通信ライブラリ Suci for Java の性能改善と評価

Performance improvement and evaluation of communication library "Suci for Java"

山城 潤 †, 河野真治 ‡
Jun Yamashiro, Shinji Kono
† 琉球大学理工学研究科情報工学専攻
Information Engineering Course, University of the Ryukyus.
yamajun@cr.ie.u-ryukyu.ac.jp
‡ 琉球大学工学部情報工学科
Dept. of Information Engineering, University of the Ryukyus.
kono@ie.u-ryukyu.ac.jp

本研究室が開発している通信ライブラリ Suci for Java は、PC クラスターのようなスループットとレスポンスの調整、あるいは、多数の競合的な通信が要求されるネットワークのために、UDP による通信に信頼性を付加しフロー制御機構を用意するものである。

Suci for Java のベースとなった C 言語版 Suci は、TCP より高いスループットを出していたが、当初の Suci for Java は TCP と比べて低いスループットしか出せなかった。

本論文では、Suci for Java における性能とその改善について論じる。

1 はじめに

PC クラスターの普及にともない、並列分散プログラムが実用的に使われるようになっており、並列分散アルゴリズムで使われるデータ構造も単純な配列から決定二進木、スパース行列など複雑なものが使われるようになっている。

現在、PC クラスターでは MPI や SCore のように、通信プロトコルに TCP を利用するライブラリが主に使われている。しかし、TCP はインターネットでの通信のために作られたプロトコルであり、主に専用ネットワークの上で構築される PC クラスターには適さない場合もある。また、TCP は連続的なストリーム通信を前提としているので、多数のノードに多様なメッセージを送る並列分散アルゴリズムの実装には適していない。例えば、通信する分だけソケットをオープンしなければならないし、メッセージのチェックもソケット分だけ行う必要がある。

そこで、本研究室では PC クラスター向けに、UDP ベースの通信ライブラリ Suci[1][2](Simple User level Communication Interface) を Prolog と C の組合せによって実装し、利用してきた。しかし、Prolog と C による実装では違う言語で作られたプログラム間のデータ受渡しでオーバーヘッドが生じるなどの不都合があった。

その問題を解決するために、Suci の Java による実装 Suci for Java を実装し昨年発表 [3] した。

しかし、開発当初の Suci for Java のパフォーマンスは十分なものとはいえるものではなく、その改良が必要であった。

本論文では、Suci for Java の改良とその性能に付いて述べる。

2 Suci の概要

Suci[1] は、PC クラスターのような高速、低損失なネットワーク上で利用することを前提として開発された Prolog と C による通信ライブラリである。

Suci の特徴は以下の通りである。

- UDP ベースの通信ライブラリ
- UDP 通信への信頼性付加
- ユーザーレベルフロー制御 API

Suci はトランスポート層プロトコルに UDP を利用することで、通信の軽量化をはかっている。また、

UDP に欠けている信頼性を補うために再送処理などのフロー制御 API を提供している。

これまでの研究 [2] では、C で実装された Suci はピンポン転送ベンチマークにおいて、C で実装された TCP プログラムの数倍程度スループットが向上することが明らかになっている。

2.1 ユーザーレベルフロー制御

PC クラスター上の通信は、インターネットと異なり、高速な LAN や専用ネットワークを使うことが多い。そのため、インターネットでは禁止されている帯域の独占的利用も必要ならば可能である。しかし、TCP ではそのようなネットワークの利用は TCP スタックが提供しているフロー制御機構によって制限される。

Suci では、フロー制御機構を持たない UDP を利用し、フロー制御 API をアプリケーション層で実装することで、ユーザーがフロー制御を自由に行うことができるようにしている。

3 Suci for Java

3.1 Suci for Java のクラス構成

Suci for Java は、API を提供するクラス `Suci`、パケットを表現するクラス `SuciPacket`、ノードのアドレスを保持するクラス `AddressDatabase`、個々のノードの IP アドレス、ポート番号を保持するクラス `SuciAddress` から構成されており、その関係は図 1 で表される。

3.2 改良前の性能

改良前の Suci for Java のスループットはピンポン転送ベンチマーク上で TCP の 42〜184 分の 1 しか出せなかった (図 2)。

4 パフォーマンスの改善

Suci for Java のパフォーマンスが低下していた要因には、オブジェクト生成 (`new` 構文) の多さと、送受信データのコピー (`System.arraycopy()` メソッド) 回数が不必要に多かったことにある。

ソースコードとプロファイラーの出力をチェックすることで、オブジェクトの生成をできる限り減らし、また可能な限りプログラム起動時にオブジェクトを生成するように変更することで、実行時のオブジェクト初期化の回数を半分以下に削減した。

