日本データベース学会和文論文誌
Vol. 25-J, Article No. 1, 2027年 3月

工場IoTに適したクラウドオブジェクトストレージ利用関係データベースのキャッシュストレージ容量の評価

黒川能毅[1]　渡辺聡[2]　西川記史[3]　茂木和彦[4]
清水晃[5]　藤原真二[6]

工場IoTにおいては不良原因の追究のため，各工程のログをたどり機器の状態を収集し分析をする．分析では収集したログを使い工程間の関係をグラフ構造で表したデータをたどる処理を行う．関係データベースシステムでグラフ構造を扱うためには再帰クエリを使用する．一方，クラウドでのデータ格納にはコスト観点からオブジェクトストレージを選択するケースが多いが，レイテンシの問題がある．工場IoTでは再帰クエリが高速に動くことが低コスト化の条件となる．仕掛品削減の努力で複数の工程が短時間のうちに処理されることにより，そのデータは近い時刻のタイムスタンプを持つ．これにより関連するデータは近い時間でデータベースに格納され，結果物理的近傍に配置される傾向を持つ．これを生かすために，オブジェクトストレージにキャッシュとして高速ストレージを利用した関係データベースシステムを使用し，データ特性を利用した読み飛ばしにより低コスト化を行った．工場IoTを模した4mbenchを用いて，データを日時順にインポートし，再帰クエリ時間を計測した．結果，キャッシュストレージ容量をインポートデータの約3%に抑えつつ，レンジインデックスによる読み飛ばし機能で再帰クエリの性能を維持できることを確認した．これよりクラウドでのオブジェクトストレージを利用した関係データベースシステムで，工場IoTの不良原因追及を低コスト化する見込みが立った．

1　研究の背景

2018年ごろからIoTを活用した工場での生産が開始されている [3]．工場IoTでは生産時の各工程でのログデータを取得し，データを統合，データから製品個体の生産状況や材料を検索し紐づけ，検証することにより不良品判定，歩留まり改善，品質向上などを狙う [4].

いくつもの工程からなる工場ラインのログデータは．各工程のログを統合して扱う必要がある．このためにグラフ構造を用いて生産される製品に紐づくデータを結びつける．そこで，現在一般に普及している関係データベース（RDB）を用いてグラフ構造を扱う事が検討されている [9]．日立でも工場IoT向けに分析用RDB使用してグラフ構造のログデータを扱っている [12, 19, 20]．工場IoTでは，ランニングコストが直接製品価格を押し上げるため，コスト削減が重要となる．一方，これまで工場内でオンプレミスなシステムを使用して扱われていた工程ログデータが，データのセキュアな取り扱い環境が整ったパブリッククラウドでも扱えるようになった．パブリッククラウド上で行うことにより，ベンダーは工場IoTをサービスとして顧客に提供でき [4]，また，スケールアウトが簡単に行う事ができる特徴を生かし，サービス規模を簡単に拡大できるメリットがある．

しかし，パブリッククラウドではストレージが高価であり，大きなデータ量を取り扱う必要があるIoT分野では，データ保持のコストがかかるデメリットがある．工場IoTをパブリッククラウドでサービス化する事を考えた場合，ストレージのコストが直接製品製造コストとなる．この解決策として，パブリッククラウドでは，Amazonが一般的なストレージをブロックとして扱うのではなく，オブジェクトとして扱う安価なオブジェクトストレージを発表し [2, 7]，セキュアにコストを下げる事ができるようになった．しかし，RDBでは先に触れたブロックとして扱うブロックストレージを前提としているため，低速かつオブジェクト毎の扱いとなるオブジェクトストレージを活用するためには，仕掛けが必要になる．仮にデータをオブジェクトストレージに一括で保存した場合，そのサイズは搭載メモリを上限とすることになる．これはオブジェクトをメモリに搭載可能なサイズの複数に分割することで解決は可能であるが，性能が低くなる問題は引き続き存在することになる．

2　工場IoTでのアプリケーション

2.1　アプリケーションの概要

工場IoTにおいては不良原因の追究のため，各工程のログをたどり機器の状態を収集し分析をする．これには，以下の手順となる．①工場の各ライン中の工程で使用されているログを集約し，グラフを使用したSQL処理を行う．②ログ間の関係を結びつける．③個々の製品で使用した部品ロットや作業を特定する．④それらの製品が正しく生産されたかを検証する．各工程が出し入れする部品や製品をノードと見立てたグラフ処理が使用される．例えばシリアル番号，ロット番号，製造ログ番号などをグラフのノードとして，それらを検索によってエッジを検出し，ノードを順次たどり，結果として関連するデータを引き出す．これらグラフ処理を表形式のデータとしてSQLでの処理を考える場合，次のノードを探すために表結合処理（ジョイン）を繰り返すことで，各製品のノードをたどる処理を表現することができる．これは，工程の段数分繰り返されるため，一般的にはジョインの再帰処理を行うSQLとなる．

2.2　4mbenchモデルと再帰クエリ

工場IoTでの処理を想定したベンチマーク4mbenchを東京大学と日立製作所の共同研究として発表した [11]．図1に4Mモデルを示す．4Mは工場の各工程を4個のM(huMan,

[1] 正会員　(株)日立製作所研究開発グループ
yoshiki.kurokawa.ee@hitachi.com
[2] 正会員　(株)日立製作所研究開発グループ
satoru.watanabe.aw@hitachi.com
[3] 正会員　(株)日立製作所研究開発グループ
norifumi.nishikaw.mn@hitachi.com
[4] 正会員　(株)日立製作所研究開発グループ
kazuhiko.mogi.uv@hitachi.com
[5] 正会員　(株)日立製作所サービスプラットフォーム事業本部
hanako@sample.email.ac.jp
[6] 正会員　(株)日立製作所サービスプラットフォーム事業本部
shinji.fujiwara.yc@hitachi.com

