2004－HPC－99　(34)
2004／8／1

協調動作する並列 TCP ストリームへの Packet Spacing の適用とその評価

中村 誠
東京大学情報基盤センター
makoto@is.s.u-tokyo.ac.jp

亀沢 寛之
富士通 (株) サーバシステム事業部
kamezawa.hiroyu@jp.fujitsu.com

稲葉 真理　平木 敬
東京大学大学院情報理工学系研究科
{mary, hiraki}@is.s.u-tokyo.ac.jp

概要　非常に大きな帯域遅延積を必要とする環境で並列 TCP を用いてデータ転送を行う場合、複数の TCP ストリームが同時に通信を開始すると輻輳ウィンドウが十分に大きくなるはるか以前にパケットロスが発生し、よい初期性能が得られない。また、New Reno 方式を採用した TCP はウィンドウサイズの回復が非常に遅く、大きな帯域遅延積にまでウィンドウが成長するには非常に小さな回線エラー発生率と多くの時間を要する。本稿ではまず、並列 TCP に対しパケットの送出レートを制御する TCP を用いる事で通信初期段階での性能改善が可能であることを示す。さらに、協調動作しストリーム間の通信速度差を緩和する DECP アルゴリズムと HighSpeed TCP を並列 TCP に対し使用した場合の性能への影響と振る舞いを観察した。

Co-operative Parallel TCP Streams with Packet Spacing

Makoto Nakamura
Information Technology Center,
the University of Tokyo

Hiroyuki Kamezawa
Fujitsu Ltd.

Mary Inaba　Kei Hiraki
Graduate School of Information Science and Technology,
the University of Tokyo

Abstract　When using parallel TCP streams to transfer data on network of huge bandwidth delay product, starting multiple TCP streams simultaneously causes premature packet losses far before TCP congestion windows become enough large, and we can get only poor initial performance. In addition, TCP New Reno recovers window very slowly, and it requires very small error rate and long time.

In this paper, we first demonstrate that Transmission Rate Controlled TCP improves initial performace of parallel TCP streams. In addition, we examine the influence and behavior of co-operative algorithm for parallel streams, DECP, and HighSpeed TCP applied against parallel TCP streams.

1　はじめに

科学技術計算用ファイル共有システム、データレゼボワールは Super-SINET や米国 Abilene といった超高速ネットワークや、APAN や IEEAF といった大容量の大陸間接続回線を利用して、科学実験や自然科学シミュレーションなどにより生成されるテラバイトオーダーのデータを遠距離ネットワークを

通じて共有するという目的を達成するために開発されている。データレゼボワールは大量のディクスに蓄積されたデータを効率よく転送するため、iSCSI プロトコルによるディスク-ディスク間データ転送を並列 TCP を用いて行っている。その開発過程において、非常に大きな帯域遅延積 (Bandwidth Delay Product, BDP) を必要とする環境における並列 TCP の性能問題に直面した [1]。

並列 TCP の性能に関する問題点としては、パケットロスがストリーム間で非同期に不公平に発生する事が挙げられる。さらに、その確率は、ストリームのマクロな転送速度に比例しない。そのため、個々のストリームの転送速度が不均衡になり、転送データ量が同一である場合には、転送所要時間も差がついてしまう。我々は、並列 TCP が協調動作する流量制御アルゴリズム DECP [2] により転送速度の不均衡を緩やかではあるが解消できる事を確認した。

また、並列 TCP が同時に通信を開始すると利用可能な帯域を超過する以前に最初のパケットロスが発生してしまう。さらに、New Reno 方式はウィンドウサイズの回復が非常に遅いため、通信初期で性能が低いと、利用可能な帯域まで転送速度が増加するために時間を要する。以前の実験では、32 台構成のクラスタ間で並列 TCP によるデータ転送を行った際、通信遅延が 300ms 弱の通信路において 128 ストリームで転送速度を 5Gbps 増加するため約 25 分もかかってしまった [2]。

