Vol.2017-DBS-165 No.29
Vol.2017-IFAT-128 No.29
2017/9/20

高信頼性とスケーラビリティを備えた 分散システムアーキテクチャの提案

藤澤宏明
北陸先端科学技術大学院大学
情報科学研究科

鈴木正人
北陸先端科学技術大学院大学
先端科学技術研究科

概要：分散アプリケーションにおける RDB を用いたキューDB（非同期でキューイング処理を行うデータベース）に関して（1）キューDB が SPOF(Single Point of Failure)であり，停止時はシステム全体が停止する（2）キューDB がボトルネックとなり，アプリケーションサーバを追加しても性能が向上しにくい（3）障害発生時やシステムの高負荷時に業務 DB と相互に影響が出る，といった問題が指摘されている．
本稿ではこれらの問題を解決することを目的とし，KVS を用いた新しいアーキテクチャを提案し，実験により有効性を確認する．非同期でキューイング処理を行い，データを配信するサービスを例として，RDB に基づく従来のアーキテクチャと比較する．比較は（1）スループット，及びスケーラビリティ（2）障害検知とそれからの復旧（3）DB 停止時の性能劣化測定を行った．結果として，提案するアーキテクチャはスケーラビリティと可用性に優れており，今後のシステム構築において，KVS を用いたアーキテクチャを利用する見込みが立った．

キーワード：KVS，メッセージキューイング，スループット，スケーラビリティ，SPOF

1. はじめに

DB を含む分散システム（参考文献[1][2]）では，利用するユーザの業務処理と非同期に行うデータ処理の重要性は日々高まっており，処理量や提供するサービスが年々増加する中，高い信頼性とスケーラビリティが求められている．非同期処理の要求に対する応答はユーザに基本的に返らないため，要求を受け付けた後はシステムの障害発生時もその要求を失わずに実行する必要がある．この機能を実現するため，現行の仕組みでは非同期処理の要求をキューイングし（参考文献[3]），RDB(Relational Data Base 参考文献[4])を用いてデータの一貫性や処理の正当性を保証している（キューDB）．

この RDB を用いたキューDB は，全てのアプリケーションで共通して同一のデータベースを使用する構成が多い．アプリケーションサーバは複数を冗長化した構成のため，一部で故障が発生してもユーザにサービスを提供し続けられる．しかしキューDB は 1 か所でのみ稼働するため，もし停止した場合は全てのアプリケーションが使用できなくなる SPOF(単一障害点．Single Point of Failure)である．加えてユーザ数やデータ量の増加に伴い，システムへの要求数が増加すると，キューDB に対する要求数も増加する．キューDB に負荷が集中し処理が遅延すると，システム全体が過剰に遅延し，ユーザに満足のいく応答時間を提供できなくなる．この状態でアプリケーションサーバを追加してもシステム全体のスループットが向上しない．この他，特に中小規模のシステムでは業務用に使用しているデータベース（業務 DB）とキューDB との相互影響が問題となる．これは作業負担と費用を減らすよう，簡易的に業務 DB と同一のハードウェア（HW）上にキューDB を構築されるためである．結果，障害発生時は論理的な独立性が失われシステム全体が停止し，システムの高負荷時にはスループットが低下する．

本稿の提案は RDB に代わり KVS(Key-Value Store 参考文献[5])をキューDB に用いて，上記の問題解決を試みる点に独自性がある．キューDB を SPOF とせず，業務 DB との相互影響をなくし，処理要求数に応じたスケーラビリティを実現することで，ユーザはサービスを常に同じように享受でき，開発者は性能要件（非機能要件）に応じた柔軟なシステムの構築が可能になる．本稿の目的は，KVS を用いたアーキテクチャ（KVS 型）が利用可能であり，RDB を用いたアーキテクチャ（RDB 型）より有効なことを示すことである．

RDB 型も KVS 型も，非同期処理の要求をキューイングしてデータを永続化するためにデータベースを利用する．RDB 型は一つのデータベースを全てのアプリケーションが共通で利用する．KVS 型は各アプリケーションが別々に，各アプリケーションと同一の HW 上に存在するデータベースを利用するため，以下の点が有効になると予想される．

