大規模トランザクション処理の性能対応

Performance Management for Large-scale Transaction System

高　井　健　志

要　約　大規模開発では，アプリケーション機能の開発がほぼ終了し統合テストに入ってから性能問題が発覚し，アプリケーションチューニング等で大幅なオーバーヘッドがかかり，その結果プロジェクト全体のスケジュール遅延やコストの増大につながることがある．

当該状況に陥らないためには，1)早期にアーキテクチャを決定しアプリケーションフレームワーク（共通機能）を構築する，2)構築したフレームワークの下でプロトタイプアプリケーションのベンチマークテストを実施し，最終形アプリケーションでの性能予測および性能に対するキーポイントを明確にする，3)アプリケーション開発前に性能検証の戦略を立て、単体テスト時より工程毎に性能を継続的に検証する，という対応を実施することが重要である．

本稿では，性能リスクにいかに対応したかを記述し，開発を通じ注力すべきポイントを明確化する．

Abstract　When development of an application function is mostly completed in large-scale development and after going into an integration test stage, performance issues may arise. It will increase the overhead of work to tune up application software, and as a result, it may lead to a schedule delay of the whole project, or an increase of cost.

In order not to fall into the situation concerned, it is important to take the following measures, 1) Determine the architecture at an early stage and build an application framework (common function), 2) Clarify the key points of performance prediction and performance related to the application in final form by carrying out a benchmark test with the prototype application under the mechanism concerned, 3) Work out a strategy for performance verification before application development, and verify performance continuously for every process from the time of a unit test.

This paper describes how measures were taken for performance risk, and how a lot of highly efficient transactions processing were realized, and clarifies the key points that should be focused on thru the development.

1. は　じ　め　に

全日本空輸株式会社（ANA）の次世代国内旅客系システム基盤“ANACore”は，1978 年以降米国 Unisys 社製メインフレームにて稼働してきた国内旅客系（予約・発券・搭乗）システム“able-D”の次世代システムとするために開発したシステムである．ANACore は，able-D が抱える問題・課題を解決し，また able-D の優位な点を踏襲しつつ，今後の環境の変化に耐えうる拡張性，柔軟性を備え持つオープンシステムとすることを基本コンセプトとしている．但し，これを実現させるためのプロジェクトを計画立案した当時の懸念事項として，当システムに求められる性能要件である“メインフレームの able-D システムと同等の処理能力”

をどのように実現するかが掲げられた.

当プロジェクト計画当時（2006 年）には，オープンシステムにてエアライン基幹業務を実現している事例はなく，一般的にもメインフレームの方がオープン系よりも処理性能が高いと言われている状況では，期待された処理性能を実現できないリスクがあった. また，当プロジェクトのような大規模開発では，アプリケーション機能の開発がほぼ終了し統合テストに入ってから性能評価が行われ，問題が発覚し，開発済みのアプリケーションをチューニングする等で大幅なオーバーヘッドがかかり，プロジェクト全体のスケジュール遅延やコストの増大につながることもリスクと考えられていた（以降，大規模開発のリスクと呼ぶ）.

本文では，このリスクにいかに対応し，大規模開発で，大量高性能のトランザクション処理をいかに実現したかを明確にし，開発を通じ注力すべきポイントを明確化する.

2. 事例概要

本章では，ANACore のシステムやその開発プロジェクトについて紹介する.

2.1 システム概要

前章で述べた通り，ANACore は able-D が抱える問題・課題を解決し，また able-D の優位な点を踏襲することがコンセプトである. able-D が抱える問題・課題は以下のとおり.

1） メインフレームの保守・更改コストの増加
2） ソフトウェアの老朽化対応の累積による生産性の低下/プログラムのスパゲッティ構造化
3） able-D の主要記述言語である FORTRAN 技術者の要員確保の難しさ

対して able-D の優位な点は以下のとおりである.

