性能検証プロセスの導入により 性能問題を未然に防止または迅速に解決する取り組みと 高度な技術者育成について

川村 冠東[†1]　和田 美江子[†1]　中越 修[†2]

概要：性能問題はプロジェクト終盤で検出されることが多く，開発遅延やコスト増大の原因となる．プロジェクト終盤での性能問題を防止するためには，システム開発のなるべく早い段階で性能問題の原因を解消することが必要である．本稿では，性能問題を早期に検出できるようにするための取り組みとして，過去に発生した性能問題の原因の分析，システムの特徴応じた性能リスクを低減するための性能検証プロセスの策定，性能測定・分析ツールの選定と標準化の活動を報告する．また，本プロセスの適用により性能問題を早期に検出したプロジェクト事例を紹介する．また，高度な技術を持った人材幾瀬の取り組み付いても紹介する．

キーワード：性能，性能問題検出，開発プロセス

Measure to prevent or quickly solve performance problems with introducing performance verification process, And human resource development with advanced technology

KANTO KAWAMURA[†1] MIEKO WADA[†1]
OSAMI NAKAGOSHI[†2]

Abstract: Performance problems are often appeared at the end of the project, and it cause development delay and cost increase. In order to prevent performance problems, detection at an early stage of development is important. In this paper, we analyzed the cause of performance problem from our case, proposed a performance verification process to reduce performance risk according to system characteristics, and select and standardization of performance measurement and analysis tools. In addition, we introduce examples of projects that detected performance problems early by application of this process. In addition, I will show measure of human resources development with advanced technology.

Keywords: Performance, Performance bottleneck identification, Development Process

1. はじめに

システム開発におけるプラットフォームは多様化しており，マイクロサービスアーキテクチャを採用したシステムなど，多数のコンポーネントの組み合わせで構成されるシステムが増加している．このようなシステムで性能問題が発生すると，その影響は多岐に渡る．特に運用中のシステムや，出荷直前で性能問題が発生した場合，後追いで性能問題を解決するために膨大な時間とコストが必要になる．

著者が所属する部門は，システム品質向上に向けたプロセス整備とプロジェクト支援を行う部門であり，筆者は性能支援グループに所属している．支援活動としては，性能問題が発生したプロジェクトから連絡を受け，現場に行って性能分析を行い，分析結果をプロジェクト側にフィードバックするという活動を行っている．近年は特に，性能問題の防止に力を入れて取り組んでいる．

本稿では，これまで支援対応した性能問題を分析し，特徴を述べる．続けて，問題を防止するための方法として，システムの特徴応じた性能リスクを低減するための性能検証プロセスを整備し，標準ツールを選定した活動を報告する．最後に，本プロセスの適用により実際に性能問題を早期に検出したプロジェクト事例を紹介する．

2. 過去に支援した性能問題の特徴

当部門では過去の性能問題の発生の事例を分析し，その原因と解決方法に関するノウハウを蓄積し，再発を抑制するための技法，プロセス，ツールを整備することで，全社的な SI 品質の向上を目指している．

これまでに対応した性能問題の概要と，その原因の例を挙げる．

(1) 利用実績のないハードウェアやミドルウェアの利用に起因する問題

ハードウェア性能の限界や，特定のリソースのみ処理が集中するといった原因で，性能問題が発生する．特に初めて

†1 日本電気株式会社
NEC Corporation
†2 NEC ソリューションイノベータ株式会社
NEC Solution Innovators, Ltd.

使用する機器，ミドルウェアでは想定通りに性能を出すことが難しく，後工程で問題が発生した場合，ハードウェア構成や利用ソフトの検討からやり直す必要がでてくる．

(2) **実装したアプリケーションに起因する問題**
ミドルウェア，フレームワークなどの使用方法が誤っているといった単純な問題から，複数トランザクションによるリソース競合が考慮されていないなど，設計，実装の誤りが性能問題の原因となる．テスト工程で問題が発生した場合，どのコンポーネントがボトルネックかを切り分け，設計，実装をやり直す必要がでてくる．

(3) **高負荷に起因する問題**
DB 処理やネットワークの遅延，多数のプロセスの同時アクセスによるリソース競合など，設計時に想定していない事象が原因で性能問題を引き起こすことがある．原因を特定するためには,各ソフトウェアの挙動を一元的に把握し，問題となる箇所を検出する必要がある．

