性能トラブル解決の勘所

鈴木 勝彦
株式会社日立ソリューションズ
masahiko.suzuki.yh[at-mark]hitachi-solutions.com

要旨

ソフトウェア保守では，性能トラブルが発生すると原因究明までに時間がかかることが多い．これは，性能トラブル発生時の調査手順が確立されておらず，場当たり的で試行錯誤しながらの調査となっているためである．また，性能トラブルの原因としてどのような要素があるかなどが体系的にまとめられていないことにも起因している．

本研究では，性能トラブルの要素などを 4W1H の概念で体系化し，実際の性能トラブル発生時の調査の勘所をまとめた．性能トラブルでお悩みの方々のお役に立つことができれば幸いである．

1. はじめに

ソフトウェア保守では，ソフトウェアが稼働してからトラブルが発生することは避けられないが，原因究明にかかる時間は，原因によって短い時もあれば長期化することもある．中でも性能トラブルが発生すると，原因究明までに時間がかかることが多い．特に，解決までの時間は，担当者の経験，知識や技能によって全く違う．また，解決のノウハウがドキュメント化されていないため，ノウハウの伝承がうまくいっていない．著者は，一定の技術者であれば，性能トラブルが発生しても一定の時間で解決できる資料が必要と考えた．

当初は，著者自身が過去に経験した多くの事例集を作成した．しかし，ソフトウェア・メインテナンス研究会のメンバから単なる事例集では実際の性能トラブル発生時の調査では活用が難しいとのご指摘を受け，調査ポイントを列挙して段階的に調査できる形式に再作成した．しかし，すべての事例は網羅できないため，調査時の観点を体系化できないかを試行錯誤し 5W1H の概念が使えるのではないかと閃いた．最終的には，性能トラブルの要因などを 5W1H から「Where」と「Why」を除き，「Whom」を追加した 4W1H での体系化に至った．体系化したことで，過去に列挙した事例だけでなく，経験と体系を基に組合せることで，考えられる事例を追加し，より網羅性のある事例集を作成することができた．この論文を使用することで，一定のスキルを有する技術者であれば 4W1H の観点を思い浮かべながら，事例集を基に調査することで，解決までの時間を短縮することが可能と考えられる．

本論文では，最初に 4W1H による体系的な説明を実施し，最後にいくつかの性能向上施策方法を記述した．

2. 性能トラブル時の調査観点を 4W1H で考える

性能トラブルに対して，経験豊富な特別な技術者でなく，普通の技術者でも 4W1H による体系的な観点に基づいて調査すれば，試行錯誤することなく，解決までの時間短縮に有効であると考えた．

「4W1H」とした経緯は，体系的にするにあたって「5W1H」という考え方に基づき，過去 100 例ほどの性能トラブルに対して，When(いつ)，Where(どこで)，Who(誰が)，What(何を)，Why(なぜ)，How(どうやって)の6項目毎に観点を追加する作業を試みた．その際に，Why(なぜ)の項目は人の動機に該当し，物理現象には該当しないため除外した．さらに，外部影響によって性能トラブルとなる事例の分類の必要性から，Whom(誰によって)の項目を追加した．

また、Where の場所の特定はプログラムの場合には Who のプロセスの特定と同じ結果となり，Who に該当するプロセスに関する情報は，統計情報などで確認できるが，Where に関する情報は簡単には得られないため，Where も項目から削除し，結果として「4W1H」とした．一般的に情報伝達する時に「5W1H」を意識する必要があるように，性能トラブル時には，「5W1H」に対応する「4W1H」によって事象を正しく把握し，結果として原因究明ができるようになる．

「4W1H」のそれぞれについて，どのように捉えるかを以下に説明する．

・Who(誰が)は，原因となるプロセスを究明する．
　性能トラブルが Who(誰が)に該当するプロセスかを究明し，さらにそのプロセス内のどのソースコードかを究明する．

・How(どのように)は，原因となる動作を究明する．
　性能トラブルは，動作がスムーズに処理できていないため発生する．現象などから，どのような振舞いによって遅くなっているかを究明する．

・What(何を)は，原因となる資源を究明する．
　プログラムは，実行する時にさまざまな「資源」を消費しながら処理している．プログラムが，システムで供給できる「資源」を上回る消費を行おうとすると遅延が発生するため，不足している「資源」を究明する．

・When(いつ)は，原因となるきっかけを究明する．
　性能トラブルは，問題なく動いていても突然発生することがある．しかし，プログラムは，入力する情報が同じであれば，同じ結果になるハズである．遅延の発生は，ある時点(When)から入力情報が異なることを示すため，変わった時点を究明する．

・Whom(誰によって)は，原因を誘発したものを究明する．
　通常の性能トラブルは，遅くなっているプロセスに起因することがほとんどであるが，外部の影響を受けて発生することもある．このため，調査する時には自プロセスだけでなく，システム全体または他システムも含めて全体を把握する必要がある．

3. Who(誰が)は，原因となるプロセスを究明

性能トラブルが発生した時には，Who(誰が)に相当するどのプロセスに起因して発生しているかを特定する必要がある．プログラムは，1 つのプロセスだけでなくいろいろなプロセスが連携して動作している．また，1 つのプロセスの中でも自分が作成したソースだけでなく，同梱しているライブラリであったり，call している API などのソースもある．問題となるプロセスは，場合によっては自ホストとは限らず，他ホストに起因することもある．図 1 は，原因となるプロセスが Who(誰が)を分類したものである．

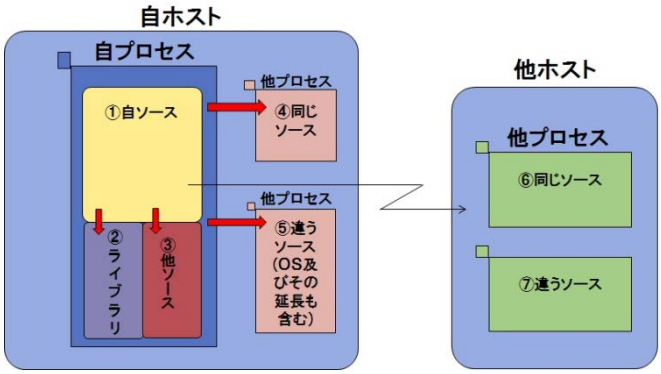

図 1.　Who(誰が)の原因となるプロセス究明時の観点

3.1　性能トラブルがどのプロセスで発生しているかの観点

性能トラブルが発生した時は，現象などから見た目の動作が遅いプロセスを特定するが，場合によっては他のプロセスが起因している可能性もある．実際には，プロセス単位の CPU 使用率，DISK 入出力回数と入出力バイト数，データ通信量などの統計情報などから問題のプロセスを絞り込んでいく．さらに，プロセスの中のどの部分で発生しているかを絞り込んでいく．Who(誰が)の絞り込みであるが，場所の特定という意味では Where(どこで)の要素も含まれている．

