マルチメディア通信と分散処理 74－6
グループウェア 15－6
（1996. 1. 25）

計算機ネットワークに自律的に適応する分散システムの検討

谷江健 角川裕次 藤田聡 山下雅史
広島大学工学部

分散システムの最適な並列性や実行形態は、分散システムが実行される計算機ネットワークの状態に大きく依存する。本研究は、計算機ネットワークの状態に依存して、その並列度や実行形態を自律的に変える分散システムの可能性について議論する。

A STUDY OF DISTRIBUTED SYSTEM AUTONOMOUSLY ADAPTED TO COMPUTER NETWORKS

Takeshi Tanie, Hirotsugu Kakugawa, Satoshi Fujita, Masafumi Yamashita
Faculty of Engineering, Hiroshima University

Optimal parallelizm and execution form of distributed systems depend on the state of computer networks. This paper suggests potentialities of the distributed systems which changes it's parallelizm and execution form autonomously.

1 はじめに

複数の計算機を、ネットワークでつないで相互に通信できるような計算機ネットワークが普及してきた。このためユーザーはネットワークを通して手軽に他の計算機資源を利用することが出来るようになってきている。そこで、複数の計算機を用いて問題を分散して解くことで計算の応答時間を短縮することができる。我々は、処理手順は同一でデータのみを分割するアルゴリズムである分割統治法に基づき分散化を行うことにする。

その際に注意しなければならない点が二つ挙げられる。一つは、通信遅延によるオーバーヘッドのため問題を分散したからといって計算の応答時間が短くなるとは限らないことである。二つ目は、計算機資源を占有できないことである。これは、マルチタスクマルチユーザーシステムでの計算機プロセッサは複数のユーザと共有して使われるため計算機の負荷が重くなり本来の計算能力より劣った計算機を使用することになる。したがって、負荷の状態によりアルゴリズムの動作を変えることで応答時間を短縮できるかも知れない。この二点より単純に並列度を上げれば計算の応答時間が短縮されるわけではないことがわかる。シーケンシャル実行が最も応答時間が短い場合もあるのである。それらは計算機ネットワークの状態に依存するので分散の仕方をユーザーが決定するのは困難であるし手がかかる。本稿では、アプリケーション自身が計算機ネットワークに適応して自分自身の動作を変えることで自律的に最適な分散システムを構築する可能性を検討する。

自律的に最適な並列性を獲得するアルゴリズムの評価のために 10Mbps の Ethernet でつながれた同機能同性能な計算機群を用いてマージソートを行った。2章では、並列実行時間がインスタンスサイズと計算機負荷により決定できることを示す。3章では、自律的に並列度を変えるためのアルゴリズムを示す。4章で、そのアルゴリズムを実験的に評価する。5章で、本研究のまとめを述べる。

2 並列実行時間

分割統治法によるアルゴリズムの実行時間が計算機の負荷とインスタンスサイズに依存することを示す。マージソートを例題として、その実行時間を実験式と理論式から見積もる。n をインスタンスサイズ、lを計算機負荷、p を計算機台数として、ソートの実行時間を $T_{sort}(n,l,p)$ で表す。また、ソートアルゴリズム中のマージ操作時間を $T_{merge}(n,l)$、通信時間を $T_{comm}(n,l)$、そして部分問題の遠隔起動時間を $T_{rexec}(l)$ とする。マージソートの計算構造は 2 分木で表せる。2 分木の各階層のマージ操作の計算複雑度は $\frac{1}{2}n$ であり、2 分木の高さは $\lg n$ だからシーケンシャルマージソートの計算複雑度は $\frac{1}{2}n\lceil\lg n\rceil$ である。

2.1 モデル化

並列度 4 のマージソートを図 1のような計算構造で行う。ノード中の番号はホスト番号を表しており、親と子が同じ番号ならばローカルな再帰呼出しを表す。そうでなければリモートな再帰呼出しである。並列度 2 では、図 1の深さ 1 までの 2 分木となる。チャート図を図 2に示す。t_0〜t_1 は $T_{rexec}(l)$、t_1〜t_2 は $T_{comm}(\frac{n}{2},l)$、t_2〜t_3 は $T_{sort}(\frac{n}{2},l,1)$、$t_3$〜$t_4$ は $T_{comm}(\frac{n}{2},l)$、t_4〜t_5 は $T_{merge}(\frac{n}{2},l)$ である。チャー