- 障害発生時の影響範囲が限定的あり，システム全体が停止せず，ユーザは継続してシステムを利用可能．
- システムのボトルネックが発生しにくく，アプリケーションサーバ数に比例して処理性能が向上する．

本稿では下記を定義している．

（1）分散システム
- 複数のコンピュータの協調した動作を行う．
- 複数のコンピュータが別々にデータの取扱可能．
- 論理的及び物理的に異なる構成が可能．

（2）中央集約型システム
- 複数のコンピュータが協調した動作を行わない．
- 1 か所でデータを取り扱い，集中して処理を行う．
- 論理的及び物理的に異なる構成が困難．

この定義から RDB 型は中央集約型，KVS 型は分散システムと捉え，それぞれの特徴を考えることができる．

Vol.2017-DBS-165 No.29
Vol.2017-IFAT-128 No.29
2017/9/20

2. 分散データベース調査

企業向けシステムのデータベース使用例を記載する．

- 商品の受発注データの作成および参照
- 請求処理，日次や月次での集計処理
- 物流データ(在庫，倉庫)との連携
- 売上データの管理，POS システムとの連携

いずれもデータの重要度が非常に高く業務と密着している．データの消失がないよう，冗長化及び分散した配置が必要である．またシステム停止は業務停止または圧倒的な作業効率の低下となるため，可用性も求められる．

さらに日々データ量は増大し，データの一貫性を保つため集中的に一つのデータベースで処理を行うことが多く，相対的にデータベースの処理時間が大きくなりボトルネックとなりやすい．このため，システムに対する要求数やデータ量が増加した場合の性能も求められる．

これらの要求に対応する RDB の代表例をあげる．（参考文献[5][6][7]）．

- MySQL のマスタ/スレーブを利用したレプリケーション
- Oracle の RAC
- クラスタリングソフトウェアを用いた冗長構成

同様に KVS の代表例を記載する(参考文献[8][9][10])．

KVS 型（Oracle NoSQL）

Master と Replica の 2 種類のノードで構成され，全てのノードは同じデータを持つ．アプリケーションはいずれかのノードにアクセスして処理を行うが，更新処理は Master でのみ行われる．参照処理は各ノードが応答を返すため負荷が分散されるが，データの一貫性を優先しているため，データの更新は常に Master が先，Replica が後となり，競合が発生しやすく，Master と Replica 間でデータの不整合が発生する時間帯も存在する．

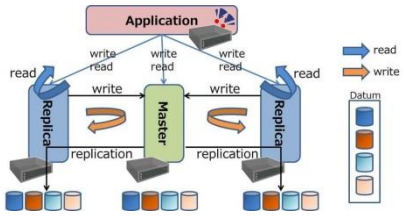

図 2-1: Oracle NoSQL

KVS 型（HBase）

各ノード（Region）はデータの参照及び更新を同じように行うが，データの配置先は Master が割り当てる．このためアプリケーションは，必ず接続先を確認し（Zookeeper に接続し該当ノードの確認を行う），その後に実際の処理を行う．データの保存先は HDFS(Hadoop Distributed File System)で分散して保存され，HDFS を利用した他のアプリケーション（Hadoop, Pig, Hive 等）と連携しやすい．一方，システムの信頼性を高めるためには Zookeeper, Master の冗長化が必要となる．

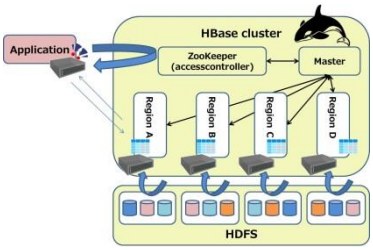

図 2-2: HBase

KVS 型（Cassandra）

各ノードの役割は均等であり，アプリケーションはいずれかのノードにアクセスして処理を行う．データは各ノードに分散して配置される（データのキー情報をハッシュ化し，複数のノードに冗長的に配置される）．データの更新時は更新したデータの情報が非同期で伝播するため，各ノード間でデータの整合性が失われる時間帯が大きくなるものの，特定のボトルネックがなく，スケーラビリティが向上するアーキテクチャとなっている．