1） 24 時間＆ 365 日の無停止＆高安定稼働が実現している
2） 大量高性能のトランザクション処理を実現している
3） 旅客業務と密に連携し，操作員も長年の利用で親しんでいる木目細かな機能

これらを新システム上で実現するためのプラットフォームとして，米国 Unisys 社が開発した Java 言語で構築された Airline 向けパッケージ AirCore（Airline Core Systems Solutions）を選定し，カスタマイズと再構築をすることで，ネットワークキャリア[*1] では世界初（2006 年 4 月時点）となるオープンシステムにて今後 20 年間使用可能な拡張性・柔軟性を備えたシステム基盤の確立を目指すこととした.

2.2 システム構成

最終的にシステムテストを完了し確定した本番稼働時のシステム構成概要を図 1 に示す.

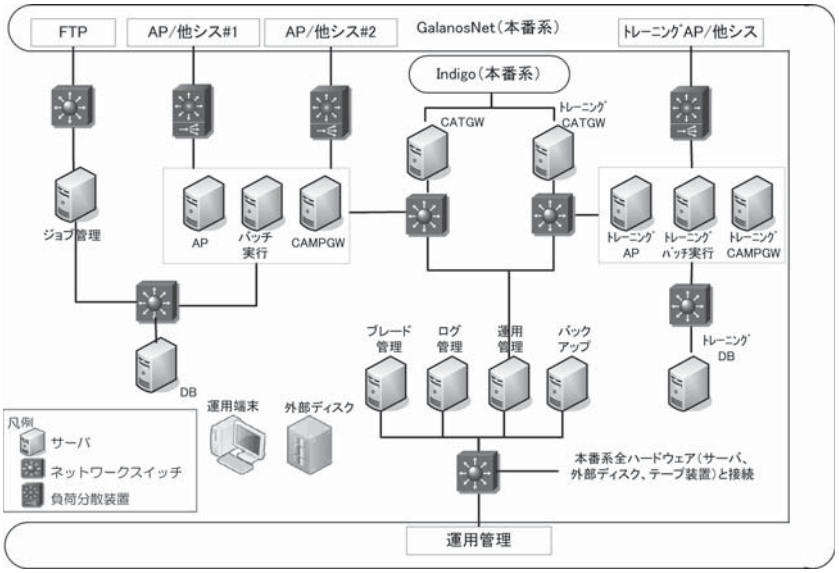

図 1　ANACore システム論理構成図

2. 3　イテラティブな開発プロセス概要

ANACore の開発では反復開発（イテレーション開発）を採用しており，以下のような機能グループ分けをしたイテレーション開発を行った．

機能グループ 1：主要業務（オンライン）機能の開発

機能グループ 2：サブ業務（オンライン）機能の開発

機能グループ 3：バッチおよび他システム接続機能の開発

機能グループ 4：開発期間中に現行 able-D システムで実施された現行改修案件の取込（3 回に分けて実施）

各機能グループの開発で，要件定義・設計・実装/単体テスト・結合テスト・機能テストを実施し，一つのイテレーションで評価可能な成果物が構築されるプロセスとした．この構築されたアプリケーションを基に段階的に検証を繰り返し，開発初期段階で大きなリスクに対処することでリスク軽減を図ることを目的としたプロセスである．当該開発プロセスは，各グループ終了時に検証可能な成果物ができあがることで，4. 3 節で述べる段階的な性能検証が可能となり，性能問題に対しても効果のあるプロセスとなっている．

3.　解決すべき課題と対応方針

3. 1　性能要件

プロジェクト計画時に ANACore に求められた「現行システムと同等のサービスレベルの実現（処理性能要件）」とは，具体的にはサーバ処理性能（1 秒あたりの処理件数）と端末のレスポンスタイム（空港設置端末と一般端末のそれぞれ）だったが，これらの正確な値は諸般の事情により非公開とする．

3. 2　対応方針

大規模開発のリスクに対応するための方策として以下を挙げた．

1） 早期にアーキテクチャを決定しアプリケーションフレームワーク（共通機能）を構築する
2） 構築したフレームワークのもとでのプロトタイプアプリケーションでベンチマークテストを実施し，最終形アプリケーションでの性能予測および性能に対するキーポイントを明確にする
3） アプリケーション開発前に性能検証の戦略を立て、単体テスト時より工程毎に性能を継続的に検証する

