社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

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

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@tkl.iis.u-tokyo.ac.jp

あらまし 仮想化された計算・記憶資源をネットワークを介して利用するクラウド環境は，自らハードウェアを所有するオンプレミス環境とともに広く用いられるようになってきている．クラウド環境の多くはユーザの要求の変化に応じて柔軟に利用する資源の量を調整する機能を実現しており，この機能は多数のユーザに支持されている．しかしながら，オンプレミス環境における一定量の計算・記憶資源を前提に設計された従来のデータベースエンジンでは，クラウド環境の有する資源の伸縮性を活用することは難しい．本論文では，データベースエンジンにおいてクエリ処理の実行中に利用する計算資源を調整する手法を提案し，さらに当該手法を用いたデータベースエンジンの試作実装に基づく小規模環境での実験結果を示すことで提案手法の有用性を検証する．
キーワード クラウド，データベースエンジン，動的資源調整

A Study on Prototyping of Fast Database Engine with Runtime Resource Adjustment Mechanism in Cloud Environments

Akihiro OKUNO†, Yuto HAYAMIZU†, Kazuo GODA†, and Masaru KITSUREGAWA†,††

† Institute of Industrial Science, The University of Tokyo
†† National Institute of Informatics
E-mail: †okuno@tkl.iis.u-tokyo.ac.jp

Abstract Cloud environments are becoming widely used together with on-premise environments. Cloud environments allow users to adjust the computation and storage resources on their demand. However, since the conventional database engine has been designed for the on-premise environments, which provide a fixed amount of resources, it has been difficult to exploit the elasticity of the resources in the cloud environments. In this paper, we propose a method to adjust the computing resource usage during query runtime, and verify the effectiveness of our method by showing the experimental results.
Key words Cloud, Datbase Engine, Dynamic Resource Adjustment

1. はじめに

情報システムを構築する際に，自らハードウェアを所有・管理するオンプレミス方式に加えて，Amazon，Google などの事業者が提供する計算資源やストレージを借り受けてインターネットを介して利用するパブリッククラウド方式が広く用いられるようになってきている．パブリッククラウドの市場は年平均 21%で成長すると予想されており [1]，情報システムの構築におけるパブリッククラウド環境の重要性は今後更に増していくものと考えられる．

パブリッククラウド環境では極めて多数の計算・記憶資源が仮想化され提供されており，ユーザは必要な量の資源を即座に調達することが可能である．またパブリッククラウド環境の多くは，利用者の要求の変化に応じて利用する資源の量を増やす，あるいは減らす機能を提供している．パブリッククラウド環境の有するこれらの特徴を活用することにより，大規模な情報システムを迅速かつ柔軟に構築することが事が可能となる．

他方，関係データベース管理システムは情報システムにおいて堅牢なデータの保護と迅速なデータ処理を可能とし，今日の社会・経済において重要な役割を果たしているソフトウェアの一つである．しかしながら，従前の関係データベース管理システムはオンプレミス環境の一定量の資源を前提としており，利

用する資源の量を柔軟に調整可能というパブリッククラウド環境の特徴を利用することは困難である．

関係データベース管理システムは情報システムにおいてデータの格納・処理を担う中心的なソフトウェアとして，ユーザからの多種多様な要求に応える必要がある．関係データベース管理システムにおいて，パブリッククラウド環境が提供する資源の伸縮性を活用してクエリ実行での利用資源を柔軟に調整可能とすることは，ユーザおよびシステム管理者の両方に多数の利点をもたらすと期待される．例えば，関係データベース管理システムのユーザにおいて，実行の途中であるクエリに対しても計算資源を追加して投入することにより当該クエリの処理を加速することが可能となり，想定より長い実行時間を要したクエリに対しての実行性能の調整の機会を提供する．また，複数のユーザが資源を共有するマルチテナント環境の関係データベース管理システムにおいて，利用可能な資源が少ない状況で高い優先度のユーザがクエリを開始した場合に，低優先度のユーザが実行しているクエリから高優先度のユーザのクエリへと資源を振り替えることにより，よりユーザ毎の優先度に即した資源の提供が可能となるだろう．

