WWW クラスタにおける同期を考慮したコンテンツ更新機構の設計と実装
—夏の高校野球 WWW 中継における運用—

The Design and Implementation of Contents Update Mechanisms on a WWW Server Cluster
—Operation at the National High School Baseball Tournament—

西馬一郎 * 河合栄治 † 知念賢一 *
Ichiro Nishiuma Eiji Kawai Ken-ichi Chinen
吉田豊一 § 香取啓志 § 山口英 *
Toyokazu Yoshida Keishi Kandori Suguru Yamaguchi

概要

アクセスが集中するWWWサイトでは、負荷を分散させるため、複数のWWWサーバホストを用意してクラスタを構成することが一般的である。しかし、コンテンツの更新時に、クラスタ内でコンテンツの一貫性が損なわれる問題がある。そこで本研究では、公開時刻を制御することによりコンテンツの一貫性を保つ機構の設計・実装を行なった。また、このシステムを高校野球WWWサイトに導入して運用実験を行ない、実環境における有効性を確認することができた。

1 はじめに

大規模なWWWサイトでは、WWWサーバホストを複数用意してWWWクラスタを構築し、DNSラウンドロビンやレイヤ4またはレイヤ7スイッチと呼ばれる装置を用いて負荷分散を行なう。これにより、クラスタに到着するリクエストを複数のホストに分散し、クラスタ全体で高いスループットを達成する。

リクエストが集中するサイトではコンテンツが頻繁に更新されることが多い。ここで、コンテンツとはWWWページを構成するファイルであり、HTMLファイル、JPEGファイル、GIFファイルなどを指す。コンテンツの更新頻度が高いクラスタでは、複数のサーバで更新が同時に完了しなければ各サーバによって公開するコンテンツに差異が生じ、コンテンツの一貫性が損なわれる時間が存在することになる。特にリクエストが集中して大量に到来するクラスタでは、このような時間が問題になる。さらに、チケットの予約、期間限定サイトなど、リアルタイム性が要求されるコンテンツの場合、問題が顕著となる。そのため、このようなサイトを想定すると、クラスタのコンテンツの一貫性を保持することが重要となり、従来手法よりも高い精度で更新を完了させる必要がある。

本研究では、更新時にクラスタ内のコンテンツの一貫性を保つため、コンテンツの公開制御機構を提案する。提案方式は、サーバがコンテンツを格納しても即時に公開せず、他の全てのサーバにコンテンツが配布されたことを確認し、公開時刻を決めることによって同期をとる。これにより、サーバ間でのコンテンツの一貫性の保持が実現される。提案方式の優位性を検証するために提案方式と他方式を比較する実験を行ない、さらに、高校野球のWWWサイトに適用し実運用によって提案方式を評価した。

* 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
† 科学技術振興事業団
Japan Science and Technology Corporation
§ 朝日放送
Asahi Broadcasting Corporation

2 問題点

複数のサーバにおいてコンテンツを更新する時、現状では更新が同時に実現されなければサーバ間でコンテンツの不一致が発生する。一般的に、コンテンツの更新とは既存のコンテンツに変更を加えることである。本研究では、新規にコンテンツを作成することもコンテンツの更新と捉える。クラスタにおいてコンテンツの更新を実現するためには、変更したコンテンツをクラスタを構成する全サーバに配送しなければならない。コンテンツの転送、格納の処理に要する時間がクラスタ内のそれぞれのサーバで同じとは限らない。よって、コンテンツを格納する時刻に差が生じる。通常、コンテンツを転送し格納すると即時にそのコンテンツは公開される。このため、コンテンツの格納のタイミングのずれが公開のタイミングのずれになり、ユーザが閲覧するコンテンツに差が生じてしまう。また更新されたコンテンツの配送に失敗した場合、そのコンテンツはクラスタ内で一致していない。

更新の処理時間の差はそれぞれのサーバの負荷、状態などによって異なる。サーバに与える負荷はクライアントからのリクエスト数に依存し、各サーバの負荷は一様ではない。また、ネットワークを介して更新を行なう時、ネットワークの状況も更新の処理時間の差となる。

複数のサーバを用いてクラスタを構築する際、WWW サーバを使う場合とリバースプロキシを使う場合がある。コンテンツの更新を実現する手法はそれぞれ異なる。ここではその場合の更新実現方法と問題点を指摘する。

2.1 WWW サーバを使う場合

WWW サーバに対してコンテンツの更新をする時、図 1(a) に示すように WWW サーバが公開しているファイルシステムにコンテンツを転送する。このような手段には、rsync [1] や rcp あるいは FTP [2] などがある。各 WWW サーバで格納が終了すると、その時点でコンテンツは公開される。このため、複数の WWW サーバへの転送処理を同時に行ない、かつ同時に格納処理が終了しなければ同時公開は実現できない。

