並列データベースシステムにおける入出力追跡による耐障害型問合せ実行方式の提案とパブリッククラウドにおける実験

別所　祐太朗[1]　早水　悠登[2]　合田　和生[2]　喜連川　優[2,3]

概要：意思決定支援をはじめとする多様なビッグデータ応用のニーズに牽引され，データベースの容量には増大の傾向が見られ，問合せの実行は長時間に渡ることがある．並列データベースシステムが 1 つの有力なアプローチであるが，当該システムは多数のノードから構成されるため，問合せ処理の実行中にノードに障害が発生する確率が高まる．問合せ処理の実行中にノードに障害が発生した場合，通常は問合せ処理を再度実行し直す必要があり，ユーザが問合せの応答を得るまでに掛かる時間が増大するという問題がある．本論文は，共有ストレージ構成の並列データベースシステムを対象に，ストレージに対する入出力を起点として問合せ処理の進捗を追跡することにより，ノードに障害が発生した際に動的に他のノードがその実行状態を引き継ぎ，問合せ処理を継続することを可能とする手法を提案する．また，当該手法に基づく並列データベースシステムの試作機を示し，パブリッククラウドにおいて 16 インスタンスを用いて行った試験結果を示し，当該手法が正しく機能することを示す．

1. はじめに

意思決定支援をはじめとする多様なビッグデータ応用が登場し，そのニーズに牽引され，データベースが蓄積するデータ量には増大の傾向が見られる．既に 2009 年の時点において，ペタバイト規模の運用例が報告されており [1]，近年多様な分野で導入が試みられている IoT 機器類が発するセンサ情報の蓄積が本格化することにより，今後，当該傾向はより顕著になるものと期待される [2]．

大規模データベースの管理には，並列データベースシステムが広く利用される．当該システムは多数の計算機（ノード）を並列に駆動するものであり，問合せの実行中に予期せずいずれかのノードが故障する恐れがある．一般に，ノードの故障が生じた場合には，正常に動作しているノードの実行状態を破棄して，問合せ処理を初めから再実行する．大規模データベースの分析を行う場合，問合せ処理は数時間から数日を要することも珍しくなく，再実行に伴う時間や運用コストは無視できない．

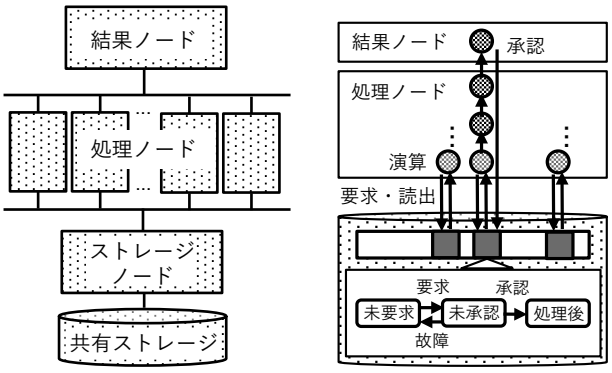
本論文は，共有ストレージ構成の並列データベースシステムを対象として，問合せ処理の実行中に，ストレージに対する入出力の毎に処理ノードにおける実行状態を追跡することにより，ノードに障害が発生した場合に，正常なノードを以って当該問合せ処理を継続することを可能とする手法を提案する．また，当該手法に基づく並列データベースシステムの試作実装と，パブリッククラウドにおいて 16 インスタンスを用いて行った試験結果を示し，当該手法によって正しく問合せ処理が継続されることとその性能特性を明らかにする．

本論文の構成は以下の通りである．第 2 章では本論文が対象とする共有ストレージ構成の並列データベースシステムと当該システムにおけるノード障害について述べる．第 3 章では入出力追跡による耐障害型問合せ実行方式を示す．第 4 章では当該方式の試作実装を示すとともに，当該実装を用いて行ったパブリッククラウドにおける実験を示し，最後に第 6 章において論文を纏める．

2. 並列データベースシステムと故障

共有ストレージ方式における並列走査：並列データベースシステムは，複数の計算機（ノード）を並列駆動することにより，大規模データベースに対する高速な問合せ処理を実現するものである．当該システムは，共有ストレージ構成と無共有構成に分類することができるが，本論文では前者を対象として，議論を進める．共有ストレージ構成は，複数のノード間でストレージを共有するものであり，スト