性能問題が発生したプロジェクトにおいて，問題を早期に検出できなかった理由についてヒアリングをおこなったところ，次のコメントを受けた．

- 機能要件への対応が優先され,性能要件の検証は開発終盤まで行っていなかった
- 処理時間の測定以外に，何をどうやって測定し，検証すればよいかわからない

これらへの対応が性能問題の早期検出に必要となると考えて，性能問題を防止するためのプロセスの策定と，担当者が性能検証を容易に実施できるようにするためのツール，技法の整備を進めている．以降の章で，その取り組みを紹介する．

3. 性能問題を防止するためのプロセスの策定

性能要件への対応を後回しにしないためには，それぞれのプロジェクトで開発対象システムの性能リスクを把握し,性能リスクに応じた対策をシステム開発計画の段階から組み込んでおくことが重要である. 2 章で述べた支援の実績に基づき，対応すべきプロセスを策定した．

(1) **利用実績のないハードウェアやミドルウェアの利用に起因する問題**
ハードウェア，ソフトウェアの挙動を予測した上で，システム全体のアーキテクチャを見積もり，要件定義で定めた性能要件との妥当性を確認する．

性能見積もりについては，シングルプロファイル法[1]やシステムモデルベース SI 支援環境[4]などを利用して実施することができる．

特に，これまで利用実績がないハードウェア，ソフトウェアを採用する場合は,テスト機やプロトタイプを用意し，代表的な処理を実測し，処理の挙動とシステムリソースの使用量を把握することで，実環境により近い情報で見積もりを行うことが必要となる．

また，プロトタイプなどでその挙動，性能を見える化することで，処理時間を増大させる要因を把握しておくことが，性能問題発生リスクを抑制する上で重要である．

(2) **単体テストでの性能検証**
単体のコンポーネント，シングルトランザクション処理の性能検証を行い，設計と照らし合わせ，性能ボトルネックとなる処理，振る舞いの有無を確認する．特に，データベース関連の処理，ネットワークに関する処理を行うコンポーネントは,単体で性能問題があることが多く,処理時間，通信回数，データ送受信量などが設計と照らし合わせ妥当かを確認することが必要である．

(3) **結合テストでの性能検証**
機能単位に処理を測定し，レスポンス時間の遅れの要因の有無を検証する．加えて，複数トランザクション実行状態においても同様に検証を実施する．特に，マルチトランザクションの性能検証では，トランザクション間のリソース競合の有無と，その遅延が設計と照らし合わせ妥当かを確認することが必要である．

(4) **総合テスト，負荷テストでの性能検証**
実環境または同等の環境で，実際に想定する負荷シナリオでの処理を測定し,性能要件を満たしていることを確認し，問題があれば原因の分析と対策を施す．性能要件を満たすことを確認すること以外に，高負荷時におけるシステムとしての限界を把握すること，長時間連続稼働時のリソース解放漏れ，枯渇の傾向を把握することが性能問題発生のリスクを抑制する上で有効である．

各フェーズの性能検証で，性能問題を検出するためには，二つの観点が必要である．
一つ目は複雑なシステムを効率的かつ正確に測定を行い，挙動，性能を見える化する手段をもつことである．性能問題は利用ソフトウェアと連携して動作するミドルウェアの複合的な要因により発生するなど製品個別のログのみでは原因特定に時間がかかるケースが多かった．過去に支援した性能問題で，性能遅延となったボトルネックの要因の例を以下に示す．性能検証でボトルネックを検出するためには，これらの挙動，性能を見える化し，設計と照らし合わせ妥当かを確認する必要がある．

表 1　ボトルネックの例

分類	問題の例
システムリソース	CPU，メモリ，ディスク，ネットワークなどのリソースが不足している
関数呼び出し	関数の呼び出し，繰り返し処理など，不要，非効率な処理を多数実行している
スレッド処理	多数の CPU コアがあるが，スレッド分散処理が正しく実装できていないため，CPU コアを有効に使えていない
プロセス間通信	プロセス間通信が多数発生しているため，遅延が発生する
マシン間通信	クライアントとサーバ間，AP サーバと DB サーバ間など，マシン間の通信が多発しているため，遅延が発生する
トランザクション処理	トランザクション間で，スレッド，DB などの排他制御が行われているため，処理待ちが発生する

二つ目は性能要件にあった適切なテストの実施である．高負荷についても，性能要件として想定される負荷によるテストを効率的に実施できるようにすることが必要である．また現状の構成でどこまで処理可能かという，システムの性能限界を把握しておくことが役に立つ．

4. 社内標準とする性能検証ツールの選定

性能検証を行える要員が少ないという課題に対しては抜本的な対策が必要であると認識した．性能検証は全てのプロジェクトが実施するものであり，前節で述べたプロセスを各プロジェクトで着実に実施できるようにする必要がある．