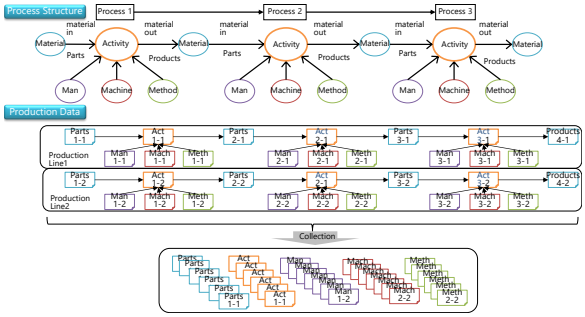

図1　4Mモデル

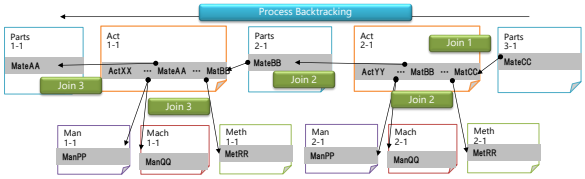

図2　バックワード処理の様子

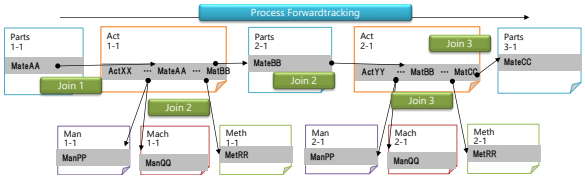

図3　フォワード処理の様子

Machine, Method, Material)の要素で表すモデルで，工程が人間(huMan)，機械(Machine)，方法(Method)で構成され，入力される材料(Material)から出力される製品(次工程に入力される材料Materialに該当)となるモデルである．

本モデルをもとに，製造ログが各要素で整理することができる．4mbenchではこのモデルで想定されるログファイルを作成し，これをもとにデータベーステーブルを構成，製品のログのバックトレース（製品がどのような材料でどう作成されたか）やフォアードトレース（材料がどの製品に使われているか）などを検索するものである． 4mbenchでの再帰処理のSQLの一例として図2にバックワード処理の様子，図3にフォアード処理の様子を示す．各工程をグラフとして処理し，そのクエリは再帰クエリで構成されるのが最大の特徴となる．

2.3　再帰クエリのアクセス特性

この再帰クエリは工程の段数だけ再帰でジョイン処理をくりかえすため，非常に作業量の多い，メインの処理となる．工場IoTでは再帰クエリが高速に動くことがシステム性能を左右することとなる．

ここで，再帰クエリの特性を説明するために，ある製品を生産する工場IoTでの一例を用いて紹介する． SNSの関係性解析などで使用されるグラフ構造とは大きく異なる． SNSの関係性解析では，幅広いリンク数のバリエーションを持つノードが不均一に分布している構造を有し，データをたどるにはほとんどランダムアクセスに近いアクセスになる [21]．一方，工場IoTでのデータ

```
/* Equipment failure influence analysis (4mQ.5a) */
WITH T_OUT (LID, E_EID, E_ENAME, EL_SENSOR, EL_READING, MLID_SRC, OL_TSEND,
            ML_MTYPE, ML_MLID, ML_OLID_SRC, ML_OLID_DST) AS (
  SELECT EL_LID, E_EID, E_ENAME, EL_SENSOR, EL_READING, ML_MLID, OL_TSEND,
         ML_MTYPE, ML_MLID, ML_OLID_SRC, ML_OLID_DST
  FROM EQUIPMENT_LOG
     INNER JOIN EQUIPMENT ON EL_EID=E_EID AND EL_LID=E_LID
     INNER JOIN OPERATION_LOG ON EL_EID=OL_EID AND EL_LID=OL_LID  AND OL_TSBEGIN=EL_TS
     INNER JOIN MATERIAL_LOG ON OL_LID=ML_LID AND OL_OLID=ML_OLID_SRC
  WHERE
    EL_EID=4 AND EL_SENSOR='PRESSURE' AND (EL_READING < 10 OR EL_READING > 20)
    AND EL_TS BETWEEN DATE'2022-07-28' AND DATE'2022-07-28' + 1 DAY
  UNION ALL
  SELECT T.LID, T.E_EID, T.E_ENAME, T.EL_SENSOR, T.EL_READING, T.MLID_SRC, T.OL_TSEND,
         M.ML_MTYPE, M.ML_MLID, M.ML_OLID_SRC, M.ML_OLID_DST
  FROM T_OUT T INNER JOIN MATERIAL_LOG M ON T.LID=M.ML_LID AND T.ML_OLID_DST=M.ML_OLID_SRC
  WHERE M.ML_MTYPE IN ('MT09', 'MT08', 'MT07', 'MT06')
)
SELECT LID, E_EID, E_ENAME, EL_SENSOR, EL_READING, MLID_SRC, OL_TSEND, ML_MTYPE, ML_MLID
FROM T_OUT ;
```

図4　再帰処理の例 4mbench 4mq5aバックワード処理

は必ず材料から製品への因果関係が存在し，時系列で材料から各工程を経て製品となる一連のつながりがグラフとなっている．つまり，SNSの関係性グラフと大きく異なる部分は，工場IoTグラフは時間的に限られた範囲への関係性しか存在しない．そのため，例えばある日の不良解析のために生産された製品が通過した工程や使用した材料をバックトレースする場合，関連する工程や材料は範囲が時間的に限定された部分に集中する．これは，ある一定期間の部品や生産された仕掛品がロットとして管理されることからもわかるように，集中的に生産される仕掛品や製品は限られた材料や上流の仕掛品のロットから生産されるため，範囲が限定されることになる．つまり時間的に近い製品は，限定されたロットに関連する材料や仕掛品との関係性しか持たない性質を持つことになる．これは同一日の製造品だけではなく，仕掛品を別工場などで別日に生産している場合でも時間的には製品との関連は薄くなるものの，ロットでの管理により，製品に関連することになる仕掛品は限定されたものとなる．

この他にも工場IoTでは過去の同様の不具合が発生した場合の影響分析なども行われる．これらの分析にはデータ全体に対しての検索が必要となるが，各段にリソースを使用する処理となり，毎日行う必要もない．その日の不良品のバックトレースなどのように，一定の間隔で一定の時間内に終了させるような実時間性が求められるような処理では，限定された材料や製品に対して関連するデータが限定される性質を利用できると考えられる．

