Web Workers を用いた演算処理性能の差による CPU コア数の推定

桐生 直輝† 磯 侑斗† 金子 洋平† 齋藤 孝道‡

†明治大学大学院
214-8571 神奈川県川崎市多摩区東三田 1-1-1
{ce36022, ce36004, ce36017}@meiji.ac.jp

‡明治大学
214-8571 神奈川県川崎市多摩区東三田 1-1-1
saito@cs.meiji.ac.jp

あらまし HTTPヘッダやJavaScriptなどから採取できる情報を用いてWebサイト閲覧者を識別する手法が知られている. 識別を好ましく思わない利用者の中には, 対策ツールを用いて, 識別情報の採取を避ける者もいる. しかし, 端末の処理性能を識別に用いる場合, 従来の対策ツールでは有効でない場合がある. その一例として, 本論文では, HTML5 APIの一つであるWeb Workersを用いることで, 利用者端末のマルチスレッド処理の性能を測定し, CPUコア数とHyper Threading Technologyの有効/無効を, Web経由で推定することが可能であることを示す.

Estimation of Number of CPU Cores Using with Web Workers

Naoki KIRYU† Yuto ISO† Yohei KANEKO† Takamichi SAITO‡

†Graduate School of Meiji University
1-1-1, Higashimita, Tama-ku, Kawasaki-shi, Kanagawa, 214-8571, JAPAN
{ce36022, ce36004, ce36017}@meiji.ac.jp
‡Meiji University
1-1-1, Higashimita, Tama-ku, Kawasaki-shi, Kanagawa, 214-8571, JAPAN
saito@cs.meiji.ac.jp

Abstract It is getting popular to identify visiting users by information collected with HTTP headers or JavaScript. Someone who doesn't want to be identified uses countermeasure tools for avoiding collection of the information. However, in cases of identifying with hardware's performance, users cannot avoid to be identified even with those tools. As an example, in this paper, we show a way of estimating the number of CPU cores and enable/disable Hyper Threading Technology by measuring performance of multi-thread process with Web Workers API.

1 はじめに

Web 技術の発展により, Web を利用した様々な商業目的のサービスが提供されるようになった. 商業目的の Web サイトの運営者は, 利用者の Web ページ閲覧状況から, 利用者の嗜好を判断し, 効果的な広告を配信する, 所謂行動ターゲティング広告の導入を始めている.

一般的に，利用者の端末の識別は HTTP クッキー（正しくは HTTP クッキーを用いてやり取りされる識別子）を用いることによって行われる．広告事業者などは，複数の Web サイトにコンテンツを埋め込み，サードパーティクッキーによって利用者の端末を識別し，利用者の Web ページ閲覧状況を収集する．これを Web 行動追跡と呼び，行動ターゲティング広告に利用する．

Web 行動追跡は，利用者の許可を得ることなく行われるので，プライバシーの観点で問題視されている．そこで，HTTP クッキーによる Web 行動追跡の拒否を希望するための仕様である Do Not Track（DNT）[13]の策定を始めとする，Web 行動追跡を拒否するための技術が普及してきている．

近年，新たな端末の識別方法として，Web Browser Fingerprinting と呼ばれる技術が登場した．Web Browser Fingerprinting とは，Web ブラウザと Web サーバが通信する際にやり取りされる HTTP ヘッダや，JavaScript などから採取される情報を用いることで，利用者の端末を識別する手法である．これに対して，採取される情報の書換えや制限によって，端末の識別を防ぐ技術が開発されている[8][9][10]が，実際には値の書換えが困難な情報が存在するので，対策は不完全である．

他方，HTML の新しい仕様である HTML5 の機能や JavaScript API が，ブラウザに実装され始めた．その中の一つである Web Workers API は，従来シングルスレッドで実行されていた JavaScript の処理を，マルチスレッドで実行することを可能とした．

本論文では，HTML5 API の一つである Web Workers API を用いたマルチスレッド処理の処理性能の差によって，Web Browser Fingerprinting に用いる情報の一つとして，利用者端末の CPU コア数に加えて，Hyper Threading Technology の有効/無効の推定方式を提案し，その実装と評価を示す．ここで，処理時間を基にした推定は，Support Vector Machine（以降，SVM という）を用いて行う．

2 Web Browser Fingerprinting

本節では，現在使用されている Web Browser Fingerprint や近年の研究で新たに登場した Web Browser Fingerprint，及び Web Browser Fingerprinting への対策について説明する．