本稿では、(1) TCP 通信のウィンドウ制御に従って流量制御を行う TRC-TCP [4] を用い並列 TCP の通信初期での性能改善を図り、(2) 帯域遅延積の大きな環境向けの TCP 輻輳制御方式 HighSpeed TCP を使用した並列 TCP に DECP アルゴリズムを適用した場合の性能への影響と振る舞いを観察する。

1.1 DECP

DECP (Dulling Edges of Co-operative Parallel Streams) [2, 3] は、並列 TCP が協調し、転送速度の速いストリームの速度増加を抑え、遅いストリームの速度増加を促進し、不均衡を解消するアルゴリズムである。

並列 TCP ストリームがネットワークに送出されているパケット量 (inflight) を定期的に収集し、ヒストグラムを作成する。その際、輻輳イベントによるウィンドウサイズの急激な変化に緩やかに反応するために、過去のヒストグラムを加味しつつヒストグラムを更新する。最頻値域より適度に大きなウィンドウサイズを各ストリームのウィンドウサイズの上限値とする事で、転送速度の比較的速いストリームの速度増加を抑制する。

1.2 TRC-TCP

TRC-TCP (Transmission Rate Controlled TCP) [4, 5, 6] は、TCP の輻輳制御の拡張で、輻輳ウィンドウサイズ $cwnd$ と遅延時間 RTT を基にパケットの送出レートを制御する方式である。マクロな転送速度 $cwnd/RTT$ とパケットレベルのウィンドウ制御方式とその実装に起因するミクロな送出速度の差を緩和する事により、余った帯域がある状態でのパケットロスを軽減する。

Clustered Packet Spacing (CPS) という、Slow Start 時にある程度までパケットの送信間隔が $RTT/cwnd$ (この $cwnd$ は RTT 時間毎にウィンドウサイズが 2 倍になると仮定した時の値) となるように OS カーネルの TCP スタックにおいてスケジュールする方式を用いた。スケジュールには擬似割り込み tasklet を用いた。その粒度は CPU 負荷や割り込み頻度に左右されるが、概ね 1μs 以上の間隔であれば十分に実現が可能である (図 1)。

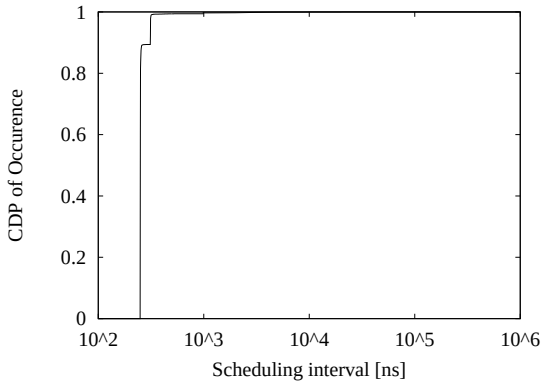

図 1 tasklet 割り込みのスケジュール間隔分布 (ディスク上のデータを Iperf で送信時)

Slow Start 時には RTT 時間毎にウィンドウサイズが 2 倍になり、遅延帯域積が大きいとウィンドウサイズの増加量が異常に大きくなってしまい、大量のパケットがロスする可能性が非常に高く、ネットワークに対する負荷も大きい。CPS の有効性を確認するため、Slow Start 時のウィンドウサイズの最大値 $w_{max\_slowstart}$ を予め設定し、ウィンドウサイズ w が $w > w_{max\_slowstart}/3$ の時点から RTT 時間当りのウィンドウサイズの増加量を $max\{\frac{w_{max\_slowstart}-w}{2}, 2\}$ とする方式 CPS+ も用いた。