3. 2 原因となるプロセスの絞り込み方法について

原因となるプロセスを特定するには，最初にシステム全体及びプロセス単位での CPU 使用率，DISK 入出力回数と入出力バイト数，データ通信量を確認し，さらに次のような観点でプロセスを絞り込んでいく．

(1)自分のプロセスの CPU 使用率が高い，DISK 入出力回数が多い，または入出力バイト数が多い，データ通信量が多い場合は，自プロセスの可能性が高く，次の 3 つが考えられる．

・自プロセスの自分で作成したソースで発生．
・自プロセスだが，取り込んでいるライブラリで発生．
・自プロセスだが，call している API の延長で発生．(システム関数の場合も含む)

(2)他プロセス(自分で作成したソース)の CPU 使用率が高い，DISK 入出力回数が多い，または入出力バイト数が多い，データ通信量が多い場合は，他プロセスの可能性が高く，次の2つが考えられる．

・自プロセスの延長で他プロセスが実行されるが，他プロセスからの戻りが遅いことで自プロセスの性能がでないことがある．

この場合は，他プロセスに関して調査する．

・自プロセスは，他プロセスと通信しながら処理しているが，他プロセスからの戻りが遅いために性能がでないことがある．この場合は，他プロセスと通信の状態に関して調査する．

(3)他プロセス(他人が作成したソース)の CPU 使用率が高い，DISK 入出力回数が多い，または入出力バイト数が多い，データ通信量が多い場合は，他プロセスまたは，システム全体の可能性が高く，次の2つが考えられる．

・特定の他プロセスだけ(システムプロセスも含む)がリソースをたくさん使用し，自プロセスから call している延長で発生しているかを確認する．
・特定の他プロセスだけ(システムプロセスも含む)がリソースをたくさん使用し，自プロセスが確保するリソースと競合が発生していないかを確認する．

(4)システム全体の CPU 使用率が高い，DISK 入出力回数が多い，または転送バイト数が多い，データ通信量が多い場合は，他プロセスまたは，システム全体の可能性が高く，次のことが考えられる．

・システム全体が資源をたくさん使用している場合には，自プロセスが確保する資源と競合が発生していないかを確認する．
・システム全体のハードウェアのリソースが不足していないかを確認する．

3. 3 プロセスを特定した後，さらにどのソースコードで発生しているかを調査する方法について．

(1) 自分のプロセスでかつ自分のソースで発生

・自分のソースなので，トレースログを強化すれば，場所の絞り込みも可能．

(2)自分のプロセスであるが，取り込んでいるライブラリ部分で発生

・ライブラリの前後でトレースログを出力すれば，絞り込み可能．

(3)自分のプロセスであるが，API の延長(他人のソース)で発生

・API の前後でトレースログを出力すれば，絞り込み可能．

(4)他プロセスであるが，自分と同じソースで発生

・自分のソースなので，トレースログを強化すれば，場所の絞り込みも可能．

(5)他プロセスでかつ，他人のソースの部分で発生

・API の前後でトレースログを出力し，該当する他プロセスが自分の発行している API の延長(OS なども含む)であるかを確認する．
・他プロセスが特定の資源を大量に消費していないかを確認する．

(6)他ホストであるが，自分と同じソースで発生．

・通信処理の前後でトレースログを強化すれば，場所の絞り込みも可能．他ホストとの通信がある場合には，他ホストからのレスポンス待ちで自ホストの自プロセスが遅く見える場合がある．原因の究明は，自ホストと同様の方法で他ホストのプロセスを調査する．

(7)他ホストでかつ，他人のソースで発生

・通信処理の前後でトレースログを強化すれば，場所の絞り込みが可能．しかし，他ホストの他人のソースが起因する場合には，原因の究明は難しい．

4.　How(どのように)は，原因となる動作を究明

性能トラブルが発生した時には，How(どのように)に相当するどんな事象が起きているかを特定する必要がある．性能トラブルの事象の原因は多種多様ではあるが，分類すると大きく6種類になる．図2は，性能トラブルの事象の原因を How(どのように)の観点で分類したものである．

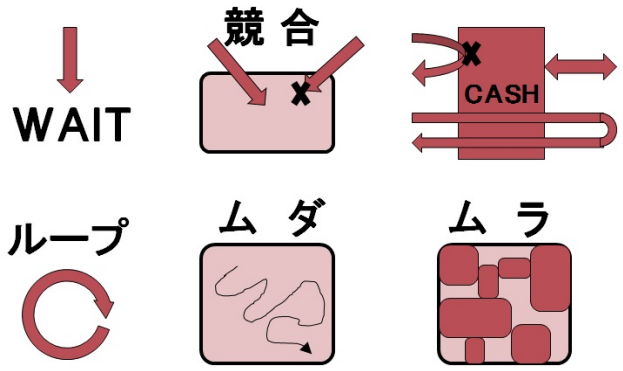

図 2. How(どうした)の原因となる動作を究明時の観点

4. 1 性能トラブルがどのようにして発生しているかの観点

3章では，性能トラブルがどのプロセスで発生しているかを究明する方法を記載したが，原因究明には，そのプロセスが具体的にどのようにして性能低下しているかを究明する必要がある．

性能トラブルの原因のメカニズムに関しては，大きく「WAIT」，「競合」，「キャッシュ」，「ループ」，「ムダ」，「ムラ」に分類できる．性能トラブル解析時には，この 6 つのどの原因で発生しているかを特定する必要がある．

4. 2 原因となる事象の絞り込み方法について

(1) WAIT による性能トラブルについて

WAIT(待ちが発生)は，性能トラブルの中でも一般的には原因究明が容易な方である．しかし，非常に短い WAIT が頻発している場合には，原因究明に時間がかかることがある．

排他資源の解放待ちは，「WAIT」と排他資源の「競合」の両方の要素がある．複数資源の排他によるデッドロックの場合には，自プロセスでわかる排他資源は 1 つであり，他プロセスの排他待ちで異なる排他資源を究明する必要があり，時間がかかることがある．

WAIT の具体的な要因としては，以下のものがある．

・タイマー値が大きい
・タイマー値が大きくリトライあり
・タイマー値が小さくリトライを多発
・相手の処理待ち
・無限待ち
・デッドロック(「競合」でもある)

(2) 競合による性能トラブルについて

競合は，ハードウェア資源の競合と，排他資源の競合がある．ハードウェアの競合は，リソースの状況を確認することで判明することが多いが，ソフトウェアでの排他資源の競合は OS の稼働情報では分からないので，判明までに時間がかかることが多い．

競合の具体的な要因としては，以下のものがある．

・CPU
・メモリ
・DISK
・通信
・排他資源

(3) キャッシュによる性能トラブルについて

キャッシュに起因する性能トラブルは，原因究明に時間がかかることが多い．例えば，DISK の I/O では，ユーザープログラムがレコードをブロッキングしたり，OS のメモリキャッシュでの I/O，VM のキャッシュ，ハードウェアのキャッシュといろいろな階層でのキャッシュが存在している．つまり，それぞれの階層でキャッシュのサイズ，動作タイミング，動作特性などの相性があり複雑な動きとなる．これに I/O の同期，非同期の組み合わせも加わる．

