RC-003

社会インフラ監視用途のための分散データストレージ設計
Design of Distributed Data Storage Systems for Infrastructure Monitoring

小林 大 † 三宅 弘士 † 上村 純平 † 山川 聡 ‡
Dai Kobayashi Hiroshi Miyake Junpei Kamimura Satoshi Yamakawa
石川 健一郎 ‡ 長谷部 賀洋 ‡
Ken-ichiro Ishikawa Yoshihiro Hasebe

概 要

社会インフラの劣化状況診断や異常監視のため，何十万ものセンサー装置を設置し，無線を利用し高頻度かつリアルタイムにデータをセンターに集約して高度な解析処理を行う社会インフラ監視システムが実現されようとしている．このようなセンサーが生成する数十から数百バイトの小サイズの大量データを格納し，解析処理や可視化システムに即座に提供するデータストレージには，OPS(Operations per second) 性能が重視される．従来の WEB や OLTP 等で用いられてきたインメモリ分散キャッシュの設計は求められる負荷傾向に合致せず，システムが巨大化してしまい不経済的である一方この用途では，遅延の大きい無線経由のアクセスのため，アクセスレイテンシ要求は緩和することができる．

本稿では，社会インフラ監視用途に特化した Staged ストレージ設計に基づいた分散インメモリデータストレージについて提案する．Staged デザインは近年 DBMS のクエリ処理等に提案されており，CPU キャッシュやメモリバス利用を効率化できる手法である．我々はこれを応用し分散データストレージの実行パスを，バッファリング，索引更新，複製作成等のステージに分割しそれぞれ一括に処理することとした．さらに，纏めて処理する要求数を調整することで，レイテンシ性能を犠牲にしストレージノード 1 台当たりの OPS 性能を大幅に向上し，従来よりも小規模で経済的なシステムで同等性能を実現する．提案する Staged ストレージ設計に基づくプロトタイプシステムを InfiniBand で結合した Linux クラスタ上に実装した結果，従来技術で 200 台以上の計算機を有する OPS 性能をわずか 8 台で実現可能であることを示した．

1. まえがき

社会インフラの異常や劣化を監視するため，何十万ものセンサー装置を設置しリアルタイムにデータをデータセンターで収集しインフラの保全運用や抑止制御へつなげるシステムが提案されている．音響振動センサーや圧力センサー等を用いた，建築構造物監視 [1] や，水道配水網監視 [2] 等は老朽化進行する社会インフラの維持管理に重要であり，また Connected Cat と呼ばれる自動車プローブ情報の無線活用 [3] やスマートフォン上のセンサーを利用した行動解析や都市最適化も実用化されようとしている．このようなシステムはサイバーフィジカルシステム [4] あるいは IoT（Internet-of-Things) と呼ばれることもある．

データセンターでは収集したデータをリアルタイム解析システムを利用して数秒以内に処理することで，異常の検知や予測，重大事故の予兆検知を含んだ従来より高度な可視化システムや，動的に流量を制御するシステム，あるいはエネルギー効率を最大化する都市運用などにつなげることができる．ストリームデータマイニング [5,6] や CEP(Complex Event Processing) [7] と呼ばれる，メモリ上でデータを格納前に処理することで高速化する技術はこのリアルタイム解析を実現可能である．しかしながら，データを格納し活用することも同様に重要である．なぜなら新規生成データと過去データを突き合わせ深く解析することで，新しい有用な解析ルールを生み出すことができるからである [8]. OLAP や MapReduce のような大量格納データを何度もアクセスし解析する深い解析（ここでは”Deep Dive 解析”と呼ぶ）がこれを可能にする．よって，リアルタイム解析システムが遅滞なくデータを処理できる速度で，データを格納処理し，後の Deep Dive 解析に提供可能なデータストレージ技術が必要である．

大量のユーザやアクセス要求を処理する従来システムとして，計算機の並列クラスタの主記憶領域で構成する分散インメモリキャッシュが利用されてきた．例えば Memcached § は WEB システムの前段で効果を発揮し，GemFire [9] は OLTP アクセスの高速化を実現する．一方，社会インフラ監視では数十バイトから数百バイトのセンサーデータが秒間何百万，何千万と要求されるため OPS(Operations Per Second) 性能が要求されるが，従来のシステムでは，このようなワークロードは考慮されておらず，必要な性能を満たすためにシステム内の計算機台数が肥大化するという欠点がある．例えば一千万個のセンサーが毎秒 30 バイトの観測データを生成し，データを三重冗長で格納する場合，後述する我々の実験による性能から推定すると既存の memcached ではデータ量にすると高々毎秒 300MByte であるにもかかわらず 200 台以上のサーバが必要とされ不経済である．

このような OPS 性能上の問題は，要求処理の数に比例したオーバヘッドに起因する．ストレージの典型的な Write 処理のデータフローに含まれる索引データ構造の更新や複製作成の各処理は，メモリ領域アクセスへの排他ロックの獲得・解放や広範囲のアドレス空間に対する細粒度のアクセスを含む．これらのメモリアクセスは CPU 間，および CPU とメモリ間の共有バス

