デスクトップ画像共有システムのための，トーナメントアルゴリズムを使った負荷分散機構

山之上卓[†1]　杉田裕次郎[†1]　小荒田裕理[†1]　小田謙太郎[†1]　下園幸一[†1]

HTML5 を使ったデスクトップ画像共有システムにおいて，大量の Web クライアント間で短い遅延を保ったまま，画像共有を実現するための，負荷分散機構について述べる．この画像共有システムは複数の Web サーバと大量の Web クライアントによって実現されている．この負荷分散機構はトーナメントアルゴリズムを用いて実現している．80 台の端末を用いて本システムおよびアルゴリズムの有効性を確認した．

A Load-balancing Mechanics using Tournament Algorithm for a PC Desktop Screen Sharing System

TAKASHI YAMANOUE[†1]　YUJIRO SUGITA　KENTARO ODA[†2]
YUURI KOARATA　KOICH SHIMOZONO

1. はじめに

我々は授業やゼミや会議のとき，パソコン画面をプロジェクタでスクリーンに投影することをよく行う．しかしながら，この方法では，遠くから小さな字や絵を見ることが困難になる．このような問題に対処するため，SOLAR-CATS[1][4] や Multi VNC[7] などの PC 間画像放送システムが存在する．しかしながら，これらのシステムは，会議参加者それぞれに，これらのソフトウェアをインストールしたパソコンが必要になる．

最近，我々の多くはスマートフォンやノートパソコンを携帯している．もし，発表者のパソコンのデスクトップ画像を，このような機器で，特別なソフトウェアやアプリをインストールすることなしに，共有することができたら，授業や会議やゼミは，より効果的になる可能性がある．我々は，このような要求を満たす「Web Screen Share」[5]を開発し，我々のゼミで利用している．Web Screen Share は，HTML5 の Web Socket 技術を使ったパソコンデスクトップ画面共有システムである．これは，一般的なパソコンやスマートフォンで利用可能な，複数の，HTML5 対応 Web ブラウザに，画像中継 Web サーバを通じて，送信元パソコンのデスクトップ画面をリアルタイムで配信する．

Web Screen Share は，参加者が２０名程度までなら，問題なく動作していた．しかしながら，この人数を超えると，動画を表示するときに，動きの滑らかさが失われてしまうことが分かった．

我々は，この問題を，平衡２分木上に接続されたサーバ間データ転送ノードを使うことによって，解消した．送信元パソコンのデスクトップ画像は，このサーバ間データ転送ノードを使って複数の画像中継 Web サーバに配送される．Web ブラウザは，複数の画像中継 Web サーバに，自動的に，均等に，接続される．問題を解消した Web Screen Share を，Distributed Web Screen Share と呼ぶ．
この論文では，Web ブラウザの接続を，自動的に，均等に，複数の Web サーバに割り当てる機構を，トーナメントアルゴリズムを使って実現したことと，Distributed Web Screen Share によって，動画表示の動きの滑らかさを損なうことなく，より多くの Web クライアントによって同じデスクトップ画面を共有できたことについて述べる．

2. Web Screen Share

Web Screen Share[5]は Client-Server 型の，HTML5 の Web Socket 技術を使った，パソコンデスクトップ画像を実時間で共有システムである．図 1 に Web Screen Share の概要を示す．Web Screen Share は 1 台の Web Server と，複数の Web Client と，画像取得とその画面をサーバに送信する *Screen Sender* から構成されている．

Screen Sender はデスクトップ画像を繰り返し取得し，通常の Socket を使って Web Server に送信する．Screen Sender から画像を受け取った Web Server は，その画像を各 Web Client に対応した Queue の最後に追加する．Queue

