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

電気通信大学大学院

情報システム学研究科

博士(工学)の学位申請論文

2020 年 3 月

大規模分散処理システムの検証の効率化に関する研究

博士論文審査委員会

主査 植野 真臣 教授
委員 大須賀 昭彦 教授
委員 田中 健次 教授
委員 田野 俊一 教授
委員 栗原 聡 教授
委員 田原 康之 准教授

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

Masayoshi Umeda

Abstract

We have recently seen the spread of large-scale distributed processing systems typified by Hadoop which is capable of managing and processing big data. With such systems, it is necessary to verify those systems (system verification) intended for commercial operation to ensure product quality within the period determined by the budget. If we verify these systems in the same manner as general information processing systems, we will be faced with the issue of there being an extremely long verification period.

However, with large-scale distributed processing systems, it is difficult to complete verification within the period estimated by the conventional method. To complete verification within a particular period, (1) the verification period can be reduced by streamlining the verification work, and (2) the trade-off between verification accuracy and verification period can be controlled by assigning importance to verification items.

This study analyzes the importance of verification work first, and clarifies the work that should be streamlined so that verification can be completed whithin the designated period. Next, “planning” the activities in the verification process and “development of test environment” reduce verification periods by rationalizing the work, and so the verification period can be estimated. In addition, the importance of verification items is analyzed during “test generation”, and these items are verified in order of importance during the verification period. This study proposes a way of controlling the trade-off between verification accuracy and the planned verification period.

For the planning activity, this study proposes three methods: (1) a method for stable and high-speed registration of the verification data by tuning network input/output (I/O) and disk I/O, (2) method of improving the time-estimation accuracy for executing verification when using large quantities of data resources, and (3) method of allocated machine partitioning and executing in stages verification items and those of non-dependent machine classification.

The first method is the stable and high-speed registration of verification data for storing large-scale data in advance. This study ascertains the bottlenecks to be the network and disk I/O. This study resolves that write speeds beyond the performance requirements with the network I/O

tuning is achieved by tuning a TCP buffer size 1.5x lager than the default size, achieved stable throughput is achieved with the disk I/O tuning Merge Compaction randomization.

The second method is the test-time estimation accuracy for a more accurate verification schedule. This study ascertains that a large gap to be using a large quantity of data resources. This study resolves this by taking common items and making an estimate by provisionally verifying in advance in a planned manner when drawing up the schedule.

The third method is the efficient allocation of verification machines for problem solving and handling. This study resolves the problem, the number of machines is partitioned and several stages of the verification environment. The verification enviroment are allocated from small scale to large scale depending on the verification item. And the verification is carried out in stages.

For the test-generation activity, this study determines the importance of verification items and verifies the items within a period.

(1) From the viewpoint of verification, we evaluate “resource efficiency”, “fault tolerability”, and “recoverability”, which are characteristics of large-scale distributed processing systems. Then, (2) feedback the problems extracted in the verification are then feed back to the items. The verification is performed in order of importance during the planned period. This method controls the trade-off between verification accuracy and verification period.

For the test-environment-development activity, this study proposes two methods: (1)a method of creating a framework for confirming performance and functionality in which read-write-data control and logs can be confirmed and (2)a method of reducing the impact on verification by securing standby devices and using sequential monitoring by using tools.

The first method is the creation of a benchmark tool for efficiently verifying work in a several-hundred-unit-scale machine environment. As a framework for confirming performance and functionality in detail, (1) a concentrated control mechanism using a test program on multiple units for improving work efficiency, (2) a mechanism in which the write data content can be flexibly changed using parameters, and (3) a mechanism that can measure throughput and confirm data content are used.

The second method is the establishment of detecting machine faults that would cause a problem for verification and a method of recovery that does not impact the verification in a several-hundred-unit-scale machine environment. From the actual machine faults, it was found to be valid to store 10% of the total number of machines as standby devices. This method is able to control verification, resolves detecting faults in machines affecting verification using tools such as Shell script and OSS Nagios, and replacing the faulty machines sequentially.

This study proposed these methods to resolve actual issues for the above three activities when verifying large-scale distributed processing systems. This study applied these proposed methods to the commercial development of collecting and searching large amounts of data on the Internet. As a result, by applying these methods, the verification could be controled, and executed within the verification period without delay. The verification is half as short when these methods are not applied.

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

概要

近年，BigData 処理の需要の拡大から，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．また，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おり，見積もった期間内で検証を完了する必要がある．

しかし，大規模分散処理システムでは，従来の方法で見積もった期間で，従来の方法で検証をすると検証作業が膨大になってしまうため，期間内で検証を完了することが難しい．限られた検証期間で検証を完了するためには，(1)検証作業の合理化による検証期間の削減，(2)検証項目の重要度付けによる検証精度と検証期間とのトレードオフのコントロール，の 2 点を行う必要がある．

本研究では，まず検証作業の重要度を分析し，期間内で完了するための効率化すべき作業を明確にする．次に効率化すべき検証作業で検証工程のアクティビティの「計画」と「テスト環境の開発」で作業の合理化による検証期間の削減を行い，見積もりで検証期間の計画を行う．また，「テストケースの生成」で検証項目の重要度を分析し，網羅的な項目を検証するのではなく，計画した期間で重要度の順に検証する．この検証精度と計画した検証期間とのトレードオフのコントロールと，これらの手法を利用する方法とを提案する．

計画のアクティビティでは，「I/O デバイスをチューニングした高速データ登録手法」，「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」，「検証マシン台数分割と段階的検証による検証実施手法」の 3 点の提案手法を示す．

1 点目の手法は，検証データとして必要なデータの質で必要なデータ量を準備する．データの準備の作業は，I/O デバイスをチューニングすることで実データのデータ量を変更する．具体的には，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，ネットワーク I/O の「TCP バッファのサイズを変更」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る．このデータ量の変更により，大量の実データを安定して高速登録することができ，検証精度を下げずに期間を減少できる．

2 点目の手法は，全ての項目を調査することなく検証期間を見積もる．見積もりは，データを大量に使う項目の検証時間を調査することで，既存で要する見積時間で精度を調整する．具体的は，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，選別した項目の実施時間を測定することを提案する．この見積もり精度の調整により，予測精度の向上した検証計画が作成できる．

3 点目の手法は，マシン台数を全て使って検証した精度で検証を加速する．検証の加速は，マシン台数を分割し検証項目に割り当てることで調整する．具体的には，バグを効率的に摘出するマシンの分割の仕方を突き止め，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する．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することを提案する．この検証作業の調整により，発生した問題の解析と対処を迅速化することができ，検証精度を下げずに期間を減少できる．

テストケース生成のアクティビティでは，網羅的な項目を検証するのではなく，検証項目の重要度を決め，重要度に合わせた項目数で期間内に検証する「システム特徴と問題発生傾向による項目削減と重要度による期間内検証手法」の提案手法を示す．具体的には，(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」を評価する．そして，(2)検証で摘出された問題を項目にフィードバックする．これら重要な観点以外で重要度の低い，運用では利用しない処理の項目を削減し，計画した期間で重要度の順に検証する．これにより，検証精度と検証期間とのトレードオフをコントロールする．

テスト環境の開発のアクティビティでは，「データ生成とログの出力を組み込んだテストドライバ(TP)の作成手法」，「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」の 2 点の提案手法を示す．

1 点目の手法は，検証作業を手動の確認ポイントを押さえ自動化して作業をする．自動化の作業は，データ生成とログの出力を組み込んだテストドライバを作成することで検証作業を制御する．具体的には，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うために，プログラムの機能とデータの読み書きの性能を確認できる仕組みとして，(1)複数台の TP(Test Program)を集中制御できる仕組み，(2)書き込みデータの圧縮率等を考慮して外部パラメータで任意に内容を変更できる仕組み，(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る．この検証作業の制御により，効率的に検証作業ができ，検証精度を下げずに期間を減少できる．

2 点目の手法は，テスト環境の迅速な正常化により検証精度を保つ．テスト環境の迅速な正常化は，監視ツールで故障を発見し，一定数の予備機を保有して入れ替えることで，故障対処の作業を操作する．具体的には，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う．そして，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る．この故障対処の作業の操作により，検証のやり直しなどの検証への影響を最小限にでき，検証精度を下げずに期間を減少できる．

以上のように，大規模分散処理システムの検証において，主要な課題を抽出し，解決手法をそ

れぞれ提案する. これらの提案手法を, インターネット上の大量データを収集し, 検索する商用の開発に適用した. その結果, これらの提案手法により, 検証期間の削減が実現し, 検証精度と検証期間のトレードオフをコントロールすることで, 限られた検証期間で検証を完了し, 検証期間を従来の 1/2 以下に短縮できた.

目 次

図 目次

表 目次

1. はじめに

1.1 背景・目的

様々なデータ形式の BigData を活用し，データ量の増大に合わせてマシン台数をスケールアウトさせるサービスをする場合，大規模分散処理の技術が必要となる[1][2][3]. 大規模分散処理システム は，(1)多数のサーバを繋げて一つの大きな処理システムとすること，(2)並列分散処理により大量のデータの格納・処理を最適化すること を特徴としている[4]. 大規模分散処理システムが適用されているクラウドサービスの市場規模は，2017 年には 1,640 億ドルであったが，2020 年には 3,047 億ドルに拡大すると予想されている[5].

大規模分散処理システムで運用されているサービスの一例として，インターネットの検索サービスがある. インターネットの検索サービスは，システムに格納されている大規模データを処理することにより，キーワードによる検索や付加価値情報を創出するサービスである. 検索サービスの利用者数は，2002 年の 1,646 万人から 2008 年には 4,775 万人に伸びている[6]. 検索サービスの具体例としては，Web コンテンツを収集してコンテンツの検索を提供するサービスや，購入履歴の情報などを分析してユーザの嗜好に合致した情報を検索結果の付加情報とするレコメンデーションサービス等がある. これらのサービスで取り扱うWebコンテンツやログ情報は，数十 TB（テラバイト）～数百 TB，場合によってはそれ以上の規模になる. このため，数台のマシンにより構成されるシステム（以降，従来の情報システムと呼ぶ）ではなく，多数の PC サーバを用いて大規模データを分散処理する大規模分散処理システムが利用されている.

大規模分散処理の技術への関心の高まりから，OSS（オープン・ソース・ソフトウェア）のコミュニティとして大規模分散処理システムを開発する動きがある. 著名な例としては Apache プロジェクトの Hadoop[7][8][9][10]がある. 筆者は，Common IT Base over Cloud Computing（CBoC）タイプ 2 システム[12]という大規模分散処理システムの開発を進めてきた. この CBoC タイプ 2 システムは，携帯端末で利用される商用の検索サービスのシステムとして運用されている.

商用開発におけるシステム検証は，開発を計画する前の段階で，投資する予算が決定しており，さらにサービスを開始する日取りからさかのぼって，これまでの開発経験から期間が決定している. そのため，検証作業のスケジュールは，これまでの開発経験からの期間と決定している費用とに応じた要員のもとに，従来の情報システムにおける方法で，システムの開発規模や開発工数から算出した期間で計画する. 検証作業は計画した期間内で完了する必要がある. しかし，これまで大規模分散処理システムにおいて，プロダクト全体を連携して動作し，要求仕様を確認する検証プロセスを扱う事例研究は公知となっていない. このため，計画した期間内で完了できない問題を整理し，解決策を導き出すことが重要である.

商用開発では，不確実な方法をできるだけ排除し実績のある方法が採用される. そのため，大規模分散処理システムにおける検証の事例がなく適した方法が不明な場合は，事例の多い従来の情報システムにおける方法（1.2.3 項参照）で検証せざるを得ない. そのため，大規模分散処理システムを検証する際，従来の情報システムにおける検証方法で検証をすると，1.2 節で詳述する

ように計画した検証期間以上の期間を要してしまうという問題がある.

このように大規模分散処理システムの検証は，従来の方法で見積もり計画した期間で，従来の方法で検証をすると検証作業が膨大になってしまうため，期間内で検証作業を完了することが難しい. このような背景から，本研究では，(1)検証作業の合理化による検証期間の削減，(2)検証項目の重要度付けによる検証の妥当性や正確性（以降，検証精度と呼ぶ）と検証期間とのトレードオフのコントロール，の 2 点を行い，計画した期間で作業を完了することを目的とする.

1.2　大規模分散処理システムにおける検証の課題

システム検証は以降に詳述するように，プロダクト全体を連携して動作し，要求仕様を確認する検証である．大規模分散処理システムにおいて，この検証の実例は少なく，かつ 2.3 節で後述するように，(1)検証作業の合理化による検証期間の削減，(2)検証項目の重要度付けによる検証精度と検証期間とのトレードオフのコントロール，という検証の効率化に関する文献は，これまで存在していない．そのため，実際には従来の情報システムにおける検証の方法を適用する．しかし，従来手法で大規模分散処理システムの検証を行うと，計画した以上の期間を要してしまう問題が発生する．本節では，従来のシステム検証における作業概要を示した後，大規模分散処理システムにおけるシステム検証の主要な問題とその具体的な課題を議論する．

1.2.1　　システム検証について

検証には後述する 2.3 節のように，形式言語で仕様や設計を記述してプログラムの検証をする静的検証と，プログラムを実行してその結果から仕様の確認を行う動的検証とがある．本論文での検証は，運用を想定してプログラムの動作確認をすることから，実際にプログラムを実行して検証する動的検証を指す．以降に，大規模分散処理システムのプログラムを実際に実行して検証するシステム検証について示す．

以下の図 1 に示すとおり，一般的なソフトウェア開発における開発工程と検証工程との対応関係を V 字型に整理した．各検証工程での検証単位と品質達成レベルを以下の表 1 に示す．システム検証は，表に示すように V 字モデル[13]の検証工程の最後の工程であり，総合検証や要求検証とも呼ばれる．そのため，システム検証は，要求仕様を確認するプロダクト全体を連携した最終的な検証であり，実運用時に発生する問題を事前に改修するという意味において重要である．

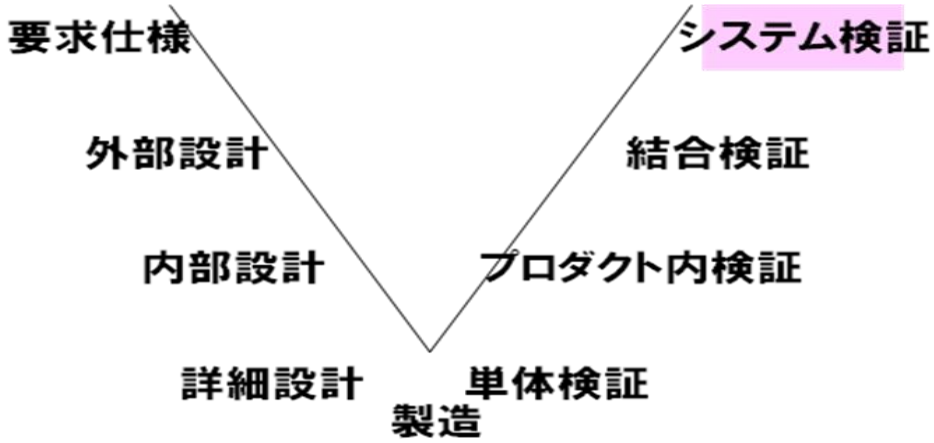


図 1. V 字モデルにおけるシステム検証

表 1. 各試験工程でのシステム検証のレベル

検証工程	検証単位	品質達成レベル	抽出される不具合例
単体検証	サブシステム(各プロダクトを構成するシステム)内の単体レベルの品質を確保する	C0/C1 100%	単体ロジック誤り, コーディング誤り
プロダクト内検証	サブシステム間の検証によりプロダクト単体の品質を確保する	機能・性能の確保	プロダクト内ロジック誤り, 単体性能劣化, 詳細設計誤り
結合検証	プロダクト間をまたがったプロダクト単体の品質を確保する	機能・性能の確保	プロダクト間ロジック誤り, IF 誤り, 基本/機能設計誤り
システム検証	プロダクト全体を連携したシステムの品質の確保する	実運用上のシステム安定性確認	ユースケースのロジック誤り, 性能劣化, リソース不足, 仕様誤り

1.2.2　検証工程のアクティビティ

本論文では, 検証の作業工程をアクティビティの単位として分類する. 検証アクティビティは IEEE のソフトウェア エンジニアリング基礎知識体系の SWEBOK[14]にある, ソフトウェアテスティングに規定されている分類を用いる. その分類を用いて大規模分散処理システムにおける検証の課題を整理する. 表 2 に SWEBOK の検証アクティビティを示す.

計画アクティビティの作業は, 人員調整, 利用可能な検証環境に対するマネジメント, および予想される不良な実行効果が得られたときの対策を行う. プロジェクトの計画を立案してプロジェクトのスコープと期間と工数や費用などのリソースを決定するという, 工程全体に関わる作業であり, プロジェクトマネジメント[15]では最も重要な作業である.

テストケース生成アクティビティは, 実施されるテストレベルおよび用いられるテスト技法に基づいた, 検証項目の作成を行う. 作成する検証項目は, 品質を確保可能な検証の内容であり, かつ検証期間内に実施可能な数である必要がある. 検証項目の作成は, 検証が開始される前に完成している必要があり, スケジュールの厳守が求められる作業である

テスト環境の開発アクティビティは, テスト環境を構築する作業であり, ソフトエンジニアリングツールとの互換性を保ち, 検証項目の生成および制御を容易にする. 同時に, 期待された結果, 記述およびその他の検証のための材料を記録し, 再現できる検証環境の開発を行う. テストケース生成と同様に検証における上流の作業工程であり, スケジュールの厳守が求められる作業である.

実行アクティビティの作業は, 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の明確に定義された版を用いた検証の実施を行う. いわゆる検証の手順書に従って検証することである.

テスト結果の評価アクティビティの作業は, 検証の結果が手順書に記載された仕様や設計書の内容に合致し合格しているかを確認することである. 結果が合格であるということは, ソフトウェアが

仕様どおりに動作し，仕様から外れた実行結果が発生しなかったことを意味する．検証の結果は品質の基礎データとなるため，必要があれば結果の再評価を実施する．例えば，仕様から外れた実行結果が直ちに問題につながるわけではなく，単純な誤差と判定されることもあるため，仕様の再確認が必要な場合がある．

問題報告／テストログアクティビティの作業は，検証ログより，いつ検証が実施されたか，だれが実施したか，どのようなシステム構成をベースにして行ったか，およびその他関連する識別情報が分かるようにし，予期しない，または不正な結果を問題報告システムに記録する．検証した結果を確認できるようにシステムに記録し，摘出した問題は問題報告書としてシステムに記録することである．

欠陥追跡アクティビティの作業は，欠陥に対する分析を行い，いつそれらがソフトウェア内に入り込んだか，どのエラーが欠陥を生じる原因になったか，およびいつそれらがソフトウェアの中で発見されたか，などの決定をし，どの時点で改善が必要かの追跡をすることである．

表 2. 検証アクティビティ

アクティビティ	内容
計画	人員調整，利用可能な検証環境に対するマネジメント，および予想される不良な実行効果が得られたときの対策．
テストケース生成	実施されるテストレベルおよび用いられるテスト技法に基づいた，検証項目の生成．
テスト環境の開発	ソフトエンジニアリングツールとの互換性を保ち，検証項目の生成および制御を容易に行えるようなものであると同時に，期待された結果，記述およびその他の検証のための材料を記録し，再現できる検証環境の開発．
実行	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の明確に定義された版を用いた検証の実施．
テスト結果の評価	検証が成功したか，検証の結果の評価．
問題報告/テストログ	検証ログより，いつ検証が実施されたか，だれが実施したか，どのようなシステム構成をベースにして行ったか，およびその他関連する識別情報が分かるようにし，予期しない，または不正な結果は問題報告システムに記録．
欠陥追跡	欠陥に対する分析を行い，いつそれらがソフトウェア内に入り込んだか，どのエラーが欠陥を生じる原因になったか，およびいつそれらがソフトウェアの中で発見されたか，などの決定．

以上のように，検証のアクティビティにおいて，特に計画アクティビティ，テストケース生成アクティビティ，テスト環境の開発アクティビティは，重要な作業であることが分かる．

1.2.3 従来の情報システムにおける検証

従来の情報システムにおける検証の作業概要を表 3 に示す．表 3 の作業概要は，以降でアクティビティの単位毎に説明する．なお，それぞれのアクティビティ単位の作業で使用される技術の詳細は，2.3 節で詳述する．

表 3. 従来の情報システムにおける検証の作業概要

	アクティビティ	作業概要
1	計画	要求仕様から検証の内容を決定し計画を立案
2	テストケース生成	網羅的に検証項目を抽出し，同値分割，境界値分析等により項目数の精査をして項目を完成
3	テスト環境の開発	検証に必要な検証の作業環境を手作業で準備 ツールによる効率化は，すぐに利用可能なOSSやShell scriptを使う
4	実行	手作業やツールでプログラムを実行し，コンソールに表示されるメッセージやログを逐次確認
5	テスト結果の評価	実行結果をログから分析し確認
6	問題報告/テストログ	テストログを手動やツールで収集し問題のある部分を有識者が確認
7	欠陥追跡	検証結果の分析は品質報告で実施

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[14]．

テストケース生成のアクティビティでは，検証項目は，以降に記述する技術を用いて，1.2.1 項に記述した検証のレベルで作成される．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[41][42][43]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[42]．

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[14].

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[14][45][46][47][48]．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[43]．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[43]．静的検証では形式手法による検証方法[34][35][36]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[14].

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[14].

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[14].

上記のように，従来の情報システムの検証では，ソフトウェアの品質を確保するために，各アクティビティで漏れなく網羅的に検証する技術はあるが，効率化に関する技術はない．

1.2.4　大規模分散処理システムにおける検証の主要な問題と具体的な課題

商用の開発において，従来の情報システムにおける検証の方法で，大規模分散処理システムの検証期間を計画し，検証すると予想外の問題が発生し，計画した以上の期間を要してしまう問題がある．その主要な要因を全て抽出したところ，82 項目の問題が抽出された．これらの 82 項目から数週間（2 週間以上）の計画より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する．2 週間以上を抽出する理由は，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を抽出する理由は，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[15]．問題を整理することで，効率化すべき主要な課題 6 つを抽出する．

課題 6 つを解決できると，検証を効率化することができ，検証時間の正確な見積もりができるため，検証期間内に完了することができると考える．主要な問題と具体的な課題をそれぞれアクティビティで整理した結果を以下の表 4 に示す．

表 4. システム検証で発生した問題と具体的な課題

アクティビティ名	問題	具体的な課題
計画	(1)検証データが 100 テラバイト～ペタバイト級のため，検証前のデータ登録が長期化し，検証開始が 2 週間以上遅延する	課題 1:大量な検証データの早期データ準備 (3.3 節で記述)
	(2)検証項目の実施時間を見積もる精度が悪く，見積もりで計画より 2 週間以上の期間を要する	課題 2:検証実施時間の見積もり精度向上(3.4 節で記述)
	(3)全てのマシンを利用すると，問題の解析の間はシステム検証が止まり，計画より 2 週間以上の検証期間を要する	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(3.5 節で記述)
テストケース生成	(4)網羅的に作成された検証項目は項目数が多くなり，計画より 2 週間以上の検証期間を要する	課題 4:大量な検証項目の項目削減と期間内の検証完了(4.3 節で記述)
テスト環境の開発	(5)検証で利用する大規模のマシンに対して読み書きの性能確認や機能確認が容易にできない	課題 5:性能確認や機能確認の効率化(5.3 節で記述)
	(6)通信障害などのマシン故障の状態で検証をしてしまうため，機能確認や性能測定は再実施しなければならない	課題 6:故障検知と故障対応の迅速化(5.4 節で記述)
実行	(前アクティビティである「計画」，「テストケース生成」，「テスト環境の開発」の対策で問題解消)	(前工程の対策で課題解消)
テスト結果の評価		
問題報告/テストログ		
欠陥追跡		

計画アクティビティでは，(1)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 2 週間以上遅延する問題，(2)検証項目の実施時間を見積もる精度が低いため，商用の開発では計画した期間より 2 週間以上の期間を要する問題，(3)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い，計画より 2 週間以上の検証期間を要する問題が発生

した．本研究では，こうした従来の検証方法で効率化しきれない問題を整理し，検証期間の削減が必要な 3 つの課題「大量な検証データの高速データ登録」（課題 1），「検証実施時間の見積もり精度向上」（課題 2），「効率的にバグ摘出するマシン台数分割による検証実施」（課題 3）を取り上げる．

テストケース生成のアクティビティでは，(4)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ため，商用の開発で計画より 2 週間以上の検証期間を要する問題が発生した．本研究では，従来の検証方法で効率化しきれない問題を整理し，検証精度と検証期間とのトレードオフのコントロールが必要な課題「大量な検証項目の項目削減と期間内の検証完了」（課題 4）を取り上げる．

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，故障時の検証は正確ではないため，再検証が必要とな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，問題を整理し検証期間の削減が必要な 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）を取り上げる．

実行アクティビティより後のアクティビティでは，計画アクティビティ，テストケース生成アクティビティ，テスト環境の開発アクティビティの問題が解消されることにより，効率化が必要な問題は発生しない．

上記のように，商用の開発で大規模分散処理システムにおける検証の効率化すべき主要な問題と具体的な課題とが整理された．

1.3 提案手法

本節では，大規模分散処理システムにおける検証（システム検証）において，「計画」，「テストケース生成」および「テスト環境の開発」のアクティビティで発生した課題を解決する提案手法について，それぞれ概要を述べる．

1.3.1 計画アクティビティにおける提案手法

計画のアクティビティにおいて，大規模分散処理システムの検証では，1.2 節に示したように 3 点の課題「大量な検証データの早期データ準備」（課題 1），「検証実施時間の見積もり精度向上」（課題 2），「効率的にバグ摘出するマシン台数分割による検証実施」（課題 3）を取り上げる．これらの課題に対する対策として，それぞれ「I/O デバイスをチューニングした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，検証期間の削減ができ，課題を解決できる．

手法 1 の「I/O デバイスをチューニングした高速データ登録手法」は，事前に実施する大量データ蓄積を加速するための検証データの登録が，安定的かつ高速に登録ができない問題を解決する．データ書き込みの性能要件は要求仕様で規定されている．システムはその要件のとおりに作られているため，運用時の環境におけるデータ登録は，最速でもその性能要件で必要な時間を要する．性能要件を超える高速なデータ登録は，ネットワーク I/O に起因する高速登録のボトルネックと，ディスク I/O に起因し処理が集中することにより，性能が不安定になるため不可能であった．事前に実施するデータ登録は，検証準備において一時的に必要な処理である．そのため，プロダクトのプログラム修正と比べ，一時的なデータ登録後に元に戻すことが容易なパラメータ設定の変更による I/O チューニングの手法が望ましい．パラメータ設定は，ボトルネックとなる I/O のネットワーク I/O とディスク I/O に着目して，段階的な調整を行いながら最適なチューニング値を見つける．チューニングはシステムのログやネットワーク監視の結果や性能値を確認して，性能劣化が生じないように実施する．提案する解決手法では，(1)ネットワーク I/O のチューニングは TCP バッファのサイズを 1.5 倍に変更する，(2)ディスク I/O のチューニングはデータ登録時にバッファリングされたデータが，ディスクに書き出される処理を同一時刻ではなくランダム化し，負荷分散させる．

手法 2 の「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」は，見積もりの精度が低いため，見積もられた検証実施時間と実際の検証実施時間が大きく乖離してしまう問題を解決する．特に大量にデータがあり，ディスクアクセス時間や処理時間がかかる検証項目は，机上で計算されたものと乖離する．具体的な検証項目の例として，大量データの書き込み／読み出し項目や，データのリカバリの項目は，従来の情報システムにおける見積もりでは，精度の高い見積もりを行うことができない．この問題を解決するために，大量にデータを使用する検証項目に着目する．これらの項目は，項目全体の約 1/3 程度の項目数を占めており，スケジュール全体に影響を与える．これらの項目から項目数を絞って事前検証を行い，検証時間から見積もりを行う方法で課題を解決

する. 検証項目数の絞り方は, 類似の項目が2項目以上ある場合, そのうちの1項目を検証項目とする. そして, その検証項目をスケジュール作成時に事前に検証を実施する. 提案する解決手法では, 大量にデータを使用する項目で, 複数項目ある中から代表とした 1 項目については, スケジュール作成時に事前の検証を実施し, 大量にデータを使用しない項目は, 従来の見積もり手法とする.

手法3の「検証マシン台数分割と段階的検証による検証手法」は, 発生した問題の解析と対処を効率化するため, サービスイン時のマシン台数が利用可能なシステム検証において, どのようにマシン台数を複数に分けて検証環境にするのかと, どのように検証環境を検証項目に割り当て検証を実施するかの問題を解決する. 従来の情報システムにおけるシステム検証では, 検証項目に運用条件である運用時のマシン台数を割り当てる. 大規模分散処理システムの検証で, 従来のように検証項目に運用時の最大のマシン台数を割り当てると, 余分なマシンはないため検証と並行してマシンを使った問題の解析はできない. そうすると, 発生した問題の解析時には, 検証を止める必要があり, その間は検証の進捗がなく時間を要してしまうことになる. 検証項目に運用時の最大のマシン台数を割り当てるのは, システム検証を運用時のマシン台数で検証するという前提だけではなく, バグを効率的に摘出するマシンの分割の仕方と, 検証項目への割り当て方法が分からないからである. 検証環境として商用のマシン台数しか持たない場合, 提案する解決手法では, 検証マシン台数の分割と, マシン台数が少ないものから多いものまで数段階の検証環境を検証項目に応じて割り当てて検証を実施する.

以上のように計画のアクティビティでは, 上記 3 つの解決手法を適用することで, 検証期間の削減ができ, 課題を解決できる. その結果, 事前にリスクを軽減し, 精度のよい見積もりで計画したスケジュールで検証することができる. これらの課題と解決手法については, 本論文の 3 章において詳述する.

1.3.2 テストケース生成アクティビティにおける提案手法

テストケース生成のアクティビティにおいて, 1.2 節に示したように 1 点の課題「大量な検証項目の項目削減と期間内の検証完了」(課題 4)を取り上げる. この課題の対策として「システム特徴と問題発生傾向による項目削減と重要度による期間内検証手法」を提案する. この手法を適用することにより, 検証精度と検証期間とのトレードオフのコントロールができ, 課題を解決できる.

手法 4 の「システム特徴と問題発生傾向による項目削減と重要度による期間内検証手法」は, 検証する項目数が多く, その全てを検証期間内に実施できない問題を解決する. 従来の情報システムにおけるシステム検証では, 検証条件を利用して精査する同値分析や境界値分析等により, 項目数の絞り込みを行う. しかし, 大規模分散処理システムの検証においては, 網羅的に抽出する検証項目数が膨大な量になるため, 従来の情報システムにおける方法では検証期間内に実施可能な項目数にまで削減することができない. この問題を解決するために, 網羅的な項目を検証するのではなく, 検証項目の重要度を決め, 重要度に合わせた項目数で期間内に検証する「システム特徴と問題発生傾向による項目削減と重要度による期間内検証手法」を適用する. 具体的には,

(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」を評価する. そして, (2)検証で摘出された問題を項目にフィードバックする. これら重要な観点以外で重要度の低い, 運用では利用しない処理の項目を削減し, 計画した期間で重要度の順に検証する.

以上のようにテストケース生成のアクティビティでは, 上記の手法を適用することで, 検証精度と検証期間とのトレードオフのコントロールができ, 課題を解決できる. その結果, 品質を確保したまま効率的な検証と期間内での検証を完了することができる. 上記の課題と解決手法については, 本論文の 4 章において詳述する.

1.3.3 テスト環境の開発アクティビティにおける提案手法

テスト環境の開発アクティビティにおいて, 1.2 節に示した 2 点の課題「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)を取り上げる. 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テストドライバ(TP)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, 検証期間の削減ができ, 課題を解決できる.

手法 5 の「データ生成とログの出力を組み込んだテストドライバ(TP)の作成手法」は, 数百台規模のマシン環境を利用し, 分散されているプログラムの性能を計測する際に, 検証条件の変更や結果の集計のための機能を決定する. OSS で利用可能な大規模分散処理システム向けのベンチマークツールは, 性能を比較する機能はあるものの機能確認や問題解析の機能は含まれていない. そのため, 機能確認や問題解析の検証で必要となる, 書き込む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の条件を細かく変化させた測定ができない. この問題を解決するために提案する手法では, 独自ベンチマークツールである TP で, AP の処理を模擬しプログラムの機能とデータの読み書きの性能を確認するために, (1)複数台の TP を集中制御, (2)書き込みデータの内容をパラメータで任意に変更, (3)スループットの計測やデータ内容を確認という 3 つの仕組みをツールに具備する. (1)は, TP の数が数百に及ぶ場合には一つ一つ手作業で起動や管理することは時間がかかるため, 起動や停止を集中管理するコントロールサーバ上の TP の一括起動停止ツールを動作させる. これにより, 各 TP を一括管理し, 設定変更もコントロールサーバで実施する. また, AP の処理と同じに書き込みや読み出しの TP の起動や終了タイミングを合わせることで, AP の模擬が可能となる. (2)は, データの内容を固定ではなく変更できることにより, 圧縮率などデータの内容によるシステムの性能への影響を確認する. また AP のデータ内容を模擬することで, 実際のシステムを模擬した機能確認が可能となる. (3)は, 各 TP のログから結果を集計する仕組みを持ち, 数百台規模のマシンから時刻を合わせて集計した結果を一箇所で確認する.

手法 6 の「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」は, 数百台のマシンの故障を検知して検証に影響を与えないよう回復するという問題を解決する. 従来の情報システムにおける故障検知と故障対応は, 数百台規模の検証環境では故障の頻度が高いにも関わらず, リ

アクティブなものであった. 特に性能検証では, メモリ故障やディスクの部分故障, さらにはサイレント故障など, 大規模分散システムの故障検知で切り離しがされない場合は, 検証結果に影響を及ぼす. この問題を解決するために提案する手法では, 監視ツールによる故障監視と, 予備マシンの用意による故障マシンの入れ替えとで対応する. 故障率がこれまでの運用の記録からある程度範囲が分かるため, 予備マシンは故障率を元に想定した故障マシン数と同程度を準備する. また, 通常のマシン監視に加え, それだけでは発見できない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をする.

以上のようにテスト環境の開発のアクティビティでは, 上記 2 つの解決手法を適用することで, 検証期間の削減ができ, 検証を加速することができる. これらの抽出した課題と解決手法については, 本論文の 5 章において詳述する.

以上, 本研究では, 大規模分散処理システムの検証について, 作業を効率化すべき主要な課題を抽出・整理し, 解決するための手法を提案する. 解決手法については, 実際の開発に適用することで有効性が確認でき, 汎用的な解決手法としての可能性が高まる.

1.4　本論文の構成

本論文の構成を図 2 に示す. 本論文は, 商用の大規模分散処理システムの検証における課題を抽出･整理し, 解決策を導き出したものである. 本論文の構成を以下に示す. 第 2 章では, 大規模分散処理システムとシステム検証の定義を示し, 本研究の位置づけの説明を行う. 第 3,4,5 章では, 第 1 章で示した課題に対する, 商用開発のシステム検証において有効性を確認した実践的な解決手法を提案する. 第 6 章では, 第 3,4,5 章で示した課題の解決手法について考察をし, 第 7 章で結論と今後の課題を示す.

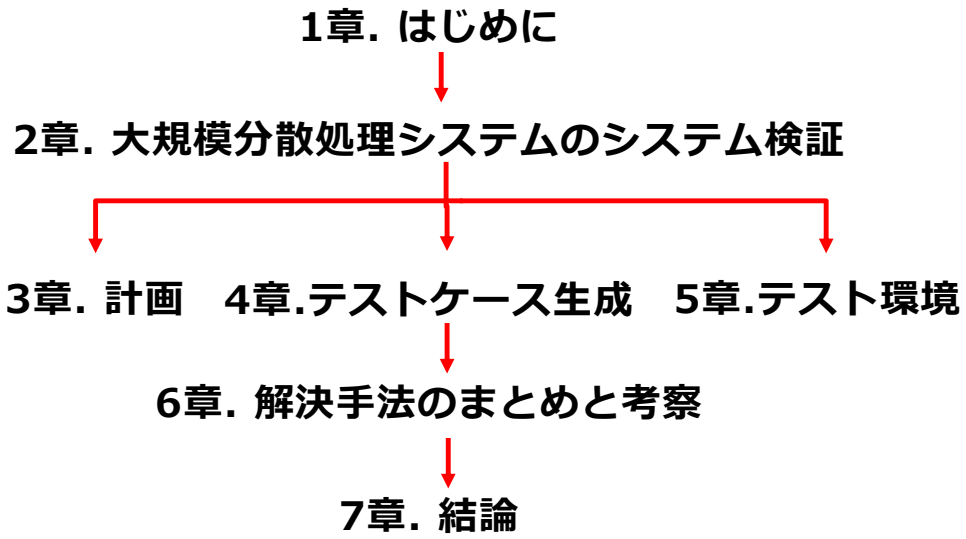


図 2. 本論文の構成

2. 大規模分散処理システムにおける検証

本章では，本論文でシステム検証の対象となるシステム，その処理の概要について述べ，大規模分散処理システムにおける検証の効率化に関する先行研究のレビューを行う．

2.1 大規模分散処理システムの概要

本節では，本研究の対象となるシステムの概要として，システムの特徴とサービス例と運用例を示す．併せて具体的にプロダクトの構成を示し，状態遷移と主要な処理とデータ量の効率化処理を示す．

対象となるシステムは，図 3 のように汎用サーバを大量に並べることで，100 テラバイト～ペタバイトクラスの大規模なデータ蓄積，データ処理を可能にする大規模分散処理システムである．システムの特徴は，サーバ数を動的に変化させることで，要求されるデータ量や処理量をスケーラブルに変化することができる「スケールアウト性（分散かつ大規模）」を備えていることである．加えて，大量のコモディティサーバを扱うことに伴い頻発する，サーバやネットワークの故障に対する「耐障害性（可用性）」を備えており，故障を前提としたシステム構成となっていることである．

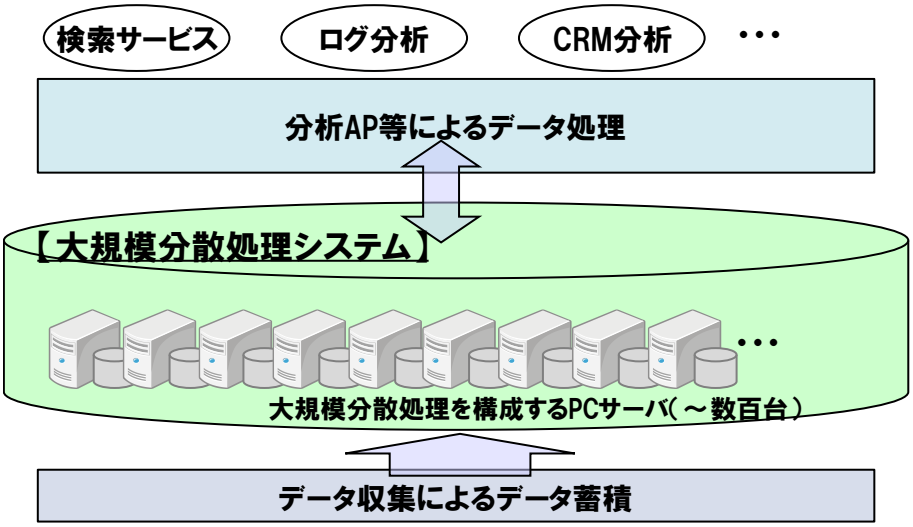


図 3. 大規模分散処理システム

大規模分散処理システムは，大量のデータを蓄積，処理するため，NoSQL[11]である KVS(Key Value Store)でデータ格納・管理をしている．KVS は，SQL の RDBMS とは異なり，データの蓄積と処理を分散環境で実行可能である点（「分散かつ大規模」）に特徴があり，分散環境において可用性を実現したデータストレージである．大規模分散処理システムと一般的な情報システムである RDBMS について，データ量とスケールアウトにおける位置づけを図 4 に示す．このように大規模分散処理システムは，RDBMS と比較して扱うマシン台数とデータ量が多い（「分散かつ大規模」）という特徴がある．（付録 1，付録 2 参照）「スケールアウト性（分散かつ大規模）」については，以下で具体的に説明する．

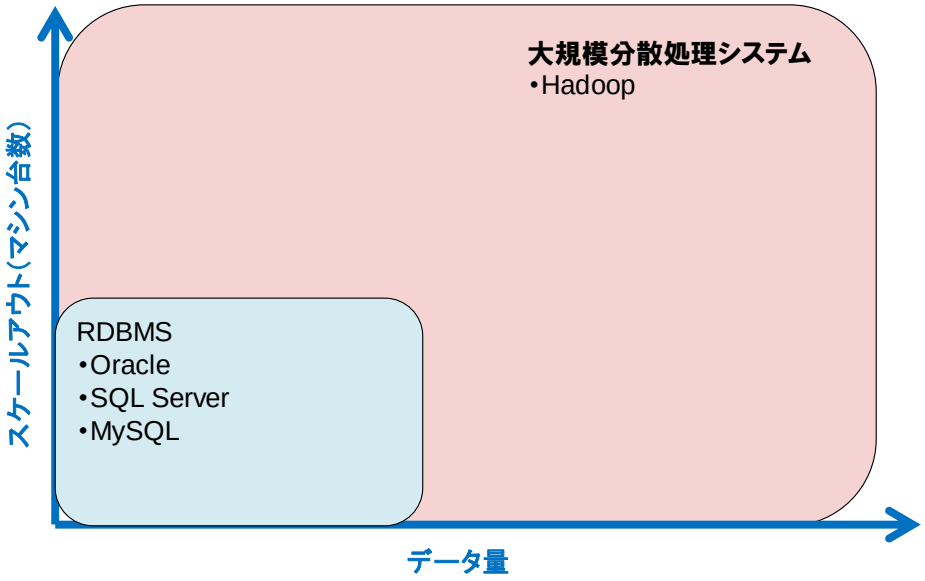


図 4. データ量とスケールアウトにおけるシステムの位置づけ

大規模分散処理システムの「スケールアウト性（分散かつ大規模）」は，以下に示す 2 点の特徴がある[4]．

(1) 複数のサーバを束ねて一つの大きな処理システムとする
マスターサーバで全てのサーバを束ねて管理することで，利用者は1台の大きなサーバとして利用することができる．また，サーバの容量が不足した場合は，利用者に意識させずにサーバを追加しスケールアウトできる．

(2) 並列分散処理により大量のデータの格納・処理に最適化
例えば，HDD は 200MB/sec で読み出しができるが（7,200rpm の HDD），5TB のデータを読み出すと約 7 時間（5*1000*1000/200 = 25000 秒 = 6.9 時間）を要する．そこで，データを複数のサーバに分割して格納し，処理時には複数のサーバからそれぞれデータを読み出す並列分散処理を活用（水平分散による負荷分散）する．サーバ 1000 台で処理すると，200GB/sec

の読み出しのスループットとなり，5TBのデータを読み出す時間は，25秒と高速化できる．このように大量のデータを格納・処理できるが，大量のデータを読み出して変更するには，格納されたデータを操作しなければならない．そのため，データのリアルタイムな処理よりは，バッチ処理に適している．

以上のような特徴から，大規模分散処理システムは，図 5 のようにデータサイズが多く，バッチ処理に適したシステムである．大規模分散処理システムを活用したサービスの例としては，以下のようなサービスがある．

- インターネット上のWebデータを蓄積して，キーワードによりWebページ内のテキストデータを検索する
- 過去のアクセス履歴を格納・処理して，ユーザ毎の嗜好（特徴量）を抽出し，コンテンツ最適化やレコメンドを行う
- オンラインゲームサービスにおいて，蓄積されたユーザ行動ログを分析し，解約低減を図る
- 工場の機械，自動車，船舶に付けたセンサーからのデータを格納・処理し，故障検知を行う

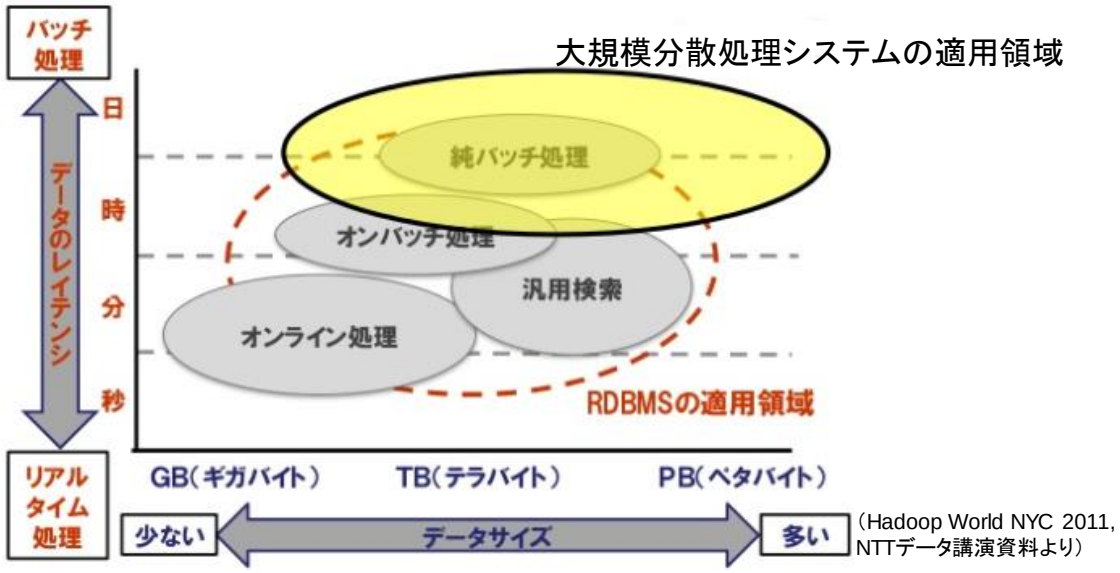


図 5. 大規模分散処理システムの適用領域

2.1.1 大規模分散処理システムの商用サービス例

対象とした商用のサービスの適用イメージを示す. 検索サービスのシステムである CBoC タイプ 2 は, 大規模で多種多様なデータの加工・分析を目的として, 数百台の PC サーバで連続に安定して大規模データを分散し蓄積・処理をする.

