記事
Tomohiro Iwamoto · 2020年6月5日 27m read

InterSystemsデータプラットフォームとパフォーマンス-パート4: メモリの確認

この記事では、InterSystemsデータプラットフォームで実行するデータベースアプリケーションにおけるグローバルバッファ、ルーチンバッファ、gmheap、locksizeなどの共有メモリ要件のサイジングアプローチを説明し、サーバー構成時およびCachéアプリケーションの仮想化時に検討すべきパフォーマンスのヒントをいくつか紹介します。 これまでと同じように、Cachéについて話すときは、すべてのデータプラットフォーム（Ensemble、Health Share、iKnow、Caché）を指しています。

このシリーズの他の記事のリストはこちら

最初にCachéを使い始めたころ、ほとんどの顧客オペレーティングシステムは32ビットで、Cachéアプリケーションのメモリは少なく、高価なものでした。 一般的に展開されていたIntelサーバーにはコアが数個しかなく、スケールアップするには、大型のサーバーを使用するか、ECPを使って水平方向にスケールアウトするしかありませんでした。 今では、基本製品グレードのサーバーでさえも、マルチプロセッサ、数10個のコア、TBにまで増設できる可能性も備わった128GBまたは256GBの最低メモリが搭載されています。 ほとんどのデータベースインストールでは、ECPは忘れられ、単一サーバー上でアプリケーショントランザクションレートを大幅に拡張できるようになりました。

Cachéの主要機能は、共有メモリでデータを使用する方法で、通常データベースキャッシュまたはグローバルバッファと呼ばれています。 手短に言えば、適切なサイズにして「より多くの」メモリをグローバルバッファに割り当てることができれば、通常はシステムパフォーマンスを向上させることができます。メモリ内のデータには、ディスク上のデータよりもはるかに高速にアクセスできるからです。 32ビットシステムが主流であったころ、「グローバルバッファにはどれくらいのメモリを割り当てればよいのか」という問いへの回答は簡単でした。「できるだけ多く！」という答えです。割り当てられる量も特に多かったわけではないため、OS要件、OSとCachéプロセスの数とサイズ、およびそれぞれが使用する実メモリを計算して残りを割り出し、できるだけ多くのグローバルバッファを割り当てるために、総計の計算が入念に行われていました。

潮流の変化

最新世代のサーバーでアプリケーションを実行しているのであれば、Cachéインスタンスに大量のメモリを割り当てることができ、「安価」なメモリが豊富にあるため自由放任な姿勢で臨むことができます。 ところが、潮の流れは再び変わり、現在では非常に大きなシステムを除き、デプロイされているほぼすべてのシステムが仮想化されています。 そのため、「モンスター」VMのメモリフットプリントは必要に応じて大きくできるにしろ、システムの適正化が再びスポットライトを浴びているのです。 サーバーの一元化を最大限に活用するには、利用可能なホストメモリを最大限に活用するキャパシティプランニングが必要となります。

メモリを使用するもの

Cachéデータベースサーバーの主なメモリ消費者は、大まかに4つあります。

- オペレーティングシステム(ファイルシステムキャッシュを含む)。
- Caché以外のアプリケーション(インストールされている場合)。
- Cachéプロセス。
- Caché共有メモリ(グローバルバッファ、ルーチンバッファ、GMHEAPを含む)。

必要な物理メモリ量は、おおまかにリストの各項目の要件を単純に合計することで得られます。 上記はすべて実

メモリを使用しますが、仮想メモリを使用することもできます。キャパシティプランニングでは、ページングが発生しない、もしくは最小化されるように、またはメモリをディスクから戻す必要のあるハードページフォールトの発生を緩和または失くすために、十分な物理メモリを得られるようにシステムのサイズを決定することが重要となります。

この記事では、Caché共有メモリのサイジングと、メモリパフォーマンスを最適化するためのいくつかの一般的な規則について説明します。 オペレーティングシステムとカーネルの要件はオペレーティングシステムごとに異なりますが、ほとんどの場合数GBです。
ファイルシステムキャッシュは多様で、リスト内の他の項目に割り当てられた後に使用可能な量となります。

Cachéは主にプロセスです。アプリケーションの実行中のオペレーティングシステムの統計には、Cacheプロセス(cacheまたはcache.exe)が表示されます。 そのため、アプリケーションのメモリ要件を確認するには、オペレーティングシステムのメトリックを調べるのが単純な方法と言えます。 たとえば、Linuxの vmstat または ps コマンド、またはWindows Process Exploreで、使用中の実メモリ量を合計し、増加とピーク時の要件を推定することができます。 一部のメトリックは共有メモリを含めた仮想メモリを表示するため、それに注意して実メモリ要件を収集してください。

