■6 群（コンピュータ －基礎理論とハードウェア）－ 5 編（コンピュータアーキテクチャ（II）先進的）

3 章 分散コンピューティングシステム

（執筆者：合田憲人）［2010 年 5 月 受領］

■概要■

分散コンピューティングシステムは，単体で独立して動作する計算機をネットワークで接続することにより構成される計算機システムを意味する．本章では，分散コンピューティングシステムの概要を説明するとともに，分散コンピューティングシステムを活用するために必要となるプログラミング技術やミドルウェアについて説明する．

【本章の構成】

3-1 節では，分散コンピューティングシステムとは何かについて説明するとともに，その例として PC クラスタ，グリッド，ボランティアコンピューティングについて紹介する．分散コンピューティングシステムを活用するためには，並列化したプログラムを作成しなければならない．3-2 節では，分散コンピューティングシステム上でのプログラミング技術について説明する．また，分散コンピューティングシステムでは，独立した計算機群を連携して利用するためのソフトウェアが必要となる．3-3 節では，このためのミドルウェアについて説明する．

■6 群 － 5 編 － 3 章

3-1 構　成

（執筆者：合田憲人）［2009 年 3 月 受領］

分散コンピューティングシステムは，プログラムを複数の処理に分割して並列に実行する並列計算を目的として利用される計算機システムの一つである．並列計算を目的とした計算機システムは，マルチプロセッサや，PC クラスタ，グリッドなど様々なものに分類できる．これらのうち，単体でも独立して動作できる計算機を集めてネットワークで接続することにより構成されるシステムが分散コンピューティングシステムと呼ばれる．

例えば，マルチプロセッサは，複数の CPU やメモリをネットワークで接続して構成されたものであるが，マルチプロセッサを構成する CPU やメモリはそれら単体では計算機システムとして動作することはできない．したがって，マルチプロセッサは分散コンピューティングシステムとは呼ばれない．これに対して PC クラスタは，単体で動作可能なパーソナルコンピュータ（PC）をネットワークで接続して構成されるため，分散コンピューティングシステムと呼ぶことができる．また，マルチプロセッサのような計算機システムでは，システム専用に開発された高速なネットワークによって CPU やメモリが接続される．これに対して PC クラスタのような分散コンピューティングシステムでは，イーサネットなどの汎用のネットワークによって PC が接続される点も，分散コンピューティングシステムの特徴といえる．

独立した計算機の集まりを一つの計算機システムとして利用するためには，計算機を連携させるためのソフトウェアが必要となる．例えば，分散コンピューティングシステム上で並列プログラムを実行するためには，異なる計算機上で実行されるプロセス間で通信を行うためのライブラリが必要となる．また，ユーザが多くの計算機の中から 1 台の計算機を選んで計算を実行させることは，ユーザにとって不便であると同時に管理者にとっても計算資源の有効利用が難しくなる．この場合，計算機への計算の割り当てをユーザに代わって行うソフトウェア（ジョブスケジューラやバッチスケジューラと呼ばれる）が重要な役割を果たす．

以後，本節では分散コンピューティングシステムの例を紹介する．

3-1-1 PC クラスタ

PC クラスタは，複数のパーソナルコンピュータ（PC）をネットワークで接続することにより構成される分散コンピューティングシステムである．当初は複数のワークステーションを用いて構成されていたため，ワークステーションクラスタと呼ばれることが多かったが，PC の高性能化と低価格化による普及から，PC クラスタという呼び名が一般的になった．

PC クラスタの最も簡単な作り方は，汎用の PC をイーサネットなどの汎用ネットワークで接続することである．この場合，PC クラスタを構成する個々の部品はすべて安価な汎用品を用いることが可能なため，簡単かつ安価に PC クラスタを作ることができる．一方，PC クラスタの考え方は，高性能サーバやスーパーコンピュータでも取り入れられている．この場合は，ラックマウントサーバやブレードサーバと呼ばれる PC（PC クラスタの部品として製造された PC）をラックに搭載する方式を用いられており，限られたスペースにより多くの PC を搭載し，省スペース性や運用性を高めている．

