メインフレームだからできる「CPU のムダとり」手法

株式会社アイビスインターナショナル

■ 執筆者Ｐｒｏｆｉｌｅ ■

有賀　光浩

1985 年　富士通株式会社入社
2003 年　富士通株式会社退職
2004 年　株式会社アイビスインターナショナル
　　設立　代表取締役

■ 論文要旨 ■

GS 初である「CPU のムダとり」手法を確立し，お客様の本番システムで効果をあげている事例を紹介する．

高信頼が強みである GS 上でも性能品質の悪化が進んでいる．運用中のシステムに性能品質を作り込んでいくには，リプレース時の①性能予測，運用時の②GS メタボ診断，③CPU のムダとりの３点がポイントである．

CPU のムダとりは 12 項目からなる性能評価プロセスに従って実施する．改善のターゲットはユーザアプリであり，改善作業はお客様が行ない，私どもは支援をする．工夫点は，①CPU のムダが一目で見える仕組み（CPU/IO 頻度分析），②CPU 時間の削減予想，③プログラム内の問題を見つけ出す手法（詳細分析＋ヒアリング），④お客様が改善したくなる場作りである．ムダは想像以上にたくさん存在している．

CPU のムダに関するリテラシーを高め，オープンシステムにも展開していきたい．

■ 論文目次 ■

■ 図表一覧 ■

１．はじめに

１．１　当社の概要

株式会社アイビスインターナショナル（所在地：東京都）は，富士通メインフレーム（GS21，PRIMEFORCE）の性能コンサルティングを事業としている（ホームページ：http://www.ibisinc.co.jp/）．

代表の有賀光浩(Ariga Mitsuhiro)は富士通株式会社で 18 年間 SE として活躍，1992 年からの 11 年間は共通技術部門でメインフレームの性能に関する技術支援，顧客システムの性能トラブル対応を担当した．対応システム数は国内外合わせて 1,000 以上に及ぶ．

富士通退職後，2004 年に株式会社アイビスインターナショナルを設立．お客様が満足感を持ち，安心してメインフレームを使い続けて頂けるためのコンサルティングを提供している．

１．２　背景

（１）メインフレームの国内市場

(社)電子情報技術産業協会(JEITA)の調査によると，2009 年度のメインフレームの出荷台数は 562 台（前年度比 94%），金額は 1,186 億円（前年度比 99%）であった．オープンサーバ全体の出荷台数 33 万台（前年度比 96%），金額 3,168 億円（前年度比 78%）と比べると未だに検討していることがわかる．メインフレームの稼動台数は公表されていないが，出荷実績から算出すると 2009 年度は約 4,700 台と予想する．

2010 年のトピックスとしては，1 月に富士通が GS21 1600 と 1400 の出荷を開始，7 月に IBM が zEnterprise System を発表している．

（２）富士通メインフレームの動向

2010 年 5 月に開催された FUJITSU FORUM 2010 で，富士通メインフレーム（以降，GS と略す）の次期エンハンスは 2014 年頃との説明があった．

興味深いサービスに「CPU 能力変動型アウトソーシングサービス」がある．これは，お客様の要求に応じて 1 年ごとに CPU 能力を UP/DOWN できる画期的なサービスであるが，実際に運用することは想像以上にむずかしい．図 1 に示すように，性能条件と性能目標を明確にした上で情報システムを掌握し，お客様自身が性能をコントロールしながら CPU 能力を調整する必要がある．

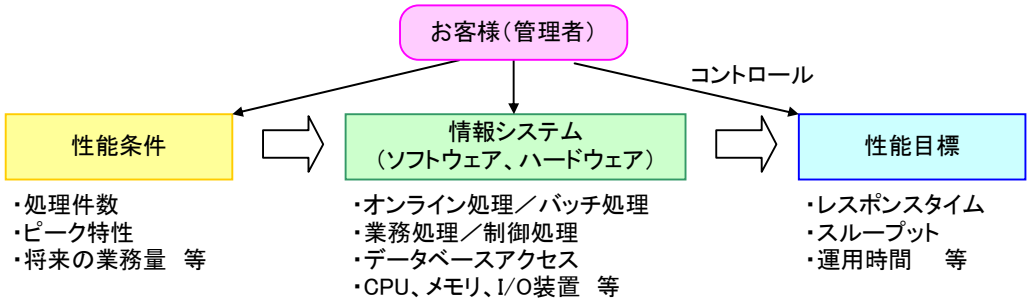


図１　性能をコントロールする

参考までに，GS でのシステム統合（CPU 能力 UP）や CPU ダウングレード（CPU 能力 DOWN）の考察は，図２に示すように 2006 年（★４，参考文献４）と 2009 年（★７，参考文献７）の FUJTISU ファミリ会論文に投稿している．