具体的なサービスの処理は図 6 に示すとおり, クローラーが Web サイトからのデータを収集して CBoC タイプ 2 に書き込む. クローラーは, 定期的に Web データを収集する. そのため, クローラーは CBoC タイプ 2 にランダムライト(不規則な書き込み)を行う. そして, データ分析が CBoC タイプ 2 からデータを読み出して, 検索エンジンに必要な検索インデックスを作成する. データを読み出す際に, 検索要求に対する内部の処理を行うアプリケーションとして, 収集データ編成ツール(MapReduce)とスマートキャッシュがある. 収集データ編成ツールは, 書き込まれたデータをシーケンシャルリード(順次読み出し)で CBoC タイプ 2 から読み出す. スマートキャッシュは, 検索時のランダムリード(不規則な読み出し)でキャッシュとして利用する. データ分析は, 収集データ編成ツールによって CBoC タイプ 2 からシーケンシャルリードによるデータにより検索インデックスを作成したり, テキストの内容(分析 AP)や画像などのマルチメディアデータ(マルチメディア分析)を分析して結果をランダムライトで書き込んだりする. ユーザからのデータの検索問い合わせ時には, 検索エンジンがインデックスを利用し蓄積されたデータから検索をして, 結果をユーザに返却する. システム検証では, 上記に示した処理を検証項目として作成し機能要件を確認する.

これらのユースケースを表 5 に, 処理のタイミングを図 7 示す. ユースケースにおけるランダムライト, シーケンシャルリード, ランダムリードは, ワーカにあるデータに対してランダム(不規則)にアクセスするか, シーケンシャル(順次)にアクセスするかということである. 各 AP は, それぞれのタイミングで動作しているため, タイミングにより同時に動作する. また, それぞれの AP はマシンの集約により同一マシンで動作する設計が可能である. 同一マシンで動作する場合, 各 AP が同時に動作することを考慮し, マシンのメモリやネットワーク(NW)を互いに共有して動作する必要がある. そのため, キャッシュメモリの使用量やネットワークのバッファの上限値を決めて, 互いの動作に影響が及ばないようにしている. もし大量のデータを書き込む場合は, バッファの上限値を変更しなければならない.

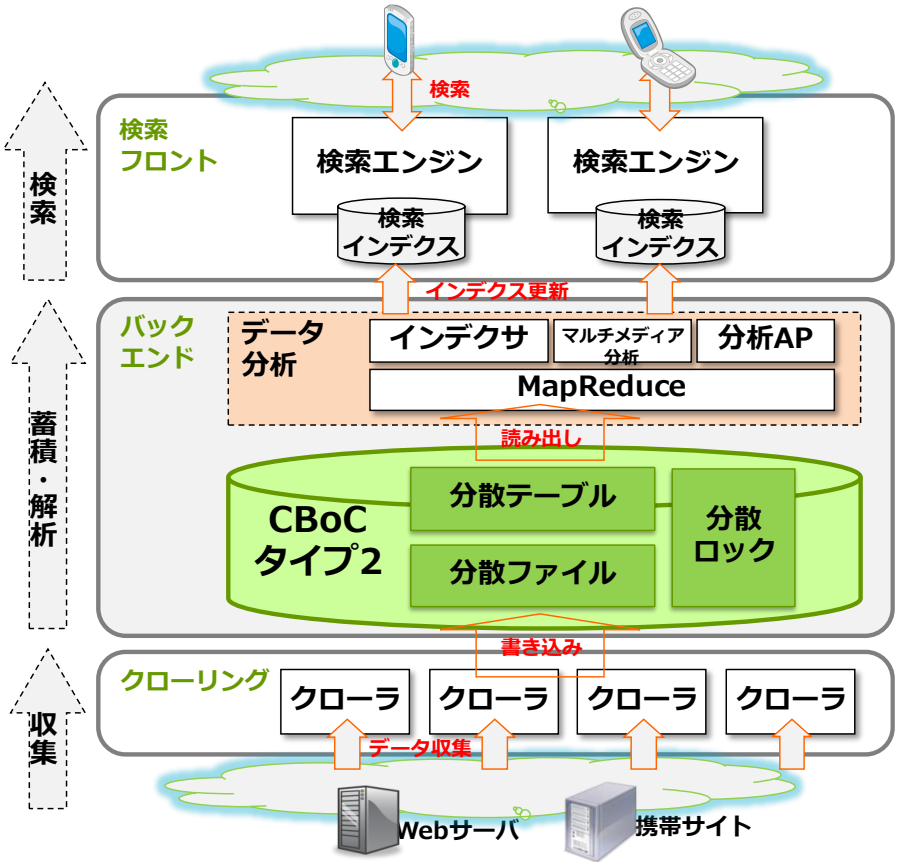


図 6. CBoC タイプ 2 の検索サービスにおける適用イメージ

表 5. ユースケース(例)

ユースケース	AP	処理 (タイミング)	説明
データ収集 処理	クローラー	ランダムライト (定期的)	Web データを収集してそのデータを システムに書き込む
検索要求 処理	収集データ編成 ツール	シーケンシャル リード(定期的)	Web データを分析用に編成する
	スマートキャッシュ	ランダムリード (随時)	検索を高速化させるための事前の データキャッシュ機能
データ分析 処理	テキスト内容分析	ランダムライト, ランダムリード (随時)	検索の問い合せに対しテキストの内容 を分析し結果を返却する
	マルチメディア分析		検索の問い合せに対しマルチメディア データを分析し結果を返却する

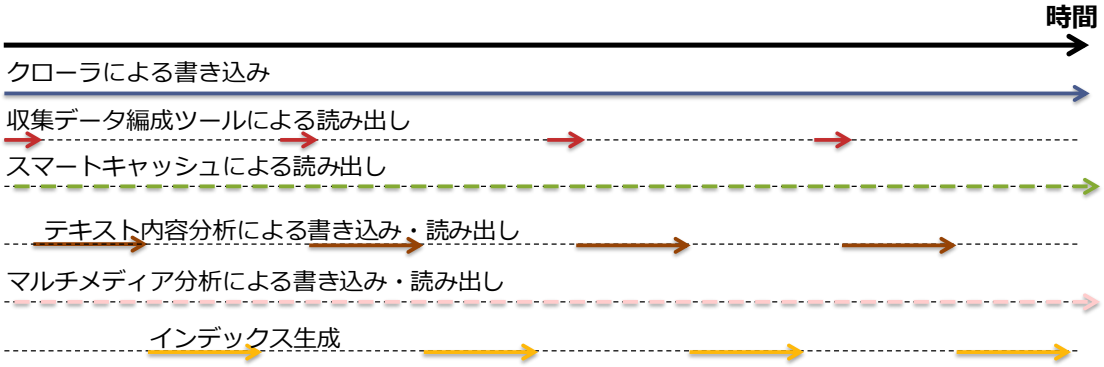


図 7. AP の走行タイミング

一方，非機能要件としては，耐障害性や性能や精度の検証がある．表 6 に CBoC タイプ 2 において，1行目のデータ格納規模で規定されている耐障害性，性能要件，精度を例として示す．

耐障害性は，システムを構成する 1 台のサーバで故障が発生しても，システム全体は継続して運用が可能であることを確認する．

性能の確認は，ユースケースにおいて，設計された必要な性能である性能要件を満たしていることを確認する．CBoC タイプ 2 のシステム検証における性能の確認は，サーバ数百台規模でデータの読み出し／書き込みが規定された性能要件の数値以上であることを確認する．

精度の確認は，テキストの内容分析とマルチメディア分析では異なる．テキストの内容分析では，問い合わせにキーワードを使用し，パターンマッチングによって返却された結果を確認する．マルチメディア分析では，特徴量を使用し，類似度検索によって返却された結果を確認する．特徴量を使うことにより，検索時間を短縮し，検索精度を向上することができる．（付録 4 参照）

表 6. 要件（例）

	要件
データ 格納規模	サーバ数百台規模で 30 億レコードのデータを格納 （1 レコードは，1Web ページの情報を格納）
耐障害性	単一障害が発生しでもシステム全体は継続して運用が可能であること
データ 書き込み性能	テキスト：30 億レコードを 1 日で書き込みができること マルチメディア：30 億レコードの 1/100 を 1 日で書き込みができること
データ 読み出し性能	テキスト：30 億レコードを半日で読み出しができること マルチメディア:30 億レコードの 1/100 を 1 日で読み出しができること
精度	テキスト：パターンマッチングでキーワード検索できること マルチメディア：類似検索ができること

2.1.2　大規模分散処理システムの運用例

大規模分散処理システムの運用は，図 8 のように数百台規模のマシンを数十台のラックに入れて運用する．各ラックでは製品で規定されている電源容量があるため，マシンの搭載台数は，その電源容量を超えないように調整する．各ラックは，エッジスイッチ（Edge Switch）と呼ばれるラック毎のスイッチに収容される．エッジスイッチはそれらを収容するコアスイッチ（Core Switch）で統合される．これらのラックとスイッチの構成により，各マシン間は相互に通信が可能となる．

運用中のマシンの監視は，5.1.2 項に記述するように，監視ツールにより監視をする．大規模分散処理システムは，故障したマシンを自動的に切り離す死活監視機能を具備している．しかし，死活監視機能が働かない故障の場合，故障したマシンにプロダクトを稼動させたままだと，NW（ネットワーク）や処理に関係する故障の場合，システム全体の性能や動作に影響が及ぶ．システム検証では，機能の動作や性能測定の測定値を決められた要件で検証をする必要があるため，故障したマシンを速やかに検知し，正常なマシンと入れ替える必要がある．

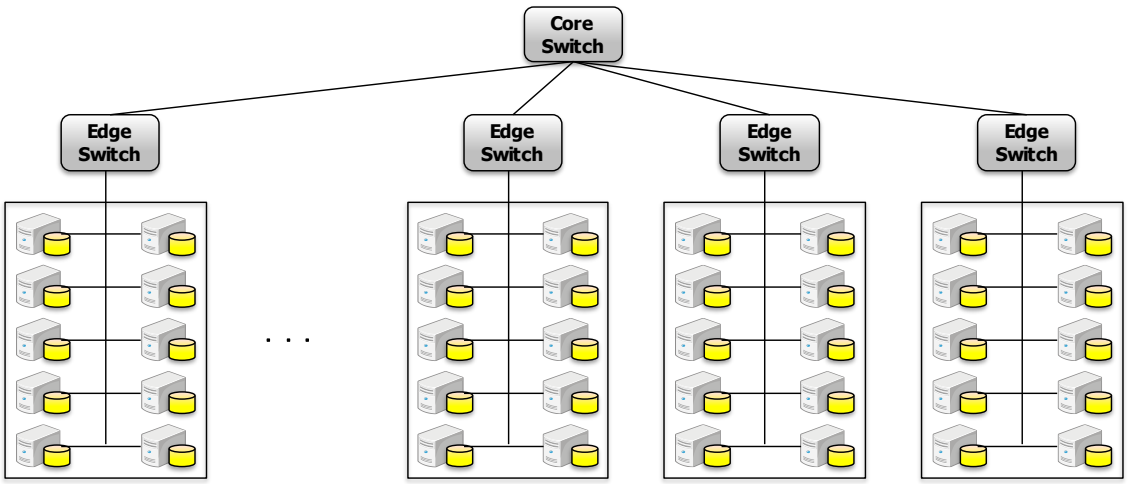


図 8. 大規模分散処理システムの運用

2.2 大規模分散処理システムのプロダクトと処理

大規模分散処理システムは，分散ロック，分散ファイル，分散テーブルと3プロダクトのシステムで構成されており，それぞれのプロダクトが連携して動作する．本節では，大規模分散処理システムのプロダクトと主要な処理と効率化の処理について示す．

2.2.1 大規模分散処理システムのプロダクト

以降に大規模分散処理システムのプロダクトについて示す．大規模分散処理システムは，分散ロック，分散ファイル，分散テーブルの 3 プロダクトから構成される．以降に 3 プロダクトについてそれぞれ説明する．

●分散ロック：

分散ロックシステムは，分散処理を実行するマシン間で資源の分散調停・死活監視を行うためのシステムである．分散ロックシステムの構成を冗長構成にすることにより，分散ロックシステムの可用性を実現する．構成要素であるサブプロダクトとしては，分散ロックシステムの機能を提供する分散ロックサーバがある．

(1) 分散ロックサーバ

分散ロックを制御する機能と関連するコマンドを提供する

●分散ファイル：

分散ファイルは，大規模データを分散した多数のマシン上で管理する，スケールアウト可能なファイルシステムである．構成要素であるサブプロダクトとしては，分散ファイルのメタデータ（META）を管理し全体を制御する分散ファイルマスタと，各マシンでユーザデータを管理する分散ファイルワーカがある．

(1) 分散ファイルマスタ

分散ファイルを制御する分散ファイルマスタと関連するコマンドを提供する

(2) 分散ファイルワーカ

分散ファイルのファイル入出力を制御する分散ファイルワーカと関連するコマンドを提供する

●分散テーブル：

分散テーブルは，大規模な構造データを分散サーバ環境で管理するためのシステムであり，分散サーバ環境での構造データの読み書きを実現する．構成要素であるサブプロダクトとしては，テーブルを特定する管理情報のメタデータ（META）とメタデータを特定する管理情報のルート（ROOT）とを管理し制御する分散テーブルマスタと，各マシンでユーザテーブルを管理する分散テーブルワーカがある．

(1) 分散テーブルマスタ

分散テーブルを制御する分散テーブルマスタと関連するコマンドを提供する

(2) 分散テーブルワーカ

テーブルの入出力を制御する分散テーブルサーバと関連するコマンドを提供する

これらのプロダクトが連携することで，「分散かつ大規模」，「可用性」を実現する． このように大規模分散処理システムは，複数のサブプロダクトが組み合わさり連携して動作するシステムである．そのため，組み合わせが変更になった場合を考慮する必要があり，検証項目は最大構成だけではなく，変更が可能な全ての組み合わせの検証をすることになる．

2.2.2 大規模分散処理システムの処理モデル

大規模分散処理システムの主要な処理である，データの書き込み／読み出し処理とリカバリ処理の処理モデルについて説明する．データの書き込み／読み出し処理では，サーバ台数に応じて処理量にスケーラブルに適応し，「スケールアウト性（分散かつ大規模）」を実現するデザインとなっている．リカバリ処理においては，「耐障害性（可用性）」を実現するデザインとなっている．検証項目は，データの書き込み／読み出し各処理の確認のみならず，書き込み／読み出しの処理が同時の場合の確認を行う項目である必要がある．また，その際のリカバリ処理による可用性の確認も行う項目である必要がある．以下では，データの書き込み／読み出し処理とリカバリ処理の各処理モデルについて説明する（図 9～図 11 参照）．

●書き込み／読み出しの処理モデル：

書き込み／読み出し処理では，クライアントからのリクエストに応じて，データの書き込み／読み出し処理を実施する．その際，処理の分散やデータの複製処理が実施される．処理の分散では，クライアントからの書き込み／読み出しのリクエストを複数のサーバ上にあるワーカに分散させて処理することによって，システム全体の処理性能が向上する．以下に，分散テーブル，分散ファイルの分散方式を示す．

(1) 分散テーブルにおける処理分散方式：

分散テーブルでは，あるサーバ上のマスタが処理を担当するワーカを決定する．マスタによって決定されたワーカの管理情報はメタデータ（META）として管理され，さらにメタデータ（META）の管理情報は，ルート（ROOT）として管理される．そのため処理を行うクライアントは，ROOT から META の情報を得て，META からワーカの情報を得る．複数の META で分散してワーカの情報を持っているため，クライアントの処理は分散される（図 9，図 10 の①～④）．

(2) 分散ファイルにおける処理分散方式：

分散ファイルでは，リクエストを処理するワーカの決定をマスタが行う．分散ファイルへのリクエストは，リクエストを処理するワーカの情報をマスタに問い合わせることで処理が分散される（図 9，図 10 の⑤～⑥）．

一方，データの複製処理では，書き込み時に複数個の複製を作成し，かつ複製間の整合性を保つことでデータの消失を防ぐ．読み出し時は，ネットワークの距離が最も近いワーカの複製データを読み出す．

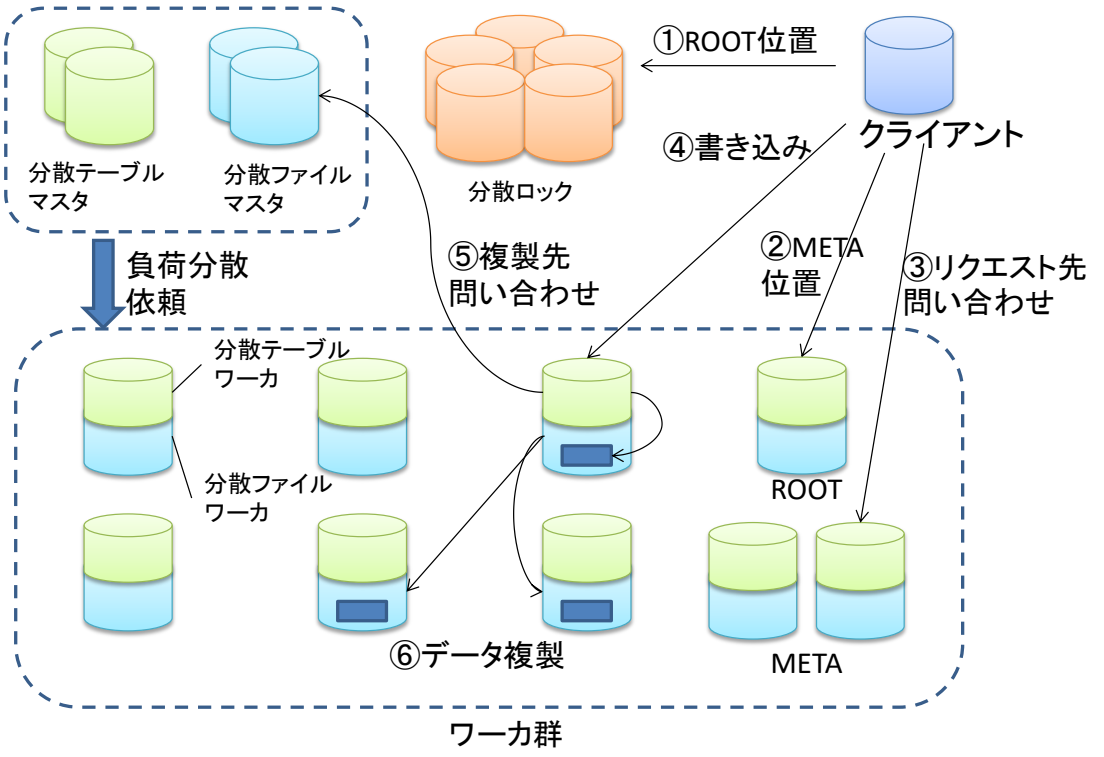


図 9. 書き込みの処理モデル

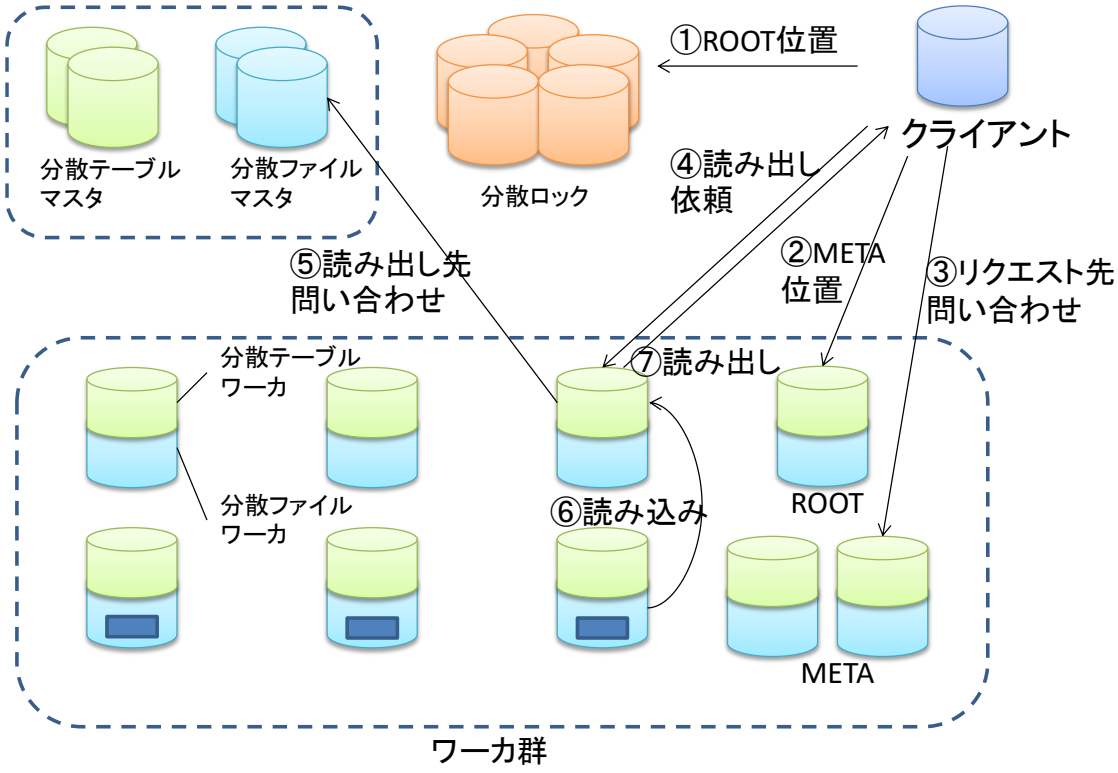


図 10. 読み出しの処理モデル

●リカバリの処理モデル：

リカバリ処理では， ネットワークの分断やサーバ障害が発生した場合に，分断された小サーバ群や障害発生サーバを自動的に切り離し，別のサーバに切り離したサーバの役割を担わせる．図 11 に示すように具体的な処理は，分散ロックによりワーカ群の障害を検知し，分散テーブルと分散ファイルのマスタに障害を通知する．障害の通知を受けたマスタは，フェールオーバによりサーバの役割を交代させるため，別のワーカに処理の割り当てを変更する．その一方で，マスタからのリカバリ依頼により複製データをコピーすることによってデータの消失を防ぐ．これにより，システム全体を停止させずにクライアントからのリクエストに応答することが可能である．

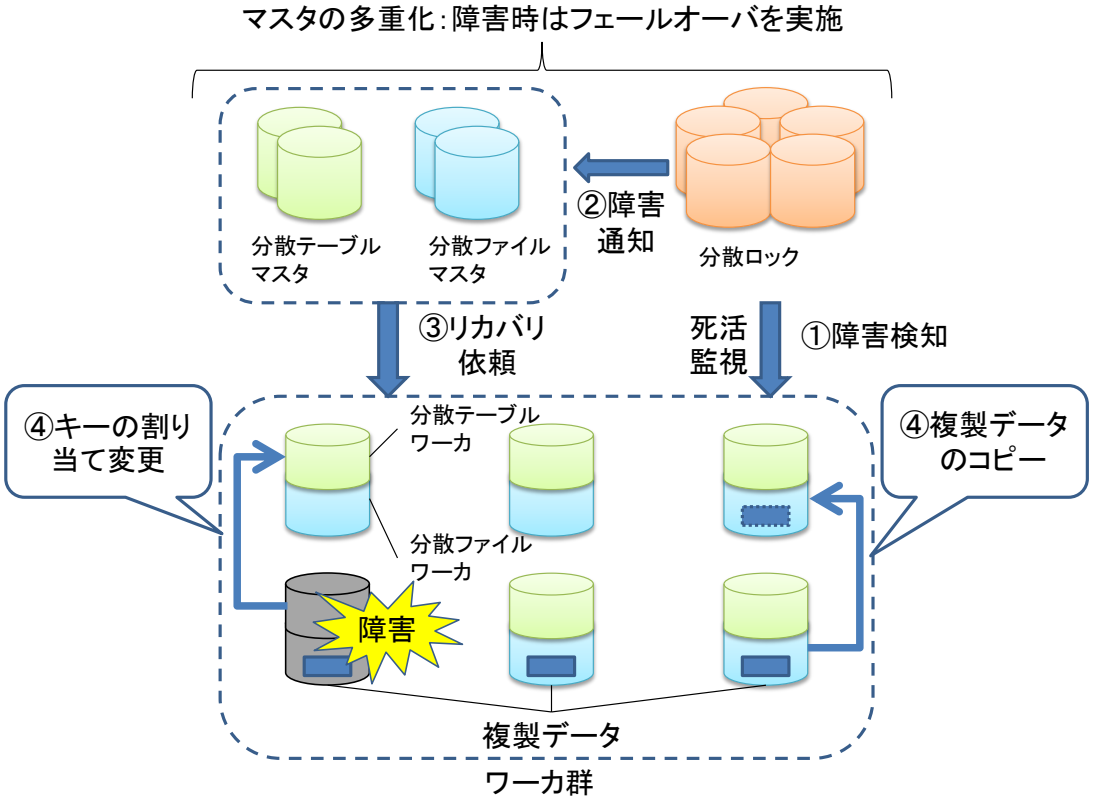


図 11. リカバリの処理モデル

2.2.3　大規模分散処理システムのデータ量の効率化処理

本項では，データ量の効率化処理であるデータ圧縮機能とコンパクションについて示す．この効率化処理は，大量のデータを読み書きする処理で効果的である．

●データ圧縮機能：

データ圧縮機能とは，指定した圧縮方式に従ってデータの圧縮・展開を行う機能である．分散ファイルシステムに対してデータの入出力を行う処理の性能向上を図る方法として，物理的に異なる複数のサーバプロセス間で送受信するデータの量を小さくすることが挙げられる．本機能を使用して送信バッファにデータを送出する際にデータの圧縮を行い，分散ファイルシステムに対して圧縮済みデータを入出力することによって，処理全体としての性能を向上させ，クローリングして収集したデータを効率よく蓄積することができる．具体的なデータの圧縮・展開を以降に示す．

本機能では，複数の圧縮方式を実装し，分散テーブルは設定情報によって，データの圧縮方式を切り替えることができる．データの圧縮は，分散テーブルでデータの圧縮・解凍の要求を受け付けた後，使用する圧縮方式を決定し，データの圧縮を実行する．データの解凍は，圧縮時に取得した圧縮種別を元に解凍方式を決定し，データの解凍を実行する．なお，分散テーブル上にデータがすでに登録された状態で， 圧縮方式を他の方式に切り替えると，異なる方式で圧縮されたデータが混在することになるが，この場合でも分散テーブルサーバは，圧縮種別により正しくデータを解凍することが可能である．

●Compaction（コンパクション）機能：

Compaction 機能とは，追加や更新操作を通してレコード件数が増大した際に，データを整理し圧縮(Conpaction)する機能である．具体的には，(1)利用しているメモリ容量の削減を目的としたメモリ上のデータをディスクへの保存，(2)データサイズの削減を目的とした利用しているファイルの統合，(3)利用しているファイルの再構築を行う機能である．Compaction 機能には，それぞれ MinorCompaction，MergeCompaction，MajorCompaction が存在する．それぞれ以下に示す．

(1) MinorCompaction

MinorCompaction 機能は，メモリ上のデータをディスクに保存する機能である．この機能は，1 つのサーバのメモリ使用量を削減することと，サーバがダウンした後のリカバリ処理の時に読み込むログの量を削減することである．メモリのデータ量が設定した閾値を超えた場合，MinorCompaction が発生する．

(2) MergeCompaction

MergeCompaction 機能は，いくつかのディスクのデータを 1 つのデータファイルに統合する機能である．この機能は，データファイルを統合することにより，検索性能を向上させることを目的とする．サーバ上のデータのファイル数が設定した閾値を超えた場合，MergeCompaction が発生する．

(3) MajorCompaction

MajorCompaction 機能は，システム全体のメモリ上のデータとディスクのデータを再構築する機能である．この機能は，有効でないデータを削除してディスク容量を減らすことと，データを再構築することで検索性能を向上させることを目的とする．サーバ上のデータ量が設定した閾値を超えた場合，MajorCompaction が発生する．

上記の処理の中で，マシン負荷は MajorCompaction が一番高く，次に MergeCompaction である．しかし，MergeCompaction が大量データの投入時にシステム全体で同時に発生すると，MajorCompaction のマシン負荷と同等となる．

2.3 大規模分散処理システムの検証の効率化に関する先行研究

本節では大規模分散処理システムにおけるシステム検証の効率化に関する先行研究をレビューする. 最初に大規模分散処理システムの検証の効率化について, 次に従来の検証である従来の情報システムにおける検証の効率化の研究について紹介する. システム検証で発生する問題に関する研究は, 3,4,5 章の各章で記述する.

1.1 節で記述したように, 対象とする大規模分散処理システムは, 近年の BigData に対応するシステムとして注目されている. しかし, 以降に示すように, 大規模分散処理システムに関する論文やソフトウェアの検証に関する論文はあるが, 大規模分散処理システムにおけるシステム検証の効率化に関する研究は報告されていない. 報告の少なさは, 大規模分散処理システムが注目されて間もないため, 従来の情報システムと比較すると開発例が多くないことや, 企業にとって検証の効率化は費用削減のアドバンテージになることから, 率先して公表をしないことが一因と考えられる.

倉持[24]は, 大規模分散処理システムにおいて, 処理のトレースが可能なライブラリや, サービス間の呼び出し関係と処理時間をトラッキングする API を開発して利用する報告をしている. また, LIU ら[25]は, 実際に実行されているプロセスの動作と, 定義された仕様の動作とを自動的に比較するツールを開発し利用する報告している. これらの方法は, 工数をかけてライブラリや API やツールを開発する必要があるが, 機能が少なく開発工数が開発全体に影響しない場合, デバックの効率化に効果的である. その他には, Jerry Gao ら[26], 坂西ら[27], Haryadi S. Gunawi ら[28], Sergiy VILKOMIR[29], Lian Yu ら[30], Leah Muthoni Riungu ら[31]は, ホストマシン上の仮想的なバーチャルマシン（VM）で構成された大規模分散処理システムにおいて, 従来の情報システムにおけるシステム検証と比較し, ハードウェアに依存する検証や実運用の環境でのシステム検証を不要とする報告をしている. この方法は, 物理的なマシンではなく VM を利用する場合に有効である.

一方, 従来の情報システムにおける検証については, Cem Kaner ら[41], Glenford J. Myers ら[42], IEEE の SWEBOK[14] で詳しく報告されている. これらは, 検証の方法について記載されており, 大規模分散処理システムの検証に全て適用が可能である. しかし, 1, 2 章で記述したとおり, 大規模分散処理システムに特有である大規模なデータ量で, 多くのマシン台数を利用する検証作業は, 従来の情報システムにおける方法では時間を要してしまう. 期間が定められた商用開発においては, 時間を要したことで期間を延伸して検証を完了することは許されない. 従来の情報システムにおけるシステム検証で利用する技術を, 1.2.2 項で示したアクティビティの分類を利用して以下の表 7 に示す.

表 7. 従来の情報システムにおいてシステム検証で利用する技術

アクティビティ	利用する技術	適用可否 (可:○, 否:×)
計画	・プロセス計画	○
	・成果物の決定	○
	・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	○
テストケース生成	・論理網羅テスト(全数テスト)	×
	・組み合わせテスト(直交表, HAYST 法, オールペア, k-way)	×
	・同値分割, 境界値分析	×
	・原因-結果グラフ(デシジョンテーブル, CFD 法)	×
	・機能テスト	×
	・ユースケース	○
	・統計的テスト	○
	・シナリオテスト	○
	・リスクベース	○
	・欠陥仮設法(エラー推測, 異常値, 特異値分析)	○
	・例外ユースケース(探索テスト)	×
	・状態遷移テスト	×
	・モデル検査(カバレッジ)	×
	・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(モンキーテスト)	×
テスト環境の開発	・ソフトウェアエンジニアリングツール ➢ テストジェネレータ ➢ テスト実行フレームワーク ➢ テスト評価ツール ➢ テストマネジメントツール ➢ 性能分析ツール	×
	・保守ツール	×

実行	・動的検証 ➢ ホワイトボックステスト ✧ パステスト(組み合わせテスト, エラー推測, 欠陥仮設法) ✧ トランザクションフローテスト(タイミングテスト) ✧ データフローテスト(同値分割, 境界値分析) ✧ 状態遷移(グラフ)テスト(原因-結果グラフ) ➢ ブラックボックステスト ✧ ドメインテスト(ユースケース, 統計的テスト, シナリオテスト, リスクベース, 例外ユースケース) ✧ ランダムテスト ・静的検証 ➢ 構文テスト ➢ 論理テスト(カバレッジ, モデル検査, 静的解析)	○ △(構文テストは適用可)
テスト結果の評価	・テストされるプログラムの評価 ➢ テスト計画およびテスト設計におけるプログラム計量 ➢ フォールトタイプ／クラス分け／統計 ➢ フォールト密度 ➢ ライフテスト／信頼性評価 ➢ 信頼度成長モデル ・実施されたテストの評価 ➢ カバレッジ／徹底度計量尺度 ➢ フォールトの人為的種まき ➢ 得意仕留め得点数 ➢ 種々の技法の比較および相対的有効度	○ ○
問題報告／テストログ	・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	○ ○ ○
欠陥追跡	欠陥／増補／論点および問題追跡ツール	○

計画アクティビティで利用する技術は, 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[14].

・プロセス計画は, ウォータフォール, スパイラル, 進化的プロトタイピングといったソフトウェアライフサイクルモデルの選定, および適切なソフトウエアライフサイクル・プロセスの適応化, および割り

付け配置の技術である.

・成果物の決定は，タスクそれぞれから成果物を仕様化し決定する技術である．その過程において，先に開発した成果物や他のプロジェクトで開発した成果物の再利用が検討される.

・工数，工程，およびコスト見積もりは，タスク，入力，出力の要素それぞれの工数として過去の開発における規模対工数データが利用および参照可能であれば，そのデータに基づいて見積もりをする技術である．利用および参照ができない場合は，エキスパートによる判定のような開発のデータを分析する方法で見積もりをする．この工数，工程，およびコスト見積もりの技術は，類似なシステム開発のデータが存在する場合に有効である．そのため，類似なシステムが少ない大規模分散処理システムでは適用ができない.

・資源割り付けは，ガントチャートなどを用いて装置，設備，および人員をタスクに割り当てる技術である．この資源割り付けの技術は，工数，工程，およびコスト見積もりの技術の結果を反映しており，結果が正しい場合は有効である．見積もりの技術が確立していない大規模分散処理システムでは適用ができない.

・リスクマネジメントは，意思決定樹やプロセスシミュレーションによるリスク査定法により，リスクの識別と分析を行いリスクの軽減，リスクの発生による処理を行う技術である．品質マネジメントは，プロダクトに対して適切に定量的，定性的に定義し，全プロセスに渡って実施されるプロダクトに対する検証と妥当性確認をする技術である.

・計画マネジメントは，プロジェクト全体をどのようにマネジメントするか，および計画そのものをどのようにマネジメントするのかをマネジメントツール[15]を活用しながら計画する技術である.

上記のとおり，プロセス計画，成果物の決定，リスクマネジメント，品質マネジメント，計画マネジメントの技術は，どのソフトウェア開発に適用可能な技術である．しかし，工数，工程，およびコスト見積もり，資源割り付けの技術は，大規模分散処理システムでは適用ができない.

テストケース生成アクティビティで利用する技術は，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[42].

・論理網羅テスト（全数テスト）は，プログラムのソースコードを網羅的に実行する技術である.

・組み合わせテスト（直交表を用いたテスト）は，テストケース生成の技術を組み合わせて検証する技術である．任意の 2 因子についての組み合わせを決定するために，直交表を利用する.

・同値分割は，2 つ以上の同値のテストケースからいくつかのテストケースを選択する技術である.

・境界値分析は，テストケースで入力条件と出力条件から 1 つまたは複数個の要素を選ぶ技術である.

・原因−結果グラフ（デシジョンテーブル）は，原因−結果グラフのツールを使って問題の摘出効果の高いテストケースを選択する技術である．原因−結果の判定の組み合わせを決定するために，デシジョンテーブルを利用する.

・機能テストは，システムの機能が規定された役割や処理ができているかどうかを確認する技術である．
・ユースケースは，ユーザの利用シーンを想定した検証の技術である．
・統計的テストは，統計的に利用が多い代表的なテストケースを作成し検証する技術である．
・シナリオテストは，業務を想定したシナリオを作成し検証する技術である．
・リスクベースは，バグが出た場合の影響度を品質リスクと捕らえ検証期間の早い段階で品質リスクを低減する検証の技術である．
・例外ユースケースは，ユーザの利用シーン以外やトラブル状態を想定した検証の技術である．
・欠陥仮設法は，品質特性から欠陥の兆候の仮説を立て，仮説に基づいて検証をする技術である．
・エラー推測は，経験から問題を抽出できそうなテストケースを作成し検証する技術である．
・状態遷移テストは，状態遷移図や状態遷移表に基づいて検証する技術である．
・モデル検査は，システムを有限後の状態を持つモデルで表現し，システムが満たすべき性質がモデル上で成り立つかどうかを，モデルが取りえる全ての状態およびパス上で機械的かつ網羅的に検証する技術である．
・タイミングテストは，イベントの同時発生などタイミングが関係する箇所に着目し検証する技術である．
・静的解析によるピンポイントテストは，ソースコードを解析してコーディングルールに合っているかどうかや，データフローや制御フローを解析し，エラーが発生しやすい箇所を分析し検証する技術である．
・ランダムテストは，ランダムな入力を与えて検証する技術である．

上記のとおり，ユースケース，統計テスト，シナリオテスト，リスクベース，欠陥仮説法の技術は，どのソフトウェア開発に適用可能な技術である．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，例外ユースケース，状態遷移テスト，モデル検査，タイミングテスト，静的解析によるピンポイントテスト，ランダムテストは，大規模分散処理システムでは，検証項目数が多くなることから適用ができない．

テストケースが膨大な場合，品質を確保し効率的に検証をするには，利用する検証技術は，図 12 の縦軸と横軸が交差する中心近くに位置する検証技術の必要がある．これは，横軸の「ピンポイント」に位置する検証だと項目の網羅性がなく，それ以外だと大規模分散処理システムで検証項目を抽出すると項目数が多くなってしまうからである．しかしながら，どの技術を使えば品質を確保し効率的に検証ができるかは明らかになっていない．

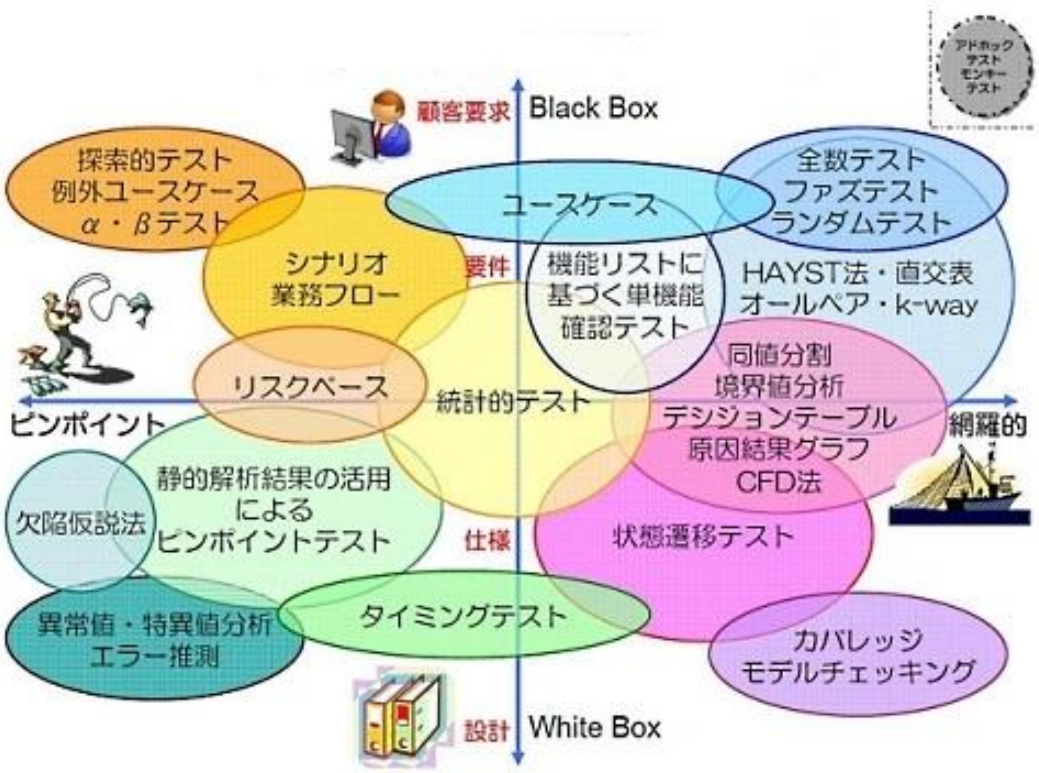


図 12. 検証技術のポジショニングマップ（ソフトウェアテストシンポジウム JaSST'12 Tokyo：http://jasst.jp/symposium/jasst12tokyo/outline.html より）

テスト環境の開発アクティビティで利用する技術は，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[14].

