

VNC を用いた授業用画面共有システムの設計と実装

谷成 雄 大城 信康 河野 真治

各クライアントをツリー型に接続し、親が配信したデータをリレーさせることで分散 VNC アプリケーションを実装した。通常の VNC では配信者へ負荷が集中する設計となっている。例えば、大学の講義等で VNC を用いて画面共有を行った時、クライアントの増加に比例して配信者への負荷が増えてしまう。この問題を解決する為に、ツリー構造にクライアントを接続させ、Top のクライアントから子供へデータを送ることでスケーラビリティを持たせた。その結果、クライアントの数を増やしてもサーバ側への負荷を抑えることができた。また、VNC Reflector との性能比較も行う。

1 はじめに

普段授業を行う際、プロジェクタなどを使って授業を進めている。しかし、後ろの席から見えにくいなどの不便を感じるがよくある。授業をうけている生徒の手元にパソコンがあるならば、そこに先生のスライドを表示して授業を進めれば後ろの席に座っても手元に画面があるので見えづらいという問題は解消される。

VNC を使えば、スライドを生徒の手元の画面に表示することができる。しかし、多人数の生徒が先生のパソコンに同時に接続してしまうと処理性能が落ちて授業の進行に画面がついていかなくなってしまう。この問題は一つのコンピュータに多人数が繋がる時に起こる問題である。

本研究では多人数で画面共有ができるようにクライアントをツリー構造に接続させ、上から順番にデータを流していくという方法で新しい VNC クライアント、TreeVNC の設計と実装を行った。

2 VNC について

VNC(Virtual Network Computing) は、Rfb プロトコルを用いて遠隔操作を行うリモートデスクトップソフトである。VNC はサーバ側とクライアント(ビューア)側に分かれていて、サーバを起動し、クライアントがサーバに接続を行い遠隔操作を可能にする。

2.1 Rfb プロトコル

Rfb (remote frame buffer) プロトコルは、GUI 操作をリモートアクセスで行うためのプロトコルである。画面の描画の更新は画面の差分が発生した部分を矩形毎で送り描画される。また、画面の描画データに使われるエンコードが多数用意されており、また独自のエンコードを実装することもできるシンプルなプロトコルである。

3 TreeVNC の方針

まず、多人数が参加している授業で VNC を使う場合に起こる問題は、最初に述べたように、一つのコンピュータに多人数が繋がり、処理性能が大幅に落ちてしまうところが問題である。この問題を解決する為に、クライアント同士を接続させ、画面描画のデータ

を受け取ったクライアントが次のクライアントにデータを流すという方法を行うことにした。画面共有を行いたいクライアントが一種の VNC サーバ自体にもなる。また、クライアント同士の接続はツリー構造で行うことで管理がしやすくなると考えた。クライアント同士の接続の管理はツリーの一番上にいる PC(Top)で行い、この Top だけが VNC サーバへ接続を行うようにする。

今回作成した TreeVNC は、上記の実装でツリー状にクライアントを接続していくように実装を行った。画面の共有だけを行うように実装した。

4 先行事例

Vnc サーバへの負荷を軽減する先行事例としては Vnc Reflector がある。

4.1 Vnc Reflector

Vnc Reflector は、Vnc サーバとクライアントとの間に入り、Vnc サーバとの通信を代わりに行うプログラムである。クライアントは Vnc Reflector へ接続するので、Vnc サーバとの接続は Vnc Reflector のみとなり、Vnc サーバ側の負荷を減らすことができる。

5 TreeVNC の実装

TreeVNC は tightVNC の java 版のビューアを元に作成を行った。実装の細かい内容は以下で説明する。

5.1 tightVNC Viewer

tightVNC は tightVNC というプロトコルをサポートしたフリーの VNC 用ソフトである。2011 年 8 月 9 日現在と C++ で作成された VNC サーバ用と Windows 版、それと Java 版のビューアが公開されている。

