Vol.2013-MBL-69 No.15
2013/12/20

移動端末からクラウドサービスを利用するためのプロキシサーバ構築に関する研究

鈴木俊裕[†1]　張勇兵[†1]　計宇生[†2]

近年、スマートフォンやタブレット端末などのような移動端末が急速に普及し、クラウドサービスを利用する機会が増えてきているが、移動端末の処理能力や通信帯域幅、電力容量などの制限が考慮されていないため、効率的な利用が実現できていない。本研究では、上記移動端末の制限に関する問題を考慮し、移動端末に近いネットワーク上にプロキシサーバを設けることにより、移動端末がクラウドサービスを効率的に利用できるシステムを提案した。それにより、移動端末から要求されたデータをそのアクセス特性にしたがって近隣にあるサーバにキャッシュし、また、移動端末の処理負荷をサーバ側に効率よく分散することにより、応答時間の短縮および移動端末の消費電力を減らすことができる。提案システムは従来のウェブプロキシサーバ squid を拡張し、また、JSON-RPC を使って移動端末のアプリケーション実行（負荷）をサーバにオフロードを実現した。提案手法の有効性を評価するため、実システムより得られたアクセス履歴データを使用した。

Proxy Server for Mobile Users to Use Cloud Services

TOSHIHIRO SUZUKI[†1]　YONGBING ZHANG[†2]　YUSHENG JI[†3]

The number of mobile devices such as smartphones and tablet terminals has increased significantly and accesing the cloud services using those mobile devices also become common. However, due to the limitations of the mobile devices such as low processing capability, small storage capacity, narrow network bandwidth, and limited battery capacity, the access to the cloud services by the mobile devices is not efficiently realized. In this paper, we propose to allocate proxy servers on the network to cache the users' accessed data and offload the users' computation loads to the proxy servers in order to resolve the previously mentioned problems. We implemented the proposed mechanism by extending the well-known proxy server, Squid, and realized the offload mechanism using JSON-RPC. We evaluated the proposed mechanism by simulation using the trace data obtained from a real web proxy server system.

1. はじめに

近年、スマートフォンやタブレット端末などのような多機能移動情報端末が急速に普及してきている。総務省平成24年版情報通信白書[1]によれば2011年には世界市場におけるスマートフォンの販売台数は約4億7千万台であったが、2016年には約13億台になり、年平均22%の成長になると予測されている。また、移動端末はデスクトップ型コンピュータと同様にウェブ閲覧やファイル転送などのネットワークアプリケーションを利用できるだけでなく、クラウドサービスを利用することも可能である。その一方、移動端末はバッテリーによる駆動するものがほとんどであり、また、その処理能力や記憶容量、通信帯域幅などの性能は一般的にはデスクトップ型コンピュータより劣っている。従って、クラウドサービスを効率的に利用するには、デスクトップ型コンピュータと異なる手法を開発する必要がある。

上記移動端末の弱点を補い、クラウドサービスを利用する際には、ネットワーク上にプロキシサーバ（キャッシュサーバとも呼ばれる）を配置することにより必要とされるデータの処理や計算処理などをプロキシサーバで行い、必要なデータをプロキシサーバに一時保存することが考えられる[2]。プロキシサーバにはフォーワーダプロキシ型とリバースプロキシ型があるが[3]、本研究では移動端末がクラウドサービスを利用する際、リバースプロキシ型のキャッシュサーバを使用する。プロキシサーバは、ユーザからのクラウドサービス要求を中継し、取得したデータのキャッシュをする。それにより、移動端末が必要の計算や処理を行うための最小限のデータだけをプロキシサーバから取得し、その他のデータをサーバにキャッシュすることができる。そのため、データアクセス時間を短くし、通信トラヒックを削減することが期待できる。

また、移動端末での処理負荷を軽減し、バッテリーの消費量を削減する方法として、移動端末の処理負荷をサーバにオフロードすることが考えられる[4]。ここで、オフロードとは移動端末が処理すべき計算などのアプリケーション負荷をプロキシサーバに移送して処理することを言う。サーバにオフロードするかどうかは処理の計算量および伝送

