Vol.2013-UBI-38 No.19
2013/5/16

# ライフログセンシングのための分散ストレージシステム

名生 貴昭[1,a)]　安田 充志[1,b)]　安積 卓也[1,c)]　西尾 信彦[1,d)]

**概要**：近年，センサデバイスを備えた携帯端末から取得できるログを蓄積・解析する研究が盛んに行われている．本稿ではセンサログを常時蓄積するシステムの課題として，大量のレコードを含むログの蓄積と瞬間的な過負荷の 2 点に注目し，ログ挿入専用の API で HBase を拡張したストレージシステムである $L^2$base を提案し性能評価を行った．本 API では write ahead log の削減および同期タイミングの最適化，排他制御の削減およびインデックスの追記型更新によって上記課題を軽減する．

**キーワード**：ライフログ，データベース，分散処理

# A distributed storage system optimized for lifelog sensing

Myojo Takaaki[1,a)]　Yasuda Atsushi[1,b)]　Azumi Takuya[1,c)]　Nishio Nobuhiko[1,d)]

***Abstract:*** In recent years, studies on accumulation and analysis of logs retrieved from a mobile terminal equipped with sensor devices have been conducted. This paper proposes $L^2$base, which extends HBase by providing specialized API for log insertion, and conduct the performance evaluation focusing on 2 points: accumulation of logs that contains large quantities of records, and tolerance to instantaneous overload as an issue on the system to constantly accumulate sensor logs from mobile devices. $L^2$base solves the above issue by reduction of the write ahead log, optimization of synchronizing timing, reduction of exclusion control and support for updating index without using index traversal.

***Keywords:*** Lifelog, Database, Distributed computing

## 1. はじめに

### 1.1 研究背景

今日，スマートフォンは一般に，測位デバイス，無線通信デバイス，センサデバイスなどを備えている．こうしたデバイスの観測値はユーザの行動と密に関係していると考えられ，人間の行動をデジタルデータとして記録したライフログと呼ばれるものの一種だといえる．これらライフログの蓄積，解析技術が発達すれば，多様なコンテキストアウェアサービスの実現が期待できるため，ユーザの行動解析やコンテキスト認識，そして，それらを支えるプラットフォームの研究が盛んに行われている [1][2]．我々も，個人レベルの長期にわたるセンシングに注目し，研究に取り組んでいる [3][4][5]．

携帯端末の記憶資源には限りがあるため，バックエンドにあるデータベースは，携帯端末から送られてくるログを喪失しないように遅延なく書き込むことを他の処理より優先すべきである．そして，ライフログを持続的に蓄え，処理し続けるためにデータ量の増加に対するスケーラビリティを確保する必要がある．さらに，携帯端末からのログのアップロードは，センサネットワークのように周期的に起こるとは限らない．ユーザの状況によって，データベースに送られてくるデータ量，タイミングはともに不規則である．結果として，ライフログのアップロード要求が瞬間的にデータベースの処理能力を大きく上回り，データベースが機能不全に陥る恐れがある．したがって，継続的なライフログセンシングを行う上で，瞬間的な過負荷にも耐え

[1] 立命館大学
Ritsumeikan University
[a)] myotaka@ubi.cs.ritsumei.ac.jp
[b)] ats@ubi.cs.ritsumei.ac.jp
[c)] takuya@cs.ritsumei.ac.jp
[d)] nishio@cs.ritsumei.ac.jp

られる必要がある．

そこで，本研究ではライフログデータの蓄積に最適化されたデータストアである $L^2$base を提案する．$L^2$base は一般的な CRUD 処理とともに，後述する特別な特徴を備えたログ挿入専用の API を提供することで上記の大量のレコードを含むログの蓄積と瞬間的な過負荷という問題を解決する．これにより，より少ない計算機資源でのライフログセンシングを可能とする．

### 1.2 本稿の構成

本稿は 6 節で構成される．以下，2 節でライフログセンシングの現状と課題を，3 節で関連研究を紹介し，それらを踏まえたうえで $L^2$base の設計について 4 節で述べる．4 節の設計を実際に実装し，それを用いた評価を 5 節に示す．最後に 6 節で本稿をまとめる．

## 2. 現状と課題

### 2.1 ライフログセンシングとそれを支えるプラットフォームの現状

ライフログセンシングを行うためのプラットフォームとして，義久ら [6] は開発者によって記述された内容に従って，携帯端末側とサーバ側の両方でセンサデータを解析し，収集するセンサデータ処理収集フレームワークを提案している．新垣ら [7] はライフログの収集端末として Android OS を搭載した携帯電話を用いて位置情報を取得し，大規模なログの保存と解析のために Hadoop[8] と HBase を利用したライフログシステムを試作している．

我々も，個人レベルの長期にわたるセンシングに注目し，研究に取り組んでおり，ライフログデータ処理のためのフレームワーク [9]，携帯端末–サーバ間の省電力なデータ転送機構 [10] などを提案してきた．