グローバルバッファのサイジング - 単純な方法

高トランザクションデータベースの場合、キャパシティ計画の目標の1つは、アプリケーションのできる限り多くのワーキングセットがメモリ内に存在できるようにグローバルバッファを適正化することです。 こうすることで、読み取りIOPSが最小化されるため、全体的にアプリケーションのパフォーマンスが向上することになります。 また、オペレーティングシステムやCachéシステムといったほかのメモリ利用者がページアウトされることなく、ファイルシステムキャッシュのメモリが十分であるように、バランスを取ることも必要です。

このシリーズのパート2
では、ディスクからの読み取りが過剰になると何が起きるかという例を紹介しました。あの場合は、大量の読み取りは、不良なレポートまたはクエリが原因で発生しましたが、グローバルバッファが小さすぎでアプリケーションがディスクからデータブロックを読み取り続けなければならない場合にも同じような効果が見られます。 補足として、ストレージの状況が常に変化していることにも注目する価値があります。ストレージはSSDの進歩により加速化が進んでいますが、メモリ内のデータが実行中のプロセスに近いことが依然として最適です。

もちろん、すべてのアプリケーションは異なるため、
「有用性は異なる場合がある」
と言っておくことが重要ですが、アプリケーションに合った共有メモリのキャパシティプランニングに着手するには、一般的な規則があります。 それに従った後で、特定の要件に合わせて調整することができます。

どこから始める？

残念ながら、その問いを魔法のように解決する答えはありません。しかし、前の記事で話したように、必要なピーク時トランザクションレートについて、ピーク処理時間のCPUの使用率が約80%になるようにシステムのCPUキャパシティを調整することが適当な実践と言えます。 アクティビティにおける短期的な増加または予期しない急増に備えて、20%のヘッドルームを残しておくということです。

たとえば、私がTrakCareシステムのサイジングを行っている場合は、顧客サイトのメトリックのベンチマークと確認から、トランザクションレートのCPU要件がわかっているため、Intelプロセッサベースのサーバーに対して幅広い経験則を使用できます。

経験則: 物理メモリのサイズは、CPUコア当たり n ギガバイトで決定する。

- TrakCareデータベースサーバーの場合、n は8 GBです。 より規模の小さなWebサーバーの場合は、4 GBです。

経験則: Cachéグローバルバッファには、n%のメモリを割り当てる。

- 小規模から中規模のTrakCareシステムの場合、n%は60%で、オペレーティングシステム、ファイルシステ

ムキャッシュ、およびCachéプロセスに40%を残します。 ファイルシステムキャッシュがたくさん必要な場合やプロセス数が多い場合には、50%などに変更できます。
または、大規模なシステムで非常に大量のメモリ構成を使用する場合は、%を高く指定します。
- この経験則では、サーバー上のCachéインスタンスが1つしかないことを想定しています。

そのため、たとえばアプリケーションに10個のCPUコアが必要な場合、VMのメモリは80 GBで、グローバルバッファに48 GB、その他すべてに32 GBが割り当てられることになります。

メモリサイジングの規則は、物理システムまたは仮想システムに適用されるため、TrakCare VMにも同じ1 vCPU : 8 GBのメモリ比率が適用されます。

グローバルバッファの調整

サイジングがどれほど効果的かを確認するために注目すべきことがいくつかあります。
オペレーティングシステムツールを使用して、Caché以外の空きメモリを観察できます。 最良の計算に従ってセットアップし、経時的なメモリ使用量を確認します。常に空きメモリがある場合、グローバルバッファを増やすかVMを適正化してシステムを再構成することができます。

グローバルバッファの適切なサイジングのもう1つの重要な指標は、読み取りIOPSをできる限り低くすることです。Cachéキャッシュの効率性が高くなります。 mgstatを使用して、PhyRdsとRdRatioに対するさまざまなグローバルバッファサイズの影響を観察できます。これらのメトリックの見方の例はこのシリーズのパート2で説明しています。メモリにデータベースが丸ごと格納されていない限り、ディスクからの読み取りは必ず発生するため、この目的は、読み取りをできるだけ低く抑えるところにあります。