†1 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†2 国立情報学研究所
National Institute of Informatics

すべきアプリケーションのサイズにより決まる。その際、計算量が大きく、伝送トラヒックが少ない処理はオフロードした方がよい一方、計算量が少なく、伝送トラヒックが多い処理は移動端末で処理した方がよい。その移送の決定[4]は処理をサーバへ送るかどうかを決定するフェーズと、サーバへ送る処理を分割するフェーズに分けられる。

そこで、本研究では、移動端末に近いネットワーク上にプロキシサーバを設置することにより、移動端末が効率的にクラウドサービスを利用できる手法を提案する。具体的には、従来のウェブプロキシサーバ[5]を拡張し、ユーザの属性(例えば、端末言語やデータの地域性など)により、そのデータのキャッシュ優先順位を考慮に入れるキャッシュ手法を提案し、優先順位の高いユーザデータを先にキャッシュするようにする。また、移動端末での処理時間およびアプリケーションファイルのサイズに応じてプロキシサーバへオフロードする仕組みを実現する。本研究で提案する手法を評価するため、実システムから得られたウェブプロキシサーバのアクセス履歴データを使って、実装実験を行い、先行研究手法と比較した。

本論文の構成は以下のようになっている。第 2 章では、プロキシサーバソフトウェア Squid で実装されているキャッシュ置換ポリシーや遠隔プログラムを実行する仕組み JSON-RPC (JavaScript Object Notation - Remote Procedure Call)について紹介する。第 3 章では、本研究で提案するデータキャッシュ方式およびオフロード方式の仕組みについて説明する。第 4 章では、提案システムの実装および評価実験について述べる。最後に第 5 章で結論を述べる。

2. 関連研究

2.1 プロキシサーバ

ウェブプロキシサーバはユーザのアクセスしたデータをキャッシュし、そのキャッシュ記憶容量を超えてしまった場合、合計アクセス数の少ないデータからキャッシュから削除して行くようになっている。これまでウェブアプリケーションを利用する際、ユーザのアクセスしたデータをキャッシュする手法[5][6][7]が多く提案されており、実システムで広く使われているものが Squid[5][6]と呼ばれるプロキシサーバがある。Squid[5][7]では、キャッシュ容量を超えた場合、キャッシュにあるデータを置き換える方式として、LRU (Least Recently Used)および GDSF (Greedy-Dual Size Frequency)、LFUDA (Least Frequently Used with Dynamic Aging)の３つが提案されており、その中で GDSF 方式のデータ項目ヒット率が最もよい。

LRU 方式は最も簡単なキャッシュ手法であり、キャッシュサーバ内で最も長い時間参照されないデータ項目から削除していくものである。一方、GDSF および LFUDA[6]では、各データ項目のアクセス回数およびデータ項目の伝送コストを考慮するキャッシュを決定するための評価値を定義しており、その評価値はデータアクセス回数およびデータの伝送コストに正比例するように設定している。GDSF 方式では、さらにデータ項目の大きさを考慮に入れ、キャッシュ評価値はデータサイズに反比例するようにしている。GDSF 方式はオブジェクトヒット率（データ項目のヒット回数とキャッシュ可能なオブジェクト総数との比）を最大化することを目的としているが、LFUDA 方式は、バイトヒット率（キャッシュヒットしたデータ項目のバイト数とキャッシュ可能のバイト数容量との比）を最大化することを目的としている。

しかし、これらのデータキャッシュ方式では、異なるユーザのデータ項目の区別が考慮されていない。クラウドサービスを利用する際、ある特定の地域に属するユーザは同じ、または、関連のあるデータを要求することが多いと考えられ、他の地域から来る訪問者のアクセスするデータより高い優先度でキャッシュすることが望ましい。

