大規模分散処理システムの検証の効率化に関する研究

梅田昌義[†1]

概要：商用運用を目的としたシステム全体の検証では，商用サービスを開始する日取りが決定しているため，決められた期間内で検証を実施する必要がある．しかし，従来の方法で大規模分散処理システムのシステム検証を行うと，検証作業に時間を要して決められた期間を延伸してしまう問題があった．本研究では，決められた期間を延伸させてしまう主要な 6 点の課題を解決する手法が他のシステムに適用できるかどうかを考察した．その結果，それぞれの手法について適用可能な領域が判明した．

キーワード：大規模分散処理システム， システム検証， 効率化

A study on efficient verification of large-scale distributed processing systems

MASAYOSHI UMEDA[†1]

1. はじめに

近年，BigData の管理と処理が可能なことから，Hadoop に代表される大規模分散処理システムの利用が広まっている．商用運用を目的としたシステム全体の検証(システム検証)では, 商用サービスを開始する日取りが決定しているため，決められた期間内で検証を実施する必要がある．しかし，従来の方法で大規模分散処理システムのシステム検証を行うと，検証作業に時間を要して決められた期間を延伸してしまうことが問題である．筆者は決められた期間を延伸させてしまう主要な 6 点の課題を抽出し，それぞれの解決手法について整理した[1][2][3].

本研究では，大規模分散処理システムのシステム検証で，主要な 6 点の課題を解決する手法が，他のシステムに適用できるかどうかを考察した．その結果，それぞれの手法について適用可能な領域が判明した．

本稿では，検証の工程の分類をアクティビティに基づいて説明する．具体的には， IEEE で作成したソフトウェアエンジニアリングの基礎知識体系の SWEBOK[4]に記述されている工程の分類に基づいて説明を行う[a].

2. 関連研究

本章では，従来のシステム検証での作業を示し，システム検証の効率化に関する既存の研究を示す.

2.1 従来のシステム検証

計画のアクティビティでは，要求仕様から検証の内容を決定し，計画を立案する．計画の立案は，従来の情報システムでの開発における検証の経験や，類似なシステム開発のデータに基づいて行う．計画アクティビティで利用される技術は，プロセス計画，成果物の決定，工数，工程およびコスト見積もり，資源割り付け，リスクマネジメント，品質マネジメント，計画マネジメントである[4].

テストケース生成のアクティビティでは，検証項目は，実施される検証のレベルおよび用いられる手法に基づいて作成される．検証項目の作成は，網羅的に検証項目を抽出し, 同値分割, 境界値分析[5][6][7][8]等により検証内容の精査をして完成させる．テストケース生成アクティビティで利用される技術は，論理網羅テスト，組み合わせテスト，同値分割，境界値分析，原因-結果グラフ，機能テスト，ユースケース，統計的テスト，シナリオ，リスクベース，例外ユースケース，欠陥仮設法，エラー推測，状態遷移テスト，モデルチェッキング，タイミングテスト，静的解析によるピンポイントテスト，ランダムテストである[39].

テスト環境の開発のアクティビティでは，検証の作業環境は, 検証項目の作成を容易に行えるものであると同時に，検証の結果を記録し，検証を再現できるようにしなければならない．検証に必要な検証の作業環境は，手作業と OSS や Shell script を使って構築する．テスト環境の開発アクティビティで利用される技術は，ソフトウェアエンジニアリングツール，保守ツールである[4].

実行のアクティビティでは，検証は文書化された手続きに沿って, 検証対象であるソフトウェアに対して実施する．既存のツールが利用できればそのツールを利用し，利用できない場合は手作業で検証して結果をコンソールやログで確認する．実行アクティビティで利用される技術は，動的検証と静的検証に分類される[4] [9][10][11][12]．動的検証は，さらにホワイトボックステストとブラックボックステスト結果の評価」、「問題報告/テストログ」、「欠陥追跡」の 7 工程で分類されている．

†1 日本電信電話株式会社
NTT.

a) 「計画」、「テストケース生成」、「テスト環境の開発」、「実行」、「テス

ストに分類される.ホワイトボックステストの検証方法は，パステスト，トランザクションフローテスト，データフローテストの技術がある[8]．ブラックボックステストの検証方法は，ドメインテスト，状態遷移（グラフ）テスト，ランダムテストの技術がある[8]．静的検証では形式手法による検証方法[13][14][15]がある．

テスト結果の評価のアクティビティでは，実行結果はログから分析して確認する．テスト結果の評価アクティビティで利用される技術は，テストされるプログラムの評価，実施されたテストの評価である[4].