### 2.2 ライフログセンシングを支えるデータベースにおける課題

ライフログセンシングにおけるプラットフォームにおいて最も重要なことは，携帯端末から送られてくるログを喪失しないように遅延なくサーバ側のデータストアに書き込みを行うことである．ライフログセンシングによって得られるログは，センサデバイスからの 1 回の観測値だけでみると高々数ビットから数十バイト程度であるが，すべての観測値を蓄積しようとするとデータ量が膨大になる．携帯端末の記憶資源には限りがあるため，ログを喪失しないようにバックエンドのストレージに遅延なく書き込むことが他の処理より優先されるべきである．そして，ライフログを持続的に蓄え，処理し続けるためにデータ量の増加に対するスケーラビリティを確保する必要がある．さらに，ログがアップロードされるタイミングは，センサネットワークのように周期的に起こるとは限らない．結果として，ライフログのアップロード要求が瞬間的にデータベースの処理能力を大きく上回り，データベースが機能不全に陥る恐れがある．したがって，継続的なライフログセンシング行う上で，瞬間的な過負荷にも耐えられる必要がある．そこで，上記の要件を既存のデータベースが満たしているかを評価するため，予備実験を行った．

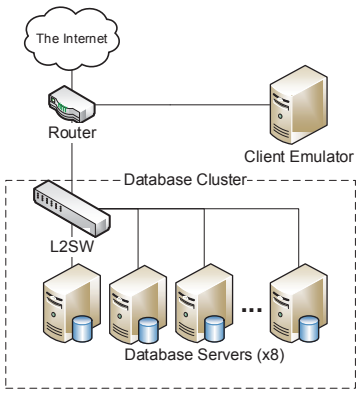
#### 2.2.1 予備実験

スケーラビリティを備えた既存のデータベースとして，分散関係データベース，P2P 型の分散 Key-value ストア，Master-slave 型の分散 Key-value ストアなどが挙げられる．現在では様々な分散データストアが公開されており，Master-Slave 型の分散 key-value ストアでは Bigtable[11], HBase などが，P2P 型の分散 key-value ストアでは Cassandra, Dynamo[12] などが，分散関係データベースでは VoltDB[13], MySQL Cluster などが有名である．予備実験には，これら分散データベースのなかでも安定して動作が可能で，カラムによるデータ管理が可能なものとして，Master-Slave 型の分散 key-value ストアからは HBase，P2P 型の分散 key-value ストアからは Cassandra，分散関係データベースからは MySQL Cluster を選び出し，評価対象とした．

上記の 3 つの分散データストアに対して実際のログを用いた性能評価を行い，ライフログセンシングに適したデータベースかを検証した．書き込み処理の性能評価として，1 日分のログ × 30 人分 (約 2.7GB・11271631 レコード) を挿入するのに要した時間を，クラスタ内の台数を 2 台，4 台，8 台と増やした場合について行った．

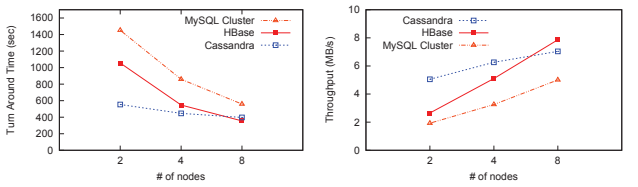
性能測定の実験に用いた環境を図 1 に示す．各機器の間は 1000BASE-T のケーブルで繋がれており，L2SW およびルータはギガビットイーサネットに対応したものを用いている．各ノードは 1 台の L2SW を中心としたスター型で構成されている．**図 2** に書き込み処理の評価結果を示す．

図 2 から分かるように，クラスタノードが 8 台のときのスループットだけをみると HBase，Cassandra，MySQL Cluster の順に性能が良いが，スケーラビリティの点では HBase, MySQL Cluster, Cassandra の順に良い結果となっている．Cassandra はデータの検索に分散ハッシュテーブル（Distributed hash table．以下，DHT）を用いているため，通常では範囲検索が出来ない．そのため，範囲検索が可能になるようにスキーマ設計を行ったことが，Cassandra の性能が台数の増加によって向上していない原因だと考えられる．MySQL Cluster と HBase については台数の増加に伴って性能も向上しており，スケーラビリティを備えてはいるが，スループットでみると数 MB/s 程度である．分析の結果，我々が取得しているライフログの特性としてバイト数あたりのレコード数が非常に多く，ログの挿入処理におけるロック処理，インデックスの更新，WAL (write ahead log, 先行書き込みログ) の生成・同期がボトルネッ




| Database & Emulator Servers | |
|---|---|
| CPU | Intel Core i7-2600K 3.4 GHz |
| RAM | 16 GB |
| OS | Ubuntu 10.04.1 Server |
| **Router & L2SW** | |
| Router | YAMAHA RTX1200 |
| L2SW | NETGEAR GS108 |