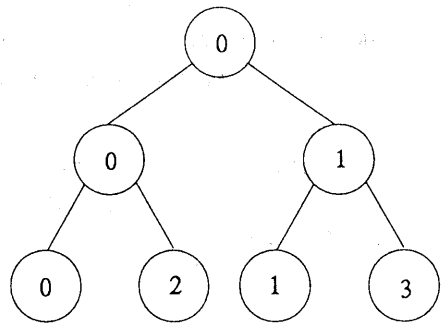

図 1: 計算構造と計算機との対応

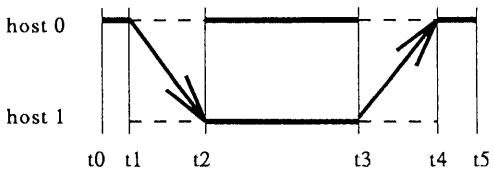

図 2: 並列度 2 のチャート図

ト図(図2)から$T_{sort}(n,l,2)$を導くと次式となる。

$$\begin{aligned} T_{sort}(n,l,2) &= T_{sort}(\tfrac{n}{2},l,1)+2T_{comm}(\tfrac{n}{2},l) \\ &+ T_{rexec}(l)+T_{merge}(\tfrac{n}{2},l) \end{aligned}$$

イーサネット上での通信は1本のバスを計算機群が共有して使うため同時に複数の通信はできない。したがって、並列度4のときのチャート図は図3の様になる。t_0〜t_1は$T_{rexec}(l)$、t_1〜t_2は$T_{comm}(\frac{n}{2},l)$、t_2〜t_3は$T_{rexec}(l)$、t_3〜t_4とt_4〜t_5は$T_{comm}(\frac{n}{4},l)$、t_5〜t_6は$T_{sort}(\frac{n}{4},l,1)$、$t_6$〜$t_7$は$T_{comm}(\frac{n}{4},l)$、$t_7$〜$t_8$は$T_{merge}(\frac{n}{4},l)$、$t_8$〜$t_9$は$T_{comm}(\frac{n}{2},l)$、$t_9$〜$t_{10}$は$T_{merge}(\frac{n}{2},l)$である。チャート図(図3)から$T_{sort}(n,l,4)$を導くと次式となる。

$$\begin{aligned} T_{sort}(n,l,4) &= T_{sort}(\tfrac{n}{4},l,1)+2T_{comm}(\tfrac{n}{2},l) \\ &+ 3T_{comm}(\tfrac{n}{4},l)+2T_{rexec}(l) \\ &+ T_{merge}(\tfrac{n}{4},l)+T_{merge}(\tfrac{n}{2},l) \end{aligned}$$

2.2 実験式

実験によりシーケンシャル実行時間、マージ操作時間、通信時間、リモート起動時間を求めて次のような実験式を作成した。

- **シーケンシャル実行時間**
  $T_{sort}(n,l,1)=1.35(l+1)n\lg n\ [\mu sec]$
- **マージ操作時間**
  $T_{merge}(n,l)=T_{sort}(n,l,1)/\lg n\ [\mu sec]$
- **通信時間**
  計算機負荷に比例する定数と通信媒体のみに関係する定数が送信すべきインスタンスサイズに比例すると次のような式となる。定数は実験により決定した。図4がその実験結果である。
  $T_{comm}(n,l)=(2.355l+5.846)n\ [\mu sec]$
- **リモート起動時間**
  $T_{rexec}(l)=0.6(l+1)\ [sec]$

2.3 結果

図5の実線は各々実際のマージソートを実行してその応答時間を計測したものである。そして破線は先のモデル式をプロットしたものである。同様に図6をみても破線は実線とほぼ一致しておりモデル化が正しいことを示している。しかしながら、インスタンスサイズが大きければ通信は最大パケット長の繰り返し送信なので問題ないが、インスタンスサイズが小さいと可変パケット長となるのでばらつきが大きくなりモデルどおりには行かない。