本研究では，クラウド環境が有する資源の伸縮性を最大限に活用可能なデータベースエンジンの実現を目的とする．本論文ではクラウド環境が提供する資源のうち計算資源を対象とし，クエリ実行中に利用する計算資源量を調整する手法を提案する．さらに，提案手法を実装したデータベースエンジンの試作と当該実装を用いた小規模環境における実験結果を示すことで当該手法の有効性を示す．

本論文の構成は次の通りである．2. 章ではクラウド環境を指向したデータベースエンジンの設計を示す．3. 章では 2. 章で示したデータベースエンジンにおいてクエリ実行時に利用資源量を調整する手法を提案する．4. 章では当該データベースエンジン設計および動的資源調整手法を用いた初期実装を示し，小規模環境における性能評価実験を行って当該手法の有効性を議論する．5. 章では関連する研究および本論文との関係について記し，6. 章では本論文のまとめ，および今後の展望について述べる．

2. クラウド環境におけるデータベースエンジンの設計

データベースエンジンは長い研究の過程において多数の設計方法が提案されているが，本章の始めにクラウド環境が有する特徴を活用するためのデータベースエンジンの設計について議論したい．クラウド環境における利点の一つとして，利用する計算・記憶資源の量を柔軟に調整可能であることが挙げられる．本論文では研究の端緒として，クラウド環境が提供する資源のうち，伸縮性を利用する資源として計算資源を取り上げる(注1)．

(注1)：クラウド環境では計算資源の他，ストレージ資源においても性能の伸縮性を提供している．ストレージ資源における性能の伸縮性を利用するためには，当然ながら本論文で提案する計算資源の伸縮性を対象とした設計とは異なる手法が必要になると考えられる．ストレージ性能の伸縮性を利用する設計については今後の課題としたい．

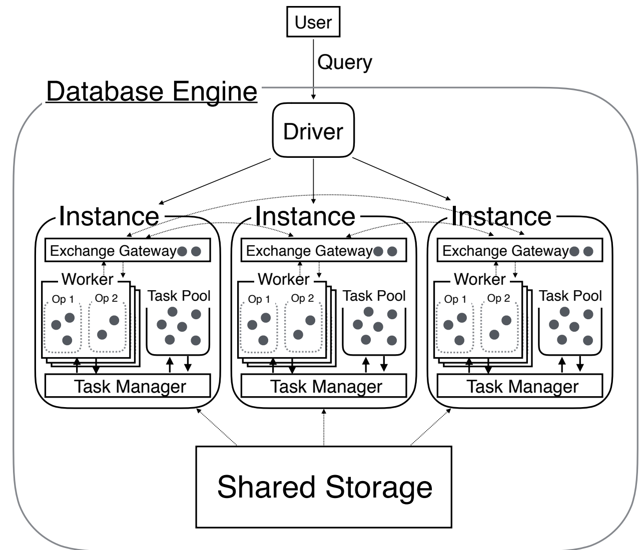

図 1 本論文で提案するクラウド指向の高速データベースエンジンのシステム構成

クエリ実行中に計算資源を柔軟に調整するために，本章で提案するデータベースエンジンでは，計算資源とストレージを分離し，各計算機からネットワークを介してストレージを共有する共有ストレージ型のアーキテクチャを採用する．

従前の固定された計算資源を前提としたデータベースエンジンでは，クエリの実行開始前に各計算機への負荷割り当てを決定する．この方法では，クエリの実行時に計算資源の調整を行う際に，各計算機への負荷割り当ての再計算と，計算機間での処理の移送が必要となり，容易に利用資源の調整を行うことができない．そのため，クラウド環境の計算資源の伸縮性を利用するデータベースエンジンでは，実行時に動的にタスクを生成して処理を実行する手法でクエリを実行することが望ましい．

図 1 は，本論文で提案するデータベースエンジンのシステム構成を示す．データベースエンジンはクラウド上の複数の計算機（インスタンス）を利用し，当該インスタンス群はネットワークを介して唯一のストレージを共有する．インスタンス群はドライバなるプロセスから処理内容を受け取り，ユーザからのデータベースエンジンへの操作はドライバを介して行われる．