4. 実践した課題対応内容

3.2 節で挙げたリスク対応の方策について，本章で詳しく説明する．

4.1 早期のアーキテクチャの決定とアプリケーションフレームワークの構築

開発初期段階で，システム化要件で求められている処理性能を ANACore でも実現可能であることを検証し，適正なハードウェア構成を検討する必要がある．そのためには、早期にアーキテクチャを決定しアプリケーションフレームワーク（共通機能）を構築した上で，当該機構上で性能要件の実現性を確認することが必要となる．また、リソースの拡大に応じた処理量の拡大＝スケーラビリティを得るためには，その前提として個々の機能がシステムリソースを多量に消費する等の非効率な処理をしないことが前提であり，それを実現するベースとなる機構と，個別機能の最低限の性能品質を確保するための開発ガイドラインや規約が重要になる（非効率的な SQL を書かないための具体的な記述指針等）．

4.2 プロトタイプアプリケーションでのベンチマークテスト

前節のフレームワーク上で国内旅客の主要業務アプリケーション機能[*2]についてプロトタイプアプリケーションを作成する．当該プロトタイプを使用してベンチマークテストを実施し，プロトタイプによる負荷増を基準に本番アプリケーション実装時の負荷増を想定し最終アプリケーションの性能を机上算定する．

ここでのキーポイントは，フレームワーク（共通機能）の主要部分[*3]をできる限り最終形に近い形で構築することである（図 2）．これにより性能評価をより信憑性のあるものにできる．また，全てのアプリケーションは当該フレームワーク上で構築されるので，一部機能でのプロトタイプによるベンチマークテストを基に，最終形の性能予測が可能となった．

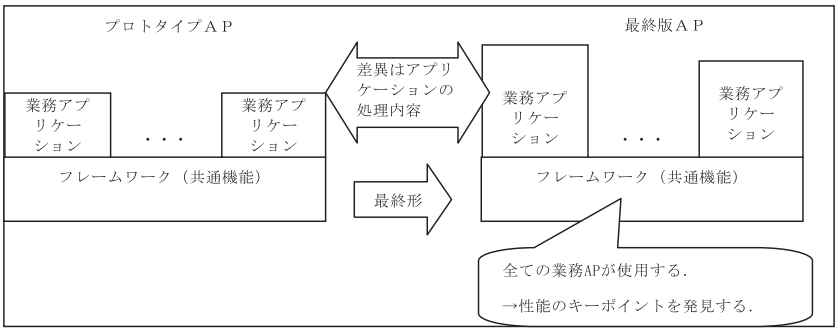

図 2 プロトタイプによるベンチマーク概念図

また，このベンチマークテストを，予め候補として挙げた複数のプロダクトセット（ハードウェア，ソフトウェア）にて実施することで，プロダクトセットの選定にも利用し，同ベンチマークテストの結果より，本番に向けた機器構成を算定[*4]した．なお，機器構成算定にあたってはサーバ毎の特徴を考慮して拡張方針を定義（スケールアップ，スケールアウト）し，万一性能の問題が発生した場合にも，リソースの拡大で対応が可能な構成とした．プロダクトセット選定作業および作成物の流れを図3に記述する．

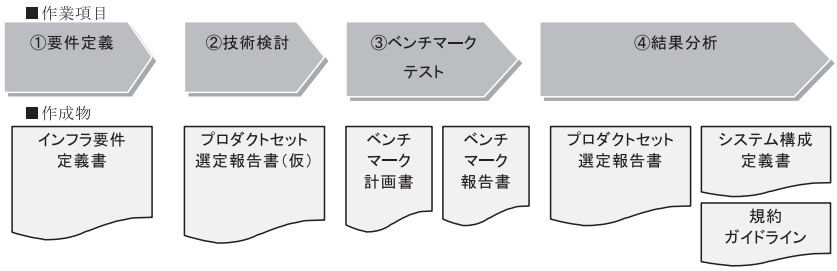

図3　性能要件実現性の確認およびハードウェア構成検討フェーズ作業フロー図

4.3　アプリケーション本格開発開始後の戦略

4.3.1　個別アプリケーション開発時（機能テスト実施時）の継続的性能チェック

大規模開発のリスクに陥ることを防止するため，開発工程毎に性能品質成長レベルを定義し，個別アプリケーション開発時の各工程での性能のチェックのクライテリアを設け，またアプリケーション開発の各フェーズで，性能測定および分析を行い，次フェーズでのアプリケーション開発にフィードバックすることとした．性能品質を本番稼働レベルまで段階的に成長させていく戦略として，図4に示すLEVEL0からLEVEL4までの成熟シナリオを策定した．