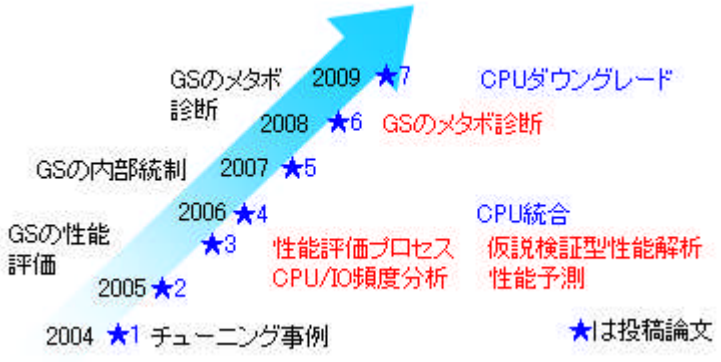


図２　FUJITSU ファミリ会投稿論文

（３）GS は汎用機にあらず

弊社で最近対応した GS システムについて，実メモリ量と DISK 容量を集計した結果を図３に示す．（AVM 使用時は代表的な仮想システムを一つ抽出）

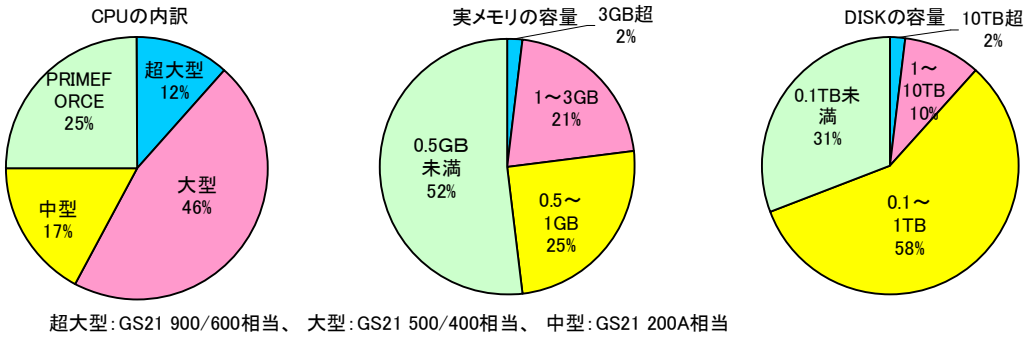


図３　GS のメモリ量と DISK 容量

図左は CPU の大きさで集計しているが，市場での分布と大きな偏りはないと考える．図中央は搭載している実メモリ量を示している．１GB 以上が全体の 23%しかなく，半数以上が 0.5GB 未満である．図右は DISK 容量を示している．１TB 以上は全体の 12%のみで，100GB 未満が 31%もある．

即ち，GS21 500 クラスの大型機であっても，標準的なシステム構成は実メモリ量が１GB 程度，DISK 容量は数 100GB である．メモリや DISK の容量は，あなたが使っている PC とほとんど変わらないのが実態である．

このように一台の小さな GS 上で，信頼性の高いデータベースシステムを構築し，数百ものオンラインプログラムが動作し，並行してバッチ処理が多重で実行されている．これを支えているのが，OS とミドルウェアであり，その規約に準拠して COBOL でコーディングされている高品質なユーザプログラムである．

GS は汎用機とも呼ばれていたが，もはや選ばれた基幹システム専用のコンピュータであるといえる．

２．性能品質の作り込み

2006 年の論文（図２★３，参考文献３）で，GS システム上の性能品質が悪化している現状について警告をした．性能品質が悪いとは，ある機能を実現するのに必要以上のリソース（特に CPU）を使っていることを意味する．どうしてこのような現象が表面化してきたのか考察をする．

２．１　理想的な性能品質の作り込み

性能品質を作り込むためには，システムのライフサイクルにおいて図４に示すような取組みが必要である．これは GS・オープン共通の考え方である．

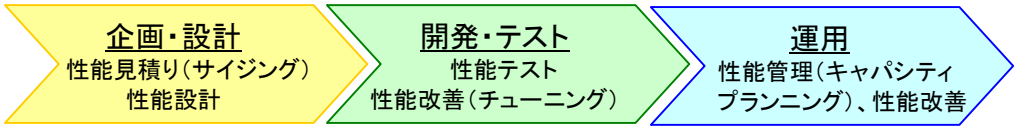


図４　性能品質の作り込み（理想）

GS システムにおいては，図４の性能見積り～性能管理等が積極的に行われなかったのが現実である．それにもかかわらず，性能問題が表面化しにくかったのは，最初の導入時に AIM やデータベース設計がしっかりしていたことと図５に示すように CPU 能力に十分余裕があったことが理由と考えられる．

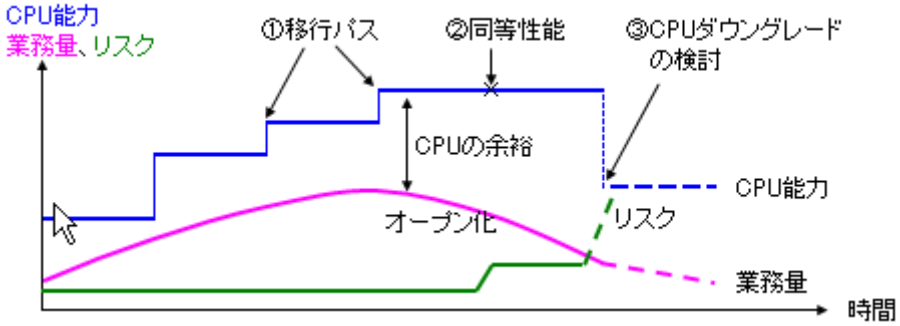


図５　GS の CPU 能力，業務量，リスクの関係

CPU に余裕ができたのは，メーカが推奨している CPU への移行を繰り返したためである（①移行パスと言う）．これにより，「性能管理レス」での運用が可能となり，手間のかかる性能見積り（サイジング）も結果的に不要になった．

GS 上の業務のオープン化が進み，コスト削減の要請が強まった結果，CPU 能力が適正なのか問題視されるようになってきた．直近の CPU リプレース（図５の②）は同等性能の機種を採用しても，次のリプレース③では CPU のダウングレードか GS の撤去が決行される流れである．

②以降は「性能管理レス」で運用してきた前提条件が崩れたため，性能問題が表面化するリスクが高まっている．リスクを低減するためには，適切な性能管理をするとともに，性能品質の悪いプログラムはあらかじめ対処することが望ましい．最近では，内部統制やセキュリティ強化の対応も必要となっている．

２．２　新・GS 性能品質の作り込み

運用中のシステムに性能品質を作り込んでいくには図６のような対応が効果的である．理想（図４）との比較を表１に示す．ポイントは性能予測，GS メタボ診断，CPU のムダとりの３点である．