2.4　再帰クエリの最適化方針

図4に2.3で示した工場IoTでの再帰処理のSQLの一例を示す．再帰処理ではアンカーと呼ばれる部分により，再帰の土台となる中間テーブルを生成し，再帰の繰り返しの中では，中間テーブルとデータを保存しているテーブルとのジョインにより次の中間テーブルを作成する．これを工程分繰り返し，最終的にバックトレースであれば，材料，フォワードトレースであれば製品にたどり着くように動作する．再帰処理時には土台及び再帰中にジョインするテーブルについて，前述のように特定の日付や特定のロットというような限定をつけることにより，全データの検索を避け一部のデータのみの読み出し，つまり不要データの読み飛ばしが可能となると考えられる．読み飛ばしの条件はアンカーに記載されているが，この条件を再帰の中間テーブルとのジョイン時にも適用することで，再帰クエリの再帰部分での読み飛ばしも可能となる．読み飛ばしを行うことで，繰り返される全テーブルの読み出しを全て回避することができ，性能向上が期待できる．

読み飛ばしを判定するにはインデックスを使用するが，インデ

ックスの中でもレンジインデックスと呼ばれるものを使用する．レンジインデックスは一定の範囲のテーブルデータに対してインデックスの対象とするカラムの値について最大値と最小値を持つ．インデックスを引くときは，インデックスに設定されている最大値と最小値を先に読み出し，対象カラムの検索範囲がそのテーブルデータ範囲に含まれるかを判定してそのテーブルデータ範囲を読み出すか読み飛ばすかの判定を行う．このレンジインデックスを今回の読み飛ばし条件に使用し，再帰クエリにも適用できるように実装することで，再帰クエリの高速化が図れると考えられる．

2.5 オブジェクトストレージ使用関係データベースの最適化

前節で示したように再帰クエリの高速化が図れると考えられる一方，工場IoTを含め前出のクラウドでのデータ格納にはコスト観点からオブジェクトストレージを選択するケースが多い．オブジェクトストレージは，データをオブジェクトとして扱い，オブジェクト単位でのデータの入出力を行う記憶装置で，通常はデータを使用するホストからネットワークを介して接続され，オブジェクトの入出力を行う．特徴として以下の２点がある． (特徴1) ネットワークを介しての接続のため多くの場合はそのサーバが持つ通信バンド帯域が最大のアクセス速度となる．今回実験で使用したインスタンスでは10Gbps，つまり1.2GB/s程度が限界となる． (特徴2) ネットワークを介してリクエストを発行して結果が帰るのを待つため，レイテンシが数10msから数100msと，帯域と比較して非常に長い特徴を持つ．つまり，ネットワーク帯域の上限まで性能を使うためには，同時に並列に多数のリクエストの発行と受け取りを行う必要がある．

データベース全体は数TBとなることも多く，データベースのデータをオブジェクトとして扱う場合にデータ全体を１つのオブジェクトにすると搭載するメモリの量が足りなくなる可能性がある．そのため，データを小さな単位に分割し，それらをオブジェクトを処理する部分から順に読み出すことによって処理するようにする．分割単位は，データベースとして管理する単位であると処理がスムーズになるということと，小さすぎると読み出しデータ量に対してレイテンシが大きすぎて性能が出ないため，数MBから数10MBの大きさが適当と考えられる．

ここで全てのデータをオブジェクトストレージから読み出すと，メモリに乗り切らないオブジェクトから消去されて，仮に後で同じオブジェクトから別なデータを読み出す必要が出ても，消去された後では再度オブジェクトストレージからとなり，効率が悪い．そこで，著者らは高価ではあるが，高速なSSDストレージを併用し，一度読み出したオブジェクトをSSDストレージにも保存し，再度同じオブジェクトのリクエストがあったら，代わりにSSDから読み出すことで高速にデータをアクセスできるようにする構成とした．

この場合の課題としては，データベースエンジンの動作として一度はオブジェクトストレージからの読み出しを行わなければならないため，クエリの初回動作時，とくにコールドスタートからのクエリ初回動作ではオブジェクトストレージからの読み出しとなり，速度が低下する事が考えられる．

工場IoTでの処理を想定した場合，処理ネックとなるのは前述の通り，バックワードトレース，フォア―ドトレース時の再帰クエリとなる．ここで再帰クエリでのデータのアクセスパターンを考える．データは工場の各工程のログとしてデータベースに保存される．データの保存はシステム性能上の問題からトランザクションが使われる事はまれで，数十分から数時間に１回集中して一括インポート処理で行われる．このため，データはインポート単位で物理的にかたまった場所に保存される．オブジェクトストレージの場合も，分割する数量を少なくするために，一括インポートで少ないオブジェクト数にするようにデータを保存するため，結果としてインポート単位でデータがかたまることになる．また，仕掛品削減の努力で複数の工程が短時間のうちに処理されることにより，そのデータは近い時刻のタイムスタンプを持つ．これにより関連するデータは近い時間でデータベースに格納され，結果物理的近傍に配置される傾向を持つ．つまり，データの空間的な局所性を持ち，キャッシュシステムが機能する可能性が高い．再帰クエリにおいては，再帰１回目の読み出しでオブジェクトストレージから読み出されたオブジェクトをキャッシュとしてSSDに保存しておくことによって，再帰２回目以降はキャッシュとしてSSDに保存されたオブジェクトを再利用することで性能を落とさずに処理をする構成とした．

以上のようにオブジェクトストレージを使用した関係データベースを構成可能であるが，キャッシュとして使用するSSDの，システム性能が維持できる具体的なサイズは不明である． SSDは高コストな要素のため，サイズはシステムのコストに影響を与えるためなるべく小サイズでの実装が求められる．以降，3章では関係データベースでのキャッシュ構成についてとキャッシュの挙動について検討し， 4章で工場IoTにおいての最適なSSDサイズについての評価を行う．

3 クラウドオブジェクトストレージ利用関係データベース

前節までに述べた方針に基づき，ブロックデバイスアクセス方式の超高速データベースエンジンHitachi Advanced Database[*1](以降HADBと記載)をベースに試作した．

3.1 ベースとしたHADBの特徴およびキャッシュの構成

2.3で述べたように，工場IoTのデータアクセスは各製品の細項目をほぼランダムに読み出すため，データアクセスは粒度の細かいアクセスとなる．このため，クラウド及びオブジェクトストレージに対応するクラウド版は、ブロックストレージ版HADBで採用されているブロックデバイスアクセス方式をベースとした．このブロックデバイスを小さく分割したファイル（キャッシュ単位オブジェクト）に置き換え，それらをオブジェクトストレージに配置し，必要となるデータが含まれるキャッシュ単位オブジェクトを読み出し，その中のデータをブロックデバイスと同様にブロック単位の整数倍の大きさでアラインメントが取れているデ