†NEC クラウドシステム研究所, NEC Cloud System Research Laboratories
‡NEC グリーンプラットフォーム研究所, NEC Green Platform Research Laboratories

§http://memcached.org

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

を占有し，またメモリアクセスの局所性を低下させるため要求処理の性能オーバーヘッドとなる．本稿で想定するような小サイズデータアクセスの大量要求ではその数が多いため相対的にこのようなオーバヘッドは顕著になる．

想定するワークロードは高い OPS 性能を要求する一方，レイテンシ性能については従来よりも寛容である．インフラに設置されたセンサー端末とデータセンターの間は，LTE 等の無線網で接続され，そのレイテンシは数十ミリ秒から数百ミリ秒に及ぶ [10]．よって，システムアプリケーションはあらかじめ数百ミリ秒の遅延が発生しても正常に動作するよう設計されていることが多い．従来のデータストレージは 1 ミリ以下あるいはマイクロ秒レベルの性能を要求されていたが，センサー端末と解析アプリケーションを結ぶデータストレージでは，数十ミリ秒の遅延を許容することができる．

本稿では我々は斯様な環境を鑑み，レイテンシへの要求性能を緩和し，高い OPS 性能を経済的に実現できる社会インフラ監視・制御向け分散インメモリストレージ実現を目的とする．我々は Staged ストレージ設計を採用し，さらにステージ間での一括処理要求数を調整することでストレージノードあたりの OPS 性能を高めるアプローチを提案する．Staged 設計は DBMS [11] やタスクのスレッド分割で用いられる考え方で，処理の実行パスを複数のステージに分割し，個々のステージごとに処理する．ステージ内一括処理により各ステージ内で資源利用の局所性を高めることができ，オペレーションの数に起因するオーバーヘッドを削減することができる．我々は，この Staged デザインを分散インメモリストレージに応用し，実行パスにおける，データバッファリング，索引作成，複製作成等の一連の処理をステージに分割し各ステージ内で一括処理した．これにより，複数ステージの実行待ちに起因するレイテンシ増加を犠牲とし，各ステージにおけるメモリアクセスや排他制御の数の削減による OPS 性能の大幅向上を達成できる．

我々は Staged ストレージ設計によるインメモリデータストアのプロトタイプを InfinibaBand で結合した Linux クラスタ上で実装し従来技術と比較することでその効果を検証した．その結果，我々のプロトタイプは従来技術に比べ 10 倍から 2000 倍のレイテンシ悪化の代わりに 24 倍のノードあたり OPS 性能向上が得られた．また，プロトタイプの性能は 8 ノードまで良好にスケールし 1000 万 OPS を達成したことを確認した．

本稿の構成を以下に示す．続く 2. では，我々が想定する全体システム構成とストレージへの要求について論ずる．3. では提案する Staged ストレージ設計に基づくインメモリストレージについて記す．4. ではプロトタイプによる実験と結果について述べる．5. では関連研究を紹介し，6. で全体をまとめる．

2. 社会インフラ監視システムの全容

我々が想定する社会インフラ監視システムは，インフラに設置されたセンサー端末と，データストリームを受けて処理をするデータセンターによって構成される．図 1 に全体像を示す．データセンター内はゲートウェイサーバ，データストレージ，Real-time 解析エンジン, Deep-Dive 解析エンジンによって構成される．センサー端末で生成されたストリームデータは，無線ネットワークを介しデータセンターにあるゲートウェイサーバに到達する．ゲートウェサーバでは通信プロトコルやデータ構造を変更したデータをデータストレージに保存する．格納されたデータは各種解析エンジンへ提供される．

2.1. 各コンポーネントのデータストレージアクセス

各システム要素は次のようにデータストレージにアクセスする

ゲートウェイサーバ: 各ゲートウェイサーバはそれぞれ数十万コネクションをセンサー端末と構成する．センサー端末が生成する一つのデータオブジェクトをデータストレージへの一つの Write アクセスとする．オブジェクトの識別子はセンサー端末の識別子とユニークなシーケンス番号あるいはタイムスタンプからなる．一つのオブジェクトは数個から数百個の数値や文字列値からなるデータ構造で，格納後に更新されることは稀である．ゲートウェイサーバは，センサー設置業務時を除き，データストレージに Read アクセスを発行しない．ゲートウェイサーバ自体の信頼性は期待せず，ステートレスなサーバとする．つまり，サーバは現在発行している Write の格納が完了してから，次の Write を発行し，格納完了前のデータはセンサー端末上のバッファで保持する．

Real-time 解析エンジン: ストリームデータを連続的に解析する．基本的に格納データを利用せず，メモリ上に残ったわずかなデータ（Window) を利用して処理を行う．処理の内容はルールやパラメータとして展開されており，状況に応じて書き換えることができる．例として Esper ¶ や JBoss Drools ‖ 等の CEP 実装による簡単な条件マッチや，Jubatus [5] 等の高度なオンライン機械学習が利用できる．Real-time 解析エンジンによる処理結果は，SCADA 等の状況可視化システムへの統合や異常検知アラームへの展開できる．データストレージからは，最新データの Read，解析結果の Write に加え，最新データの解析に利用するアドホックな Read を発行する．よって，ストリーム数と同等の規模数のアクセス要求を発行する．センサーから取得されたオリジナルデータの更新は行わない．また，処理データに加え，Real-time 解析エンジンの状態データをデータストレージに逐次保持することにより，エンジンを構成する計算機の一部の障害時に，状態を新しい別の計算機に復帰しサービスを継続することもできる．