1.3 HighSpeed TCP

HighSpeed TCP [8] は、Floyd によって提案された大きな輻輳ウィンドウを必要とする長距離広帯域ネットワーク上で TCP 通信性能を改善するための新しい輻輳制御方式ある。W を平均ウィンドウサイズ (単位はパケット)、p をパケットロス率とした時に、通常の TCP の Response Function が $W = 1.2p^{-0.5}$ なのに対して、HighSpeed TCP は Response Function として $W = 0.12p^{-0.835}$ となるよう設計されている。

TCP の輻輳制御アルゴリズムはパケットレベルでは、RTT 時間当たりのウィンドウサイズの増加量 $a(w)$ と、輻輳イベント発生時の減少率 $b(w)$ の 2 つのパラメータで定義される。通常の TCP が $a(w) = 1, b(w) = 0.5$ なのに対して、HighSpeed TCP は

$$a(w) = 2w^2 p(w)\frac{b(w)}{2 - b(w)}$$

$$b(w) = 0.5 + (b_{high} - 0.5)\frac{\log w - \log w_{low}}{\log w_{high} - \log w_{low}}$$

(ただし、$b_{high} = 0.1, w_{low} = 38, w_{high} = 83000$) とする。つまり、ウィンドウサイズが大きくなるにつれて増加量が増え、減少率が小さくなる。

2 実験環境

図 2 のようにホストを GbE でスイッチに接続し、2 台のスイッチ間を GbE で繋ぎ、単一のボトルネック帯域を共有する”ダンベル”型のトポロジーを構成した。このスイッチ間接続を回線遅延エミュレータを介し、片方向 100ms 以下の一定時間の通信遅延を擬似的に発生させた。

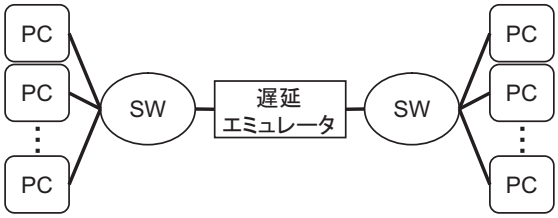

図 2　ネットワーク実験環境

ホストは IBM x345 で、Dual Xeon 2.4GHz, 2GB メモリ, Intel 82546EB チップ搭載の 1000BASE-T から構成されている。OS は Linux 2.4.26 を用い、TCP バッファサイズを 32MB、ネットワークデバイスの送信キュー (TxQueue) を 18,000 パケットに設定した。

ネットワーク性能測定ツール Iperf [11] をデータの送受信プログラムとして用いた。転送されるデータはメモリ上で処理されるため、ディスクの I/O 性能は考慮しなくてよい。

3 並列 TCP の通信開始時の性能改善

並列 TCP が同時に通信を開始すると Slow Start のよって利用可能な帯域をを超過する前に、最初のパケットロスが発生してしまう。これは、Slow Start によるパケット送信がバースト的に発生し、スイッチ内のバッファを溢れさせてしまうせいだと推測される。そこで、パケットの送出レートを CPS および CPS+ を用いて制御した場合と、ストリームの開始時間をずらす ad hoc な回避策と比較した。

RTT を 200ms に設定し、4 台のホストが同時に 1 ストリームを開始した時に、Slow Start の終了時に設定される ssthresh (パケットロス検出時にウィンドウサイズの 1/2) を調べた (表 1)。各方式とも 10 回ずつ測定した。CPS によって $RTT/cwnd > 100\mu$s の間はパケットの送出レートを制御した。CPS+ の $w_{max\_slowstart}$ は、転送速度 250Mbps に相当する 4058 パケットに設定した。オフセットとは、各ストリームの開始時間を 50ms ずつずらした場合である。

	最大	最小	平均	標準偏差
通常	121	32	60.4	25.5
CPS	3137	2074	2741.3	153.9
CPS+	2044	1929	2027.0	33.2
オフセット	6183	2048	3458.8	1412.7