†1 鹿児島大学
†2 マルチメディア，分散，協調とモバイルシンポジウム

に溜まった画像の量が一定値を超えると，その値を超えないように Queue の先頭部分にある画像を削除する．現在，この一定値は 3 としている．

Web Client は HTML5 の Web Socket を使って Web Server に対して get コマンドを繰り返し発行する．Web Server が get コマンドを受け取ると，それぞれの Web Client に対応した Queue の先頭にある画像を，get コマンドを受け取った Socket に対して送信し，その画像を Queue から削除する．画像を受け取った Web Client はその画像を表示する．開発当時は Web Socket が利用できる Web サーバとして Jetty を利用した．

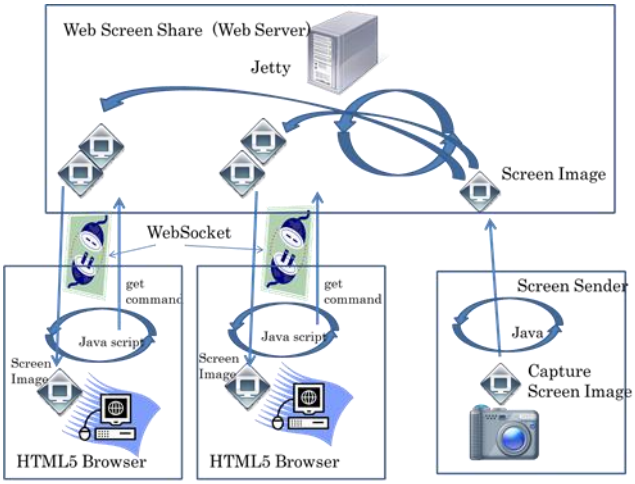

図 1. Web Screen Share

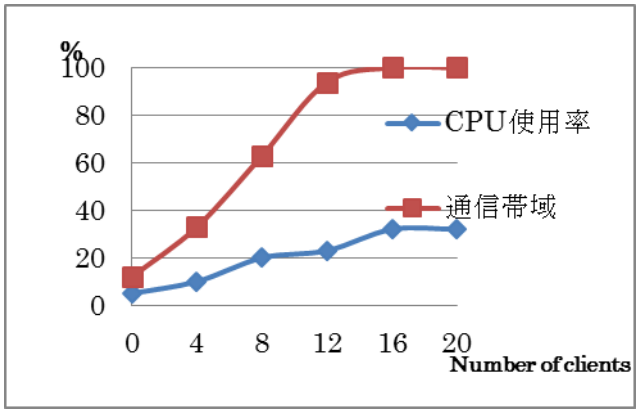

図 2. Web Screen Share のサーバの接続端末数増加による負荷の変化

Web Screen Share に接続する台数により，Web Server の CPU 使用率と通信帯域がどのように変化するかを計測した．ここで利用した機材は，Web Server, Web Client, Screen Sender のいずれも，CPU はインテル Celeron デュアルコア E1200 1.6GHz 相当で，メモリは 2GB であった．また，Screen Sender は 5fps でデスクトップ画面(解像度 1440 x 900)を Web Server に送信させた．NIC は 1Gbps であるが，ネットワークスイッチは 100Mbps を利用した．

この結果，Web Client 数が 12 を超えるとネットワークの利用は 100% に近くなり，これ以上 Web クライアント数が増えると Web Client に送信される単位時間あたりの情報量が減少することが予想される．CPU 利用率も 12 台で 20% を超えており，例えネットワーク容量が十分あったとしても，Web クライアント数が 60 を超えると，サーバ側の処理能力が低下し，Web Client での動画表示性能が劣化することが予測される．

3. Distributed Web Screen Share

大規模な会議にも対応できるようにするため，数百台の Web クライアントが画面共有できるようにしたい．Web クライアント増加による性能劣化を緩和するため，Distributed Web Screen Share (DWSS)を開発した．DWSS は *Node System* を平衡 2 分木状に接続し，葉の位置にある *Node system* の Web Server に Web Client をできるだけ均等に割り当てるようにするものである(図 3). Node System は Web Server と ノード間を接続するための *Inter Server Connection Node* (ISC Node) を接続したものである．Screen Sender は DWSS のどれか 1 つの Web Server に接続され，Screen Sender で取得された画像はこの Web Server を経由して，すべての Node System の Web Server に転送される．画像のすべてまたは 1 部が Queue を通じて Web Client に送信され，そこで表示される．