2.2 アプリケーションの遠隔実行

遠隔にあるコンピュータ上で処理を行う仕組みとして、そのコンピュータ上にある関数・プロシージャを呼び出す RPC(Remote Procedure Call)[8]の仕組みがある。RPC 機構では、呼び出される関数・プロシージャはサーバ上に存在し、呼び出される関数名を指定するとともにそれへの引数を引き渡すことによりその関数を遠隔コンピュータで実行することになる。RPC の仕組みを拡張し、HTTP を通じて、関数呼び出しを実現するための XML-RPC(Extensible Markup Language - Remote Procedure Call)プロトコル[9]がある。XML-RPC では、関数の呼び出しは XML にエンコードされ、サーバ側に送られる。また、XML 機構を異なる言語や OS 環境などに拡張し、新たな機能を追加したものに SOAP(Simple Object Access Protocol)[10]がある。

また、最近に HTTP 以外のプロトコルにも対応でき、XML に比べ軽量であるプロトコルとして、JSON-RPC プロトコル[11]が提案されている。JSON は、その多様性および軽量性により、無線通信システムのような通信帯域幅が限られている環境に適している[12]。JSON プロトコルでは、クライアントは JSON-RPC ブリッジを通じて JSON 書式にエンコードされたデータをサーバに伝送し、そして、サーバは元のオブジェクトにデコードしてから呼び出す関数に渡して処理するようになっている。

移動端末が処理能力や記憶容量などが限られているため、一度に処理できるデータ量が少なく、処理が長時間かかることがあるため、サーバ側にオフロードすることが考えられる。JSON-RPC を利用することにより、移動端末の種類を問わず、端末のアプリケーションをネットワーク上にあるサーバに簡単に分散することができる。

3. 提案システム

本研究で提案するシステムは図 1 に示されるような構成になっている。ユーザがクラウドサービスを利用する際、

ポータルサーバを経由するようになっており、ユーザからのリクエスト要求はポータルサーバによりその現在位置に最も近いプロキシサーバにリダイレクトされるようになっている。プロキシサーバはユーザの属性に従ってそのアクセスデータのキャッシュ優先順位を決定し、また、移動端末での処理負荷やサーバへの伝送時間などを考慮して、サーバへのオフロードを決定する。

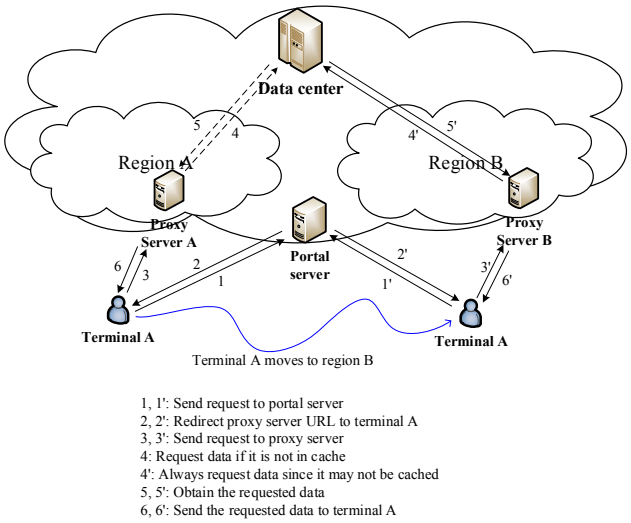

図 1　提案システム

Figure 1　Proposed system.

3.1 データキャッシュ方式

本研究で提案するキャッシュ方式は GDSF 方式[7]を拡張したものである。GDSF 方式では、キャッシュ決定はデータ項目の合計アクセス数を考慮しているため、遠い過去に頻繁にアクセスのあったデータ項目がキャッシュに残される可能性が生じてしまう。そこで、本研究では、一定時間間隔における平均アクセス回数が用いられ、最近にあったアクセス回数は優先順位が高くするようにしている。また、GDSF 方式のキャッシュ決定では、ユーザの属性やその現在位置などの要素が考慮されていない。例えば、同じ地域に属するユーザは同じ、または、似ているデータ項目を使用することが多いと考えられるため、その地域で多くのユーザが必要とされるデータ項目を優先的にキャッシュする一方、他の地域からの訪問者がアクセスするデータ項目の優先順位を低くすることが望ましい。提案システムでは、ユーザの属性に従って、キャッシュする優先度を設定できるようにしている。