ハードウェアの ” 食品群 ” とそのバランスを正しく取ることに注意してください。グローバルバッファのメモリが多くなるほど読み取りIOPSは低くなりますが、システムが短時間でより多くの処理を実行できるようになるため、CPU使用率は増加する可能性があります。ただし、IOPSを低くすることはほぼ必ず良い結果を生み出し、顧客により速い応答時間を提供することができます。

物理メモリ構成への要件の適用については、以下のセクションを参照してください。

仮想サーバーの場合は、本番VMメモリ、特にCaché共有メモリをオーバーサブスクライブしないように計画してください。これについては以下でも説明します。

あなたのアプリケーションのスイートスポットは、CPUコア当たり8 GBの物理メモリですか？
私にはわかりませんが、あなたのアプリケーションでも同様の方法が使用できるかを確認してください。
コア当たり4 GBであろうが10 GBであろうが関係ありません。 グローバルバッファのサイズを適正化するにあたりほかの方法を見つけた方は、ぜひ下にコメントを残してください。

グローバルバッファの使用状況の監視

Cachéユーティリティ「^GLOBUFF」は、グローバルバッファがある時点で何をしているのかに関する統計を表示します。 たとえば、上位25をパーセントで表示するには、次のように記述します。

```
do display^GLOBUFF(25)
```

これは、次のように出力されます。

```
Total buffers: 2560000    Buffers in use: 2559981  PPG buffers: 1121 (0.044%)

Item  Global                                Database          Percentage (Count)
1     MyGlobal                              BUILD-MYDB1        29.283 (749651)
2     MyGlobal2                             BUILD-MYDB2        23.925 (612478)
3     CacheTemp.xxData                      CACHETEMP          19.974 (511335)
```

```
4     RTx                                BUILD-MYDB2        10.364 (265309)
5     TMP.CachedObjectD                  CACHETEMP          2.268 (58073)
6     TMP                                CACHETEMP          2.152 (55102)
7     RFRED                              BUILD-RB           2.087 (53428)
8     PANOTFRED                          BUILD-MYDB2        1.993 (51024)
9     PAPi                               BUILD-MYDB2        1.770 (45310)
10    HIT                                BUILD-MYDB2        1.396 (35727)
11    AHOMER                             BUILD-MYDB1        1.287 (32946)
12    IN                                 BUILD-DATA         0.803 (20550)
13    HIS                                BUILD-DATA         0.732 (18729)
14    FIRST                              BUILD-MYDB1        0.561 (14362)
15    GAMEi                              BUILD-DATA         0.264 (6748)
16    OF                                 BUILD-DATA         0.161 (4111)
17    HISLast                            BUILD-FROGS        0.102 (2616)
18    %Season                            CACHE              0.101 (2588)
19    WooHoo                             BUILD-DATA         0.101 (2573)
20    BLAHi                              BUILD-GECKOS       0.091 (2329)
21    CTPCP                              BUILD-DATA         0.059 (1505)
22    BLAHi                              BUILD-DATA         0.049 (1259)
23    Unknown                            CACHETEMP          0.048 (1222)
24    COD                                BUILD-DATA         0.047 (1192)
25    TMP.CachedObjectI                  CACHETEMP          0.032 (808)
```

これは、たとえばワーキングセットのどれくらいがメモリに保持されているのかを確認する場合など、さまざまな点で役立ちます。 このユーティリティが便利だと思った方は、役に立った理由を下のコメント欄でほかのコミュニティユーザーに共有してください。

ルーチンバッファのサイジング

アプリケーションが実行しているコンパイルされたクラスなどのルーチンは、ルーチンバッファに格納されます。ルーチンバッファの共有メモリをサイジングする目的は、すべてのルーチンコードをロードし、ルーチンバッファに常駐させることにあります。
グローバルバッファと同じように、ディスクからルーチンを読み取るのは高くつき、非効率です。
ルーチンバッファの最大サイズは1023 MBです。 ルーチンをキャッシュすることでパフォーマンスは確実に大幅に向上するため、原則として、必要以上のバッファを設定するとよいでしょう。

ルーチンバッファはさまざまなサイズで構成されています。 デフォルトでは、Cachéが各サイズのバッファ数を決定します。2016.1のインストール時のデフォルトは、4、16、および64 KBです。 さまざまなサイズに対するメモリの割り当てを変更することはできますが、キャパシティプランニングに着手するには、特別な変更理由がない限り、Cachéのデフォルトを使用することをお勧めします。
詳細については、Cachéドキュメンテーションを参照してください。『Caché Parameter File Reference（Cachéパラメータファイルリファレンス）』の場合は、付録「config」の「routines（ルーチン）」、『Caché System Administration Guide（Cachéシステム管理ガイド）』の場合は、「Configuring Caché（Cachéの構成）」の章にある「Memory and Startup Settings（メモリと起動の設定）」で説明されています。