2.1 Web Browser Fingerprint

Web ブラウザが Web サーバへアクセスする際，Web サーバが取得できる Web ブラウザの情報を特徴点と呼ぶ．特に，端末や Web ブラウザの利用者の特定につながるものを指す．特徴点を 1 つ以上組み合わせたものを，Web Browser Fingerprint と呼ぶ．本論文では，以降 Fingerprint と呼ぶこととする．特徴点の組合せから生成したハッシュ値や，単一の特徴点を Fingerprint と呼ぶこともある．

Web ブラウザが Web サーバへアクセスした際に，Web サーバが Web ブラウザから Fingerprint を採取する行為を，Web Browser Fingerprinting と呼ぶ．本論文では，以降 Fingerprinting と呼ぶこととする．アクセスしてきた Web ブラウザに対して Fingerprint を採取するための JavaScript コードを送信し，Web ブラウザで実行した結果を Web サーバへ転送させることで，Fingerprinting を行うことができる．

2.2 既存の Fingerprint に関する研究

Eckersley[4]は，Fingerprinting の研究を行い，JavaScript や Flash などから採取できる Fingerprint を用いて，利用者の端末を 94.2%一意に識別できることを示した．Eckersley が使用した特徴点には，User-Agent，HTTP-ACCEPT ヘッダ，ブラウザのプラグイン，タイムゾーン，スクリーンサイズ，システムフォント，HTTP クッキーの使用の可否，supercookie に利用されるストレージの対応などがある．

Mowery[5]は，HTML5 の canvas API や WebGL を用いて文字列や画像を描画し，描画結果の画素レベルの差から利用者端末の GPU，OS，及び，Web ブラウザの組合せが特定できることを示し，300 個のサンプルを収集した際のエントロピーは 5.73 ビットであったとある．また，Mowery ら[6]は，JavaScript を用いたベンチマークを用いて，Web ブラウザの特定，OS や CPU の識別を行い，その実験結果を示している．Web ブラウザの識別に関しては 98.2%の確率で識別できるとしている．

Mulazzani ら[7]は，JavaScript エンジンの実装状況を特徴点として捉え，Web ブラウザの識別を行った．ここでの実装状況とは，JavaScript エンジンが正しく実装されているか否かを指す．

著者らの研究グループでは，クライアント端末のハードウェアの特徴点の採取を行っており，文献[1]では，CPU のコア数，GPU レンダリングの有無，メディアデバイスの有無や，タブレット端末は表示向きなどの情報の採取に成功している．また，文献[2]では，Mowery ら[6]のアプローチと同じとなるが，JavaScript の様々な処理のパフォーマンスによるベンチマークの採取により端末を推定し，それを特徴点とすることを独自に試みた．さらに，文献[3]では，CPU 拡張命令である SSE2（Streaming SIMD Extensions 2）の対応の有無による CPU アーキテクチャの推測に成功している．

2.3 Fingerprinting 対策技術

Fingerprintingには，対策ツールが存在する．その例として，Ghostery Enterpriseが開発したGhostery[8]や，Mozilla Firefoxのアドオンであ るFireGloves[9]，通信を匿名化するWebブラウザであるTor Browser Bundle[10]などがある．

GhosteryはWebブラウザの拡張機能の一つである．予めGhosteryがWeb行動追跡を行っていることが既知であるドメインをブラックリストとして保持しており，ユーザがリスト内にあるドメインへアクセスしようとした際に検知し，Web行動追跡をブロックする．

FireGlovesは採取されるFingerprintの書換えや，Webブラウザの機能の制限することによって，サーバが正確なFingerprintを採取することを防ぐ．しかし，FireGlovesを用いた際，画面解像度などの情報が書き換えられると，Webブラウザ上での描画が崩れてしまうなどの，デメリットも存在する．また，Fingerprintを書き換えた結果，通常ありえないFingerprintが生成されてしまうと，それ自体が一意なFingerprintとして利用されてしまうこともある．

Tor Browser Bundleは匿名でWebブラウジングを行うことを目的としたWebブラウザである．Tor Browser BundleはTorを利用するので，IPアドレスなどTCP/IPに関する情報を秘匿することが可能である．更に，プラグインの利用やFingerprintingに利用される可能性のあるAPIの利用を制限しており，Fingerprintingへの対策も施されている．

3 周辺技術

本節では，提案方式に用いる Web Workers API，CPU の並列処理を実現する技術である Hyper Threading Technology，2 クラス分類の機械学習手法の SVM について説明する．

