一般
投稿論文

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

梅田 昌義[†1]　鬼塚 真[†2]

[†1]日本電信電話（株）　[†2]大阪大学

近年，ビッグデータの管理と処理を行うために，スケールアウトの容易な大規模分散処理システムの必要性が増してきている．本稿では，大規模分散処理システムの検証における3つの課題である「検証データの安定した高速登録」，「検証項目の実施時間の見積もり精度向上」，「効率的な検証マシンの割り当て」に対して，「ネットワークとディスクのI/Oのチューニング手法」，「大量にリソースを使用する項目の見積もり手法」，「数段階の検証環境の割り当て手法」の解決方法を提案する．これらの解決方法をシステムに適用することにより，事前のデータ蓄積時間を短縮し，システム検証の遅延リスクを減少させて，システム検証をスケジュールどおりに完了したプラクティスについて述べる．

1. はじめに

近年，大規模データを処理するマシンの性能向上や自律分散処理の発達によって，大規模分散処理システムによるビッグデータの活用が期待されている．現在，このシステムを実現する代表的なソフトウェアは，Apacheプロジェクトの Hadoop[1]である．筆者らは，Hadoopと同様に Key-Value Store[2]による大量の非構造化データを管理する CBoC タイプ2（Common IT Base over Cloud Computing タイプ2）システム[3]を開発している．このシステムは，「分散ロック」，「分散ファイル」，「分散テーブル」のコンポーネントから構成される．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．そのデータの読み込みと書き込みができる API を利用した AP（Application）により，携帯検索サービスを提供する．検証の対象は，このサービスが動作するシステムである．

一般に商用ソフトウェア開発の場合，運用前のサービスの最終確認としてシステム全体の検証（以下，システム検証と呼ぶ）を行う．大規模分散処理システムの場合，手順の複雑さやログによる問題解析の困難さがあるため，あらかじめ計画したシステム検証のスケジュールが遅延することがある．特に新システムの場合，システム検証の事例がないために解決方法が見つからず，検証作業が止まってしまうこともあり，検証により多くの時間を要してしまう．

本稿では，商用の大規模分散処理システムにおけるシステム検証における3つの課題である「検証データの安定した高速登録」，「検証項目の実施時間の見積もり精度向上」，「効率的な検証マシンの割り当て」に対して，「ネットワークとディスクのI/Oのチューニング」，「大量にリソースを使用する項目の見積もり」，「数段階の検証環境の割り当て」の解決方法を提案する．また提案方法の適用により，スケジュールどおりに完了したことについて述べる．第2章では，大規模分散処理システムの関連研究を紹介する．第3章では，大規模分散処理システムのシステム検証の課題について述べる．第4，5，6章では，検証の3つの課題解決の提案と実施した結果についてそれぞれ述べる．第7章では，まとめと今後の課題について述べる．

2. 関連研究

この章では，大規模分散処理システムのシステム検証の効率化に関する研究について，「大規模分散処理システム」，「分散処理システム」，「一般的な情報システム」の3つに分類して述べる．(a)大規模分散処理システム，(b)分散処理システム，(c)一般的な情報システムでは，発生した問題の解決方法や検証の効率化についての報告はされていない．

(a) 大規模分散処理システムの検証

このシステムでは，数百台規模のマシンをリソース

として検証する必要がある．しかし，このシステムのアーキテクチャは新しいことから，システム固有の問題を考慮した検証の効率化に関する研究は，ほとんど報告されていない．従来の大規模分散処理システムの論文[4],[5],[6]は，システム検証の性能と項目作成の問題，あるいはクラウドとレガシーシステムの検証を比較してクラウドの有効性を報告するにとどまっている．また，Google[7]やMicrosoft[8]のシステム検証は，ログの出力や，APIを利用したトラッキングや，モニタリングによる動作や，ボトルネックの確認方法が報告されているだけである．これらの報告は，大規模分散処理システムの検証として参考になる．しかし，システム固有の確認方法であるため，大規模分散処理システムには，一般的に適用できない．また，これらの論文では，発生した問題の解決方法や検証の効率化について報告されていない．