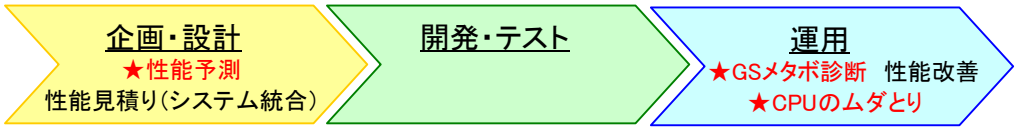


図６　新・GS 性能品質の作り込み

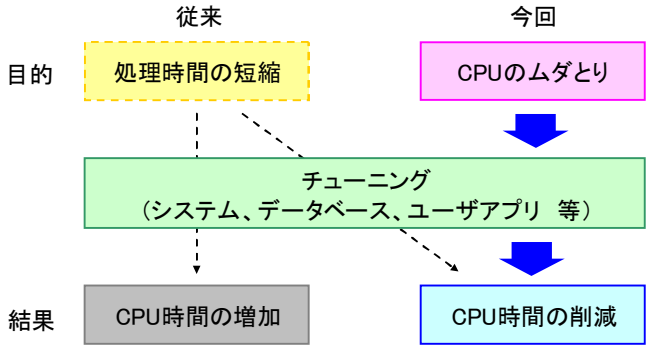
表１　新・GS 性能品質の作り込みと理想の比較

フェーズ	理想（図４）	GS の実態	新・GS 性能品質の作り込み（図６）
企画・開発	性能見積り（サイジング）	移行パス（実施せず）	移行パスが採用されなかたときの対応 単純移行時： CPU 能力 UP/DOWN 時の性能予測 システム統合時： 性能見積りと性能予測を合わせて実施
	性能設計	実施せず	（個別対応）
開発・テスト	性能テスト	実施せず	（個別対応）
	性能改善	実施せず	（個別対応）
運用	性能管理(CP)	移行パス（実施せず）	従来のキャパシティプランニング(CP)では効果が期待できない GS メタボ診断でリスクを見える化する
	性能改善	トラブル発生時に実施	リスクを低減させる改善を実施 CPU のムダとりを実施

注「実施せず」は，多くの GS ユーザでは実施されていないことを意味している．
一部の大規模システムでは当然厳格に実施されている．

今まで，CPU のムダとりについて議論されたことはなかったと思う．図７に示すように，処理時間の短縮のためのチューニング（SQL 文の見直し等）を行い，結果として CPU 時間が削減されるようなことはあった．逆に CPU 時間が増加するケースも珍しくない．

今回は，CPU のムダとりが目的である．目標値を設定し，チューニングをして CPU 時間を削減するものであり，ユーザアプリケーションの改善が中心となる．



注　処理時間の短縮のためにデータベースをメモリ常駐すれば CPU 時間は増加する．

図７　CPU 時間の削減の違い

３．CPU のムダとり手法

ユーザ業務に関する専門知識が無くても，短期間で実現できる CPU のムダとり手法について説明をする．

３．１　性能評価プロセス

性能データの数字を拾って EXCEL でグラフを作り，時代遅れの指標値と比較してコメントをつけて体裁を整える．これは性能データの集計であり，性能評価ではない．

性能評価の基本となるプロセスを図８に示す．これは 2006 年の論文（図２★３，参考文献３）で発表し，ブラシュアップしたものである．重要な点を以下に示す．

・性能評価の目的と Output を明確する．性能評価すること自体を目的化しない．
・使用する性能データ(Input)を選定し，性能評価プロセス(Process)を調整する．
・Input と Process に想像力や柔軟性がないと，Output と目的が乖離する．
・特定の性能データを信用せず，２つ以上のデータで相互チェックする．