- ソフトウェアエンジニアリングツールはさらに，テストジェネレータ，テスト実行フレームワーク，テスト評価ツール，テストマネジメントツール，性能分析ツールなど，テストケース生成および制御を容易な環境にする技術から成る.
- 保守ツールは，テスト環境を保守するツールやスクリプト（Shell Script）などでテストの結果を再現できるようにテストの環境を保守する技術である.

上記のとおり，ソフトウェアエンジニアリングツール，保守ツールは，開発対象のプロダクトに依存するため，類似なシステム開発の検証には適用が可能であるが，類似でないシステム開発の検証には適用できない．そのため，類似なシステムが少ない大規模分散処理システムでは適用ができない.

実行アクティビティで利用する技術は，松本[14]，黒坂ら[45]，根路ら [46]，Y. Falcone ら[47]，進ら[48]により，動的検証と静的検証に分類されることが示されている．動的検証は，実際にプログラムを実行することにより，仕様と合致しているかの確認をする技術である．CBoC タイプ 2 のシステム検証は動的検証である．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に分類

される. ホワイトボックステストは, システムの内部構造を理解した上で, ロジックや制御の流れが正しいかどうかを検証する技術である. ブラックボックステストは, システムの内部構造は考慮せず, 仕様を満たしているかどうかを検証する技術である.

ホワイトボックステストの検証方法としては, 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[43].

・パステストは, 構造モデルとしてプログラムの制御フローを使い, プログラムから一連のテストパスを選んで検証する技術である. 検証の実施の完全性に対する記述がいくつかあれば, それに基づいて一連のテストパスを選択することが可能である. 例えば, 全てのソースコードの文を少なくとも1回は実行するパスを選び出す.

・トランザクションフローテストは, トランザクションフローグラフを使用して動作モデルを作成し, 機能テストを実施する技術である. トランザクションフローグラフとは, システムの動作をモデル化したものである.

・データフローテストは, 制御フローを使って, データに発生する不正を検出する技術である. 例えば, データは使用前に初期値化されていることや, 定義済みのデータがなんらかの目的に使用されていることを検証するパスを選択するなどである.

ブラックボックステストの検証方法としては, 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[43].

・ドメインテストは, 機能仕様書やソースコードを基に検証をする技術である. 機能仕様書に基づいて検証をするとドメインテストは機能テストであり, ソースコードを基にすると構造テストとなる. ただし, 検証項目による検証において, 入力変数が1つの場合や, 2 つであっても組み合わせが簡単な場合は, ドメインテストの実行に相当する. 例えば, 通常の変数の範囲を超えた値による検証がこれに相当する.

・状態遷移(グラフ)テストとは, 状態グラフで表現されたシステムのモデルを用いてソフトウェアの構造, 動作, 仕様を検証する技術である.

・ランダムテストは, プログラムにランダムな入力や操作を行い, エラーの検出や信頼性を確認する技術である.

上記の動的検証であるパステスト, トランザクションフローテスト, データフローテスト, 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は, どのソフトウェア開発でも有効である.

一方, 静的検証は, 形式言語で仕様や設計を記述してプログラム構造の検証をする技術である. 来間[34], 中島[35], 中島[36]は, 静的検証では形式手法による検証方法を示している. これらは, プログラム言語のように文法が形式的に与えられた言語で仕様や設計を記述する. その仕様や設計からシステムを論理的に検証することにより, 品質を高めることが可能である. 静的検証における形式手法としては, 構文テスト, 論理テスト, 状態遷移(グラフ)テストの技術がある.

・構文テストとは, 検証対象となるシステムのソースコードをBNF(Backus-Naur Forrm)のような形式的言語で表現し, 構文の完全性と一貫性を検証する技術である.

・論理テストとは, 仕様書の内容をデシジョンテーブルやブール代数を用いて記述し, 仕様の完全

性と一貫性を検証する技術である.

上記の静的検証である構文テストは, コンパイラや構文チェックのシステムがあるため, どのソフトウェア開発で有効であり, 論理テストは, 全ての動作が膨大にはなく論理的に検証することが可能なシステムに対しては有効である. また, 山根[37], Kavi K.M. [38], Bryant R.E. [39], McMillan K.K. [40]は, さらにその手法を分散システムに対応させた報告をしている. これらは, 状態遷移を形式的に仕様として記述することで状態遷移が明確化されるため, 動作が複雑なシステムの設計を確認する結合検証やプロダクト内検証で有効である. 論理テストは, 大規模分散処理システムでは動作と状態の組み合わせが膨大となるため適用ができない.

テスト結果の評価アクティビティで利用する技術は, 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に分類される[14].

・テストされるプログラムの評価は, ソースコードの行数やファンクションポイントにより, テストを作成するテスト計画およびテスト設計において役立つプログラム計量の技術から成る. 加えて, どのようなタイプの問題が発生するかを推測するフォールトタイプ／クラス分け／統計, 抽出された問題の数をプログラムサイズで割ったフォールト密度, 信頼度の達成, および評価をするライフテスト／信頼性評価, 問題に基づいた信頼度成長モデルにより, テストされるプログラムを計量し, 計画や実行を最適化し, 検証の進捗度を評価する技術から成る.

・実施されたテストの評価は, 仕様や設計で識別された要素と検証した要素とのカバレッジや, 徹底度計量尺度による確認や, 問題であるフォールトの人為的種まきから, 問題をどれだけ摘出できたかの確認により, 検証の網羅性を評価する技術から成る. 加えて, 人為的な変更を摘出する変異仕留め得点数や種々の技法の比較および相対的有効度により, 実施された検証が目的に達成したかどうかを評価する技術から成る.

上記のとおり, 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の技術は, どのソフトウェア開発においても有効な技術である.

問題報告／テストログアクティビティで利用する技術は, 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[14].

・データの収集は, データを収集し検証され記録する技術である.

・データの分析および情報プロダクトの開発は, データ分析で集約され分析される技術である. 結果の伝達は, 分析された結果は文書化されユーザおよび利害関係者に伝達する技術である.

上記のとおり, 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, 結果の伝達の技術は, どのソフトウェア開発においても有効な技術である.

欠陥追跡アクティビティで利用する技術は, 欠陥, 増補, 論点および問題追跡ツールである[14]. 欠陥／増補／論点および問題追跡ツールは, 検証中に抽出される問題を分析し, 原因や改善方法を情報として後に利用できるようにする技術である. 欠陥／増補／論点および問題追跡ツール

の技術は，どのソフトウェア開発においても有効な技術である．

以上をまとめると，表 8 のように大規模分散処理システムにおけるシステム検証において，計画アクティビティと，テストケース生成アクティビティと，テスト環境の開発アクティビティに関する作業の効率化は，従来の手法が適用できず，まだ解決手法が明確にはされていないオープンな課題である．これらの計画とテストケース生成とテスト環境のアクティビティの解決手法がない部分を 2.2 節の図 4 を利用してマップすると図 13 の赤枠で囲まれた部分になる．赤枠で囲まれた部分は、従来の情報システムである RDBMS の領域以外の部分であり、計画アクティビティはデータ量とマシン台数に関わる部分，テストケース生成とテスト環境の開発のアクティビティは，マシン台数に関わる部分が解決できていない領域である．PMI[15]は，プロジェクトマネジメントでは計画アクティビティは重要な作業工程としている．計画アクティビティと同様に，上流の作業工程であるテストケース生成とテスト環境の開発のアクティビティは重要度が高い．そのため，これらの課題を解決することは，検証作業のスケジュールに影響を与えないためにも必須である．

表 8．従来手法適用可否

	技術	従来手法適用可否
計画	工数およびコスト見積もり	適用否領域
	資源割り付け	
テストケース生成	項目作成	
テスト環境の開発	エンジニアリングツール	
	保守ツール	
実行		適用可領域
テスト結果の評価		
問題報告/テストログ		
欠陥追跡		

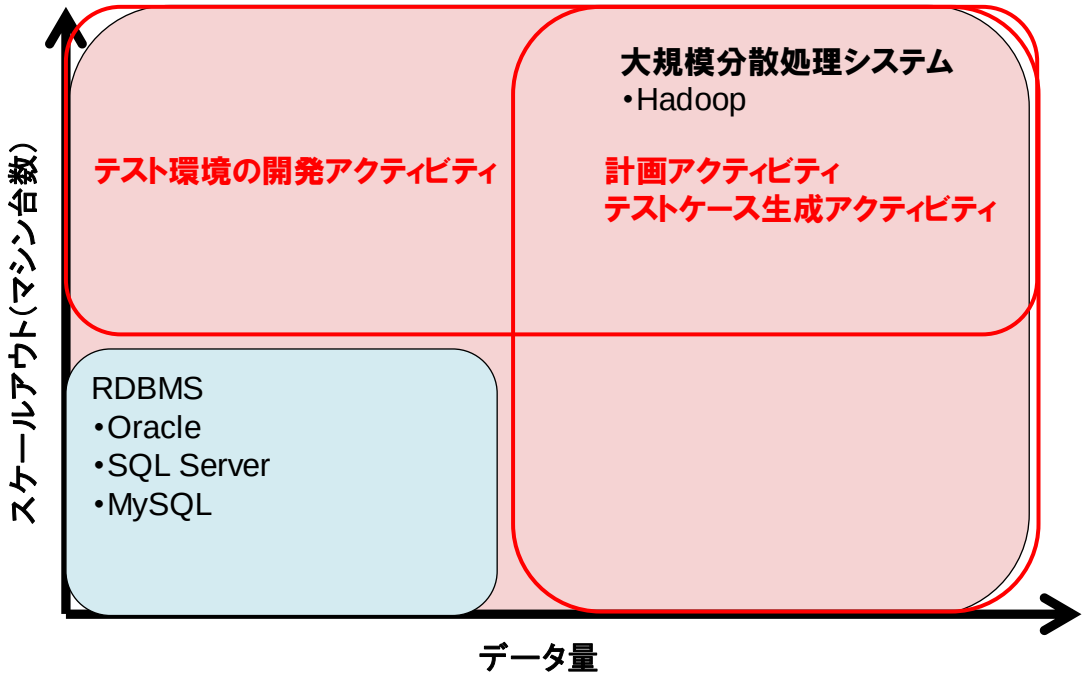


図 13. 解決手法がない部分

3. 計画アクティビティの課題と解決手法

リソースに対するマネジメント計画である計画アクティビティにおいて，主要な課題は「大量な検証データの早期データ準備」（課題 1），「検証実施時間の見積もり精度向上」（課題 2），「効率的にバグ摘出するマシン台数分割による検証実施」（課題 3）の 3 つの課題である．課題 1 に対しては「I/O デバイスをチューニングした高速データ登録手法」（手法 1），課題 2 に対しては「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），課題 3 に対しては「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）でそれぞれ効果がある．

本章では，計画アクティビティにおいて，検証作業の合理化により検証期間を削減する解決手法を 3 つの課題それぞれに対して説明する．3.1 節で先行研究，3.2 節で課題を示した後，3.3 節で課題 1 に対して効果のあった手法 1 について，3.4 節で課題 2 に対して効果のあった手法 2 について，3.5 節で課題 3 に対して効果のあった手法 3 について説明する．

3.1 計画アクティビティの課題に関する先行研究

本節では，1.2 節に示した計画アクティビティの課題「大量な検証データの早期データ準備」，「検証実施時間の見積もり精度向上」，「効率的にバグ摘出するマシン台数分割と検証項目への割り当て」で利用される技術それぞれについて，関連する先行研究を示す．2.3 節で示したとおり，従来技術のプロセス計画，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，計画マネジメントは，大規模分散処理システムで発生した課題以外において適用できる．しかし，発生した課題においては，以降のようにそのまま適用ができない．

3.1.1 大量な検証データの早期データ準備に関する先行研究

データ登録の研究に関して，ネットワーク I/O のチューニングの研究では，Niels ら[53]が Linux 上に一度に複数のシグナルを受信することができるインターフェースを開発し多重化する手法を提案している．またその他では，H.Kamezawa ら[52]は，Linux の TCP スタックにおいて，並列 TCP ストリーム間で輻輳ウインドウサイズを交換し，データ量を平均化する改善で，高速化によるネットワークのボトルネックを解消している．これらの手法は大量のデータを送受信することを想定して，これらの手法を組み込んでシステムを開発する場合に効果がある．しかしながら，これらの高速化の手法は OS に変更を加えるなど，特別なインターフェースを用いる．システム検証では，運用時の設定で検証するため，コンパイルが必要となる OS の変更などの特別な手法ではなく，簡易な手法で高速化する必要がある．簡易な手法として，公知となっている Linux の OS のパラメータ変更によるチューニングがある．

一方，ディスク I/O のチューニングの研究は，G.AS.Whittle ら[54]や J.Grifficen ら[55]のファイルアクセスのパターンを抽出し，ディスクキャッシュのヒット率を改善した高速化や，Becerra ら[56]により，IBM Cell BE プロセッサを搭載したアクセラレータボードにオフロードするという CPU のアクセラ

レーションで，安定した高速化が実現されている．これらの手法は，高速化されたハードウェアを導入して運用する場合に効果がある．しかしながら，システム検証への適用については，特別な手法ではなく簡易な手法で高速化する必要がある．加えて CPU アクセラレーションでは，Terasort の大量の I/O が発生するデータ処理を高速化できなかったことが記述されており，検証用データの準備など大量のデータ登録には不適であると考えられる．また，ディスクにデータを出力する Compaction の研究は，白木原ら[57]がデータをディスクに出力するのではなく，メモリに一旦蓄積することで高速に保存する方式として実現している．しかしながら，大規模分散処理システムは，基本機能としてメモリに一旦蓄積する方式を採用しており，さらに高速にデータ登録の必要がある．

大量のデータを分散ファイルに格納することについては，伊藤忠テクノソリューションズ[49]が，時間を要することが実運用上で問題であることが報告している．そのため，倉持[50]は有効な対策方法として，データが書き込まれるファイルシステムの高速化を提案している．また，大規模分散処理システムのデータの高速読み出しについては，M. Klems ら[62]や I. Konstantinou ら[63]の提案がある．これは，データの読み出しにロードバランサーの利用とインメモリ化を利用し，データの読み出しを高速化する技術である．これらの手法は高速化されたファイルシステムや新たなハードウェアを導入するシステムの場合に効果がある．しかしながら，システム検証への適用については，専用ハードの利用や，プログラムをリコンパイルする手法ではなく，元の商用運用での設定や状態にすぐに戻せる手法で高速化する必要がある．

3.1.2 検証実施時間の見積もり精度向上に関する先行研究

検証実施時間の見積もりに関する研究としては，戸田ら[64][65]が机上で検討可能な手法として，工数見積もりのモデルから分析する手法を提案している．さらに，工数見積もりモデルを効率的に実施する手法としては，K. Srinivasan ら[66]や M. Shepperd ら[67]が，重回帰モデル，ニューラルネット，事例ベース推論などが提案している．工数見積もりモデルとは，開発のプロジェクトの特性値（開発規模，開発者数など）を説明変数とし，目的変数である工数との関係を数学的に表すものである．これらの手法は，対象のマシン台数が限られた従来の情報システムであり，過去の開発のプロジェクト特性値が得られているものについては効果がある．しかし，過去の工数を算出するモデルがない場合の見積もりは，全ての項目の誤差を 1 週間以内に収めることが困難と考えられる．

3.1.3 効率的にバグの摘出可能な検証マシン台数分割と検証項目の割り当てに関する先行研究

検証マシン台数分割に関連する研究としては，川勝ら[68]や Francois Legillon ら[69]が，あるユースケースで必要な最小限度のマシン台数を算出する研究をしている．この研究で提案されている手法は，必要なサービスのリクエスト量と SLA(Service Level Agreement)を満たすように，CPU 処理能力とNW（ネットワーク）I/O から必要なマシン台数を計算する手法である．この手法はユースケースが決まっているが，マシン台数が決まっていない場合に効果がある．しかし，大規模分散処理システムでは，ユースケースに合わせてマシン台数を決めているので，この手法では最大のマシン台数で検証を実施することになり，効率化を図ることができない．

3.2　計画アクティビティの問題と具体的な課題

計画アクティビティの課題を検討するにあたり，表 9 に示すとおり計画アクティビティにおける作業を「準備」，「見積もり」，「実施計画」の 3 つに分類する．これらの作業における問題と具体的な課題を以下に説明する．

(1) 準備：この作業は，検証前の事前準備のことである．事前に実施するデータ蓄積に要する日数が長期化するという問題があり，大量にデータを登録することにより，その日数を削減する必要がある．

(2) 見積もり：この作業は，検証作業の時間を見積もることである．実施時間の長い検証項目で見積もりよりも多くの時間を要するという問題があり，これがスケジュール遅延の原因となるため，実施時間の見積もり精度向上が必要である．

(3) 実施計画：この作業は，見積もりした検証作業をどう実施するかの計画を立てることである．前工程における解決すべき問題の解析と対処に時間を要するという問題があり，解析と対処が効率的に実施できるマシン割り当てにより，その時間を削減する必要がある．

上記の 3 つの課題は，システム固有のものではなく，大規模分散処理システムで必ず実施する作業で発生する共通の問題である．そのため，これらの課題を解決することができれば，見積もりの確度が高いスケジュールが完成し，作業を合理化して期間内での検証完了に貢献ができると考える． 以降から，作業を合理化できる解決手法を課題それぞれについて詳細に説明する．

表 9.「計画」アクティビティで発生する問題の概要と具体的な課題

作業分類	発生した問題の概要	具体的な課題 （具体的な内容）
準備	検証データが 100 テラバイト～ペタバイト級のため，事前の検証データ登録が長期化し，検証開始が 2 週間以上遅延する	課題 1:大量な検証データの早期データ準備 （事前に実施するデータ蓄積の日数を削減）
見積もり	検証項目の実施時間を見積もる精度が悪く，見積もりで計画より 2 週間以上の期間を要する	課題 2:検証実施時間の見積もり精度向上 （実施時間の見積もりが困難な項目の見積もり精度向上）
実施計画	全てのマシンを利用すると，問題の解析の間はシステム検証が止まり，計画より 2 週間以上の検証期間を要する	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 （発生問題に対する解析と対処の時間を削減）

3.3 「大量な検証データの早期データ準備」(課題 1)と解決手法

表 10 は，従来の情報システムと大規模分散処理システムとのデータ登録の違いを示したものである．CBoC タイプ 2 のような数百台規模の大規模分散処理システムの場合，データ量は 100 テラバイト～ペタバイト級になり，これだけ大規模なデータを運用で利用する AP で事前に蓄積するには，約 1 か月程度の時間を要する．そのため，データ蓄積を高速にするなどして日数を削減することが必要になる．

表 10. データ蓄積に関わる違いとデータ登録の問題

	データ蓄積	データ登録の問題
従来の情報システム	データ規模は数テラバイトであり，数時間で準備可能	設計時のデータ登録の条件で蓄積が可能．登録が短時間なため，他の処理の影響も少なくスムーズで問題なし
大規模分散処理システム	データ規模は大規模データ(CBoC タイプ 2 では 100 テラバイト～ペタバイト級のデータ)を扱う．データの蓄積は実運用のユースケースでは検証期間と同等の 1 ヶ月程度を要する	後述する Compaction 処理が自律的に動作するため，高負荷をかけて Compaction が同時に発生する場合にスループットが低下して安定しない

3.3.1　解決手法「I/O デバイスをチューニングした高速データ登録手法」(手法 1)

課題 1 の解決手法としては「I/O デバイスをチューニングした高速データ登録手法」(手法 1)が効果的であると考えた．なぜなら，手法 1 による安定した高速データ登録は，性能向上の高速化はネットワーク I/O，安定稼動はディスク I/O をそれぞれチューニングするため，課題を解決できると考えたからである．それぞれの具体的手法について以下に説明する．

- ネットワーク I/O チューニング方式:

  大規模データの投入はクライアントから書き込みが行われる．そのため，データ投入のチューニングは，クライアントが動作する OS のインターフェースで実施した．チューニングのポイントは，クライアントから OS にデータ書き込みをする通信のコネクションに関する TCP バッファのサイズであった．TCP バッファのサイズのパラメータチューニングにより，1.5 倍の性能を確保する．詳細について以下に述べる．

  TCP バッファとは，TCP の通信時にパケットを一時的に保管しておくバッファのサイズである．CBoC タイプ 2 では，書き込みと読み出しの処理は同一マシンで実行する AP とリソースを調整できるようにするため，バッファのサイズに関する値はパラメータで変更可能である．パラメータ変更では，ボトルネックとなる上限の値が分かっていないものはデフォルトの値を設定，文献や実際の測定で分かっているものはその値を設定する．バッファのサイズを大きくする変更は，送

信するパケットサイズを大きくすることでパケットの送信回数を少なくすることで，大量のデータが送信できるようになるからである[1]．この手法で性能要件以上の書き込みを行ったところ，エラーが発生しないことを確認した．具体的には，Linux の OS 設定での TCP バッファに対して，送受信バッファの net.ipv4.tcp_wmem と net.ipv4.tcp_rmem を，デフォルトの値の 87380 バイトから 129024 バイトの約 1.5 倍に変更する．このパラメータの値は，1 ソケットあたりのデータ送受信の際に利用するメモリの使用量で，システムに他の AP などの処理（以降，負荷と呼ぶ）がかかっていない状態での割り当て量である．また負荷がある場合は，OS によってそのメモリの使用量が自動に調整される．最大 2Gbps の NW 帯域でデータの書き込みを行い，効果が高いデフォルトの 1.5 倍に設定する．さらにバッファの最大値の net.core.wmem_max と net.core.rmem_max をデフォルトの 131071 バイトから 4194304 バイト（許容されている最大値）に変更して，受信したデータを最大限処理可能にする．その結果，提案手法によって，1.5 倍の書き込みスループットを確保することが可能となる．

- ディスク I/O チューニング方式：

  ディスク I/O のチューニングは，プロダクトが制御しているディスク I/O に対するチューニングにより実施する．プロダクトが制御しているディスク I/O に対するチューニングにより，安定した性能が得られる．詳細について以下に記述する．

  プロダクトが制御しているディスク I/O に関しては，データ登録時にバッファリングされたデータをディスクに書き出す Compaction の処理がある．Compaction が処理性能への影響の支配項になっていることが分かったため，原因となっている Merge Compaction の実行をデフォルトの設定値以上になったときの実行から，設定値に乱数値を足してランダムな実行に変更し，複数のマシンで同時に Merge Compaction が発生しないようにする[2]．複数のマシンにおいて Merge Compaction の同時発生を回避したことにより，ディスク書き込みのボトルネックが解消できる．

以上のように，TCP バッファのサイズのチューニングによって，性能要件の 1.5 倍の書き込みスループットを実現する．またプロダクトのディスク書き込み制御の設定変更によって， 図 14 に示すとおり書き込みスループットを性能要件から常に超えるスループットとなるように安定させる．

[1] 一般的にソケットのバッファサイズを大きくすると，スループットを上げる効果がある．（参考：http://d.hatena.ne.jp/tt_clown/20091130/1259567150）

[2] 商用の運用条件では，データ投入のスループットが事前投入と比べて低く，データは各サーバへ不均衡に蓄積されるため Merge Compaction が同時に発生することはない．また，設定値の変更としたのは，機能追加では設計からやり直す必要があるためである． プロダクトの通常処理に Merge Compaction のランダムな実行の処理を加えることは，ランダム化したことによりバッファがオーバーフローしないよう閾値を超える前に実行するなど，実行条件を検討して機能追加をするため時間を要する．

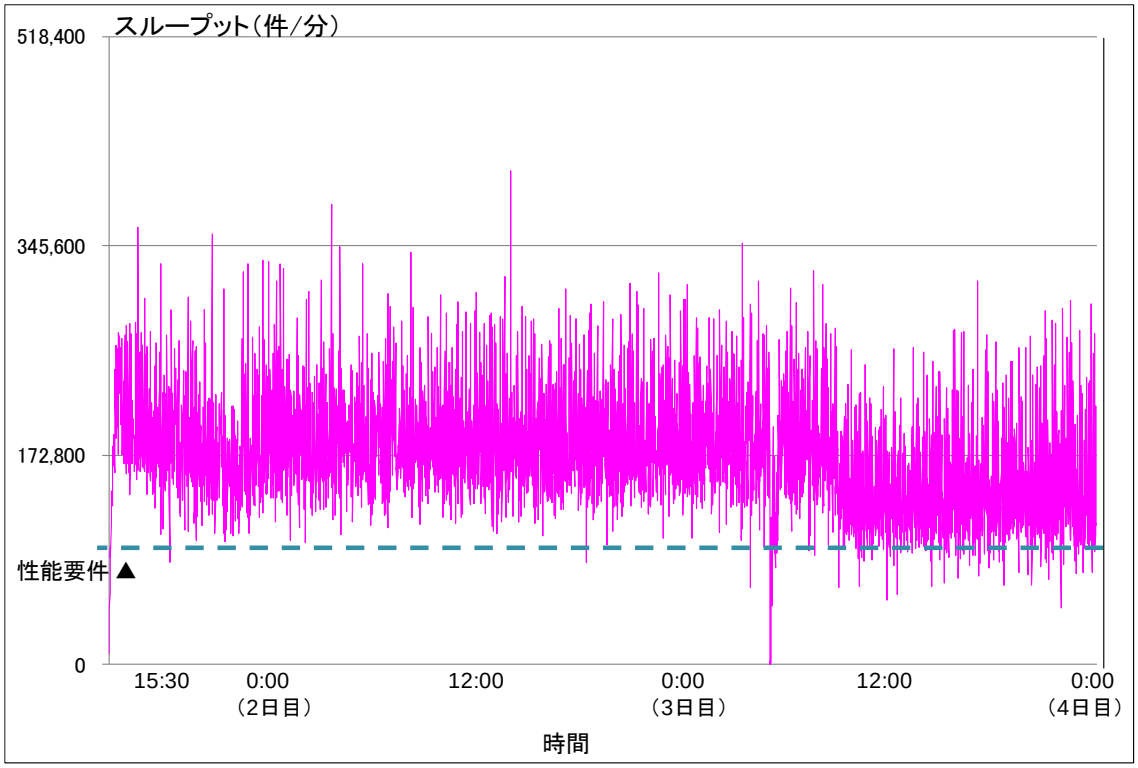


図 14. データ投入例（書き込みスループット一定）

3.3.2　　手法 1 の適用結果

適用結果については，従来の方法と提案手法による方法とで蓄積に要する期間で評価した．CBoC タイプ 2 へ解決手法を適用し，実際にシステム検証をした結果，図 15 に示すとおり，I/O デバイスをチューニングした高速データ登録により，実データの蓄積を期間内に完了できた． 具体的に以下に示す．

計画上のデータ蓄積は，1 ヶ月で行うことを求められていた．しかし，運用で利用する AP を使用し，従来の性能要件で定められた書き込み性能のままでは，3 世代までのデータ蓄積は 960 時間であった．これに対し，3.3.1 項の提案手法に従い，性能要件の 1.5 倍の書き込み速度でデータを投入した．この結果，1.5 倍の書き込みスループットが安定的に維持され，データ登録時間は 650 時間で登録することができた．以降に手法の考察を示す．

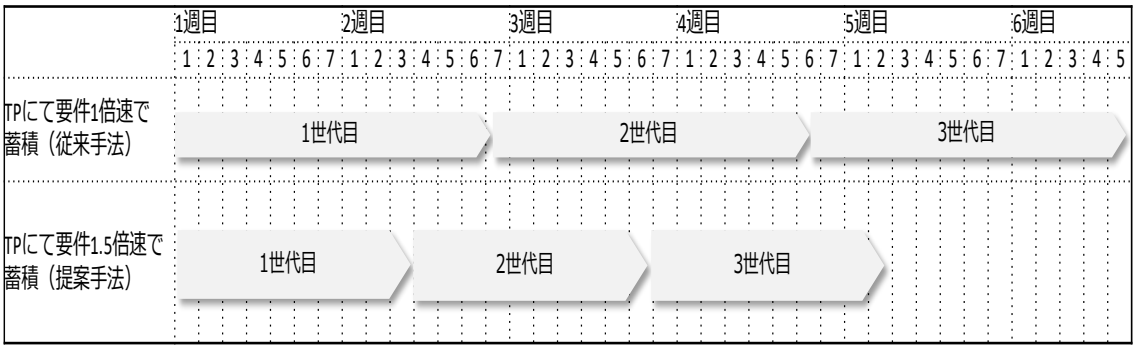


図 15. 事前データ登録の具体的スケジュール

注：世代はデータの版数を意味する．

解決方法「I/O デバイスをチューニングした高速データ登録手法」（手法 1）について，以下の

- 課題 1 の「大量な検証データの早期データ準備」を解決する方法
- ネットワーク I/O チューニング方式
- ディスク I/O チューニング方式
- 適用可能な領域

の観点で考察する．

- 課題 1 の「大量な検証データの早期データ準備」を解決する方法として，

  (1) 検証データの高速登録

  (2) データをコピーして増やす

  の 2 つの案が考えられる．(1)について商用システムでは，ユースケースにおいて，同時動作するデータ登録やデータ分析の書き込みと読み出し両方の処理の影響を試算して，性能要件を決定している．このため，性能要件以上の書き込み速度でのデータ蓄積は動作が保証されない．また，CBoC タイプ 2 のシステムは，図 6 で示したようにサービスのバックエンドのシステムであり，安定動作を重視している．そのため，データ分析処理アプリケーションとの同時処理があるが，安定動作することを考慮して，Linux の proc/sys ディレクトリ配下の OS レイヤの設定は，他のアプリケーションに影響のないデフォルトにしている．しかしその一方，事前のデータ登録は，読み出し処理やアプリケーションの実行と同時に行う必要がないため，OS レイヤの設定変更や性能要件を超えた書き込み速度でのデータの投入が可能である．

  もう一つの手法として，(2)のデータをコピーして増やし，検証データの蓄積を効率化する案がある．このデータをコピーする案は，すでにディスクに蓄積されたデータを利用し，ネットワークを介さずにディスク I/O を最大限活用してデータを追加し増やすことができる．しかしこの案は，前述の性能要件を超えたデータ登録の案と比べ，コピーするツールの作成とメンテナンスに工数を要し，データを蓄積する期間と合わせると 2 倍以上かかるため除外した．

- ネットワーク I/O チューニング方式の考察について以下に示す．大規模データの投入はクライアントから書き込みが行われる．そのため，データ投入のチューニングの調査は，

  (1) クライアントが動作する OS のインターフェース

  (2) クライアントが連携するアプリケーション

  の観点で実施した．その結果，(1)のチューニングのポイントは，クライアントから OS にデータ書き込みをする通信のコネクションに関する TCP バッファのサイズであると判断した．(2)については，クライアントとアプリケーションのコネクションプールに対するパラメータチューニングであると判断した．TCP バッファのサイズのパラメータチューニングについては，性能を確保した方法，コネクションプールについては，効果の確認方法を具体的に以下に示す．

  (1) TCP バッファに対して，送受信バッファの net.ipv4.tcp_wmem と net.ipv4.tcp_rmem を，デフォルトの値の 87380 バイトから 129024 バイトの約 1.5 倍に変更した理由を以下に示

す. 最大 2Gbps の NW 帯域でデータの書き込みを行い, ボトルネックとならない範囲で 1.5 倍, 2 倍と試した結果, 効果が高い数値だったのがデフォルトの 1.5 倍であったためである. また, 書き込み速度のスループットを性能要件の 1.5 倍にしたのは, 1.5 倍, 2 倍, 3 倍と試し, 限界値に近い速度が 1.5 倍であったためである.

(2) コネクションプールは, クライアントとアプリケーションが通信する際のスレッドの数を管理する機能であり, CBoC タイプ 2 は, TCP バッファと同様に同一マシンで実行する AP とリソースを調整できるようにするため, プールする数をパラメータで変更可能としている. 該当するパラメータの値は, クライアントの設定で可能であるが, 商用運用の実績値であり, データの書き込みに効果がある値かどうかは確認できていなかった. このことから, 試験的にコネクションプール数を 100 から 300 へと変更して確認した. その結果, データ投入直後は 2 倍のスループットとなった. しかしその後, 確認できていない他の要因と思われるが, データ投入を継続した 2 日目には 1 倍のスループットに戻り, 変更した効果を確認するには至らなかった.

- ディスク I/O チューニング方式の考察について以下に示す. チューニングの調査は, ディスク I/O のチューニングが可能な, OS とアプリケーションとの観点で実施した. その観点とは,

  (1) マシンのハードウェアのディスク I/O

  (2) プロダクトが制御しているディスク I/O

  に対するチューニングである. マシンのハードウェアのディスク I/O については, パラメータチューニングによる効果の確認方法, プロダクトが制御しているディスク I/O については, 性能を確保した方法を具体的に以下に示す.

  (1) ハードウェアのディスク I/O に関しては, 転送の高速化のためにドライブが直接メモリにデータを保存する DMA(Direct Memory Access)について, ハードディスクの設定を検証した. しかし, Linux ではパラメータのチューニングの効果はなかった.

  (2) プロダクトが制御しているディスク I/O に関しては, データ登録時にバッファリングされたデータをディスクに書き出す Compaction の処理と, プロダクトが運用上必要なログの書き込み処理という 2 つのポイントがある. これらの処理は, 商用運用時の負荷では性能要件に影響を及ぼさないように設計されている. しかし, 2 章に示した性能要件を超えてデータ投入した場合, ディスク I/O が高くなり性能に影響を及ぼす. 設計書から確認できるが実際の処理を確認したところ, Compaction のデータ量はログのデータ量よりも倍以上多いことが確認できた.

検証データを高速登録して蓄積を加速する際に，安定して登録する考え方について述べる．検証データの高速登録の案に基づいて，書き込み速度を上げて事前登録を行い，スループットを測定した結果を図 16 に示す．13 時半頃（図の△部分）にデータ書き込み速度を 4 倍に上げた．スループットが上がったところで一定になると予想していたが，実際には 15 時前頃からスループットが低下し，16 時以降には性能要件よりも低いスループットになった．これは，データ投入の処理において，メモリ内にあるデータの量がある一定以上になると，圧縮してディスクへ書き出す処理（Compaction）が起動して高負荷になることが原因であった[3]．

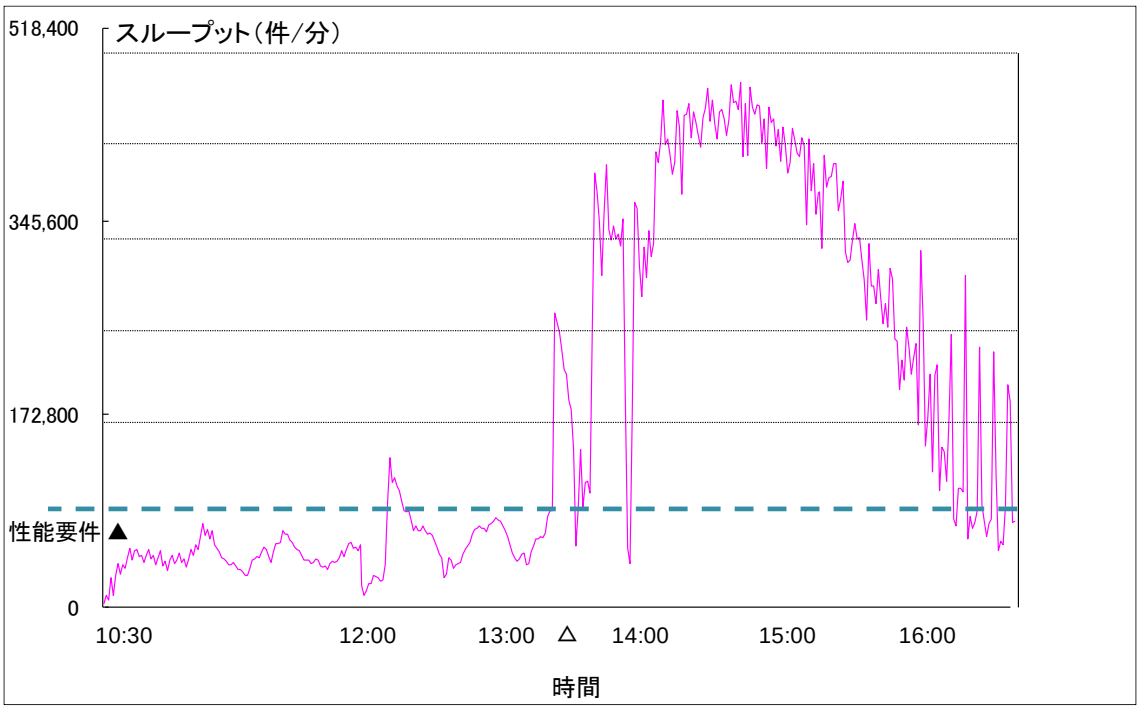


図 16．データ投入時の書き込みスループット低下

注：検証環境は以下のとおり

マシン：Intel(R) Xeon(R) CPU L5520 2.27GHz，NW（ネットワーク）：2G(Bonding)，OS：Redhat Linux

[3]商用運用時のシステムでは事前登録のような大量データの書き込みはない．そのため，Compaction の発生が分散され，ディスクへ書き込みが集中して高負荷になることがなく，スループットへの影響もない．

この Compaction は，2.2.3 項に記載したとおり，Minor Compaction， Merge Compaction，Major Compaction の 3 つがある．それぞれ，(1)サーバのメモリ上に記録されたデータサイズが一定数に達したときにディスクに保存する Minor Compaction，(2)Minor Compaction によって作成されたファイルの数が一定数に達したときに 1 つのファイルに統合する Merge Compaction，(3)サーバ上のデータ量が一定数に達したときにメモリとディスクのデータを再構築する Major Compaction である．1 回に処理されるデータ量の多さから，Merge Compaction と Major Compaction が処理性能への影響の支配項であることは，設計から自明であり実際の処理でも確認できた．さらに発生する頻度は Merge Compaction が Major Compaction が多く処理性能への影響の支配項であることも，設計から自明であり実際の処理でも確認できた．検証用のデータは，各サーバに均等に蓄積されるように調整されたデータである．そのデータを大量に投入する場合では，各サーバのデータサイズの一定数に同時に達してしまうため，Merge Compaction が複数のマシンで同時に発生することになる．Merge Compaction が，複数のマシンで同時に発生すると，I/O の負荷が高くなるため I/O の処理待ちとなり，クライアントの処理要求を受け付けることができない状態となった．

以上に示したとおり，検証データの安定的な高速登録という課題を解決するためには，そのボトルネックを解消することが必要である．データ登録は，検証準備において一時的に必要な処理である．そのため，プロダクトのプログラム修正と比べデータ登録後に元に戻すことが容易なパラメータ設定の変更による I/O チューニングによる手法が優位である．なお，プロダクトのプログラム修正は，設計変更であり検証を最初から実施しなおす必要がある．パラメータ設定の変更による I/O チューニングは，システムのログやネットワーク監視の結果や性能値を確認することによって，性能劣化を生じない設定をした．

- 適用可能な領域について以下に示す．2.2.3 項で示したように，大規模分散処理システムのデータ登録では，データの追加型で非同期に書き込むため，大量のデータを書き込むができる．手法 1 は，ネットワーク I/O とディスク I/O のチューニングというアプローチであり，これらは汎用的なものである．そのため，ネットワークとディスク I/O のボトルネックがある他のシステムにもこれらの提案手法が適用可能である．提案手法を適用することで，性能要件を越えた書き込みスループットと安定性を実現できる．

  しかし，手法 1 はベアメタルではなく VM（Virtual Machine）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しない場合に適用すると効果が少ない．また，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．

3.4 「検証実施時間の見積もり精度向上」(課題 2)と解決手法

従来の情報システムと異なり，大規模分散処理システムの検証には，時間を要する異常系の検証項目や性能測定の検証項目がある．時間を要する検証項目の実施時間を見積もるのが正確でないと，不足している誤差が数時間や日単位になり，検証のスケジュールを圧迫することになる．大規模分散処理システムでは，システム検証を実施するにあたり参考にできる類似システムが少ない．また，そのシステム検証は，数多くのプログラムで多数のマシンを動作させ，検証条件をチューニングしながら実施するため，プログラムの動作時間を厳密に見積もることが難しい．例えば，性能の上限値を測定する場合は，数多くのプログラムの影響から机上で算出した値と異なるため，あらかじめ計画しておいた検証条件以外に，新たに検証条件を加えるため，検証実施時間が長くなる．また，データのリカバリなどデータ量に依存した検証項目は，データ量が多いため検証実施時間が長くなる．こうしたことから，従来の情報システムにおけるシステム検証の実績を基に，大規模分散処理システムにおけるシステム検証の見積もり精度を上げることは困難である．

この問題は，検証項目を事前に実施して時間の見積もりを行うことで解決が可能だが，検証項目を絞って実施しないと長い期間がかかるという問題が残る．そのため，検証項目を絞った事前検証による見積もり作業時間の短縮を行い，見積もり精度を向上させることが必要になる．3.1.2 項に示した既存の手法で見積もりを実施することも可能であるが，初回の検証において見積もりの精度は上がらない．そのため，何度か検証で見積もりを実施することで見積もりの精度を上げることになる．したがって，1 項目の検証実施時間が従来の情報システムに比べ，検証実施時間が長い大規模分散処理システムでは 1 回で高精度の見積もりをする必要がある．

3.4.1 解決手法「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)

課題 1 の解決手法としては「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)が効果的であると考えた．なぜなら，手法 2 は，大量にリソースを使用する項目を分類して代表項目を選定し，事前検証することで，精度を上げるべき項目に着目し必要最低限の作業で見積もりをするため，課題を解決できると考えたからである．具体的に以下に説明する．

解決手法は，大量にリソースを使用する項目を分類して代表項目を選定し，事前検証することである．検証項目には，ユースケースにおける処理を確認する正常系と，サービス復旧が可能な準正常系，およびサービス復旧ができない異常系の 3 つがある．大項目を性能測定等の検証の種類とすると，事前検証の手法は以下の手順で実施する．

(1) 精度を上げるべき項目として，正常系は性能測定の大項目，準正常系はデータをリカバリ処理する復旧性能の大項目，異常系はラックダウンの複合・多重障害の大項目と大量にリソースを使用する項目を抽出する．

(2) 大項目では，多くの検証項目が同じ項目内に分類されてしまうため，その中から事前検証の項目となる一部の項目を選定することは，同じ分類の項目の内容をさらに詳細に確認する必要がある．この内容を詳細に確認する作業は，項目が多い大規模分散処理システムでは工数がか

かってしまう. 工数を少なくするため, 分類した大項目レベルをもう一段階詳細化した中項目レベルで選定する. 具体的には, 表 11 のように中項目レベルで検証観点が同じ項目において, システムへの書き込みや読み出しのツールが異なるなど検証の条件が異なるが,「ツールを実行する」や「メッセージを確認する」といった作業手順が同じ項目を類似の項目として分類する[4]. その類似の項目において手順の数の差が所要時間の差につながるという考えのもと, 手順が最も多い 1 項目を代表的な検証項目として選定する.