(b) 分散処理システムの検証

分散処理システムの検証の論文[9], [10]では，分散環境で問題が発生することを未然に防ぐ方法が提案されている．しかし，問題が発生した場合の解決方法や検証の効率化について報告されていない．

(c) 一般的な情報システムの検証

マシン数台により構成されるシステムを，一般的な情報システムと呼ぶと，このシステムのソフトウェア開発の検証の論文[11], [12], [13], [14]が報告されている．しかし，これらの論文では，検証の課題や発生した問題の解決方法や検証の効率化に関する記述はない．そのため，今回の大規模分散処理システムに特有な，データ量の多さから発生する問題や，複数コンポーネントから構成されている検証項目の多さの問題を解決することができない．

3. 大規模分散処理システムのシステム検証の課題

システム検証は，開発工程のV字モデルにおける最後の工程として，要求仕様を満たしていることを確認することが目的である．本稿では，検証工程をIEEEソフトウェアエンジニアリングのSWEBOK[15]にある，ソフトウェアテスティングに規定されている工程に基づいて，図1のように前工程の計画から後工程の欠陥追跡までアクティビティとして分類する．そのうち計画アクティビティは，プロジェクトの計画を立案して，スコープ，期間，工数および費用のリソースを決定するため，工程の全体にかかわる重要な工程である[16]．

計画アクティビティにおいて，過去に表1に挙げた問題が発生してスケジュールが1カ月遅延した．これらの問題を解決するには，期間の短縮や作業の効率化が必要になった．この章では，計画アクティビティのうち，これらの問題が発生する理由，解決に向けての課題および対応について述べる．

3.1 課題について

計画アクティビティの課題を検討するにあたり，表1のようにその作業を「準備」,「見積もり」,「実施計画」の3つに分類する．

(a) 準備：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課題があり，高速にデータを登録することにより，その日数を短縮する必要がある．

(b) 見積もり：実施時間の長い検証項目は見積もりよりも多くの時間を要するという課題があり，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．

(c) 実施計画：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課題があり，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，その時間を短縮する必要がある．

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

図1 SWEBOKのアクティビティ

表1 発生した問題の概要と具体的課題

作業分類	発生した問題の概要	具体的課題（具体的内容）
準備	検証データが100テラバイト級のため，事前の検証データ蓄積が長期化し，検証開始が遅延	検証データの安定した高速登録（事前に実施するデータ蓄積の日数短縮）
見積もり	検証項目の実施時間の見積もり精度が悪く，計画よりスケジュールが遅延	検証項目の実施時間の正確な見積もり（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	問題の解析に時間を要するため，その間はシステム検証が止まり，スケジュールが遅延	効率的な検証マシンの割り当て（発生問題に対する解析と対処の時間短縮）

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，汎用性がある方法で見積もりの確度が高い条件でスケジュールを計画することができ，システム検証を期間内に完了することが可能になる．本稿では，期間とリソースに制約条件がある商用開発において，システム検証を所定の期間内に完了するための対応方法と適用事例を，3つの課題それぞれについて説明する．

4. ネットワークとディスクの I/O のチューニング手法

表2は，一般的な情報システムと大規模分散処理システムでのデータ登録の違いを示したものである．大規模分散処理システムのデータの場合，データ量は100テラバイト級になり，これだけ大規模なデータを事前に蓄積するには約1カ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短縮することが必要になる．

4.1　課題解決の考え方

データ蓄積の処理時間を短縮する方法として，「検証データの高速登録」と「データをコピーして増やす」という2つの案がある．

商用システムでは，ユースケースを使って，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保証されない．また，データ分析処理アプリケーションとの同時処理があるため，OSレイヤの設定値変更がそのほかのアプリケーションに影響することを考慮して，Linuxのproc/sysディレクトリ配下のOSレイヤの設定をデフォルトにしている．しかしその一方，読み出し処理やアプリケーションの実行は事前のデータ登録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処理の投入が可能である．

もう1つの方法として，データをコピーして増やし，処理時間を短縮する案がある．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やメンテナンスの工数が2倍以上かかるために除外した．