例えば，ユーザープログラムでレコードをブロッキングし，ブロック単位で同期 I/O を実施した時に，DISK のキャッシュサイズより小さい時は，キャッシュに書き込んだ直後に戻るが，大きなサイズになると，実際の DSIK の円盤への書き込み処理との同期が発生して，非常に時間がかかるようになる．

また，実行時間の短いユーザープログラムで大量データの I/O を非同期で実行した場合，途中の I/O の時間は短いがファイルのクローズまたはプロセスの終了時に OS のキャッシュ上のデータの書き込み処理の同期待ちが発生し，プロセスの終了までに時間がかかることがある．

キャッシュの具体的な性能劣化要因としては，以下のものがある．

・ヒット率低下

例えば，DISK 上に A データ，B データ，C データ．．．Z データのように複数のレコードを連続して書き込んだファイルに対して，Z データから前の方向に読み込み処理をするとキャッシュのヒット率が低下することがある．DISK の読み込み処理の場合には，OS レベルなどで先読みしている場合が多く，順方向であれば，キャッシュに先読みしたデータがありヒット率が高くなるが逆読みすると先読みの効果が得られずにヒット率の低下となる．

・キャッシュの容量不足
・ハードウェアなどのキャッシュのアクセススピードが遅い
・キャッシュが無効

(4) ループによる性能トラブルについて

ループに起因する性能トラブルは，原因究明が短い時と長期化する時の2極化する傾向がある．ループする範囲が小さい場合には，原因究明が短いことが多いが，広い範囲でのループになるとループしてしまう原因究明に手こずることがある．特にループの中にネストしてループの処理を追加した場合には，ループの回数は積数となるので，どちらかのループ回数が増加しただけで急激に増加してしまう．

ループが起因するものとしては，以下のものがある．

・無限ループ
・非常に回数の多いループ
・リトライ回数が多くなる可能性があるループ
・ループ中のループ
・ループの処理中にシステムコールなどの重い関数を追加

(5) ムダな処理による性能トラブルについて

ムダな処理に起因する性能トラブルは，いろいろな原因が考えられるので究明に時間がかかることが多い．特に，想定していた処理件数では全く性能的に問題なく動作していたものが，件数の増加とともに指数関数的に性能が劣化するケースがあり，原因究明に時間がかかることがある．

ムダな処理の具体的な要因としては，以下のものがある．

・輻輳
・メモリ確保関数を頻発

・I/O 処理を頻発
・同期処理
・メモリのコピーの実装が悪い
・ソート, コンペアの実装が悪い
・サーチ, インデックスの実装が悪い
・キューイング処理の実装が悪い
・ループ中のループ

(6) ムラな処理による性能トラブルについて

ムラに起因する性能トラブルは, 動作が特定することが難しく, 原因究明が長期化することが多い. 例えば, プロセスのプライオリティが低いような場合には, 他のすべてのプロセスがどのようにCPUなどを使用しているかで振舞いが変わるため, 該当プロセスだけの振舞いを調査しても原因究明には至らない. DISK の断片化による性能トラブルは, 突然発生するのではなく, 少しずつ性能低下していくので, 気づかないことが多い.

ムラな動作の具体的な要因としては, 以下のものがある.

・プライオリティ
・DISK の断片化
・メモリの断片化
・バッファサイズの不適切な値

(7) その他の性能トラブルについて

性能トラブルは, (1)から(6)のような事象に分類されることが多いが, プログラム固有の実装ロジックがあったり, ハードウェアの特性に起因することもある. 例えば CPU などでは, 動作時の温度によってクロック数が変わるものがある. これなどは, ソフトウェアを開発する人にとっては, 再現テスト時の観点として, ハードウェア特性の観点はなかなか想像できないことが多い.

その他の具体的な要因としては, 以下のものがある.

・同居製品固有の実装ロジック
・OS 固有の実装ロジック
・ライブラリ固有の実装ロジック

例えば, 文字列を処理する append()関数などは注意が必要である.

・自プロセスの固有の実装ロジック
・通信品質の特性(特に無線 LAN)
・ハードウェア特性

4. 3 どのような現象が起きているかを具体的に調査する方法について

4. 3. 1 該当プロセスの CPU 使用率, メモリ使用量, DISK I/O 量, 通信量を確認する.

(1)該当プロセスの CPU 使用率が 0%の時

(a)CPU 使用率が 0%の時は, WAIT で止まっている可能性が高いので, 止まっている場所を特定する.

・プログラムのトレースログを確認して場所を推定する.
・スナップダンプを取得し, 場所を特定する.
・システムコールトレースから, 場所を推定する.
・システムコールであれば, システムコール名称とシステムコール時の引数を確認する.

(b)止まっている原因を特定する.

・排他処理であれば, 排他資源名を確認して, デッドロックになっていないかを確認する.
・タイマー設定後の WAIT 関数であれば, タイマー値を確認し, 値が大きすぎないかを確認する.
・システムコール等に続く WAIT 関数であれば, エラー後のリトライ間隔が大きすぎないかを確認する.
・システムコール等に続く WAIT 関数であれば, エラー後のリトライ回数が大きすぎないかを確認する.
・他プロセスと通信しながらの処理があり, 他プロセスからの返信待ちになっていないかを確認する.
・システムコール等で無限待ちの引数でコールしていないかを確認する.
・排他の範囲が大きく他のプロセスとの競合が発生し易くなっていないかを確認する.
・排他の範囲を小さくするために, 排他を細切れにして発行回数が多くなっていないかを確認する.

(2)該当プロセスの CPU 使用率が 100%に近い時

該当プロセスの CPU 使用率が 100%であっても, 8 コアの CPU だと, システム全体の CPU 使用率が 12. 5%と表示されるので, プロセス毎の CPU 使用率を確認する必要がある.

(a)プロセスの CPU 使用率が 100%に近い時は, ループしている可能性が高いので, ループしている場所を特定する.

・プログラムのトレースログを確認してループしている場所を推定する.
・スナップダンプを複数回取得し, ループしている場所を特定する.
・システムコールトレースから, ループしている場所を推定する.

(b)ループしている原因を特定する.

・スナップダンプからループカウンターの値を確認する.
・ループ箇所のソースコードから無限ループに陥ることがないかを確認する.
・ループ箇所のエラー処理で無限にリトライすることがないかを確認する.
・ループ箇所の中に排他エラーの時に無限にリトライすることがないかを確認する.
・ループの中にループするような処理がないかを確認する.

(c)実装方法の確認

・コピーの実装が悪い
・ソート, コンペアの実装が悪い
・サーチ, インデックスの実装が悪い
・キューなどの実装が悪い
・文字列操作などの実装が悪い

(3)該当プロセスのメモリサイズだけでなく, メモリの確保/解放回数も確認する.

(a)処理するデータ件数が多くなった時の振舞いを確認する.

・メモリの確保/解放のロジックを確認し, 件数に比例する以上のメモリを使用しないかを確認する.
・件数の増加に伴って, メモリの確保回数も多くならないかを確認する.
・レコード1件読む毎にメモリを確保し, チェーンでつないでいないかを確認する.