クエリ実行は，ドライバがユーザからクエリを受け取ることによって開始される．ドライバはユーザからクエリを受け取ると，まずは当該クエリを実行計画へと変換する．当該実行計画は複数のステージから構成されており，複数のインスタンスにおいてステージ単位で並列に実行することが可能である．実行計画はステージ単位に分割され各インスタンスへと配布される．インスタンスは実行計画を受け取ると，ワーカなるプロセスを立ち上げ，実行計画に含まれる演算に基づいて処理を開始する．入出力処理が必要となる演算においては，ワーカからネットワークを介した共有ストレージに対して入出力処理が発行される．また，並列クエリ処理にでは，ステージをまたぐ処理において，ワーカ間で中間結果データを交換することが必須となるが，各インスタンス毎に唯一のエクスチェンジゲートウェイなる機構を設け，ワーカ間のデータの交換は当該機構を介して実

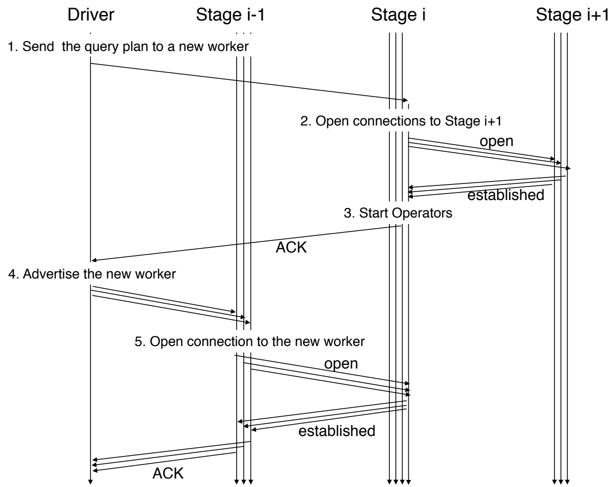

図 2　Stage i に対するワーカ追加のシーケンス

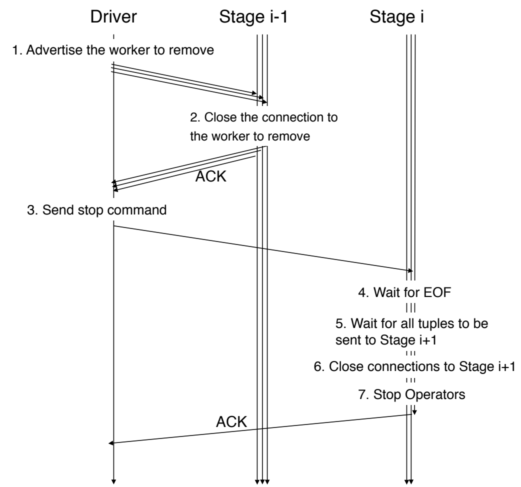

図 3　Stage i に対するワーカ削除のシーケンス

行される．ワーカでは実行計画に含まれる演算に基いてクエリ処理を複数のタスクへと分解し並列実行する．提案設計では，インスタンス毎にタスクマネージャなる機構を設けることで，生成されるタスクへの計算資源の割り当てを管理している．各ワーカにおいてタスクを生成する際にはタスクマネージャへと要求を行い，タスクマネージャは要求を受け取ると，予めインスタンスの有する資源に応じて確保しておいたタスクプールからワーカへと割り当てを行う．

3. クエリ実行時の利用資源の調整手法

本章では 2. 章で提案したデータベースエンジンの設計において，クエリ実行時に利用する計算資源を調整する手法，即ちクエリに用いるインスタンス利用数を実行時に調整する手法を説明する．本論文で提案するデータベースエンジンの設計では，あるクエリにおけるインスタンスの利用数は，当該クエリの実行計画における各ステージにおいて，ワーカを起動するインスタンスの合計数によって決定される．ワーカはステージ毎に分割された実行計画をインスタンスが受け取ることによって起動される．即ち，クエリ実行時のインスタンス数の調整は，当該クエリの実行計画における，あるステージ i に対するワーカの追加・削除という操作に帰着される．ここで，あるステージ i が実行計画内のどのステージであるかによって，ワーカを追加・削除するために必要な操作が異なることが予想されるが，本論文では議論の簡単のため，集約を含まないクエリであり，かつ開始・終端のステージでない，即ち，クエリが総数 n のステージからなる時，$i(1 < i < n)$ のステージにおけるワーカの追加・削除を対象としたい．本論文では，研究のファーストステップとして計算資源を対象としており，ストレージの配置としては共有ストレージ型を想定しているため，ワーカの追加・削除にあたってストレージにおけるデータの再配置を行う必要はない．(注2)