5.2 tree structure

今回は、ホストに対しクライアントがツリー状に繋がっていくように実装した。ツリーの構成は以下の手順で行う。

1. クライアントが接続する際、ホストに接続をしているプロキシ（今後このプロキシのことを Top

と記述する）に接続する。

2. Top はクライアントにどこに接続すれば良いかを知らせる。（このときに親の番号と自分の番号それからリーダーであるかどうかを一緒に知らせる）
3. クライアントは Top から指定されたノードに接続を行う。

5.2.1 tree の再構成

今回の実装はクライアントがツリー状に繋がっているので、親ノードが落ちると子ノードも一緒に落ちてしまう。そこで、tree の再構成が必要になる。

1. 親ノードが落ちた際に、子ノードの中で一番若い番号の子ノードが Top に対して自分の親ノードが落ちたことを報告する。（親ノードの番号を知らせる）
2. Top は木の番号が一番大きいノードに対して 1 で報告を受けた親ノードの代わりになるように命令を出す。
3. 親ノードがいなくなった子ノードたちは Top に対して、2 で新しく繋がった親ノードの IP アドレスを教えてもらいそのアドレスに対して接続をおこなう。上記の構成の場合、一つのノードが落ちた場合に再接続を行うノードは 2 分木の場合 3 ノードである。

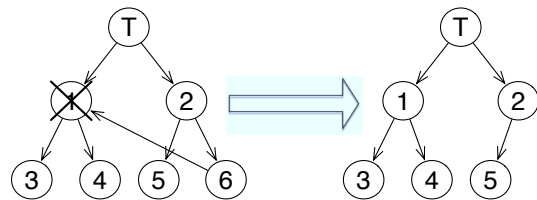


図 1 1 番の木が落ちたときの再接続の処理 (T は Top)

5.3 クライアントとの通信

TreeVNC は、受け取った画面の描画データをそのまま自分に繋がっている次のクライアントに送信する。描画データを受け取ったクライアントはまた次の

クライアントヘデータをそのまま送信する。内部では、まず受け取った描画データの読み込みを先に行い Bytebuffer でコピーを行う。次にクライアントへの送信と自身のビューアへの描画を並列に行う。

5.3.1 FramebufferUpdate

Rfb プロトコルでの画面の描画の更新は、FramebufferUpdate で行われる。FramebufferUpdate を受け取ることで画面の再描画が行われる。FrameBufferUpdate では、メッセージタイプと画面の矩形の数がまず送られ、次に x 座標、y 座標、横幅、縦幅、エンコードのタイプ、描画データが矩形の数だけ送られてくる。描画データはエンコードのタイプに従った方法で送られてくる。

5.3.2 描画データの先読み

1 回の FramebufferUpdate で送られてくるデータ量はエンコードタイプまで読みこめば知ることができる。例えば、RAW エンコードの場合は $width * height * 4$ のバイト数が送られてくる (4 はピクセルのデータ量である)。計算で求めた数の分とヘッダーを合わせた数だけバイトを読み込むことで 1 回分の FramebufferUpdate のコピーを行うことができる。

5.3.3 MulticastQueue

画面が更新された際に更新をクライアントに伝えなければならない。ノードが多数ある場合、一人一人に更新を知らせるのではなく、同時に画面の更新を知らせたい。同時に更新を知らせるために、CountDownLatch を用いて MultiCastQueue を作成した。

CountDownLatch 一回 CountDown されたときに待機しているスレッドを解放するように宣言する。更新情報が来るまで await を用いてスレッドを待機させる。更新情報が来たとき CountDown を行う。すると、スレッドが開放されるので同時に更新情報を参照することができる。

5.3.4 timeout

MultiCastQueue を使ったデータの取得には問題が発生した。それは、接続してきたクライアントがデータを取得しない状況、例えばサスペンド状態になったときに Top のメモリの中にデータが残ってしまうというものである。メモリに残り続けたデータはや

