

リレーショナル・データベース動向

本コラムでは、毎号、最も旬なソフトウェアの最新テクノロジー動向をご紹介します。連載記事最後の今回は、リレーショナル・データベース(Relational Database:以下、RDB)の最新の利用技術をいくつか取り上げ、解説します。

日本アイ・ビー・エム株式会社
ソフトウェア事業
ソフトウェア・テクノロジー推進会議
議長

米持 幸寿



Edgar F. Codd によって 1970 年に提唱されてから 40 年近く利用され続けている RDB ですが、その人気はまだまだ衰えず、今でもデータベースの定番モデルです。前回のコラムのように、違う型式のデータベースが出てきたにもかかわらず、RDB はそのままに、以前とは違う使い方や動作方法によって進化を続けています。

大量のデータ処理に分散キャッシュ

データベース処理を高速化する技術として古くからポピュラーな方法に「キャッシュ」があります。多くの場合、DBMS (データベース管理システム:データベース機能を実現する DB2[®] などのミドルウェア) の中にキャッシュを作るのが一般的です。そのほか、「同一クエリーなら、結果を残しておき、クエリーそのものを投げない」という思想のキャッシュもあります。これは、オブジェクト・キャッシュなどと呼ばれるたぐいのもので、アプリケーション側で簡単に実装できますし、アプリケーション・サーバーが持つオブジェクト・キャッシュ機能をオンにするだけでも同様の効果を得られます。

とても大きなデータ・セットを扱うとき、キャッシュに入りきらないとキャッシュがうまく働きません。そこで、プロセスにまたがってキャッシュにデータを入れて、分割処理する方法が登場しています。

複数の Java[™] VM (以下、JVM[™]) にキャッシュを分散させ、別々に処理させるためのフレームワークを WebSphere[®] XD で提供しています。これを「Object Grid」と呼びます。Object Grid では、データの処理を分散させることで、キャッシュを有効に働かせることができるため、「分散キャッシュ」としても知られています。

例えば、数十万件もあるデータを比較したり集計したりするとき、複数の JVM を起動してある範囲ごとに分

割して処理することができます。こうすると、特定の範囲のデータだけがキャッシュ内に入りますので、キャッシュに残る可能性が高くなるわけです。

Object Grid では複数起動する一つ一つのサーバーを「Grid サーバー」と呼び、その中に「パーティション」を作ります。「MapGridAgent」と「ReduceGridAgent」という二つの Java インターフェースを用意しており、これを継承した「エージェント」と呼ばれるオブジェクトをパーティションに自動的にばらまくことで処理を実行していきます。以前エージェント技術というものがありましたが、そのコンセプトにも近いものです。

Map エージェントは、マップ (コレクションの一種) をパーティションごとに作るためのエージェントで、Reduce エージェントは集計処理などによって、データを絞り込んで行く処理をパーティション分散するためのものと考えればよいでしょう。パーティションごとに集計が終わったものを、最終的に総合することで処理が終わるようにすると、分散処理が可能、ということです (図1)。

知識のある方はすでにお気づきかと思いますが、

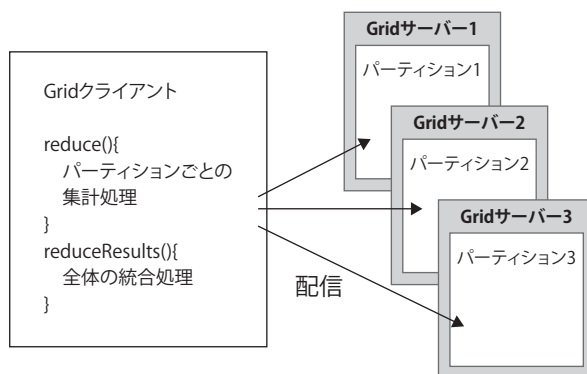


図1. Reduceの例

Google のシステムやオープンソースの Hadoop (分散処理型ソフトウェア) にも使われている MapReduce[1] という方法論です。

お手軽超高速化、インメモリーDB

IBM solidDB では、データをすべてメモリー上に配置して検索処理を行うため、キャッシュよりもさらに超高速なデータベース処理が可能なエンジンとなっています。メモリー上だけで実現されているデータベースはほかにもありますが、ディスク・ベースのエンジンと一つの接続で利用できたり、バイナリーが 4MB 程度、というとても小さかったりすることが特徴的です。