CPU のムダとりには，②見える化，④原因分析（詳細分析），⑦処理時間分析，⑩改善（チューニング）がポイントとなる．

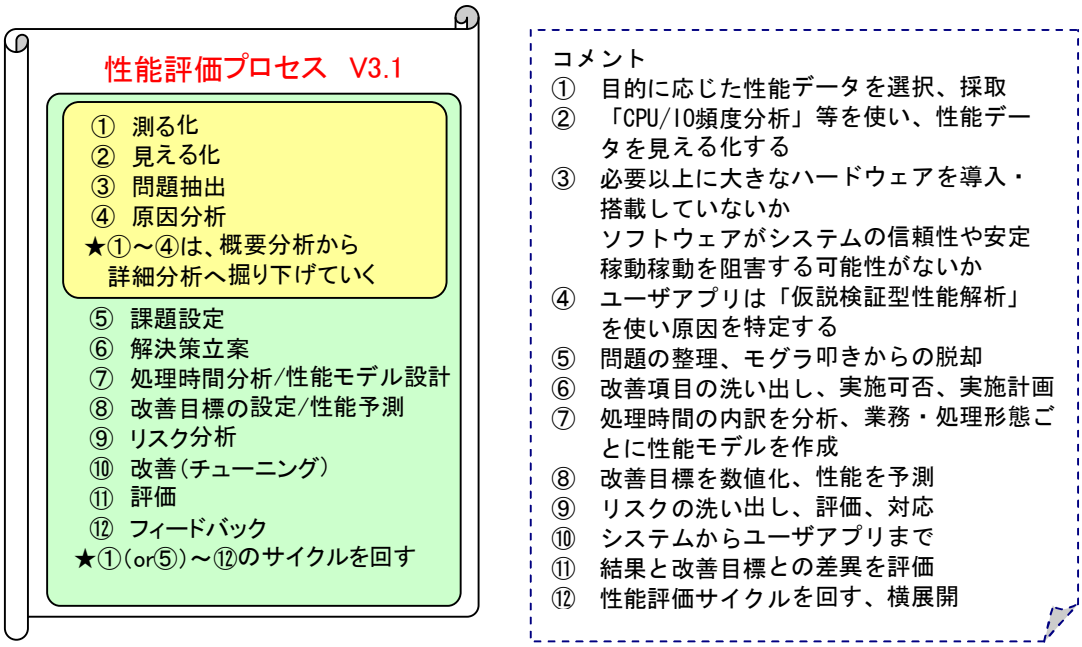


図８　性能評価プロセス V3.1

３．２　CPU のムダとりのステップ（見える化）

（１）CPU のムダの見える化（システム）

見える化の核となる技術は CPU/IO 頻度分析である．

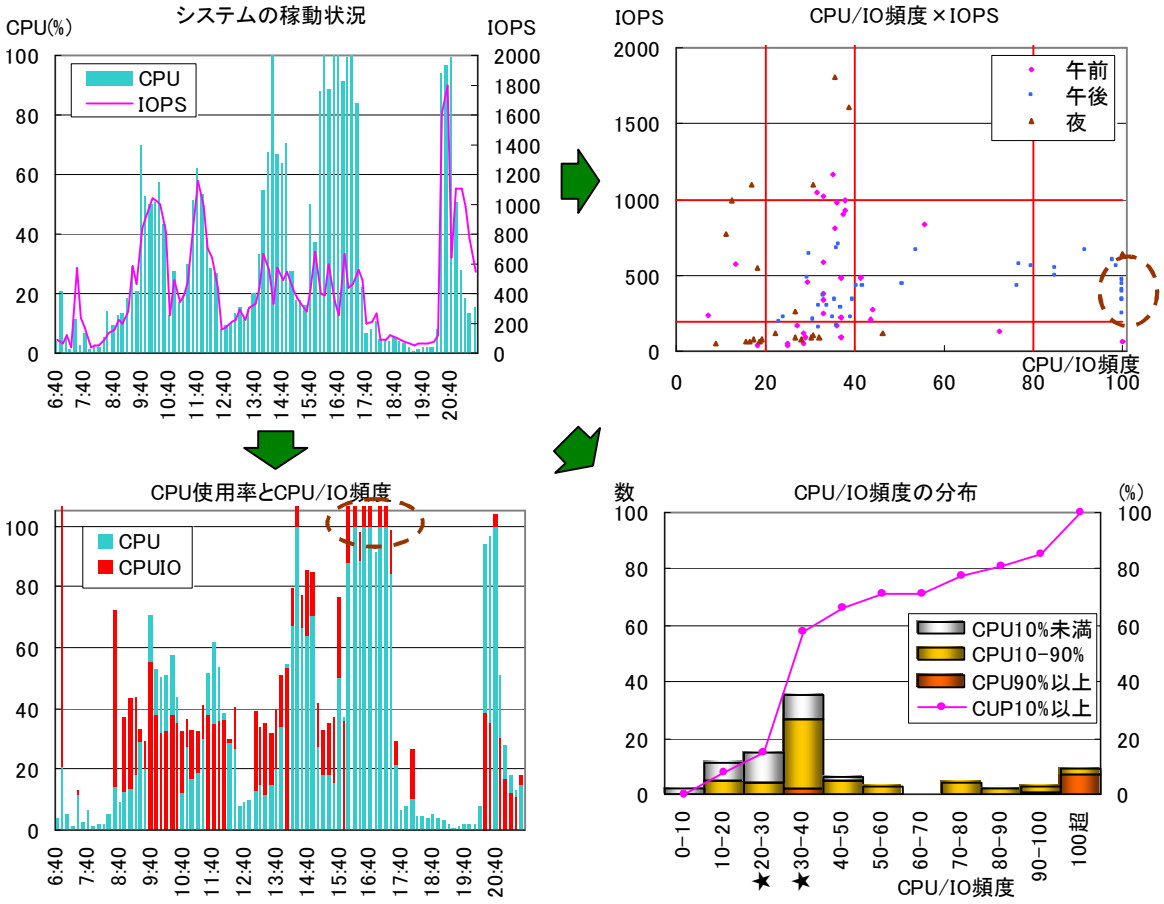
CPU/IO 頻度とは，コンピュータ内部で使っている CPU と発行する I/O のバランスを，標準が一定の値になるように数値化したものである．現時点では GS 独自の評価手法であり，表２に示すように標準値は 20～40（単位はない）になるように算出している．20 未満のとき「I/O 頻度が高い」，40 以上は「CPU 頻度が高い」と定義する．システム全体の評価だけでなくジョブの評価も可能である．

表２　CPU/IO 頻度の指標値（システムとジョブ）

CPU/IO 頻度	20 未満	20～40	40～80	80 超
システム	I/O 頻度が高い	標準	CPU 頻度が高い	CPU ループ

CPU/IO 頻度	20 未満	40 以下	40～100	100～1000	1000 超
ジョブ	I/O 頻度が高い	標準	CPU 頻度が高い	CPU ループ	超 CPU ループ

システムの CPU/IO 頻度分析の活用例を図９に示す.



左上：14:00 頃, 16:00 から約 1 時間, 20:30 頃, CPU 使用率（棒グラフ）が高い.
右上：CPU/IO 頻度が 100 を超えている点（10 分間の測定値）が複数ある.
　　IOPS は 200～500 出ている.
左下：特に 16:00 から約 1 時間は CPU 使用率が高く CPU/IO 頻度も 100 を超えている.
右下：CPU/IO 頻度は 30～40 の次に 100 超が多い.

図９　CPU/IO 頻度分析（システム）

従来のように（図９左上）CPU 使用率だけで評価すると，CPU 負荷が高いことはわかるが，その妥当性を客観的に評価することはむずかしい．図９左下のように CPU 使用率と CPU/IO 頻度を重ねると，14:00 頃と 16:00 から約１時間は突発的に CPU 使用率と CPU 頻度が高くなっており，ユーザアプリが CPU をムダに使っている可能性が高いことがわかる．CPU 使用率と CPU/IO 頻度の関係を表３に示す.