(b)処理するデータサイズが大きくなった時の振舞いを確認する.

・最初のデータ格納サイズが小さく，大きなデータになると少しずつメモリ増分していないかを確認する．
・C++のライブラリで「strcopy」などの関数を使用していないかを確認する．

(4)該当プロセスの DISK I/O の確認
(a)該当プロセスの DISK I/O 回数が非常に多い
コンピュータのパフォーマンスを測定するツールで DISK の読み込み回数，書き込み回数を確認する．
(b)該当プロセスの DISK I/O のサイズが大きい
コンピュータのパフォーマンスを測定するツールで DISK の読み込みサイズ，書き込みサイズを確認する．
(c)DISK の I/O を同期 I/O が不必要な所で実施していないかを確認する．

(5)該当プロセスの通信の量が多い時
(a)ネットワークのパフォーマンスを測定するツールで送信量，受信量を確認する．
(b)プログラムを確認して，プロトコルデータのサイズと通信バッファのサイズを確認する．
(c)OS の通信バッファのサイズを確認する．

4. 3. 2 他のプロセスの CPU 使用率，メモリ使用量，DISK I/O 量，通信量を確認する．
(1)他のプロセスの CPU 使用率が 100%に近い時
・プログラムのトレースログを確認して，他プロセスを呼び出しているかを確認する．
・プログラムのトレースログを確認して，他プロセスを頻繁に呼び出しているかを確認する．
・スナップダンプを取得し，他プロセスを呼び出しているかを確認する．

(2)system の CPU 使用率が 100%に近い時
・システムコールトレースから，システムコールが頻繁に発行されていないかを確認する．
・プログラムのトレースログを確認して，システムコールの呼び出しの延長かを確認する．
・プログラムのトレースログを確認して，システムコールの呼び出しを頻繁に実施しているかを確認する．

(3)全体の CPU 使用率が 100%に近いが，自プロセスの CPU 使用率が低い時
・自プロセスに十分な CPU が割り当てられていない状況かを確認する．
・自プロセスのプライオリティを確認する．

(4)他のプロセスのメモリの使用量が多い時
・システム全体のメモリ使用量を確認し，システム全体でページフォルトが頻発しているかを確認する．
・システムコールトレースで，システム全体のメモリの確保/解放が頻発しているかを確認する．
・スラッシングが発生しているかを確認する．

(5)システム全体の DISK I/O を確認
(a)システム全体の DISK I/O 回数が非常に多い
・DISK I/O のパフォーマンスを測定するツールで読み込み回数，書き込み回数を確認する．
(b)システム全体の DISK I/O のサイズが大きい
・DISK I/O のパフォーマンスを測定するツールで読み込みサイズ，書き込みサイズを確認する．

(6)システム全体の通信量が多い時
(a)送受信の量が多い
・ネットワークのパフォーマンスを測定するツールで送信サイズ，受信サイズを確認する．

4. 3. 3 通信相手の他プロセスを確認する．
現象が表面化するプロセスに問題がなくても，通信相手のプロセスからのレスポンスが悪いことがある．このような場合には，通信相手のプロセスの調査が必要になる．

(1)通信相手の他プロセスも自作のプログラムの場合
・通信相手のプロセスのトレースログなどを確認する．
・通信相手のプロセスの CPU 使用率，メモリ使用量，DISK I/O，通信などの負荷状況を調査する．
・通信相手のプロセスが他ホストの場合，他ホストのシステム全体の CPU 使用率，メモリ使用量，DISK I/O，通信などの負荷状況を確認する．

(2) 通信相手の他プロセスが自作のプログラムでない場合
・通信相手のプロセスの CPU 使用率，メモリ使用量，DISK I/O，通信などの負荷状況を調査する．
・通信相手のプロセスが他ホストの場合，他ホストのシステム全体の CPU 使用率，メモリ使用量，DISK I/O，通信などの負荷状況を確認する．

(3)自プロセスと通信相手のプロセスの間で通信の輻輳が発生していないかを確認する．

4. 3. 4 プロセスのプライオリティを確認する．
(1)プロセスのプライオリティの確認
・自プロセスだけでなく，他のプロセスも nice 値とプライオリティを確認する．

4. 3. 5 各種キャッシュの状況を確認する．
キャッシュは，ヒット率，キャッシュサイズ，キャッシュの種類，キャッシュの有効/無効などを確認する．

(1)DISK I/O 時の自プログラムのメモリ上のキャッシュを確認する．
例えば，複数レコードを1ブロックで書き込むような場合．
(2)DISK I/O 時の OS のメモリ上のキャッシュを確認する．
(3)DISK I/O 時の DISK 上のキャッシュを確認する．
(4)通信時の自プログラムのメモリ上のキャッシュを確認する．
(5)通信時の OS のメモリ上のキャッシュを確認する．
(6)通信時の通信装置によるキャッシュを確認する．
(7)メモリのキャッシュを確認する．
科学計算のようなインコアで処理するロジックがある場合には，
・CPU にある一時キャッシュのサイズ，二次キャッシュのサイズとアクセススピードを確認する．
・マシンが SMP(Symmetric Multiprocessing)構成かを確認する．VM の場合に，割り当てられるメモリキャッシュが他の CPU になる場合があり，メモリアクセス性能が悪くなる場合がある．

5. What(何を)は，原因となる資源を究明

性能トラブルが発生した時には，What(何を)に相当するソフトウェアが使用する資源を特定する必要がある．性能トラブルの事象の原因は多種多様ではあるが，What(何を)に相当する資源の観点で分類すると大きく6種類になる．図3は，性能トラブルの事象の原因をWhat(何を)の観点で分類したものである．

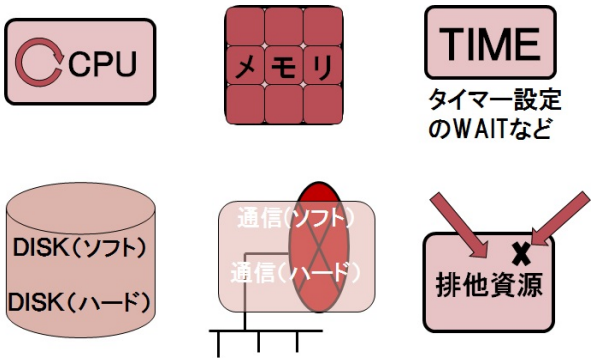

図 3. What(何を)の原因となる資源を究明時の観点

5. 1 性能トラブルがどのような資源で発生しているかの観点

性能トラブルが発生しているということは，性能が思ったように出ていないということであり，要因として資源が不足あるいは資源が競合していることが考えられる．不足するかは元の資源の能力との見合いなので，ハードスペックも確認する必要がある．究明するには，いろいろな資源に関係する性能資料を採取して解析すると有効である．

