Vol.2009-OS-112 No.5
2009/8/5

クライアント資源を利用した 堅牢なマッシュアップサービスの実現

堀江 光[†1] 浅原理人[†1] 河野健二[†1]

Web サービスの応答遅延や異常停止等の一因として Flash Crowds と呼ばれる突発的なアクセス集中が知られている．Flash Crowds は発生時期や規模の予測が困難なため，正確な予測を前提としない対策が求められる．また，近年の Web サービスでは動的生成コンテンツや外部のサーバから提供されるコンテンツと組み合わせたマッシュアップも増えてきたため，これらを念頭に置いた Flash Crowds 対策が必要である．本論文では，クライアントの資源を用いることで Flash Crowds への耐性を持ち，動的 / 外部コンテンツにも対応したコンテンツの配信手法を提案する．本手法は，クライアント間で形成する Peer-to-Peer ネットワーク上でキャッシュを管理・共有することで，サーバへのアクセスを軽減する．また，クライアントが取得したコンテンツを，そのクエリをキーとして管理し共有することで，静的 / 動的 / 外部コンテンツの区別なく共有可能にする．10,000 台の仮想クライアントを用いた実験ではサーバのリクエスト処理数とネットワーク帯域使用量が約 98 % 減少することを確認した．

Robust Mashup-Service Using Client Resources

Hikaru Horie,[†1] Masato Asahara[†1]
and Kenji Kono [†1]

Web services are often delayed or stopped by flash crowds. In flash crowds, many clients suddenly connect to a web server to get a service. It is difficult to deal with flash crowds due to the difficulty of predicting the time and the scale of flash crowds. On the other hand, a recent web server often uses mashups to provide rich and complex services. Mashups use dynamically generated contents or contents provided from an external server. A web server must deal with flash crowds even if it uses mashups. In this paper we propose a novel technique which enables a web with mashups server to provide its service in flash crowds. Our technique is in a peer-to-peer fashion; it makes a client to share caches of a content between clients. To be adaptive to mashup contents, our technique uses a query which is used for generating a content as a key assigned to a cache of the content. We evaluate our technique with 10000 emulated clients. Experimental results demonstrate that our technique reduces the number of requests received by a web server and the network usage of the server by 98 %.

1. はじめに

近年 Web サービスの利用範囲は拡大し，個人や私企業だけでなく公的機関でも多く用いられるなどその重要度も高まってきている．一方で Web サービスの拡大により，サーバの応答遅延や異常停止といった障害発生が大きな影響を及ぼすようになってきた．サービス障害は多数の利用者に不利益を与えるため，安定的なサービス提供は益々重要性を増している課題である．障害の原因として，サーバに対して不特定多数のアクセスが突発的に集中する Flash Crowds と呼ばれる現象が大きな問題となっている[1)]．Flash Crowds は多くの場合長くは続かず，数分から数十分程度の短時間で沈静化すると言われている．しかし Flash Crowds 発生時のアクセス負荷は平常時の数倍から数百倍になることもあり，サービスが停止することもしばしば起こる．

現在主流の対策はキャッシュサーバの設置によるアクセス負荷の分散である．サーバの運用には多くのコストを要するため必要以上のサーバを設けるのは適当ではなく，また，必要量に満たないサーバではアクセスに対処しきれないことから，サーバ台数を適切に見積もることが重要である．しかし，Flash Crowds は様々な要因から起こる現象であるため，その発生時期や規模の予測は難しく，故に必要なキャッシュサーバの台数を見積もることは非常に難しい．Flash Crowds の原因としては例えば，Slashdot effect と呼ばれる現象が有名である．これは Slashdot[2)] で紹介されたことをきっかけに，ある Web サイトへのアクセスが爆発的に増えてサービス停止を引き起こすものである．このような Flash Crowds の原因は他にも無数に存在するため，それらがいつ起こり，どの程度のアクセス増加を引き起こすかを予測するのは極めて困難である．また，突然の負荷増に備えるためにサーバは平時から稼働しておく必要があるが，ほとんどの場合でこれらは過剰な稼働でありコスト面で大きな無駄が生じるという問題もある．

このように予測が困難な Flash Crowds に対処するために，Peer-to-Peer (P2P) 技術を用いることで高いスケーラビリティを実現する手法が提案されている．この手法はクライアント

†1 慶應義塾大学
Keio University

間で取得済みのデータを共有することでサーバの負荷軽減するもので，例えば BitTorrent[?)] は大容量ファイルの配信に用いられている．一方で，近年多くの Web サービスでは，クライアントのリクエストに応じて動的に生成されるコンテンツ (動的コンテンツ) や，外部のサーバから提供されるコンテンツ (外部コンテンツ) によるマッシュアップが見られる．従来の手法ではマッシュアップを用いた Web サービスへの対応は十分でない．P2P 技術を用いた多くの手法ではあらかじめ，コンテンツを分割したものやそれを復元するためのインデックス情報を用意しておく必要があるため，動的コンテンツを配信するには別途サーバ側で対応する必要がある．また，外部コンテンツは各クライアントが直接外部のサーバから取得するため，管理対象のサーバから外部コンテンツを共有可能な形で提供するように改変することは難しい．