[*1] 内閣府の最先端研究開発支援プログラム「超巨大データベース時代に向けた最高速データベースエンジンの開発と当該エンジンを核とする戦略的社会サービスの実証・評価」（中心研究者：喜連川情報・システム研究機構機構長／東大特別教授）の成果を利用．「Hitachi Advanced Data Binder」は2025年10月より「Hitachi Advanced Database」へ名称を変更．

ータページを高速に読み書きすることで性能を担保する．図5に概念図を示す．図中，右のストレージがオブジェクトストレージ(S3)で， DBMSサーバを構成する仮想サーバ向けインスタンスとはネットワークを介して接続される．サーバ向けインスタンスはSSDストレージ上にキャッシュ領域を持つ．サーバがクエリを受け取ると，それを解釈しクエリプランを作成，実行する．実行時にデータの読み出しが発生すると，そのデータが含まれるデータページが存在するキャッシュ単位オブジェクトを選択し，キャッシュ領域に存在するかを確認する．存在しない場合，S3から読み出す．こうしてキャッシュ領域上に配置されたキャッシュ単位オブジェクトから，所定のページを読み出し，クエリを実行する．

また，データを入力する場合も同様の動作を行い，書き込みページを保有するキャッシュ単位オブジェクトがキャッシュ領域に存在するかを確認し，存在しない場合，S3から読み出す．キャッシュ領域にキャッシュ単位オブジェクトが配置されたら，所定のページを書き込こむ．書き込みが終了しそれ以上書き戻しがない，またはコミットされた時点でS3への書き戻しを行い，データ入力が完了する．データ入力には，一括でデータを入力するインポートと，トランザクション処理により一件ずつ変更または書き込む方法とあるが，双方実装している．

3.2 キャッシュヒットおよびミスの挙動

これらにより，キャッシュがヒットしている場合はキャッシュ管理オーバーヘッドが必要であるがほぼ従来版であるブロックストレージ版HADBと同等の速度で動作する．しかしキャッシュのミスが頻発するケース，特にシステムを起動してキャッシュ単位オブジェクトが全くキャッシュ領域に読み込まれていない場合などでは，データのアクセスにS3ファイルアクセスのレイテンシがオーバーヘッドとして追加される．逐次アクセスの場合等は，読み込んだキャッシュ単位オブジェクトの残りの部分がキャッシュヒットケースとして動作するため実行時間もほとんどがキャッシュヒットケースの時間となる．しかし，最悪なケースとして想定される，データベース全体のデータにランダムに散らばっているデータに対して 1 件ずつトランザクション処理するケース等では，対象となる 1 行を読み出すために，キャッシュ単位オブジェクトを読み出すため，ブロックストレージの100倍以上となるオブジェクトストレージのレイテンシの影響により，一桁以上の性能劣化の可能性がある．

3.3 キャッシュ最適化の検討

前節でキャッシュシステムを構築するとしたが，一般的に複数記憶階層を利用したキャッシュシステムを構築・利用するためにはデータに局所性があることが前提条件となる．一般的な商用CPUでのキャッシュシステムは，CPUで使用する命令列やデータにおいて2種類の局所性によって成立している [14]．1種類目は時間局所性，つまり時間的に近いデータが再利用される可能性が高いことである．ことのため，一般的に高速で低容量かつ高価な記憶階層を少量持つだけでも，そこに保存された命令やデータが再利用されるため，キャッシュとして成立する．2種類目は空間局所性，つまり記憶デバイス中の現在使用している命令，データから距離の近い命令，データが次に利用される可能性が高い

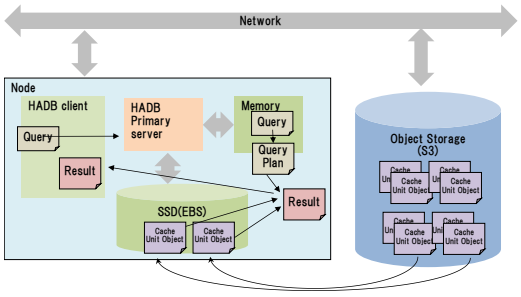

図5 オブジェクトストレージ対応Hitachi Advanced Database概略

ことである．このため，付近の命令，データを一括で読み出し，キャッシュしておくことによって，付近の命令，データを高速な記憶階層のレイテンシで使用でき，高速化に貢献し，キャッシュとして成立する．

これらの局所性はデータベースデータにも当てはまり [17]，これにより前節で述べたHADBのキャッシュシステムも効果を発揮すると考えられる．これら 2 つの局所性のうち，空間局所性については，データベースデータの記憶デバイス上の配置及び，検索条件に大きく依存する．例えばキャッシュ階層のサイズより大きなテーブルの全データを確認しなくてはならない検索条件の場合，キャッシュは一旦テーブル全体のデータを読み込もうとするが，サイズが小さいためキャッシュ階層があふれ，以前に記憶したデータの中からデータを選び消去して別のデータを保存する事を繰り返す．つまり，全体のデータを読み出し終了後にもう一度同じテーブルを使用しても，最後に読み出したテーブルの一部分のみがキャッシュに残るだけとなり，また最初からデータ読み出しを繰り返すことになる．これをスラッシングと呼ぶ．キャッシュストレージ容量が十分でない場合残されるキャッシュ単位オブジェクトに残されるテーブルデータの量が減るため，たとえキャッシュストレージ容量全てにキャッシュングしていても次回動作でその中に利用可能なデータが含まれる可能性は低くなっていくと考えられる．よって，一般的なキャッシュにおけるデータ配置の課題が工場IoTにおいてどのように影響を与えるか評価する必要がある．

4 評価

4.1 評価方法および評価環境

前出の工場IoTを模した4mbenchを用いて，全6クエリの計測を行った．今回，スラッシングが起き性能低下を起こす可能性がある場合とない場合を観測したいため，キャッシュストレージ容量を64GB，32GB，16GBとして，それ以外の条件を揃えて時間および，キャッシュされているテーブルおよびインデックスの容量を測定した．