表 1　通信開始直後の Slow Start 終了時の ssthresh。単位はパケット。オフセットは各ストリームの開始時間を 50ms ずつずらした場合

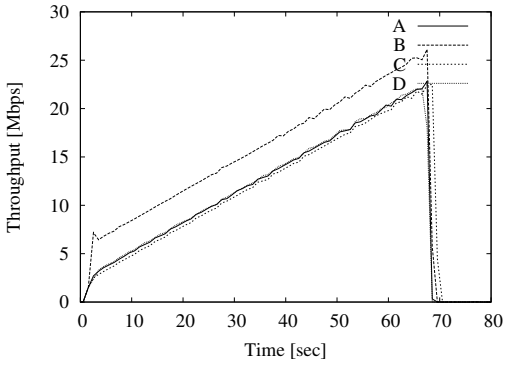

図 3　通常 TCP を使用した並列 TCP

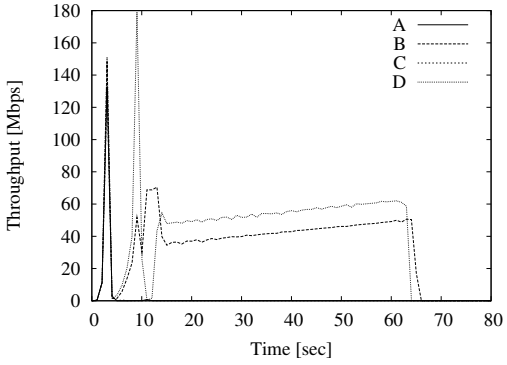

図 4　CPS を使用した並列 TCP

通常の並列 TCP は各ストリームが平均約 7.4Mbps の転送速度までウィンドウが成長した時点でパケットロスが発生してしまっている。CPS を用いるとボトルネック帯域を埋めつくすまで確実にウィンドウが成長する一方、その成長が急激すぎてパケットロスを検出した時点ではウィンドウサイズの合計がボトルネック帯域を大幅に超えてしまう。オフセットの結果から通信開始をうまくずらす事ができれば、CPS 同様の効果が得られる一方ウィンドウサイズのボトルネック帯域に対する超過分が非常に大きく、ばらつきも大きい。その差は、CPS の場合輻輳がすべてのストリームに対して影響するのに対して、オフセットの場合輻輳がバーストなパケット送信が衝突する一部のストリームにのみ影響するためと考えられる。

図 3 から図 6 までは各方式の転送速度の変化の様子を測定したものである。CPS やオフセットでは Slow Start により瞬間的に 100Mbps 以上の転送速度を出す事が出来る一方で、パケットロスを Fast Retrasmit/Fast Recovery により回復できないストリームが多く、再送処理の終了に数十秒から数分要する場合もあった。

4　HighSpeed TCP を用いた並列 TCP

図 7 は HighSpeed TCP を用いた時の並列 TCP の転送速度の変化の様子である (CPS+ も使用している)。HighSpeed TCP では $a(w)$ がウィンドウサイズと共に大きくなるため、RTT が 200ms であれば 0Mbps から約 60 秒で 100Mbps、約 180 秒で 1Gbps に転送速度が大きくなる反面、並列 TCP に用いるとストリーム間の転送速度の差は拡大するという負の側面もある。

4.1　並列 HighSpeed TCP への DECP の適用

DECP アルゴリズムは速度差を緩和する事ができるため、並列 HighSpeed TCP のデメリットを補う事ができる。しかし [2] でのパラメータは $a(w) = 1$ 用に設定されているため、そのままでは HighSpeed TCP の特徴を生かせない。

そこで、全ストリームのウィンドウサイズがヒストグラムの 1 つの値域に収まっている場合には、HighSpeed TCP の $a(w)$ を基に $w_{min} + m\frac{a(w_{min})}{RTT}$ (m は DECP の適用間隔) を上限値として指定するよう修正した。