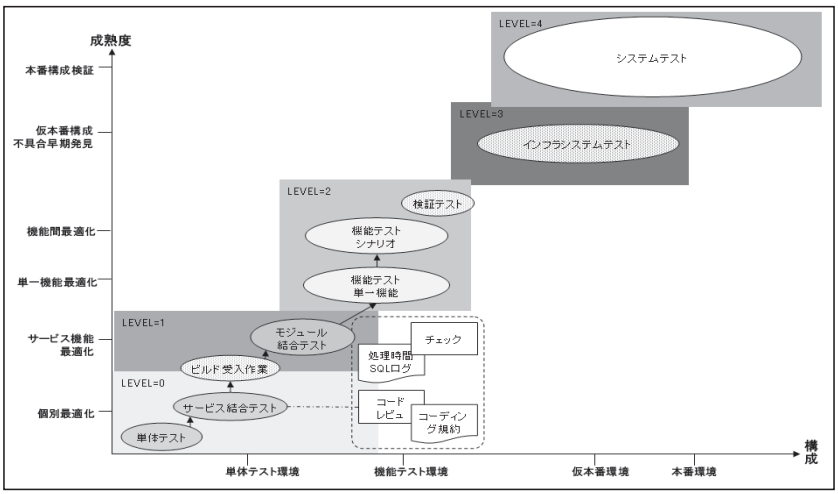

図4　性能品質成長戦略

性能品質はアプリケーションで見る性能とアプリケーションとインフラを含め統合的に見るシステムとしての性能の両面から対処した．図4のLEVEL0～LEVEL2までがアプリケーション単体で見る範囲であり，LEVEL3～LEVEL4がシステムとして見る範囲である．図の縦軸が性能品質の成熟度であり，横軸が性能測定環境の構成である．

それぞれの性能品質のレベルを次のように定義した．

1） 性能品質 LEVEL0（表1）

LEVEL0はコードレベルの検証でコーディング規約レベルの検査をパスしていることを意味する．

表1　性能品質 LEVEL0 の定義

LEVEL0	コードレベル ― サービス単位評価可能な状態
クライテリア	コーディング規約レベルの検査をパスしている 性能面に関する考慮がされたJavaコーディング規約とSQLコーディング規約に準拠していることを静的チェックツール(FindBugs/CheckStyle)及び，コードレビューを受け，指摘事項が解消されている
評価単位	クラス
工程	実装～単体テスト
補足	静的検証は環境に依存しないため，評価環境は規定しない 単体テストはDBアクセスを行わないため，実際にDBと接続したテストは次工程で行う

2） 性能品質 LEVEL1（表2）

LEVEL1はサービスレベルの検証で，結合テスト環境にて性能面で悪影響を及ぼすSQL正当性の検査をパスしていることを意味する．

表2　性能品質 LEVEL1 の定義

LEVEL1	サービスレベル ― 機能単位評価可能な状態		
クライテリア	SQL ログにより，SQL 発行回数が想定以上でないことが確認されている．対象外のTABLE が記録されていない．		
評価環境	項目	対象	備考
	単位	サービス	他モジュールはSTUBを利用
	DB	テスト用DB	データ量はテスト用
	HW 環境	開発PC	レスポンス時間は評価しない
工程	対象リリース	工程	備考
	全リリース	結合テスト	
補足	結合テストは自モジュール内に限定したDBアクセスが行われる．STUBを廃止した機能及びレスポンスタイムの評価は次工程で行う．ここでは自モジュール内で意図した DB のアクセスを回数や対象となるTABLEの観点で評価を行う		