(3) 選定した代表的な数項目について, 所要時間を検証の事前に確認する. このことにより, 他の同じ手順の項目について, 項目の実施時間を見積もることができる.

事前に実施しない項目は, 従来の情報システムにおける検証の実施時間から類推[16]する[5]. この理由は, 1 項目あたりの検証実施時間が 30 分以内であり, 事前検証をする 1 項目あたりの検証実施時間が 1 時間程度の項目と比較して, 見積もり誤差は多くて 30 分で検証全体のスケジュールに与える影響が少ないからである.

表 11. 類似項目

大項目	中項目	小項目	検証条件
複合 ・多重障害	ラックダウン ・復旧	ラックダウン時の SquallWorker チャンクリペア (実 AP)	・3AP は稼働中 ・上記の状態で Squallworker が格納されたラックをダウンさせる
		ラックダウン時の SquallWorker チャンクリペア (TP)	・3AP を擬似した TP は稼働中 ・上記の状態で Squallworker が格納されたラックをダウンさせる
		ラックダウン時の SquallWorker チャンクリペア (背景呼なし)	・3AP／TP の稼働なし ・上記の状態で Squallworker が格納されたラックをダウンさせる

[4] 類似の項目とは, 具体的に表 11 の検証項目の場合では, どの小項目についても検証条件の「実 AP」,「TP」,「背景呼なし」が異なるだけであり, 手順が同じ項目のことである.

[5]見積もりの結果が予定した検証期間を超えた場合は, 従来の情報システムと同様, 計画を再度立案する.

3.4.2　　手法 2 の適用結果

適用結果については，見積もりに要する期間とぶれがどれだけなくなったかで評価した．CBoC タイプ 2 へ解決手法を適用し（図 15 の検証データの準備をした後の検証について示す），実際にシステム検証をした結果（後述する数段階の検証環境の割り当て手法における検証の一部）を 図 17 に示す．従来は後述するようにデータ蓄積や検証中の異常系の検証による遅延が発生していたが，事前検証の手法によって，検証項目の実施時間を見積もる精度が向上し，計画された期間内にシステム検証を完了することができた ．具体的な結果を以下に示す．

解決手法を適用し，大量にリソースを使う複合・多重障害と復旧性能の項目と，書き込みおよび読み出しの処理で時間がかかる性能測定の項目に関して事前検証した．検証実施時間の見積もり結果を表 12 に示す．この場合の事前検証の所用時間は，従来の情報システムで適用されている類推で見積もった以外の項目全てで 61.5 時間となった．これにより，1 日約 8 時間の稼動の場合では，8 日間（61.5÷8=7.7）の事前検証で，実施スケジュールを完成することができた．以降に手法の考察を示す．

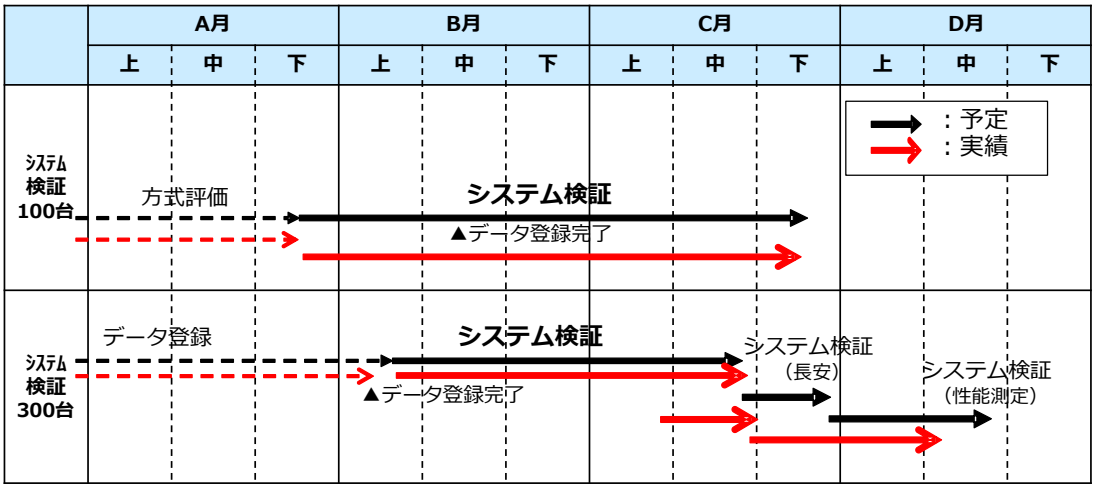


図 17. 提案手法による検証計画と実績

注：事前検証のスケジュールは A 月の前に実施．

表 12. 検証実施時間の見積もり

大項目名	中項目名	検証実施時間
性能測定	ランダムライト単体性能	8.5
	シーケンシャルリード単体性能	13.3
	ランダムリード単体性能	3.8
	複合性能	24
	時間区間指定読出しの高速化	(1)
系構成変更	フェールオーバ	(0.8)
	組み込み	(0.8)
単一障害	プロセスダウン	(2.6)
	NW 障害	(0.7)
複合・多重障害	ラックダウン・復旧	6.7
復旧性能	リカバリ	5.2
長安	長期安定	(168)
コマンド	ファイル操作	(0.5)
	ファイル運用コマンド	(0.5)
	分散テーブル操作	(1.5)
	運用ツール	(1.5)
ファイル更新	無停止アップデート	(0.5)
	プロトコルバージョンチェック	(0.8)

注:()内は解決手法以外(従来の情報システムで適用されている類推[14])で見積もった値

解決手法「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)について，以下の

- 課題 2 の「検証実施時間の見積もり精度向上」を解決する方法
- 適用可能な領域

の観点で考察する

- 課題 2 の「検証実施時間の見積もり精度向上」を解決する方法の考え方を以下に示す．過去の事例では図 18 に示すとおり，システム検証を実施した結果，ほぼ全ての検証項目で，当初の見積もり時間を越えていた．この検証は，新しいバージョンの OS(以降別 OS と呼ぶ)や VM 環境で性能測定を実施したもので，当初の計画ではシステム検証は見積もりどおりに進捗することを想定した．小項目 1 項目の実施時間は，準備を入れて 2 時間程度とし，中項目 1 つが 1 週間で完了する計画にした．

  しかし，これまで事前検証を実施して時間の見積もりをしなかったため，表 13 に示すとおり環境に起因する問題が発生(計 4 件)して進捗が 2 週間遅れた．発生した別 OS の検証の問題と VM 検証の問題は，ブラックボックスの OS に起因する問題であり，問題の切り分けや解析と対策は，プログラム単体の問題解析よりも時間を要したためであった．

  ここで，大量にリソースを使用する検証項目が実施時間を要していることに着目する．こうした項目は，中項目レベルで 6 項目程度の項目全体の約 1/3 を占めており，スケジュール全体に影響を与える．このうち検証の観点で類似の項目が 2 項目以上ある場合，そのうち 1 項目を代表的な検証項目として，スケジュール作成時に事前に検証を実施する．事前の検証を数項目に絞ることによって見積もり期間を短くする．この事前検証の見積もりは，中項目レベルで約 15 項目程度であり，1 日に 1 項目あたり 90 分以上要するため 3，4 項目程度の見積もりができる．その結果，スケジュール作成は 3，4 日程度で実施可能である．中項目レベルに着目する理由は，この粒度の分類で，検証実施時間に関わる検証の手順が同様なものとして見積もりを行うことが可能と考えたからである．

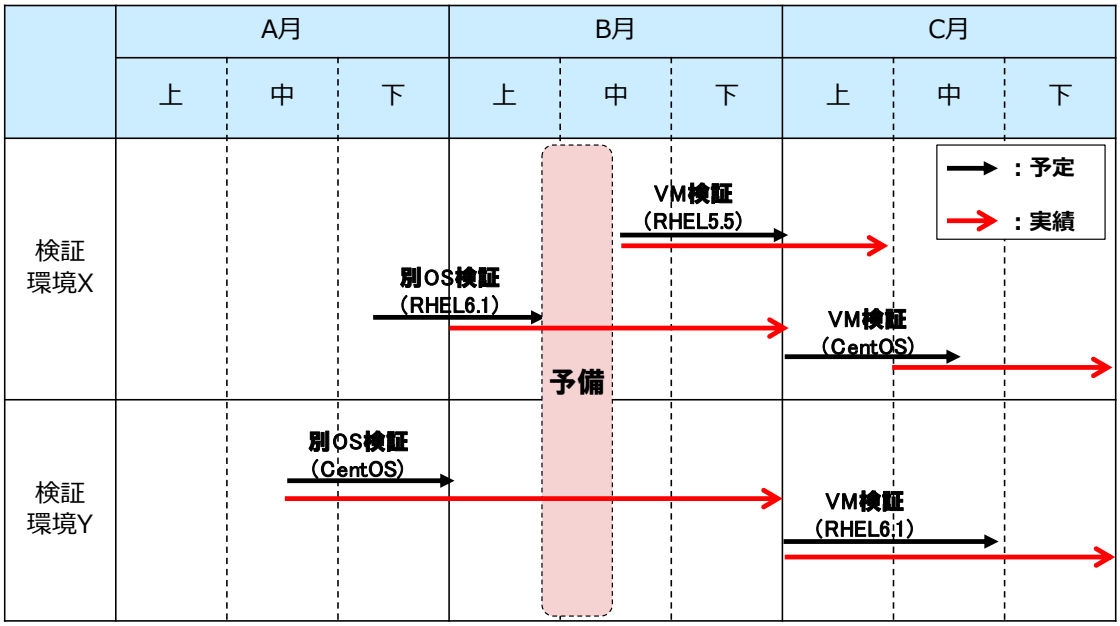


図 18. 従来手法による検証計画と実績

表 13. 各システム検証で発生した問題の例

別 OS 検証	・24 時間周期でスループットが低下する. ・シーケンシャルリード実施中にサーバ上で大量の Swap 発生のため遅くなる.
VM 検証	・VM に割り当てられているメモリが少ないため，サーバ上で Swap 発生し，分散ファイルの Worker と分散テーブルの AreaServer が停止. ・chunk is broken 発生. chunk の途中がオール 0 となっていた.

- 適用可能な領域について以下に示す. 大量にリソースを使用する項目のうち，代表的な検証項目について事前に検証を実施する手法 2 によって，1 回の見積もりの精度を上げ，スケジュールの精度を向上させることができる. 手法 2 は，リソースの大きい項目に着目した事前の検証というアプローチであり，これは汎用的なものである. この手法は，大量のデータを処理する場合，他のシステムにも適用が可能である.

  しかし，手法 2 は，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.

3.5 「効率的にバグ摘出するマシン台数分割による検証実施」(課題 3)と解決手法

大規模分散処理システムを検証する場合，品質確保の観点から，全ての検証項目をサービスイン時の数多いマシンを利用して検証するという手法がある．この場合，検証対象となる処理が複数プロダクトの連携で実現されているため，検証で問題が発生した場合には，プロダクトを連携させたまま原因を解析する必要がある．その解析に要する時間は，従来の情報システムよりも長くなる傾向があるほか，異常系の検証項目は，問題を対処した後の確認検証にさらに時間を必要とする．検証項目の内容に応じて，摘出する問題に合ったマシン台数で実施することができれば，問題の解析時間が最短になり，マシン占有時間も短縮することができる．

実作業を調査した結果，過去のシステム検証において発生した問題は 285 件であった．利用可能な全てのマシンを使用したと仮定し，これらの問題に対する平均対応日数を実際の検証における1日/件として順次処理すると，問題対応だけで約1年を要してしまう．問題が局所的で同時に対応ができたり，対応が容易な問題ばかりだとして，対応日数が 2 日/件となっても約半年を要する．

以上を考慮すると，問題摘出が可能で，かつ発生した問題の解析と改修した後の確認検証がしやすいように，必要最小限のマシン台数を検証項目に割り当てて，検証のスケジュールを作成することが，検証作業を調整でき期間内での検証を完了するための重要な課題となる．

3.5.1 解決手法「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)

課題 3 の解決手法としては「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)が効果的であると考えた．なぜなら，手法 3 は，マシン台数を分割し検証項目に割り当てることで，並行に検証が可能となり，検証の効率化ができると考えたからである．具体的に以下に説明する．

検証環境を，小規模は数 10 台，中規模は 100 台，大規模は数 100 台のオーダと想定して解決手法を決定する．大規模環境はサービスイン時のマシン台数規模，小規模環境は最小構成規模，中規模環境はその中間で運用上の都合によって環境が縮退した場合を想定する．大規模環境，中規模環境，小規模環境は 3.5.1 項の考え方に基づいて検証項目に割り当てる．

これらの検証環境は，問題解析や再現の検証にも利用する．ただし，検証条件や手順が一部異なるなどの類似の項目が 2 項目以上ある場合，代表的な 1 項目を実施して効率化を図る考えに基づいて，1 項目以外は大規模環境ではなく中規模環境を割り当てて，検証実施の効率化を図ることができる．同様に，中規模環境の項目は小規模環境を割り当てて，効率化を図ることができる．逆に，スケジュールで中規模環境が空いている場合，小規模環境を割り当てている項目は中規模環境で実施することもできる．

検証環境を項目に割り当てたのち，異常系の検証項目を中心に小規模から中規模，さらに大規模環境と段階的に検証を実施し，問題摘出と解析を効率化する．小規模環境は複数の環境で並行して実施ができる．小規模環境を割り当てた全体の 1/3 の項目は， n 個の環境で並行に検証すると 1/n の検証時間で完了することができる．中規模環境も並行できるとさらに検証を効率化することができる．

3.5.2　　手法 3 の適用結果

適用結果については，検証が充分に実施している状態を保ったまま実施時間が短縮できたかで評価した．CBoC タイプ 2 へ解決手法を適用し，実際にシステム検証をした結果を以下に示す．全ての問題を 300 台の大規模環境で摘出して対応した場合には 7 ヶ月を要していた．対して，提案の手法を適用した場合，図 19 のように検証と問題の解析を分割したそれぞれの検証環境で行った結果，5 ヶ月の期間内で品質が確保され，プロダクトアウトを可能にすることができた．具体的な結果を以下に示す．

解決手法に基づき，表 14 に示すとおり割り当てた検証項目に従ってシステム検証を実施した．小規模 30 台，中規模 100 台，大規模 300 台と，それぞれの検証環境でどう問題数が推移したかを図 19 に示す．システム検証の初期段階の小規模環境で発生する問題が多いものの，しばらくするとその数の増え方は緩やかになる．中規模環境でも同様に問題が発生するが，やはりしばらくすると増え方が緩やかになる．さらに，大規模環境でも一時的に問題が多く発生するが，以降は増え方が緩やかになっている．このように，あるマシン規模で問題が全て出尽くしたように見えても，規模を大きくするとまた新たな問題が発生する．このように，最初から中大規模環境で検証しないことで，小規模環境でデバッグ可能な問題を中大規模環境で行うことなく，解析や対処確認の再現試験の効率が下がるのを防ぐことができた．

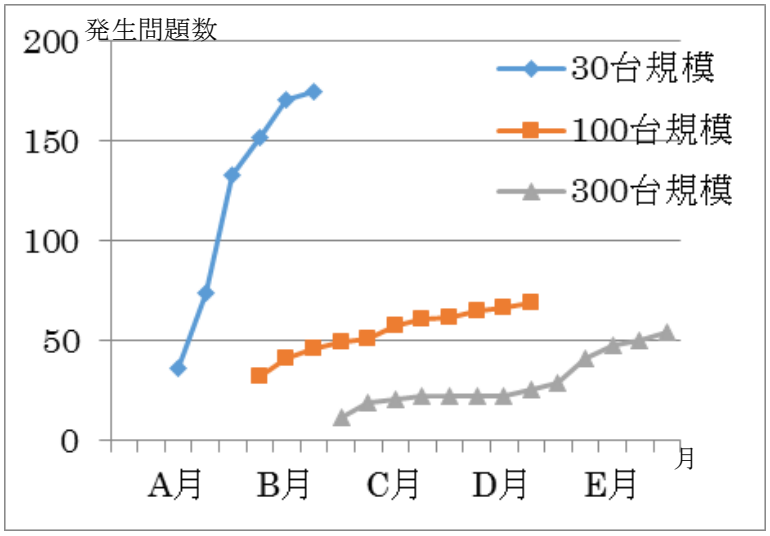


図 19．検証環境別の発生問題数の推移

注:グラフでは，大規模環境が中規模環境と並行して検証を実施しているように見える時期がある．これは，大規模環境で検証を開始したところ問題が多発し，中規模環境に切り替えて検証を継続したため，その間の数値がグラフ上に示されていることによるものである．また，小規模環境は複数の環境で並列に実施している．

表 14. 検証環境毎の項目割り当て

分類	大項目	大規模環境(項目数)	中規模環境(項目数)	小規模環境(項目数)
機能要件	単一障害	29	10	5
	複合・多重障害	1	28	42
	マシン系構成変更	9	11	1
	コマンド	12	1	2
	データ消失	1	1	1
	ファイル更新	1	5	1
	新規開発項目観点	2	2	0
	リグレッション項目	5	0	3
	過負荷・高負荷	2	2	1
	長期安定	1	2	0
非機能要件	性能	23	25	0
総計		86	87	56

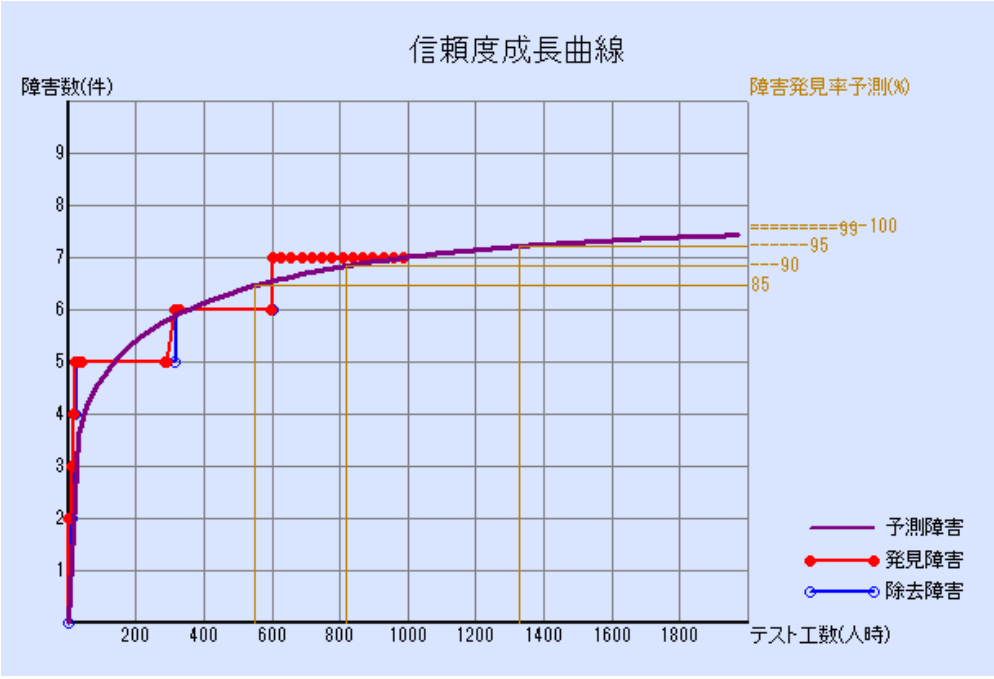


図 20. 問題の収束状況

検証が充分に実施できているかどうかの確認は，信頼度成長曲線により行った．ソフトウェアの開発では，ゴンペルツ曲線[6]やロジスティック曲線[7]などの信頼度成長曲線により検証の充分な実施と品質確認をしている[70]．これはソフトウェアの代表的な品質特性である信頼性を計量化して，検証における達成状況やプロダクトアウトの品質の確認をしていることによる．この信頼度成長曲線は，ソフトウェアの開発では最も妥当性や有効性が高いとされており[71][72]，ソフトウェア開発の検証工程や運用段階における障害をソフトウェアの信頼度成長過程として記述する．ここで障害とは，ソフトウェア開発プロセルで作りこまれたバグと呼ばれる欠陥や誤りによりソフトウェアが期待通りに動作しないことである．信頼度成長曲線を用いた確認方法は，検証実施時間とテストにより摘出された障害の数を信頼度成長曲線にあてはめ，曲線の収束値を求めて収束度合いにより判断する．提案の手法による検証の品質の状況を図 20 の信頼度成長曲線[8]で示す．図 20 の信頼度成長曲線が示すとおり，検出された障害は収束傾向にあることが分かる．

以上のように，134 日要していた検証は，手法 3 により 60 日で終了することができた．以降に手法の考察を示す．

---

[6] $y=ab^{e-cx}$ （a は障害の数，b とcは任意のパラメータ）で表される．

[7] $y=a/(1+be^{-cx})$ （a は障害の数，b とcは任意のパラメータ）で表される．

[8] 適用した信頼度成長曲線は，従来の代表的な信頼度成長曲線を統合した曲線であり，以下の式で表される．

$d(y+\sigma)/dt\cdot(y+\sigma)^{(\gamma-1)}=\alpha\cdot e^{-\beta t}$

(A) $\beta>0$，$\gamma>0$ の場合

$y=N(1-a\cdot e^{-\beta t})^c-\sigma=N\{(1-a\cdot e^{-\beta t})^c-(1-a)^c\}+y_0$.

ただし，

$N=\{(\alpha\cdot\gamma+\beta(y_0+\sigma)^\gamma)/\beta\}^{1/\gamma}$,

$a=\alpha\cdot\gamma/(a\cdot\gamma\ \beta(y_0+\sigma)^\gamma)$,

$b=\beta$,

$c=1/\gamma$.

(B) $\beta>0$，$\gamma=0$ の場合

$y=N\exp(-ke^{-\beta t})-\sigma=N\{\exp(-ke^{-\beta t})-\exp(-k)\}+y_0$.

ただし，

$N=(y_0+\sigma)\exp(\alpha/\beta)$,

$k=\alpha/\beta$,

$b=\beta$.

(C) $\beta>0$，$\gamma<0$ の場合

$y=N/(1+\phi e^{-\beta t})^{cc}-\sigma=N\{1/(1+\phi e^{-\beta t})^{cc}-1/(1+\phi)^{cc}\}+y_0$.

ただし，

$\phi=-a>0$，　$cc=-c>0$.

解決方法「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)について，以下の

- 課題 3 の「効率的にバグ摘出するマシン台数分割による検証実施」を解決する方法
- 適用可能な領域

の観点で考察する.

- 課題 3 の「効率的にバグ摘出するマシン台数分割による検証実施」を解決する方法の考え方を以下に示す. 課題を解決するために，「(1)検証項目に対応したマシン台数の割り当て方法」「(2)検証の実施方法(実施順番)」の 2 つの観点から問題対応時間を短縮する方法を検討した. それぞれの方法について以下に述べる.

(1) マシン台数の分割と検証項目に対応したマシン台数の割り当て方法

大規模でないマシンで実施できる検証項目は，ユースケースであっても 2 プロダクト間に閉じ，大容データを必要としない検証項目である. 2 プロダクト以外のプロダクトは最小構成でよい. また，2 プロダクト間に閉じる検証項目では，運用確認や性能といったシステム全体の要求仕様を検証する必要がない. そのため，サービスイン時の大規模マシン構成の検証は不要になる. 以上より，システム全体の要求仕様を確認する検証は大規模マシン構成で実施し，それ以外の検証は大規模でないマシン構成で実施できることから，以下の 3 つの考え方でマシンを割り当てる.

(1) 小規模構成は，マシン台数を数十台規模とし，台数に依存しない機能要件と多重障害の検証を行う. この構成は，検証の実施，問題の解析がいずれも容易で，少ないデータ量で実施できる.

(2) 中規模構成は，マシン台数を百台規模とし，初期の運用時のマシン台数が少ない状態や，古いマシンの入れ替えなど運用上の都合により環境が縮退している場合を想定し，機能検証とマシン台数に依存しない非機能要件(可用性)の検証を行う.

(3) 大規模構成は，マシン台数を数百台規模とし，商用運用環境時の非機能要件(性能，信頼性，安定性)や，データ量・台数依存項目の検証を行う. この構成はサービスの確認や検証に適している.

例えば，CBoC タイプ 2 の分散テーブルは 5 台，分散ファイルは 2 台，分散ロックは 1 台という最小台数で構成することができる. この結果，分散テーブルと分散ファイルの結合検証は，最低 7 台のマシン台数で実施可能になる. こうして，検証項目に対して問題摘出に見合ったマシン台数で検証を実施できることになる.

(2) 検証の実施方法(実施順番)

(1)による検証項目に応じたマシン台数を割り当てた後，小規模から大規模へ段階的に検証を実施する. この方法では，摘出する問題に合わせた規模での検証実施が可能となり，問題の摘出と対処に要する期間を短縮することができる.

(3) 大規模構成の未実施項目に対するサンプリング検証

全ての検証項目をサービスイン時の大規模なマシン台数で実施していないことから，未実

施項目を大規模の環境でサンプリング実施する手法によって，最終的な品質の確認作業を期間内に行うことが必要である[9].

- 適用可能な領域について以下に示す．解決手法は，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．適正なマシン台数を検証項目に割り当てるこれらの手法は，大規模分散処理システムの検証を効率化することができ，他の大規模分散処理システムにも適用が可能である．さらに，小規模環境と中規模環境を同時に利用して並列に検証項目を実施すれば，より一層の効率化が可能である．

  しかし，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が，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．

[9] (2)の検証で充分であるが，実際の検証では大規模構成で実施していない検証項目を(3)のサンプリングして検証することにより，最終的にプロダクトの品質を確認した．

4. テストケース生成アクティビティの課題と解決手法

実施されるテストレベルおよびテスト技法の作成という，テストケース生成アクティビティにおいて主要な課題は「大量な検証項目の項目削減と期間内の検証完了」（課題 4）である．その課題 4 に対しては「システム特徴と問題発生傾向による項目削減と重要度による期間内検証手法」（手法 4）で効果がある．

本章では，テストケース生成アクティビティにおいて，検証項目の重要度による検証精度と検証期間とのトレードオフをコントロールする解決手法を課題とともに説明する．4.1 節で先行研究，4.2 節で課題を示した後，4.3 節では課題 4 に対して効果のあった手法 4 について説明する．

4.1 テストケース生成アクティビティの課題に関する先行研究

本節では，1.2 節に示したテストケース生成アクティビティの課題「大量な検証項目の項目削減と期間内の検証完了」で利用される技術について関連する研究を示す．2.3 節で示したとおり，従来技術の論理網羅テスト，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，ランダムテストは，大規模分散処理システムで発生した課題以外において適用できる．しかし，発生した課題においては，以降のようにそのままでは適用ができない．

4.1.1 大量な検証項目の検証の効率化に関する先行研究

大量な検証項目の検証の効率化研究として，ソフトウェアテストの技法として検証項目を削減する方法と計算機で効率的に検証する方法とがある．ソフトウェアテストの技法としては，Cem Kaner ら[41]や，Glenford J. Myers[42]やボーリス・バイサー[43]が，同値分割や境界値分析等による項目の削減方法を記している．この手法は削減の数が限られるため，項目数が組み合わせで多くなることはない場合に効果がある．しかし，大規模分散処理システムの検証の項目は，その手法で効率化を行ってもなお項目数が多く，全ての項目の検証は検証期間内に実施できない．また，準正常系や異常系の手順が複雑でデータのリカバリが発生する長時間の項目には，同様な項目が少なく適用できない．理由は，組み合わせるプロダクトや検証条件が異なっているため，大幅な削減ができないからである．

計算機で効率的に検証する方法としては，モデル検査の手法がある．モデル検査とは，システムの状態遷移を表すモデルに対して，全ての状態遷移が成り立つかどうかを計算機で網羅的に検証する技術であり，E.M. Clarke ら[58]により示される形式手法の一つに位置づけられる．この手法は，状態遷移による組み合わせが多くない場合に効果がある．しかし，協調動作が満たすべき性質をしらみつぶしに確認するため，大規模分散処理システムの状態遷移の組み合わせによる膨大な項目を，システム検証の限られた期間内で全て実施するのは現実的ではない．

4.2 テストケース生成アクティビティの問題と具体的な課題

1.2 節で示したとおり，プロダクトが複数あり確認する条件の数が多いため，検証項目数は組み合わせ的に膨大になる問題がある．検証項目を精査する既存の手法としては，同値分割と境界値分析等の手法があるが，その精査により項目の削減を行ってもなお項目数が多く，全ての項目の検証は数週間の期間を要する．これは，組み合わせるプロダクトや検証条件が異なっているため，上記の精査の手法では大幅な削減ができないためである．

上記に示したとおり，大規模分散処理システムのテストケース生成アクティビティでは，作業をコントロールすべき問題が発生しており表 15 に示す．発生した問題は，大規模分散処理システムで検証項目を網羅的に作成すると，膨大な量の項目数となるため，検証に時間を要することである．その具体的な課題は，網羅的な検証による品質を確保したまま，大量な検証項目の項目削減と計画した期間内での検証完了である．以降から，作業をコントロールできる解決手法について詳細に説明する．

表 15. テストケース生成アクティビティで発生する問題の概要と具体的な課題

作業分類	問題の概要	具体的な課題 （具体的な内容）
項目作成	網羅的に作成された検証項目は項目数が多くなり，計画より 2 週間以上の検証期間を要する	課題 4: 大量な検証項目の項目削減と期間内の検証完了 （網羅的な検証項目の半数以下の項目削減）

4.3 「大量な検証項目の項目削減と期間内の検証完了」(課題 4)と解決手法

テストケース生成アクティビティでは，JIS X 0129-1(ISO9126-1) [87]で示された品質特性から，システムの特徴と対応する検証観点を抽出する必要がある．その理由は，網羅的に検証観点を抽出して項目作成する手法では，プロダクトの数や状態遷移の組み合わせで検証項目を作成することから，項目数が膨大になるからである．膨大な数のままの項目数では，検証期間内に完了できない．したがって，作成する検証項目は，重要度を決め，運用時のユースケースに従ったシステムの処理を中心に絞り込みをする．そしてその項目を検証期間内に検証する．これが，検証精度と検証期間とのトレードオフをコントロールするための重要な課題となる．

4.3.1 解決手法「システム特徴と問題発生傾向による項目削減と重要度による期間内検証手法」(手法 4)

課題 4 の解決手法としては「システム特徴と問題発生傾向による項目削減と重要度による期間内検証手法」(手法 4)が効果的であると考えた．なぜなら，手法 4 は，システム特徴と問題発生傾向で重要度を決め，重要度順に検証することで課題が解決できると考えたからである．システムの特徴に基づく削減手法(「資源効率性」/「障害許容性」/「回復性」の観点による検証実施)と，問題の要因分析に基づく削減手法(問題分析による検証項目へのフィードバック)と，重要度の順の検証について，それぞれ具体的な解決手法を以下に示す．

- システムの特徴に基づく削減の手法(「資源効率性」/「障害許容性」/「回復性」の観点による検証)：

実際に実施した検証項目を表 16 に示す．検証項目数は，網羅的に抽出した時点では約 600 項目あったが，運用時のシステムの処理を確認し，重要度を決定して抽出した「障害許容性」，「回復性」，「資源効率性」の検証観点において精査をして検証項目を作成した結果，約 260 項目となった．

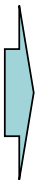
従来の精査は，「同値分割」や「境界値分析」等を用いて実施するが，通常は項目抽出時にすでにこの精査は行われており，更なる精査を行う必要があった．当初は「システム起動・停止」，「上位アプリからの書き込み」，「上位アプリからの読み込み」，「コマンド確認」，「正常系負荷」，「スケールアウト」の内容を網羅的に抽出していたが，運用時のシステムの処理を確認した結果，必ずしも動作することはない処理があることと正常系機能試験で確認を兼ねることができるため，大幅に項目を削減する．一方，「上位アプリのプロセス中断」，「フェールオーバ」，「CBoC プロセスダウン」については，「障害許容性」，「回復性」の観点で項目を追加した．最後に，「組み込み」については「資源効率性」の観点で項目を追加する．

以上により，ユースケースの観点と，大規模分散システムの特徴に基づく「障害許容性」，「回復性」，「資源効率性」の重要な観点で精査を行い，大規模分散処理システムに特化した項目が完成する．

表 16. 精査をした大規模分散処理システムの検証項目

大項目	中項目	
ユースケース機能試験	システム起動・停止	次版削除
	上位アプリからの書き込み	次版削除
	上位アプリからの読み込み	次版削除
	コマンド確認	次版削除
高負荷，異常系複合試験	正常系負荷	次版削除
	異常系複合	
	機能試験用負荷	次版削除
性能		
長期安定試験	スケールアウト	次版削除
	長期安定性試験	
運用コマンド		

約600項目



大項目	中項目	
ユースケース機能試験	正常系機能	今回追加 次版削除
	上位アプリのプロセス中断	
CBoCの系構成変更	フェールオーバ	今回追加
	サーバ組み込み	今回追加
CBoCの単一障害	CBoCプロセスダウン	今回追加
	NW障害	今回追加
複合・多重障害	CBoCフェールオーバ（1次、2次）	今回追加 次版削除
	CBoCプロセスダウンの組み合わせ	
性能		
長期安定試験		
運用コマンド		

約260項目

● 問題の要因分析に基づく削減手法(問題分析による検証項目へのフィードバック):

大規模分散処理システムに特化した項目で検証した際に問題が発生する. 発生した問題を要因分析し, 分析の結果で得られた有効な検証観点を表 16 の右の検証項目にフィードバックした結果を表 17 に示す. 表 17 の検証項目へのフィードバックは,「資源効率化」のフィードバックとして「上位アプリ性能」,「性能」の項目の強化を実施し,「障害許容性」,「回復性」のフィードバックとして「CBoC プロセスダウンの組み合わせ」,「運用コマンド」の強化を実施する. 一方,「CBoC フェールオーバ」は, プロセスダウンの組み合わせとラックダウンの検証で確認を兼ねることができるため, 項目を削減する.

表 18 の検証項目数の割合では障害・復旧試験が大半を占めており, 従来の検証における割合の, 正常系 60%, 異常系 15%, その他が 25%とは異なる. これは, 障害発生プロセスや障害種別, 同時障害発生数などの組み合わせを基に検証項目を抽出しているためである. システム特徴の観点との対応については,「資源効率性」は性能・高負荷試験や長期安定性試験,「障害許容性」「回復性」は障害・復旧試験, 長期安定性試験に対応している.

表 17. 問題の要因分析によるフィードバックを行った項目

大項目	中項目	
ユースケース機能試験	正常系機能	今回追加 次版削除
	上位アプリのプロセス中断	
CBoCの系構成変更	フェールオーバ	今回追加
	サーバ組み込み	今回追加
CBoCの単一障害	CBoCプロセスダウン	今回追加
	NW障害	今回追加
複合・多重障害	CBoCフェールオーバ（1次、2次）	今回追加 次版削除
	CBoCプロセスダウンの組み合わせ	
性能		
長期安定試験		
運用コマンド		

約260項目

大項目	中項目	
ユースケース機能試験	上位アプリ性能	今回追加
CBoCの系構成変更	フェールオーバ	
	サーバ組み込み	
CBoCの単一障害	CBoCプロセスダウン	
	NW障害	
複合・多重障害	CBoCプロセスダウンの組み合わせ	強化（フェールオーバと組み合わせ）
	ラックダウン・復旧	今回追加
性能	データを変化させた場合の測定	強化
長期安定試験		
運用コマンド		強化（異常系と組み合わせ）

約130項目

表 18. 検証実施項目数の割合

検証項目分類	主な試験内容	項目数割合
正常/準正常系	ユースケースでの読み書き，読み書き中の保守コマンド投入	23%
障害・復旧	障害プロセス・障害種別・試験時の負荷の組み合わせ（二重/三重故障含む），大規模故障，障害復旧中の障害	43%
性能・高負荷	読み書き，NW・DISK 高負荷，障害復旧性能	25%
長期安定性	長期運転時の障害・復旧（保守コマンド投入含む）	8%

- 重要度の順の検証:

  検証の実施は，前述した重要な観点による絞り込みを行った項目に基づき，重要度の順に計画した期間内に検証する． 重要度の順は，システムの特徴である「障害許容性」，「回復性」，「資源効率性」の検証観点をまず検証し，次にユースケースとしてシステムが必ず動作する観点の項目を検証し，最後にユースケースとして特定のパターンで動作する観点の項目を検証する．これは，後述する表 21 に示した◎の検証項目をまず検証し，次に〇の項目を検証し，最後に△の項目を検証することである．このように，重要な項目から検証を実施することで，検証期間内に品質を確保して検証を実施することができる．

4.3.2　手法 4 の適用結果

適用結果については，検証に要する期間で評価した．また，検証が充分に実施できているかどうかの確認は，3.5.1 項で述べたように，ソフトウェアの開発では最も妥当性や有効性が高いとされている信頼度成長曲線の収束値の収束傾向により評価した．実際の検証で確認した結果，重要度を決め，大規模分散処理システムの特徴である「分散かつ大規模」と「可用性」を中心に，絞り込みを実施することが重要であることが判明した．また，検証によって問題を摘出した場合，問題を摘出できた検証項目を重点化することで，システムの特徴による絞り込みに加えてさらに，計画した期間内に重要度順に検証することで，検証精度と検証期間とのトレードオフをコントロールできた．具体的な結果を以下に示す．

検証環境としては，300/100/30 台の 3 環境（15～20 サーバ/1 ラック）で検証を実施し，それぞれ 300TB/7.4TB/21TB 規模のデータを登録した状態で検証を実施した．長期安定性試験や，障害系機能試験における負荷としては，1 サーバあたり 0.41MB/sec（ランダムライト），6.9MB/sec（シーケンシャルリード），2.1KB/sec（ランダムリード）で実施した．検証項目に関しては図 21 に示すとおり，30 台環境では，ほとんどが障害系観点の検証であり，小規模環境による設定変更の容易さを活かし，短時間で数多くの検証を実施した．100 台環境では，当初の実商用想定環境であり，正常機能の確認とともに，運用時に発生すると想定される準正常・異常系機能の確認を行った．300 台環境では，性能（資源効率性）を中心に正常機能（その他観点），準正常・異常機能（障害許容性・回復性）の確認を行った．

その結果，検証に要した期間は，削減をしない方法では，網羅的な約 600 項目の検証で収束に 160 日要していた．対して，提案の手法を適用して，システムの特徴でない観点で組み合わせ的に多くなっていた項目や，問題の発生しない項目を削減した検証では，85 日で完了ができた．

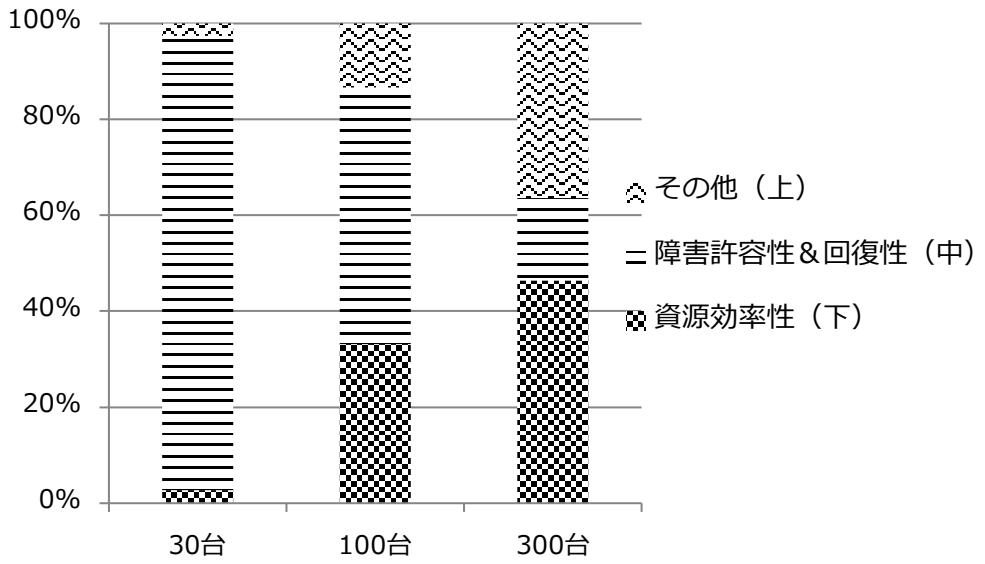


図 21. 30/100/300 台の環境の項目割合

表 16 で示した 260 項目のシステム検証の実施で得られた，信頼度成長曲線を図 22 に示す．3.5.1 項で記述したように，信頼度成長曲線は，検証実施時間とテストにより摘出された障害の数を成長曲線にあてはめて作成する．検証が充分に実施できているかどうかは，曲線の収束値を求めて収束度合いにより判断する．図 22 では，横軸に検証実施時間を縦軸に障害の数を設定し曲線の収束値を求めた結果，収束傾向にあると判断できた．プロダクトアウト後の実運用では，運用開始後1年間の障害の数が 2 件であり，サービスが継続できない問題は発生することなく安定運用ができた．このことから，提案する観点の絞込みで網羅的な項目抽出と同等な検証が可能であることが判明した．

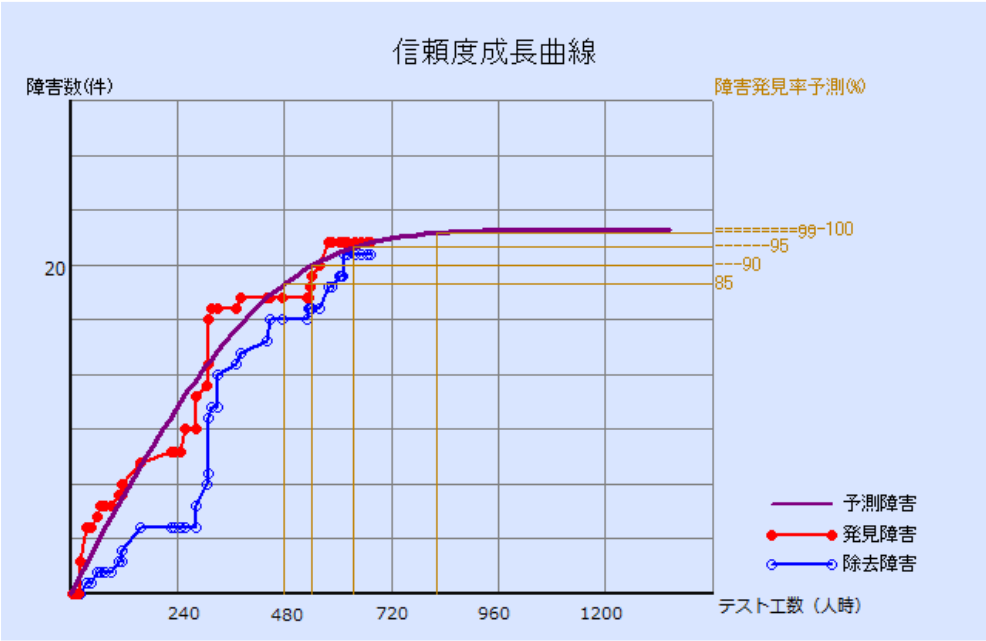


図 22. 表 16 の完成項目での信頼度成長曲線

表 19. 検証環境別の問題発生割合

	資源効率性	障害許容性 &回復性	その他	平均
30 台	0.000	0.083	1.000	0.105
100 台	0.071	0.132	0.000	0.094
300 台	0.062	0.042	0.000	0.036
平均	0.065	0.102	0.014	0.069

また，表 17 で示した130項目のシステム検証の実施で得られた，信頼度成長曲線を図 23 に示す．信頼度成長曲線の収束値を求めた結果，収束傾向にあると判断できた．またプロダクトアウト後の実運用では，運用開始後1年間の障害の数が 1 件であり，サービスが継続できない問題は発生することなく安定運用ができた．この結果から，提案する注力すべき項目に絞った検証では，観点の絞り込みからさらに効率的に検証できることが判明した．環境別の問題発生件数では，表 19 を見ると検証1項目あたりに発生する平均問題数が 0.069 件であり，30 台・100 台では障害許容性と回復性の観点に対応する問題が平均より多く抽出できることになる．また，資源効率性についても，100 台・300 台環境で平均程度の問題数を抽出できる．これは，狙い通りに効率良く検証が実施できることを示している．一方，その他の観点では，ログの設定の問題摘出のみであったことから，システム検証における重要性は低い．以降に適用した手法の考察を示す．

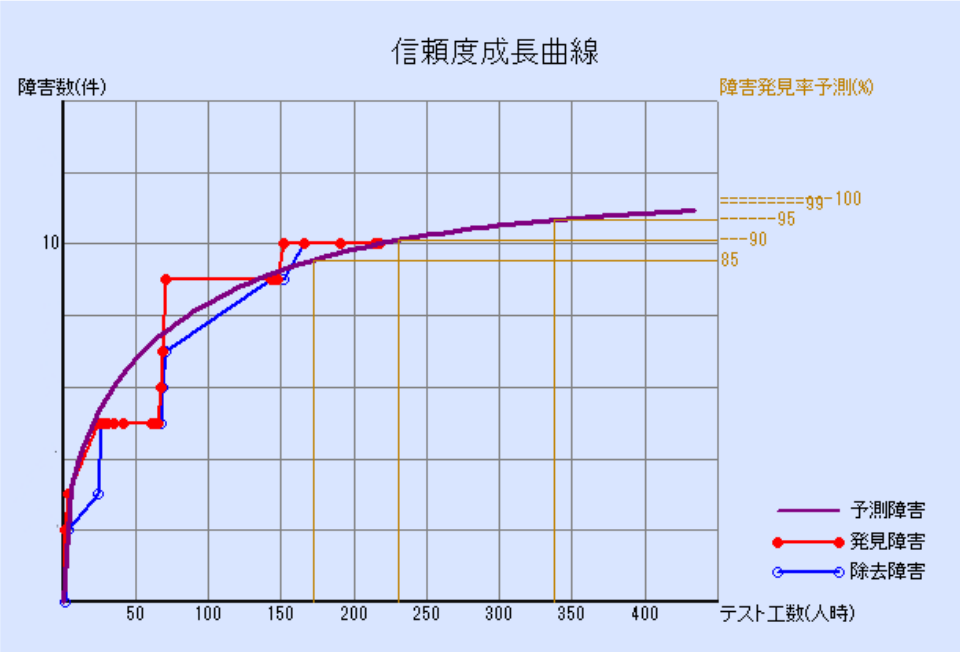


図 23. 表 17 の完成項目での信頼度成長曲線

解決手法「システム特徴と問題発生傾向による項目削減と重要度による期間内検証手法」(手法4)について，以下の

- システムの特徴に基づく削減手法(「資源効率性」/「障害許容性」/「回復性」の観点による検証)
- 問題の要因分析に基づく削減手法(問題分析による検証項目へのフィードバック)
- 手法の適用方法
- 適用可能な領域

の観点で考察する.