問題報告／テストログのアクティビティでは，検証結果は，検証ログに管理し，予期しないまたは不正な結果は問題報告に記録する．検証ログは，ツールや手動で収集し結果を有識者が確認をする．問題報告／テストログアクティビティで利用される技術は，データの収集，データの分析および情報プロダクトの開発，結果の伝達である[4].

欠陥追跡のアクティビティでは，検証中に摘出された問題は，品質報告の資料を作成するときに，ソフトウェアの欠陥に基づく．欠陥に対する分析を行う．欠陥追跡アクティビティで利用される技術は，欠陥，増補，論点および問題追跡ツールである[4].

上記のように，各アクティビティではソフトウェアの品質を確保するために，漏れなく網羅的に検証する方法は示されているが，効率化に関する方法は示されていない．

2.2 システム検証の効率化の研究

大規模分散処理システムに対するシステム検証効率化の課題に対して，既存の研究を下記(1)〜(3)の観点で調査・分析した結果を示す.

(1) **大規模分散処理システムのシステム検証に関する研究**

大規模分散処理システムのシステム検証は,通常のマシン数台により構成されるシステム（以降，一般的な情報システムと呼ぶ）の検証とは異なり，数百台規模のマシンをリソースとして管理し検証を実施する必要がある．しかし，大規模分散処理システム自体のアーキテクチャが新しいことから，大規模分散処理システム固有の問題を考慮した検証の効率化に関する公知の文献は，これまでほとんどなかった.既存の論文[2][16][17]は，システム検証における性能の問題と項目作成の問題，あるいはクラウドシステムの検証におけるレガシーシステムの検証と比較した際の有効性,を扱うにとどまっている．また，Google のシステムに関する検証[18]や Microsoft のシステムに関する検証[19]は，ログの出力や API を利用したトラッキングやモニタリングによる動作やボトルネックの確認であり，システム固有の確認方法であるため，他の類似した大規模分散処理システムでは確認ができない.

(2) **分散処理システムの検証に関する研究**

分散処理システムの検証に関しては，論文[20][21]があるが，数百台規模のマシンを利用した検証効率化の観点はない.

(3) **大規模システムと一般的な情報システムの検証に関する研究**

システム検証を一般的な情報システムのソフトウェア開発の検証については文献[5][6][7][8]が詳しい.しかし,検証の実施に際しては，検証の課題やその課題の対処方法の具体的な記述はない．例えば，今回の大規模分散処理システムに特有な，マシン台数も多く，プロダクトも複数あることから発生する検証実施に時間を要する問題や，故障検知の問題等に適用することができない.

3. 課題と解決手法

従来の情報システムにおけるシステム検証の方法で，大規模分散処理システムの検証期間を計画し，検証すると，即日に解決しきれない問題が発生し，計画した以上の期間を要してしまった.これらの発生した問題から数週間 (2 週間以上）の期間を要するもので，上流の作業工程におけるアクティビティを抽出した．2 週間以上というのは，検証期間が 6 ヶ月の場合，検証期間の 1 割以上を占めることになり，全体への影響が大きいからである．また，上流の作業工程のアクティビティは，プロジェクトマネジメントでは上流の作業工程が重要であるからである[22]．その結果,主要となる課題 6 つを抽出した．そして，それぞれの課題に対して，表 1 のように解決手法を整理した．以降にそれぞれの課題と解決手法について詳述する.

3.1 計画アクティビティにおける課題と解決手法

計画アクティビティでは，検証前に準備するデータ蓄積の作業においてデータ量が大量のため時間を要し検証開始が遅延する問題，検証項目の実施時間を見積もる精度が低く計画した期間を遅延する問題，発生する問題の解析により検証項目の実施が止まってしまう問題が発生した.

これらの課題に対する対策として,それぞれ「I/O デバイスのチューニングによる高速データ登録手法」（手法 1)，「大量データ観点の事前検証による見積もり精度向上手法」（手法 2)，および「検証マシン台数分割と段階的検証による検証実施手法」（手法 3）を提案する．これら 3 つの手法を適用することにより，このアクティビティにおける数週間に及ぶスケジュールの遅延をなくすことができた.