3.1 Web Workers API

Web Workers API は，W3C の Web Workers[11]で定義されている（広義の）HTML5 API である．これは，JavaScript にスレッド機能を提供する．Web Workers API を用いない場合，JavaScript はシングルスレッドで処理される．そのため，高負荷な処理を実行する際，他の処理がブロックされてしまう．Web Workers API を用いると，JavaScript をマルチスレッドで処理することが可能となるので，高負荷な処理にブロックされることなく，他の処理を行うことができる．Web Workers API で提供されるスレッドは Worker と呼ばれ，メッセージパッシングによって Worker 間の通信を行う．以下に Web Workers の使用例を示す（図 1，

図 2 参照).

```
1  var worker = new Worker('worker.js');
2  worker.onmessage = function(e){
3    console.log(e.data);
4  }
5  worker.postMessage('');
```

図 1. Worker の生成と呼出の例

```
1  onmessage = function(e){
2    postMessage('received: '+e.data);
3  }
```

図 2. Worker による処理の例(worker.js)

図 1 に示した JavaScript コードでは，Worker コンストラクタによって worker.js の処理を実行する Worker オブジェクトを生成する．次に worker.js からのメッセージを受け取った際の処理を onmessage イベントハンドラとして登録する． postMessage メソッドを用いて，Worker にメッセージを送信することで，Worker に処理を開始させる．図 2 に示した JavaScript コードでは，onmessage イベントハンドラを用いて Worker の処理を定義している．処理の最後で，postMessage メソッドを用いて，呼び出し元に処理の完了を通知する．

3.2 Hyper Threading Technology

CPU の高速化技術の一つで，命令の各サイクルを並列で行う技術を Hyper Threading Technology という．本論文では，以降 HTT と記す．HTT は，一つの物理コアで二つのスレッドを実行する．これは一つのスレッドの実行に必要な情報を保持するレジスタセットを二つ用意することで実現している．

一つの物理コアで一つのスレッドを実行する場合は，処理の内容によっては演算ユニットに待機状態が発生する．そこで HTT を利用することで，待機状態の演算ユニットで他方のスレッドの命令を実行することで，処理全体の高速化を実現している．

3.3 Support Vector Machine

Support Vector Machine(SVM)は，2 クラスの分類を行うための教師あり機械学習手法の一つである．現在は多クラスの分類を行うことが可能な手法と，そのライブラリも存在する．

SVM によるクラスの境界面は，学習データの中から他方のクラスに近い位置にあるデータを基準として，そこからの距離が最も大きくなる位置に設定する．

多クラスの SVM には，one-against-all 方式と，pairwise 方式がある．one-against-all 方式では，分類データがそのクラスに属しているか否かの判定を，各クラスに対して行う．学習にかかる時間は長くなるが，分類性能は比較的高いとされている．pairwise 方式では，複数あるクラスの内 2 つを選び，どちらのクラスに近いかをトーナメント方式で判定していく．学習にかかる時間は比較的短いが，one-against-all 方式に比べて精度は下がるとされている．

4 実験

本節では，実験を行った環境，及び実験結果について記す．

4.1 実験環境

本研究における実験は，以下の環境で行った．

- クライアント
  - ・OS:
    Microsoft Windows 7/8
  - ・ブラウザ:
    Google Chrome 36.0.1985.125 m
    Mozilla Firefox nightly 34.0a1
    Mozilla Firefox 31.0
- サーバ
  - ・OS(distribution):
    Ubuntu 14.04 64bit
  - ・Web サーバ:
    Apache HTTP Server2.4.7
  - ・Web アプリケーションサーバ
    PHP5.5.9

4.2　データ収集方法

データを収集するための Web ページを作成し，ローカルネットワーク内でデータの収集を行った．

トップページで端末名，CPU のコア数とスレッド数，CPU 名を入力し，ボタンをクリックすることで測定を開始する．測定は，図 3，図 4 に示した JavaScript によって行う．図 3 に示した JavaScript では，Worker の生成と，Worker からメッセージを受け取った際の処理の定義を行う．ここで複数の Worker による処理を行うことで，複数のプロセスにより CPU に負荷をかけることができる．図 4 には各 Worker で行う処理を示しいている．

```
 1  function workerBenchmark(data) {
 2    var i, num = 16, done = 0, target = 1,
 3    startTime, endTime, workers = [], ;
 4    data.worker = '';
 5    for (i = 0; i < num; i++) {
 6      workers[i] = new Worker('js/worker.js');
 7      workers[i].onmessage=function(event) {
 8        var i;
 9        done++;
10        if (done === target) {
11          endTime = performance.now();
12          data.worker +=
13          (endTime -startTime) + ',';
14          done = 0;
15          target++;
16          if (target <= 16) {
17            startTime = performance.now();
18            for (i = 0; i < target; i++) {
19              workers[i].postMessage('');
20            }
21          } else {
22            data.worker=data.worker.slice(0,-1);
23          }
24        }
25      };
26    }
27    startTime = performance.now();
28    workers[0].postMessage('');
29  }
```