(a) 従来手法の課題調査

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2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時頃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．一方で，商用運用時のシステムでは事前登録のような大量の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．

(b) 課題解決へのアプローチ

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，コンポーネン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チューニングの方法が優位である．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ように設定する．ここでは，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．この方法は，ネットワークとディスクI/Oのボトルネックがあるほかのシステムにも適用可能である．これらの方法を適用して，性能要件を超えた書き込みスループットと安定性を実現する．

表2　データ蓄積の違いと問題

	データ蓄積	データ登録の問題
一般的な情報システム	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条件で蓄積が可能．登録が短時間なため，ほか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100テラバイト級の大規模データを扱うため，データ登録は検証期間と同等の1カ月程度を要することがある	後述するCompaction処理が自律的に動作するため，特に高負荷をかけた場合にスループットが低下して安定しないことがある

4.2　具体的な解決方法

(a) 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

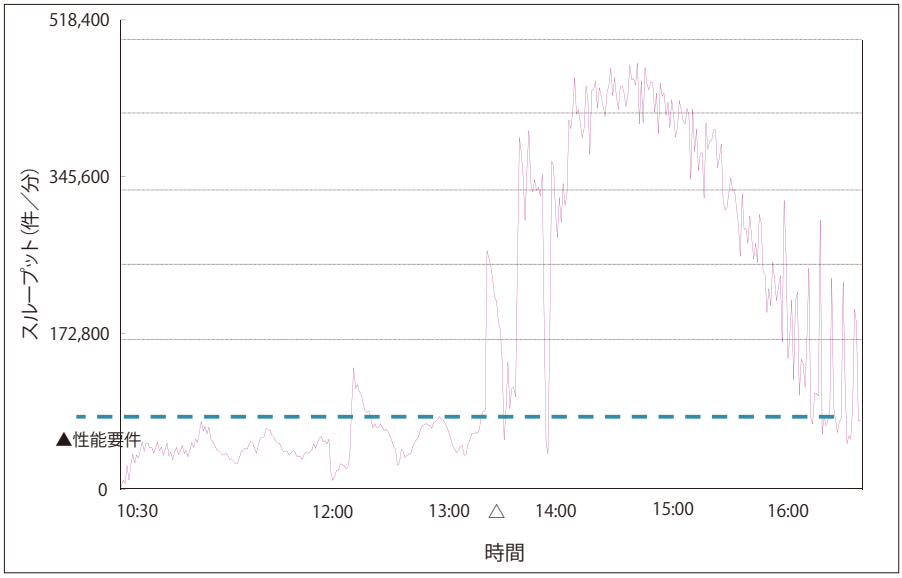


図2　データ投入時の書き込みスループット低下

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)については，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バッファであり，(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バッファのパラメータチューニングでは，1.5倍の性能を確保したが，コネクションプールについては，効果を確認することができない結果となった．それぞれについて以下に述べる．

- TCPバッファのパラメータは，ボトルネックとなる上限の値が分かっていないものは最大値として設定，分かっているものはその値を設定する．この方法で性能要件以上の書き込み速度を行ったところ，エラーが発生しないことを確認した．まず，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87380バイトから129024バイトの約1.5倍に変更した．このパラメータは，1ソケットあたりのメモリの使用量で，システムに負荷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によって自動に調整される．最大2Gbpsのネットワーク帯域でボトルネックとならない1.5倍，2倍と試した後，デフォルトの1.5倍に設定した．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できるようにした．そして，性能要件の1.5倍，2倍，3倍と試し，システムの余力はあるが限界値に近い書き込み速度を求めた．このように段階的に調整する方法によって，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った．
- コネクションプールは，アプリケーションレイヤにおけるコンポーネントの設定で可能であるが，適正な値が不明であった．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，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．

(b) ディスクI/Oチューニング方式

ディスクI/Oのチューニングが可能な，アプリケーションとOS間のポイントを調査した．そのポイントとは，マシンのハードウェアのディスクI/Oと，コンポーネン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では，I/Oスケジューラやプロセススワップの抑制などパラメータのチューニングの効果が得られなかったが，コンポーネントが制御しているディスクI/Oについては，安定した性能が得られる結果となった．それぞれについて以下に記述する．

