特集論文

次期基幹系業務システムプラットフォームへの移行に向けた取組み

板倉建太郎*
熊谷雄太*
林　和史*

Method of Migration to Next Enterprise System Platform

Kentaro Itakura, Yuta Kumagai, Kazufumi Hayashi

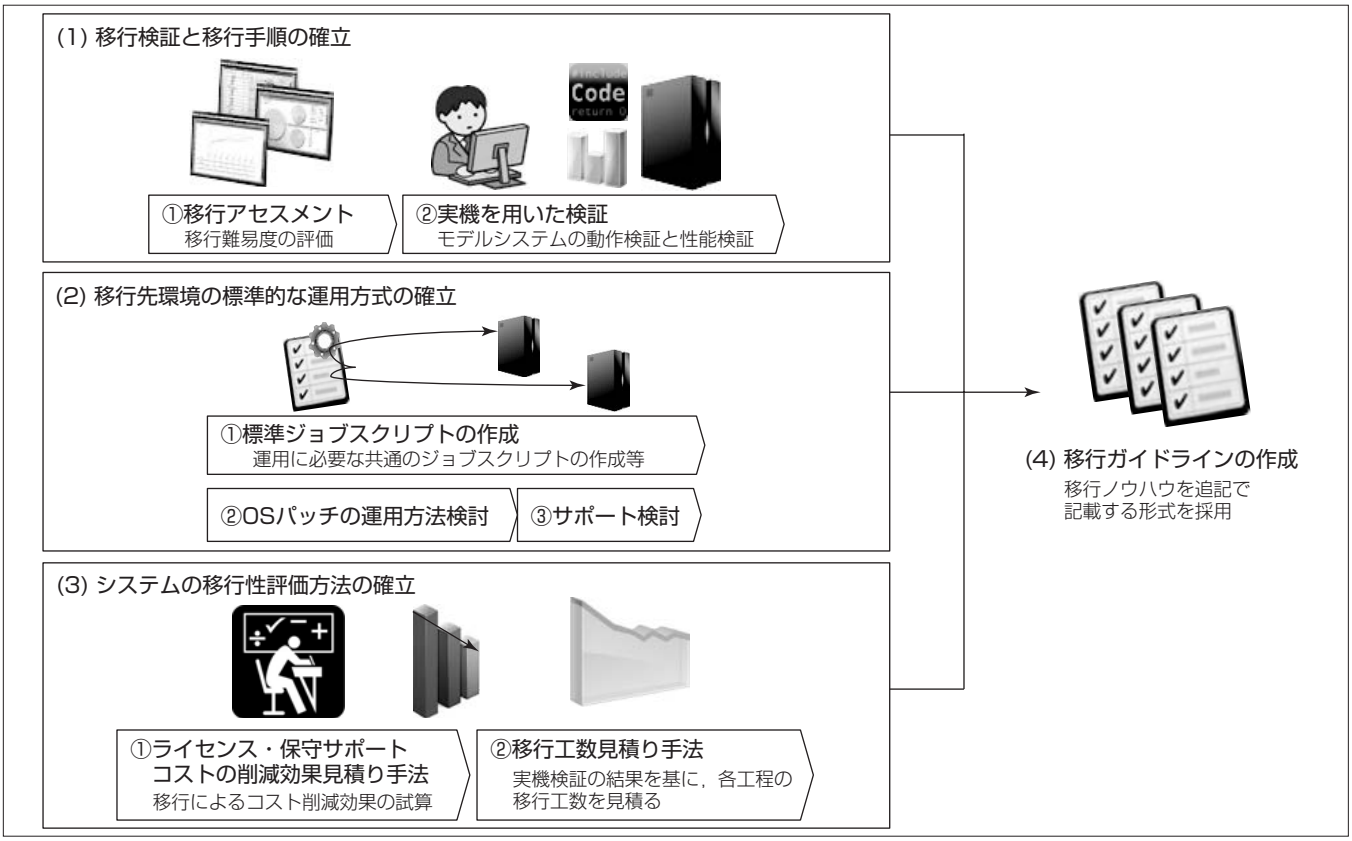
要　旨

三菱電機では，基幹系業務システムのプラットフォームにUNIX(注1)サーバを広く利用している。古くはメインフレーム(大型汎用機)を利用していたが，2000年代初頭にライセンスや保守サポート費用のコスト削減を目的としてオープン系プラットフォームへのダウンサイジングを実施した。その際，オープン系プラットフォームの中では性能や信頼性が比較的メインフレームに近いUNIXサーバを採用した。もう1つのオープン系プラットフォームであるIA(Intel(注2) Architecture)サーバは，当時は性能や信頼性が低かったが，その後改良が繰り返され，現在は基幹系やミッションクリティカルなシステムでの利用に耐え得るまで性能・信頼性が向上してきている。

