

4 D運用管理とトラブルシューティング

上級管理者のための知識とトラブルシューティング

2012 DevCon

目次

4D 運用管理とトラブルシューティング	1
ベストプラクティスを考える	5
データベース設定	5
コンパイラー	5
データベース	6
Web	6
SQL	8
互換性	9
まとめ	10
Unicode モードを理解する	10
Unicode とは	10
ICU とは	11
4D と Unicode	11
まとめ	12
バックアップと復元	12
バックアップのスケジュール	12
復元	14
ログ管理	17
まとめ	20
ミラー	21
ベストプラクティス	23
メインテナンス	23
標準モードとメインテナンスマード	23
検証	25
ベストプラクティス	28
オペレーティングシステム	28
Windows	28
ベストプラクティス	30
ハードウェア	30
メモリ	30
アプリケーションメモリを理解する	30
仮想メモリ	32

アプリケーションメモリを測定する	33
4D のメモリ使用量を計算する	35
データベースキッシュ	37
Solid-State Drive (SSD)	40
なぜストレージの速度が重要なのか	40
ディスクの速度が何故それほどに重要なのか	41
SSD を理解する	42
SSD のパフォーマンス低下について理解する	44
SSD メモリ: フラッシュセル	46
コントローラー	46
SSD のメインテナンス	49
SSD でもっとも重要な機能	50
SSD を使用する	52
ベンチマーク	53
4D とデスクトップ仮想化	58
RDPCLIP	58
プリンタードライバー	58
クライアントロード	59
避けるべきこと	59
問題解決の考え方	60
4D CACHE LOG ANALYZER	60
キャッシュログと分析	60
4D CACHE LOG RECORDER	62
I. 概要	62
II. インストール	62
III. インターフェースを使用する	62
V. キッシュのクリア	64
VI. 情報の記録を開始する	64
VII. クライアント/サーバー	66
VIII. インターフェースを表示しない	66
IX. デバッグのためのログ	67
X. 生成されたデータの送信	67
XI. コンポーネントのアンインストール	67
STATISTICS OVERVIEW	68
4D INFO REPORT COMPONENT	90
4D Info Report コンポーネントの共有メソッドマニュアル	95
4D Info Report コンポーネントを直接起動する	98
生成されたレポートのフォーマットと内容	103

4D POP DATA ANALYZER	109
4D NETWORK LOG ANALYZER	116
NETWORK LOG ANALYZER を使用してログの作成/読み込みを行う	120
4D ログファイル - ログツール	124
WINDOWS リソースモニター	127

ベストプラクティスを考える

4Dにおけるベストプラクティスについて考える事にしましょう。特に4Dデータベースの運用と管理に関して深く考えて行きたいと思います。もちろん開発に際して、これらの情報が役に立つ事は言うまでもありません。

ここで紹介する開発アプローチやガイドラインは、これまでの開発で得られた経験的なノウハウと相容れないものかもしれません。しかし、上級デベロッパーとしてエンドユーザに優れたシステムを提供するためには、裏打ちされた知識に基づくノウハウであり、ぜひ守ってもらいたいものであることに間違はありません。

データベース設定

4Dには、開発時、運用時、ランタイム実行中に設定できる設定項目があります。まずはこれらについて細かく見ることにしましょう。

とはいっても、ここではすべての設定を紹介するわけではありません。この目的は、ベストプラクティスとなるであろう項目について妥当な情報を提供することにあります。そのためタイムアウトに関するものは大きく省かれています。理由はいくつかあります。

- ・ タイムアウトは一般的に大きなトピックであり、限られた時間でカバーしきれない
- ・ すでにテクニカルノートがある
- ・ タイムアウトについてはネットワークトラブルシュートで解説

また、バックアップについては別に取り上げます。

注

ここでは特に 4D v12 のデータベース設定について取り上げます。

コンパイラ



コンパイラ設定は開発時に使用されるのですが、運用やメインテナンスに関する重要な項目が2つあります。

- ・ コンパイルパスを「自動変数定義は行わない」に設定する
すべての変数は明示的に型宣言されているべきです。この設定により、型情報がない場合エラーを返すよう4Dに指示することができます。これにより運用中に型の衝突が発生する前にそれを発見できるようになります。コンパイル時に型の衝突を見つけることができれば展開前にそれを取り除くことができます。
- ・ 「64-bit プロセッサー用にもコンパイルする」の意味を理解し、適切に設定する
64-bit プロセッサーでアプリケーションを動作させるからといって、このオプションを有効にする必要はありません。この設定は、コンパイルしたファイルを 4D Server 64-bit アプリケーションで動作させる時に必要なものです。この場合のみ、このオプションを選択してください。そうでないのにこのオプションを選択する

こともできますが、64-bit コンパイルコードが追加されるため不必要にアプリケーションのサイズが大きくなっています。

データベース



動的キャッシュの計算について

この設定を選択しても動的にキャッシュサイズが変化するわけではありません。また実行時のリソースの状態に応じてキャッシュサイズを計算するものでもありません。例えばマシンの空きメモリや実行アプリケーションの数を考慮に入れるものではありません。

この設定の目的は単純です。インストールされる環境を事前に開発者が知ることができないようなときに使用するものです。この場合マシンメモリを独占しないよう注意深くキャッシュサイズを設定しなければなりません。使用されるハードウェア構成を事前に知ることができませんから、インストールされたハードウェア構成に基づき、キャッシュ量を計算する仕組みが用意されています。

マシン構成が分かっている場合、動的キャッシュの計算を使用する優位性はなく、もっと言えばキャッシュサイズの設定が不必要に複雑になります。この場合のベストプラクティスは動的キャッシュの計算をオフにして直接値を設定することです。

キャッシュを物理メモリに保持する

「キャッシュを物理メモリに保持する」は無効にします。この設定を選択すると、未知の副次効果が発生するかもしれません。この設定は、第一義的にメモリーク発生を発見するのに役立つものです。なぜならこの設定により 4D はキャッシュ全体の割り当てを約束するよう強制されるからです。設定を無効にすることで項目がキャッシュに追加されるたびに使用されるメモリが動的に増えるようになります。

キャッシュサイズの設定については後ほど解説します。

Web



通常 4D はスタティックなコンテンツのみを配信するわけではありません。つまり純粋な Web サーバーとしてではなく、動的なリクエストに対応してデータにアクセスする Web アプリケーションプラットフォームとして使用されます。このためパフォーマンスは変化し、データベース設定やクライアントロードに依存します。それにしても考慮すべき重要なポイントがあります。

Web アプリケーションのパフォーマンスは変化するため、4D Web アプリケーションの最も重要なベストプラクティスはロードテストを行うということです。Web アプリケーションのロードテストツールはいくつかあります。ツールを選択し、以下のようなテストを行います：

- 適切なターゲットロードを選択する
おそらくこれが一番大切です。Web アプリケーションが想定する利用量を理解し、テストをそれに一致させることが重要です。
- ロードテストを行う
もちろん許容誤差を考慮するためより高いロードでテストすることも推奨されます。

- 必要に応じて微調整を行う
詳細は後述。

代わりの方法として上限値をテストすることもできます:

- サーバーハードウェアを決定します。このマシンを元に適切なパフォーマンスの上限値を決定します。
- サーバーが応答しなくなるまでサーバーのロードテストを行う。
- サーバーがその上限に達しなくなるよう調整を行う。

4D アプリケーションに対するロードが問題になる場合、プロクシ、ロードバランス、リライトなどの技術を検討します。

プロクシやリライトを使用すれば、例えば 4D がスタティックコンテンツを配信する必要がなくなります。これによりアプリケーションは動的なコンテンツを処理すればよいだけになります。これはつまり 4D を他の Web アプリケーション環境に統合できることを意味します。

ロードバランスは、ひとつの 4D インスタンスではロードを処理できない場合に考慮します。主な選択肢は 2 つあります:

- 複数の 4D アプリケーションのコピーをホストする
主な欠点は各コピー間で同期をとる必要があり、複雑さが増すことがあります。
- 4D Server を使用し、Web リクエストを 4D クライアントで処理します。2 つの利点があります:
データの同期をとる必要がありません。
各クライアントからの DB4D リクエストはサーバー上でプリエンプティブに処理されます。

他にパフォーマンス改善の選択肢としてパーティショニングがあります。

4D Server を目的毎に別々に分割して運用します。分割の方法は色々とあると思いますが、例えば会員管理を行うデータベースと商品管理を行うデータベースを分け、負担を分散させることもできるでしょうし、商品の概要だけを取り扱うデータベースと、詳細情報だけを取り扱うデータベースに分割してしまう方法もあると思います。

4D Web キャッシュ

4D Web キャッシュは以下の制約下でスタティックなコンテンツをキャッシュするために使用します。

HTML、スタイルシート、100KB 未満の GIF や JPEG のみがキャッシュされます。
キャッシュがいっぱいになり追加のスペースが必要な場合、4D は最も使用されないコンテンツの中からもっとも古いものをアンロードします。

必要に応じて 4D Web キャッシュを使用します:

デフォルトでオフになっています。
開発時は使用しないほうがよいでしょう。
スタティックコンテンツのサイズに応じてサイズを設定します。ただしマシンリソースを考慮してください。すべてのスタティックコンテンツを保持できる値に設定するとよいですが、マシンのリソースを食いつぶさないようにしなければなりません。

WEB CACHE STATICICS や /4DSTATS url でキャッシュの利用状況を取得できます。

Web プロセス

Web プロセスについて心に留めておくべき一番重要なことは、Web プロセスが4Dプロセスであり、コオペラティブだということです。これを踏まえると、次のように言えます。

4D Server を Web サーバーにしている場合、CPU/コアの数を増やすことに利点はありません。4Dプロセスとして正しく振る舞うためのテクニックが、Web プロセスでも必要です。

- 他のプロセスに制御を渡すために IDLE コマンドを使用
- 必要であれば DELAY PROCESS を使用
- 同期呼び出しを避ける

同期呼び出しによりアプリケーション全体がロックされてしまうかもしれません。例えばすべてのプラグイン呼び出しは同期的です (プラグインが 4D に制御を渡すことができるとしても)。LAUNCH EXTERNAL PROCESS もデフォルトでは同期的です。

Web プロセスに関する注意点

- 非動作プロセスのタイムアウト
サーバリソースを考慮したうえでこの設定を調整します。常に利用可能にしておくプロセスの数とメモリの利用量とのバランスを取らなくてはなりません。プロセスの作成には時間がかかり、プロセスの保持はメモリを消費するからです。後述のメモリの節を参照してください。
- 最大同時 Web プロセス
これは DOS 攻撃によるサーバーのオーバーロードを避けるのに有用です。念のために言えば、4D を使用して DOS 攻撃を防ぐことはできません。それはここでのポイントではありません。ポイントは DOS 攻撃により 4D 自身が落ちる事態になることを避けることがあります。Web サイトは落ちるかもしれません。しかし 4D は他のユーザー (クライアントや SQL アクセス) に応答することができます。
適切な値を設定しなければなりません。
 - 低すぎると HTTP 503 レスポンスが返されます。
 - 高すぎるとサーバーに悪影響があります。
 - デフォルトは 100 です。100 同時リクエストを予定していますか？

テキスト変換

文字コードは UTF-8 に設定しましょう。

Unicode に移行することが重要です。特に複数言語をサポートする場合はなおさらです。Web の場合 UTF-8 が効率的です。HTML タグで使用されるような一般的な文字が 1 バイトで構成されるため、データ転送量が節約されます。UTF-8 は Web のデファクトスタンダードで、最大のサポートと互換性が保たれます。

Keep-Alive 接続

この設定を使用してサーバーへの TCP 接続を制御します。Web リクエスト処理ではありません。ベストプラクティスは IT インフラに基づくものであり、4D 側に影響を与えるものではありません。言い換えれば Keep-alive 接続は TCP/IP のコンセプトであり、4D や Web のコンセプトではありません。Keep-alive については 4D ではなく当該のリファレンスを参照してください。

SQL



SQL サーバー公開

起動時に SQL サーバーを公開という設定がありますが、4D SQL サーバーは常に実行されている点に注意してください。この設定は、SQL サーバーを外部に公開するかどうかを決めるに過ぎません。このダイアログに表示されているポートは、常に 4D Server により使用されています。SQL サーバー接続は 4D リモート接続の重要な部分です。

この設定を切り替えることで、4D データベースへの外部アプリケーションからの接続を起動時に許可できます。4D ODBC ドライバーや4D間の SQL 接続を行うために、この設定を使用しなければなりません。これらの接続が必要ない場合、この設定を非選択にするべきです。しかしそれにより SQL サーバーが無効になるわけではありません。

この設定を有効にしたら、セキュリティのため、ドキュメントに基づき On SQL Authentication データベースメソッドを実装すべきです。

互換性



注

互換性ページは過去のバージョンから変換した場合のみ表示されます。

クライアント／サーバー

クライアント上ではなく、サーバー上でフォーミュラを実行させる設定が複数あります。パフォーマンスと最適化の観点からこれらの設定はとても重要です。これらの設定は QUERY BY FORMULA などのコマンドを 4D がどのように処理するかに影響します。

互換モードにおいて、フォーミュラはクライアント上で評価されます。クエリ実行時、各レコードはクライアントに送信され評価されます。これはとても非効率的です。

設定を有効にするとフォーミュラはサーバー側で評価されます。結果のみがクライアントに送信されます。

デフォルトでこの設定が有効になつてない理由は、実行コンテキストが変わることで結果に影響する可能性があるためです。例えばフォーミュラが 4D メソッドで、クライアント上のプロセス変数の値が評価に使用されている場合などです。

QUERY BY FORMULA は今まで遅く非効率的なコマンドとして知られていました。特にテーブルリレーションを使用する場合は顕著です。互換性設定を使用すれば QUERY BY FORMULA で SQL JOIN を使用することができるようになり、クロステーブルの検索パフォーマンスが劇的に向上します。

重要

4D v13 では、Web のコンテキストモードオプションが完全に取り除かれました。この機能の使用は、止めなければなりません。

Unicode モードはとても重要なので、別に解説します。

まとめ

- タイムアウトについては Mastering 4D Timeouts & Connectivity Settings を参照してください。
- コンパイルパスには「自動変数定義は行わない」を設定する。
- 4D Server v12 64-bit を利用するときのみ「64-bit プロセッサー用にもコンパイルする」を有効にする。
- 「動的キャッシングの計算」はマシン構成をコントロールできない場合にのみ使用する。
- 「物理メモリにキャッシングを保持する」はトラブルシューティングのみに使用する。
- Web アプリケーションはロードテストする。
- Web アプリケーションで必要に応じてプロクシ、リライト、ロードバランス、分離テクニックを使用する。
- 4D Web キャッシュを使用する(開発時を除く)。
- 予定するロードにマッチするよう、Web プロセス設定を調整する。
- Web 文字セットには UTF-8 を使用する。
- ネットワークの状況に応じて Keep-Alive を設定する。
- SQL サーバー公開設定は SQL サーバー自体を有効/無効にするわけではありません。外部からの接続のみを制御する。
- 外部接続を許可する場合は On SQL Authentication を常に実装する。
- フォームュラを使用するコマンドを実行する場合は互換性設定を調整すると、フォームュラがサーバー上で実行され、パフォーマンスが向上する。
- QUERY BY FORMULA を実行する場合は互換性設定を調整して JOIN を使用できるようにすると、パフォーマンスが向上する。
- Web のコンテキストモードは使用しない。

Unicode モードを理解する

4D v11 SQL からは文字エンコーディングに Unicode 標準が使用されるようになりました。さらに 4D v11 SQL Release 2 からはソート、クエリ、データ比較に ICU コレーションアルゴリズムが使用されるようになりました。ここでは 4D で Unicode を使用する際の重要なポイントについて説明します。

注

この内容はテクニカルノート 11-22 *ICU in 4D: Impact on Queries, Sorts and String Comparisons* から取られています。

Unicode とは

Unicode は一連の統合文字コードであり、世界中の標準的な文字を実質的に管理しようとするものです。Unicode はモールス符号や ASCII のような文字符号化システムで、文字に対し U+nnnn の形式でコードを割り当てます。エンコーディングを統一することで異なる国間でデータ交換が可能となります。

US-ASCII 文字のコードは 1-127 の間に含まれているのに対し、Unicode の上限は 65,000 を超えます。

Unicode のコード化にはいくつかの方法があります。例えば:

- UTF-16 は 16-bit 整数を使用。
- UTF-32 は 32-bit 整数を使用。
- UTF-8 は 8-bit 整数を使用。UTF-8 ではある種の可逆変換を提供します(後述)。

ASCII が 1 文字に 1 バイト消費するのに対し、UTF-16 では 2 バイトを使用します。UTF-16 の最初の 256 文字は ASCII で定義されたものとほぼ同じですが、コード可能な拡張文字のコーディングにはシステムゾーン(この場合連続した範囲、面、そしてブロック)を使用します。

異なる目的に適した複数の形式が Unicode には存在します。現在広く使用されているのは UTF-8 で、これは可変サイズのエンコーディングです。特定の Unicode 文字によって文字は最大 4 バイトのグループで表現されます。これにより ASCII 使用する場合システムは文字をそのまま使用できますが、より複雑なコードを必要とする場合は、必要に応じて Unicode 文字を呼び出すことができます。

ICU とは

International Components for Unicode (ICU) はオープンソースプロジェクトであり、Unicode、様々なタイプのデータ交換、そしてソフトウェアのグローバル化をサポートする C/C++ および Java ライブラリを提供します。ICU はすべてのプラットフォームそして C/C++ と Java ソフトウェアにおいて同じ結果と振る舞いを提供します。

ICU を使用することで Unicode 文字列、文字プロパティ、文字セット変換、正規表現などを処理できるようになります。

ICU は文字、単語、行の切れ目、さらには大文字小文字を区別した文字列比較、コレーション、検索なども管理します。さらに ICU は正規化や大文字小文字、文字体系の変換(ある文字体系システムから他のものにテキストをシステムチックに変換する方法)なども可能にします。

さらに ICU は複雑なテキストレイアウト(アラビア語、ヘブライ語、タイ語など)や、日付、時刻、数字、通貨、ルールに基づくメッセージなどのフォーマット化や解析もサポートします。

4D と Unicode

4D v11 SQL より前では、4D はサポートする各言語ごとに文字セットを含んでいました。文字セットは文字と数値を対応付けるものです。異なる文字セットでは同じ数値に異なる文字が割り当てられることがあります。4D は起動時にシステム言語に対応する文字セットを選択しました。1 つの数値に異なる文字が割り当てられるため、同時に複数の言語を管理することは不可能でした。

4D バージョン 11 より、すべての文字データは UTF-16 で格納されます。同時に 4D はランゲージオブジェクトに UTF-16 を使用します。通信の場合は例外で、この場合は UTF-8 を使用します。UTF-8 は A-Z や 0-9 などの文字がコンパクトに表現されるからです。

互換モードでの 4D の動作はほぼ以前のバージョンの 4D と同じです。つまり文字データは各言語のデフォルト文字セットを使用しているものとして扱われます。しかしこれを実現するため、互換モードでは必要に応じて UTF-16 とデフォルト文字セット間で変換が行われている点に注意が必要です。これはパフォーマンスに大きな影響を与えます。

1 から 127 までの Unicode 文字は ASCII 文字に対応します(日本語環境では一部例外あり)。

4D v11 SQL Release 2 以降、Unicode モードでアプリケーションが実行されるとき、4D は ICU コレーションアルゴリズムを使用するようになりました。これはソート、クエリ、そしてデータ比較に使用されます。このアルゴリズムは 2 つの文字列を高度な方法で比較します。ユーザー や デベロッパー には透過的です。しかし ICU 特有のルールがあり、4D コマンドの振る舞いが大きく変わることがあります。「無視可能な文字」の取り扱いはその一例です。

```
C_TEXT($textToFind_t;$textToSearch_t)
C_LONGINT($pos_l)
$textToFind_t:=Char(1)
$textToSearch_t:="a"+Char(1)+"b"
```

```
$pos_l:=Position($textToFind_t;$textToSearch_t)
```

Unicode モードで上記のコードを実行すると \$pos_l が受け取る結果は 1 になります。Char(1) は無視される文字として定義されていて、文字列比較の際には考慮されません。Char(1) が無視される結果、\$textToFind_t の検索パターンは空文字となり、1 文字目の右がヒットします。

特定の 4D コマンド (Length、Replace string、Position、Delete string、Substring、Change string) にはオプションの * 引数を指定することで、ICU ルールではなく文字コードベースの検索を強制することができます。

まとめ

- 4D の Unicode モードを理解する際には 2 つの重要なファクターがある。
 - すべてのデータベースを Unicode モードにし、必要な変更を行う。
 - Unicode のサポートとは何を意味するかについて理解すること。
-
- Unicode は比較的複雑なトピックであり、開発に大きく関連。そのためここでは詳細に触れません。この点について書かれたテクニカルノートがあるので、Unicode モードに変更する際開発に与える影響についてはそちらを参照。この文書はすべての 4D デベロッパーが読むべきものであり、Unicode を理解することはデータベースの管理のみならず、テキストに関連するトラブルシュートにも重要。
- Web には UTF-8 を使用。ほとんどの場合データがコンパクトになり、互換性が高まる。UTF-8 は Web のデファクトスタンダードである。
- 4D がアクセスする XLIFF は UTF-8 でなければならない。それ以外のエンコーディングの場合正しく処理されない可能性がある。エディターが他のエンコーディングを使用しないよう確認する。
- キーワード検索を使用する場合は、ICU がどのようにキーワードを作成するか理解し確認する。
 - http://www.unicode.org/unicode/reports/tr29/#Word_Boundaries
 - キーワードの作成は ICU のサポートにより v11.2 より変更されました。また日本語のコレーションは、v11.6 で変更されています。以前のバージョンで作成されたインデックスは作り直す必要があります。

バックアップと復元

ここでは最適なデータベースの整合性を保つための詳細な情報とチップスを紹介します。

バックアップのスケジュール

この節はバックアップスケジューラーの使用方法についてではありません。スケジューラーは直感的に操作できます。そうではなく定期的なバックアップが完了しない場合のトラブルシュートについて詳細な情報を提供します。

定期的なバックアップは 4D の「内部タイマープロセス」と呼ばれるカーネルプロセスで管理されます。内部タイマープロセスは定期的にバックアップ設定をチェックし、バックアップを起動するかどうかを判断します。

プロセス名	セッション	タイプ	Num	状況
DB4D Flush		DB4Dサーバー	0	実行中
DB4D Index builder		DB4Dサーバー	0	実行中
DEMON		ストアドプロシージャー	5	モーダルダイアログを隠す
タスクマネージャー		SQLサーバー	0	実行中
Webサーバ	-	Webサーバー	4	実行中
クライアントマネージャ	-	アプリケーションサーバー	3	フラグ待機中
ユーザインターフェース	-	アプリケーションサーバー	1	イベント待機中
内部タイマープロセス	-	アプリケーションサーバー	2	実行中

内部タイマープロセスには、他のタスクもあります。例えばサーバー管理ウインドウに表示するためのプロセス情報を収集します。このプロセスが何らかの理由でスタックすると、定期的なバックアップは行われなくなります。内部タイマープロセスがスタックしているかどうかを知る簡単な方法は、管理ウインドウが更新されているかどうかを確認することです。

通常、内部タイマープロセスの状態は 4D Server 上で「実行中」、4D クライアント上で「実行中」または「イベント待ち」です。スタックしている場合、ランタイムエクスプローラでプロセスの状態を確認できます。ランタイムエクスプローラは、ユーザーインターフェースプロセスでプロセス情報を収集いるからです。

内部タイマープロセスには定期的なバックアップ以外にも他の多くのタスクがあるので、処理をブロックする理由はバックアップ以外にあるかもしれません。このプロセスが処理するタスクにはほかに処理モニターデータの収集、クライアントサーバー進捗インジケーターの更新、クライアントへのリソースフォルダー変更通知、ストラクチャー更新のチェックなどがあります。これらのタスクのいずれかが予期しないエラーに直面し、内部大麻プロセスがスタックすると、定期的なバックアップを行えなくなります。

内部タイマープロセスが正しく動作しているとすれば、バックアップが実行するタイミングが検知され、「BackupProcess」という名前のプロセスが起動されます。

プロセス名	セッション	タイプ	Num	状況
DB4D Flush		DB4Dサーバー	0	実行中
DB4D Index builder		DB4Dサーバー	0	実行中
DEMON		ストアドプロシージャー	5	モーダルダイアログを隠す
タスクマネージャー		SQLサーバー	0	実行中
BackupProcess	-	アプリケーションサーバー	9	実行中
Webサーバ	-	Webサーバー	4	実行中
クライアントマネージャ	-	アプリケーションサーバー	3	フラグ待機中
ユーザインターフェース	-	アプリケーションサーバー	1	イベント待機中
内部タイマープロセス	-	アプリケーションサーバー	2	実行中

ここで問題がある場合には BackupProcess の状態をモニターすることが重要です。

- ・ プロセスが存在するか？
- ・ 実行されているか？
- ・ 状態は？

内部タイマープロセスと BackupProcess の状態のモニターは PROCESS PROPERTIES コマンドを使用して自動化できます。例えば定期的なバックアップの問題が疑われる場合、ストアドプロシージャでこれらのプロセスの状態を監視するとよいでしょう。

バックアップ開始直前には On Backup Startup、終了直後には On Backup Shutdown データベースメソッドが呼び出されます。これらのメソッド呼び出しをログに取ることで、バックアップ処理の途中で問題が発生しているのか、バックアップ処理の外なのかを知ることができます。これはどこでバックアップ処理に失敗しているのかを知るために重要な情報です。

On Backup Shutdown データベースメソッドは\$1 にステータスコードを受け取ります。このステータスコードをログに書き出しが、何らかの理由で特定のバックアップしいっぱいが発生しているのか知るために重要です。さらにバックアップ設定に基づき、4D はバックアップが失敗したときにバックアップを再試行します。これらの情報をすべて収集することが重要です。

データベース管理者にバックアップの完了や、\$1 のエラーコードを通知するメールを送信する機能を On Backup Shutdown に追加することは良いことです。

復元

データベースをバックアップする際、4D はファイル名に番号を追加します。ログファイルが有効になっていれば、ログファイルのバックアップも作成します。ログファイルのバックアップの番号は常にデータベースバックアップ番号マイナス 1 です。この番号の相違により、復元の際にどのファイルを使用すべきか迷うかもしれません。

これに対応するため、復元を行う場合、以下の 2 つの方法を使用できます。

- 番号を無視し、単純なルールに従う
- 設計を理解する（この場合でも単純なルールを適用できます）

単純な方法で復元を行う

バックアップファイルの番号付けルールを理解していないなくても、データベース復元の際には 1 つの単純な従うべきルールがあります。

常に番号が一致するファイルを選択する

例えば以下のようなファイルが存在する場合を例に挙げて説明します。

Name	Date modified	Type	Size
Logs	7/20/2011 1:21 PM	File folder	
Preferences	7/20/2011 1:27 PM	File folder	
Resources	7/20/2011 1:21 PM	File folder	
Rollback.4DB	7/21/2011 11:55 AM	4D Structure File	385 KB
Rollback.4DD	7/21/2011 11:55 AM	4D Data File	193 KB
Rollback.4DIndy	7/21/2011 11:55 AM	4D Structure Index...	193 KB
Rollback.journal	7/21/2011 11:55 AM	4D Journal File	1 KB
Rollback.Match	7/21/2011 10:24 AM	4D Structure info	1 KB
Rollback[0000].4BL	7/20/2011 1:27 PM	4D Backup Journal...	2 KB
Rollback[0001].4BK	7/20/2011 1:27 PM	4D Backup file	164 KB
Rollback[0001].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0002].4BK	7/20/2011 1:27 PM	4D Backup file	164 KB
Rollback[0002].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0003].4BK	7/20/2011 1:27 PM	4D Backup file	165 KB
Rollback[0003].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0004].4BK	7/20/2011 1:27 PM	4D Backup file	165 KB
Rollback[0004].4BL	7/20/2011 1:28 PM	4D Backup Journal...	1 KB
Rollback[0005].4BK	7/20/2011 1:28 PM	4D Backup file	165 KB

ここには 5 つのバックアップファイルが存在します。バックアップ#3 を復元したい場合、次のファイルを指定します。

Rollback[0003].4BK
Rollback[0003].4BL

ファイルが一致するので、自動で同じフォルダーにファイルが復元され、データベースが利用可能になります。

ルールは単純であり、データベース管理のトレーニングに有効です。

バックアップファイル番号付けのルールを理解する

先に示した通り、4D のバックアップはカレントデータベースバックアップ番号より小さい番号のログファイルバックアップを作成します。これは意図されたものです。バックアップされるデータベースにはカレントログファイルに書かれた情報がすでに含まれています。このバックアップが復元されるときにはカレントログファイルを統合する必要はありません。

復元には 2 つのシナリオがあります：

最新のバックアップから復元を行う場合。データベース起動時に、壊れたデータベースのカレントログファイルを選択します（ログファイルが壊れていないという前提で）。カレントログファイルを選択できない場合、新しいログファイルを作成する以外選択はありません。

最新でないバックアップから復元を行う場合。先に示した単純なルールに従います。同じ番号のログバックアップファイルを選択します。統合すべきそれ以降のログファイルがある場合、番号順にログバックアップファイルを選択し、最後にカレントログファイルを選択します。

次の例は、少し複雑です。

Name	Date modified	Type	Size
Logs	7/20/2011 1:21 PM	File folder	
Preferences	7/20/2011 1:27 PM	File folder	
Resources	7/20/2011 1:21 PM	File folder	
Rollback.4DB	7/21/2011 11:55 AM	4D Structure File	385 KB
Rollback.4DD	7/21/2011 11:55 AM	4D Data File	193 KB
Rollback.4DIndy	7/21/2011 11:55 AM	4D Structure Index...	193 KB
Rollback.journal	7/21/2011 11:55 AM	4D Journal File	1 KB
Rollback.Match	7/21/2011 10:24 AM	4D Structure info	1 KB
Rollback[0000].4BL	7/20/2011 1:27 PM	4D Backup Journal...	2 KB
Rollback[0001].4BK	7/20/2011 1:27 PM	4D Backup file	164 KB
Rollback[0001].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0002].4BK	7/20/2011 1:27 PM	4D Backup file	164 KB
Rollback[0002].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0003].4BK	7/20/2011 1:27 PM	4D Backup file	165 KB
Rollback[0003].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0004].4BK	7/20/2011 1:27 PM	4D Backup file	165 KB
Rollback[0004].4BL	7/20/2011 1:28 PM	4D Backup Journal...	1 KB
Rollback[0005].4BK	7/20/2011 1:28 PM	4D Backup file	165 KB

このようなファイル構成のときに、データベースが壊れて復元しなければならなくなつたときのことを考えましょう。このときバックアップ#1から復元を行うとします。それには、次の作業を行います。

- 4Dを起動します。
- データベースを開きます。これはMSCの復元インターフェースにアクセスするために必要です。新しい空のデータベースを作成することができます。
- MSCを開きます。
- 復元タブを選択します。
- 選択ボタンをクリックし"Rollback[0001].4BK"を選択します。
- 「復元後にひとつ以上のログファイルを統合」をチェックします。
- 復元ボタンをクリックします。
 - 4Dは"Rollback[0001]"フォルダーにRollback[0001].4BKを復元します。
- 4Dはキャンセルがクリックされるまで統合するログファイルを選択するためのダイアログを表示します。
 - Rollback[0001].4BLを統合します。4Dは"Rollback[0001].integrate"フォルダーにログファイルを配置します。
 - Rollback[0002].4BLを統合します。4Dは"Rollback[0002].integrate"フォルダーにログファイルを配置します。
 - Rollback[0003].4BLを統合します。4Dは"Rollback[0003].integrate"フォルダーにログファイルを配置します。
 - Rollback[0004].4BLを統合します。4Dは"Rollback[0004].integrate"フォルダーにログファイルを配置します。
- Rollback[0001]フォルダー内のデータベースを今まで使用していたデータベースフォルダーに移動またはコピーし、壊れたデータベースに入れ替えます。
- 復元したデータベースを4Dで開きます。
- 指示されたら、カレントログファイル"Rollback.journal"を選択します。4Dは残りの変更を統合し、データベースを利用可能にします。

4Dの復元機能の利点で見逃されるものに、復元後一致するファイルが自動で同じフォルダーに配置されるというものがあります。1回の復元のみが必要な場合、より単純な方法を使用できます。例えば先の例で#3のバックアップのみを復元したい場合には次のように作業を行います。

- 4D を起動します。
- ファイルメニューから復元...を選択します。
- "Restore[0003].4BK"を探し、復元します。
- "Restore[0003].4BL"にも同じステップを繰り返します。
- デフォルトでデータベースとログファイルは同じフォルダーに展開されます。この時点ではデータベースを使い始めるすることができます。
- データベースが 4D で起動されると、ログファイル 0003 に記録された変更が統合されます。結果データベース 0003 はデータベース 0004 と同じになります。

復元されたバックアップが利用可能かどうかは何がバックアップされているかによります。例えばアプリケーションがプラグインを使用している場合で、これらのプラグインがバックアップに含まれない場合、復元後にインストールする必要があるかもしれません。復元したファイルを今まで使用していたフォルダーにコピーすることもできます。復元処理時のミスに備えるため、ファイルの入れ替えを行う前に今まで使用していたデータベースファイルのコピーを取ることを忘れないでください。

ログ管理

4D ログファイルの管理機能は(読み込みのみデータベースを除き)データベースの整合性を保守するための重要な部分です。ログファイル機能の原則はとても単純です。

4D はランタイム処理の高速化のためにメモリキャッシュを使用し、データベースのデータにアクセスする際のディスクへのアクセスを減らしています。データベースオブジェクトの変更(レコードの CRUD、インデックスの更新、ストラクチャーの変更)が発生すると、変更はディスクではなくメモリに保持されます。もちろんこれらの変更を永続化するため、定期的にディスクにフラッシュされます。クラッシュなどのためデータベースがクラッシュし、キャッシュ中に永続化されていないデータが残っていた場合、その変更は失われます。

注

4D v11 よりログファイルの拡張子は .4DL から .journal に変更されました。

ログファイルはキャッシュと完全に異なります。データに関する処理はディスク上のログファイルに即座に書き込まれます。ログファイルは他のデータベースファイルと異なる方法で管理され、この即座のディスクアクセスは効率化されています。結果キャッシュ中のデータがすべてフラッシュされていないデータベースがクラッシュしても、次回データベースを起動する際に失われた変更をログファイルから統合することができます。

さらにログファイルはバックアップから復元されたデータベース以降の変更をすべて統合する機能も提供します。先の節の例題の通り、復元したデータベースに順番にログファイルを統合して、最新の状態に戻すことができます。

最後にこのパワフルな機能を使用すれば 4D の論理ミラーを可能にできます。

ログファイルを使用した復元

バックアップ #1 空の復元を行う先の例題で 4D ログファイルの重要性は説明しました。他にもログファイルに関するパワフルな能力があります。

アクティビティ解析

MSC にはアクティビティ解析と呼ばれるページがあり、ユーザーはこのページを使用して選択したログファイルに含まれる処理を確認することができます。デフォルトでカレントログファイルが表示されていますが、他のファイルを選択することもできます。この UI を使用してユーザーはログファイルに含まれるメタデータを見るることができます。

アクティビティ分析

以下のリストは、最後のバックアップ以降ログファイルに記録された操作を表示します。

操作#	アクション	テーブル	レコード/	プロセス	サイズ	日付	時間	ユーザ
17871	コンテキス…			18		2011-11-21	13:36:04	kebukawa
17872	シーケンス番号	Mail_box		18		2011-11-21	13:36:04	kebukawa
17873	追加	Mail_box	2594	18	408	2011-11-21	13:36:04	kebukawa
17874	シーケンス番号	Mail_box		18		2011-11-21	13:36:04	kebukawa
17875	追加	Mail_box	2595	18	434	2011-11-21	13:36:04	kebukawa
17876	シーケンス番号	Mail_box		18		2011-11-21	13:36:04	kebukawa
17877	追加	Mail_box	2596	18	238	2011-11-21	13:36:04	kebukawa
17878	シーケンス番号	Mail_box		18		2011-11-21	13:36:04	kebukawa
17879	追加	Mail_box	2597	18	406	2011-11-21	13:36:04	kebukawa
17880	シーケンス番号	Mail_box		18		2011-11-21	13:36:04	kebukawa
17881	追加	Mail_box	2598	18	434	2011-11-21	13:36:04	kebukawa
17882	シーケンス番号	Mail_box		18		2011-11-21	13:36:04	kebukawa
17883	追加	Mail_box	2599	18	362	2011-11-21	13:36:04	kebukawa
17884	シーケンス番号	Mail_box		18		2011-11-21	13:36:04	kebukawa
17885	追加	Mail_box	2600	18	450	2011-11-21	13:36:04	kebukawa
17886	シーケンス番号	Mail_box		18		2011-11-21	13:36:04	kebukawa
17887	追加	Mail_box	2601	18	258	2011-11-21	13:36:04	kebukawa
17888	シーケンス番号	Mail_box		18		2011-11-21	13:36:04	kebukawa
17889	追加	Mail_box	2602	18	376	2011-11-21	13:36:04	kebukawa
17890	シーケンス番号	Mail_box		18		2011-11-21	13:36:04	kebukawa

特定のフィールドを表示するには、カラムヘッダを右クリックします。

解析 選択... 書き出し...

一覧に以下のフィールドを追加したり取り除いたりすることができます。

追加
削除

▶

- アクション
- テーブル
- ユーザ
- 日付
- 時間
- プロセス
- レコード/BLOB
- サイズ
- 値
- 操作#

何より重要なのは、この UI の内容をテキストファイルに書き出して、他のツール等で解析できることでしょう。この点についてはプレクラスの第三部で触れます。

ロールバック

ログファイルに関連するもう一つのパワフルな機能はデータベースに対して行われた変更をロールバックできることです。

ユーザーあるいは自動化処理により、データベースに誤った更新が行われたとします。管理者はロールバック機能を使用してログファイル中の最後の有効な処理を選択し、それ以降の無効な処理をなかつたことにできます。これは MSC のロールバックページで行うことができます。

この図の例では、Mail_box にレコード 8058 の追加処理を含めて、その後の処理をすべて無効にします。

ロールバック

以下のリストは、最後のバックアップ以降ログファイルに記録された操作を表示します。

操作#	アクション	テーブル	レコード/	プロセス	サイズ	日付	時間	ユーザ
30529	シーケンス番号	Mail_box		30		2012-01-06	10:24:38	
30530	追加	Mail_box	8055	30	260	2012-01-06	10:24:38	
30531	シーケンス番号	Mail_box		30		2012-01-06	10:24:38	
30532	追加	Mail_box	8056	30	258	2012-01-06	10:24:38	
30533	シーケンス番号	Mail_box		30		2012-01-06	10:24:38	
30534	追加	Mail_box	8057	30	282	2012-01-06	10:24:38	
30535	シーケンス番号	Mail_box		30		2012-01-06	10:24:38	
30536	追加	Mail_box	8058	30	460	2012-01-06	10:24:38	
30537	シーケンス番号	Mail_box		30		2012-01-06	10:24:38	
30538	追加	Mail_box	8059	30	272	2012-01-06	10:24:38	
30539	シーケンス番号	Mail_box		30		2012-01-06	10:24:38	
30540	追加	Mail_box	8060	30	282	2012-01-06	10:24:38	
30541	シーケンス番号	Mail_box		30		2012-01-06	10:24:38	
30542	追加	Mail_box	8061	30	406	2012-01-06	10:24:38	
30543	シーケンス番号	Mail_box		30		2012-01-06	10:24:38	
30544	追加	Mail_box	8062	30	464	2012-01-06	10:24:38	

特定のフィールドを表示するには、カラムヘッダを右クリックします。

上のリストから特定のログアクションを選択して、ロールバックボタンをクリックすると、4Dは現在のデータファイルを閉じ、最新のデータベースのバックアップから復元を行い、選択されたものを含めログに記録された操作を統合します。

ロールバック操作を取り消すことはできません。しかし現在のデータファイルはリネームされてディスク上に保存されます。

カレントログファイル

ロールバック

ロールバックは実際に変更をロールバックするわけではない点に留意してください。4D は代わりに最新のバックアップから復元を行い、選択されたロールバック位置までログファイルに書かれた処理の統合を行います。

重要

ロールバックを行うためには最新のバックアップが必要です。このためログファイルを有効にする場合はバックアップも有効にしなければなりません。

最新ではない過去のバックアップにまでロールバックを行うのは、この処理を複雑にします。この点について説明します。

バックアップまでロールバックする

ロールバック機能を動作させるため、最新のバックアップが MSC から選択できなければなりません。最新のバックアップの選択メカニズムはバックアッププロジェクトファイル (Preferences フォルダー内の Backup.xml) をチェックすることに依存します。この機構はカレントログファイル以前の部分までロールバックを行いたい場合に問題となります。

無効な変更が行われ、1回以上のバックアップが行われるまでその事に気がつかなかったとしましょう。ここからロールバックを行うために、最新のバックアップではないバックアップを復元する必要があります。動作中のデータベースでこれを行うことは不可能です。なぜならバックアッププロジェクトファイルは最新のバックアップファイルを記録しているからです。この場合変更が行われた時点のデータベースの状態を再現する必要があります。今回の例題で Roolback.4dbase の現状が以下であるとき：

注

この例題ではリスクを最小化するために異なるフォルダーで処理を実行します。このためバックアッププロジェクトファイルが最新のバックアップを指すようパスを編集しなければなりません。

このデータベースには 5 つのバックアップ ([0001]から[0005]) があります。データベースは以下のパスにあります。

バックアップ 0004 に問題があることが判明します。これを修正するには以下の通りに行います。

重要な注意

バックアップ設定には *Backup.xml* (実際には *Preferences* フォルダー全体) が含まれています。これにより復元やロールバックが容易になっています。

- 処理を行うための新しいフォルダーを作成します (C:\Share\4D\Projects\Misc\Test)。
- バックアップ 0004 とログファイル 0004 をこのフォルダーにコピーします。
 - バックアップ 0004 を C:\Share\4D\Projects\Misc\Test\Rollback[0004] に復元。
 - ログファイル 0004 を C:\Share\4D\Projects\Misc\Test\Rollback[0004] に復元。
- この時点でバージョン 0004 のデータベースが配置されました。このデータベースにはロールバックしたいポイントが含まれています。しかしバックアップ 0004 に対応する最新のバックアップファイルが依然として必要です。
 - バックアップファイル 0003 を C:\Share\4D\Projects\Misc\Test\Rollback[0004]\Rollback[0003].4BK にコピーします。
- これで 0004 に対応する最新のバックアップファイルが配置されましたが、まだ十分ではありません。4D はどこからこのファイルを探せばよいか知る必要があります。Preferences フォルダーはバックアップ内に含まれるため、正しい *Backup.xml* はすでに復元されています。しかし今はバックアップ 0004 を動作時とは異なるフォルダーに復元したため、バックアップ 0003 へのパスを更新する必要があります。
 - C:\Share\4D\Projects\Misc\Test\Rollback[0004]\Preferences\Backup にある復元した *Backup.xml* を開き、C:\Share\4D\Projects\Misc\Rollback.4dbase を C:\Share\4D\Projects\Misc\Test\Rollback[0004] で置き換えます。
- 4D で C:\Share\4D\Projects\Misc\Test\Rollback[0004]\Rollback.4DB を開きます。
- MSC のロールバックページを開きます。
- ロールバックしたい行を選択します。

結果ロールバックボタンがクリック可能になり、ロールバックを実行することができます。

まとめ

ロールバック機能を動作させるため、*Backup.xml* 内にはその前のバックアップファイルのパスが正しく指定され、そのファイルを 4D が探せなければなりません。最新のバックアップよりも前のバックアップを使用してロールバックを行うためには *backup.xml* を編集して 4D が最新のバックアップを探せるようにします。

チップ

データベースバックアップに *backup.xml* を含めることにより、*backup.xml* 内の他の値 (最新のバックアップ番号等) も復元対象のデータベースに対し正しいことを意味し、結果編集する必要がありません。この例題で復元の必要なのはパスを編集することだけです。

ロールバックを行うために XML ファイルを新規作成することも可能ですが、しかしこれを行うとより処理の複雑さが増します。バックアップ設定を使用して *backup.xml* をバックアップするほうが簡単です。

もちろん動作中のデータベースの場所にファイルを復元することも可能です。この場合 backup.xml の編集は必要ありません。しかしこの場合何かが発生した場合のリスクがあります。もしもの時のためにコピーを作成しましょう。

LogTools

4D のログファイルは実際ある種のデータベースです。例えばレコードが更新されると、レコード全体のコピーがログファイルに格納されます。データベースの破壊が激しい場合、ログのデータを展開して新しいデータベースに読み込むこともできます。

Business Brothers (<http://bbsp.com/>) から、これを行うための高く推奨されるサードパーティ製ツール、LogTools がリリースされています。LogTools については第3部で紹介します。

ミラー

4D v11 SQL では 2 つの新しいコマンドを使用して論理ミラーを構築できます。

New log file: このコマンドを使用してカレントログファイルを閉じ、新しいログファイルを作成できます。

INTEGRATE LOG FILE: このコマンドを使用してログファイルを統合します。

メカニズムは極めて単純です。運用中のデータベースで New log file コマンドを使用し、新しいログファイルを作成します。閉じたログファイルをミラーデータベースに送信します。そしてミラーデータベース側でそれを統合します。これらのログファイルをセグメントあるいはログセグメントと呼ぶことにしましょう。この方法でミラーデータベースは運用中のデータベースと同等となり、データの完全なコピーが作成されます。

ログセグメントの管理は開発者が行うことができるので、論理ミラーの実装はインフラに合わせて作成することができます。この機能について不明な点は 4D のドキュメントを参照してください。

ここでの目的はミラーについて説明することではなく、なぜミラーがあらゆるデータベースにとってベストプラクティスとみなされるのか、またこれを利用する利点は何かを示すことにあります。

注: 4D のミラーは制限のない冗長性を提供します。つまり複数のミラーを実装することが可能です。追加のコピーを保守するためにはすべてのミラーにログセグメントを送信するだけです。

データベースの生のバックアップを保守することに加え、ミラーには他にも利点があります。

ホットスワップ

4D の論理ミラーの明らかな利点はアプリケーションをホットスワップできることにあります。主たるサーバーがダウンしたときにはミラーサーバーをその場所に配置するか、通信をミラーに転送することができます。これにより問題が解決されるまでダウンタイムを最小化できます。

オフラインメインテナンス

ミラーリングによりデータベースの運用と管理タスクを分離することが可能になります。

- バックアップ
 - 論理ミラーを使用している場合、実際のところ運用中のデータベースをバックアップすることはできません（これを行うと論理ミラーが行えなくなります）。代わりにミラーをバックアップします。これによりバックアップ中の運用データベースの無応答を取り除くことができます。4D のバックアップ中もデータの読み取りやトランザクションベースのデータアクセスを引き続き行うことはできますが、運用サーバーからバックアップ処理を取り除くことでパフォーマンスをかなり向上させることができます（特に巨大なデータベースの場合、バックアップにも時間を要します）。

- ミラーのバックアップ中はログセグメントの統合が行われない点に留意してください。バックアップが終了するまでは完全には最新ではなく、バックアップ終了後に残りのログを統合する必要があります。
 - ログファイルの統合はデータベースの使用よりも速いので、一般的にミラーが運用データベースよりも遅れを取ることを考慮する必要はありません。しかしロードが極端であるデータベースではこの点を考慮する必要があるかもしれません。
- 検証/復元
 - 4D バックアップと同様、MSC タスクをミラー上で行うことが可能ですが。例えばデータベース実行中は復元処理を行えません。復元を運用データベースで行うとその間データベースをオフラインにする必要があります。
 - (いくつかある) 検証処理は運用データベースで行うことができますが、パフォーマンスに影響します。
 - これらの処理をミラーに移行することで、運用サーバーに与えるパフォーマンスの影響を取り除くことができます。
 - データベースは論理的に同じものなので、運用サーバーの問題を見逃すリスクを大きく削減できます。言い換えればミラーで発見された問題は、おそらく運用中のサーバーにも存在します。さらにメインテナンスタスクに運用サーバーが影響されることはないので、より頻繁にメインテナンスを行うことができます。

オフライン圧縮

注

これは技術的にはオフラインメインテナンスの一部ですが、説明を加えるために別の節として独立させました。

データベースの圧縮は時間を要します。データベースが大きくなり、またフラグメントが大きければ、圧縮にかかる時間は長くなります。

オフライン圧縮は、運用データベースをそのまま実行し、データベースのコピーを圧縮する処理です。メインテナンスが終了したら、ダウンタイムは圧縮したデータファイルを運用マシンにコピーし、論理ミラープロシージャを実行する間だけです。

例えば 2 つのシステム Production と Mirror があるとき

- Mirror データベースの停止
- Mirror データベースの圧縮
- Mirror データベースを起動し圧縮中に、Production のすべてのログセグメントを統合
- Mirror がすべてのログを統合したら、Production を停止
- Production のカレントログファイルを Mirror に統合します。この時点で圧縮された Mirror データベースが完全に最新になります。
- Mirror データベースを Production にコピーし、論理ミラーの設定を行います。

この方法で Production システムはファイルのコピーとミラーの設定のときだけダウンします。

この方法が常に最適ではないということに留意してください。論理ミラーの設定時にはバックアップが起動されます。バックアップ処理が圧縮処理よりも時間がかかる場合、Production システムがバックアップのためにダウンする時間が長くなるため、上記の方法をとる利点はなくなります。このときは Production 上で圧縮を行うべきです。

ベストプラクティス

- On Backup Startup と on Backup Shutdown データベースメソッドを実装しましょう。各メソッドの各呼び出しをログに取ることで、バックアップのどの時点で問題が発生したのかを追跡する助けになります。
- データベースとログファイルのバックアップを復元する際、バックアップの番号が一致しているかを確認します。
- 4D のログ機能を使用しましょう。この機能には単純なクラッシュからの復元以外にも多くの利点があります。
- MSC を使用してログファイルのメタデータを書き出し、外部ツールで解析できます。
- 復元後のインデックス再構築を避けるためにはインデックスファイルをバックアップに含めます。
- backup.xml をバックアップ対象に含めましょう。それによりロールバック機能が使いやすくなります。
- バックアップは異なるドライブに取ることが推奨されますが、外部ドライブは避けましょう。バックアップファイルを例えばネットワーク上のボリュームに移す必要がある場合、OS の機能を使用してください。4D から外部ストレージにアクセスしないようにしましょう。例えば On Backup Shutdown でスクリプトを起動することができます。
- 4D のバックアップ機能を使用しましょう。4D をバックアップするための最適なツールです。ログ管理機能を使用することもできるようになります。
 - Retrospect を 4D に対して使用するとデータが壊れるケースが報告されています。
 - 4D をバックアップするために Time Machine を使用することはできません。開かれたファイルはバックアップできません。
- その逆に、4D は 4D データベースのバックアップのためだけに使用します。汎用的なファイルバックアップシステムとしては使用しないでください。例えば 4D のバックアップは何百ものファイルのバックアップには適していません。

メインテナンス

この章では 4D データベースのメインテナンスに関するベストプラクティスを論じます。特に以下の節では 4D Server 管理ウィンドウや MSC について説明します。