本論文では，マッシュアップサービスを用いたサーバに対して，各クライアントが取得したコンテンツをキャッシュとして管理・共有する手法を提案する．本手法は，本来サーバへ発行されるリクエストをクライアント間でも処理することで，サーバの処理するリクエスト数を減少する．さらに，クライアントがサーバへ発行したクエリとそれにより取得したコンテンツを一対一に対応づけることで，動的／外部コンテンツについても静的コンテンツと同様にクライアント間で管理し共有することができる．

P2P 技術による従来のコンテンツ配信手法は，キャッシュを共有するための情報の生成・管理をサーバが行っていた，これに対して，本手法ではサーバからコンテンツを取得したクライアントが共有するための情報を生成し，キャッシュの管理は P2P ネットワーク上で行う．これにより，サーバに負荷を集中させずにキャッシュを管理し共有することができる．また，キャッシュを管理する P2P ネットワークの構築には分散ハッシュ表 (DHT) を用いることで，多くのクライアントが存在する状況下でもキャッシュを高速に検索し取得することができる．

提案手法の有用性を検証するために，ネットワークシミュレータ上に実装し，10,000 台の仮想クライアントにより，サーバに毎秒約 200 件のリクエストを発行する実験を行った．全てのクライアントがサーバから直接コンテンツを取得した場合と，提案手法を利用した場合とで，サーバの負荷を比較した．その結果，サーバの 1 秒間当たりのリクエスト処理数は約 98 % 減少し，ネットワーク帯域使用量も約 98 % 減少することを確認した．

本論文の構成は以下の通りである．2 章では関連研究について述べる．3 章では本論文の提案機構について説明する．4 章では提案機構の実装について説明する．5 章では提案機構の有用性を確認した実験について述べる．最後に 6 章で本論文をまとめる．

2. 関 連 研 究

2.1 キャッシュサーバによる負荷分散

キャッシュサーバの設置による負荷分散は現在主流の Flash Crowds 対策である．この手法は，複数台用意したキャッシュサーバにコンテンツのキャッシュを配置し，クライアントに提供する．キャッシュサーバの台数に応じてサーバ 1 台あたりの負荷の減少を見込むことができる．しかし， Flash Crowds に対応するには平常時から十分な台数のキャッシュサーバを稼働しておく必要がある．また，用意したキャッシュサーバ群の処理能力を上回る負荷には対応できない．Flash Crowds の発生時期や規模の予測が性格にできれば無駄を抑えたキャッシュサーバの稼働が可能になる．しかし，そもそも Flash Crowds の予測が難しいため，対応は困難である．また，動的コンテンツへの対応は，クライアントのリクエストに応じてキャッシュを配置する仕組みをとることで対応することが可能である．しかし，外部コンテンツはクライアントが外部のサーバから直接取得してしまうため対応が難しい．例えば，ある Web サービスにおいて API (Application Programming Interface) を用いて Youtube[3)] の動画を埋め込む場合， クライアントは Youtube のサーバへ直接アクセスして動画を取得するため，用意したキャッシュサーバが利用されることはない．

キャッシュサーバを用いたコンテンツ配信システムに Akamai[4)5)] がある．Akamai では，世界中に配置された多数のキャッシュサーバを専用の高速回線で結んでおり，オリジナルのコンテンツを提供するサーバ (オリジンサーバ) へアクセスしてきたクライアントに対して独自の Domain Name System サーバ (DNS サーバ) によって最寄りのキャッシュサーバを割り振る．キャッシュサーバが対象コンテンツのキャッシュを保持していない場合，クライアントに変わってキャッシュサーバがオリジンサーバからコンテンツを取得した上でクライアントに提供する．キャッシュサーバが対象コンテンツのキャッシュを保持している場合，キャッシュサーバはオリジンサーバへアクセスすることなくクライアントにキャッシュを提供する．この仕組みにより，オリジンサーバへのアクセス数は軽減されるため Flash Crowds への耐性が得られる．しかし Flash Crowds への耐性は同社が用意している資源に依存する．以前，Akamai 社の DNS サーバが分散型サービス不能攻撃 (DDoS 攻撃) によって障害を起こし，その DNS サーバを利用した Web サービスが利用不能に陥った事例がある．[6)] DDoS 攻撃と Flash Crowds は別であるが，アクセス集中によって障害を発生する可能性が露呈されたと言える．また，本システムは同社に登録したオリジンサーバにのみ適用されるため，外部コンテンツは対象としていない．

