記事
Tomohiro Iwamoto · 2020年8月17日 9m read

SQLベースのベンチマークを行う際に、実施していただきたい5つの項目

本稿について

本稿では、InterSystems
IRISを使用してSQLベースのベンチマークを行う際に、実施していただきたい項目をご紹介します。
Linuxを念頭においていますが、Windowsでも考慮すべき点は同じです。

メモリ自動設定をやめる

パフォーマンスに直結する、データベースバッファサ
イズの自動設定
はデフォルトで有効になっています。自動設定は、実メモリの搭載量にかかわらず、データベースバッファを最大で1GBしか確保しません。

> 更新: 2020年11月20日
> バージョン2020.3から、確保を試みるデータベースバッファが実メモリの25%に変更されました。

搭載実メモリ64GB未満の場合は実メモリの50%程度、搭載実メモリ64GB以上の場合は実メモリの70%を目途に、明示的に設定を行ってください。

設定するにはiris停止状態で、iris.cpfファイル(IRISインストール先\mgr\iris.cpf)を変更します。下記はブロックサイズ8KB用(既定値です)のデータベースバッファサイズの自動構成を4096(MB)に変更する例です。

修正前

```
[config]
globals=0,0,0,0,0,0
```

修正後

```
[config]
globals=0,0,4096,0,0,0
```

詳細はこちらです。

また、Linuxの場合、ヒュージ・ページ有効化
の設定を行ってください。設定値の目安ですが、IRIS起動時のメッセージから、確保される共有メモリサイズ(下記だと749MB)を読み取り、その値めがけて設定するのが簡単です。

> コンテナバージョンは非rootで動作するため、ヒュージ・ページは利用できません

```
$ iris start iris
Starting IRIS
Using 'iris.cpf' configuration file
Starting Control Process
Allocated 749MB shared memory: 512MB global buffers, 64MB routine buffer
$ cat /proc/meminfo
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

ページサイズが2048kBですので、750MB(丸めました)を確保するには750/2=375ページ必要になります。構成を変えると必要な共有メモリサイズも変わりますので、再調整してください。

```
$ echo 375 > /proc/sys/vm/nr_hugepages
$ iris start iris
Starting IRIS
Using 'iris.cpf' configuration file
Starting Control Process
Allocated 750MB shared memory using Huge Pages: 512MB global buffers, 64MB routine bu
ffer
```

メモリ関連の設定値は、パフォーマンスに大きく影響しますので、比較対象のDBMSが存在するのであればその設定値に見合う値を設定するようにしてください。

データベース関連ファイルの配置

特にクラウド上でのベンチマークの場合、ストレージのレイアウトがパフォーマンスに大きな影響を与えます。クラウドではストレージごとのIOPSが厳格に制御されているためです。

こちら
に細かな推奨事項が掲載されていますが、まずは下記のように、アクセス特性の異なる要素を、それぞれ別のストレージに配置することをお勧めします。

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdc        850G  130G  720G  16% /irissys/data      (IRIS??????????????????)
/dev/sdd        800G  949M  799G   1% /irissys/wij       (???????????????)
/dev/sde        200G  242M  200G   1% /irissys/journal1  (?????????)
/dev/sdf        100G  135M  100G   1% /irissys/journal2  (???????????????)
```

サイズは用途次第です。WIJに割り当てているサイズが大きめなのは、IOPS性能がファイルシステムの容量に比例するクラウドを使用するケースを意識したためで、IOPSとサイズが連動しないオンプレミス環境では、これほどのサイズは必要ありません。

テーブルへのインデックス追加

IRISのテーブルへのインデックス追加は一部のDWH製品のように自動ではありません。多くのRDBMS製品と同様に、CREATE TABLE命令の実行時に、プライマリキー制約やユニーク制約が指定されている場合、インデックが追加されますが、それ以外のインデックスはCREATE INDEX命令で明示的に追加する必要があります。

どのようなインデックスが有用なのかは、実行するクエリに依存します。
一般的に、ジョインの結合条件(ON句)となるフィールド、クエリやUPDATE文の選択条件(WHERE句)となるフィールド、グルーピングされる(GROUP BY)フィールドが対象となります。

