ユーザレベル通信ライブラリ “Suci” の スナップショット・アルゴリズムへの応用

Applying User Level Communication Library “Suci” to SnapShot Algorithm

屋比久 友秀† **河野 真治**‡
Tomohide Yabiku Shinji Kono

１． まえがき

並列分散環境において PC クラスタなどの普及により、50 台以上の分散環境も比較的容易に利用する事ができるようになってきた。PC クラスタのような比較的密結合な分散環境では、MPI や PVM 等の並列ライブラリが利用されている。これらの多くは TCP ベースの通信を採用しているが TCP ベースの通信では TCP 特有の問題があり、比較的密な分散環境において無視できない問題が幾つかある。たとえば、輻輳崩壊の問題[1][2][3][4]である。この問題を解決する為に、Jacobson、Brakmo らは TCP の改善を試み[5][6][7]、Stevens は TCP アルゴリズムの標準化[8]を行った。いずれの改善も、ネットワーク上すべてのホストで一様に送出パケットを抑えるようにアルゴリズムを採用している。通信の公平性を保証する場合はそれでも良いが、通信に優先度が存在するようなアプリケーションおいて、優先したい通信に十分なスループットを得る事ができない場合がある。このような通信を行う場合は、輻輳制御やフロー制御を TCP のようにシステムレベルに隠蔽するのでは無く、ユーザレベルで行う必要がある。

また、別の問題として、多数の TCP ソケットを開く際に、巨大な送受信バッファがカーネル内部に生じ、メモリ空間を圧迫する場合がある。これは、大規模な分散環境において多数のノード間で通信を行う完全結合(全てのノードが全てのノードと同時に通信を行うモデル)の場合に顕在化する。通常 TCP の最大ウィンドウサイズは、TCP ヘッダのウィンドウサイズフィールドが 16bit であるため、2 の 6 乗 = 64KByte である[9]。1000 台以上の完全結合型通信を TCP で実現しようとした場合、64Kbyte x 1000 x 2 = 約 128Mbyte もの送受信バッファがカーネル内部に生じることなる。

このような問題が TCP にあるためシステムレベルの輻輳制御やフロー制御では特定のモデルに対しては不十分である。我々はこれらの問題を解決するために、ユーザレベルにトランスポート層の仕組みを実装し、アプリケーションプログラムから輻輳制御、フロー制御を行えばよいと考え、ユーザレベルのフロー制御と輻輳制御を備えた通信ライブラリを提案し評価した[10]。本稿では、Suci を使った応用例としてスナップショット・アルゴリズム[13]を取り上げ、ユーザレベルでフロー制御を行う場合と TCP ベースの通信ライブラリとを比較し、ユーザーレベルでフロー制御を行う場合の優位性を示す。

†琉球大学大学院 理工学研究科

‡琉球大学、科学技術振興事業団さきがけ研究 21

2．Suci の設計

TCP が抱えるこのような問題を解決するために、我々は UDP ベースの通信ライブラリ[11][12]を拡張した Suci (Simple User Communication Interface)を実装し、評価した[10]。

Suci はユーザ空間に再送用のキューを持ち、メッセージにシーケンス番号を付加し、UDP パケットの欠落や順序の入れ替えに対するメッセージの配送を保証している。UDP を利用する事で一つのソケットで複数のプロセスと通信ができため、システム内に大きなバッファを用意する必要が無い。

2.1 輻輳制御

輻輳制御も並列アルゴリズムに密着した記述を行えば、効率よく並列処理を進められる可能性がある。そのためには、複数の並列プログラムが協調してフロー制御を行う仕組みが必要である。

また、1 対 1 の送受信のネゴシエーションが行われるフロー制御に対して、輻輳制御は複数の送受信ノードが参加する 1 対多のフロー制御として捉えることもできる。以下に Suci の輻輳回避の手順を示す。

(1) 輻輳が生じた場合、Ack が取れていなかったメッセージを再送キューに格納する。
(2) 送信側は再送キューの数により輻輳状態を検出する。
(3) 送信側は受信側に、再送が必要なキューのサイズと一緒に再送許可の要求を送る。
(4) 受信側は送信された再送キューのサイズと、最大スループットから再送を許可するかどうかを判断し、送信側に再送の可否を送る。
(5) 送信側は許されたメッセージサイズを送信する。

このような輻輳制御を行えば、TCP のような一様なスループットの低下がなくなり、通信毎に必要な帯域を割り当てることができる。

3. スナップショット・アルゴリズムへの応用

スナップショット・アルゴリズムは、システム全体の状況を計算し記録するアルゴリズムである。スナップショット・アルゴリズムは分散環境においてアルゴリズムの終了、デッドロック、トークンの紛失などの定常特性の検出および判定に用いられている[13]。また、故障から復帰するためのチェックポイント、ロールバックアルゴリズム[15][17]や分散プログラムのデバック[15][16]にも利用されている。ネットワーク形状の無矛盾性を考慮したスナップショット・アルゴリズム[18]も提案されている。

3.1 スナップショット・アルゴリズムのフロー制御

スナップショット・アルゴリズム[13]ではマーカー(*marker)* と呼ばれる特別なメッセージを利用する。このマーカーによって無矛盾な状況を記録している。このマーカーの送受信を行う際に、受信側でマーカーを受け取れなかった場合、TCP では受信するまで再送を行う。しかし、受信側が実際に欲しいのは送信側の現在の状況であって、昔の状況では無い場合がある図１。