図 1　性能評価の測定環境




(a) 挿入に要した時間　　(b) 各データベース別スループット

図 2　書き込み処理の評価結果

クになっていることが分かった．

WAL とはデータベースの実際のテーブルデータに更新を反映するより前に，データベースへの変更処理の全てをログとして記録しておくものである．あるトランザクション処理を行っている最中にマシンがクラッシュした場合でも，WAL をチェックすることで，マシンがクラッシュ時に本来行われていなければならなかった処理が実際に完了したかが分かる．これにより，プログラムは処理の開始時までロールバックするか，中断された処理を再開して完了させるか，そのままの状態にしておくかを判断できる．

## 3. 関連研究

### 3.1 Logbase

ログはその性質上，時系列的に生成される．Logbase[14] はこの性質に着目し，WAL なしでの永続性を保証しつつ，蓄積処理の高速化を実現している．データに対して時系列順にインデックスを付ける場合，ログは一般的なデータと異なり，メモリ上でソートする必要がない．そこで，WAL を作成せずに直接データファイルをディスク上に生成することで，ディスク I/O を削減している．これにより，Logbase はオリジナルの HBase 比 2.5 倍の書き込み性能を実現している．

Logbase はメモリをバッファとして使用せずに，データファイルを直接ディスク上に生成している．これは，蓄積されるログのサイズがメモリ内で完結する場合であっても，ディスク I/O が発生してしまうという問題をはらんでいる．したがって，継続的なライフログセンシングを行う上で重要となってくる，瞬間的な過負荷という問題を解決できていない．

### 3.2 モバイル端末とクラウドコンピューティングを用いたセンサ情報蓄積手法の提案と設計

丹羽ら [15] はセンサ情報の密度に応じて動的なネットワークを構築し，負荷を分散させることが可能なクラウド内 P2P ネットワークを用いたセンサ情報蓄積手法を提案している．地理的範囲ごとに管理ノードとモバイル端末による P2P ネットワークを形成することで，目的とする地理的位置に関連付けられたセンサ情報を効率良く取得できる．丹羽らのシステムは位置情報を携帯端末から収集し，ゲリラ豪雨の検知や予報に活用するといった，短期的な情報のみを一時的に蓄積することを前提としている．この前提のもと，センサ情報をモバイル端末側で保持することで，ユーザ数の増加に比例して必要なストレージ容量の増加の問題を解決している．

しかし，我々が行っているライフログセンシングは数ヶ月，数年といった長期的なログの蓄積を前提としている．数 GB から数 TB のデータを各携帯端末で保持し続けるのは現実的ではない．したがって，我々が対象としているライフログセンシングにこのアプローチを適用することはできない．

### 3.3 MD-HBase

西村ら [16] はデータサイズに対するスケーラビリティを備えるとともに，多次元データに対する効率的な範囲検索を実現した MD-HBase を提案している．MD-HBase はそのインデックスに多次元インデックスの 1 つである空間分割ベースの Kd-Tree の空間分割法を用いている．これにより，一般的な手法と比較して無駄なインデックス走査を削減し，大規模な多次元データに対する効率的な範囲検索を実現している．空間充填曲線の 1 つである Z 曲線でインデックス付けしたオリジナルの HBase との比較では，書き込み処理のスループットが数パーセント程度向上した結果も出ている．しかし，ノード数によってはオリジナルの HBase よりも低い場合もあり，書き込み処理において MD-HBase の方が優れているとは一概には言えない．

MD-HBase と本研究は，大規模なセンサデータを効率よく管理するという目的においては同様である．しかし，我々はセンサからの観測値を喪失なく蓄積することに，MD-HBase は多次元データに対する効率的な範囲検索に主眼を置いている．したがって，本研究とは補完関係にはあ

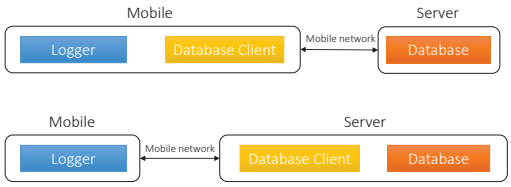



図 3　ライフログセンシングにおける各要素の構成パターン

るものの，競合するものではないと考えられる．

## 4. $L^2$base の設計

我々は，前節までに述べた諸課題を鑑み，ライフログデータの蓄積に最適化されたストレージシステムである $L^2$base を提案する．$L^2$base は瞬間的な過負荷と大量のレコードを含んだログの書き込みという問題に対して，$L^2$base クライアントとの連携による WAL の削減，排他制御の削減とインデックスの追記型更新および携帯端末からのアップロード単位での WAL の同期という 3 つの手法を用いることで解決する．なお，本節における従来のデータベースは HBase を想定している．

### 4.1 想定環境

ライフログセンシングを行う上で必要な要素は，実際にセンシングを行う Logger，サーバ側のデータベースにログを挿入する Database Client，そしてログを蓄積する Database の 3 つに大きく分けられる．これら各要素の構成は図 **3** に示した 2 つのパターンが考えられる．

