グリッドコンピューティングと
並列ライブラリ

李　森

平成16年5月12日

1.グリッドコンピューティング

１．１グリッドコンピューティングとは

ネットワークを介して複数のコンピュータを結ぶことで仮想的に高性能コンピュータをつくり、利用者はそこから必要なだけ処理能力や記憶容量を取り出して使うシステムである。

複数のコンピュータに並列処理を行わせることで、一台一台の性能は低くとも高速に大量の処理を実行できるようになる。ビジネス利用や学術研究など、多くの可能性が模索され、実現に向けてさまざまな試みが行なわれいている。グリッドユーザーは、インターネットユーザーがWebコンテンツを見るように、大規模な仮想コンピューターをひとつのシステムとして利用することができるのである。

グリッドコンピューティングの重要な部分は、異機種混合環境や各コンピューティング資源が分散 It環境下でのコミュニケーションを可能にする、オープンな標準プロトコルに基づいています。

Webがインターネット・コミュニケーションの最初の段階であるのに対し、グリッドでは、あるひとつの目標に対して、インターネット上の複数の資源がコラボレーションで作用することを可能にしています。Webは、Webブラウザーで情報への自由なアクセスを可能にしました。それに対してグリッドは、行政機関や民間組織による利用のために、コンピューターやデータ・ストレージといった、さまざまなインターネットで接続されたIT資源へのアクセスを可能にします。

ピア・ツー・ピア・コンピューティングは2ユーザー間で資源を共用しますが、グリッド・コンピューティングは、複数のユーザー間での資源を共有します。

グリッド技術の普及や標準化を進めているThe Global Grid Forumを始め、学術機関を中心に研究が進められていたが、最近ではIBM社が商用化を目指すプロジェクトを立ち上げている。

a.可能性

グリッドコンピューティングは強力なコンセプトで、その主な魅力は一台のサーバのコンピューティングパワーを最大限に活用されることにある。現在のところ、コンピューターは動作していない時間がかなりあり、プロセッサは処理するデータを待っているような状態となっている。グリッドの世界では、停止している数百台、数千台のサーバのコンピューティングパワーを活用したり、大量の処理能力を必要とする人に「貸し出し」たりすることが可能だといわれている。

b．重要な要素

グリッドコンピューティングを実現するためには、地域的なものを全然意識しなくていいシームレスであることと、ネットワーク上の各種資源をその場所を意識せずに利用できる高いトランスペアレンシであることが求めている。

c．問題点

技術的な問題が山ほどあるのは当然ながら、最も難しい問題は、社会的および政治的な次元にあるといわれている。例えば、信頼関係がまるでない見ず知らずの他人の間におけるコンピューティングの共有をどのように促進すればいいのかといったことである。また、セキュリティーをどのように確保するかというなども問題であるといわれている。

１．２グリッドコンピューティングのメリット

a.異機種のデバイスおよびシステムの統合：

グリッドの技法はオープン・スタンダードに基づいているため、分散した資源を、中央で管理可能な1つに統合されたインスタンスにまとめることができる。また、グリッド・コンピューティングは、一連の水平統合機能を提供し、企業間、部門間のIT資源統合の課題に効率よく取り組むとともに、そのソリューションを複数の企業間に拡大することもできる。また、グリッドの異機種統合機能によって、企業は今までは使用できなかった特殊な装置を利用することもできるようになる。たとえば、研究開発グリッドに参加している科学者は、今まで利用できなかった国立研究所のサイクロトンに接続している特殊なスーパーコンピューターを利用できるようになり、問題解決処理を大幅に改善することができる。

b.オペレーティング環境の費用対効果を向上：

グリッドコンピューティングは、異種のIT機能間における資源の統合、プーリング、共有および管理を仮想化することで、オペレーティング環境およびその管理を単純化かつ合理化し、管理オーバーヘッドを軽減する。また、テクノロジー資源の利用がさらに効率化されるので、企業は、既存のテクノロジー投資をフル活用する、コスト効率に優れたITインフラストラクチャーを構築することができ、機能満載で巨大なコストを要するITインフラストラクチャーに依存する必要がなくなる。

c.柔軟でセキュアな「仮想コラボレーション」ドメインの作成：

今日のセキュリティー・ドメインおよび資源共有ドメインは、クローズドかつ専有のシステムおよびスタンダードに基づいており、柔軟性に欠け、管理が煩雑になる傾向がある。ユーザーおよび資源は一度設定すると、その追加/除去は、非常に困難な作業になります。これに対し、グリッド技法では柔軟性、選択の自由、そしてオープン・スタンダードといった概念が取り入れられている。グリッド・コンピューティングでは、グリッドは集約する資源についての「予備知識」を持たないという原則がベースになっている。そのため、グリッドには、多種多様で絶えず変化するIT環境をダイナミックに検出し、それに適応する能力が必要となる。これを可能にするため、グリッド・コンピューティングは、IT担当者が、ビジネスのニーズに合わせてセキュアな資源共有ドメインの各種パラメーターを簡単に設定、再設定および変更できるようにする。