2.2 クライアント資源の利用による負荷分散

クライアント資源を利用する手法では，P2P 技術を用いクライアント間でキャッシュを共有することでサーバの負荷軽減を図る．この手法で，コンテンツは断片 (Chunk) として扱われ，各クライアントは Chunk を収集することで目的のコンテンツを得る．各クライアントが互いに取得済みの Chunk を提供することで，サーバが各クライアントに直接コンテンツを提供する場合よりも，サーバの通信量を軽減できる．コンテンツを取得済みのクライアントが多く存在することは， Chunk を提供するクライアントが多く存在することを意味する．すなわち，Flash Crowds 発生時のように特定のコンテンツに多数のクライアントが集中している場合，コンテンツを要求するクライアントの増加と共に，Chunk を提供するクライアントも増加する．この手法は，こうしてクライアント資源を利用することで高いスケーラビリティを実現するため，Flash Crowds の発生時期や規模の予測を必要とせず，多数のクライアントが集中する場合でも安定的なコンテンツ配信を可能とする．

クライアント資源を利用した負荷分散手法として BitTorrent[7]，Antfarm[8]，Flashback[9] がある．BitTorrent，Antfarm は多数のクライアントに対して大容量コンテンツを配信することを目的としている．各クライアントは，目的のコンテンツを構成する Chunk の情報とサーバのアドレスを含むメタデータファイルを Web などから入手し，サーバにアクセスする．サーバはクライアントに対し，リクエストされたコンテンツの Chunk と，そのコンテンツの Chunk を保持する他のクライアントの情報 (IP アドレス等) を提供する．クライアントは互いに Chunk を交換しながら収集し，集めた Chunk とメタデータファイルの情報を用いて元のコンテンツを復元する．この仕組みによって，サーバにおける通信量は各クライアントに対して直接コンテンツを提供する場合よりも大幅に削減される．しかし BitTorrent，Antfarm ともに，各クライアントが安定的に接続されている状況を想定しているため，クライアントの入れ替わりの激しい Flash Crowds 発生時には配信効率が低下することがある．Flashback は，Interval Skiplist[10] による通信量を抑えた Chunk 管理や，ハンドシェイクに Chunk 保持の確認を含めた独自の接続プロトコルを用いることで，Flash Crowds 発生時のようなクライアントの入れ替わりの激しい状況においても効率の低下しないコンテンツ配信を実現する．これらの手法はいずれも，多数のクライアントに対する安定的なコンテンツ配信を実現しているが，サーバを基点にして Chunk の管理を行っているためサーバがボトルネックとなる可能性がある．また，動的コンテンツについてはサーバがリクエストを受けた際に Chunk を生成することで対応可能だが，外部コンテンツについてはクライアントが外部のサーバへ直接リクエストするため対応できない．

3. 提 案 手 法

3.1 概　　要

本論文の提案手法は，クライアント間における P2P ネットワークでキャッシュを管理・共有することで，サーバの処理するリクエスト数を減少させ，さらに，動的 / 外部コンテンツへの対応を実現する．本手法の設計にあたり，Flash Crowds は多数のクライアントが同じコンテンツを求めて殺到している現象である点と，動的 / 外部コンテンツの多くは同じクエリに対し同じコンテンツが得られるという点に注目した．ここでクエリとは， URL，GET / POST パラメータ，サーバに送信される Cookie の内容等を指す．同じコンテンツを要求するクライアントが多く存在するほど，そのコンテンツを共有することでサーバの処理するリクエスト数を大きく削減できる．また，クエリによって一意にコンテンツが定まることため，取得したコンテンツをキャッシュとして共有する際にクエリをキーとして管理できる．これにより，動的 / 外部コンテンツについても静的コンテンツと同様に管理し共有することが可能となる．ただし，ショッピングカートの中身のようにクライアント毎に内容の異なるコンテンツについては，他のクライアントが利用することがないため，本手法による負荷軽減の対象としない．

本手法では，各クライアントはコンテンツの取得にあたり，まず共有ネットワークに対してクエリを発行し，キャッシュの取得を試みる．そして，キャッシュを取得できなかった場合にのみサーバに対してクエリを発行し，コンテンツを取得する．このとき，サーバからコンテンツを取得したクライアントは，共有ネットワークに取得したコンテンツをキャッシュとして提供する．この仕組みにより，キャッシュとして共有されているコンテンツを要求する後続のクライアントは，サーバに対してクエリを発行することなくコンテンツを得ることができ，サーバの処理するリクエスト数が削減される．図 1 にこの仕組みを示す．コンテンツ A は共有されているため，クエリ A を発行したクライアントはサーバにアクセスすることなくコンテンツ A を取得できる．一方，コンテンツ C は共有されていないため，クエリ C を発行したクライアントは共有ネットワークからコンテンツ C を取得できない．そのため，サーバに対してクエリ C を発行してコンテンツ C を取得する．このとき，取得したコンテンツ C はキャッシュとして共有ネットワークに提供する．