測定にはAmazon社AWSのインスタンスおよびストレージとしてEBS(Elastic Block Storage)，オブジェクトストレージとしてS3を使用した．表1に測定条件を示す．また，表2に使用したAWSのインスタンスのスペックを示す． 2xlarge, 4xlage, 8xlargeと数字が倍になるごとにvCPU数とメモリサイズが倍となる構成となっている．

表1　測定条件

項目名	測定条件
使用コンピュータシステム	Amazon社AWSインスタンスr5bシリーズ
メインストレージ	Amazon社オブジェクトストレージS3
キャッシュストレージ	Amazon社Elastic Block Storage (EBS)
時間測定方法	timeコマンドによる実行時間測定

表2　使用AWSインスタンススペック

AWSインスタンス名	r5b.2xlarge	r5b.4xlarge	r5b.8xlarge
vCPU数	8	16	32
メモリ[GiB]	64	128	256
ネットワーク速度[Gbps]	最大10	最大10	固定10
EBS帯域[Gbps]	最大10	10	10

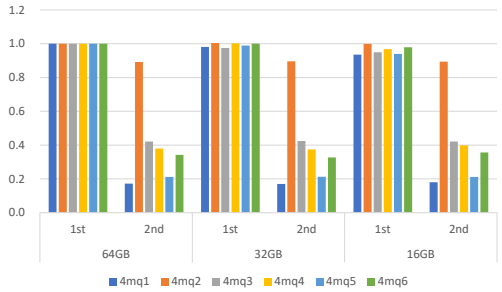

図6　4mbench実行時間測定結果（キャッシュストレージ容量の違い）

4.2　評価結果

図6に実行時間の比率を64GBの各クエリの実行時間を1.0とした相対比率で示した．クラウド環境での測定のため，ネットワークの負荷が不定であることと，オブジェクトストレージが他サービス群との共用であり，さらにキャッシュ機能を有するため，アクセスレイテンシを安定的に取ることができない．それらの影響を排除するために，測定時間帯を揃えて測定し，さらに５回測定し，上下１回の記録を外して中３回の平均をとることとした．

必ずキャッシュミスを起こす１回目の実行時間測定結果はキャッシュストレージ容量による変化はほとんどない． 16GBが少し高速であるのは測定回が64GB，32GB，16GBの順序で測定したため，前述のオブジェクトストレージのキャッシュ効果によるものと推測される．しかしオブジェクトストレージのキャッシュ機能の詳細については非公開のため，定量的な考察はできない．

キャッシュ溢れを起こしていなければ必ずキャッシュストレージの速度となる２回目の実行時間の測定結果についても，キャッシュストレージ容量による変化はほとんどない．後述するが，キャッシュ溢れを生じている16GBのキャッシュストレージ容量についても動作時間に大きな変化は観測されなかった．わずかに4mq4について動作時間が増加しているのが観測された程度である．理由については，後で考察する．4mq2が1回目と比較して他のクエリに対してあまり高速化されないのは，クエリのボトルネックがCPUであるためである．

図7に実行時間の比率をAWSのインスタンスr5b.2xlargeの各クエリの実行時間を1.0とした相対比率で示した．インスタンスを2倍のvCPU数を持つr5b.4xlargeと4倍のvCPU数を持つr5b.8xlargeの動作速度と比較する．一回目の実行時間は4mq2のみvCPU数に比例して実行時間が短縮する．これは前述のとおり，クエリの処理ボトルネックがCPUのためである．それ以外のクエリはvCPU数の増加により多少実行時間短縮となるが，r5b.4xlargeで8割付近となり，r5b.8xlargeでも前後するものの大きくは変化しない．これは，ボトルネックがネットワークを介したオブジェクトストレージにあるためである．インスタンスの構成上，r5b.2xlargeは他の２つのインスタンスと違い，ネットワーク速度がクレジット制をとり，連続アクセスをすると速度が低下する仕様となっている．一方r5b.4xlargeとr5b.8xlargeは常にインスタンスの最高速度でのアクセスが可能でありかつ同一の最高速度を持つため，ボトルネックであるオブジェクトストレージの性能で頭打ちとなり，似たような動作時間となったと考えられる．

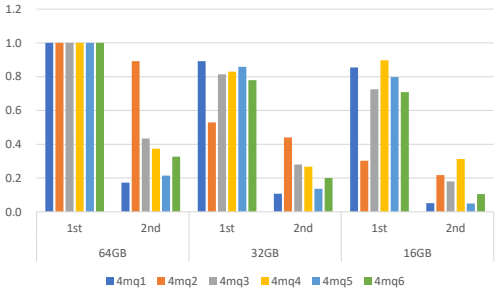

図7　4mbench実行時間測定結果（AWSインスタンスの違い）

一方，２回目は4mq4を除くクエリについてCPUネックとなったため，vCPU数に応じた動作速度短縮が観測されている．4mq4に関しては，キャッシュ溢れが生じているため，性能が向上しない．

次にキャッシュストレージ上のキャッシュ単位オブジェクト占有状況を図8に示す．クエリ毎にキャッシュストレージ容量を振り，クエリ実行後の状態について，テーブルデータ(data)，インデックス(index)，空き(space)に分けて示している．前章で議論した読みとばしによる効果によりディスクにはテーブルのうち約1日分のデータに相当する量のデータのみが残り，２回目で同じものが使用され1日分のデータでのみ高速に動作できているのがわかる．また，残りの30GB程度はレンジインデックスの領域となる．レンジインデックスは読み飛ばしのための情報が含まれており，テーブルデータと比較して少ないデータ量の読み込みでテーブルデータを絞っている．キャッシュストレージ容量16GBについては4mq3～4mq6についてキャッシュ溢れが生じている．今回のクエリでは，DBMSが読み込むのはレンジインデックスが圧倒的に多く，LRUアルゴリズムにより追い出されるキャッシュ単位オブジェクトも圧倒的にレンジインデックスが多いため，テーブルデータは追い出されず，レンジインデックスのみの入れ替えとなっている．

また，レンジインデックスはその一部のみが有用なレンジインデックスとなっており，有用な部分のみメモリにキャッシュされるため，キャッシュ単位オブジェクトへの読み出しを行わなくて

