開発技術と開発プロセス

年金管理システム開発における パフォーマンス改善プロセス

金融システムビジネスユニット
年金業務ソリューション第二構築センター
岡 村 智 幸

1．はじめに

今回の年金管理システム開発では、開発過程で、オンライン部分、バッチ部分双方にパフォーマンス問題が発生した。これらには、純粋に技術的な問題に起因するものと、プロジェクトの推進プロセスに起因するものとがあったが、最終的には全体で当初の10分の１程度に処理時間を短縮し、無事にカットオーバーを迎えることができた。パフォーマンス問題が発生したこと自体は決して自慢できないが、得られた知見も多いのでレポートにまとめ発表することとした。

2．問題の発覚時点と深刻さ

今回のプロジェクトでは、総合テスト当初からパフォーマンス問題が発覚していたが、本格的に対応を始めたのはカットオーバー予定の約４ヵ月前で、機能テストの終盤段階であった。月次で処理の重い日には最大で100時間程度の処理時間を要するケースが想定され、改善できなければカットオーバーは無理という状態であった。オンライン画面では、画面ユーザーが多いわけではなく、エントリーDBを利用しているため総トランザクション処理量が問題となるケースはないものの、画面遷移で非常に時間がかかる場合があり、重い画面では30秒待たないと次の画面に遷移しないことが判明した。これについてもそのままではカットオーバーを延期せざるを得ないといった状況であった。

3．改善結果

現時点では、通常処理日において夜間バッチ全体で２時間程度、月次で処理の非常に重い日でも６時間程度（いずれもバックアップ処理などを含む時間）になっており、メインフレーム時代よりむしろ速いという評価を得るまでに改善している。また、オンライン処理については、画面遷移の時間で最大約８秒と若干反応が遅いと言わざるを得ないケースが残るものの、運用に耐えるパフォーマンスの確保に成功している。

4．解決までのプロセス

問題の深刻さから考えても、パフォーマンス問題はカットオーバーを左右する重大な問題であり、住友信託銀行殿とも協議の上、専任の解決チームを編成することとし、筆者がCAC側のマネジメントを担当することとした。具体的な施策としてただちに実施したのは以下の３点である。

・数値目標の設定
・パフォーマンスチューニングの専門家のアサイン
・優先順位の設定

この３点の施策はパフォーマンスチューニングの推進の観点から過不足のないものであったと考えているが、この段階に至る前に、アーキテクチャ確定の段階で事前に問題分析ができていれば、未然に問題発生は防止できたと思う。以下に、３つの施策について個々に詳細を述べることとする。

4.1　数値目標の設定

改善プロセスのスタートは具体的で客観的な目標を設定することであった。ホストコンピュータで稼動している旧システムの処理件数、処理時間を調査し、性能目標値を１秒100件と割り出してこれを当初の目標値とした。次に、統合テストや総合テストの結果を用いて各ジョブの実績での処理能力と目標値との乖離率を算出した。まずは、乖離

が大きい（１秒あたりの処理件数が少ない）もので、なおかつ業務的要件が高いものから優先的に検討に入った。改善当初はほとんどのジョブの乖離率が500から1000といったレベルであった（乖離率100は、１秒に１件しか処理できないということ）。

4.2　パフォーマンスチューニングの専門家のアサイン

改善を進めるにあたり、まずチューニングの専門家をアサインした。

具体的には、DBやWindowsのスキルに長けたメンバーに加えて、MCS （Microsoft Consulting Service）のコンサルティング担当者を対象とした。これらの専門家と設計担当者の合同による改善対応策の検討会を実施した。

まずは、検討対象のプログラムの処理の構造について、設計者から設計書をもとに説明させる。この説明とプロファイリングツールの出力結果を照らし合わせてボトルネック分析を開始する。処理の構造から考えても不自然なモジュールの呼び出し回数、処理件数に対してありえない処理時間などが浮き彫りにされ、ボトルネックが次第に特定されていく。しかしボトルネックは１つとは限らない。発見しては実験し、さらに分析する、というプロセスを何度も繰り返していくうちに、処理の根幹にかかわる問題が明らかになっていく。このような作業を１ヵ月ほど続けていくうちに、プログラムに関する原因はほぼすべてが特定できた。