また、NFS (Network File System) を利用して複数の WWW サーバが一つのファイルシステムを共有することにより、コンテンツを公開する手法も考えられる。この場合、複数のサーバホストにコンテンツを分配する必要はないが、NFS サーバに負荷が集中するなどの問題があり、大規模 WWW システムで用いるのは難しい。

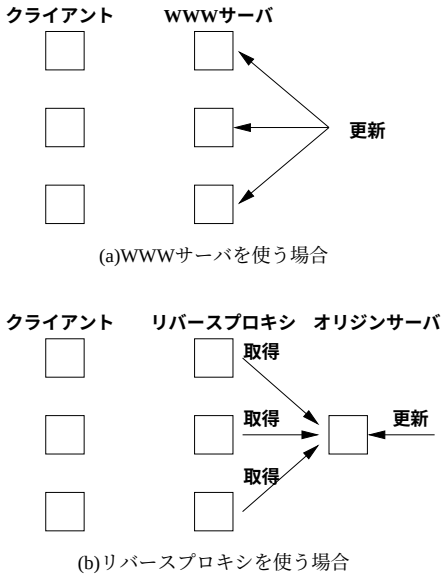

図 1: WWW サイトの構成例

2.2 リバースプロキシを使う場合

WWW サーバに対する負荷を軽減するために、クライアントとサーバの間にキャッシュシステムを設置する手法がとられることが多い。このキャッシュシステムをリバースプロキシと呼ぶ。リバースプロキシは図 1(b) のように、WWW サーバ (オリジンサーバ) からコンテンツを取得してキャッシュに格納しクライアントに応答する。このキャッシュを再利用することにより、オリジンサーバの負荷の軽減を実現する。

リバースプロキシは自律的にオリジンサーバからコンテンツを取得する。よってディスクが故障するなどシステムの障害からの復旧が容易である。また、管理者やコンテンツ供給者によるコンテンツの管理を必要としない。よってリバースプロキシは WWW クラスタへの導入が容易である。

ただし、クラスタを構成するリバースプロキシ全てで更新を制御するのは難しい。リバースプロキシはクライアントからの条件付き要求 (If-Modified-Since ヘッダ付 GET) を受けた時、オリジンサーバ

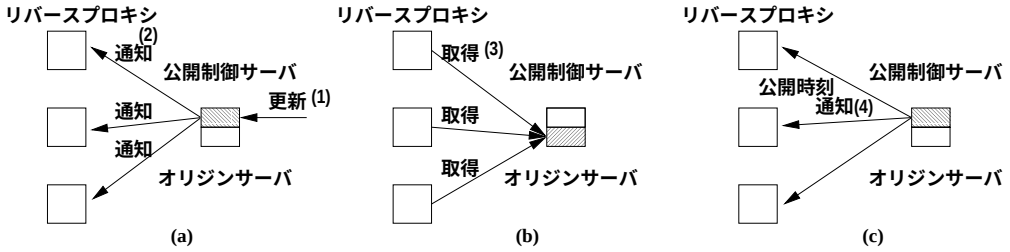

図 2: リバースプロキシとコンテンツ制御サーバ、オリジンサーバの通信

に対して、そのコンテンツが更新されているかどうか確かめる。また、要求内の Cache-Control ヘッダに no-cache が指定されるとキャッシュを用いず、オリジンサーバから直接コンテンツを取得する。このようなリクエストは全体的には少なく、これを用いてオリジンサーバの更新を反映することは難しい。さらにこのような更新処理は単体のリバースプロキシで発生するため全てのリバースプロキシに、反映できない。これらの理由から、コンテンツの一貫性を保つためには、明示的にコンテンツの更新を全てのリバースプロキシに指示する機構が必要となる。

3 設計

前章での問題点を解決するために以下のようなシステムを設計した。本システムでは、格納時間に起因する公開時間のずれを吸収するため、コンテンツの格納段階と公開段階に分離した。格納処理が終了した後に公開時刻を通知し、一斉公開することによって、更新したコンテンツの同期をとる。

本システムでは、前章で述べたリバースプロキシによる WWW クラスタを用いた。自律的に動作するリバースプロキシに格納されている全てのコンテンツを把握し、それらをオリジンサーバと一致させるよう制御するのは難しい。そこで、本システムでは更新したコンテンツに関してのみリバースプロキシを制御することでコンテンツの一貫性を保つよう設計した。ここで必要となるのは、更新したコンテンツをリバースプロキシに通知する機能と、リバースプロキシにおいて更新したコンテンツの公開を管理する機能である。それぞれの機能を満たすため、コンテンツ制御サーバを新しく設けた。リバースプロキシはコンテンツ制御