手法 1 は，大量の検証データを蓄積する日数を短縮するために，検証データを安定して高速登録するという課題を解決するものである．本研究では，高速登録のボトルネックがネットワーク I/O にあることと，性能が不安定となる要因がディスク I/O であることを突き止めた．ネットワーク I/O の「TCP バッファのサイズを 1.5 倍」にするチューニングにより，システム運用時の性能要件以上の書き込み

速度を実現し，ディスク I/O の「Merge Compaction のランダム化」のチューニングにより，データ登録の最中に不安定にスループットが低下することなくデータ登録が実施できた．

手法 2 は，限られた検証期間内で計画した検証のスケジュールの確度を高めるために，検証項目の実施時間を効率的にかつ精度よく見積もるという課題を解決するものである．本研究では，見積もりと実施時間の差が大きい検証項目が大量にデータを使用する項目であることを突き止め，大量にデータを使用する項目を選別して，選別した検証項目の実施時間を測定した．この手法とデータ量が多くない項目を従来の方法とすることにより，従来では期間を超えていた検証を見積もりどおりの期間内に完了できた．

手法 3 は，発生した問題の解析と対処に要する時間を短縮するために，限られた検証用のマシン台数を検証項目にどのように割り当てるのかの課題を解決するものである．サービスイン時と同数のマシンを利用して大規模分散処理システムのシステム検証をする場合，発生した問題の解析と，改修した後の検証をしやすいようにする必要がある．本研究ではマシン台数を小規模から大規模までのように複数に分けて，これらを検証環境とした．そして，発生した問題の解析と改修した後の検証がしやすいように，複数に分けた検証環境を検証項目の内容に応じて割り当て，小規模から大規模へと段階的に検証を実施する方法で課題を解決した．

3.2 テストケース生成アクティビティにおける課題と解決手法

テストケース生成のアクティビティでは，従来の方法で網羅的に項目を作成すると項目数が膨大になる問題が発生した．本研究では，この策定した検証のスケジュールを数週間遅延させた問題から課題「大量な検証項目の項目削減」（課題 4）を抽出した．

この課題の対策として「システム特徴と問題発生傾向の 2 段階精査による検証項目削減手法」を提案する．この手法を適用することにより，このアクティビティにおける数週間に及ぶスケジュールの遅延をなくすことができた．

手法 4 は，検証期間の短縮のために，膨大な検証項目を半数以下に削減するという課題を解決するものである．本研究では，(1)検証観点を大規模分散処理システムの特徴である「資源効率性」，「障害許容性」，「回復性」に重点を置いて網羅的に検証し，それ以外の検証項目を減らす精査をした．そして，(2)検証で摘出された発生確率の高い問題に重点を置いて網羅的に検証し，それ以外の検証項目は減らす精査をした．この手法を適用することで，網羅的に抽出した項目数の 1/4 の試験項目が完成した．

表 1. 検証の課題と解決手法

アクティビティ	検証の課題	解決手法
計画	課題 1:大量な検証データの高速データ登録	手法 1 : I/O デバイスのチューニングによる高速データ登録手法
	課題 2:検証実施時間の見積もり精度向上	手法 2 : 大量データ観点の事前検証による見積もり精度向上手法
	課題 3: 効率的にバグ摘出するマシン台数分割による検証実施	手法 3 : 検証マシン台数分割と段階的検証による検証実施手法
テストケース生成	課題 4:大量の検証項目の項目削減	手法 4 : システム特徴と問題発生傾向の 2 段階精査による検証項目削減手法
テスト環境の開発	課題 5:性能確認や機能確認の効率化	手法 5 : データ生成とログの出力を組み込んだ検証ツールの作成手法
	課題 6:故障検知と故障対応の迅速化	手法 6 : 予備機の入れ替えと監視ツールによる検証環境の正常化手法

3.3 テスト環境の開発アクティビティにおける課題と解決手法

テスト環境の開発アクティビティでは，検証環境は検証を効率よく実施でき，マシンの故障に影響されない環境であることが必須である．商用運用と同様な環境を準備し検証すると，インターネット接続によるデータ収集が可能なようにグローバル IP を取得する必要があるなど運用条件のある AP（Application Program）を工夫しながら利用することになった．また，AP を利用していたのでは，検証で必要な書き込みデータと書き込まれたデータの整合性の確認や，データサイズの変更などデータ条件を細かく変化させた測定ができないという問題が生じた．また，検証マシンの台数が多く，マシンの故障が日々発生する場合は，再検証により時間を要してしまう．そのため検証を期間内に完了できなくなるという問題が生じた．本研究では，テスト環境の開発アクティビティにおける主要な問題を整理した結果，策定した検証のスケジュールを数週間遅延させた問題から 2 つの課題「性能確認や機能確認の効率化」（課題 5），「故障検知と故障対応の迅速化」（課題 6）を抽出した

される．

これらの課題の対策としてそれぞれ「データ生成とログの出力を組み込んだ検証ツールの作成手法」（手法 5），「予備機の入れ替えと監視ツールによる検証環境の正常化手法」（手法 6）を提案する．この解決手法を適用することにより，このアクティビティにおける数週間に及ぶスケジュールの遅延をなくすことができた．