図 8 は並列 HighSpeed TCP に上記修正を施した DECP を適用した場合の転送速度の変化の様子であ

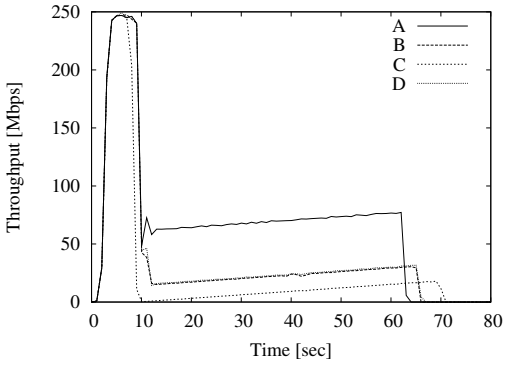

図 5　CPS+ を使用した並列 TCP

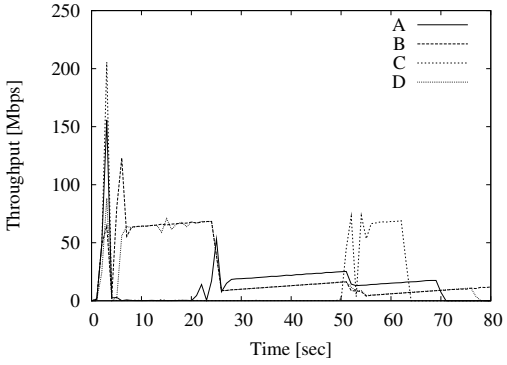

図 6　開始時間をずらした通常 TCP による並列 TCP

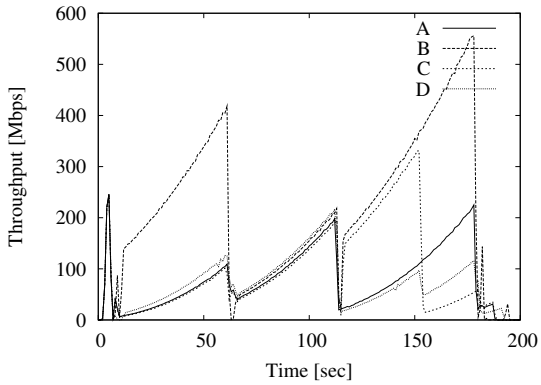

図 7　HighSpeed TCP を使用した並列 TCP

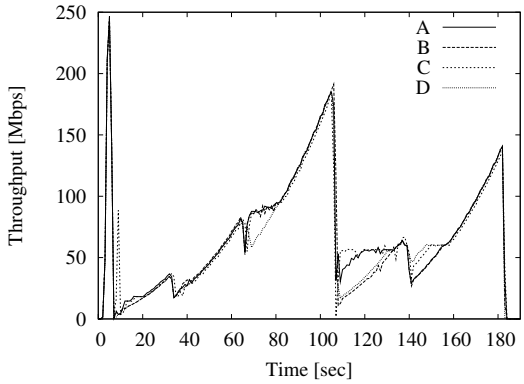

図 8　並列 HighSpeed TCP に DECP を適用した場合

る。65 秒や 110 秒、140 秒地点にてパケット再送処理後にウィンドウサイズにストリーム間で差が生じているが、DECP により速いストリームの速度変化が抑えられて、遅いストリームは HighSpeed TCP 本来の振る舞いにより速やかに他のストリームに追いつけている。ただし、これらのウィンドウサイズの縮退は、HighSpeed TCP を用いる副作用として並列 TCP の合計帯域がボトルネック帯域までにまだ余裕がある状態でありながらパケットロスが発生した事に起因している。

5　関連研究

Floyd の Limited Slow-Start [9] は Slow Start 時の RTT 時間当りのウィンドウサイズの増加量を $w > max\_ssthresh$ の場合 $max\_ssthresh/2$ に制限して、急激なウィンドウサイズの増加による大量のパケットロスを回避し性能低下を防ぐ。$max\_ssthresh$ の推奨値は 100 である。