3.2 設　　計

提案手法では高いスケーラビリティを実現するために DHT を用い，サーバや特定のクライアントが特別な役割を持たない Pure-P2P 型の共有ネットワークを構築する．Pure-P2P

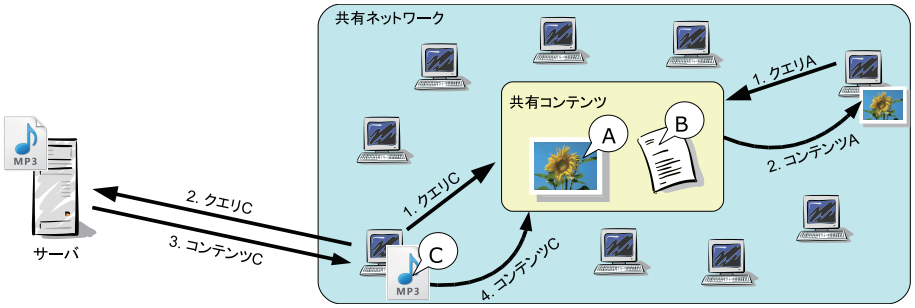

図 1 提案手法の概要

型とすることでボトルネックが生じる可能性を抑えることができる．また本手法では，クエリによってコンテンツが一意に定まるとしているため，キーと値を一対一で管理する DHT はこの点でも適している．しかし，コンテンツ単位という粗い粒度のままデータを管理すると，大容量コンテンツや人気の高いコンテンツを保持するクライアントの負荷が他に比べ著しく増大してシステムのスケーラビリティを下げる可能性があるため，本手法ではコンテンツを分割して Chunk として扱う．このとき，Chunk を収集して元のコンテンツを復元するためのメタデータが必要となる．本手法ではこのメタデータを File Meta Data (FMD) と呼ぶ．FMD には Chunk を特定するためのキーや Chunk を結合する順番などを含む．コンテンツは 1 つの FMD と複数の Chunk として共有される．FMD と Chunk の生成および共有ネットワークへの提供は，サーバからコンテンツを取得したクライアントが行う．各クライアントは FMD と Chunk からコンテンツ復元するが，各クライアントのデータ管理負荷はより均一化されるため，ボトルネックが生じる可能性をさらに抑えることができる．

3.2.1 DHT による共有ネットワーク

提案手法は DHT を用いて FMD と Chunk を管理するクライアントを定める．共有ネットワークに利用する DHT ではキーと値として

- キーをクエリのダイジェスト，対応する値を FMD を保持するクライアント情報
- キーを Chunk のダイジェスト，対応する値を Chunk を保持するクライアント情報

を管理する．ダイジェストを用いるのはキーの形態として統一性を持たせ，FMD と Chunk を同じ DHT で管理できるようにするためである．また，コンテンツを形成する全ての Chunk のダイジェストを FMD に含むことで，クエリから FMD，FMD から全ての Chunk を得ることができる．すなわち，本来サーバに対して発行されるクエリさえあれば，共有ネットワークからキャッシュを得ることが可能になる．