- 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について，ハードディスクの設定を検証した．RedHat5では，パラメータのチューニングの効果はなかった．
- コンポーネン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，コンポーネン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，性能要件以上のデータを

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．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分かった．このことから，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．このCompactionは，サーバのメモリ上に記録されたデータを対象としており，データサイズが一定値に達したときのMinor Compactionと，このMinor Compactionによって作成されたデータの数が一定数に達したときに複数のデータを1つのデータに統合するMerge Compactionの2つがある．1回に処理されるデータ量の多さから，Merge Compactionが処理性能への影響の支配項であることが分かった．そのMerge Compactionは，複数のマシンで同時に発生すると処理負荷が高くなり，CPUで処理しきれず，クライアントの処理要求を受け付けることができなくなることがある．大規模データが投入されている場合には，この発生頻度はさらに高くなる．そのため，システムのMerge Compactionの実行の設定に乱数による設定を加え，ランダムな実施に変更した．このことで複数のマシンにおいてMerge Compactionが同時発生するのを回避している．

TCPバッファ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した．また図3のように，コンポーネントのディスク書き込み制御の設定変更によって，書き込みスループットを図2と比べて安定させた．

4.3　適用結果

CBoC タイプ2への適用結果を以下に示す．計画上のデータ登録は，1カ月で行うことを求められていた．しかし，データ量はバックアップデータも含めて30億Webページ規模であり，性能要件で定められた書き込み性能のままでは，計算上は図4の上のように6週間かかる．これに対し，4.2の提案方法に従い，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た．そして，図4の下に示すように，約1カ月でデータ蓄積を完了した．

5. 大量にリソースを使用する項目の見積もり手法

一般的な情報システムと異なり，大規模分散処理システムのシステム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．そのため，実施時間の長い検証項目の見積もりが正確でないと，誤差が数時間や日単位になり，スケジュールを圧迫することになる．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課題が残る．したがって，検証項目の事前検証による見積もり作業時間の短縮に加え，見積もり精度を向上させることが必要になる．

5.1　課題解決の考え方

大規模分散処理システムでは，参考にできる類似システムが限られている．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

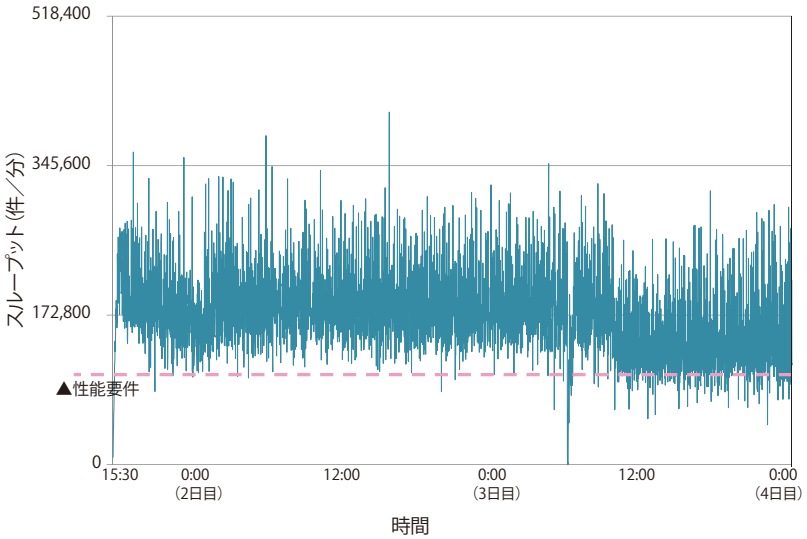
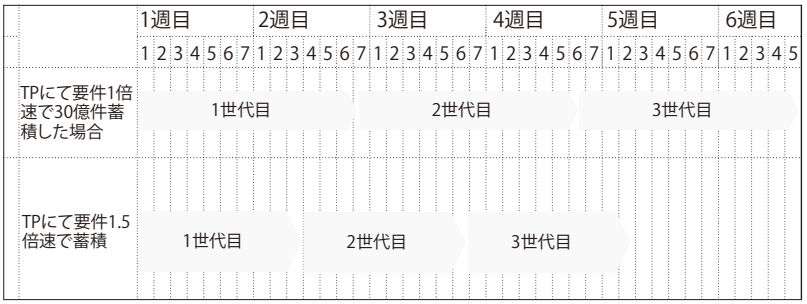