d.需要の変動に対応するためのコンピューター資源容量の増加：

グリッド技法は、分散した資源を集約し未使用の能力を利用する機能を提供することで、使用可能なCPU資源およびデータ資源の量を大幅に増やす。

グリッド・コンピューティングは、予期せずトラフィックおよび使用率が急増した場合でも迅速に対応できるITインフラストラクチャーの構築に役立ちます。これは特に「オンデマンド」時代において協力な武器となる。

e.インフラストラクチャーの信頼性を高め、運用の弾力性を向上させる：

グリッド資源を従来の災害時回復シナリオの代替手段として利用することで、IT部門は、重複システムを使用した場合のコストの数分の一のコストでテクノロジー・インフラストラクチャーの信頼性および可用性を大きく高め、運用の弾力性を向上させることができる。さらに、プールされた資源を仮想化することにより、管理者は、数多くの異機種デバイスに分散しているジョブの進行状況を、あたかも1つのシステムのように容易にモニターすることができる。

1.3 グリッド環境

実際のグリッド・アプリケーションではグリッド・アプリケーション同士のさまざまな組み合わせが使用されますが、一般的には、次の3種類の資源から構成されている。

a.デスクトップ

デスクトップCPU有効活用グリッドによって、大容量の処理能力が利用できるようになる。デスクトップCPU有効活用グリッドでは、このタスクの実行に未使用のデスクトップ・コンピューティング・サイクルを利用する。グリッドはエンド・ユーザーのマシンのバックグラウンドで実行されるように設計されており、通常は「スクリーンセーバー」などPCが使用されていない時にのみ機能する。このため、ユーザーはグリッドの存在をほとんど意識することはない。デスクトップCPU有効活用グリッドは、科学や研究の分野で使用されるような、高度に並列化された分散アプリケーションで特に力を発揮する。

b.サーバー

サーバー・グリッドは、共有資源の活用に焦点をあてている点ではデスクトップCPU有効活用グリッドと同様ですが、サーバー・グリッドでは、フルに活用されていないサーバー資源を利用します。さらに、サーバー・グリッドを使用すると、特殊な計算または処理を行うために必要な専用デバイスを利用することができます。

c.データ

データ・グリッドは、共有およびコラボレーション用に単一のデータ・ソースを提供するように設計されている。また、データ・グリッドを使用して、大規模なコラボレーションにおいて複数のデータ・ソースを1つのビューで表示するための仮想ビューを作成することもできます。この処理を「データ・フェデレーション（連合）」という。たとえば、National Digital Mammography Archive （NDMA）と英国のeDiamondグリッドは、大規模なデータ・セット（この場合はデジタルX線と関連する臨床情報）を多くの処理施設で共有することを目的としています。この膨大なデータ・セットの可用性とグリッド・コンピューティングの強力な処理能力を結合することにより、科学者や研究者は、収集した情報を分析するためのアプリケーションを作成することができるようになる。情報のパターンや特長を調べることで、科学者は病気の環境要因や遺伝子要因について新たな事実を発見および究明できるようになる。

1.4 代表的な例

a.RBC保険会社

グリッド技術の採用により保険物件の評価に伴うリスク査定・保険金額の査定などの業務が12時間から32分に短縮

b.Charles Schwab証券取引会社

オンライン証券取引会社の最大手であるチャールズ・シュワブ・コーポレーション社は、グリッド技術の採用によりお客様からの投資相談へのレスポンスを4分から15秒に短縮

２．並列ライブラリ

2.1 並列計算と並列ライブラリ

並列計算機は、大きく分けて二つの種類があり、一つは共有メモリ型の計算機、もうひとつは分散型の計算機である。前者は一つの筐体に複数のCPUが乗っている、特にサーバーマシン等がよい例だ。後者は、ネットワークで接続されたパソコンやEWS(Engineering WorkStation)の集合体と考えればよいだろう。これらの違いは、並列ライブラリを使うことによって、特に意識する必要はなくなる。

並列ライブラリとは、並列プログラムを書く際に、プロセッサ間の通信等の処理を一手に面倒を見てくれる便利な関数群である。代表例はMPIである。

2.2MPI(Message Passing Interface)

2.2.1 Message Passing 方式とは

Message Passing方式とは、プロセッサ間でメッセージ交信しながら並列処理を実現する方式である。並列処理では、複数のプロセッサが通信を行いながら同時に処理を進めていく。そこで問題になるのがプロセッサ間通信となるのだが、メッセージパッシング方式ではプロセッサ間での通信を互いのデータの送受信で行う。そのため、「プロセッサｉ上でプロセッサｊにデータを送る。」というプロセスと「プロセッサｊ上でプロセスｉからデータを受け取る。」というプロセスが必ずペアとして成り立たなければならない。