表３　CPU 使用率と CPU/IO 頻度の関係

		I/O 頻度が高い	CPU 頻度が高い
CPU使用率	低	・問題ない	・問題ない（システムの CPU 頻度）
	中	・（一時的）問題ない ・（常時）I/O ネックで CPU 負荷が上がっていない可能性がある	・（一時的）問題ない ・（常時）ユーザアプリが CPU を使いやすい
	高	・（一時的）ジョブの多重度が高い，無駄に多量のデータベースアクセスをしている ・（常時）CPU 能力が不足している	・（一時的）ユーザアプリが CPU をムダに使っている ・（常時）ユーザアプリが CPU を使いやすい

（２）CPU のムダの見える化（ユーザジョブ）

CPU 頻度の高いジョブは，SMF 等から抽出することができる．I/O 回数の求め方は OS やシステムパラメタ値，使用しているデータベースによって変わり，SMF 以外の性能データ（I/O トレース等）が必要なこともある．SMF やジョブログにある EXCP 回数は I/O 回数とは異なる．

図９のシステムで CPU 時間が 10 秒以上のバッチジョブの稼動状況を図１０に示す．CPU/IO 頻度をグラフの「パターン」で表している．14:00 頃と 16:00 から約１時間で CPU をムダに使っている可能性のある３つのジョブに線を引いている．

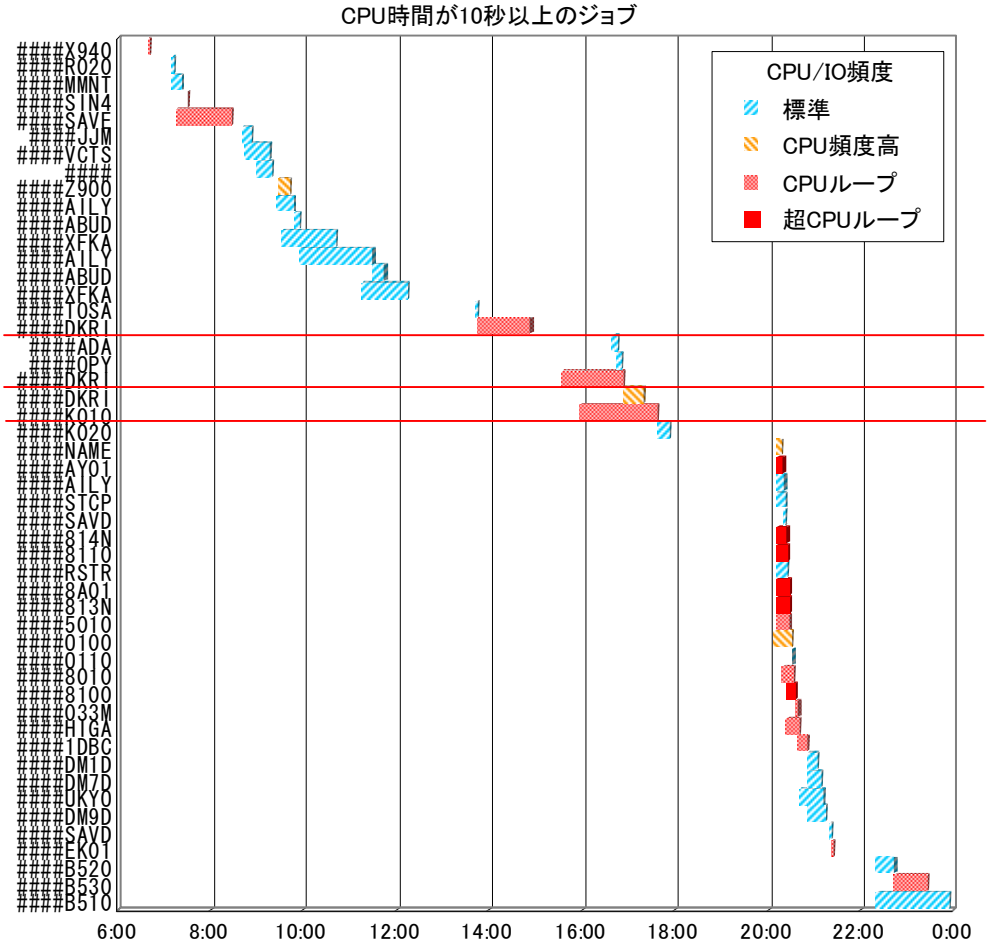


図１０　バッチジョブの稼動状況（CPU 時間 10 秒以上）

図１０に線を引いた３つのジョブの基礎情報を表４に示す.

表４　CPU をムダに使っている可能性のあるジョブ

ジョブ	開始	終了	処理時間	CPU 時間	I/O 回数	CPU 占有率	CPU/IO
①####DKRI	13:47	14:54	67 分	30 分	80 万	45%	137
②####DKRI	15:34	16:54	80 分	25 分	61 万	32%	167
③####K010	15:57	17:38	61 分	42 分	60 万	42%	284

（３）CPU のムダをどのくらい削減できるか

CPU/IO 頻度を基に，チューニングでどの程度 CPU 時間を削減できそうか予想することができる（表５）．経験的に，CPU/IO 頻度＝100 までは比較的容易に改善が可能であり，頑張っても CPU/IO 頻度=60 までと考えている．目標 CPU/IO 頻度(60～100)から目標 CPU 時間を逆算し，現在の CPU 時間との差分が削減予想 CPU 時間である.

CPU のムダとりの対象ジョブとすべきか否かの判断材料として使用する.

表５　CPU 時間の削減予想

CPU 時間		①####DKRI	②####DKRI	③####K010	備考
現　在		30 分	25 分	42 分	
CPU/IO 頻度=100	目標	22 分	15 分	15 分	比較的容易な目標
	削減予想	▲8 分	▲10 分	▲27 分	
CPU/IO 頻度=60	目標	14 分	9 分	9 分	高い目標
	削減予想	▲16 分	▲16 分	▲33 分	

３．３　CPU のムダとりのステップ（原因分析～改善）

（１）原因分析（詳細分析）

CPU をムダづかいしていると思われるユーザジョブが見つかったとき，その原因を正確かつ短時間で分析するには図１１に示す仮説検証型性能解析が効果的である.