性能トラブルが発生している時は，資源として大きく「CPU」，「メモリ」，「TIME」，「DISK」，「通信」，「排他資源」に分類できる．性能トラブル解析時には，この 6 つのどの資源が原因で発生しているかを特定する必要がある．

5. 2 原因となる資源の絞り込み方法について

「CPU」，「メモリ」，「DISK」と「通信」については，「ハードウェアの能力」を「システムが使用する量」が超えることはできない．このため，最初にハードウェアのスペックと OS の各種統計情報を確認する必要がある．「TIME」は，WAIT などによって，ある意味「時間」という資源を消費していると考えて「資源」の 1 つとした．「排他資源」も WAIT で待つが競合するものが時間ではないので別物として扱う．

5. 2. 1 ハードウェアスペックを確認する．

(1) CPU を確認する．

・CPU の種類，CPU 数，コア数，クロック性能と CPU の 1 次/2 次のキャシュサイズを確認する．

・高温環境下だとクロック性能が抑制される場合があるので，動作温度を確認する．

・VM の場合には同一筐体内の他の OS の CPU 使用率も確認し，該当 VM に割り当てられているコア数を確認する．

・SMP(Symmetric Multi Processor)構成のマシンかを確認する．4 台のサーバを 1 個の OS で 1 個のマシンのように扱うため，OS 自身の排他処理の時はすべての他のマシンも含めて止まってしまうので4倍の性能にはならないことが多い．

(2)メモリを確認する．

・メモリの種類，メモリのサイズを確認する．

・SMP(Symmetric Multi Processor)構成のマシンかを確認する．4 台のサーバを 1 個の OS で 1 個のマシンのように扱うため，他のマシンの CPU のメモリキャッシュまで同期が必要になるので効率が悪くなることがある．

(3)DISK を確認する．

・DISK 単体の記録密度，回転速度，転送速度を確認する．

・DISK のキャッシュサイズを確認する．

・DISK のキャッシュのヒット率を確認する．

・DISK の接続方法を確認する．
SCSI，SAS，DAS，ファイバチャネル等．

・NAS であるかを確認する．

・DISK 装置とのケーブルの転送速度を確認する．

・DISK 装置の構成を確認する．
例えば，接続形態がデイジーチェイン(daisy chain)方式かを確認する．

・RAID の種類を確認する．

・RAID のキャッシュサイズを確認する．

・ディザスタ構成と方式を確認する．
実現方式がハードウェアかソフトウェアかを確認する．また，同期か非同期かも確認する．

・VM の場合には複数の VM から 1 つの DISK の玉にアクセスする構成の場合には，他 VM の DISK I/O の状況も確認する．

(4) 通信を確認する．

・論理的な通信速度を確認する．

・通信の帯域を確認する．

・通信相手との距離を確認する．

・VM で同一筐体内の通信かを確認する．

・NIC の情報を確認する．

・接続ケーブルの種類を確認する．
10BASE，100BASE，1000BASE など

・無線の場合は，周波数を確認する．

・途中のルーターなどで「SYN ACK」をシミュレートする装置かを確認する．

・ネットワーク構成図を確認する．

(5)画面表示に関係するものを確認する．

・グラフィックボードや GPU を確認する．

(6)プリンタを確認する．

・印刷がある場合には，プリンタのスペックを確認する．

(7)その他

・上記以外の入出力がある場合には，ハードウェアのスペックを確認する．

5. 2. 2 OS の性能情報から状況を確認する．

(1) OS の CPU 情報を確認する．

・システム全体の CPU 使用率とプロセス毎の CPU 使用率を確認する．

・各プロセスのプライオリティを確認する．

・VM の場合には同一筐体内の他の OS の CPU 使用率も確認する．
VM の設定によっては，VM の CPU 数が固定ではなく，最大 CPU 数と最少 CPU 数のような形式で指定できるものがある．

この場合には，他のVMが使用しているCPU量に応じてダイナミックにCPU数が変化するので注意が必要である．

⑵ OSのメモリの情報を確認する．
- ・システム全体のメモリ使用量を確認する．
- ・プロセス毎のメモリ使用量を確認する．
  時間の経過とともに増減がある場合には，傾向も確認する．
- ・システム全体のページング状況を確認する．
- ・プロセス毎のページフォルト状況を確認する．
- ・共用メモリの状況を確認する．
- ・仮想メモリのサイズを確認する．

⑶ OSのDISK情報を確認する．
- ・プロセス毎のDISK I/Oの読み込み回数とバイト数と書き込み回数とバイト数を確認する．
- ・OSのDISK I/OのI/Oチェーン数を確認する．
- ・DISK I/Oのキャッシュのヒット率を確認する．
- ・DISKがビジーな状態かを確認する．
- ・VMの場合には，他のVMマシンのDISK情報も確認する．

⑷ OSの通信情報を確認する．
- ・OS全体の通信速度を確認する．
- ・NICやワイヤレスネットワーク毎の通信量を確認する．
- ・レスポンス時間を確認する．
- ・OSのバッファサイズを確認する．
- ・ポートの使用状況を確認する．
- ・実際に最大でどのくらいの通信速度までででるかを確認する．
- ・ネットワークトレースを採取し，内容を確認する．

⑸ 時間(=WAIT)の確認をする．
- ・WAITにより，長時間止まっていないかを確認する．プロセスのCPU使用率が0に近い値になっている．
  スナップダンプなどを取得し，WAITで止まっているかを確認し，止まっている理由がタイマー，排他資源，I/O，OS関数なのかを確認する．タイマーの場合には，WAIT時間を確認する．
- ・短い時間のWAITが頻発しているかをログ等から確認する．ログがなくCPU使用率が予想より低ければ，スナップダンプなどを複数回取得し，WAITしている場所が毎回同じで止まっているかを確認し，止まっている理由がタイマー，排他資源，I/O，OS関数なのかを確認する．同じ場所の場合には，WAITが頻発している可能性がある．WAITの理由がタイマーの場合には，WAIT時間と頻発する原因を確認する．

⑹排他資源を確認する．
- ・長時間WAITにより，長時間止まっていないかを確認する．プロセスのCPU使用率が0か，0に近い値の場合には，デッドロックの可能性がある．デッドロックは，2つのプロセスだけでなく3つ以上のプロセスの場合もあり，同じ排他資源を使用するプロセスの洗い出しが必要である．
  スナップダンプなどを取得し，WAITで止まっているかを確認し，止まっている理由が排他資源かを確認する．デッドロックであれば，同様の現象が発生している他のプロセスを探し，排他待ちであるかを確認する．
- ・上記と同様の時に，資源の確保と解放で，排他資源名を間違えていないかを確認する．
- ・短い時間のWAITが頻発しているかをログ等から確認する．ログがなくCPU使用率が予想より低ければ，スナップダンプなどを複数回取得し，WAITしている場所が毎回同じで，WAITで止まっているかを確認し，止まっている理由が排他資源かを確認する．同じ場所の場合には，WAITが頻発している可能性がある．排他資源が他のプロセスと競合していないかを確認する．
- ・排他資源に対して，必要以上に頻繁に排他したり，排他範囲が広すぎないかを確認する．