プロジェクトの限られた要員と工数で効率的に性能検証を進めるにはツールの活用が欠かせない．前章で述べた複雑なシステムの挙動，性能を見える化するためのツールとして，性能測定・分析ツールを，性能要件にあった適切なテストの実施を行うためのツールとして，負荷テストツールを標準ツール選定の対象とした．

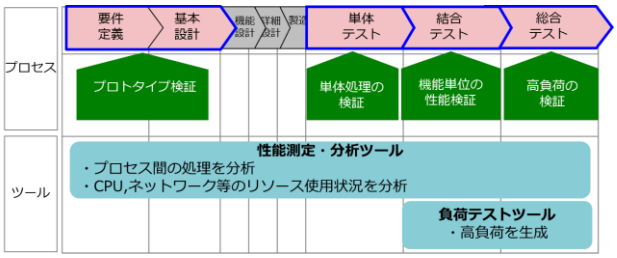

図 1 開発フェーズと性能検証ツールの対応

著者の所属部門では，社内へ展開する標準ツールの選定とその展開を行っており，標準ツールの選定については以下の基準がある．今回の選定では次表の確認項目に沿って選定した．

表 2　標準ツールの選定基準

選定基準	選定での確認項目
適用範囲	・対応 OS，言語，通信プロトコル ・ツール習得の難易度
実績	・社内の利用プロジェクト数 ・適用効果
サポート体制	・社内ユーザに対する問合せ，導入支援の提供ができるか

4.1 性能測定・分析ツールの選定

OS や言語に依存せずに利用ができ，社内ツールとして 10 年に渡る利用実績がある mevalet(メバレット)を採用した．mevalet はイベント・トレース手法[1]によるデータ取得と可視化を実装したツールである．イベント・トレース手法とは，プロセス，CPU，ディスク，通信などのリソースのイベントの時系列を採取して処理フローを生成し，ユーザからの処理要求(トランザクション)に対する処理時間の内訳をリソース使用状況と紐づけてプロセス単位で分析する手法である．

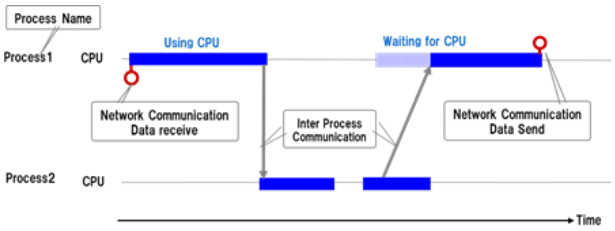

図 2　性能情報の可視化の画面イメージ

mevalet ではプロセス情報として名前と ID が確認できる．プロセス処理時間は CPU 使用時間が対応し，当該プロセス行の横棒線で表示される．また，プロセス処理の開始(再開)待ち時間については CPU 待ち時間が対応し，横棒線(網掛け)で表示する．さらに右クリックメニューから，プロセス処理で利用した CPU 番号，システムコールの発行順番などの情報も確認することができる．

通信に関して，プロセス間通信はプロセスからプロセスへの線で表示される．プロセス間通信に要する時間は，線の起点の時刻と終点の時刻の差分で把握できる．また，他のコンピュータで動作するプロセスとの通信についても，時刻，通信先と送受信データ量が確認できる．

同手法によるオープンソースのツールも公開されており LTTng，Trace Compass などが利用できる．

4.2 負荷テストツールの選定

オープンソースの Apache JMeter を選定した．主要な通信プロトコルをサポートし，社内での利用実績が多くサポート部門もあるためである．

5. 適用事例と成果

5.1 適用事例

前述で述べた各フェーズでの性能検証において，実際に性能問題を検出した事例を紹介する．

(1) 設計フェーズで検出した性能問題の事例

背景：これまで利用実績がないミドルウェアを採用することになり，設計段階でプロトタイプを作成し，その挙動，性能の見える化を行った．

結果：リソース使用量を測定する他に，多重度を段階的に増やし，処理時間を増大する要因を調査した．その結果，特定のプロセスがボトルネックとなり，マシンリソースを増やしても処理性能が一定以上伸びないことが検証できた．ソフトウェアの挙動を考慮したハードウェア構成，アーキテクチャを検討することで，後工程での性能問題の発生を抑制することができた．