このテクノロジーは Solid 社買収により、IBM 製品と統合されたものです。テーブルはメモリーだけに置くこともできますし、ディスクに置くこともできます。メモリーだけに置いてあるテーブルの更新ログをディスクに保存して、万が一の障害のときに復旧を可能にすることもできます。旧来のデータベースのフロント・エンドにおいて、永続性のあるキャッシュのように使うこともできます。

セカンダリー・サーバーを立ち上げておいて切り替える機能も俊敏で、99.9999% の可用性を実現しています。

インメモリー・テーブルでは、インデックスに trie (Radix Tree: キーを分解して木構造にストアする方法) を使い、メモリー上のデータ・アレイを直接ポイントする構造が使われています。キャッシュのヒット検査なども省略されているインメモリー専用エンジンなので、キャッシュより高い効果が得られることがあります。ただし、SCAN などでは、B+Tree (インデックス構造の一つ) の方が早いことがあったり、更新処理は苦手だったりするので、注意も必要ということです。

solidDB は、DB2 や IDS (Infomix Dynamic Server) のフロント・エンドとして使うこともでき、既存のデータベースの高速化エンジンとしても利用できます。

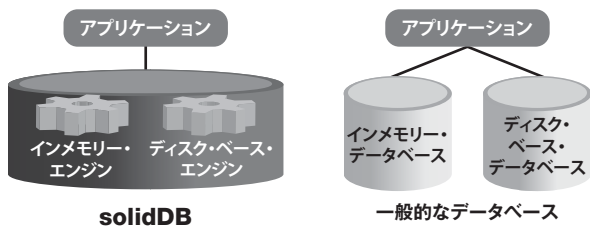


図2. solidDBはシングル・コネクションで利用可能

RDBをデータ・モデルとツールで隠ぺい

Groovy や PHP といったスクリプト言語を利用できる Web インフラストラクチャー・ソフトウェアである ProjectZero では、とても簡単に RDB にデータを保存できます。

Zero Resource Model (以下、ZRM) というフレームワークを使うと、RDB を簡単に隠ぺいできます。ZRM では「モデル」と呼ばれるデータでデータベースの項目、すなわちスキーマを定義します。例えば、ストリングで 512 文字の「name」列、ストリングで 512 文字の「address」列、日付の「birthday」列、といった具合です。このモデルをコマンド行で「zero model sync」と実行するだけで、必要な RDB テーブルが CREATE され、データが保存できるようになります。このような定義を行うための、Web ブラウザーで動作する簡単なツールを提供していますので、RDB の知識がなくても RDB 上にテーブルを準備することができます (図3)。

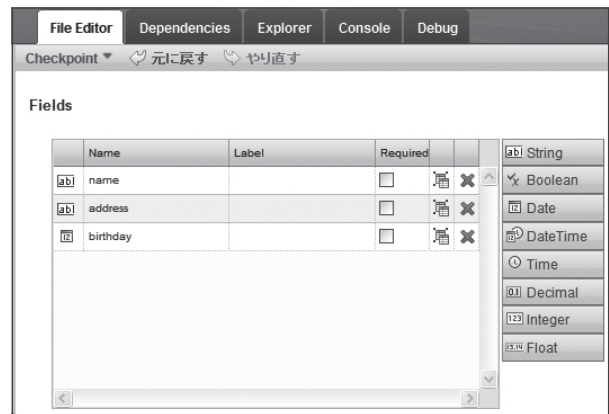


図3. ProjectZero のモデル・エディター

逆に、既存の RDB 表からモデルを生成することもできます。モデルを変更して、RDB テーブルを再作成したりすることもできます。しかも、一度入れてしまったデータもそのまま残しておくことができるなど、手が込んでいます。