標準モードとメインテナンスマード

MSC には標準とメインテナンスマードがあることを理解することが重要です。

標準モード

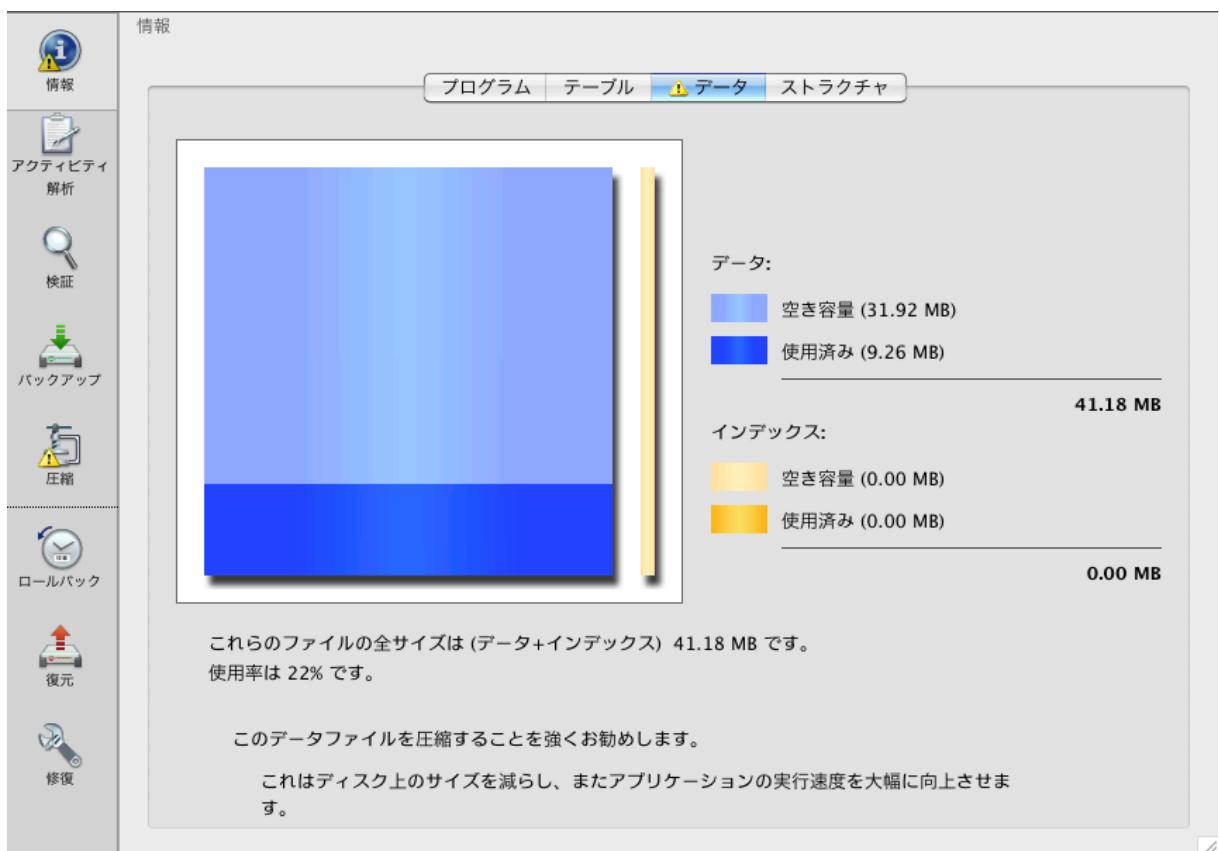
4D がデータベースを開いている際、MSC は標準モードで実行されます。この場合データベースは実行中です。MSC はデータベース中のレコード数や、データおよびストラクチャーファイルの利用スペースなどの統計情報にアクセスすることができます。

注

MSC におけるアクセス権についての詳細は後で説明します。

重要な注意

データベースのサイズによっては、情報ページを表示するときに遅延が生じるかもしれません。



遅延が生じるのは、MSC がデータファイルとストラクチャーファイルを解析し、このページで表示すべき情報を収集しなければならないからです。表示されたメッセージに対する対応を行う前にファイルの解析を待ってください。例えば上の図に表示されている「データファイルを圧縮することをお勧めします」のメッセージは正しいものですが、もしこのページを表示したときに、イメージが表示されておらず同時に使用済みの容量が 0GB となっているような時は、未だ情報収集中ですからメッセージは意味なく表示されているだけです。そのようなときには、データファイルが完全に解析されるまで待つ必要があります。解析が終了すると UI が更新されます。

解析が終了する前に他のページ（例えば圧縮ページ）を選択すると、MSC アイコンはファイルが解析されたときに更新されます。上の図では、圧縮ボタンに注意アイコンが付加され圧縮が必要であることを表しています。解析が終了して、本当に圧縮が必要なときには、上の図のようにボタンの表示も変わるはずです。

解析に時間が掛かるようなときには、別のページをチェックしながら、ファイルの解析を待ってください。

標準モードで行われるタスク（特に検証とバックアップ）はリアルタイムに行うことができます。つまりユーザーがログインしている間も行うことが可能です。いくつかの制約があります。

タスクの実行中新しい接続を行うことはできません。

既存の接続は読み込みのみ行うことができます。書き込みが試行されると、メインテナンスが完了するまでそのプロセスは一時停止されます。

メインテナスマード

メインテナスマードでは 4D はデータベースを開いていません。Startup コードは実行されず、データベースキャッシュは作成されません。クライアントからの接続もできません。このモードではメインテナスマスクのみ行うことができます。例えば情報ページにデータやストラクチャーファイルに関する情報は表示されません。

アクセス権

特定の MSC 機能は、使用される 4D アプリケーションのタイプ、MSC モード、そしてカレントユーザーのアクセス権によっては使用することができません。ガイドラインは以下の通りです：

アプリケーションストラクチャーに関連する機能（検証、復元、圧縮）は 4D ローカルモードおよび 4D Server からのみアクセスできます。4D リモートモードや 4D Desktop では対応するページが隠されます。

データやストラクチャーファイルの内容に関する情報は、4D がデータベースを開いている場合のみ利用できます。これは先に説明したとおり、4D が標準モードで MSC を実行している場合です。

データ圧縮、ロールバック、復元、修復は 4D が開いていないデータファイルに対してのみ行うことができます。これは先に説明したとおり 4D がメインテナンスモードで MSC を実行している場合です。これらのタスクのいずれかがリクエストされると、MSC はデータベースを閉じメインテナンスモードで聞く旨の警告を表示します。

パスワードが有効化されている場合、データ圧縮、ロールバック、復元、修復は Administrator および Designer のみ行うことができます。

検証

データベースの検証はストラクチャーと、4D データベースファイルの内容のチェックを実行します。検証はあらゆる 4D データベースにとって重要なタスクであり、できる限り早く問題を検知するために定期的に行われるべきものです。

データベースの検証には複数のオプションがあり、またそれにアクセスする方法も複数あります。この節では各オプションについて説明します。主に開発タスクであるストラクチャーの検証についてはここでは触れません。

データの検証は 3 つのうちいずれかの方法でアクセスできます。

- MSC の検証ページ

The screenshot shows the 'Validation' section of the 4D Server Management window. On the left, a sidebar lists icons for 'Information', 'Activity Analysis', 'Validation' (which is selected and highlighted with a red border), 'Backup', 'Compression', 'Rollback', 'Restore', and 'Repair'. The main area has a title 'Validation' and a sub-section 'Select an action from the following list:'. It contains three validation options, each with an icon and a status message:

- Validate records and indexes**: Shows a magnifying glass icon. Status: 'All records (51/955) and indexes (1) are validated.' Question: 'The database file is not validated.'
- Validate records only**: Shows a magnifying glass over a document icon. Status: 'Only records (51/955) are validated.' Question: 'Records are not validated.'
- Validate indexes only**: Shows a magnifying glass over a book icon. Status: 'Only index of the database file (1) is validated.' Question: 'Indexes are not validated.'

Below these is another section with a gear icon: **Validate application**. Status: 'All modules, forms, and all design objects are validated.' Question: 'The application is not validated.'

A 'Tables list' button is at the bottom.

- 4D Server 管理ウインドウのメインテナンスページ



- VERIFY CURRENT DATA FILE コマンド

検証には 4 つの異なる種類があります。

- レコードの検証
- インデックスの検証
- レコードとインデックスの検証
- すべてを検証

これらに対応する VERIFY DATA FILE コマンドや VERIFY CURRENT DATA FILE コマンドの引数があります。いくつかは UI から使用できるオプションに対応します。以下は GUI タスクとランゲージの対応です:

タスク	4D ランゲージ
管理的な検証	VERIFY CURRENT DATA FILE(Verify Records+Verify Indexes;0;"")
MSC のすべてを検証	VERIFY CURRENT DATA FILE(Verify All;0;"")
MSC のレコードを検証	VERIFY CURRENT DATA FILE(Verify Records;0;"")
MSC のインデックスを検証	VERIFY CURRENT DATA FILE(Verify Indexes;0;"")

検証処理に要する時間を最短なものにするために、いくつかの検証オプションから選択ができるようになっています。先に説明したとおり検証中は新規接続が行えず、既存の接続は読み込みのみになります。検証されるデータベースパーティと、データベースの可用範囲がトレードオフになります。一般的に完全な検証よりレコードとインデックスの検証のほうが速いです。

しかし MSC のレコードとインデックスを検証と、管理ウィンドウのレコードとインデックスを検証は同じでないことに留意してください。MSC のオプションでは Verify All を行います。この操作は完全な検証を処理するデータエンジン機能を呼び出します。Verify Records と Verify Indexes オプションは C++コードを使用して高レベルで管理され、各検証処理の特定の部品を処理するデータベースエンジンのみが呼び出されます。2 つのポイントがあります:

管理的な検証はデフォルトでクライアントから実行可能な検証のみを行います。

MSC による検証は一般的により速いです。なぜならデータエンジン内で実行され、高度に最適化されているからです。

結果データベースの検証で推奨されるベストプラクティスは、可能な限り MSC を使用することです。または次節で説明する通りランゲージを使用します。

4D ランゲージサポート

以下のコマンドは多くの MSC タスクをサポートしています。

- VERIFY DATA FILE
- VERIFY CURRENT DATA FILE
- Compact data file

実際先の節で示した通り、MSC はこれらの 4D コマンドと同じエントリポイントを呼び出します。これが意味することは 2 つです:

- 自動化
- 細かな制御

注

カレントデータベースに対するメインテナنسコマンドはクライアント上でなくサーバー上で実行する必要がります。このために EXECUTE ON SERVER や「サーバー上で実行」メソッド属性を使用できます。

自動化

MSC のメインテナンスタスクに 4D コマンドからアクセスできることで、データベースメインテナансを完全自動化できるようになっています。特に検証タスクで有効です (MSC をメインテナансモードで起動する必要がありません)。

これらのコマンドが提供する機能のうちもっとも重要なもののひとつはコールバックシステムであり、これによりメインテナンスタスクの状態を開発者が検知でき、アクションをとることができます。

VERIFY CURRENT DATA FILE(Verify All;0;"callback_Verify")

このコードを使用すると検証処理中定期的に "callback_Verify" メソッドが呼び出されます。コールバックメソッドは 4 つの引数を受け取ります。

イベント	\$1	\$2	\$3	\$4
検証中	1	0	検証中メッセージ	経過量(%)
オブジェクト検証終了	2	Object type	OK メッセージ	テーブル/インデックス数
エラー発生時	3	Object type	エラーメッセージ	テーブル/インデックス数
検証処理終了時	4	0	終了メッセージ	0
警告	5	Object type	警告メッセージ	テーブル/インデックス数

注:\$5 引数は予約されていますが使用されていません。

このシステムによりカレントデータファイルに対する検証タスクを開発者が完全に自動化することができ、さらに発生した問題に応じてカスタマイズされたアクション (データベース管理者にメールを送るなど) を取ることができます。

細かな制御

MSC で利用可能な操作に対抗して、4D ランゲージメインテナансコマンドではタスクの対象とするデータベースオブジェクトをフィルターすることができます。以下はデータベースの 3 つのインデックスのみを検証するコード例です。

```

ARRAY LONGINT($tableNums_al;0)
ARRAY LONGINT($indexNums_al;2;0)

$indexNums_al{1}{0}:=4 // テーブル番号
APPEND TO ARRAY($indexNums_al{1};1) // フィールド番号
$indexNums_al{2}{0}:=5 // テーブル番号
APPEND TO ARRAY($indexNums_al{2};2) // フィールド番号

```

```
APPEND TO ARRAY($indexNums_a1{2};3) // フィールド番号  
VERIFY CURRENT DATA FILE(Verify Indexes;0;"cb_Verify";$tableNums_a1;$indexNums_a1)
```

この例では table 4 の field 1、table 5 の field 2、table 5 の field 3 のインデックスのみがチェックされます。同様に \$tableNums_a1 配列を変更することで特定のテーブルのレコードのみを検証させることができます。これにより開発者はデータベースのメインテナンスを細かく制御できます。

先に示した通り、標準モードのメインテナンス操作は、データベースを読み込みのみモードにするため、可能な限り速く行われるべきです。メインテナントタスクの処理対象をフィルターすることでメインテナントタスクを分割(30 分ごとに 1 テーブルを検証等)したり、必要のない処理を行わない(データが変更されていないテーブルを検証対象から外すなど)などが可能です。これによりメインテナンスをユーザに対して目立たなくするすることができます。

ベストプラクティス

- 標準モードとメインテナンスマードの違いを理解する
 - 標準モードタスクはデータベース利用中も実行できる
 - メインテナンスマードタスクはデータベースが利用されていないときのみ実行できる
- MSC の制約を理解する
 - クライアント上では管理ウィンドウの検証のみを実行できる
 - パスワードアクセスシステムが有効な場合、Administrator と Designer だけが MSC タスクを行える
- 素早く完全な検証を行うには、“Verify All”(MSC のレコードとインデックスを検証)を使用する
- MSC ではなく 4D コマンドを使用すれば自動化とより細かな制御が可能

オペレーティングシステム

OS に関連して現在 4D にとって最も重要なことは、Mac OS X と Windows とも 64-bit OS を使用することです。これにより 4D が使用することのできるメモリの量が(32-bit 4D を使用している場合でも)最大化されるためとても重要です。アクセス可能な総メモリ量に加え、64-bit OS のメモリのパーティション方法と利用方法はより効率的です。メモリについてはハードウェアの項で説明します。

もちろん 64-bit の 4D Server を実行するためには、Windows 64-bit OS が必要になります。

32-bit から 64-bit OS への移行はソフトウェアのアップグレードにすぎないことに留意してください。64-bit をサポートしないモダンな CPU を探すことはとても難しいです。実際 Mac OS X は数年にわたり 64-bit カーネルを使用しています。64-bit の Windows OS へのアップグレードは新しいソフトウェアのインストールと同じです。

注

初期の Intel ベース Mac (mini, MacBook, iMac) は 32-bit Intel コア (Core Solo, Core Duo) を使用しています。これらのマシンでは 64-bit OS がサポートされません。次に使用された Core2 (Core 2 Solo, Core 2 Duo) や i5/i7 CPU は 64-bit プロセッサーであり、64-bit OS をサポートします。

Windows

4D Server を実行する Windows マシンのベストプラクティスは以下の通りです。

Windows Server 2003

Windows Server 2003 上で 4D を動作させる場合、推奨されるパフォーマンス設定があります。

- プロセッサのスケジュール:バックグラウンドサービスではなくプログラムに設定
- メモリ:システムキャッシングではなくプログラムに設定
- 仮想メモリ:カスタムサイズではなくシステム管理サイズに設定

これらの設定の場所は OS により異なりますが、探すべき設定の名前は上記のとおりです。これらは以下のステップで探すことができます。

1. スタートをクリックし、実行ボックス内に”sysdm.cpl”とタイプ
2. システムプロパティダイアログボックスで詳細設定タブをクリックし、パフォーマンス下の設定をクリック

これらの推奨設定は、ログインユーザーであろうとサービスとして実行されていようと、すべての 4D アプリケーション (スタンドアロンとサーバー) に当てはまります。

Windows Server 2003 で Microsoft はサーバーへの同時 TCP/IP 接続のためのキューサイズを減らすセキュリティ機能を有効にしました。これはレジストリの SynAttackProtect キーで確認できます。この設定はサービス拒否攻撃に対して有効ですが、4D で大きなロードが発生する状況では、有効な TCP/IP 接続がサービス拒否攻撃と判定されてしまうかもしれません。この問題に対しては以下のチップスを参照してください。

<http://kb.4d.com/search/assetid=76127>

4D Server を実行する 2003 サーバーマシンでは、以下で説明される通り、ネットワークプロパティをネットワークアプリケーションに最適化するようにしてください。

[http://technet.microsoft.com/en-us/library/cc728171\(WS.10\).aspx](http://technet.microsoft.com/en-us/library/cc728171(WS.10).aspx)

Windows Server 2008

4D のパフォーマンスを最高にするために、マシンのロールを「アプリケーションサーバー」にします。

[http://technet.microsoft.com/en-us/library/cc754024\(WS.10\).aspx](http://technet.microsoft.com/en-us/library/cc754024(WS.10).aspx)

これは Windows Server 2008 の「ロールの追加」ウィザードで行えます。

<http://technet.microsoft.com/en-us/library/cc732263.aspx>

Microsoft は Windows Server 2008 のシステムファイルキャッシング機能に問題を見つけています。この問題はシステムがファイルを頻繁にメモリにキャッシングしようとするというものです。このため Windows Server 2008 R2 を使用することがとても重要です。

<http://support.microsoft.com/kb/976618>

Mac OS X

不必要的サービスを無効にし、4D が利用できるリソースを最大にしてください。これは Mac OS X Server で特に重要です。

ベストプラクティス

- ハードウェアがサポートする限り、64-bit OS を使用
- Windows 7 や Windows 2008 Server を使用(TRIM をサポートしているため。詳細はハードウェアの項を参照)
- Windows XP 64-bit は避ける(実装がよくない)
- Windows Server 2003
 - プロセッサのスケジュール:バックグラウンドサービスではなくプログラムに設定
 - メモリ:システムキャッシングではなくプログラムに設定。
 - 仮想メモリ:カスタムサイズではなくシステム管理サイズに設定
 - SynAttackProtect の状態をチェックし、適切に設定
 - ネットワークプロパティをネットワークアプリケーションに最適化するよう設定
- Windows Server 2008
 - 「アプリケーションサーバー」ロールに設定
 - Windows Server 2008 R2 を使用
- Mac OS X
 - 不必要なサービスを停止

ハードウェア

この章では 4D を実行するハードウェアに関するベストプラクティスについて説明します。

メモリ

4D に関するメモリの話では主に 2 つの話題について取り上げます。

- アプリケーションメモリ
- データベースキャッシング

データベースキャッシングはアプリケーションメモリの一部を使用するので、これらは関連します。ここでは 2 つの問題について考える事にします。4D が利用できる総アプリケーションメモリを最大化する方法、およびキャッシングサイズを最大化する方法です。最後に非典型的な状況について説明します。

注

この節では OS のコンテキストで「プロセス」という用語を使用します。4D のプロセスではありません。Mac OS と Windows 両方で、「プロセス」は典型的にひとつのアプリケーションです。

注

この節ではバイナリ値を表現するために IEC 単位を使用します。例えば GB は 10 進値であり $10^9 = 1,000,000,000$ バイトです。Gib (Gibibyte) はバイナリ値であり、 $2^{30} = 1,073,741,824$ バイトです。メモリはバイナリ単位で指定されるため、バイナリ値を使用することが重要です。 2^{32} は 32-bit プロセッサー、 2^{64} は 64-bit プロセッサーです。

アプリケーションメモリを理解する

4D のアプリケーションメモリを最大化する方法を見る前に、アプリケーションメモリの意味を理解することが重要です。マシンの物理 RAM よりも大きなメモリ「アドレス空間」を実現するために、モダン OS はストレージ技術との組み合わせを使用します。RAM は物理メモリと仮想メモリという用語を使用して総メモリから区別されます。

総アプリケーションメモリは OS に直接関連します。OS がメモリを管理する方法が、アプリケーションに割り当てるメモリ量を決定します。実際 OS の制約は 4D が使用することのできるメモリ量を計算しようとするだけでもある種混乱するシナリオのもととなります。ここではこの問題について検証します。

物理メモリ

物理メモリという用語は典型的に RAM を指します。RAM は高速なメモリですが、相対的に高価です。なので RAM の量は一般的に総アドレス空間サイズよりもはるかに小さくなります。

OS が使用できる物理メモリ量は 32-bit か 64-bit かによって基本的に大きく異なります。1 つのメモリアドレスはトータルで 32-bit または 64-bit のいずれかであり、1 バイトのメモリを参照します。

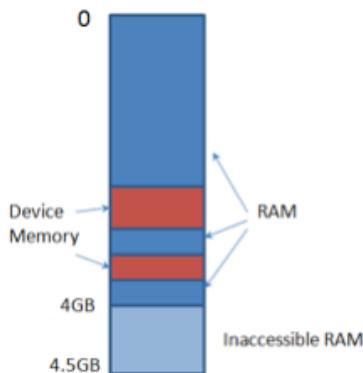
- 32-bit は 2^{32} のストレージをアドレスできる
 - 4,294,967,296 アドレス
 - 4GiB
- 64-bit は 2^{64} のストレージをアドレスできる
 - 18,446,744,073,709,551,616 アドレス
 - 16Eib (16 exbibytes)

つまり 32-bit OS は 4GiB のメモリのみにアクセスできますが、64-bit OS ではさらにその 4,294,967,296 倍をアドレスできます。

事実上 OS がサポートできると主張する物理メモリ量はそれよりも少なくなります。ハードウェアの制限と SKU の制限が最も一般的な制限です。

32-bit OS ではハードウェアの制限があります。

- 総 RAM とマシンの他の物理メモリが「物理アドレス空間」を構成します。他の物理メモリはビデオカードやネットワークアダプターのそれが含まれます。
- メモリマップ I/O (MMIO) と呼ばれる技術を使用することで、OS は物理アドレス空間内のハードウェアアドレスを追跡します。さらには、互換性のために、32-bit Windows は 4 GiB を上限とするハードウェアアドレスをマップします。例えば 32-bit マシンに 1 GiB RAM のビデオカードと 4 GiB のメインメモリが搭載されている場合、アドレス可能な総メインメモリは 3 GiB です。他のハードウェアデバイスは一般的に RAM を搭載していて、結果実際に利用可能な総 RAM はさらに減少します。各ハードウェアの RAM により、メインメモリは 4 GB の外側に押しやられてしまいます。



- Intel は、36-bit のメモリアドレスを使用できるようにする物理アドレス拡張 (PAE) と呼ばれる技術を開発しました。
 - しかし PAE を使用しても、すべての 32-bit Windows クライアント SKU は最大 4 GB の物理メモリをサポートします。これが事実上の制限であることが分かります。32-bit Windows に 4 GB を超える物理メモリのサポートを追加しようとする試みにおいて、Microsoft はテストしたシステムがクラッシュ、ハング、さらにはブート不能になることを発見しました。これは不適切に実装されたデバイスドライバーに起因します。問題のあるクライアントドライバーエコシステムのため、理論的にアドレス

可能であるにもかかわらず、クライアント SKU は 4 GB を超えるところに存在する物理メモリを無視する決定を行いました。

64-bit OS も MMIO を使用しますが、64-bit アドレス（およびアプリケーション）は容易に 4 GiB を超えるメモリにアクセスできるので、ハードウェアのアドレスは実際の制限になりません。

製品の違いによるメモリの制限も一般的です。例えば Microsoft は製品ごとにサポートする物理メモリ量を変えています。以下は Windows Server 2008 R2 (64-bit エディション) でサポートされる製品ごとのメモリ量です：

バージョン	制限
Windows Server 2008 R2 Datacenter	2 TB
Windows Server 2008 R2 Enterprise	2 TB
Windows Server 2008 R2 Foundation	8 GB
Windows Server 2008 R2 Standard	32 GB
Windows HPC Server 2008 R2	128 GB
Windows Web Server 2008 R2	32 GB

完全なリストは以下で確認できます：

[http://msdn.microsoft.com/en-us/library/aa366778\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/aa366778(v=vs.85).aspx)

製品による違いに加え、2 TB の制限が事実上の上限である点に留意してください。Microsoft はハードウェア上でのテストを行い、Windows Server 2008 では 2 TB 物理メモリを最大量としています。

Apple の Mac コンピューターは理論上の一般的なリストを公開していますが、OS というよりはマシンごとにサポートされる RAM が決められています。

仮想メモリ

仮想メモリという用語はページメモリと混同して用いられてきました。ページメモリは仮想メモリではありません。ページメモリストレージは仮想メモリの一部であることを理解することがポイントです。

注

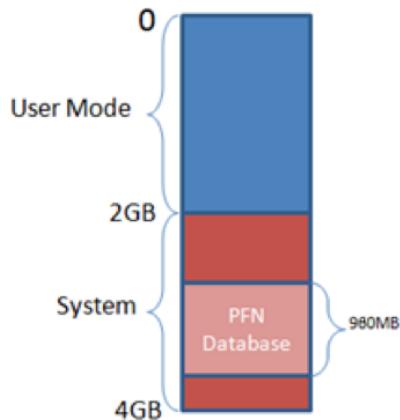
メモリのページングとは、物理メモリのページをディスクストレージにコピーし、物理スペースを他の目的で使用できるようにするものです。ページングにより、モダン OS ではマシンにインストールされた物理メモリを超えるメモリにアクセスできるようになっています。他方ページングにはパフォーマンスのコストがかかります。

仮想メモリは特別なハードウェア部品というよりはコンセプトです。仮想メモリは OS がアクセスできる総アドレス空間のことです。このアドレス空間の一部にはまったく物理的な実体がないこともあります。例えば Mac OS X は 64-bit メモリアドレスを使用する場合は、16 EiB のメモリをアドレスできるかもしれません。しかしこの量のメモリがインストールされた Mac マシンは存在しません。仮想メモリは文字通り仮想なのです。

Mac OS X と Windows のすべてのアプリケーションはこの仮想アドレス空間の一部にアクセスします。この部分は Windows で「ユーザー モードアドレス空間」、Mac OS X で「プロセスアドレス空間」と呼ばれます。メモリが要求されると (4D キャッシュのようにそれが即座に使用されない場合でも) 仮想アドレス空間のアプリケーション部からのアドレスが割り当てられます。

物理メモリと同様、32-bit アプリケーションは論理的に 4GiB の仮想メモリにアクセスでき、64-bit アプリケーションは 16 EiB にアクセスできます。しかし実際のアプリケーション用仮想メモリサイズは以下のように変化します：

- 32-bit OS におけるプロセスごとの仮想メモリは、以下のような理由で実質利用可能な部分は少なくなります。
 - システムによる予約: 例えば Windows は空間の半分をシステムに予約し、残りをプロセスに与えます。これは 32-bit Windows においてプロセスがアドレス可能なメモリは 2 GiB に制限されることを意味します。
 - 搭載された RAM を超えるメモリにアクセスするためにそれらの場所へのマッピングを格納する技術が 32-bit OS に存在します。しかしこのマッピングはプロセス用の仮想メモリに格納されなければなりません。例えば Windows でこのアドレス可能なメモリを定義するデータは PFN データベースと呼ばれます。PFN データベースはプロセスがアクセスできるシステムメモリの一部を使用します。



- /3GB パラメーターを使用して Windows がシステムに予約するメモリを半分に強制することができます。結果 32-bit プロセスは 3 GiB にアクセスできます。
- 64-bit OS の場合そのような問題はありません。OS はシステムメモリや PFN データベースを処理する十分な上部空間を持つので、各 32-bit アプリケーションは 4 GiB 仮想アドレス空間全部にアクセスできます。しかし物理メモリのように事実上、64-bit Windows のプロセスは 8 TB 仮想メモリに制限されます。

以上のことから、アプリケーションメモリは実際のところ仮想メモリです。仮想メモリには物理メモリが含まれるため、このコンテキストで 4D がアクセスできる最大物理メモリ量について話すことは、重要ではありません。物理メモリを増やす（ページングが少なくなることで）パフォーマンスが良くなります。アクセス可能な最大メモリサイズが増えるわけではありません。しかし先に示した OS の制限のため、物理メモリ量は 4D がアクセスできる総仮想メモリ量に影響します。

アプリケーションメモリを測定する

4D は物理メモリではなく仮想メモリにアクセスします。そこで問題は「4D が使用できるメモリ量」ではなく「4D が使用できる仮想メモリの量」です。

この質問に答えるため仮想メモリ量を知る必要があります。Mac OS X と Windows には多くの異なるメモリ測定ツールがあります。

Windows

タスクマネージャー、リソースモニター、Perfmon により以下の計測が得られます：

- ワーキングセット - プロセスが使用中の物理メモリ (RAM)。仮想メモリのため、これは総使用メモリを正しく反映したものにはなりません。
- コミットサイズ - ページファイル以外のバックストアがない、ページ可能な仮想アドレス空間の総量。4D からすると、これがほぼ最大メモリサイズとなります。

Windows には他の計測方法もありますが、4D が使用する総仮想メモリ量を実際に表示することはできません。

Mac OS X

アクティビティモニタを使用して以下の計測を行うことができます:

- 実メモリ - 現在のプロセスが保持する物理ページの大まかな値。このコンテキストではこの値は重要ではありません。物理メモリ量は 4D が使用している総仮想メモリ量についてまったく反映していないからです。4D は仮想メモリを使用し、OS が物理メモリを使用するかどうか決定します。
- 仮想メモリ - 4D が実際に使用している仮想メモリ量 (GUI に表示される際に丸められますが)。

実際のところ両 OS とも、4D が使用する総メモリ量を計測するためのベストなツールはありません。Windows では利用可能な計測ツールで、プロセスが使用する仮想メモリを実際に計測するツールはありません。この情報は Windows API を使用してプロセスが取得できますが、レポートツールには表示されません。Mac OS X ではより状況がよくなり、仮想メモリの計測は正確です。実際は両プラットフォームで 4D アプリケーションが使用的メモリを計測する良い方法があります。

GET CACHE STATISTICS

GET CACHE STATISTICS (GCS) コマンドを使用して使用仮想メモリ量を取得できます。

例えば以下の条件であるとします。

- 64-bit OS
- 32-bit 4D アプリケーション
- ループの中で、4D が使いきるまでメモリを割り当てる

GCS から返される最大使用仮想メモリサイズは 4 GiB あるいは 4,294,967,296 bytes (GCS はバイト単位で値を返します) に近くなるはずです。GCS から返される値がこの値に近くなると、アプリケーションはメモリをほぼ使い尽くす状態になります。

この例において、マシンにインストールされた RAM の量は関係がないことに留意してください。4D はいずれにしても 4 GiB の仮想アドレス空間を割り当てることができます。インストールされた RAM は 4D が使用できるアクティブな物理メモリ量を決定するだけです。ページングにより、すべての仮想メモリにアクセスが可能です。RAM を増やすことでパフォーマンスは向上しますが、最大アドレス可能メモリサイズが変わることはありません。

結果 4D が使用するメモリを計測する最も良い方法は GET CACHE STATISTICS を使用することです。

4D 用に最大メモリを獲得する

4D が使用するメモリを計測するのと異なり、4D が利用可能なメモリを最大化することはマシンの物理メモリに直接関連します。先に示した制限により、ベストパフォーマンスを引き出すためにはマシンに十分な RAM が搭載されていなければなりません。

1 GiB しか RAM が搭載されていないマシンでも、32-bit 4D は 4 GiB の仮想メモリにアクセスできます。しかし OS はページングを煩雑に行わなければならないため、パフォーマンスが落ちることになります。他方仮想メモリの実装上、RAM のみを使用するよう強制することはできません。常にいくつかのページファイルは使用されます。

すべてのケースで 32-bit 4D は 4 GiB の仮想メモリにのみアクセスできます (先に示した OS の制限により減らされる場合があります)。4D Server v12 64-bit にはメモリアクセスに事実上制限はありません。しかし先に説明したとおり、Microsoft は人為的に制限を設けています。以下は十分な RAM があると仮定したときに 4D がアクセス可能な総メモリのまとめです:

- 32-bit OS - 2GiB
- PAE が有効な場合 3 GiB
- 64-bit OS, 32-bit 4D - 4 GiB
- 64-bit OS, 64-bit 4D Server - 8 TB (Windows のみ)

4D 用のメモリを最大化させるために以下のガイドラインに従ってください:

- 64-bit OS を使用する
- 32-bit 4D アプリケーションが 4 GiB のアドレス全体にアクセスする必要がある場合、マシンに最低でも 6 GB の RAM をインストールします。
- 64-bit 4D Server の場合事象はより複雑です。32-bit 4D は 4 GiB の固定された制限があり、そのため追加の RAM がどれくらいあればよいか示すことは簡単です。64-bit 4D Server には事実上制限がないため、アプリケーションに関する分析が必要です。キャッシュサイズとアプリケーションの 4D プロセスが使用する量でメモリ量が決定されます。例えば 4D アプリケーションがキャッシュに 100 MB を使用し、ランタイムにもう 100 MB のみを使用する場合、6 GB の RAM をインストールする必要はありません。ひとつのゴールは、最大メモリ使用量に十分な RAM を用意することです。例えばデータベースが 16 GB のキャッシュを持ち、最大ユーザー接続時に追加で 3 GB のメモリを使用するとき、インストールすべき RAM はおよそ 36 GB です。
- OS システムキャッシュを忘れないでください。4D が使用しないメモリは他のアプリケーションや OS プロセスだけのためのものではありません。Mac OS X と Windows 両方において、システムキャッシュは重要です。
- 4D の最大メモリ使用量を計算する方法については次節を参照してください。

関連する資料

以下の資料はメモリアクセスについて理解するために有用です。

- Windows
 - <http://blogs.technet.com/b/markrussinovich/archive/2008/07/21/3092070.aspx>
 - <http://support.microsoft.com/kb/888732>
 - <http://support.microsoft.com/?kbid=291988>
 - [http://msdn.microsoft.com/en-us/library/aa366778\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/aa366778(v=vs.85).aspx)
- Mac OS X
 - http://developer.apple.com/library/mac/#documentation/Performance/Conceptual/Memory_Mgmt/Articles/AboutMemory.html
 - nagingMemory/Articles/AboutMemory.html
 - http://support.apple.com/kb/TA27734?viewlocale=en_US

4D のメモリ使用量を計算する

4D アプリケーションのランタイムメモリ使用量を計算するよりも、検証によって推測することが可能です。推測可能なサイズが主に 3 つのあります。

- データベースキャッシュ
- プロセススタックサイズ
- プロセスおよびインタープロセス変数

データベースキャッシュサイズは直観的です。計算にはデータベース設定で指定した値を使用します。

プロセススタックサイズについては後述します。

コンパイラーが出力するシンボルテーブルを使用してデータベースの変数が使用するメモリコストに関してヒントを得られます。シンボルテーブルを生成して各変数のサイズを合計します (倍長整数と実数は 4 バイト、整数は 2 バイト、テキストは 2 バイト/文字等)。

次に、実際にどのように計算するか、例を挙げて行います。

注

このテクニックはデータベースのメモリ使用量の指標を計算するのに有効です。実際に使用されるメモリを計算するものではありません。この指標を使用してハードウェアの仕様を決定したり、あるいは全体的なスタックサイズを経らう執拗があるなどの決定を行うことができます。

基本的な必要メモリ量

クライアントプロセスがない状態の 4D Server アプリケーション。

システムライブラリ	1,000 MB
キャッシュ	1,200 MB
インタープロセス変数	10 MB
4D Server スレッド	10 MB
合計	2,220 MB

- システムライブラリに 1,000MB (概算)
- キャッシュに 1,200MB (データベース設定で設定した値)
- インターパロセス変数用の 10MB は、いくつかの大きな BLOB が使用されることも考慮したおおよその上限値です。
- カーネルのプリエンプティブ / コオペラティブスレッド (キャッシュマネージャー、インデックスビルダー、インターフェース、サーバーリスナー等) 用に~10MB

プロセスごとに必要なメモリ量

ここではクライアントプロセスのスタックサイズと、(シンボルファイルから計算可能な) プロセス変数のコストについてみていきます:

注

SDP は *SET DATABASE PARAMETER* のセレクター 53 を使用してプリエンプティブプロセススタックサイズを減らしたときの値です。

プロセス	デフォルト	SDP 512K	SDP 256K
コオペラティブ (twin)	.25 MB	.25 MB	.25 MB
プリエンプティブ (twin)	1 MB	.5 MB	.25MB
小計	1.25 MB	.75 MB	.5 MB
SQL (triplet)	1 MB	.5 MB	.25 MB
小計	2.25 MB	1.25 MB	.5 MB
500 KB のプロセス	.5 MB	.5 MB	.5 MB
twin との合計	1.75 MB	1.25 MB	1 MB
triplet との合計	2.75 MB	1.75 MB	1.25 MB
1.5 MB のプロセス変数	1.5 MB	1.5 MB	1.5 MB
twin との合計	2.75 MB	2.75 MB	2 MB
triplet との合計	3.25 MB	2.75 MB	2.25 MB

SQL プロセスは SQL 呼び出しを処理するために必要に応じて作成されるため、別に記載しています。クライアントから SQL 呼び出しが行われない場合、この部分は計算から除外できます。

2 つのプロセス変数の例題は以下のように選択されています。

- 500KB プロセス変数/プロセスはスカラー変数のみを使用する場合に妥当です。
- 1.5MB プロセス変数/プロセスはテキスト、BLOB、ピクチャー、配列等を使用する場合を反映しています。

計算の例題

以下の条件を仮定します。

- 32-bit 4D Server
- 64-bit OS
- 6GB RAM
- 4D SQL コマンドは使用しない
- アプリケーションプロセス内で処理を実行する (1 クライアントごとに 1 プロセスのみ)
- 500 同時ユーザーをゴールとする

32-bit 4D Server の最大仮想メモリは 4GiB であることを覚えておいてください。先の計算式を適用してこのシナリオでのサイズは以下の通りとなります。

500 KB 変数	デフォルト	SDP 512K	SDP 256K
基本	2,220 MB	2,220 MB	2,220 MB
500 プロセス	875 MB	625 MB	500 MB
合計	3,095 MB	2,845 MB	2,720 MB

1.5 MB 変数	デフォルト	SDP 512K	SDP 256K
基本	2,220 MB	2,220 MB	2,220 MB
500 プロセス	1,375 MB	1,125 MB	1,000 MB
合計	3,595 MB	3,345 MB	3,220 MB

警告

この計算は簡略化しております。通常のオペレーション下でサーバー上で実行される処理 (セット、配列、BLOB、ピクチャー、トランザクション等) については考慮されていません。しかし見積もり段階でも、デフォルトスタック設定である 1.5MB 変数サイズ/プロセスは 4 GiB の制限に対しすでに危険域であることが分かります。500KB/プロセスの SDP 256K 設定に比べ、1GB 近い追加のメモリが必要となっています。後者の設定の場合、パフォーマンスのためにデータベースキャッシュサイズを増やすことができます。

データベースキャッシュ

様々な意味で、キャッシングは 4D におけるすべてのデータベースエンジン I/O の核です。データファイル、ストラクチャーファイル、インデックスファイル、複製ヘッダー等はキャッシングを経由して行われます。キャッシングは 4D に不可欠で重要な部分です。

4D v11 SQL で、キャッシングは劇的に改善されました。64-bit メモリアドレスのサポートや 4D におけるメモリ管理の向上などにより、キャッシングサイズをより大きくすることができるようになりました。キャッシングは特にサイズの増加に対して、より効率的になるよう再設計されました。さらには開発者がキャッシングの利用状況追跡やトラブルシュートができるようにするための新しいツールが追加されました。

4D v12 でもキャッシングの進化は継続していて、パフォーマンスが改善し、管理が向上しています。

ここでは、4D データベースキャッシングに関するベストプラクティスについて説明します。

最大キャッシング

キャッシングはアクセスされるデータをディスクから RAM にコピーすることで、パフォーマンスを向上させる方法のことです。理論的にキャッシングが大きくなれば、4D のパフォーマンスは向上します。実際は常にそうなるとは限りませんが、原則は有効です。

4D のキャッシュサイズを最大化させることに注目した場合、2 つの考慮すべきポイントがあります:

- キャッシュは最大どれくらいになるか
- それをサポートするためにマシン仕様はどうあるべきか

データベースキャッシュの最大サイズは以下の通りです:

- 32-bit: 2,384 MB
- 64-bit: 事実上無制限 (Windows では 8TB、Mac OS X では 16 EiB)

4D v11 SQL では最大キャッシュサイズが 4,095 MB とされていましたが、これは大きすぎる値でした。32-bit アプリケーションはトータルで 4 GiB の仮想メモリにしかアクセスできないので、4,095 MB のキャッシュを設定すれば 4D に残されたメモリはほぼなくなってしまいます。32-bit の 4D の最大キャッシュサイズは 2,384 MB に減らされました。4D に残されたメモリが適切でない場合、クラッシュしてしまうからです。一方でキャッシュが少なすぎれば動作が遅くなります。

64-bit の場合、キャッシュの制限は事実上ありません。64-bit アドレスである最大 16 EiB に達するシステムはまだないからです。必要なハードウェア構成が準備されている限り、4D Server v12 64-bit でデータファイルに適切なキャッシュサイズを必要なだけ設定できます。

ただし最大キャッシュサイズを実現することは、データベース設定に大きな数値を入力するだけでは不十分です。もっとも大きな問題は、キャッシュで使用するメモリが連続していなければならないということです。4D は起動時に、OS に対し連続したメモリ領域を要求します。4D は OS から完全に孤立して実行されることはありません。すべてのアプリケーションプロセスはシステムライブラリやドライバーを必要とします。これらのアイテムはすべてプロセスマメリ空間の一部を使用します。結果 4D がキャッシュメモリを要求しても、要求した分 OS から渡されるとは限りません。これは特に 32-bit OS の制限となります。そこでキャッシュを最大にする推奨設定は以下のとおりとなります:

- 32-bit 4D
 - すべてのケースで 64-bit OS を使用します。
 - 最低 6 GB の RAM をインストールします。
- 64-bit 4D Server
 - 当然 64-bit OS が必要です。
 - 必要なサイズを割り当てることができます (RAM が少ないマシンでも)。
 - しかし最低でもキャッシュとしてリクエストするサイズと OS が使用する追加分の RAM を実装してください。これはいくつかの理由で重要です:
 - システムキャッシュ:
Windows と Mac OS X 両方とも、ディスク内容の RAM キャッシュを使用します。これはシステムキャッシュと呼ばれます。4D がすべてのメモリを独占することでシステムに逆らつたりしないようにすることが重要です。システム自身がファイルキャッシュを行えるようにメモリを残しておかなければなりません。
 - ページング:
4D とシステムは両方とも大量の仮想メモリを消費するかもしれません。ページングが発生するまでにメモリを消費すると、4D キャッシュはほとんど無視されます。
 - 4D がディスクにキャッシュをフラッシュしたいとします。
 - 4D はキャッシュからページを読み取ろうと試みます。しかしページはすでにディスクにページアウトしているため、ページインする必要があります。つまり 4D はディスクから読み取りを行います。これはデータベースキャッシュの目的に全く反する状況です。
 - キャッシュページを RAM に読み込んだのち、4D は変更をフラッシュ (ディスクに書き込み) する必要があります。
 - さらに OS はペナルティを支払います。4D が要求するページを RAM に読み込むために、何かをページアウトしなければならないからです。

フラッシュの間隔

4D v12 のデータベース設定でのデータをディスクに保存する間隔のデフォルト値は 20 秒です。以前のバージョンの 4D では 15 分でした。

4D v12 ではキャッシングマネージャーが改善され、フラッシュするものがキャッシングに少ないほどより効率が良くなりました。つまりより頻繁にフラッシュを行ったほうが効率的だということです。ゴールはフラッシュサイズをより少なくすることです。これは特に SSD を使用する際に有効です。

注

変換されたデータベースでは以前に設定された値が使用されます。しかしキャッシングのパフォーマンスに問題を感じない場合は、この設定を変更する必要はありません。

動的キャッシングの計算

注

これは前述のデータベース設定と同じ内容です。説明を分かりやすくするために、改めて説明しているだけです。

この設定を選択しても、動的にキャッシングサイズが変化するわけではありません。また、実行時のリソースの状態に応じてキャッシングサイズを計算するものでもありません。例えばマシンの空きメモリや実行アプリケーションの数を考慮に入れるものではありません。

この設定の目的は単純です。インストールされる環境を事前に開発者が知ることができないようなときに使用するものです。この場合マシンメモリを独占しないよう注意深くキャッシングサイズを設定しなければなりません。使用されるハードウェア構成を事前に知ることができませんから、インストールされたハードウェア構成に基づき、キャッシング量を計算する仕組みが用意されています。

マシン構成が分かっている場合、動的キャッシングの計算を使用する優位性はなく、もつと言えばキャッシングサイズの設定が不必要に複雑になります。この場合のベストプラクティスは動的キャッシングの計算をオフにして直接値を設定することです。

最適でないキャッシング

この節では可能な最大のキャッシングサイズを実現する方法を説明します。これは最適なハードウェア構成のベストケースシナリオからの推定になります。ハードウェアが最適でない場合はどうでしょう。どれくらいキャッシングを設定すればよいでしょうか。また最大キャッシングサイズが必要と仮定します。これはまれなケースです。とても大きなデータベースでリクエストがたくさんやってくるようなシステムだけが大きなキャッシングサイズを必要とします。

キャッシングは 4D が使用するメモリの一部に過ぎないこと、また OS にメモリを要求するのは 4D だけではないことを考慮することが重要です。他方キャッシングを小さすぎないようにすることも重要です。実際のシナリオを想定してみました。

- 3 GB RAM マシンで 2.3 GB キャッシュを設定した場合
 - このような設定では OS によるページングが頻繁に発生し、パフォーマンスが低下します。
- 16 GB マシンに 25 ユーザーが接続する場合で、100 MB をキャッシングに割り当てる
 - 以前のバージョンでは 100 MB がデフォルトでしたが、現在は 400 MB に引き上げられています。
変換されたデータベースではキャッシングが少なすぎないかチェックしてください。

キャッシュを取りすぎてマシンのメモリを占有しないよう注意してください。他方小さすぎないようにすることも重要です。

最適なキャッシュサイズを決定する方法は、キャッシュの利用量を実際に観察することです。第3部でキャッシュ解析について説明しますが、ここで例を紹介します。

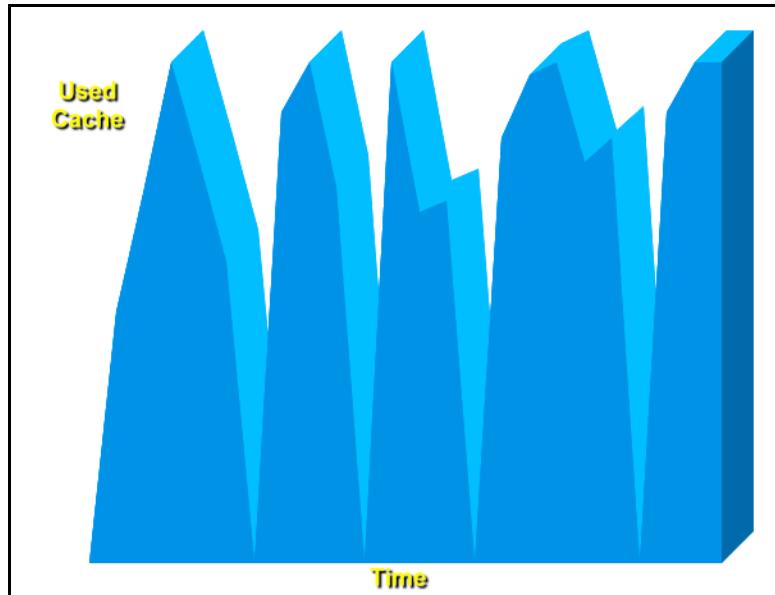
キャッシュの解析は GET CACHE STATISTICS コマンドを使用します。

GET CACHE STATISTICS(1 ; arrNames ; arrValues ; arrCount)

このコマンドは使用キャッシュサイズを返すので、このコマンドを繰り返し呼び出してキャッシュの利用量を追跡することができます。

以下の図はキャッシュが少なすぎる場合の例です：

通常のキャッシュページは総キャッシュ量の 25%です。この例ではすべてのキャッシュが何度もページされています。これはキャッシュが少なすぎる典型例です。



Solid-State Drive (SSD)

SSD の利用が推奨されています。この節ではなぜ SSD が推奨されるのかについて説明します。

4D はディスクを集中的に使用するアプリケーションです。一般的にハードディスクはフラッシュメモリや RAM に比べて遅いメディアであると認識されています。今まで SSD を推奨する文言は HDD がいかに遅いメディアであるかを強調することで行われてきました。しかし HDD を SSD に置き換えるべきがうまくいくのでしょうか。

この節では 4D を中心に据えて、この疑問を解消するための調査を行います。

なぜストレージの速度が重要なのか

SSD が 4D に与える影響を考慮するには、4D にとってストレージがいかに大切な理解する必要があります。他のデータベースと同様、4D はディスクを集中的に使用するアプリケーションです。データはデータファイルとしてディスク上に存在しますが、それ以外にも以下の関連する部分に目を向ける必要があります。

- データとインデックスはそれぞれデータファイルやインデックスファイルとしてディスク上に存在します。クエリ速度を最大化させるためにはインデックスを可能な限り素早く読み込む必要があります。
- 4D はキャッシュに連動してディスクを使用します。キャッシュ速度のためにストレージ速度も重要となります。
- インデックスはディスクを使用して構築されます。インデックスが作成される際 "temporary files" フォルダーにファイルが作成されるのを見ることがあるかもしれません。
- データログファイルのためにも素早いディスクアクセスが重要です。すべての 4D アプリケーションにとってデータログファイルは必須であると考えるべきです。このファイルはデータベースにエラーが発生したとき、素早く復旧するために使用されます。最新のバックアップから最新のデータベース更新までの処理を統合するために使用されるものです。
- キャッシュフラッシュ速度はディスク速度の影響を直接受けます。

これらは 4D の中で重要な部分の一部であり、ディスクに依存し、ゆえにパフォーマンスが重要となります。

素早いデータアクセスは重要ですが、いくつかの理由からこれらの論点はデータベースデザインに関連付けられます。

- 4D はデータアクセスの多くの領域ですでに最適化されていて、必要なデータだけをロードするようになっています。
- 4D はディスクに関連するパフォーマンス低下を緩和するため、RAM 中のデータキャッシュを使用します。最もアクセスされるデータがキャッシュ中に保持されるようデータベースがデザインされていれば、ディスクアクセス速度はさほど重要な要素ではありません。

4D にとって最も重要なのは読み書きのランダムアクセス速度です。データベースとして 4D は相対的にディスクの小さなピースにアクセスします。4D は数ギガバイトのファイルを移動させる用途には通常使用されないので、シーケンシャルアクセスはさほど重要ではありません。ランダムアクセスの速度が重要です。

ディスクの速度が何故それほどに重要なのか

ディスクが速くなれば 4D が速くなる、これは明白のように思えます。なぜこのポイントについて話をする必要があるのでしょうか。

ここでの新しいトピックは、4D にとってディスクがどれほどのボトルネックになり得るのかが一般的に理解されてきていたなかったことにあります。今まで HDD よりも速いディスクを使用することが難しかったことが影響しているでしょう。RAM ディスクや SSD などは高価すぎて典型的な 4D アプリケーションでは使用することが困難でした。