手法 5 は，数百台規模のマシン環境において，効率的に検証作業を行って検証実施時間を短縮するために，どのような機能を検証ツールに具備するかの課題を解決するものである．本研究では，性能と機能を確認できる仕組みとして，(1)複数台の TP(Test Program)を集中制御可能とする仕組み，(2)書き込みデータの内容をパラメータ等で柔軟に変更可能とする仕組み，(3)スループットの計測やデータ内容の確認ができる仕組みを用いた．この手法を適用することで，仕組みを利用しない場合では，想定では約 1 ヶ月要するところを 7 日で完了し，検証実施時間の短縮を実現することができた．

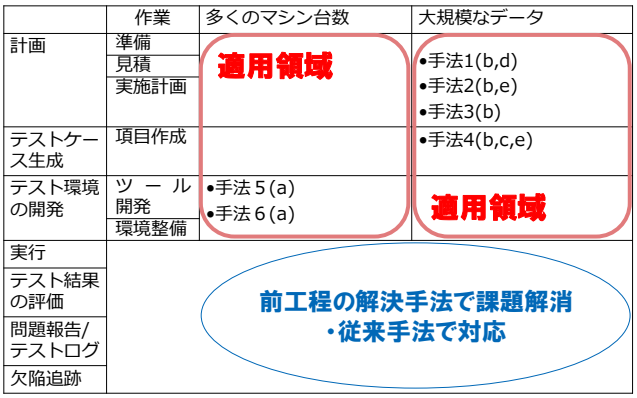
手法 6 は，検証の手戻りをしないように，数百台のマシン環境において，検証で問題となるマシンの故障をどのように検知し，検証に影響を与えないように回復できるかという課題を解決するものである．本研究では，利用する特定プロトコルの故障を速やかに検知するために，通信経路の確認やログ監視をスクリプトや Cron による定期チェックで故障検知を行い，検証に影響の出るマシンを逐次，確保した予備のマシンと入れ替えた．この手法を適用することで，検証への影響を最小限にし，手戻りのない検証が実現できた．

4. 他システムへの適用の考察

提案した解決手法について，他システムへの適用する場合，多くのマシン台数（スケールアウト（分散））であること，大規模なデータ量（大量なデータ処理）であるという 2 つのシステムの特徴に基づき適用領域を表 2 に整理した．手法 3，4，5，6 の解決手法は，多くのマシン台数利用が要因の課題を解決することから，この領域に効果を発揮する手法であることが判る．よって，手法 3，4，5，6 の解決手法は，スケールアウト（分散）するシステムに適用が可能ということが考えられる．一方，手法 1，2 の解決手法は，大規模なデータ量の利用が要因の課題を解決することから，この領域に効果を発揮する手法であることが判る．よって，手法 1，2 の解決手法は，大規模なデータ量（大量なデータ処理）のシステムに適用が可能ということが考えられる．以降での他のシステムへの適用を考察した結果，以下の適用可能性が具体的に判明した． 手法 3 と手法 5 と手法 6 は， 1 サイトあたり多くのマシン台数からの処理を有するシステムへの適用が可能と考えられる．手法 4 は，多くのマシン台数からの処理を有するだけではなく，読み書きを主とした処理を有し，データ複製処理を有するシステムへの適用が可能と考えられる．手法 2 は，大規模なデータを有するシステムに適用が可能と考えられる．手法 1 は，大規模なデータを有するだけではなく，トランザクション処理を有するシステムに適用が可能と考えられる．

以降，他のシステムへの適用について考察を詳述する．最初に他の大規模分散処理システムへの適用について，次に他の大規模分散処理システムと RDBMS を含めて適用について述べる．

表 2. 解決手法の分類と適用領域



手法（）内はシステムの処理特性： a.多くのマシン台数(/サイト)， b.大規模なデータ，c.読み書きが主な処理， d.トランザクション処理， e.データ複製処理

4.1 解決手法の分類と適用領域

各課題とそれぞれの解決手法について，分散ロックと分散ファイルと分散テーブルのプロダクトで構成される大規模分散システムで，どのような検証に適用が可能かを整理する．

計画アクティビティにおける課題の解決手法は，ネットワーク I/O とディスク I/O のチューニング，リソースの大きい項目に着目した事前の検証，適正なマシン台数を検証項目に割り当てる，というアプローチであり，これらは汎用的なものである．手法 1 は VM（Virtual Machine）ではなくベアメタルのマシンのように，ネットワークとディスクの I/O のボトルネックが存在する場合に適用すると有効である．分散ロックと分散ファイルと分散テーブルのプロダクトで構成される大規模分散システムであっても，数十テラバイト級のデータ量を管理するシステムでは，数百テラバイト級のデータ量を管理する場合と比較すると効果が少なくなる．手法 2 は大量のデータを検証で利用する場合に適用すると有効である．しかし，大量にデータを利用しないシステムの検証では，検証実施時間を要することはないため，効果を得ることができない．手法 3 は大規模分散