アプリケーションが実行されると、ルーチンはディスクからロードされ、そのルーチンが入る最も小さなバッファに格納されます。 たとえば、ルーチンが3 KBの場合、理想的には4 KBのバッファに格納され、4 KBのバッファがない場合は、それより大きなバッファが使用されます。 32 KBより大きなルーチンの場合は、64 KBルーチンバッファが必要な数だけ使用されます。

ルーチンバッファの使用状況の確認

mgstatメトリック RouLas

ルーチンバッファの大きさが十分であるかを理解するには、mgstatメトリックのRouLas（ルーチンのロードと保存）を使用する方法があります。 RouLasはディスクからのフェッチまたはディスクへの保存です。 ルーチンのロ

ード/保存の数が多いと、パフォーマンスの問題が現れる場合があります。その場合は、ルーチンバッファを増やすことでパフォーマンスを改善でできます。

cstat

ルーチンバッファを最大の1023
MBに増やしてもRouLasが高
くなってしまう場合は、より詳細な調査を行うために、cstat
コマンドを使ってどのルーチンがバッファ内にあり、どのくらいが使用されているのかを確認することができます
。

```
ccontrol stat cache -R1
```

これを実行すると、ルーチンバッファとキャッシュ内のすべてのルーチンのリストを含む、ルーチンメトリックのリストが生成されます。
たとえば、デフォルトのCachéインストールの部分リストは、次のように生成されます。

```
Number of rtn buf: 4 KB-> 9600, 16 KB-> 7200, 64 KB-> 2400,
gmaxrouvec (cache rtns/proc): 4 KB-> 276, 16 KB-> 276, 64 KB-> 276,
gmaxinitalrouvec: 4 KB-> 276, 16 KB-> 276, 64 KB-> 276,

Dumping Routine Buffer Pool Currently Inuse
 hash   buf  size sys sfn inuse old type   rcrc     rtime   rver rctentry rouname
   22: 8937  4096   0   1     1   0  D  6adcb49e  56e34d34    53 dcc5d477  %CSP.UI.Po
rtal.ECP.0
   36: 9374  4096   0   1     1   0  M  5c384cae  56e34d88    13 908224b5  %SYSTEM.Wo
rkMgr.1
   37: 9375  4096   0   1     1   0  D  a4d44485  56e34d88    22 91404e82  %SYSTEM.Wo
rkMgr.0
   44: 9455  4096   0   0     1   0  D  9976745d  56e34ca0    57 9699a880  SYS.Monito
r.Health.x
 2691:16802 16384   0   0     7   0  P  da8d596f  56e34c80    27 383da785  START
   etc
   etc
```

上記の2行目の「rtns/proc」は、276個のルーチンがデフォルトの各バッファーサイズにキャッシュできることを示しています。

この情報を使用したルーチンバッファのサイジングには、アプリケーションを実行して、実行中のルーチンをcstat -R1で一覧表示する方法があります。 一覧表示した後、使用中のルーチンのサイズを計算します。たとえば、このリストをExcelに入力し、サイズ順に並べ替えて一体どのルーチンが使用中であるのかを確認できます。 各サイズのすべてのバッファを使用していない場合は、ルーチンバッファが十分に用意されていることになりますが、すべてを使用している場合は、ルーチンバッファを増やす必要があるか、各バケットサイズの数の構成をより積極的に行うことができます。

ロックテーブルのサイズ

locksiz構成パラメータは、異なるプロセスがデータの特定の要素を同時に変更するのを防止する並行処理制御のロックを管理するために割り当てられるメモリのサイズ(バイト単位)です。 内部的には、メモリ内のロックテーブルには現在のロックと、それらのロックを保持するプロセスに関する情報が含まれています。

ロックの割り当てに使用されるメモリはGMHEAPから取得されるため、GMHEAPに存在する以上のメモリをロックに使用することはできません。 locksizのサイズを増やす場合は、下のGMHEAPセクションに記載された式に従って、GMHEAPを対応するサイズに増加してください。 アプリケーションによるロックテーブルの使用に関する情報は、システム管理ポータル(SMP)を使用して監視できます。また、APIを使うとより積極的に監視できます。

```
set x=##class(SYS.Lock).GetLockSpaceInfo()
```