Deep-dive 解析エンジン: データストレージ上の過去データを解析し，統計的な情報や特徴パターンを解析する．例として，格納された全データに Map 処理と Reduce 処理を適用する MapReduce 処理がある．多くの場合データストレージ上のすべてのデータまたは，日次更新データなどまとまった量のデータに対する Read

¶http://esper.codehaus.org/
‖http://www.jboss.org/drools/

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

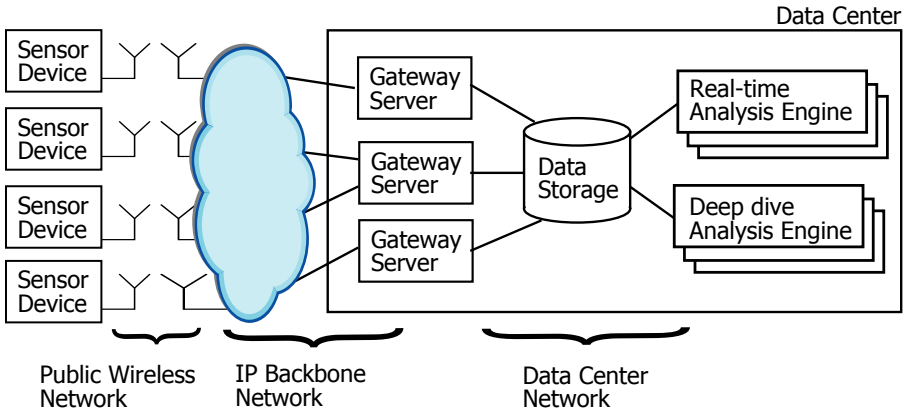

図 1: 目的とする社会インフラ監視システムの全体像

を発行し，数分から数時間かけて処理する．そしてオリジナルデータの更新は行わず，解析結果は新たなメタデータもしくは付随データとしてデータストレージに格納する．

2.2. データストレージの性能課題

ゲートウェイサーバと Real-time 解析エンジンは OPS 性能を重視し，Deep Dive 解析エンジンはデータ量に対するスループット性能を重視する．我々は，このゲートウェイサーバと Real-time 解析エンジンは OPS 性能に対する性能に着目し OPS 性能向上を課題とした．

以降ではデータストレージにアクセス要求を発行する各システムコンポーネントを，クライントサーバと呼び，分散ストレージを構成する各計算機をストレージノードと呼ぶ．

アクセス要求はクライアントサーバ上のクライアントライブラリから発行され，あるストレージノードに到達後，処理され，結果がクライアントサーバに返答される．Write 要求は対象データを含み，返答は正常に格納されたか否かを示すステータス情報が含まれる．Read 要求は対象データの識別子を含み，返答は対象データとステータス情報が格納される．

典型的な Write 処理のデータフローでは次のような処理で構成される．

- まずストレージノード上にアクセス要求をバッファリングする
- ストレージノード上にデータを格納する領域を確保する
- 確保した領域にデータをコピーしその成否を確認する
- データの場所を示す索引構造を更新する
- 冗長性確保のため他のストレージノードにデータを送付し複製格納を依頼，その成否を確認する
- 最後に返答をクライアントサーバに返す

Read 処理のデータフローでは，アクセス要求をバッファリングし，索引構造を走査し，結果データをバッファリング領域にコピーし，結果を返答する．

同時に発行されるアクセス要求の数が増加すると，要求数に比例したオーバヘッドが増加しシステム全体のオーバヘッドに占める割合が顕著になる．オーバヘッド要因の一つは，アクセス要求の処理の間で共有するバッファや索引等のデータ構造に対する排他制御である．近年の CPU アーキテクチャは，物理メモリ空間を CPU で分割して管理する NUMA 構成が浸透してきている．排他制御は CPU コア間での命令のやり取りが発生する [12] ため，CPU 間インターコネクト資源消費が他の要求処理を妨げ性能オーバーヘッドになる．別の要因として，広範囲のメモリアドレス空間に対する細粒度のアクセスに伴う，データキャッシュおよびメモリ管理用 TLB のミス率の増加があげられる．インメモリデータストアでは主記憶のメモリ空間にランダムアクセスを行う．そのため，メモリアクセスの局所性が低下し，局所性を活用した高速化手法であるキャッシュや，アドレス論理物理変換の高速化に用いる TLB の効率が低下することが知られている [13]．

その他のオーバヘッド要因として，ストレージノード間での複製作成に伴う通信コストや要求ごとに処理を行うスレッド間のコンテキストスイッチがあげられる．

3.Staged ストレージ設計による OPS 性能向上

分散インメモリストレージとして構成したデータストレージの OPS 性能を向上するため，Staged ストレージ設計を導入した．データストレージは Key-value インターフェースによりデータの入出力を行い，格納データは複数のストレージノード間に複数格納することで冗長性を確保する．この各ストレージノードにおけるデータ要求処理を複数のステージに分割し各ステージ内で多数の要求処理を一括処理することで資源利用の局所性を増やし，OPS 性能を向上する．

