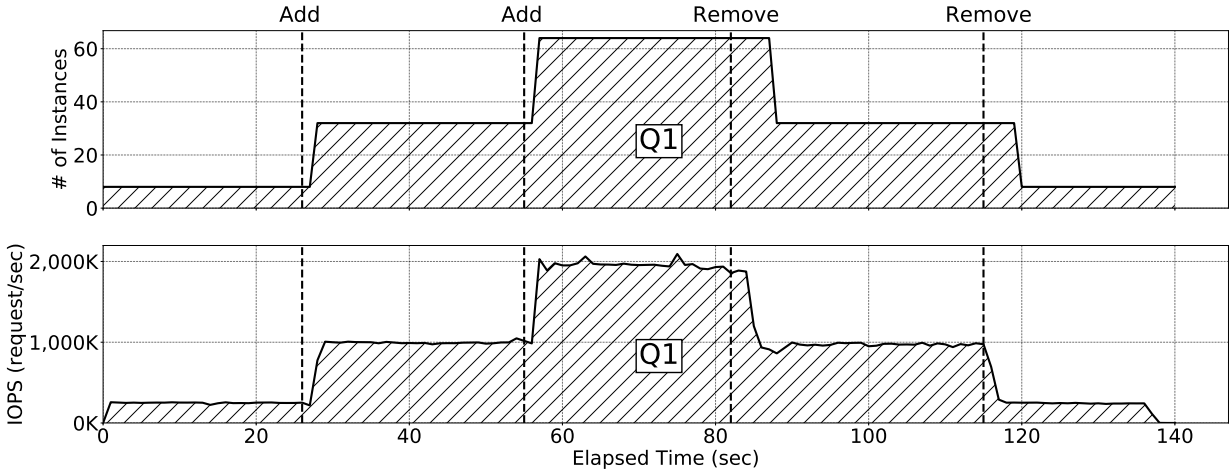


図 10: クエリ Q1 に対する動的演算資源調整

Fig. 10 Dynamic resource adjustment for Q1

型データベースエンジンのストレージとしてキーバリューストアを用い，当該データベースエンジンにおいて分析クエリを高速に実行する手法 [20] などが提案されている．本論文は，共有ストレージ型データベースエンジンにおいて，クエリの実行時に演算資源を調整することを目的としており，これらの研究とは目的を異とする．

演算資源を処理の実行中に調整する手法として，SAN によってディスクを共有したクラスタを利用する研究や [21–23] や，MapReduce[24] を対象とした研究が行われている．提案手法は関係データベースエンジンにおけるクエリを対象として動的に演算資源の調整を行うものであり，これらの研究は対象を異とする．

6 おわりに

本論文では，著者らが取り組んでいる共有ストレージ型データベースエンジンにおける動的演算資源調整手法において，当該調整手法がどのように実行時間へ影響を与えるかを明らかにするために，試作実装を用いたパブリッククラウドにおける評価実験の結果を示し，データベースエンジン内部において処理待ちとなっているタプルが資源調整の実行に与える影響について論じた．インスタンス数に対する静的なスケーラビリティの評価実験では，64 インスタンスを用い，3 つのクエリについて，それぞれ 54.8 倍，56.9 倍，58.3 倍の性能向上が得られた．クエリ実行に割り当てるインスタンス数に応じてインスタンス削除の完了に要する実行時間が増加することを示し，処理待ちタプルが多く存在しているステージにおけるワーカの停止に要する時間によって演算資源削除全体の実行時間が律速されることを示した．また，エクスチェンジにおけるバッファサイズの調整によって演算資源削除の実行に要する時間を短縮が可能であることを示した．今後はクエリ実行時におけるデータベースエンジンの状態に応じて動的に各パラメータを調整し演算資源調整の実行に要する時間を削減する研究に取り組んでゆきたい．

謝辞 本研究の一部は，内閣府革新的研究開発推進プログラム（ImPACT）「社会リスクを低減する超ビッグデータプラットフォーム」の支援の下に実施したものである．また，本研究の一部を進めるに当たっては，Amazon Web Services, Inc より AWS Cloud Credits for Research Program による実験環境資源の提供を受けた．感謝する次第である．

文　献

[1] Spending on IT Infrastructure for Public Cloud Deployments Will Return to Double-Digit Growth in 2017, According to IDC.

[2] 奥野晃裕, 早水悠登, 合田和生, 喜連川優. クラウド環境に於けるクエリ実行時の資源調整機構を備えた高速データベースエンジンの試作に関する一考察. 信学技報, DE2017-25, Vol. 117, No. 374, pp. 7–12, Dec 2017.