平衡 2 分木のすべての ISC Node に画像を中継するため，以下の *multi-casting* 手続きを利用する．

if I receive a message of an image from one of the links
then
forward the message to other links;
process the message;
fi

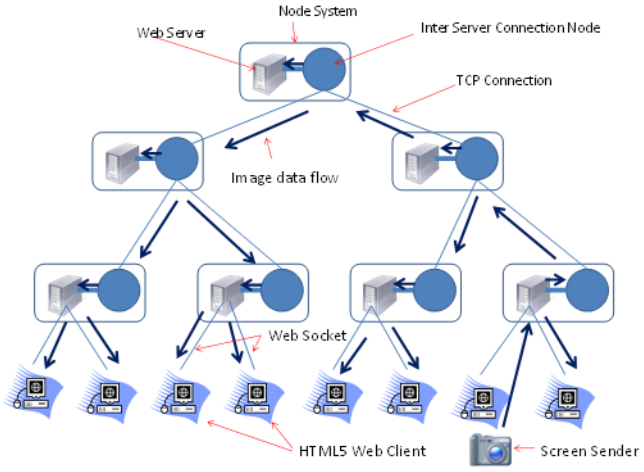

図 3. Distributed Web Screen Share (DWSS)の概要

平衡 2 分木状にノードが接続されているため，すべてのノード間の通信速度と ISC Node の性能が同じであれば Screen Sender から送信された画像がすべての葉ノードの Web Server に届くまでの時間は木の高さに比例する．

4. ISC ノードによる平衡２分木の構成

Node System を平衡 2 分木状に接続するため，文献[6]のアルゴリズムを利用している．このアルゴリズムを図 4 に示す．図 4 の中で利用される手続きを図 5 で示す．2 分木を構成する ISC ノードの集合を「グループ」とする。このアルゴリズムは２分木の任意の ISC ノードにおいて，一定期間ごとに左右の子孫のノード数の合計値を左右の子孫から受け取り，それぞれ値を left weight, right weight として記憶し，自分が根ノードでない場合はその合計値を親ノードに送信する．このときの通信は ISC ノードの生き死に確認にも使われる．グループに新たに ISC ノードが加わるとき，新たなノードはグループの根ノードに対して，「参加の問い合わせ」を行う．任意のノードに新たなノードの参加が問い合わされたとき， 左右の子孫の数を比較して，再帰

- I am an ISC node, and I have the following variables.

var
up_link, left_link, right_link : tcp_connection;
left_weight, right_weight : integer;

NULL is assigned to *up_link, left_link,* and *right_link,* and zero is assigned to *left_weight* and *right_weight,* initially.

- If I am the root node of the group, start the procedure of *balancing*.

```
procedure    balancing;
co_begin
     start the procedure of message_transmitting;
     start the procedure of message_receiving;
co_end
```

- When I would like to be a member of the group, start the procedure of *join*.

```
procedure join;
begin
       send the join message, which includes the address of
       this node, to the root node of the group;
      wait for receiving the message which shows the address
       of the member node and left_link or right_link;
      connect this up_link to the link at the member node;
      when this up_link is connected, this node become the
       member and start the procedure of balancing;
end
```

- When I am not the root node and a member of the group, and the node at the up_link seems down, start the *join* procedure.

図 4. 平衡 2 分木構成アルゴリズム

```
procedure message_transmitting;
begin
     left_weight:=0;    right_weight:=0;
     repeat forever
       send left_weight+right_weight + 1 to the node at
            up_link if up_link is not NULL, using the up_link;
        wait interval_term_for_weight_transmitting;
     taeper
end;

procedure message_receiving;
begin
     repeat forever
     if I receive the weight from left_link then assign the weight
          to the left_weight; fi
     if I receive the weight from right_link then assign the
          weight to the right_weight; fi
     if I receive join message with the address then execute the
          procedure of add_node(address); fi
     taeper
end

procedure add_node(address);
begin
     if left_link is NULL then
       tell the node at the address to connect to the left_link of
        mine;
       add one to left_weight;
     else
     if right_link is NULL then
      tell the node at the address to connect to the right_link
        of mine;
      add one to right_weight;
     else
     if left_weight <= right weight then
       forward the join message to the node at the lift_link;
      add one to left_weight;
      else
        forward the join message to the node at the right_link;
        add one to right_weight;
      fi
end
```