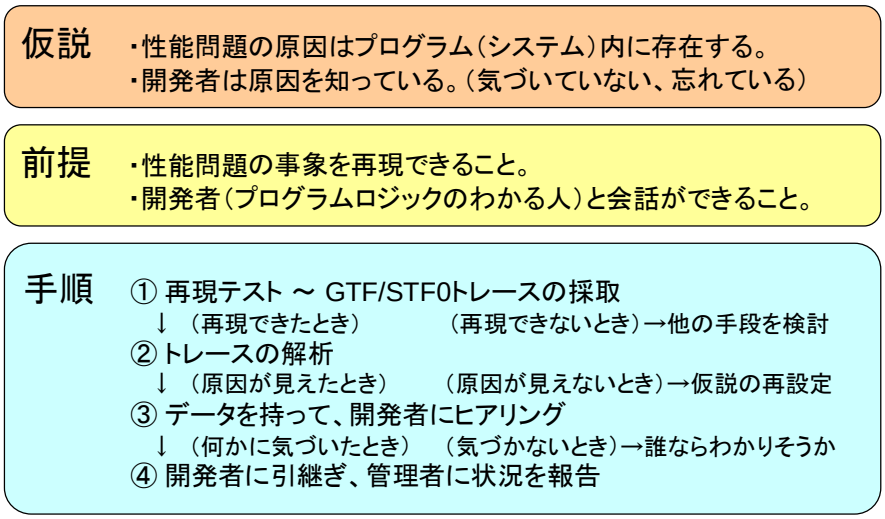


図１１　仮説検証型性能解析

表４の①のジョブをテスト機で実行し，仮説検証型性能解析を行った．

・テスト機でも本番機と類似した事象が起きることを確認．
・トレースを約１分×数回採取して解析した結果，COBOL の CHECK オプション用のモジュールが多量に走行していることが判明．
・お客様にヒアリングしたところ，CHECK オプションをつけており，運用上はずせないとのこと．
・該当プログラムが使っている全サブルーチンを対象に，CHECK オプションをはずれるものとはずせないものを整理．
・リコンパイル，再テストした結果 CPU 時間が 38%削減できたことを確認（表６）．

表６　CPU 時間の削減結果

環境	処理時間	CPU 時間	I/O 回数	CPU 占有率	CPU/IO
本番機　ジョブ①	67 分	30 分	80 万	45%	137
テスト機－改善前	51 分	19 分	87 万	38%	88
テスト機－改善後	50 分	11 分	87 万	24%	54

注　テスト機ではデータベースの更新処理など一部のステップを省略している．
　　テスト機の方が本番機よりも単体 CPU 性能がよい．

図１１－手順③のヒアリングでは，お客様と一緒にプログラムソースを追ったり，データベースのアクセスロジックを考えたりもすることもある．以下に実例を紹介する．

例１　CPU 時間が長く，CPU/IO 頻度は標準

特定のデータベースに 1000 万回もアクセスしているようなケースがある．このときは，まずプログラムの仕様から 1000 万回の妥当性を確認する．問題があればデータベースへのアクセスの仕方や SQL の発行状況等を調査し，ヒアリングを重ね，原因を究明する．（→（２）処理時間分析を参照）

例２　CPU 時間が長く，CPU 頻度が高い，デバッグオプションはついていない

トレースから特徴を探す．特定の I/O，モジュール，SVC 命令，処理のパターン等．手掛かりが見つからないときは，COBOL の COUNT オプションを採取し，お客様とソースを追い原因を究明する．CPU 頻度が高いということは頻繁に呼ばれる不適切なロジックが必ず存在するはずである．

例３　CPU 時間は短いが，実行回数が多く，CPU 頻度が高い

例２と同様に，CPU 頻度が高くなる箇所を特定し，原因を究明する．

例４　CPU 時間が長く，CPU 頻度が高いユーティリティ

SORT や SymfoWARE のユーティリティが該当し，チューニングの対象となることがある．（→４．CPU のムダとり事例を参照）

（２）処理時間分析

例１のようなプログラムについては，アクセスしているデータベースやファイルを調査し，処理時間の内訳を分析する．この作業を行なうと，処理の流れを理解でき，ヒアリングを効率よく進めるためのツールとしても活用できる．図１２にサンプルを示す．

ジョブ名：　####10@　　ステップ番号：　6　　プログラム名：　####0140
【処理概要】 ＃＃チェック
####0320サブをCALLして、＃＃マスターを作成する。
####0130で作成した、＃＃チェック用の入力ファイルをINのパラメータにセットして、####L000サブより＃＃チェックを行う。
U20に、＃＃データのチェック結果を出力する。U21に、＃＃入力データのチェック結果を出力する。
＃＃チェック用＃＃マスターを削除する。

【プログラムの特徴、改善策等】
・スプールアクセス、ライブラリアクセスが多い。
・データベースアクセスが多くてもCPU頻度が高くならない。
・処理件数が増えるとMDやRACF管理簿のアクセスが増える。
・####PB210で排他待ちが起きる。

【処理時間の分析】
(1) SMFより
10:24:22～10:24:28　①処理時間：5.56秒　CPU時間：1.113秒　I/O回数：2,161　CPU/IO頻度：37

(2) GTFトレース、SMF98より

データセット		VOL=SER	I/O回数	I/O時間	総I/O時間	POINT	GET	PUT	ERASE
ライブラリ			1,139	2	2,278 ms				
スプール			328	1	328 ms				
入力(U10)、出力(U20,U21)			6	1	6 ms				
VTOC、ワーク等			278	2	556 ms				
####DB061.MD			30	1	30 ms				
####PB000-####CTL	共通		6	1	6 ms	60	1,650		
####PB001-####AIx	共通		11	1	11 ms	5	7	2	
####PB040-####30x	＃＃記録		21	1	21 ms	4	18		
####PB200-####0x,####0x	標準　＃＃マスター		59	1	59 ms	6	6		
####PB210-####01～318	標準　＃＃マスター		185	1	185 ms	22	22	15	15
####PB220-####0x	＃＃マスター		30	1	30 ms	10	114		
####PB230-####0x	＃＃テーブル		92	1	92 ms	24	140		
####PB240-####0x	＃＃テーブル		24	1	24 ms	5	108		
計			2,209		3,626 ms	136	2,065	17	