しかし、帳票出力の処理部分についてはチューニングの専門家でも原因が特定できずにいた。これについては、帳票作成ツールのベンダーであるウイングアーク テクノロジーズとサポート契約を結び、ボトルネックとなっているジョブのデバッグログをもとに分析と改善策の提案を依頼した。この効果はすぐに現われ30%～50%の改善が可能になった（図１）。

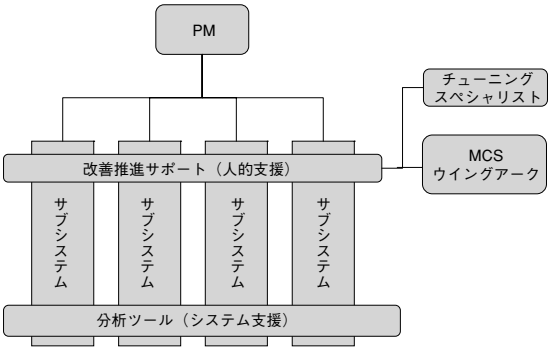

図1　パフォーマンス改善の体制

4.3　優先順位の設定

年金管理システムは、VB.NETプログラムの総ステップが300万行以上あり、ジョブの数にして400以上である。これらすべてのジョブを手当たり次第に改善することは、非効率であるだけでなく時間も足りなかった。

そこで、ジョブの優先順位付けを設定した。まず、エンドユーザーにお願いし、すべてのジョブの中からエンドユーザーから見て重要と思われるものをピックアップしてもらった。この中から、開発側の視点で処理タイプの網羅性を考慮し20ジョブを選び出した。これら20ジョブについて専門家を交えた会議を重ね改善策を打ち出し、改善を実施した。20ジョブの改善を開始して２ヵ月ほど経過した時点で乖離率が５～10に落ち着き、このタイミングで全ジョブへの展開を行った。結果的には、最初の20ジョブで検討した改善策を、残りの数百を超えるジョブにそのまま適用することができた。

５．主な改善点

1）クラスの生成タイミングと生成回数

オブジェクト指向プログラミングでは、必要な時点でオブジェクトを生成し、不要になったら破棄するというのが一般的である。しかし、オブジェクトの生成自体にコストのかかるクラスもあり、こういったオブジェクトの生成と破棄をループ処理にて繰り返すことが、パフォーマンスに大きく影響を及ぼしているケースが多数あった。これらのケースでは、オブジェクトの生成タイミングを見直し、オブジェクトの生成回数を極力押さえるような方式による改善作業を展開した。この方式での改善効果は大きく、これだけで処理時間が20%～30%改善された。

2）共通部品の呼び出し回数過多

システム共通部品として日付や文字を扱う小さな部品があるが、こういった部品の呼び出しに負荷がかかっているケースがあった。これについては、呼出回数を抑えるような対策とともに、共通部品側の処理も軽くするような対策を施した。

3）環境設定の不備

帳票作成ツールであるSVFは、ツールのオプションでデバッグログを吐くことができるのだが、開発期間中このログを最大限に出力していたことがわかった。その設定のままシステムテストを続けていた。これを抑止することで、帳票作成処理すべてが50%以上スピードアップした。

4）委託者分割による並列処理

委託者を分割して並列実行する仕組みは本システムのバッチ管理アーキテクチャの特徴であるが、これを利用せず、全委託者について直列処理を行っているために処理時間の増大を招いているケースがあった。こういったジョブについては、サーバーを有効に活用するため委託者分割および委託者内分割を行うよう修正を施した。

5）DBアクセスクラスの不要なコーディング

VS.NETのコンポーネントデザイナを使用して作成したDBアクセス処理には、データベースとDataSetの間の整合性を保つため、データベースの更新後に再度データベースを読み込む処理が含まれている場合がある。この再読込がパフォーマンスを劣化させているケースがあり、再読込が不要なDBアクセス処理については、これを除外することによりパフォーマンスの改善を行った。

6）SOAPフォーマッタによるシリアライズとデシリアライズ

エントリオブジェクト等をDBへ保存、あるいはDBから取り出す際に使用するSOAPフォーマッタによるシリアライズとデシリアライズの処理が予想以上に重く、パフォーマンスに影響を与えていた。これについては、フォーマッティングの処理を見直すことにより改善を行った。