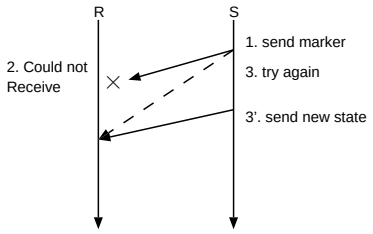

図１　スナップショットの再送

図１では、はじめに S(Sender)から R(Receiver)にマーカーを送信するが、R が何らかのトラブルで受信できない場合、TCP ベースの通信ではもう一度同じ状態を R に送信する。しかし、もう一度同じ内容を送信するよりは、より計算の進んだ段階を送っても良い場合が考えられる。このような要求のあるアプリケーションでは TCP ベースの通信を利用する事ができないが、Suci では、ユーザレベルでフロー制御を行うため、R が受信できなかった事を検出し、それにあわせて、もう一度スナップショットを取る事ができる。

3.2 リング形状スナップショットにおける Acknowledge の削減

リング形状のスナップショットでは、マーカーをリングに沿って送受信を行う。図２の点線が TCP をベースとした通信の場合である、この場合、各ノード間で Ack や Nack の送受信を行いメッセージ到達を保証している。Suci の場合はユーザレベルでフロー制御が可能なため、図２のように太線で一方向でメッセージの送受信が可能になり、確認応答の為のメッセージが大幅に減少する事ができる。この場合、N1 から送信されたメッセージが最終的に N1 に戻ってきた場合に、メッセージ到達が保証される。

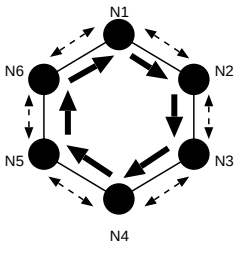

図２　リング形状スナップショット

4. まとめと今後の課題

本稿では、完全結合型の分散環境において TCP の問題を指摘し、その解決策としてユーザレベルでフロー制御を行う通信ライブラリ Suci の概要を説明し、Suci をスナップショット・アルゴリズムに応用した場合の優位性について述べた。今後の課題として実際にスナップショット・アルゴリズム[13]を Suci ベースで実装し、他の通信ライブラリと比較し、Suci ライブラリの有効性を検証する必要がある。

参考文献

[1] 尾屋祐二, 後藤滋樹, 西尾章治朗, 宮原秀夫, 村井純 編: トランスポートプロトコル, 岩波書店, 2001.

[2] 西田 佳史: TCP 発展の歴史と未来, コンピュータソフトウェア, Vol. 17, No.1, pp73-80, 2000.

[3] Nagle, J.: Congestion Control in IP/TCP Internetworks, Computer Communication Review, Vol.14, No.4, 1984

[4] Nagle, J.: Congestion control in IP/TCP Internetworks, RFC 1191, 1990.

[5] Jacobson, V.: Modified TCP Congestion Avoidance and Control, *Proceedings of ACM SIGCOMM '88, pp.314-329, 1994.*

[6] Jacobson, V., Braden, R. and Borman, D.: TCP Extensions for High Performance, RFC 1323, 1992.

[7] Brakmo, L., O'Malley, S. and Peterson, L.: TCP Vegas, New techniques for congestion detection and avoidance, *Proceedings of ACM SIGCOMM '94, pp.24-35, 1994.*

[8] Stevens, W. R.: TCP Slow Start, Congestion Avoidance, Fast Restransmit, and Fast Recovery Algorithms, RFC 2001, 1997.

[9] Postel, J. (ed.): Transmission Control Protocol, DARPA Internet Program Protocol Specification, RFC 793, 1981.

[10] 屋比久 友秀, 河野 真治: 並列分散ライブラリ Suci の実装と評価, 情報処理学会 第 90 回システムソフトウェアとオペレーティングシステム研究会, 2002.

[11] 河野 真治, 神里 健司: UDP を使った分散環境とその応用, 日本ソフトウェア科学会大会論文集, 2000.

[12] 河野 真治, 池村 正之: 状態集合の分割による時相論理の検証の並列化, 電気学会・電子情報通信学会合同講演会, 1998.

[13] K. Chandy and L. Lamport.: Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems, Vol. 3, No. 1, pp63-75, 1985.*

[14] R.E. Strom and S. yemini. Optimistic recovery in distributed systems. *ACM Transaction on Computer Systems,Vol. 3,No. 3, 1985.*

[15] 青柳, 真鍋. 分散デバッガのためのチェックポイント・ロールバックアルゴリズム, 日本ソフトウェア科学会ソフトウェア研究会(関西)資料, SW-92-10-4, 1992.

[16] 増澤, 都倉. 分散デバッガのための Causal Distributed Breakpoint を求めるアルゴリズム, 電子情報通信学会秋季大会, SD-1-8, 1992.

[17] 守屋 宣, 櫟 粛之. インターネットエージェントのための動的スナップショットアルゴリズムと部分ロールバックアルゴリズム, 電子情報通信学会 技術研究報告, Vol. 101, No. 44, pp17-24, 2001.

[18] 佐藤 泰朗, 井上 美智子, 増澤 利光, 藤原 秀雄. 分散移動システムにおけるスナップショット・アルゴリズム, 情報処理学会 研究報告 アルゴリズム No. 47-4, 1995.