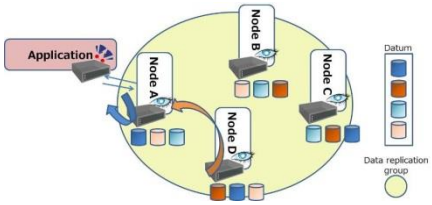

図 2-3: Cassandra

次に RDB と KVS の比較を表 2-1 に記載する．RDB は中央集約型であり，障害発生時の影響が大きく，性能ボトルネックとなりやすいが，KVS は分散してデータを配置し信頼性を向上させ，要求数やデータ量に応じたスケーラビリティを実現可能なアーキテクチャである．

表 2-1 RDB と KVS の比較

point	RDB	KVS
data format	テーブル形式 各値を個別に定義可能	キーとペアになる値 値は大きく一括りに取り扱われ、詳細定義は各KVSで異なる
data access	SQLを用いたデータ加工、集計、条件検索に優れる	キーアクセスが基本 データの格納に優れる
policy	データを一元管理	データを分散して管理

KVS の比較を表 2-2 に記載する．Oracle NoSQL は Master ノード，HBase は Master 及び Zookeeer が中央集約型であり SPOF でスケール性に乏しい．対して Cassandra は SPOF がなく，スケーラビリティに優れている．他の KVS より効果的なアーキテクチャを実現できると考えられるため，Cassandra を採用して，KVS 型の開発を行った．

表 2-2 KVS 型の比較

point	Oracle No SQL	HBase	Cassandra
data access	各ノードに直接アクセス	マスタ管理 Zookeeper経由	各ノードに直接アクセス
data update	マスタノードのみ	各ノードで更新	各ノードで更新
storage	各ノードのローカル	共通HDFS	各ノードのローカル

3. アーキテクチャ

一般的なシステムの達成目標を下記にあげる．

- 障害発生時の影響範囲を最小化
- メンテナンス時間の最小化
- ユーザに対する応答時間の最小化

いずれもユーザがストレスなくシステム利用することを目的としている．機能要求や非機能要求に応えるため，上記を踏まえてアーキテクチャを決定する．

本稿では，サーバ構成とソフトウェア構成及びデータの冗長化をアーキテクチャの対象としており，システムを利用するユーザ側の処理や接続方式，及びネットワーク環境と通信特性は対象外としている．

RDB 型と KVS 型のアーキテクチャをそれぞれ記載する．

3.1 RDB 型アーキテクチャ

キューDB に RDB を用いたアーキテクチャである．

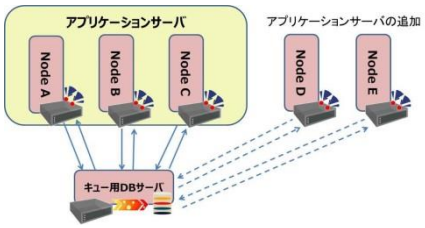

図 3-1: RDB 型アーキテクチャ

RDB 型は中央集約型であり，下記の問題がある．

- キューDB が停止するとシステム全体が停止する．
- 要求数が多い場合，キューDB への負荷が集中し，キューDB の性能が上限となり，アプリケーションサーバを追加してもスループットが向上しない．
- 業務 DB と同一の HW にキューDB を構築すると，障害発生時の利用不能及び高負荷時の性能等，相互影響が発生する．

3.2 KVS 型アーキテクチャ

RDB 型における問題点を解決するため，キューDB に KVS 型のデータベースを使用したアーキテクチャを提案する．アプリケーションサーバ上と同一環境にキューDB を構築することで，分散してデータを利用する点が特徴となる．

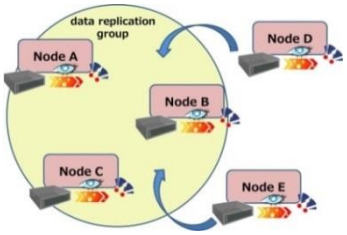

図 3-2: KVS 型アーキテクチャ

RDB 型の問題点に対し，下記の改善が期待できる．

- アプリケーションサーバと同じ粒度で分散処理を行う，SPOF がない構成
- 局所的な負荷が発生せず，理想的にはアプリケーションサーバ数に比例したスループット向上
- 業務 DB と異なる HW 上に構築されるため，性能や障害発生時の相互影響なし