処理システムとして数百台規模のような大規模のマシン台数を使用し，他に利用可能なマシン台数がない場合に適用すると有効である．マシン台数の分割をする手法のため，検証に利用するマシン台数が少なく他に利用可能なマシンがある場合は，効果を得ることができない．

テストケース生成アクティビティにおける課題の解決手法は，システムの特徴の観点とバグ傾向を考慮した検証項目の絞込みというアプローチであり，これは汎用的なものである．手法 4 は分散ロックと分散ファイルと分散テーブルのプロダクトで構成されるように，複数のプロダクトによる組み合わせで項目数が多くなる場合に適用すると有効である．しかし，マシン台数が少なく，データの欠損を許容する場合は，プロダクト毎のマシン故障による組み合わせにより，データ欠損や処理不具合を摘出するための検証をする必要が無く，検証項目の数が膨大にならないため効果が少なくなる．

テストケース環境の開発アクティビティにおける課題の解決手法は，TP（Test Program）の集中制御，故障検知と故障マシンの入れ替えというアプローチであり，これらは汎用的なものである．手法 5 は，多くの AP（Application Program）による数客台規模のマシンに同時の読み書きの処理がある場合に有効である．しかし，１台や複数台のようにマシン台数が少ない場合では，AP を動作させるための検証ツールを作成せず，手動で AP を動作させることが可能であるため効果を得ることができない．手法 6 は数百台規模のマシンを利用しており，故障したマシンを使い捨てにせず再利用する場合に有効である．マシン数十台規模の利用で，保守メンテナンスの修理で対応できる場合は，マシン１台/1 年の故障率で最大で１台／月程度の故障となるため，効果を得ることができない．また，Google のシステムのように故障したマシンを使い捨てにする場合も効果を得ることができない．

4.2 代表的な他の分散処理システムや RDBMS への適用

大規模データに対応したシステムとしては，分散処理システムや一般的な情報システムである RDBMS を大規模対応したシステムがある．従来の情報システムの RDBMS や Hadoop というレコードオリエンテッドストア な大規模分散処理システムへの提案手法の適用可能性だけでなく、カラムオリエンテッド な分散処理システムやドキュメントストア といった,大規模データを管理可能なシステムについて，提案手法の適用可能性を考察する．さらに，大規模分散処理システムの AP である MapReduce を利用したシステムについても提案手法の適用可能性を考察する．

以降に各システムの種類毎について適用可能性示す．ユースケースは，インターネット上からクローリングしたデータを検索する処理である．表 3 に各システムと提案手法の適用可能性を示す．

表 3. 他システムへの提案手法の適用可能性

システム種類		提案手法の適用可能性（効果の度合い）					
		手法1	手法2	手法3	手法4	手法5	手法6
		b,d	b,e	b	b,c,e	a	a
NoSQL	(1)大規模分散処理システム（レコードオリエンテッドストア）	○	○	○	○	○	○
NoSQL	(2)カラムオリエンテッドストア	×	○	○	○	○	○
NoSQL	(3)ドキュメントストア	○	○	○	○	○	○
(4)RDBMS		×	○	×	×	×	×
(5)MapReduce		NoSQL のシステムに依存					

(1) 大規模分散処理システム（レコードオリエンテッドストア）は，行単位でデータを書き込み，読み込みをするのに優れており，すべての提案手法の適用が可能である．

(2) カラムオリエンテッドストアは，列方向のデータをまとめて扱うシステムであり[14]，列の値の集計処理に優れる．代表的なシステムとしては Cassandra ，Oracle Exadata ，Netezza ， SAP HANA ，がある．カラムオリエンテッドストアは，以下のシステム構成とデータ管理と処理機能を有している．

・システム構成は，ノードと呼ばれる千台規模のサーバで構成される．
・管理可能なデータ容量はペタバイト級である．
・データ管理は，千台規模のノードで分散したデータを管理し，複製データはすべてノードに非同期でコピーされる．カラム単位の読み書きであり，データの排他制御は簡易な実装である．そのため，レコード追加ではすべての列に追加の操作が必要となる．列毎にデータを管理しているため，列の値の集計処理に優れる．