(注2)：ストレージの配置として無共有型，，共有メモリ型のアーキテクチャを採用した場合には，ワーカの追加にあたってデータの再配置が必要となり，本論文で提案した手法をそのまま適用することはできない．これらのアーキテクチャにおける資源調整の手法については別に議論したい．

3.1 ステージ i に対するワーカ追加手順

図 2 にステージ i に対するワーカ追加の手順を示す．ワーカ追加・削除にあたっては各ワーカが協調して手順を進めることが必要となるが，協調動作の為の全ワーカの操作はドライバを介して行う．図において，各ステージにおける縦線は当該ステージにおける各ワーカを表している．また，ステージ間あるいはステージとドライバ間を結ぶ線は，ネットワークを介した通信が行われることを表している．

ステージ i におけるワーカ追加では，まず追加の対象するインスタンスを決定し，当該インスタンスに対してステージ i の実行計画の送信を行う．実行計画を受け取ったインスタンスは，データの送信先となるステージ $i+1$ の全ワーカに対して新たにコネクションを開く．コネクションが確立された後に，インスタンス内でワーカを立ち上げて実行計画に含まれる各オペレータの処理を開始し，ドライバに受領確認信号（ACK）を返す．この時点では，新たに追加されたワーカでは処理を行う準備は整っているが，ステージ $i-1$ からデータが送られて来ないために，実際に処理が行われることはない．ドライバは追加ワーカから ACK を受け取ったら，追加ワーカの情報をステージ $i-1$ の全ワーカに送信する．ステージ $i-1$ の各ワーカは，追加ワーカの情報を受け取ったら当該ワーカへのコネクションを開いて中間結果の送信先に追加し，ドライバに ACK を返す．ドライバはステージ $i-1$ の全ワーカから ACK を受け取ったら，ステージ i のワーカ追加処理を終了とする．著者らが提案する手法では，このようにステージ i での準備が整って，受け取ったデータの処理を行いステージ $i+1$ に送信することが可能になってから，ステージ $i-1$ からのデータ送信を開始することで，クエリ処理の実行中にワーカの追加を行っても，全てのデータが正しく処理される．

3.2 ステージ i に対するワーカ削除手順

次に，図 3 にステージ i におけるワーカ削除の手順を示す．ステージ i におけるワーカ削除では，まず削除の対象とするワーカを決定し，ステージ $i-1$ の全ワーカに対して削除する

ワーカの情報を送信する．ステージ $i-1$ の各ワーカは削除するワーカの情報を受け取ったら，そのワーカへのコネクションを中間結果の送信先から削除して当該コネクションを閉じ，ドライバに ACK を返す．ドライバはステージ $i-1$ の全てのワーカから ACK を受け取るまで待機する．ステージ $i-1$ の全てのワーカから ACK を受け取った時点で，削除するワーカには新しいデータが送信されなくなる．その後，ステージ i の削除するワーカに対して停止命令を送る．ワーカは停止命令を受け取ったら，まずステージ $i-1$ とのコネクションから全てのデータを受け取るまで，即ち読み取り終端信号（EOF）を受け取るまで待機する．EOF を受け取ったら，ワーカ内のオペレータにおいて処理途中のデータが全て処理を終えてステージ $i+1$ に送信されるまで待つ．全てのデータを送信し終わった後，ワーカ内のオペレータを全て停止してからドライバに ACK を返す．ドライバは削除するワーカから ACK を受け取ったら，ステージ i におけるワーカ削除処理を終了とする．ステージ i の停止したワーカに対してデータを送信してしまうと，当該データはステージ i で適切に処理されず失われてしまう．著者らの提案する手法では，ワーカの削除処理おいて，まずステージ $i-1$ からのデータ送信を止めることで，この問題を回避している．