このAPIは、「空き領域、使用可能領域、使用中領域」の3つの値を返します。 適切な値を大まかに計算するには、「使用可能領域」と「使用中領域」を確認します(一部のロック領域はロック構造に予約されています)。
詳細については、Cachéドキュメンテーションを参照してください。

注意: locksizの設定を編集すると、変更内容はすぐに反映されます。

GMHEAP

GMHEAP(汎用メモリヒープ)構成パラメータは、Cachéの汎用メモリヒープのサイズ(キロバイト単位)として定義されています。 これが、ロックテーブル、NLSテーブル、およびPIDテーブルにも割り当てられる割当量です。

注意: GMHEAPを変更するには、Cachéを再起動する必要があります。

アプリケーションに合ったサイジングを行うには、APIを使ってGMHEAPの使用状況に関する情報を確認することができます。

```
%SYSTEM.Config.SharedMemoryHeap
```

このAPIには、使用可能な汎用メモリヒープを取得する機能があり、構成用のGMHEAPパラメータを推奨することができます。
たとえば、DisplayUsage
メソッドは、各システムコンポーネントが使用するすべてのメモリと使用可能なヒープメモリの量を表示します。
詳細については、Cachéドキュメンテーションを参照してください。

```
write $system.Config.SharedMemoryHeap.DisplayUsage()
```

任意の時点のGMHEAPの使用状況と推奨を取得するには、RecommendedSizeメソッドを使用できます。
ただし、システムのベースラインと推奨を得るには、これを何度も実行する必要があります。

```
write $system.Config.SharedMemoryHeap.RecommendedSize()
```

経験則:
繰り返しになりますが、アプリケーションの改善率はそれぞれに異なりますが、サイジングを始めるとすれば、次のいずれかから始めることができます。

```
(??128MB) ??? (64 MB * ???) ??? (2x locksiz) ??????????
```

GMHEAPは、ロックテーブルを含めたサイズにする必要があることに注意してください。

LargePages / HugePages

LinuxでHugePagesを有効にするとパフォーマンスを大幅に改善できる理由について、Mark Bolinskyが素晴らしい記事を書いています。

危険！ WindowsのLargePagesと共有メモリ

Cachéは、すべてのプラットフォームとバージョンで共有メモリを使用しており、パフォーマンスを大幅に改善することができます。それが常に使用されるWindowsも例外ではありませんが、注意すべきWindows固有の問題が

いくつかあります。

Cachéが起動すると、データベースキャッシュ(グローバルバッファ)、ルーチンキャッシュ(ルーチンバッファ)、共有メモリヒープ、ジャーナルバッファ、およびその他の制御構造に使用される単一の大きなチャンクの共有メモリが割り当てられます。 共有メモリの割り当てには、Cachéの起動時に、SmallPagesまたはLargePagesを使用することができます。 Windows 2008 R2以降では、CachéはデフォルトでLargePagesを使用するようになっていますが、それまでにシステムが長時間稼働していた場合はメモリが断片化しているため、Cachéの起動時にメモリを割り当てられず、代わりにSmallPaegsを使用して起動することがあります。

CachéをSmallPagesで起動してしまうと、構成で定義された量より少ない共有メモリでCachéが起動してしまう可能性があります。または
起動に時間が掛かかったり、起動できなかったりすることもあります
。個人的には、バックアップサーバーが長い間データベースサーバーとして使用されていないフェイルオーバークラスタを備えたサイトでこれが発生したのを見たことがあります。

ヒント: 緩和策として、オフラインのWindowsクラスタサーバーを定期的に再起動することができます。
また、Linuxを使用することもできます。

物理メモリ

物理メモリは、プロセッサの最適な構成によって決まります。
不適切なメモリ構成は、パフォーマンスに大きな影響を与えかねません。

Intelメモリ構成のベストプラクティス

これはIntelプロセッサに限定される情報です。
ほかのプロセッサに適用されるルールについては、ベンダーにご確認ください。

最適なDIMMのパフォーマンスは以下のような要因ににによって決定します。

- DIMMの種類
- DIMMのランク
- クロック速度
- プロセッサとの位置関係(近い/遠い)
- メモリチャネル数
- 必要な冗長機能

たとえば、NehalemとWebmereのサーバー(Xeon 5500と5600)には、プロセッサあたり3つのメモリチャネルがあるため、メモリはプロセッサあたり3個を1セットとしてインストールする必要があります。 たとえばE5-2600といった現行のプロセッサの場合、プロセッサあたり4つのメモリチャネルがあるため、メモリはプロセッサあたり4個を1セットとしてインストールする必要があります。