両者の違いは Database Client が携帯端末側で動作するか，サーバ側で動作するかである．前者におけるメリットとしては，Database Client をサーバ側から排除することで，Database がサーバの計算機資源を十分に活用できることである．しかし，この構成における問題点は Database Client と Database の間が携帯電話回線でつながっていることである．現状，携帯電話回線は接続性が非常に不安定であるとともに，通信速度が非常に低速である．したがって，Database 側でエラーが起きた場合，リトライするごとに携帯電話回線を介した通信を行う必要がある．後者の場合，Database Client と Database が同居する関係上，計算機資源を共有する必要はあるが，上記の問題は発生しない．したがって，Database Client がサーバ側で動作する構成が望ましい．以降は Database Client はサーバ側で動作している構成を前提とする．

### 4.2 $L^2$base クライアントとの連携による WAL の削減

従来のログの挿入までの流れ（図 **4**）は，まず Logger がセンシングし，そのデータをローカルなディスクに永続化する．次に，一定のタイミングでログを Database Client に送信し，Database Client は受け取ったログをデータベースへ挿入する．このとき，データベースは WAL をディスクに書き込むとともにログをオンメモリのデータ構造に挿入する．Database Client はデータベースへの書き込みが完了すると Logger へ挿入完了を通知し，それを受け取った Logger は永続化が完了したログを削除する．メモリ上のデータが一定量を超過するとディスクに書き出され，書き込みが完了した範囲の WAL が削除される．この場合，Logger のローカルなディスクとデータベースの WAL の 2 つを用いてログの永続性を保証しているが，前述の通り WAL を用いる事でスループットは劇的に低下する．

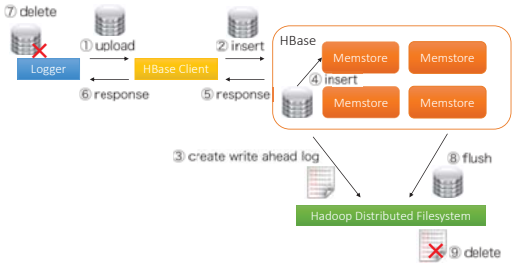


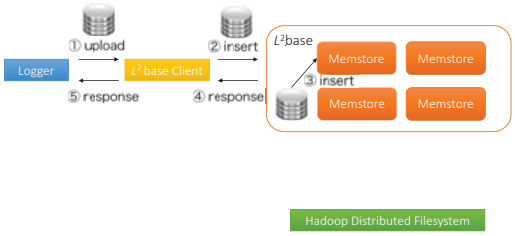

図 4　現状のライフログ蓄積の流れ

そこで，$L^2$base クライアントが携帯端末内のログを削除するタイミングを制御することで，WAL なしでの永続性を保証する．WAL を削減することで一時的にログの蓄積を高速化し，瞬間的な過負荷に対応する．図 **5** に処理の詳細を示す．ログのアップロードに関しては現状のライフログ蓄積と同様であるが，WAL の書き出しは行わずメモリ上にログが保存された段階で Logger にレスポンスを返す．Logger は蓄積されたことをレスポンスとして受け取るが，端末内の送信済みログは削除しない (図 **5(a)**)．そして，$L^2$base のメモリ上に保持されているログは，一定量を超過するとディスクに書き出される．この後，再び Logger からログがアップロードされるタイミングで，ログがディスクに書き込まれたことを $L^2$base クライアントが Logger に通知する．Logger は $L^2$base 側でディスクへの蓄積が完了したことを確認すると，端末内の蓄積済みログを削除する．以上の手順を踏むことで，WAL なしでログの永続性の保証する (図 **5(b)**)．

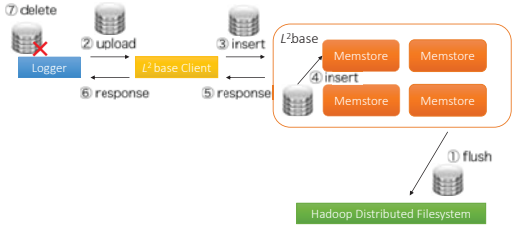
上記の手法を用いることで WAL の書き込みを削減し，ログの蓄積を高速化する．しかし，携帯端末の記憶資源は乏しいため，長期間に渡って携帯端末側でログを保持することはユーザの負担になる．さらに，メモリ内のデータをディスクに書き込む前にデータベースがクラッシュした場合，喪失したデータを携帯端末が再送しなければならないといったデメリットも存在する．したがって，データベース側の負荷に応じて WAL の有無を切り替えることが肝要だと考えられる．

### 4.3 排他制御の削減およびインデックスの追記型更新によるオンメモリ処理の最適化

瞬間的な過負荷について，アップロードされるログのサ




(a) WAL なしでのログ書き込み




(b) 永続化が完了したログの削除

**図 5** $L^2$base クライアントとの連携による WAL なしでの永続性の保証

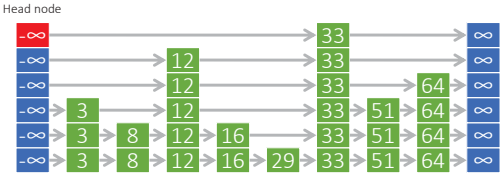



**図 6** skip list の例

イズがメモリ内で完結する場合，オンメモリ処理の高速化が重要となってくる．したがって，瞬間的な過負荷に対応するには，ディスク I/O だけなく，オンメモリ処理も最適化する必要がある．$L^2$base では排他制御を削減し，インデックスの追記型更新を実現することでオンメモリ処理を高速化している．