[3] 奥野晃裕, 早水悠登, 合田和生, 喜連川優. 共有ストレージ型データベースエンジンにおける動的演算資源調整手法の提案. 情報処理学会論文誌データベース（TOD）, Vol. 11, No. 2, pp. 30–43, jul 2018.

[4] The TPC-H Benchmark.

[5] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In *SIGMOD Conference*, pp. 1041–1052, 2017.

[6] Sérgio Fernandes and Jorge Bernardino. What is BigQuery? *IDEAS*, pp. 202–203, 2015.

[7] Ryan Marcus and Olga Papaemmanouil. WiSeDB - A Learning-based Workload Management Advisor for Cloud Databases. *PVLDB*, 2016.

[8] Dimokritos Stamatakis and Olga Papaemmanouil. SLA-driven workload management for cloud databases. *ICDE Workshops*, pp. 178–181, 2014.

[9] Liangzhe Li and Le Gruenwald. An SLA and Operation Cost Aware Performance Re-tuning Algorithm for Cloud Databases. *CLOUD*, pp. 966–969, 2016.

[10] Benoît Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. The snowflake elastic

data warehouse. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pp. 215–226, 2016.

[11] Michael Stonebraker. The case for shared nothing. *IEEE Database Eng. Bull.*, Vol. 9, No. 1, pp. 4–9, 1986.

[12] David J. DeWitt and Jim Gray. Parallel database systems: The future of high performance database systems. *Commun. ACM*, Vol. 35, No. 6, pp. 85–98, 1992.

[13] Erfan Zamanian, Carsten Binnig, and Abdallah Salama. Locality-aware partitioning in parallel database systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pp. 17–30, 2015.

[14] Carlo Curino, Yang Zhang, Evan P. C. Jones, and Samuel Madden. Schism: a workload-driven approach to database replication and partitioning. *PVLDB*, Vol. 3, No. 1, pp. 48–57, 2010.

[15] Abdul Quamar, K. Ashwin Kumar, and Amol Deshpande. SWORD: scalable workload-aware data placement for transactional workloads. In *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pp. 430–441, 2013.

[16] Jeff Shute, Mircea Oancea, Stephan Ellner, Ben Handy, Eric Rollins, Bart Samwel, Radek Vingralek, Chad Whipkey, Xin Chen, Beat Jegerlehner, Kyle Littlefield, and Phoenix Tong. F1: the fault-tolerant distributed RDBMS supporting google's ad business. In K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pp. 777–778. ACM, 2012.

[17] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Elastras: An elastic, scalable, and self-managing transactional database for the cloud. *ACM Trans. Database Syst.*, Vol. 38, No. 1, pp. 5:1–5:45, 2013.

[18] Matthias Brantner, Daniela Florescu, David A. Graf, Donald Kossmann, and Tim Kraska. Building a database on S3. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pp. 251–264, 2008.

[19] Justin J. Levandoski, David B. Lomet, Mohamed F. Mokbel, and Kevin Zhao. Deuteronomy: Transaction support for cloud data. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, pp. 123–133, 2011.

[20] Markus Pilman, Kevin Bocksrocker, Lucas Braun, Renato Marroquin, and Donald Kossmann. Fast scans on key-value stores. *PVLDB*, Vol. 10, No. 11, pp. 1526–1537, 2017.

[21] 合田和生, 田村孝之, 小口正人, 喜連川優. San 結合 pc クラスタにおけるストレージ仮想化機構を用いた動的負荷分散並びに動的資源調整の提案とその評価. 電子情報通信学会論文誌. D-I, 情報・システム, I-情報処理 = The transactions of the Institute of Electronics, Information and Communication Engineers. D-I, Vol. 87, No. 6, pp. 661–674, jun 2004.

[22] Kazuo Goda, Takayuki Tamura, Masato Oguchi, and Masaru Kitsuregawa. Run-time load balancing system on san-connected PC cluster for dynamic injection of CPU and disk resource - A case study of data mining application. In *Database and Expert Systems Applications, 13th International Conference, DEXA 2002, Aix-en-Provence, France, September 2-6, 2002, Proceedings*, pp. 182–192, 2002.

[23] Masato Oguchi and Masaru Kitsuregawa. Runtime data declustering over san-connected PC cluster system. In *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, p. 275, 2002.

[24] Sven Groot, Kazuo Goda, and Masaru Kitsuregawa. Towards improved load balancing for data intensive distributed computing. In *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011*, pp. 139–146, 2011.