図3　データ投入例（書き込みスループット一定）



注：世代はデータの版数を意味する．

図4　事前データ登録の具体的スケジュール

時間を厳密に見積もることが難しい．また，データ量も多いため，データのリカバリのようなデータ量に依存した検証項目は検証時間が長くなる．こうしたことから，一般的な情報システムの検証の結果をもとに見積もりを行い，大規模分散処理システムの検証項目の実施時間の精度を上げることは困難である．

そこで，少ないテスト回数でより広い範囲をカバーするための手法として広く利用されているのが，同値分割と境界値分析の方法[9]である．この方法は，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ような長時間の項目には，同様の項目が少なく適用できない．適用可能な項目は実施時間が1時間以内のものであるため，これらの項目は全体の実施時間の支配項とはならないため，スケジュールの精度の向上にはつながらない．検証の効率化について説明した文献[6], [7], [8], [9]はあるものの，事前実施や類似検証からの類推の見積もりに関しての記述や，どの観点で見積もりを行うかに関しての記述はない．

一方，類似システムを参考にする方法で見積もりを実施することも可能である．この方法では，1回の見積もりで見積もりの精度は上がらないため，見積もりの精度を上げるには数度の実施が必要になる．1回の検証時間が一般的な情報システムに比べ，検証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．

(a) 従来手法の課題調査

過去の事例では図5に示すように，システム検証を実施した結果，ほぼすべての検証項目で，当初の見積もり時間を超えていた．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．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週間で完了するように計画した．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3のように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．別OSの検証の問題とVM検証の問題は，OSに起因する問題であり，問題の切り分けや解析と対策に時間を要したためであった．

(b) 課題解決へのアプローチ

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，項目全体の約1／3程度を占めており，スケジュール全体に影響を与える．このうち代表的な検証項目について，スケジュール立案時に事前に検証を実施する．その際，代表的な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3，4項目程度の見積もりができる．中項目レベルに着目する理由は，この粒度の分類で，検証時間にかか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．この解決方法を採用した場合，見積もりの作業はスケジュール立案時に2，3日程度で実施可能である．

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方法によって，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．この方法は，大量のデータを処理する場合，ほかのシステムにも適用可能である．

5.2 具体的な解決方法

解決方法は，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，ユースケースを確認する正常系と，ユースケースとは異なるが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．事前検証

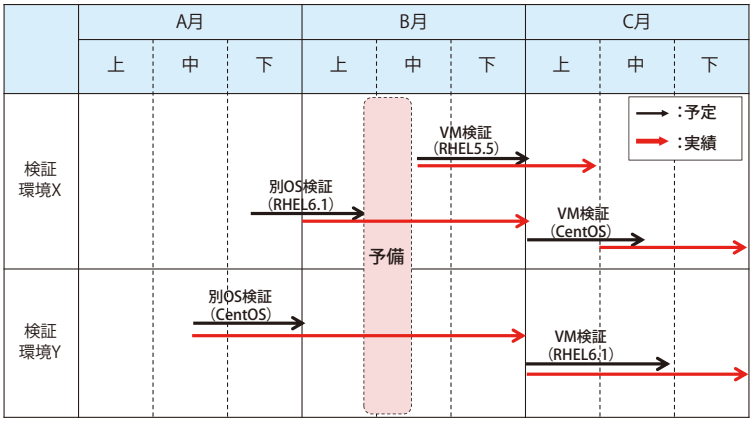


図5　実例の計画と実績（事前検証未実施）

表3　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 が発生するため遅くなる．
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 が発生し，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．