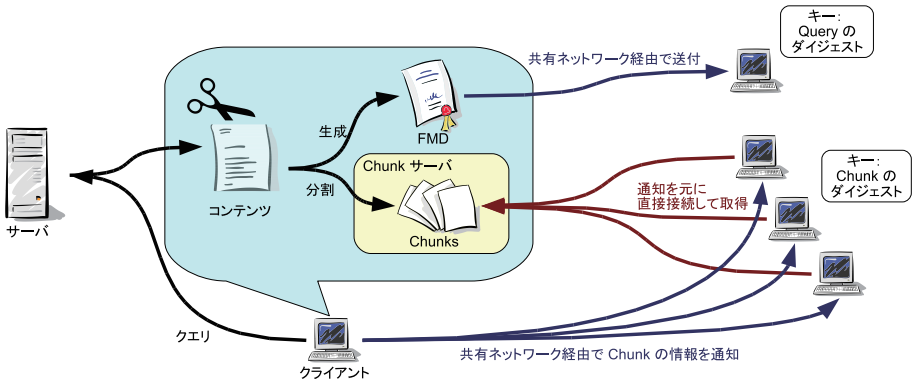

図 2 コンテンツの取得から FMD・Chunk の配置までの流れ

3.2.2 FMD と Chunk によるコンテンツ共有

提案手法ではサーバからコンテンツを取得したクライアントが FMD と Chunk を生成し，共有ネットワークに配置する．図 2 にこの流れを示す．まず，サーバに対してクエリを発行しコンテンツを取得したクライアントは，コンテンツを分割して Chunk を生成する．次に，各 Chunk のダイジェストと Chunk の結合方法を FMD に記述し，クエリのダイジェストにより特定されるクライアントに FMD を送付する．続いて，生成した Chunk を Chunk サーバで公開し，各 Chunk のダイジェストから DHT に従って特定されるクライアントに対して，Chunk を取得するための情報を通知する．最後に，通知を受けた各クライアントが，通知を元に Chunk サーバから Chunk を取得することで， Chunk の配置が完了する．コンテンツデータを先頭から一定サイズに分割するという最も単純な例では，分割したそれぞれを Chunk とし各 Chunk のダイジェストを分割順に FMD に記述する．クライアントは， FMD に記載された順で Chunk を結合することでコンテンツを復元することができる．各 Chunk に対し暗号化や圧縮処理を加える必要がある場合，暗号形式とパスワードや圧縮形式等の情報を FMD に付加することで実装することも可能である．

図 3 にクライアントがクエリを発行してからコンテンツを得るまでの手順を示す．各クライアントはクエリのダイジェストをキーとして DHT から FMD の所在を取得し，FMD を要求するメッセージを送る．FMD を取得できた場合にはそれに含まれるキーを用いて Chunk を収集し， FMD の情報を用いてコンテンツを復元する．FMD を取得できなかっ

```
 1: クエリから FMD を取得するためのキーを生成;
 2: 求める FMD を担当するクライアントを DHT から取得;
 3: そのクライアントに FMD を要求;
 4: if (FMD が返ってきた) {
 5:     foreach (FMD に含まれる Chunk のキー) {
 6:         求める Chunk を管理するクライアントを DHT から取得;
 7:         そのクライアントに Chunk を要求;
 8:         if (クライアントから Chunk を得られなかった) {
 9:             break;
10:         }
11:     }
12:     if (全 Chunk を集められた) {
13:         FMD を基に Chunk を結合してコンテンツを得る;
14:         return コンテンツ;
15:     }
16: }
17: サーバから直接コンテンツを取得;
18: コンテンツを分割し FMD と Chunk を生成;
19: FMD を担当するクライアントに FMD を送信;
20: foreach (生成したすべての Chunk) {
21:     Chunk を担当するクライアントを DHT から取得;
22:     そのクライアントに Chunk を取得するように要求する;
23: }
24: return コンテンツ;
```

図 3 クライアントにおけるコンテンツ取得の手順

た場合や Chunk が揃わなかった場合には，サーバに対してクエリを発行してコンテンツを取得する．この時サーバから取得したコンテンツを共有するために，FMD と Chunk を生成して共有ネットワーク上に配置する．すべてのクライアントに対してこの手順を課すことで，新たに需要が生じたコンテンツや構成データに欠損が生じたコンテンツなどもすぐに共有ネットワークへ反映することが可能となる．

FMD と Chunk の授受には，共有ネットワーク上におけるメッセージと，共有ネットワークを介さずに直接接続しての通信を併用する．FMD に比べて Chunk のデータサイズは大きく，共有ネットワークを介して授受すると効率が悪いため，FMD に含まれるダイジェストから Chunk を保持するクライアントの情報を得て，直接接続して取得する．一方 FMD はダイジェスト等を含むだけでデータサイズが比較的小さいため，改めて直接接続することはせず共有ネットワーク上のメッセージに FMD を含めて授受する．

表 1 各種メッセージとそのハンドラ

メッセージ	内容	ハンドラ (メッセージを受けたクライアントが行う処理)
HandoverFMD	FMD	FMD を管理テーブルに追加する．
HandoverChunk	Chunk 取得のための情報	情報を元に Chunk をダウンロードし，管理テーブルに追加する．
RequestFMD	クエリのダイジェスト	管理テーブルから該当する FMD を探して返信する．
RequestChunk	Chunk のダイジェスト	管理テーブルから該当する Chunk を探し，取得のための情報を返信する．
ReplyFMD	FMD	FMD に含まれる Chunk のダイジェストで特定されるクライアントに，RequestChunk メッセージを送る．
ReplyChunk	Chunk 取得のための情報	メッセージの内容を元に Chunk をダウンロードし，管理テーブルに追加する．

メッセージ共有ネットワークに参加しているクライアント間では共有ネットワークの保守の他に，FMD・Chunk を共有するために各種メッセージが授受されている．提案手法で独自に設けたメッセージは表 1 に示す 6 つである．

3.2.3 クライアントの構成

提案手法では Pure-P2P 型の共有ネットワークを構築するため特別なクライアントは存在せず，各クライアントは同様の仕組みを備えている．図 4 にクライアントの構成を示す．DHT とルーティングサービスは，共有ネットワークに参加するために用いる．HTTP クライアントは，サーバからコンテンツを取得するために用いる．FMD と Chunk ジェネレータは，取得したコンテンツから FMD と Chunk を生成する．Digest ジェネレータは，クエリ や Chunk のダイジェストの生成に用いる．FMD テーブルは，FMD とその有効期限を Chunk のダイジェストをキーにして保持する．Chunk テーブルは，Chunk ファイルとその有効期限を，Chunk のダイジェストをキーにして保持する．Chunk サーバ / クライアントは，他のクライアントと Chunk を授受するために用いる．