- システムの特徴に基づく削減手法(「資源効率性」/「障害許容性」/「回復性」の観点による検証)の考え方を以下に示す. テストケース生成のアクティビティにおいては，検証項目を作成するため，重要とする検証観点を抽出する必要がある. 検証観点は，JIS X 0129-1(ISO9126-1)で示された品質特性の中からシステムの特徴と対応する検証観点を抽出する必要がある. 具体的には大規模分散処理システムの場合， 2.2 節で述べたとおり，大規模分散処理システムの特徴である「スケールアウト性」と「耐障害性」を検証する内容が含まれるべきである. 以下に各特徴に対する検証観点を述べる.

  「スケールアウト性」の検証観点としては，「資源効率性」が主要な品質特性である. その理由は，システムを構成する多数のサーバ上では，要求されるデータ量や処理量が均等に分散することでスケーラブルに適応することから，「資源効率性」としてシステムの構成上で処理の集中する箇所がボトルネックになっていないかを検証し，リソース全体を効率的に利用して所望の性能が得られるか否かを検証するからである． 特に，データの蓄積量の増加に伴って，サーバの台数を増やした際に，それに応じた処理性能の向上が得られるか否かの検証を行う必要がある.

  「耐障害性」の検証観点としては，「障害許容性」，「回復性」が主要観点である. その理由は，システムが動作するサーバやネットワークが故障した際には，システム全体が正常に機能を提供し続けるという，「障害許容性」を検証する必要があるからである. つまり，一部のサーバやネットワークが故障で機能しなくなった際に，システム全体として，正常に機能を提供し続けることを確認する検証を行うことがポイントとなる. ただし，システムの基幹となる役割を担うマシンが故障を受けた場合には，スタンバイ中のサーバが機能を有効にするまでの間，短期間ではあるが，サービスが停止することを考慮する必要がある. 一方，「回復性」の検証としては，上記の場合において，サービスが停止する期間，サーバの役割が切り替わる時間，さらには，これらの機能が正しく動作するかなどの検証を行う必要がある.

  運用時のシステムの処理を確認して検証項目を作成し，縦軸を検証項目の大項目とし，横軸を JIS X 0129-1(ISO9126-1)で示された品質特性とした表に分類した. その結果，従来のシステム検証における分類の表 20 と異なり，大規模分散処理システムの分類は，表 21 の分類と

なった. ◎や○の付いている欄は, ユースケースとしてシステムが必ず動作する観点であり, △の付いている欄は, ユースケースとして特定のパターンでのみ動作する観点である. ◎はシステムの特徴に基づく観点であり, 重点化する項目である. 機能を確認する検証項目は, 機能の正常性, 準正常, 復旧障害および性能・高負荷の項目を網羅的に作成している. その上で, 必ず動作する処理の検証項目は網羅的な項目の作成とし, 特定のパターンでのみ動作する処理の検証項目は必要な項目だけとした. その結果, 精査した後の項目数は, 項目を網羅的に抽出した場合の半数以下となった.

表 20. 従来のシステム検証における検証項目抽出

	機能性	信頼性	使用性	効率性	保守性	移植性
正常系・準正常系	○	○	○	○	○	○
障害復旧	○	○	○	○	○	○
性能・高負荷	○	○	○	○	○	○
長期安定	○	○	○	○	○	○

表 21. 大規模分散処理システムにおける検証項目抽出

	機能性	信頼性	使用性	効率性	保守性	移植性
正常系・準正常系	○	△	△	△	△	△
障害復旧	△	◎	○	△	○	△
性能・高負荷	△	△	△	◎	△	△
長期安定	△	◎	○	△	○	△

- 問題の要因分析に基づく削減手法(問題分析による検証項目へのフィードバック)の考え方を以下に示す. 問題の要因分析に基づく削減手法は, 検証により発生した問題の検証項目へのフィードバックにより実現する. 発生した問題の検証項目へのフィードバックは, 実際に検証で発生した問題に関して要因を分析し, その問題が抽出可能となる検証項目を作成し, 元の検証項目に反映するということである. 前述した「資源効率性」/「障害許容性」/「回復性」の観点による検証実施によって抽出できた問題を表 22 に示す. ここでは, 問題発生要因を分析し, その分析結果から検証項目を精査する方法について示す.

表 22. 抽出した具体的問題

サービスの不具合	抽出した問題
正常に動作しない	マシン不具合
上位 AP が停止	通信途絶時のリカバリ不具合による管理情報の欠如
登録のスループットが低下	分割の閾値が大きすぎた
サーバの切り替えが突然発生	サーバ側での旧接続に関する後処理の抜け
圧縮の処理のタイムアウトが発生	ログレベルの設定誤り
登録のスループットが低下	通信途絶時でのデッドロック
登録したデータが見つからない	デッドロック
復旧処理時にデータが破損	他の管理情報の混在による影響
サーバ追加ができない	復旧処理の改修漏れ
登録のスループット低下	ロック取得による処理遅延
サーバ追加ができない	古いセッション管理情報を使ってしまっている
サーバの切り替わりが突然発生	リソース不足による切り替わり発生
収集したデータが見つからない	通信ディスクリプタのマップからの削除漏れ
障害復旧の処理が完了しない	要求処理の滞留
上位 AP が停止	リカバリが完了する前にクローラーのリトライアウト
障害復旧できずにプロセス停止	差分データが大きい場合の処理不具合
上位 AP が停止	MapReduce でのタイマとの差異
障害復旧の処理が完了しない	ロック期間の調整不足
上位 AP が停止	システムコール不具合
障害復旧の処理が完了しない	ハーフオープンされた File Description 処理の考慮漏れ

表 23. 発生した問題の主な要因

検証観点	要因分類	要因の概要
資源効率性	1. 処理集中	・大量データを分散させる閾値誤りによる処理集中
	2. 排他処理	・異なるデータ処理の競合 ・通信処理でのデッドロック
	3. リソース設定	・リソース設定誤り
障害許容性	1. 大規模障害	・大規模障害による処理遅延
	2. 障害復旧タイミング	・接続情報のキャッシュ削除漏れ ・複製データへのアクセスタイミングの考慮漏れ
	3. 障害復旧パターン	・通信障害パターン考慮漏れ ・旧接続のリンク切り漏れ ・想定外の戻り値に起因するリトライ動作誤り
	4. OS	・システムコールの動作不良
回復性	1. キャッシュ	・キャッシュの削除漏れ

発生した問題に関して要因の分析を実施した結果を表 23 に示す. 以降では, 各要因について詳細を述べた後に, 分析について述べる.

(1) 要因分類①-1 処理集中(資源効率性)

アクセス集中の問題では, 分散テーブルのマスタが処理を担当するワーカを決定し, その情報をテーブルの META に保持するが, その META へアクセス集中が見られた. META は, データ量・サーバ台数の増大で応答が遅くなるため, あるデータ量の閾値で分散する必要がある. しかし, 検証で設定した閾値ではアクセスを分散する前に応答が遅くなり, アクセスが集中した状態が継続してしまった.

(2) 要因分類①-2 排他処理(資源効率性)

大量データ処理との競合の問題では, 分散ファイルのバックグラウンド処理であるデータの圧縮処理と書き込みとの競合が発生した. この場合, データ量が多いほど圧縮する時間も長くなり, 競合の発生時間も長くなっていた.

一方, 通信処理におけるデッドロックの問題では, 分散テーブルへのリクエストが強制切断された際にファイルを削除するため, 書き込みとのデッドロックが発生した. 問題の発生はタイミング依存であった.

(3) 要因分類①-3 リソース設定(資源効率性)

リソース設定誤りの問題では, 長期運転中に分散ロックでメモリプールサイズの不足が発生した.

(4) 要因分類②-1 大規模障害(障害許容性)

大規模障害による処理遅延の問題では, 多数のワーカのダウンが原因で, 分散ファイルマスタの管理情報からダウンしたワーカ情報の削除が遅れた. そのため, マスタからダウンしたワーカ

に要求が送られてしまい，ダウンしたワーカは要求を処理できず，マスタで処理中の要求が増加し，マスタがビジー状態となった．

(5) 要因分類②-2 障害復旧タイミング（障害許容性）

接続情報のキャッシュ削除漏れの問題では，クライアントでキャッシュしている以前の接続情報（ソケットのファイルディスクリプタと IP アドレス）が削除されないタイミングがあり，同じファイルディスクリプタに対し 2 つの IP アドレスが存在してしまった．

一方，複製データへのアクセスタイミングの考慮漏れの問題では，分散ファイルのクライアントがワーカの位置情報のキャッシュを参照した際，ダウンした最近傍のワーカにアクセスし，次のワーカに読みに行くまでのリトライで処理遅延が発生した．これは，キャッシュがすぐに削除されない読み方だったために問題が発生した．

(6) 要因分類②-3 障害復旧パターン（障害許容性）

通信障害パターン考慮漏れの問題では，udp が通信可能であるのに対し，tcp のみ通信不可であったため，分散ロックは udp の死活監視でサーバが正常に動作していると判断した．しかし，分散テーブルや分散ファイルは tcp の通信障害が原因で正常に動作しなかった．また，旧接続の切り漏れの問題では，分散ロックのクライアントとサーバ間の死活監視で接続が一時的に切れたため，再接続を開始したが，クライアントから遅れて届いた FIN パケットによって接続が再度，切断されてしまった．

最後の想定外の戻り値に起因するリトライ動作誤りの問題では，コアスイッチの一部のポートが閉じられたため，分散ロック間でシステムコールを用いてセッションクローズを要求したが，スイッチ上でパケットがドロップし，システムコールからエラーが返却されなかったため，リトライが実施されなかった．

(7) 要因分類③-1 キャッシュ（回復性）

キャッシュの削除漏れの要因では，分散ロックのフェールオーバ後に実施される復元処理で，キャッシュのフラグが無効になっているケースがあり，フェールオーバ後のキャッシュ関連処理で問題が発生した．

大規模分散処理システムは，大規模環境で安定動作すること，一部の機器における故障発生時も動作し続けることが重要である．抽出した問題の大部分は，これらの点に反する問題であり，「資源効率性，障害許容性，回復性」の項目を重視したからこそ発見できた問題である．仮に上述の項目を重視せず，これらの問題を発見できなかった場合，システムの再立ち上げ，スループットが不安定，データ消失による再投入，上位アプリにおける処理の再実施等の事象が発生すると考えられ，非常に使い勝手の悪いものになる．

発生した問題の要因から考察して導き出した，「資源効率性，障害許容性，回復性」の各検証観点において有効な検証（確認観点含む）を表 24 に示す．「資源効率性」の観点では，「処理集中」において，大規模なデータの処理という大規模分散処理システムに特有な処理の発生前後に，性能低下が発生している．したがって，検証を実施する際は，上記処理の発生状況や発生前後の性

能について注意する必要があることが分かった．また，「排他制御」に関しては，バックグラウンド処理や運用コマンドにおいて，大規模分散処理システムの特徴である大規模データを取り扱う際に，性能への影響を確認する必要があることが分かった．他にも，大規模分散処理システムは，ネットワーク(NW)を介して各サーバが協調して動作するため，NW 系の高負荷時に通常とは異なる動作（システムコールの失敗や，タイムアウト等）が発生することで，動作に支障がないかも確認する必要がある．

一方，「障害許容性」や「回復性」の観点では，大規模分散処理システムがある程度の多重障害時でも動作するように設計されていることから，多重障害の検証を重視して実施することで，様々な障害パターン，タイミングでの試験が実施され，問題が顕在化するケースが見られた．このことから，障害系の加速検証も有効であると思われる．また，大規模障害によって処理遅延が発生し，1 台の故障では発見できないような問題も抽出できた．他には，長期安定性試験で，特殊ケースの障害に関する問題の抽出ができると考える．

表 24. 大規模分散処理システムで有効な検証や観点

検証観点	検証項目分類	有効な試験（確認観点含む）
資源効率性	・性能，高負荷試験 ・長期安定性試験	・大規模サーバ環境でデータが分散する過程と分散完了後の性能差が設計通りか． ・バックグラウンド処理（データの圧縮処理等）が設計通りに動き，性能への影響が想定通りか． ・リード/ライト中かつ大規模データで運用コマンドが要件時間内に完了するか． ・バックグラウンド処理数の時間推移が想定通りか． ・NW の高負荷が断続的に発生した場合，上位アプリへの影響は設計通りか．
障害許容性 回復性	・障害復旧 ・長期安定性試験	・多重障害（障害中/復旧後の障害） ・大規模障害時（ラックダウン等）の障害復旧とサービス持続性 ・障害系加速試験（クライアントからの連続接続・切断時の安定動作等） ・特殊障害発生時（特定プロトコル，一部ポートの遮断等）に設計通りの動作か．

- 手法の適用方法について以下に示す. 手法 4 は検証項目の重要度付けと重要度順の期間内の検証により, 検証精度と検証期間とのトレードオフをコントロールする.

  一般的に検証期間が長ければ検証精度を高くすることができるが, 検証期間が短いと検証精度を低くしなければならない. 検証のリソースが限られている商用開発では, 必要な検証精度とそのための検証期間を決定しなければならず, 検証精度と検証期間とのトレードオフをコントロールすることが必要である. 重要度の観点での項目精査により, 項目の削減量が多いので期間に収まる場合がほとんどであるが, 項目精査で削減しても項目が多い場合は, 期間に収まらず, 検証精度とのトレードオフのコントロールができない場合がある. トレードオフのコントロールができない場合は, 検証精度や検証期間の決定のやり直しが必要である.

  具体的に検証精度と検証期間のトレードオフのコントロールができる場合は, 手順は表 25 となる. まず, 1 の検証精度と見積もりで検証期間を決定する. 次に, 2 の検証項目の重要度付けを行い, 検証項目の削減を実施する. そして, 3 の計画した期間内に重要度の順に項目を検証する.

  しかし, 3 で検証期間内に検証が完了できない場合は, 検証精度を低くするか検証期間を延ばすかの調整をする. または, 要員を追加するなどのリソースの条件を変更する. これらの調整や変更の後に再度, 1 の検証精度と検証期間の決定からやり直す. 以上のように, 1 から 3 の手順を繰り返すことで検証精度と検証期間のトレードオフをコントロールする.

  3 の検証期間内に検証ができない場合は, システムのユースケースが多くかつシステムで確保する品質が高い場合に当てはまる. したがって, ユースケースと品質の程度が検証精度と検証期間とのトレードオフのコントロールに大きく影響する. 重要度付けは, システムのユースケースやシステムで確保する品質の程度で変わる. つまり, ユースケースが多くかつ確保する品質が高い場合は, 重要度付けが多くなる. その場合は 1 の検証期間を増やす決定が必要となる.

表 25. 検証精度と検証期間とのトレードオフのコントロールの手順

	作業項目	備考
1	検証精度と検証期間の決定	3 で検証期間内に完了できず, 検証精度と検証期間のトレードオフのコントロールができない場合は, 検証精度や検証期間の調整またはリソース条件の変更後 1 からやり直す
2	検証項目の重要度付けと項目削減	
3	計画した期間で重要度の順に検証	

また，見積もりの期間が期間内であることが判明した場合，手法 4 により期限まで重要度に従い残りの検証項目を検証する．さらに信頼度成長曲線を使い，検証が充分に実施できているかどうかを確認することで，検証精度の向上が可能となる．検証では，品質指標値と呼ばれる最低限実施すべきプロダクトの規模や機能数あたりの検証項目数や障害摘出数が規定されている[92]．品質指標値が達成できる時期を見積もることで，手法 4 と信頼度成長曲線により，可能な期間内にさらに検証精度の向上を検討することが可能となる．

以上のように，重要度付けがポイントであり，1から 3 の手順の実施により，重要度の順に検証をして期間内に完了できるように検証精度と検証期間とのトレードオフをコントロールする．この手順は，大規模分散処理システムでは必要となる手順であり，また，項目が多くなる他のシステムでも適用が可能である．

- 適用可能な領域について以下に示す．この解決の考え方は，重要な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」を中心にすることで項目を絞り，検証期間内に重要度順に実施するアプローチであり，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，他の大規模分散処理システムでも適用が可能である．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システムのように，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．

5. テスト環境の開発アクティビティの課題と解決手法

制御が容易で結果を記録，再現可能なテスト環境の開発という，テスト環境の開発アクティビティにおいて主要な課題は「性能確認と機能確認の効率化」（課題 5）と「故障検知と故障対応の迅速化」（課題 6）の 2 つの課題である．課題 5 に対しては「データ生成とログの出力を組み込んだテストドライバ（TP）の作成手法」（手法 5），課題 6 に対しては「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）でそれぞれ効果がある．

本章では，テスト環境の開発アクティビティにおいて，検証作業の合理化により検証期間を削減する解決手法を 2 つの課題それぞれに対して説明する．5.1 節で先行研究，5.2 節で課題を示した後，5.3 節で課題 5 に対して効果のあった手法 5 について，5.4 節で課題 6 に対して効果のあった手法 6 について説明する．

5.1 テスト環境の開発アクティビティの課題に関する先行研究

本節では，1.2 節に示したテスト環境の開発アクティビティの課題「性能確認や機能確認の効率化」，「故障検知と故障対応の迅速化」で利用される技術それぞれについて関連する研究を示す．2.3 節で示したとおり，従来技術のソフトウェアエンジニアリングツール，保守ツールは，大規模分散処理システムで発生した課題以外において適用できる．しかし，発生した課題においては，以降のようにそのままでは適用ができない．

5.1.1 性能確認や機能確認の効率化に関する先行研究

システムの性能確認と機能確認を効率的に実施するツールとして，Tom White[8]や B.Cooper ら[19]は，ベンチマークツールによる異なる大規模分散処理システム間での測定を記述している．このツールは，ある設定されたデータサイズで短時間の測定をする場合に効果がある．しかし，このツールは，データサイズの変更などデータの条件を変化させた性能の測定ができないことや，長時間の検証に必要な途中で測定を中断し再開することができない．また，データを可変にする性能の測定は，データ生成ツールを組み合わせるなど，新たなツールとして作成が必要である．

一方，Apache プロジェクト[20]は，従来の情報システムにおけるデータウェアハウス向けベンチマークである TPC-H を，大規模分散処理システム向けの Hive 上で，動作可能としたツールを提供している．このツールは，Apache プロジェクトのプロダクトを利用する場合に有効であり，TPC-H のベンチマークの結果により，他 DBMS との性能の比較ができる．しかし，Apache プロジェクトのプロダクトに特化しているため，他の大規模分散処理システムに対応するには，汎用化するための改造が必要である．例えば，上記の機能確認や問題解析の検証や，長時間の検証に対応するための改造が必要である．

5.1.2 故障検知と故障対応の迅速化に関する先行研究

大規模分散処理システムの故障検知機能で検知できない場合に有効となる，マシンの監視ツールを調査した結果を以下に示す．

現在の大規模なマシンの監視は，寺島[86]が紹介するように，Nagios，Hobbit，ZABBIX，Hinemos 等が既存の監視 OSS で運用されている．これらのツールを使うことにより，大量に運用されているマシンのハードウェア故障を即時に検出することができる．ただし，大規模分散処理システムで一部のプロセスが動作していないため性能が出ないといった，特定プロトコルの通信障害やマシンの性能劣化などは，既存の監視 OSS の機能で監視ができない[10]．また，障害が発生した際の対応については，紹介記事には情報がない．

故障対応においては，Barlow，R ら[61]は，故障箇所の交換方法として，故障交換，個別交換，一斉交換と分類し，その分類に故障分布をあてはめて，どの方式が効果的かを示している．この方式は，交換する費用をできるだけ安くする方法として有効である．しかし，商用運用では交換作業自体をできるだけ短くすることが重要であるが，その方法については記述がない．

5.2 テスト環境の開発アクティビティの問題と具体的な課題

大規模分散処理システムのテスト環境は，マシン台数が多く検証対象のプロダクトが複数あり，扱うデータ量が大量であるため，大量のマシンに対して一斉に検証ができるようにコントロールしかつ効率よく検証の結果を確認しなければならない問題がある．実際に CBoC タイプ 2 のシステム検証では，数百台規模のマシンに対して書き込みや読み出しの処理を擬似する必要があり，そのツールの準備が必要である．また，利用するマシンの台数が多く故障は日々発生するため，マシンの故障による性能の低下を防ぐなど，検証への影響回避が必要である．

上記のとおり，テスト環境の開発アクティビティにおけるシステム検証では，検証期間の短縮や検証作業の効率化が必要となる問題が発生している．主要な問題を表 26 に示す．

表 26.「テスト環境の開発」アクティビティで発生する問題の概要と具体的な課題

作業分類	発生した問題の概要	具体的な課題（具体的内容）
検証実施	検証で利用する大規模のマシンに対して読み書きの性能確認や機能確認に手間を要する	性能確認と機能確認の効率化 （読み書きデータのスループットやデータ量の調整を細かく制御することが可能な TP を作成し，性能の基礎データを取得する）
環境確認	通信障害などマシン故障により，機能確認や性能測定を再実施しなければならない	故障検知と故障対応の迅速化 （マシン故障を速やかに検知し，検証に影響が出ないようにマシンの切り離しなどの対策をとる）

[10] この障害は，大規模分散処理システムの死活監視機能では監視ができないため，切り離しができない．

問題の 1 点目は，大規模分散処理システムの検証では，数多くのマシンがあるため，同時に性能確認や機能確認をする際に，検証の同期合わせの手間と，集計するデータを決めて分散しているマシンから収集する手間とを要することである．この問題の具体的な課題は，性能確認と機能確認の効率的な実施である．

問題の 2 点目は，日々マシンの故障が発生することにより，検証結果にその影響が及ぶため，機能の動作や性能測定の測定値が要件と異なり，原因の調査や再試験で工数を要してしまう問題である．この問題の具体的な課題は，マシン故障に伴う検証への影響を回避する，速やかな故障検知と故障対応である． 以降から，作業をコントロールできる解決手法を課題それぞれについて詳細に説明する．

5.3 「性能確認と機能確認の効率化」(課題 5)と解決手法

大規模分散処理システムでは，2.2 節で示したとおり複数の実アプリケーション(AP)からのデータの書き込みと読み出しの処理がある．そのため，検証のユースケースとして AP(Application Program)によるデータの読み書きには，シーケンシャルリード，ランダムライト，ランダムリードと種類があり，そのデータについてもテキストデータもあれば画像などのマルチメディアデータもある．これらの処理を実現し，さらに機能確認や問題解析の検証で必要な書き込む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るベンチマークツールであるテストドライバ(以降，TP(Test Program)と呼ぶ)が必要となる．

CBoC タイプ 2 では，数百台規模のマシンに対して，大量データに対する読み書きを一斉にコントロールしたり，機能確認やデータの整合性確認を自動で実行したりする機能が求められる．さもなければ検証実施時間は長くなり，与えられた期間で実施する他の作業を圧迫してしまう．

検証作業の制御により数多くのマシンに対して一斉にデータの書き込み／読み出しを可能とし，システム検証で利用できるツールを表 27 に示す．

表 27. データの書き込み／読み出しのツール

	利用ツール	ツールの説明	利点	欠点	主な用途
1	実アプリケーション	商用で運用される実アプリケーション（AP）そのものを用いる	実利用における正確な性能確認が可能	・利用に際してNW 環境に利用条件がある ・書き込みデータが対象となるサーバ毎に異なりデータ量のコントロールがしづらい	・ユースケースの性能要件達成確認
2	汎用ベンチマークツール	OSS 等のベンチマークツールを目的に合わせて取捨選択 TestDFSIO [11]， TeraSort [12]，MRBench[13], YCSB[14], TPC-H－Hive[15]など	・開発不要 ・基本性能の把握が可能	実利用におけるデータ量は考慮していないため参考値	・類似システムとの性能比較
3	独自ベンチマークツール	実利用を模擬したデータ発生用のテストドライバ（TP）を独自に開発	実利用を想定した測定が可能	開発リソースが必要	・ユースケースの性能要件達成確認 ・限界性能・性能特性の把握

1 点目の実アプリケーション（AP）は，商用で運用されるアプリケーションそのものである．商用のユースケースにおける処理を忠実に再現でき，機能を正確に確認することができる．しかし，インタ

[11] Hadoop に付属の分散ファイルシステムベンチマークツール[8],
[12] Hadoop に付属の巨大ファイルソートプログラム[8],
http://hadoop.apache.org/docs/current/api/org/apache/hadoop/examples/terasort/package-summary.html
[13] Hadoop に付属の MapReduce ベンチマークツール[8]
[14] Yahoo!が公開している NoSQL 用ベンチマークツール[19]
[15] データウェアハウス向けベンチマークである TPC-H を Hive 上で動作可能としたツール[20],
https://github.com/rxin/TPC-H-Hive/tree/master/

ーネット上のデータを取得するには，グローバルな IP アドレスを利用したインターネット環境が必要となるなど利用に際しての制約がある場合がある．また，例えば書き込みデータ量が対象となるサーバ毎に異なるなど，性能検証時にデータ量がコントロールできない場合もある．

2 点目の汎用ベンチマークツールは，OSS のベンチマーク[19]が主流であり，特定の条件の書き込みや読み出しの基本性能の把握ができる．また，OSS 公開されているものは，利用者が多く，その利用者が測定した類似システムの結果が公開されているため，そうした類似システムの結果と比較して自システムの優劣の確認ができる．しかし，その結果は，特定条件の書き込みや読み出しの基本性能値しか取得できないため，実際のデータ特性を模擬した性能や，過負荷時や運用によるマシンメンテナンス作業など，商用での実利用を想定した性能値を取得することができない．さらに，問題発生時の解析に必要な機能は具備していない．

3 点目の独自ベンチマークツールは，商用で運用される AP の処理を模擬した TP であり，実利用を想定した性能要件や限界性能の特性を把握することが可能である．従来の情報システムにおけるシステム検証では，主に汎用ベンチマークツールと実アプリケーションを利用することが多い．しかし，大規模分散処理システムの検証においては，汎用ベンチマークツールでは商用の実利用を想定した性能が取得できないこと，AP では環境の制約やデータ量のコントロールができないことから，これらを解決する独自ベンチマークツール（TP）の作成が必要となる．しかし，新たに TP を作成するには，開発リソースを計画に入れる必要があり，TP 開発の負荷を見積もっておく必要がある．そこで作業見積もりをしたところ，TP の作成は，実アプリケーションを工夫して利用し，かつ取得できない性能値を手作業で確認するのに必要な期間（我々の経験では 0.5ヶ月）よりも短い期間（7 日間）で作成可能であり，期間短縮の効果が期待できることが判明した．

5.3.1 解決手法「データ生成とログの出力を組み込んだテストドライバ（TP）の作成手法」（手法 5）

課題 5 の解決手法は，5.3 節の 3 点目の独自ベンチマークの方法である「データ生成とログの出力を組み込んだテストドライバ（TP）の作成手法」（手法 5）が効果的であると考えた．本項では，手法が効果的となる商用のシステム検証で必要なテストドライバ（TP）の機能を明らかにする．そのため，CBoC タイプ 2 の性能測定兼システム検証用に独自開発した独自ベンチマークツール（TP）の機能をどのように決定したか説明する．

TP については，AP の処理を模擬し性能を測定できるようにするため，以下の機能を具備する必要がある．

A) 複数台の TP を集中制御できる仕組み
B) 書き込みデータの内容をパラメータで任意に変更できデータの内容の確認ができる仕組み
C) スループットの値やデータの整合性の確認ができる仕組み

A の機能は，TP 数が数百に及ぶ場合には一つ一つ手作業で起動や管理することは時間がかかるため，起動や停止を集中管理するコントロールサーバ上の TP の一括起動や停止を行うツールを動作させることにより，各 TP を一括管理する．また設定変更もコントロールサーバで実施する．この機能により，AP の処理と同じように，書き込みや読み出しの TP の起動や終了タイミングを合わせ

ることができ，AP の模擬ができる．

B の機能は，データの内容を固定ではなく変更できるようにすることにより，データの内容によるシステムの性能への影響を確認できる．また AP のデータ内容を模擬することで，実際のシステムを模擬した機能確認ができる．

C の機能は，各 TP のログ等からの結果の集計を仕組みとして持つことにより，数百台規模のマシンから時刻を合わせて集計した結果を容易に確認することができる．これにより，逐次性能を確認することができ，ログのメッセージや監視ツールの突合による問題発生時の即時対応ができる．

これらを実現すると， 図 24 のように 1 箇所のコントロールサーバに配置したデータ生成 TP から各ワーカに配置した TP を制御することができる．以降で A，B，C の各機能の詳細について記述する．

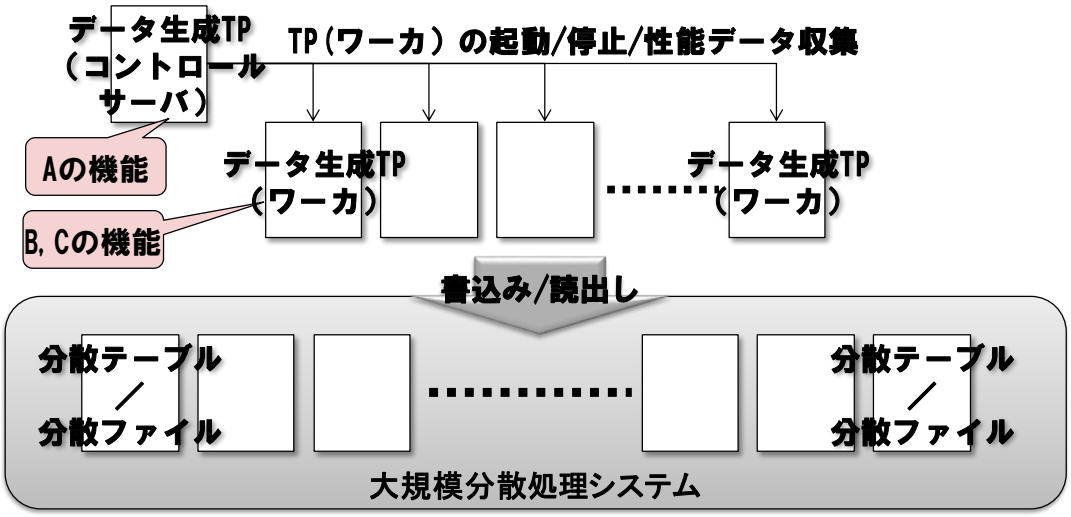


図 24. TP の構成と集中制御

A の機能については，機能として必要な「ランダムライト／ランダムリード」と「シーケンシャルリード」を独自ツールの TP として作成し，集中制御した．「ランダムライト／ランダムリード」と「シーケンシャルリード」とを別々に作成するのは，AP がそれぞれを別サービスとして処理するためで，その結果，それぞれが別プロセスで動作するからである．TP の構成は， 図 24 のように 1 台のコントロールサーバと複数台のサーバ上にある「ランダムライト／ランダムリード」と「シーケンシャルリード」の処理をするワーカで構成される．オペレーションは全てコントロールサーバで行い，コントロールサーバから各ワーカに対して起動／停止／性能データ収集の制御を行う．

B の機能について，書き込みデータは，表 28 に記載する「ランダムライト／ランダムリード」の TP を使ったデータ生成により自動生成した．生成したデータにおいて，検索されるキーとなるデータの RowKey は，複数のサーバ／プロセス／スレッドによる書き込み／読み出しの処理が分散されるように，該当のサーバやプロセスの情報を用いたハッシュによる変換処理を行い算出した値で作成する．また，そのデータ自体は，Rowkey で使用した該当のサーバやプロセスの情報からハッシュ算出した値で作成するとともに，データの書き込み時の圧縮機能により，指定した圧縮率で作成する．

この手法により，生成されたデータの内容を把握した検証ができる．例えばデータを書き込む際の異常系の検証では，書き込むデータと書き込まれたデータの整合性がチェックできる．「ランダムライト／ランダムリード」TPと「シーケンシャルリード」TPにおけるデータの書き込み／読み出しの管理の仕組みで工夫した点を以降に示す．これらの工夫は汎用ベンチマークツールでは実現されていない．

表 28.「ランダムライト／ランダムリード」の TP におけるデータ生成方式

	生成項目	内容
1	RowKey	処理を行うサーバの IP アドレス，シーケンス番号，プロセス ID，スレッド ID からハッシュ算出
2	データ （テキスト）	処理を行うサーバの IP アドレス，シーケンス番号，プロセス ID，スレッド ID，世代番号（パラメータ指定）からハッシュ算出 圧縮機能により，データ登録時に指定する圧縮率になるようにデータを作成．圧縮率はパラメータとして指定可能
3	データ （マルチメディア）	処理を行うサーバの IP アドレス，シーケンス番号，プロセス ID，スレッド ID，世代番号（パラメータ指定）からハッシュ算出

注：世代はデータの版数を意味する．

(1) 「ランダムライト／ランダムリード」TP のデータ生成に必要な項目を表 29 に示す．データ書き込みの量を容易に増やしたり，データの生成バリエーションを増やしたりするため，以下の機能が必要である．

① スレッド数／プロセス数やデータの送信間隔で読み込み／書き込みデータの発生量を調整

② 書き込むデータサイズは一様ではなく正規分布や F 分布，ポアソン分布のようにデータサイズを分布に基づいて書き込み

③ 書き込みを中断した場合に中断点からの再開（レジューム機能）

必要な項目は，書き込み／読み出しのデータとそのデータを生成と指定するための情報である，RowKey サイズ，データサイズ分布，データサイズ，レジューム，およびスループット計測周期とした．これらの項目について，YCSB，TestDFSIO，TeraSort ではレジュームやスループットの計測周期を指定することができない．例えば，詳細な分析が必要な場合でも計測周期は短くすることができず，途中で中断した場合は最初からの処理になりその時間が無駄になる．表中の×は，その項目を指定できないことを示している．表中の△はスレッド数を指定できるが，その他のインターバル，総件数を指定できないことを示している．YCSB，TestDFSIO，TeraSort は，ファイルシステムの I/O の性能や，大量データのソート速度を測定するものとして作成されており，ユースケースの機能確認や性能確認には合致しないため指定できない項目がある．

(2) 「シーケンシャルリード」TP のデータ生成に必要な項目を表 30 に示す. 必要な可能な項目は, 読み出しのカラムとそのカラムを指定するための情報であり, 検索対象となるデータの timestamp, 同時読み出しプロセス数とした. これらの項目について YCSB, TestDFSIO, TeraSort では読み出しカラムの指定, つまり, 1 カラムだけの指定をして読み出しをすることができない.

表 29.「ランダムライト／ランダムリード」TP で必要な項目

	指定項目	YCSB	Test DFSIO	Tera Sort
1	データ発生量(スレッド数, インターバル, 総件数)	△	△	×
2	RowKey サイズ	○	×	×
3	データサイズ分布	○	×	×
4	データサイズ	×	○	○
5	レジューム	×	×	×
6	スループット計測周期	×	×	×

表 30.「シーケンシャルリード」TP で必要な項目

	指定項目	YCSB	Test DFSIO	Tera Sort
1	読み出しカラム	×	×	×
2	検索対象となるデータの timestamp	×	×	×
3	同時読み出しプロセス数	○	○	×

表 31.「ランダムライト／ランダムリード」TP の主な出力ログ

	ログ出力項目	YCSB	Test DFSIO	Tera Sort
1	ログ出力時刻	×	×	○
2	アクセス数	○	○	×
3	平均処理時間	○	○	○
4	処理時間の分散	×	○	×
5	処理時間の最大値	○	×	×
6	処理時間の最小値	○	×	×
7	スループット	○	○	×
8	処理時間が閾値以上の件数	△99%のみ	×	×
9	読み出しデータ無しの件数	×	×	×
10	読み出しデータの不一致の件数	×	×	×

C の機能は，スループットやデータ内容の確認ができるようにするもので，ログに必要な情報を記録する仕組みと，その記録を集計する仕組みからなる．複数のワーカに配置した TP で出力されたログをコントロールサーバで集計できるように作成することで作業の効率化を図った．「ランダムライト／ランダムリード」TP と「シーケンシャルリード」TP におけるデータの書き込み／読み出しの計測の仕組みで工夫した点を以降に示す．

(1) 「ランダムライト／ランダムリード」TP の性能を計測する仕組みとして必要となる主なログを表 31 に示す．特に性能の特性を確認するために工夫した項目は，以下の 2 点である．

① 各処理の平均処理時間に加え，最大／最小／分散等を出力

② レスポンスタイムがある閾値以上のデータの個数を出力

具体的な項目については，ログ時刻，アクセス数，平均処理時間，処理時間の分散，処理時間の最大値，処理時間の最小値，処理時間のスループット，処理時間の閾値以上の件数，読み出しデータ無しの件数，および読み出しデータの不一致の件数とする．

(2) 「シーケンシャルリード」TP の性能を計測する仕組みは，読み出しによる結果の確認のためのログ出力が必要である．主なログを表 32 に示す．「シーケンシャルリード」の TP については，読み出し開始/終了時刻，読み出しデータ量，読み出し件数，読み出しスループット，および読み出し Area（水平分割されたテーブルの一部）数を主な出力ログとする．

表 32.「シーケンシャルリード」TP の主な出力ログ

	ログ出力項目	YCSB	Test DFSIO	Tera Sort
1	読み出し開始/終了時刻	×	△開始時刻のみ	○
2	読み出しデータ量	×	○	○
3	読み出し件数	○	○	○
4	読み出しスループット	○	○	×
5	読み出し Area（水平分割されたテーブルの一部）数	×	×	×

5.3.2　手法 5 の適用結果

適用結果については，作成した TP を利用した確認の作業に要する期間で評価した．この手法を適用した結果について示す．作成した TP では，AP の利用による環境制約の考慮やデータ量コントロールの工夫が必要なく， 図 24 に示したように検証作業の制御により 1 箇所のコントロールサーバに配置したデータ生成 TP から各ワーカに配置した TP を制御して，機能検証と非機能検証ができた．特に，汎用ベンチマークツールでできなかった，書き込んだデータのチェックや，データの書き込みの中断や，データによる性能の確認を 100 テラバイト級のデータに対してできるようになった．

従来の手法では，汎用ベンチマークで自動的な確認に 1 日かかっており，汎用ベンチマークで確認できずデバッガーなどで値をトラッキングして取得するなど工夫して確認するのに約 0.5 ヶ月要していた．対して，提案する手法では，全て自動的に1日で確認ができ，ツールの作成を含めて計画どおり 7 日で完了し，検証実施時間の短縮を図ることができた． 以降に手法の考察を示す．

解決手法「データ生成とログの出力を組み込んだテストドライバ（TP）の作成手法」（手法 5）について，以下の

- 課題 5 の「性能確認と機能確認の効率化」を解決する方法
- 適用可能な領域

の観点で考察する．

- 課題 5 の「性能確認と機能確認の効率化」を解決する独自ベンチマークツール（TP）を作成する際に，以下の 2 つの案を検討した．

(1) ツール自身で書き込みのデータを作成し，書き込まれたデータを読み出すツールを作成する

(2) AP で蓄積された実際のデータを別ディスクに保管し，そのデータを蓄積データとして登録しなおし，その蓄積されたデータを読み出すツールを作成する

この 2 つの案を比較したところ，(1)，(2)両方ともツールを作成する工数を要する点は変わらない．しかし，(2)は蓄積されたデータが 100 テラバイト～ペタバイト級であるため，別ディスクに保管するだけで数週間を要した．その上，蓄積データとして登録しなおすのに，さらに数週間を要した．(1)は(2)と比較してデータを作成する機能を新たに作成する工数を要するが，1 度作成すればよく再利用ができる．加えて(1)のデータを蓄積する機能は，3.3 節で示したとおり時間を短縮することができる．以上より，(1)の案がよいと判断した．

- 適用可能な領域について以下に示す. この課題解決の考え方は, TP の集中制御というアプローチであり, 多くの AP による同時のデータの読み書きがある場合, 他の大規模分散処理システムでも適用が可能である. 手法 5 は, 多数の 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テストドライバを作成せず, 手動で AP を動作することが可能であるため効果を得ることができない.