の方法を以下の手順で実施する．

(1) 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) 検証観点が同じであるため，検証手順にも同じ部分が多くなる．このことから，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，それぞれの中項目レベルから，手順が最も多い1項目を選定する．

(3) 選定した代表的な数項目について，所要時間を事前に確認する．このことにより，ほかの同じ手順の項目時間を見積もることができ，実施時間の把握が可能となる．

それ以外の項目は，一般的な情報システムの検証の実施時間から類推[15]する．この理由は，今回事前検証をする項目と比較して見積もり誤差が小さく，検証全体のスケジュールに与える影響が少ないためである．見積もりの結果が予定した検証期間を超えた場合は，一般的な情報システムと同様，計画を再度立案する．

5.3　適用結果

CBoC タイプ2への適用結果について以下に示す（図5の検証とは別の検証）．この方法を適用し，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込みの処理で時間がかかる性能測定の項目に関して，それぞれ1項目（中項目）を事前検証した．検証時間の見積もり結果を表4に示す．この場合，約1週間の事前検証で実施スケジュールを完了することができた．実際にシステム検証をした結果を図6に示す．（後述する数段階の検証環境の割り当て手法の検証の一部）従来は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，事前検証の方法によって検証項目の実施時間の見積もり精度が向上し，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．さらに，表3に示すとおり過去の事例では，システム検証中は問題摘出もあることから，事前検証の方法を適用することで問題を事前摘出できる効果もあった．このように，システム検証において時間を要する検証項目による遅延のリスクは減り，計画したスケジュールでの完了が可能となった．

6.　数段階の検証環境の割り当て手法

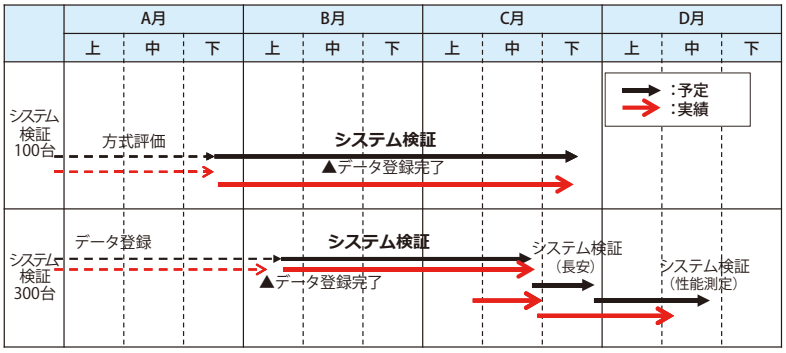
大規模分散処理システムを検証する場合，品質確保の観点から，すべての検証項目をサービスイン時の大規模なマシン台数の環境で検証する方法がある．しかし，検証対象となる処理が複数コンポーネントの連携で実現されているため，検証で問題が発生した場合にはコンポーネントを連携させたまま原因を解析する必要がある．この場合，その解析に要する時間は一般的な情報システムよりも長くなる傾向があるほか，異常系の検証項目は，問題を対処した後の確認検証にさらに時間を必要とする．

以上を考慮すると，問題摘出が可能で，かつ発生した

表4　検証時間の見積もり

大項目名	中項目名	見積もり（時）
系構成変更	フェールオーバ	(0.8)
	組込み	(0.8)
単一障害	プロセスダウン	(2.6)
	ネットワーク障害	(0.7)
複合・多重障害	ラックダウン・復旧	6.7
長期安定	長期安定化確認	(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)
復旧性能	リカバリ	5.2
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
	時間区間指定読み出しの高速化	(1)

注：（ ）内は事前検証の方法以外（一般的な情報システムで適用されている類推[15]）で見積もった値



注：事前検証のスケジュールはA月の前に実施

図6　実例での計画と実績（事前検証実施）

問題の解析と対処の確認がしやすいよう，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計画することが，期間内での検証を可能とするための重要な課題となる．

6.1 課題解決の考え方

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．

(a) 従来手法の課題調査

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．利用可能なすべ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．