本手法は，オリジナルコンテンツが更新されているにも関わらず，古いコンテンツが延々と共有され続けることを避けるために，FMD / Chunk テーブルでは各データの有効期限も管理している．有効期限には例えば，データをテーブルに登録した時刻から一定時間経過後に破棄することや，データを一定回数提供した後に破棄すること等の実装が考えられる．有効期限による管理は，FMD や Chunk の不必要な破棄によるサーバアクセスを発生させる他，オリジナルコンテンツ更新の反映を遅らせるといった問題点も含む．しかし，無駄な破棄が発生しても本手法が通常時よりサーバへのアクセス数を減少させていることには変わ

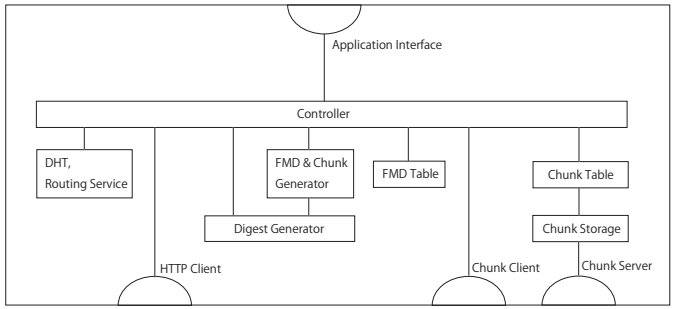

図 4 クライアントの構成

りなく，Flash Crowds 対策として有効である．また，各データの有効期限を必要以上に長くしなければオリジナルコンテンツの更新が延々と反映されないということも起こりづらいと考えられる．よって，本手法では有効期限によってコンテンツの更新を行うものとする．

4. 実　　装

提案手法の有用性を示すために Overlay Weaver[11] 上に実装した．Overlay Weaver はオーバーレイネットワーク構築ツールキットであり，多数のノードが参加するネットワークのシミュレーションも行うことができる．

DHT には Overlay Weaver で提供されている Chord[12] の実装を利用した．Chord は参加ノード数 n に対して $O(\log n)$ のホップ数で目的のノードに到達することができる．すなわち，Chord は参加ノード数が多い状況でも比較的高速に目的のノードを探索することが可能なルーティングアルゴリズムであり，多数のクライアントが共有ネットワークへ参加する状況に適している．

今回は評価を簡単にするため，コンテンツデータを先頭から 40 KB 毎に分割したものを Chunk とし，FMD には分割した順番通りに各 Chunk のダイジェストを記録した．ダイジェストの生成には MD5[13] を用いた．また，FMD と Chunk 管理テーブルで管理される有効期限には，コンテンツの提供回数を制限する手法をとった．コンテンツ提供の制限回数を示す正の整数値として Time To Live (TTL) を設定した．各クライアントは対象の FMD と Chunk が管理テーブルから取得される度に TTL をデクリメントし，TTL がゼロになった時点でその FMD・Chunk を破棄する．Chunk サーバには Apache HTTP サーバ[14] を用いた．提案手法で独自に設けたメッセージは Overlay Weaver で提供されているメッセージングサービスを利用して実装した．

表 2 実験マシンの詳細

	クライアントエミュレータ	コンテンツサーバ
OS	Linux-2.6.20	Linux-2.6.27
CPU	Intel(R) Xeon 3.00 GHz (4 Cores)	Intel(R) Pentium4 2.80 GHz
メモリ	12 GB	2 GB
ネットワーク	Gigabit ethernet	

5. 実　　験

多数のクライアントによるアクセスが集中する状況において，本手法によってサーバへのリクエスト数が減少し，それに伴ってサーバにかかる負荷が軽減されることを示すため以下の実験を行った．

5.1 方　　法

Overlay Weaver を用い 1 台の計算機上で 10,000 台のクライアントノードをエミュレートした．また，コンテンツサーバとして 1 台の計算機を用いた．2 台の計算機は専用のネットワークインターフェース及び LAN ケーブルで直に接続した．機器の詳細は表 2 の通りである．クライアントエミュレータには Overlay Weaver-0.8.6 ，コンテンツサーバには Apache-2.2.9 をデフォルトのパラメータ設定で用いた．各クライアントにおける Chunk サーバには Apache-2.2.4 をデフォルトのパラメータ設定で用いた．

コンテンツサーバで提供するコンテンツには /dev/urandom の内容を dd コマンドで 100 KB 分取得して生成したファイル，すなわち静的コンテンツを用いて評価を行った．ここまでに説明したように本手法はその仕組みから静的／動的／外部コンテンツの区別なく適用可能なため，静的コンテンツでも評価可能である．