ここ数年で SSD の値段は低下し、ボトルネックの一つを解消する機会を得ることができますようになりました。SSD を使用することによるパフォーマンスの向上は衝撃的です。

注

他のハードウェア検討事項と同様、データベースのパフォーマンスはデータベース設計に依存します。あなたの 4D アプリケーションではボトルネックがディスクではないかもしれません。ただ SSD が試験のためには十分安価になり、ディスクを交換することでもたらされるパフォーマンスの改善を見ることができるようになった点が異なります。

ただし、正しい SSD を選択するために注意しなければならないことがあります。SSD の歴史はまだ浅く、革新は今も続いている。ここでは 4D デベロッパーに SSD に一般情報と 4D に関連する事柄を紹介します。またベンチマークも提示します。

SSD を理解する

SSD のテクノロジーについて手短に説明しましょう。詳細情報が必要であれば、オンライン上に多くの SSD に関する情報があります。

SSD は HDD と同じコンポーネントで構成されています。

- コントローラー
- RAM キャッシュ
- ストレージメディア

SSD と HDD の主な違いはストレージメディアです。ハードドライブはメタルコーティングされた回転する磁気ディスクを使用します。データの読み書きを行うために機械的なアームが動きます。SSD はフラッシュメモリを使用します。

SSD の利点

パフォーマンスの観点から SSD を見た場合:

- 動く部品がない
- ファイルのフラグメンテーションがパフォーマンスに影響しない
- コントローラー (この点は少し複雑であるため別途説明します)

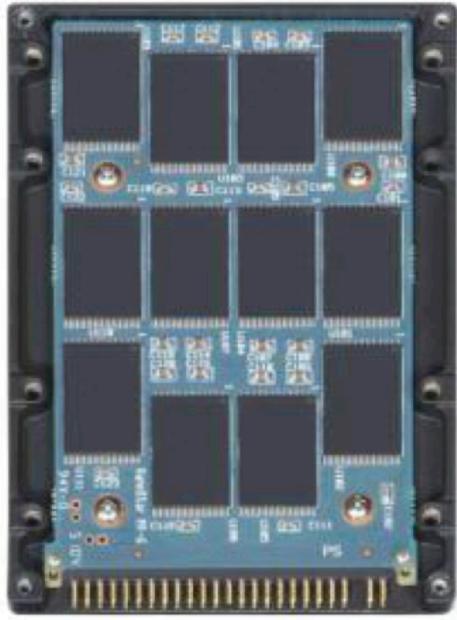
動く部品がないことは SSD に以下の利点をもたらします。

- 動作させるためのパワーが少ない
- より低温で動作する
- 特にシークタイムとランダムアクセスに関してより速く動作する

HDD においてファイルフラグメンテーションがパフォーマンスに影響を与えることは、動くパーツが存在することに直接関連します。散在したファイルを読み込むためにアームやディスクが動かなければなりません。SSD にはこの動作がないので、ファイルのフラグメントはまったく速度に影響しません。



Traditional hard disk drive



Solid state hard drive

SSD の欠点

SSD には欠点もいくつかあります。

- コスト
- 容量
- 寿命 *
- 使用するに伴い遅くなる

SSD は HDD に比べるとまだまだ高価です。4D を実行するためにそこそこの SSD を購入して\$1,000 費やすとしましょう。それでも容量は 100-500 GB 程度です。フラッシュ技術は進化していて、容量も増える傾向にありますが、HDD 今日主流である 2TB 程度のものを購入しようとなれば何千ドルもかかります。

寿命の問題についていえば、HDD で使用される磁気ディスクよりも、フラッシュメモリの寿命は短いものです。より詳しく説明すると、1 つのフラッシュセルは書き込み処理回数に上限数があります。他方 HDD には動く部品があるので、メディアが壊れる前に他の部品が壊れるかもしれません。

注

寿命については製造会社が発表を行っています。この点については後ほどさらに詳しく説明しますが、寿命自体は実際の問題ではありません。

SSD の詳細

ここでは、断片化やコントローラーなどについても説明しますが、それらはもっと重要な話題の中で出てくるトピックに過ぎません。その話題とは、SSD は使用していく中で遅くなるという事です。

SSD のパフォーマンス低下について理解する

すべての SSD は使用 (書き込み処理) する間にパフォーマンスが低下していきます。これはフラッシュが HDD とは異なり、直接データの上書きを行えないことに起因します。書き込みをできるようにするためににはまず消去を行わなければなりません。これはアーキテクチャー上の制限です。これはフラッシュメモリ以外のストレージを使用しない限り回避できません。

注

SSD メーカーは、すでにこの問題に取り組んでおり、効果的なソリューションを提供しています。

モダンな OS 上でファイルを削除しても、ドライブレベルではなにも行われません。OS はファイルアロケーションテーブル (FAT) を更新してその空間が空いていることを宣言しますが、ドライブ上に実際のデータは残っています。もちろん新しいファイルが書き込まれる際には、古いファイルが上書きされることになります。

注

この仕組みによりファイル復旧ソフトは動作しています。そういったソフトはドライブ上に存在するファイルを復旧させているだけです。

HDD のように SSD も固定長のブロックに分割されています。各ブロックにはメモリの複数ページが含まれます。SSD は一度に複数ページを読み書きできます。しかし削除はブロック単位でしか行えません。

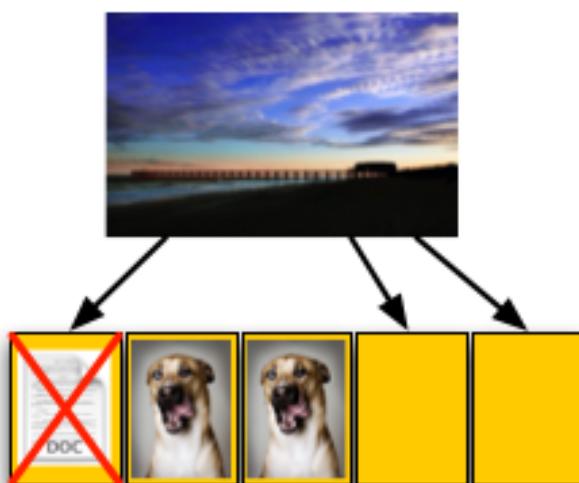
そこで SSD はどのファイル (ページ) が削除されたのか知ることができず (問題 1)、また SSD がファイルを書き込む際、削除はブロック単位でしか行えません (問題 2)。

使用するにつれ、ブロックには使用中のファイルと削除されたファイルの両方が含まれるようになります。その結果、すべての書き込み処理は複数回の書き込み処理になります。

書き込み処理増幅

SSD にファイルを保存する処理を考えてみましょう。この例題では 1 ブロックに 5 ページあります。このシナリオで 1 ページを使用するドキュメントファイルが SSD に書き込まれ、後で削除されます。そして 2 ページを使用する犬の画像もディスクに存在します。ここで 3 ページを表示する海岸の画像をディスクに書き込みます。次の図 ×印のボックスは、削除されたドキュメントが占めている1ページを現しています。

1



SSD は、どのブロックが使用中であるか分かりますが、先に説明したとおり、ドキュメントファイルが削除済みであることが分かりません。SSD は、ただブロック全体を削除することしかできず、それは既存のデータも消去することになり、安全ではありません。ではどうやってファイルを保存できるのでしょうか。

まず SSD はキャッシュ/バッファーにブロックを読み込まなければなりません。次の図の Cache と銘打ったボックスが、キャッシュ/バッファーだとイメージしてください。

2



図のように1ブロック、5ページの情報を読み込んだら、空きスペースを確認し、削除されたファイルをクリアします。(これはキャッシュ中で行います、ディスクではありません)。

3



次にキャッシュ中に新しいファイルを書き込みます。

4



最後に元のブロックを消去し、新しいブロックをディスクに書き込みます。

5



結果たった 3 ページのデータを書き込むために、ドライブは実際 5 ページを書き込みます。すべてのブロックに使用中のページが存在するようになると、すべての書き込み処理が読み込み/更新/消去/書き込みになります。

この問題は書き込み処理増幅と呼ばれ、以下の式で表すことができます。

$$\frac{\text{USER DATA WRITTEN}}{\text{ACTUAL DATA WRITTEN}} = \text{WRITE AMPLIFICATION}$$

これはすべてのブロックに使用中のページが含まれるようになるまで性能が低下しますが、それ以上は進行しない点に留意してください。

注

2011 年において利用可能な SSD の典型的な性能低下は 25-35% です。

SSD メモリ：フラッシュセル

先に示した通り、SSD フラッシュセルの書き込み処理には寿命があります。SSD メインストリームは典型的に以下 2 形式のいずれかを使用します。

- マルチレベルセル (MLC)
- シングルレベルセル (SLC)

これらの違いは、各セルが表現するビット数です。SLC はビット毎に 1 セルを使用します。MLC は 2 ビットで 1 セルを使用します。MLC は 2 倍のビットを表現できるので、より安く製造できます。他方 SLC は寿命が長くなります。MLC はセルに 2 ビット含まれるので、各セルは SLC に比べ 2 倍の書き込みが行われるからです。

ここで指摘したことにより、SSD を使用することに不安を感じるかもしれません、そのような必要はありません。次の節を見てください。

コントローラー

コントローラーは SSD にパフォーマンスにおいて重要な役割を持っています。

実際 SSD コントローラーは様々な SSD の違いを表す一つの機能です。ほとんどの製造者はフラッシュメモリを同じ所から仕入れています。なのでメモリの速度や寿命は製品の違いにはなりません。コントローラーにより製品に違いが出るのであります。

コントローラーの機能は 2 つのカテゴリに分けられます。

- 寿命管理
- パフォーマンス

寿命管理

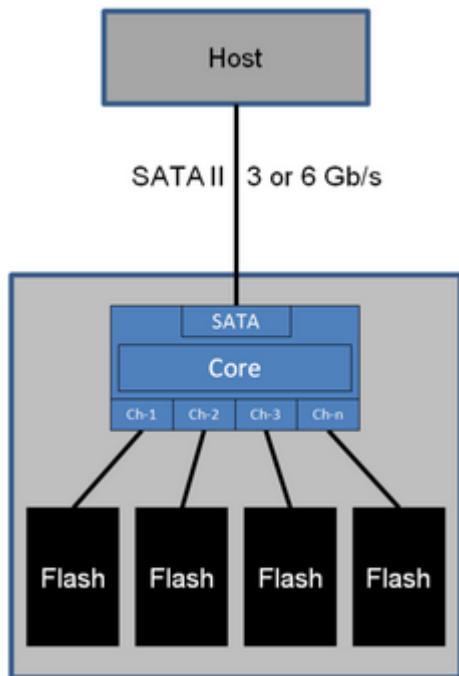
すでにフラッシュセルには書き込み/削除の寿命がある点を指摘しました。コントローラーはドライブが故障するのを避けせるための動作を行います。すべての SSD はウェアレベリングと呼ばれるシンプルなスキームを実装しています。

100MB のファイルを保存したいとします。HDD はディスクの片方から開始し、空きスペースがある場所にシーケンシャルにファイルを書き込みます。ファイルをなるべく連続して書き込むことで、読み込みが速くなります。

SSD コントローラーは可能な限りディスク上のすべてのブロックにファイルを散らばらせるよう試みます。それは以下の 2 つの理由からです：

- パフォーマンス
- 整合性

パフォーマンスの観点からは、故意に書き込みを散らばらせるのは、並行書き込みという明確な機能です。良い品質の SSD は複数のチャンネルを通じてフラッシュメモリにアクセスする複数のレイヤーを持ちます。



SSD コントローラーは同時にこれらのメモリチャンネル書き込む能力を持ちます。先の例の続きで、100MB のファイルはチャンクに分割され、複数のチャンクは並行に書き込まれます。一般により高価な SSD ほど、より多くのフラッシュチャンネルを持ちます。これにより HDD に比べパフォーマンスが大幅に向上します。

注

さらに一般に容量の大きな SSD ほど多くのチャンネルをもらいます。なので SSD のパフォーマンスはサイズに伴って向上することがあります。

整合性の観点からすると、ファイルがシーケンシャルに、あるいはもっと悪いことに連続的に保存されると、ファイルが更新されるたびに同じフラッシュセルに対して消去と書き込みが行われることになります。ドライブ全体に

ファイルデータを分散することで、SDD コントローラーは各セルへの書き込み数を減らそうとします。言い換えれば SSD コントローラーは故意にファイルをフラグメント化させます。これは”ウェアレベリング”と呼ばれます。

ここで、すでに説明したパフォーマンスの低下に関する問題について考えましょう。ウェアレベリングは私たちが SSD のパフォーマンスに関して知るすべてに対して悪く働くように見えます。故意にファイルを SSD 全体に分散することは、いつかすべてのフラッシュブロックが使用済みページで埋まる 것을意味します。これはまったくそのとおりです。ウェアレベリングは直接的に書き込みの増幅を引き起こすか、あるいはコントローラーがシーケンシャルに書き込みを行うのに比べ書き込み増幅問題が表れるまでの時間を速めます。これはまさにパフォーマンスと寿命の間で適切なバランスを取らなければならないということです。

このコンテキストでもうひとつ示しておかなければならぬ重要なことは、すべての SSD (特に MLC SSD) には示されているよりも多くのフラッシュメモリが積まれているということです。コントローラーはこの追加のスペースを使用して追加的なウェアレベリングを行い、ドライブの寿命を延ばし、書き込み増幅の発現を遅らせようとします。

パフォーマンス機能

ドライブコントローラーの重要なパフォーマンス機能は以下の通りです。

- 並列化
- ガーベージコレクション
- TRIM
- キューディシティの最適化

初期 (2008 年ころ) のメインストリーム SSD ではこれらは確かに大きく有効な関心事でした。ほとんどすべての SSD に書き込み増幅の問題があり、パフォーマンス低下の問題を修復する唯一の方法はドライブを再フォーマットすることでした (クイックフォーマットでは使用済みページがそのまま残ってしまうため、有効ではありません)。実際 JMicron 製のコントローラーを搭載するすべての世代の SSD にはこの問題に対する解決策がありません。JMicron ベースの SSD を購入した場合、購入時と比較して 25-35% パフォーマンスが落ち、ドライブのフォーマット以外に解決法はありません。

いくつかの SSD 製造者は SSD をクリーンにするためのツールをリリースしました。これらのツールは SSD に対し削除されたページを通知するものですが、手動で実行しなければなりません。

他の SSD 製造者は独自のガーベージコレクションを実装しました。SSD のガーベージコレクションで、コントローラーは SSD をスキヤンし、使用済みページを識別します。そして使用済みページの塊を移動し、ひとつのブロックを埋め、他のブロックを解放するようにします。想像できる通り、この処理の間ドライブを使用することはできなくなります。ガーベージコレクションはドライブがアイドルのときにしか行えません。

新しい解決策は ATA コマンドに TRIM と呼ばれるコマンドが追加された結果実装されました。OS が TRIM をサポートしていれば、OS は SSD から削除されたファイルを追跡し、ドライブコントローラーに通知します。ドライブコントローラーはどのページが使用されていないか知ることができ、ファイルの保存やウェアレベリングの処理をより賢く行うことができます。ただし TRIM について留意すべき点もあります:

- 4D OS サポート
このドキュメント記述時点では Windows 7 と Windows Server 2008 のみがネイティブに TRIM をサポートしています。ネイティブ TRIM サポートを期待するのであれば Windows を使用しなければなりません。
 - Mac OS X Lion は TRIM サポートを追加していますが、マシンに内蔵された SSD のみが対象です。
- ドライブサポート
ほとんどの SSD (多くが第二世代になっています) は TRIM をサポートしていますが、とにかく SSD を選択する際に TRIM サポートを確認することが重要です。SSD は進化しているので、TRIM サポートはすべての主要な OS やドライブに統合されるでしょうが、それでも気にすべき重要な機能です。
- RAID サポート
TRIM をサポートする RAID コントローラーは現時点ではありません。

異なる SSD を比較する際にはキュー深度の最適化が重要となります。キュー深度という用語は単純にドライブへの同時アクセスを意味します。ドライブにより多くのアプリケーションやスレッドが要求を行えば、キュー深度は高くなります。キューという単語がキーです。SATA ドライブへの並行アクセスのようなものはありません。そこでリクエストはキューされます。

キュー深度が増加すると、SSD のパフォーマンスは製品ごとにだいぶ異なるようになります。高キュー深度に最適化されたコントローラーを持つ SSD (Intel 等) は大量の同時リクエストに対しても大変よく動作します。

注: キュー深度のパフォーマンスは異なる SSD を比較するときに重要であることを強調しておきます。貧弱なキュー深度しか持たない SSD であっても、メインストリームの SSD は HDD よりも速いです。

SSD のメインテナンス

フラグメンテーション

フラグメンテーションはどうして SSD に影響しないのでしょうか。

SSD にピクチャーを保存する例をもう一度考えてみましょう。各ブロックにはすでに使用済みページがあります。ウェアレベリングについてお話したので、SSD は内部的にピクチャーをフラグメントすることがお分かりいただけます。フラグメンテーションによりフラッシュセルの寿命が延びるのであります。

フラグメンテーションが SSD のパフォーマンスに影響を与えない理由は極めて簡単で、動く部品がないからです。HDD のようなシークタイムはありません。データを探すためにヘッドが動くこともありません。確かにデータにアクセスするために、SSD でもいくらかの時間はかかりますが、HDD よりけた違いに速いです。SSD の場合、連続したデータとフラグメントしたデータでロードの時間は同じです。

ファイルフラグメントは問題にならないので、SSD 上でデフラグのプログラムを実行しないことが重要です。実際ウェアレベリングのため、SSD のデフラグを行うことは不可能です。SSD のデフラグは不必要的書き込みを行うことで単に寿命を減らすだけの作用しかありません。

圧縮

SSD のデフラグを行ってはならないということとは対照的に、データベースの圧縮は SSD においても重要です。これは 4D がファイルシステムへのアクセスを最適化する方法に関連します。

データファイルから読み出しを行うとき、4D はブロック単位でそれを行います。4D が 1 レコードを必要とするとき、近くの他のレコードも必要とされるかもしれません。より明確に言うならば、シーケンシャル検索のコンテキストを考えてください。

データファイルが圧縮されていれば、連続するレコードは同じブロックに存在する可能性が高くなります。データファイルがフラグメントしていれば、連続するレコードが別のブロックに存在する可能性が高くなります。

相違点はブロックの処理数です。圧縮されたデータファイルを使用すれば、4D がデータファイルにアクセスする際に行わなければならない処理量を減らすことができます。以下は実際の経験に基づく例です。

- 大きなデータベース - 50GB 以上、300 クライアント、100,000 新規レコード/日、定期的な圧縮なし
- MSC 検証: 2 時間
- MSC 圧縮: 5 時間
- 圧縮したデータファイルの検証: 30 分
- 再圧縮: 2 時間

日常的な圧縮が SSD においても推奨されます。圧縮を行う間隔はデータベースの利用状況により異なります。読み込みのみのデータベースであれば、データが変更されないわけですから、圧縮を行う必要はないでしょう。

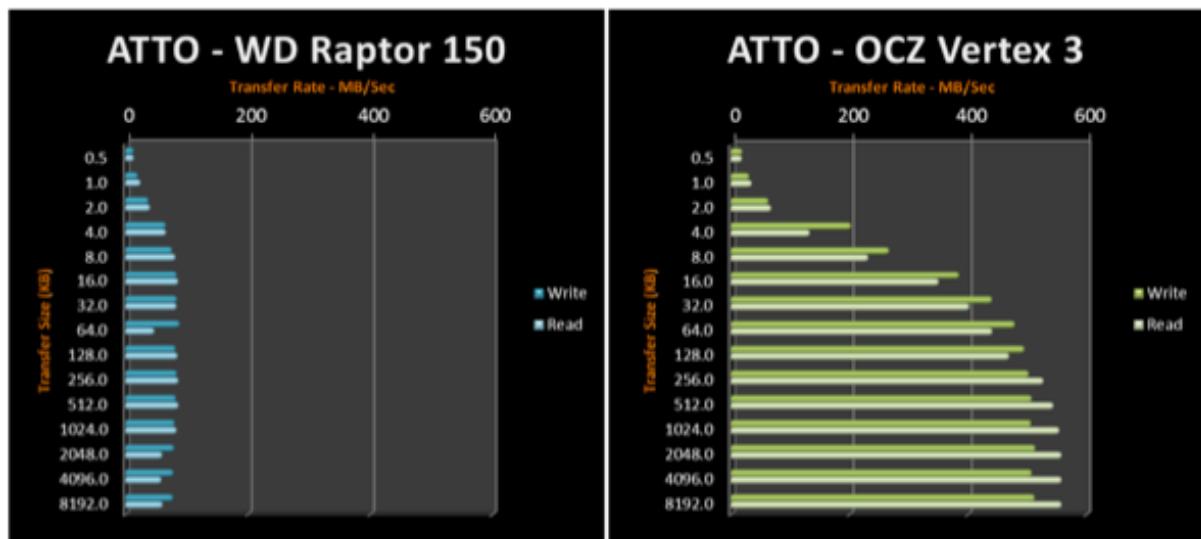
なぜわざわざこの点を述べたのでしょうか。なぜ単に、常に圧縮しろと言わないのでしょうか。すべてのデータベースのメインテナンスは SSD ドライブに対する不必要な書き込みを行わないということとのバランスを取らなければなりません。圧縮はデータベースのフルコピーを作成することです。圧縮を行うべきでないときは、行うべきではありません。

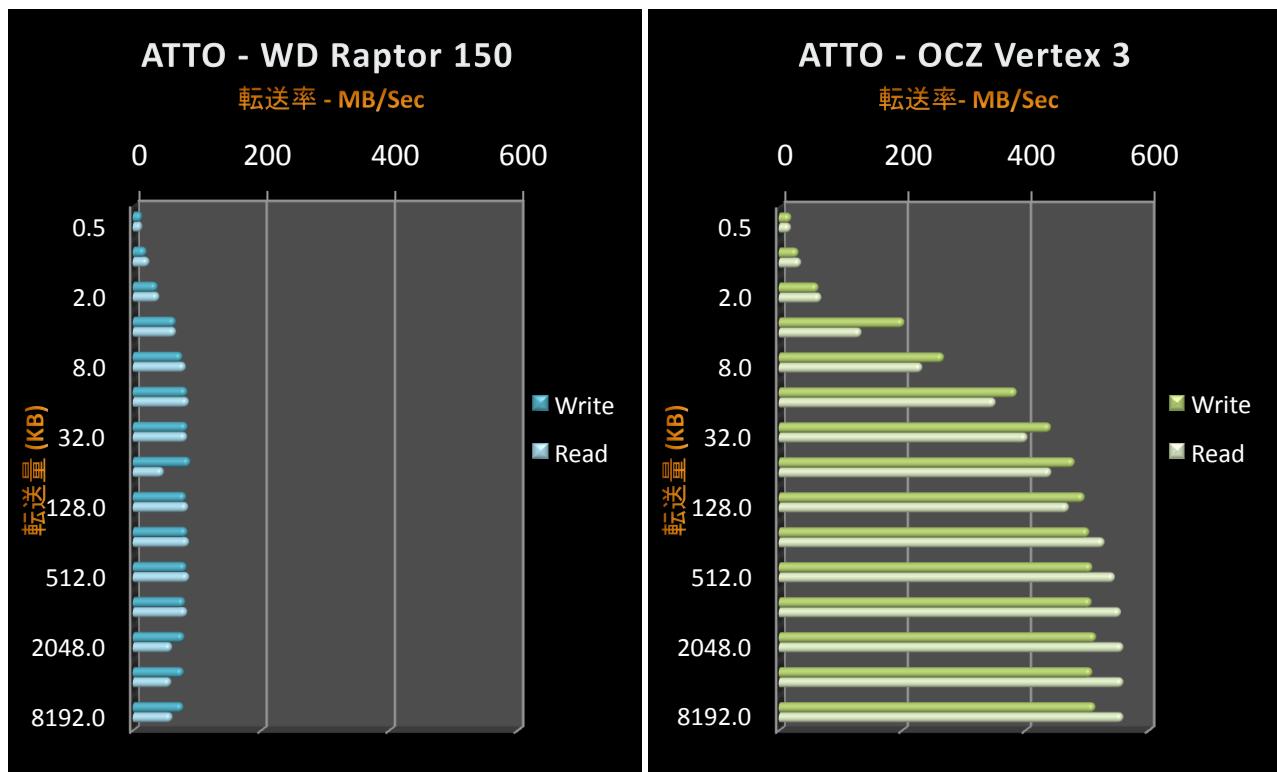
4D v12 では GET TABLE FRAGMENTATION という新しいコマンドが追加されている点に留意してください。このコマンドを使用してデータファイルのフラグメンテーションを計測でき、圧縮が必要かどうかを判断することができます。

SSD でもっとも重要な機能

- TRIM サポート
 - または最低でも独自の方法でドライブをクリーンにする能力。ドライブ全体を消去することなく最適なパフォーマンスに復旧したいとお考えでしょう。
- 平均故障間隔 (MTBF)
 - 文字通りドライブが故障するまでの時間で、時間単位であらわされます。
 - また関連して保証も考慮すべきです。
- SATA 6GB/s 接続
 - この点については後で説明します。

第二世代のメインストリーム SSD ドライブは SATA3 標準をサポートしています (第一世代は SATA2 でした)。SATA2 の最大バンド幅は 3GB/s で、SATA3 は 6GB/s です。HDD の場合これは無意味です。ほとんどの HDD は SATA1 接続 (1.5GB/s) でさえ使い切ることはできません。第二世代の SSD はすでに SATA3 の上限に達しています。以下は ATTO ベンチマークの結果です。





このベンチマークは(512Kから8MBまで) サイズが増加するデータをドライブに読み/書きし、各サイズにおけるバンド幅を推定します。このテストにおけるキュー深度は4です。

HDDはWestern Digital Velociraptorを使用しました。これは10,000RPMの高パフォーマンスHDDです。SSDはOCZ Vertex 3を使用しました。これはSATA3をサポートするメインストリームSSDです。

ここで見られるとおり、HDDは100MB/sも出せません。それに対してSSDは600MB/sに近くなっています。

注

SATA3のパフォーマンスを得るためにマザードライブのコントローラーもSATA3に対応していなければなりません。

4Dにとって最も重要な2つのSSD機能は次の通りです。

- ランダムな読み書き速度
 - ドライブ製造者は通常この値を公表しています。
- 高キュー深度パフォーマンス
 - この点を検証する最善の方法はドライブのベンチマークをリサーチすることです。

ランダムアクセスはすべてのSSDにおける基本的な要求であり、ベンチマークに必ずしも必要というわけではありません。Intelの第一世代のSSDはとても控えめなランダム書き込み速度を表示していました。しかしその時、Intelだけが、実際に利用できるコントローラーパフォーマンスを備えたコンシューマーレベルのドライブを提供していました。具体的にいえばIntelはキュー深度最適化により競争に打ち勝ったのです。Intelの読み書き速度が他のドライブに対し読み書き速度で仕様表記上劣っていても、Intelをテストすれば常に、特にキュー深度が高くなれば、実際のパフォーマンスは勝っていたのです。SSDのベンチマークをレビューする際には、キュー深度がファクターとして組み込まれているかどうかをチェックしてください。

これはIntelドライブを無条件に推奨するものではありません。例えばSandforceコントローラーベースの第二世代SSDは高キュー深度でも高いパフォーマンスで動作し、第二世代のIntel製ドライブを超えることもあります。

このセッションの目的は、読者に特定の SSD を購入させることではなく、どのような SSD を購入すべきか情報を提供することにあります。

SSD を使用する

一般的および 4D を使用する際の、SSD 利用に関する重要な Tip を紹介します。

- SSD のデフラグは絶対に行わないでください。パフォーマンスは一切向上しません。ドライブの寿命を縮めるだけです。
- OS によるインデックス作成を無効にすることを考慮してください。SSD のパフォーマンスはとても高く、ファイルインデックスは必要ないかもしれません。またインデックスの作成は必要にドライブの寿命を縮めます。
- 異なるドライブにバックアップを取りましょう。これは SSD でなくとも必要なことですが、バックアップにより、不必要的書き込みを減らすことができるというメリットがあります。

良い結果を得られる状況

SSD 利用を考慮する場合、ディスク I/O のモニターが重要です。例えば Windows ではディスクアクティビティを知るために、リソースモニターが使用できるでしょう。スループットが~50MB/s 以下であれば、おそらく SSD は HHD よりシーケンシャル処理で速くなることはないでしょう。100MB/s に近くなれば、SSD のメリットが出てきます。これは結局ボトルネックがどこにあるのかということが重要であり、しばしばデータベースデザインレベルで考慮すべきことです。

他方ランダムアクセスについてはスループットを知ることに意味はありません。パフォーマンスの違いを見るためにはデータベースのベンチマークを行ったほうがよいでしょう。

キャッシングのフラッシュは重要であり、決定的に異なる点です。キャッシング中のオブジェクトは通常ディスク上で連続していません。SSD はその点を気にしません。4D 社でテストを行ったところ、キャッシングフラッシュにかかる時間が 15-30 秒から、検知できないまでに短くなったことがあります。キャッシングフラッシュはユーザーアクションにも影響することに留意してください。より短くなればユーザーにもメリットがあります。

SSD を検討する際現実的になることが重要です。SSD は魔法の杖ではありません。ほとんどの場合磁気ディスクより SSD のほうが速いとしても、CPU/キャッシング/RAM 速度には及びません。他のリソースに比べ、ディスクが依然としてボトルネックになる可能性はあります。SSD は確かによりよく動作しますが、それでもディスクアクセスがボトルネックとなる可能性が消えるわけではありません。ですから、ディスクアクセスを減らすという今までのセオリーは、今でも有効です。

OS 設定の最適化

4D にとって特別なことはありませんが、SSD を使用する際の OS 微調整については多くのオンライン資料があります。

一般的に次のように言われています。

- データベースだけでなく、OS も SSD にインストールする。
- OS のインデックスを無効にする。
- OS のキャッシングを無効にする。
- OS のデフラグを無効にする。

Mac OS X: <http://blogs.nullvision.com/?p=357>

Windows: <http://elpamsoft.com/Downloads.aspx?Name=SSD%20Tweaker>

ベンチマーク

すべてのベンチマークは模造的であるという点に留意してください。ここで紹介する数値は実世界とは異なるかもしれません。とはいっても説得力のあるものです。

テスト設定

テストに関する重要な点は以下の通りです。

- すべてのテストは、TRIM サポートがある Windows 7 で行いました。
- SSD は OS をインストールしていません。これは 3 つのドライブをなるべく公平に比較するためです。SSD に OS を置くと、実際はもっと速い結果が得られるでしょう。
- 特に記載しない限りキャッシュは空の状態で行いました。キャッシュにデータが入ると RAM 上のそれが使用され、ディスクは関係なくなってしまうからです。
- ネットワークに関する問題を取り除くため、すべてのコードはサーバー上で実行されました。

マシン仕様

テストマシンの仕様は以下の通りです。

プロセッサー	Intel Core 2 Quad Q6600, 2.40 GHz
RAM	8192 MB
OS	Windows 7 Home Premium (64-bit)
4D バージョン	4D Server v11 SQL

テストしたドライブ

以下のドライブをテストに使用しました。

ドライブ	本ドキュメント中での略称
Western Digital Raptor 150 GB	HDD
Intel X25-M 80 GB	Intel
OCZ Colossus 120 GB	OCZ

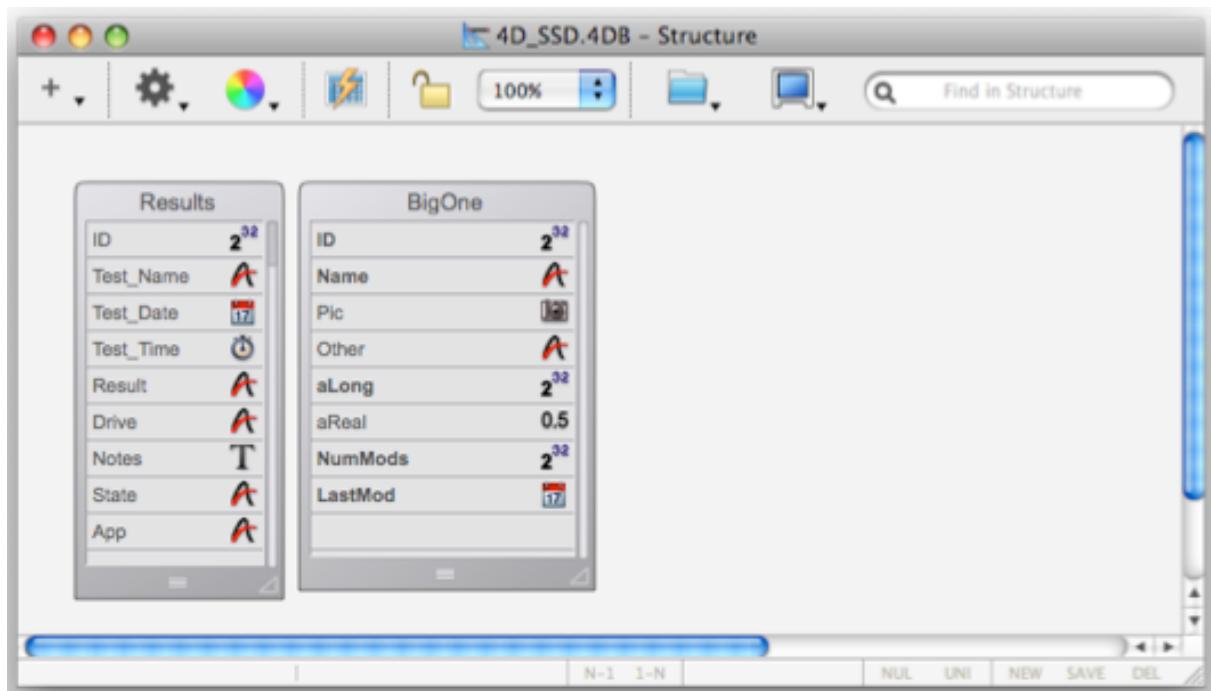
ドライブについての詳細は次の通りです。

- Western Digital Raptor
Raptor は 10,000 RPM のハードドライブで、大きなキャッシュと高シークタイムを持つ高パフォーマンス HDD です。この速いドライブは HDD を使用した場合の最高なメインストリームシナリオを作成するために選択されました。その他多くの HDD は Raptor よりも遅い点に留意してください。
- Intel X25-M
すでに示した通り、Intel SSD ドライブは第一世代のメインストリームレベル SSD の最適な例です。このドライブをテストに選択したのは、利用が容易であるからです。
- OCZ Colossus
この OCZ ドライブはいくつかの興味深い理由で選択しました:
Colossus は TRIM をサポートしません。その代わりユニークな内部 RAID 設定を持ち、2 つの SSD をペアにします。この独自の RAID コントローラーにはガーベージコレクション技術が含まれます。おそらく主な違いは、OCZ のガーベージコレクターは OS の TRIM 機能ではなく、バックグラウンドで独自に実行されているでしょう。
 - OCZ ガーベージコレクターがこの仕事を行うためにはアイドル状態でなければなりません。
Colossus は 3.5 インチフォームファクターで販売されています。これで SSD をインストールするのがとても楽になります。ほとんどの SSD は 2.5 インチまたは 1.8 インチであり、適切なアダプターが必要です。Colossus は標準の 3.5 インチ SATA HDD と簡単に交換できます。

- これは特に最近の Mac Pro や多くのラックサーバーで顕著です。なぜなら SATA 接続がフレキシブルなケーブルでなく、固定的にマウントされているからです。ドライブ（もしくはアダプター）は 3.5 インチフォームファクターに対応していなければなりません。
- Colossus は特にシーケンシャルアクセスにおいて Intel 製のドライブに性能が近接していますが、キュー深度が深くなると低下します。そのため比較のために選択しました。

テストデータベース

テストデータベースには 600 万レコードのテーブルとテスト結果を格納するテーブルで構成されています。ストラクチャーは次の通りです。



[BigOne]テーブルに 600 万レコードが格納されています。このテーブルのデータはピクチャーフィールドを除きランダム化されています。すべてのレコードには同じ 5K のピクチャーが格納されています。データファイルサイズは 32GB でインデックスファイルサイズは 413MB です。

より現実的なシナリオを表現するためにいくつかのフィールドにはインデックスが付けられています。テスト中インデックスが追加されたり削除されたりすることはありません。

テスト 1: 500,000 件レコードの更新

このテストではデータを変更するためにランダムにアクセスを行う状態をシミュレートします。単純なループを行い、ランダムにレコードを検索して更新します。600 万レコードのうち 50 万レコードが更新されます。インデックスが付けられたフィールドも更新されるため、4D はインデックスも更新します。

テストコード

```
C_LONGINT($numberOfMods_l)
C_LONGINT($rangeOfRecords_l)
C_LONGINT($i;$id)
  ` 更新を行うレコード数
$numberOfMods_l:=500000
  ` 使用するレコード ID の範囲
```

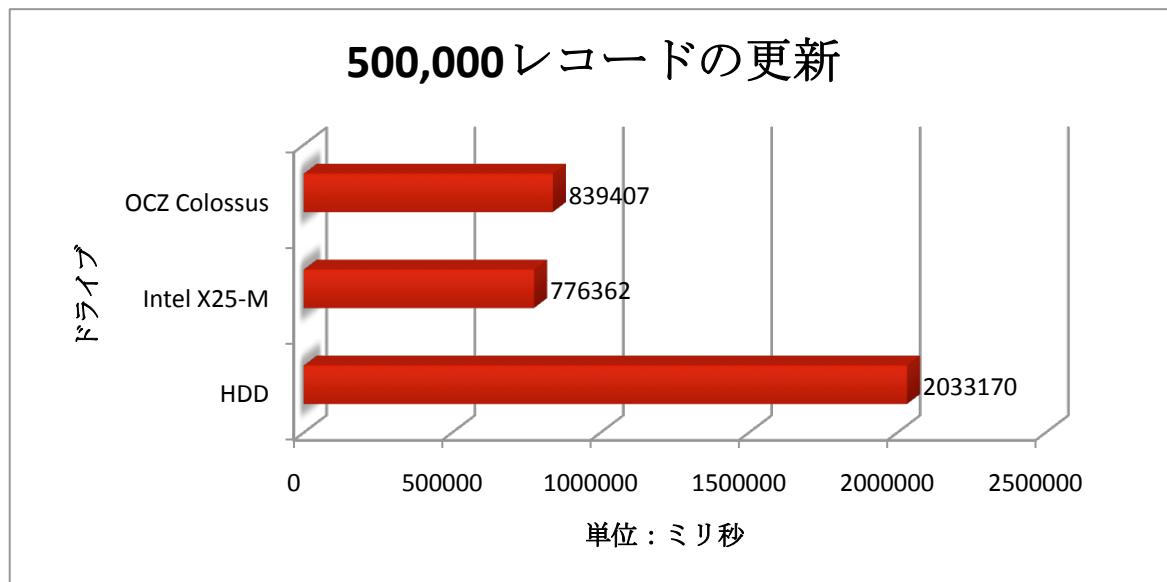
```

$rangeOfRecords_I:=6000000
For ($i;1;$numberOfMods_I)
    $id:=(Random%($rangeOfRecords_I))+1
    QUERY([BigOne];[BigOne]ID=$id)
    [BigOne]Name:="name"+String(Random)+"x"
    [BigOne]aLong:=Random
    [BigOne]aReal:=Random
    [BigOne]NumMods:=[BigOne]NumMods+1
    [BigOne]LastMod:=Current date(*)
    SAVE RECORD([BigOne])
End for

```

結果

最初のテストから、SSD を使用することでパフォーマンスの向上を見ることができました。



このテストではループを使用し、またデータを作成するために、CPUを集中的に使用します。そのため結果はディスク速度だけに結び付けられるものではありません。それでもここで興味深いのは、HDDをテストする際、ディスクがアクセスされる際にCPUアクティビティが上下を繰り返していたことです。SSDをテストしている際にはCPUアクティビティ(4D ランゲージなので1コア)は最高のままです。より速いドライブを使用すると、4DはよりCPUを使用できるようになります。

ここでは Intel SSD が OCZ を 1 分以上上回っていることを見ることができます。

テスト 2: 100,000 ランダムレコードのロード

このテストでは前のテストに比べ CPU をあまり使用せず、できるだけクリーンにランダムアクセスをシミュレートします。結果はディスク速度に完全に左右されます。

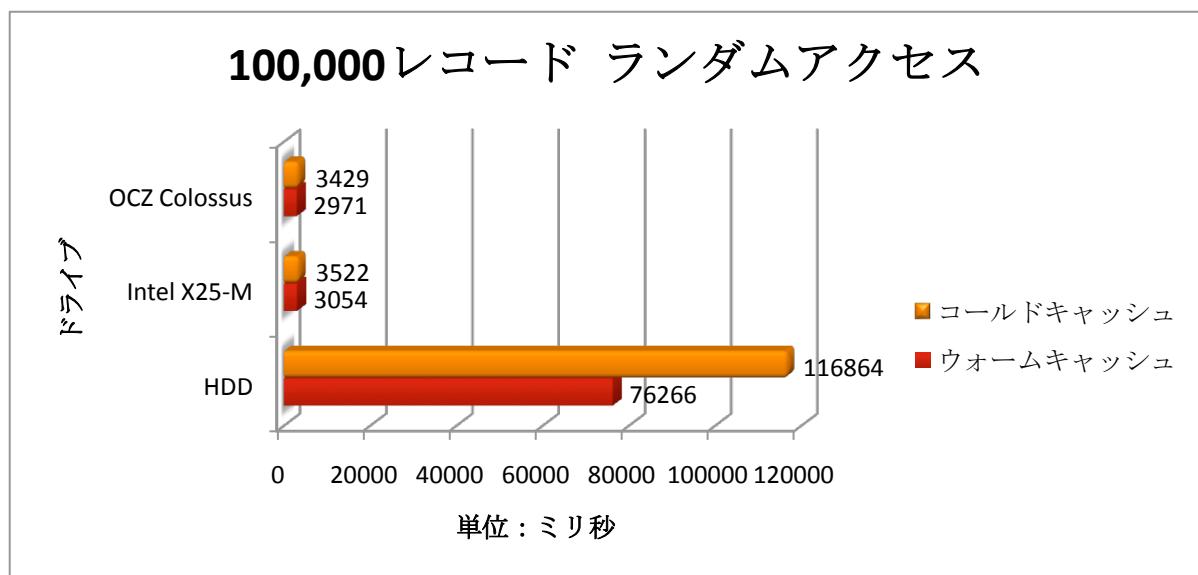
テストでは単純にループを行い、GOTO RECORD でランダムにレコードをロードします。このテストでは 600 万レコード中 10 万レコードをロードしました。

テストコード

```
C_LONGINT($numberOfMods_I)
C_LONGINT ($rangeOfRecords_I)
C_LONGINT ($i;$id)
    ` ロードするレコード数
$numRecsToGoto_I:=100000
    ` 使用するレコード ID の範囲
$rangeOfRecords_I:=6000000
For ($i;1;$numRecsToGoto_I)
    GOTO RECORD([BigOne];(Random*Random)%$rangeOfRecords_I)
End for
```

結果

予想以上の結果に驚くかもしれません、これが結果です。



HDD と SSD では、全くレベルが異なることが分かります。

このテストではまずデータベースを起動し直すことでキャッシングを空にしたうえで一度目のテストを行っています。またより実際の状況を比較するため、データがキャッシングに取られた後、もう一度テストを繰り返し行っています。いずれにしてもレベルの違いは明らかです。

テスト 3: 6,000,000 レコードのシーケンシャルソート

このテストでは[BigOne]テーブルに対するシーケンシャルソートを行います。

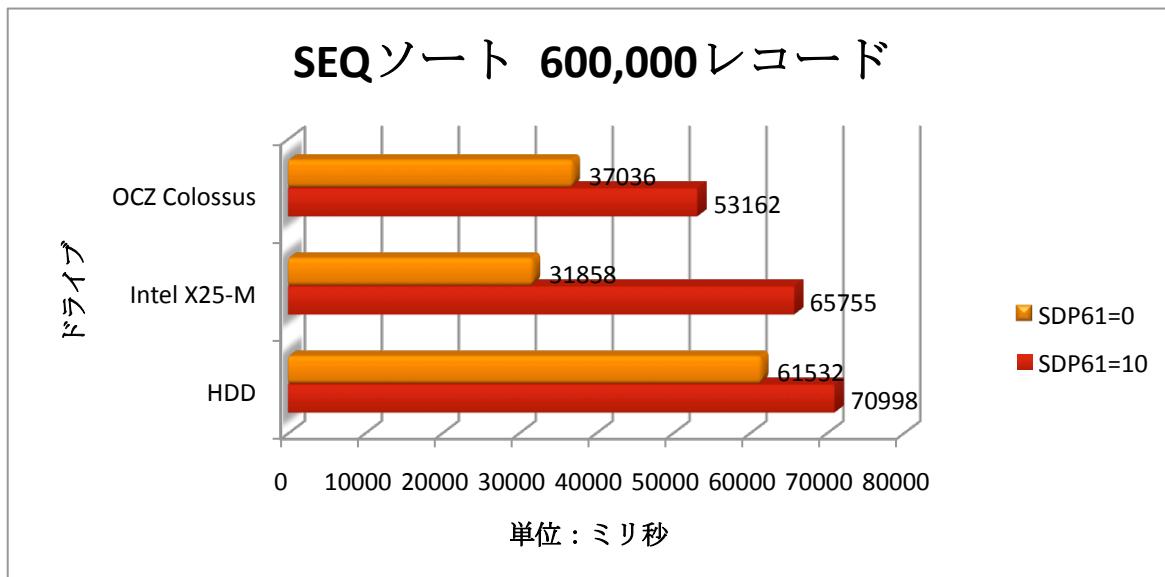
テストコード

```
ORDER BY([BigOne]aReal)
```

aReal フィールドにはインデックスが設定されていません。

結果

これは SSD が万能薬ではないことを示すテストです。



テストは以下の違いを持たせ 2 回行っています:

4D v11 SQL Release 5 の時点では、SET DATABASE PARAMETER (SDP) に、4D がシーケンシャルソートを行う際のキャッシュメモリ量を制限するための新しいセレクター (61) があります (シーケンシャルソートはデフォルトでキャッシュ中で行われます)。デフォルトは 0 で無制限 (キャッシュサイズに左右されます) を意味します。このテストでは無制限と 10MB の値を使用しています。4D はキャッシュが上限に達すると、カレントバッファーをディスク上のファイルにフラッシュします。

デフォルト値では明白に SSD が勝っています。しかし 4D が使用できるメモリの量が制限されると違いはなくなっています。この根本的な原因はシーケンシャルなディスク I/O にあります。このテストで 4D は多くの 10MB ファイルをディスクに書き込み、これが大量のシーケンシャル処理となります。

SATA2 SSD は本質的にシーケンシャルなディスク処理は速くありません。例えば Intel のドライブは相対的にシーケンシャルな書き込みパフォーマンスが貧弱です。第二世代の SSD は並行化と SATA3 のサポートにより、この部分で大きく向上しました。

二番目の問題はよりわずかなものです。先に示した通りほとんどのカレントシステム上のすべての SATA2 ドライブは、シーケンシャルスループットがだいたい 250-280MB/s に迫っています。これは SATA2 接続の上限です。この壁を超えるには SATA3 接続が必要です。

まとめ

4D 社が特定の SSD を公式に推奨するということはありません。新しいハードウェアを選択する際、必要とされるゴールは状況により異なるはずです。新しいハードウェアを網羅的にテストすべきです。

ベストプラクティス

- メモリ
 - 32-bit 4D は 4 GiB 仮想メモリを使用できます。
 - 64-bit 4D Server v12 は 8TB の仮想メモリを使用できます。

- GET CACHE STATISTICS コマンドの使用済み仮想メモリ戻り値を使用して、アプリケーションメモリを計測します。

4D が使用できるメモリ量を最大化するために

 - 64-bit OS を使用します。
 - 32-bit 4D の場合最低 6GB の RAM を積みます。
 - 64-bit 4D Server v12 の場合、サーバーの使用履歴を参照してトータル RAM サイズのバランスを決定します。
- 上と同じルールを適用して 4D が使用できるキャッシュ量を最大化します。
 - ハードウェアとデータベース両方のキャッシュサイズを適切に設定してください。
- キャッシュフラッシュの間隔は秒単位で設定できるようになりました。4D のキャッシュマネージャーはフラッシュする内容が少ないほどより効率的に動作します。データの変更が頻繁に行われるようなデータベースではフラッシュ間隔を短くするとよいでしょう。
- そうする必要がない限り動的キャッシュの計算は使用しないでください。
- SSD
 - *ぜひ使用してください。
 - *重要な機能
 - TRIM サポート
 - 高ランダム I/O パフォーマンス
 - 高キュー深度パフォーマンス
 - SATA3 サポート
 - *利用法
 - SSD をデフラグしてはいけません。
 - ドライブのインデックスを無効にすることを考慮する。
 - 物理的に異なるドライブにバックアップを取る。
 - SSD に OS をインストールする。
 - OS のキャッシュを無効にすることを考慮する。
 - OS 設定を自動で管理するための SSD ツールが存在します。
 - *期待されること
 - 速い HDD を使用してディスク I/O が 50MB/s 以下の場合は、SSD を使用するメリットはさほど得られません。
 - キャッシュフラッシュ間隔は短く設定すべきです。
 - SSD は速いですが、RAM ほどではありません。

4D とデスクトップ仮想化

4D を仮想デスクトップ（シンクライアント）上で使用する際にはいくつかの推奨事項があります。特記しない限り、これらは Microsoft Terminal Service と Citrix 両方に当てはまります。

RDPCLIP

4D Server と Microsoft Remote Desktop とのクリップボード共有には互換性の問題があります。これを解決するには 2 つの方法があります:

- 4D Server が応答しなくなったら rdpclip.exe プロセスをアボートする（この状況ではこのプロセスが 99% の CPU タイムを消費します）。
- Terminal Services Configuration でクリップボードマッピング機能を恒久的に無効にする。
 - リモートデスクトップ接続設定からも無効にできます。

プリンタードライバー

プリンターが何もインストールされていない場合、デフォルトで 4D はシステムプリンタードライバーを使用します。このドライバーが更新されていない場合や 4D の実行ユーザーと同じユーザーでインストールされていない場合、問題の原因となります。これらの問題を避けるために、ダミーのローカルプリンター（例えば HP 汎用プリン

タードライバー) をインストールし、それをデフォルトプリンタードライバーに設定することを推奨します。これはもちろんローカルプリンターがインストールされていない場合です。

クライアントロード

シンクライアントサーバー毎に、15-20 クライアントを超えないことを推奨します。

避けるべきこと

デザインモード

- オブジェクトフォントプロパティでフォントやフォントサイズのポップアップを表示する。
- プロパティリストで変数タイプポップアップを表示する。
- フォーム上にオブジェクトを描画したり移動したりする。

アプリケーションモード

- 4D Chart で四角オブジェクトをリサイズする。
- TSE/Citrix 環境で SET TIMER の利用は推奨しません。

データベース設定

CPU 優先度はすべてデフォルトの真ん中のままにしてください。SET DATABASE PARAMETER コマンドによりこの値を間違って設定することのないよう注意してください。