以下に提案手法の適用可能性について示す．

・手法 1: 大規模分散処理システムのようなレコードオリエンテッドではなく，カラムオリエンテッドで列方向にデータをまとめて管理しているため，行を追加するには，それぞれのカラム毎すべてに追加の処理をする必要がある．手法 1 による行毎の大量データ投入をすると，時間を要するため，手法 1 の適用の効果は低いと考えられる．
・手法 2: ペタバイト規模のデータを管理するため，検証に要する時間のばらつきがあると考える．そのため，大量データ観点の事前検証による見積もりの効果が高いと考えられる．
・手法 3: 千台規模のサーバで構成されることから，検証で有しているマシン台数が運用上のマシン台数と同じ場合があり，効率的にバグを摘出できないことが考えられる．そのため，検証マシン台数分割と段階的検証の効果が高いと考えられる．
・手法 4: ランダムリード・ランダムライトの処理や複製データによるデータ保障の確認を千台規模のサーバで確認することから，試験項目数が多くなることが考えられる．そのため，システム特徴と問題発生傾向の 2 段階精査による項目削減の効果が高いと考えられる．
・手法 5: 千台規模のマシンを利用し，複数のスレッドをコントロールする必要があるため，複数のスレッドをコントロールするデータ生成やログ出力を組み込んだ検証ツールの効果は高いと考えられる．
・手法 6: 千台規模のサーバで構成されることから，マシンの故障は頻発することが考えられる．予備機の入れ替えと監視ツールによる効果が高いと考えられる．

提案手法の 1 以外で効果があることから，12 ヶ月要するところを 6 ヶ月短縮して 6 ヶ月にできる．

(3) ドキュメントストアは，半構造化データを格納するためにスキーマレスでデータ構造が柔軟にできるよう設計しているシステムであり[14], 分散処理が可能である．代表的なシステムとしては MongoDB, amazon DynamoDB がある．ドキュメントストアは，以下のシステム構成とデータ管理と処理機能とを有している．

・システム構成は，１台のプライマリサーバと千台規模のセカンダリサーバの構成である．
・管理可能なデータ容量は数ペタバイト級である．
・データ管理は，1 台のプライマリサーバでデータ管理し，複製データをセカンダリサーバで管理する．データの排他制御はデータベース全体やドキュメント単位で排他制御をかけることができる．これにより，トランザクション処理では，データ一貫性を保障する．データ複製は全てのサーバに確保するのではなく，ある一定台数のサーバに同期することで処理の高速化を図っている．
・データ構造は，スキーマレスで柔軟なデータを扱うことを可能としている．また，データの読み書きは，単純なデータの追記と読み出す処理をすることで高速化を図っている．

以下に提案手法の適用可能性について示す．

・手法 1: スレッドを排他する機能はないため，大量のデータの書き込みがあった場合，I/O デバイスのボトルネックが発生する可能性があり，I/O デバイスのチューニングによる効果が高いと考えられる．
・手法 2: ペタバイト規模のデータを管理するため，検証に要する時間のばらつきがあると考える．そのため，大量データ観点の事前検証による見積もりの効果が高いと考えられる．
・手法 3: 千台規模のセカンダリサーバで構成されることから，検証で有しているマシン台数が運用上のマシン台数と同じ場合があり，効率的にバグを摘出できないことが考えられる．そのため，検証マシン台数分割と段階的検証の効果が高いと考えられる．
・手法 4: シーケンシャルリードの処理や複製データによるデータ保障の確認を千台規模のサーバで確認することから，試験項目数が多くなることが考えられる．そのため，システム特徴と問題発生傾向の 2 段階精査による項目削減の効果が高いと考えられる．
・手法 5: 複数のマシンを利用し，複数のスレッドをコントロールする必要があるため，複数のスレッドをコントロールするデータ生成やログの出力を組み込んだ検証ツールの効果は高いと考えられる．
・手法 6: 千台規模のスレーブで構成されることから，マシンの故障は頻発することが考えられる．予備機の入れ替えと監視ツールによる効果があると考えられる．

提案手法の全てで効果があることから，12 ヶ月要するところを 7 ヶ月短縮して 5 ヶ月にできる．

(4) RDBMS は，１台のサーバで集中してオンラントランザクションの処理を得意とする RDBMS であり，ＡＣＩＤ特性[15]のあるシステムである．RDBMS のシステムには，SQL Server，MySQL , Oracle Database などがある．RDBMS は行レベルロック，読み取り一貫性，堅牢性，移植性の特徴を有し，以下のシステム構成とデータ管理等を有している．

・システム構成は，１台のサーバと数十台規模のクライアントの構成である．
・管理可能なデータ容量は数百テラバイト級である．
・データ管理は，１台のサーバにより，数十台規模のクライアントに分散されたデータを管理する．レコードオリエンテッドなシステムであり，データの排他制御は，カラム単位，レコード単位，テーブル単位でできる．トランザクション処理における２フェーズコミットやジャーナル処理により，データ一貫性を保障する．
・データ構造は，スキーマを定義し，カラムは型とデータサイズが決まっている．データの挿入時は排他制御とインデックス作成の処理により時間を要するが，読み出し