以上，図 2，図 3 の 2 つの操作によって，ステージ i に対するワーカの追加・削除を行うことができ，即ちクエリの実行中にワーカの割当数を任意に変更することが可能となる．図 2，図 3 においては説明の簡単の為，追加・削除を行うワーカ数を 1 としたが，複数のワーカの追加・削除についても，ドライバからの送信，および ACK の待機を複数行うという拡張のみで実行可能である．

3.3 開始・終端ステージにおけるワーカ削除・追加

本章で提案した手法は，クエリが総数 n のステージからなる時のステージ $i(1<i<n)$ に対する追加・削除の操作が対象であるが，終端ステージ $i(i=n)$ に対するワーカ追加・削除の操作についても，ステージ $i(1<i<n)$ の操作から僅かの変更で拡張可能である．即ち，ワーカの追加については，追加するインスタンスが実行計画を受け取った後にすぐにワーカを開始して ACK を返せばよい．ワーカの削除については，ステージ $i(1<i<n)$ に対するワーカ削除の手法でステージ $i+1$ は関わらないため，ステージ $i(i=n)$ に対するワーカの削除も，ステージ $i(1<i<n)$ の場合と全く同じ手法によってワーカの削除を実行することが可能である．

一方，開始ステージ $i(i=1)$ については，ワーカの追加・削除にあたって，テーブルをもれなく読み込むために他ワーカで読み込み済みの位置を管理する要求が必然的に発生する．その為，本論文で議論したステージ $i(1<i<n)$ に対するワーカ追加・削除の手法とは異なる手法が必要となると考えられる．また，集約演算を含むクエリについては，特にワーカの削除において計算済みの中間結果をどう扱うかという問題が発生する．本論文で対象外とした開始ステージに対する追加・削除の操作，および集約演算を含むクエリについては今後の課題としたい．

表 1　AWS（N. Virginia region）実験環境諸元

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

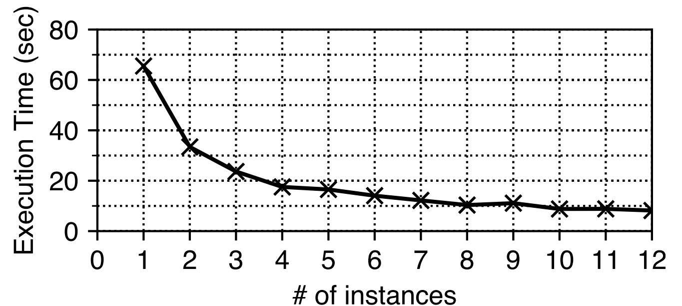

図 4　インスタンス数に対するクエリ実行時間

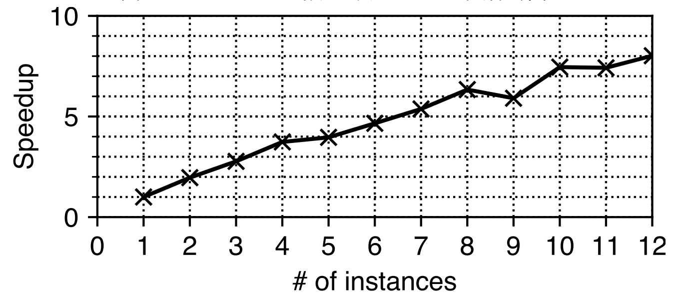

図 5　インスタンス数 1 を基点とした相対性能

4. データベースエンジンの試作と小規模環境における評価試験

4.1 データベースエンジンの試作

著者らは 2. 章で述べたクラウド環境を指向したデータベースエンジンの設計，および 3. 章で提案したワーカ割当数の調整手法を実装したデータベースエンジンの試作を行った．当該データベースエンジンではクエリの実行方式としてアウトオブオーダ型実行方式 [2] [3] を用いた．また，当該データベースエンジンにおいて 3. 章で提案した資源調整手法を実装するために，データベースエンジンのドライバに対して，インスタンス追加・削除コマンドなる機能を設けた．当該コマンドをドライバへ発行すると，新規インスタンスでのワーカの起動，もしくはワーカ起動中のインスタンスでのワーカ停止が実行される．