2.2.2 MPI(Message Passing Interface)とは

MPIとは、分散メモリ環境における並列プログラミングの標準な実装である。その名前の通りメッセージパッシング方式に基づいた仕様であり、MPIの仕様に準じた実装ライブラリは、複数存在する。その中の幾つかはフリーで配布されており、UNIX系を中心としてWindows、Macとほぼすべてのシステム、アーキテクチャに対応している。そのため、どのような環境においてもMPIはフリーで使用することができる。

2.2.3 MPI通信方式

MPIにおいてデータを交換するための通信関数には、任意の二つのプロセス同士だけ関わる一対一通信と、任意のグループに付属するプロセスが全て関わる集団通信が用意されている。さらに、一対一通信、集団通信のそれぞれにブロッキング通信とノンブロッキング通信が用意されており、それぞれを適所でしようすることができる。

a.ブロッキング通信

ブロッキングとは、操作が完了するまで手続きから戻ることがない場所のことを意味する。この場合、各作業はその手続きが終了するまで待たされることになり効率が悪くなる場合がある。

例えば、送信、受信が宣言されてから完了するまで処理は待機状態になるため、非常に無駄が多くなってしまう。ただし、各操作の完結が保証されているためノンブロッキングに比べて簡単である。

b.ノンブロッキング通信

ノンブロッキングとは、操作が完了する前に手続きから戻ることがあり得る場合のことを意味する。送信、受信が宣言されてからも処理を続くことができるため、ブロッキング通信に比べ通信待ちの時間少なくなり、処理時間の軽減を計ることができ、より効率良いプログラムを作成することができる。特に、非同期通信などを行う場合にはノンブロッキング通信は必要不可欠をなる。しかし、操作の完了が保証できないため、ノンブロッキング通信を完了するための関数(MPIWait())を呼び出す必要がある。基本的には一対一通信のみにしかノンブロッキング通信は存在しない。

２.2.4 MPI を用いた簡単なプログラム例

```
#include<mpi.h>//ヘッダファイルの読み込み
 define N

 int calc(int a);

 main(int argc,char *argv[])
{
 MPI_Status status;
 int node_id,mode_size;
 int start,end;
 int *out,i;
 out=(int*)calloc(N,sizeof(int));

 MPI_Init(&argc,&argv);//MPI ライブラリを利用するための初期化
 MPI_Comm_rank(MPI_COMM_WORLD,&node_id);//プロセスが自分の rank を取る
 MPI_Comm_Size(MPI_COMM_WORLD,&node_size);//プロセス数を取る
 start=node_id*(N/node_size);
 end=(node_id+1)*(N/node_size);

 for(node_id==0){
       out[i]=calc(i);
 }

if (node_id==0){
       for(j=0,j<node_size;j++)
       {
       MPI_Recv(out+j*N/node_size,N/node_size*sizeof(int),
MPI_INT,j,100,MPI_COMM_WORLD,&status);//データを受信バッファから取り出す
       }
```

```
 }
else{
        MPI_Send(out+start,N/node_size*sizeof(int),MPI_INT,0,100,MPI_COMM_WORLD);
                //基本的なブロッキング送信の操作を行う
 }
MPI_Finalize();//MPI の実行環境を終了する。

free(out);
}
```

関数説明：

MPI_Init(&argc,&argv)

MPI ライブラリを利用するための初期化

MPI_Comm_rank(MPI_COMM_WORLD,&node_id)

プロセスが自分の rank を取る。コミュニケッタ（お互いに通信を行うプロセスの集合）内の全てのプロセスは、プロセスが初期化された時にシステムによって示された ID を持っている。これは 0 から始まる連続した正数が割り当てられる。プログラムはこれを用いて、処理の分岐、あるいはメッセージの送信元や受信元を指定することができる。この関数により node_id には自分の rank 番号が入力される。

MPI_Comm_Size(MPI_COMM_WORLD,&node_size)

コミュニケッタ内のプロセス数を取って、node_size に入力される。

MPI_Recv(void *buf,int count,MPI Datatype datatype,int source,int tag,
MPI Comm comm,MPI Status *status)

要求されたデータを受信バッファから取り出す、またそれが可能になるまで待つ。

void *buf：受信バッファの開始アドレス

int count：データの要求数

MPI Datatype datatype：データタイプ

int source：送信元の rank

int tag：メッセージタグ、メッセージを認識するために割り当てられる任意の整数

MPI Comm comm：コミュニケッタ

MPI Status *status：送信元の rank や送信時に指定されたタグの値を格納する

MPI_Send(void *buf,int count,MPI Datatype datatype,int dest,int tag,MPI Comm comm,)

基本的なブロッキング送信の操作を行う、送信バッファのデータを特定の受信先に送信する。

int dest:受信先の rank。

ほかのは MPI_Recv と同じだ。

MPI_Finalize()

MPI の実行環境を終了する。