問題解決の考え方

「時折サーバーが遅くなる」、あるいは「サーバーが期待通りに動作しなくなることがある」といったような問題はもつとも対応が難しい問題です。特に大人数がアクセスするようなサーバーで、顧客に「問題の発生時に何を行ったか」と尋ねることができない場合はなおさらです。

ログの取り方やその解析方法を学ぶことで、これらの問題の多くを見つけ出し、理解することができます。さらに良いことには、通常時からサーバーの振る舞いを監視するツールを使用して、発生する可能性のある問題を素早く検知したり解決したりできます。

4D Server バージョン 11 以降様々なログファイルを作成することが可能であり、オフサイトでそれらを解析することで、サーバーの振る舞いや問題を理解することを助けてくれます。このパートではこの目的のために使用する 4 つのツールについて説明します。

4D CACHE LOG ANALYZER

以前のバージョンの 4D では、キャッシュは速度向上のためにのみ使用されていました（ハードディスクへのアクセスを減らせばオブジェクトへのアクセス速度を高められます）。4D v11 SQL 以降ではカレントセレクションやトランザクション、セット、結合や並び替えなどに称される内部オブジェクトなどもキャッシュを使用します。4D v11 SQL ではキャッシュマネージャーが完全に書き直されています。

キャッシュサイズとオブジェクトを格納するための空スペースは、アプリケーションの速度に大きな影響を与えます。極度にメモリが足りない状況ではサーバーがハンギング（フリーズ）したりクラッシュすることがあるでしょう。

4D v11 SQL の拡張性により、1,000 プロセスが作成されるようなサーバーでも 4D を使用することができるようになりました。キャッシュマネージャーに必要とされるのは低メモリの状況下で未使用的オブジェクトを素早く開放することと、同時に 64-bit モードで 20GB 以上のキャッシュサイズを処理できるようにすることです。キャッシュマネージャーは 4D v11 SQL Release 8 および 4D v12.1 で劇的に向上しました。ここでお話しする内容はすべてこの新しいキャッシュマネージャーが対象です。

理想はサーバー上で利用可能なメモリの上限がなく（現実的には必要十分なメモリがあり）、キャッシュログの検証やキャッシュ中で何が行われているか知る必要がないことです。実際には 2 つの状況があります：

- ・極端にメモリの少ない既存のサーバー。何が起きているか理解し、設定を最適化する。そして必要であればメモリを付け足す必要があるか（それで問題が解決されるか）検証を行う。
- ・新しいサーバー設定のための最適値を求める。ユーザー数などの環境に応じて特定のアプリケーションの必要条件を理解しなければなりません。既存の顧客のプロファイルを作成し、ユーザーが増えるに従い、どれだけメモリの必要量が増していくか明確な数字を得られるようにします。

サーバーに十分なメモリが搭載され、すべてがうまくいくという 3 番目の状況もあります。このケースについてはお話しする必要はないでしょう。

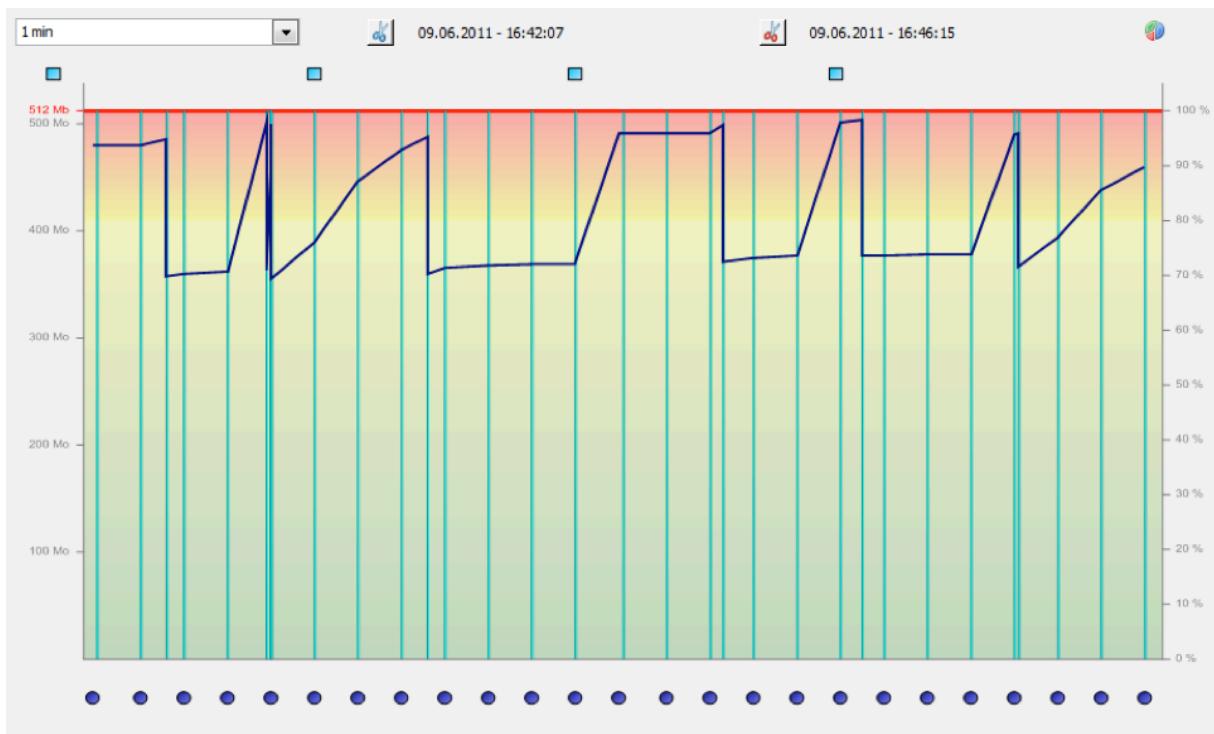
キャッシュログと分析

4D には GET CACHE STATISTICS というコマンドがあり、キャッシュの内容に関する情報を返します。このコマンドを定期的に実行・ログに記録し、分析することができます。ここではログの記録を行う 4D Cache Log Recorder コンポーネントと、記録したログの読み込みとグラフィカルな表示を行う 4D Cache Log Analyzer を紹介します。

ここで使用するサンプルはキャッシュサイズ 512MB に設定された 4D Server v12.2 で動作し、10 ユーザー /210 プロセスが実行され、1 日の動作が記録されています：

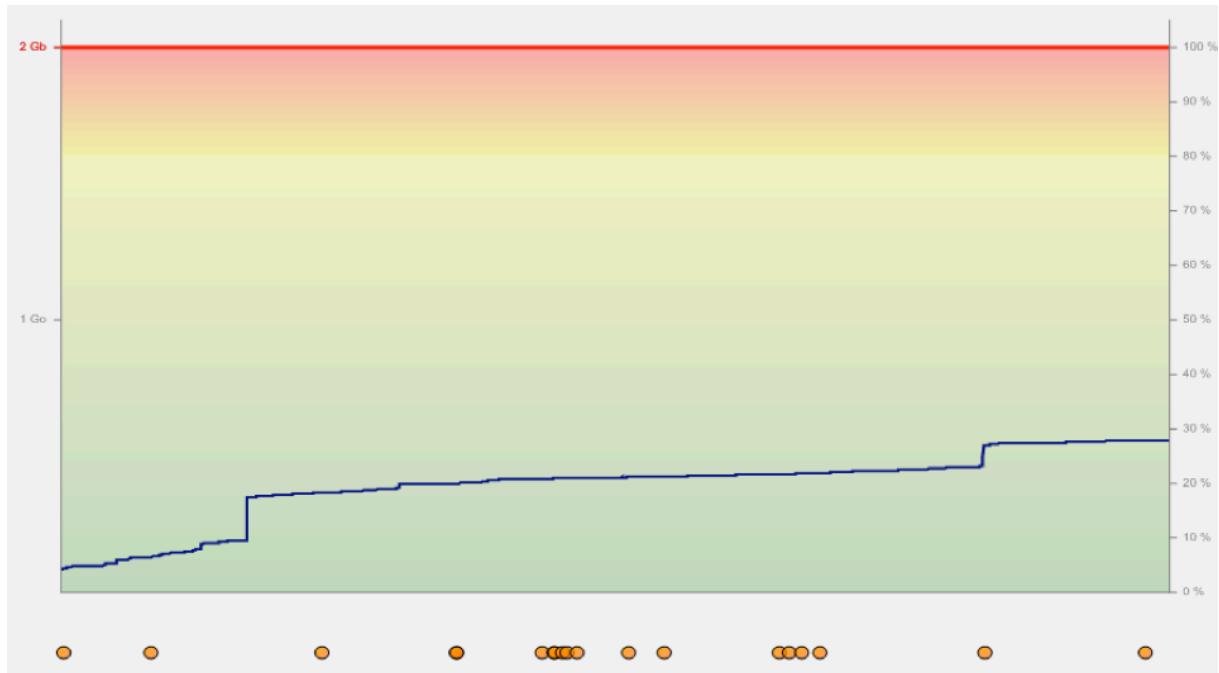


この図をみると、定期的にキャッシュが必要となり、約 100MB がページされ、リロードが行われていることが分かります。画面をズームすることができますので、ズームアップして見た所が次の図になります。



下部に現れた青い点は 4D が処理した定期的なフラッシュバッファのタイミングです。このデータベースは、4D v12 のデフォルト値である 20 秒毎に設定されています。縦線はコマンドの実行、あるいは必要とされたために行われた追加のフラッシュです。この表を見ると分ごとに数回のページやデータのリロードがしばしば必要とされていることが分かります。そのためサーバーが遅くなっていると考えられます。

以下の図は同じ顧客、同じアプリケーション、2GB のキャッシュを割り当てた 64-bit 4D Server を 2 日間動作させたときのログです。



この状況であれば、サーバーの RAM を少なくすることも可能ですし、同じメモリであるならより多くのユーザーを処理することができます。

4D CACHE LOG RECORDER

I. 概要

4D Cache Log Recorder コンポーネントは 4D データベース利用中の使用キャッシュメモリに関する情報を記録するためのツールです。情報はファイルに記録され、4D Cache Log Analyzer を使用して解析できます。

II. インストール

他の 4D コンポーネントと同様、4DCacheLogRecorder.4dbase という名前のパッケージ (Mac OS X) あるいはフォルダー (Windows) をデータベースストラクチャーと同階層に作成する Components フォルダーにコピーします。このコンポーネントを動作させるために必要な最低バージョンは 4D v11 SQL Release 5 (11.5) です。v12 用のコンポーネントは 4D v12 および v13 で使用できます。

III. インターフェースを使用する

コンポーネントをインストールしデータベースを開いたら、CLR_Interface メソッドを呼び出してコンポーネントインターフェースを表示できます。以下のダイアログが表示されます:

IV. オプションの設定

次の図はデフォルト設定を表しています。各オプションを設定できます。



1. Recording every

ここではログを取る間隔を分単位で設定します。複数日にわたって情報を記録したい場合、この値を増やすことができます。デフォルト設定は1分毎です。1分毎に一週間記録したログを読み込む場合、最近のコンピューターでも4-5時間かかります。間隔を5分や10分に設定すれば読み込み時間を短くできます。しかし情報の粒度は低くなるので、たとえ読み込みに時間がかかったとしても、短い間隔でより多くのログを取るべきです。

2. Flush cache

このポップアップメニューでは3つのオプションから選択できます。

- None (デフォルト): 通常の方法でデータベースはキャッシュを記録します。
- Simple: "Recording every"オプションに基づきキャッシュを記録します。
- With purge: "Simple"オプションと同じ処理を行いますが、キャッシュがディスクにフラッシュされるとき、キャッシュの内容全体をページします。4Dのテクニカルサポートからこのオプションを選択するよう指示された場合のみ選択してください。このオプションを選択するとデータベースの動作が遅くなります。

3. Memory info

4Dのテクニカルサポートから指示されない限り、このチェックボックスは常に選択されているべきです。

4. Cache statistics

4Dのテクニカルサポートから指示されない限り、このチェックボックスは常に選択されているべきです。

5. C++ Objects

このオプションを選択するとパフォーマンスが低下します。4D のテクニカルサポートから指示された場合のみこのオプションを選択してください。

6. Other objects

このオプションを選択するとパフォーマンスが低下します。4D のテクニカルサポートから指示された場合のみこのオプションを選択してください。

7. Blocks

このオプションを選択するとパフォーマンスが低下します。4D のテクニカルサポートから指示された場合のみこのオプションを選択してください。

8. Internal watching

このオプションは 4D v11.6 (および次のバージョン) で使用され、キャッシング監視メカニズムを有効にして、CacheLog_timestamp_n.txt のようなログファイルを生成します。
このオプションは現在のバージョンでも使用すべきです。“Statistics Overview”が作成され、より速く表示されます。

9. Memory watching

このオプションは N ステップ毎のメモリ監視メカニズムを有効にし、CacheMem_time ログファイルを生成します。
4D のテクニカルサポートから指示された場合のみこのオプションを選択してください。

V. キャッシュのクリア

CLR_Launch_PeriodicalClearCache メソッドを使用して定期的にキャッシングをページできます (すべてのオブジェクトがキャッシングから削除されます)。このメソッドを実行するとダイアログが表示され、分ごとに間隔を指定できます。

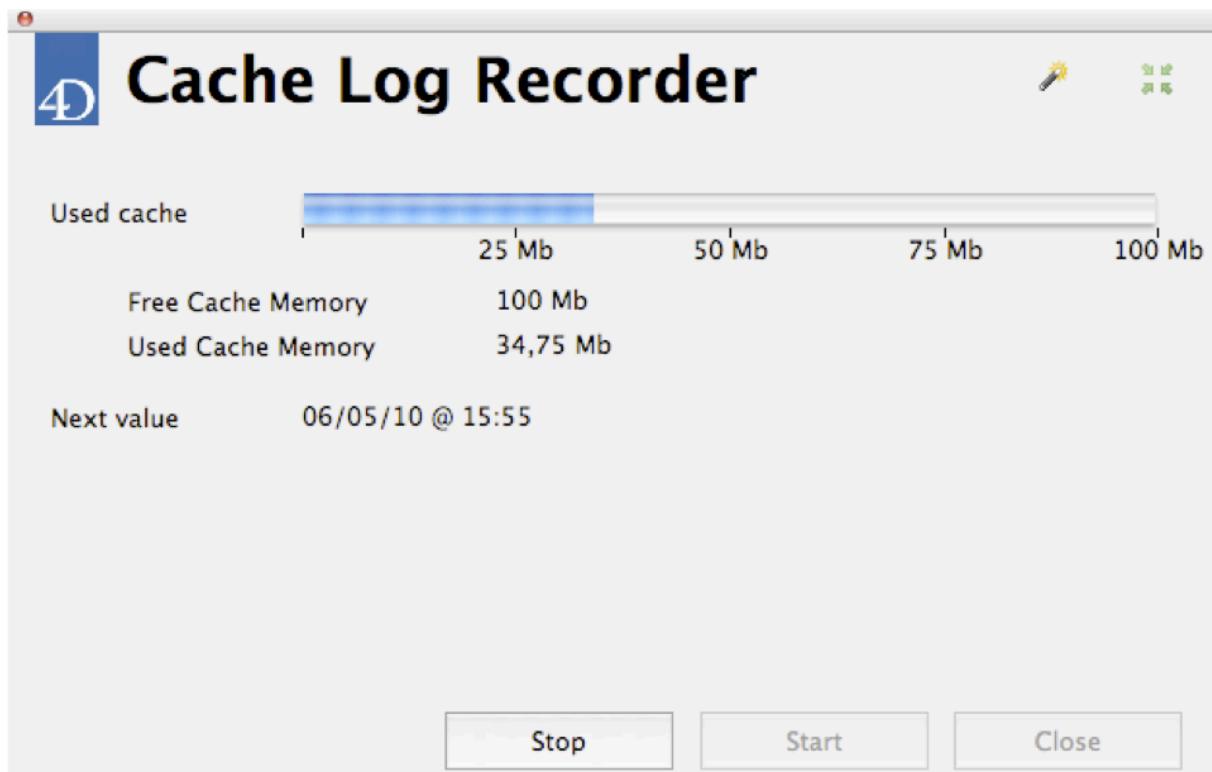
例えば 1440 分を指定すればキャッシングは 24 時間ごとにクリアされます。

このプロセスを終了するには CLR_Stop_PeriodicalClearCache メソッドを実行します。

ページを行った後、キャッシングにオブジェクトが格納されるまでの間速度が遅くなる点に留意してください。

VI. 情報の記録を開始する

オプションを設定したら“Start”ボタンをクリックして情報の記録を開始できます。ダイアログは次のように変化します。



1. Used cache

サーモメーターには利用可能なキャッシュサイズに対して実際に使用されているキャッシュのサイズが表示されます。メモリに関する数値がサーモメーターの下”Free Cache Memory”と”Used Cache Memory”に表示されます。

2. Next value

次に値が更新される時間を表します。

3. Stop

このボタンをクリックすると記録が停止されます。

4. 魔法の杖アイコン

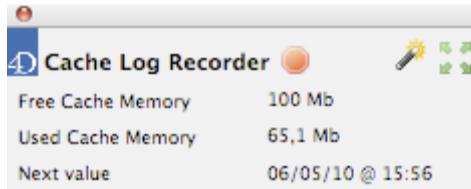


このアイコンをクリックすると手動でキャッシュをクリーンにできます。結果キャッシュは空になります。ページを行った後、キャッシュにオブジェクトが格納されるまでの間速度が遅くなる点に留意してください。

5. ウィンドウリサイズアイコン



このアイコンをクリックするとウィンドウを小さくできます。



VII. クライアント/サーバー

クライアント/サーバー環境ではインターフェースを 4D クライアント上で実行しなければなりません。しかし情報の記録中、CLR_LOGGER という名前のプロセスはサーバー上で実行され、必要な処理を行います。

VIII. インターフェースを表示しない

ログの記録をインターフェースなしで行うには以下のコードを使用します:

```
<>clr_logger:= Execute on server  
("CLR_LOGGER";128*1024;"CLR_LOGGER";1;Record_Frequence;  
Flush_cache;info_type1;info_type2;info_type3;info_type4;info_type5;b_Surveillance  
;b_Surveillance2  
;Record_Every;Record_For;*)
```

引数は以下の通りです。

Record_Frequence	デフォルトで 1 (分単位)
Flush_cache	1 : フラッシュしない(デフォルト) 2 : 記録時に Flush Buffer の実行 3 : 記録時に Flush Buffer(*)の実行
info_type1	0 1 : Flush Cache 下のチェックボックスの設定
b_Surveillance	0 1 : SET DATABASE PARAMETER(65)を有効にする
b_Surveillance2	0 1 : メモリリークチェックを有効にする
Record_Every	メモリリークチェック間隔(回数毎) (デフォルト=5)
Record_For	メモリリークチェック間隔(回数毎)

推奨値

```
$pr:= Execute on server  
("CLR_LOGGER";128*1024;"CLR_LOGGER";1;1;1;1;0;0;0;1;0;5;1;*)
```

サーバー上で実行されるインターフェースなしのログを停止する方法:

```
<>clr_logger_stop := True  
CLR_Reactivate
```

クライアントからは

```
<>clr_logger_stop := True
```

```
VARIABLE TO VARIABLE(-1;<>clr_logger_stop;<>clr_logger_stop)  
$pr:=Execute on Server("CLR_Reactivate";32*1024;"CLR_Reactivate")
```

IX. デバッグのためのログ

CLR_DebugLog_Interface

このコンポーネントメソッドは、クライアント側にダイアログを表示します。このダialogから、デバッグログセレクターを設定し、デバッグログを開始/停止できます。

CLR_DebugLog_Interface_Server

このコンポーネントメソッドは、サーバー側にダイアログを表示します。このダialogから、デバッグログセレクターを設定し、デバッグログを開始/停止できます。

LOG EVENT(4;"mylogmessage")

4Dのコマンドを使用して直接キャッシュログにコメントを書き込みます。これによりコードのどの部分が現在実行されているのか分かりやすくなります。

X. 生成されたデータの送信

4D テクニカルサポートや自分の環境にデータを送信するためには、データベースストラクチャーと同階層にある"Logs"フォルダー内に生成されたデータを取り出します。このフォルダーの中には CacheLog フォルダーがあり、そこに必要な情報が格納されています。

常にフォルダー全体を圧縮します。ファイルを直接送信するとメールアプリケーションにより行末が変更されて、壊れてしまうことがあるからです。CacheLog フォルダー内には、すでに圧縮された Cache_Log_XXX.4DLog ファイルがあります。

XI. コンポーネントのアンインストール

データベースからコンポーネントを取り除くには、Components フォルダーからコンポーネントを取り除きます。CLR_XXX コンポーネントメソッド呼び出しをアプリケーションから取り除くことを忘れないでください。そうしないとコンパイル時にエラーが発生することになります。

4D CACHE LOG ANALYZER

4D Cache Log Analyzer は v12 アプリケーションとして提供され、4D Cache Log Recorder バージョン v11, v12 そして v13 のログを読み込んで解析できます。各顧客ごとまた各データファイルごとにこのアプリケーションのインスタンスを使用することをお勧めします。

File メニューから Import を選択します。フォルダー内のすべての.txt 拡張子ファイルまたは.4DLog 拡張子のファイルを選択します。解析には両拡張子のファイルが必要です。

4D は各記録ジョブ毎に、タイムスタンプ+"_1"を名称にした.txt ファイルを作成します。このファイルサイズが 10MB になると追加のファイル"XXX_2.txt"を作成します。

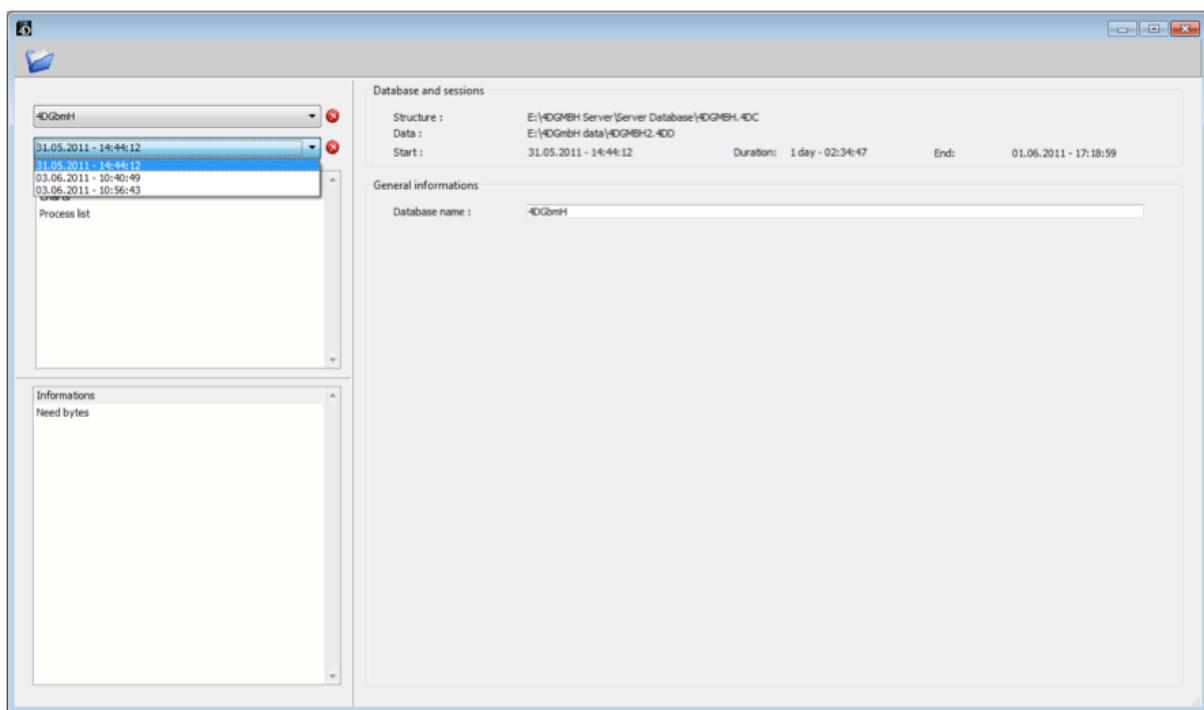
これらのファイルは"Statistics Overview"メニューで必要とされます。すべてのファイルが必要である点に留意してください。ファイルをスキップしたり、最後のファイルだけを選択することはできません。

コンポーネントはさらに"Cache_Log_xx.4DLog"ファイルも作成します。ログの記録を数日間行うと数千のファイルが出来上がります。これを読み込むには数時間かかります。これらのファイルは"Statistics Overview"メニューでは必要とされませんが、"Statistics Detail"で必要となります。これらのファイルの読み込みには時間がかかるため、素早い概要の把握が必要であれば、.txt ファイルだけを読み込んで Overview ダイアログにフォーカスすることができます。

読み込みコマンドはまずプロジェクト名を指定するよう求めてきます。この機能を使用して複数の読み込みをグループ化できます。これは異なる顧客やデータベースをひとつのデータファイルに読み込む際に便利です。

STATISTICS OVERVIEW

このダイアログではキャッシュの状況に関する概要をつかむことができます。



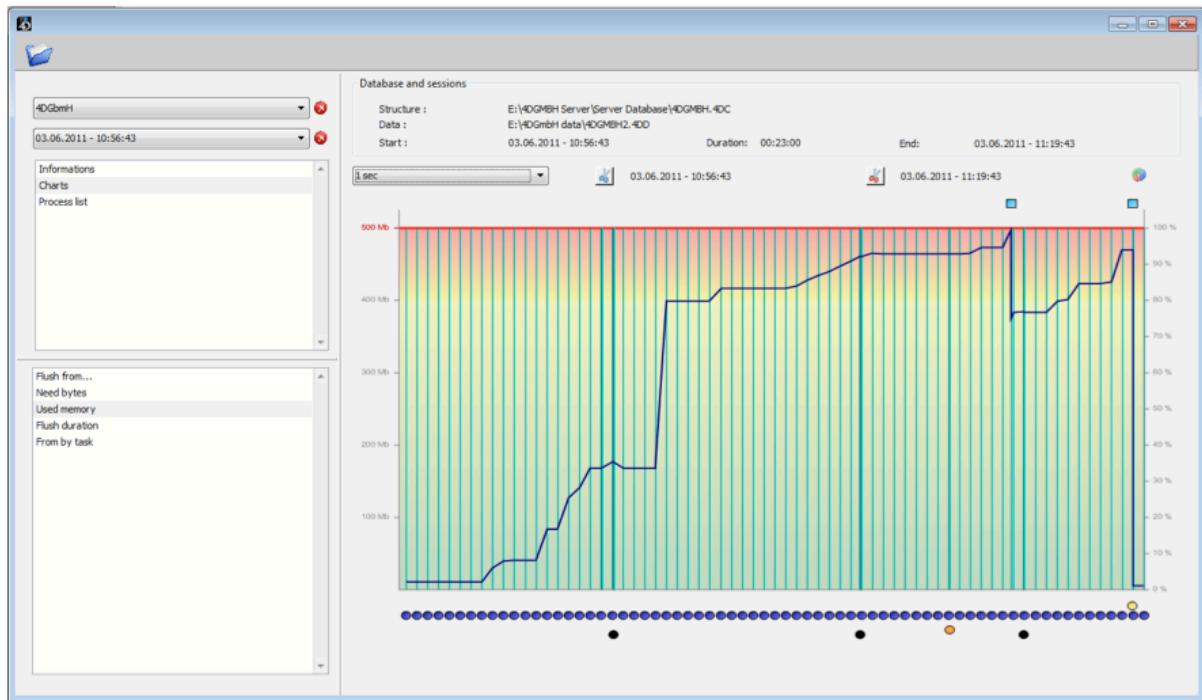
左上のポップアップで解析するロググループを選択できます。このグループは読み込み時にあなたが指定した名前です。顧客ごとなど複数のデータファイルを作成することが可能である点に留意してください。レコードの量を減らすことができれば解析の速度も向上します。

その下のポップアップではひとつのログファイルあるいは同じグループの複数の読み込みいずれかのタイムフレームを表示します。

その下のリストボックスには統計グループが表示されます。さらにその下のリストボックスからは異なるビューを選択できます。

USED MEMORY

"Chart"および"Used Memory"を選択することから開始し、ログの最初の印象を得ましょう。

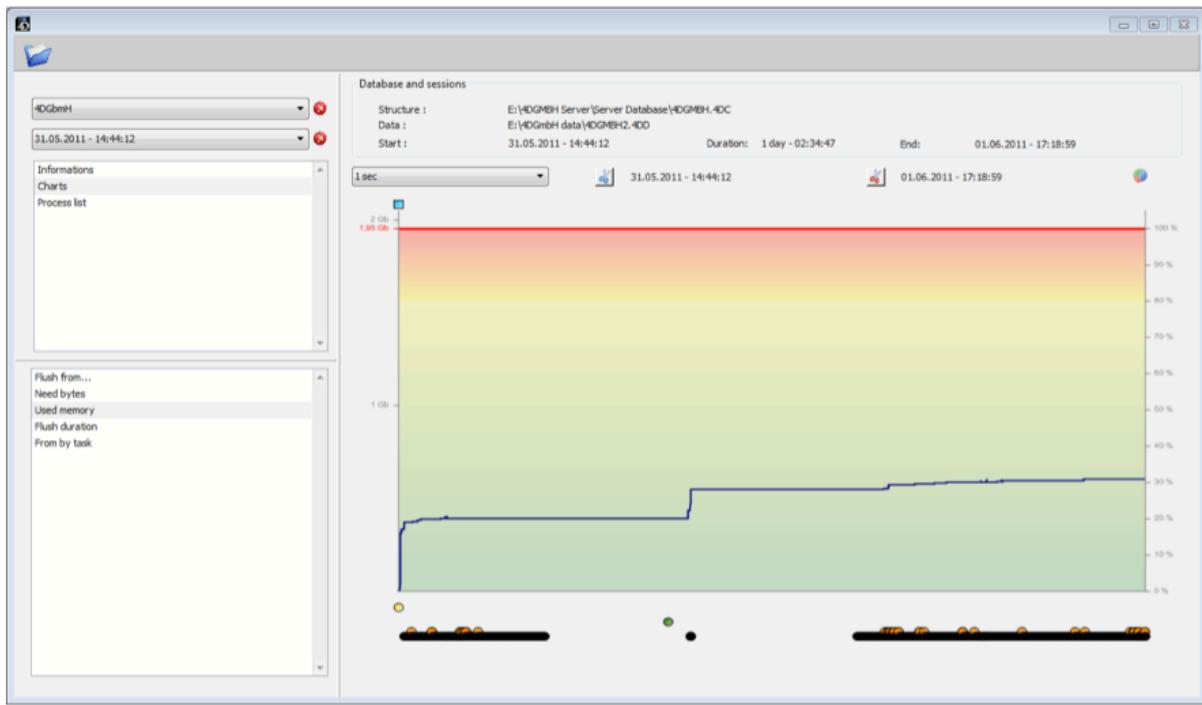


この例では 23 分という短い時間を表示しています。情報の一番上には開始時刻と終了時刻、そして経過時間が表示されています。

その下の図にはキャッシュの利用状況が表示されています。このサンプルでの総キャッシュサイズは 500MB で、すべて使い切られたところ（グラフが頂点に達したところ）に、青い四角が表示されます。この四角はサーバー上で利用可能なメモリよりも多くのキャッシュが必要になったことを示し、結果キャッシュの一部が解放されます。二番目の四角は偽のすべてのキャッシュメモリクリアリクエストで、FLUSH BUFFERS(*)によって引き起こされました。

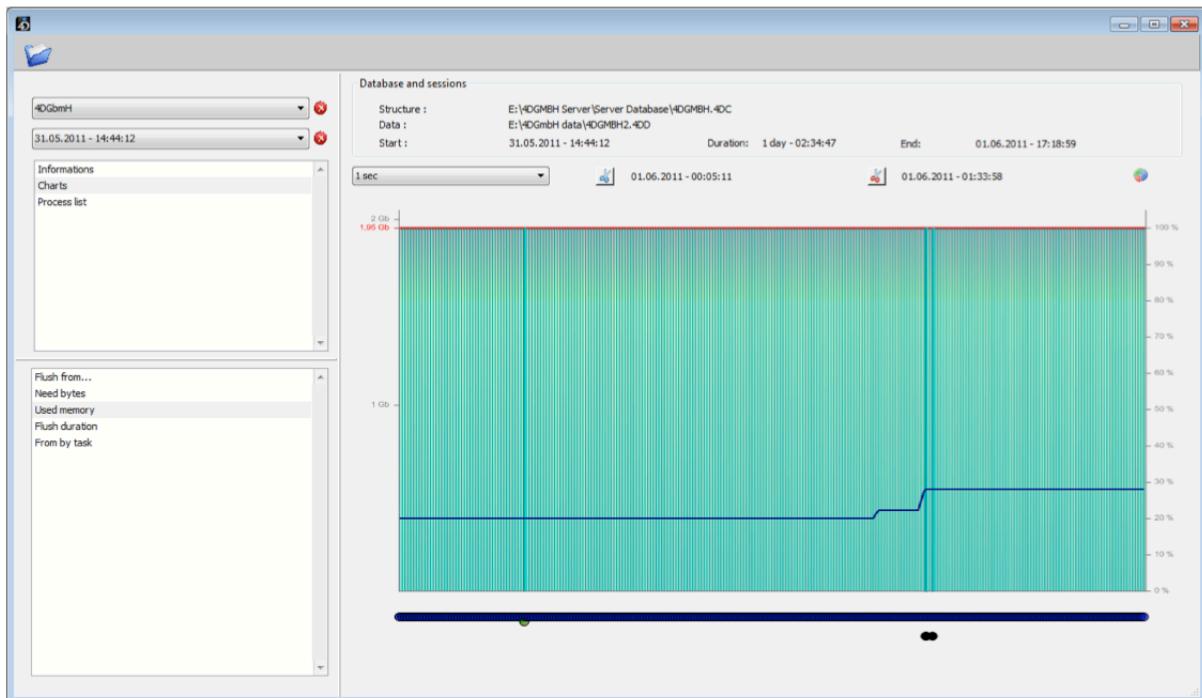
図中の縦線はスケジューラー、4D Backup、あるいはプログラム言語により、いつ FLUSH BUFFER が行われたかを示します。下部の青丸はスケジューラーによる定期的なフラッシュを示します（ここでは 20 秒ごと）。黒丸はサーバー上の、オレンジはクライアントプロセスからの FLUSH BUFFER 実行を示します。黄色は FLUSH BUFFER(*)です。

縦線および丸記号は与えられたタイムフレーム内であまりにも頻繁に行われた場合隠されます。サンプルを一日以上動作させた場合、以下の通り縦線と青丸が表示されないことがあります。

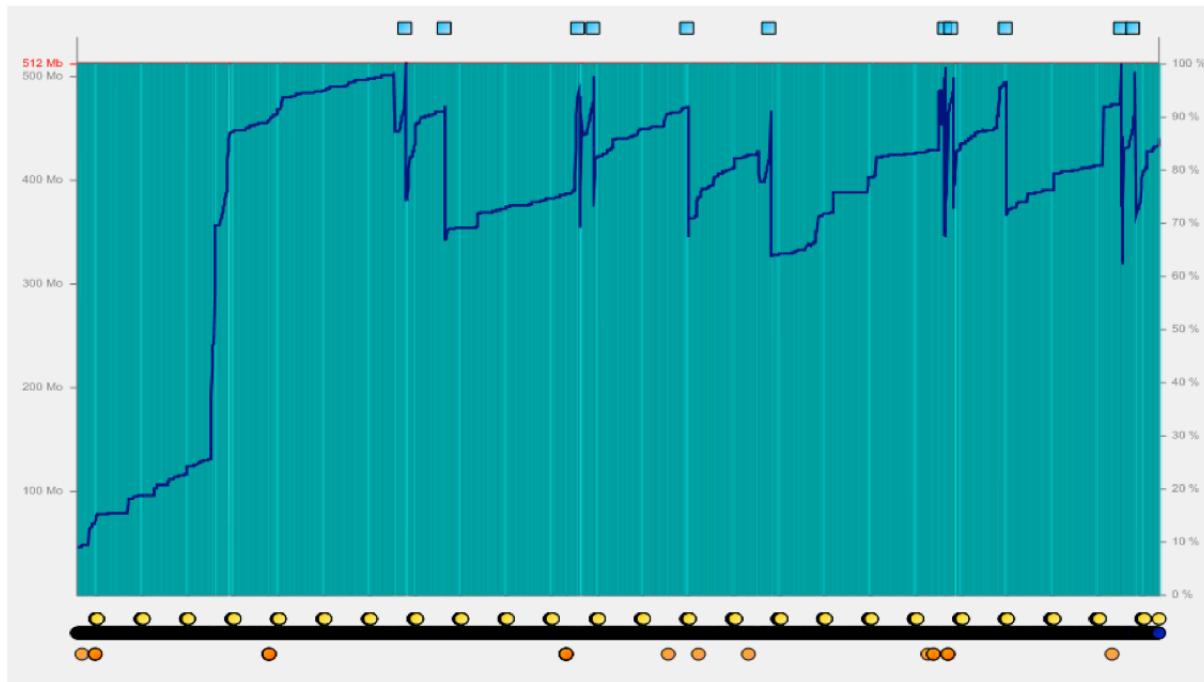


黒い四角はサーバープロセスが毎分(夜以外)呼び出した FLUSH BUFFER を示しています。

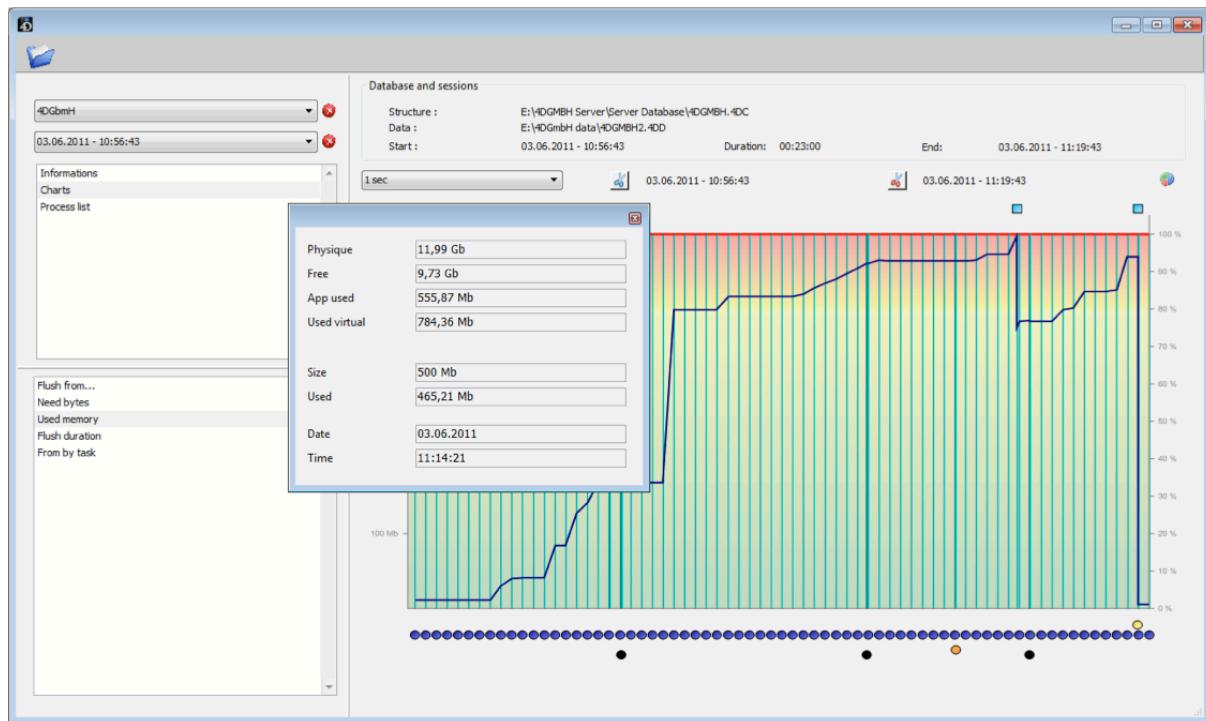
拡大表示するために、上部にある青と赤のハサミを使用してタイムフレームのサブセットを選択できます。青ハサミをクリックしてチャート中をクリックすると、チャート中のそれ以前の部分がカットされます。赤で同様のことを行えばそれ以降がカットされます。



最初の 500MB の例では、この期間においては、キャッシングが少なかったです。2GB は必要とされるよりも多すぎました。以下の例では 512MB のキャッシングでは少なすぎる例です。



これらの例を見ると、この図が情報源としてとても重要であり、通常 4D Log Analyzer で最初にチェックすべき点です。後ほどさらにサンプルを見ていきます。



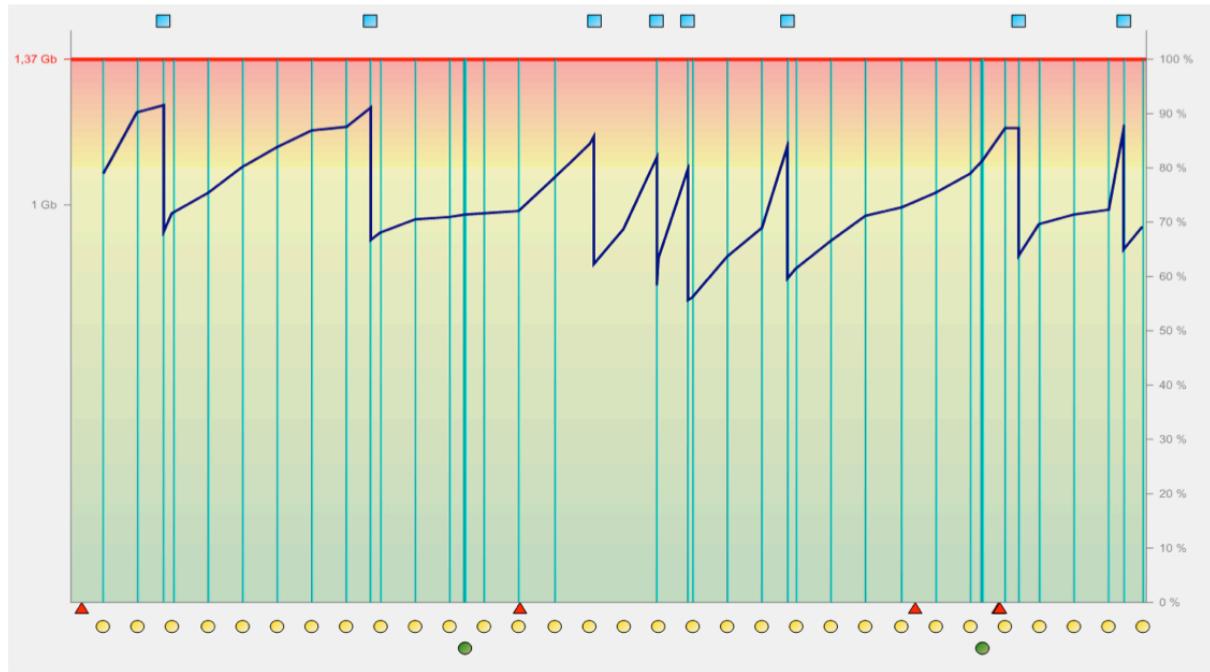
上部の小さな円グラフアイコンをクリックすると詳細ウインドウが開かれます。表中の任意の場所をクリックすると、この瞬間にに関する詳細情報を取得できます。

これによると 500MB 中 465MB のキャッシュが使用され、コンピューターには 9.7GB の空きメモリがあることが分かります。

LOG EVENT

4Dランゲージの LOG EVENT コマンドには、ドキュメント化されていない4番目のオプションがあり、使用メモリ表にコメントを入れられます。

LOG EVENT(4;"my comment")

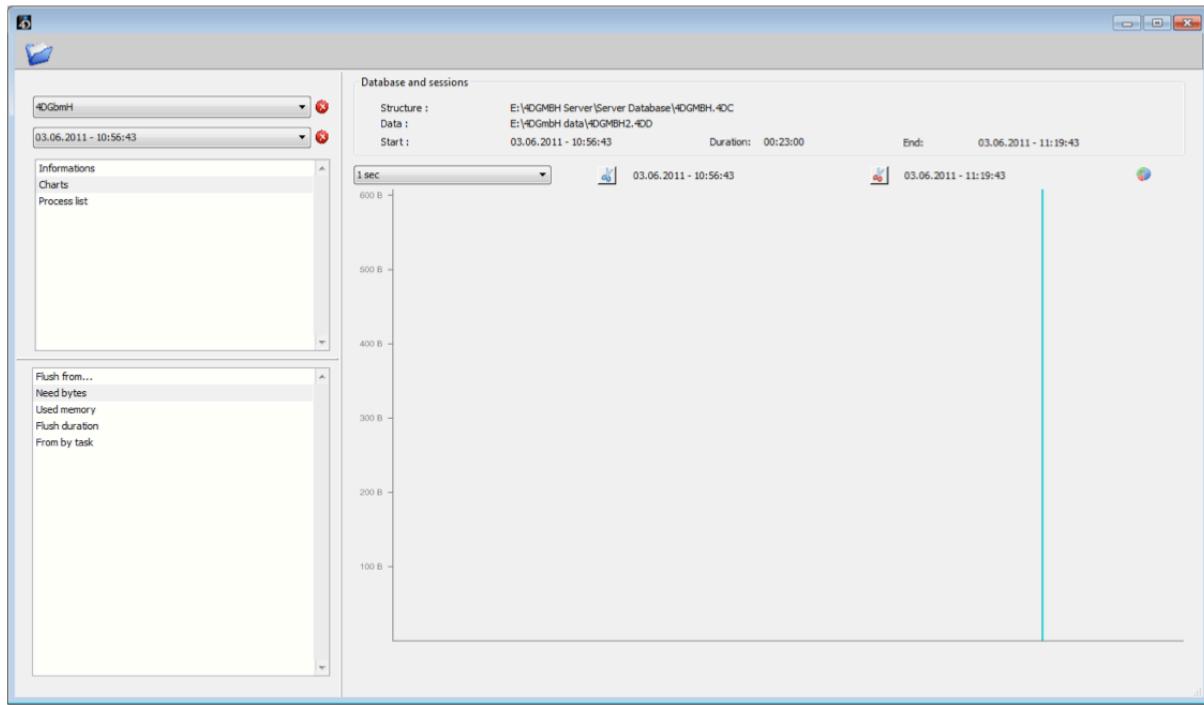


表に赤の三角が表示されます。三角上にマウスを重ねるとコメントが表示されます。このオプションを使用してキャッシュの利用状況とイベント/アクションを関連付けられます ("started statistic method xyz" や "method xyz line 123"など)。

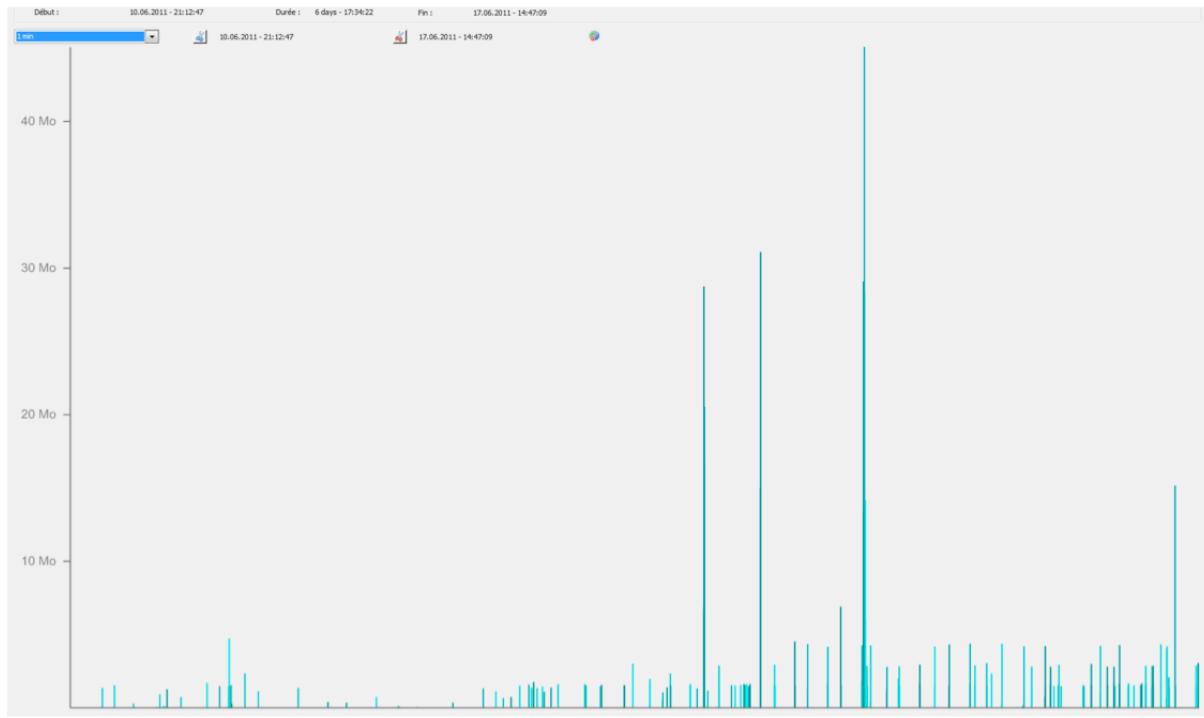
NEED BYTES

表上部に青い四角が表示されている場合、利用可能なメモリより多くのメモリが必要になったことを示します。

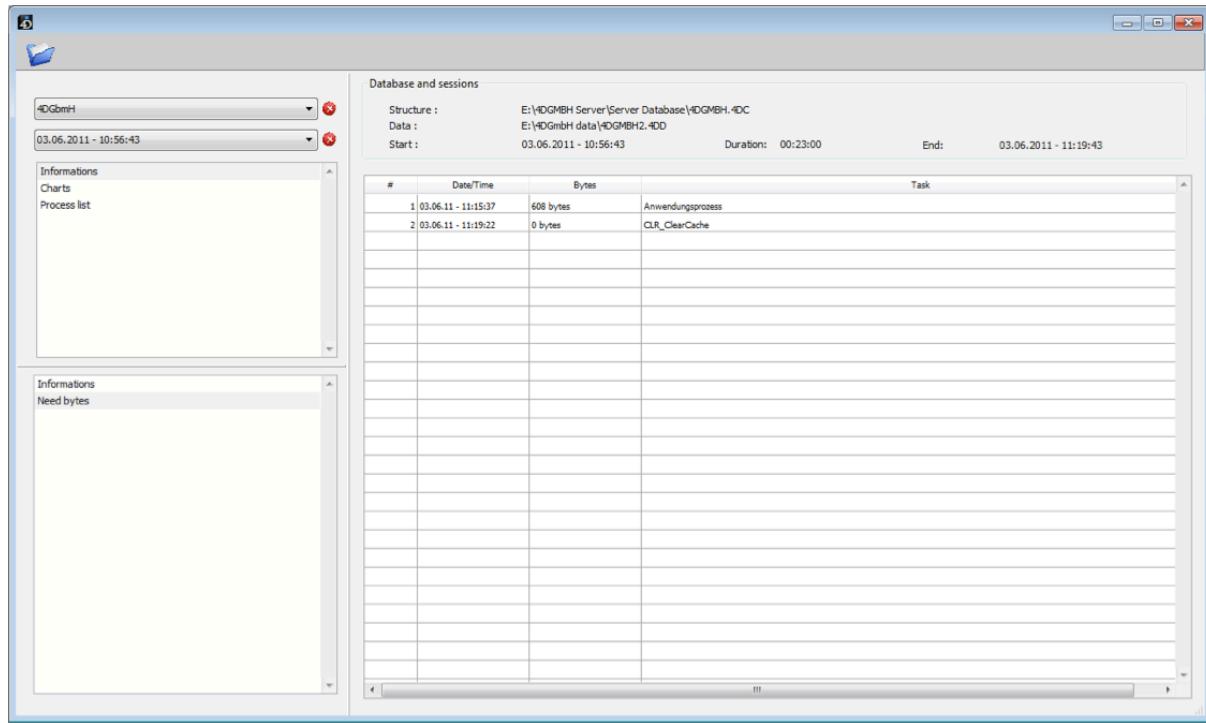
詳細情報を得るために "Used memory" をクリックします。



このサンプルでは 608byte を必要とする小さなリクエストがあるだけです。次のサンプルはより多くのメモリを必要としています。



必要なメモリをより詳細に表示する二番目の画面があります。上のリストボックスで"Information"をクリックし、下のリストボックスで"Need bytes"をクリックします。