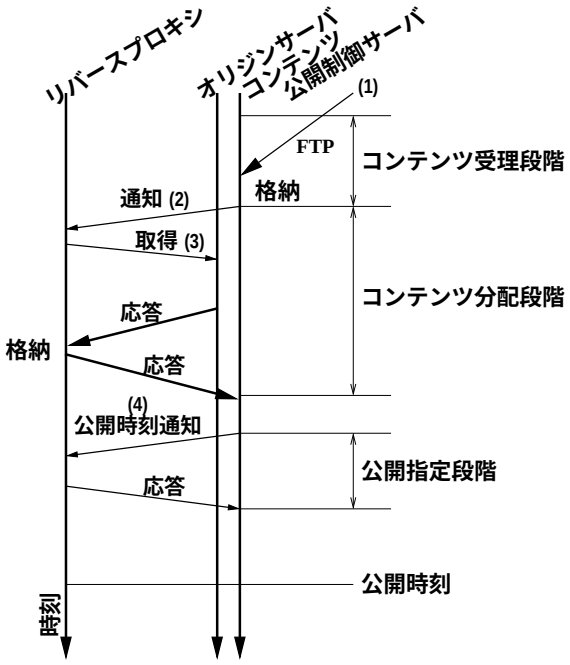

図 3: 公開までの段階

サーバの指示に従う。

また、更新したコンテンツ以外、つまり更新されていない、あるいは保持していないコンテンツに対するリクエストについては、リバースプロキシが自律的にオリジンサーバよりコンテンツを取得するので特に処理を追加しなくてもオリジンサーバのコンテンツが反映される。

以下に本システムを構成するリバースプロキシおよびコンテンツ公開制御サーバの設計について述べる。

3.1 コンテンツ公開制御サーバの設計

更新したコンテンツの公開を制御するサーバは、クラスタを構成するリバースプロキシに対してコンテンツの取得と公開を促すリクエストを発行し、各リバースプロキシのコンテンツの公開を制御する。コンテンツ制御サーバとリバースプロキシ間の通信を図 2 に示す。

コンテンツ公開制御サーバは図 2 (a) (1) にあるように、コンテンツ作成者から更新したコンテンツを受理する。これによりコンテンツの更新を検知し、そのコンテンツを格納する。コンテンツ公開制御サーバはオリジンサーバとファイルシステムを共有する。コンテンツ公開制御サーバがコンテンツを受理するまでをコンテンツ受理段階と呼ぶ。

次に、コンテンツ公開制御サーバは、図 2 (a) (2) にあるように全リバースプロキシに対し検知したコンテンツの取得を促す要求を発行する。リバースプロキシは、この要求に応じて図 2 (b) にあるようにコンテンツをオリジンサーバから取得し、その成否を示す応答をコンテンツ公開制御サーバに返す。コンテンツ公開制御サーバのコンテンツ格納先をオリジンサーバの公開するファイルシステムと共有したのは、リバースプロキシのコンテンツ取得先がオリジンサーバだからである。この通信により、コンテンツ作成者から受理したコンテンツは全てのリバースプロキシに存在することになる。コンテンツ公開制御サーバがリバースプロキシにコンテンツ取得要求を発行して、応答を受けるまでをコンテンツ分配段階と呼ぶ。

リバースプロキシは、コンテンツ制御サーバの要求に従いオリジンサーバからコンテンツを取得する。この時点では、このコンテンツをクライアントに対して公開しない。全リバースプロキシがコンテンツを取得したことを確認すると、コンテンツ公開制御サーバは公開時刻をリバースプロキシに通知する (図 2 (c)) 。これに従い、リバースプロキシは指定公開時刻にこのコンテンツを公開する。それにより一斉公開を実現することができる。この段階を公開指定段階と呼ぶ。

以上に述べた通信と段階を図 3 に示す。これら一連の処理により、コンテンツ作成者は、更新したコンテンツをコンテンツ公開制御サーバに転送するだけで、すべてのリバースプロキシにそのコンテンツが分配され公開される。

3.2 リバースプロキシの設計

リバースプロキシは前節で述べたように、コンテンツ更新時にコンテンツ制御サーバに従って動作する。

リバースプロキシは、更新の通信が滞るおそれがあることを想定してこれを回復するために、一定期間保持したコンテンツについてオリジンサーバにリクエストを発行してコンテンツが更新されたかどうか確認する。