6．今回の活動で得られた経験

1）運用要件から目標値を導出し改善を推進した

まず、運用要件から具体的に1秒100件処理という明確な目標値を立てた。さらに、目標達成のために、何段階かのステップを経てゴールに向かう道筋を示し、これを制作メンバーに展開した。このように運用要件から目標を設定すると、ユーザーから見て改善状況が具体的に理解でき、制作メンバーにとってもまだどれぐらい努力すべきかの量的な感覚がつかめて、進捗状況報告や改善推進には効果的であった（図2）。

2）客観的数値をもとに分析できるツールを利用した

ボトルネックを探るためのプロファイリングツールを導入することで、客観的な数値をもとに分析が可能になった。客観的な数値を用いることにより、具体的な乖離率などの考え方が導入できたことは有効であった（次ページ図3）。

3）外部の支援をうまく利用できた

MCSやウイングアーク テクノロジーズなどの外部コンサルタントや製品ベンダーの支援を活用した。チューニングの専門家だけでは解決できないOS、DBMS、帳票作成ツールにかかわる部分の外部のサポートを、プロジェクトに展開することができた。

4）設計者のアーキテクチャの十分な理解が品質を向上させる

設計者の十分なアーキテクチャ理解と、パフォーマンスの観点からの考察が、プロジェクト全体の品質向上に貢献し、開発コストも下がる。言うまでもなく下流工程での大幅な見直しは、莫大な修正コストが発生するだけでなくデグレードの可能性も高くなる一方である。

5）制作フェーズ・コーディングサンプルの充実とコード

改善作業工程					
作業段階	第0 STEP 事前準備	第1 STEP コーディング作業	第2 STEP 設計見直し作業（1）	第3 STEP 設計見直し作業（2）	第4 STEP 検証作業
作業の概要	ボトルネック の検出	現在までに横展開した作業	ロジックの修正が少なくても 効果のある改善作業	大幅なロジックの修正が 必要とされる改善作業	次の展開に進めるかどうか精査する
乖離度 (100件/1秒の何倍か)	—	100以上	100～50	50～5	5以下
作業内容の詳細	プロファイリング	・プログラムロジックの修正 ・共通クラスの修正 （共通クラス自体の修正 利用者側の修正） ・業務セットの修正 （業務セット自体の修正 利用者側の修正） ・ADO.NETの使い方の見直し	・簡易な設計の見直し ・Select文の見直し	・大幅な設計の見直し ・DBアクセスロジックの見直し ・共通処理の切り出し	・データの角度・精度 ・データ量 ・自動トランザクション ・ジョブネット
作業後に期待 できる乖離度	—	100～10	10以下	5以下	調整が必要なものはさらなる改善を行なう
対応する要員 のスキル	特別なスキルは 不要	基本的なプログラミング	高度なプログラミング	高度な設計能力とプログラミング	業務知識と運用面の検討もできる人材
工数	小	小	小～中	大小	