[1] 東京大学 大学院情報理工学系研究科 Graduate School of Information Science and Technology, The University of Tokyo, 〒 113-8656 7–3–1 Hongo, Bunkyo-ku, Tokyo
[2] 東京大学　生産技術研究所 Institute of Industrial Science, The University of Tokyo, 〒 153–8505 4–6–1 Komaba, Meguro-ku, Tokyo
[3] 国立情報学研究所 National Institute of Informatics, 〒 101–8430 2–1–2 Hitotsubashi, Chiyoda-ku, Tokyo



(a) システム構成　(b) 入出力追跡

図 1　並列データベースシステムの構成と入出力追跡

レージの管理をノードの管理と分離して行うことを可能とし，システムの管理容易性の観点から，商用のデータベース管理システムにおいて採用されている．

共有ストレージ構成の並列データベースシステムでは，ストレージをノード間で共有していることから，データの読込みは複数のノードが協調して並列に行う．例えば，巨大なログファイル等の関係表を複数ノードで走査する場合，ストレージが単一の読出しオフセットを管理し，問合せ処理を実行している最中に，ノードの要求に応じて，ストレージは関係表のどの部位を当該ノードが読み出すべきかを決定して読出しを行う [3]．処理ノードの処理能力に応じてオンデマンドに入出力を行うことにより，ノード間で優れた負荷分散が実現される．

ノード故障時の問合せ再実行の非効率性：並列データベースシステムは多数のノードから構成されていることから，問合せ処理の実行中に何れかのノードにおいて故障が発生する確率が高まる傾向がある．一般に，並列データベースシステムは，処理の効率性の観点から，演算をパイプライン化して実行し，逐一その結果を永続化しない．したがって，問合せ実行時にノードの障害が生じた場合には，問合せ処理を最初から再実行する必要がある．当然のことながら，再実行に伴い，ユーザが結果を得るまでの時間が大幅に増加するほか，運用に掛かるコストを増大する．

3. 並列走査の進捗追跡による耐ノード故障問合せ実行方式

本論文では，共有ストレージ構成の並列データベースシステムを対象に，問合せ処理の実行中に，ストレージに対する入出力の毎にノードにおける実行状態を追跡することにより，ノードに障害が発生した場合に，正常なノードを以って当該問合せ処理を継続することを可能とする手法を提案する．

システム構成および障害モデル：並列データベースシステムの構成として図 1 (a) に示すものを前提として説明する．即ち，ストレージは入出力を仲介するノード（ストレージノード）を挟んで問合せ処理を並列に実行するノード（処理ノード）群に接続され，処理ノード群は結果を集約するノード（結果ノード）に接続される．処理ノードは必要に応じて関係表をストレージノードを介して読み出し，出力を結果ノードに送信する．以降は，一つ以上の処理ノードが予期せず停止する障害を想定し（ネットワーク障害は対象外），その障害を検知する手法が利用可能である前提で議論を進める．

提案する問合せ実行方式：問合せ処理の実行中にいずれかの処理ノードに障害が発生した際に，正常に動作している処理ノード（および追加で投入する処理ノード）を以ってその処理を継続させて正確な問合せの結果を得るためには，障害発生時点において，問合せの入力である関係表のうち次の 2 つの条件の何れにも該当しない範囲を特定し，これをいずれかの処理ノードに割り当てる必要がある．

(1) 正常に動作している処理ノードが既に処理を引受けている．
(2) 既に処理結果が結果ノードの入力として確定している．

障害発生時に上述の範囲を正しく特定するために，本論文では，入出力に基づく問合せ処理の実行状態の追跡を提案する．即ち，ストレージノードが，問合せ処理の進捗と実行状態を都度把握し，処理ノード間の入力割当ての調停を行う．図 1 (b) にその概要を示す．簡単のために，ここでは処理ノードへの入力はタプルの単位で行われるものとし，当該タプルを入力として実行される一連のパイプライン化された演算の実行状態を，ストレージが「未要求」,「未承認」,「処理後」の 3 つの状態に分類して管理する．