本実験では提案手法を利用しない場合と利用する場合 (TTL は 10, 20 の 2 通り，Chunk サイズは 40 KB) を比較した．利用しない場合はすべてのクライアントがコンテンツサーバに直接リクエストを発行し，利用する場合は各クライアントは優先的に共有コンテンツを取得した．サーバ負荷の評価指標としてリクエスト処理数とネットワーク帯域使用量を用いた．リクエスト処理数を用いたのは，本手法がサーバへのリクエストそのものを減少させることを示すためである．ネットワーク帯域使用量を用いたのは，リクエストの減少によってサーバへの負荷が実際に減少していることを示すためである．

また，与えるワークロードとしてリクエストの発行頻度が毎秒 200 件となるようにクラ

表 3 TTL とサーバが処理したリクエスト数の関係

提案手法	なし	あり	
TTL	-	10	20
リクエスト処理数 [#]	10,000	1,000 (-90.0 %)	500 (-95.0 %)
平均リクエスト処理頻度 [#/sec]	188.7	9.2 (-95.1 %)	4.6 (-97.6 %)

イアントエミュレータを設定した．但し，これはクライアント側において毎秒 200 件の頻度でコンテンツのリクエスト処理を開始することを意味しており，この頻度でサーバへリクエストが到達することを保証するものではない．今回は各クライアントに 1 件ずつリクエストを発行させるため， 50 秒間で合計 10,000 件のリクエストが発行される．ただし，ディスクへの書き込みによる遅延を緩和するために，Chunk の保存には ramdisk を用い，生成したコンテンツはディスクへ書き込まずに /dev/null に破棄している．

5.2 評　　価

5.2.1 リクエスト処理数と頻度

表 3 は提案手法の有無及び TTL の違いによる，サーバにおけるリクエスト処理数と処理頻度の平均値を比較したものである．また，図 5 にサーバでのリクエスト処理頻度の時間変化を示す．横軸は時間経過，縦軸はサーバにおける 1 秒間あたりのリクエスト処理数である．なお，縦軸は対数軸である．

提案手法を用いなかった場合はちょうど 10,000 件のリクエストをサーバが処理しており，平均毎秒 188.7 件というリクエスト処理数も与えたワークロードとほぼ一致している．平均値が 200 件に達しなかったのは各リクエストの処理に要する時間があるためと考えられる．

提案手法を用いた場合のリクエストの処理数は TTL = 10, 20 でそれぞれ 1,000 件，500 件となり提案手法を用いない場合の 1/10, 1/20 となった．これは FMD と Chunk がそれぞれ TTL で設定した台数のクライアントに対して有効に働いたことを示している．このことから提案手法によって，サーバへのリクエスト数が減少したと言える．リクエスト頻度も 1/10, 1/20 に減少すると予想していたものの，TTL = 10, 20 のそれぞれの場合においてリクエスト頻度はそれぞれ 1/10, 1/20 よりもさらに小さくなっており，図 5 から全てのリクエストが処理されるまでの時間が長くなっていることが確認された．提案手法を用いなかった場合には毎秒 188.7 件のリクエスト処理ができていることからサーバの過負荷により処理時間が長くなったとは考えづらい．そのため，各クライアントでの処理に時間がかかった分リクエストの発行が遅れたと思われる．この遅れは 1 台の実機上で多数のクライアントをエミュレートしていることによると考えられる．しかし，リクエスト数が 1/TTL に減少していることから，理論的にリクエスト頻度が 1/TTL を超えることはなく，TTL によってサーバへの負荷軽減の程度を調整可能である．また，この実験により TTL を 20 に設定することで約 98 % の負荷軽減ができることが確認された．

表 4 ネットワーク帯域使用量の比較

提案手法	なし	あり	
TTL	-	10	20
平均ネットワーク帯域使用量 [MB/sec]	21.29	1.01 (-95.3 %)	0.51 (-97.6 %)
リクエスト処理頻度 × コンテンツサイズ [MB/sec]	18.8	0.92 (-95.1 %)	0.46 (-97.6 %)

5.2.2 ネットワーク帯域使用量

表 4 は提案手法の有無及び TTL の違いによる，サーバのネットワークインターフェースにおけるネットワーク帯域使用量の平均値を比較したものである．下段の「リクエスト処理頻度 × コンテンツサイズ」はネットワーク帯域使用量と比較するために記載したもので，通信のオーバーヘッドを考慮しなかった場合はこの分のデータ量がサーバから送信されている．また，図 6 にネットワーク帯域使用量の時間変化を示す．横軸が時間経過，縦軸はサーバのネットワークインターフェースにおける 1 秒間あたりのネットワーク帯域使用量である．なお，縦軸は対数軸である．