3.3 実験対象システムの定義と構成

本稿では，株価アラート配信システムを対象として，RDB 型と KVS 型の両方を実装して実験を行い，KVS 型によって改善した効果を確認する．

株価アラート配信システムのユーザは株の取引を行っている．株価アラート配信システムの目的は，株式市場に変動があった場合，取引を判断するためのデータ提供を目的として，PUSH 型で各ユーザにデータを配信する事である．最終的にモバイル端末にデータが届くが，これは別システム（MDS: Message Delivery System）を利用している．株価アラート配信システムの対象はユーザに届くメッセージを作成し MDS に受け渡すまでである．ユーザは事前に，配信してほしい情報を登録する．この情報から，株価アラート配信システムが各ユーザ別に異なるデータを作成する．

機能要求を下記に記載する．

- モバイル端末にデータを配信すること．
- 各ユーザは欲しいデータをあらかじめ登録する．このデータを株価配信アラート配信システムが株価アラート生成システムからデータを受信し，対象のユーザにメッセージを作成して MDS に配信する．
- 連続したデータは，受信順にデータを配信する．

性能要求を下記に記載する．

- 最大同時生成メッセージ数：3 万件
- メッセージ配信端末数：5 万台
- 許容配信時間：20 秒以内（図 3-3 ①～⑤が対象）

この性能要件は，取引開始時や緊急のビッグニュース発生時といった突発的に処理負荷が高騰する場合も満たす必要がある．いずれも平日，株式市場の取引時間帯が対象となる．一般的な設計目標は，上記の処理負荷が高騰する場合も性能劣化（配信時間の遅延）を 30%程度に抑え，性能劣化する時間幅は 1 時間以内である．

Vol.2017-DBS-165 No.29
Vol.2017-IFAT-128 No.29
2017/9/20

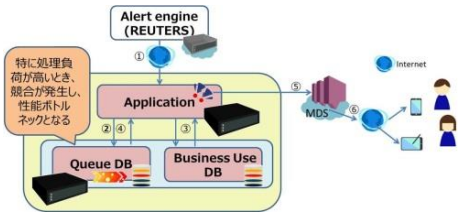

図 3-3：株価アラート配信システム構成

図 3-3 に示したシステムの動作シーケンスは以下となる．

① 株価アラート生成システムからデータ受信
② キューDB に受信データ格納
③ ユーザに配信するメッセージ作成
④ キューDB に配信するメッセージを格納
⑤ MDS に送信
⑥ MDS が各ユーザにアラートデータを配信

システムに対する処理要求数の増加は下記に影響する．

- ①の受信数(株価市場の変動が大きい)
- ⑤の配信数(ユーザ数, 配信データ数の増加)

3.4 モデル化

株価配信アラート配信システムのボトルネックや処理時間への影響を明確にし，スケーラビリティが向上する見込みを検討するため，論理的なモデル化を行った．データの提供元からデータを受信しユーザにメッセージを配信する機能のみを対象とし，配信対象人数の変化と性能への影響と有効性を確認可能にした．対象機能は下記となる．

- 事前に登録される画一的な情報を基に，各ユーザに対し別々にメッセージを配信する．
- 株価が変動し，登録済の閾値を超えた際に通知する．

下記を前提条件とし，モデルを図 3-4 に示す．

- メッセージはサイズに差がなく，性能に影響ない．
- メッセージは株価の変動監視対象データを受信した後，各ユーザのモバイル端末に届く内容が作成される．
- メッセージは何らかの要因による更新や喪失はない．
- メッセージはシステム内外の影響や相互干渉がない．

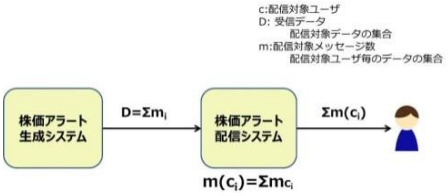

図 3-4：株価アラートシステム配信メッセージ数の関係

各値の説明は以下の通り．

- 配信対象ユーザ:c 各ユーザ:ci
- 配信対象ユーザごとに作成するメッセージ:m
- 各メッセージ:mi(m は c により異なる)
- 受信データの集合:D(m の集合)
- 各ユーザに配信するメッセージ:m(c) = m の集合