まず，各タプルに対する一連のパイプライン化された演算の実行状態の初期状態は「未要求」である．処理ノードがタプルが要求されると，ストレージノードはこれを「未要求」状態から「未承認」に遷移させ，要求した処理ノードの識別子を記録した後に，当該タプルのデータを処理ノードに提供する．受信した処理ノードは，処理結果を対応するタプルの識別情報を付随させて結果ノードに送信し，結果ノードは処理ノードの出力を入力として承認すると，その旨をストレージノードに送信する．承認を受信したストレージノードは，該当タプルに対する実行状態を「処理後」に遷移させる．

ストレージが処理ノードの障害を検知すると，故障ノードに割り当てられた「未承認」のタプルを「未要求」状態に戻す．これによって，故障ノードが引受けており，かつ結果ノードの入力としては未確定である処理が，その他の正常に動作している処理ノードに引き継がれる．

全ての読み出しタプルが「処理後」状態に遷移すると問合せが完了する．「未承認」のタプルは前述の条件 (1) を満たす範囲に，「処理後」のタプルは条件 (2) を満たす範囲に相当するので，提案のプロトコルに従うことにより，障害が発生する環境下でも漏れと重複の無い処理の分配を達成できる．

表 1　実験環境の諸元

	処理・結果ノード	ストレージノード
Instance	t2.medium	t2.xlarge
CPU	2 VCPUs	4 VCPUs
Memory	4 GB	16 GB
Storage	8 GB	512 GB (25,000 IOPS)
	(general-purpose)	(provisioned)
OS	Amazon Linux 2	Amazon Linux 2

集約演算への拡張：問合せが集約を含む場合は，処理ノードにおける局所集約と，結果ノードにおける大域集約を併用することとする．この際，処理ノードでは複数のパイプラインの出力が併合されて，その結果が纏めて結果ノードに通知される．この場合には，纏められた複数のパイプラインの各々を結果ノードが識別できるように，タプルの識別情報を添えて結果ノードに通知する．

4. 評価実験

提案手法に基づく並列データベースシステムの問合せ実行エンジンを試作し，パブリッククラウドにおいて評価実験を行った．

実験環境：Amazon Web Services が提供するパブリッククラウドを用い，表 1 に諸元を示すノード群を図 1 (a) に示すとおりに相互接続し，実験環境を構築した．

評価用問合せと進捗の追跡：問合せの対象として，TPC-H ベンチマークの 3 表（`customer`，`orders`，`lineitem`）及び `orders` 表の `o_custkey` 列，`lineitem` 表の `l_orderkey` 列の B+木索引を生成し，共有ストレージに格納した．なお，scale factor は 100 とした．

図 2 に示す問合せの処理を，索引結合を用いて行った．即ち，処理ノードは `customer` 表のタプルを読み出し，選択されたタプルの結合キーをストレージに送信し，結合タプルを問い合わせる．ストレージは索引を利用して該当キーを持つ `orders` 表のタプルを検索し，処理ノードに提供する．続いて同様に，処理ノードは `lineitem` 表の結合対象のタプルをストレージより取得する．集約は，先に述べたように，処理ノードによる局所集約と結果ノードによる大域集約により行った．局所集約においてはバッファが充足されるまでパイプラインを併合して集約することとした．このバッファの長さを，以降局所集約バッファ行数と呼ぶことにする．

問合せ進捗の追跡は，`customer` 表の各タプルの 3 状態管理により行った．処理ノードが `customer` タプルを読み出すと，ストレージは該当タプルを「未承認」状態に遷移させる．結果ノードが処理ノードによる局所集約の出力を承認し，ストレージノードがその通知を受けると，入力に該当する `customer` タプルを「処理後」状態に遷移させる．

ノード故障時の動的障害回復の動作挙動：Scale factor 100 のデータセットに対して問合せを実行し，途中でノードの停止（500 秒後，530 秒後）と投入（570 秒後，600 秒後）を行った．結果，750 秒以内に問合せ処理が完了した．ストレージノードの送信スループット，及び各処理ノードのストレージ側からの受信スループットの推移を図 3 に示す．処理ノードの同時稼働台数に比例した読出しスループットの推移が確認できる．なお，ノード故障を発生させた場合と発生させない場合とで問合せ結果を比較し，同じ結果が得られていることを確認した．

```
SELECT l_orderkey, sum(l_extendedprice * (1 - l_discount)),
o_orderdate, o_shippriority FROM customer, orders, lineitem
WHERE c_mktsegment = 'BUILDING' and c_nationkey < 2
and c_custkey   = o_custkey  and l_orderkey = o_orderkey
and o_orderdate < 1995-03-15 and l_shipdate > 1995-03-15
GROUP BY l_orderkey, o_orderdate, o_shippriority
```