本研究では、データ項目 i をキャッシュすべきかを評価する値（これから、単に評価値と呼ぶ）は K_i で表される。また、与えられた一定時間間隔 T にデータ項目 i をアクセスした平均回数は A_i で表され、現在の時間間隔において観測したアクセス回数は F_i で表される。データ項目 i のデータサイズと伝送コストはそれぞれ S_i と C_i で表される。よって、時間間隔 T におけるデータ項目 i の平均アクセス回数は次のように計算される。

$$A_i = \alpha F_i + (1-\alpha)A_i^p.$$

ここで、A_i^p は現在の時間間隔までの平均アクセス回数を表し、また、$\alpha(0 < \alpha \leq 1)$ は平均値をとるためのパラメータである。データ項目の平均アクセス回数の計算は、遠い過去のアクセス回数と最近のアクセス回数による重み付けを使って計算されており、パラメータ α の値によりそれらの重みが決まる。パラメータ α の値が大きい場合は、最近のアクセス回数を重視するが、α の値が小さい場合は、過去のアクセス回数を重視することになる。

データ項目 i の評価値 K_i は次式で与えられ、その評価が高いほどキャッシュされる可能性は高くなる。

$$K_i = \frac{A_i P_i C_i}{S_i}.$$

ここで、P_i はプロキシサーバがデータ項目 i をキャッシュする属性（重要度）$(0 < P_i \leq 1)$、C_i はデータ項目をキャッシュするためのコスト、また、S_i はデータ項目の大きさを表す。上の式から、データ項目は、平均アクセス回数が多く、伝送コストが高く、また、キャッシュ重要度が高いほど、キャッシュされる可能性は高くなることが分かる。一方、データ項目のサイズが大きく、または、キャッシュ重要度が小さい場合は、キャッシュされる可能性は小さくなる。データ項目のキャッシュ重要度は、前述の通りそのデータがプロキシサーバの設置地域との関係により決まる。例えば、図 1 に示されるように、ネットワークにプロキシサーバ A, B があるとする。サーバ A の領域に属するユーザ A はプロキシサーバ A の近くにいる場合、アクセスしたデータはサーバ A にキャッシュする可能性は大きく、逆にユーザ A がサーバ B の領域に入り、ユーザ A がアクセスするデータ項目はプロキシサーバ B にキャッシュする可能性が低くなる。

3.2 オフロード方式

移動端末で実行するアプリケーションのオフロードは、実行時間の短縮や移動端末消費電力の削減など異なる性能目標としている[6]。本研究では、オフロード方式は、移動端末での処理負荷をプロキシサーバに分散することによりアプリケーションの実行時間を短くし、また、移動端末の消費電力を削減することを目的としている。具体的には、移動端末でのアプリケーション推測実行時間がプロキシサーバで処理する時間より長い場合は、そのアプリケーションをサーバに伝送（オフロード）して、サーバで実行することにする。サーバで実行する場合の所要時間は、サーバでそのアプリケーションを実行する時間と、そのアプリケーションがサーバまで伝送して、そして、その処理結果が移動端末までに返送するまでの往復時間との和で計算される。

アプリケーションが移動端末で処理される時間とは端末で実行しはじめてから応答があるまでの時間であり、T_l で

表される。同様に、サーバでの処理時間はT_sで表される。また、アプリケーションが端末からサーバに伝送し始めてから応答が返ってくるまでの所要時間はアプリーションを伝送する時間とサーバで処理する時間との和であり、T_rで表される。ここで、アプリケーションの伝送時間を計測するため、定期的に単位長パケットをサーバに送信し、端末とサーバとの往復伝送時間Rを測定するようにする。以上の計測結果により、アプリケーションがサーバで処理される時間は次のように計算される。

$$T_s = T_r - DR.$$

ここで、パラメータDはデータ長を表す。

プロキシサーバの処理能力が高いため、アプリケーションの処理時間はそれほど変動ないことが考えられるため、単位パケットにより計測した往復伝送時間Rを使用すれば、$T_r(T_r = T_s + DR)$を推測することができる。したがって、もし$T_l > T_r$ならば、アプリケーションはプロキシサーバにオフロードすることになる。