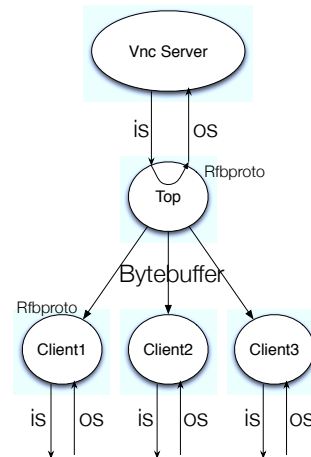


図2 クライアントへは並列にデータを送信する。

がメモリオーバーフローを引き起こしてしまうのである。その様子を図3に示す。

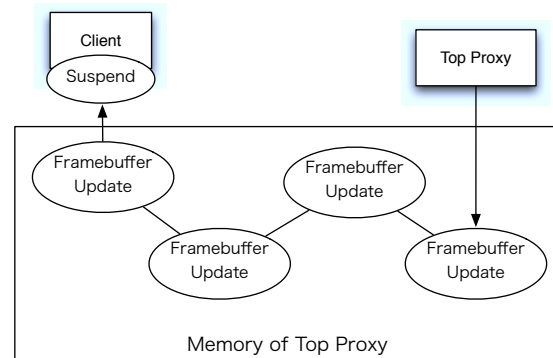


図3 クライアントサスペンド時の Top のメモリの様子。データが残り続けメモリを圧迫してしまう。

そこで、ある一定の時間がたつと代わりにデータを取得してくれる Timeout 用のスレッドを作成した。Timeout スレッドはサスペンドしているクライアントの代わりにデータを取得する。

Timeout スレッドがクライアントの代わりにデータを取得することで、MultiCastQueue の中からデータが削除され Top のメモリを圧迫することがなくなった。

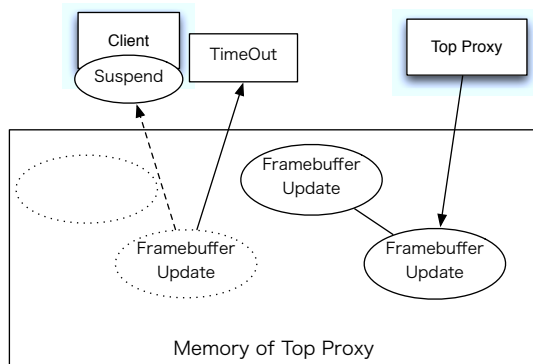


図 4 TimeOut が代わりにデータを取得する

5.4 圧縮の問題

VNC で扱う Rfb プロトコルには、使えるエンコーディングのタイプの 1 つとして ZRLE (Zlib Run-Length Encoding) がある。ZRLE は Zlib で圧縮されたデータとそのデータのバイト数がヘッダーとして付けられ送られてくる。Zlib はフリーのデータ圧縮及び解凍を行うライブラリである。可逆圧縮アルゴリズムの圧縮と解凍が行える `java.util.zip.deflater` と `java.util.zip.inflater` を実装している。

5.4.1 `java.util.zip.deflater` の実装の問題

Zlib 圧縮は辞書を持っていて、その辞書に登録されているデータを元に解凍が行われる。しかし、`java.util.zip.deflater` は現在持っている辞書を書き出すこと (flush) ができないことが分かった。辞書を書き出すことができない為、Zlib 圧縮されたデータを途中から受け取ってもデータが正しく解凍を行うことができない。

5.4.2 ZRLEE (ZRLE Economy)

そこで、Top が ZRLE で受け取ったデータを unzip し、データを zip し直して最後に `finish()` を入れることで初めからデータを読んでいなくても解凍を行えるようにした (毎回新しい辞書を使うようにした)。このエンコードは ZRLEE エンコードと定義した。一度 ZRLEE エンコードに変換してしまえば、そのデータをそのまま流すだけで良い。よって変換は Top が行う一回だけですむ。ただし、`deflater`, `inflater` では前回までの通信で得た辞書をクリアしないとイケな