キーによって取り出したり、検索した結果を、JSON (JavaScript のオブジェクト表記法) 形式でデータを変換して Web ブラウザーに送信したり、Web ブラウザーから送信されてきた JSON データを RDB 表に INSERT・UPDATE したり、削除したりする「RESTful」

サービスを簡単に作ることができるのです。RESTful サービスとは、HTTP の「GET」「POST」「PUT」「DELETE」などのコマンドを使ってデータの生成、読み出し、更新、削除 (CRUD) 処理を行う方式のことで、Web 2.0 でとてもポピュラーな Web サービス方式です。ZRM で RDB データ (モデル) をアクセスできるようにするためには、次のたった 1 行のコードを書くだけです。

```
ZRM.delegate()
```

このデータ・モデルで Web 上に公開 (パブリッシュ) された RDB データは、WebSphere sMash に付属している Dojo ツールキットの拡張コンポーネント (ZRM フォームやデータ・グリッド) を使って、とても簡単に Ajax アプリケーションとして Web 画面上に表示し、編集作業をすることができます。一度も SQL を書くことはありません。

手元に保存、どこからでもクエリー

昨今、流通にしても、医療にしても、情報が爆発的に増えすぎていて、旧来からあるデータベース・システムを拡張していっただけでは、その処理コストに見合わなくなってきています。

alphaWorks に掲載されている「Gaian データベース」は、オープンソースの RDB「Derby」をベースに作られた、「A Dynamic Distributed Federated Database (動的分散連邦データベース: 以下、DDFD)」[2] の実装です。DDFD とは、数多くのデー

タベースがネットワークを結んで連邦するような仕組みのことを指しています。MapReduce をはじめとする、さまざまな分散モデルを参考に、既存データベースやファイルを持つノード同士をつなぎ合わせる新しいデータベース・モデルとして考案されています。データベース装置そのものを大規模にするのではなく、多数のものをつなぎ合わせよう、というコンセプトです。センサー・ネットワークや、ニューロン・ネットワークにヒントを得て考えられたといわれています。

最初、どれかのノードにクエリーが渡ると、近くのノードにクエリーを伝播します。次から次へと伝播していき、その答えを統合して元へ戻していきます。一度クエリーが渡ったノードでは、そのクエリーを重複して伝播しないようにしています (図4)。

GaianDB には「Store locally, query anywhere (SLQA: 手元に保存、どこからでもクエリー)」という思想が使われています。それぞれのデータベースやファイルは、それぞれが個別に更新し、遠くにあってもクエリーできるという考え方です。

まとめ

前回紹介したような RDB 以外のデータベース、今回紹介したような RDB 分散や隠べいの技術の多くは、近年のコンピューティング・モデルがネットに深く入り込んでいることの現れともいえます。クラウド時代に入り、これらの技術をチェックしておくことは今後重要とってくるでしょう。

参考文献

- [1] J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.
- [2] A Dynamic Distributed Federated Database. <http://www.usukita.org/files/Page238.pdf> (2008/9) .

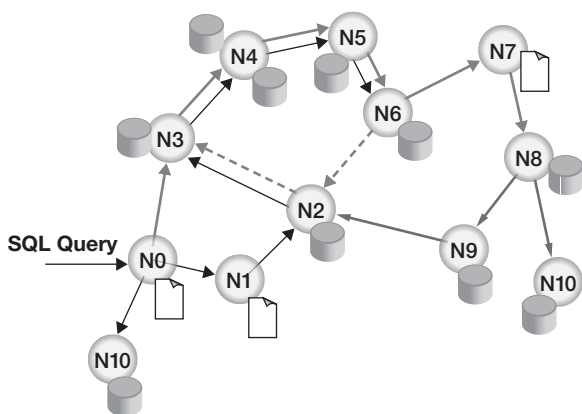


図4. GaianDBのコンセプト