現在，価格性能比の観点でUNIXサーバはIAサーバと比較して機器費用や保守料が割高であることが課題となっている。サーバを仮想化・集約することでコスト削減を図る際にも，仮想化技術・製品がOSごとに異なるUNIXサーバでは，標準化推進が阻まれる懸念がある。また，将来的なクラウド化を見据えるとクラウドではIAサーバが標準であることからも，IAサーバへ移行を検討する必要性が生じた。

そのため，実際のシステムをモデルシステムとして移行の検証を実施し，具体的な移行ノウハウを獲得して移行ガイドラインに集約し，横展開することを目指した取組みを実施した。

(注1)　UNIXは，The Open Groupの登録商標である。
(注2)　Intelは，Intel Corp. の登録商標である。



次期プラットフォームへの移行に向けた4つの取組みとその活動項目

(1)ではモデルシステムをIAサーバに移行させて移行ノウハウや課題の対策を検討した。(2)ではUNIX現行環境と移行先環境の運用面について検討した。(3)では移行工数見積り等のコストの観点で検討を行った。そこで獲得した移行ノウハウを(4)で移行ガイドラインに集約した。

*IT戦略室

1. まえがき

UNIXサーバは社内で多数の利用実績があるが，IAサーバと比較して機器費用や保守料が割高である。しかし，当社はUNIXサーバからIAサーバへの移行実績が少なく，具体的に移行を行う際のノウハウを持っていない。そのため，モデルシステムで移行作業を実践し，ノウハウ獲得や課題の洗い出しと，その対策の検討を実施した。

IAサーバに搭載するOSは，主にWindows(注3)かLinux(注4)である。LinuxはUNIXから派生したOSであるが，オープンソースソフトウェアであるため，まずはUNIXサーバと同等のサポートが得られるWindowsへの移行を検討し，実現性を見極めることにした。また，OSの移行検討に伴い，データベース製品もWindowsと親和性の高い製品への変更を併せて検討した。

移行に向けた取組みは次の4つである。

(1) 移行検証と移行の手順の確立
(2) 移行先環境の標準的な運用方式の確立
(3) システムの移行性評価方法の確立
(4) 移行ガイドラインの作成

本稿では，(1)と(4)の取組みを主体に述べる。

(注3) Windowsは，Microsoft Corp. の登録商標である。
(注4) Linuxは，Linux Torvalds氏の登録商標である。

2. 移行検証と移行手順の確立

2.1 モデルシステムの選定

今回，移行検証を実施する対象モデルシステムとして，全社的に使用されている基幹系業務システムから3つのシステムを選定した。各システムの開発言語は，Java(注5)，COBOL(Common Business Oriented Language)，UNIXシェルスクリプト，及びデータベース内のストアドプロシージャ等である。この3つのシステムの主要部分をUNIXサーバからIAサーバへ実際に移行(UNIXからWindowsへの移行と，データベース製品の変更)した。

(注5) Javaは，Oracle Corp. の登録商標である。

2.2 移行先検証環境の構築

現行環境の構成を参考に，移行先の環境(必要なサーバ数，OSのバージョン，必要なミドルウェアのバージョンやビット数等)を検討し，ソースコード変換や，動作検証，及び性能検証を実施するための移行先検証環境を構築した。

2.3 移行アセスメント

机上検証として移行アセスメントを実施し，移行先のデータベースとして選定した製品への移行難易度を評価した。そのデータベース製品に付属する移行サポートツールは，現行環境で稼働しているデータベース内のオブジェクト(表，索引，ストアドプロシージャ等)やアプリケーションが発行するSQL文を移行先のデータベース製品上で動作できる形へ自動変換する。3つのモデルシステムに対して，オブジェクトのおおむね90%は自動変換が可能(手動での書き換えが不要)，又は変換不要でそのまま移行先のデータベース製品で動作可能という結果となり，データベース製品の移行作業は少ない工数で実施可能という結果となった。