3.1. 分散インメモリストレージの Staged 設計

我々は Staged ストレージ設計を採用し，さらにステージ間での一括処理要求数を調整することで, アクセス要求ごとに個別に発生する性能オーバーヘッドを削

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

減し，高い OPS 性能をもつ分散インメモリデータストレージを実現した．

ステージングは DBMS [11] やフロー処理のスレッド分割 [14] で用いられる考え方で，クエリ実行を複数のステージに分割し，個々のステージごとに多数の要求の同じ部分を一括処理する．ステージ内一括処理により各ステージ内でメモリバスや通信など資源利用の局所性を高めることができ，オペレーションの数に起因するオーバーヘッドを削減することができる．資源利用の局所性は，スレッドを各ステージごとに占有すること，および各ステージが個別にメモリ領域を一元管理することでさらに高めることができる．

ステージングの欠点は，レイテンシの伸長である．各ステージに最初に投入された要求は実際にステージの処理実行が開始されるまで遅滞される．またステージ間でアクセス要求を移動する処理も時間を要する．ステージ段数が増えると，その分全体の実行時間は伸びる．

我々は，この Staged デザインを分散インメモリストレージに応用し，実行パスにおける，データバッファリング，索引作成，複製作成等の一連の処理をステージに分割し各ステージ内で一括処理した．図 2 にそのデータおよび制御フローを示す．

まずクライアントサーバ内のクライアントライブラリが複数のアクセス要求をバッファし，チャンクとしてまとめる．したがってクライアントアプリケーションからの要求はノンブロッキング API で処理する必要がある．

チャンクに一定量の要求が保持されるか，あるいは指定したタイムアウト時間が経過した後，チャンクはあるデータ分散格納アルゴリズムに基づきストレージノードに送信される．アルゴリズムの例としてデータオブジェクトの識別子をキーとしたコンシステントハッシュ方式を利用することができる．この場合，各チャンク内には同一のストレージノード向けオブジェクトが固められる必要がある．

ストレージサーバはデータチャンクをプライマリデータとして受信し，まずバッファリングする．

続いて，各チャンクはオブジェクトに分解され，各オブジェクトごとに複製作成先ストレージノードが計算される．そして，各ノード向けの送信バッファにコピーされる．この送信バッファ内オブジェクト群も改めてチャンクとしてまとめられる．前述のコンシステントハッシュの場合，プライマリデータが同じノードに保持されるオブジェクトでも，複製先は異なる場合が多い．

一定量もしくは一定時間の経過後，複製作成要求チャンクが送信される．各ストレージノードは自身のプライマリデータと，他のノードの複製データの両方を保持する．ストレージノードでは，チャンクから永続データ領域にデータをコピーする．コピー終了後，コピーの成否を含むステータスを作成し，返答バッファにステータスをコピーする．

返答は，複製作成の返答であればプライマリノードに返答される．プライマリ作成の返答であれば，プライマリ担当ストレージノードはすべての複製作成要求が成功していることを確認したのち，クライアントサーバへ返答する．

ここで我々の設計では，返答送信後，データ Read 用の索引構造を更新する設計とした．索引構造は，データオブジェクトの識別子をもとに，データの格納されたメモリ上の位置を引くためのデータ構造であり，新しいデータの挿入ごとに排他制御をする必要がある．クライアントサーバが同一データに対する Read/Write 混在のアクセスをする際は索引構造の更新は返答返却前に終了する必要があることに注意されたい．

3.2. 方式の利点と欠点

以上のような各ステージに分割してストレージノード内処理を設計することで，CPU キャッシュアクセスや共有メモリバスへのアクセスを効率化できる．各 CPU コアは一つのステージにおいて多数のアクセス要求の同じ部分を一括処理することで，ジャイアントロックにより排他制御数を削減し，メモリへのシーケンシャルなバルクアクセス発行によりキャッシュやメモリ管理に関わるオーバヘッドを削減可能である．その結果，CPU オーバーヘッドによって律速されていた OPS 性能を大きく向上できる．

Staged デザインの主要な欠点としては，先述の通りレイテンシの伸長があるが，そのほかに，メモリアクセス量の増大がある．ステージ数を増やすと，各ステージ間でのデータ転送が加わるためである．各ステージが別の CPU コアに割当たっている場合には，データ自体の転送によりメモリバスが圧迫される．

本稿では我々は Staged ストレージ設計によって基本的なデータ入出力機能を実現したが，新しい機能をステージとして挿入することも可能である．具体的には，データ量削減のためのオンライン重複排除や，メモリ上データ破壊に備えたエラー訂正符号の挿入など，エンタープライズストレージに備わる機能をステージとしてデータフローの一部に挿入することができる．また，求められるストレージの機能要件や性能要件に従い，Stage の順序を入れ替えることもできる．たとえば，先述の索引更新ステージの例がある．

データストレージとしての OPS 性能はもっとも遅いステージの性能に律速される．新しい機能を導入する場合，レイテンシが増加すること，ステージの処理速度を他のステージと同等にすることを考慮して設計する必要がある．