インデックスの対象を参照ください。

データベースファイルのサイズ

IRISのデータベースファイルの初期値は1MBです。既定では、ディスク容量が許容する限り自動拡張を行いますので、エラーにはなりませんが、パフォーマンスは劣化します。ベンチマークプログラムでそのことに気づくことはありません。お勧めは、一度ベンチマークを実行して、必要な容量まで自動拡張を行い、データを削除(DROP TABLEやTRUNCATE TABLEを実行)した後に、再度ベンチマークを実行することです。

統計情報の取得

IRISはクエリの実行プランを最適化するために、テーブルデータに関する統計情報を使用します。
統計情報の取得処理(TuneTable/TuneSchema)は、初期データロード後に明示的に実行する必要があります。

> IRISバージョン2022.1以降、統計情報の取得処理が一度も実行されていない場合、初回のクエリ実行時に自動実行されるようになりました。そのため、初回のクエリに若干のオーバヘッドが発生します。

TuneTable(テーブル対象)/TuneSchema(スキーマ対象)を実行すると、既存のクエリプランがパージされるので、次回の初回クエリ実行時はクエリ解析・プラン生成にかかる分だけ時間が多めにかかります。
また、TuneTable/TuneSchema実行後にインデックスを追加した場合には、そのテーブルに対してTuneTableを再実行してください。

これらを考慮すると、ベンチマークで安定した計測結果を得るには、テストデータのロード後に、明示的に下記を実施することが重要になります。
- TuneTable/TuneSchemaを実行
- 最低でも一度は全クエリを実行する

> もちろん、これらの性能を計測したい場合は、その限りではありません。

TuneTable/TuneSchemaは、管理ポータル及びCLI
で、手動で実行可能
です。ベンチマーク測定時は、データの再投入やインデックスの追加や削除といった、統計情報の更新を繰り返し実行する必要性が高くなることを考慮すると、手動での更新は避けて、下記のObjectScriptのCLIやSQL文で実施するのが効果的です。

下記コマンドはスキーマMySchemaで始まる全てのテーブルの統計情報を取得します。

```
MYDB>d $SYSTEM.SQL.Stats.Table.GatherSchemaStats("MySchema")
```

下記コマンドは指定したテーブルのテーブルの統計情報を取得します。

```
MYDB>d $SYSTEM.SQL.Stats.Table.GatherTableStats("MySchema.MyTable")
```

あるいはSQL文として、データロード実行後にベンチマークプログラム内で実行する事も可能です。

```
TUNE TABLE MySchema.MyTable
```

タイムスタンプ型に%TimeStampではなく%PosixTimeを使用する

更新: 2020年11月20日 使用方法を、SQL文の修正が不要な方法に変更しました

> IRISバージョン2022.1以降、SQLのTimeStamp型の既定値が%PosixTimeに変更されました。この変更によりテーブルを新規作成した際のTimeStamp型は%PosixTimeになります。

```
create table TEST (ts TIMESTAMP)
```

これを、下記のように変更してください。

```
create table TEST (ts POSIXTIME)
```

デフォルトではSQLのTIMESTAMP型は、IRIS内部では%TimeStamp型で保持されます。
%TimeStamp型は、データを文字列として保存するのに対して、%PosixTimeは64bitの整数で保持します。その分、ディスクやメモリ上のデータサイズや比較演算処理などで有利になります。両者はxDBC上は、共にTIMESTAMP型となり、互換性がありますので、DML文の修正は不要です(敢えて指摘するとすれば、ミリ秒以下の精度が異なります)。以下は、実行例です。%PosixTimeに対するクエリのほうが、程度の差こそあれ、高速になっています。

```
create table TEST (ts TIMESTAMP, pt POSIXTIME)
create index idxts on TABLE TEST (ts)
create index idxpt on TABLE TEST (pt)
insert into TEST ... 100???????INSERT

select count(*),DATEPART(year,pt) yyyy from TEST  group by DATEPART(year,pt)
?? 669???
select count(*),DATEPART(year,ts) yyyy from TEST  group by DATEPART(year,ts)
?? 998???

select count(*) from TEST  where DATEPART(year,pt)='1990'
?? 533???
select count(*) from TEST  where DATEPART(year,ts)='1990'
?? 823???

select count(*) from TEST where pt>='1980-01-01 00:00:00' and pt<'1991-01-01 00:00:00
'
?? 350???
select count(*) from TEST where ts>='1980-01-01 00:00:00' and ts<'1991-01-01 00:00:00
'
?? 381???
```