画面にはメモリが要求された日付、サイズ、そしてプロセス名が表示されます。上のサンプルの2番目は FLUSH BUFFERS(*)によるものです。

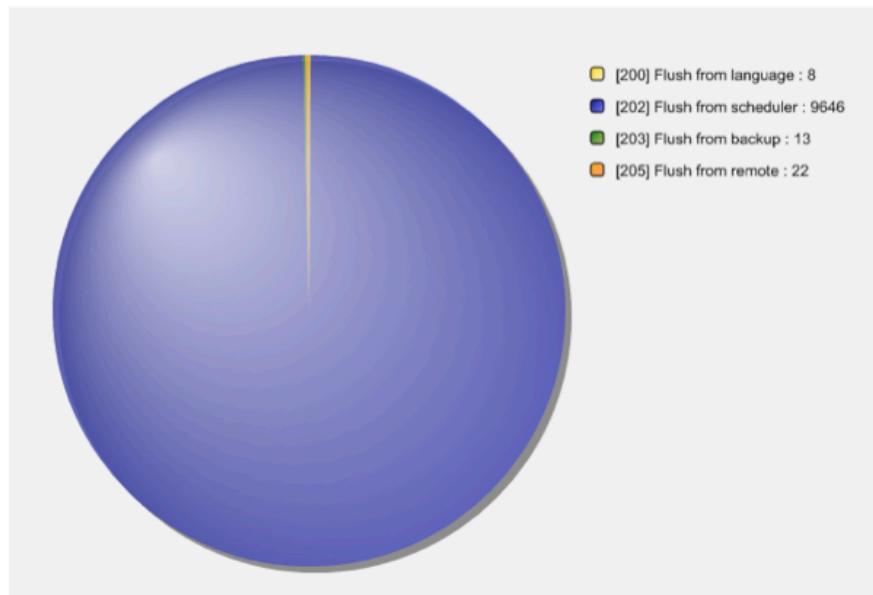
サンプルの同じ画面では 45MB メモリが必要とされています。

#	Date/Time	Bytes	
137	15.06.11 - 14:46:15	1,47 Mb	
138	15.06.11 - 14:47:17	2,27 Mb	
139	15.06.11 - 14:47:38	3,34 Mb	
140	15.06.11 - 14:47:42	3,34 Mb	
141	15.06.11 - 14:47:42	3,34 Mb	
142	15.06.11 - 14:47:42	3,34 Mb	
143	15.06.11 - 14:47:43	3,34 Mb	
144	15.06.11 - 14:47:43	3,34 Mb	
145	15.06.11 - 14:47:43	3,34 Mb	
146	15.06.11 - 14:47:43	3,34 Mb	
147	15.06.11 - 14:48:38	3,34 Mb	
148	15.06.11 - 14:48:39	3,34 Mb	
149	15.06.11 - 14:48:39	3,34 Mb	
150	15.06.11 - 14:48:39	3,34 Mb	
151	15.06.11 - 14:50:24	2,38 Mb	
152	15.06.11 - 14:51:21	1,48 Mb	
153	15.06.11 - 14:51:52	1,85 Mb	
154	15.06.11 - 14:52:57	2,35 Mb	
155	15.06.11 - 14:53:57	2,39 Mb	
156	15.06.11 - 14:54:07	2,38 Mb	
157	15.06.11 - 14:54:21	1,49 Mb	
158	15.06.11 - 14:55:12	3,52 Mb	
159	15.06.11 - 14:55:14	3,52 Mb	
160	15.06.11 - 14:55:14	3,52 Mb	
161	15.06.11 - 14:55:14	3,52 Mb	
162	15.06.11 - 14:55:14	3,52 Mb	
163	15.06.11 - 14:55:15	3,52 Mb	
164	15.06.11 - 14:55:15	3,52 Mb	
165	15.06.11 - 14:55:15	3,52 Mb	
166	15.06.11 - 14:55:38	2,84 Mb	
167	15.06.11 - 14:55:41	3,52 Mb	
168	15.06.11 - 14:55:42	3,52 Mb	
169	15.06.11 - 14:55:42	3,52 Mb	
170	15.06.11 - 14:55:43	3,52 Mb	
171	15.06.11 - 14:56:14	3,53 Mb	
172	15.06.11 - 14:56:14	3,53 Mb	
173	15.06.11 - 14:56:14	3,53 Mb	
174	15.06.11 - 14:56:15	3,53 Mb	
175	15.06.11 - 14:56:15	3,53 Mb	
176	15.06.11 - 14:56:15	3,53 Mb	
177	15.06.11 - 14:56:16	3,53 Mb	

14:55:14 のように時折 3.5MB のリクエストがあるのが分かります。すべて同じプロセスから、時には毎秒要求されています。キャッシングメモリを増やすことに加え、このプロセスを解析するという選択肢も取れます。解析を行うために SET DATABASE PROPERTIES(Debug Log Recording;3)や LOG EVENT(4;"Methode XZY line 123")でキャッシングログにコメントを挿入することができます。

FLUSH FROM

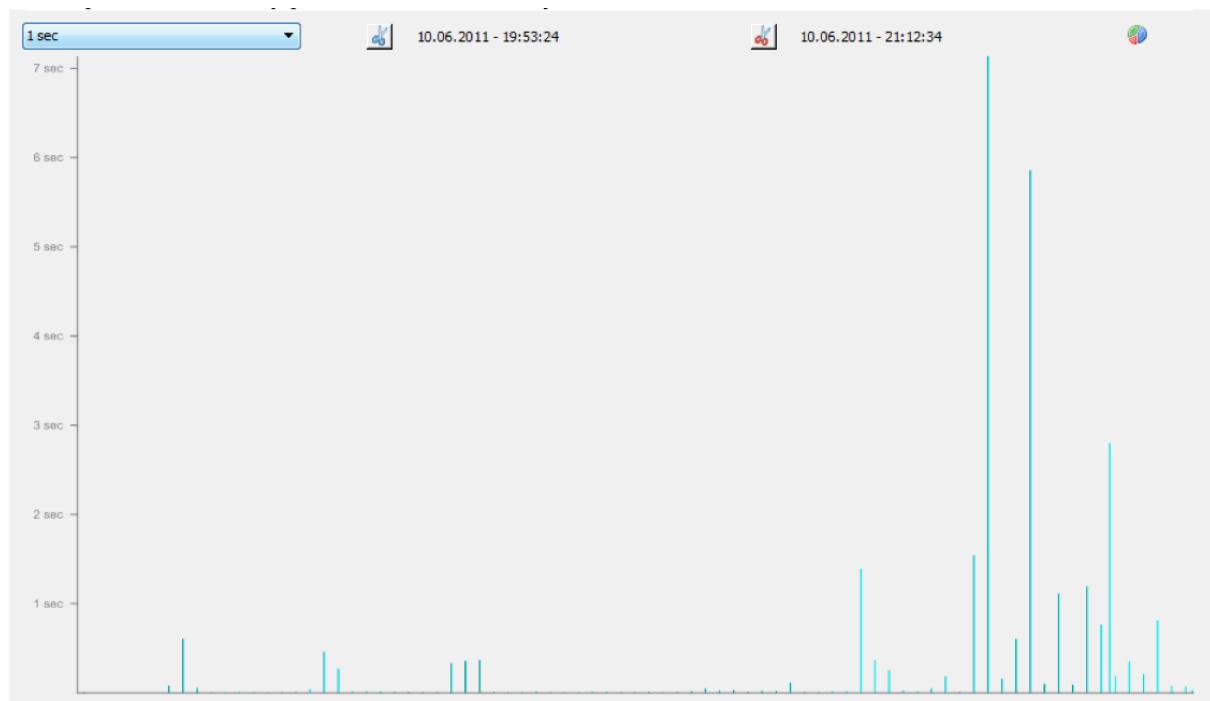
この表は統計情報を得る目的でのみ使用し、なぜバッファーのフラッシュが呼び出されたかを示します：



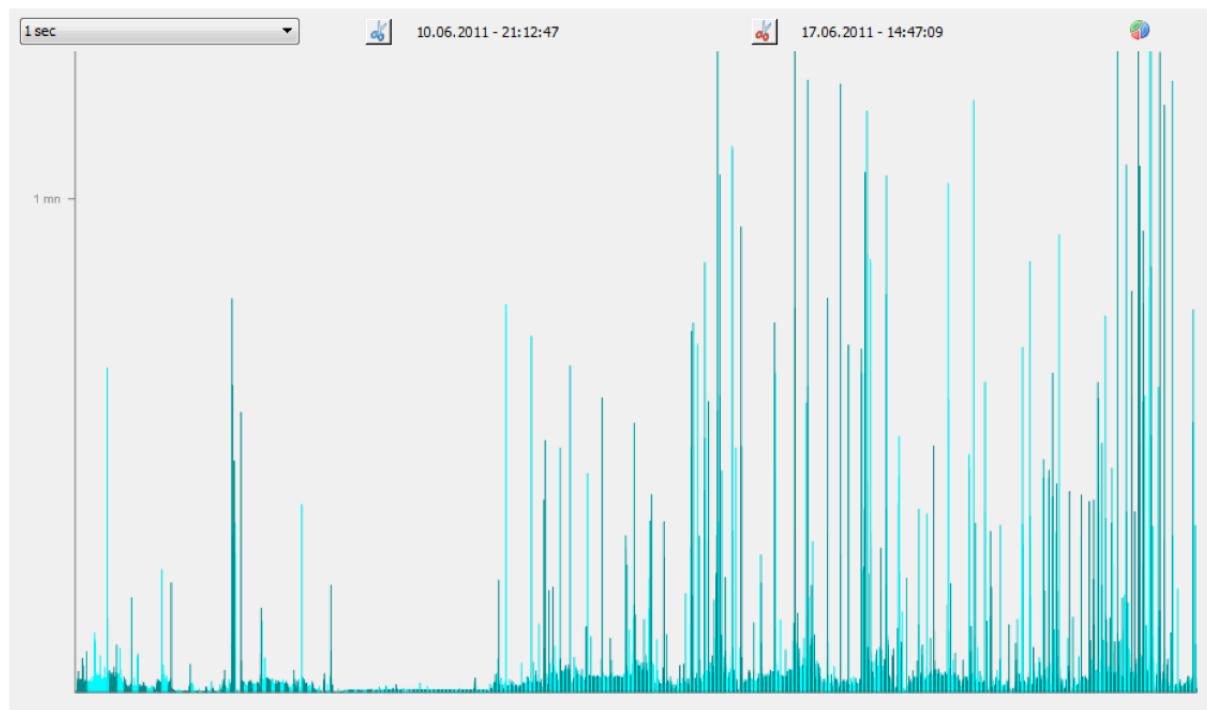
以前の 4D (特にバージョン 6) では、データを確実にディスクに書き込むために、重要な処理の後に FLUSH BUFFER を呼び出すことが一般的でした。4D Backup が統合されたことによりこの必要性は薄まりました。また自動バックアップが統合されたこと、およびキャッシュアーキテクチャーが変更され、毎 20 秒ごとにフラッシュされるようになったことでこの必要性は全くなくなりました。多くのアプリケーションが昔のコードのまま FLUSH BUFFER を呼び出しています。このダイアログを使用してこのことを素早くチェックできます。

FLUSH DURATION

フラッシュに要した時間をこの表で見ることができます。通常これは数秒です。

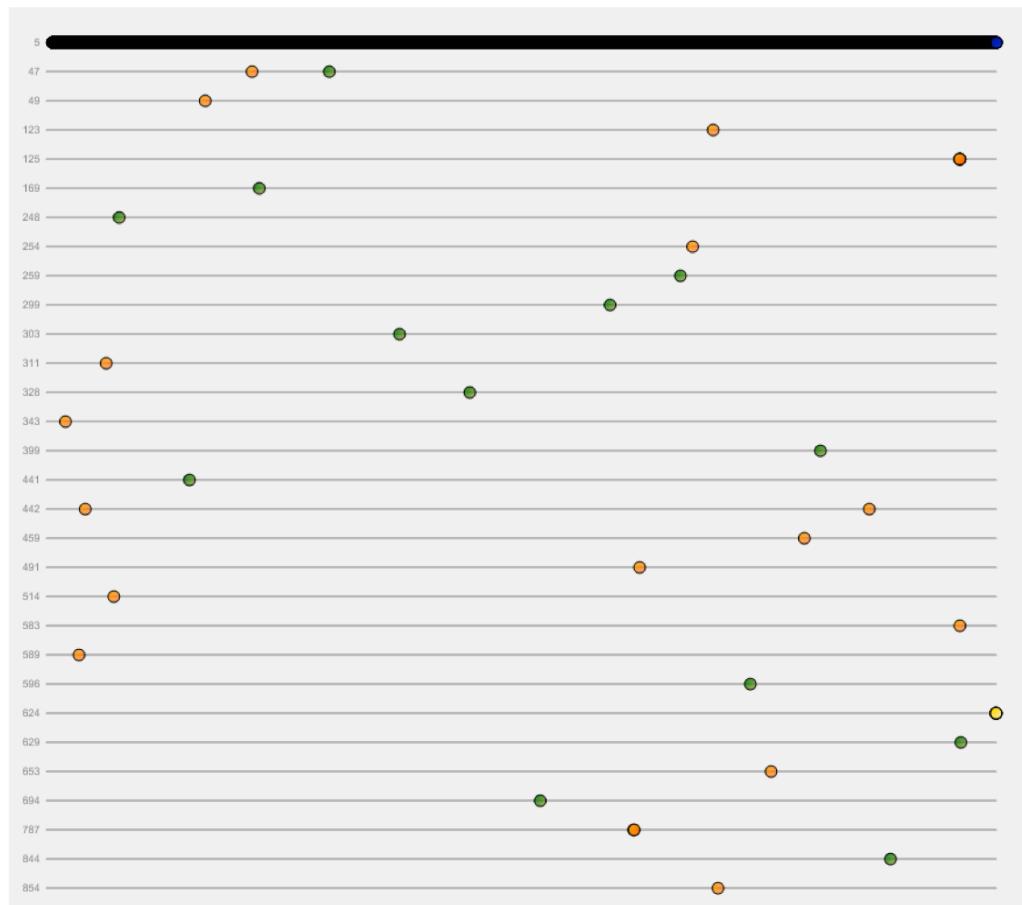


分単位で時間がかかるっている場合、サーバーが忙しすぎるか、ハードディスクが遅すぎるかの警告となります。



FLUSH BY TASK

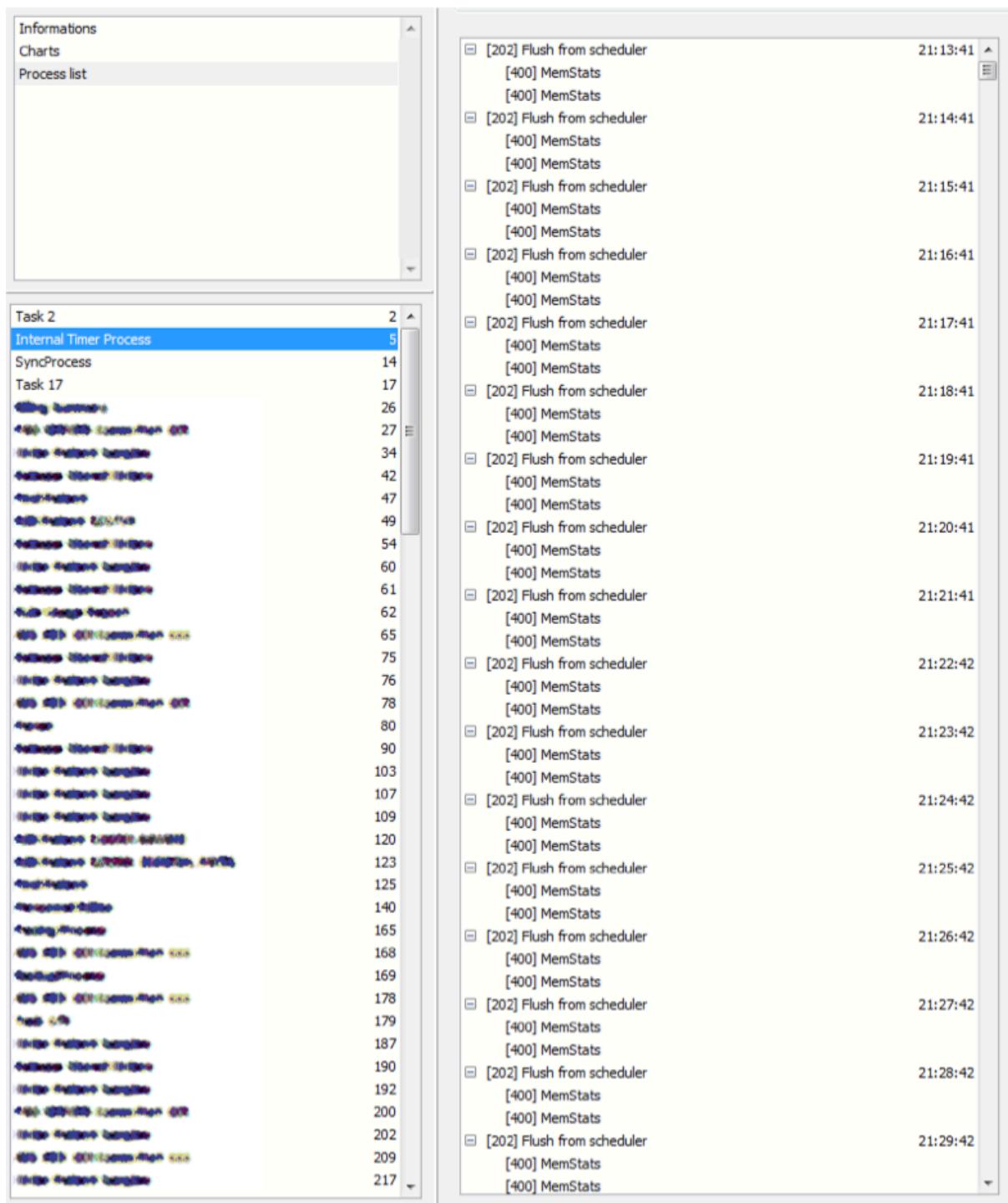
この画面ではフラッシュを実行したプロセスが、プロセス番号でソートされて表示されます。



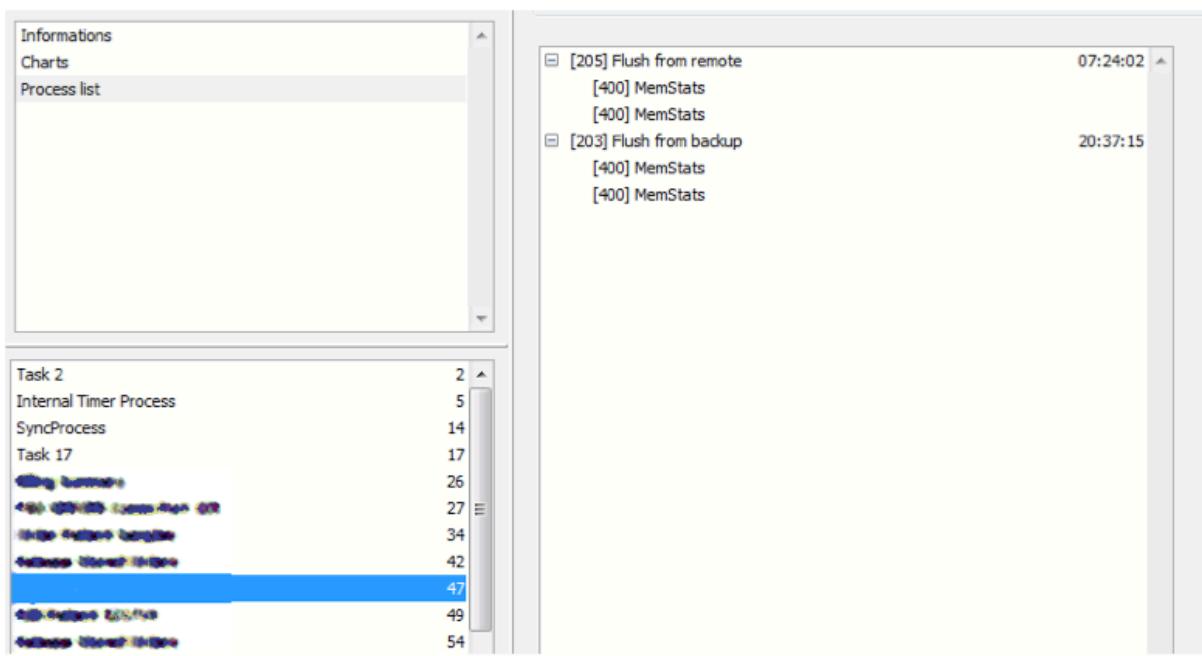
色は”Used memory”と同じ意味です。このサンプルでプロセス 5 はスケジューラーであり、通常 20 秒毎にフラッシュを行いますが、この例では毎分です。

フラッシュの時間は”Overview”の最後の画面を使用して詳細に解析できます。

PROCESS LIST



上の図は毎分バッファーのフラッシュを呼び出す自動スケジューラーを表示しています。次の図は先の表のプロセス 47 で、クライアントから一回バッファフラッシュを呼び出し、そして 4D Backup を開始しています。

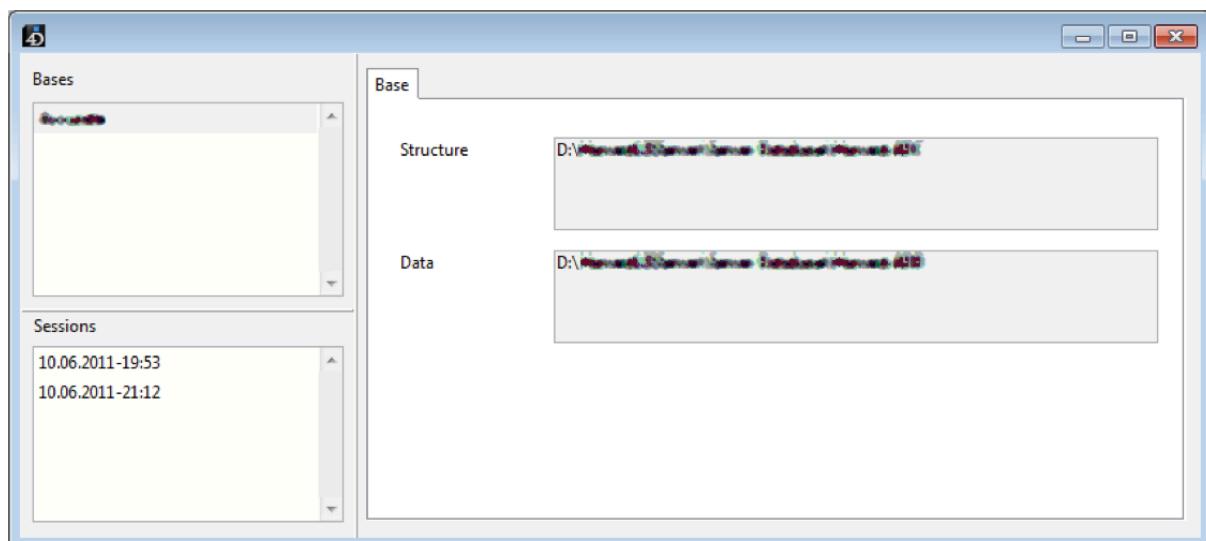


メモリが足りないためフラッシュが実行されると、下図のようになります。

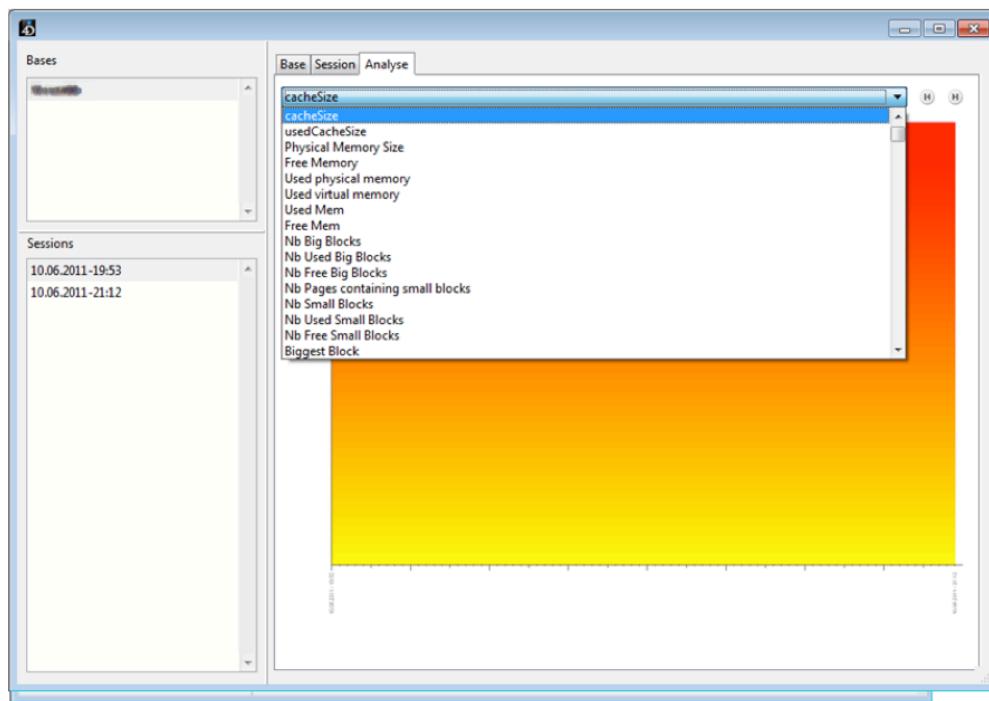


STATISTICS DETAILS

このダイアログではキャッシングの状態に関する詳細情報を得ることができます。



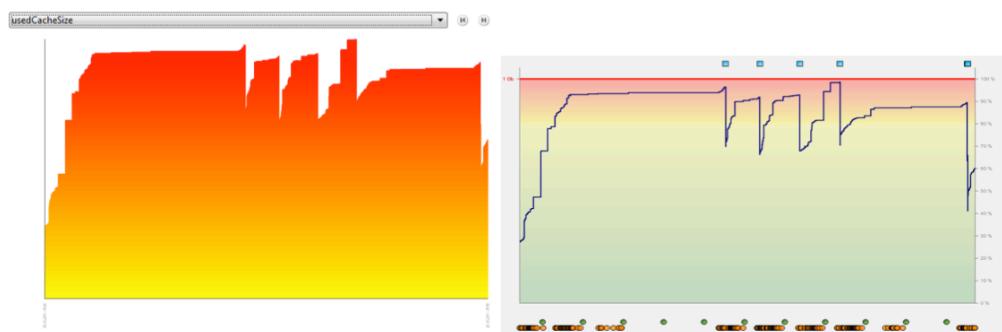
タイムフレームを選択すると、ダイアログにより多くのオプションが表示されます。



ポップアップを使用してどの情報を表示するか選択できます。

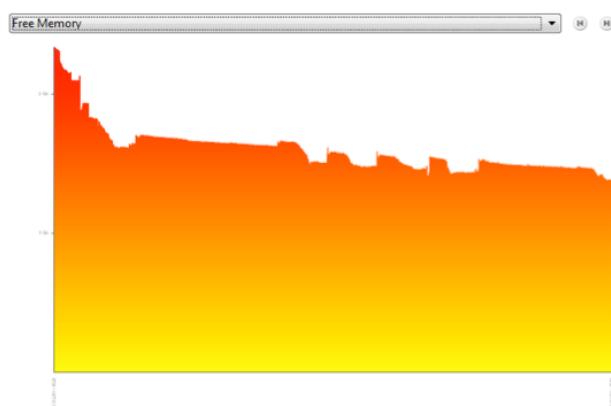
USED CACHE SIZE

この表では使用済みキャッシュサイズと同じ結果を見ることができます。



FREE MEMORY

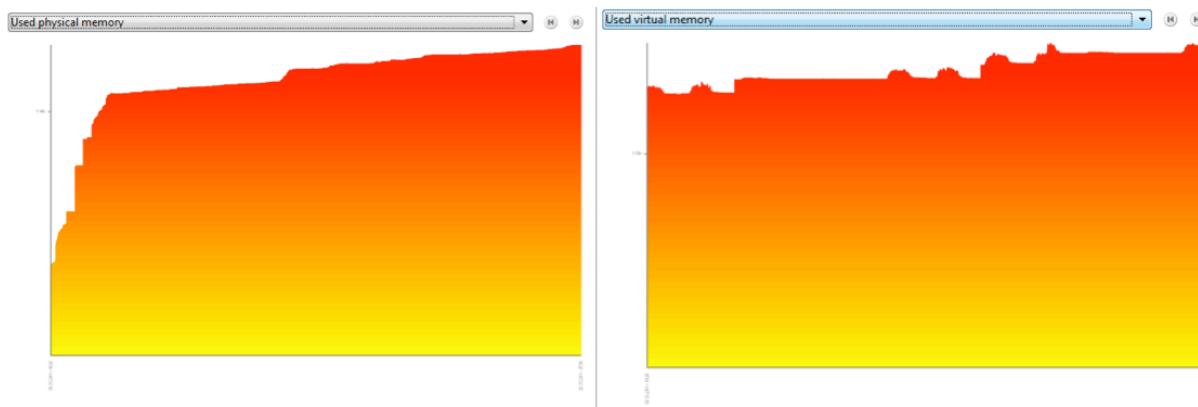
この表では OS からみた空きメモリを見ることができます。これらのデータは Windows タスクマネージャーやアップラクティビティモニターと同じデータです。



Windows は 60%以上メモリの空きがあると認識しているのに、4D が使用できるキャッシュメモリはほとんどいっぽいであることが分かります。

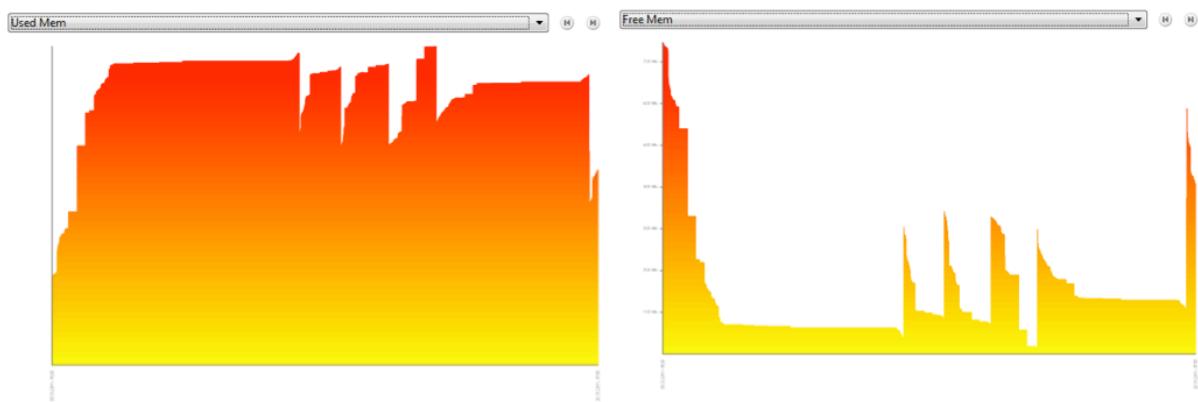
USED PHYSICAL MEMORY と USED VIRTUAL MEMORY

“Free Memory”には物理および仮想メモリが含まれているのに対し、“Used physical memory”はより緻密に使用中の物理 RAM の量を表示します。“Used virtual memory”と比べることができます。



USED MEM と FREE MEM

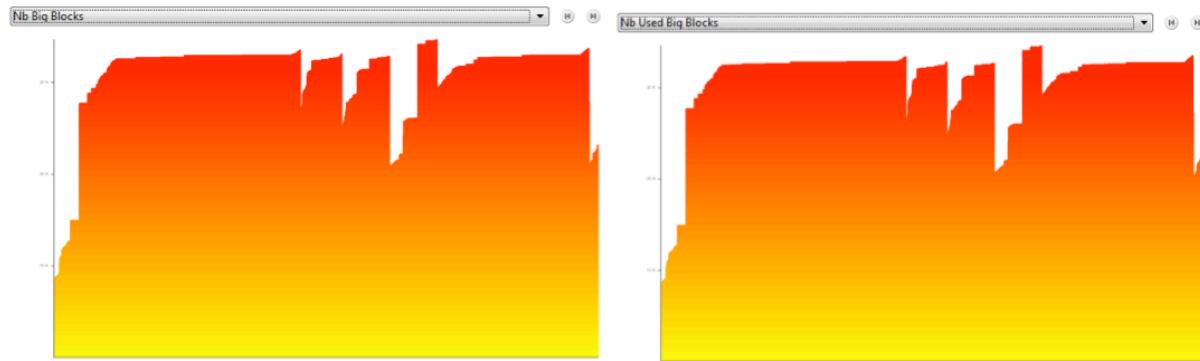
Used physical/virtual memory が OS からみたものであるのに対し、Used Mem and Free Mem は 4D からみた値です。Total memory はキャッシュおよびカーネルメモリ両方を意味します。



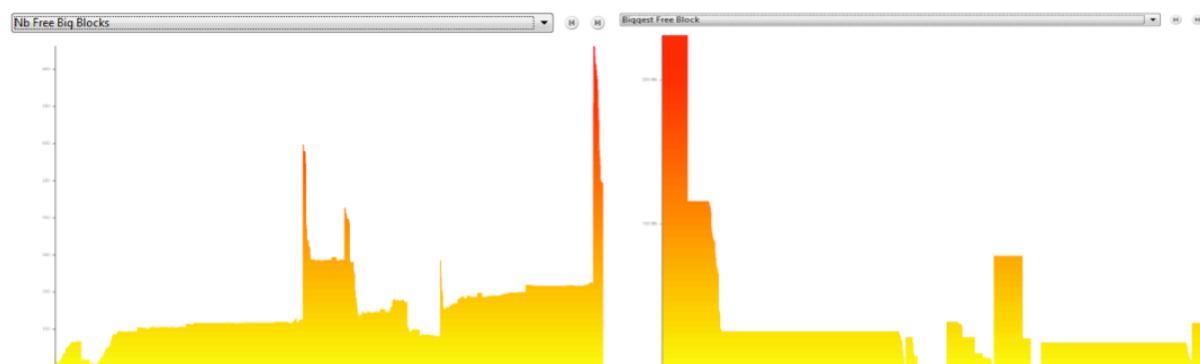
小ブロックと大ブロック

4D は小さなオブジェクトと大きなオブジェクトを異なる方法で処理します。小さなブロックを探すのが簡単であるのに対し、大きなブロックサイズの連続したメモリ領域を探すのは困難な場合があるからです。キャッシュの問題が発生したときは、(大きなセットや大きな命名セレクションなどのため) 大きなブロックを確保するのに十分な空きがなかったためと考えられます。

大ブロック

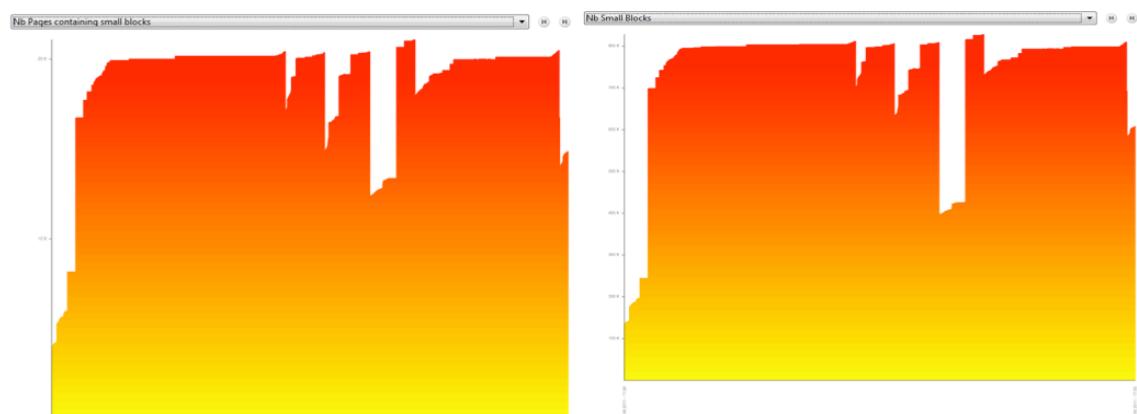


(残り 10k, 20k, 30)

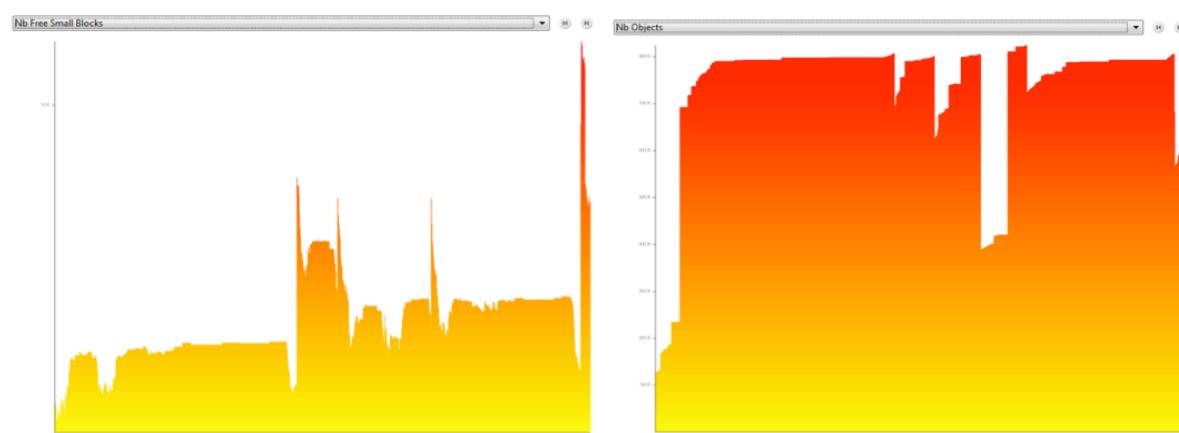


(残り 100, 200, ..., 800) (残り 100 MB, 200 MB)

小プロック



(残り 10k, 20k) (残り 100k..800k)

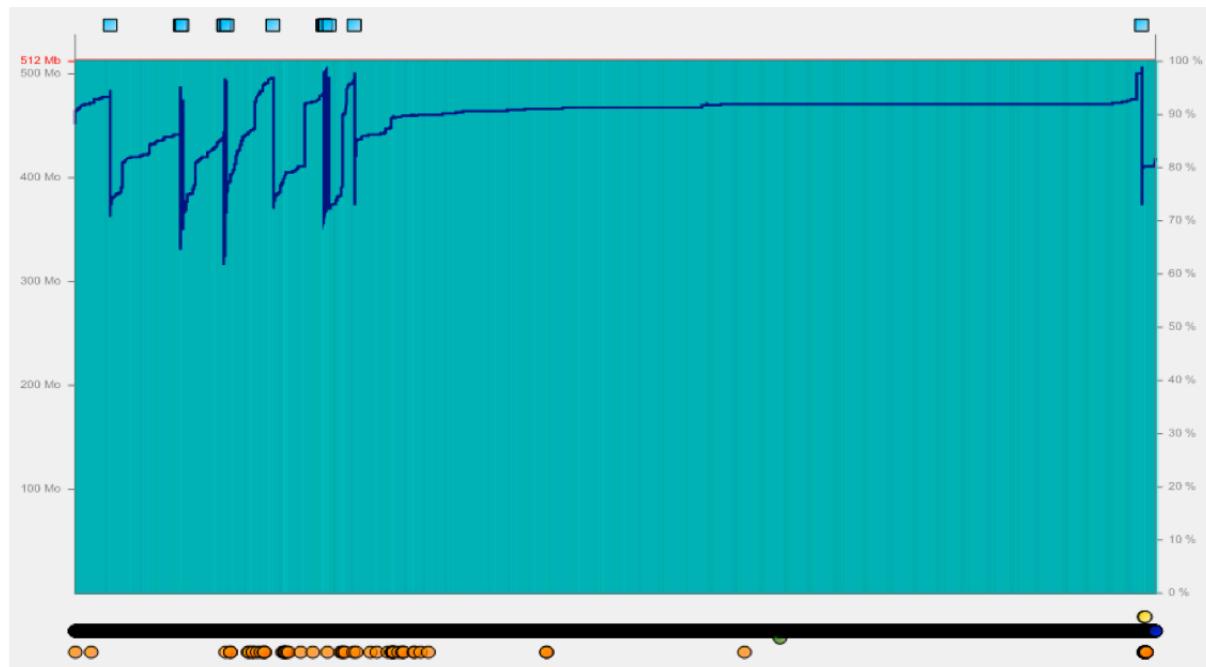


(残り 50k) (残り 100k..800k)

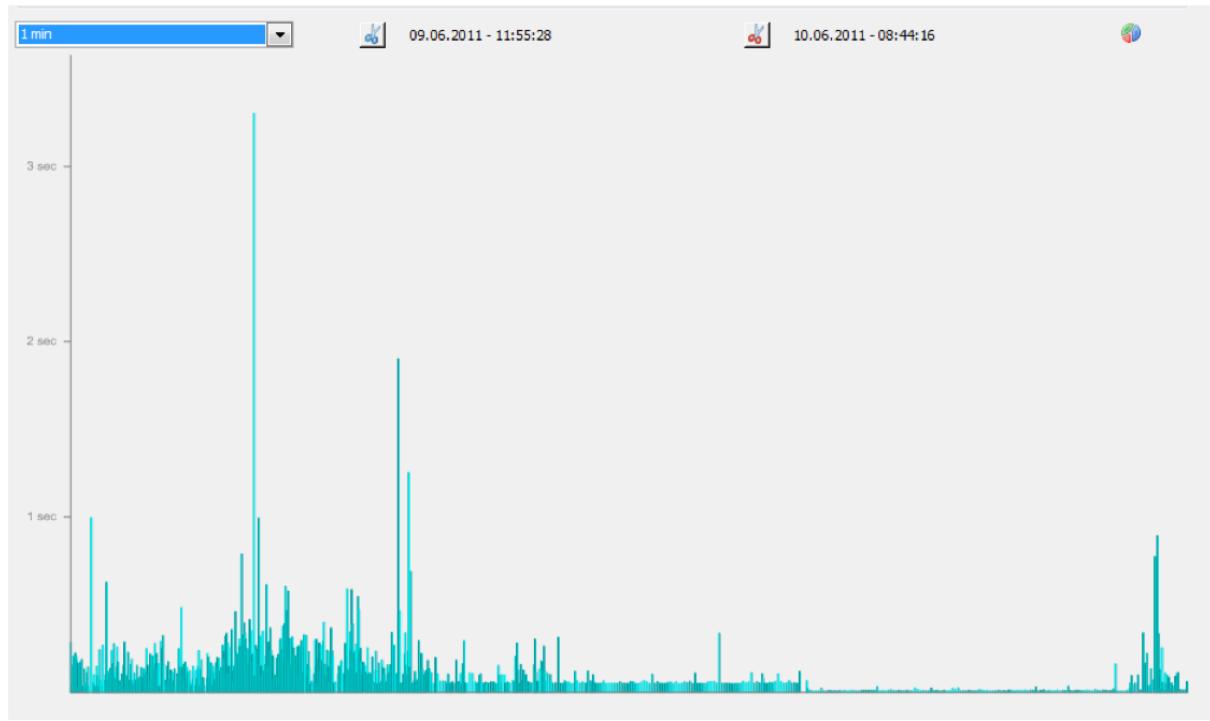
このとおり多くの情報を得ることができます。難しいのはそれを理解することです。手助けのため以降いくつかのサンプルを提供します。これらはいずれも実際のアプリケーションから収集されたものです。

サンプル

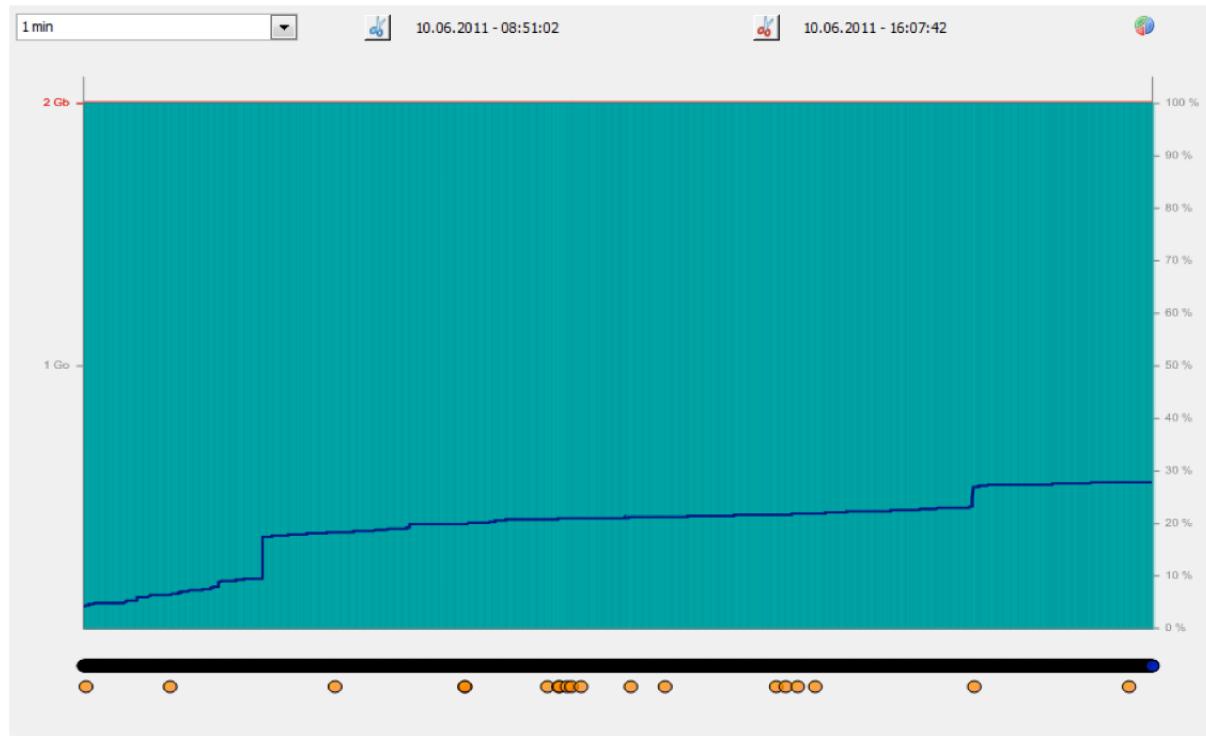
20 ユーザー、またユーザーごとに 10 プロセスを作成するサーバーからはじめます。このアプリケーションは Windows Server 2008 64-bit および 8GB RAM のコンピューターで実行されていましたが、4D のキャッシング設定は 512MB でした。



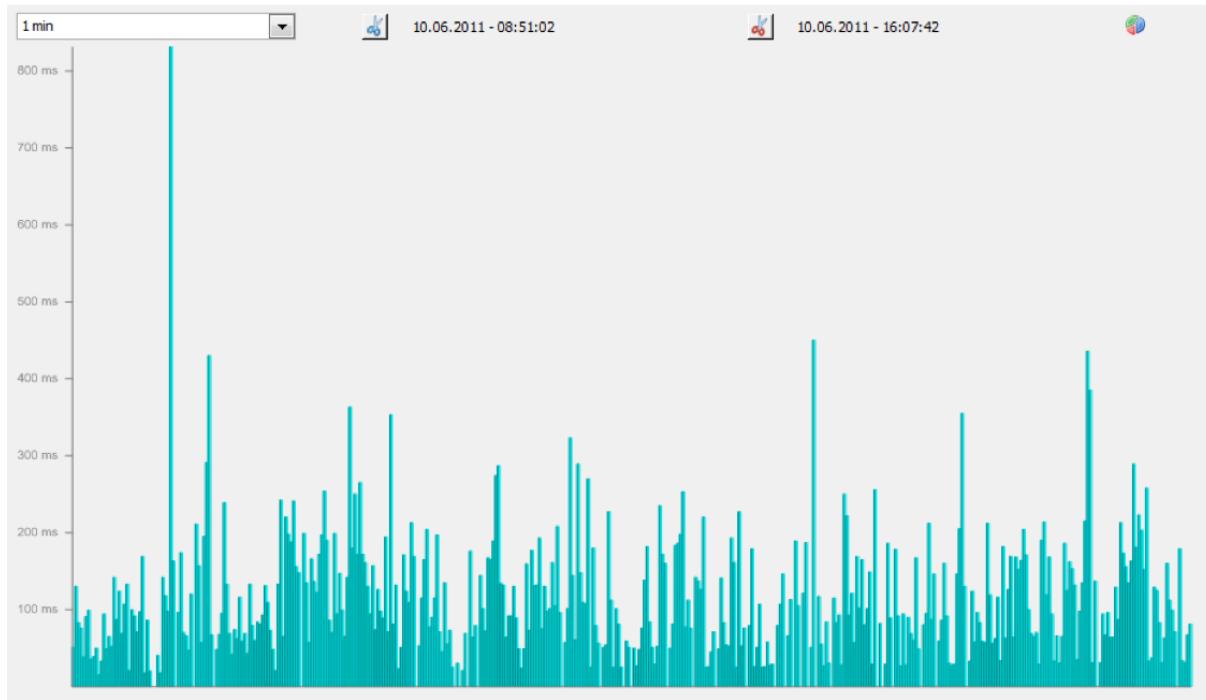
上図はキャッシング使用量を示しています。より多くの RAM を要求していることが簡単に分かります。450MB で安定しているのは夜間です。翌朝になるとまた上下を繰り返します。下図はキャッシングのフラッシュに要した時間です。ほとんどのケースで 1 秒未満ですが、時に長くかかります。いずれにしても問題はありません。