4. プロトタイプによる効果検証

我々は前述の Staged ストレージ設計に基づいた分散インメモリデータストレージとして DoverStor と呼ぶシステムプロトタイプを実装し，InfiniBand(QDR) で結合された Linux クラスタ上でその性能特徴を明らかにした．

4.1. プロトタイプ実装

DoverStor プロトタイプは C++を用い実装した．実装の簡便化のため，チャンクデータを一時的に保存するバッファとしてオープンソースの Key-Value ストアである Redis を，データ格納領域の管理と索引構造に，データ挿入性能に優れた独自のインメモリカラムスト

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

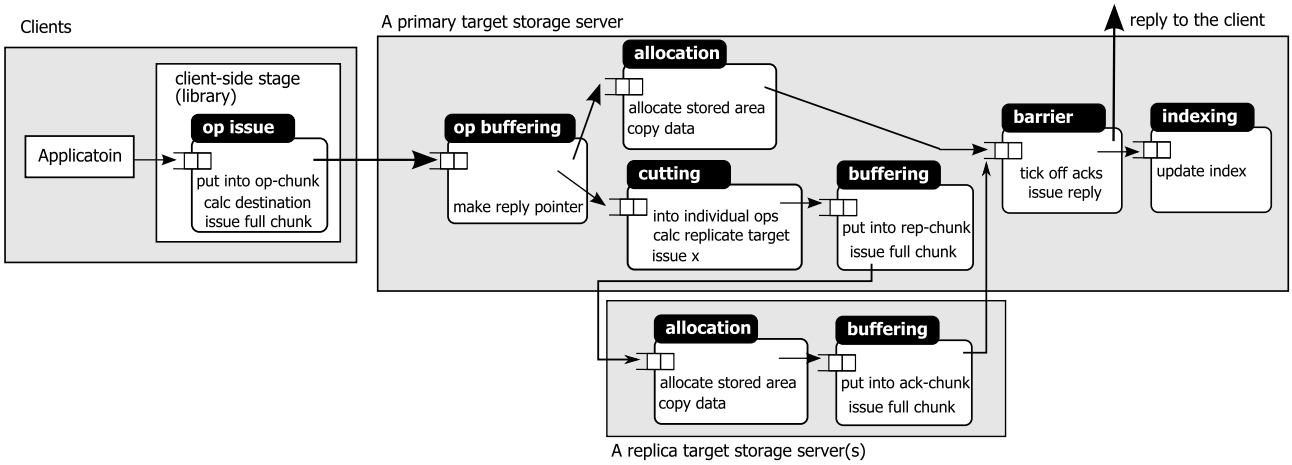

図 2: Staged ストレージ設計による Write 要求のデータフローおよび制御フロー

表 1: 実験に用いた環境の性能緒元

CPU	Intel Xeon X5550 2.66GHz (4 core)
RAM	96GB DDR3-1333
Network	QLogic QLE7340 (HBA) QLogic 12300-BS01 (switch)

アを用いている．アプリケーションが用いるクライアントライブラリには独自のノンブロッキング read/write API を実装した．通信には QDR(40G) の InfiniBand を用い，データグラム方式で RDMA を利用できる RDS プロトコルを用いた．またデータ配置の制御には DHT に類似した独自のデータ分散アルゴリズム ASURA [15] を用いた．複製配置戦略は前述のとおりプライマリバックアップ方式を採用し，一つのプライマリデータに対して二つの複製データをそれぞれ異なるストレージノードに配置する三冗長構成とした．DoverStor では Staged ストレージ設計以外にも社会インフラ監視に特化した一貫性管理やリード高速化の仕組みを実装しているが本稿の考慮外であり割愛する．

実験に用いたサーバおよびネットワークの概要を表 1 に示す．ネットワーク性能について事前に `qperf` [**] を用いた測定を行った結果，IPoIB 上の TCP では 1.82 GB/s (レイテンシ 22.7 ミリ秒), RDS では 0.8 GB/s (同 17.3 ミリ秒) であった．なお参考までに，同サーバー上の 1000Base-T 上の TCP では 0.1 GB/s (同 40 ミリ秒) であった．比較対象となる既存システムとして，memcached 1.4.15 と libmemcached クライアントライブラリによるシステムについても同環境の測定を行った．

測定に用いたアクセス負荷として，独自のベンチマークを構築した．ベンチマークプログラムでは，100 万のセンサーストリームが同時にデータオブジェクトを発行し，各ストリームは現在の Write 要求の返答を受領後，次のオブジェクトの Write 要求を発行することとした．

ストレージノード数およびデータオブジェクトサイズについては実験ごとに異なるため後述する．クライアントサーバ数については負荷生成がボトルネックにならないよう資源状況を監視した結果，memcached ではサーバと同数とし，我々のプロトタイプ向けでは 1 台とした．