図 3. Worker の生成と呼出，処理時間の測定

```
 1  onmessage = function(event){
 2    n=0, imax = 30000000;
 3    for(var i=0;i<imax;i++){
 4      var x=Math.random();
 5      var y=Math.random();
 6      if(Math.pow(x,2)+Math.pow(y,2)<1){
 7        n++;
 8      }
 9    }
10    var pi=n/imax*4.0;
11    postMessage(pi);
12  };
```

図 4．モンテカルロ法による円周率の計算（worker.js）

図 3 に示したコードでは，変数 num で指定される数の Worker を生成し，それぞれに onmessage イベントハンドラを登録する．onmessage イベントハンドラでは，Worker の起動から，指定された個数の Worker の処理が終了するまでにかかる時間の測定を行う．これを 1 個の Worker から 16 個の Worker に増やしてそれぞれ実行する．図 4 に示したコードでは，HTT の有効/無効によって処理時間に差が生じる処理の一例として，モンテカルロ法を用いて円周率の計算を行う．その結果を Web アプリケーションサーバへ送信し，そのデータを基に，SVM を用いて CPU コア数と HTT の有効/無効の推定を行う．

4.3　実験結果の例

本論文での方式では，各収集データにおいて，target=1 で処理を行った際にかかった時間を 1 とした場合の各 target における処理時間の比率を求める．図 5 に Windows 版 Google Chrome を用いて採取したデータのグラフを例示する．ここでは，2 コア 4 スレッド，4 コア 4 スレッド，4 コア 8 スレッドの CPU を用いてデータを収集した．

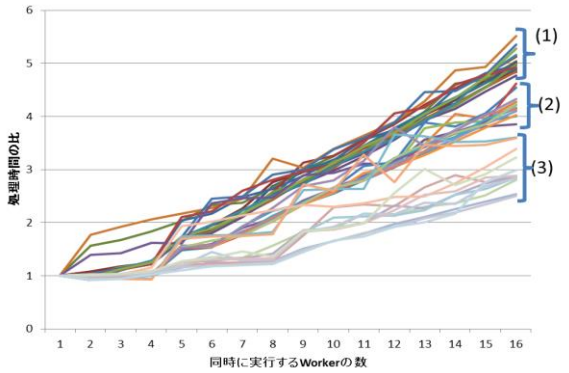

図 5. 各 Worker の処理にかかった時間の比率

縦軸を時間の比率，横軸をtargetの値としている．target の値が大きくなると，グラフが 3 つのグループに分かれている．グラフが最上段のグループ(1)は 2 コア 4 スレッド，中段(2)は 4 コア 4 スレッド，下段(3)は 4 コア 8 スレッドの CPU である．CPU の物理コア数及び HTT の有効/無効によってグラフに差がでていることがわかる．[1]ではHTTの有効/無効を分類することができなかったが，提案手法ではそれを分類することができている．

5 考察

5.1 演算処理性能の差

CPU のコア数と HTT の有効/無効によって処理結果に差が生じた理由として，CPU の命令実行ユニット内部の整数演算ユニットと小数演算ユニットが，独立して動作することが考えられる．整数演算のみを HTT を用いて行う場合，同じ物理コアに割り当てられたスレッド間で共有する整数演算ユニットが実行中の処理を完了するまで次の処理に移ることができず，1 スレッドずつの処理しか実行出来ない．しかし，整数演算と小数演算を同時に行う場合，整数演算を行いながら，小数演算を行うことができるので，1 つの物理コアで 2 スレッドに近いパフォーマンスを発揮することが可能となる．そのため，物理コア数，HTT の有無によって処理時間に差が生じると推測する．

5.2 Worker におけるループ回数と識別能力の関係

図 4 に示したコードにおいて，ループ回数を変化させた場合の識別能力の差を検証した．ここでの識別能力は，SVM によって分類可能であるかどうかによって定める．SVM の実装として LIBSVM[12]を用いた．LIBSVM による pairwise 法による多クラス分類を，以下の手順で行った．