図2　改善作業工程

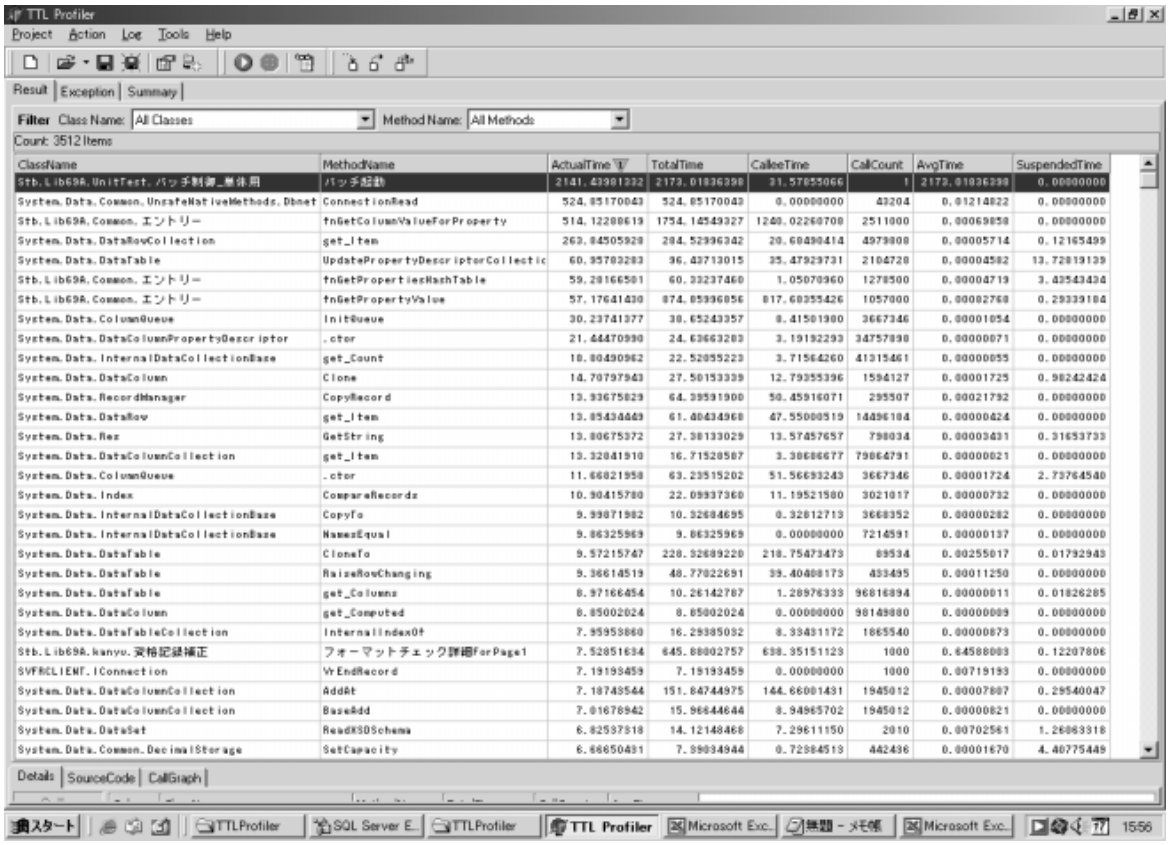

図3　プロファイリングツールの出力イメージ

チェックの徹底

これは、品質保証の基本的な手順ではあるが、事前に準備されたコーディングサンプルの充実と、制作フェーズにおける有識者を交えたコードインスペクションが、改めて必要だと思われる。

6）早期のプロトタイプでの検証を十分に実施する

新技術を導入するプロジェクトでは、初期段階のプロトタイプ検証は必須である。本プロジェクトでもプロトタイプでの検証は実施したが、大量データを用いたパフォーマンス検証までは十分ではなかった。

7）設計不良が発覚した場合は大きな決断を

不幸にしてパフォーマンスの観点からの大きな設計不良がわかった場合、早期に作り直しを検討する。大幅な作り直しはリスクを伴いコストも増大するのだが、中途半端な対応による改善ではその効果に限界がある。経験から言っても、早期に大幅な修正を行ったジョブには期待以上の改善結果が出ており、中途半端な対応しかできなかったジョブでは必ず改善の壁に突き当たった。

8）推進チームは権限をできるだけ強化する

改善推進チームの持つ権限は強力でなければならない。問題が大きなジョブについては、強制力をもってシステムテストのスケジュールを見直してでも改善作業をさせるべきである。せっかく専門家を入れ改善策が判明しても、対応を徹底できなければ意味がない。１ジョブで見るとわずかな改善でも、プロジェクト全体が改善された場合、バッチ全体を通して見ると大きな効果が得られる。

７．終わりに

第２次リリースを2004年１月に終え、半年に及ぶパフォーマンス改善のサブプロジェクトは、いったん終了した。しかしながら、カットオーバー後に管理対象者が順調に増加していることもあり、パフォーマンス改善要求は再び高まっている。今後も、今回と同様の手法を採用できると考える。今回実施した推進プロセスが完全なものだったとは思っていないが、大規模プロジェクトでかつ下流工程で発生した問題の改善事例が、今後のプロジェクト開発の参考になれば幸いである。