メモリが3つまたは4つを1セットとしてインストールされていなかったり、メモリDIMMのサイズが異なっていたりなど、メモリ構成が不均衡である場合は、それにより23%のメモリパフォーマンス低下となる可能性があります。

Cachéの機能の1つはメモリデータ処理にあるため、メモリから最高のパフォーマンスを得ることが重要であることを忘れてはいけません。 また、最大帯域幅のサーバーは、メモリ速度が最速になるように構成する必要があることにも注意してください。 Xeonプロセッサの場合、最大メモリパフォーマンスは、チャネルあたり最大2個のDIMMのみでサポートされるため、CPUを2個備えた一般的なサーバーの最大メモリ構成は、CPU周波数とDIMMサイズ(8 GB、16 GBなど)といった要因によって決まります。

経験則:

- バランスの取れたプラットフォーム構成を使用します。各チャネルと各ソケットに、同じ数のDIMMを使用してください。
- プラットフォーム全体でサイズ、速度、ランク数が同じDIMMを使用してください。
- 物理サーバーの場合、ホストサーバーの物理メモリの合計を、Intelプロセッサのベストプラクティスに従って、64 GB、128 GBなどの自然な区切りに切り上げます。

VMware仮想化に関する考慮事項

Cachéを仮想化する場合のガイドラインについては、今後別の記事で説明するつもりですが、 メモリ割り当てについては、以下の重要なルールを考慮する必要があります。

ルール: 本番システムにVMwareの予約メモリを設定すること。

前に説明したように、Cachéが起動すると、グローバルバッファ、ルーチンバッファ、GMHEAP、ジャーナルバッファ、そしてその他の制御構造で使用される単一の大きなチャンクの共有メモリが割り当てられます。

共有メモリのスワッピングは避けたいため、本番データベースVMの予約メモリを、少なくともCaché共有メモリにCachéプロセス、オペレーティングシステム、およびカーネルサービス用のメモリを加えたサイズに設定する必要があります。 適切かどうかわからない場合は、本番データベースVMの全メモリを予約してください。

原則として、本番サーバーと非本番サーバーを同じシステムに混在させる場合、非本番サーバーには予約メモリを設定しないでください。 非本番サーバーには、残っているメモリで対処してもらいましょう (^_)☆
VMwareは、8個を超えるCPUを搭載したVMを「モンスターVM」と呼ぶことがよくあります。
高トランザクションのCachéデータベースサーバーは多くの場合、モンスターVMです。 モンスターVMに予約メモリを設定する場合には、ほかの考慮事項があります。たとえば、モンスターVMがメンテナンスにより移行される場合や高可用性によって再起動がトリガされた場合には、ターゲットホストサーバーに十分な空きメモリが必要となります。 これに対して計画するにも戦略がありますが、NUMAを最大限に活用する計画といった他のメモリ関連の考慮事項と合わせて、今後の記事で説明したいと思います。

最後に

これは、メモリのキャパシティプランニングの開始部分であり、厄介な領域です。CPUのサイジングほど明確ではありません。 ご質問やご意見がございましたら、ぜひコメントを残してください。

この記事が投稿されるころ、私はグローバルサミット2016に向かっています。 今年このイベントに参加される方は、パフォーマンス関連のトピックで2つのプレゼンテーションを行いますので、ぜひご覧ください。また、開発者エリアで直接お会いしましょう。

#システム管理 #パフォーマンス #Caché #InterSystems IRIS #InterSystems IRIS for Health

---

ソースURL:
https://jp.community.intersystems.com/post/intersystems%E3%83%87%E3%83%BC%E3%82%BF%E3%83%97%E3%83%A9%E3%83%83%E3%83%88%E3%83%95%E3%82%A9%E3%83%BC%E3%83%A0%E3%81%A8%E3%83%91%E3%83%95%E3%82%A9%E3%83%BC%E3%83%9E%E3%83%B3%E3%82%B9-%E3%83%91%E3%83%BC%E3%83%884%C2%A0%E3%83%A1%E3%83%A2%E3%83%AA%E3%81%AE%E7%A2%BA%E8%AA%8D%C2%A0