3 アルゴリズム動作

3.1 基本的な考え方

アルゴリズムにサイズnのインスタンスを与えるとする。アルゴリズムはまず初めにこのインスタンスを半分に分けて並列度2で実行を試みる。具体的には$\frac{n}{2}$を関数の再帰呼出し(ローカルな再帰呼出し)によって割り当て、残りの$\frac{n}{2}$をリモートホスト上に割り当てて実行(リモートな再帰呼出し)する。そのとき分散したインスタンスサイズ$\frac{n}{2}$をスレッショルド(θとする)として記憶しておく。ここで、シーケンシャルな実行時間(T_{seq}とする)が分かっていれば、並列度2のパラレル実行時間(T_{par}とする)は自分自身の実行時間を計測することで得られるから、この二つの実行時間を比較することでサイズnのインスタンスの並列化の効果を判定できる。もし並列化の効果がなければ、次回の実行からはこのインスタンスサイズではシーケンシャル実行をすれば良い。逆にパラレル実行時間の方が短ければ、並列化の効果があるので次回の実行では更に並列度を上げて実行してみる。

1. もし $(T_{seq} \leq T_{par})$ ならば θを大きくして集中化傾向を強める。
2. もし $(T_{seq} > T_{par})$ ならば θを小さくして分散化傾向を強める。

3.2 スレッショルドの設定と更新

実行時に与えられるインスタンスサイズをnとする。初めは並列度2で実行する必要があるためθの初期値は$\frac{n}{2}$とする。また、直前の実行時に並列化の効果があったかどうか覚えておくために$lower$と$upper$の2変数を導入する。この2変数の初期値は負値とする。それ以降の実行では、以下のようにθを設定する。アプリケーション起動後、θを設定するために$lower$と$upper$を保持しているファイルを読み込んで次を実行する。

case 1: $(lower \geq 0)$ かつ $(upper < 0)$ ならば
$\theta \leftarrow 2 \times \theta$

case 2: $(lower < 0)$ かつ $(upper \geq 0)$ ならば
$\theta \leftarrow \frac{1}{2} \times \theta$

case 3: $(lower \geq 0)$ かつ $(upper \geq 0)$ ならば
$\theta \leftarrow \frac{1}{2}(lower + upper)$

プログラムの再帰呼出し部分をθを閾値として用いて再帰呼出しをローカルかリモートにするかを決定する。そしてアプリケーション実行後に次のようにして二分探索の要領で最適なθの範囲を決定する。もしシーケンシャル実行時間がパラレル実行時間より短ければθはもっと大きくてもいいのでこのθの値を$lower$に保存する。逆に、パラレル実行時間の方が短ければ、θは分散化傾向を強めるため小さくすべきなので$upper$に保存する。但し、シーケンシャル動作しかしなかったときは何も更新せずに終了する。次回実行時に読み出してθを設定するために更新した$lower$と$upper$はファイルに保存しておく。

3.3 シーケンシャル実行時間の推定

シーケンシャル実行時間を推定するためにプログラミングユーザにインスタンスサイズnにおけるそのアルゴリズムの計算量をnの関数の形で記述してもらう。与えられた関数を$f(n)$、インスタンスサイズをnとして並列度2の実行が行われたときに同じインスタンスサイズのシーケンシャル実行時間を推定する。この時、子供のノードは$n/2$のインスタンスサイズのシーケンシャル実行を行っていると見なせる。アルゴリズムがこの子供の実行時間を自分自身で計測すれば、$f(n/2)$に対応する実時間$T(n/2)$が得られる。インスタンスサイズnのときの実行時間$T(n)$は、

$$T(n) = \frac{f(n)}{f(n/2)} \times T(n/2)$$

として計算により求められる。

4 実験

4.1 実験環境

10Mbpsのでつながれた同機能同性能なワークステーション4台を用いて実験を行った。機種は、Sun Microsystems社のAS4080/30EGXワークステーション(CPU:Super SPARC, クロック周波数:36MHz, cache:36KB, main memory:36MB)である。記述言語はC言語を用いた。