5.4 「故障検知と故障対応の迅速化」(課題 6)と解決手法

従来の情報システムにおけるシステム検証に使われるマシンは数台であり，故障発生の頻度は 1 年に数度である．そのため，故障対応をリアクティブに実施しても検証スケジュールに影響することがなく，問題は顕著化しなかった．

それに比べ，大規模分散処理システムでは，利用するマシンの台数が数百台規模となり，マシンあたり 1 年に 1 度の故障率でも日々マシンが故障する状況となる．この大規模分散処理システムでは故障したマシンを切り離して運用する機能が備わっているが，マシンを切り離してしまうと決めたマシン台数の性能測定ができなくなってしまう．また，マシンの性能劣化や特定のプロトコルで接続できないなど，故障したマシンを切り離す機能に検知されずに故障したマシンが運用される場合がある．これは，処理の継続を重視した設計となっているため，処理性能の劣化の対応は即時対応としていないためである．故障したマシンの放置は，故障したマシン台数分の処理性能が落ちるだけでなく，故障したマシンの処理がボトルネックとなって機能に影響を与えることもある．そのため，マシンの故障を検証前に検知して交換等の対応をし，検証への影響を回避する必要がある．仮想化技術を利用しない CBoC タイプ 2 は，マシン台数が多いため，効率的に実施しないと検証開始が遅れることになってしまう．

実際，データの書き込みにおいて，図 25 に示すとおり 63 時間経過した付近と 73 時間経過した付近で，応答時間が長くかかっている時間帯があった．性能測定の条件としては，数百台のマシンを利用し図 24 の構成で TP によりランダムライトを実施していた．データサイズはクローリングされたデータと条件を合わせ平均レコードサイズ 5k Byte から 40k Byte とし，分布も合わせていた．

調査は応答時間とマシンのリソースの観点で行った．その結果，次の 2 点が明らかとなった．

(1)特定のマシンで応答時間が遅くなっていないか調査をした結果，特定の 1 台ではなく，複数台で遅い傾向があった．

(2)各マシンのリソース状況を調査した結果，特定のサーバで I/O が一時的に高くなっていた．

これらの調査結果から，特定のサーバで I/O の処理が一時的に遅くなり，それが全体的な応答時間の悪化につながったと考えられるが，この調査に数週間を要することになった．上記のとおり，検証に影響のある故障を速やかに検知して回避することが重要である．

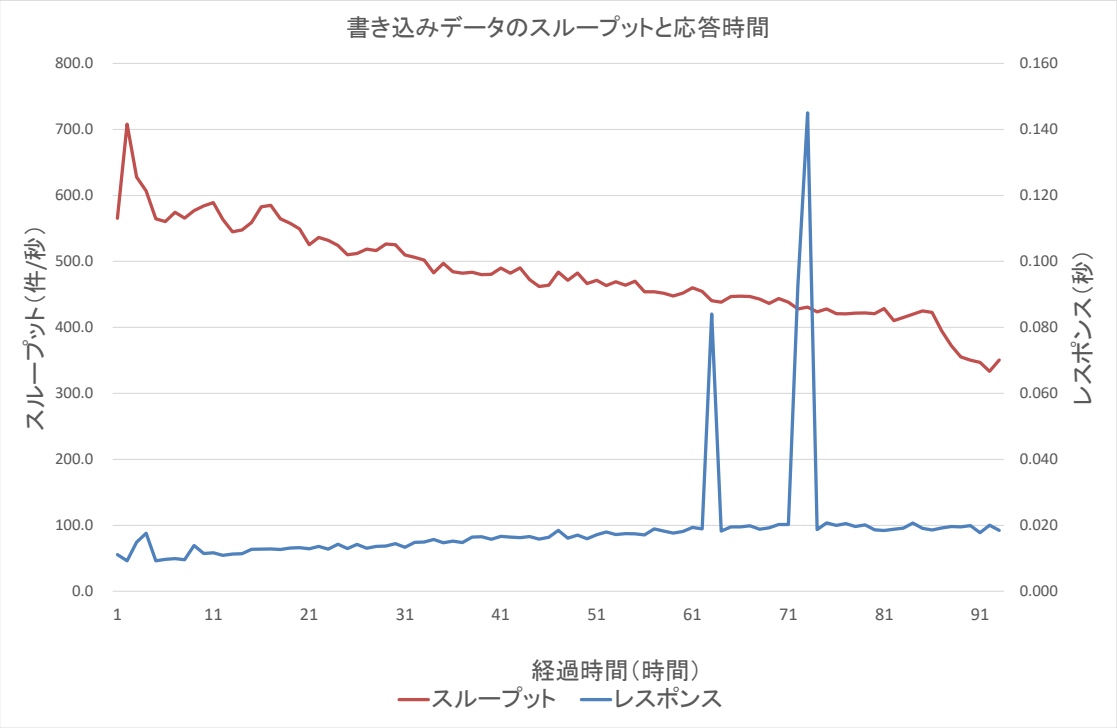


図 25. 書き込みデータのスループットと応答時間

5.4.1　　解決手法「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)

課題 6 の解決手法としては「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)が効果的であると考えた. 理由は, 監視ツールによる故障監視と, 予備マシンの用意による故障マシンの入れ替えで対応することで速やかに故障対応ができると考えたからである. 以下に具体的に示す.

解決方法は, ツールと監視 OSS である MRTG と Nagios や Crane に加えて, シェルスクリプトによるツールと Cron により故障を検知する手法を採る. 表 33 に示すとおり, 監視 OSS による監視は, 従来のシステムでの監視内容である CPU の負荷といった CPU の状態, メモリの使用率といったメモリの状態, ディスク I/O やディスク容量といったディスク状態や, NW の不具合等の状態を数値の異常により監視する. 加えて, 利用した監視 OSS では, syslog 等のシステムログに出力されたエラーメッセージだけでなく, 追加の設定により大規模分散処理システムのエラーメッセージも監視する.

表 33. 監視概要

種別	監視内容	監視方法	ツール	OSS		
				MRTG	Nagios	Crane
従来の情報システム	CPU 状態	監視 OSS	×	○	○	×
	メモリ状態		×	○	○	×
	Disk 状態		×	○	○	×
	NW 状態		×	○	×	×
大規模分散処理システム	ログメッセージ	ツール	○	×	○	○
	プロセス状態		○	×	×	○
	特定プロトコル疎通確認		○	×	×	×
	統計情報		○	×	×	×

スクリプトや Cron のツールでは，大規模分散処理システムで特有の内容である，ログメッセージやプロセス監視，特定プロトコルの疎通確認の他に，運用に必要となる統計情報の確認を行う．スクリプトや Cron のツールを定期的に実行することにより，ログメッセージやプロセスの監視や特定プロトコルの疎通確認を自動化し，問題を検知した場合，即時に監視画面等で通知する．

商用システムの CBoC タイプ 2 の検証では，20 台のマシンを積んだラックを 15 ラック利用していた． 図 26 に 2 年間の故障の実例を示す．故障の発生には，当初は「ある検証を実施すると発生する」，「あるプロダクトのマシンで発生する」等の仮説を立てたが，特定の作業による故障の傾向は見られなかった．一方で，故障したマシンは再度故障するという傾向が見られた．また，特定プロトコルで接続できない状態のマシンは，1 ヶ月あたり数%の割合で発生していた．この状況から，故障率とラックに搭載できるマシン数を考慮し，予備マシン数は全体台数の 1 割とする．

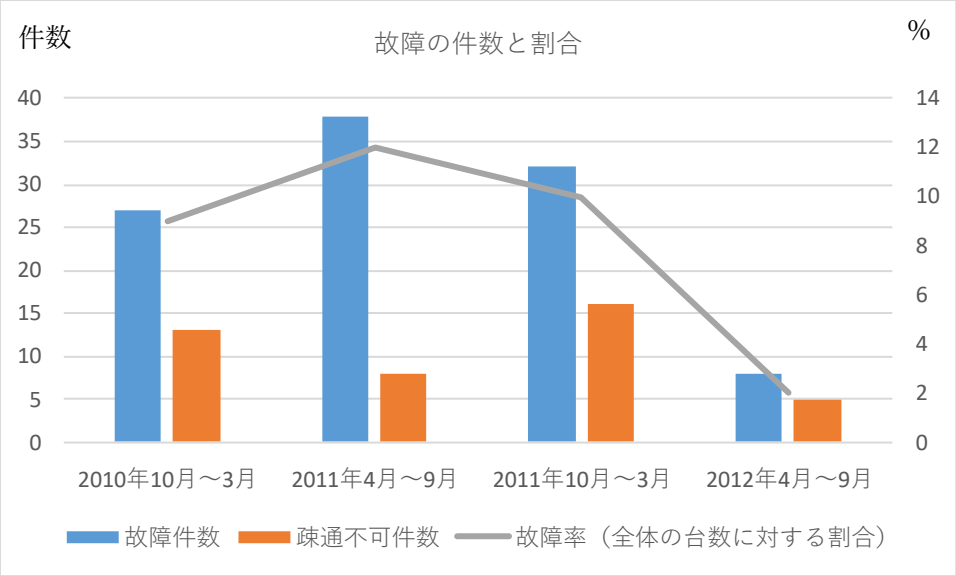


図 26. 適用した期間の故障の実例

5.4.2　手法 6 の適用結果

適用結果については，故障を検知し，故障したマシンを入れ替える作業に要する時間で評価した．商用システムである CBoC タイプ 2 の検証用のマシン環境に，提案方法を適用した結果を以下に示す．予備マシンが無いときは，利用できるマシンの確認や設置の処理など調達に 5 日間（1 週間），環境設定と組み込みに数日と 14 日間（2 週間）要していた．対して，故障率などが上記の商用システムの検証環境において，提案の解決手法を適用した結果，予備マシンが準備済みであることにより，入れ替え作業のみの 30 分で復旧ができた．また，OSS による通常の監視システムでは故障の対象にならず，発見が遅れることが多い特定プロトコルでの接続不良など，大規模分散処理システムに特有な故障にも対応ができ，再実施のない検証の実現ができた．以降に適用した手法の考察を示す．

解決方法「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）について，以下の

- 課題 6 の「故障検知と故障対応の迅速化」を解決する方法
- 適用可能な領域

の観点で考察する．

- 課題 6 の「故障検知と故障対応の迅速化」を解決する故障マシンの入れ替えの方法として，表 34 に示すように 3 つの案を検討した．

表 34．故障マシンの入れ替え方法

案	解決方法	費用	復旧期間
1	故障の都度にマシンを入れ替える	○	3 週間
2	運用の倍のマシンを用意して使い捨てにする	×	30 分
3	一定の割合の予備機を用意して入れ替える	△	30 分

1 点目の案は，マシンの故障を発見した場合に，マシンの調達を行って調達できたマシンから故障機と入れ替えることである．この方法は，故障の都度にマシンを調達するので，費用は故障したマシンの分のみでよいが，調達や設置に時間を要する．当初この案で運用をしていたが，故障から復旧に 2～3 週間程度を要しその間の検証が止まってしまうため他の案に替えた．商用の運用時であれば，システムが稼動していれば対応ができるが，マシン台数を揃えた機能検証や性能検証は対応ができない．

2 点目の案は，マシンの故障の頻度は考慮せず，利用するマシンのバックアップとしてもう一式マシンを用意することである．この方法は，マシンの故障を発見時には，バックアップマシンとすぐに切り替えができ，故障から復旧までの時間は 30 分程度ですむ．しかし，利用する倍の台数のバックアップマシンが必要となるため，利用するマシンの倍の費用が必要になる．

3 点目の案は，これまでの故障の実績を基に，これから故障するマシンの台数を割り出し，そのマシンの台数を予備機として用意し，マシンが故障した場合に，その予備機と切り替えることである．この方法は，費用としては故障する頻度が少ないときには，余分に費やしてしまい，故障する頻度が多いときには，1 点目の案と同様にマシンの調達を行う必要がある．しかし，故障する頻度が予備機の数と合っている場合は，最少の費用と故障から復旧までの最短時間で故障したマシンの入れ替えができる．

以上から 1 点目から 3 点目の中では，3 点目の案がよいと判断した．解決の考え方としては，監視ツールによる故障監視と，予備マシンの用意による故障マシンの入れ替えで対応する．故障率はこれまでの運用の記録からある程度範囲が分かるため，それを元に想定した故障マシン数と同程度の予備マシンを準備しておく．これにより，検証に影響がないよう速やかに故障マシンを切り離し予備マシンに入れ替えることができる．また，通常のマシン監視に加え，それだけでは発見できないシステム独自で使用している特定プロトコルの故障を速やかに検知する．

- 適用可能な領域について以下に示す．この解決の考え方は，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，マシン台数が数百台規模の場合，他の大規模分散処理システムでも適用が可能である．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．

  しかし，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．

6. 解決手法のまとめと考察

6.1 各課題と解決手法のまとめ

本研究では，計画された期間内で検証を完了することが難しい大規模分散処理システムにおいて，検証作業の重要度を分析し，期間内で完了するための効率化すべき作業を明確にした．そして，計画した期間で重要度の順に検証することで，検証精度と計画した検証期間とのトレードオフをコントロールする手法を提案した．

明確にした重要度の高い各課題とそれぞれの解決手法とを表 35 に整理して示す．計画のアクティビティでは，「大量な検証データの早期データ準備」，「検証実施時間の見積もり精度向上」，「効率的にバグ摘出するマシン台数分割による検証実施」の課題を取り上げた．テストケース生成のアクティビティでは，「大量な検証項目の項目削減と期間内の検証完了」の課題を取り上げた．テスト環境の開発のアクティビティでは，「性能確認や機能確認の効率化」，「故障検知と故障対応の迅速化」の課題を取り上げた．各課題に対してそれぞれ解決するための手法を提案した．

表 35. 検証の課題と解決手法

アクティビティ	作業	検証の課題	解決手法
計画	準備	課題 1:大量な検証データの早期データ準備	手法 1:I/O デバイスをチューニングした高速データ登録手法
	見積もり	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
	実施計画	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	項目作成	課題 4:大量の検証項目の項目削減と期間内の検証完了	手法 4:システム特徴と問題発生傾向による項目削減と重要度による期間内検証手法
テスト環境の開発	ツール開発	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テストドライバ(TP)の作成手法
	環境整備	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

そして，網羅的な項目を検証するのではなく，計画した期間で重要度の順に検証する．これをインターネット上の大量データを収集し，検索する商用の開発に適用した．その結果，これらの提案手法により，検証期間の削減が実現し，検証精度と検証期間のトレードオフをコントロールすることで，限られた検証期間で検証を完了した．図 27 に示すとおり，従来の情報システムレベルで計画した検証期間が 6 ヵ月の開発において，解決手法を適用しない場合は 1 年の期間が必要であったが，解決手法を適用した結果，作業をコントロールでき，計画された検証期間内で完了させることができた．

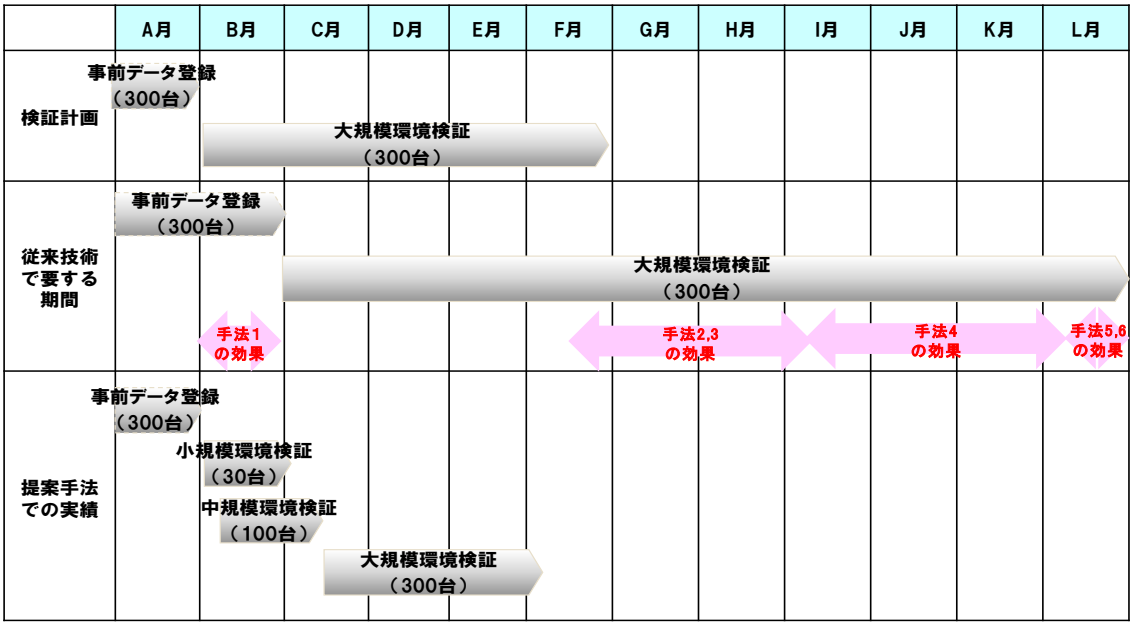


図 27. 検証の計画と実績

6.2 課題の解決による効果

課題を解決することで，検証作業をコントールでき，品質を確保して期間内での検証を完了できる．提案した各アクティビティにおける課題の解決手法とその効果を表 36 に整理する．

表 36. 課題解決による効果

アクティビティ	解決手法	効果(到達点)	
計画	手法 1 (検証作業の合理化による検証期間の削減)	検証計画:1 ヶ月 従来手法で要する期間:2 ヶ月 提案手法での実績:1 ヶ月	・計画したスケジュールどおりの検証の実現 ・事前のリスク軽減
	手法 2 (検証作業の合理化による検証期間の削減)	検証計画:5 ヶ月 従来手法で要する期間:約 6 ヶ月 提案手法での実績:5 ヶ月	
	手法 3 (検証作業の合理化による検証期間の削減)	検証計画:5 ヶ月 従来手法で要する期間:約 7 ヶ月 提案手法での実績:5 ヶ月	
テストケース生成	手法 4 (検証項目の重要度付けによる検証精度と検証期間とのトレードオフのコントロール)	検証計画:5 ヶ月 従来手法で要する期間:約 8 ヶ月 提案手法での実績:5 ヶ月	・品質を確保して網羅的な検証項目の半分以下の工数での検証完了
テスト環境の開発	手法 5 (検証作業の合理化による検証期間の削減)	検証計画:7 日 従来手法で要する期間:0.5 ヶ月 提案手法での実績:7 日	・検証実施時間の短縮 ・再実施のない検証の実現
	手法 6 (検証作業の合理化による検証期間の削減)	検証計画:0 従来手法で要する期間:0.5 ヶ月 提案手法での実績:0	

1 点目の課題「大量な検証データの早期データ準備」の解決手法は，手法1「I/O デバイスをチューニングした高速データ登録手法」である．手法1は，ネットワーク I/O には「TCP バッファのサイズを 1.5 倍」にするチューニングを適用して性能要件以上の書き込み速度の実現し，ディスク I/O には「Merge Compaction の実行契機のランダム化」のチューニングを適用してスループットを安定させることで，高速登録を実現した．手法 1 により，検証期間を削減でき解決手法の適用前では約 2 ヶ月要するところ，適用後では計画どおり 1 ヶ月の期間内で完了させることができた．

2 点目の課題「検証実施時間の見積もり精度向上」の解決手法は，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」である．手法 2 は，大量にリソースを使用する検証の観点で類似の項目が 2 項目以上ある場合，そのうち 1 項目を代表的な項目として事前に検証を実施することで見積もり精度の向上を実現した．手法 2 により，検証期間を削減でき解決手法の適用前では約 1 ヶ月の遅れを生じていたが，適用後では計画どおり 5 ヶ月の期間内で完了させることができた．

3 点目の課題「効率的にバグ摘出するマシン台数分割による検証実施」の解決手法は，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」である．手法 3 は，検証マシンを分割し，マシン台数が小規模から大規模までの数段階の検証環境に検証項目の内容に応じて項目を割り当てて段階的に検証することで，問題の解析を容易にすることができ効率的な検証を実現した．手法 3 により，検証期間を削減でき解決手法の適用前では約 7 ヶ月要するところ，適用後では計画どおり 5 ヶ月の期間内で完了させることができた．

上記のとおり，計画のアクティビティでは，上記 3 つの課題を解決することで，検証期間を削減でき，かつ事前のリスクを軽減し，計画したスケジュールどおりの検証をすることができた．

4 点目の課題「大量な検証項目の項目削減と期間内の検証完了」の解決手法は，手法 4「システム特徴と問題発生傾向による項目削減と重要度による期間内検証手法」である．手法 4 は，重要な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」に絞り込み，さらに発生した問題を中心とした項目に絞り込むことで 1/4 に削減した．そして，計画した期間で重要度の順に検証した．手法 4 により，検証精度と計画した検証期間とのトレードオフをコントロールでき解決手法の適用前では 8 ヶ月要するところ，適用後では計画どおり 5 ヶ月の期間内で完了させることができた．この作業のコントロールにより，品質を確保して網羅的な検証の約半分の工数で検証を完了することができた．

5 点目の課題「性能確認や機能確認の効率化」の解決手法は，手法 5「データ生成とログの出力を組み込んだテストドライバ（TP）の作成手法」である．手法 5 は，プログラムの機能とデータの読み書きの性能を確認できるように，(1)作業効率化のために複数台の TP（Test Program）を集中制御，(2) 書き込みデータの圧縮率等を考慮して外部パラメータで任意に内容を変更，(3)スループットの計測やデータ内容の確認，の 3 つを可能にする仕組みを具備した．手法 5 により，検証期間を削減でき解決手法の適用前では 0.5 ヶ月要するところ，適用後では計画どおり 7 日で完了することができた．

6点目の課題「故障検知と故障対応の迅速化」の解決手法は，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」である．手法6は，実際の故障実績に基づきマシン全体の1割の台数を予備に持ち，Shell script や OSS の Nagios などのネットワークやマシンの監視ツールを用いて故障を検知し，故障したマシンを予備機と逐次入れ替えることで，検証への影響を最小限にした．手法 6 により，検証期間を削減でき解決手法の適用前ではマシンの調整の対応で 0.5 ヶ月要するところ，適用後では対応することなく完了することができた．

上記のとおり，テスト環境の開発のアクティビティでは，上記 2 つの課題を解決することで，検証期間を削減でき，検証実施時間の短縮と再実施のない検証を実現することができた．

6.3　解決手法の分類と適用領域

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 37 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．スケールアウト（マシン台数）とデータ量を軸とした 2 章の図 4 にそれぞれの手法とシステムの領域とをマップすると，それぞれの手法の適用可能なシステムが判明する．

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 37．解決手法の分類と適用領域

<table>
<tr><th></th><th>作業</th><th>多くのマシン台数</th><th>大規模なデータ</th></tr>
<tr><td rowspan="3">計画</td><td>準備</td><td rowspan="3">適用領域</td><td rowspan="3">•手法1(b,d)
•手法2(b,e)
•手法3(b)</td></tr>
<tr><td>見積</td></tr>
<tr><td>実施計画</td></tr>
<tr><td>テストケース生成</td><td>項目作成</td><td></td><td>•手法4(b,c,e)</td></tr>
<tr><td rowspan="3">テスト環境の開発</td><td rowspan="2">ツール開発</td><td rowspan="2">•手法５(a)
•手法６(a)</td><td rowspan="2">適用領域</td></tr>
<tr><td></td></tr>
<tr><td>環境整備</td><td></td><td></td></tr>
<tr><td>実行</td><td colspan="3" rowspan="4">前工程の解決手法で課題解消
・従来手法で対応</td></tr>
<tr><td>テスト結果の評価</td></tr>
<tr><td>問題報告/テストログ</td></tr>
<tr><td>欠陥追跡</td></tr>
</table>

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

6.3.1　他の大規模分散処理システムへの適用

他の大規模分散処理システムへの提案した解決手法の適用は，大規模分散処理システムのOSSで著名なものの一つであるHadoopへの適用が考えられる．一般的にOSSでは設計思想や詳細な設計が全て明確ではないため，複雑な問題対応や最適なチューニングには限界がある．しかしながらHadoopは，図 28と図 29のようにCBoCタイプ2とプロダクト構成など共通する部分が多い．そのため，CBoC タイプ 2 と同様に，マシンを数百台規模使い，プロダクトが複数分散し連携して動作する Hadoop は，提案した解決手法を適用することで検証の効率化ができると考えられるからである．

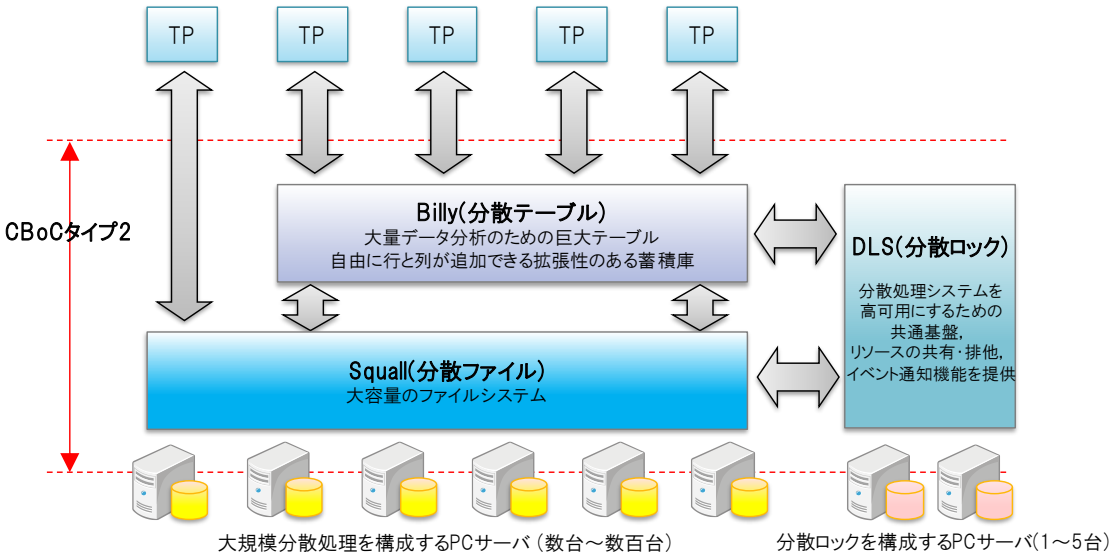


図 28. CBoC タイプ 2 のシステム構成

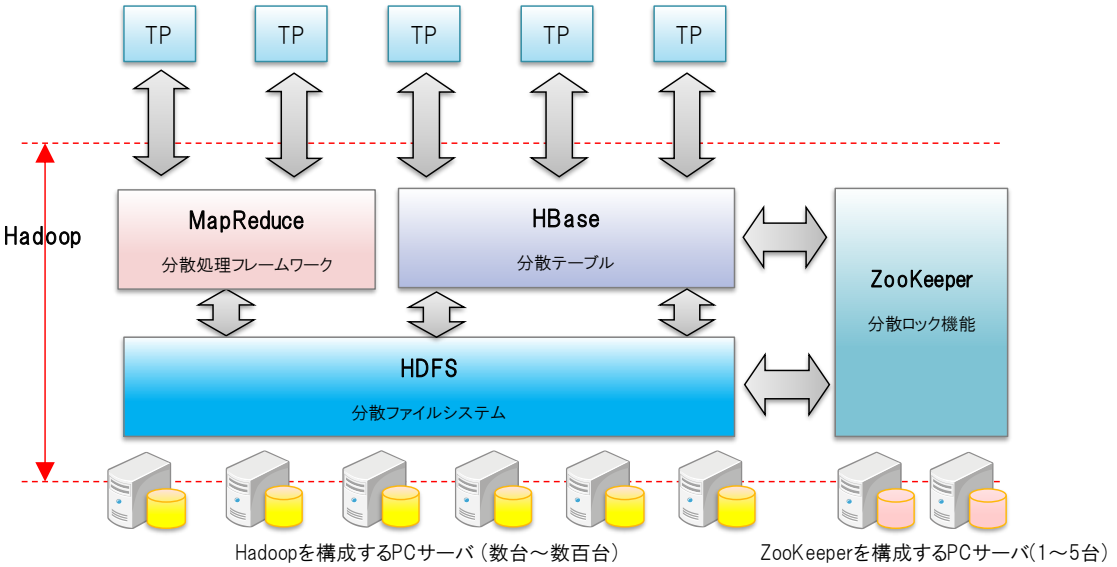


図 29. Haddoop のシステム構成

実際に Hadoop に機能検証や性能検証として適用したところ，本手法はそのまま適用できることを確認できた．このことから，解決手法の考え方は，OSS である Hadoop にも共通する知識であり，他の大規模分散処理システムにおける検証の課題解決に役立つと考えられる．

6.3.2 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 や前項の Hadoop というレコードオリエンテッドストア[16]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[17]な分散処理システムやドキュメントストア[18]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，2.2.3 項に記述した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 38 に各システムと提案手法の適用可能性を示す．

表 38. 他システムへの提案手法の適用可能性

システム種類		システムの特徴 ・構成とデータ管理 ・特徴	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	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
			関係する処理特性	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）	（2.2 節を参照） ・レコード単位のデータ操作に優れる ・シーケンシャルリードに優れ，小粒度のランダムリードは劣る	a.1K～10K b.～PB c.シーケンシャルリード，ランダムライト d.単一レコードのみのトランザクション（レコード単位のロック） e.全レプリカ同期更新	○	○	○	○	○	○

[16]https://docs.aws.amazon.com/ja_jp/redshift/latest/dg/c_challenges_achieving_high_performance_queries.html#columnar-data-storage

[17] https://www.techcrowd.jp/nosql/column/

[18] https://www.techcrowd.jp/nosql/documentdb/

(2)カラムオリエンテッドストア	・ノードと呼ばれる複数台のサーバで構成され分散してデータ管理 ・カラム単位のデータ集計に優れる	a.1K～10K b.～PB c.ランダムリード，ランダムライト d.レコード追加ではすべての列の操作が必要（排他制御は簡易） e.複製データはノードに非同期でコピー	×	○	○	○	○	○
(3)ドキュメントストア	・1 台のプライマリサーバと複数のセカンダリサーバで構成され分散してデータ管理 ・スキーマレスで柔軟なデータを扱うことが可能	a.～1000 b.～PB c.シーケンシャルリード d.データ一貫性を保証したトランザクション処理 e.複製データは一定のサーバで同期	○	○	○	○	○	○
(4)RDBMS	・1 台のサーバで複数のクライアントで分割されたデータを管理 ・トランザクション管理されデータは保障される	a.～100 b.～TB c.ランダムリード，ランダムライト d.2 フェーズコミットによるデータ一貫性を保証したトランザクション処理（テーブル/レコード/カラム単位のロック処理） e.ジャーナル処理によりデータをバックアップ	×	○	×	×	×	×
(5)MapReduce	NoSQL のシステムと同じ特徴・処理特性		NoSQL のシステムに依存					

凡例:「○」効果高,「×」効果低

(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，2.2 節で記述したとおり，行単位でデータを書き込み，読み込みをするのに優れている．そのため，6.3.1 項で記述したとおり，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[17]，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[19]，Oracle Exadata[20]，Netezza[21]，SAP HANA[22]，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重要度の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テストドライバ(TP)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．

[19] http://cassandra.apache.org/
[20] https://www.oracle.com/technetwork/jp/database/exadata/overview/index.html
[21] https://www.ibm.com/jp-ja/analytics/netezza
[22] https://www.sap.com/japan/products/hana.html

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[17]，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 デバイスをチューニングす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重要度の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テストドライバ（TP）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[23]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[18]のあるシステムである．RDBMS のシステムには，SQL Server[24]，MySQL[25]，Oracle Database[26]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 デバイスをチューニングす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バックグラウンドで処理がされるため，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．そのような工夫のもと大規模な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重要度の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テ

[23]https://www.kddi.com/yogo/%E6%83%85%E5%A0%B1%E3%82%B7%E3%82%B9%E3%83%86%E3%83%A0/RDBMS.html
[24] https://www.microsoft.com/ja-jp/sql-server/sql-server-2019
[25] https://www.mysql.com/jp/
[26] https://www.oracle.com/jp/database/products.html

ストドライバ（TP）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[91]．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 2 章の図 4 の領域を用いて図 30 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 30 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 30 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 30 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，重ねて表示はしていない．

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 30 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 3，手法 4 は， 主に大量なデータ（ペタバイト級のデータ）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分散されたマシンのシステムの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．手法 2 は，数百テラバイト級のデータを管理するシステムの RDBMS に適用可能である．

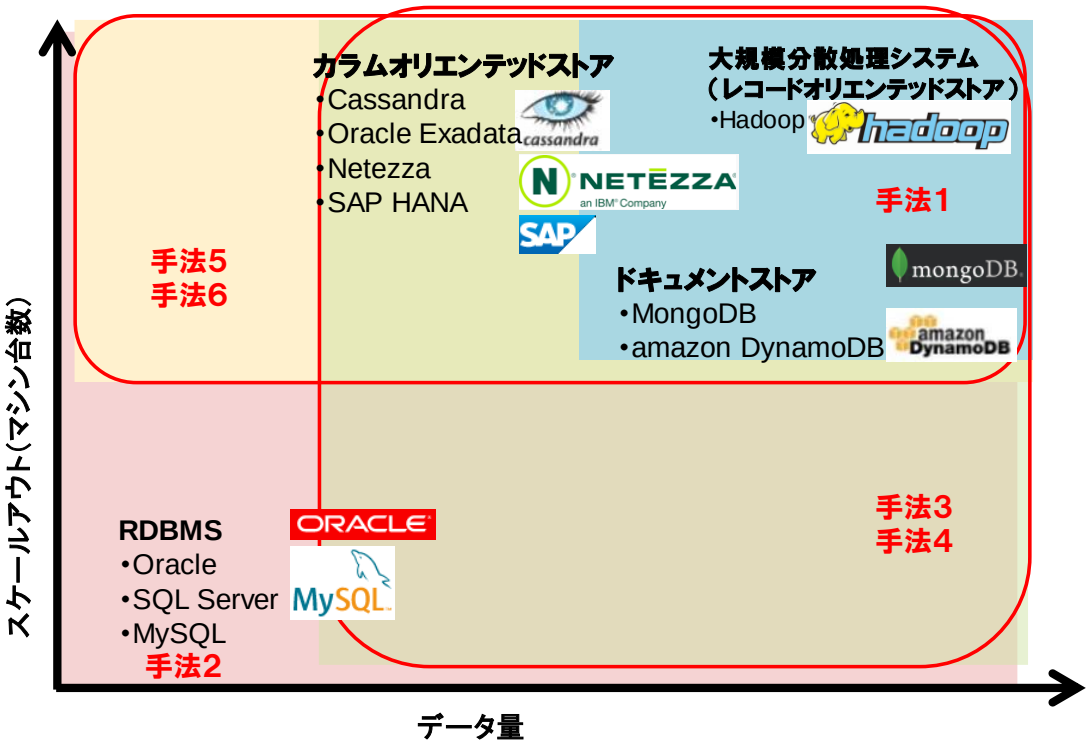


図 30. 代表的な他の分散処理システムやRDBMS における本成果の適用範囲

7. 結論

7.1 本研究の到達点

本研究は，プロダクト全体を連携して動作し，要求仕様を確認する検証プロセスを扱う事例研究が公知となっておらず，計画した期間内で検証を完了することが難しい大規模分散処理システムにおいて，(1)検証作業の合理化による検証期間の削減，(2)検証項目の重要度付けによる検証精度と検証期間とのトレードオフのコントロール，の 2 点を行い，計画した期間で作業を完了することを目的とした．

本研究の結果，まず検証作業の重要度を分析し，期間内で完了するための効率化すべき作業を明確にした．次に効率化すべき検証作業で検証工程のアクティビティの「計画」と「テスト環境の開発」で作業の合理化による検証期間の削減を行い，見積もりで検証期間の計画を行った．また，「テストケースの生成」で検証項目の重要度を分析し，網羅的な項目を検証するのではなく，計画した期間で重要度の順に検証した．この検証精度と計画した検証期間とのトレードオフのコントロールと，これらの手法を利用する方法とを提案した．

これらの提案手法を，インターネット上の大量データを収集し，検索する商用の開発に適用した．その結果，これらの提案手法により，検証期間の削減が実現し，検証精度と検証期間のトレードオフをコントロールできることで，限られた検証期間で検証を完了し，検証期間を従来の 1/2 以下に短縮できた．加えて，これらの提案手法について，他の大規模分散処理システムへの適用範囲を整理した．

7.2 研究の展望

実施例の少ない検証では，実際に発生した課題の解決が重要である．しかし，課題の解決だけではなく，課題の抽出と整理そのものが，実施例の少ない検証の体系化に役立つ可能性がある．今後も実施例において，継続した課題の抽出と整理とが期待される．

本研究で対象とした BigData は，分析の処理をする対象のデータであり，センサーデータやログデータや画像や音楽などの大量でかつ形式が様々なデータである．この BigData の分析の処理をする大規模分散処理システムは，ハードウェア性能の向上に伴い，更なる大量で形式が様々なデータを高速に，かつ複雑な処理することが予想される．また，コモディティ化された PC を使っての大規模分散処理システムは，エコシステムといった他システムとの連携が進むことが予想される．今後は，これらの様々な要求条件や環境の変化に対応したシステム検証が必要になると予想され，検証手法の研究の発展が期待される．

7.3 今後の課題

本研究で課題は漏れなく抽出し整理した．そして，主要な課題の解決手法については，実際の開発に適用することで効果が確認できた．しかし，本論文で扱わなかった課題については，解決

手法の提案と解決手法の大規模分散処理システムへの適用評価はできていない．例えば，他の課題は計画アクティビティでは，環境トラブルによる蓄積データの損失という課題がある．テストケース生成アクティビティでは，精査し追加した検証で，新たに発生する問題の対応という課題があり，テスト環境の開発アクティビティでは，検証を効率的に実施する TP(Test Program)のパラメータの決定という課題がある．また，提案した解決手法は，他の分散処理システムへの適用確認ができていない．今後も継続して解決手法の調査を進める必要がある．

付録

付録 1　大規模分散処理システムと RDBMS との違いと CBoC タイプ 2 の特徴

大規模分散処理システムと RDBMS(Relational Database Management System)との違いを以下に示す. KVS(Key-Value Store)は RDBMS とは異なり, 簡易トランザクション・簡易クエリを分散環境で実行する点に特徴があり, 分散環境において一貫性あるいは可用性を実現したデータストレージである.

KVS を利用している大規模分散処理システムと RDBMS との違いを図 31 に示す. 違いの 1 点目として, データの一貫性の違いがある. 大規模分散処理システムはパラレルにデータの書き込みを行い, 大量のデータ処理を可能とする. しかし, 単一レコード単位での一貫性はあるが, 複数レコードにまたがる一貫性は保証しない. RDBMS のデータ処理は, 書き込み対象のロック行うとともにアクセス制御によりトランザクションによって, 複数レコードにまたがる一貫性を保証する. RDBMS ではロックコントロールモジュールにより実現している. しかし, 大規模分散処理システムの分散ロックはシステムの監視のみである. 違いの 2 点目として, データ量による検索処理の違いがある. RDBMS の処理において, 大規模分散処理システムでサポートしない処理は, データの書き換え処理の「Update」と, 大規模なデータ処理となる「Sort」, 「Join」, 「Order by」等である. 「Update」の理由は, データは追加と削除のみで書き換えを行わないことから対応できないからである. 「Sort」, 「Join」, 「Order by」の理由は, 分散ファイルは 100 テラバイト～ペタバイトレベルの超巨大なデータを扱うが, RDBMS の DB はテラバイトレベルの検索対象を限定したデータを扱うため, 大規模分散処理システムで全データを操作する処理は, 処理時間が膨大となるため実運用に耐えることができないからである.