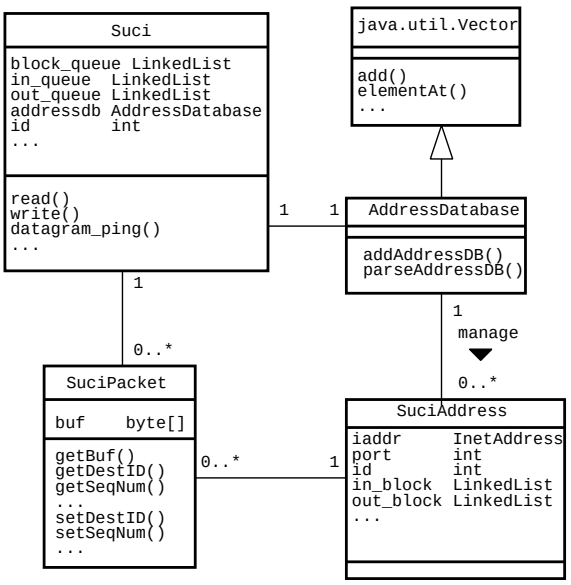

図 1: Suci for Java の UML クラス図

	改良前	改良後
ソース中の new の数	37	28
初期化回数	177691	82882
ソース中の arraycopy() の数	14	9
arraycopy() 実行回数	19780	15910

これらの改良により、Suci for Java のスループットは改良前との比較で最大 51 倍改善された (図 2)。

5 ベンチマーク結果

改良された Suci for Java の性能を確かめるために、本研究室の PC クラスター上の 2 つのノード間で通信を行い、新旧バージョンの Suci for Java と、TCP を使った Java プログラムとを比較したベンチマークテストを行った。

5.1 ピンポン転送

2 つのノード間でパケットを往復させ、スループットと往復時間を計測する。

ピンポン転送では、Suci for Java が TCP を上回るのは 1 度だけで、それ以外では常に TCP が 2〜53 倍高いパフォーマンスを記録している。

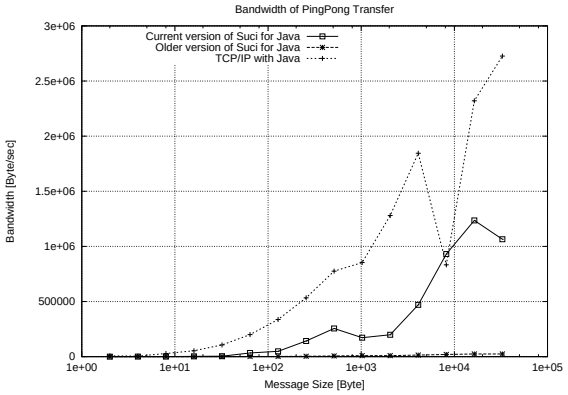

図 2: 新旧 Suci for Java と TCP とのスループット比較 (ピンポン転送)

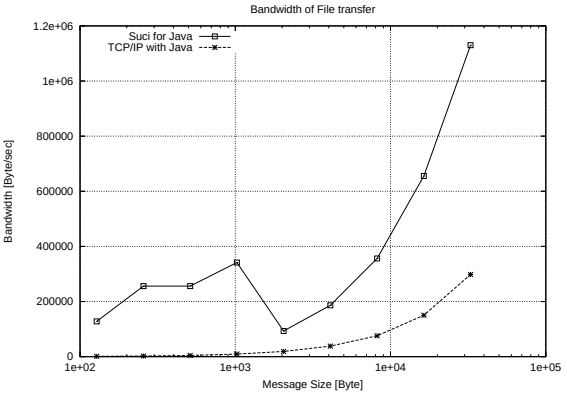

図 4: Suci for Java と TCP とのスループット比較 (ファイル転送)

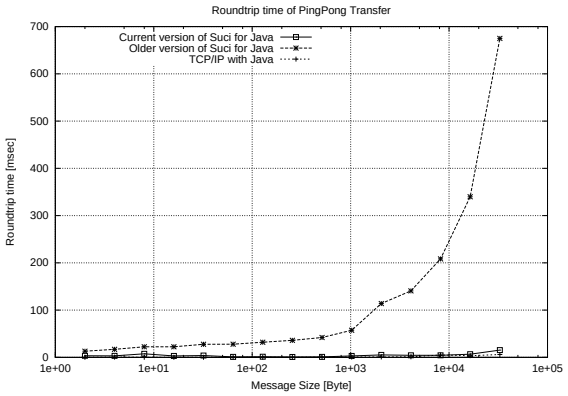

図 3: 新旧 Suci for Java と TCP との往復時間の比較 (ピンポン転送)