3） 性能品質 LEVEL2（表3）

LEVEL2は，単一機能レベルの検証で，機能テスト環境にて機能個別レスポンスタイムの検査をパスしていることを意味する．

表 3　性能品質 LEVEL2 の定義

LEVEL2	単一機能レベル ― 全体評価可能な状態		
クライテリア	単一機能として並行処理負荷を掛けない状態でレスポンスタイムが評価されている． STUB を廃止した状態で測定され，意図しない DB アクセスがない． 個々の SQL の発行内容，応答時間が評価され，悪さが明確であり，対応策ができている． BATCH は同時実行の条件が明確で，並行処理の前提となる基準値が評価されている		
評価環境	項目	対象	備考
	単位	機能	
	DB	テスト用 DB+移行 DB[※5]	
	HW 環境	機能テスト	
対象	対象	評価条件	備考
	リアル系 バッチ系 他システム接続系	ELAPS SQL 発行回数/必要**有無** 単一 SQL の処理時間	それぞれの機能テストのテストケース条件に従う 性能としての特定の条件（DB 容量，検索条件）は規定しない
	フレームワーク系	なし	フレームワーク系は特定の機能に依存しないため，全体として評価する
工程	対象リリース	工程	備考
	全リリース	機能テスト	
補足	機能テストのログを評価．実施した機能テストケースで，処理時間が一定クライテリアに満たないものは改善を施し，全体並行処理負荷でボトルネックとならないようにする		

2. 3 節で述べた通り当プロジェクトでは，機能グループに分割した開発を行っている．当プロセスでは，各機能グループの開発（要件定義から機能テスト）での LEVEL0 ～ LEVEL2 の評価を毎回開発プロセスの一部として実施することとした．

4）　性能品質 LEVEL3（表 4）

LEVEL3 は，全体評価レベルの検証で，システム全体（同時並行処理下）の性能テストを行う．

LEVEL3 検証は，2. 3 節の開発プロセスで示した各機能グループの機能テスト終了毎に，開発済アプリケーション機能を使用して行うこととした（現行改修案件の取込は 3 回それぞれで実施した）．当該 LEVEL の検証により初めてシステム全体として性能要件を達成可能なレベルにあることが確認される．従って LEVEL3 検証時には業務状況の変化等による性能目標の見直しをした上で性能測定を行い，最終的に性能要件を達成できるかを評価し，分析結果から得られた改善策を次の機能グループの開発にフィードバックする．この PDCA サイクルを回すことにより段階的に性能のリスクを低減する計画とした（図 5）．

性能測定 PDCA サイクルは以下の通り

ⅰ）業務状況の変化等による，性能目標値の見直しを行う．

ⅱ）上記ⅰ）の見直しに従い性能測定モデルの定義・見直しを行う．

ⅲ）設計した構成による実機での性能測定を実施する．

ⅳ）ベンチマークテスト結果より，結果を考察し最終性能を予測する．ここで，性能要件が満たされていない場合は，原因を解析してチューニングを実施し，再度測定を行う（性能測定モデルが不適切な場合は，性能測定モデルの再定義から実施する）．

ⅴ）当該サイクル中で検出された性能課題の分析結果を，コーディング規約・SQL 規約・アプリケーション設計ガイド等へ反映するとともに開発者に周知し，次グループの開発の入力とする．また，当該課題がアプリケーション共通の場合は，アプリケーションフレームワークの改善を実施する．

表 4　性能品質 LEVEL3 の定義

LEVEL3	全体評価レベル ― 業務品質評価可能な状態		
クライテリア	性能要件の評価対象と評価条件が定義され，ベンチマークにて負荷を掛けた状態で評価されている 同時並行処理を阻害する重障害は対策を施され，結果が確認されている 性能目標値に到達していない機能は原因が明確であり，対応策ができている		
評価環境	項目	対象	備考
	単位	システム	システムを代表する機能を選択して実施
	DB	移行 DB	移行 DB を基に評価に必要なデータを追加 追加の必要なデータはテストテーマ別に選出
	HW 環境	本番相当構成	稼働前に構成見直し
対象	対象	評価条件	備考
	リアル系	性能目標で定義されたトランザクションミックスを用いて，レスポンスとスループットを評価	業務状況の変化等により性能目標を見直し，対象機能のテスト条件を規定する
	バッチ系	運用要件	バッチの処理順序の依存関係，期待される終了時刻など
	他システム接続系	処理形態に依存	リアル系，バッチ系に分類する
	フレームワーク系	テーマ別に設定	
工程	工程		備考
	各開発グループの機能テスト終了時		並行処理及び，負荷を掛けた状態での悪さを明らかにする．
補足	ANACore システムとして初めて負荷を掛けた状態での並行処理をテストする工程となる．		