いため、Top とクライアント側では毎回新しく作る必要がある (クライアント側は `inflater` だけ)。また、ZRLEE はクライアント側が対応していなければならないという問題がある。

5.4.3 ZRLE と ZRLEE のデータ圧縮率の比較

RAW, ZRLE, ZRLEE のデータ量の比較を行った。図 5 は 1920 * 1080 の画面の全描画にかかるデータ量を測った結果を示した図である。ZRLE は 3471994 バイト、ZRLEE は 3266544 バイトと、ZRLEE の方がデータ量が少ない。これは、ZRLE (Zlib) が初めに送られた辞書を用いての解凍が余り有効的に働いていない場合があるからだと思われる。つまり VNC の場合は ZRLEE の様に毎回辞書のデータを付加させて送ってもデータ量に差がでない可能性があることが分かった。

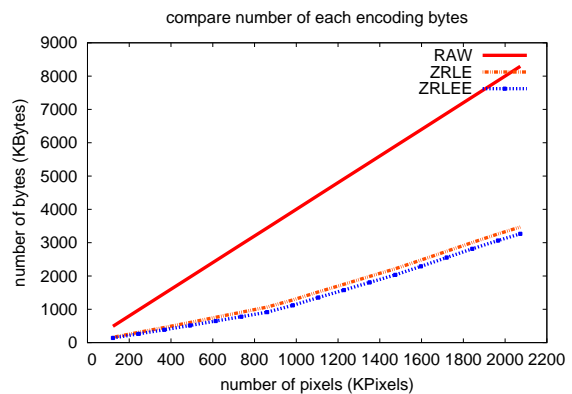


図 5 RAW, ZRLE, ZRLEE による 1 画面 (1920*1080) 描画にかかるデータ量。x 軸はピクセル数、y 軸はバイト数を表している。

6 評価

TreeVNC の実装を行い、Vnc Reflector との比較を行った。テスト環境は Blade サーバ上に VM を 48 台用意し、Blade サーバの外に TreeVNC と Vnc Reflector を起動させた PC を置き、VM 48 台にアクセスさせて実験を行った。

6.1 Vnc Reflector との比較結果

一極集中型の Vnc Reflector は、スループットが 2 メガから 5 キロまで落ちた。一方 TreeVNC の方はスループットを 2 メガのままをキープすることが確認できた。これは、1 本の通信帯へ 48 台がアクセスする Vnc Reflector と違い、クライアント同士が通信を行う為である。

6.2 TreeVNC の利点

1. クライアント同士がデータのやり取りをするので Vnc サーバへ負荷が少ない。
2. 一極集中型ではないので、多人数で使う際に画面表示のストレスが少ない。
3. Top 自身がビューアを持つこともできる。

6.3 TreeVNC の欠点

1. 独自のエンコーディング形式を使っているため、圧縮する場合専用のクライアントが必要。
2. みんなの通信速度が上がる分、スイッチへの負荷が高い。

7 まとめ

本研究では、VNC とツリー構造を用いて画面共有システムを開発した。結果スループットを落とすこと

なく 48 台で接続を行うことができた。

Zlib 圧縮に使用される `java.util.zip.deflater` の問題も発見したので、ZRLE の代わりとなる ZRLEE を用意した。

クライアントへのデータ転送をマルチキャストで行う等、実装には改良の余地がある。

参考文献

- [1] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber: Bigtable : A Distributed Storage System for Structured Data
- [2] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati , Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall , Werner Vogels: Dynamo: Amazon's Highly Available Key-value Store , SOSP (2007)
- [3] Shoshi TAMAKI, Shinji KONO: Cassandra を用いた CMS の PC クラスタを用いたスケーラビリティの検証, ソフトウェア科学会 (2010)
- [4] Shoshi TAMAKI, Shinji KONO, Yu TANINARI: Cassandra を使ったスケーラビリティのある CMS の設計, 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 (OS)