6. When(いつ)は，原因となるきっかけを究明

性能トラブルが発生した時には，When(いつ)に相当するソフトウェアが性能トラブルを発生するきっかけを特定する必要がある．性能トラブルの事象の原因は多種多様ではあるが，When(いつ)に相当するきっかけの観点で分類すると大きく5種類になる．図4は，性能トラブルの事象の原因をWhen(いつ)の観点で分類したものである．

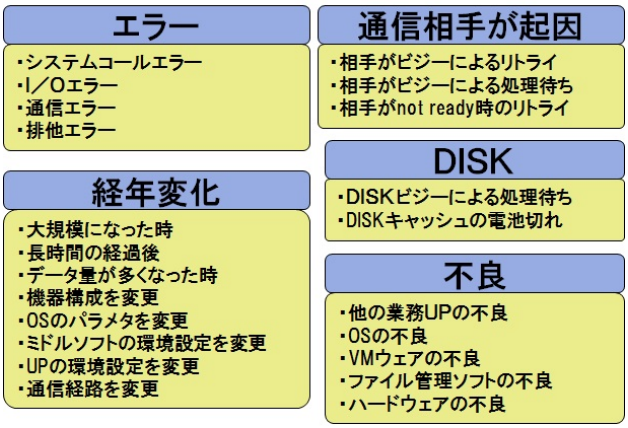

図4. When(いつ)の原因となるきっかけの究明時の観点

6. 1 性能トラブルが発生したきっかけの観点

性能トラブルの発生は，今までは問題なく動作していたが，あるきっかけを契機に発生することがある．

昨日までは，問題なく動作していたものが，なぜか突然に性能が悪くなることは，要因としてなんらかの変化があったと考えられる．昨日と何が変わったのかは，いろいろなログ，操作履歴や構成変更履歴に関係する資料を集めて解析すると有効である．

性能トラブルが発生するきっかけとしては，大きく「エラー」，「通信相手が起因」，「経年変化」，「不良」，「DISK」に分類できる．性能トラブル解析時には，この5つのきっかけが原因で発生しているかを特定する必要がある．

6. 2 原因となるきっかけの絞り込み方法について

⑴ エラーを確認する．

エラーの発生の有無は，性能トラブルが発生した製品のログだけでなく，システムのログも確認する．エラーの発生がいつからか，何回発生しているかを確認し，性能トラブルとの関係の有無を確認する．場合によってはハードウェアのログも確認する．エラーの種類としては，大きく以下の4種類になる．

- ・システムコールエラー
- ・I／Oエラー

・通信エラー
・排他エラー

(2) 通信相手が起因しているかを確認する.

他のマシンと通信があるプログラムの場合, 性能トラブルは製品が動いているマシンだけを調査しても判明しないことがある. 通信相手や通信の状況に起因して性能トラブルが発生することがある. 次の 3 つの観点で調査して, 通信相手が起因しているかを追究し, 場合によっては通信相手側のプログラムの調査を実施する.

・相手がビジーによるリトライ
・相手がビジーによる処理待ち
・相手が not ready 時のリトライ

(3) 経年変化の要因を確認する.

性能トラブルは, 突然発生する場合もあるが, 時間の経過とともに少しずつ遅くなることもある. 少しずつ遅くなる場合には, 気づくのが遅くなることが多く, 原因究明も難しくなることが多い. これは, 長い時間の間には変化する対象が多くなるため, 原因究明が難しくなる.

・大規模になった時
・長時間の経過後
・データ量が多くなった時
・機器構成の変更
・OS のパラメタを変更
・ミドルソフトの環境設定の変更
・UP の環境設定の変更
・周辺装置の変更
・通信経路の変更

例えば, LAN と無線の両方を使用している場合に, LAN が有効な状態でもある時から無線が使われることがある.

・DISK のフラグメンテーション
・メモリのフラグメンテーション
・ハードウェアのファームウェアの変更

(4) DISK の状況を確認する.

最近は, 複数の DISK が複数のコンピュータと接続されているケースが増えている. このため, 1 つの DISK が複数のコンピュータからもアクセスされることがあり, 他のコンピュータの I/O が頻繁にあると影響を受けることがある.

・DISK の接続構成の確認
・DISK ビジーによる処理待ち
・DISK キャッシュのヒット率
・DISK キャッシュの電池切れ

(5) 各種不良がないかを確認する.

性能トラブル発生時でも, OS やハードウェアなどの既知の不良がないかを確認する.

・他の業務UPの不良
・OS の不良
・VM ウェアの不良
・ファイル管理ソフトの不良
・ハードウェアの不良

7. Whom(誰によって)は, 原因を誘発したものを究明

性能トラブルが発生した時には, 自分のプログラムに関してだけ調査しても判明しないことがある. 時々ではあるが, 自分ではなく他からの影響を受けている場合があり, Whom(誰によって)を特定する必要がある. 性能トラブルの事象の原因は多種多様ではあるが, Whom(誰によって)から誘発されたかの観点で分類すると大きく3種類になる. 図5は, 性能トラブルの事象の原因を Whom(誰によって)の観点で分類したものである.

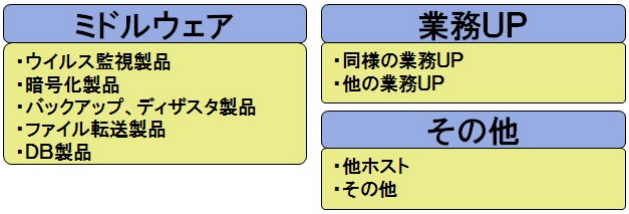

図 5. Whom(誰によって)の原因を誘発したものの究明時の観点

7. 1 性能トラブルの原因を誘発しているものの観点

自分のプログラムの性能が悪いとどうしても, 自分のプログラムに問題があり, その原因究明に注力してしまい, 問題が見つからずに長期化することがある.

このため, 性能トラブル発生時には, 問題が発生している自分のプログラムだけでなく, システム全体の性能情報の解析から始めることが大切である.

性能トラブルを誘発する原因としては, 大きく「ミドルウェア」, 「業務 UP」, 「その他」に分類できる. 性能トラブル解析時には, この 3 つの他の誰かによって誘発されて発生しているかを特定する必要がある.

7. 2 原因となる他の誰によって誘発されているのかの絞り込み方法について

(1) ミドルウェアを確認する.

同一マシンにミドルウェア製品がインストールされている場合には, ミドルウェアが大量のリソース(CPU, メモリ, DISK I/O, 通信)を消費している場合があり, その影響を受けて性能トラブルが発生している場合がある.

・ウイルス監視製品

UP の I/O や通信の延長で動作する場合があり, パターンファイルの更新を契機に性能遅延が発生することがある.

・暗号化製品

DISK I/O の延長で書き込み時の暗号化と読み込み時の複合化処理が動作することがあり, データサイズが大きいと CPU を消費することがある.

・バックアップ製品

バックアップ製品実行中は, 高速化のためにデータサイズを非常に大きくして書き込むことがある. このため RAID のキャッシュなどを大量に消費し, UP の I/O のキャッシュのヒット率が低下するなどの影響を受けることがある.