4.2. 実験結果

まずはじめに，3 ノード構成のプロトタイプにおけるステージ間一括処理オペレーション数 (チャンクサイズ）を変化させ，OPS 性能，および最大レイテンシの関係の変化を測定した．データオブジェクトのサイズは 26 バイトとした．図 3 に結果を示す．図より我々のプロトタイプではチャンクサイズ 100KB の時，最大レイテンシ 12 ミリ秒で約 320 万 OPS の性能が得られることがわかる．また，チャンクサイズの増減により，100 ミリ秒のレイテンシを許容することで最大 400 万 OPS の OPS 性能を得られることがわかる．この実験により，Staged ストレージ設計が我々の求める，レイテンシ性能を犠牲にした高い OPS 性能を実現できることがわかる．

続いて，従来システムとの比較を行う．図 4 は，同仕様の計算機利用下においてストレージノード 1 台当たりの OPS 性能について，memcached について 2 種類のネットワークそれぞれ，および我々のプロトタイプ (チャンクサイズは 100KB) を比較したものである．なお memcached は冗長複製を作成しないことに注意されたい．図より，memcached は InfiniBand によるネットワーク高速化の恩恵を活用できず，冗長性のない設定で 13 万 OPS である一方，我々の DoverStor プロトタイプでは三冗長の設定において 110 万 OPS を示している．一方レイテンシでは memcached が 6 マイクロ秒に対し，DoverStor では，13 ミリ秒であった．このことから，我々のプロトタイプは同一計算機資源において，220 倍長大なレイテンシの替わりに，24 倍の OPS 性能向上を得られたことがわかる．なお，最新の研究成果として，InfiniBand に最適に実装された memcached も

**http://www.openfabrics.org/

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

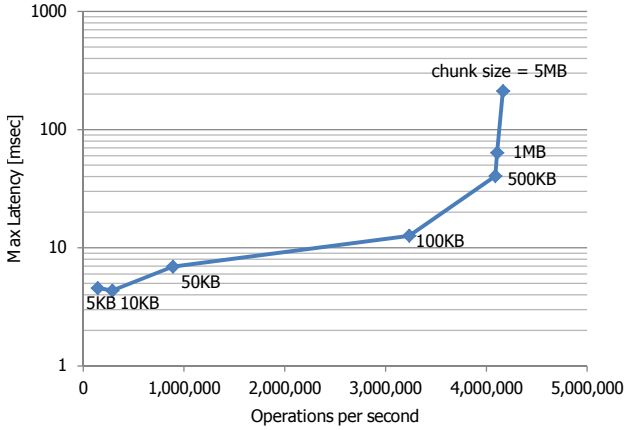

図 3: 一括処理要求数を変化させたときの最大レイテンシと OPS 性能の関係 (ノード 3 台構成)

報告されている [16]．用いている機材の性能が違うため参考比較ではあるが，同論文内で報告されている複製なし 190 万 OPS と比べても現在の DoverStor は 1.7 倍の OPS 性能を示しており，OPS 性能を第一とする社会インフラ監視用途に合致した設計であるといえる．

続く実験ではノード数を最小構成の 3 台から 8 台まで変化させ OPS 性能の変化を測定した．図 5 に示す実験結果ではストレージノード数に対し理想的に比例した OPS 性能が確認でき，8 台で 1000 万 OPS を超える性能が得られた．前述の memcahched は libmemcached などのクライアントライブラリによる分散システム構築により良好にスケールするシステムを構築できるが，理想的にスケールした 3 冗長システムが実現できたとしても，我々が 8 台で実現した 1000 万 OPS を達成するには上記実験値 13 万 OPS/ノードから約 230 ノードを要すると推定される．これより，我々の技術では高い OPS 性能のストレージをより経済的に実現できると言える．

3. で述べたように Staged ストレージ設計の欠点の一つはステージ間のデータ転送による資源利用である．この影響はシステムのデータ量に対するスループット性能に現れる．図 6 はデータオブジェクトサイズを変化させたときの OPS 性能とデータ量に対するスループットの関係を測定したものである．図より，ノードあたり 350MB/s 程度でシステムスループット性能が律速されていることがわかる．今後，センサーだけでなく動画・静止画像を利用したインフラ監視まで目的を拡大し本設計を活用するためには，データ量に対するスループット性能の向上は今後の重要な課題である．

5. 関連研究

大量のメモリと低遅延ネットワークを利用した汎用的なシステムとしては memcached や RAMCloud がある [17]．これらのシステムは DRAM の低遅延性とネットワークの低遅延性を活用し，高頻度に参照更新されるシステムと親和性が高い一方，我々が求める比較的