4.2 実 験 環 境

著者らは前項で述べたデータベースエンジンの試作実装を用い，小規模環境における評価実験を行った．実験環境として Amazon が提供するパブリッククラウドである Amazon Web Service（AWS）のバージニア北部リージョンを利用しており，計算資源として EC2 インスタンスを，ストレージとして DynamoDB をそれぞれ用いた．実験に用いた環境の諸元を表 1 に示す．

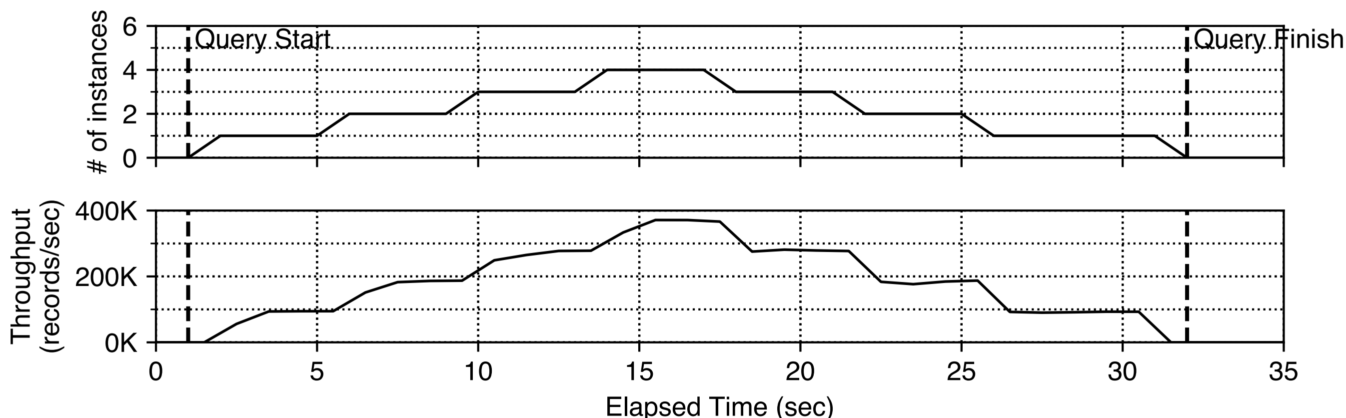

図 6　インスタンス追加・削除の操作を行った時のインスタンス数とスループット

データセットとしては，分析的なワークロードの標準ベンチマークとして広く用いられている TPC-H [4] を用いた．TPC-H の Scale Factor=1 で生成した 1GB のデータを，DynamoDB のテーブルとして読み込み，データセットとして利用した．この時，結合処理に索引を用いたアクセスを行えるように，DynamoDB の各テーブルの外部参照列にグローバルセカンダリインデックスを作成した．

対象クエリとしては，本論文での提案手法は集約処理を対象外としているため，結合処理のみを行う全 2 ステージからなるクエリを用いた．

4.3　データベースエンジンのスケーラビリティ

まず始めに，2. 章で示したデータベースエンジンの設計において，利用する EC2 のインスタンス数に対して性能向上が得られるかどうかを明らかにするための評価実験を行った．ステージ 1 のインスタンス数を 2 で固定し，ステージ 2 のインスタンス数を 1〜12 の間で変化させた時の，クエリの実行時間を計測した．計測したクエリ実行時間を図 4 に，インスタンス数 1 の性能を基点とした相対性能を図 5 に示す．インスタンス数の増加に応じて性能が向上し，インスタンス数 12 ではインスタンス数 1 の場合と比較して 8 倍の性能が得られた(注3)(注4)．この結果から，本論文で提案したデータベースエンジンの設計では利用した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　インスタンス追加・削除の操作とスループット