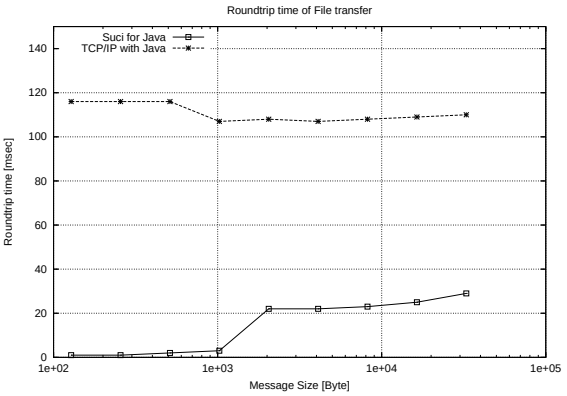

図 5: Suci for Java と TCP との往復時間の比較 (ファイル転送)

5.2 ファイル転送

ノード A からノード B へファイルを送信し、スループットと往復時間を計測する。

ファイル転送においては、Suci for Java は TCP を常に上回っており、4 倍〜116 倍のスループットを記録している。これは、Suci for Java は単純に一方通行にデータを送る能力に関しては TCP を上回るだけの性能を持っていることがわかる。

```
void main(String[] args) {
 // 初期化、ファイル読み込みなど

 //ostream.write(sendBuf); // TCP の場合
 suci.write(sendBuf);
 // 全パケットが送信されたことを
 // 確認するまで再送処理
}
```

図 6 ファイル送信プログラムの概要

```
void main(String[] args) {
 // 測定開始
 start = System.currentTimeMillis();

 // 受信処理
 // TCP の場合
 //while ((size=istream.read(readbuf)>0) {
```

6 考察

ピンポン転送とファイル転送での Suci for Java の性能の違いはベンチマークプログラムの違いに原因があると考えられる。

```
 while ((size=suci.read(readbuf)>0) {
   // 受信したパケットの ACK を送信 (Suci のみ)
   recvSize += size;
   if (recvSize >= full) break;
 }
 // 測定終了
 end = System.currentTimeMillis();
}
```

図 7 ファイル受信プログラムの概要

```
// クライアント側の疑似コード
// サーバーの場合は送受信処理が逆になる。
for（指定した範囲でパケットサイズを増加）{
 // 測定開始
 start = System.currentTimeMillis();
 for（パケット送受信の繰り返し）{
   // 送信処理
   //ostream.write(sendBuf); // TCP
   suci.write(sendBuf);
   // 全パケットが送信されたことを
   // 確認するまで再送処理 (Suci のみ)

   // 受信処理
   // TCP の場合
   //while ((size=istream.read(readbuf)>0) {
   while ((size=suci.read(readbuf)>0) {
    // 受信したパケットの ACK を送信 (Suci のみ)
    recvSize += size;
    if (recvSize >= full) break;
   }
 }
 // 測定終了
 end = System.currentTimeMillis();
}
```

図 8 ピンポン転送プログラムの概要

ファイル転送プログラムは 1 個のファイルを送って終了し、ピンポン転送プログラムは、同じサイズのパケットの送受信を指定の回数 (本論文のベンチマークでは 100 回) 繰り返している。そのため、パケット送受信の頻度が高くなることによる輻輳が原因と考えられたが、ピンポン転送での繰り返し回数を減らしても、性能差の変化はあまりなかった。

ファイル転送プログラムでは、送信と受信の処理は送信用/受信用プログラムに分担されているが、ピンポン転送プログラムでは、双方のノードが送信と受信両方の処理を行っており、測定中に送信のための `write()` メソッドと再送信のための処理が実行されていることが原因と考えられる。

しかし、正確にデータを送るためには再送処理を実行することは必要なことであり、 速度が要求される部分でのデータ送信を避けるようにプログラムを設計するか、再送処理の高速化を行う必要がある。

7 まとめ

改良の結果、不十分とはいえ Suci for Java の性能向上という目標はある程度達成された。しかし、C バージョンの Suci が達成している TCP より高いスループットは達成できていない。

今後は、TCP を越えるスループットを達成することと、本研究室の研究システムへの導入が課題になる。

参考文献

[1] 河野 真治, 神里 健司. UDP を使った分散計算環境とその応用. 日本ソフトウェア科学会第 16 回大会論文集, 1999

[2] 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予稿集, 2002

[3] 山城 潤, 河野 真治. Java によるユーザレベルトランスポート層の実現と評価. 情報処理学会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予稿集, 2003

[4] Jcluster `http://vip.6to23.com/jcluster/`