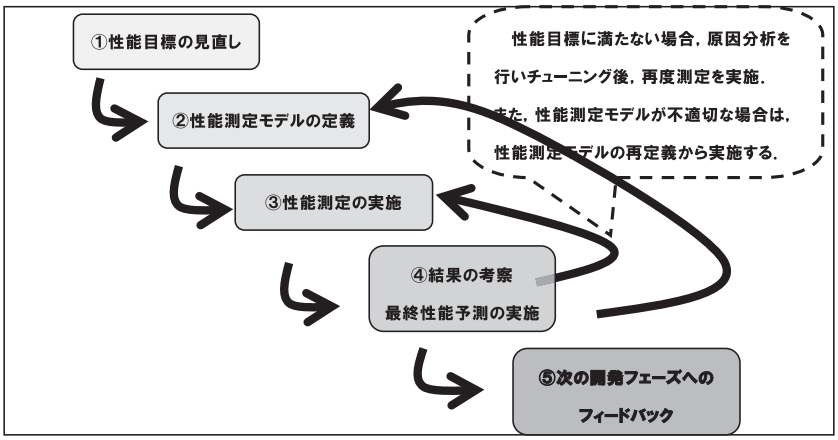

図 5　性能測定 PDCA サイクル概念図

5）性能品質 LEVEL4（表 5）

業務品質レベルにあることの検証として，本番リリース版アプリケーションを使用しシステムテスト環境にてシステム全体の性能テストを行う．この結果，性能要件にパスしている事が確認できれば，LEVEL4 に到達している．

表 5　性能品質 LEVEL4 の定義

LEVEL4	業務品質レベル ― 業務適用可能な状態		
クライテリア	システムに求められる性能要件が満たされている		
評価環境	項目	対象	備考
	単位	システム	
	DB	本番 DB	ANACore 稼働前は本番 DB に able-D 側から移行した DB を用いて，データ量及び内容がリリースされる稼働環境同等とする
	HW 環境	本番相当構成	
対象	対象	評価条件	備考
	リアル系	非機能要件として ・空港端末のレスポンス ・一般端末のレスポンス ・サーバスループット ・ストレステスト ・ロングランテスト	評価は事前に LEVEL3 で行い，問題点が明確になった後，最終確認をこの工程で行う
	バッチ系	バッチの運用要件	
	他システム接続系	他システムの運用要件	
	フレームワーク系	-	フレームワークを利用する機能の性能にて評価
工程	工程		備考
	現行改修案件取込のシステムテスト		
	全体システムテスト		
補足	SLA の基準とする対象機能，条件を定め，今後の改修時に同じ条件で評価可能とする		

4. 4　性能測定ツールとテスト内容

本節では，各テストで使用した性能測定ツールとテスト内容について記述する．

4. 4. 1　パフォーマンステスト（ラッシュテスト＆ストレステスト）

1）パフォーマンステストツール

今回のシステムは，able-D と ANACore で，基本的には入力電文の変更がなかった．able-D で採取した入力ログから作成されたトランザクションを ANACore へ投入する機構を構築することにより，本番業務のさまざまなピーク時間帯の処理状況をより正確にシミュレートすることが可能となる．当該機構を使用して入力内容・入力量を調整し，実際の本番システムでの入力電文を基にして、発売ピーク時間帯、搭乗ピーク時間帯等のさまざまなピーク時間帯をシミュレートすることが可能となった（図 6）．