次に，3. 章で提案した利用資源の調整手法の有効性を示すために，事前に決めたあるシナリオに沿って，ステージ 2 を対象としたインスタンス追加・削除コマンドを実行し，その間のステージ 2 の入力操作におけるスループットを 1 秒毎に計測した．本実験で用いたシナリオでは，まずステージ 1 のインスタンス数を 2 で固定し，ステージ 2 のインスタンス数を 1 にしてクエリを開始した．その後 4 秒ごとにインスタンス数を 1 つずつ増やしていき，インスタンス数 4 まで増やした．その後，4 秒ごとにインスタンス数を 1 つずつ減らしていき 1 まで減らした．結果を図 6 に示す．この図では，x 軸は経過時間を表し，上図はその時点でのステージ 2 のインスタンス数を，下図はその時点でのステージ 2 の入力操作のスループットを表している．図 6 を見ると 4 秒ごとのインスタンスの追加・削除に応じてスループットの向上・減少しており，各インスタンス数において定常状態に達した後はインスタンス数に比例したスループットが得られていた．しかしながら，インスタンを追加してからのスループットの増加は緩やかであり，インスタンスの追加から定常状態のスループット性能が得られるまでに数秒程度の遅延を要した．クエリの実行はインスタンス数が再び 1 に戻った後，32 秒時点で全てのデータを処理し終えて終了した．クエリから得られた結果は，追加・削除の操作を行わない場合と一致していた．これらの結果から，本論文で提案した手法は，クエリの実行中に結果に影響を与えること無く利用する計算資源の調整を可能とすることを明らかにした．しかし，計算資源を追加した後にスループットが定常様態に達するまでには遅延が発生することが分かった．この遅延はより細かく資源を調整することが要求された時に障害となる可能性があり，本研究における今後の課題の一つである．

(注3)：インスタンス数 9，11 では，インスタンス数 8，10 の場合と比べて性能が低下していたが，この原因については十分に解明されておらず今後の課題としたい．

(注4)：インスタンス数 5 から性能の向上が鈍化しているが，これはクエリ開始から最大スループット性能が得られるまでの遅延の影響であると考えられる．この遅延の詳細については別の機会に議論する．

5.　関 連 研 究

並列データベースエンジンは 1980 年代に Gamma [5] や Grace [6] などが発表された後に盛んに研究が行われ，近年では計算機間のデータ通信を削減することにより結合処理を高速化する手法 [7] や，無共有型アーキテクチャにおける効率的なデータ配置の手法 [8]〜[10] などが提案されている．並列データベースエンジンは研究のみならず，Vertica や Netteza などに商用化され利用されているほか，従前の並列データベースエンジンに加えて Hadoop や Spark といった並列データ処理系も広く用いられるようになってきている．これらの並列データベースエンジンや並列データ処理系では，複数の計算機にデータを分散して配置し当該計算機クラスタにおいて並列にデータの処理を行うことで高速な処理を実現しており，多数の計算資源を効率よく利用することが可能であるが，計算資源の調整にあたっ

てはデータの再配置が必要となるため，柔軟に利用資源の調整を行うことはできない．

計算の実行中に利用資源を調整を行い負荷を分散する手法として，データマイニング[11], [12] や MapReduce [13] を対象としたものが提案されている．また，BigTable [14] や Dynamo [15] といった分散 KVS では負荷に応じて，利用する計算資源を柔軟に調整することが可能であるが，提供する機能は単純な読み書きのみであり，関係データベースエンジンが提供する複雑なクエリを処理することは不可能である．

Amazon が提供する Aurora [16] はクラウドの計算・記憶資源を用いることで高速な処理と高可用性を実現したマネージド型の DBaaS である．Aurora は計算資源とストレージとを分離するアーキテクチャを採用しており，クエリを受け付けたまま利用する計算資源の量を調整することがすることが可能であるが，個々のクエリの実行は固定された計算資源のもとで実行される．また，ユーザが計算資源を共有するマルチテナント型の DBaaS における資源割り当ての手法である Thrifty [17] が提案されている．Thrifty ではユーザごとの資源利用の状況を記録しておき，それに基づいてクラスタ内におけるデータの配置およびクエリへの資源の割り当てを決定することで，クラスタ全体における利用資源の最適化を図っている．Thrifty において調整されるのはクエリ毎の資源の割り当てであり，既に実行されたクエリの資源を調整することはできない．