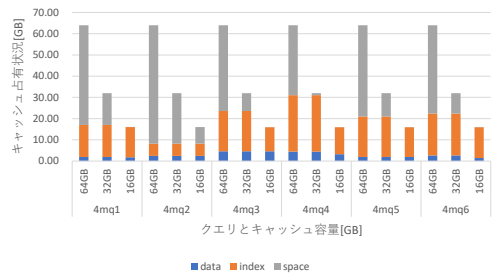

図8　テーブルデータ及びインデックスのキャッシュ占有状況

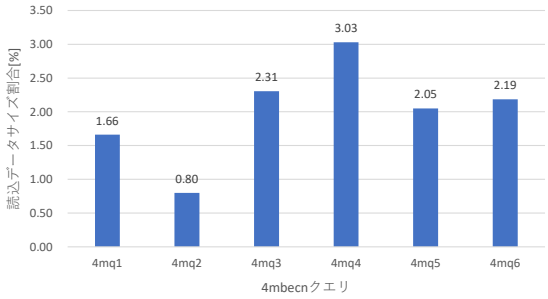

図9　入力データサイズに対する読み込みデータサイズ割合

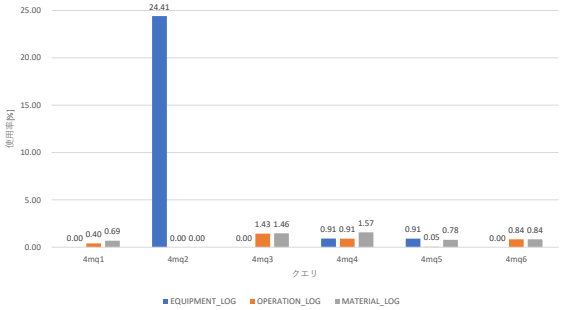

図10　テーブル全体に対しての相対的な読み込み割合

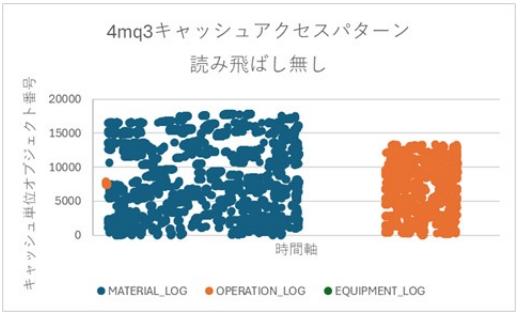

図11　4mq3キャッシュアクセスパターン読み飛ばし無し

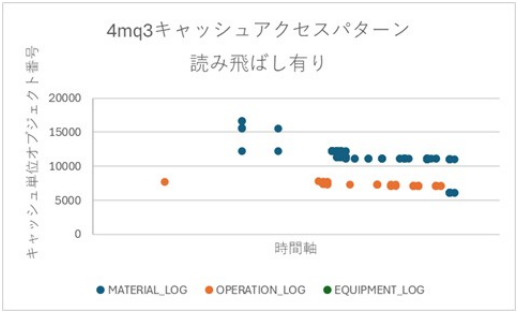

図12　4mq3キャッシュアクセスパターン読み飛ばし有り

も良い．このためスラッシングが起こる率が低くなり，性能が低下しないと考えられる．また，テーブルデータについてもキャッシュが保存され再読み込みがほとんど発生しないため，キャッシュ溢れの影響が比較的小さく抑えられている．唯一4mq4についてはキャッシュ溢れによりテーブルデータの追い出しが起こり，実行時間の増加が起きたと考えられる．

図9に入力データサイズに対する読み込みデータサイズの割合を示す．最大となる4mq4でも3.03%となる．インポート前データサイズに対して約3%のキャッシュ領域を確保することでキャッシュ溢れ無しで4mbenchを動作させることができることが分かった．

次にレンジインデックスによる読み飛ばしの効果がどれだけあったかを4mbenchのテーブルデータとして大きいMATERIAL_LOG, OPERATION_LOG, EQUIPMENT_LOGの3テーブルについて，テーブル全体に対しての相対的な読み込み割合を算出した．図10に示す．4mq2を除き読み飛ばしの効果により1日から2日分程度のデータ量の読み出しで済んでおり，高速化に貢献していることが分かる．4mq2についても，クエリのデータ絞り込み条件によって，約1/4まで絞りこめている．また，EQUPMENT_LOGテーブルは最大のテーブルであるMATERIAL_LOGの4%程度の大きさしかなく，読み出し量としては他クエリの他テーブルデータと同程度の読み出しとなり，今回の性能のネックとはなっていなく，4mq2はCPUネックとなっている．

図11及び図12に4mq3を例にとり，読み飛ばし無と読み飛ばし有りのキャッシュのアクセスパターンを図示した．横軸は時間軸，縦軸は便宜的につけたキャッシュ単位オブジェクトの番号である．キャッシュのファイルにユニークな番号を振り，番号が違えば，違うキャッシュ単位オブジェクトを読み込んでいることを示している．MATERIAL_LOGとOPERATION_LOGという２つの比較的大きなテーブルの再帰ジョインを繰り返すクエリである．各テーブルは約17000のキャッシュ単位オブジェクトとして構成されている．図11では読み飛ばしが無いため，キャッシュ番号全体に広がった読み出しが発生していることが分かる．また再帰クエリで発生するデータ読み出しに起因する同一オブジェクトについて複数回の読み出しが観測され，キャッシュ単位オブジェクトが頻繁に入れ替わっている様子が観測されている．一方，図12では読み飛ばしにより絞り込まれた限定的な数量のオブジェクトのみが読み出されており，オブジェクトストレージに対する負荷が非常に小さくなっていることが読み取れる．

これらの結果から，レンジインデックスによる読み飛ばし機能による効果により，すべてのクエリにおいて読み飛ばしによる性能向上が確認でき，またコストに直結すると考えられるキャッシ

ュストレージ容量をインポートするデータ量の3%程度に抑えることが可能であることが分かった．

本評価では，キャッシュストレージ容量をインポートデータの約3%に抑えつつ，レンジインデックスによる読み飛ばし機能により再帰クエリの性能を維持できることを確認した．この結果は，クラウド環境でのストレージ層最適化のみで実用的な性能を確保できることを示している．