4. 提案システムの実装

本研究では、移動端末は Galaxy Nexus で、OS は Android 4.0.2 をしているものを使用した。また、プロキシサーバは CPU が Core 2 Duo 2.53GHz、メモリが 4GB で、OS が CentOS 6.4 を使用し、その上にキャッシュサーバソフトウェア Squid を拡張したものを実装した。移動端末とプロキシサーバとの接続は IEEE 802.11g 無線 LAN を使用した。図 1 に示されるように、ポータルサーバは移動端末からのデータ要求（http リクエスト）を受け付け、そして、そのリクエストからその端末 IP アドレス情報を取得し、その端末に近いプロキシサーバの情報を移動端末にリダイレクトする。移動端末はプロキシサーバの情報を取得してから自動的にプロキシサーバにリクエストを送信する。

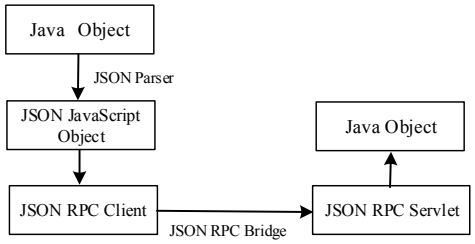

図 2　JSON-RPC による Java コードの遠隔実行
Figure 2　JSON-RPC based Java code remote execution.

本研究では、移動端末のアプリケーションを遠隔実行するため、JSON ライブラリ JSON-RPC v2.0 をサポートする jabsorb 1.3.2 を使用した。また、実験用移動端末が Android OS に基づくものをであるため、端末とサーバでのプログラミング環境は Java ベースのものに限定したが、他のシステム環境においても同様な仕組みで実現できると思われる。

以下の例は、JSON-RPC を使って、クライアントがサーバ上にあるメソッド `subtract` を実行するものを示している。メソッド `subtract` とは 2 つの引数を持ち、その引数による引き算を行うメソッドである。この場合、クライアントは次のようなメッセージをサーバに送ることになっている。

```
--> {"jsonrpc": "2.0", "method": "subtract",
"params": [42, 23], "id": 1}
```

ここで、`jsonrpc` は JSON-RPC のバージョンを示し、`method` はサーバで実行するメソッド名を示す。`params` は実行するメソッドの引数を表し、`id` は呼出メソッドの識別子を示す。一方、サーバ側はメソッドの実行終了後に以下のようなメッセージをクライアントに送り返すことになっている。

```
<-- {"jsonrpc": "2.0", "result": 19, "id": 1}
```

サーバが送り返すメッセージはクライアントからのメッセージに似ており、`jsonrpc` は JSON-RPC のバージョンを表し、`result` はメソッドの戻り値を示し、`id` はクライアントからもらったクライアントの呼出識別子である。以上のようなやり取りにより、クライアントはサーバ上にある引き算メソッド `subtract` を呼び出すことができる。

5. 提案システムの評価

本研究では、IRCache プロジェクト[13]によるウェブプロキシサーバのキャッシュ履歴データ（1995 年 8 月 28 日から一週間）を使って、シミュレーション実験を行った。ここで、データ項目はランダムに 2 つの言語（言語 A と B）に分け、そのデータ項目数はそれぞれ 9 割と 1 割とした。履歴データにおいては、データ項目の総数は計 31,033 個あり、ランダムに分けられた言語 A と B はそれぞれ 27,925 と 3,108 個である。また、データ項目数のデータ量は約 317MB あり、言語 A と B のデータ量はそれぞれ 284MB と 32MB である。データ項目へのアクセス総数は 1,488,673 であるが、シミュレーション実験はキャッシュが空の状態から行うため、最初の 10 万アクセスは初期状態として結果の統計に入れていない。本実験では、先行研究と比較するため、GDSF 方式を使用した。プロキシサーバのキャッシュ容量Cは全データ量の 10%, 20%, 40%とした。また、評価指標値としてデータ項目のヒット率（データ項目がキャッシュで見付かるときのリクエスト数とすべてのリクエスト数との比）を使用した。