図１２　処理時間分析のサンプル

【解説】
CPU/IO 頻度は標準であるが CPU 時間が長いと思われるプログラム
・SMF 等から処理時間，CPU 時間，I/O 回数を数パターン調査する．
　バッチ処理－ジョブステップ単位，オンライン処理－トランザクション単位
・データセット～総 I/O 時間は，I/O トレースを分析している．
・POINT～ERASE はデータベース（SymfoWARE）が発行しているマクロ回数．
・I/O の半分がライブラリへのアクセスであり，他のプログラムと比べても回数が多い．
　→　ヒアリングから，動的プログラム構造になっていることが判明．
・帳票出力していないのにスプールアクセスが多い．
　→　詳細のトレース分析から，ジョブジャーナルを取得していることが判明．
・データベース####PB000-####CTL は，60 回(POINT)INDEX 経由でアクセスし，その先で合計 1,650 回(GET)テーブルアクセスされているが，6 回(I/O 回数)しか実 I/O が発生していない．→　CPU も多く使っている可能性大，更に SQL を調査

（３）改善（チューニング）

ユーザアプリの改善作業はお客様自身にお願いをすることが多い．主導権をお客様に握ってもらうことが従来のシステムチューニングとの大きな違いであり，むずかしい点でもある．このような環境下で，CPU のムダとりプロジェクトを成功に導くためには，関係者にできるだけ早く改善の成功体験と感動体験を得てもらうことが効果的である．表６を例

にとると，19 分だった CPU 時間を，何分位削減できそうだと予想して，実際に 11 分にするのである．目の前で改善結果が見えることはお客様のモチベーションを上げるのはもちろん，一部のネガティブな意見に対しても丁寧に説明することができる．

仮説検証型性能解析～改善は，お客様とのコミュニケーションと改善したくなる場作りが大切になる．

３．４　GS だからできる CPU のムダとり

あるジョブが使っている CPU が「ムダか」「ムダでないか」の判断は難しいため皆様一人一人にお願いしたい．ここでは，なぜ「GS だからできるのか」について考察する．

① コストメリットが大きい
- CPU 能力を１ランク下げるだけで費用は数千万円～１億円以上違ってくる．

② CPU/IO 頻度分析ができる（→３．２）
- データをアクセスする際，DISK に実 I/O を発行することが前提となる手法である．（データをメモリ展開しているようなシステムには適用できない．）

③ システム内部の詳細分析が可能である（→３．３）
- OS レベルのトレースによる一次分析が可能である．

④ システムが安定している
- システムが過負荷状態（CPU 使用率が 100%，ページング多発，デッドロック多発等）になってもシステムは問題なく動作する．

⑤ ユーザ業務が基幹システムである
- 業務内容は違っても，処理のパターン，DBMS，ログ管理，言語等は同じである．

逆に，GS だからできないと言われる代表的な理由は，
- お客様でユーザアプリの保守ができない，改善するパワーがない．

である．CPU のムダとりプロジェクトの作業分担と平均的な作業期間を表７に示す．まず１ヶ月間程度（対象 A の改善と評価まで）で成果を出すことが重要である．

表７　CPU のムダとりプロジェクトの作業分担と作業期間

作業内容	性能評価プロセス(図８)	お客様	弊社
性能データの採取	①	(1W～4W)	(支援)
システム全体の性能評価[概要]	②～④(概要)	ー	1W
業務の性能評価[概要]	②～④(概要),⑦～⑧(概要)	ー	1W
→ムダとり対象の洗出し(A,B,C,・・・)			
対象 A の詳細分析	②～④(詳細)	(テスト支援)	1W
対象 A の改善策立案，期待効果	⑤～⑨	(ヒアリング)	1W
→改善実施の可否判定			
対象 A の改善と評価	⑩～⑪　※並行して実施	1W(改善)	1W(評価)
→フィードバック（横展開）と次のムダとり対象 X の選定			
対象 X の詳細分析			1W
対象 X の改善策立案，期待効果			
対象 X の改善と評価		1W(改善)	1W(評価)
→以降，繰返し			