5 考察

工場IoTにおいては，2.3節で述べたように，製造のパイプラインのデータを扱うためデータ間に時系列の因果関係が必ず存在すると考えられる．つまりデータは時間的因果関係を持つ空間的配置となる．具体的には，あるパイプライン内のある地点で製造している仕掛品のデータの発生時刻と，その下流の工程でその仕掛品が使用されている時刻は近く，時系列でデータが保存された場合，それら２地点のデータは関係データベースで使用するストレージ内の空間的に近い位置に保存される可能性が高いと考えられる．つまり，製造が1日に収まる場合，フォワードトレースやバックワードトレースでアクセスするデータは１日分に抑えられたと考えられる．

今回使用した4mbenchでは，１つの工場で閉じる生産について原料から製品までを模している．しかし，現実の生産は部品を生産する工場が上流にある場合がほとんどで，トレーサビリティによる品質管理には，複数工場間のサプライチェーンを含めた原料，仕掛品，工程の追跡が必要となる．このような場合においても，各工場内の管理は今回使用した4mbenchと同様なクエリによる生産データの管理が適用可能である．さらに，自動車の生産のように，それらがさらに運送で結ばれる複数工場がカスケードされる生産においても，運送を１つの工程と見立てることによって同様の戦略でのデータ管理が可能である．その時も，原料や仕掛品と製品との因果関係は保たれる．4mbenchと同様なクエリによる生産データのトレースを行った場合，ある製品工場で使用している中間製品の生産工場においても，その工場の製品である中間製品のロットは近い時間で生産され，同じかまたは時間的に近い製造を行った部品が材料となる．このため，中間製品の製造時刻と製品の製造時刻が離れていても，製品から原料や中間製品へのバックトレースにおいて工場をまたいだ場合も対象データは限定的となり，レンジインデックスを張ることにより，読み飛ばしによって高速化が可能と考えられる．

また，4mbenchのクエリは再帰処理によってバックワードトレース，フォワードトレースを行うため，工程の深さが変わっても対処可能である．部品によって工程の深さが違うのが一般的であり，それぞれの部品に対してのバックワードを一種類のクエリで管理可能であることも適用を容易にしている．

さらに，4mbenchは組み立て製造の工場を模しているが，他の製造工場，例えば切れ目のない原料を加工していく食品工場のような分野 [6]においても，原料をある程度の量で区切り，それをロットとして管理することによって4mbenchと同様の手法でのックワードトレース，フォワードトレースを行うことが可能と考えられる．

本提案手法は，オブジェクトストレージ利用時のストレージ層最適化に重点を置いている．一方，再帰クエリのオペレータ層での最適化として動的プルーニング手法が提案されており，両者は補完関係にあり，動的プルーニング＋レンジインデックスはオペレータレイヤでのI/O削減を狙い，キャッシュ＋レンジインデックスはメモリ／ストレージレイヤでのレイテンシ緩和を狙う．これらを組み合わせることで，クラウド環境での再帰クエリ処理における多層的な最適化が可能となる．今後は，ストレージ層のキャッシュ最適化とオペレータ層の動的プルーニング [16]を統合し，クラウド環境での再帰クエリ処理における多層的な最適化を実現することで，同一コストでのさらなる性能向上を目指す．

6 関連研究

スケールアウト方式の性能増強を提供するクラウドデータベースとして，Amazon社のRedshift [8, 10, 13]，Snowflake社のSnowflake [18]がある．双方ともスケールアウト方式の性能増強を行なっている．Redshiftでは4MBのデータオブジェクトを多数準備し，データをクラスターに一気に読込み，ノード間でのデータの調整を行った後，クエリを動作させる．この方式であれば，TPC-H [1]のQuery-6のようなクエリはテーブルを多数のノードに分散させ，分散処理を行い集計するだけなので，高速化が期待できる．また，ノードの割り振りなどリソース管理も自動で行うため，ユーザが実際に動作するノード数などを設定する必要が無く，スケールアウトでの高速化を行う事ができる．また，Snowflakeにおいては，4MBから50MB程度の可変の大きさのデータオブジェクトを多数準備し，Redshiftと同様に多数のノードに分散させて分散処理を行い，集計を行うため，前述のTPC-H Query-6のようなクエリの高速化が期待できる．Redshiftと比較してデータオブジェクトのサイズが大きく，可変とすることで，より高速なデータの読み出し，割り振りを実現していると考えられる．また，Snowflakeはウェアハウス構築時にコストに比例した形でノード数を倍増することができる．例えばあるデータセットでのクエリが遅い場合，ウェアハウスをアップグレードすることによって，ノード数を増加させ，結果より大きなデータセットを扱い，高速に結果を得たりなどのチューニングを行う事が可能となる．双方とも複数ノードからオブジェクトストレージに一気にリクエストを出すため，大きな読み出しバンド幅を使用して高速に全データを読み出す事ができ，全件検索が必須となる比較的単純なクエリで高性能を発揮すると考えられる．つまり，どのように蓄積されたかなどの前情報が無い大量のデータがあり，分析に全件検索を選択せざるを得ない場合など，強力な分析手段となると考えられる．

クラウド環境におけるデータベースストレージ最適化や再帰クエリ高速化に関する研究として，IBMによるDb2 Warehouseのクラウドネイティブ化研究 [15]では，LSMツリー構造とRocksDBを基盤とするKeyFile層を導入し，COS，NVMeキャッシュ，WALを組み合わせることで高レイテンシを吸収する設計を提案している．しかし，この研究はクラウドデータウェアハウスの汎用的な高速化を目的としており，工場IoT特有のアクセスパターンやキャッシュ容量の最適化指針には言及していない．

再帰クエリの高速化に関しては，動的プルーニングを提案する研究 [16]がある．この手法は，再帰ステップごとに実行時の中間結果を利用してプルーニングフィルタを生成し，不要データの読み込みを抑制することでI/Oを削減する．TPC-Hや4mbenchを用いた評価で最大135倍の高速化が報告されているが，クラウドオブジェクトストレージのレイテンシやコスト構造には対応していない．これらに対し，本研究は工場IoTのトレーサビリティ分析に特化し，クラウドオブジェクトストレージ利用時のレイテンシ問題を解決するために，高速SSDキャッシュとレンジインデックスによる読み飛ばし機能を組み合わせ，キャッシュ容量をインポートデータの約3%に抑える設計指針を定量的に示した．また，4mbenchを用いた評価により，工場IoT特有の時間的・空間的局所性を活用したキャッシュ戦略の有効性を実証した．さらに，オペレータ層での動的プルーニングとの併用により，クラウド環境での再帰クエリ処理のさらなる高速化が可能である．また本研究は，クラウドオブジェクトストレージ利用時のストレージ層最適化に焦点を当て，キャッシュ＋レンジインデックスにより低キャッシュ比率でも十分な性能を達成した点に特徴がある．一方，再帰クエリのオペレータ層での最適化として動的プルーニング手法が提案されており，4mbenchのQ3〜Q5で最大39.5倍の高速化が報告されている [16]．両者は補完的であり，キャッシュ層とオペレータ層の最適化を組み合わせることで，クラウド環境での再帰クエリ処理のさらなる高速化を実現している [5].