1. N 個のサンプルで構成される集合から，1 つのサンプル N1 を取り出す．
2. N1 以外のサンプルを学習データとして，教師データを作成する．その際，放射基底関数を用いた境界面を設定するものとし，ペナルティパラメータ C を 1000 とする．
3. 作成した教師データを用いて，N1 が正しく分類できるか検証する．
4. 全サンプルに対して，1～3 を繰り返す．
5. 正しく分類できたものの割合を算出する．

各 Worker によって処理されるループの回数が 50000000 回，30000000 回，10000000 回，7000000 回，5000000 回，1000000 回の場合についてデータを収集し，上記の方法で正当率を算出した．その結果を以下に示す．

表 1. Worker のループ回数ごとの分類正答率

ループ回数(回)	正答率(%)	所要時間(ms)
50,000,000	100%	100,090
30,000,000	100%	61,568
10,000,000	98.0%	22,613
7,000,000	95.6%	14,569
5,000,000	93.3%	10,516
1,000,000	86.7%	2,501

表 1 に示した所要時間は，2 コア 4 スレッドの CPU で実験を行った際の平均の処理時間である．この結果より，ループ回数を多くした場合，ループ回数が少ない場合に比べて識別能力が高くなることがわかる．これは，ループ回数が少ない場合，CPU に負荷がかかりきらず，処理性能に差が生じにくくなることが原因として考えられる．しかし，ループ回数が多い場合，推定の処理にかかる時間が長くなるというデメリッ

トもある.

図 5 から，同時に実行する Worker の数が 1 個，2 個，16 個の場合のみの処理時間から，CPU コア数の分類が可能であると推察できる．実際，図 6 のとおり，1 個，2 個，16 個の場合のみの処理は明らかに分かれる．

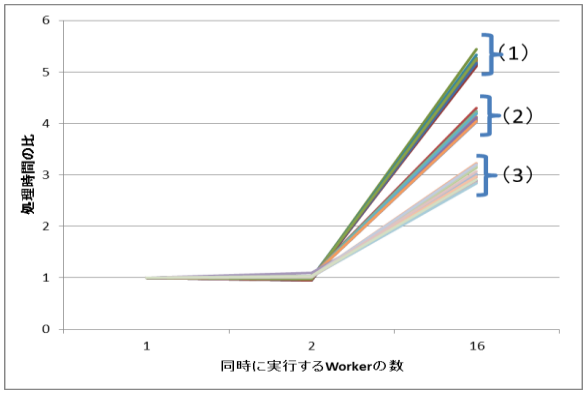

図 6. テスト回数を減らした分類

グラフが最上段のグループ(1)は 2 コア 4 スレッド，中段(2)は 4 コア 4 スレッド，下段(3)は 4 コア 8 スレッドの CPU である．

よって，一つの Worker で処理を行った場合を 1 とし，16 個の Worker で処理を行った場合の処理時間の比率を用いることで，処理時間を短縮しながら，CPU コア数及び HTT の有効/無効を判別することが可能であることがわかる．

5.3 対策技術への適用

提案方式による推定を，FireGloves をインストールした Web ブラウザを用いて行った．FireGloves をインストールした Web ブラウザでは，FireGloves をインストールしていない場合と比較して，識別に影響が出るほどの差は生じなかった．Tor Browser Bundle を用いる場合は，処理が大幅に遅くなる．その影響によって，HTT の有効/無効による差が小さくなる場合があり，分類の正答率が低下することがある．図 7 に Tor Browser Bundle を用いた場合の処理時間の比のグラフを示す．

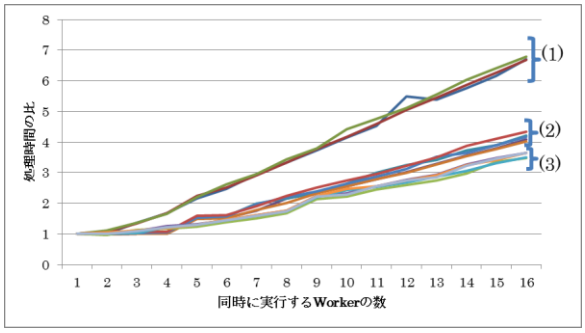

図 7. Tor Browser Bundle を用いた際の処理時間の比

図 7 に示したグラフにおいても，図 6 に示したグラフと同様に，上段(1)に 2 コア 4 スレッド，中段(2)に 4 コア 4 スレッド，下段(3)に 4 コア 8 スレッドと，処理結果に差が生じているが，4 コア 4 スレッドのグループと 4 コア 8 スレッドのグループの差が小さく，SVM を用いた際の分類正答率は低くなる．