3.3 コンテンツ更新時のファイルの取り扱い

コンテンツ公開制御サーバがコンテンツを受理してから公開するまで、コンテンツの転送・格納は、一時的ファイル名を介して行なう。これは既存プログラムとの整合性をとるためである。更新を実現するために同じファイル名で中身の異なるファイルを同時に二つ持つことが必要だが、一般的なファイルシステムでは実現できない。

コンテンツ公開制御サーバは、コンテンツ作成者からファイルを受けとると、このファイルを一時的ファイル名で保存する。その後、コンテンツ公開制御サーバは全リバースプロキシにこのコンテンツの取得を促すリクエストを発行する。リバースプロキシは一時ファイル名のコンテンツを取得し、これを格納する。公開制御サーバでは、公開時刻に到達した時点で一時的ファイル名を公開するべきファイル名に置き換える。一方、リバースプロキシ側でも公開するべきファイル名に置き換え、クライアントに提供する。このように公開時刻になるまでコンテンツを一時ファイル名で保持することにより、公開時刻を制御することができる。

3.4 コンテンツ公開制御サーバ・リバースプロキシ間の通信プロトコル

コンテンツ公開制御サーバ・リバースプロキシ間のコンテンツ公開制御のための通信をクライアントからのリクエストと別扱いするよう、HTTP にヘッダを加えた。これにより、コンテンツ公開制御サーバから全リバースプロキシへコンテンツの分配、公開時刻の指定を行なう。用いたメソッドは GET である。

コンテンツ分配段階では、コンテンツ公開制御サーバは X-Notify ヘッダを含んだリクエストを全リバースプロキシに対して発行する。X-Notify ヘッ

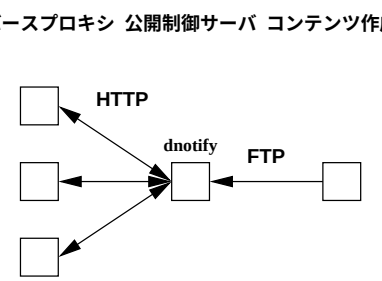

図 4: 提案手法によるコンテンツの更新

ダを含むリクエストを受けたリバースプロキシは該当するコンテンツをオリジンサーバから取得し、コンテンツ公開制御サーバからのリクエストに対する応答を行なう。この時のリクエストを以下に示す。

```
GET /index.html HTTP/1.0
X-Notify:
```

公開指定段階では、コンテンツ公開制御サーバは X-Public ヘッダを含むリクエストを全リバースプロキシに送信する。これは、コンテンツ公開制御サーバが全リバースプロキシに対してコンテンツの公開時刻を通知するものである。この時のリクエストを以下に示す。日時を表すフォーマットは RFC1123 [4] 形式を用いた。

```
GET /index.html HTTP/1.0
X-Public: Mon, 19 Jul 2001 19:39:24 GMT
```

3.5 コンテンツの識別方法

提案システムでは、全リバースプロキシやコンテンツ公開制御サーバの間でコンテンツの内容が一致するもの、または内容が異なるものを検出する必要がある。この時、更新されたコンテンツは名前が同じであるので、そのコンテンツの中身で識別しなければならない。ファイルサイズや最終更新時刻などでも実現できるが、今回は厳密さを期するため、HTTP/1.1 に記述されている ETag を用いた。

4 実装

4.1 コンテンツ公開制御サーバ

今回、コンテンツを受理し、これを更新通知としてリクエストを発行するプログラム、dnotify を開発した。図 4 のように、コンテンツ公開制御サーバは FTP サーバとして動作し、コンテンツ作成者より FTP でコンテンツを受理し、格納する。また、そのコンテンツについてリバースプロキシと通信をした時刻、レスポンスコード、ETag などを管理する。

コンテンツ公開制御サーバは、リバースプロキシとの公開指定段階の通信時間の平均値を直近の履歴から算出し、それに基づいて公開時刻を決定する。これはリバースプロキシとの通信時間がネットワークの状況、双方の負荷や状態により変化することに対応するためである。また、公開時刻は秒単位で指定する。これは、HTTP による時刻表現の最小単位であり、表現できる限界だからである。

本システムにおけるコンテンツの分配や公開は、リバースプロキシとの通信に依存する。コンテンツ公開指定段階では、クラスタ内の全リバースプロキシがコンテンツの格納を完了したことを確認して公開時刻を通知するので、クラスタ内のリバースプロキシの障害に対応しなければならない。そこで通信に失敗したリバースプロキシを公開のための通信の対象から除く機能を加えた。