2.4 実機を用いた検証

実機を用いた検証では，実際にIAサーバ上へモデルシステムの一部の機能を移行してバッチジョブやオンラインサービスが正しく動作するかを検証する“動作検証”と，レスポンスや処理時間が適正かを検証する“性能検証”を実施した。

2.4.1 ソースコード変換

Java，COBOL，データベース内のストアドプロシージャ，UNIXシェルスクリプト等を，Windows及び移行先のデータベース製品上で動作できる形へ実際に変換した。Java，COBOLの修正箇所はソースコードに埋め込まれているSQL文とSQL呼び出し部分，及びデータベース製品変更に伴う接続定義部分を手動で書き換えるにとどまり，それらを除いたソースコード本体の修正は発生しないことが分かった。

また，データベース内のストアドプロシージャは，先に述べた移行サポートツールを利用することで約90%を自動変換でき，変換工数を抑制した。

一方，UNIXシェルスクリプトは，Windows上で同等の機能を提供するWindows Power Shellへの変換が必要となるが，この変換をサポートするツールは存在しないため全て手動で書き換える必要があった。これはスクリプトの量によっては，移行工数を大きく増加させてしまうため，課題として残ることになった。

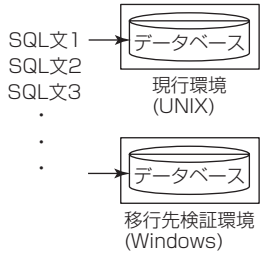
2.4.2 動作検証

変換したソースコードをコンパイルし，構築した移行先検証環境へ配置して動作検証を実施した。バッチジョブの動作検証は，現行環境と移行先開発環境の双方に同じインプットデータを使用してバッチジョブを実行し，出力したアウトプットデータが同じであることを確認した。オンラインサービスでは，通常のアプリケーション開発手順と同様に動作検証用シナリオを実行した。エラーが発生した場合は原因を調査し，対象部分を修正するという手順を踏む。今回は，約60件のエラーが発生したが，致命的なものはなく，全て修正して検証対象のバッチジョブとオンラインサービスが正しく動作することを確認した。主にプラットフォームの相違によるものがエラーの原因であったが，その詳細については後に述べる。

2.4.3 性能検証

バッチジョブの性能検証は，移行先での処理時間が現行環境の処理時間とほぼ同等かそれ以下となれば，性能的に問題がないことを確認できる。ただし，現行環境と移行先検証環境のハードウェアスペックが異なっている今回のケ

同じデータベース製品(同じバージョン)を双方のサーバーに導入し，基本性能を取得する複数のSQL文を双方のサーバーで実行し，処理時間を計測する。



処理結果(秒)	SQL文1	SQL文2	・・・	平均
現行環境(UNIX)	15	6	・・・	15
移行先検証環境(Windows)	21	5	・・・	10

この例では，スペック差係数は1.5(15/10)となる。
移行先環境での処理時間を1.5倍した値と現行環境での処理時間を比較することで，性能検証を行う。

図1．ハードウェアスペック差の係数の算出

ースでは，ハードウェアスペックの差を吸収する数値(係数)が必要となる。そこで，現行環境と移行先検証環境の双方の環境上で同じデータベースで同じ処理を数パターン実行させ，処理時間平均の差から係数を算出した(**図1**)。この係数を移行先での処理時間に適用することでハードウェアスペックの違いを吸収し，性能検証を実施した。

バッチジョブの性能検証の結果，約40%は現行環境と同等かそれ以上の性能を確認できた。残りの60%はチューニングを実施することで性能が改善し，同等の要件を満たすことを確認した。ただし，本番システムの移行時には，現行環境よりも高いスペックの最新サーバへ置き換えることで，チューニングが必要なバッチは60%よりも少なくなると予想している。