図 2　評価用問合せ

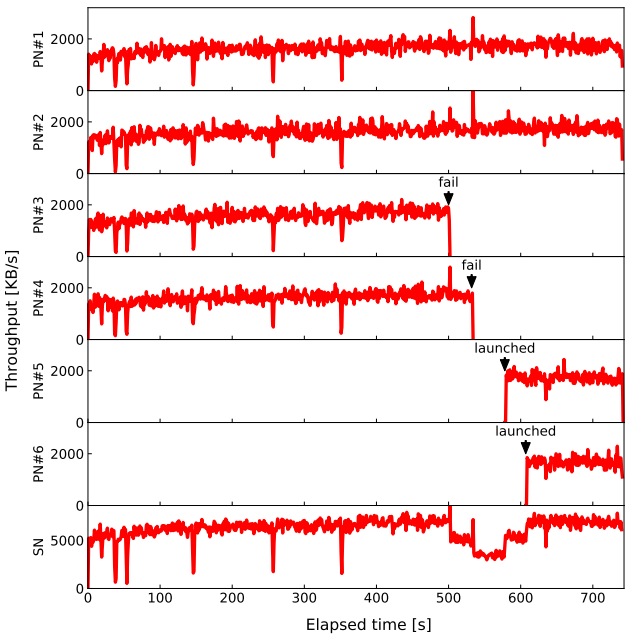
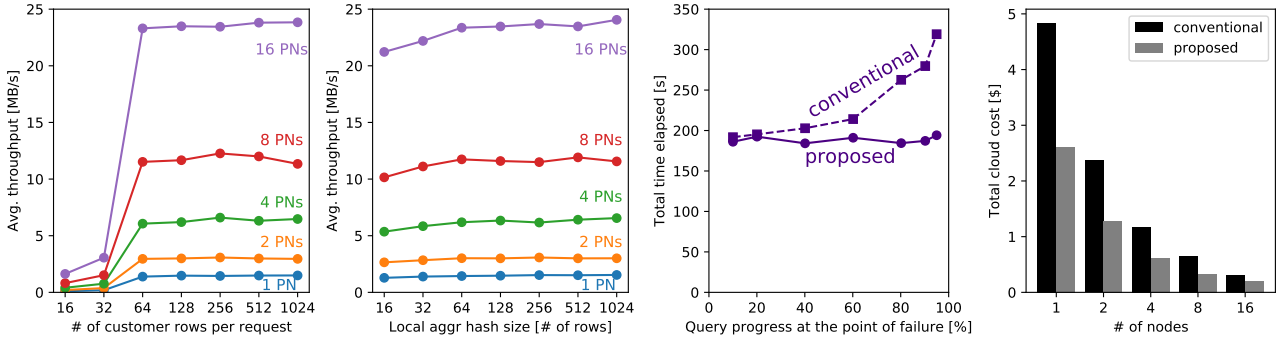


図 3　ノード故障時の動的障害回復の動作挙動．上 6 段が各処理ノード，最下段がストレージノードの挙動を表す．

性能特性とスケーラビリティ：各種パラメタを変化させて，処理性能を測定し，性能特性を確認した．まず，処理ノードが一度に要求する `customer` 表の行数と平均スループットの関係（局所集約バッファ行数は 128 に固定）を図 4(a) に示す．一度に要求する `customer` の行数，すなわち入出力追跡の粒度がある程度大きく，性能特性が安定と限らないクラウド環境上でも，凡そ処理ノード数に比例したスループットが観測できた．次に，局所集約バッファ行数を変化させた際のスループットの変化を図 4(b) に示す（処理ノードが一度に要求する `customer` 表の行数は 128 に固定）．この場合も，局所集約バッファ行数がある程度大きければ，処理ノード数におよそ比例した性能が得られた．

障害回復時間の削減効果：問合せ処理の実行中に単一ノード故障を発生させ，従来手法により問合せ処理を最初



(a) 入出力追跡の粒度によるスループットの変化　(b) 集約バッファサイズによるスループットの変化　(c) 故障時の問合せ進捗率と実行時間の関係　(d) 進捗率 90%の故障におけるクラウドコストの削減