6. おわりに

本論文では，パブリッククラウド環境が提供する柔軟な計算資源を活用可能なデータベースエンジンの開発を目的として，クラウド環境を前提としたデータベースエンジンの設計を示し，当該データベースエンジンにおいてクエリ実行時に利用資源の調整を可能とする手法を提案した．そして，当該手法を実装したデータベースエンジンの試作を用いて，小規模環境における評価実験を行い，当該手法の有効性を明らかにした．評価実験では，まず当該データベースエンジンにおいて利用した資源に応じた性能向上が得られることを確認し，TPC-H のデータセットを用いた実験では，インスタンス数 1 の場合の 8 倍の性能向上が得られた．さらに，本論文で提案した資源調整手法は，クエリの結果に影響を与えずに利用資源の調整を行うことが可能であることを明らかにした．

今後の課題としては，より一般のクエリ及び任意のステージに対して資源調整の手法を拡張していくとともに，複数クエリの同時実行時におけるクエリ間の資源調定や，ストレージ資源における性能の伸縮性の活用などにも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] “Worldwide Public Cloud Services Spending Forecast to Reach $266 Billion in 2021, According to IDC”. https://www.idc.com/getdoc.jsp?containerId=prUS42889917
[2] 喜連川優，合田和生，“アウトオブオーダ型データベースエンジン OoODE の構想と初期実験，” 日本データベース学会論文誌，vol.8，no.1，pp.131–136，2009.
[3] 合田和生，早水悠登，喜連川優，“100 ドライブ規模のディスクストレージ環境におけるアウトオブオーダ型データベースエンジン OoODE の問合せ処理性能試験，” 電子情報通信学会論文誌 D，vol.J97-D，no.4，pp.729–737，April 2014.
[4] “The TPC-H Benchmark”. http://www.tpc.org/tpch/
[5] DeWitt, David J, Gerber, Robert H, Graefe, Goetz, Heytens, Michael L, Kumar, Krishna B, and Muralikrishna, M, “GAMMA - A High Performance Dataflow Database Machine.,” VLDB J., pp.228–237, 1986.
[6] S. Fushimi, M. Kitsuregawa, and H. Tanaka, “An Overview of The System Software of A Parallel Relational Database Machine GRACE.,” VLDB J., pp.209–219, 1986.
[7] S. Chu, M. Balazinska, and D. Suciu, “From Theory to Practice - Efficient Join Query Evaluation in a Parallel Database System.,” SIGMOD Conference, pp.63–78, 2015.
[8] E. Zamanian, C. Binnig, and A. Salama, “Locality-aware Partitioning in Parallel Database Systems.,” SIGMOD Conference, pp.17–30, 2015.
[9] C. Curino, Y. Zhang, Evan P. C. Jones, and S. Madden, “Schism: a Workload-Driven Approach to Database Replication and Partitioning.,” PVLDB, vol.3, no.1, pp.48–57, 2010.
[10] A. Quamar, K.A. Kumar, and A. Deshpande, “SWORD: scalable workload-aware data placement for transactional workloads.,” EDBT, pp.430–441, 2013.
[11] 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ならびに動的資源調節の提案とその評価，” 電子情報通信学会論文誌，vol.J87-D-I，no.6，pp.661–674，June 2004.
[12] K. Goda, T. Tamura, M. Oguchi, and M. 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 —,” Database and Expert Systems Applications, pp.182–192, Sept. 2002.
[13] S. Groot, K. Goda, and M. Kitsuregawa, “Towards improved load balancing for data intensive distributed computing,” Proceedings of the 26th ACM Symposium on Applied Computing, Technical Track on Cloud Computing (SAC2011), pp.27–32, May 2011.
[14] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, “Bigtable: A Distributed Storage System for Structured Data.,” OSDI, pp.205–218, 2006.
[15] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: amazon’s highly available key-value store,” SOSP, pp.205–220, 2007.
[16] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao, “Amazon aurora: Design considerations for high throughput cloud-native relational databases,” SIGMOD Conference, pp.1041–1052, 2017.
[17] P. Wong, Z. He, Z. Feng, W. Xu, and E. Lo, “Thrifty - Offering Parallel Database as a Service using the Shared-Process Approach.,” SIGMOD Conference, pp.1063–1068, 2015.