次に，オンラインサービスの性能検証は，負荷テストツールを利用して現行環境と移行先検証環境で同じ条件(同時アクセスユーザー数など)・同じシナリオで実行した。レスポンスタイムを測定した結果，1つのシナリオだけチューニングによる性能改善を必要としたが実施ハードウェアスペックの差を考慮しても性能的に問題がないという結果を得た。

これらの結果から，IAサーバへ移行する際，チューニングによる性能改善検討が必要な処理は，バッチジョブのうち多くても60%程度，オンラインサービスは0%と判断した。この割合(%)は，今後，実際に移行した際の実績値を加えることで精度を高めていく。

3. 検証結果を活用した移行ガイドラインの作成

移行アセスメントと実機を用いた検証結果を基に，移行ガイドラインを作成した。移行ガイドラインの内容は，移行工数の見積りに使う“見積り指標”と，実際に移行を実施する際に利用できる“移行手順書”の大きく2つから成る。

3.1 見積り指標の作成

ソースコード変換，動作検証，及び性能検証にかかった工数の実績値をベースとして，各工程の工数見積りの計算式を作成した。例えば，ソースコード変換では，各ソースの種類(Java，COBOL，UNIXシェルスクリプト)ごとに**表1**の指標を計測し，最終的にソースコード1,000行当たりの変換工数(ソースコード変換工数レート)を見積もる計算式を導出をした。

ただし，この“変換工数”にはエラーや不具合対応の工数が含まれており，同じエラーを2回目以降に対応する場合，その工数は初めて対応する工数に比較して大幅に削減できると考えられる。つまり，移行のノウハウを蓄積することで，ソースコード変換工数レートも小さくなる。

表1．ソースコード変換での見積り指標

見積り指標	内容
変換工数	ソースコードの変換にかかった工数
ソースコード行数	現行環境のソースコード総行数
ソースコード有効行数	ソースコード行数から“空白行”と“コメント行”を除外した行数
ソースコード変換工数レート(1,000行当たり)	(“変換工数”/“ソースコード有効日行数”)×1,000行

表2．エラーや不具合の分類

分類	エラーや不具合の原因
分類A	OSやデータベース製品変更によって，修正が必要となったもの
分類B	アプリケーション構造を把握していなかったことによるもの (アプリケーションに詳しい担当者であればすぐに対応できたもの) 例：特定機能から警告メッセージが出力されるが，これは現在使用されていない機能のため，調査不要など
分類C	設定内容の記載ミスや設定ファイルの不足等
分類D	今回の検証環境だけに依存するもの

表3．ノウハウ蓄積率を反映したソースコード変換工数レートの計算式

ソースコードの種類	1,000行当たりの変換工数レート(単位：時間)
ストアドプロシージャ	$1.76x + 4.15(1-x)$
UNIXシェルスクリプト	$18.4x + 20.8(1-x)$

x：ノウハウ蓄積率(0～100%)

3.2 ノウハウ蓄積率を考慮した見積り指標

実施要員のスキルレベルやノウハウの保持具合を“ノウハウ蓄積率：x”として，これを考慮した変換工数レートを算出するため，次の作業を実施した。まず，ソースコード変換，動作検証，及び性能検証の各工程で発生したエラーや不具合を**表2**のように分類し，その分類ごとに対応工数を集計した。

例えば，動作検証では約60件のエラーや不具合が発生したが，その対応工数を**表2**に従って分類した。その結果，分類Aが56%を占め，分類Bが22%，分類Cが21%，分類Dが1%という結果となった。分類Aは移行ノウハウが蓄積されることで，ある程度の工数削減が可能であり，分類Bも体制のフォローアップ(アプリケーションに詳しい担当者がサポートする等)によって工数削減が可能と考えられる。一方，分類Cはどのような場合でもある程度は発生する可能性がある。これらの対応工数を除いた工数の組み合わせで，ノウハウ蓄積率を考慮した変換工数レートの計算式を**表3**に示す。例えば，ノウハウ蓄積率が60%の場合，