(2) 単体テストで検出した性能問題の事例

背景：性能要件が厳しいシステムで，マルチコア CPU を使った分散処理を実装した．単体レベルで性能問題がないことを確認するため，性能検証を行った．

結果：イベント・トレースツールで，スレッドごとの CPU 使用時間，待ち時間を見える化した．その結果，処理時間は要件を満たしているが，分散処理が正しく実装できていないため，CPU コアが有効に使えていないことが検証できた．当問題を対処しないと，後工程の複数コンポーネントの組み合わせや，高負荷時に性能問題を引き起こす可能性があり，これらのリスクを回避することができた．

(3) 結合テストで検出した性能問題の事例

背景：ミドルウェア上に独自の業務 AP を作成するシステムで，結合時の動作を確認するため，性能検証を行った．

結果：イベント・トレースツールで，機能単位の処理を測定し，システム全体の挙動，性能を見える化した．その結果，不要な CPU 利用・待ちの発生，設計にないプロセス間通信が多発していることが分かった．計測結果をミドルウェアの製品部門と共有し，API の利用方法，設定などの誤りを検出し対処した結果，結合テストの段階で処理時間を 1/5 に改善することができた．

5.2 成果

2017 年度に本手法を 91 プロジェクトに適用した．その成果を表 3 にしめす．91 プロジェクト中，73 プロジェクトは性能問題がなかった．性能問題があった 18 プロジェクトのうち，総合テストより前に性能問題を検出できたプロジェクトは 10 プロジェクトあった．これは，従来の手法であれば総合テストで初めて検出する問題であり，後戻りを回避することができた．

表 3　本手法による性能問題検出状況

性能問題の有無	検出フェーズ	PJ 数
性能問題あり	設計	5
	単体，結合テスト	5
	総合テスト	8
性能問題なし	—	73
	合計	91

一方，総合テストで検出できなかった 8 プロジェクトは，本番環境と開発環境の違いなどで，本来総合テストで検出すべき問題であり，妥当な結果であると認識している．

6. 人材育成への取り組み

性能問題を防止するために各開発フェーズで確認すべき内容とその検証方法をプロセスとして整備したが，その技法を開発現場で実際に適用させるためには，全社的な人材育成への取り組み欠かせない．

まず 2005 年度に，上流工程における基本的な性能設計および見積技法に関する教育コースを開発した。その後、2010 年に下流工程での性能検証技法を学ぶ教育コースを開発した．今では，両コースとも広く一般に受講を開放している．なお，両コースとも知識の整理を主眼としており、以下に示すような性能問題を防止するための基本的な考え方についてまとめてある．

(1) 上流工程から性能を作り込む

- 性能要件を定量的に仕様化する
- できるだけ数値を集めて性能見積を行う．測定できるなら性能実測値を収集して利用する
- 性能問題が起きたときに必要となるデータをあらかじめ設計に入れておく

(2) 節目節目で性能を検証する

- 単体テストフェーズから性能を見える化し問題ないことを確認する
- 限界性能を把握する
- 負荷など性能要件が変わったら性能見積りからやり直す

さらに 2015 年には，全社から選抜された要員に対して，プロフェッショナルなシステムアーキテクトを育成するた

めのプログラムが開始され，その中に性能に関する内容が盛り込まれることになった。前述の知識の整理に主眼を置いた教育コースの強化が求められ、具体的には，大規模ミッションクリティカルシステム開発を想定したプラットフォーム設計における性能検証や性能面を考慮した運用設計についてなどを追加し、半期に一度講義を実施している．

人材育成の成果として，性能設計を経験した要員がリーダとなり,自らプロジェクトメンバを指導するのみならず,自組織内に横展開するなど，広がりを見せている．今後も人材育成を強化・継続していく予定である．

謝辞 本稿の執筆にあたってプロジェクトメンバやその他の関係者の方々から色々なアドバイスをいただきました．ここに感謝いたします．

参考文献

[1] Takashi, Horikawa.. Application of Event Trace Framework for Performance Problem Solutions, IPSJ SIG Technical Report, 2003.
[2] Kanto, Kawamura.. Performance measurement and analysis tool “mevalet”, NEC Technical Journal Vol 60, 2007.
[3] Takashi, Horikawa.. Performance Evaluation Method for DBMS Systems, IPSJ SIGARC Technical Report, 1994.
[4] Sayaka Izukura.. Evaluation of Performance and Availability based on Model-based System, IPSJ SIG Technical Report, 2010.
[5] Atsuhiro Tanaka.. An Evaluation method for Open Systems, IPSJ SIG Technical Report, 1994.
[6] Takashi, Horikawa.. An Analysis method for Scalability Bottlenecks and Case Study on Database Management Systems, IPSJ SIG Technical Report, 2009.