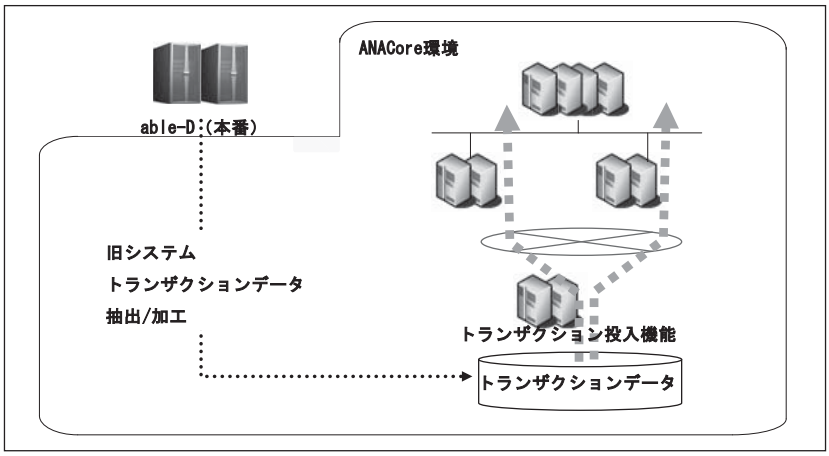

図6　パフォーマンステスト概念図

2）　パフォーマンステスト内容

■ラッシュテスト

性能要件に定義されているシステムのピーク時間帯の入力をシミュレートし，性能要件（スループットおよび応答時間）が満たされていることを，トランザクションの秒あたりの処理件数と測定対象期間の平均応答時間を集計して確認する．また，リソースの消費状況の評価としてシステム資源の使用状況（CPU・メモリ・IO 状況等）と基本プロダクトとして Oracle 稼働状況，WeblogicServer の稼働状況を分析し，期待通りの状態にあることを確認する．

■ストレステスト

ピーク時の負荷が一定時間維持されてもシステムが安定していることを確認する．ラッシュテストで対象とした入力をピーク入力以上の限界性能まで入力した場合のシステムの動作について応答時間，処理件数に大幅な劣化がないこと，リソース消費状況に大幅な増加がないことを確認する．

4. 4. 2　ロングランテスト

1）　ロングランテストツール

前項でも述べたが，able-D と ANACore は基本的には入力電文に変更がなかった．そこで，able-D への入力をリアルタイムで ANACore へ横取りし入力する機構を用意した．これにより，able-D と同タイミングで同入力を可能とし新旧の同一性を保った状態での長時間の安定稼働のテスト（図7　ロングランテスト）を実施した．

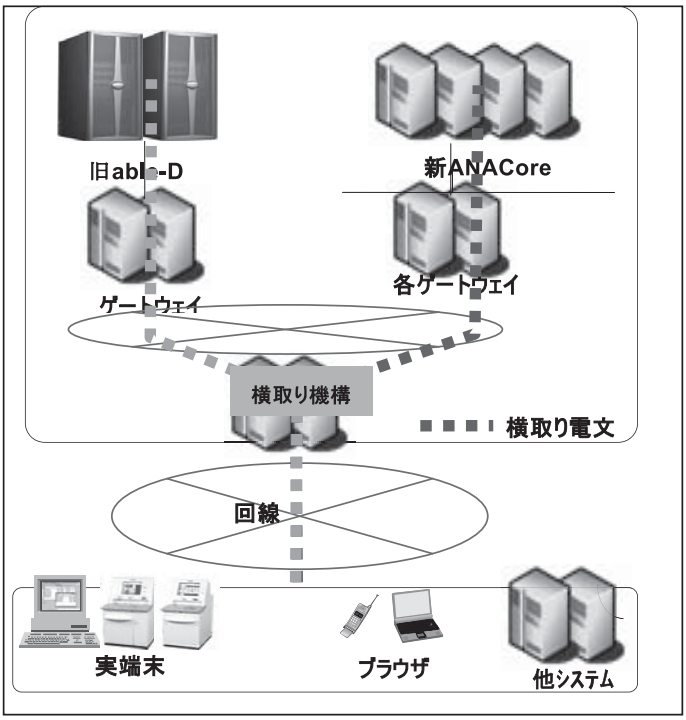

図 7　横取り機構を用いたロングランテスト概念図

2）　ロングランテスト内容

able-D のオンライントランザクションを用いて連続稼働させた状態での安定性を確認する．本番と同等の入力状況で長時間連続稼働（able-D と同等の処理状況での 48 時間の連続稼働）した状況で，以下を確認する．