・ディザスタ製品

ディザスタ製品は, 同期/非同期, ハードウェアによる実装だけでなく, ソフトウェアによって実装されるものもある. 特に, 同期の実装方法によっては, 性能に大きく影響する場合がある.

・ファイル転送製品

ファイル転送製品実行中は，通信がビジーになる可能性が高くなる．製品によっては複数のコネクションを接続して同時に通信する製品もあり，UP が使用できる帯域が小さくなり，通信の待ちキューも長くなる場合がある．

・DB製品

DB 製品実行中は，大量の I/O だけでなく，同時に大量に CPU やメモリを消費することがある．複雑な SQL などを並行で実行すると，インコア処理が並行して実行するので，複数の CPU 資源を大量に消費することがある．

(2) 業務 UP を確認する．

同一マシンには，いろいろな業務 UP が動作しており，他の業務 UP が大量のリソース(CPU，メモリ，DISK I/O，通信)を消費している場合があり，その影響を受けて性能トラブルが発生している場合がある．

・類似の業務 UP

業務 UP の中で似ている UP は，消費するリソースも似ている傾向があり，同時に多くの類似した UP を実行すると，性能トラブルが発生することがある．特に，排他資源で同一の資源をシリアライズしている場合には，性能トラブルが発生し易いので注意が必要である．

・異なる業務 UP

1 つのシステムでも，実行される業務 UP は，いろいろとありかつ，多数の業務 UP が同時に実行されていることがある．いつもであれば，同時に実行されることがない UP でも，データ件数の変化などにより，同時に実行されてしまうこともある．そのような時に，I/O が集中してしまうことがあったり，同一 DISK の同一ドライブに負荷がかかることがある．性能トラブル発生時は，自分の UP だけでなく，同時期に実行している他の UP も調査する必要がある．

(3) その他．

・他ホスト

他ホストと連携して処理するような UP の場合には，他ホストからのレスポンスが悪いと，性能トラブルが発生することがある．

・ウイルスソフトに感染している

ウイルスソフトに感染するといろいろなファイルへのアクセスや通信が発生し，性能トラブルを誘発する．

・その他

Dos 攻撃や Ddos 攻撃を受けている場合には，OS などがその対応で CPU を消費し，UP が CPU を十分に使えなかったり，通信速度が低下するなど性能トラブルが発生することがある．

8. 性能トラブル発生時の具体的な調査方法

はじめにで記述したように，本論文を作成する前に現象などから調査ポイントを段階的に実施すれば原因究明できるような資料として，「性能トラブル解決の手引き−事例編」としてソフトウェア・メインテナンス研究会で作成したものがある．

123個の事例を表形式でまとめたものである．表には，項目として「現象」，「調査ポイント 1」，「確認結果 1」，「調査ポイント 2」，「確認結果 2」，「原因」，「調査場所大分類」，「調査場所中分類」，「調査場所小分類」，「観点」，「調査観点」，「備考」などがあり，現象から調査ポイントの確認結果の内容に応じて段階的に絞り込むことで，原因を究明するものである．表 1 は，「性能トラブル解決の手引き−事例編」の抜粋である．

「性能トラブル解決の手引き−事例編」の全文は，ソフトウェア・メインテナンス研究会の HP に掲載してある．

http://www.serc−j.jp/

また，ログなどの調査を開始すると同時にシステム変更点，運用の変更点なども同時に確認する．

(1)システム変更の確認

・ハードウェア構成の変更
・新たに導入したプログラムの有無の確認
・ソフトウェアのアップデートの確認
・ソフトウェアのパラメタ変更の変更

(2)運用の変更の確認

・いつもと違うオペーレションを実施していないかの確認
・週末・月末などの業務量などの変化の確認

	現象	調査ポイント1	確認結果1	調査ポイント2	確認結果2	原因	調査場所大分類	調査場所中分類	調査場所小分類	観点	調査観点	備考
55	処理が重い	メモリ使用量の確認	メモリ使用量が急激に増加しているかを確認する。	(1)メモリサイズの時間の経過による推移を確認する。 (2)スナップダンプによる解析(メモリの使用状況の確認) (3)ソースでメモリの確保、開放処理の洗い出し。 (4)システムコールトレースでメモリ確保関数の回数の確認	(1)メモリサイズの変化とシステムコールトレースから、大きなメモリサイズの確保が、小さなサイズを大量に確保したかを確認する。 (2)ダンプを解析し、確保したメモリサイズと数を確認する。 (3)ソースでメモリの確保時のサイズと回数を確認する。	(1)メモリの確保サイズが巨大。 または (2)メモリの確保する個数が膨大	ソース	メモリの確保/開放のロジック		メモリの確保/開放	時々だが急激に重くなるような状況があるかを確認。	OSによっては、大きなメモリを確保しても、実際にメモリへの書き込みがないと実メモリを割り当てないケースがあるので、大量のメモリへの書き込みを同時に実行しないと発生しないこともある。
56	処理が重い	メモリ使用量の確認	メモリ使用量が急激に増加しているかを確認する。	(1)メモリサイズの時間の経過による推移を確認する。 (2)スナップダンプによる解析(メモリの使用状況の確認) (3)ソースでメモリの確保、開放処理の洗い出し。 (4)システムコールトレースでメモリ確保関数の回数の確認	(1)メモリサイズの変化とメモリの確保/開放のシステムコールが大量に発行されていないかを確認する。 (2)ダンプを解析し、確保したメモリサイズと数を確認する。 (3)ソースでメモリの確保時のサイズと回数を確認する。	データコピー時にコピー先の領域を1バイトずつ拡張しながらコピーしている。 データサイズが大きくなると指数関数的にメモリ処理量する。またCPUも消費し処理に時間がかかる。	ソース	メモリの確保/開放のロジック		メモリの確保/開放	メモリ確保のシステムコールが大量に発行されていないかでわかる	speedが関数などだと関数の延長でこのような実装がされているので注意が必要。
57	処理が重い (長時間経過後に時間の経過と共に遅くなる)	CPU使用率の確認 UNIX,Linuxの場合には、sysとusrの比率も確認	CPU使用率が増加しているかを確認する。	(1)スナップダンプによる解析(メモリの使用状況の確認) (2)ソースでメモリの確保、開放処理の洗い出し。 (3)システムコールトレースでメモリ確保関数の回数の確認	(1)ダンプを解析し、メモリが断片化していないかを確認する。 (2)ソースでメモリの確保時のサイズより小さいサイズで開放するなどの処理がないかの確認。 (3)メモリの確保サイズが可変で頻繁に実行されていないかの確認。	メモリの断片化。 100kバイトを確保し90Kバイトだけ解放のような処理を繰り返すと、メモリが細片化してメモリの確保に時間がかかるようになる。)	ソース	メモリの確保/開放のロジック		メモリの確保/開放	・自分のソースのメモリの確保/解放処理 または、使用しているライブラリの延長で発生していることもあるので、注意が必要。	・メモリ使用量(見た目の使用メモリは少なくてもフラグメンテーションのためメモリ不足エラーになることがある)
58	処理が重い 但し一時的 (長時間経過後に時々急激に遅くなることがある)	CPU使用率の確認 UNIX,Linuxの場合には、sysとusrの比率も確認	CPU使用率の時間の経過による推移を確認する。	時々CPU使用率が高い状態が発生し、やがて解消されるような傾向があるかを確認する。	(1)一時的に重い時に、CPU使用率が以上に高いかを確認する。 (2)OSがJavaのようなメモリのガーベージコレクションが実行していないかを確認する。 (3)ダンプ解析でメモリの内容を確認する。	OSのガベージコレクションが実行されている。	OS	ガベージコレクション処理の実装方法		ガベージコレクション処理	時々だが急激に重くなるような状況があるかを確認。	
59	処理が重い 但し一時的 (長時間経過後に時々急激に遅くなることがある)	CPU使用率の確認 UNIX,Linuxの場合には、sysとusrの比率も確認	CPU使用率の時間の経過による推移を確認する。	時々CPU使用率が高い状態が発生し、やがて解消されるような傾向があるかを確認する。	(1)一時的に重い時に、CPU使用率が以上に高いかを確認する。 (2)Javaのメモリのガーベージコレクションが実行していないかを確認する。 (3)ダンプ解析でメモリの内容を確認する。	Javaのガベージコレクションが実行されている。	Java	ガベージコレクション処理の実装方法		ガベージコレクション処理	時々だが急激に重くなるような状況があるかを確認。	使用しているJavaによって挙動が異なる。
60	処理が重い 突然遅くなる	CPU使用率の確認	CPU使用率が高い状態が続く	(1)システムコールトレースで通信の回数を確認する。 かつ (2)該当プロセスの通信量を計測する。 (3)ソースを解析し、通信処理の周辺のロジックを確認する。 かつ (4)通信相手(他ホストの場合もある)のプロセスも高負荷状態にあるかを確認する。	(1)通信の回数が非常に多いかを確認する。 (2)通信のデータ量が多いかを確認する。 (3)通信相手が受け取れない時に、再送処理があり 輻輳状態になるロジックがあるかを確認する。 (4)通信相手のプロセスがビジー状態で通信を受け取れない状況であるかを確認する。	他プロセスとのデータの受け渡しがあるが、他プロセスが処理しきれないデータを戻す処理があり 2つのプロセス間の通信処理で輻輳が発生している。	ソース	通信処理のロジック		輻輳	件数とサイズを変化させて性能測定し、グラフ化する。 ・いろいろな件数で処理時間の計測	通信相手のプロセスが処理しきれている間は、問題が発生しない。受信プロセスの処理性能 < 送信プロセスの送信量 の状態が続くと限界を超えると急激に遅くなる。