株価アラートシステムの性能目標は，配信するメッセージの作成処理(m を配信対象人数ごとに作成する処理を m_{ci} とし，全メッセージの作成は $m(c) = \Sigma m_{ci}$ となる）の時間を最小化することである．株価アラート生成システムからのデータ受信頻度の増加，データベースに格納されるデータ量の増加，及び配信対象人数が増加しても，処理時間を最小にし，スループットを最大としたい．

キューDB に対する処理は図 3-5 となる．

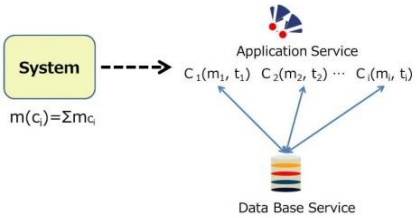

図 3-5：キューDB へのアクセスと処理時間の関係

各ユーザに配信するメッセージを作成する処理時間を t, C はアプリケーションが行う処理で，C = C(m, t)となる．T が小さいとシステムの処理性能が向上する．アプリケーションは, データの永続化を行うサービスが RDB でも KVS でも全く同じ処理を実行するので，キューDB の要素を変更すると性能が向上する可能性がある．

システムに対する要求数の変化に伴うシステムの状態遷移図は図 3-7 のようになる（RDB 型も KVS 型も同じ）．

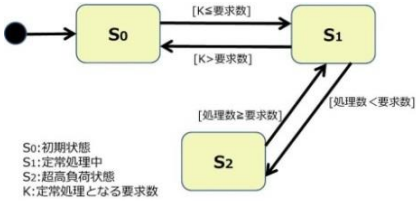

図 3-6：処理負荷の変化による状態遷移図

システム起動時の状態は S_0，システムに対する要求数が一定の数に達するとシステムが定常的に処理を行う状態 S_1 に遷移し，さらにシステムの処理性能を上回る要求数が発生すると S_2 に遷移する．時間の経過とともに処理を待つキューが減少し，最終的に要求数が 0 となるため，$S_2 \rightarrow S_1 \rightarrow S_0$ と状態が戻る．システム構築時は S_2 を避けることが目標となる．S_2 になると，ユーザにデータを配信するまでの時間が過剰に大きくなり，CPU の枯渇やメモリ不足等のシステム障害が発生する可能性が高くなる．

RDB 型でキューDB がボトルネックとなる問題が解消された場合，下記の効果が期待できる．

- KVS 型の方が S_1 から S_2 に遷移しにくくなる．
- S_2 となっても, KVS 型の方が S1 に戻る時間が短い．

アプリケーションがデータベース（キューDB 及び業務 DB）に行う処理は，RDB 型も KVS 型も全く同じである．

Vol.2017-DBS-165 No.29
Vol.2017-IFAT-128 No.29
2017/9/20

4. 実験と分析

4.1 実験の目的

正常時の性能を測定する．1 台のアプリケーションサーバで実現可能なスループット，及びアプリケーションサーバ数の増加に対する処理性能の増加傾向を把握し，必要なシステム構成の設計が可能となる．

4.2 実験環境

実験は複数の物理サーバ上に VMWare を用いて構築された仮想サーバ環境（JAIST Cloud）上で実施した．実験に使用した HW 構成及び SW 構成は表 4-1 の通り．

表 4-1 HW/SW 構成

項目	アプリケーションサーバ	データベースサーバ	実験データ取得用サーバ
台数	4	1	1
CPU	Intel(R) Xeon(R) CPU X5690 @ 3.47GHz		
仮想CPUコア数	8	16	16
仮想メモリ	12	24	24
仮想HDD	500	500	500
OS	CentOS Linux release 7.2.1511 (Core)		
その他	JDK 1.8.0_111 Cassandra 3.0.11.1564	MySQL 5.7.17	JDK 1.8.0_111

4.3 測定結果

以下の 2 つのパラメータを変化させて測定を行う．

- 配信対象人数
- アプリケーションサーバ数

配信対象人数は 10, 50, 100, 150, 200，アプリケーションサーバを 1～4 台と変化させて測定している．

以下の値を RDB 型と KVS 型それぞれに対して測定する．

- TPS（Transaction Per Seconds）
- 各サーバの CPU 使用率

TPS はシステム全体で処理する性能を把握する．また CPU 使用率はボトルネック箇所を把握する．TPS は最大値，及び平均値を測定し，CPU 使用率は最大値を測定している．