実験結果は図 3-図 12 に示されている。ここで、言語 A の重要度P_Aは 1 であるとし、言語 B の重要度P_Bはそれぞれ 0.0, 0.5、1.0 とした。$P_B = 0$の場合、キャッシュサーバに空いているならば、言語 B のデータはキャッシュされるが、キャッシュに空きスペースが足りないなら、キャッシュされている言語 B のデータが削除されてしまうことになる。一方、$P_B = 1$の場合、言語 A および B によるデータの区別がなく、同様に扱われることになる。データの平均

アクセス回数を計算するために使われるパラメータ α 値の影響を調べるため、実験では α=0.5, 0.8, 1.0 にした。

実験結果から、システムパラメータを適切に調整すれば提案方式は GDSF 方式より良い性能が得られることが分かったが、改善の度合いは残念ながら顕著なものではなかった。例えば、キャッシュ容量が全データ量の 10%、α=0.8、時間間隔T=16 時間のとき、言語 A、B が同等に扱われる、すなわち、$P_A = P_B = 1$なら、データ項目ヒット率は GDSF 方式より 2%程度良く、また、言語 B の重要度P_Bが 0.5 のとき、言語 B のデータヒット率がさほど悪くならない一方、言語 A のデータヒット率はより良くなることが分かった。

図 3　言語 A データのヒット率（C=5%, α=0.5）
Figure 3　Hit rate of language A (C=5%, α=0.5).

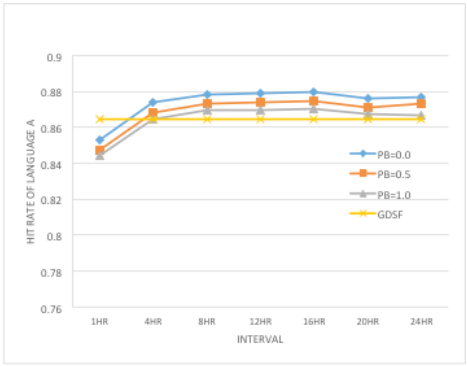

図 4　言語 A データのヒット率（C=5%, α=0.8）
Figure 4　Hit rate of language A (C=5%, α=0.8).

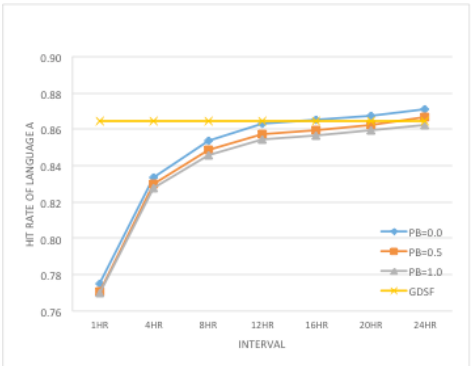

図 5　言語 A データのヒット率（C=5%, α=1）
Figure 5　Hit rate of language A (C=5%, α=1).

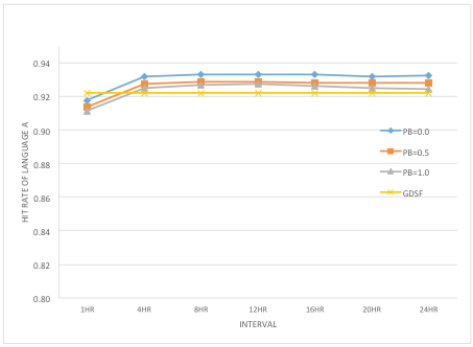

図 6　言語 A データのヒット率（C=10%, α=0.5）
Figure 6　Hit rate of language A (C=10%, α=0.5).