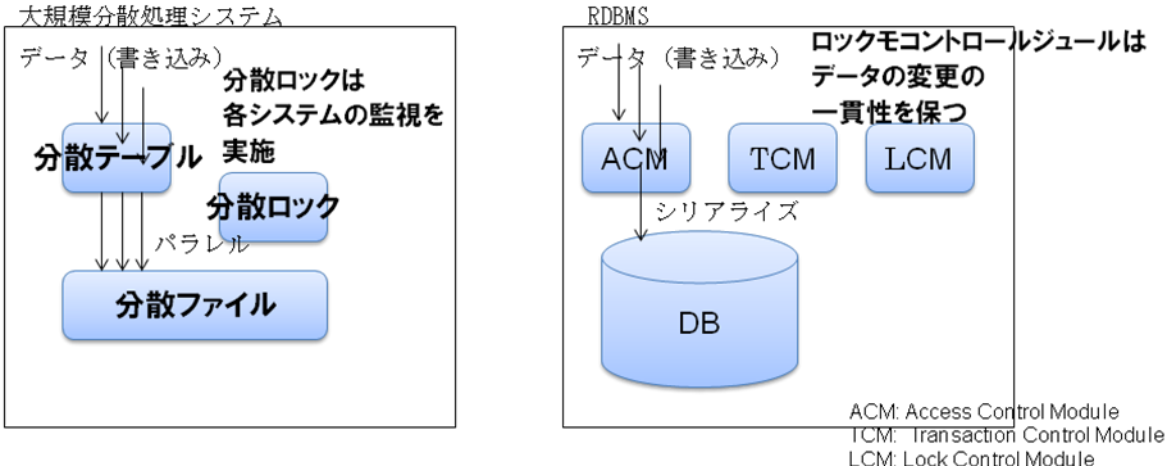


図 31 大規模分散処理システム(KVS)と RDBMS との違い

次に，CBoC タイプ 2 におけるシステムの特徴について以下に示す．

- 数百台規模のマシンを扱い，100 テラバイト～ペタバイト級のデータを扱い，2 重故障までは許容し運用可能
- 可用性（Avilability），分散性（Partition）を満たし，一貫性（Consistency）はできる限り満たす

以下にシステム設計上の特徴を項目として示す．

A) シーケンシャル読み出し性能 ＞ 書き込み性能 ＞ ランダム読み出し性能
B) 可用性 ＞ レイテンシ
C) データレプリケーション

CBoC タイプ 2 では，上記の特徴を以下の 3 つのプロダクトで実現している．

① 分散テーブルにおいて多次元テーブルを持ち大量の構造化データの読み書きを可能とする設計により実現(A)
② 分散ロックにおいてプロセスの死活監視を実施(B)
③ 分散ファイルにおいてファイルのレプリケーションを複数持って動作(C)

上記のとおり，CBoC タイプ 2 は，分断耐性，可用性が強いシステムである．ユーザは自身でデータの所在の確認をする必要がなく，ネットワークが分断された際でも分断されたサーバ群のうち，小規模な群を自動で切り離して正常に動作することができる．また，単一機器の障害によるデータ消失はなく，リカバリ機能によって障害が発生したデータをレプリカから復旧することができる．

付録 2　大規模分散処理システムの研究

近年，多く利用されてきている大規模分散処理システムの研究として，システムの問題の研究とNoSQL のシステムの研究について示す．

付録 2.1　大規模分散処理システムの問題の研究

大規模分散処理システムの問題は，大きく 2 つあると言われている[21]．一つはマシンのメモリやディスクの資源管理ともう一つは，性能最適化と言われている．以下その理由について示す．
リソース管理については，特にマルチテナントのシステムについては，1 つのマシンの資源を複数の OS で共有する場合，それぞれに互いに影響が及ばない設定をする．図 32 は左から右へ，アプリケーションから OS へ資源を分けた場合を示している．左の場合，利用するユーザは自分の資源は確保されており，他のユーザから影響は受けないが，余ったマシン資源を利用することはできなくなる．

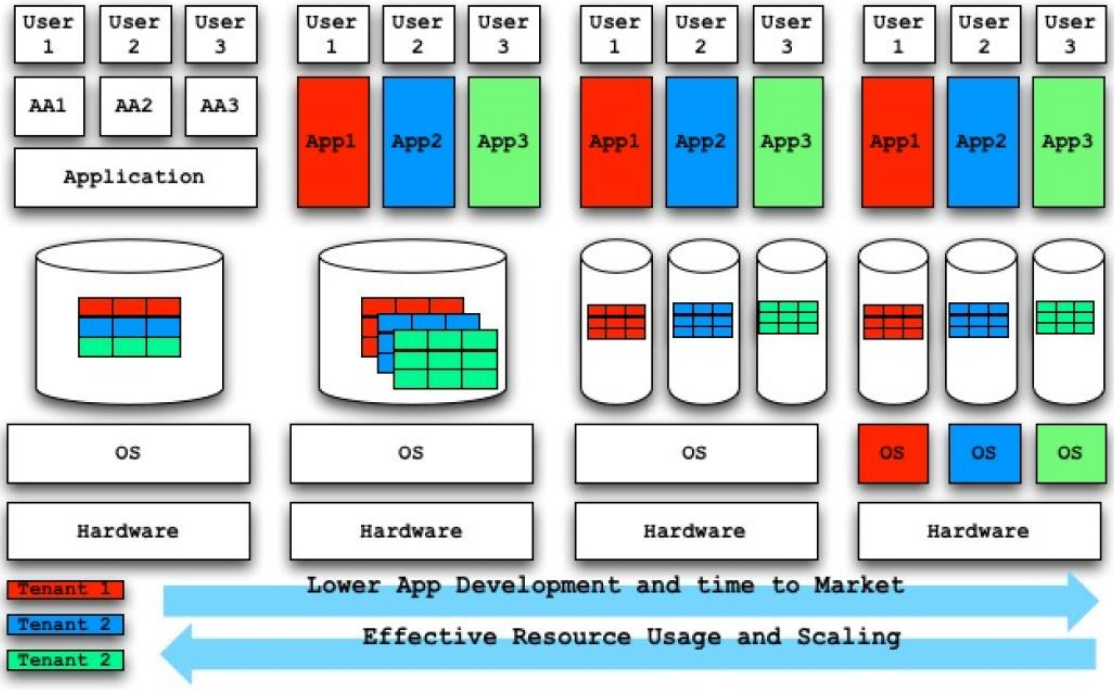


図　32．マルチテナントモデル(Bo Li: Survey of Recent Research Progress and Issues in BigData[21]より)

性能最適化については，通常の複数マシンにより構成されるシステム（以降，従来の情報システムと呼ぶ）においては重要なトピックで，アプリケーションを利用した実験から得られた結果や技術を使い，性能最適化に対応している．大規模分散処理システムも同様にアプリケーションの実験により，性能最適化に対応している．

また，システム全体の問題の観点としては，資源管理と性能最適化は，大規模分散処理システムにも当てはまる．特に，具体的な課題にある検証データ用のデータ登録では，データ投入量を最大にするため，マシンの資源と性能最適化が必要である．

CBoC タイプ 2 では 1 マシン 1OS であり，マルチテナントモデルではないため，他の OS との資源の競合は発生せず，マシン資源を最大限に活かすことが可能である．

付録 2.2 NoSQL の研究

対象である大規模分散処理システムは，NoSQL[11]である．NoSQL のシステムとしては[10][16][17]にあるように，キーバリューストア(KVS)，ドキュメントストア，カラムオリエンテッドストアの 3 種類のシステムがある．以下の表 39 に NoSQL の分類の代表的なシステムを一覧にする．

表 39. NoSQL の分類とシステム(Cattell, Rick: High Performance Scalable Data Store[17]より)

分類	システム
キーバリューストア	Redis Scalaris Tokyo Tyrant Voldemort Riak
ドキュメントストア	MongoDB CouchDB SimpleDB Terrastore
カラムオリエンテッドストア	Cassandra HyperTable

表中の一部のシステムについて，特徴を以下に示す．

キーバリューストアのシステムとして Redis と Tokyo Tyrant を示す．

(1) Redis

高速で小規模の容量のデータを扱うのに適している．詳しくは以下に示す特徴を持つ．

① メモリデータシステムである．システム走行時はメモリ上にデータを載せておき，それ以外の時はハードディスク上に保存する．読み書きを 10 万レコード/s で実行可能な性能を持つ．

② リストとリストのセットの組み合わせる操作を可能とする．

③ 1GB のサイズの値を扱うことが可能．

④　大規模の容量のデータは扱えない.

(2) Tokyo Tyrant

日本の mixi に使われているものである. TC という高ストレージエンジンと TT というマルチスレッドで高同時制御のサーバから構成されており, 4 万～5 万回/s の読み出しと書き込みの性能を持つ. TC はデータの信頼性を保つ機構を持っている. TC はキーバリューの構造だけではなく, リレーショナルデータベースの構造も持っている. 加えて, ページングやソートといった単純な問い合わせのみをサポートしている.

カラムオリエンテッドストアとして, Cassandra と Hypertable を示す.

(3) Cassandra

Cassandra は Facebook で使われているオープンソース(OSS)であり, 以下の特徴がある.

①　スキーマが柔軟で構築時に必ずしもスキーマの定義が必要ではなく, フィールドの追加, 削除が容易である.

②　キーとしてレンジの問い合わせをサポートしている.

③　1 台が故障しても, 全体に影響を及ぼさずリニアにスケールする, 高スケーラビリティを実現している.

(4) Hypertable

オープンソースの検索エンジンである. 1000 ノードのシステムで設計しており, 書き込みは 7MB/s, 読み出しは 1M セル/s である. しかしながら, 高い読み出し速度や大きなストレージのアプリケーションの利用は向いていない.

ドキュメントストアとして, CouchDB と MongoDB を示す.

(5) CouchDB

Apache プロジェクトのシステムで, フレキシブルでフォールトトレラントのシステムである. 加えて, ACID に対応している. データのレプリケーションを持っており, P2P の分散処理システムのサービスに対応している. インターフェースとしては, HTTP の REST ベースのみであり, 読み出しと書き込みの性能は他と比較して高くない.

(6) MongoDB

リレーショナルデータベースと非リレーショナルデータベースの中間に位置するシステムである.

①　非リレーショナルデータベースであるが, リレーショナルデータベースの機能を持つ

②　複合データを格納できる

③　単一テーブルに対してリレーショナルデータベースの問い合わせが可能

④　50GB を超えるデータに対し，MySQL の 10 倍のアクセス速度を持つ

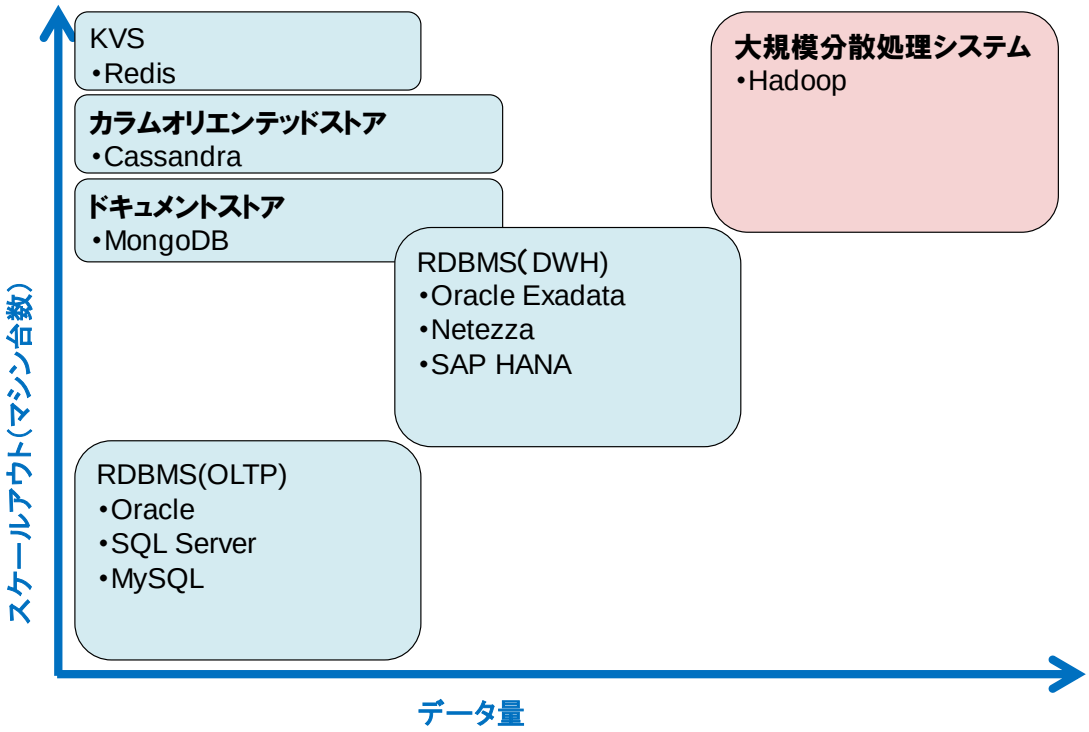


図 33. 各システムの特徴による位置づけ

キーバリューストア(KVS)，ドキュメントストア，カラムオリエンテッドストアと大規模分散処理システムと RDBMS の関係を図 33 に示す．大規模分散処理システムである Hadoop は，カラムオリエンテッドストアのスループットが上がるように，データ処理を並列化する仕組みを取り入れている．このように，大規模分散処理システムは，スケールアウト（分散）と高スループットが特徴なシステムである．

CBoC タイプ 2 は，Hadoop のシステムと同様なプロダクト構成をしたシステムである．構成するプロダクトの分散テーブルは，カラムオリエンテッドストアの Hypertable や HBase(Hadoop)のシステムと同じである．CBoC タイプ 2 の分散テーブルは，Hypertable と比較して書き込みに性能を高くするようにしているが，大きなストレージのアプリケーションの利用も可能である．HBase(Hadoop)と比較して，YCSB(Yahoo! Cloud Serving Benchmark)[19]のベンチマークを利用して書き込みや読み出しのスループットの性能は，CBoC タイプ 2 が約 2 割以上よい性能であった．CBoC タイプ 2 は大量の小さいデータサイズの変更は得意である．CBoC タイプ 2 と Hadoop とでは，1 カラムのみの更新では，Hadoop の性能が良い．ただし，全カラムを更新の対象とし，更新するデータのサイズを大きくした場合には，CBoC タイプ 2 の性能が Hadoop を上まわる．

付録 3　検証項目の並行実施

大規模分散処理システムのシステム検証では，可用性の確認のために大規模の環境で長安や単一障害等の検証をする必要がある．しかし，問題発生時にはプロダクト間にまたがった問題解析や再現検証に検証マシンを利用するため，利用可能な全てのマシンを用いた大規模環境を利用すると，同時に並行して検証と問題発生時の再現試験や問題解析ができない．このため，限られた検証環境を効率的に活用した検証項目の変更実施による検証期間の更なる短縮が必要となる．

3.5 節の効率的な規模の検証環境への検証項目割り当てを実施した後，実施順序を考慮して検証項目の並行実施することが重要な課題となる．この具体的な課題について，検証環境への検証項目割り当てを簡略化した例で以下具体的に確認する．図 34 上段に示すとおり，例えば検証期間が 2 週間のプロジェクトで，検証項目 A, B, C・D はそれぞれ検証環境規模の異なる項目であり全て 1 日を要する項目とする．項目 A は 4 件，項目 B は 3 件，項目 C と D は 2 件あり，それぞれの項目の切り替わりでは環境整備に 1 日要すると仮定する．検証項目を C,D,B,A の順に逐次実行する場合は，検証期間は 14 営業日必要であり，予定の検証期間 2 週間をオーバしてしまう．そのため，マシンリソースと人的リソースの並行化が行われるように検証計画を立案する必要がある．一方で，効率化のため検証環境を組み合わせるが，最初から多重障害を実施すると問題が多く発生するため，単一障害を実施し問題を対処した後に多重障害を実施するなど，順序性を無視して検証項目を組み合わせることができない．

付録 3.1　検証項目実施の並行化に関する研究

スケジューリングに関する研究としては，生産工場における機械が処理するタスクのスケジューリングの研究[73]されている．タスクの処理時間に不確実性がある場合や突発的な機械故障が生じる環境において，事前にオフラインでスケジュールを最適化するのではなく，環境変化に追従したリアルタイムスケジューリングする研究[74]がある．また，オフラインスケジューリングとリアルタイムスケジューリングを融合した，ローリングスケジュールの研究[75]もある．また，作業タスク時間の利用率を最大化するスケジューリングの研究[76]もある．これは，時間と重要度により，タスク数の制約のもと，時間の利用率を最大化することで，スケジューリングを行う．これらの手法による検証項目の検証マシンの割り当てとスケジューリングは，マシン配分とタスクのスケジューリングの考え方を組み合わせることで対応が可能である．しかし，大規模分散処理システムのシステム検証は，マシン台数の組み合わせや検証項目の順序性の考慮や，問題の発生の対応など，その他の条件を組み合わせないと，効率的なスケジューリングができない．

付録 3.2　課題解決の考え方

項目の実施順序を考慮して検証項目を並行実施する具体的な課題の解決手法として，マシンリソースを分割した小規模/中規模環境の並行実行の手法および，類似検証項目の同時実行の手

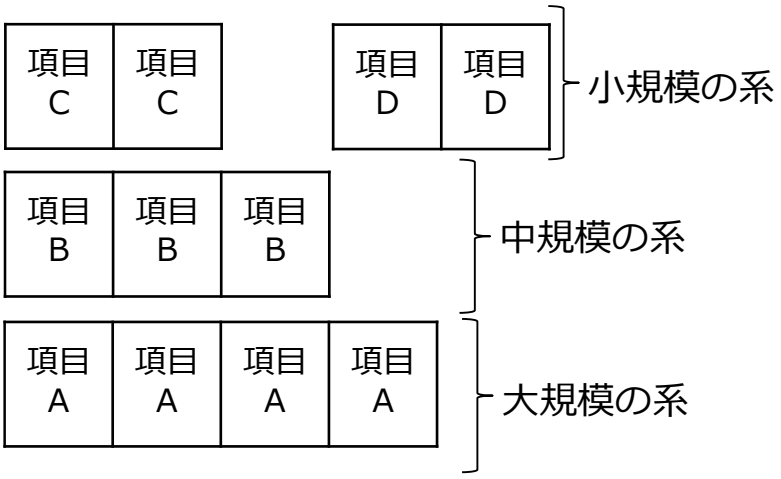
法を提案する. 4.3 節での検証項目割り当てを実施した後に段階的検証実施に基づき小規模と中規模の環境を組み合わせて期間短縮を図るとともに, 各々の規模に割り当てられた検証項目には類似の項目があることから, 検証手順が同じ検証項目をまとめて同時に実行することで, 人的リソースの効率化を図り検証期間の更なる短縮を図る. 大規模分散処理システムのシステム検証では検証項目量は多いが, 異常系の検証では組み合わせの項目で類似項目が多いため, 数多くのマシンを操作可能な検証実行ツールによる効率化が可能となる. 類似でないその他の項目は, 項目の順所性のみを考慮して実施する.

付録 3.3 具体的な解決手法

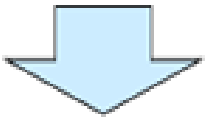
具体的には分割した小規模/中規模環境の並行実行の手法では, 図 34 下段の簡略化した例のように分割された大規模, 中規模, 小規模の検証環境を組み合わせ, 検証を並行化する. 類似検証項目の同時実行の手法では, 図 35 の簡略化した例のように分解された小規模/中規模環境に割り当てた検証項目を, 並行実行可能なように同時期に計画し, 検証実行ツールにより 1 人で検証項目を実行可能なようにする. この手法により, 22 人日から 15 人日に工数を削減することができる. また, 項目実施の順番は, 4.3 節の手法による段階的検証実施とし, 効率的に問題を摘出し解決可能とする. さらに検証の実施には検証項目の順序性も考慮する. 例えば, 異常系は小規模環境で実施し, その後に中規模や大規模環境の異常系が可能で, 複合の試験も可能となる. 単純な正常系の検証である性能の確認の項目は, 小規模環境の異常系の検証と並行して実施という項目の順序とする. 加えて, 時間をかけて事前に蓄積した検証用データが, 検証の最初に異常系の試験等で破壊されないよう, 検証用データの破壊の項目は検証の最後とする.

付録 3.4 適用結果

具体的な商用開発における適用実例として, CBoC タイプ 2 への適用結果について以下に示す. まず, 小規模, 中規模, 大規模の検証環境を並行して実行できるように組み合わせを計画した. その際には, 4.2 節での検証見積もりを適用し, 以下の点に考慮しスケジューリングを実施した. 大規模環境では, 最初のデータ蓄積は非常に時間がかかることから, 検証開始前から実施した. システム検証の後半ではデータが破壊される可能性がある検証項目を計画し, データ蓄積から再度実施となった際の影響を最小限にとどめるようにした. 中規模環境は, 検証の複雑さが少なく問題の少ないと考えられる検証項目から実施した. また, 同じくシステム検証の後半では, 異常系やデータが破壊される可能性がある検証を実施した. 小規模環境は異常系や障害の解析を中心に実施することとし, 早めの問題の先出しと解決により, その後に続くその他の運用の検証に影響が出ないように計画した. 次に, 検証項目を同時に実行できるツールの利用を可能とするため, 各検証環境の類似検証項目を同時期にまとめるようスケジュールを作成した. その実施結果として, 図 36 に示すとおりシステム検証の期間を計画の 5 ヶ月の期間内に収めることが可能となった.



注：項目Aは検証系大の項目，
項目Bは検証系中の項目，
項目C,Dは検証系小の項目



	月	火	水	木	金
1週	環境整備	項目B	項目B	項目B	予備
	項目C	項目C	環境整備	項目D	項目D
2週	環境整備	項目A	項目A	項目A	項目A
3週	（検証なし）				

検証系を分解し割当

図 34. システム検証計画例(1)

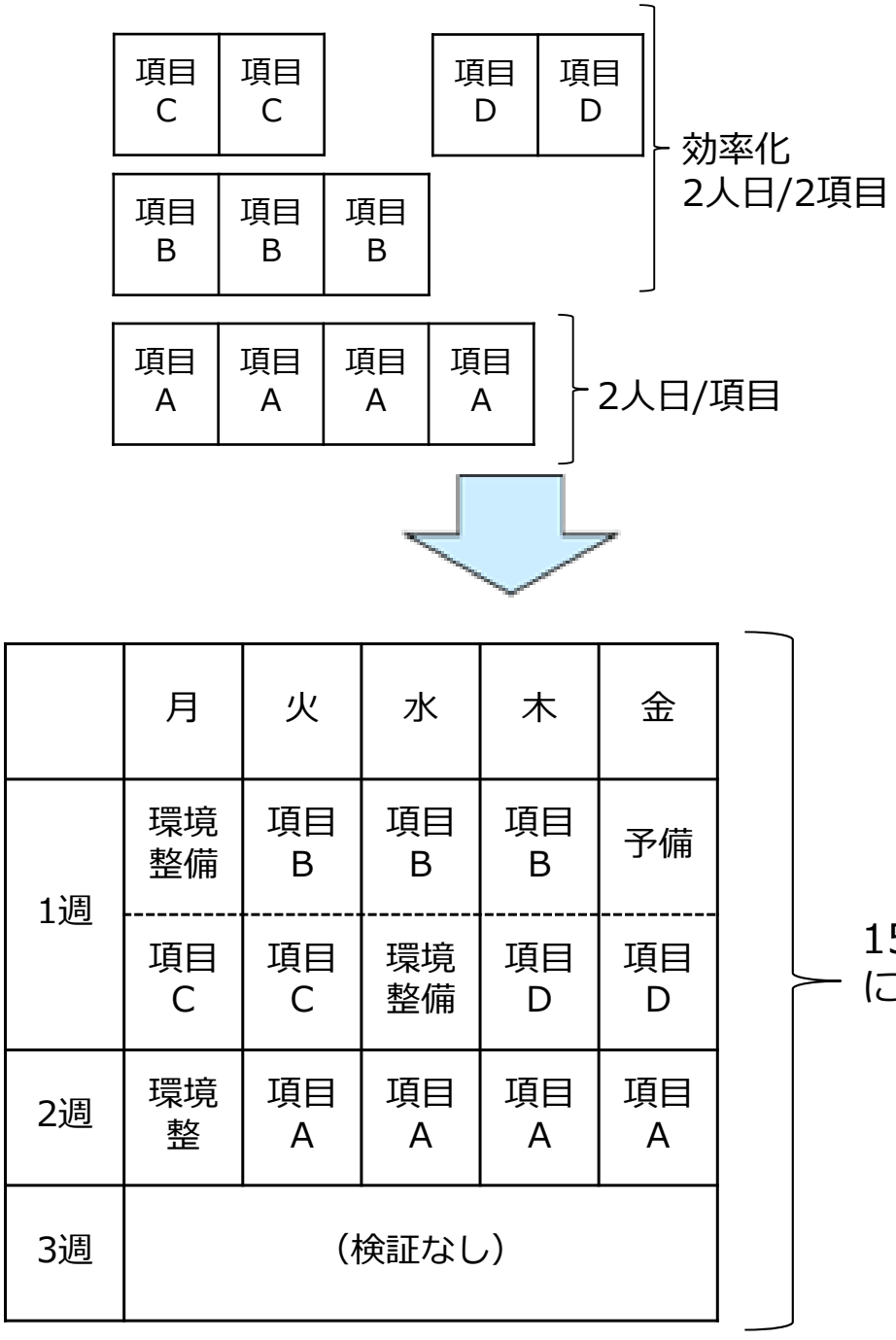


図 35. システム検証計画例(2)

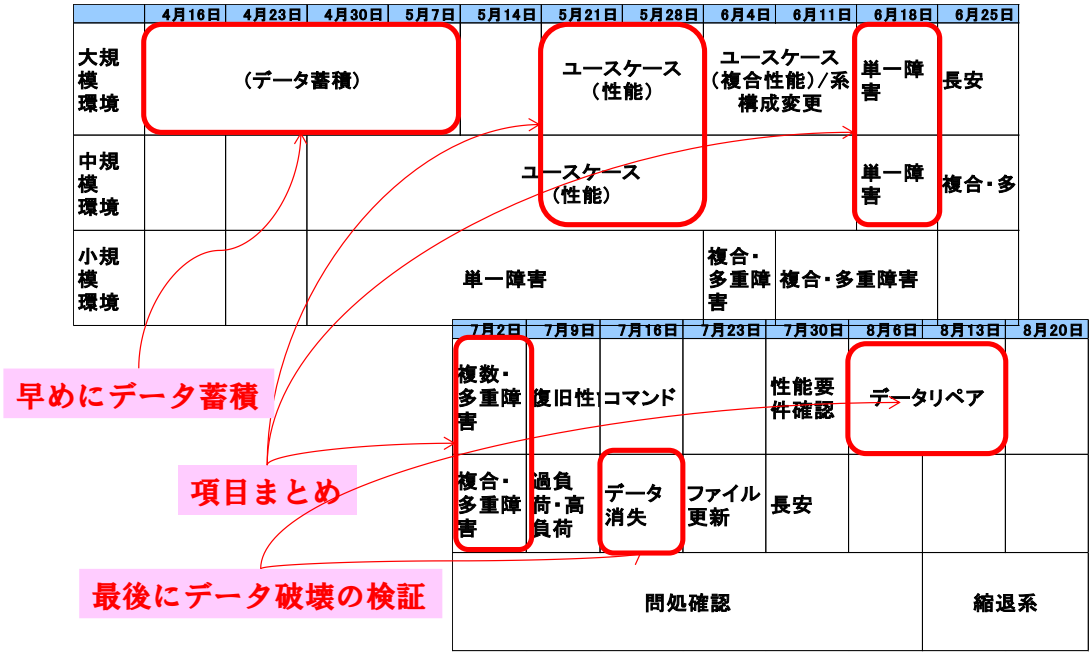


図 36. 検証環境毎の検証項目とスケジュール

付録 4　主要な検証パラメータを組み合わせるパラメータ選定の手法

大規模分散処理システムの検索は，テキスト同士の検索だけではなく，画像や音楽といったマルチメディアデータ同士の検索を行う．したがって，システム検証では，マルチメディアデータ同士の検索の機能検証を行うとともに，性能と精度を満たしているかを確認するために，非機能検証を実施する．マルチメディアデータの検索では，特徴量を使用して検索をする方式[78]があるが，特徴量のデータが大きいと検索に時間を要し，特徴量のデータが小さいと検索の精度が落ちるという問題がある．また，検証パラメータが特徴量であり，特にマルチメディアデータは，テキストデータの文字と比較すると，テキストデータは，文字として認識することであるが，他の音や動画はバイナリデータで認識する必要があり，特徴量を抽出する方法が異なる．そのため，どれが有効な特徴量かすぐに分からないという問題もある．そのため，複数のデータの小さい特徴量を使用し精度を高めることになるが，非機能検証において，特徴量を検証パラメータとし，全てを検証で確認することは，組み合わせとして数が膨大となり確認に時間を要する．そのため，特徴量のパラメータを選定して実施する必要がある．

実際のマルチメディア検索は，1 章の図 6 に示したユースケースにおいて，図 37 に示すとおりクローリングしたデータを分析するアプリケーションの処理となる．この分析のアプリケーションでは，マルチメディアデータによる検索や，マルチメディアデータでどのようなデータが検索のトップにあるかのランキング分析や，マルチメディア検索時のインデックス生成に必要なデータを抽出している．マルチメディアデータの検索時に多くの特徴量で検索すると，検索キーのデータ量が多くなるため検索時間がかかる以外に，システムへの負荷の上昇や，回線のボトルネックが発生する．そのため，検索時の特徴量を少なくし検索速度の性能を高めながら精度を高める必要がある．

マルチメディアデータの中で音楽データの場合は，テキスト検索と同じ検索性能を確保したまま検索精度を向上させることが必要である．本節では，マルチメディアデータ検索として，音楽データの検索を対象とする．

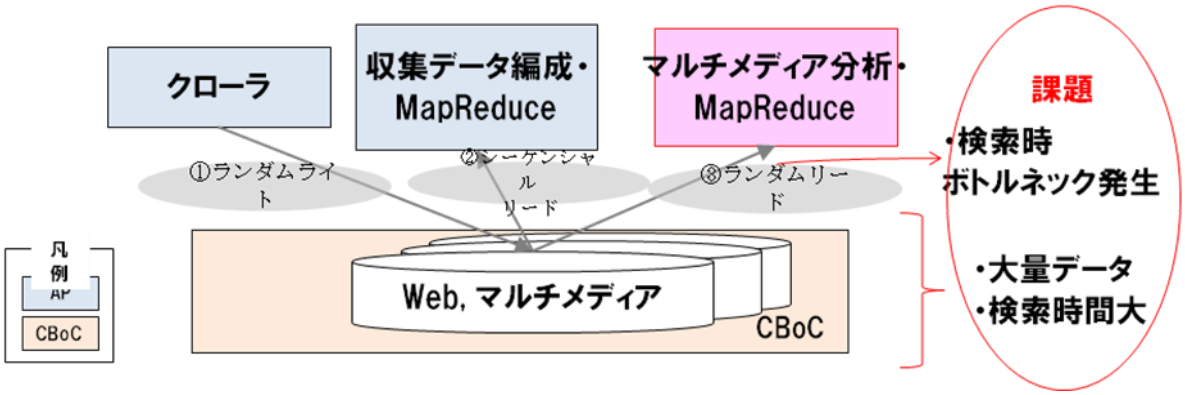


図 37. マルチメディア検索の概要

付録 4.1.1 検証パラメータの選定方法に関する研究

複数の検証パラメータがあった場合，どのパラメータの組み合わせが有効かを判別する手法として，多変量分散分析の手法[59][60]を使うことが知られている．多変量分散分析では，パラメータである従属変数間の相関を考慮し，全体の優位差があるかの検定を行う．しかし，大規模分散処理システムのように，機能確認と性能確認のみの検証の場合，要求仕様で規定されている機能と性能を満たしているかの確認のみでよく，検証結果の優位差の検定は不要である．

システム検証でどの検証パラメータを使うのがよいかは，対象とするデータをどのように分析するかによる．分析には特徴量と呼ばれるデータの一部を使い分析する．マルチメディアデータを分析する際には，テキストデータの文字と比較すると，テキストデータは，文字として認識することであるが，他の音や動画はバイナリデータで認識する必要があり，特徴量を抽出する手法が異なる[78][81]．

音楽データの分析については，マルチメディアデータの分析と同様であり，元のデータをそのまま扱うのではなく，特徴量として扱う[79][80][83]．その際に，MIDI などの楽譜情報を特徴量とする手法[79]があるが，楽譜情報に変換する精度が高くないと元のデータの特徴とならないことと，全て楽譜にする処理に時間がかかることが課題である．また，特徴量をパラメータとした場合，パラメータ 1 つで分析すると高速だが対象とする結果の精度が低い，しかしパラメータを複数にすると，対象とする音楽データの分析結果の精度は高いが検索速度の性能が悪くなる．したがって，パラメータの組み合わせが重要となる．

付録 4.1.2 大規模 マルチメディアデータ同士の検索に関する研究

マルチメディアデータの検索の関連研究を調査した結果を示す．画像や動画や音楽等のマルチメディア検索の研究は，データ周辺情報か抽出したキーワードのタグによる検索[82]や，特徴量による検索[78][81]などがある．この特徴量による検索は，本研究の手法と同じである．具体的には，画像データそのもので検索するシステム(図 38)で，画像の色や形などの特徴量を抽出し，結果を返却(図 39)する．

音楽検索[79][80][83]で類似検索する場合は，画像検索と同様である．しかし，特徴量を全体の音の長さなど単一の特徴量をパラメータとして検索キーとしている．これでは，特徴量自体のデータがテキストデータより多く，テキストの検索と同等の検索性能が出ない．音楽検索では，音を MIDI などの楽譜情報に変換して検索する方法[79]が主流である．タグを抽出する方法では，マルチメディアデータそのものの情報ではないことから，あいまい性を排除できない．また，MIDI などの楽譜情報に変換するのでは，変換する精度が高くないと検索精度が向上しないことと，全て楽譜にする処理に時間がかかることが課題である．

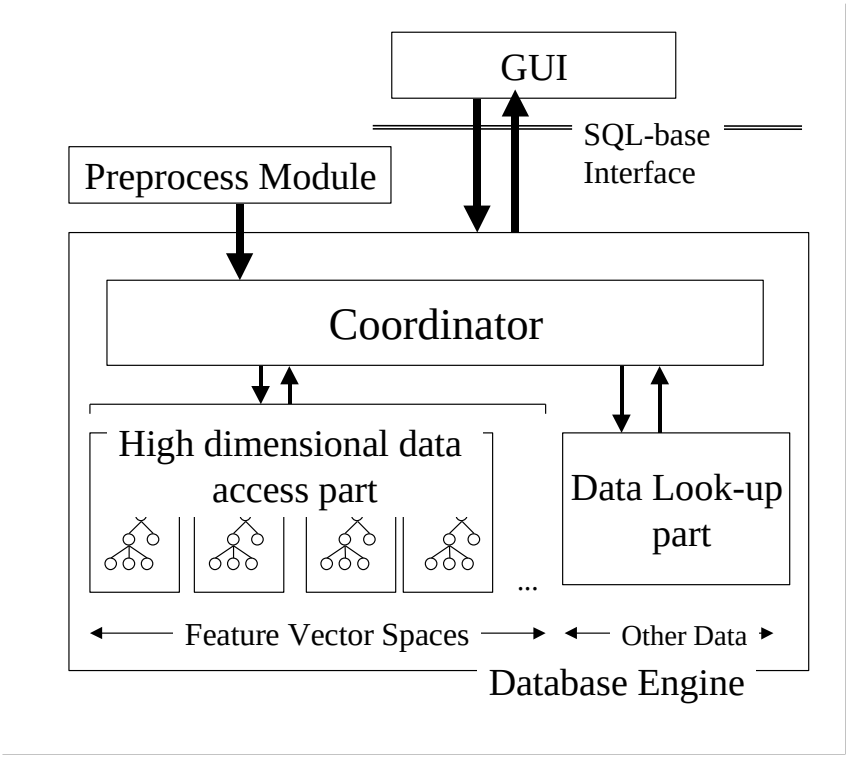


図 38. ExSight[78]のシステム構成

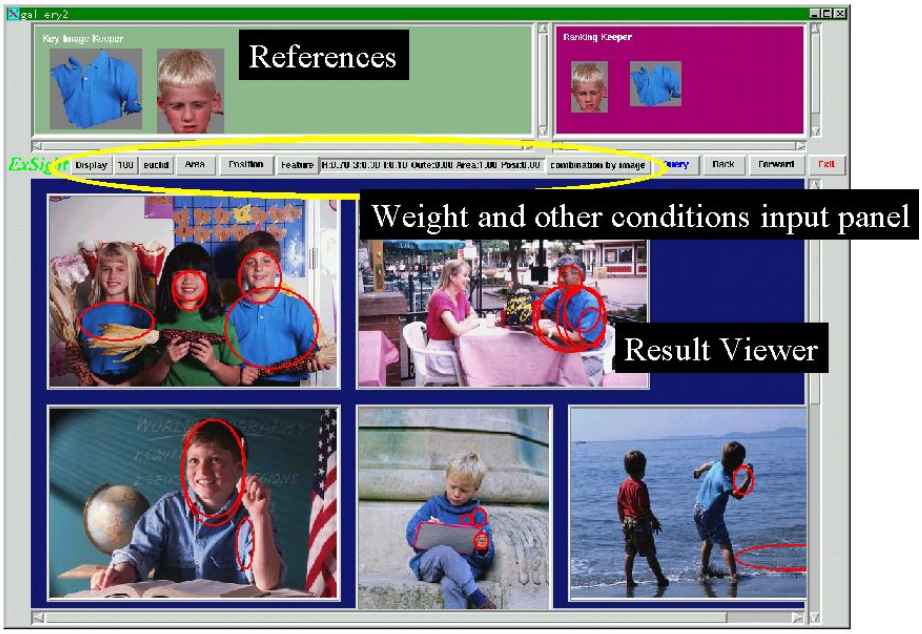


図 39. ExSight[78]の検索画面

付録 4.2　課題解決の考え方

検索速度の性能を上げる手法としては，音楽を特徴づけている音楽全体のメロディと曲の主旋律を中心とした「音長」と「音圧のピーク」と「音圧の平均」と「音圧の差分」という特徴量を利用する．また，これらを組み合わせて検索することで検索精度を上げる．ここでのメロディとは和音のように音を組み合わせた全体の曲調を指し，主旋律とは曲の中心となる音の旋律を指す．これまでは，データの波形などの特徴量1つで検索を実施していた[79]が，提案する手法は，特徴量を組み合わせることで検索精度を上げる点が異なる．特徴量の抽出方法において，検索に最小限必要な特徴的なデータを選定できれば，対象データと検索キーのデータを減らすことができ，組み合わせても検索速度を上げることができる．この解決の考え方は，特徴量を組み合わせて検索に利用するというアプローチであり，マルチメディアデータの検索をする場合，他の大規模分散処理システムでも適用が可能である．

付録 4.3　具体的な解決手法

特徴量はベクトルとして管理し，以下の 4 つを特徴量として選んだ．①対数の間隔の音圧成分（音長との面積），②周波数成分の音圧のピークとなる周波数，③周波数をある一定区間に正規化したその周波数の音圧のピークとなる周波数，④③から得られた音圧の高い周波数の時系列的に差分を取ったものである．

① 対数の間隔の音圧成分（ベクトルの次元数：18 次元）
② 周波数成分の音圧のピークとなる周波数（ベクトルの次元数：24 次元）
③ 周波数をある一定区間に正規化したその周波数の音圧のピークとなる周波数
　（ベクトルの次元数：28 次元）
④ ③から得られた正規化した周波数の音圧のピークについて時系列に差分を取った値
　（ベクトルの次元数：28 次元）

これらは音楽全体のメロディと曲の主旋律を中心とした特徴量であり，音楽の特徴を抽出している．これらの特徴量の利用で，検索速度の性能と検索精度を向上させる．以降にそれぞれの特徴量について，内容と選定した理由を説明する．

(1) 対数の間隔の音圧成分（音長の面積）

元の音楽データを FFT(Fast Fourier Transform)変換することで，その音高の合算値（図 40 のグラフの面積に相当）を特徴量のデータとして使用する．この特徴量を使用することで，テンポを調整された音の音楽データでもメロディによる検索が可能となる．また，その特徴ベクトルは対数間隔である．その理由は，人間の聞こえる音の感覚が対数間隔であることに基づいている．他の特徴ベクトルも同様に対数間隔である．

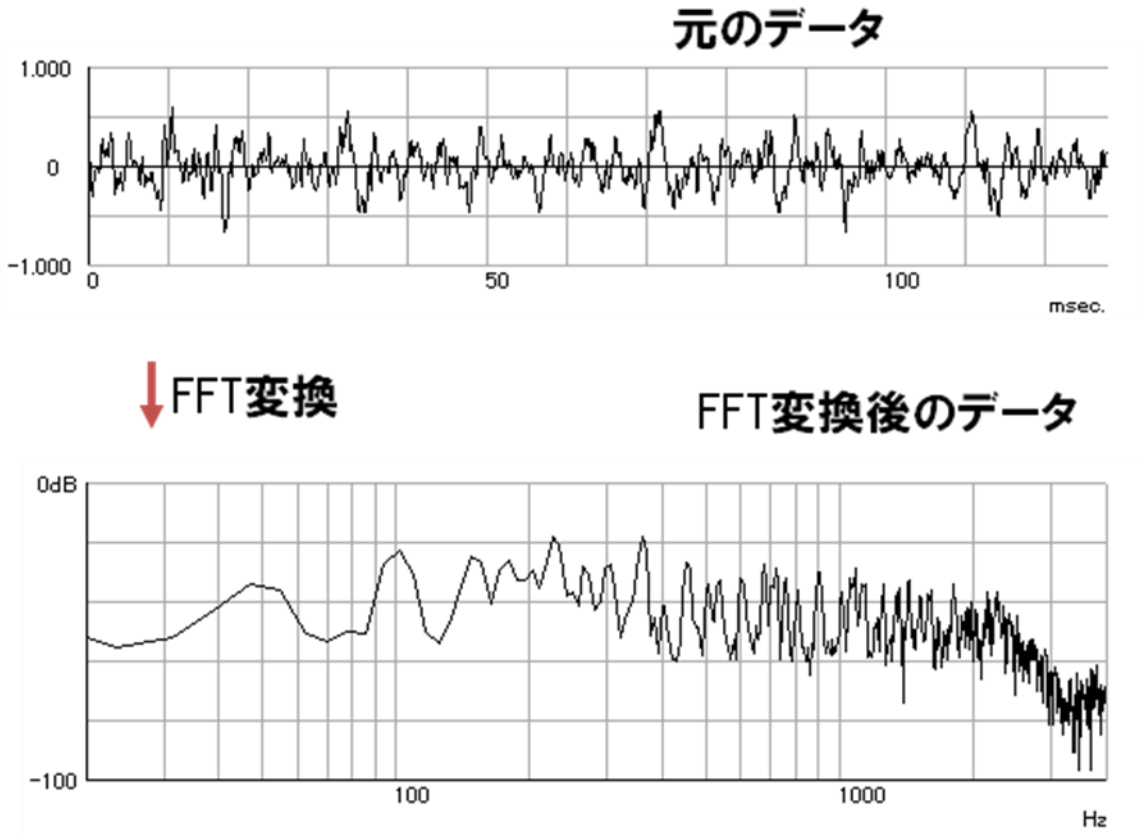


図 40. 音圧の FFT 変換

(2) 周波数成分の音圧のピークとなる周波数

FFT 変換されたグラフで，最大音圧の周波数のデータを特徴量とする．この特徴量を使用することで，いわゆるサビと呼ばれる曲が盛り上がる主旋律のデータを検索できる．それは，主旋律は音圧が一般的に高いからである．特徴量の取得は，図 41 に示すとおり，周波数帯域（グラフの横軸に相当）を 100Hz 単位の周波数帯域に分割し，その中で最大の音圧となっている周波数のデータを特徴量として取得し，検索に使用する．

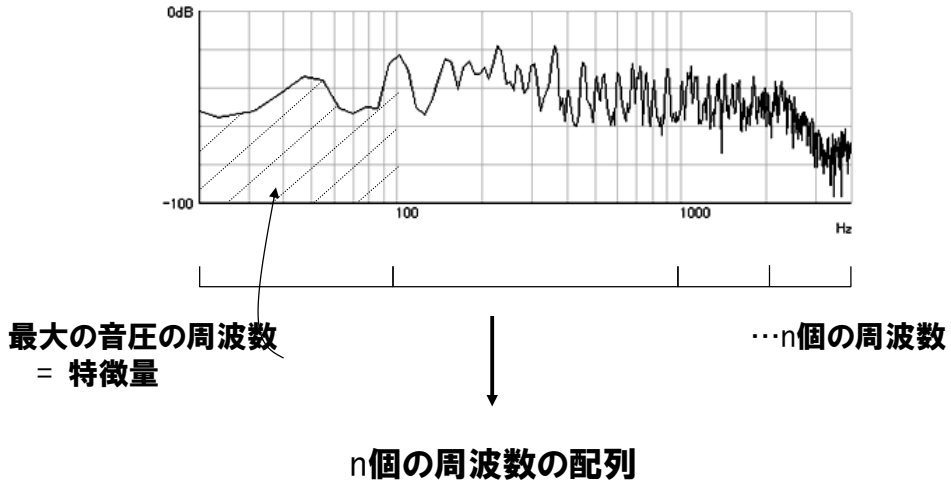


図 41. 周波数の分割

(3) 周波数をある一定区間に正規化し音圧のピークとなる周波数

元の楽曲の最大の音圧の周波数を，図 42 に示すとおり平均した値を特徴量とする．この特徴量を使用することで，どんなオクターブに変化しても検索することが可能である．なぜなら，正規化によってオクターブの変化は，それぞれの区間で統合されているからである．