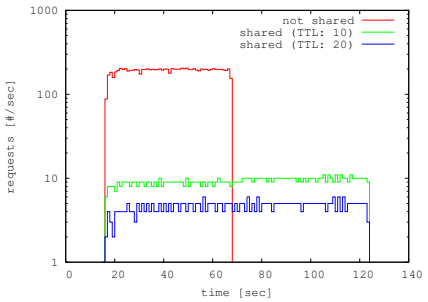

図 5 リクエスト処理頻度の比較

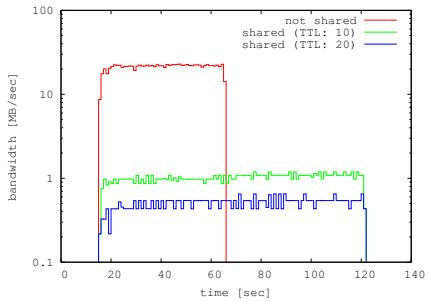

図 6 ネットワーク帯域使用量の比較

図 5 と 6 の比較から，ネットワーク帯域使用量は概ねリクエスト処理頻度を反映していることがわかる．本手法を用いなかった場合は平均して 21.29 MB/sec の転送量があったが，本手法を用いた場合には TTL の設定に応じて 1.01 MB/sec, 0.51 MB/sec と大幅に減少している．すなわち，本手法の利用によってサーバのネットワーク帯域にかかる負担が

小さくなった．これにより，リクエスト処理頻度の減少によってサーバの負荷を軽減できることを確認した．また，リクエスト処理頻度と同様に TTL を 20 に設定することで約 98 % の負荷軽減ができた．

6. おわりに

本論文では，マッシュアップを利用する Web サーバに対して，Flash Crowds への耐性を持ったコンテンツ配信を実現する手法を提案した．本機構では，クライアント間で Pure-P2P 型のネットワークを形成し，取得済みのコンテンツを管理し共有する．各クライアントは共有ネットワークから優先的にコンテンツの取得を試みるため，サーバが処理するリクエスト数を減少させることが可能となる．また，コンテンツの管理と共有の機能をサーバから完全に切り離したことで，どのサーバからどのように提供されたかに関係なくコンテンツの共有を実現した．すなわち，静的／動的／外部コンテンツの隔たりなくクライアント間での共有が可能となり，現状の Web サービスとの親和性も高い．実験によりサーバへのリクエスト頻度とサーバのネットワーク帯域使用量が約 97 % 減少することを確認した．今後の課題としては，同じクエリを発行しても異なる結果が得られる場合への対応，FMD と Chunk の更新基準の改善，コンテンツ取得の効率化などが挙げられる．

謝辞 本研究の一部は，文部科学省科学研究費補助金特定領域研究「情報爆発時代に向けた新しい IT 基盤技術の研究」による支援を受けている．

参考文献

1) Jung, J., Krishnamurthy, B. and Rabinovich, M.: Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites, pp. 293–304 (2002).
2) SourceForge, Inc.: Slashdots. http://slashdots.org/.
3) YouTube, LLC: Youtube. http://www.youtube.com/.
4) Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R. and Weihl, B.: Globally Distributed Content Delivery, *IEEE Internet Computing*, Vol.6, No.5, pp.50–58 (2002).
5) Akamai Technologies: Akamai. http://www.akamai.com/.
6) Akamai Technologies: Akamai Provides Insight into Internet Denial of Service Attack. http://www.akamai.com/html/about/press/releases/2004/press_061604.html.
7) Cohen, B.: Incentives Build Robustness in BitTorrent, *Proceedings of Workshop on Economics of Peer-to-Peer Systems* (2003).
8) Peterson, R.S. and Sirer, E.G.: Antfarm: Efficient Content Distribution with Managed Swarms, *Proceedings of USENIX Symposium on Networked Systems Design and Implementation* (2009).
9) Deshpande, M., Amit, A., Chang, M., Venkatasubramanian, N. and Mehrotra, S.: Flashback: A Peer-to-Peer Web Server for Flash Crowds, *Proceedings of IEEE International Conference on Distributed Computing Systems* (2007).
10) Hanson, E.N. and Johnson, T.: The interval skip list: A data structure for finding all intervals that overlap a point, *Proceedings of Workshop on Algorithms and Data Structures* (1992).
11) 首藤一幸，田中良夫，関口智嗣：オーバレイ構築ツールキット Overlay Weaver，情報処理学会論文誌：コンピューティングシステム， Vol.47, No.SIG12 (ACS 15), pp. 358–367 (2006).
12) Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, *Proceedings of ACM Special Interest Group on Data Communications Conference* (2001).
13) Rivest, R.: The MD5 Message-Digest Algorithm, *RFC1321* (1992).
14) The Apache Software Foundation: Apache HTTP Server. http://httpd.apache.org/.