図 5. 平衡 2 分木構成アルゴリズムで利用する手続き

的にノード数が少ない側の子ノードに参加の問い合わせが行われ，左右のどちらかの子ノードを持っていないノードにおいて，新規参加ノードがそのノードの子ノードとして接続される．左右のバランスが崩れている部分があっても，新しいノードが次々と参加するとき，根ノードから再帰的なノード参加処理が行われることにより，左右のバランスの改善が行われる．根ノード以外の任意の節ノードに障害が発生した場合，障害が発生したノードの左右の子ノードは，根ノードに対して自分自身の追加を問い合わせる．

5. トーナメントアルゴリズムによる負荷分散

Web Client の接続先により性能の差ができるだけ生じないように，Web Client を Node System の葉ノードにある

Web Server にできるだけ均等に割り当てるようにして負荷分散を行う. これを実現するため, 図 6 で示すトーナメントアルゴリズムを利用する. 図 6 のアルゴリズムで利用する手続きを図 7 で示す. このアルゴリズムは, 葉ノードの Node System においては, その Web Server に接続された Web Client 数を, その Web Server の URL と共に, 親ノードに送信することを一定期間ごとに繰り返す. 任意の節ノードにおいては, 左右の子ノードから送られてきた Web クライアント数を比較し, Web Client 数が少ないほうの URL を勝者として記録し, その Web Client 数を Web Server の URL と共に親ノードに送信することを一定期間ごとに行う. 根ノードにおける勝者が全 Web Server のうち, 最も Web Client 数が少ない Web Server の URL となる. 新規に Web Client が DWSS に接続するとき, その Web Client はまず, 根の Node system に接続を行い, 次に, 根の Node system に記述されている勝者の URL に re-direct されることにより, 最も Web Client の少ない Web Server に新規の Web Client が接続する.

このアルゴリズムを DWSS に実装し, 実際に 80 台の Web Client による画面共有を行い, Node System の数と Web Client 数の増加によりどのくらい性能に影響があるか計測した. ここで性能は Web Client での単位時間当たりの画像表示枚数(fps)とした. Screen Sender からは毎秒 5 枚の画像が送られる. 図 8 に実験の様子を示し, 図 9 に実験結果を示す. Node System はすべて実験結果で示したように, Node System を 3 から 7 に増やすことにより, fps の低下が抑えられていることがわかる. なお, この実験では, Web Client は葉ノードの Web Server のみに接続されているため, Web Client が接続される Web Server は Node system の数が 3 のとき 2, 7 のとき 4 となる.

- I am an ISC node and I also have the following variables. The right hand side value of each assignment is the initial value.

```
var
   left_candidate_host_address :=NULL;
   right_candidate_host_address:=NULL;
   candidate_host_address:=
      the web server address at me;
   number_of_left_candidate_clients:=infinite,
   number_of_right_candidate_clients:=infinite,
   number_of_candidate_clients:=0,
   number_of_max_clients:=infinite
   node_assignment_policy:="leafFair";
```

- When I am a member of the group, execute the procedure of *get_candidate*

```
procedure   get_candidate;
co_begin
    start the procedure of message_transmitting2;
    start the procedure of message_receiving2;
co_end
```