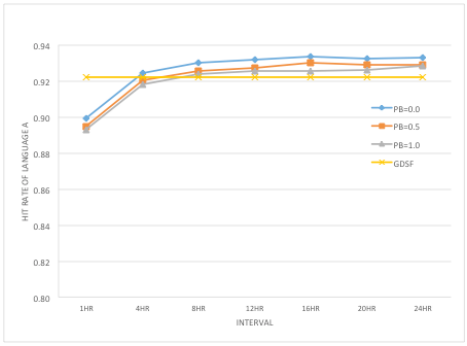

図 7　言語 A データのヒット率（C=10%, α=0.8）
Figure 7　Hit rate of language A (C=10%, α=0.8).

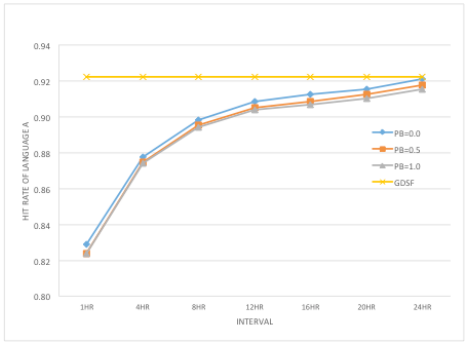

図 8　言語 A データのヒット率（C=10%, α=1）
Figure 8　Hit rate of language A (C=10%, α=1).

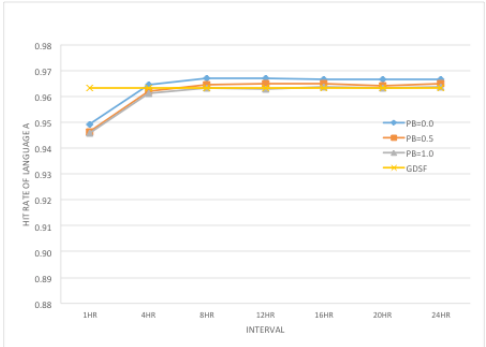

図 9　言語 A データのヒット率（C=20%, α=0.5）
Figure 9　Hit rate of language A (C=20%, α=0.5).

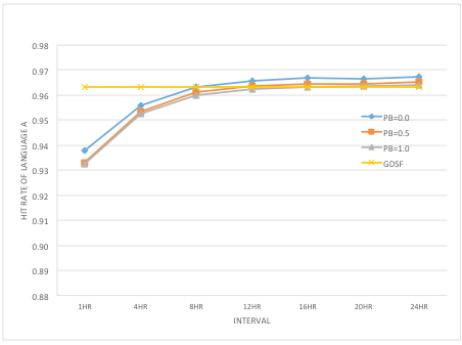

図 10　言語 A データのヒット率（C=20%, α=0.8）
Figure 10　Hit rate of language A (C=20%, α=0.8).

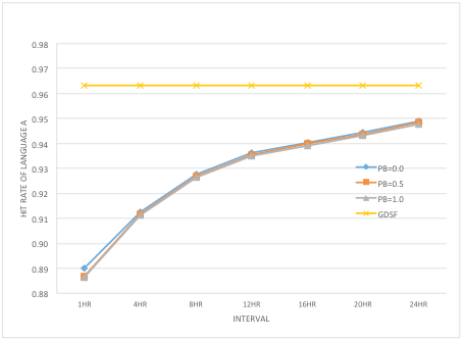

図 11　言語 A データのヒット率（C=20%, α=1）
Figure 11　Hit rate of language A (C=20%, α=1).