4.2 実験方法

整数のソーティングをマージソートにより行う。インスタンスサイズnを10万、20万、30万、40万、50万まで変化させる。そして計算機の負荷を4台とも同程度となるように調節して、負荷値lを0、1、2、3、4、5まで変化させた。それぞれシーケンシャル実行時間とパラレル実行時間(並列度2と4)および本研究で提案するオートノマス実行時間を測定する。実験は10回繰り返して、その平均値をグラフにプロットした。

4.3 実験結果

実行時間が計算機の負荷とインスタンスサイズより決まると仮定すると、lが2のときは図7より、インスタンスサイズが0〜16万まではシーケンシャル動作が最適であり、15万〜40万では並列度2のパラレル動作が最適であり、40万より大きな所では並列度4のパラレル動作が最適であることが分かる。図9と図10はオートノマス実行が自律的に最適な並列度を獲得している様子を表している。図9では1回目の実行(並列度2)で並列化の効果があったので、2回目の実行では並列度を更に上げて並列度4で実行している。その結果、3回目の実行で最適な並列度は2であることが分かり、それ以降は並列度2で実行している。図8は、図9の実行時のスレッショルドの動きを表しています。図7で、シーケンシャルと並列度2のグラフがクロスするインスタンスサイズはおよそ16万なので、スレッショルドは8万程度となるはずである。図8では、スレッショルドは2回目の実行で5万から10万の範囲となって、5回目の実行ではおよそ7万に収束している。

5 おわりに

本研究によって計算機ネットワークの状態により最適な並列度が存在することがわかり、アルゴリズムが自律的に変化するシステムの有効性を確認した。今後は計算機負荷の動的変動に対しての適応について研究する予定である。

参考文献

[1] H.E.Bal, J.G.Steiner, A.S.Tanenbaum, "Programming Languages for Distributed Computing Systems," ACM Computing Surveys, Vol.21, no.3, Sept, 1989.

[2] K.Schwan, R.Ramnath, S.Vasudevan, and D.Ogle, "A Language and System for the Construction and Tuning of Parallel Programs," IEEE Trans. Software Eng., Vol.14, no.4, pp.455-471, Apr, 1988.

[3] 前川 守, 所 真理雄, 清水 謙多郎, "分散オペレーティングシステム UNIX の次にくるもの," 共立出版株式会社, 1991.