図 6. 負荷分散を行うトーナメントアルゴリズム

```
procedure message_transmitting2;
begin
    repeat forever
        number_of_candidate_clients:=
            current clients number of the web server at me;
    if number_of_left_candidate_clients <=
       number_of_right_candidate_clients
                and
           number_of_left_candidate_clients <=
           number_of_max_clients
        then
       candidate_host_address:=
           left_candidate_host_address;
       number_of_candidate_clients:=
           number_of_left_candidate_clients;
       else
    if number_of_right_candidate_clients <
        number_of_left_candidate_clients
                    and
          number_of_right_candidate_clients <=
          number_of_max_clients
        then
       candidate_host_address:=
           right_candidate_host_address;
       number_of_candidate_clients:=
          number_of_right_candidate_clients;
           fi
    if node_assignment_policy is "leafFair" then
          send values of candidate_host_address   and

number_of_the_candidate_clients
          to the node at up_link of the node;
     fi
        wait interval_term_for_candidate_transmitting
    taeper
end

procedure message_receiving2;
begin
   repeat forever
    if I receive the candidate_host_address, and
       number_of_candidate_clients from left_link
    then
       left_candidate_host_address:=received address;
       number_of_left_candidate_clients:=
           received clients number;
     else
    if I receive the candidate_host_address, and
       number_of_candidate_clients from right_link
    then
       right_candidate_host_address:=received address;
       number_of_right_candidate_clients:=
           received clients number;
     fi
   taeper
end
```

図 7. 図 6 のトーナメントアルゴリズムで利用する手続き

図 8. 80 台の端末で DWSS を使っている様子.

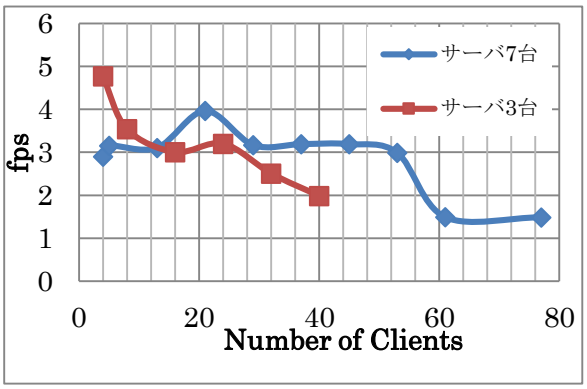

図 9. Web Client 数を変化させたときの性能 (Web クライアントで表示される動画の FPS)の変化.

6. アルゴリズムの拡張

DWSS は Web Client の増加による性能低下を緩和するために, 1 台の Web Server に接続可能な Web Client 数の上限を設定し, あふれた Web Client を節ノードに配置する機能や, 動的に Node System を追加する機能を持っている. また, Web Client 数が減少している状況において, 葉の Node System に接続される Web Client 追加を停止することにより, 葉の Node System の方から, これに接続される Web Client 数を 0 にしていき, 葉の Node System を削除可能にする機能も持っている. 葉ノードの方から空にすることにより, Node System の削除のノード再配置を省くことが可能となり, DWSS に参加している Web Client への影響を小さくすることができる.

図 6 のアルゴリズムの *number_of_max_clients* は 1 台の Web Server に接続される Web Client の最大値を表す変数である. この変数の値を状況によって変更させたい場合がある. グループ内の, すべての変数の値を一斉に変更するため, 第 3 章の multi-casting 手続きを利用する.

7. 関連研究

7.1 SOLAR-CATS

我々は SOLAR-CATS[1][4]を開発している. これは気軽に利用できる教育支援システムであり, デスクトップ画面を共有する機能も持っている. また, SOLAR-CATS は DWSS と同様に, 2 分木を使った構造的 P2P を利用している. しかしながら, SOLAR –CATS は一般的な Web ブラウザで利用することができない.

7.2 拡張 Edutab

鈴木らは HTML5 技術を使った電子薄板の共有を行う遠隔教育支援システムを開発している[7]. このシステムは 1 台のサーバを利用しているため, 接続されるクライアント数が増加すると描画スピードなどの性能が劣化する恐れがある.