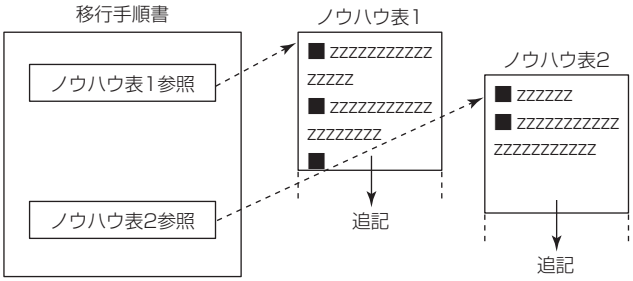


図２．移行手順書の構成

ストアドプロシージャを変換する工数レートは，2.716（＝1.76×0.6＋4.15×（1－0.6））時間となる。

同様に，動作検証と性能検証についても，ノウハウ蓄積率を反映した形で工数レートの計算式を導出し，各工程の工数を精度高く見積もれるようにした。

3.3　移行手順書の作成

今回の移行検証で得たノウハウは移行手順書という形でドキュメント化した。その際，全ての実施項目を手順書の中に記載してしまうと，実際の移行作業のたびに新たに得たノウハウを手順書に追記する必要が発生する。そこで，手順書本体とは別に，ノウハウを追記型で記載する形式でまとめることで，手順書を変更することなくノウハウを蓄積することを可能にした（**図２**）。

4. 推進上の課題とその対応

今回の移行検証で，機能や性能面では移行が可能であることを確認できた。ただし，UNIXシェルスクリプトは移行を支援するツールが存在しないため，全て手動での書き換えが必要であり，スクリプトの量によっては移行工数を大きく増加させてしまう可能性がある。この課題に対して，次の２つのアプローチで対策を検討した。

⑴　代替ソフトウェアの活用

Windows上でUNIXシェルスクリプトを動作させることができるソフトウェアの活用を検討した。フリーソフトウェアCygwin等の複数の候補があるが，ソフトウェア自体が異常終了した際にその上で稼働する全てのシェルスクリプトが異常終了する事態となる。これは現行環境よりリスクが高くなると判断し，不採用とした。

⑵　標準化したジョブスクリプトの活用

移行先のWindows，データベース製品の運用に必要なジョブスクリプト（バックアップ，ログ圧縮など）をあらかじめ標準化して作成しておき，移行時にそれらの標準ジョブスクリプトを活用することで，書き換え対象のシェルスクリプトを減らして移行工数の増加を抑制する。これによって，OSやミドルウェア関連のシェルスクリプトの書き換え工数削減を図ることができたが，アプリケーション関連のシェルスクリプトについてはアプリケーション固有の要素が多く標準化できないため，やはり手動での書き換えが必要であった。

5. 運用局面での検討事項

今回のプロジェクトでは，モデルシステムを使用した移行検証が中心であるが，運用するサーバがUNIXからIAサーバへ変更するに当たり他にも検討すべき項目がある。

その１つとして，定期的にOSやミドルウェアにパッチを適用するという検討事項がある。Windowsは，UNIXと比較してOSのパッチ適用を頻繁に実施しなければならないが，過去の一定期間のパッチ公開履歴を調査して適用の要否を検討した結果，ブラウザなど基本的にサーバでは使用しないソフトウェアのパッチ適用は省略可能と判断した。また，待機系のサーバを用意し，適用時に待機系のサーバへ切換えを併用することで，パッチ適用回数を年数回程度に抑え，かつ適用によるサービス停止時間も数十秒程度に抑えることが可能であると判断した。

6. む　す　び

UNIXサーバのOSであるUNIXからIAサーバのOSの１つであるWindowsへの移行と，データベース製品の変更が可能であることを確認した。しかし，UNIXシェルスクリプトの量によっては移行工数の削減が難しいことが分かった。そのため，新規システムの構築については，コストパフォーマンスの高いIAサーバ（Windows）を検討するが，既存システムの移行については数年に一度のハードウェア更改や，アプリケーションの大規模改修等に合わせて実施を検討する。その際，Windowsへの移行によるコスト削減が見込めない場合は，IAサーバのもう１つの選択肢であり，UNIXシェルスクリプトをあまり書き換えずに動作させることが可能なLinuxへの移行を検討する。社内ではLinuxへの移行検討も実施しており，最終的にはIAサーバ（Windows，Linux）への移行ガイドラインとして当社グループで活用していく予定である。

参　考　文　献

⑴　下出聖子，ほか：汎用コンピュータからオープン環境への基幹系システム全面移行，三菱電機技報，**82**，No.10，666～669（2008）