(b) 課題解決へのアプローチ

この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) 検証項目に対応したマシン台数の割り当て方

システム検証の前工程（単体検証，結合検証）に着目する．単体検証はユニット内で行われるため，システムの最小構成で実施可能である．結合検証はコンポーネント内で行われるため，そのコンポーネント以外は，最小のシステム構成で実施可能である．また，コンポーネント単独の検証では，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．このことから，以下の3つの考え方でマシンを割り当てる．

- 小規模構成は，マシン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．
- 中規模構成は，運用上の都合により環境が縮退した場合を想定し，機能検証とマシン台数に依存しない非機能要件（性能と可用性）の検証を行う．
- 大模構成は，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・マシン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．

たとえば，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．

適正なマシン台数を検証環境に割り当てるこれらの方法は，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，ほかの大規模分散処理システムにも適用可能である．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，より一層の効率化も可能である．

6.2 具体的な解決方法

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方法を検討した．このうち大規模環境については，サービスイン時のマシン台数を設定する．また，大規模環境はサービスイン時のマシン台数規模以上，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．検証項目は，大規模環境，中規模環境，小規模環境は6.1節の考え方に基づいて検証環境に割り当てる．

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，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に割り当てて，検証実施の効率化を図ることができる．同様に，中規模環境の項目を小規模環境に割り当てて，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は，小規模環境に割り当てている項目を中規模環境で実施することもできる．

項目を検証環境に割り当てたのち，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．

6.3 適用結果

CBoC タイプ2への適用結果について述べる．6.2節の解決方法に基づき，表5のように割り当てた検証項目に

従ってシステム検証を実施した結果を図7に示す．このように，問題を摘出して収束することができ，そして，計画どおりの検証期間での製品出荷ができた．大規模分散処理システムの開発において初めてシステム検証を実施する場合，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のように問題数が推移したかを図8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すべ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とが分かる．しかし，最初から中大規模環境で検証するのでは，小規模環境でデバッグ可能な問題も中大規模環境で行うことになり，解析や対処確認の再現試験の効率が下がる．6.1節で示したとおり，過去のシステム検証の例では，すべての問題を300台の大規模環境で摘出して対応した場合には約1年を要するのに対して，提案の方法を適用した場合には6カ月で品質が確保され，製品出荷が可能になった．

表5　検証環境ごとの項目割り当て

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

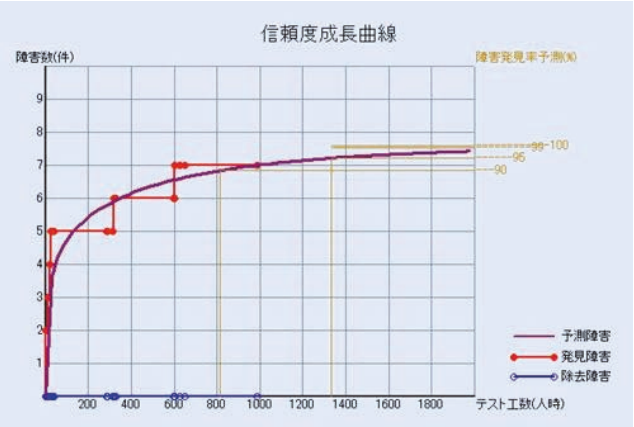
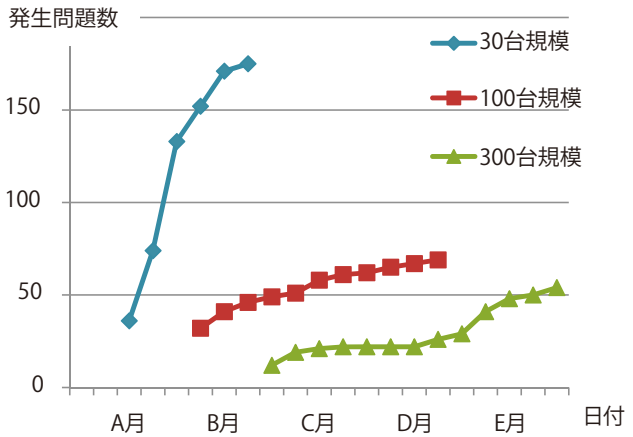