実験を実施し，下記の結果を得た．特徴的な例として，最も処理負荷の高くなるケース（アプリケーションサーバ 4 台，配信対象人数 200）を掲載している．

TPS

図 4-1 に TPS（最大値）の値を示す．TPS の値は大きい方がシステムで処理できる要求数が多く，性能が高いと考えられる．全体的に RDB 型より KVS 型の方が大きい値となっている．

スケーラビリティをみると，RDB 型はアプリケーションサーバ数が増加すると TPS の増加が頭打ちになる傾向がみられる．KVS 型の増加傾向は線型であり，アプリケーションサーバ数に比例して TPS が増加している．

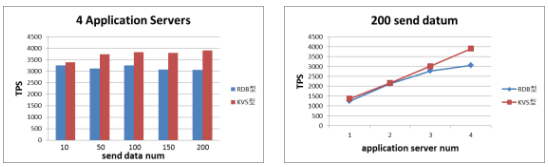

図 4-1：TPS 比較（最大値）
左：データ数変更　右：サーバ数変更

TPS（平均値）は処理負荷の低い場合に RDB 型の方が高い値になる結果があったものの，処理負荷の高い場合は KVS 型の方が高い値となった．

CPU 使用率

アプリケーションサーバの CPU 使用率は 90%を超過しない限り，大きな値が好ましい（ボトルネックがなく，システム全体の TPS が高くなるため）．90%を超えると CPU リソースが枯渇し，アプリケーションが正常に動作しない可能性がある．

図 4-3 にアプリケーションサーバの CPU 使用率を示す．全体的に KVS 型の方が大きく，3 倍程度の値である．処理負荷が高くなるほど RDB 型は減少傾向だが，KVS 型は一定以上の CPU を使用し，処理負荷の増加に応じて CPU を使用している．

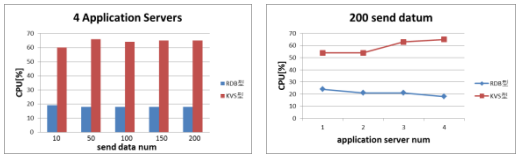

図 4-3：CPU 比較（アプリケーションサーバ）
左：データ数変更　右：サーバ数変更

DB サーバの CPU 使用率は小さな値が好ましい．これは DB サーバがボトルネックにならず，アプリケーションサーバの増加に対し，システム全体の TPS が向上するためである．

図 4-4 に DB サーバの CPU 使用率を示す．全体的に KVS 型の方が小さく，CPU 使用率の差で 20%程度の余裕がある．

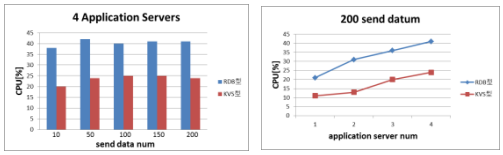

図 4-4: CPU 比較（DB サーバ）
左：データ数変更　右：サーバ数変更

この他，キューDB を強制停止することで，データベースの障害検知と復旧に関する実験を行っており，A2_耐障害性実験データ.pdf に結果を記載している．またキューDB の一部を停止した状態で性能劣化を確認する実験を行っており，結果として KVS 型はキューDB の障害発生後もデータベースの接続の切替及び切替後の動作の継続に問題がなく，キューDB が一つ停止した状態でも，大きく性能が低下することなく使用可能であることが分かった（実験の全データの記載については紙面の都合により割愛）．

4.4 分析

4.4.1 性能特性

配信するメッセージ数が 10 → 200 と増加するとシステムの処理負荷は高くなる．特にメッセージ数が 100 以上の場合，RDB 型より KVS 型の方が，TPS が高い値になることを示した．この理由として以下の 2 点が考えられる．

- キューDBに関して，RDB型はアプリケーションサーバと異なるVM上に存在するデータベースにアクセスするが，KVS型はアプリケーションサーバと同一のVM上に存在するデータベースにアクセスするため，通信処理のオーバーヘッドが少ない．
- RDB型は同一のテーブルに複数のアプリケーションからアクセスするため，処理の競合が発生する．同一の行データに対する競合は発生しないが，INDEX更新やデータベースのセッション管理，更新データの保存等で同時実行数及び競合が増える．対してKVS型は1データベースに1アプリケーションからのアクセスしか発生しない．RDB型と比較して，複数アプリケーション間で処理の競合が発生せず，応答時間が小さくなる．どちらもwrite-write，read-writeの競合は発生するが，KVS型の方が頻度が少なく，競合する確率が低い．