４．CPU のムダとり事例

オンラン処理，バッチ処理，TSS およびユーティリティでの CPU のムダとり事例を紹介する．

事例１～オンライン処理の CPU のムダとり

a. システムの概要

CPU：大型機　　メモリ：１GB　　CPU 使用率：日中約 60%（平常期）

対象：オンラインジョブ　CPU 時間：1,823 秒/日　CPU/IO 頻度：1,158

主たるプログラムの平均 CPU 時間：483ms/TRN

備考：業務の性能評価［概要］で対象ジョブ，プログラムを絞込み，処理の内容をヒアリング．

業務上，最も重要な処理でピーク期にはデータ量が２倍以上に増加．

CPU 頻度が非常に高い(1,158)ため大幅な削減を期待．

b. 原因分析と効果

仮説検証型性能解析より，以下の CPU のムダが見つかり改善．

① 以前使っていた０件データベースへのアクセスを削除（CPU35%削減）

② 使っていない SELECT COUNT 命令を削除（CPU28%削減）

③ 画面スクロールの先読みを改善（CPU15%以上削減）

c. お客様の作業期間

テスト機での改善・評価まで２週間弱．

d. 感想

お客様：データベースのアクセス回数など具体的な数値が示されわかりやすかった．

①～③の観点でプログラムと共通サブルーチンに横展開を実施した．

事例２～バッチ処理の CPU のムダとり

a. システムの概要

CPU：大型機　　メモリ：256MB　　CPU 使用率：100%（夜間）

対象：バッチジョブ　CPU 時間：56 分　CPU/IO 頻度：547

備考：業務の性能評価［概要］で対象ジョブを絞込み，処理の内容をヒアリング．

b. 原因分析と効果

・サーバへのファイル転送の前処理で，製品が提供しているサブルーチンを使って文字コード変換を行い CPU を多量に使っていることが判明．

→　文字コード変換をサーバ側で行なうことにした（CPU80%削減）．

c. お客様の作業期間

本番機への適用まで数日．

d. 感想

お客様：メーカに提供して頂いたプログラムでノーチェックだった．

事例３～TSS の CPU のムダとり

a. システムの概要

CPU：超大型機　　メモリ：0.8GB　　CPU 使用率：70～100%

対象：TSS 全体　CPU 使用率：35～50%　CPU/IO 頻度：30

TSS ユーザ数は 10 未満

備考：性能評価［概要］で TSS 空間の CPU 使用率が異常に高いことが判明.

CPU/IO 頻度は標準だが，CPU 使用率 35～50%が異常値であると気づくことが最大のポイント.

b. 原因分析と効果

・GTF トレースを解析したところ，複数の TSS 空間が頻繁にスプールをアクセスしていることが判明.

→　ヒアリングをしたところ，リアルタイムモニタを使いスプールを 1 秒間隔で監視していることが判明.

→　監視の間隔を 10 秒に変更，端末を制限（CPU90%以上削減）.

c. お客様の作業期間

本番機への適用まで数日.

d. 感想

導入支援した SE：こんな使い方は想定外だった.

お客様：こんなに CPU を使うとは思ってもみなかった.

事例４～ユーティリティ(SORT)の CPU のムダとり

a. システムの概要

CPU：超大型機　　メモリ：1 GB　　CPU 使用率：ほぼ 100%（夜間）

対象：バッチジョブ（単純な SORT 処理）

処理時間：150～206 秒　CPU 時間：78～86 秒　CPU/IO 頻度：876～982

備考：業務の性能評価［概要］で対象ジョブを絞込み，ステップを分析したところ SORT の CPU 時間が大きく CPU 頻度も高いことが判明.

b. 原因分析と効果

・GTF トレースを解析しても原因はわからず．開発元からは SORT の仕様と回答.

・メモリアクセスの方法がおかしいのではないかと仮説を立てる.

→　SORT が使えるメモリサイズを小さくしてテストをする．具体的には拡張 REGION を使えないように設定.

→　処理時間はほとんど変わらず CPU 時間を約 60%削減.

処理時間：164～273 秒　CPU 時間：30～34 秒　CPU/IO 頻度：125～143

・処理時間が 273 秒のときは，強制 SWAPOUT が多発していた.

・I/O 回数は約 24,000 回→65,000 回に増加したが影響はなかった.

・すべての SORT 処理がおかしい訳ではなく，今回の方がレアケースである.

c. お客様の作業期間

本番機への適用まで数日.

d. 感想

お客様：昔から使っているユーティリティなので何も意識していなかった.

５．評価と課題

CPU の処理能力が目覚しく向上し，コストが下がり，クラウドも現実的になってきた．この時代に，GS は存在価値があるのだろうかと模索し，たどり着いたところが GS のメタボ診断であり，今回の CPU のムダとりであった．ムダは巨大な CPU 能力でカバーできるというのがグローバル・スタンダードな考えであろう．しかし，GS の強みは，CPU のムダが外から見えて，容易に改善できることであると考える．

（１）評価

CPU のムダとり手法を確立し，ユーザアプリの改善に積極的に取り組み，効果を上げることができた．

① CPU のムダの見える化
- CPU/IO 頻度分析を使い，ムダの状況が一目で見える仕組みを確立．
- ジョブ（プログラム）を特定し，CPU 時間の削減予想も可能．

② ムダの原因分析
- 仮説検証型性能解析を使い，プログラム内の問題箇所を特定できる．
- 処理時間分析を行い，お客様とムダを掘り下げる．

③ 改善
- お客様のモチベーションを上げ，改善作業に取り組んでもらう．
- 改善したくなる場作りをする．

（２）課題
- CPU のムダとりの普及（SE，お客様）
- CPU 能力変動型アウトソーシングサービス等との連携
- 基幹系のオープンシステムへの展開

以上

参考文献

［１］有賀光浩，「メインフレームのパフォーマンス変革　－お客様が感動したチューニング効果とコスト削減のプロローグ－」　FUJITSUファミリ会2004年後期論文

［２］有賀光浩，「性能管理・改善のブレークスルー　－自らの常識を破り，どのように性能改善を実現したか－」　FUJITSUファミリ会2005年前期論文

［３］有賀光浩，「メインフレーム変革！　お客様視点・現場重視の性能評価　－３割のコスト削減・性能向上・満足度向上運動－」　FUJITSUファミリ会2006年前期論文

［４］有賀光浩，「メインフレーム変革！　性能品質向上への挑戦　－CPU統合の見える化と改善－」　FUJITSUファミリ会2006年後期論文

［５］有賀光浩，「メインフレームの特徴を生かしたIT全般統制(ITGC)構築のポイント」FUJITSUファミリ会2007年論文

［６］有賀光浩，「メインフレームのメタボ化とその診断手法」　FUJITSUファミリ会2008年論文

［７］有賀光浩，「メインフレームのメタボ診断の応用」　FUJITSUファミリ会2009年論文

用語説明

GS21, PRIMEFORCE	富士通メインフレームの機種名
AVM	仮想化を実現する製品
AIM	オンラインシステムを支援するサブシステム
CPU/IO頻度	CPUとI/Oのバランスを示す指標
IOPS	１秒当たりのI/O回数
SMF	課金データ
CPU占有率	CPU時間／処理時間
GTF/STFO	OSの内部トレースを取得するプログラム
SymfoWARE	GSのリレーショナル型データベース