Xu らの BIC-TCP [10] は CPS+ と同様に w_{max} に対して $a(w) = \frac{w_{max}-w}{2}$ をウィンドウサイズの増加量とする方式を採用している。また、$w > w_{max}$ の場合には Slow Start のように $a(w) = w - w_{max}$ を増加量とする。ただし, $S_{min} < a(w) < S_{max}$ という制限を設け、w_{max} は輻輳イベント時に再計算した値を用いている。

6　まとめ

本稿では、パケット送信レートを制御する TRC-TCP の 1 種である CPS を用いることで、Slow Start のよるパケット送信のバースト性を緩和し、並列 TCP の通信開始段階での性能を改善できる事を示

した。CPS によりウィンドウがボトルネック帯域を埋め尽くすまで成長する一方、Slow Start による急激で行き過ぎたウィンドウの成長により Fast Retransmit/Fast Recovery では回復できないようなパケットロスを発生させてしまう事が判った。

また、協調動作しストリーム間の転送速度差を緩和する DECP アルゴリズムと HighSpeed TCP を並列 TCP に用いることにより、転送速度のばらつきをすばやく解消でき、並列 TCP が足並みを揃えて現実的な成長速度でウィンドウサイズを大きくなるように制御できる事を示した。しかし、単純に HighSpeed TCP を並列動作させるだけでは、ボトルネック帯域を使い切る前にパケットロスが発生させてしまい、その動作が不安定になってしまう事が判った。

参考文献

[1] 中村, 来栖, 坂元, 古川, 生田, 下國, 下見, 陣崎, 玉造, 稲葉, 平木, “高レイテンシ環境下におけるデータレゼボワールの性能評価”, 情報処理学会研究報告 2003-HPC-93, pp.37-42, Mar. 2003.

[2] 亀澤 寛之, 中村 誠, 稲葉 真理, 平木 敬, 陣崎 明, 下見 淳一郎, 来栖 竜太郎, 中野 理, “長距離・高バンド幅通信における並列 TCP ストリーム間の調停の実現”, SACSIS2004 論文集, pp.425-432, May 2004.

[3] H. Kamezawa, M. Nakamura, M. Inaba, K. Hiraki, “Coordination between parallel TCP streams on Long Fat Pipe Network”, NETWORKING 2004, LNCS 3042, pp.962-973, May 2004.

[4] M. Nakamura, J. Senbon, Y. Sugawara, T. Itoh, M. Inaba, K. Hiraki, “End-node transmission rate control kind to intermediate routers - towards 10Gbps era”, 2nd International Workshop on Protocols for Fast Long-Distance Networks, Feb. 2004.

[5] 中村 誠, 稲葉 真理, 平木 敬, “ギガビットイーサネット上での遠距離 TCP 通信における Packet Spacing”, 電子情報通信学会技術研究報告 IA2003, pp.13-18, Oct. 2003.

[6] M. Nakamura, M. Inaba, K. Hiraki, “Fast Ethernet is sometimes faster than Gigabit Ethernet on LFN — Observation of congestion control of TCP streams”, 15th IASTED International Conference on Parallel and Distributed Computing and Systems, pp.854-859, Nov. 2003.

[7] N. Hu, P. Steenkiste, “Improving TCP Startup Performance using Active Measurements: Algorithm and Evanluation”, 11th IEEE International Conference on Network Protocols, pp.107-118, Nov. 2003.

[8] S. Floyd, “HighSpeed TCP for Large Congestion Windows”, RFC 3649, Dec. 2003.

[9] S. Floyd, “Limited Slow-Start for TCP with Large Congestion Windows”, RFC 3742, Mar. 2004.

[10] L. Xu, K. Harfoush, I. Rhee, “Binary Increase Congestion Control for Fast Long-Distance Networks”, Infocom 2004, Mar. 2004.

[11] http://dast.nlanr.net/Projects/Iperf/