PC クラスタ上で並列プログラムを開発する場合は，MPI[1] に代表される通信ライブラリが

用いられることが多い．MPI では，プロセス間でのデータ通信を行うための API が用意されており，API を用いてプロセス間の通信をプログラム中に記述することにより，異なる計算機上で実行されるプロセス間でデータの授受が可能となる．

PC クラスタ上で利用可能なバッチスケジューラは, 商用製品からフリーソフトウェアまで数多くのものが利用されている [2), 3), 4)]．ユーザは，PC クラスタ上のノード（PC）に自らログインして計算を直接起動する代わりに，ユーザの PC からバッチスケジューラに対して計算（ジョブ）を投入する操作を行う．バッチスケジューラは PC クラスタ上の各ノードの負荷を監視し，負荷の低いノードにユーザのジョブを割り当てる．

端末室や実習用の教室に設置されている PC 群を PC クラスタとして利用することもできる．例えば, 実習などでユーザに利用されることにない夜間に PC のソフトウェア設定を変更し，複数の PC 上で並列計算を行うことが実際に行われている．また昼間でも，PC の利用状況を監視し，誰にも利用されていない PC に対してジョブを割り当てることにより，計算能力の有効利用を図るソフトウェアも登場している [4)]．

3-1-2 グリッド

ネットワーク上に分散した計算機を連携させて計算を行うコンピューティンググリッド（計算グリッド，または単にグリッドと呼ばれることもある）も，分散コンピューティングシステムの一つである．前項で説明した PC クラスタは一つの組織内で管理されるが，コンピューティンググリッドは，異なる複数の組織により管理される計算機群から構成されている点が大きく異なる．

異なる組織により管理される計算機は，アーキテクチャやソフトウェアの仕様が異なるだけでなく，ユーザ管理や運用方法も組織の方針により異なる．また，異なる組織の計算機間の通信はインターネットなどの広域ネットワークを経由して行われるため, PC クラスタに比べて通信時間が非常に大きい．したがって，コンピューティンググリッドでは，このような計算機の仕様や運用方法の違い，通信性能の問題を解決しなければならない．この問題を解決するためのソフトウェアがグリッドミドルウェアである．グリッドミドルウェアは，上記の問題を解決し，ネットワーク上に分散した様々な計算機を連携させるためのサービスを提供する．グリッドに関する詳細は，8 章または文献 5) を参照されたい．

3-1-3 ボランティアコンピューティング

オフィスや家庭にある PC は，常に 100 %の能力で稼働しているわけではなく，ユーザの退席時や夜間など，多くの時間は利用されていないといわれている．ボランティアコンピューティングは，この使われていない間の計算能力（余剰計算能力）を集めて，大規模な問題を解くための方法である．

ボランティアコンピューティングでは，対象とする問題ごとにプロジェクトが作られ，参加者を募集する．プロジェクト参加者には専用のソフトウェアが配布され，参加者の PC にはこのソフトウェアがインストールされる．このソフトウェアは，PC の利用状況を監視し，PC が利用されていない間，データをサーバからダウンロードして計算を実行する．ボランティアコンピューティングのプロジェクトとしては，SETI@HOME [6)] が有名である．

■参考文献

1) Message Passing Interface Forum, http://www.mpi-forum.org/
2) PBS Pro, http://www.pbsgridworks.jp/
3) Sun Grid Engine, http://jp.sun.com/products/software/gridware/
4) Condor, http://www.cs.wisc.edu/condor/
4) 合田憲人, 関口智嗣(編著), “グリッド技術入門,” コロナ社, 2008.
5) SETI@HOME, http://setiathome.ssl.berkeley.edu/

■6 群 － 5 編 － 3 章

3-2 プログラミング技術

（執筆者：木村啓二）［2008 年 10 月 受領］

分散コンピューティングシステムのもつ計算能力を利用するためには，並列化されたアプリケーションプログラムを開発し，システム内の計算ノードに対して適切に処理を割り振る必要がある．本節では，アプリケーションプログラムの並列化ついて，その概要を説明する．ここでは特に，数値計算アプリケーションの並列化を対象とする．