はインデックスを利用するため高速である．

以下に提案手法の適用可能性について示す．

・手法 1: 1 台のサーバによりトランザクション制御やレコード単位，テーブル単位などの細かい排他制御を実施するため，I/O デバイスのボトルネックは発生することはなく，I/O デバイスのチューニングによる効果が低いと考えられる．また，コンパクション処理に相当するジャーナル処理は，コンパクションのように分散して発生せず，システムで管理されて一括でバックグラウンド処理がされるため，提案の手法の効果が低いと考えられる．

・手法 2: RDBMS で管理するデータは，スキーマで定義されたデータである．一方，大規模分散処理システムで管理するデータは KVS データであるため，RDBMS でデータを管理する場合は，カラムの値を最大の値で定義するなど工夫が必要である．データを管理できた場合，検証に要する時間のばらつきがあると考える．そのため，大量データ観点の事前検証による見積もりの効果が高いと考えられる．

・手法 3: 数十台規模のマシンで構成されることから，マシン台数が不足する問題は発生しないことが考えられる．そのため，検証マシン台数分割と段階的検証の効果は低いと考えられる．

・手法 4: 数十台規模のマシンで構成されることから，マシンで動作するプロダクトの組み合わせによる試験項目数の増大はないと考えられる．そのため，システム特徴と問題発生傾向の 2 段階査による項目削減の効果は低いと考えられる．

・手法 5: 複数のマシンを利用し，複数のスレッドをコントロールする必要があるが，数十台のマシンに対してであることから，スレッドをコントロールするデータ生成やログの出力を組み込んだ検証ツールの効果は低いと考えられる．

・手法 6: 数十台のマシンの管理でよいことから，マシンの故障が頻発することはないと考えられる．そのため，予備機の入れ替えと監視ツールによる効果は低いと考えられる．

提案手法の 2 のみで効果があることから，12 ヶ月要するところを 1 ヶ月短縮して 11 ヶ月にできる．

(5) MapReduce は，分散処理システム上に構築する APL で，シンプルな API である Map 関数 reduce 関数からできている[89]．そのため，分散処理特有な複雑な負荷分散や可用性が隠ぺいされている．

MapReduce は，以下のシステム構成とデータ管理と処理機能とを有している．

・システム構成は，千台規模の分散処理マシン上に構築される．

・処理可能なデータ容量は数ペタバイト級である．

・データ管理やデータ構造は，構築されている分散処理システムと同じである．

提案手法の適用可能性については，手法 1～手法 6 のそれぞれについて，基盤としている分散処理に依存し異なる．

以上の各システムの種類に対する提案手法の適用性について，スケールアウト（マシン台数）とデータ量を軸とした図 1 に示す．カラムオリエンテッドストアは，管理するデータはペタバイト級で，スケールアウトし千台規模の多くのマシン台数で動作するため，大規模分散処理システムと同様に図 1 全体の範囲に位置する．ドキュメントストアは，管理するデータはペタバイト級であるが，マシンは数十台規模であることから，図 1 の中央から下方の範囲に位置する．RDBMS は，管理するデータは数ギガバイト級であり，マシンは数台規模であることから，図 1 の左下の範囲に位置する．MapReduce は，NoSQL のシステムと同じ特徴・処理特性であることから，重ねての表示はしない．

次にそれぞれ位置づけられた各システムに適用可能な提案手法を図 1 に重ね合わせる．そうすると，提案手法の手法 5，手法 6 は，主に数百台規模以上のマシンでスケールアウト（分散）するシステムに適用可能なことが判る．一方，提案手法の手法 1，手法 2，手法 3，手法 4 は， 主に大量なデータ（数百テラバイト級以上のデータ）を管理するシステム に適用可能なことが判る．ただし手法 1 は，大量のレコードデータを投入するシステムに適用可能であるため，大規模分散処理システム（レコードオリエンテッドストア）とドキュメントストアに適用可能である．

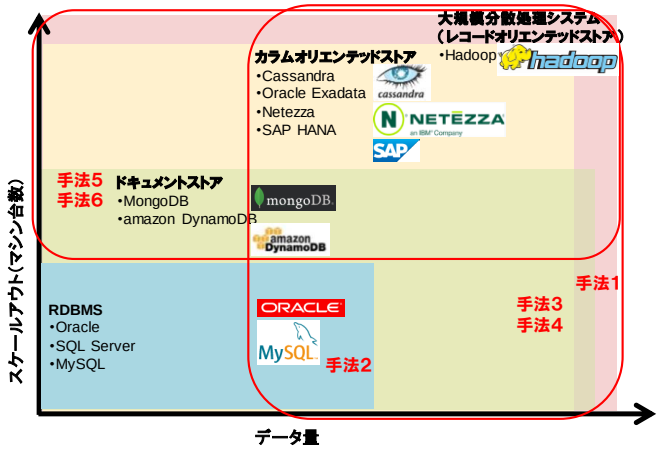


図 1. 代表的な他の分散処理システムや RDBMS における本成果の適用範囲

5. おわりに

本研究では，決められた期間を延伸させてしまう主要な 6 点の課題を解決する各手法が他のシステムに適用できるかどうかを考察した．その結果，それぞれの手法について適用可能な領域が判明した．今後は実際に他の分散処理シス