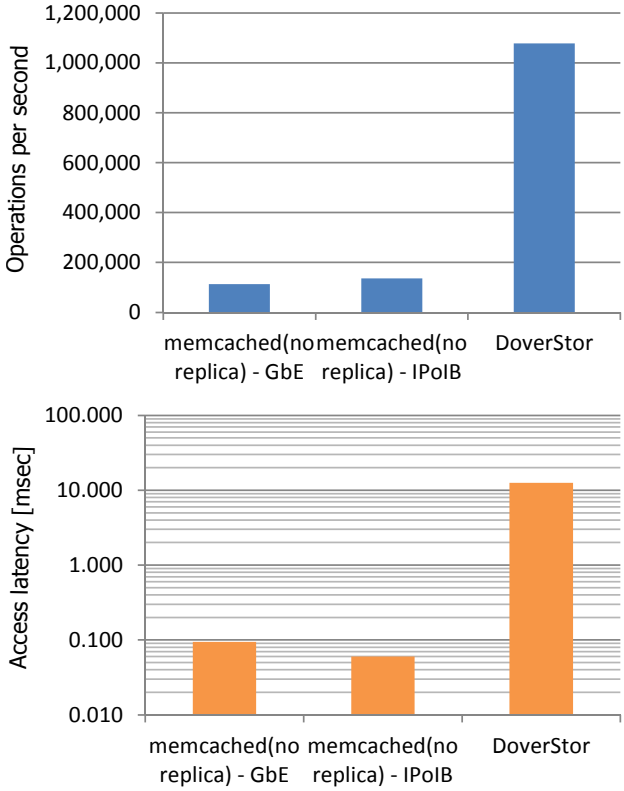

図 4: 既存技術との比較

緩いレイテンシ性能要求に対し高い OPS 性能を求められる用途では性能が上がらず，コスト効率が悪い．

一連の流れをステージングし，各ステージごとに一括処理する方式は，マルチコアの有効活用やメモリリソースの効率化として研究されている．Staged イベント駆動処理（SEDA）[14] はタスクをステージングし，各ステージへのスレッド割り当て数を調整することでリソース利用を最大化している．Staged DB [11] は DBMS においてクエリ実行をステージ分割し，各ステージでの命令キャッシュの局所性を向上している．H-store [12] もまたステージングの一種であり，複数のクエリを単一のスレッドで一括処理することで排他制御に係るオーバヘッドを削減している．我々の Staged ストレージ設計もまた同様のコンセプトの元に提案してるが，我々は特にステージングにより従来レイテンシ性能向上に利用していた資源を OPS 性能向上へと振り分ける性質に着目し利用している．

類似のアプローチに基づくシステムとして LazyBase [18] がある．LazyBase では，データ処理機能をデータストレージに内包し，データストレージ機能とデータ処理機能への資源分割を行っている．連続更新と，解析へのデータ提供を主眼に置き，処理をパイプライニングしバッチ処理で性能を向上している点で我々の提案と同様である．現在の LazyBase は HDD ベースのシステムとして設計・評価されている一方，我々のシステムはインメモリベースかつ低遅延高速ネットワークにおけるオーバーヘッドを主眼に設計している

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

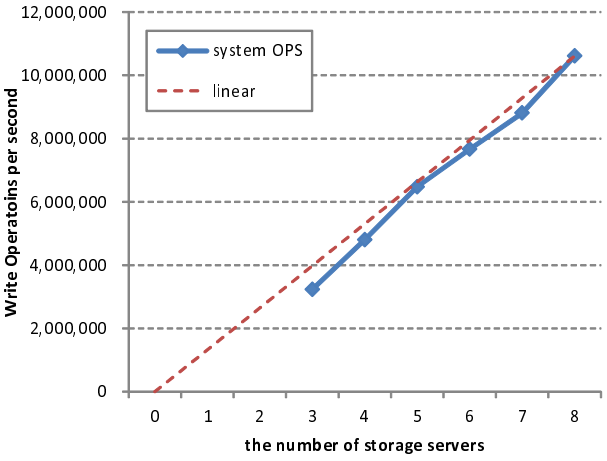

図 5: ストレージノード台数と OPS 性能の関係

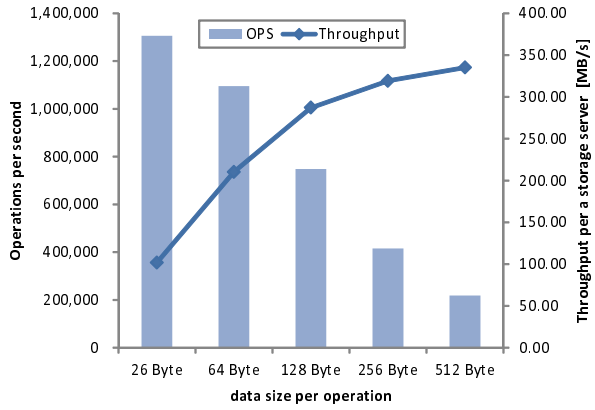

図 6: データオブジェクトサイズを変化させたときの OPS 性能とシステムトータルスループットの関係

点で設計思想が大きく異なる．この違いは，例として，LazyBase がノードをヘテロ利用し複数のステージを別のノードに配置する一方，我々の設計ではレプリケーションを除いてステージをノード内の複数コアに配置する点等で現れる．

ログ収集システムも類似システムの一種である．例えば Apache Flume ではログ発生減からのデータ受信とログ格納ストレージへのデータの送信機能をステージングしている．しかし，上記のシステムはいずれもストレージ機能は後段に外部から連結する構成を取っており，この外部ストレージが OPS 性能問題を抱える点で本稿で解決した課題が未解決である．