#### 4.3.1 排他制御の削減

複数ユーザで 1 つのテーブルを共有する場合，複数ユーザによる同時書き込み処理が行われる可能性があるため，データの一貫性を保証するために排他制御を行う必要がある．ユーザが増加するにつれて，ロック解放待ちが増加し，書き込み処理の低下につながる．そこで，各ユーザごとにログテーブルを用意することで，複数ユーザにおける排他制御を省く．さらに，ライフログの特性上，ログの更新と削除はログテーブル内で発生しないので，ログテーブル内についても排他制御を省くことができる．

#### 4.3.2 インデックスの追記型更新

データベースはその内部に蓄積されているデータに対する処理を高速に行うための仕組みとして，インデックスという機能を提供している．HBase や Cassandra などのデータベースでは，このインデックスのデータ構造に skip list（**図 6**）を用いている．

skip list では各データはキーの自然順序付けに従ってソートされており，挿入，削除，更新の計算量は $O(\log n)$

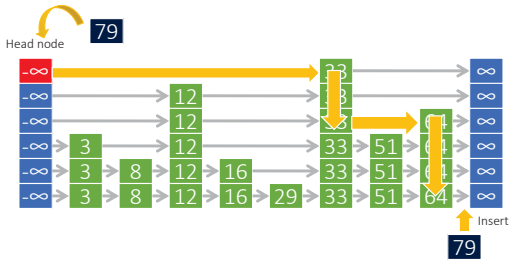



**図 7** skip list へのデータ挿入

である．skip list では挿入位置を決定するため，Head node から順に最下位にある挿入位置のノードまでを辿っていく．**図 7** には新たに”79”という値を挿入する場合についての手順を示している．まず，Head node から見て右側にあるノードは”33”であるため，”79”はこれよりも右側にあることになる．”33”の右側にはノードはないため，1 つ下のレベルに降りる．これを繰り返すことで最下位のノードまで辿っていき，最終的に挿入位置を決定する．

1 つのレコードを挿入するのに必要な計算量は $O(\log n)$ であるため，レコード数が N のライフログを挿入する場合は，それに要する時間は N 倍となる．しかし，実際のセンシングデータでは前のデータを書き換える必要はなく，時系列的に後に追加していくだけである．さらに，キーを UNIX 時間としたとき，ライフログセンシングでは新たに送られてくるデータは既にデータベース内のオンメモリのデータ構造に保持されているライフログの末尾に常に挿入されることになる．そこで，常に skip list の最後尾の位置を保持しておくことで，1 レコードを挿入するたびに発生していた挿入位置の検索を削減する．これにより高速にデータを挿入することができる．

上記 2 つのオンメモリ処理の最適化手法と $L^2$base クライアントとの連携による WAL なしでの永続化手法を活用することで，瞬間的な過負荷という問題を解決する．

### 4.4 WAL の携帯端末からのアップロード単位での同期

ライフログセンシングでは各端末は多くのレコード数を含んだログを生成する．センシング間隔や携帯端末の存在する環境にもよるが，携帯端末は 1 時間あたり，数千から数万レコードのログを生成する．

また，分散データベースでは可用性を高めるために，この WAL を複数のノードに複製している（**図 8**）．近年，注目を浴びている NoSQL の多くは，この WAL の同期を数レコード単位で行っている．しかし，我々が対象としているデータはレコード数が多く，RPC のオーバヘッドがレコード数分発生するため，蓄積処理に要する時間が劇的に増大する．

そこで，携帯端末から 1 回のアップロードで送られてくる単位で同期することにより，WAL を同期する際に生じるオーバーヘッドを削減する．$L^2$base はアプリケーショ

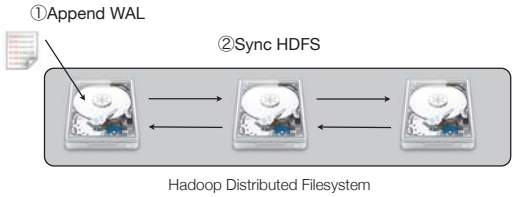
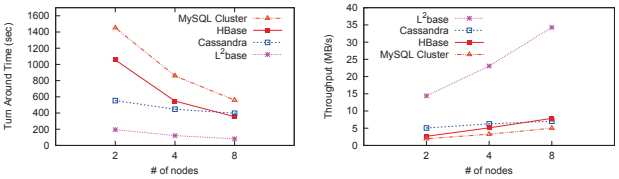