図 4　耐障害型問合せ実行方式の性能特性と動的障害回復の効果

から再実行する場合と，提案手法によりその 3 秒後に同仕様のノードを追加投入し問合せ処理を継続する場合を比較した．図 4(c) に処理ノード 16 台，異なる故障時進捗率（故障発生時点での問合せ処理全体に対する進捗の割合）における総実行時間の比較を，図 4(d) に故障時進捗率 90%，異なる処理ノード台数でのクラウド運用コストの比較を示す．なお，一度に要求する `customer` 表の行数及び局所集約バッファ行数はともに 128 とした．従来手法では故障時進捗率が高い程ペナルティが大きいのに対して，提案手法では殆どペナルティが観測されず，有効性が明らかである．

5. 関連研究

MapReduce [4] をはじめとする並列処理フレームワーク等は，処理の中間出力を都度永続化し，直近の永続化時点からの実行状態の復元が可能である．しかしながら，それに伴う性能オーバヘッドは一般に無視できない．データベースシステム一般においては，バックアップノードへの定期的同期によるシステムの障害回復が行われる [5]．ストリーミング処理を対象とした研究では，問合せ結果の正確さを犠牲にしつつ，その処理中の動的かつ効率的な障害回復を実現する方式が提案されている [6,7]．J. Smith は，障害回復に伴う重複データの排除を目的として，上流ノードによるバックアップと識別子付ログ配送手法を提案している [8]．また，別の研究では，下流ノードが行番号単位で上流ノードの処理進捗を追跡して重複を排除する手法を提案している [9,10]．特に後者は出力重複の排除だけでなく再計算コスト削減を目指しており，手法と目的が本研究に近い．しかしながら，共有ストレージにおける動的障害回復をめざす先行研究は筆者らの知る限り存在しない．

6. おわりに

本論文は，共有ストレージ構成の並列データベースシステムを対象に，ストレージに対する入出力を起点として問合せ処理の進捗を追跡することにより，ノードに障害が発生した際に動的に他のノードがその実行状態を引き継ぎ，問合せ処理を継続することを可能とする手法を提案した．また，当該手法に基づく並列データベースシステムの試作機を示し，パブリッククラウドにおいて 16 インスタンスを用いて行った試験結果を示し，当該手法が正しく機能することを示した．今後は，入出力性能によるオーバーヘッドの評価を行う．また，本論文では選択・射影・結合後に集約を行う典型的な問合せを対象としたが，今後は他の多様な問合せに適用可能とするべく，手法の拡張に挑戦したい．

参考文献

[1] Facebook, Hadoop, and Hive (online). http://www.dbms2.com/2009/05/11/facebook-hadoop-and-hive
[2] The Internet of Things: Data from Embedded Systems Will Account for 10% of the Digital Universe by 2020 (online). https://www.emc.com/leadership/digital-universe/2014iview/internet-of-things.htm
[3] Kazuo Goda, Takayuki Tamura, Masato Oguchi, Masaru Kitsuregawa. Runtime Load Balancing System on SAN-connected PC Cluster for Dynamic Injection of CPU and Disk Resource — A Case Study of Data Mining Application —. Proc. DEXA, 182-192, 2002.
[4] J. Dean, S. Ghemawat. MapReduce: simplified data processing on large clusters. Commu. ACM, 51(1), 107-113, 2008.
[5] S. Bartkowski et al. High availability and disaster recovery options for DB2 for Linux, UNIX, and Windows. IBM Redbooks, 2012.
[6] M. A. Shah, J. M. Hellerstein, E. Brewer. Highly available, fault-tolerant, parallel dataflows. Proc. SIGMOD, 827-838, 2004.
[7] J. H. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, S. Zdonik. High-availability algorithms for distributed stream processing. Proc. ICDE, 779-790, 2005.
[8] J. Smith, P. Watson. Fault-tolerance in distributed query processing. Proc. IDEAS, 329-338, 2005.
[9] J. O. Hauglid, K. Nrvg. PROQID: Partial restarts of queries in distributed databases. Proc. CIKM, 1251-1260, 2008.
[10] B. Han, E. Omiecinski, L. Mark and L. Liu. OTPM: Failure handling in data-intensive analytical processing. Proc. CollaborateCom, 35-44, 2011.