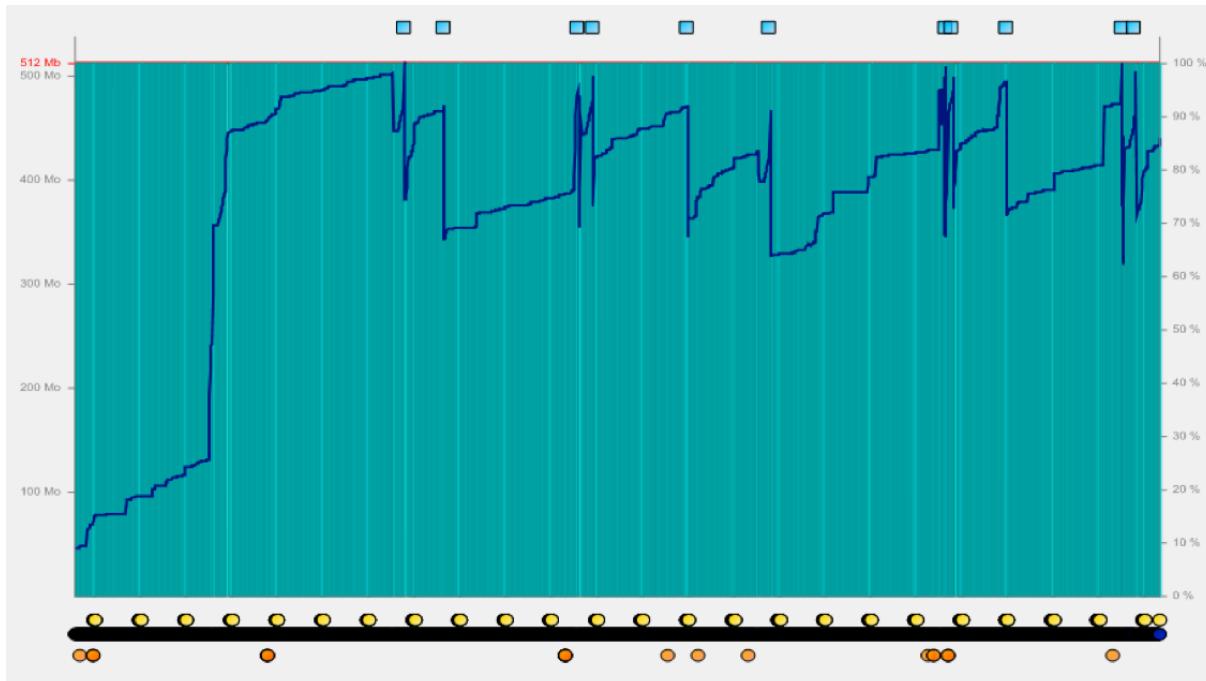
顧客はパワフルなコンピューターを使用しているので、キャッシュを 2GB に増やし再起動するよう依頼しました。



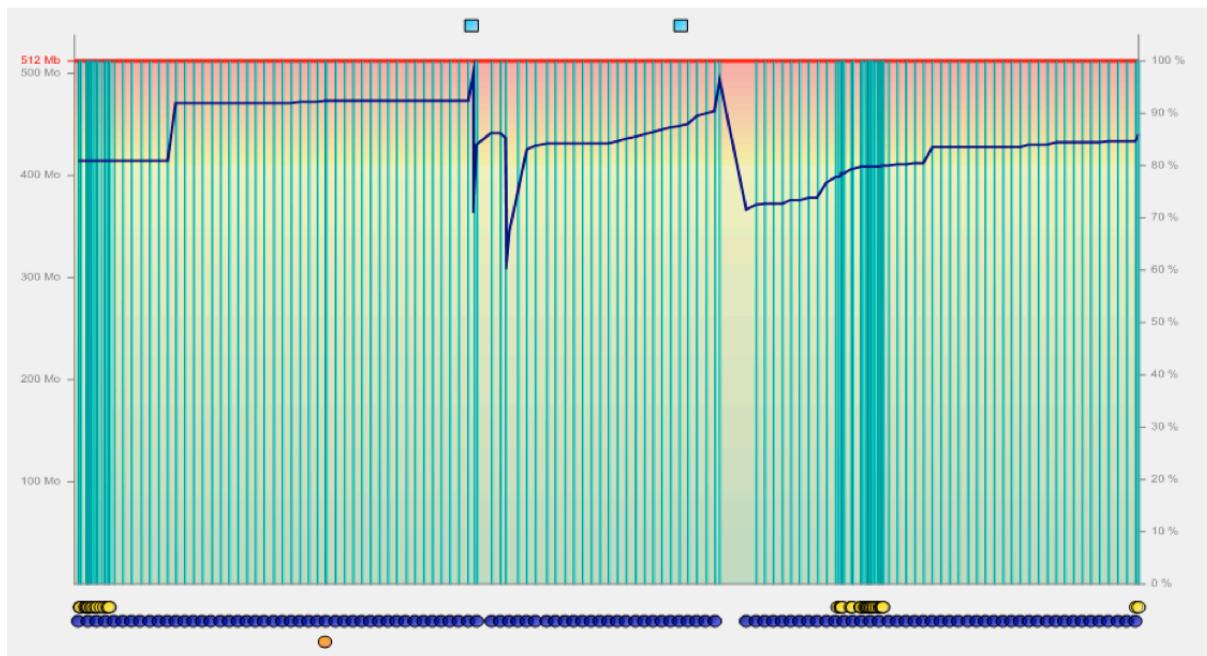
上図は同じ環境でキャッシュを 2GB に増やした後の状態を示しています。ご覧のとおり、一度キャッシュにオブジェクトが取られた後はメモリはそれ以上必要とされず、ハードディスクへのアクセスは避けられています。翌日には 60%まで増えましたがデータのページがリクエストされることはありませんでした。以下はフラッシュに要した時間です。先の図より時間がかかるっているように見えますが、単位が秒からミリ秒に変わっている点に留意してください。



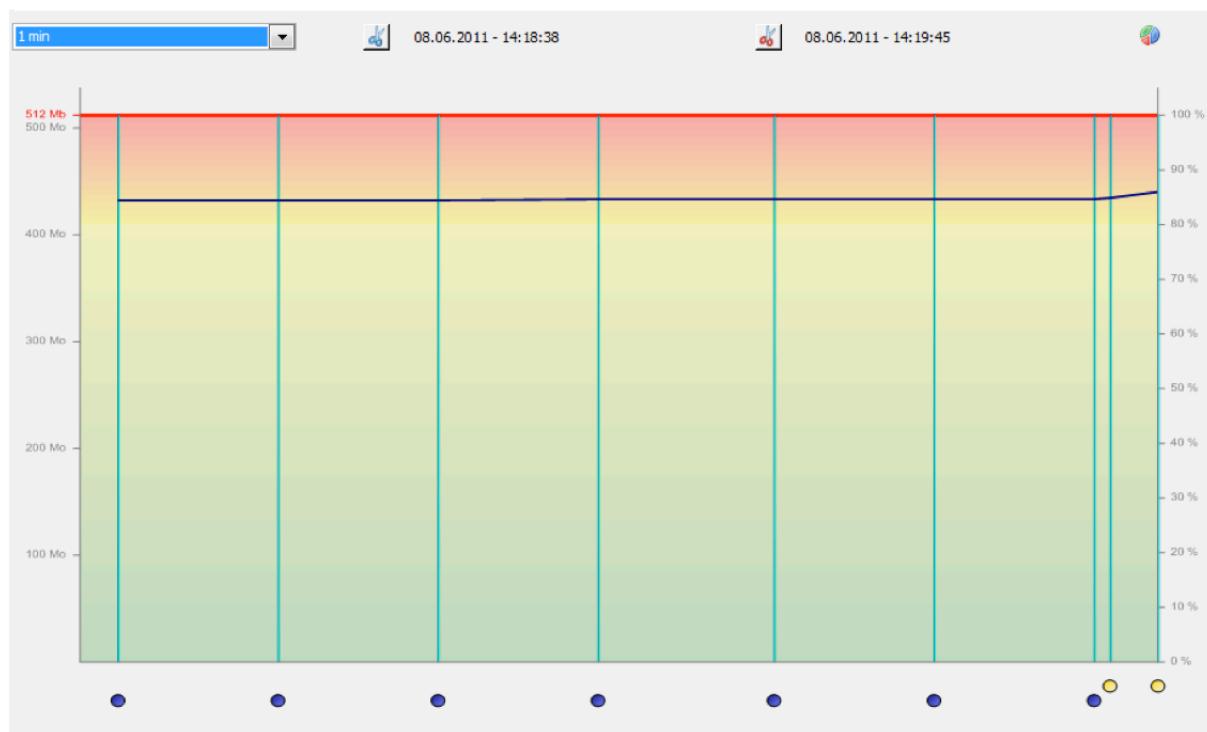
次の顧客は再現性のないランダムなクラッシュで困っていました。



何が発生しているのかを知るために Cache log recorder を使用しました。表によればアプリケーションは (40 ユーザー / 600 プロセスに対し) 512MB しかキャッシュを使用せず、あつという間にいっぱいになり、RAM が要求され、ページが繰り返されていました。最後の部分をズームしてみるとサーバーが瞬間にフリーズしていたことが分かります。緑の縦線は通常毎 20 秒ごとの自動的なフラッシュを表します。この間隔が空いているということは、メモリを解放するためにサーバーが忙しくなっており、通常のように応答できなかったことを意味します。黄丸は開発者が呼び出した FLUSH BUFFER で、1 秒に数回呼び出されています。また最後にも存在します。



クラッシュ直前をズームすると使用メモリが増加し、最後のミリ秒で FLUSH BUFFER が呼び出されているのが分かります。

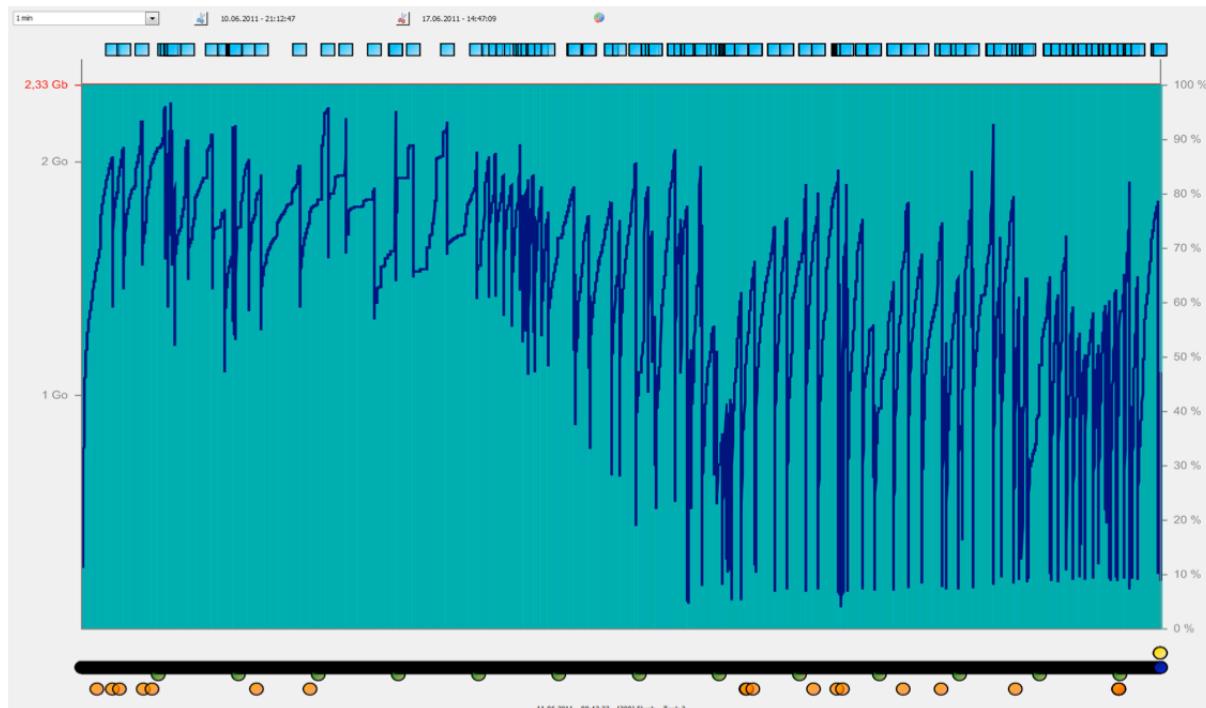


この顧客は緊急な解決策を必要としていたので、本当のクラッシュの原因は分からませんでした。解決策として次のような提案をしました。

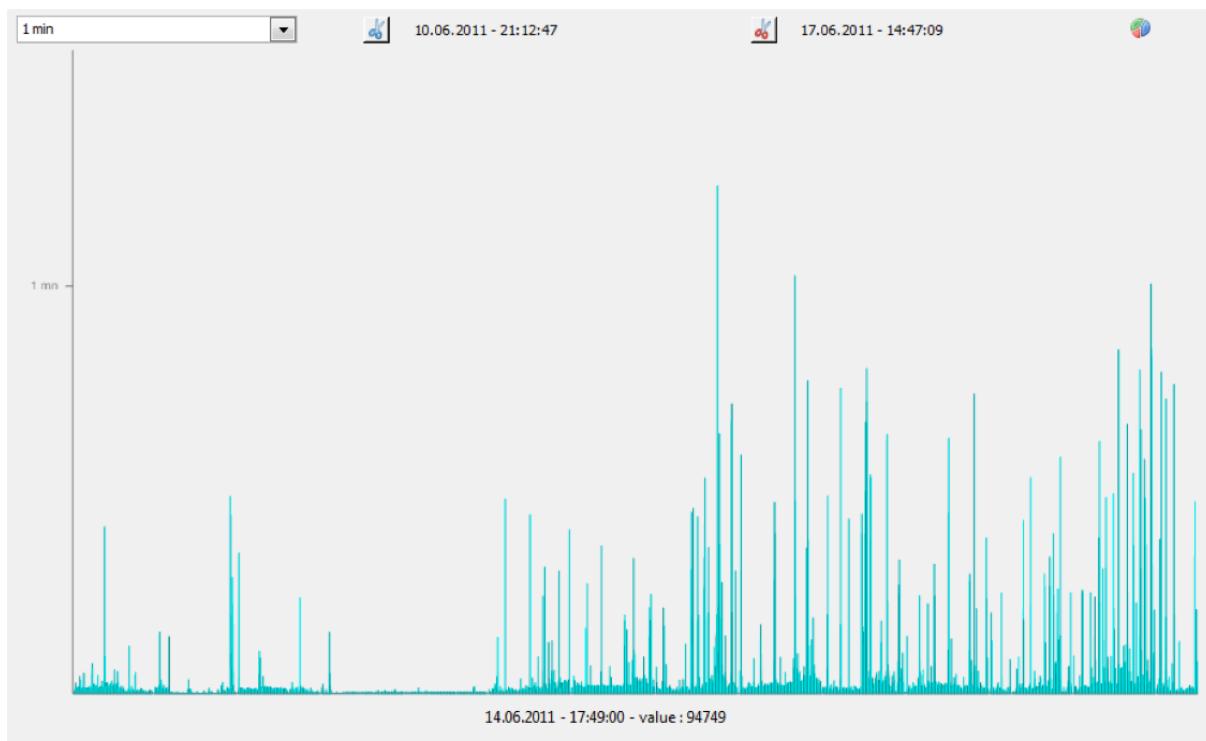
- キャッシュを 512MB から 1200MB に増やす (コンピューターが 32-bit OS のため)。
- メモリが足りない状況下で余分の作業を行わないようにするために、FLUSH BUFFER コマンドの呼び出しを取り除く。
- OS を 64-bit にアップグレードし、RAM の搭載量を増やす。

最初の 2 つの作業によりクラッシュは解消され、それ以降の原因究明は顧客の希望により行われませんでした。また後日 64-bit OS および 8GB メモリにすることで速度が向上しました。

次の表は 50 ユーザー、875 プロセスのものです。アプリケーションは 64-bit OS で動作していますが、4D Server は 32-bit です。32-bit OS の上限である 2.3GB キャッシュが割り当てられています。

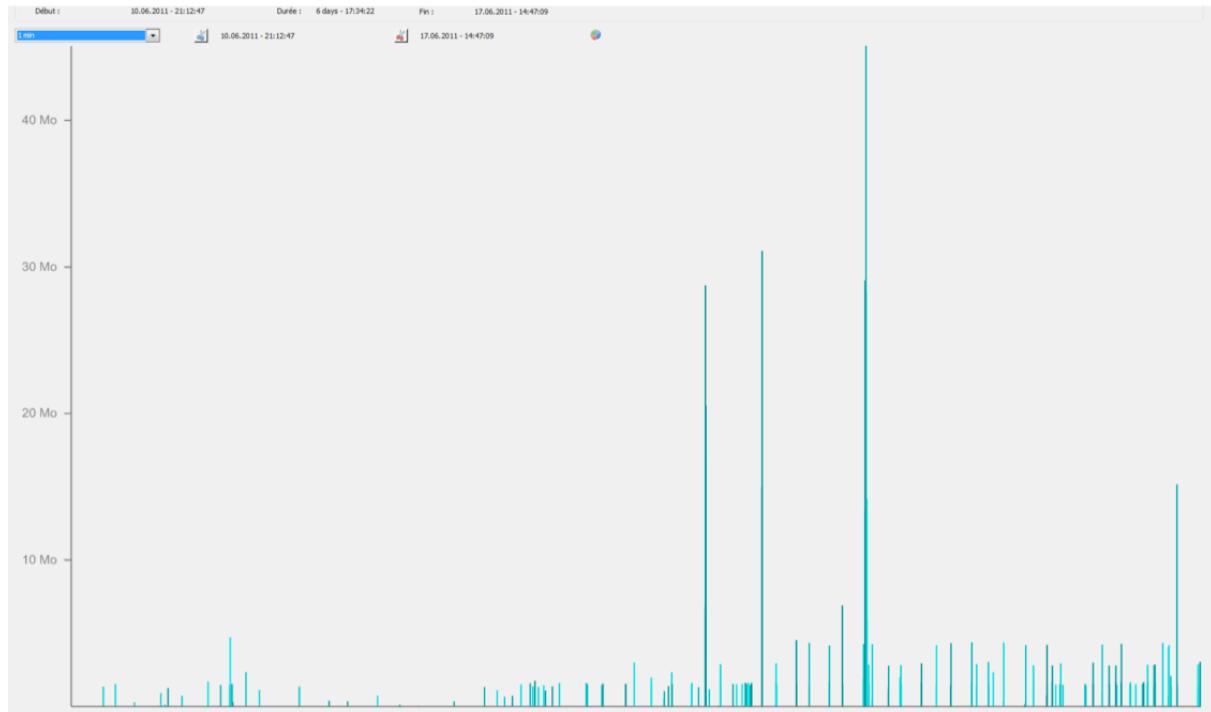


キャッシュの使用量が大きく上下しているのが分かります。上部の青い四角はサーバー上でメモリが要求され、他のオブジェクトを配置するためにキャッシュの一部をページする必要があったことを示しています。

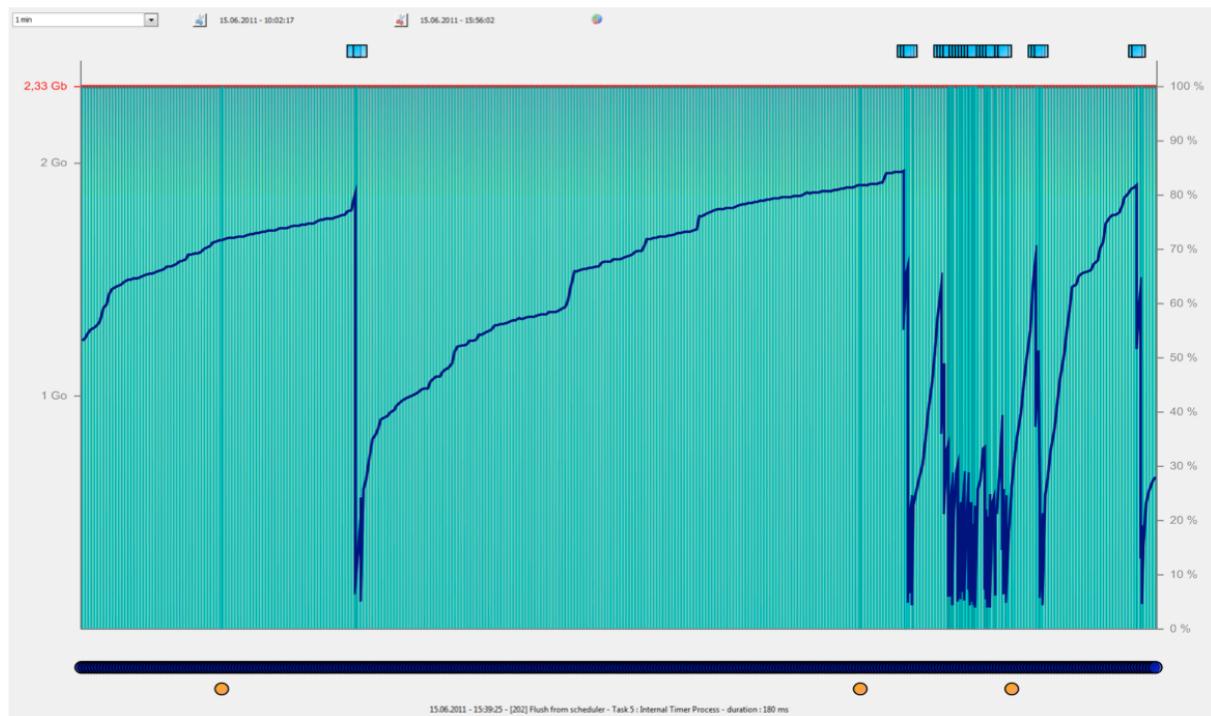


フラッシュに 90 秒以上要している箇所は、サーバーが特に忙しかったことを表しています。これはキャッシングが少なすぎて、しばしばハードディスクにアクセスする必要があったためと思われます。

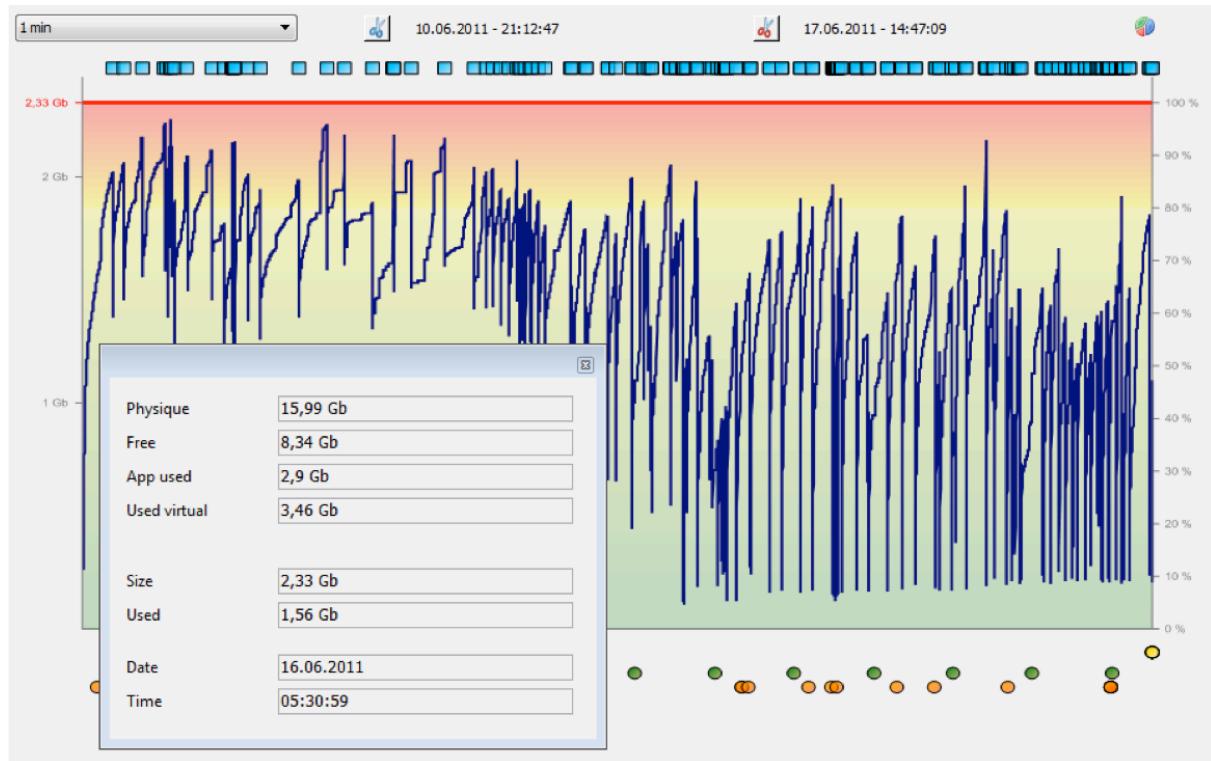
"need bytes" 表を見てみると最大 45MB のメモリリクエストが行われています。



5 時間分のサブセットを表示すると、キャッシング全体がしばしばページされていることが分かります。2~3 時間かけてキャッシングが埋められ、またページされます。そして大量のメモリリクエストが発生し、データがしばしばページされています。



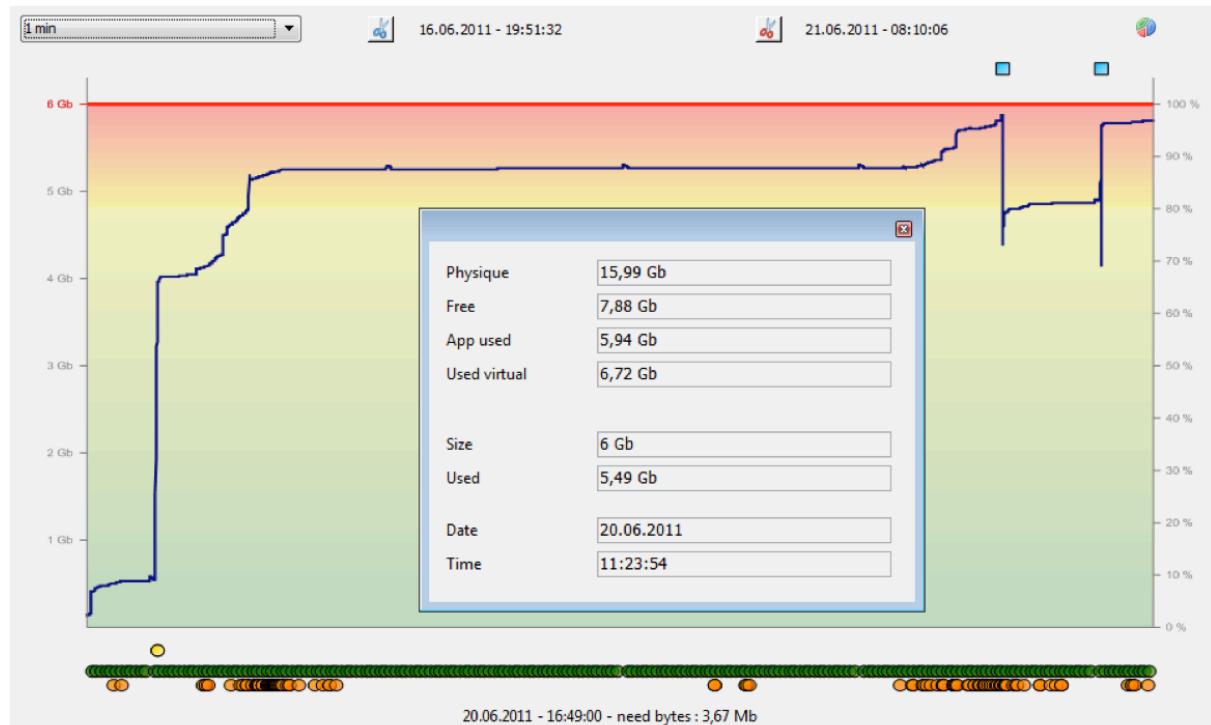
総メモリ使用量は何度か最大に近くなっています。



使用仮想メモリの最大値は 3,805,228 でこれは 32-bit アプリケーションが使用可能な 4GB に近いものです。結果、今回はキャッシュサイズを 2.3GB から 2GB あるいは 1.8GB まで減らし、4GB に達することの無いよう提案を行いました（表に基づき 2.3GB のキャッシュでも足りないと思われるケースであるにも関わらずです）。

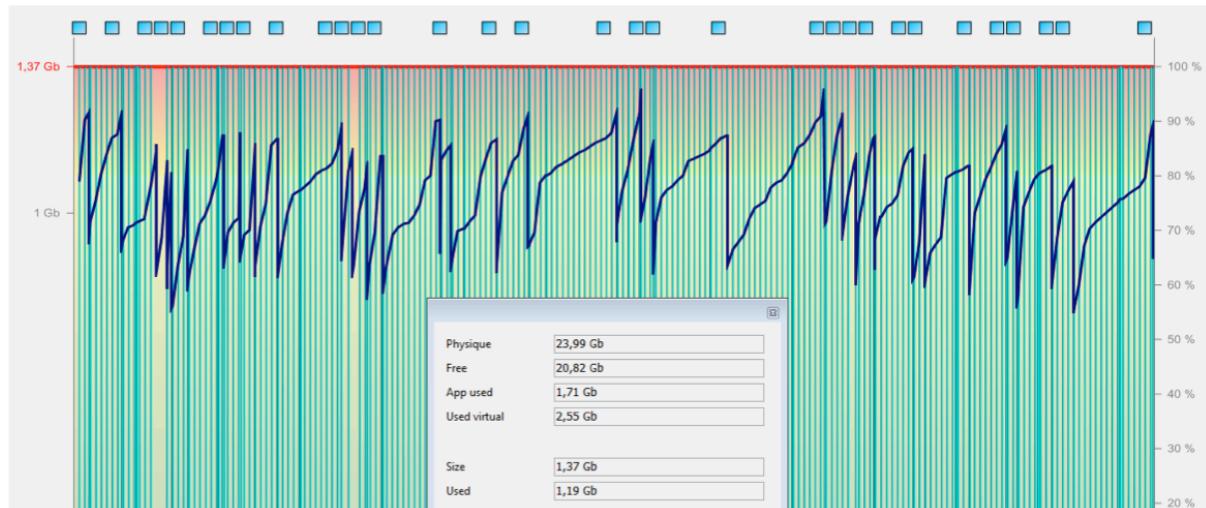
コンピューターには 16GB の RAM が積まれ、そのほとんどは使用されていないため、本当のソリューションは 64-bit バージョンの 4D Server を使用することです。

次のサンプルは 75 ユーザー、250 プロセスのものです。Windows Server 2008r2 64-bit に 16GB の RAM が搭載され、4D Server v12 64-bit が動作しています。



上の表はアプリケーションを一週間動作させたもので、とても安定したメモリの状態を示しています。2つのピークがあり、結果として小さなページが発生しています。コンピューターにはまだ 7.8GB の空きメモリがあるため、キャッシングを 6GB から 10GB に増やすことを提案しました。

次の表は 24GB のコンピューターで 32-bit サーバーを一時間動作させたものです。



このような状況では 64-bit バージョンのサーバーを使用することをお勧めします。

4D INFO REPORT COMPONENT

使用されている OS やハードウェアおよびメモリの使用状況をモニターするためのコンポーネントです。このコンポーネントはとても容易に使用でき、v11 と v12 両方で動作します。このコンポーネントは多くの情報を書き出しますが、ここではその概要と、その情報をどのように使用するかについてのヒントを提供します。

例 1

マシン仕様

コンピューター名	my4Dserver
ユーザ	SYSTEM
製造	IBM
機種	IBM eServer BladeCenter HS20 -[8832M1X]-
搭載 RAM 合計	3968 MB
CPU 数	1
CPU	Intel Xeon 3.06 GHz
合計コア数	2
合計 CPU スレッド	4
OS	Windows Server 2003 Standard Edition SP2 (32-bit)

このマシンは Linux や Web クラスターのために設計されたコンピューターです。512MB RAM、ATA-100 ドライブ、512k CPU キャッシュでシップされ、eBay で \$100 で売られています。顧客はとても高価なハイエンドサーバーに大量のメモリを搭載したと主張していました。

このレポートからはハードディスクがとても遅く、CPU は悪くはないものの速いとは言えず、最も重要なことは 32-bit OS で実行されていることです。4GB が積まれた Windows 32-bit OS では通常 3GB までしか見ることができません。1GB がシステムに使用されるので、4D は最大 2GB しか使用できません。

例2

マシン仕様

ユーザ	Adm
製造	Dell
機種	PowerEdge 2950
搭載 RAM 合計	4096 MB
CPU 数	1
CPU	Intel Xeon X5450 3.00 GHz
合計コア数	4
合計 CPU スレッド	4
OS	Windows Server 2003 Standard Edition SP2 (32-bit)

これも 4GB が搭載された 32-bit システムです。コンピューターを調べると Dell はこのマシンをデータベースサーバーに最適化し、ハイパフォーマンス、高速メモリ、そして最大 32GB の buffered RAM をサポートしていることが分かります。

顧客は高価で素晴らしいコンピューターに 32-bit OS と最低限の RAM を積んでいることになります。

なぜこのことがそんなに重要なのでしょうか。32-bit OS で 4D は最大 2GB にアクセスします。OS は残りの 1 から 1.5GB を取得しますが 2GB にはなりません。Windows はもっと必要とするかもしれません。

10 年前であれば 2GB で十分でした。4D Server v11 は 64-bit システム用に設計されています (v11 自体は 32-bit アプリケーションですが)。

64-bit システムに 8GB 以上の RAM を搭載すれば、4D は 4GB にアクセスでき、4GB が OS に残されます。4D だけがサーバー上で実行される場合、最低 8GB の RAM を搭載すべきです。

メモリについては後述します。

例3

```
Computer user: SYSTEM
Manufacturer: VMware
Computer Kind: VMware Virtual Platform
Total RAM: 3072 MB
Number of Processors: 1
Processor Name: Intel Xeon E7330
CPU Speed: 2.40 GHz
Total number of Cores: 4
Total number of CPU threads: 2
Operating System: Windows Server 2003 Standard Edition SP2
```

上図は、4D Info Report コンポーネントが収集した情報です。これはマシン仕様について収集した部分になります。

マシン仕様

ユーザ	SYSTEM
製造	VMware
機種	VMware Virtual Platform
搭載 RAM 合計	3072 MB
CPU 数	1
CPU	Intel Xeon E7330 2.40 GHz
合計コア数	4
合計 CPU スレッド	2
OS	Windows Server 2003 Standard Edition SP2 (32-bit)

この顧客は 8 コアのコンピューターに 16GB の RAM を搭載していると報告していましたが、仮想マシンを動作させていることは教えてくれませんでした。

ここからは実際に、4D Info Report コンポーネントが収集した情報を見ながら説明して聞きます。

```
===== 4D Infos =====
Application type: 4D Server
Application version: v11 SQL release 6 (F0021160)
Internal version build of 4D: 11.6 (74459)
```

ここには 2 つの重要な情報があります。4D のバージョン番号、そして内部的なビルド番号です。これにより顧客が正しいバージョンを使用しているかが分かります。

```
Data File:
"D:\Data\CRM_DATA_PROD.4DD"
  Size: 3 243 968 KB  Modif: 2010_04_09_09_28_03
Involved volume(s) list (free space):
Volume   C:\ (Operating System) 3 588 MB
Volume   D:\                 4 702 MB
```

D ドライブのデータファイルは 3.2GB と大きなものです。D には 4.7GB しか空き容量がありません。何か問題が発生し、4D の自動復旧が実行されると、ハードディスクはすぐにいっぱいになってしまいます。そしてすべてが停止します。

```
Web: (The Web Server is not started)
Count tasks: 64
Count user processes: 16
Number of users: 3
----- Database parameters -----
Database Cache: 1 699 MB
4D Server Timeout: 3 Minute(s)
4D Remote Timeout: 3 Minute(s)
Selector 28 (4D Server Log Recording): 0
Selector 34 (Debug Log Recording): 0
Selector 53 (stack size off preemptive thread): 1024 KB
Selector 54 (Idle Connections Timeout): 0 seconds.
Selector 61 (Maximum Temporary Memory Size): 0 (Max.)
----- other infos via GET CACHE STATISTICS: -----
Used cache Size : 29 777 KB
Free Memory : 3 526 996 KB
Used physical memory : 135 080 KB
Used virtual memory : 1 932 912 KB
```

この情報を記録したアプリケーションは実際使用されていたもので、ほぼ毎日クラッシュしていました。先に示した IBM のブレード上で動作し、32-bit OS で 4D Server は 2GB のメモリを使用できました。

データベースキャッシュが 1.7GB に設定されているので、残りの 300MB がカーネルに残されています。タスク数が 64 プロセスあり、プリエンプティブスレッド (selector=53) のスタックサイズは 1MB、すなわち 64MB が消費されます。4D Server には 240MB しか残されていません。

このレポートが作成されたとき仮想メモリサイズは 1.932GB で、のこり 70MB です。2000GB に達すればクラッシュします。

レポートにはより詳細な情報も含まれています。これは問題を検知するための目玉です。

これも実際のデータです。毎日クラッシュが発生していました。ここではレポート番号 1295 の後にクラッシュしています。実際この 10 分前からサーバーに接続できなくなっていました。

OS は 32-bit であり、4D は最大 2GB までアクセスできます。824MB までしか必要でないにもかかわらず 1.7GB がキャッシュに設定されていました。システムは最初から 1.933GB を使用し、作業のために 70MB しか残されていません。ユーザーが接続するたびにメモリの使用量が増えています。接続時に実行されるコードのためメモリがロックされているかりークしています。とても小さなメモリリークで通常は問題になりませんが、この場合は 70MB しか残りがなく、80 プロセス以上を開始することができませんでした。

解決のためにキャッシュサイズを 1GB に減らし、残りのメモリを 700MB にしました。さらにプリエンプティブスレッドのスタックサイズ (selector=53) を 1MB から 256KB にしました。100 プロセス必要だったので、100MB から 25MB に減ったことになります。多くないように見えますが、32-bit システムでは MB 単位は大きなものです。本当は 4GB のメモリを搭載し、64-bit Windows を使ってチューニングや最適値を探すための時間を節約し、速度を向上させたいところです。

4D Info Report コンポーネントは主に以下の情報をテキスト形式で提供するために作成されています。

- コンピューター (ハードウェア/OS)
- 4D アプリケーションのバージョン
- ホストデータベースの設定 (およびテーブル情報)

サポートされる 4D

- 4D v11 SQL release 6 以降
- 4D v12.1 (4D Server v12.1 64-bit 含む) 以降
- 4D v13 (4D Server v13 64-bit 含む)

このコンポーネントに関する情報

プラグインや他のコンポーネントには依存しません。なのでどのようなホストデータベースにもインストールできます。このコンポーネントはテーブルやレコードをホストデータベースやエクステナナルデータベースに作成したり使用したりしません。

取得する情報

このコンポーネントはレコードの内容やリソースにアクセスしたり情報を共有したりしません。ストラクチャー要素に関する基本的な情報を書き出すだけです (テーブル番号や名前、フィールドやインデックスの数、各テーブルのレコード数)。オプションの引数を使用していくつかの詳細を隠すことができます。

またコンピューター や OS、およびデータベースやアプリケーションが使用するパスも記録します。

互換性

4D_Info_Report_v3 は aa4D_Report (v2.0 以降) が作成したすべてのレポートを処理できます。

インターフェースやレポート内容でサポートされる言語

使用される 4D に基づき、英語またはフランス語 (フランス語 4D の場合) がサポートされます。

サポートされるプラットフォームと OS

4D がサポートする OS (+ Mac OS X 10.7)

4D v11 コンポーネントは PowerPC コード入りと入っていないものがあります。

命名規則

コンポーネントの名前

コンポーネントの名前は "4D_Info_Report" から始まりますが、しばしば "Information Component" と呼ばれます。名前の後にはバージョン番号 (_v3 等) が付けられます。

コンポーネントは Zip アーカイブ化され、バージョン 12 用は "_v12" のようになります。

共有メソッド名

共有メソッドは "aa4D_Report_" で始まります。

コンポーネントの共有メソッドリスト

レポートの作成

aa4D_M_CreateReport_faceless

aa4D_NP_Util_CreateReport

aa4D_NP_Util_CreateReport_Serv

aa4D_M_Report_CreateOnServerSee (4D Server 上で実行)

aa4D_NP_Shedule_Reports_Server

既存のレポートを表示

aa4D_M_Get_Last_Server_Report

ダイアログを表示

aa4D_NP_Report_Analyse_Display

aa4D_NP_Report_Compare_Display

aa4D_NP_Report_Manage_Display

ユーティリティ

aa4D_M_Get_Build_4D_Text_Call

4D Info Report コンポーネントの共有メソッドマニュアル

aa4D_M_CreateReport_faceless

aa4D_M_CreateReport_faceless ({ContentMode}; {CommentPointer})

引数 型 説明

ContentMode 倍長整数 レポートに記録するテーブル情報内容。省略時のデフォルトはすべての情報を記録。

CommentPointer ポインター レポートに含める、ホストデータベースのテキストのポインター

説明

aa4D_M_CreateReport_faceless メソッドを使用してホストデータベースのカレントプロセス内でレポートを作成できます。これは例えばコードの特定の部分でのメモリやキャッシュの利用量、あるいはループの中で使用してメモリリークに関する詳細を取得して、コードのデバッグを行うために設計されました（この目的のために ContentMode 引数を使用してテーブル情報を隠すことができます）。

aa4D_M_Get_Build_4D_Text_call

aa4D_M_Get_Build_4D_Text_call ({ContentPointer}; {PathPointer})

引数 型 説明

ContentPointer ポインター テキスト変数へのポインター

PathPointer ポインター ファイルへのフルパスを格納したテキスト変数へのポインター

説明

aa4D_M_Get_Build_4D_Text_call を使用して実行可能ファイルやプラグインのバージョンおよびビルド番号を取得できます。

メソッドはクロスプラットフォームで動作し、Mac OS 上の実行ファイルのビルド番号を取得したり、Windows 上で Mac アプリケーションの "Info.plist" を解析させることもできます（"/contents/info.plist" を含むファイルを解析する場合、このファイルをダイアログで選択します）。

引数が省略されると、ファイルを選択するためのダイアログが表示され、結果は警告ダイアログに表示されます。

第一引数にテキスト変数へのポインターを渡すと、結果はポインターが指すテキストに格納されます。

第二引数にテキスト変数へのポインターを渡すばあい、このテキストには情報を取得するファイルへの完全パス名が含まれていなければなりません。

aa4D_NP_Get_Last_Server_Report

aa4D_NP_Get_Last_Server_Report

説明

aa4D_NP_Get_Last_Server_Report コマンドを使用して、データベースの Folder_reports フォルダーに作成された最新のレポートをダイアログに表示できます。4D クライアントからも利用できます。

aa4D_M_Report_CreateOnServerSee

aa4D_M_Report_CreateOnServerSee ({ContentMode}; {CommentPointer})

引数 型 説明

ContentMode 倍長整数 レポートに記録するテーブル情報内容。省略時のデフォルトはすべての情報を記録。

CommentPointer ポインター レポートに含める、ホストデータベースのテキストのポインター

説明

aa4D_M_Report_CreateOnServerSee コマンドを使用してサーバー上にレポートを作成できます。レポートが作成されると最新のレポートがサーバーから送信され、ローカルモードでレポートを作成するときと同じダイアログにそれを表示します。取得した内容をローカルに保存することもできます。

サーバー起動後最初のレポートを作成する場合、そのレポートではなくその前のレポートが返されることがあります。理由は起動後最初のレポート作成時に追加の情報収集を行い、“Array_profiler.txt”という名称のドキュメントを作成するからです。この場合“aa4D_NP_Get_Last_Server_Report”を実行して最新のレポートを取得してください。

aa4D_NP_Report_Analyse_Display

aa4D_NP_Report_Analyse_Display ({Report Name または Full path}; {ReportContent})

引数型 説明

Report Name または Full path テキスト Define in minute(s) the delay between two reports

ReportContent テキスト Define the mode of the description of report (for tables)

説明

aa4D_NP_Report_Analyse_Display メソッドを使用して最新のレポートと過去のレポートを比較するためのダイアログを表示することができます。これにより、例えば各テーブルのレコード数を比較するなどして、どの点に変更があったのかの概要を知ることができます。

\$1 にフルパスを渡すと、レポートは直接解析され、その内容が返されます。

If you path the Report Content in \$2, the content will be parsed without opening the report file.

aa4D_NP_Report_Compare_Display

aa4D_NP_Report_Compare_Display

説明

aa4D_NP_Report_Compare_Display メソッドを使用して、ユーザー名やプロセス等の主な利用状況やメモリやキャッシュの増加状況を比較するためのリストボックスダイアログを表示します。

このダイアログから外部ファイルに結果を書き出せます。

これは外部レポートを解析するためのメインダイアログです（後述）。

aa4D_NP_Report_Manage_Display

aa4D_NP_Report_Manage_Display

説明

aa4D_NP_Report_Manage_Display メソッドを使用して、ユーザー名やプロセス等の主な利用状況やメモリやキャッシュの増加状況を比較するためのリストボックスダイアログを表示します。

このダイアログから外部ファイルに結果を書き出せます。

これは外部レポートを解析するためのメインダイアログです

aa4D_NP_Shedule_Report_Server

aa4D_NP_Shedule_Report_Server ({DelayBetween}; {ContentMode}; {CommentPointer})

引数型 説明

DelayBetween 倍長整数 2つのレポート間の間隔（分）

ContentMode 倍長整数 レポートに記録するテーブル情報内容。省略時のデフォルトはすべての情報を記録。

CommentPointer ポインター レポートに含める、ホストデータベースのテキストのポインター

説明

aa4D_NP_Schedule_Report_Server コマンドを使用してストアドプロシージャーを起動し、指定した分間隔でサーバー上にレポートを作成することができます。これにより 4D Server の利用状況を定期的に書き出すことができます。

後で (aa4D_NP_Report_Compare_Display メソッドで表示される) Compare ダイアログを使用してレポートを解析することができます。

引数を渡さない場合、デフォルトで 5 分間隔が使用されます。

ストアドプロシージャーを停止するには、第一引数に 0 を渡してこのメソッドを呼び出します。

aa4D_NP_Util_CreateReport

aa4D_NP_Util_CreateReport ({FacelessMode}; {ContentMode}; {CommentPointer})

引数 型 説明

FacelessMode 倍長整数 正数の場合、レポート作成後のダイアログを表示しない。

ContentMode 倍長整数 レポートに記録するテーブル情報内容。省略時のデフォルトはすべての情報を記録。

CommentPointer ポインター レポートに含める、ホストデータベースのテキストのポインター

説明

aa4D_NP_Util_CreateReport メソッドはこのコンポーネントのメインメソッドです。

マシン設定やデータベースの現在の状況を取得したい場合、このメソッドを実行します。

このメソッドを 4D Server 上で実行した場合、レポート作成が終了した旨のダイアログは表示されません。レポートの作成には数分かかる場合があります。

スタンドアロンで実行した場合、結果がダイアログに表示されます。このダイアログから作成されたレポートファイルにアクセスすることができます。

最初にレポートが作成されるとき:

- "Folder_Reports"という名前のフォルダーがデータファイルと同階層に作成されます (存在しない場合)。
- 複数の LAUNCH EXTERNAL PROCESS コマンドが呼び出され、コンピューターに関する情報を収集します。
- "Folder_Reports"フォルダーに"Array_Profiler.txt"ファイルを作成し、取得した値を書き込みます (4D アプリケーションを最初に起動したときのみ)。

レポートを作成したら Attention セクションを見てください。留意点が記録されているかもしれません。

aa4D_NP_Util_CreateReport_Serv

aa4D_NP_Util_CreateReport_Serv ({FacelessMode}; {ContentMode}; {CommentPointer})

引数 型 説明

FacelessMode 倍長整数 正数の場合、レポート作成後のダイアログを表示しない。

ContentMode 倍長整数 レポートに記録するテーブル情報内容。省略時のデフォルトはすべての情報を記録。

CommentPointer ポインター レポートに含める、ホストデータベースのテキストのポインター

説明

aa4D_NP_Util_CreateReport_Serv メソッドは、4D クライアントから実行されたときにサーバー上にレポートを作成することを除き、aa4D_NP_Util_CreateReport と同じです。

レポート作成は 4D Server 上で行われ、レポート作成が終了した旨のダイアログは表示されません。レポートの作成には数分かかる場合があります。

スタンドアロンで実行された場合の動作は aa4D_NP_Util_CreateReport と同じです。

レポート作成時の ContentMode 引数に関する説明

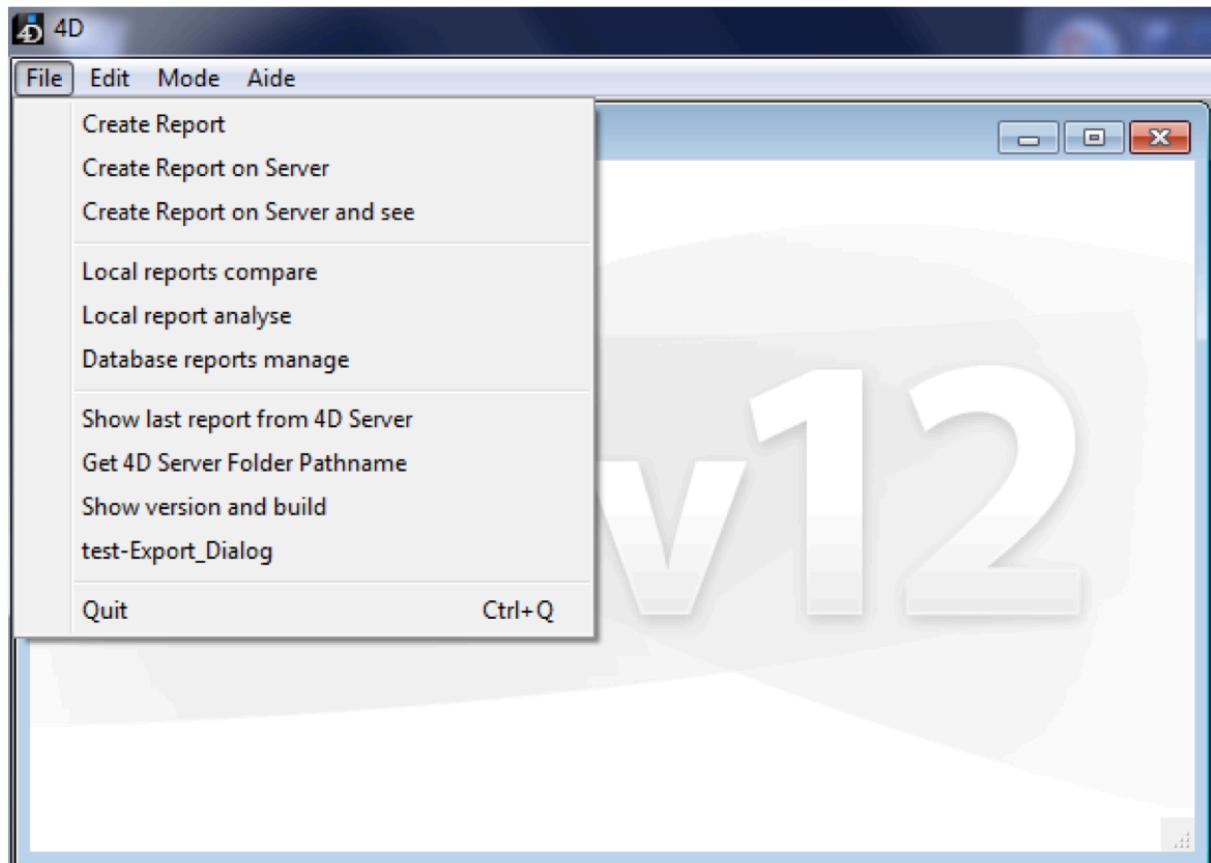
倍長整数型の ContentMode を使用してテーブル情報を含めるか、含める場合どの情報を必要とするか、設定できます。