・リソース使用率（AP サーバ及び DB サーバの CPU，メモリ）が安定している
・スループットが安定している
・レスポンスが安定している
・パフォーマンステストの監視項目に異常値が記録されていない

5.　評価

ANACore は，2013 年 2 月に本番稼働を迎えた．4. 3 節で述べた性能評価の戦略に従い，テスト・分析評価・対応策の実施を繰り返したため，システムテストでは性能の問題は殆ど発生しなかった．特に本番稼働後も性能目標値を上回る稼働状況となっている事実は特筆すべき成果と言える．大きな性能問題が発生しなかったのは，早期にアーキテクチャを決定しアプリケーションフレームワーク（共通機能）を構築したこと，その下でプロトタイプアプリケーションでベンチマークテストを実施し，最終形アプリケーションでの性能予測および性能に対するアプリケーションの特質を早期に明確化したこと，各アプリケーションの個別開発前にそれぞれの性能目標値を定義し，アプリケーション開発で継続的に性能を測定・評価・改善をしてきたことが寄与している．

段階的に性能品質を確保することで，個別機能での性能問題は性能品質 LEVEL2（単一機

能レベルの検証）までででほぼ検出され，機能テスト工程内で対応することができた．当該状況下で実施される性能品質 LEVEL3（全体評価レベルの検証）では，個別機能で大きな性能問題を発生させた機能はなく，システム全体のチューニングに注力できた．各グループ開発時の性能品質 LEVEL3 検証で，システム全体のチューニングを段階的に実施したことで，その後のグループ開発でアプリケーションが追加された場合でも，前回の LEVEL3 検証の結果と比較して性能問題の原因となっている事象を早期に発見でき，適切な対処ができた．

結果的に最終的な性能品質 LEVEL4 検証（業務品質レベルの検証）時にはチューニング済みのシステムを使用し，本番業務で想定されるさまざまな性能ピーク時間帯（発売オンライン性能ピーク時，搭乗オンライン性能ピーク時，異常運航時の性能，バッチ性能ピーク時等）の性能テストを実施することが可能となり，以後の本番業務に対するリスクはほぼ消滅した状況で本番稼働を迎えることができた．

6. お わ り に

最後に，開発途中幾多の問題に直面した際，多くの方々にご協力いただいたお陰で，無事本番稼働を迎えることができた．ProjectAI[*6] をともに進めてくださった全日本空輸株式会社関係各位，並びに ANA システムズ株式会社関係各位，プロジェクトメンバーの皆様，関連パートナー各位，および日本ユニシス関連部署の面々に深く感謝申し上げたい．

＊ 1　ネットワークキャリア：基幹空港を中心に，広範囲の市場を網羅的に取り扱う航空会社．
＊ 2　主要機能：一部機能によるテストにより最終性能評価を行うため，現行 able-D システムのピーク時の約 50％の処理トランザクション件数をカバーする機能を対象とした．（上位 9 機能）．
＊ 3　フレームワークの主要部分とはデータベースアクセス部分・全機能での共通処理等である．これにより最終形アプリケーションで，当該機能を使用する回数等を予測できれば，高い精度での性能予測が可能となる．
＊ 4　機器構成算定：必要な H/W のリソース（サーバ台数・CPU 数・メモリ数）を算出すること．構成算定は，特定の H/W 製品を仮決めして実施するが，採用製品を決定することまでは目的としていない．採用製品の決定は，必要リソース構成を組めるかということの他に，価格・ベンダーサポート等，他の事由を勘案して決める必要がある．仮決めした製品とは別の製品を採用する場合は，構成算定結果と製品性能比を勘案して実際に調達する H/W 構成を決定する．
＊ 5　移行 DB：現行 able-D システムのデータベースから移行された，本番サイズのデータベース．
＊ 6　ANACore 開発のプロジェクト名（Ana passenger system Innovation）．

執筆者紹介　高　井　健　志（Takeshi Takai）
1988 年日本ユニシス(株)入社．公共システム部門にてエアライン系のシステムサービスに従事し，主に基盤設計・開発を担当．2006 年より当システム開発に従事し現在に至る．