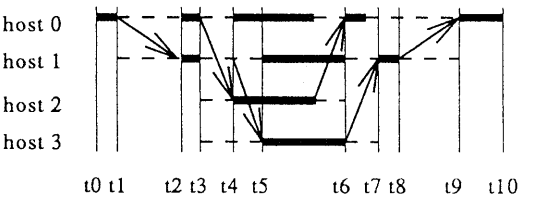

図 3: 並列度 4 のチャート図

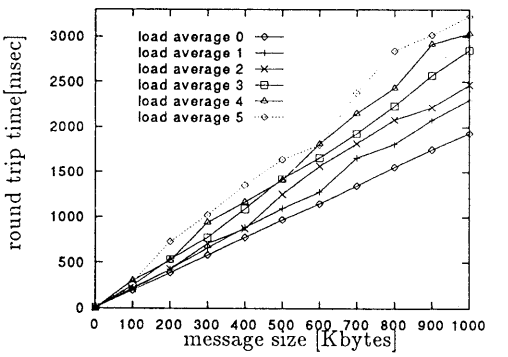

図 4: 通信時間

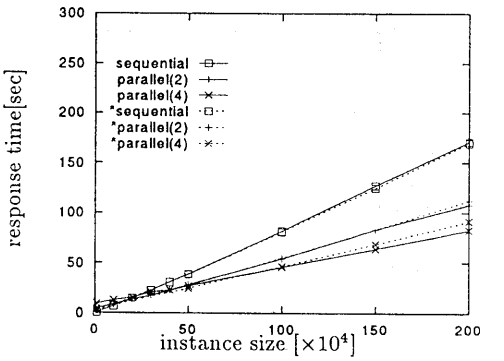

図 5: $l = 2$

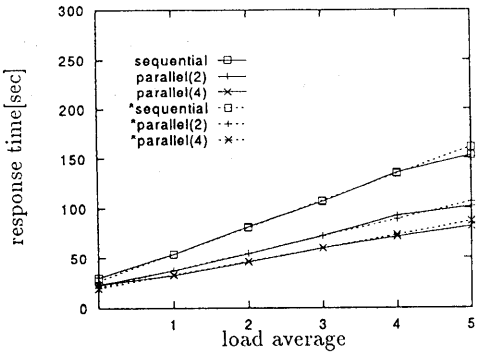

図 6: $n = 1000000$

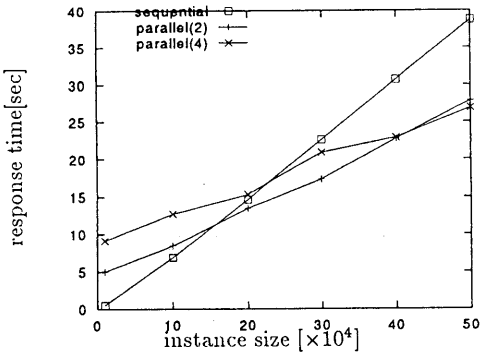

図 7: $l = 2$

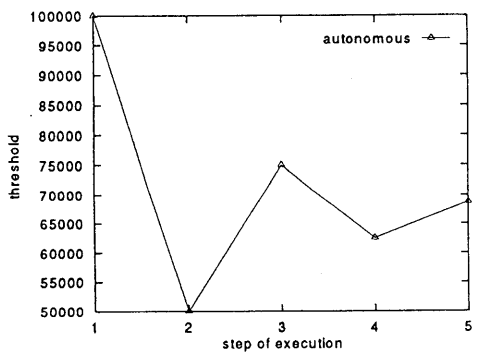

図 8: スレッショルド変化 ($n = 200000, l = 2$)

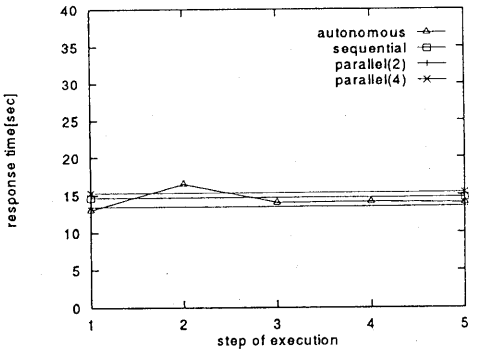

図 9: オートノマス実行 ($n = 200000, l = 2$)

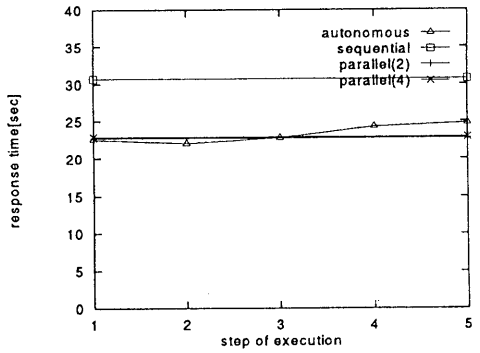

図 10: オートノマス実行 ($n = 400000, l = 2$)

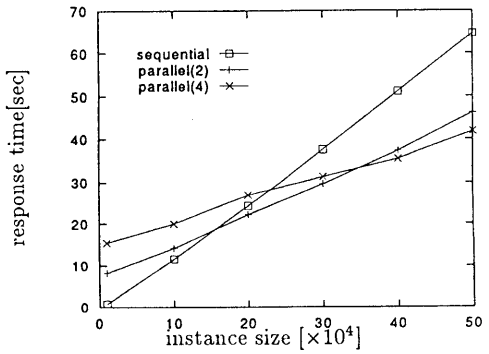

図 11: $l = 4$