レポート作成時の ContentMode 引数:

	-2	-1	0	1	2	3	4	5	6	7	8	9
総レコード数のカウント	<input type="radio"/>											
非表示テーブルを除く	<input type="radio"/>				<input type="radio"/>							
テーブルリストを隠す	<input type="radio"/>	<input type="radio"/>										
(引数省略時のデフォルト) すべてのテーブルリストとストラクチャーの詳細			◎									
テーブルリスト			<input type="radio"/>									
テーブル番号			<input type="radio"/>									
テーブル名			<input type="radio"/>									
テーブルのレコード数			<input type="radio"/>									
インデックスが設定されたフィールドの数			<input type="radio"/>			<input type="radio"/>						
サブテーブルの数			<input type="radio"/>			<input type="radio"/>		<input type="radio"/>			<input type="radio"/>	
トリガー			<input type="radio"/>			<input type="radio"/>	<input type="radio"/>			<input type="radio"/>		

4D Info Report コンポーネントを直接起動する

コンポーネントデータベースを直接 4D で開き、最初にデータファイルを作成してください。



コンポーネントを直接実行すれば、カスタムメニュー モードでどのように動作するか確認できます。

ファイルメニューのセクション1

レポートの作成指示。

- **Create Report** メニューを選択すると、aa4D_NP_Util_CreateReport メソッドが引数なしで呼び出されます。しばらくした後にレポートが作成され、結果がダイアログに表示されます。
- **Create Report on Server** (4D クライアント上で実行した場合) メニューを選択すると aa4D_NP_Util_CreateReport_Serv メソッドが実行され、4D Server 上にレポートが作成されます。レポート作成後のメッセージは表示されません。ローカルモードで実行した場合の動作は Create Report と同じです。
- **Create Report on Server and See** (4D クライアント上で実行した場合) メニューを選択すると、aa4D_M_Report_CreateOnServerSee メソッドが引数なしで呼び出されます。しばらくした後にレポートが作成され、結果がダイアログに表示されます。

ファイルメニューのセクション 2

Local reports compare will execute the shared method: “aa4D_NP_Report_Compare_Display” that display a new dialog (see captures in next page)

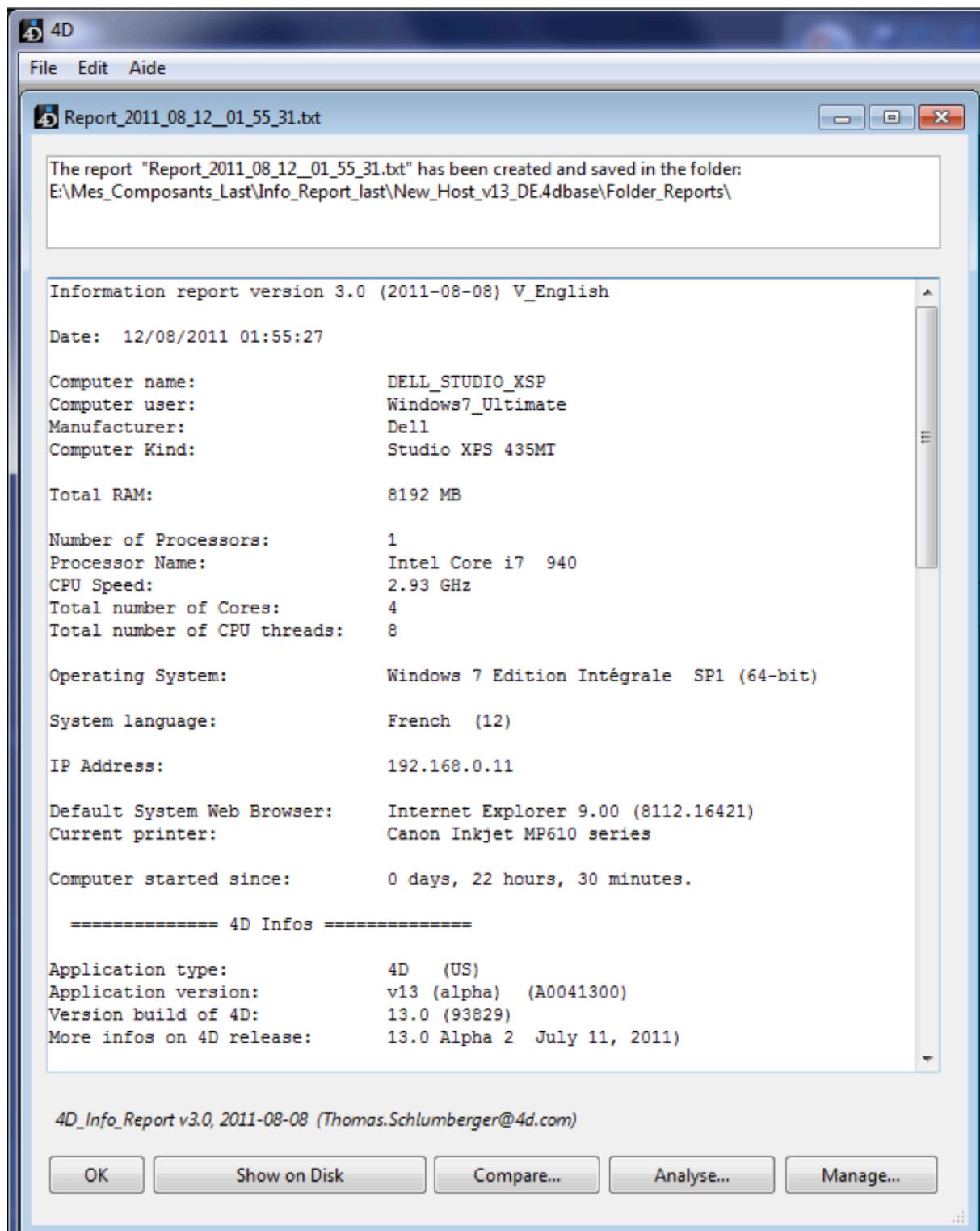
Local reports analyse will execute the shared method: “aa4D_NP_Report_Analyse_Display” that display another new dialog

Database reports manage will execute the shared method: “aa4D_NP_Report_Manage_Display” that display another new dialog

ファイルメニューのセクション 3

- **Show last report from 4D Server** メニューを選択すると aa4D_NP_Get_Last_Server_Report メソッドが実行され、4D Server 上で最後に生成されたレポートが表示されます。
- **Get 4D Server Folder Pathname** はホストデータファイルと同階層にある Folder_Reports フォルダーへのフルパスを表示します。
- **Show version and build** メニューを選択すると aa4D_M_Get_Build_4D_Text_call メソッドが実行され、表示されたダイアログで 4D アプリケーションまたはプラグインを選択すれば、以下の情報が表示されます:
 - * ファイルのバージョン
 - * ファイルのビルド番号 (存在する場合)
 - * バージョン番号とビルド番号に対応する公開バージョン
 - * 選択されたファイルのフルパス
- **test-Export_Dialog** は書き出しを行うためのダイアログを表示します。現時点ではスタンドアロンの 4D でのみ動作します。

aa4D_NP_Util_CreateReport メソッドを実行すると、数秒後にレポートが生成され、ダイアログにその内容が表示されます。



ローカルモードで「Show on Disk」ボタンをクリックすると、最新のレポートが選択された状態で「Folder_Reports」フォルダーがデスクトップ上に開かれます。

リモートモードの場合、レポートがローカルに生成されていない場合は、読み込んだレポートを保存するフォルダーを選択できます。

「Analyse」ボタンをクリックするとダイアログが表示され、2つのレポートを比較することができます (aa4D_NP_Report_Analyse_Display メソッド)。

「Manage」ボタンをクリックすると、(リモートモードでは) 4D Server 上のレコードが読み込まれ、毎 N 分ごとに新しいレポートを作成するためのストアドプロシージャーが設定されます。

Compare records

Report file name		Total Nb records	Appli Type	4D Version	Data file	
Open...	Close	Report_2011_07_06_03_14_00.txt	2 077 509	4D Server (64-bit)	12.2 (90886)	4DGMBH2.4DD
Open...	Close	Report_2011_07_13_16_03_37.txt	2 078 794	4D Server (64-bit)	13.0 (93829)	4DGMBH2.4DD
Open...	Close					

Num	Table Name	Nb Fields	Nb Index	Nb records 1	Nb records 2	Diff_rec_1_2	Diff_%_1_2
27	Logbuch	6	2	218 458	218 845	387	0,2 %
40	Kunden_Mailings	5	3	596 336	596 482	146	0,0 %
77	IT_License_Install	8	3	68 647	68 709	62	0,1 %
58	Lizenzzahlen	24	12	47 163	47 221	58	0,1 %
89	IT_LICENSE	38	6	39 572	39 630	58	0,2 %
35	Rech_Positionen	11	2	101 095	101 151	56	0,1 %
92	IT_ORDER_ARTICLE	35	2	4 429	4 477	48	1,1 %
91	Lieferscheine_Artikel	8	3	63 918	63 963	45	0,1 %
60	Aufträge_Artikel	14	2	65 149	65 191	42	0,1 %
90	IT_REGISTRATION	7	4	18 236	18 276	40	0,2 %
68	Kunden_Suchindex	2	2	88 265	88 304	39	0,0 %
31	TN_Anfragen	6	4	40 391	40 423	32	0,1 %
78	TN_Cases	4	3	1 646	1 675	29	1,8 %
6	Rechnungen	27	7	52 471	52 498	27	0,1 %
28	RegNummern	13	6	47 015	47 041	26	0,1 %
67	ServerHealth	5	2	4 227	4 251	24	0,6 %
76	IT_SEED_MACHINE	8	2	9 689	9 708	19	0,2 %
99	ServerExpansion	21	6	4 517	4 535	18	0,4 %
95	IT_QUOTE_ARTICLE	34	2	4 309	4 326	17	0,4 %
3	Lieferscheine	12	5	26 590	26 606	16	0,1 %
45	Banküberweisung	12	4	15 255	15 270	15	0,1 %

4D_Info_Report v3.0, 2011-08-31 (Thomas.Schlumberger@4d.com)

Reports management

Local folder: D:\Users\Windows7_Ultimate\Desktop\Folder_Report_test\

Show list of stored reports Select local folder

Nb. reports in the Folder_reports: 681

Nb. reports in local folder: 53

Show list of stored reports		Import Selection		Local Reports	
Report_2011_09_01_08_21_03.txt	Report_2011_09_01_06_22_36.txt	Import Selection	Import Missing	Report_2011_08_31_05_14_12.txt	Report_2011_08_31_05_13_37.txt
Report_2011_09_01_06_21_40.txt	Report_2011_09_01_06_20_43.txt	Import Selection	Import Missing	Report_2011_08_31_04_14_16.txt	Report_2011_07_31_20_34_30.txt
Report_2011_09_01_06_19_47.txt	Report_2011_09_01_06_18_29.txt	Import Selection	Import Missing	Report_2011_07_22_20_39_37.txt	Report_2011_07_12_02_52_06.txt
Report_2011_09_01_06_17_56.txt	Report_2011_09_01_06_16_45.txt	Import Selection	Import Missing	Report_2011_07_12_02_40_43.txt	Report_2011_07_12_02_20_46.txt
Report_2011_09_01_06_15_48.txt	Report_2011_08_13_05_19_14.txt	Import Selection	Import Missing	Report_2011_07_12_01_59_05.txt	Report_2011_07_12_01_30_39.txt
Report_2011_08_13_05_18_40.txt	Report_2011_08_11_10_36_56.txt	Import Selection	Import Missing	Report_2011_07_12_01_07_13.txt	Report_2011_07_12_00_51_20.txt
Report_2011_08_08_08_43_41.txt		Import Selection	Import Missing	Report_2011_07_12_00_15_13.txt	

4D_Info_Report v3.0, 2011-08-31 (Thomas.Schlumberger@4d.com)

「Compare」ボタンをクリックするとダイアログが表示され、同じフォルダー内に保存されているレポートの主な値を比較できます (aa4D_NP_Report_Compare_Display メソッド)。

Compare processes and memory

4D_Info_Report v3.0, 2011-08-08

Graph... Import... Export...

Select folder E:\Mes_Composants_Last\Info_Report_last\New_Host_v13_DE.4dbase\Folder_Reports\

		DATE-TIME REPORT:		NB USERS:	NB TASKS:	USER PROCESS:	USED CACHE (KB):	CACHE SIZE (MB):
Nb. Reports		MINIMUM:	MAXIMUM:					
102		4 juillet, 2011 09:48:54	12 août, 2011 01:55:31	1	3	2	3 736	400
							5 671	400
Ignore	ID	Report Name	DateReport	TimeReport	Users	Tasks	UsersProc	Used_Cache Cache Size
<input type="checkbox"/>	1	Report_2011_07_04_09_48_54.txt	04/07/2011	09:48:54	1	5	2	3 770 400
<input type="checkbox"/>	2	Report_2011_07_04_09_52_28.txt	04/07/2011	09:52:28	1	5	2	4 708 400
<input type="checkbox"/>	3	Report_2011_07_04_09_59_14.txt	04/07/2011	09:59:14	1	3	2	4 000 400
<input type="checkbox"/>	4	Report_2011_07_05_00_36_28.txt	05/07/2011	00:36:28	1	5	2	4 863 400
<input type="checkbox"/>	5	Report_2011_07_07_10_39_21.txt	07/07/2011	10:39:21	1	5	2	4 771 400
<input type="checkbox"/>	6	Report_2011_07_12_00_33_58.txt	12/07/2011	00:33:58	1	5	2	4 676 400
<input type="checkbox"/>	7	Report_2011_07_12_02_15_35.txt	12/07/2011	02:15:35	1	5	4	5 671 400
<input type="checkbox"/>	8	Report_2011_07_12_02_36_59.txt	12/07/2011	02:36:59	1	5	4	3 736 400
<input type="checkbox"/>	9	Report_2011_07_12_02_38_09.txt	12/07/2011	02:38:09	1	5	2	4 404 400
<input type="checkbox"/>	10	Report_2011_07_12_02_41_42.txt	12/07/2011	02:41:42	1	5	2	4 748 400
<input type="checkbox"/>	11	Report_2011_07_12_02_50_32.txt	12/07/2011	02:50:32	1	4	2	3 803 400
<input type="checkbox"/>	12	Report_2011_07_13_08_19_42.txt	13/07/2011	08:19:42	1	5	2	4 813 400
<input type="checkbox"/>	13	Report_2011_07_13_08_20_39.txt	13/07/2011	08:20:39	1	5	2	4 818 400
<input type="checkbox"/>	14	Report_2011_07_13_08_21_35.txt	13/07/2011	08:21:35	1	5	2	4 818 400
<input type="checkbox"/>	15	Report_2011_07_13_08_22_32.txt	13/07/2011	08:22:32	1	5	2	4 825 400
<input type="checkbox"/>	16	Report_2011_07_13_08_23_28.txt	13/07/2011	08:23:28	1	5	2	4 825 400
<input type="checkbox"/>	17	Report_2011_07_13_08_24_24.txt	13/07/2011	08:24:24	1	5	2	4 825 400
<input type="checkbox"/>	18	Report_2011_07_13_08_25_21.txt	13/07/2011	08:25:21	1	5	2	4 825 400
<input type="checkbox"/>	19	Report_2011_07_13_08_26_17.txt	13/07/2011	08:26:17	1	5	2	4 825 400

「Select folder」ボタンをクリックして、前回開いたフォルダーまたはレポートが格納された他のフォルダーを選択できます（例えば顧客から送られてきたレポートフォルダー）。

フォルダーが選択されるとその中のレポートが解析され、リストボックスに結果が表示されます。

「Import」ボタンをクリックすると、「Export」ボタンで作成された BLOB を読み込むことができます。その BLOB にはこのリストボックスで使用される値のみが格納されているので、より速く読み込みます。

「Export」ボタンをクリックすると、リストボックス配列の書き出しフォーマットとそのファイル名を指定できます。例えば Excel 2003 以降で開くことのできる.xml フォーマットで書き出しを行うことができます。

- リストボックスの一行をダブルクリックすると、ドキュメントが解析され、ウインドウに表示されます。
- 「Ignore」チェックボックスを選択すると、対応するレポートは MINIMUM および MAXIMUM の計算、および「Graph」ボタンで作成されるグラフから除外されます。複数の Ignore チェックボックスを同時に操作するには対象行を選択し、そのうちのいずれかの Ignore チェックボックスをクリックします。

「Graph」ボタンをクリックするとダイアログが表示され、これらの値を SVG のグラフとして表示できます。



- Draw チェックボックスの選択を切り替えると、対応する値をグラフ化するかどうかを選択できます。

グラフには 2 つの目盛りがあります。

- 左はユーザー数、タスク数、プロセス数に関連します。
- 右はメモリ使用量や使用キャッシュサイズ、キャッシュサイズ、使用 RAM、総スタックサイズに関連します。

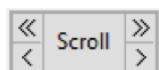
グラフ中をクリックすると一番近いレポートがハイライトされ、その値がグラフ上部に表示されます。マウスを左右に動かせば値も更新されます。このモードを終了するにはグラフの左右の外側にマウスを動かします。

もう一回クリックすると、他の部分を選択するまで選択されたレポートに固定されます。

対応するレポートをダブルクリックすると、内容が小さなウィンドウに表示されます。

MINIMUM と MAXIMUM の下にレポート名が表示されているとき、そのファイル名のボタンをクリックするとそのレポートを開くことができます。

「Scroll」の左右にあるナビゲーションボタンを使用してスクロールタイマーを設定できます。二重のボタンをクリックするとスクロールの速度が速くなります。



生成されたレポートのフォーマットと内容

- 生成されたすべてのファイルには構造化された名前が付けられます。例えばレポートが 2011 年 5 月 13 日に作成された場合、レポート名は "Report_2011_05_13_16_37_16.txt" のようになります。

すべてのレポート名は「Report_」から始まり、次に以下のタイムスタンプが付けられます:

"YYYY_MM_DD_HH_MM_SS" (Y=年, M=月, D=日, H=時, M=分, S=秒)

重要

コンポーネントを使用してあとでこのレポートを解析する場合、このファイル名は変更しないでください。

2. ドキュメントは UTF-8、改行は CRLF で書き込まれます。ファイルの先頭には 3byte の BOM が書き込まれます。

3. コンポーネントは書き込まれた内容を解析し、概要レポートを作成することができます。

A. ヘッダー: コンピューターに関する一般的な情報

```
Information report version 3.0 (2011-08-08) V_English

Date: 24/08/2011 16:37:12 // (Date format depend of the O.S. language setting)

Computer name: IP-0A6801EB
Computer user: Administrator
Manufacturer: Xen
Computer Kind: HVM domU

Total RAM: 69888 MB

Number of Processors: 2
Processor Name: Intel Xeon X5550
CPU Speed: 2.67 GHz
Total number of Cores: 4
Total number of CPU threads: 8

Operating System: Windows Server 2008 Datacenter SP2 (64-bit)

System language: French (12)

IP Address: 10.104.1.235

Default System Web Browser: Internet Explorer 8.00 (6001.18943)
Current printer: HP P3005 - Q&A (redirected 2)

Computer started since: 0 days, 8 hours, 43 minutes.
```

B. 4D アプリケーション情報

```
===== 4D Infos =====
Application type: 4D Server (64-bit) (US)
Application version: v12 release 2 (F0031220)
Version build of 4D: 12.2 (94807)
More infos on 4D release: 12.2 Hotfix 3 (August 10, 2011)
Mode: Compiled

Application:
"D:\Tests\4D\v12_2_HF3\4D Server 64-bit\4D Server.exe"

PICTURE CODEC LIST: (11)
.4pct .jpg .png .bmp .gif .tif .emf .pict .pdf .svg .wdp
```

C. データベース情報

```
===== 4D Database =====
```

```
Main UUID: 9F463893A87C478F9F3127EB3AA30DE0 (Structure)
Main UUID: 9F463893A87C478F9F3127EB3AA30DE0 (Data)

Structure File:
"D:\BenchSummit\Test64-500\Bench-64-bit-v12.4DB"
Size: 6 400 KB Modif: 2011_08_24__16_29_56

Structure File: (index)
"D:\BenchSummit\Test64-500\Bench-64-bit-v12.4DIndy"
Size: 320 KB Modif: 2011_08_24__12_39_18

Data File:
"D:\BenchSummit\Test64-500\Bench-64-bit-v12.4DD"
Size: 14 654 464 KB Modif: 2011_08_24__16_37_06

Data File: (index)
"D:\BenchSummit\Test64-500\Bench-64-bit-v12.4DIndx"
Size: 55 680 KB Modif: 2011_08_24__16_28_55

Match File:
"D:\BenchSummit\Test64-500\Bench-64-bit-v12.Match"
Size: 17 KB Modif: 2011_08_24__12_40_38

----- Plugins Bundle -----

--- Database ---
API Pack.bundle Executable Key: API Pack Version Key: 2.3.2

--- 4D Server Application ---
4D InternetCommands.bundle Executable Key: 4D InternetCommands Short Version
Key: 12.2 (94807)
4D View.bundle Executable Key: 4D View Short Version Key: 12.2 (94807)
4D Write.bundle Executable Key: 4D Write Short Version Key: 12.2 (94807)

--- (Plugin Server 64-bit) ---
4D InternetCommands.bundle

----- Plugins Bundle ----->

----- Components -----

4D SVG
4D Widgets
4D_Info_Report_v3

----- Components ----->

----- Backup infos ----->

Backup.XML location:
"\Preferences\Backup\Backup.XML"

Log File:
(No Log File found.)

Last Backup:
Last Backup Infos:
( Last Backup date & time (in Backup.XML) : 0000_01_01__00_00_00 )
( GET BACKUP INFORMATION;2 : Err.: 0 : No detected error.)
( GET BACKUP INFORMATION;0 : 0000_00_00__00_00_00 )

Next Backup:
```

```
( GET BACKUP INFORMATION;4 : 2011_08_31__03_47_37 )

Preferences Backup Advanced:

<Transaction>
WaitForEndOfTransaction: False
Timeout: 0000_00_00__00_03_00

<BackupFailure>
TryBackupAtTheNextScheduledDate: False
TryToBackupAfter: 0000_00_00__00_01_00
AbortIfBackupFail: False
RetryCountBeforeAbort: 5

<Automatic...>
AutomaticRestore: False
AutomaticLogIntegration: True
AutomaticRestart: True

----- Backup infos ----->

Involved volume(s) list (free space):

Volume C:\ (Operating System) 13 297 MB
Volume D:\ 348 809 MB

=====
----- Comment -----
(content of a pointed text variable ($3) if passed).
----- Comment ----->

Maximum number of users (license) : 1301

User list size: 2
Group list size: 0

Web: (The Web Server is not started)
Count tasks: 1248
Count user processes: 1245
Number of users: 735

----- Database parameters -----
Database Cache: 10 240 MB
4D Server Timeout: 30 Minute(s)
4D Remote Timeout: 30 Minute(s)

Selector 28 (4D Server Log Recording): 0
Selector 34 (Debug Log Recording): 0
Selector 53 (stack size of preemptive thread): 256 KB
Selector 54 (Idle Connections Timeout): 20 seconds.
Selector 61 (Maximum Temporary Memory Size): 0 (Max.)
Selector 65 (CacheLog Recording) : 1
Selector 66 (Cache Clearing size) : 1 048 576 KB
Selector 67 (Time-out forced disconnections) : 3 seconds.
Selector 68 (Log disconnections errors) : 1

----- other infos via GET CACHE STATISTICS: -----
Used cache Size : 8 172 835 KB
Free Memory : 56 092 928 KB
Used physical memory : 11 286 860 KB
Used virtual memory : 11 977 700 KB
Size of stacks : 518 525 KB

Scheduler (CPU settings):
```

```

4D Server: 0 - 8 - 0 Medium (Standard setting), recommended.

***** Attention section *****
Attention : (any Attention content, that will create an "Attention_Report.txt"
file).
Information : 4D Server (64-bit) (US) 12.2 (94807)
Information : Administration window opened on Server

```

D. テーブルに関する情報

```

===== TABLES =====
(Triggers description: N = On saving New record, E = On saving Existing record,
D = On Deleting record)

Total number of records: 58 232 680

Number of tables: 271 Number of valid tables: 271

Total number of valid fields: 1787

Num Name Nb of Records Fields (Index | SubT.) Trig

T.: 3 Cities ----- 43 607 4 ( 1 | - ) ----
T.: 2 Companies ----- 68 850 5 ( 2 | - ) ----
T.: 110 Companies0005 ----- 40 000 5 ( 0 | - ) ----
T.: 247 Companies0006 ----- 70 000 5 ( 0 | - ) ----
T.: 245 Companies0007 ----- 30 000 5 ( 0 | - ) ----
T.: 243 Companies0008 ----- 40 000 5 ( 0 | - ) ----
T.: 241 Companies0009 ----- 60 000 5 ( 0 | - ) ----
T.: 239 Companies0010 ----- 50 000 5 ( 0 | - ) ----
T.: 237 Companies0011 ----- 80 000 5 ( 0 | - ) ----

```

テーブル情報はオプションであり、書き込みを省略したり、ストラクチャー情報を隠したりできます。総レコード数は常に書き込まれます。

コンポーネントの配置

コンポーネントはストラクチャーファイルと同階層に"Components"フォルダーを作成し、その中に配置します。

このコンポーネントはホストデータベースにテーブルを作成しません。ホストデータベースが変更されることはありません。データファイルと同階層に"Folder_reports"フォルダーが作成され、レポートファイルが作成されます。

ホストデータベースからコンポーネントメソッドを呼び出す

開発中あるいはコンポーネントメソッドを呼び出すメソッド実装すれば、コンポーネントの共有メソッドを実行できます。

アプリケーションの展開を計画している場合、コンポーネントの共有メソッドを直接メソッドエディターから呼び出さないことをお勧めします。コンポーネントがインストールされていない場合にエラーとなってしまうからです。

ホストデータベースからは以下のようなコードを使用して呼び出しを行うとよいでしょう。

```

// ----- 共有メソッドの呼び出し、コンポーネント削除後もコンパイルエラーにならない ---
ARRAY TEXT($at_Components;0)
COMPONENT LIST($at_Components)
If (Find in array($at_Components;"4D_Info_Report@")>0)

```

```
EXECUTE METHOD("aa4D_NP_Shedule_Report_Server";*;5;-2)
```

```
End if
```

またインターフロセス変数を使用して、共有メソッド呼び出し毎にコンポーネントの有無をチェックしないようにすることもできます。上記のテストを起動時に一度だけ行い、例えば<>vb_Is_Component_Report_ready 変数に値を設定します。以降の呼び出しへは以下のように行えます：

```
If (<>vb_Is_Component_Report_ready)
```

```
EXECUTE METHOD("aa4D_NP_Shedule_Report_Server";*;5;-2)
```

```
End if
```

- ・"aa4D_NP"で始まるメソッドは新規プロセスで実行されるため、開発者がホストデータベースでプロセスを開始する必要はありません。またコンポーネントのいくつかのダイアログから、他の共有メソッドを直接呼び出すことも可能です。
- ・サーバー上に格納されたレポートへのアクセスや、リモートユーザーからのストアドプロシージャー設定変更権限を制限できます。これを行うには以下のコードを実装してください。

```
// ホストメソッド: aa4D_M_Host_Allow_Report_access (例題コード)
```

```
C_BLOB($1)
```

```
C_BOOLEAN($0)
```

```
C_BLOB($vxBlob)
```

```
C_BOOLEAN($Allow)
```

```
C_LONGINT($Offset)
```

```
If (Count parameters>=1)
```

```
If (Type($1)=Is BLOB )
```

```
$vxBlob:=$1
```

```
ARRAY TEXT($at_infos;0)
```

```
BLOB TO VARIABLE($vxBlob;$at_infos;$Offset)
```

```
// - ($at_infos{1}:=Current user)
```

```
// - ($at_infos{2}:=(local) I.P. address)
```

```
// - ($at_infos{3}:=Current machine)
```

```
// - ($at_infos{4}:=Current machine owner)
```

```
// --- 独自の条件に基づきアクセスを設定する ---
```

```
$Allow:=True // 条件を満たした場合
```

```
End if
```

```
End if
```

```
$0:=$Allow
```

テストする条件は開発者が独自に設定できます。上記の配列を使用する方法はあくまで例題です。

4D POP DATA ANALYZER

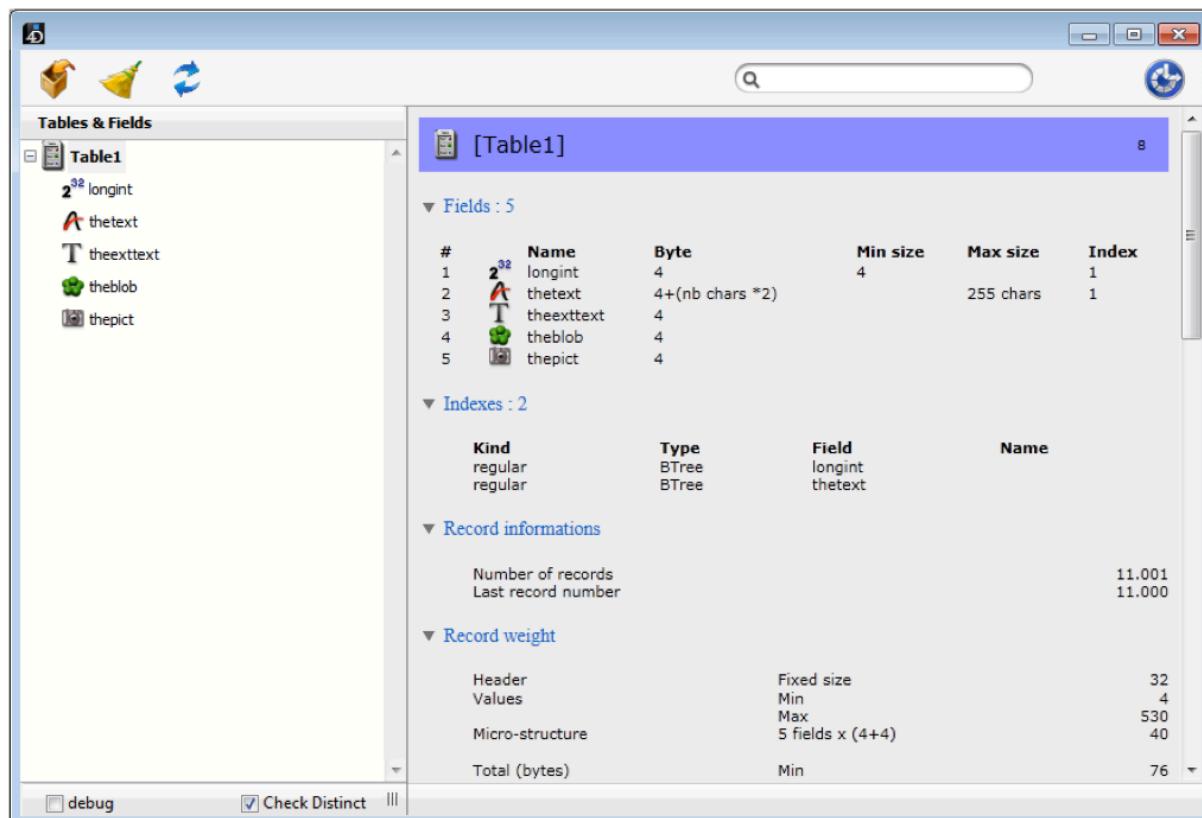
4D Pop DataAnalyzer は 4D データファイルのサイズや内容を解析するためのコンポーネントです。このツールを使用してレコードの最小/最大サイズ要件や、実際のデータに基づくサイズ要件を理解することができます。

注

コンポーネントは 4D Pop のインストールに含まれていません。

コンポーネントは v12 バージョンで配布されていますが、使用方法の節では v11、v12、v13 データファイルを使用する方法について説明しています。

概要



テーブルを選択すると基本的なデータ (テーブル名、テーブル番号、フィールドのリスト、フィールド型、ストレージサイズ、最大/最小サイズ、そしてインデックスの有無) が表示されます。

"Byte"列を見れば必要サイズを計算することができます。例えば長さ 20 に設定された文字フィールドは $4+20*2=44$ byte になります。"Min size"と"Max size"列にはそれぞれ格納に必要な最小/最大サイズが表示されます (可能な場合)。

Record information ブロックにはレコードの総数と (0 から始まる内部的な) 最大レコード番号が表示されます。10,000 レコードが存在し、最大レコード番号が 20,000 だった場合、10,000 レコードが削除されていることになります。

[Table1]					
▼ Fields : 5					
#	Name	Type	Min size	Max size	Index
1	longint	4	4		1
2	thetext	4+(nb chars *2)		255 chars	1
3	theexttext	4			
4	theblob	4			
5	thepict	4			
▼ Indexes : 2					
Kind		Type	Field	Name	
regular		BTree	longint		
regular		BTree	thetext		

Record weight ブロックではレコードが使用する論理上の最小/最大サイズを計算して表示します。必要なサイズは 3 つの部分 (Header, Micro-structure, Values) に分割されます。レコードのヘッダーサイズは 4D のバージョンにより異なる可能性がありますが、v11-v13 では 32byte です。フィールド数には左右されません。Micro-structure はレコードで使用されるフィールドを定義し、v11-v13 では 8byte * フィールド数が使用されます。Value には最小/最大サイズが計算され、表示されます。

4D はテキスト、BLOB、ピクチャーをすべて BLOB として計算し、そのサイズは 0byte から 2GB の間で可変であることを知っておいてください。Record weight ではこのサイズを計算しません。実際のデータは Table weight に含まれます。

4D は (テキストがレコード内に格納される限り) 文字とテキストフィールドを同様に扱います。結果テキストフィールドは Data Analyzer から文字として表示されるかもしれません。この場合最大サイズの値は 0 です。

上の例では 1 つのレコードが 76~602 バイトを必要とします。ハードディスクと同様 4D のデータファイルもブロック単位で管理されます。v11-v13 ではひとつのブロックサイズが 128byte です。この例のテーブルの場合、1~5 ブロック消費します。

Table weight は実際のデータファイルを解析し、結果を表示します。

Record weight sum: 選択したテーブルのレコードが消費する総バイト数。これは実際のサイズです。例えば 1 ブロックで最低 128byte が必要ですが、実際の 100byte のような値が表示されます。

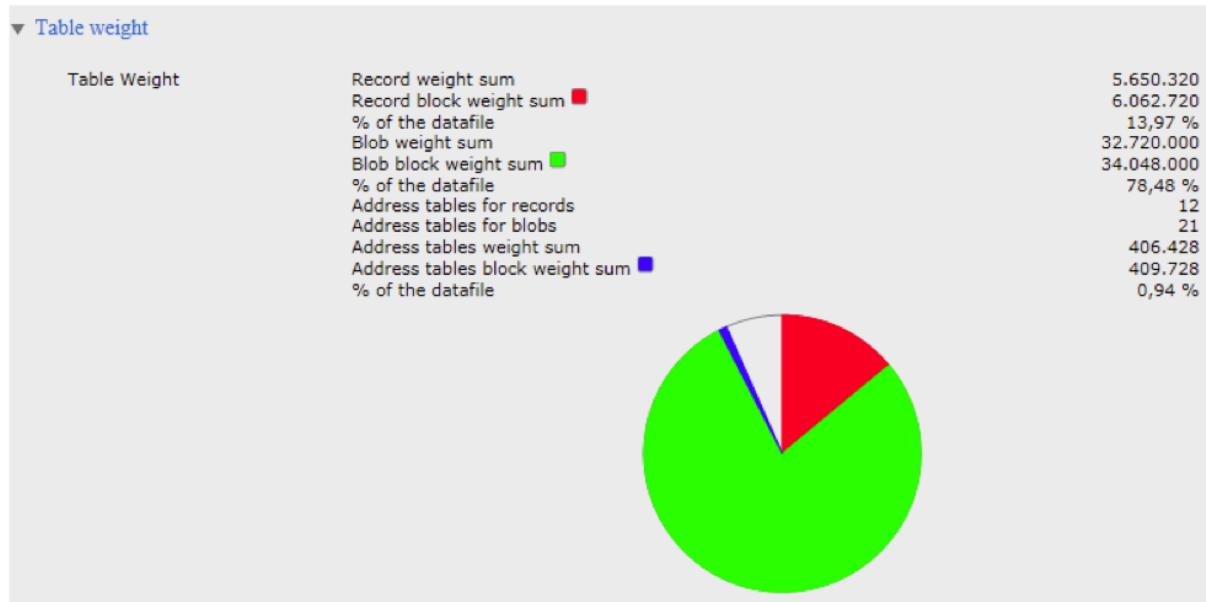
Record block weight sum: 上記と同じですが、ブロック単位でカウントを行います。これは実際にデータファイル内で使用されているサイズです。この値は下図では赤で表示されています。

% of the datafile: データファイル中でレコードブロックが占める割合です。

Blob weight sum: (レコード外に格納された) テキストや BLOB、ピクチャーが使用する総バイト数です。

Blob block weight sum: 先と同様、データファイル中で BLOB データのために使用される実際のブロックサイズです。この値は下図では緑で表示されています。

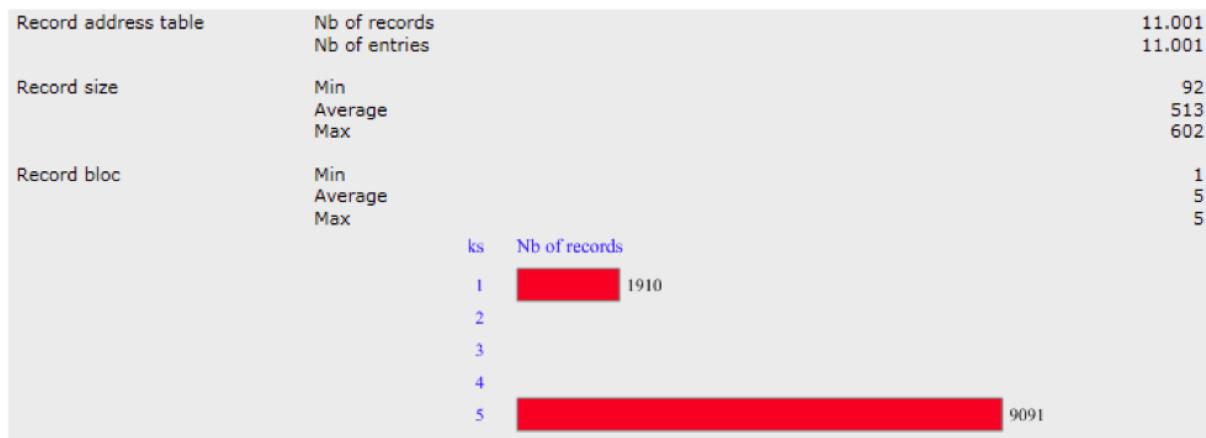
% of the datafile: データファイル中で BLOB ブロックが占める割合です。



さらにテーブルが使用するアドレステーブル(レコードおよびBLOB用)が実際のサイズとブロックサイズで計算されます。ブロックサイズは青で表示されます。

次のブロックには実際のレコードサイズに基づき最小レコード、最大レコード、平均などの統計情報が表示されます。

次の例では1910レコードが1ブロックを消費し、9091レコードが5ブロックを消費していることが分かります。2, 3, 4ブロックを使用するレコードはありません。



最後の部分ではフラグメンテーションと実際のBLOBサイズに関する情報が提供されます。

多くの開発者が、“空きスペース”はフラグメントと同じであると考えています。それは速度の低下を招きますが、ここでのフラグメンテーションとはレコード1とレコード2を読み込むときのハードディスクの機械的な移動のことです。

理想的な状態ではレコードはディスクに以下の通りに書き込まれています。

A01 - A02 - A03 - A04 - A05 - A06 - A07

MODIFY SELECTION や SELECTION TO ARRAY などにより複数のレコードを読み込む必要がある場合、4Dは自動で先読みを行い、一度に複数のレコードを読み込むとします。これは上記の例ではうまく動作し、劇的にディスクアクセスを減少させます。この例ではすべてのレコードを読み込むため、たった一度のディスクアクセスが必要です

ここで A02 レコードの "thetext" フィールドにいくつか文字を追加するとしましょう。結果レコードは 2 ブロックを消費したとします。

A01 - xxx - A03 - A04 - A05 - A06 - A07 - A02 - A02

元の位置にレコードを保存することができないため、レコードは最後に移動され、元のスペースは空きとして登録されます。4D が A01 と A02 を読み込もうとすると、一度で行うことはできません (A01 と A02 が先読みできないほど離れているとします)。

A01 が 2 ブロックを消費する程度のテキストを追加すると以下のようになります。

A01 - A01 - A03 - A04 - A05 - A06 - A07 - A02 - A02

3 ブロックの場合は以下のようになります。

xxx - xxx - A03 - A04 - A05 - A06 - A07 - A02 - A02 - A01 - A01

A01 の次に A02 を読み込もうとすると、後ろに戻らなければなりません。これは最悪のケースであり、4D は先読みを行うことが不可能になります。A01, A02 そして A03 を読み込む場合、3 回のディスクアクセスが必要です。

レポートの次の部分ではレコードや BLOB の後もどりや離れ具合を計算して表示します。

Record fragmentation	Percent	8,26 %
	Nb back jump	909
	Nb long jump	0
Blob address table	Nb of blobs	20.000
	Nb of entries	20.000
Blob size	Min	1.036
	Average	1.636
	Max	4.036
Blob bloc	Min	9
	Average	13
	Max	32
Blob fragmentation	Percent	5,00 %
	Nb back jump	1000
	Nb long jump	0

フィールドの詳細

フィールドを選択すると名前や番号などこのオブジェクトに関する詳細が表示されます。

Basics informations には以下の情報が表示されます。

Type: フィールド型

Nb distinct values: このデータファイル中での重複しない値の数。多くのレコードがあり、このフィールドがインデックス化されていない場合、この情報を取得するために時間がかかることがあります ("Check Distinct" チェックボックス参照)。

Nb values: レコードの値の数

% distinct values: 重複しない値の割合。割合が小さい場合、Bツリーではなくクラスターインデックスを選択できます。

Total size: このフィールドに必要な、データファイル中の実際の値に基づく総容量。たくさんレコードがある場合計算に時間がかかります。

Min size: 最小レコードサイズ

Average size: 平均レコードサイズ

Max size: 最大レコードサイズ

The screenshot shows the DB Browser for SQLite interface. On the left, there's a tree view labeled 'Tables & Fields' with 'Table1' expanded, showing fields: longint (2³²), thetext, theexttext, theblob, and thepict. The right panel displays detailed information for the 'longint' field under the heading '2³² [Table1]longint'. It includes sections for 'Basics informations' and 'Link informations'. The 'Basics informations' section provides statistics like Type (Long), Nb distincts values (10000), and Max size (128,92 Kb). The 'Link informations' section is currently empty.

この例題では 1 つしかレコードがないためリレーションが存在せず、Link information が空です。

次は、文字フィールドを見てみましょう。

The screenshot shows the DB Browser for SQLite interface. The left panel shows 'Table1' expanded, with 'thetext' selected. The right panel displays detailed information for the 'thetext' field under the heading 'A [Table1]thetext'. It includes a 'Basics informations' section with various statistics. For example, the 'Type' is listed as 'Alpha (255)' and 'Nb distinct values' is 3. Other statistics include 'Nb max for one value' (9100), 'Total size' (4,55 Mb), and 'Max size' (522 Bytes).