図 8　WAL の生成と複数ノードへの同期




(a) 挿入に要した時間　(b) スループット

図 9　各ノード数における挿入に要した時間とスループット

ンに対して，ログの挿入専用の API を提供し，1 回の API 呼び出しに際して送られてきた単位で WAL の同期を行う．つまり，図 8 における 2 ステップ目である WAL の同期を数千，数万レコードという非常に大きな単位で行う．これにより同期するたびに発生していた，RPC と複製の同期待ちを大幅に削減する．しかし，書き込み処理が頻繁に失敗するような環境では，1 レコードごとに同期を行っていた場合と比較して，リトライのコストが高くなる．

なお，本手法の書き込み性能は携帯端末から 1 回にアップロードされるデータの量に依存する．アップロードの間隔について，解析結果の鮮度と携帯端末の消費電力はトレードオフの関係である．本稿では，解析結果の鮮度と携帯端末の消費電力とのバランスを考え，ログは携帯端末によって 1 時間に 1 回程度の間隔でアップロードされるものとして扱う．

## 5. 実装と評価

本節では，前節で述べた設計を実装し，それを用いて提案手法の書き込み性能を評価する．性能測定の実験に用いた環境は予備実験と同様である（図 1）．なお，本研究の提案手法の 1 つである，「$L^2$base クライアントとの連携による WAL の削減」の貢献は，WAL なしでの永続性を保証することである．その有効性は，WAL を用いないことを前提とし，その上で高速化を実現している「排他制御の削減およびインデックスの追記型更新」を評価することで同時に示すことができるため，評価項目には含まれていない．

### 5.1 実装環境

$L^2$base を実装するデータベースとして，スケーラブルなデータベースには様々なものがあるが，パフォーマンス，安定性および開発のしやすさと 2.2.1 節の予備実験の結果を踏まえ，HBase を拡張する形で実装した．$L^2$base を実装する HBase には，バージョン 0.94.0 を用いている．$L^2$base の下層の分散ファイルシステムには，Hadoop Distributed Filesystem のバージョン 1.0.2 を用いた．

### 5.2 HBase を用いた $L^2$base の実装

HBase はその LOC が 633,922 と，比較的大規模なソフトウェアである [*1]．したがって，パフォーマンスを考慮しつつも，出来る限り少ない変更で $L^2$base を実現するという方針で HBase を修正し，実装した．なお，$L^2$base を実装する上で変更した主なクラスは，HRegionServer クラス，HRegion クラス，Memstore クラス，Store クラスである．

[*1] http://hbase.apache.org/source-repository.html

### 5.3 WAL の携帯端末からのアップロード単位での同期

WAL の生成・同期の単位を携帯端末からのアップロード単位で行うことよる高速化を検証するために，1 日分のログ × 30 人分を挿入するのに要した時間を測定した．さらに，各データベースのスケーラビリティを評価するため，データベースノードの台数を 2 台，4 台，8 台と増やした場合について行った．評価結果として，各ノード数における挿入に要した時間とそのスループットをそれぞれ**図 9** に示す．

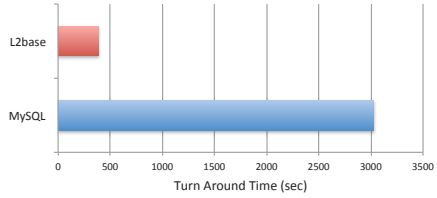
評価結果より $L^2$base は他のデータベースと比較してノード数が 2，4，8 台の時，それぞれ平均 5.25，5.10，5.36 倍高速であることが分かった．ノードが 2 台から 8 台の間での平均は 5.23 倍である．この結果より，ライフログのようなデータサイズあたりのレコード数が多いデータについては，WAL の生成・同期の単位が書き込み性能に大きく影響していることが分かった．

関連研究との比較として，評価環境，評価に用いたデータなどが異なるため一概に比較はできないが，3.1 節の Logbase はオリジナルの HBase 比約 2.5 倍高速であると評価結果において示している．したがって，本評価では HBase 比約 4.7 倍の性能を示している $L^2$base の方が Logbase よりも高速であると考えられる．

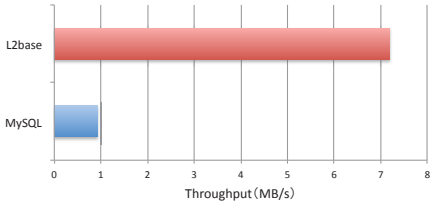
#### 5.3.1 MySQL との単体性能比較

MySQL は世界でもっとも普及しているオープンソースデータベースであり，多くのシステムで用いられている．これまでの経験から，ライフログの蓄積にも MySQL を用いたいという要求もあると考えられる．そこで，MySQL から $L^2$base へ移行するためのメリットを示すために，5.3 節と同様に 1 日分のログ × 30 人分 (約 2.7GB) を挿入するのに要した時間を測定した．比較において不公平がないように，MySQL と同様に $L^2$base のノード数も 1 台とした．MySQL のストレージエンジンには並列書き込みの性能が高い InnoDB を採用した．評価結果として，挿入に要した時間とそのスループットを**図 10** に示す．