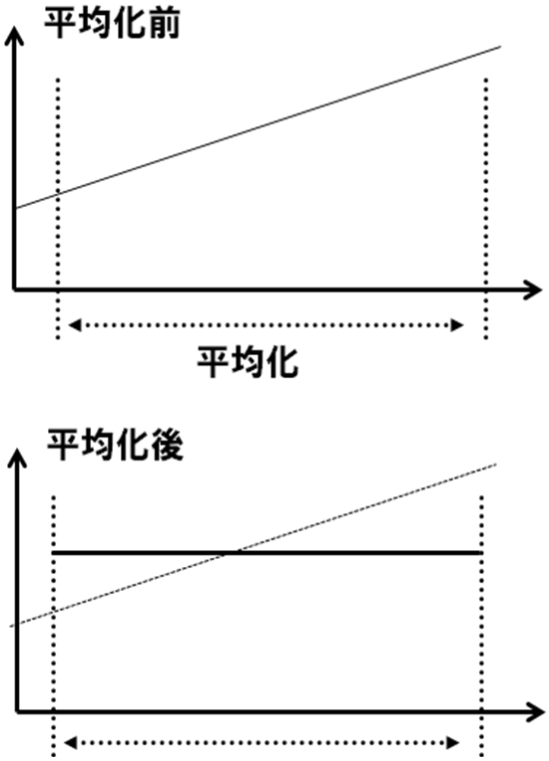


図 42. 周波数の平均化

(4) 正規化した周波数の音圧のピークについて時系列に差分を取った値

図 43 に示すとおり，一番長い周波数とその次に長い周波数の差のデータを特徴量とする．この特徴量の使用で，主旋律と次の特徴的な音との音楽的な間隔（音程差）で検索することができる．音程差の比較による検索なので，例えば，オクターブ変化したデータだけではなくテンポなどが変化していても検索ができるということである．検索結果は，音程差が近いということから，似ているメロディが検索できる．

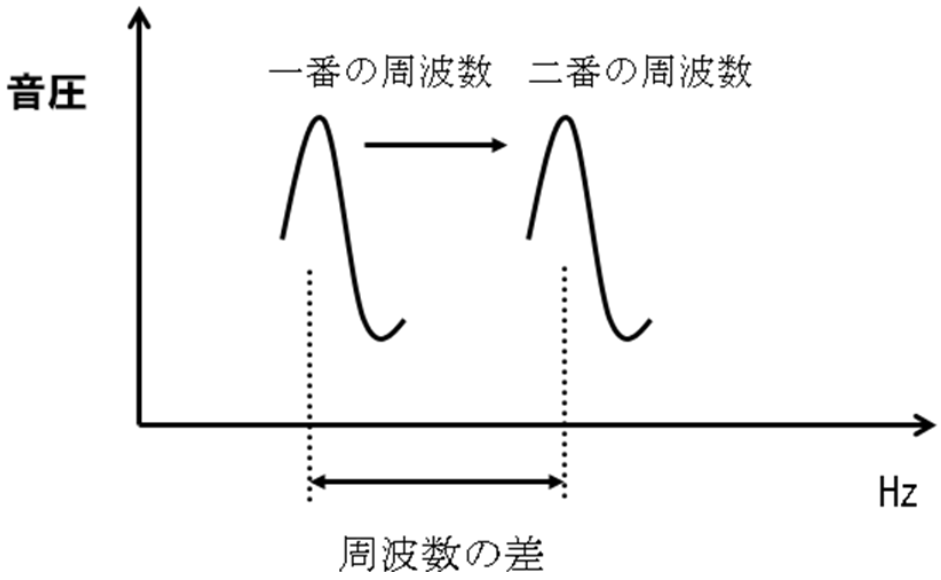


図 43. 周波数の差

付録 4.4 適用結果

この手法を適用した具体的な実例を以降に示す．上述の特徴量を検索パラメータとして使い，検索を行った．結果を表 40 に示す．50 曲のデータがデータベースとして蓄えられており，特徴量の対象となるデータは，1曲あたり 150 個程度，全体で 6400 個のデータが蓄えられている．平均検索時間は，約 0.25 秒であった．この結果，多いデータの単一の特徴量を検証パラメータとする従来の検索で，分単位のオーダであった検索時間は，1 秒以内で検索可能となった．この結果は，テキスト検索と比較して，同じオーダの検索時間であった．正解のうちどの程度が検索にヒットするかの再現率，検索結果の中にどの程度正解が含まれるかの適合率において，提案する手法では，多くの冗長なオブジェクトを抽出するため，適合率の値は特に高くはないのに対して，もし例えば多くの数の曲を問い合わせたとしても，再現率の値はかなり高い．データ量の多い音楽データの検索における高い精度の検索は，メロディと主旋律を中心とした特徴量を組み合わせることにより可能であることが判明した．検証に要した期間は，計画どおり 5 ヶ月の期間内で完了した．

表 40. 検索結果

検索回数 項目	1	2
結果数	10	10
全データ内の正解の数	5	5
再現率(%)	83	56
適合率(%)	50	50

付録 5　ツールによる検証作業の効率化

大規模分散処理システムでは，検証対象のマシンの数が数百台規模あり，手作業による検証環境の構築は，1 週間はかかってしまう．また，検証実施において結果確認は，数百台規模のログ確認に時間を要する．このため，検証環境構築と検証実施／結果確認の効率化が必要である．

従来の情報システムでの Web システム等において，検証実施は，自動化が可能[88] であり，検証結果の確認も自動で可能である．しかし，大規模分散処理システムのシステム検証では，Hadoop のツールとして環境構築をサポートするツール[89] はあるが，検証実施と検証結果の確認については，ログの解析や OSS の解析ツールの利用が考えられるが，どのツールが利用可能で，どのように利用すればよいのか公知となっていない．本節では，大規模分散処理システムのシステム検証における，検証環境構築と検証実施／結果確認の効率的な実施という課題に対し，どのようなツールが利用可能か明らかにする．

付録 5.1　ツールによる検証作業の効率化に関する研究

検証環境の構築や検証実施のツールは，従来の情報システムではそのプロダクトに付属する環境構築のツールや OSS[84][86]を利用し，不足しているところについては，ツールを自作している．構築時には，RPM のインストールには，従来の情報システムにおけるツールや OSS を利用できるが，大規模分散処理システムのプログラムのインストールは，設定ファイルの設定が必要であるものや，実行時にパラメータチューニングが必要なものや，検証時に読み書き同時の性能測定などシステム特有の操作である．これらの作業は，OSS での共通的な機能では対応することができない．実行時は，従来の情報システムでは，検証実施は Web システムなどの自動検証が可能[88]であり，検証結果の確認も可能である．しかし，大規模分散処理システムのシステム検証では，Hadoop のツールとして，環境構築をサポートするツール[89]はあるが，検証実施と検証結果の確認については，どのツールが利用可能か公知となっていない．

付録 5.2　課題解決の考え方

初期のシステム開発の場合には，ツールが整備されておらず全て手作業である．そのため，検証作業の効率化の 1 つとしてツールの利用による自動化がある．自動化の他に検証の手順を効率化するという手法もあるが，数百台規模のマシンを操作する手作業の手順に対する効率化では，同一な作業が多くないため，まとめることによる効果は少なく，効率化は限界がある．大規模分散処理システムのシステム検証では，このツールによる効率化が有効と考え，汎用的な作業については OSS の適用を行う．また，OSS が適用できない独自の作業については，検証環境構築と検証実施／結果確認それぞれにスクリプトのツールを用意するという手法で対応する．OSS が適用できる作業は OS のライブラリ等のインストールや OS／NW の状態確認であり，通常のマシン運用作業については適用ができ効率化を図ることができる． OSS が適用できない作業としては，独自開発したプロダクト等におけるインストール時の設定ファイルの設定，実行時の設定値の変更，検証時にお

ける読み書き同時の性能測定等，システム特有の作業である．これらは OSS での共通的な機能では対応できないため，独自のスクリプトツールを適用する．

この解決の考え方は，検証環境の構築や実行や結果確認における OSS の利用とスクリプトの作成というアプローチであり，多数のマシンに対する制御や監視が必要な場合，大規模分散処理システム一般に有効である．

付録 5.3　具体的な解決手法

OSS が適用できる作業については，表 41 に示すとおり効率化を図る．設計（環境構築）としては，Puppet を使い，実行（検証準備）では，iperf,bonnie を使い，実行（状態確認）には，MRTG,Nagios,Crane という利用可能な OSS を利用する．独自のスクリプトのツールについては，表 42 に示すとおり対応する．表の設計（環境構築）における RPM のインストールを例にとると，OSS ツールでも環境設定として，ライブラリのインストールができるが，プロダクトのインストールが図 44 に示すとおり設定リストを参照しながらインストールをする等，通常の作業とは異なるため，OSS ツールでは対応ができない．このため，インストール作業での独自スクリプトツールとして，表 43 の項番 5 のインストールツールを作成して効率化を図った．

付録 5.4　適用結果

この手法を適用した具体的な実例と結果について示す．これらのツールの利用により，検証作業の効率化が図ることが可能となり，さらに人為的なミスも防ぐことができた．具体的には，従来の手作業では 1 週間かかっていた構築作業が，2 時間に短縮可能となった．また，検証実施と結果確認に 1 日かかっていた作業が，3 時間に短縮可能となった．

表 41. OSS ツール(例)

分類	ツール名	作業内容
設計(環境構築)	Puppet	・必要な RPM のインストール/アップデート ・設定の配布/登録 ・データ初期化 ・ファイルの一斉配布
実行(検証準備)	iperf	NW の帯域測定
	bonnie	Disk の I/O 測定
実行(状態確認)	MRTG	リソース情報可視化
	Nagios,Crane	マシン監視

表 42. スクリプト(例)

分類	スクリプトツール例
設計(環境構築)	・サーバリストの作成 ・ssh 疎通確認 ・CBoC の RPM のインストール/アップデート/アンインストール ・設定の配布/登録 ・データ初期化 ・ファイルの一斉配布
実行(検証準備)	・CBoC の起動/停止 ・CBoC へのデータ書き込み/読み出しのツールの起動/停止 ・指定プロセスの停止 ・リモートマシンへの JOB 登録/確認/削除 ・データのバックアップ/リストア
実行(状態確認)	・CBoC 稼働状態確認 ・CBoC へのデータ書き込み/読み出しのツールのプロセス存在確認 ・CBoC へのデータ書き込み/読み出しのツールのログチェック ・定期的な CBoC 統計情報取得/取得停止 ・定期的なサーバリソース情報取得/取得状況確認/取得停止 ・tcpdump 情報の取得開始/取得状況確認/取得停止(ローカル, リモート) ・複数リモートマシンにおける同一コマンドの実行 ・複数リモートマシンからの同名ファイル収集

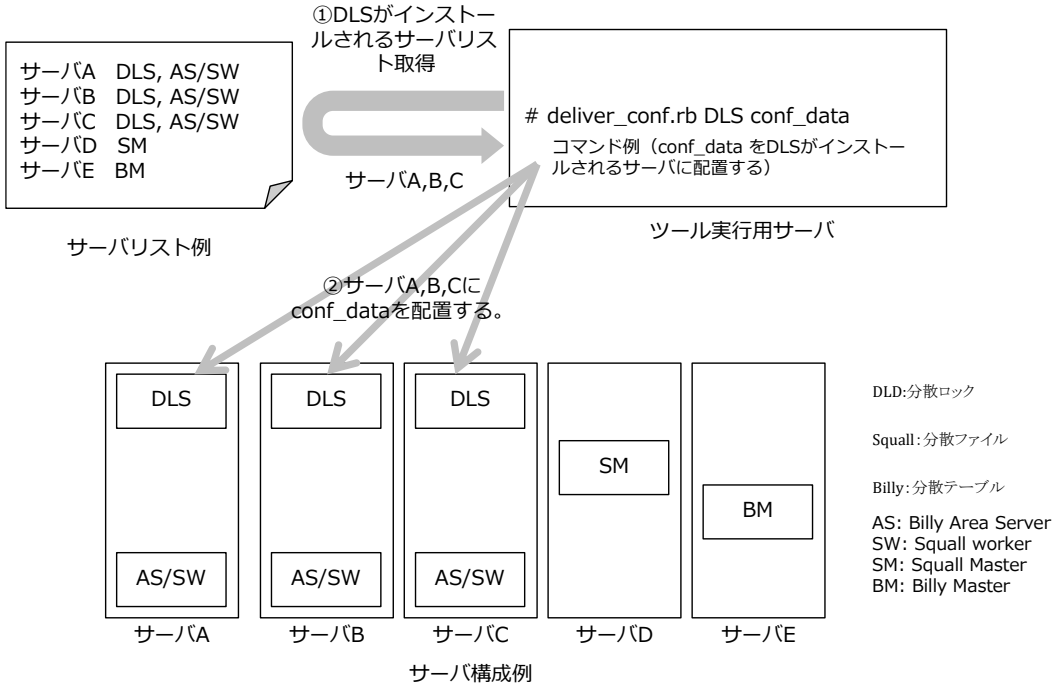


図 44. 環境構築ツールの動作イメージ

表 43. 環境構築時に用いた主な独自開発ツール

	分類	スクリプトツールファイル	機能概要
1	サーバリスト作成	generate_all_server_lists.sh	CBoC の環境構築および検証に必要な各種サーバリストファイルの生成
2	環境チェック	ssh_conn_check.rb	マシン間の ssh 接続可否確認
3	サーバ設定・配布	register_dls_cellname.sh	分散ロック セル名の /etc/hosts への登録
4		install_pf_client.sh	検証用クライアントツールの配付およびインストール
5		install_products.sh	CBoC 関連プロダクトの新規インストール
6		deliver_conf.rb	CBoC 関連プロダクトの設定ファイル配布
7		serve_files.rb	複数マシンへの同一ファイル群の配布
8	データ初期化	generate_squall_info.sh	分散ロック に登録する 分散ファイル の構成情報ファイルの生成
9		register_squall_info.sh	分散ファイルの構成ファイルの 分散ロックへの登録
10		initialize_cboc_data.sh	CBoC が保持する永続化情報の初期化
11	汎用	exec_ssh.sh	複数のリモートマシンにおける同一コマンドの実行

付録 6　NW, Disk 環境のばらつき回避

同一製造会社製のカタログスペックが同じスイッチやケーブルで構成されたネットワーク（NW）では，どの末端から測定してもスループットは同じとなるはずである．また，マシン性能についても同様である．しかし，大規模分散処理システムの「分散かつ大規模」な検証環境で数百台のマシンの NW 環境とディスク（Disk）を調べると，実際には性能にばらつきが発生していることが判明した．この NW や Disk の性能のばらつきは，1 台 1 台では小さくても，数百台を組み合わせると性能測定や機能確認に影響を与えてしまう．したがって，大規模分散処理システムのシステム検証では，NW と Disk 環境のばらつきをどのように確認するかが具体的な課題である．

従来の情報システムでは，複数マシンのみが NW で接続された環境での検証のため，1 台 1 台のばらつきの差異が小さい場合は，性能測定や機能確認に影響は与えない．本節では，大規模分散処理システムのシステム検証における，NW と Disk 環境のばらつきをどのように確認し解決するかについて説明する．

付録 6.1　NW, Disk 環境のばらつき回避に関する研究

NW（ネットワーク）や Disk（ディスク）の環境を確認するのは，通常の運用での監視[86]であり，環境が同じであることを確認していることはない．これは，従来の情報システムでは，NW 環境も Disk 環境も数は限られており，それぞれの環境の違いが機能や性能に影響されにくく，問題視されていないことが原因と考えられる．

付録 6.2　課題解決の考え方

環境の確認を行うことで，性能が均一でない部分を見つけ対処をする．環境確認のポイントは，マシン外側の NW 帯域とマシン内部の Disk I/O の 2 点であり，対処はパラメータチューニングである．実際に NW 帯域の確認において，複数のラックが論理的には同じ構成でも物理的には設定が異なっている場合がある．実際，図 45 に示すとおり中心的な NW 装置であるコアスイッチと末端のエッジスイッチ間で転送速度に約 1.5 倍の差が発生していた．また，Disk I/O の確認において，Disk ベンチマークツールの Bonnie++で測定した結果，図 47 に示すとおり同一機種でも性能の良いものと悪いものとでは，約 2 倍の性能差が発生していた．上記のとおり，環境は，想定と異なり均一でない場合があり，事前に環境を確認することが重要である．

この解決の考え方は，NW と Disk の性能確認というアプローチであり，検証環境が大規模で NW や Disk の性能のばらつきがある場合，大規模分散処理システム一般に有効である．

付録 6.3　具体的な解決手法

特に性能測定を実施する前に基本システム性能（Disk I/O スループット，NW 帯域）の測定を行い，マシン毎の性能状況や NW 帯域を確認して，必要に応じマシン交換や NW 設定の確認を行う．具体的な測定対象の環境と使用ツールを表 44 に示す．NW 帯域の確認手法としては，図 46 に

示すとおりコアスイッチとエッジスイッチ間の転送速度を OSS の iperf で一方のラックから他方のラックへデータを送信することにより測定する.

付録 6.4　適用結果

この手法を適用した具体的な実例と結果として, CBoC タイプ 2 の検証環境では環境の影響を考慮した測定結果の解析が可能となり, 解析の精度が向上したため再測定の必要がなくなった.

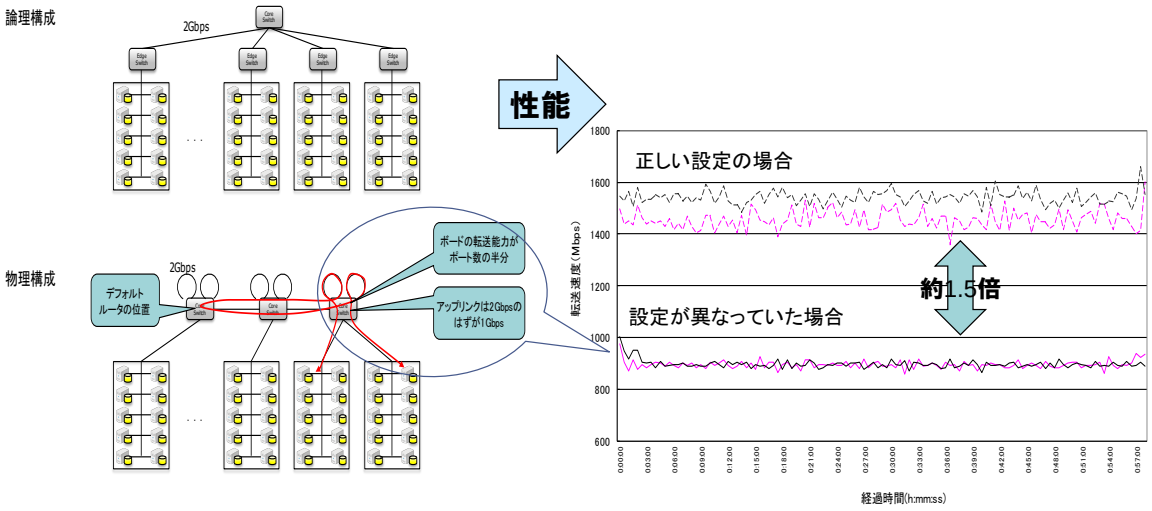


図 45. NW 帯域の違い

表 44. 測定対象の環境と使用ツール

	測定対象	使用ツール	備考
1	サーバ　ディスク I/O	bonnie++	集計例を図 47 に示す
2	サーバ　メモリ・ディスク I/O	hdparm	
3	ネットワーク帯域	iPerf	集計例を図 46 に示す

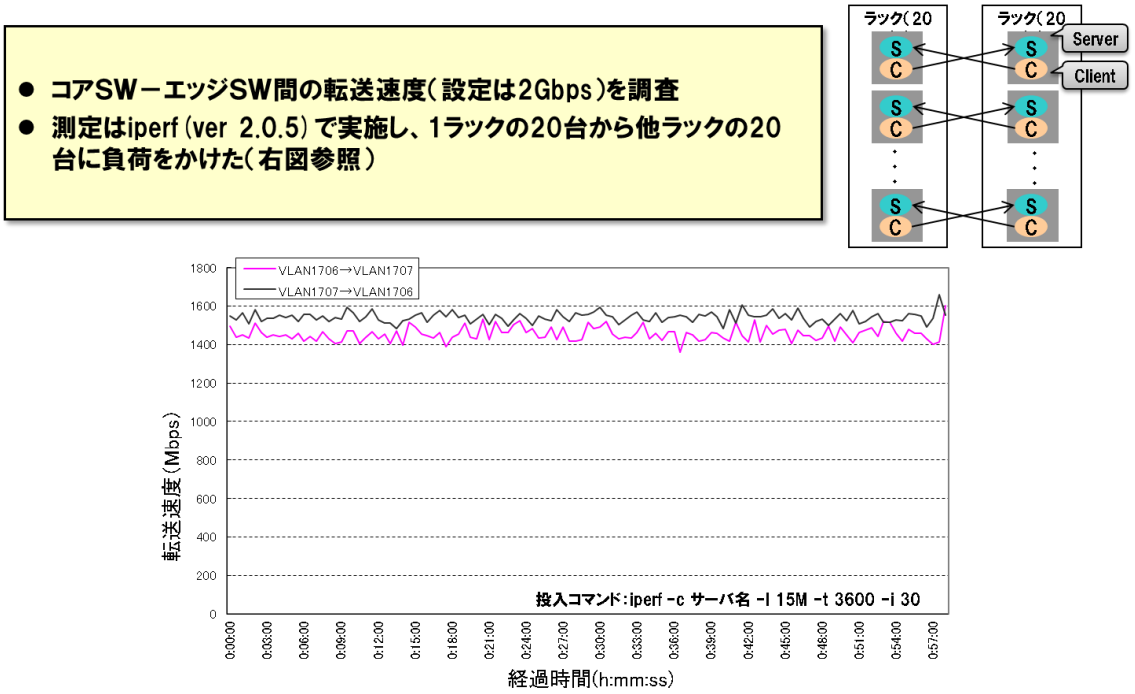


図 46. NW 帯域の確認手法

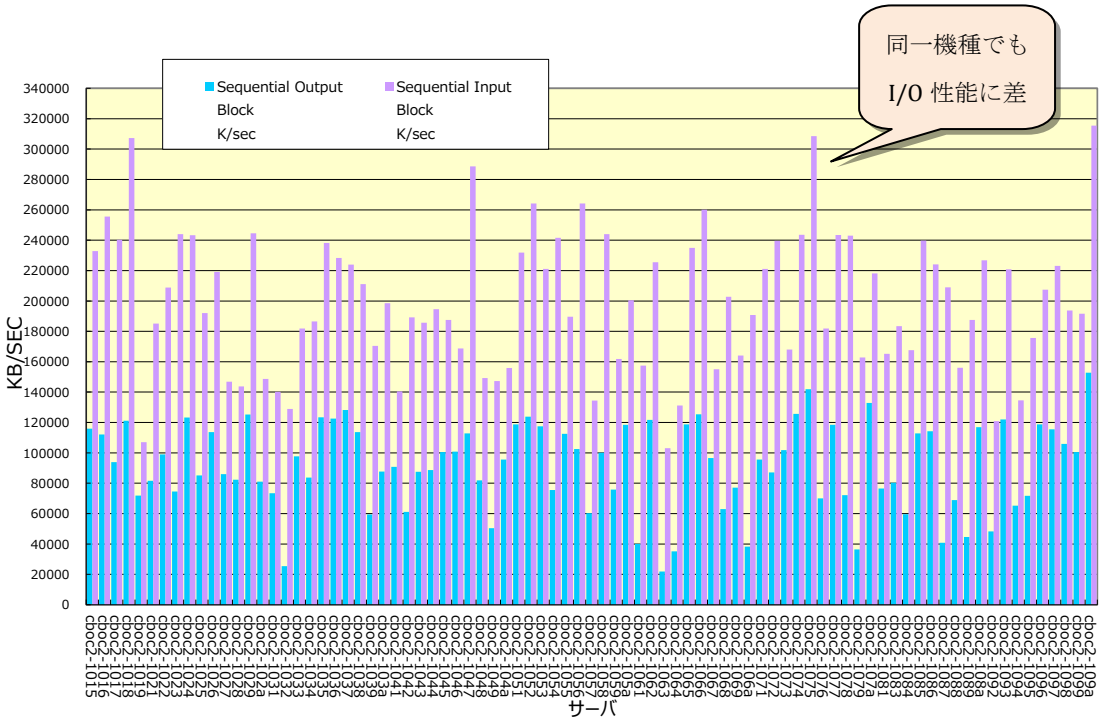


図 47. Bonnie++による測定結果

参考文献

[1] ビッグデータ競争元年,ハーバード・ビジネス・レビュー,No.293(2013/2),ダイヤモンド社.

[2] ビッグデータ競争時代,ハーバード・ビジネス・レビュー特集,2013 ダイヤモンド社

[3] 渡 佑也,欅 惇志,宮崎 純,中村 匡秀: RDBとKVSを相互に利用した多次元データに対する集約演算の効率化, DEIM Forum 2016 D5-5.

[4] 濱野賢一朗, Hadoop 徹底入門, Cloudera World Tokyo 2014.

[5] 平成 30 年度版 情報通信白書, 2019 総務省.

[6] インターネット検索エンジンの現状と市場規模等に関する調査, 2010 総務省.

[7] 西田 圭介: Google を支える技術～巨大システムの内側の世界,技術評論社(2008).

[8] Tom White, 玉川 竜司(訳):Hadoop 第 2 版, オライリー・ジャパン(2011).

[9] Apache Hadoop, http://hadoop.apache.org/.

[10] Lam C, Warren J: Hadoop in Action, Manning Publications(2010).

[11] Junichi Niino: 「NoSQL」は「Not Only SQL」である,と定着するか？, http://www.publickey1.jp/blog/09/nosqlnot_only_sql.html, 2009.

[12] 鷲坂 光一,中村 英児,高倉 健,吉田 悟,冨田 清次:大量データ分析のための大規模分散処理基盤の開発,NTT 技術ジャーナル,Vol.23,No.10,pp.22-25(2011).

[13] ISO/IEC: ISO/IEC 12207(Systems and software engineering − Software life cycle processes), ISO/IEC(1995).

[14] 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 V3.0, オーム社(2014).

[15] PMI: A Guide to Project Management Body of Knowledge, 6th ed., PMI(2018).

[16] Jing Han, Haihong E, Guan Le, Jian Du: Survey on NoSQL Database, IEEE, (2011).

[17] Cattell, Rick: High Performance Scalable Data Stores, Article of 2010-02-22, February 2010.

[18] 羽生貴史: データベース技術の今後の動向, UNISYS TECHNOLOGY REVIEW 第101 号, AUG. 2009.

[19] B.Cooper, A.Silberstein, E.Tam, R.Ramakrishnan, R.Sears: Benchmarking Cloud Serving Systems with YCSB, Proceedings of the 1st ACM symposium on Cloud computing, New York(2010).

[20] Running TPC-H queries on Hive, https://issues.apache.org/jira/browse/HIVE-600.

[21] Bo Li: Survey of Recent Research Progress and Issues in BigData, 2013.

[22] Jerry Gao, Xiaoying Bai, Wi-Tek Tsai: Cloud Testing - Issues, Challenges, Needs and Practice, SOFTWARE ENGINEERING, An International Journal, Vol1, No.1,(2011).

[23] 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, 情報処理学会 デジタルプラクティス Vol4,No.1(2013), pp51-59.

[24] 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).

[25] 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).

[26] Jerry Gao, Xiaoying Bai, Wei-Tek Tsai: Cloud Testing- Issues, Challenges, Needs and Practice, Software Engineering : An International Journal (SEIJ), Vol. 1, No. 1, Sep. 2011.pp9-23.

[27] 坂西 隆之, 小泉 仁志, 神林 亮, 佐藤 三久: プログラムテスト環境を提供するクラウドコンピューティングシステムの検討, 情報処理学会研究会報告, Vol.2009-OS-112 No.19, (2009).

[28] Haryadi S. Gunawi, Thanh Do, Pallavi Joshi,Peter Alvaro, Joseph M. Hellerstein, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Koushik Sen, and Dhruba Borthakur: FATE and DESTINI: A Framework for Cloud Recovery Testing, Proceedings of the 8th USENIX conference on Networked systems design and implementation (NSDI'11), 30 March-1 April 2011, Boston, MA, USA.

[29] Sergiy VILKOMIR: CLOUD TESTING: STATE-OF-THE ART REVIEW, INFORMATION & SECURITY. An International Journal, Vol.28, No.2, 2012, pp213-222.

[30] Lian Yu, Wei-Tek Tsai, Xiangji Chen, Linqing Liu, Yan Zhao, Liangjie Tang, and WeiZhao: Testing as a Service over Cloud, Proceedings of the Fifth IEEE International Sym-posium on Service Oriented System Engineering (SOSE), 4-5 June 2010, Nanjing, China, pp181-188; G. Candea, S. Bucur, and C. Zamfir: Automated software testing as a service: Proceedings of the 1st Symposium on Cloud Computing (SOCC'10), June 10-11, 2010, In-dianapolis, Indiana, USA; Neil Stinchcombe: Cloud computing in the spotlight,Infosecu-rity6:6 (September-October 2009): pp30-33.

[31] Leah Muthoni Riungu, Ossi Taipale, Kari Smolander: Research Issues for Software Testing in the Cloud,Proceedings of the IEEE Second International Conference on Cloud Com-puting Technology and Science, Indianapolis, Indiana USA, 30 Nov. - 3 Dec. 2010, 557-564.

[32] 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).

[33] 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).

[34] 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.

[35] 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.
[36] 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.
[37] 山根智: 時間μ-calculus による分散システムの形式的検証, 電子情報処理学会信学技報, FTS96-47(1996-10), pp49-56.
[38] Kavi K.M.: Real-time Systems, Abstratction, Languages and Design Methodologies, pp660, IEEE Computer Society(1992).
[39] Bryant R.E.:Graph-Based Algorithms for Boolean Function Manipulation, IEEE Trans. On Computers, Vol.C-35, No.8, pp677-691(1986).
[40] McMillan K.K.: Symbolic Model Checking, Kluwer,pp.194(1993).
[41] Cem Kaner, Jack Falk, Hung Quoc Nguyen:Testing Computer Software 2nd edition, John Wiley & Sons, Inc(1999).
[42] Glenford J. Myers: ソフトウェア・テストの技法 第 2 版, 近代科学社(2012).
[43] 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).
[44] 岡崎毅久: ソフトウェア品質特性に基づいたシステムテスト設計, ソフトウェア工学, pp25-30, 1996.
[45] 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).
[46] 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).
[47] Y. Falcone, et al.: A Tutorial on Runtime Verification, Engineering Dependable Software Systems, 34, IOS Press, pp.141-175(2013).
[48] 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).
[49] 伊藤忠テクノソリューションズ: 意思決定の精度向上に効く Hadoop 導入の成功に向けた意外なカギとは？, http://itpro.nikkeibp.co.jp/atclact/activesp/14b/090200017/, 日系 BP ITpro(2014).
[50] 倉持健史: 進化を遂げた NetApp ビッグデータソリューション for MapR, http://www.netapp.com/jp/communities/tech-ontap/tot-201409-mapR-jp.aspx, NetApp Tech OnTap(2014).
[51] 日立ソリューションズ: Hadoop LZO 圧縮機能の検証, http://www.hitachi-solutions.co.jp/hadoop/pdf/hadooptn.pdf, 日立ソリューションズ ホワイトペーパ(2012).
[52] H. Kamezawa and M. Nakamura and J. Tamatsukuri and N. Aoshima and M. Inaba and K. Hiraki and J. Shitami and A. Jinzaki and R. Kurusu and M. Sakamoto and Y. Ikuta, : Inter-layer coordinationfor parallel TCP streams on Long Fatpipe Networks, in Proceedings of

the IEEE/ACMSupercomputing(SC2004) (2004).

[53] Niels Provos, Chuck Lever, and Stephen Tweedie, :Analyzing the Overhead Behavior of a Simple Web Server, In 4th Annual Linux Showcase & Conference, October 2000.

[54] G.A.S.Whittle, J.F. Paris, A.Amer, D.D.E. Long and R Burns, : Using Multiple Predictors to Improve the Accuracy of File Access Predictions, in Proc. Of the 20th IEE/11th NASA Goddard Conf. on Mass Storage Systems and Techologies (MSS), pp.230-240, San Diego, US-CA, Apr. 2003.

[55] J. Griffioen and R. Appleton, : Reducing File System Latency using a Predictive Approach, In Proc. Of the USENIX Summer 1994 Tech. Conf., Boston, US-MA, Jun. 1994.

[56] Becerra, Y. Beltran, V. Carrera, D. Gonzalez, M. Torres, J. and Ayguade, E, : Sppeding Up Distributed MapReduce Applications Using Hardware Accelerators, Proc. 2009 International Conference on Parallel Processing(ICPP '09), IEEE Computer Society, pp.42-49, (2009)..

[57] 白木原敏雄, 他, : 高可用性の新技術(無停止分散システム構築ミドルウェア). 東芝レビュー. Vol.52, No.8, p.40 － 42(1997).

[58] E.M. Clarke and J. M. Wing, : Formal methods: State of the art and future directions, ACM Computing Surveys, 28(4), 626-643(1996).

[59] Tabachnick, B. G., & Fidell, : L. S. Using multivariate statistics (5th internationaled.). Boston, MA: Pearson/Allyn & Bacon(2006).

[60] Weinfurt, K. P.: Multivariate analysis of variance. In L. G. Grimm & P. R. Yarnold (Eds.), Reading and understanding multivariate statistics (pp. 245-276),Washington, DC: American Psychological Association (1995).

[61] Barlow, R. and Proschan, F.; Mathematical Theory of Reliability, 84, John Wiley & Sons, Inc. (1965)

[62] M. Klems, A. Silberstein, J. Chen, M. Mortazavi, S. A. Albert, P. P. S. Narayan, et al.,: The Yahoo!: cloud datastore load balancer, in Proc. 2012 Proceedings of the fourth international workshop on Cloud data management, pp.33-40.

[63] I. Konstantinou, D. Tsoumakos, I. Mytilinis, N. Koziris: Dbalancer: distributed load balancing fornosql data-stores. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13, pp.1037-1040, 2013.

[64] 戸田航史, 角田雅照, 門田暁人, 松本健一: 工数見積もりモデルで予測できないソフトウェアプロジェクトの特徴分析, 信学技報, 電子情報通信学会, pp67-72, 2005.

[65] 戸田航史: 工数見積もりモデルの予測精度を低下させるプロジェクトの特徴分析, 情報処理学会 ウィンターワークショップ 2006, pp71-72, 2006.

[66] K. Srinivasan, and D. Fisher, "Machine Learning: Approaches to Estimating Software Development Effort," IEEE Trans. on Software Eng., Vol.21, No.2, pp.126-137, 1995.

[67] M. Shepperd, and C. Schofield: Estimating Software Project Effort Using Analogies, IEEE

Trans. on Software Eng., Vol.23, No.12, pp.736-743, 1997.
[68] 川勝崇史， 棟朝雅晴： 分散クラウド環境における SLA を考慮した WEB システムの多目的資源割当最適化， 情報処理学会研究会報告, Vol.2013-MPS-96 No.9(2013).
[69] Francois Legillon, Noudine Melab, Didier Renard, El-Ghazali Talbi: Cost Minimization of Service Deployment in a Multi-Cloud environment, IEEE Congress on Evolutionary Computation, 2013.
[70] 菅野文友(監修): ソフトウェア品質管理事例集，日科技連出版社(1990).
[71] 菅野文友: ソフトウェア・エンジニアリング，日科技連出版社(1979).
[72] 三嵜武: ソフトウェアの品質評価法，日科技連出版社(1981).
[73] 鍋島一郎: スケジューリング理論，森北出版(1974).
[74] Hatono, I. et al: Distributed real-time scheduling for flexible manufacturing, transactions of the Society of Instrument and Control Engineers, Vol.31 No.1(1995)pp.108-115.
[75] 江口透， 大場史憲， 小崎慎太郎， 村山長： 定期的最適化とリアルタイムタイムスケジューリングの融合による処理時間の不確実性を考慮した動的スケジューリング， 日本機械学会論文集 72 巻 717 号(2006-5), 2006.
[76] 大向一輝， 武田英明， 三木光範： 多様かつ曖昧な個人タスクのための管理システムの提案と実装， エージェント合同シンポジウム (JAWS2002) 講演論文集, 2002.
[77] 廣池敦: 類似画像検索システム「EnraEnra」，人工知能学会, 29 巻 5 号, pp430-438, 2014.
[78] M. Yamamuro, K. Kushima, H. Kimoto, H. Akama, S. Konya, J. Nakagawa, K. Mii, N. Taniguchi, K. Curtis, ″ExSight - Multimedia Information Retrieval System″, Proc. of PTC'98,USA(1998).
[79] A. Ghias, J. Logan, D. Chamberlin, B. C. Smith, ″Query By Humming″, ACM Multimedia '95, pp.231-236,USA(1995).
[80] T. Kageyama, Y. Takashima, ″A Melody Retrieval Method with Hummed Melody″, Journal of The Institute of Electronics Information and Communication Engineers, pp.1543-1551,(1994).
[81] K. Curtis, N. Taniguchi, J. Nakagawa, M. Yamamuro, ″A Comprehensive Image Similarity Retrieval System that utilizes Multiple Feature Vectore in High Dimensional Space″, International Conference on Information, Communications and Signal Processing, ICIS'97, pp.180-184,(1997).
[82] 出原博， 藤本典幸， 竹野浩， 萩原兼一： WWW 画像検索における画像周辺の HTML 構文構造を考慮した画像説明文の抽出手法， 信学技報， 電子情報通信学会， pp19-24, 2005.
[83] M. Umeda, K. Kushima, M. Yamamuro, Y. Nishihara, N. Taniguchi, S. Kon'ya, "Study on the Characterization of Music and A Melody Retrieval Method using Hummed Melody", Proc. of PTC'99, USA(1999), pp1052-1056.
[84] 田澤孝之： [第 7 回] Hadoop の利用 準備編，

http://itpro.nikkeibp.co.jp/article/COLUMN/20120730/412602/, 2012.
[85] 菅原亮, 岡本隆史: Puppet のインストールと動作確認、トラブル対処法, http://www.atmarkit.co.jp/ait/articles/1410/08/news016.html, 2014.
[86] 寺島広大: オープンソース統合監視ツール導入指南, http://thinkit.co.jp/free/article/0706/21/1/, Think IT, 2007.
[87] 日本工業標準調査会: JIS X 0129-1 ソフトウェア製品の品質, 日本規格協会(2003).
[88] 大田尾一作: HTML5 時代のツール「Selenium2」で Web システムのテストを自動化, http://codezine.jp/article/detail/7427, CodeZine 2013.
[89] Savanna Project: https://savanna.readthedocs.org/en/0.3/, OpenStack Foundation 2013.
[90] 羽生貴史: データベース技術の今後の動向, UNISYS TECHNOLOGY REVIEW 第101 号, AUG. 2009, pp99-108.
[91] Dean et al., “MapReduce : Simplified Data Processing on Large Clusters”, OSDI 2004, CACM Jan 2008, pp107-113.
[92] IPA 独立行政法人社会基盤センター: ソフトウェア開発データ白書 2018-2019, IPA 独立行政法人(2018).

謝辞

本論文をまとめるにあたって多くの方にご助言をいただきました．ここに各位のお名前を記して感謝申し上げます．

まず，社会知能情報学専攻 植野真臣 教授， 大須賀昭彦 教授， 田中健次 教授， 田原康之 准教授， 情報メディアシステム学専攻 田野俊一 教授， 情報システム基盤学専攻 南泰浩 教授， 元社会知能情報学専攻（現 慶応義塾大学）栗原聡 教授には報告会や研究審査において貴重なコメントをいただきました．特に植野真臣 教授には，本論文をまとめるにあたり，的確なご助言とご鞭撻を頂き，論文の完成に至ることができました．深く感謝いたします．

また，元社会知能情報学専攻（現 大阪大学）鬼塚真 教授には，当該分野の調査から，研究の計画，研究内容に関する議論，結果のまとめ，プレゼンテーションについてご指導をいただきました．また，元社会知能情報学専攻 太田敏澄 教授には，ゼミでの発表の指導や議論，論文作成に関して，ご意見とご指導をいただきました．さらに，本研究についての助言や協力をいただいた鬼塚研究室と太田研究室のメンバーの皆様に感謝いたします．

本研究は，NTT ソフトウェアイノベーションセンタで行った研究を基としており，研究を実施していた当時に，ご指導とご協力をいただきました，吉田悟氏，中村英児氏，本庄利守氏，坂井俊之氏に感謝いたします．

関連論文の印刷公表の方法および時期

1. 梅田 昌義, 鬼塚 真
   論文題目「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」
   平成 28 年7月, 情報処理学会デジタルプラクティス,Vol7,No.3(2016) ,pp330-339.
   (第 3 章の内容に関連)

2. 坂井 俊之, 梅田 昌義, 中村 英児, 本庄 利守
   論文題目「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」
   平成 25 年 1 月, 情報処理学会デジタルプラクティス,Vol4,No.1(2013) ,pp51-59.
   (第 4 章の内容に関連)

3. 梅田 昌義
   論文題目「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」
   平成 29 年 10 月, 情報処理学会デジタルプラクティス,Vol8,No.4(2017),pp361-368.
   (第 5 章の内容に関連)

4. M. Umeda, K. Kushima, M. Yamamuro, Y. Nishihara, N. Taniguchi, S. Kon'ya
   論文題目「Study on the Characterization of Music and A Melody Retrieval Method using Hummed Melody」
   1999 年1月, PTC'99,USA(1999) ,pp1051-1056.
   (第 2 章の内容に関連)

参考論文等一覧

研究会等(査読なし)

1. 梅田 昌義, 吉田 悟: 大規模分散処理システムのシステム検証におけるテスト環境の開発アクティビティの実践, 平成27年3月, 情報処理学会 DEIM Forum 2015 B6-3.

2. 梅田 昌義: 大規模分散処理システムの検証の効率化に関する研究, 情報処理学会ソフトウェアエンジニアリングシンポジウム 2019 論文集, pp216-223 2019-08-22.

特許

1. 梅田昌義, 西原祐一, 紺谷精一: 音楽情報検索装置, 音楽情報蓄積装置, 音楽情報検索方法, 音楽情報蓄積方法およびそれらのプログラムを記録した記録媒体, 特許第 3434223 号.