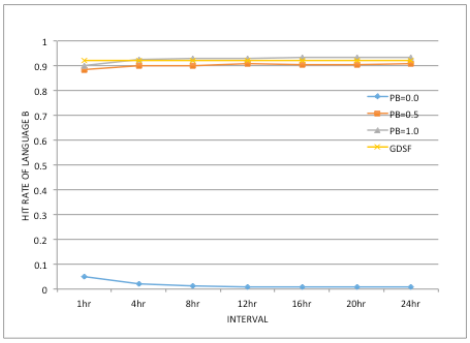

図 12　言語 B データのヒット率（C=10%, α=0.8）
Figure 12　Hit rate of language B (C=10%, α=0.8).

3.2 節で述べたように、アプリケーションの処理量が多く、しかも、そのサイズが小さい場合、アプリケーションをサーバで処理させることが望ましい。一方、アプリケーションのサイズがかなり大きい場合、移動端末で処理するのがよい。本研究では、アプリケーションのオフロード（分散）方式を検証するため、2,000 桁までの円周率を求める例を使って実証実験を行った。その結果によると、移動端末で処理する場合、約 17.8s かかるが、プロキシサーバに伝送して処理する場合、約 5.7s までに減らすことができることが分かった。

6. まとめ

本研究では、移動端末が効率的にクラウドサービスを利用するため、移動端末に近いネットワーク上にプロキシサーバを設けることにより、移動端末の必要なデータをキャッシュし、また、ユーザの属性により異なる優先順位のデータキャッシュ手法を提案した。そして、ウェブサーバへのアクセス履歴データを使って提案方式を評価したが、先行方式よりわずかに良くなることが分かった。今後は、移動端末がクラウドサービスを利用する際に必要とされるデータ特性を分析し、より実システムに近いデータのアクセスパターンを使って提案方式を評価する予定である。また、移動端末の処理時間を減らし、電力の消費を抑えるため、JSON-RPC を利用して移動端末の処理負荷をプロキシサーバに分散する仕組みを提案し、その実証実験を行った。それにより、アプリケーションの計算量が大きく、また、そのサイズが小さい場合は、移動端末のアプリケーションをプロキシサーバにオフロードするのがよいことが分かった。今後はより多くのデータの発展として、ある特定の地域にあるプロキシサーバにその地域に属するユーザのアクセスデータをキャッシュするような手法を開発する必要がある。

謝辞

本研究の一部は国立情報学研究所共同研究の助成を受けて行われた。

参考文献

[1] 総務省平成 24 年版情報通信白書: http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h24/html/nc122110.html.

[2] X. Fan, J. Cao, and H. Mao: A Survey of Mobile Cloud Computing, *ZTE Communications*, Vol. 9, No. 1, pp. 4-8 (Mar. 2011).

[3] サーバ構築研究会編著: CentOS 6 で作るネットワークサーバ構築ガイド, 秀和システム (2012).

[4] K. Kumar, *et al.*: A Survey of Computation Offloading for Mobile Systems, *Mobile Networks and Applications*, Springer, Vol. 18, No. 1, pp. 129-140 (Feb. 2013).

[5] プロキシサーバ Squid: http://www.squid-cache.org/.

[6] M. Arlitt, *et al.*: Evaluating Content Management Techniques for Web Proxy Caches, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 27, No. 4, pp. 3-11 (Mar. 2000).

[7] S. Romano and H. ElAarag: A Quantitative Study of Web Cache Replacement Strategies using Simulation, *Simulation*, Vol. 88, No 5, pp. 507-541 (May 2012).

[8] A.S. Tanenbaum and D.J. Wetherall: Computer Networks, 5th Ed., Peason, New York, 2011.

[9] RPC プロトコル xml-rpc: http://xmlrpc.scripting.com/default.html.

[10] SOAP: http://www.w3.org/TR/soap12-part0/.

[11] JSON-RPC: http://json-rpc.org/.

[12] C. Rodrigues, *et al.*: Mobile Application Webservice Performance Analysis: Restful Services with JSON and XML, *Communications in Computer and Information Science,* Vol. 220, pp. 162-169 (Oct 2011).

[13] IRCache プロジェクトによるプロキシサーバトレースデータ: http://www.ircache.net/.