5.4 提案方式への対策

Web ブラウザで行うことができる対策として，同時に実行可能な Worker の数の制限や，処理の意図的な遅延が挙げられる．

図 5 より，同時に実行する Worker の数が少ないとき，HTT の有効/無効による差が小さくなり，分類は難しくなることが言える．

ただし，たとえば，Mozilla Firefox は，同一オリジンで同時に実行できる Worker 数を 20 までと制限している[16]が，提案方式に対しては，この制限では意味を成さない．Worker 数を 10 個までに制限した場合は，提案方式での分類の正答率は 83%まで下がるので，もし，提案方式への対策をしたい場合，Worker の数は 10 より少なく制限する必要がある．

また，処理を意図的に遅らせ，CPU の処理性による差を採取させないことによって，提案方式による判別を回避することができると考えられる．

5.5 類似手法との比較

2014 年 8 月時点で，CPU のコア数推定を行

うデモページが公開されている[14]．Google Chrome は，Native Client をサポートしているので，ネイティブコードによってCPUコア数を取得することができる．他の Web ブラウザでは，本論文の提案方式と同様に，処理時間の差を用いて CPU コア数を取得する．

GitHub の 2013 年 5 月頃に登場した類似手法[14]では，処理は高速であるが，物理コアと論理コアの区別ができていないときがある．例えば，2 コア 4 スレッドの CPU で実行した際，2 コアであるという結果を出力する場合と，4 コアであるという結果を出力する場合があり，正確な識別が行えていない場合がある．

我々が 2013 年 1 月に発表した手法および今回の改良バージョンでは，HTT による差が大きくなる処理を実行しているので，処理は比較的低速だが，識別能力では勝っていると言える．

6 まとめ

Web Workers API を用いてマルチスレッドで処理を行うことで，利用者の端末に搭載されている CPU コア数の推定ができることを示した．整数演算と小数演算を同時に行う処理を行うことで，既存の CPU コア数を推定する手法[1]を改善し，さらに HTT の有効/無効の推定もできることを示した．クラスの分類には，SVM の実装の一つである LIBSVM を用いて，効率化の検討も行った．

また，対策ツールを使用した場合の検証を行った結果，FireGloves を使用している場合は，本手法による判別は可能であるが，Tor Browser Bundle を使用した場合は，識別能力が低くなることがわかった．

参考文献

[1] 後藤浩行，齋藤孝道，Web行動追跡のためのハードウェア特徴点の抽出，2013暗号と情報セキュリティシンポジウム概要集 p72-77

[2] 塚本耕司，後藤浩行，齋藤孝道，JavaScriptベンチマークを用いたCPU推定手法の提案と実装，第75回情報処理学会全国大会公演論文集 3-611，3-612

[3] 桐生直輝，後藤浩行，齋藤孝道，CPU拡張命令の対応の有無によるCPUアーキテクチャの推測，第75回情報処理学会全国大会公演論文集 3-613，3-614

[4] P. Eckersley, How Unique is Your Web Browser? In Proc. Privacy Enhancing Technologies Symposium (2010), LNCS vol. 6205

[5] K. Mowery and H. Shacham. "Pixel Perfect: Fingerprinting Canvas in HTML5." Web 2.0 Security and Privacy (W2SP). May 2012

[6] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham, Fingerprinting Information in JavaScript Implementations, In Web 2.0 Security and Privacy, 2011

[7] M. Mulazzani, P. Reschl and M. Huber, M. Leithner, S. Schrittwieser and E. Weippl, Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting, in Web 2.0 Workshop on Security and Privacy (W2SP), 2013

[8] https://www.ghostery.com/ja/

[9] http://fingerprint.pet-portal.eu/files/firegloves-1.2.3.xpi

[10] https://www.torproject.org/projects/torbrowser.html.en

[11] http://www.w3.org/TR/workers/

[12] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[13] http://www.w3.org/TR/tracking-compliance/

[14] https://github.com/oftn/core-estimator

[15] https://developer.chrome.com/native-client

[16] http://wiki.whatwg.org/wiki/Navigator_HW_Concurrency