図7　問題の収束状況

7. まとめと今後の課題

本稿では，大規模分散処理システムの検証について，3つの課題を整理した．それらの課題に対して効率的に実施可能な解決方法を提案し，有効性を確認した．1点目は，ネットワークとディスクのI/Oのボトルネックとなる場合に適用すると有効であり，ネットワークとディスクのI/Oのチューニングで解決できることを示した．2点目は，大量のデータを検証で利用する場合に適用すると有効であり，大量にリソースを使用する項目の見積もりで解決できることを示した．3点目は，マシン台数が限られている場合に適用すると有効であり，数段階の検証環境の割り当てで解決できることを示した．

提案した課題の解決方法については，効果が確認できたが，以降に示す課題がある．1点目の「検証データの安定した高速登録」では，解決方法の検討過程において，効果の見られなかった方法がある．これはパラメータを他に変更することで，効果が得られる可能性がある．2点目の「検証項目の実施時間の見積もり精度向上」では，事前検証は代表的な数項目にしている．そのため，1項目あたりの所要時間は3時間以上でぶれ幅が大きい．時間単位の精度ではなく，分単位の見積もりの精度にする方法は発見できていない．3点目の「効率的な検証マシンの割り当て」では，小規模環境と中規



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．

図8　検証環境別の発生問題数の推移

模環境を同時に利用し，検証項目を並列に実施する方法は評価中で確立できていない．これらは，引き続き検討が必要である．また，本解決方法は，大規模分散処理システムと別システムとの接続によるインタフェース確認の検証のように，検証データの準備や性能測定がない検証には適用はできない．この場合は，一般的な情報処理システムの検証を参考にする必要がある．さらに，評価ができていない計画以外のアクティビティについては，評価をする必要がある．

参考文献

1) 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)．
2) Niino, J.：「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).
3) 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).
4) Gao, J., Bai, X. and Tsai, W. T. : Cloud Testing – Issues, Challenges, Needs and Practice, SOFTWARE ENGINEERING, An International Journal, Vol.1, No.1 (2011).
5) 坂井俊之，梅田昌義，中村英児，本庄利森：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，デジタルプラクティス，Vol.4, No.1, pp.51-59 (Jan. 2013).
6) 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，デジタルプラクティス，Vol.4, No.1, pp.44-50 (Jan. 2013).
7) Sigelman, B. H., Barroso, L. A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspan, S. and Shanbhag, C. : Dapper, a Large-Scale Distributed Systems Tracing Infrastructure, Google Technical Report (2010).
8) 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).
9) 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 (Aug. 1996).
10) 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).
11) Kaner, C., Falk, J. and Nguyen, H. Q. : Testing Computer Software 2nd edition, John Wiley & Sons, Inc (1999).
12) Myers, G. J. : ソフトウェア・テストの技法 第 2 版，近代科学社 (2012)．
13) 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).
14) 長尾　真（監訳），松尾正信（訳）：ソフトウェア・テストの技法 第 2 版，近代科学社 (2012)．
15) 松本吉弘 (監訳)：ソフトウェアエンジニアリング 基礎知識体系 SWEBOK，オーム社 (2004)．
16) PMI : A Guide to Project Management Body of Knowledge, 4thed., PMI (2008).

梅田 昌義（正会員）umeda.masayoshi@lab.ntt.co.jp
NTT ソフトウェアイノベーションセンタ第二推進プロジェクト主任研究員．1991 年電気通信大学電気通信学部情報数理工学科卒業．同年，日本電信電話（株）入社．現在，大規模分散処理システムの研究開発に従事．

鬼塚 真（正会員）onizuka@ist.osaka-u.ac.jp
NTT ソフトウェアイノベーションセンタ．1991 年東京工業大学工学部情報工学科卒業．同年，日本電信電話（株）入社．分散システムのアーキテクチャの研究に従事．2013 年より電気通信大学客員教授．現在，大阪大学大学院情報科学研究科教授．博士（工学）．

投稿受付：2014 年 7 月 23 日
採録決定：2016 年 3 月 24 日
編集担当：串田高幸（日本アイ・ビー・エム（株））