まず，アプリケーションプログラムを並列化するための一般的なステップとその際の注意点を述べる．次に，並列化によって得られる性能向上の度合いに対して良い視点を与えるアムダールの法則を紹介する．最後に，アプリケーションプログラム並列化の代表的な二つの戦略について説明する．

3-2-1 並列化の手順

アプリケーションプログラム並列化の一般的なサイクルは以下のとおりである．まず，正しく動く逐次プログラムを作成し，十分に検証する．このとき，後の並列化を行いやすくするために，アルゴリズムとデータ構造を十分に検討する必要がある．一般に，実行時の挙動を把握しやすい明確なプログラムは並列化が行いやすい．次に，逐次プログラムの実行プロファイルから実行コストの高い部分を特定し，この部分の並列化を行う．その後，実行と並列化，チューニング，及びデバッグのサイクルを，目標性能に達するまで繰り返す．

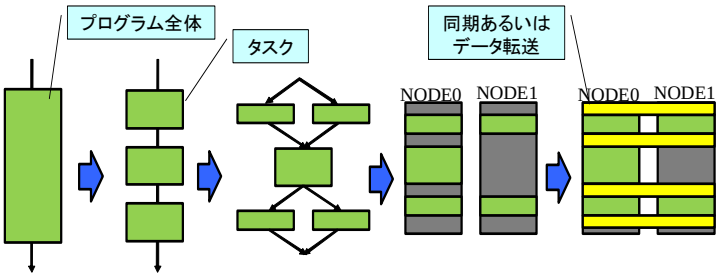

図 3・1　プログラム並列化の手順

図 3･1 に並列化の手順を示す．まず，プログラム全体を，並列処理を行う単位に分割する．この処理単位をタスクという．次に，プログラム中の各タスクに対して並列処理可能かどうかを判定し，可能である場合，そのタスクを各計算ノードで並列処理を行うための分割を行う．ここで並列化可能とは，オリジナルのプログラムにおける先行処理の演算結果と後続処理の使用の関係が，並列処理による実行順序の入れ替えにより影響を受けないことである．更に，分割後のタスクを計算ノードに実行時間が最小になるように割り当てる．このことをスケジューリングという．その後，計算順序を保証するための同期やタスク間で必要なデータの授受を行うデータ転送のコードを必要に応じて挿入する．これらのステップのすべてをプログラマが行う必要があるかどうか，更にどの程度行う必要があるかは処理系によって異

なる．例えば，共有メモリ型の並列計算機上で近年広く用いられている OpenMP [1] は，逐次プログラムに並列処理可能な箇所を指示すれば並列化可能である．その一方で，分散メモリ型の並列計算機上で広く用いられている MPI [2] では，並列化の仕方やデータ転送を細かくプログラマが指示する必要がある．また，逐次プログラムから自動的に並列化を行う自動並列化コンパイラの試みも行われている．

図 3･1 の並列化の手順で，効果的な並列処理を行うために，すなわち各計算ノードが休みなく演算処理をしている状態を保つために注意しなければならない点が二つある．一つはタスクの大きさであり，タスクの粒度と呼ばれる．もう一つはデータ転送である．タスクの粒度を小さくすると，タスクを計算ノードにスケジューリングする際に，隙間なく均等に割り当てられる可能性が高くなる．すなわち，各ノードの負荷が均衡する．しかしながら，タスクの粒度を小さくしすぎてしまうと，相対的に同期やデータ転送のオーバーヘッドが大きくなってしまい，効率のよい並列処理ができなくなってしまう．逆に，タスクの粒度を大きくすると，相対的に同期やデータ転送のオーバーヘッドが小さくなる．しかしながら，あまりに粒度を大きくしてしまうと，タスク間の実行コストに差がある場合にスケジューリング後のタスク間に隙間ができやすくなってしまい，ノード間に負荷の不均衡が発生し効率のよい並列処理ができない．データ転送も並列処理の効率を落とす原因となる．そのため，なるべくデータ転送が行われないように，データを共有するタスクを同一の計算ノードに割り当てる，あるいはタスク処理と非同期にデータ転送を行うことによりデータ転送のオーバーヘッドを隠蔽する，といった工夫が必要である．