評価結果より $L^2$base は MySQL と比較して 7.77 倍高速




(a) 挿入に要した時間




(b) スループット

**図 10** MySQL との単体性能比較

であることが分かった．この結果は $L^2$base がその下層の分散ファイルシステムである HDFS に 3 つの複製を作成しつつ発揮した性能であるため，実質的な差は非常に大きいと考えられる．さらに，$L^2$base は接続されたデータベースノードの台数を増やすことで線形に性能を向上することができる．したがって，性能，信頼性およびスケーラビリティにおいて $L^2$base が非常に優れていることが分かる．

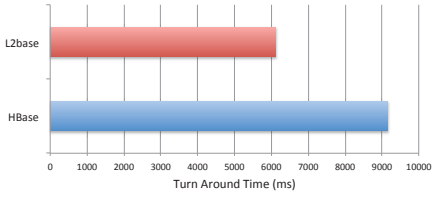
しかし，既存の SQL に慣れた開発者からすると，たとえ $L^2$base に上記の優位性があったとしてもインターフェースが異なることは非常に深刻な問題である．これについては Hive[17] や Cloudera Impala[18] などを用いることで，$L^2$base がユーザに SQL インターフェースを提供することができる．

#### 5.3.2 携帯端末からのアップロード単位での WAL の同期によるリトライコスト

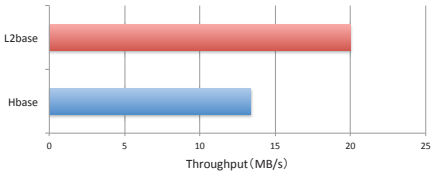
$L^2$base は WAL の生成・同期の単位を携帯端末からの 1 回のアップロードという非常に大きい単位で行うことにより，書き込み性能の向上を実現している．しかし，同期の単位を大きくすることにより，書き込み処理が失敗した場合の再試行にかかるコストが増大する．1 回の書き込みに要する時間を $t_w$，書き込みに失敗した場合に書き込みが成功するまでに要する時間を $t_f$，書き込みに成功する確率を $p$，書き込みを行う回数を $n$ としたとき，書き込み処理に要する時間 $t$ は以下の式で表される．

$$t = (t_w \cdot p + t_f(1-p)) \cdot n \qquad (1)$$

$p$ 以外の項目は測定可能であるため，$L^2$base と HBase について方程式を作ることで，$L^2$base と HBase の性能が同値になる確率を導き出せる．仮に $L^2$base と HBase の性能差を 5 倍，$L^2$base における $t_f$ を $5t_w$，HBase における $t_f$ を $t_w$ とした場合，$p = 1/5$ となる．つまり，5 回に 1 回の割合で書き込みエラーが発生する環境では書き込み処理の高速化に寄与しない．しかし，実際には書き込みエラーが起きる確率は非常に低い値であるため，携帯端末からのアップロード単位での WAL の同期による書き込み処理の高速化は有効であると考えられる．




(a) 挿入に要した時間




(b) スループット

**図 11** オンメモリ処理の最適化の評価結果

### 5.4 排他制御の削減およびインデックスの追記型更新

排他制御の削減およびインデックスの追記型更新による書き込み処理の高速化を検証するために，1 時間分のログ × 30 人分 (約 150MB) を挿入するのに要した時間を測定した．本評価実験は測定結果にオンメモリ処理以外の要素が含まれないよう，HBase，$L^2$base 両方のデータベースノードを 1 台にして評価実験を行った．**図 11** に挿入に要した時間とそのスループットを示す．

評価結果より $L^2$base はオリジナルの HBase と比較して 1.50 倍高速であることが分かった．さらに，スループットで比較すると，WAL を用いた場合と比べて 3 倍高速である．これにより，$L^2$base クライアントと連携することで瞬間的には 3 倍のトラフィックにも耐えることができると考えられる．

## 6. おわりに

### 6.1 まとめ

本稿では，ライフログデータの蓄積に最適化したストレージシステムである $L^2$base を提案した．3 種類のデータベースを用いた予備実験と我々の数年に及ぶライフログセンシングの経験と分析から，ライフログデータの蓄積における課題として，大量のレコードを含むログの蓄積と瞬間的な過負荷の 2 点にまとめた．$L^2$base は上記の課題に対して，$L^2$base クライアントとの連携による WAL の削減，携帯端末からのアップロード単位での WAL の同期，排他制御の削減およびインデックスの追記型更新という，3 つの特徴を備えたログの挿入専用の API を提供することに

よって解決した．HBase の拡張によるプロトタイプ実装を用い，$L^2$base の性能を評価した．WAL を有効にしたときの $L^2$base は，ノード数が 2 から 8 台の時で，オリジナルの HBase よりも平均 5.23 倍高速であることを確認した．さらに，MySQL との単体性能比較ではストレージエンジンに InnoDB を指定した MySQL よりも 7.77 倍高速であった．WAL の同期を行う単位の変更によるリトライコストの増加に関して，計算上では 5 回に 1 回以上の割合で書き込みエラーが発生する環境以外では高速化が有効であることを確認した．排他制御の削減およびインデックスの追記型更新によるオンメモリ処理の最適化によってオリジナルの HBase と比較して 1.50 倍処理速度が向上したことが確認できた．これにより，$L^2$base クライアントと連携し，携帯端末内のログを削除するタイミングを制御することで瞬間的には通常時の 3 倍のトラフィックにも耐えられることを示した．