テムやＲＤＭＢＳに対して各手法の適用確認を行い，調査を進めていく予定である．

参考文献

[1] 梅田昌義，鬼塚真. 計画アクティビティにおける大規模分散処理システムのシステム検証の効率化. 情報処理学会デジタルプラクティス, Vol7, No.3(2016) , pp330-339.
[2] 坂井俊之, 梅田昌義, 中村英児, 本庄利守. 大規模分散処理システムのソフトウェア試験とその実践. 情報処理学会デジタルプラクティス, Vol4, No.1(2013) , pp51-59.
[3] 梅田昌義. テスト環境の開発アクティビティにおける大規模分散処理システムのシステム検証の効率化. 情報処理学会デジタルプラクティス, Vol8, No.4(2017), pp361-368.
[4] 松本吉弘(監訳). “ソフトウェアエンジニアリング 基礎知識体系 SWEBOK “. オーム社(2004).
[5] Cem Kaner, Jack Falk, Hung Quoc Nguyen. Testing Computer Software 2nd edition, John Wiley & Sons, Inc(1999).
[6] Glenford J. Myers. “ソフトウェア・テストの技法 第 2 版”, 近代科学社(2012).
[7] 長尾 真(監訳)，松尾正信(訳). “ソフトウェア・テストの技法 第 2 版”, 近代科学社(2012).
[8] ボーリス・バイサー. “ソフトウェアテスト技法”, 日経 PB マーケティング(2011).
[9] 黒坂均, 竹村和祥, 橘昌良. システムレベル設計フローと設計言語, 情報処理学会誌 Vol.45 No.5, pp456-463(2004).
[10] 根路銘崇, 小野康一, 田井秀樹, 安部麻里. Web アプリケーションモデルに基づく JSP の動的検証, ソフトウェアテストシンポジウム 2004, pp61-67(2004).
[11] Y. Falcone, et al.: A Tutorial on Runtime Verification, Engineering Dependable Software Systems, 34, IOS Press, pp.141-175(2013).
[12] 進博正, 遠藤侑介, 片岡欣夫. ソフトウェアの動作検証支援システム ARVE, 東芝レビュー Vol.62 No.9, pp. 55-58 (2007).
[13] 来間啓伸. B メソッドと支援ツール, コンピュータ・ソフトウェア, Vol. 24, No.2, pages 8‐13, 2007.
[14] 中島震. モデル検査を用いたソフトウェアの形式検証, 日本ソフトウェア科学会チュートリアル, 2004/2005.
[15] 中島震. モデル検査法のソフトウェアデザイン検証への応用, コンピュータ・ソフトウェア, Vol.23, No.2, pages 72‐86, 2006.
[16] Jerry Gao, Xiaoying Bai, Wi-Tek Tsai. Cloud Testing – Issues, Challenges, Needs and Practice, SOFTWARE ENGINEERING, An International Journal, Vol1, No.1,(2011).
[17] 廣川裕, 林孝志, 山中章裕, 吉田悟. 商用サービス適用のための大規模分散処理システムの性能評価, Vol4,No.1(2013).
[18] Benjamin H. Sigelman, Luiz Andr´e Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, Chandan Shanbhag. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure, Google Technical Report,(2010).
[19] LIU, X., GUO, Z., WANG, X., CHEN, F., LIAN, X., TANG, J.WU, M., KAASHOEK, M. F., AND ZHANG, Z. D3S. Debugging Deployed Distributed Systems. In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI’08, USENIX Association, pp. 423–437(2008).
[20] 山根 智: 時間オートマトンによる分散システムの仕様記述と検証の方式の提案, 電子情報処理学会論文誌, Vol.J79-D-I No.8, pp.511-521(1996-8).
[21] 田村 慶信, 内田 雅也, 山田 茂, 木村 光宏: 分散開発環境に対する確立微分方程式に基づくソフトウェア信頼度成長モデルの一般化, 電子情報処理学会 信学技報, R2002-54(2002-11).
[22] PMI. A “Guide to Project Management Body of Knowledge, 6th ed”., PMI(2018).