3-2-2 アムダールの法則

アムダールの法則は，プログラムの並列化によって得られる性能向上を検討する際に良い視点を与える．アムダールの法則は，「最適化により得られる性能向上は，最適化適用可能な部分に制限される」と表すことができる．最適化による性能向上は，式(1)により算出できる．ここで，p は最適化可能な箇所の割合，n は最適化可能な箇所に対する速度向上率をそれぞれ示す．

$$speedup = \frac{1}{(1-p) + \frac{p}{n}} \tag{1}$$

例えば，オリジナルの逐次プログラムの 60 %が並列化により 10 倍高速化可能である場合，式(1)より全体で約 2.2 倍の速度向上を得ることができる．ここで，同じ箇所が 2 倍の計算ノードを使うことにより 20 倍高速化した場合でも，全体でオリジナルに対して約 2.3 倍の速度向上にしかならない．このようなプログラムの並列化に関する性質と得られる性能向上から，どの程度のコストを並列化に費やすべきか，などといった検討を行うことができる．

3-2-3 並列化の戦略

ここで，実際にアプリケーションプログラムを並列化する際の代表的な戦略を二つ紹介する．並列化の戦略は各種考えられるが，ここでは特に扱うデータ構造に着目し，配列の場合と木構造のような再帰的なデータ構造の場合のそれぞれの戦略について説明する．

配列に対して規則的な演算を行っているようなプログラムの場合は，その配列を分割して

各計算ノードに割り当てるようにタスクを分割すればよい．この様子を**図 3･2** に示す．配列の分割や各計算ノードに対するデータ転送を規則的に記述しやすいため並列化が容易である．多くのアプリケーションプログラムが，この戦略により効率のよい並列化が可能である．

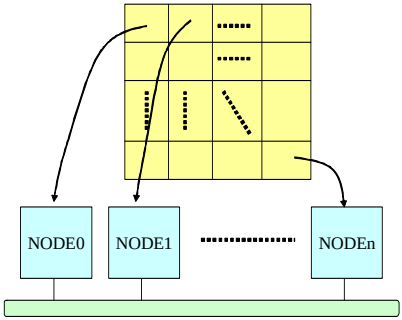

図 3・2　配列を扱うプログラムの並列化

再帰的なデータ構造の場合，**図 3･3** のようにデータ構造中のノードあるいはノード集合の各計算ノードに対する割り当てが考えられる．このとき，データ構造中のノードの計算ノードへの割り当てを静的に決定してしまうと，負荷の不均衡が発生してしまう可能性がある．このような場合は，あるノードに対する処理が終了した計算ノードに対して，残っているノードを動的に割り当てるような仕組みが有効となる．

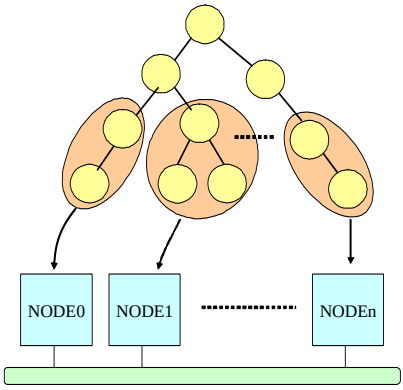

図 3・3　再帰的なデータ構造を扱うプログラムの並列化

これらのほかにも，タスク間のデータの流れに着目してパイプライン的に並列処理を行う，タスク間の事象のやり取りに着目して事象駆動的に並列処理を行うなどの戦略が考えられる．

■参考文献

1)　http://www.openmp.org/
2)　http://www.mpi-forum.org/

■6 群 － 5 編 － 3 章

3-3 基盤ソフトウェア技術

（執筆者：合田憲人）［2009 年 4 月 受領］

分散コンピューティングシステムでは，独立した計算機群を連携して利用するためのソフトウェアが必要となる．これらのソフトウェアは，各計算機の OS 上で動作してサービスを提供することから，ミドルウェアと呼ばれることもある．本節では，分散コンピューティングシステムのミドルウェアが提供するサービスを紹介する．なお，個々の技術の詳細については，文献または 7 章，8 章を参照されたい．