### 6.2 今後の課題と展望

本稿では，WAL が有効である場合と無効である場合の両方において，最適化手法を提案している．しかし，WAL を無効にした場合では携帯端末側にかかる負担が大きくなる．したがって，データベース側の負荷に応じて WAL の有無を切り替えることが肝要である．これを実現するためには，$L^2$base の内部に負荷を監視する機構を実装し，現在の負荷状態を適切に判断する必要がある．

### 参考文献

[1] Kawaguchi, N., Ogawa, N., Iwasaki, Y., Kaji, K., Terada, T., Murao, K., Inoue, S., Kawahara, Y., Sumi, Y. and Nishio, N.: HASC Challenge: gathering large scale human activity corpus for the real-world activity understandings, *Proceedings of the 2nd Augmented Human International Conference*, ACM, p. 27 (2011).

[2] Khan, W. Z., Xiang, Y., Aalsalem, M. Y. and Arshad, Q.: Mobile Phone Sensing Systems: A Survey, *Communications Surveys Tutorials, IEEE*, Vol. 15, No. 1, pp. 402 –427 (2013).

[3] 里中裕輔，西尾信彦：対話型移動支援システムの設計，情報処理学会研究報告. UBI, [ユビキタスコンピューティングシステム]， Vol. 2012, No. 38, pp. 1–5 (2012).

[4] 西尾信彦，藤井陽光，安積卓也：複数センサーを用いた屋内細粒度行動認識の自動化，情報処理学会研究報告. UBI, [ユビキタスコンピューティングシステム]， Vol. 2012, No. 38, pp. 1–6 (2012).

[5] NikenTriMahayani, Kitazawa, T. and Nishio, N.: Mining Life-log Sensing Data to Extract User's Significant Locations and Movement, 信学技報，Vol. 112, pp. 143–147 (2012).

[6] 義久智樹，西尾章治郎：スマートフォンを用いたセンサデータ処理収集のためのフレームワーク，第 4 回データ工学と情報マネジメントに関するフォーラム (DEIM 2011) 論文集 (2012).

[7] 新垣智規，城間政司，長田智和，谷口祐治，玉城史朗：Android 端末を用いた Hadoop ベースのライフログシステムの試作，情報処理学会研究報告. UBI, [ユビキタスコンピューティングシステム]，Vol. 2011, No. 5, pp. 1–4（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110008682490/⟩ (2011).

[8] Shvachko, K., Kuang, H., Radia, S. and Chansler, R.: The hadoop distributed file system, *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, IEEE, pp. 1–10 (2010).

[9] 名生貴昭，松井智紀，榎堀　優，西尾信彦：Dwarfstar: ライフログデータ処理のためのフレームワーク，電子情報通信学会論文誌. D, 情報・システム， Vol. 96, No. 5 (2013,accepted).

[10] 米田圭佑，里中裕輔，西尾信彦：センシングモバイルにおける個人特化された省電力機構，情報処理学会研究報告. UBI, [ユビキタスコンピューティングシステム]， Vol. 2012, No. 19, pp. 1–6（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009478252/⟩ (2012).

[11] Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A. and Gruber, R.: Bigtable: A Distributed Storage System for Structured Data, *Proc. OSDI*, pp. 205–218 (2006).

[12] Hastorun, D., Jampani, M., Kakulapati, G., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W.: Dynamo: amazon's highly available key-value store, *In Proc. SOSP*, Citeseer (2007).

[13] VoltDB Inc: VoltDB, VoltDB Inc (accessed 2011-06-22).

[14] Vo, H. T., Wang, S., Agrawal, D., Chen, G. and Ooi, B. C.: LogBase: a scalable log-structured database system in the cloud, *Proc. VLDB Endow.*, Vol. 5, No. 10, pp. 1004–1015 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2336664.2336673⟩ (2012).

[15] 丹羽絢也，岡田和也，奥田　剛，門林雄基，山口　英：モバイル端末とクラウドコンピューティングを用いたセンサ情報蓄積手法の提案と設計，情報処理学会研究報告. UBI, [ユビキタスコンピューティングシステム]， Vol. 2012, No. 10, pp. 1–7（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009422523/⟩ (2012).

[16] Nishimura, S., Das, S., Agrawal, D. and Abbadi, A.: MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services, *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*, Vol. 1, pp. 7 –16 (online), DOI: 10.1109/MDM.2011.41 (2011).

[17] Thusoo, A., Sarma, J., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P. and Murthy, R.: Hive: a warehousing solution over a map-reduce framework, *Proceedings of the VLDB Endowment*, Vol. 2, No. 2, pp. 1626–1629 (2009).

[18] Cloudera, Inc: Cloudra Impala, Cloudera, Inc (accessed 2011-06-22).