表1. 「性能トラブル解決の手引き−事例編」の抜粋

9. 性能向上する時の観点

性能トラブルが発生した時にデッドロック等のような明らかな不良であれば，不良を修正すればよいのだが，レスポンス向上や大規模に対応するために性能向上したい時に，どのような手法があるかの観点毎に説明する．

性能向上には，ハードウェアによる対策，ソフトウェアによる対策とハードウェアとソフトウェアの両方の対策がある．

How(どのように)の観点で見直すことと，What(何を)の観点のリソースを上手に使用することに着眼して見直すとよい．場合によってはリソースを贅沢に使用するようにして，ハードウェアを増強するという方法もある．

最近は，ハードウェアの価格が安くなってきているので，CPU の増設やグレードアップ，メモリの増設，DSIK の増設やキャッシュの増設，通信の高速化，帯域の拡大などで性能向上が可能であれば，プログラムの修正コストより安い場合もあり，ミスの危険もなく確実である．

ソフトウェアの対策には，以下のものがある．

⑴多重度を上げる．
- ・マルチプロセス化．
- ・マルチスレッド化．
- ・マルチインスタンス化．
- ・並行処理化．

⑵分散して処理する．
- ・複数エージェントでの分散実行．
- ・ロードバランサーでの分散実行．

⑶非同期にして処理する．
- ・I/O などの非同期処理．
- ・自立実行．
- ・見た目と実体の非同期処理．
- ・先読み処理．

⑷独自にキャッシュを実装する．
- ・キャッシュにより実 I/O より書き込み時間を短くする．
- ・キャッシュにより実 I/O より読み込み時間を短くする．

⑸まとめて処理する．
- ・I/O 時のレコードのブロッキング化．
- ・I/O サイズを大きくする．
- ・要求をまとめて処理する．

⑹整理する．
- ・デフラグする．
- ・ガベージコレクションする．
- ・差分のみの更新する．
- ・データは，ソート処理する．

⑺階層化
- ・Manager と Agent の構成にする．
- ・中継サーバーを構成する．
- ・インデックスを作成する．

⑻事前に準備する．
- ・先読み処理をしておく．
- ・プロセスを常駐化する．
- ・実 I/O 時に先読みしてキャッシュに入れておく．
- ・まとめて読み込みを行う．

⑼排他処理の見直しをする．
- ・排他時間を短くする．
- ・排他をしないようにする．
- ・排他範囲を小さくする．
- ・排他回数を減らす．
- ・排他エラー時のリトライ間隔を短くする．

⑽タイマー処理を見直しする．
- ・タイマー時間を短くする．
- ・タイマーのかける回数を減らす．
- ・タイマーをかけなくする．(非同期にするなど)

⑾その他
- ・ストリーム処理で実現する．
- ・ハッシュを使用する．
- ・アセンブラ化する．
- ・I/O サイズをハードのスペックに合わせる．
- ・ハードウェアの特性に合わせた実装を行う．
- ・ウイルスチェック対象から外す．

10. このドキュメントの対象について

(1) プラットフォームは，Windows，Linux，UNIX 上で動作するプログラム

(2) 通常のユーザープログラムまたは，ミドルウェアソフトのプログラム

(3)database，Application Server，Web server などのチューニングは対象外

(4) DISK，通信装置，機器構成などのチューニングは対象外

(5) OS のチューニングは，対象外

(6) VM ウェアに関しては，一部だけ対象としている

11. おわりに

個人的な経験をもとに性能トラブル解決の勘所としてまとめたが，まだ網羅性が不十分な状態であるので，この資料をベースに各自で追記して使用して頂ければ幸いである．

謝辞

本研究の資料まとめの際にソフトウェア・メインテナンス研究会の皆様に多くの助言を受けた．ここに謝意を記す．

参考文献

[1] 増井和也， 弘中茂樹， 馬場辰男， 松永真 ″ソフトウェア保守開発:ISO14764 による″ソフト・リサーチ・センター， 2007

[2] 保田 勝通，奈良 隆正 ″ソフトウェア品質保証入門″日科技連出版社，2008

[3] 鈴木勝彦 ソフトウェア・メインナンス研究会 第 24 年次報告書 D グループ報告 個人研究「性能トラブル解決の手引き」