3-3-1 負荷分散

分散コンピューティングシステムを利用するユーザが，複数の計算機の中から一つを選んで計算（ジョブ）を実行することはユーザにとって負担が大きい．ユーザの立場ではなるべくすいている（他ユーザが利用していない）計算機を選ぶことが望ましいが，そのためにはユーザ自身がジョブを実行するたびに複数の計算機の状態を調べなければならない．また，計算機の状態は常に変化するため，同一の計算機上で複数のユーザのジョブが実行されてしまい，ジョブの実行性能が著しく低下することも起こり得る．この場合，分散コンピューティングシステム全体としての利用効率が下がり，管理者のとっても望ましくない．

分散コンピューティングシステム上では，複数の計算機に対してジョブを自動的かつバランスよく割り当てるサービスが必要であり，このサービスは負荷分散と呼ばれる．負荷分散を実現するために最もよく用いられる方法は，バッチスケジューラ [1)-3)] を利用する方法である．バッチスケジューラは，ユーザに代わってジョブを計算機に割り当てるためのソフトウェアである．ユーザは，ユーザのジョブを実行するためのスクリプトを作成し，以下のようなコマンドを用いてジョブの実行依頼を行う（以下の例では，qsub がジョブの実行依頼を行うためのコマンドである）．

```
% qsub ジョブスクリプト名
```

バッチスケジューラにはユーザから依頼されたジョブを蓄えるキューがあり，依頼されたジョブはまずはキュー内で実行待ち状態となる．キュー内のジョブは，ジョブの割り当て方針（スケジューリングポリシー）に従って計算機に割り当てられ，実行される．またユーザは，実行依頼したジョブの状態（実行待ち状態，実行中，実行終了など）を確認することもできる．

スケジューリングポリシーは，分散コンピューティングシステムを運用する管理者が決める．代表的なポリシーである FCFS（First Come First Served）は，実行依頼があった時刻順にジョブを負荷の低い計算機に割り当てる方法であり，非常に簡単な方法ではあるが，多くのバッチスケジューラで採用されている．このほか，ジョブに何らかの優先度を与えて優先度順に割り当てる方法や，（ジョブが並列プログラムの場合）利用する計算機数を考慮してジョブを割り当てる方法など，様々なスケジューリングポリシーが用いられており，またより効率のよいポリシーに関する研究も数多く行われている [4)]．

3-3-2 耐故障性の確保

分散コンピューティングシステムの構成が大きくなると，一部の計算機やネットワークなどに障害が発生する確率も高くなる．しかし，分散コンピューティングシステム上で計算（ジョブ）を安定して実行するためには，一部の計算機に障害が発生しても，ジョブの実行を継続できること，すなわち耐故障性の確保が重要である．この問題を解決する方法の一つがチェックポインティングである．

チェックポインティングは，ジョブの実行中に定期的に途中結果を保存しておき，ジョブが中断された場合には，その直前に保存した途中結果を用いてジョブの実行を再開する技術である．チェックポインティングを効率よく実施するためには，途中結果の保存や再読み込みに要するオーバーヘッドを削減することが重要である．チェックポインティングを用いて中断されたジョブの実行を再開する場合，途中結果が保存された時点からジョブが中断された時点までの処理は再度実行する必要がある．そのため，途中結果の保存は，より頻繁に行う方が再開時の無駄な処理を省くことができる．しかしこの場合，途中結果の保存に伴うオーバーヘッドが増え，ジョブの実行時間が長くなってしまう．一方，途中結果の保存をあまり行わない場合はオーバーヘッドが減るが，中断再開時の無駄な処理が増える．したがって，効率よくジョブを実行するためには，対象とするジョブの性質を考慮したうえで，適切な途中結果を保存するタイミングを決定することが必要となる．

3-3-3 システムの監視

分散コンピューティングシステムの管理では，システムに障害が発生した際に速やかに原因を特定し，システムを復旧することが求められる．これを実現するためには，システムを構成する計算機やネットワークを監視（モニタリング）することが必要となるが，これを人手で行うことは特に大規模なシステムでは難しいため，システムの監視を行うサービスがよく用いられる．