TIMESTAMP型を%PosixTimeに変更するには、テーブルの作成を行う前、かつIRIS停止中に構成ファイル(IRISインストール先\mgr\iris.cpf)の下記を変更してください。以後、TIMESTAMP型で定義したカラムは%PosixTimeになります。

IRISバージョン2022.1以降、既定値がPosixTimeに変更されましたので、構成ファイルの変更は不要です。

```
[SQL]
TimePrecision=0  (???)
TimePrecision=6  (???)
[SqlSysDatatypes]
TIMESTAMP=%Library.TimeStamp (???)
TIMESTAMP=%Library.PosixTime  (???)
```

バイナリデータの保存には可能であればVARBINARYを使用する。

バイナリデータを保存する場合、サイズの上限が指定できる場合はVARBINARYを使用してください。LONGVARBINARYはサイズ上限が無い代わりに、内部でストリーム形式で保存する機構を伴うためパフォーマンスが低下します。

```
CREATE TABLE TestTable (ts TIMESTAMP, binaryA VARBINARY(512), binaryB VARBINARY(256))
```

VARBINARY使用時には、行全体のサイズに注意が必要です。SQLの行は、IRIS内部では、既定では下記のフォーマットで、1つのノードのデータ部(IRISの内部形式の用語でKV形式のバリューに相当します)に格納されます。このノードのサイズ上限は3,641,144バイトです。

長さ|データタイプ|データ|長さ|データタイプ|データ|...

この長さを超えると、\<MAXSTRING>という内部エラーが発生し、INSERTが失敗します。

Query ソース保存有効化

実行プランがコード化されたものがソースコードとして保存されます。デフォルトでは無効です。本稿では扱っていませんが、後の解析で有用になることもありますので、念のため有効化しておきます。

```
[SQL]
SaveMAC=1   <== ????0??
```

一通りのベンチマークを実行

この時点での実行結果は思った結果が得られないかもしれません。
ベンチマークプログラムによるレコードのINSERTにより、ユーザデータベースの自動拡張、一時データベースの自動拡張、WIJの自動拡張が、クエリにより、クエリプランの作成、インデックスの不足によるテーブルフルスキャンなどが発生している可能性があるためです。

(ベンチマーク実施により蓄積したデータの消去後の)2回目以降は、インデックスの不足以外の問題は解消されているので、パフォーマンスが向上するかもしれません。
また、これとは逆にジャーナルは、随時蓄積していきますので、ベンチマーク実施を繰り返すうちに、いずれはディスクの空き容量を圧迫します。
適宜ジャーナルファイルを削除してください。運用環境では絶対禁止の方法ですが、データベースミラーリングを使用しておらず、データの保全が必要のないベンチマーク環境でしたら、最新のジャーナルファイル以外をO/Sシェルでrmしてしまって構いません。

これ以後の、細かな確認作業はさておき、まずは上記の項目の実施をスタートラインとすることをお勧めします。

#SQL #パフォーマンス #InterSystems IRIS #InterSystems IRIS for Health

ソースURL:
https://jp.community.intersystems.com/post/sql%E3%83%99%E3%83%BC%E3%82%B9%E3%81%AE%E3%83%99%E3%83%B3%E3%83%81%E3%83%9E%E3%83%BC%E3%82%AF%E3%82%92%E8%A1%8C%E3%81%86%E9%9A%9B%E3%81%AB%E3%80%81%E5%AE%9F%E6%96%BD%E3%81%97%E3%81%A6%E3%81%84%E3%81%9F%E3%81%A0%E3%81%8D%E3%81%9F%E3%81%845%E3%81%A4%E3%81%AE%E9%A0%85%E7%9B%AE