Nb distinct values を見ると 11001 レコードに対し 3 つの値しかないことが分かります。つまりほとんどのレコードで同じ値が共有されています。Nb max for one value は 9100 レコードが同じ値であることを示しています。この例題はクラスターインデックスを使用すべき完璧な例です（最初のピクチャーをみると B ツリーが使用されていることが分かります）。

LINK INFORMATIONS

フィールドにリレーションが設定されている場合、ここにリストされます。

2 nd [Invoice]ID		6 ; 7
▼ Basics informations		
Type	Long	
Nb distincts values	52407	
Nb values	52407	
% distincts values	100%	
Total size	614,14 Kb	
Min size	12 Bytes	
Average size	12 Bytes	
Max size	12 Bytes	
▼ Link informations		
<- [Invoice_Items] Invoice_ID	Distincts values 52394	Nb in table 100907 % 51,92%

使用方法

1. ストラクチャーを XML 形式で書き出します。
2. ストラクチャーとデータをコピーします。
3. v11 データベースは v12 か v13 にアップグレードします。コンポーネントは v12 で提供され、v12 と v13 で利用できます。
4. コンポーネントをインストールします。
5. DataA_Explorer_Open コンポーネントメソッドを実行します。



6. いちばん左のボタンをクリックし、ストラクチャーの XML 定義を選択します。この XML ファイルを使用してコンポーネントはデータ構造とインデックス情報を読み込みます。この操作によりホストデータベースに 5 つのテーブルが作成されます。

二番目のボタンをクリックすれば作成されたテーブルとそのレコードを削除できます。

三番目のボタンをクリックするとデータファイルが再解析されます。

データファイルが巨大な場合、重複しない値のチェックを無効にすれば解析の速度を速められます。そうする場合、ダイアログ下部にあるチェックボックスの選択を外します。



データファイルの解析は速くなりますが、重複データに関する情報が得られなくなります。

重要

ストラクチャーを変更 (テーブルやフィールド、インデックスの追加や削除、変更) した場合、二番目のボタンを使用してすべてを削除してください。そして XML 定義を書き出して、一番目のボタンで再解析を行います。

レコードを変更した場合は三番目のボタンで再解析を行うだけです。コンポーネントはレコードして格納したデータを使用して各情報の計算や表示を行います。このボタンをクリックするとまずコンポーネントが作成したすべてのデータが削除されます。

デバッグログ

これは、どこかのタイミングでクラッシュするといったケースで最初のステップとして行うべきことです。

デバッグログにはクライアントあるいはサーバー上で実行されたすべてのコマンドが記録されます。ログファイルの名前は 4DDebugLog_N.txt で Logs フォルダー内に作成されます。

```
1927 p=1 puid=23 (0) cmd: New list (DoFillHierList).
1927 p=1 puid=23 (0) cmd: APPEND TO LIST (DoFillHierList).
1927 p=1 puid=23 (0) cmd: List item position (DoFillHierList).
1927 p=1 puid=23 (0) cmd: GET LIST ITEM (DoFillHierList).
1927 p=1 puid=23 (0) cmd: SET LIST ITEM (DoFillHierList)
```

各行がピリオドで終わっている点に注目してください。コマンドが実行される前に 4D は(ピリオドを除く)行をディスクに書き込みます。コマンドが実行されるとピリオドが追加されます。つまり行がピリオドで終了していればそのコマンドの実行が成功したことを表します。行末のピリオドがない場合、アプリケーションはこのコマンドを実行中にクラッシュしたことになります。これを使用すればランダムなクラッシュの原因究明の大きな助けとなります。

またログを使用すれば何が実行されたのか、どれだけの時間がかかったのかを理解することもできます。最初の値はコマンドの経過時間をミリ秒で表します。これによりアプリケーションの一部が遅いといった問題で、いつ/どこで/なぜ遅いのかを理解する助けとなります。

デバッグログは SET DATABASE PARAMETER(Debug Log Recording;1)で有効となります。二番目の引数に 2 を使用すると、プラグインの情報もログに含まれるようになります。この場合リストには様々な追加の番号が含まれるため、読み解くのが複雑になります。

```
25755 p=9 puid=12 (1) plugInName: 4D Write; externCall: -621.
25755 p=9 puid=12 (1) plugInName: 4D Write;
cmd: _4D Write; eventCode: 43.
```

この例は 4D Write エリアをフォームに表示して使用する間に作成されたものです。リストには 2 種類のエントリ、externCall と eventCode が含まれます。

externCall はプラグインが 4D を呼び出し、4D コマンドを実行したことを示します。この例では-621 が該当します。

```
#define EX_PASTEBOARD_IS_DATA_AVAILABLE -621 //  
PA_IsPasteboardDataAvailable
```

4D Plugin API の "EntryPoints.h" ファイルには利用可能なエントリポイントが列記されています。テキスト EX_PASTEBOARD_IS_DATA_AVAILABLE は何が行われたかを簡潔に示します。詳細な説明はプラグインの API ドキュメントに記載されています。

http://sources.4d.com/trac/4d_4dpluginapi/wiki/XKCMDUS.HTM

右上の "Sources" をクリックすると、"EntryPoints.h" や "PublicTypes.h" をダウンロードでき、すべての eventCode ID を見ることができます。

```
eAE_UpdateEditCommands = 43
```

第二引数に 2 を使用すると、プラグインエリアがあるフォーム使用時にログファイルのサイズが劇的に大きくなります。プラグインに問題がないと判断される場合、このオプションを使用しないことをお勧めします。問題がプラグインに存在する可能性がある場合はこのオプションを使用することになります。

v12 では二番目の引数に 3 を使用することができ、ログ機能が劇的に向上しました。インタープリターモードで引数の詳細が記録されます。

```
2979 p=1 puid=3 (8) cmd: New list.  
2979 p=1 puid=3 (8) cmd: APPEND TO LIST(3152;"Entry";2001).  
2979 p=1 puid=3 (8) cmd: List item position(3150;2000).  
2979 p=1 puid=3 (8) cmd: GET LIST ITEM(3150;2;$EntryRef;$EntryText;$sublist2).  
2979 p=1 puid=3 (8) cmd: SET LIST ITEM(3150;2000;"Phone";2000;3152;False).
```

コンパイルモードでもスタティックな値等は記録されますが、変数名などは内部的なアドレスに置き換えられます。

```
8776 p=1 puid=23 (0) cmd: APPEND TO ARRAY(8çÿÿ;"Some text")
```

ログファイルはサイズが大きくなっていますので、ログ中で特定の個所を検索するのが難しくなることがあります。以下のコードを使用すればログファイルにカスタムコメントを残すことができます。

LOG EVENT(Into 4D Commands Log; "User selected menu Accounting/End of year statistic")

このエントリはログに記録されるので、ベンチマークなどを行う際に特定の場所を見つけやすくなります。

4D NETWORK LOG ANALYZER

ネットワーククリックログにはネットワークパケットとして送受信された詳細情報が大量に記録されます。

```
Log Session D4B4091574A8B14F8A66FE119C52DEF3  
1 nth log opened on Tuesday, August 10, 2010 04:26:45 PM  
time task id component proc request bytes in bytes out duration  
08/10/10, 16:27:37 8 'dbmg' 2 11014 25 26 0  
08/10/10, 16:27:37 8 'dbmg' 2 11059 103 2585 0  
08/10/10, 16:27:37 16 'dbmg' 5 11059 103 32115 0  
08/10/10, 16:27:37 12 'srv4' 2 86 285 30 86  
08/10/10, 16:27:37 12 'srv4' 2 3 586 22 140  
08/10/10, 16:27:37 12 'srv4' 2 87 16 10 38  
08/10/10, 16:27:37 8 'dbmg' 2 11059 103 48507 0  
08/10/10, 16:27:39 8 'dbmg' 2 11009 29 51 0  
08/10/10, 16:27:39 8 'dbmg' 2 11014 46 26 0  
08/10/10, 16:27:39 8 'dbmg' 2 11014 25 26 0  
08/10/10, 16:27:39 8 'dbmg' 2 11059 103 48507 0  
08/10/10, 16:27:40 8 'dbmg' 2 11014 25 26 0
```

このログファイルは一日記録すれば何百 MB や GB 単位になることがあります、およそ利用できないものになってしまいます。しかし特定の問題を追跡したり、どれくらいのデータがネットワーク上でやり取りされているのかを知るには重要です。

リクエストログはサーバー管理ウインドウのメンテナンスページから有効にできます。またサーバープロセスから SET DATABASE PARAMETER(4D Server Log Recording;1)を実行することもできます。ストラクチャーと同階層の"Logs"フォルダーに"4DRequestsLog_N.txt"ファイルが作成されます。

最初の列はタイムスタンプです。次にタスク ID、コンポーネント、プロセス情報インデックス (infor)、そしてリクエスト ID と続きます。

最後の 3 つの列は 4D Server が受信したリクエストのサイズ、送信したレスポンスのサイズ、そしてかかった時間です。経過時間はコンポーネントにより計測が異なり、"srv4"の場合マイクロ秒、"dbmg"の場合ミリ秒です。

リクエスト ID について理解するには対応表を参照する必要があります。

概要

4D Network Log Analyzer を使用すると、先のデータがより読みやすくなります。さらにグループ化を行うことでデータがフィルターされます。パケットは(5 分のフレームで) グループ化されるか、最も大きいか遅いパケットで表現されます。

ネットワークログを一行ずつ読むのは大変困難なので、最も大きいか遅いパケットにフォーカスし、いつ/どこから/どのような理由でパケットが送信されたかの理解を試みます。このアイデアに基づき、ログを最初から最後まで読むことはせず、特別なパケットにフォーカスします。例えば 90 分間使用され、レスポンスに問題の合ったケースでは、以下のようになります。

Time	# Packets	Total Size	Max Size	Max Answer Time	Avg Answer Time
11-04-15T11:45:	50723	70.192,8	403,6	15.319,0	5,5
11-04-15T11:50:	70102	103.058,7	403,6	3.620,0	0,8
11-04-15T11:55:	75150	140.673,6	405,8	56.176,0	4,1
11-04-15T12:00:	83454	124.243,3	403,6	2.496,0	0,7
11-04-15T12:05:	29186	42.725,8	538,5	1.840,0	0,8
11-04-15T12:50:	57791	119.400,0	8.796,3	1.513,0	0,8
11-04-15T12:55:	75708	114.560,3	818,7	827,0	0,6
11-04-15T13:00:	68102	97.890,1	405,8	2.075,0	0,5
11-04-15T13:05:	77714	112.846,6	646,6	2.340,0	0,8
11-04-15T13:10:	61246	99.609,0	1.761,4	38.767,0	75,5
11-04-15T13:15:	279421	653.157,5	4.381,9	113.491,0	1,4

パケットは 5 分ごとにグループ化され、各行には送受信したパケットの総量、サイズ、最大サイズ、最大応答時間、そして平均応答時間が表示されます(時間はミリ秒単位)。

応答に 1 秒以上かかったものまた 1MB を超えるものについては黄色で表示されます。応答に 2 秒以上かかったものまた 2MB を超えるものについては赤色で表示されます。上記の例はほとんどの行が赤で表示されていて、つまり注意が必要な部分です。

行をダブルクリックするとこの 5 分間のグループのすべてのパケットが表示されます。ヘッダーエリアを使用して経過時間やサイズなどでソートできます。

この画面では経過時間でソートを行っています。以下の情報が表示されます。

Reduce to selected process

Time	Command	Bytes In	Bytes Out	Dura...	User/Process
11-04-15T13:15:06	ExecuteQuery	81	78	113.491,0	ORCHARDDB - spool - 10.128.25.80 - "Orchard Data"
11-04-15T13:15:06	ExecuteQuery	81	78	108.108,0	ORCHARDDB - spool - 10.128.25.80 - "Orchard Data"
11-04-15T13:15:38	DataToCollection	139	251.442	25.459,0	ORCHARDDB - spool - 10.128.25.80 - "Orchard Data"
11-04-15T13:15:35	SyncThingsToForget	27	14	5.101,0	ORCHARDDB - spool - 10.128.25.80 - "Orchard Data"
11-04-15T13:15:35	SaveRecord	82	38	3.869,0	ORCHARDDB - spool - 10.128.25.80 - "Orchard Data"
11-04-15T13:15:35	SyncThingsToForget	42	1.606	3.619,0	...
11-04-15T13:17:58	SortSelection	93	6.171	3.557,0	ORCHARDDB - spool - 10.128.25.80 - "Application"
11-04-15T13:15:10	ExecuteQuery	222	5.559	2.761,0	ORCHARDDB - spool - 10.128.25.80 - "orchardDB"
11-04-15T13:15:35	ExecuteQuery	81	4.229	2.574,0	ORCHARDDB - spool - 10.128.25.80 - "Workstation Service"
11-04-15T13:15:25	ExecuteQuery	134	4.326	2.028,0	ORCHARDDB - spool - 10.128.25.80 - "LabelPrinting Application"
11-04-15T13:15:25	ExecuteQuery	148	4.600	1.935,0	ORCHARDDB - spool - 10.128.25.80 - "orchardDB"
11-04-15T13:15:25	SyncThingsToForget	27	14	1.872,0	ORCHARDDB - spool - 10.128.25.80 - "Orchard Data"
11-04-15T13:15:25	ExecuteQuery	81	4.226	1.841,0	ORCHARDDB - spool - 10.128.25.80 - "orchardDB"
11-04-15T13:15:25	ExecuteQuery	130	4.302	1.732,0	ORCHARDDB - spool - 10.128.25.80 - "LabelPrinting Application"
11-04-15T13:18:02	ExecuteQuery	85	668	1.482,0	ORCHARDDB - spool - 10.128.25.80 - "Application"
11-04-15T13:16:54	ExecuteQuery	226	5.448	1.451,0	ORCHARDDB - spool - 10.128.25.80 - "orchardDB"
11-04-15T13:15:25	ExecuteQuery	110	4.596	1.404,0	ORCHARDDB - spool - 10.128.25.80 - "Orchard Data"
11-04-15T13:15:25	ExecuteQuery	145	4.184	1.404,0	ORCHARDDB - spool - 10.128.25.80 - "orchardDB"
11-04-15T13:15:25	SaveRecord	1.641	38	1.295,0	ORCHARDDB - spool - 10.128.25.80 - "Orchard Data"
11-04-15T13:15:35	ExecuteQuery	81	4.229	1.170,0	ORCHARDDB - spool - 10.128.25.80 - "Workstation Service"
11-04-15T13:16:38	DataToCollection	274	408.336	1.107,0	ORCHARDDB - spool - 10.128.25.80 - "Team QC"
11-04-15T13:15:24	ExecuteQuery	145	88	936,0	ORCHARDDB - spool - 10.128.25.80 - "Orchard Data"
11-04-15T13:15:27	DataToCollection	186	116	889,0	ORCHARDDB - spool - 10.128.25.80 - "LabelPrinting Application"
11-04-15T13:15:38	ExecuteQuery	130	4.328	873,0	ORCHARDDB - spool - 10.128.25.80 - "LabelPrinting Application"
11-04-15T13:15:38	SaveRecord	580	38	827,9	ORCHARDDB - spool - 10.128.25.80 - "orchardDB"

- サーバーのタイムスタンプ
- リクエストの名前 (後述)
- 受信バイト
- 送信バイト
- 経過時間 (ミリ秒)
- ユーザー/プロセス (コンピューターナー名 - ユーザー名 - IP - プロセス名)

このリストから、次に何をすべきかが見えてきます。この例では 110 秒ほど使用する 2 つのクエリがあります。まずこの点をチェックすべきでしょう。大量のレコードがあるテーブルに対してシーケンシャルクエリを行ったか、遅いハードディスクを使用したか、あるいはほかの理由かもしれません。3 番目に大きなジョブは DataToCollection です。SELECTION TO ARRAY やリストボックス、MODIFY SELECTION などにより 25 秒、251KB のデータが必要だったことが分かります。この行により、サーバーに問題があることが分かります。250KB のデータを取得するために 25 秒要したということはハードディスクに問題があるか、たまにしか発生しないのであればキャッシュに問題があるのかもしれません。これらは遅いクエリの理由にもなります。

パケットをダブルクリックするとこの特定のユーザー/プロセスだけが表示されます。

Time	Command	Bytes In	Bytes Out	Dura...	User/Process
11-04-15T13:12:42	ExecuteQuery	81	4.933	967,0 D	"sa"/"sa@saens"
11-04-15T13:13:01	DataToCollection	184	289.218	19.469,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	ExecuteQuery	96	6.200	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	ExecuteQuery	102	4.234	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	LoadRecord	54	88	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	DataToCollection	97	50	15,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	ExecuteQuery	122	47	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	ConnectRecord	41	30	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	ConnectRecord	41	30	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	SaveRecord	141	38	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	SyncThingsToForget	42	14	31,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	SaveRecord	599	38	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	ExecuteQuery	81	5.479	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	ConnectRecord	41	30	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	SaveRecord	471	38	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	SyncThingsToForget	42	14	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	SyncThingsToForget	42	14	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	SyncThingsToForget	42	14	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	ExecuteQuery	147	5.133	47,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	SaveRecord	85	38	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	LoadRecord	54	429	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	SaveRecord	85	38	0,0 D	"sa"/"sa@saens"
11-04-15T13:13:12	ExecuteQuery	135	4.168	0,0 D	"sa"/"sa@saens"
11-04-15T13:15:06	ExecuteQuery	81	78	113.491,0 D	"sa"/"sa@saens"
11-04-15T13:15:06	ExecuteQuery	81	71	0,0 D	"sa"/"sa@saens"

リストは自動でスクロールされ、パケットが選択されます。他のパケットも遅いものやサイズが大きいものは色が付けられます。

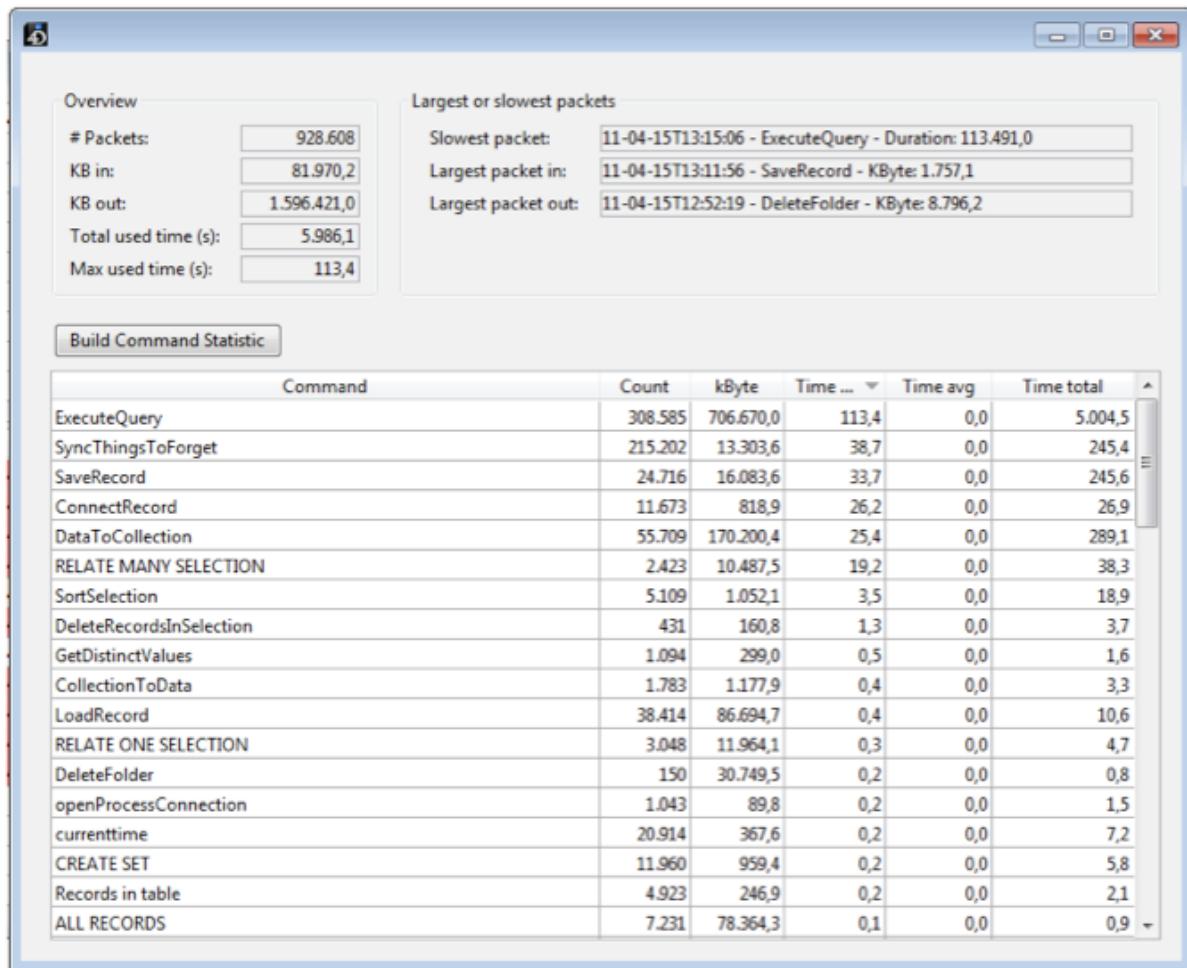
遅いパケットの前にあるコマンド（とプロセス名）が、何が行われたのかのヒントをくれるかもしれません。この例では 13:13:12 の同じタイムスタンプに複数のコマンドがあります。クエリ、クエリ、そしてレコードロードが行われています。DataToCollection は SELECTION TO ARRAY その他同様な処理を指します。そしてまたクエリが行われ、レコード保存、もう一度レコード保存、クエリ、レコード保存、クエリ、レコード保存、レコードロード、レコード保存、そして 3 回のクエリがあり、二回目は 113 秒かかっています。

大きなストラクチャーではこれらの情報から直接どのメソッドが実行されたかを探すのは困難です。しかしプロセス名や時刻、ユーザー名などの情報もあわせて、どのコードに問題があるかをつきとめます。

他のログファイルを合わせて使用し、より多くの情報を得ることもできます。クライアントのデバッグログを出力するのもよいでしょう。今回のケースではバックアップログも助けになるはずです。後ほどバックアップのログを読むためのツールも紹介します。ユーザー名から特定された 13:13:12 のログをチェックすると、5 つのレコード保存エントリを見つけることができました。どのテーブルのレコードが変更され、その内容が何であったかが分かります。これにより問題のメソッドをより容易に探すことができるでしょう。

統計

メインダイアログで "Statistics" ボタンをクリックすると、以下の画面が表示されます。



パッケージ数、総送受信 KB 数、総経過時間および最大経過時間など、ネットワークログ全体の概要が表示されます。

右側には最も遅いパケットと最も大きいパケットが表示されます。

"Build Command Statistic"ボタンをクリックすると、使用された各コマンドのサマリが作成されます。このサマリの作成には時間がかかります。そのコマンドが何回使用されたか、あるいはサイズ、時間などでソートすることができます。

この統計は、致命的な部分を見つけ、アプリケーションを最適化することを助けます。このログファイルが作成されたアプリケーションでは1日に20,914回もCurrent time(*)が実行されていました。ALL RECORD は7,231回実行され、78MBがネットワークを流れました。このアプリケーションの最も遅いパケットは5000秒中113秒を使用するクエリなので、まずこのコマンドから始めることになります。

NETWORK LOG ANALYZER を使用してログの作成/読み込みを行う

ネットワークログはサーバー管理ウィンドウのメンテナンスページで"リクエストのログを開始"ボタンをクリックして開始できます。または以下のコードを使用します。

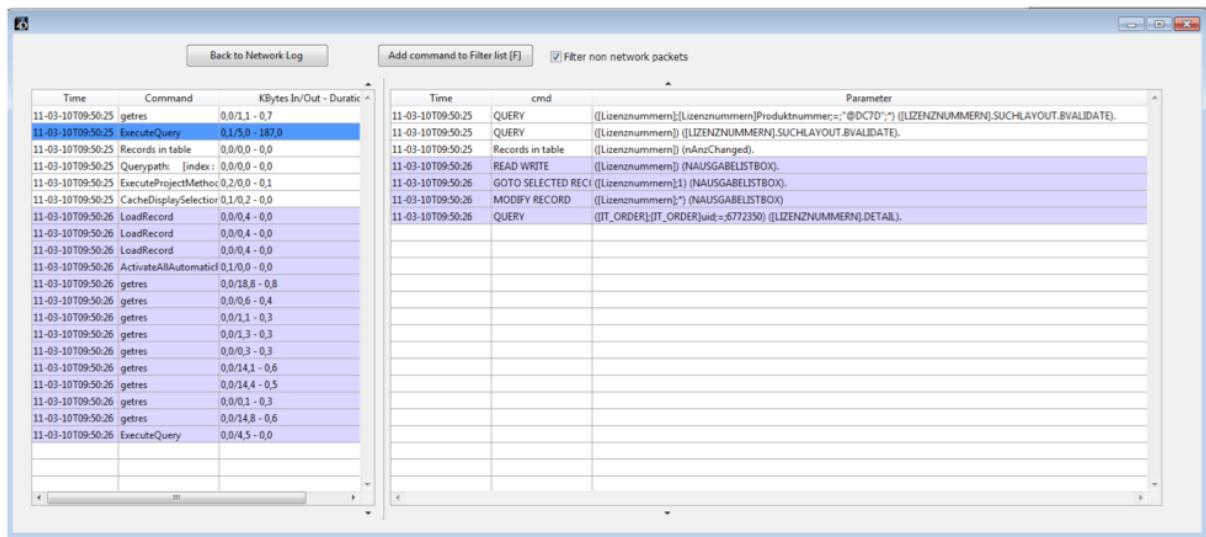
SET DATABASE PARAMETER(4D Server Log Recording;1)

4D Serverは4DRequestsLog_X.txtという名前のファイルを作成します。

作成したファイルを Network Log Analyzer で読み込むことができます。これには時間がかかります。セッション毎に新しいでたファイルを作成することをお勧めします。

上級編 - 4D DEBUG LOG とともに使用する

ネットワークログとデバッグログをともに使用すると、より詳細な情報を取得できます。次の図のように、4D Server 上で実行されたコマンドのサイズや時間とともに、このコマンドを呼び出した 4D 側のコマンドを並べてみることができます。



残念なことにこのアイデアにはいくつかの問題があります。

まずデバッグログはクライアント側に書かれ、ネットワークログはサーバー上で書かれます。二番目に 4D コマンドログとネットワークパケットを結び付けるリンクは存在しません。QUERY, QUERY BY FORMULA, QUERY SELECTION などはすべてネットワークコマンド "ExecuteQuery" を使用します。どの 4D コマンドかまでは分かりません。さらに ExecuteQuery は複数のコマンドの集合である場合があります。ExecuteQuery は UNLOAD RECORD, QUERY…, LOAD RECORD の組み合わせであり、このすべてのコマンドに対し一度だけネットワークパケットが送信されます。

NEXT RECORD, LAST RECORD, GOTO RECORD などはすべてネットワークコマンド LoadRecord にリンクされます。ほかにも同様の例があります。

Substring のようなコマンドはネットワークパケットを発生させません。Current time などは引数に応じて変わります。

READ WRITE, READ ONLY, START TRANSACTION などのコマンドは、データに関連するにもかかわらず、ネットワークに何も送信しません。このようなコマンドは実行時即座のアクションを必要としません。例えば READ WRITE は次のレコード読み込み時に影響を与えます。4D は自動的にコマンドの実行をキャッシュし、次のネットワークパケットに乗せて送信します。

このような状況のためやや複雑になりますが、それでもサーバーとクライアント両方でログを作成することができます。LOG EVENT コマンド等を使用してエントリを挿入し、同期の助けとすることができます。両マシンでは時刻設定が異なるかもしれない点に留意してください。クライアントログはミリ秒単位で記録され、サーバーログは秒単位です。なので時間差があるかもしれません。それでも動作が遅いなどといった問題のためにはこれらのログが必要となります。このテクニックは 1 秒に 1000 パケットを送信するような状況では無意味です。そのようなケースでは最適化の必要がありません。コマンドの実行に 2 秒や 50 秒かかるといった場合に問題となります。

どのサーバープロセスに属しているのかを知るために、クライアントログを識別する必要があります。これは両方のログにユーザー名と現在時刻を挿入することで行っています。このコードは On Startup と、新規プロセスが開始されるときに実行される必要があります。

Network Log Analyzer にはこの作業のために使用できるメソッドがいくつかあります。このメソッドをコピーするか、Network Log Analyzer をコンポーネントとして使用できます。

On Startup やすべてのプロセス開始時に Debuglog_Sync メソッドを実行します。メソッドをアプリケーションにコピーする場合、DebugLog_LogOnServer メソッドには”サーバー上で実行”属性が必要である点に注意してください。

複数のクライアントのデバッグログと、ネットワークログを読み込んだら、パケットを選択して”Sync with Debug Log”ボタンをクリックすると、先のような画面が表示されます。

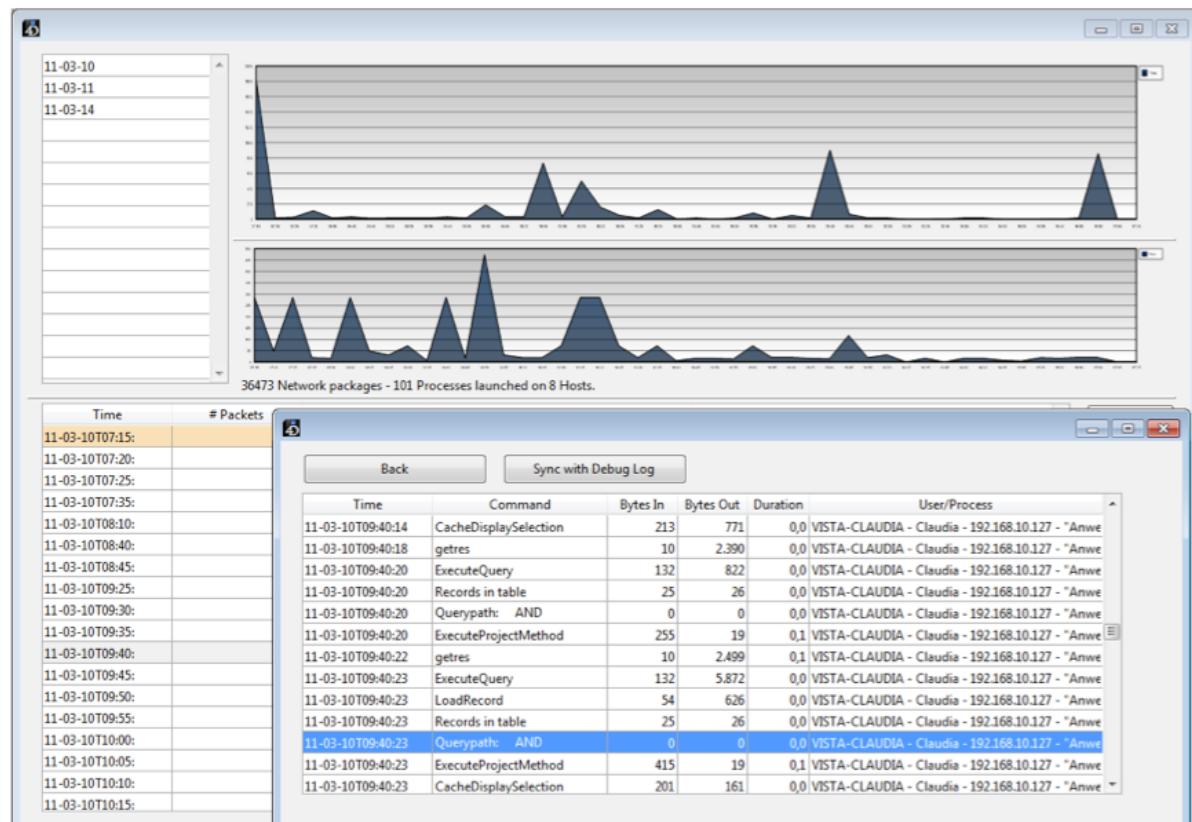
この画面では両ログが対応するようスクロールを試みます。選択されたパケットのデバッグログとネットワークログ両方を見るすることができます。前後の秒のパケットはピンクで表示されます。より古いもしくは新しいログを見る必要がある場合、リストボックスの上下にある矢印ボタンをクリックします。

Substring や Position など、ネットワークパケットを発生させないコマンドは、デフォルトですべて取り除かれます。“Filter non Network packets”のチェックボックスを外すとそれらを表示できます。

”Add command to Filter list”ボタンをクリックすると、4D コマンドをフィルターリストに追加できます。これは新しいバージョンに追加された新規コマンド等で必要になるかもしれません。

上級編 - クエリパスを含める

4D の GET LAST QUERY PATH コマンドは重要であり、遅いクエリの解析に役立ちます。Network Log Analyzer を使用すればオフラインでサーバーの振る舞いをチェックできるので、ログの中にクエリパスを含めておけばより詳細な検証が可能です。



上のログには"Querypath:"エントリがあります。ダブルクリックすればフルパスが表示されます:

```
Querypath: AND
  [index : Aufräge.LNr ] LIKE 0  (434 records found in 0 ms)
  [index : Aufräge.Nur Angebot ] LIKE false  (1 record found in 0 ms)
-> 1 record found in 0 ms
```

自動ではクエリパスは作成されませんし、ログに含まれることもありません。これは開発者が行う必要があります。通常アプリケーション内では何百・何千ものクエリコマンドが発行されているので、これらすべてにクエリパスハンドラーを追加するのには時間がかかります。先に示したデバッグログとネットワークログの組み合わせを使用すれば、クエリパスハンドラーを追加する場所を限定できます。デバッグログをオプション 3 で取得すれば **QUERY([table];[table]field="hello")** のような引数はすでにログに含まれています。カスタムクエリダイアログを作成した場合も、行われるクエリは引数付きクエリなのでデバッグログに残ります。残るは 4D 標準のクエリエディターです。この場合デバッグログ内にクエリコマンドが記録されますが、ユーザーが入力した値を見ることはできません。なのでなぜクエリが遅いのかが分かりません。

標準のクエリダイアログを開くこのようなクエリの個所を特定する必要があります。そしてクエリの前後でクエリパスの記録を有効にします。例えば以下のコードになるでしょう。

```
DebugLog_BeforeQuery (<>CreateDebugLog)
// ログを取る場合 True をそうでなければ False を渡す
QUERY([table])
DebugLog_AfterQuery(<>CreateDebugLog)
```

上級編 - クライアントコンピューターのデバッグログを収集する

サーバーとクライアントで取られたログを比較することについてお話ししましたが、各クライアント上で収集したログを一か所に集めることについてお話ししていませんでした。

コンポーネントには On Startup で呼び出すメソッドが含まれていて、このメソッドはログを圧縮し、ファイルサーバーにアップロードしてくれます。そしてクライアント上のファイルを削除します。

```
DebugLog_SaveLogs (logOption;compressOption;uploadOption;uploadPath)
```

logOption 引数は SET DATABASE PARAMETER(53;x)と同じもので、x=0 ならログを記録しない、1 は通常のログ、2 はプラグインのログも収集、3 は引数も記録するです。このツールで使用する場合 3 を選択してください。ただしアプリケーションでまだサブテーブルが使用されている場合は別です。その場合は 1 を使用してください。オプション 3 はサブテーブルと互換性がなく、クラッシュすることができます。

compressOption 引数は v12 以降で使用でき、ログファイルを圧縮するために PHP を使用します。0 はログを圧縮せずにサブフォルダーに移動します（名前はタイムスタンプを使用）。1 は PHP を使用してサブフォルダーを圧縮します。2 はサブフォルダーを圧縮して元のファイルを削除します。

uploadOption 引数を使用してファイルサーバー（マウントされているかゲストアクセスが必要）や FTP サーバー（4D Internet Commands が必要）にファイルをアップロードできます。0 はアップロードなし、1 は FTP にアップロード、2 は COPY DOCUMENT を使用してファイルサーバーにアップロードです。

uploadPath 引数でアップロードパスを指定します。ファイルサーバーの場合有効なパス名を渡します。

Windows の場合 "YYYYserver\folder\folder"。FTP の場合は

"ftp://user:password@ftp.example.com/path/folder/" のようになります。

圧縮をしてファイルサーバーにアップロードする例:

DebugLog_SaveLogs (3;2;2;"\$\$\$\$NAS\$\$Archive\$\$Logs")

ネットワークパケット名

Network Log Analyzer は自動でパケット ID の読みやすい名前を表示します。すべてのネットワークパケットが 4D コマンドとの 1 対 1 リレーションを持っているわけではないので、いくつかの名前はコマンドと同じ (ALL RECORDS など)、いくつかは機能と似ています (ORDER BY と ORDER BY FORMULA は両方とも SortSelection) ですし、名前のいくつかは何が行われたのか推測が可能 (RebuildIndexByRef、CreateFullTextIndexOnOneField、CancelShutdown 等) です。

Network Log Analyzer には[Log_Commands]テーブルがあり、各 ID に読みやすい名前を割り当てます。名前は 4D のエンジニアチームが内部的に使用します。ALL RECORDS や CREATE SET などのように 1 対 1 の関係がある場合には 4D の名前で置き換えます。置き替えのリストは[Command_Network]テーブルにあります。新しくデータファイルを作成したときはこれらのテーブルが自動的に¥Resources¥Cmd_network.txt and ¥Resources¥Cmd_table.txt の内容に基づいて埋められます。

クライアントの起動時には多くの"getres"と"DeleteFolder"パケットを見ることになります。時にはサイズが大きく時間がかかることもあります。これはストラクチャーやプラグイン、リソースの転送を行うためです。この転送は一度だけ行われ、クライアント側にキャッシュされます。

リソースフォルダーのファイルの配置方法に注意してください。ファイルが更新されると再圧縮と再転送が必要になります。トップレベルのフォルダーに何千ものファイルを入れるようなことは避けてください。ファイルをサブフォルダーに移動してください。またサブフォルダー内のファイルをなるべく更新しないようにしてください。これを行うとサブフォルダー全体が圧縮され送信されます。

4D ログファイル (.Journal) - LogTool

4D のバックアップとともに使用される.journal ログファイルはレコードの書き込みや削除だけしか記録されていませんが、それでもデバッグや問題の理解の助けになります。

Maintenance and security center

アクティビティ分析

以下のリストは、最後のバックアップ以降ログファイルに記録された操作を表示します。

操作#	アクション	テーブル	レコード/BLOB	プロセス	サイズ	日付	時間	ユーザー
52637	コンテキストの…			21578		2012-01-21	05:33:11	Administrator
52638	更新	Counter	1435	21578	36	2012-01-21	05:33:11	Administrator
52639	コンテキストの…			21578		2012-01-21	05:33:12	Administrator
52640	コンテキストの…			21579		2012-01-21	05:33:31	Administrator
52641	更新	Counter	3148	21579	36	2012-01-21	05:33:31	Administrator
52642	コンテキストの…			21579		2012-01-21	05:33:31	Administrator
52643	コンテキストの…			21580		2012-01-21	05:36:50	Administrator
52644	更新	Counter	686	21580	36	2012-01-21	05:36:50	Administrator
52645	コンテキストの…			21580		2012-01-21	05:36:50	Administrator
52646	コンテキストの…			21581		2012-01-21	05:40:02	Administrator
52647	更新	Counter	1027	21581	34	2012-01-21	05:40:02	Administrator
52648	コンテキストの…			21581		2012-01-21	05:40:02	Administrator
52649	コンテキストの…			21582		2012-01-21	05:40:15	Administrator
52650	更新	Counter	490	21582	42	2012-01-21	05:40:15	Administrator
52651	コンテキストの…			21582		2012-01-21	05:40:15	Administrator
52652	コンテキストの…			21584		2012-01-21	05:47:10	Administrator
52653	更新	Counter	929	21584	40	2012-01-21	05:47:10	Administrator
52654	コンテキストの…			21584		2012-01-21	05:47:11	Administrator
52655	コンテキストの…			21585		2012-01-21	05:50:05	Administrator
52656	更新	Counter	9154	21585	36	2012-01-21	05:50:05	Administrator

特定のフィールドを表示するには、カラムヘッダーを右クリックします。

解析 **選択…** **書き出し…**

上の図は、MSC のアクトビティ解析で.journal ログファイルを表示した所です。この画面から、データを書き出して MS Excel 等で開きチェックすることができます。

多くのレコード書き込みが行われる環境ではログファイルが大きくなることがあります。問題の個所が複数のログファイルにわたることもありますし、開くのが簡単でない場合もあります。

Business Brothers Inc. (<http://bbsp.com/>) の LogTools を使用すれば v2004-v12 のバックアップログを開くことができます。

メインのダイアログでは MSC に似たすべてのトランザクションが表示されます。

Log View

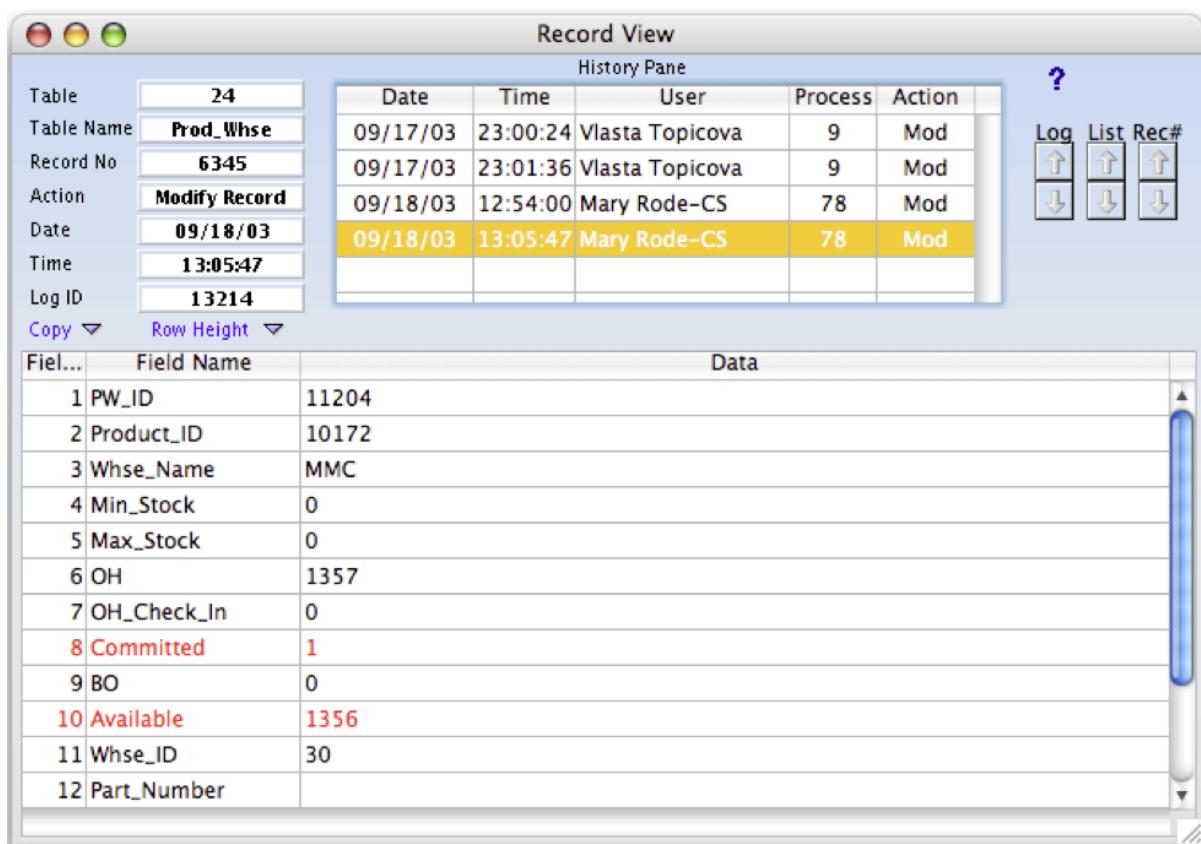
Log ID	Date	Time	Table Name	Type	Rec No	User ID	Process No	Trans ID	Tran Resolution
13210	09/18/03	13:05:29	Order	Mod	110260	test4	88	0	
13211	09/18/03	13:05:35	Inven_Draw	Add	768666	Mary Rode-CS	78	0	
13212	09/18/03	13:05:36	Inven_Draw	Mod	768666	Mary Rode-CS	78	0	
13213	09/18/03	13:05:36	MISC	Mod	49	Mary Rode-CS	78	0	
13214	09/18/03	13:05:47	Prod_Whse	Mod	6345	Mary Rode-CS	78	0	
13215	09/18/03	13:05:47	Product	Mod	8642	Mary Rode-CS	78	0	
13216	09/18/03	13:06:14	Inven_Draw	Add	768667	Mary Rode-CS	78	0	
13217	09/18/03	13:06:14	Inven_Draw	Mod	768667	Mary Rode-CS	78	0	
13218	09/18/03	13:06:14	MISC	Mod	49	Mary Rode-CS	78	0	
13219	09/18/03	13:06:17	{Start Transaction}	STR	-1	TED CORBO	106	337	VLD
13220	09/18/03	13:06:17	RETURN	Mod	12906	TED CORBO	106	337	VLD
13221	09/18/03	13:06:17	{Validate Transaction}	VTR	-1	TED CORBO	106	337	VLD
13222	09/18/03	13:06:17	RETURN	Mod	12906	TED CORBO	106	0	
13223	09/18/03	13:06:17	RETURN	Mod	12906	TED CORBO	106	0	
13224	09/18/03	13:06:26	Prod_Whse	Mod	6932	Mary Rode-CS	78	0	
13225	09/18/03	13:06:26	Product	Mod	10497	Mary Rode-CS	78	0	
13226	09/18/03	13:06:34	Order	Add	110286	Martina Zwick	22	0	
13227	09/18/03	13:06:37	MISC	Mod	750	Martina Zwick	22	0	
13228	09/18/03	13:06:37	MISC	Mod	711	Martina Zwick	22	0	
13229	09/18/03	13:06:37	Order	Mod	110286	Martina Zwick	22	0	
13230	09/18/03	13:06:38	MISC	Mod	43	Martina Zwick	22	0	
13231	09/18/03	13:06:38	Client	Mod	78	Martina Zwick	99	0	

これをテーブル、時間、ユーザーなどでフィルターすることができます。

Table View: Inven_Add

Table Inven_Add												Copy & Export	Select	Filter	0/778
Date	Time	Acti...	Rec Num	User ID	Pr...	Err	Addition_ID	Addition_Date	Quantity	Consumed	Remainder	Cost_Per	DMN_ID		
09/18/03	15:49:11	Add	18000000	Jessica Docken	85	~	424216	09/18/03	2	0	2	683.57	C	▲	
09/18/03	15:49:11	Mod	329923	Jessica Docken	85	~	424216	09/18/03	2	0	2	683.57	C		
09/18/03	15:49:12	Mod	329923	Jessica Docken	85	~	424216	09/18/03	2	0	2	683.57	C		
09/18/03	15:49:45	Add	18000000	Scott Beattie	33	~	424223	09/18/03	5	0	5	669	C		
09/18/03	15:49:45	Mod	329924	Scott Beattie	33	~	424223	09/18/03	5	0	5	669	C		
09/18/03	15:49:45	Mod	329924	Scott Beattie	33	~	424223	09/18/03	5	0	5	669	C		
09/18/03	15:50:03	Add	18000000	Jessica Docken	85	~	424224	09/18/03	1	0	1	103.18	C		
09/18/03	15:50:04	Mod	329925	Jessica Docken	85	~	424224	09/18/03	1	0	1	103.18	C		
09/18/03	15:50:05	Mod	329925	Jessica Docken	85	~	424224	09/18/03	1	0	1	103.18	C		
09/18/03	15:51:26	Add	18000000	Jessica Docken	85	~	424225	09/18/03	33	0	33	102.8	C		
09/18/03	15:51:26	Mod	329926	Jessica Docken	85	~	424225	09/18/03	33	0	33	102.8	C		
09/18/03	15:51:27	Mod	329926	Jessica Docken	85	~	424225	09/18/03	33	0	33	102.8	C		
09/18/03	15:52:43	Add	18000000	Jessica Docken	85	~	424226	09/18/03	25	0	25	89.09	C		
09/18/03	15:52:44	Mod	329927	Jessica Docken	85	~	424226	09/18/03	25	0	25	89.09	C		
09/18/03	15:52:44	Mod	329927	Jessica Docken	85	~	424226	09/18/03	25	0	25	89.09	C		
09/18/03	15:53:27	Add	18000000	Jessica Docken	85	~	424227	09/18/03	4	0	4	115.14	C	▼	

レコードの履歴を見ることが有益な場合もあります。



これによりアプリケーションのデータが失われた、値が予期せず変更されたなどの問題個所を見つけることができます。

レコードが失われるというケースがありました。顧客によればあるテーブルのレコードが突然すべて失われることでした。

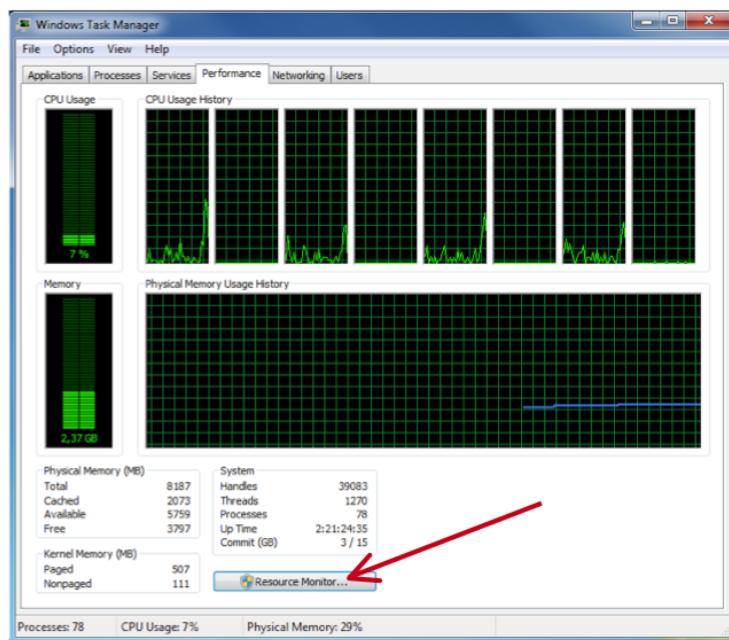
LogTool を使用してログファイルを検証したところ、このテーブルに対する TRUNCATE TABLE エントリを見つけました。顧客によればこのコマンドは一切使用していないことです。4D はテーブルレコード全体に対する DELETE SELECTION が指示されると、可能であれば内部的にこのコマンドを呼び出すことがあります（実行速度が速いため）。しかしそれも行っていないとのことでした。

同じユーザーの TRUNCATE TABLE 直前の処理を検証したところ、何が行ったのか分かりました。検索ダイアログがあり、そこで検索されたレコードを削除する機能がありました。ユーザーが何も入力しない場合もダイアログが受け入れられ、デベロッパーは値を検証せずに "@" を追加して検索していました。結果すべてのレコードが選択され、DELETE SELECTION が実行されました。

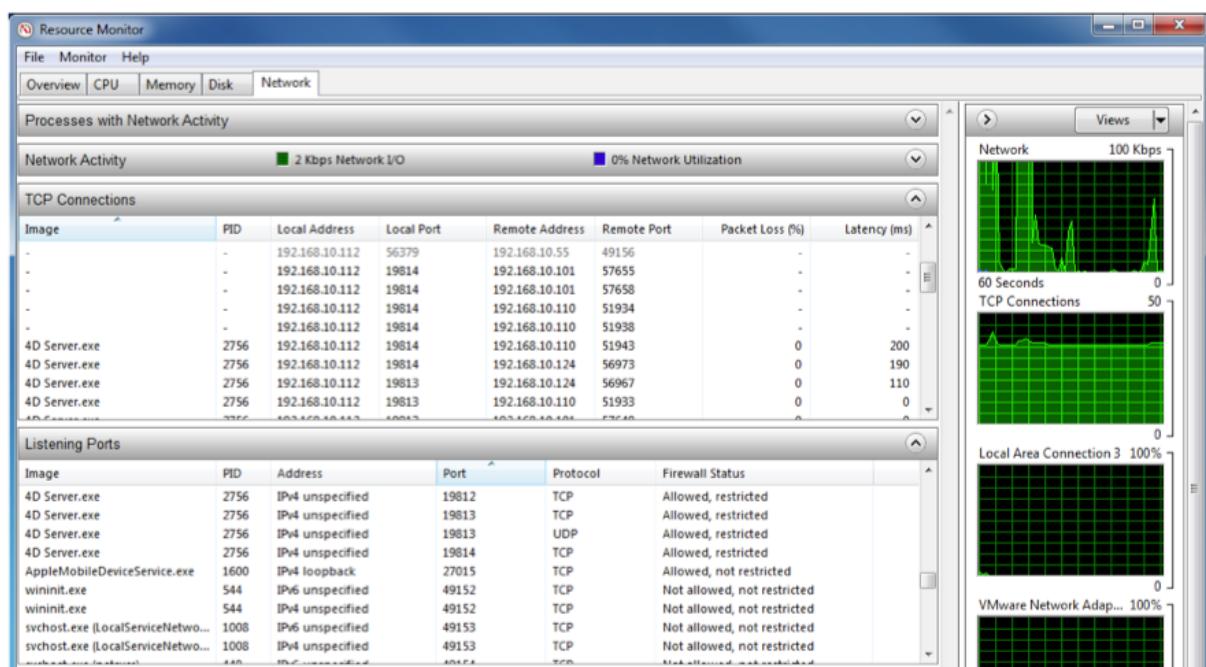
当初解決が困難と思われたケースでしたが、一時間で解決できました。

WINDOWS リソースモニター

Windows のタスクマネージャーは、ご存じと思います。CPU 使用率やメモリや仮想メモリの使用状況をモニターできるツールです。Windows 7 や Windows Server 2008 Enterprise Edition ではタスクマネージャーを拡張したリソースモニターを利用できます。

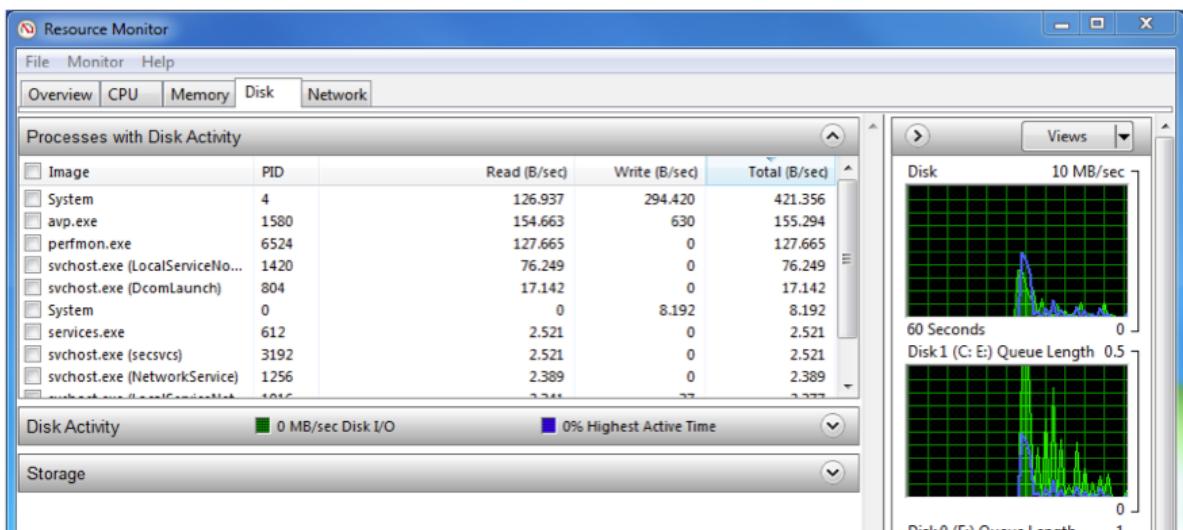


リソースモニターではCPU、メモリ、ディスク、ネットワークに関するより詳細な情報を見ることができます。



このツールを使用してどのポートが使用されているか、またそのポートに対するWindowsファイアーウォールの設定も見ることができます。また4D Serverと4Dクライアント間のようなすべての開かれた接続のステータス情報(パケットロスや潜在時間)なども見られます。

ディスクページではディスクのロードだけでなく、物理ハードディスクごとにグループ化されたキューリングの長さも見ることができ、ボトルネックの検知に役立ちます。



ディスクページで最も注目すべきは物理ディスクごとに表示される”アクティビティな時間の最高”と”キューの長さ”です。アクティビティが10%前後であったり、キューの長さが0.2未満であれば問題ありません。キューの長さが数秒間0.5や1になるようであれば、サーバーに問題があります。ディスクが実行できる以上にジョブがあり、新しいジョブは待ちキューに入れられます。フラッシュマネージャーがキャッシングをディスクに書き込んでいる間にシーケンシャルサーチやレコードの読み込みを行うなど、4D Serverがデータを読み込む必要があるとき、読み込みは待つ必要があります、すべてが遅くなります。

この問題に対する昔の解決法はオンボードのキャッシングメモリを持ったRAIDコントローラーを使用することでした。今日ではSSDテクノロジの使用をお勧めします。秒あたりのI/O(IOPS)が格段に向上します。普通のSATAディスクは約100IOPS、高価なSASは約200IOPSであるのに対し、古いSSDでも5,000-10,000IOPS、2011年の新しいSSDは約50,000IOPSです。特別なサーバーSSDはさらに500,000であり、100万というのも発表されています。

このダイアログからディスクがボトルネックになっているのかどうかを判定します。社内でSATAからSSDに切り替えたところ、キューの長さが0.5から0.01に減少しました。

ネットワークページはルーターやスイッチの問題を見つけるのに役立ちます。ここにはすべての4D Server接続がリストされます。4D Serverに1つの4Dクライアントを接続するところから始めます。クライアント側で二番目のプロセスを起動すると19814ポートが一行追加されます。クライアント側で何もしないまましばらくすると、その行が消えます(タイムアウトの設定に応じて4Dがそのポートをスリープさせます)。クライアント側で作業を行うとそのポートが再び現れます。現場にネットワークの問題がある場合、この画面と、ファイアウォール/ゲートウェイ/スイッチのモニターウィンドウを比較すると問題点が見つかるかもしれません。

この画面を使用して4D Serverのネットワーク利用方法のふつうの状態を知っておくとよいでしょう。

Mac OS Xではアクティビティモニターを使用して同じような情報を得られます。4D Serverを選択し、”詳細を表示”をクリックすると、”開いているファイルとポート”ページに情報を表示できます。

コマンドコンソール(WindowsのDOS)やターミナル(Mac OS X)でもチェックできます。
<http://kb.4d.com/search/assetid=76029>