6. まとめ

本稿では，社会インフラ監視のために設置された多数のセンサー端末と無線により結合されたデータセンター上のデータストレージ機能として，ストレージノード一台当たりの OPS 性能を高める Staged ストレージ設計を用いたシステムの提案とプロトタイプによる評価を行った．プロトタイプである DoverStor では数十ミリ秒のレイテンシ性能悪化が許容できる用途では従来システムに比べノードあたり 24 倍の OPS 性能を示し，従来 200 台以上必要と推定される 1000 万 OPS のシステムをわずか 8 台のサーバで構成可能であった．

今後の課題として，SSD の内包と階層制御による大容量化，データの格納に対して遅滞なく解析エンジンによるデータ利用が可能な一貫性保持手法，ノンブロッキング I/F と開発容易性の両立等があげられる．また将来の展望として，提案するデータストレージを含む社会インフラ監視システムの実社会へと適用することで人と地球にやさしい情報社会実現することがあげられる．

謝辞

本研究の一部は、独立行政法人新エネルギー・産業技術総合開発機構（ NEDO ）から委託を受け実施している「IT 融合による新社会システムの開発・実証プロジェクト/（データ処理基盤分野）リアルタイム大規模データ解析処理基盤の研究開発」の成果である。

参考文献

[1] M. Ceriotti, L. Mottola, G.P. Picco, A.L. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon, “Monitoring heritage buildings with wireless sensor networks: The torre aquila deployment,” Proceedings of the 2009 International Conference on Information Processing in Sensor Networks, pp.277–288, Washington, DC, USA, 2009, IEEE Computer Society.

[2] I. Stoianov, L. Nachman, S. Madden, and T. Tokmouline, “Pipeneta wireless sensor network for pipeline monitoring,” Proceedings of the 6th international conference on Information processing in sensor networks, pp.264–273, New York, NY, USA, 2007, ACM.

[3] J. Rodrigues, A. Aguiar, F. Vieira, J. Barros, and J.P.S. Cunha, “A mobile sensing architecture for massive urban scanning,” Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on, pp.1132-1137, 2011.

[4] E. Lee, “Cyber physical systems: Design challenges,” Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on, pp.363-369, 2008.

[5] S. Oda, K. Uenishi, and S. Kinoshita, “Jubatus: Scalable distributed processing framework for realtime analysis of big data,” NTT Technical Review, vol.10, no.6, 2012.

[6] B. Gedik, H. Andrade, K.L. Wu, P.S. Yu, and M. Doo, “Spade: the system s declarative stream processing engine,” Proceedings of the

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

2008 ACM SIGMOD international conference on Management of data, pp.1123–1134, New York, NY, USA, 2008, ACM.

[7] E. Wu, Y. Diao, and S. Rizvi, “High-performance complex event processing over streams,” Proceedings of the 2006 ACM SIGMOD international conference on Management of data, pp.407–418, New York, NY, USA, 2006, ACM.

[8] I. Zikopoulos, C. Eaton, and P. Zikopoulos, Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, Mcgraw-hill, 2011.

[9] “VMware vFabric GemFire. high performance, distributed main-memory and events platform.,” Technical report, VMW_10Q3_WP_vFabric_GemFire_EN_R1, VMware, Inc., 2010.

[10] N. Nikaein, and S. Krea, “Latency for real-time machine-to-machine communication in lte-based system architecture,” Wireless Conference 2011 - Sustainable Wireless Technologies (European Wireless), 11th European, pp.1-6, 2011.

[11] S. Harizopoulos, and A. Ailamaki, “StagedDB: Designing Database Servers for Modern Hardware,” IEEE Data Eng. Bull., vol.28, no.2, pp.11–16, 2005, SYSTEMS.

[12] E.P.C. Jones, D.J. Abadi, and S. Madden, “Low overhead concurrency control for partitioned main memory databases.,” SIGMOD Conference, pp.603-614, ACM, 2010.

[13] C. Kim, J. Chhugani, N. Satish, E. Sedlar, A.D. Nguyen, T. Kaldewey, V.W. Lee, S.A. Brandt, and P. Dubey, “Fast: fast architecture sensitive tree search on modern cpus and gpus,” Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp.339–350, New York, NY, USA, 2010, ACM.

[14] M. Welsh, D. Culler, and E. Brewer, “Seda: an architecture for well-conditioned, scalable internet services,” Proceedings of the eighteenth ACM symposium on Operating systems principles, pp.230–243, New York, NY, USA, 2001, ACM.

[15] K. ichiro Ishikawa, “ASURA: Scalable and uniform data distribution algorithm for storage clusters,” CoRR, vol.abs/1309.7720, 2013.

[16] J. Jose, H. Subramoni, K. Kandalla, M. Wasiur Rahman, H. Wang, S. Narravula, and D. Panda, “Scalable memcached design for infiniband clusters using hybrid transports,” Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on, pp.236-243, 2012.

[17] J.K. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, M. Rosenblum, S.M. Rumble, E. Stratmann, and R. Stutsman, “The case for ramclouds: Scalable high-performance storage entirely in dram,” SIGOPS OSR, Stanford InfoLab, 2009.

[18] J. Cipar, G. Ganger, K. Keeton, C.B. Morrey, III, C.A. Soules, and A. Veitch, “Lazybase: trading freshness for performance in a scalable database,” Proceedings of the 7th ACM european conference on Computer Systems, pp.169–182, New York, NY, USA, 2012, ACM.

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.