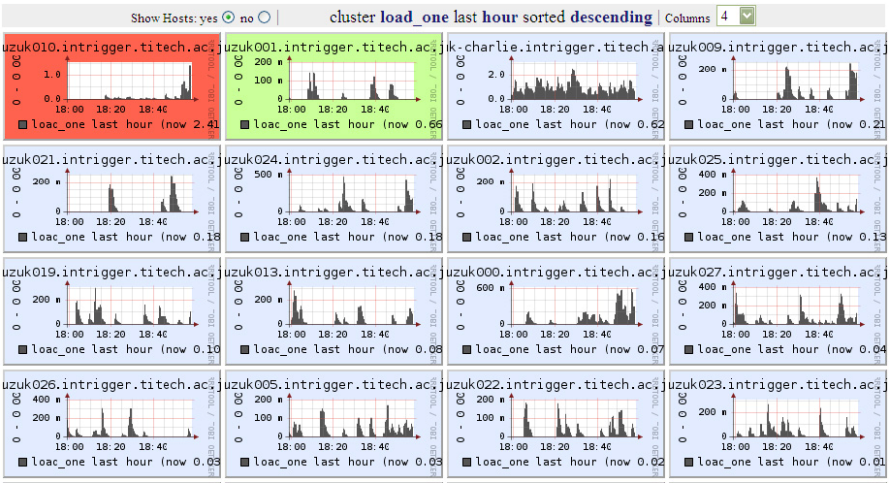

図 3・4　Ganglia による監視結果の例

図 3･4 は, 分散コンピューティングシステムの監視ソフトウェアの一つである Ganglia[5] による監視結果の例である．Ganglia では，計算機の CPU やメモリ，NIC の状態を定期的に監視し，管理者に提供する．図 3･4 の例では，システムを構成する計算機ごとの CPU 負荷が時系列に表示され，かつ現在の負荷状況が色分けして（赤が高負荷）表示されている．また，図中には表示されていないが，計算機が障害により停止している場合も画面上にその旨表示される．このようにシステムを構成する計算機の状態を画面上で確認できることにより，システム管理者は障害発生時の対応を迅速に行うことができる．

グリッドのような広域ネットワークにまたがって構成される分散コンピューティングシステムでは，ネットワークの監視も重要となる．この場合も計算機の監視と同様に，監視ソフトウェアが定期的にネットワークの状態（スループットやレイテンシなど）を監視し，管理者に提供する．グリッド上のネットワーク監視ソフトウェアの一つである Network Weather Service[6] では，グリッドを構成する任意の計算機間のネットワーク状態を監視するサービスを提供するほか，監視結果の履歴から統計的な手法を用いて将来の状態を予測する機能ももつ．

本項で紹介した監視ソフトウェアは，一般的に，システムを構成する計算機やネットワークの状態を定期的に測定するプロセスにより実装されている．例えば計算機の監視では，システムを構成する各計算機上に CPU やメモリの負荷を定期的に測定する監視用プロセスが配置され，各プロセスが測定結果を一つのサーバプロセスに報告する方式が用いられることが一般的である．またネットワークの監視では，監視するネットワーク上に測定用のパケットを定期的に送信することにより，ネットワークの状態を測定する方式が多く用いられる．この場合，このような測定のためのプロセスや通信が，分散コンピューティングシステム上で実行されるユーザのジョブの性能を妨げないことが必要となる．

■参考文献

1) PBS Pro, http://www.pbsgridworks.jp/
2) Sun Grid Engine, http://jp.sun.com/products/software/gridware/
3) Condor, http://www.cs.wisc.edu/condor/
4) B. A. Shirazi, A. R. Hurson, K. M. Kavi, “Scheduling and Load Balancing in Parallel and Distributed Systems,” IEEE Computer Society Press, 1995.
5) Ganglia, http://ganglia.info/
6) Network Weather Service, http://nws.cs.ucsb.edu/ewiki/