7 結論

工場IoTの不良原因追及をするためには，分析では収集したログを使い工程間の関係をグラフ構造で表したデータをたどる処理を行い，関係データベースシステムでグラフ構造を扱うためには再帰クエリを使用する．クラウドでのオブジェクトストレージを利用した関係データベースシステムを利用する場合では，コスト観点からオブジェクトストレージを選択するケースが多いが，レイテンシの問題がある．工場IoTでの使用を考え，以下の点から高速SSDを併用したキャッシュシステムを試作，評価した．評価の結果、以下のことが分かった。 (1)仕掛品削減の努力で複数の工程が短時間のうちに処理されることにより，そのデータは近い時刻のタイムスタンプを持つ． (2)これにより関連するデータは近い時間でデータベースに格納され，結果物理的近傍に配置される傾向を持つ． (3)レンジインデックスをデータに適用することにより，読み飛ばしによるキャッシュストレージ容量削減を達成できる．

工場IoTを模した4mbenchを用いた評価により，レンジインデックスによる読み飛ばし機能とキャッシュ最適化により，キャッシュストレージ容量をインポートデータの約3%に抑えつつ再帰クエリの性能を維持できることを確認した．今後は，動的プルーニングを組み合わせた場合のキャッシュ容量の最適なサイズを算出するアルゴリズムを検討し，さらなる最適化を目指す．

謝辞

本研究の一部は，「戦略的イノベーション創造プログラム(SIP)」「統合型ヘルスケアシステムの構築」JPJ012425の補助を受けて行った．また，本研究の一部は，東京大学生産技術研究所ビッグデータ価値協創プラットフォーム工学社会連携研究部門との共同研究の一環として実施した．

参考文献

[1] Tpc-h home. https://www.tpc.org/tpch/. Accessed 2026-03-25.
[2] What is amazon s3? https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html. Accessed 2026-03-25.
[3] スマートファクトリーの市場規模. https://robokaru.jp/factory-production-process/smart-factory-market-size/#IoT%E5%B8%82%E5%A0%B4%E3%81%AF58%E5%85%86%E5%86%86. Accessed 2026-03-25.
[4] ニュースリリース：2018年10月17日：日立. https://www.hitachi.co.jp/New/cnews/month/2018/10/1017.html. Accessed 2026-03-25.
[5] ニュースリリース：2025年6月19日：日立. https://www.hitachi.co.jp/New/cnews/month/2025/06/0619a.html. Accessed 2026-03-25.
[6] 入荷ロットと加工・包装ロットの対応づけ（内部トレーサビリティ）の解説. https://www.maff.go.jp/j/syouan/seisaku/trace/attach/pdf/index-10.pdf. Accessed 2026-03-25.
[7] James Bornholt, Rajeev Joshi, Vytautas Astrauskas, et al. Using lightweight formal methods to validate a key-value storage node in amazon s3. In *Proceedings of the 28th ACM Symposium on Operating Systems Principles (SOSP)*, 2021.
[8] M. Cai, M. Grund, A. Gupta, et al. Integrated querying of sql database data and s3 data in amazon redshift. *IEEE Data Engineering Bulletin*, 41(2):82–90, 2018.
[9] Joe Celko. *SQLグラフ原論*. 翔泳社, 8 2016.
[10] D. Dageville, T. Cruanes, M. Zukowski, et al. The amazon elastic data warehouse. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2016.
[11] Kazuo Goda, Yuto Hayamizu, Norifumi Nishikawa, and Shinji Fujiwara. 4mbench: Performance benchmark of manufacturing business database. In *Performance Evaluation and Benchmarking*, TPC Technology Conference (TPCTC 2022), pages 94–109, 3 2023.
[12] Kazuo Goda, Yuto Hayamizu, Hiroyuki Yamada, and Masaru Kitsuregawa. Out-of-order execution of database queries. *Proceedings of the VLDB Endowment*, 13(12):3489–3501, 2020.
[13] A. Gupta, D. Agarwal, D. Tan, J. Kulesza, R. Pathak, S. Stefani, and V. Srinivasan. Amazon redshift and the case for simpler data warehouses. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1917–1923, 2015.
[14] John L. Hennessy and David A. Patterson. コンピュータの構成と設計. 日経BP社, 4 1996.
[15] David Kalmuk, Kostas Rakopoulos, Robert C. Hooper, et al. Native cloud object storage in db2 warehouse: Implementing a fast and cost-efficient cloud storage architecture. In *SIGMOD Companion*, 2024.
[16] Norifumi Nishikawa, Akira Shimizu, Akira Ito, et al. Dynamic pruning for recursive joins. In *SIGMOD/PODS Companion of the International Conference on Management of Data*, pages 593–607, 2025.
[17] Alex Petrov. 詳説データベース. O'Reilly Japan, 7 2021.
[18] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, et al. Building an elastic query engine on disaggregated storage. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.
[19] 清水亮, 茂木恒一, 合田恒一, and 喜連川優. 非順序実行原理に基づく超高速データベースエンジンの詳細分析処理における性能評価. 日立評論イノベイティブ*R&D*レポート, pages 83–89.
[20] 喜連川優 and 合田恒一. 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, 8(1):131–136, 2009.
[21] 内田誠 and 白山晋. Snsのネットワーク構造の分析とモデル推定. 情報処理学会論文誌, 47(9):2840–2849, 9 2006.