いずれもモデル上ではΣm_{ci}が小さい値となり，処理時間t_iが小さくなったことで数値の差となったと考えられる．

4.4.2 スケーラビリティ特性

RDB型はアプリケーションサーバ数が増加するにつれて処理性能が頭打ちになる傾向が見られる．対してKVS型は線型に近いTPSの増加となり，アプリケーションサーバ数とほぼ比例してスループットが向上する，高いスケーラビリティとなっている．

4.4.3 負荷分散について

RDB型はアプリケーションサーバの台数が1台→4台に増加するにつれてアプリケーションサーバのCPU使用率が下がる傾向にある．これはDBサーバの負荷が高くなり，DBサーバから応答が返りにくくなったため空き時間が増加したためと考えられる．結果として，TPSが頭打ちになる傾向になったと考えられる．

KVS型はDBサーバの処理負荷が下がった分，アプリケーションサーバのCPU使用率が増加したと考えられる．

システムのボトルネックが相対的にデータベースサーバからアプリケーションサーバに移動しており，システム設計時には要求される処理量に対して，アプリケーションサーバのCPUリソースを十分に確保する必要が出てくる．

5. おわりに

非同期でキューイング処理を行うシステムにおいて，RDB型が持つ問題を解決するため，KVS型が有効となると予想を立てた．実験の結果，KVS型は実際の利用に対し十分な耐障害性を持ち，データベース停止時も大きな性能劣化はなかった．正常な状態でRDB型よりも性能が向上し，スケーラビリティに優れていることを確認できた．今後のシステム構築において，KVS型を利用する見込みが立った．RDB型の抱える問題点を払拭し，データベースの障害発生時にも継続してサービスが提供でき，ユーザ数やデータ量の増加に対し，より柔軟なシステム構築が可能となる．

他のシステムへの応用を考えた場合，キューDBやアプリケーションサーバで使用しているフレームワークとしての仕組みは同じものが使用できると考えられる．

今後の課題としては以下があげられる．

（1）システムの可用性

実際の可用性を考えるならば，「3．4 モデル化」で示したメインとなる機能だけではなく，システムの全てのアプリケーション及び運用で用いる機能が実行可能であることを網羅的に確認する必要がある．

（2）アプリケーション機能とパラメータの性能影響

4章で行った実験と，実際のシステム稼働を比較した場合，アプリケーションから実行するSQLの回数は同程度のため，さほどの影響はない．しかし，下記の相違点に関して影響が出てくると考えられる

- システムからデータを配信する端末数
- ユーザが事前に登録する配信データ種類の設定

データを配信する処理単位とアプリケーションの処理フローやロジックが変わるため，1つのトランザクション内での計算量が増加して性能が劣化する可能性がある．性能要件を満たすためのHWを用意し，現行のRDB型を用いたシステム開発でも行っているように，KVS型でも同様にテストと確認を実施する必要がある．

参考文献

[1] WikiPedia(Distributed computing)
https://en.wikipedia.org/wiki/Distributed_computing
[2] WikiPedia(Distributed database)
https://en.wikipedia.org/wiki/Distributed_database
[3] WikiPedia(Message queue)
https://en.wikipedia.org/wiki/Message_queue
[4] WikiPedia(Relational Database)
https://en.wikipedia.org/wiki/Relational_database
[5] MySQL Replication Tutorial - O'Reilly Media
http://assets.en.oreilly.com/1/event/2/MySQL%20Replication%20Tutorial%20Presentation%202.pdf
[6] Oracle RAC の概要
https://docs.oracle.com/cd/E16338_01/rac.112/b56290/admcon.htm
[7] High-availability cluster
https://en.wikipedia.org/wiki/High-availability_cluster
[8] Oracle NoSQL Database
http://www.oracle.com/technetwork/database/database-technologies/nosqldb/overview/index.html
[9] HBase: The Definitive Guide By Lars George, Publisher: O'Reilly Media
[10] Cassandra: The Definitive Guide By Eben Hewitt, Publisher: O'Reilly Media