7.3 AKAMAI

Akamai[8] は有名な商用 CDN である. DWSS も一種の CDN である. Akamai は世界中に同じコンテンツを配信するサーバを持っており, DNS re-direction を使って, コンテンツ利用者の Client が, そのクライアントから近い位置にあるサーバからコンテンツを利用できるようにする. TCP のバンド幅は遅延に大きく影響するため, Akamai のこの機能は効果が大きい. しかしながら, DWSS が想定しているような, 同じ場所にある多くのクライアントが一斉に同じコンテンツを共有するような場合, 負荷が一か所のサーバに集中してしまう.

7.4 FCAN

吉田は Flash Crowds Alleviation Network (FCAN)[3] を提唱している. FCAN は proxy server を組織化し, 予測できない急な Client の増加(flash crows)が発生した場合, Client は proxy server を利用する. しかしながら FCAN を利用するためには特殊な負荷分散を行うための特殊な DNS を利用し, DNS を変更するなどの操作も必要となる.

7.5 Ustream Live Producer

一般的なインターネット利用者が, Ustream Live Producer[9] を使うことにより, 気軽に多くの視聴者へ放送をを行うことが可能である. Ustream Live Producer はデスクトップ画面の配信を行うことも可能である. しかしながら, DWSS が対象とする配信は, 声が聞こえる範囲内の大量の端末に対するデータ配信であり, Ustream のように数十秒を超える遅延が発生するサービスでは利用が困難である.

7.6 Web RTC

Web RTC[10] は W3C で提唱されているブラウザ間の音声, ビデオチャット, P2P ファイル共有を可能にする規

格であり, Chrome や Firefox で利用可能である. DWSS に Web RTC を導入することにより, DWSS の性能をより高めることが可能となる.

8. おわりに

数百名の参加者があるような, 大規模な会議において, 参加者が持参するスマートフォンやノート PC で発表者のデスクトップ画面を共有することを可能とするシステム DWSS と, DWSS に接続される Web Client の負荷分散のために利用されているトーナメントアルゴリズムについて述べた. ノード数が減っていくときの負荷分散についてはこの論文では述べていないが, Web Client が定期的に根ノードを re-load するような HTML タグを利用することにより, このときの負荷分散も可能となる.

参考文献

1) 山之上　卓: P2P 技術を利用した分散システム上の実時間操作共有システム, 情報処理学会論文誌, vol.46, No.2, p.392-402 (Feb. 2005).
2) 芝崎 亮,千葉 大作,中沢 実,服部 進実: IT 教育向けデスクトップ管理ツール「MultiVNC」の実践報告, 情報処理学会コンピュータと教育研究会情報教育シンポジウム, Vol.2005, pp. 69-74, (Aug. 2005).
3) Norihiko Yoshida: Dynamic CDN against Flash Crowds, Springer Content Delivery Ndtworks: Principles and Paradigms (Rajkumar Buyya, Al-Mukaddim Khan Pathan, and Athena Vakari, eds), Springer, 277-298(2008).
4) Takashi Yamanoue: A Casual Teaching Tool for Large Size Computer Laboratories and Small Size Seminar Classes, Proceedings of the 37th annual ACM SIGUCCS conference on User services, pp.211-216, St.Louis, Missouri, US. (11-14, Oct. 2009).
5) 杉田 裕次郎, 小田 謙太郎, 下園 幸一, 山之上 卓: アドホックな環境で利用可能な Web ベースの画面共有システム, 電気関係学会九州支部第 64 回連合大会, (2011).
6) 山之上卓: 音声画像通信システム, 特許 No. 5186624, (Feb. 2013).
7) 鈴木 新一 , 水越 一貴 , 深澤 昌志 , 八代 一浩 , 鳥養 映子: 学校間ネットワーク上に構築した遠隔教育支援システムの接続手法の提案とその評価, 情報処理学会論文誌, vol. 54, No. 3, pp. 1050-1060 (March, 2013).
8) Akamai, http://www.akamai.com/index.html, as is March, 2013.
9) Ustream Producer, http://www.ustream.tv/producer, (As of May 17, 2013)
10) Web RTC, http://www.webrtc.org/, (As of May 17, 2013)