4.2 リバースプロキシ

今回は WWW サーバとして動作する chamomile [5] にリバースプロキシの機能を新たに実装した。クライアントからのリクエストに応答する一方、コンテンツ公開制御サーバからのリクエストに従い動作する。ヘッダに X-Notify が記されているリクエストをコンテンツ公開制御サーバから受けとると、オリジンサーバから一時コンテンツを取得し、正常に取得できたかどうかを示す応答をコンテンツ公開制御サーバに返す。ヘッダに X-Public を含むリクエストを受けとると、記されている時刻を記憶しておき、その時刻以降、新しいコンテンツをクライアントに応答する。またその時、ファイル名を一時ファイル名から公開するべきファイル名に置き換える。

更新したコンテンツが指定された時刻以降に公開されたかどうかを確認するため、ホストの OS や時刻に依存せずコンテンツの中身にのみ依存する一意に識別可能な番号を ETag として用いた。これは、コンテンツの中身からハッシュを用いて値を算出している。ETag をアクセスログに残し、公開されているコンテンツを把握した。

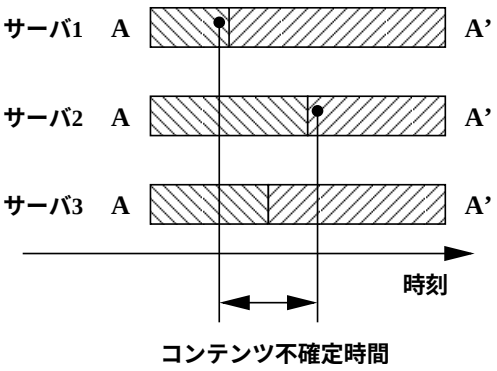

図 5: コンテンツ不確定時間

5 評価

ここでは、提案方式の評価指標について述べ、実験を通して評価を行なう。

5.1 同期の評価指標

コンテンツ不確定時間

更新したコンテンツがクラスタ内で公開される時刻を調べることにより、クラスタ内でコンテンツの一貫性が保持できているかを評価する。公開した時刻に差が存在すれば、この時間帯は、クラスタ内でコンテンツが一致せず、一貫性が損なわれている時間帯である。この時間帯をコンテンツ不確定時間と呼ぶ。サーバが更新処理中にクライアントがアクセスし、その時のサーバのログを調べることでクライアントに返すコンテンツがわかる。

図 5 に示すように複数のサーバにおいてコンテンツ A を A' に切り替える更新処理中にクライアントにどのコンテンツが提供されるのかを各サーバのログにより調べ、コンテンツ不確定時間を算出する。

同期成功率

提案方式に対して、同期に成功したかどうかを調べる指標として同期成功率を導入する。あるコンテンツについて、コンテンツ公開制御サーバが指定したコンテンツが公開時刻以降、全リバースプロキシで公開されているかどうかを調べる。同一であれば成功と判定する。

$$\text{同期成功率} = \frac{\text{同期成功回数}}{\text{同期成功回数} + \text{同期失敗回数}}$$

ここで、各リバースプロキシがコンテンツ公開制御サーバからコンテンツを取得するのに失敗した時、公開時刻以降にコンテンツ公開制御サーバが指定したコンテンツを公開していない時、公開しているコンテンツがリバースプロキシ間で同じでない時すべてを同期失敗とみなす。

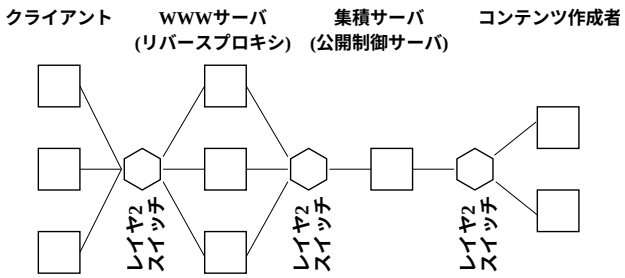

図 6: 実験 1 の構成図

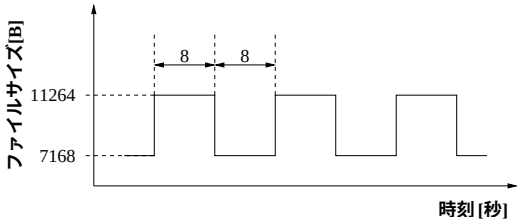

図 7: ファイルサイズと転送間隔

5.2 実験 1: 他方式による分配との比較

ここではコンテンツの更新を実現する手法として、典型的な rsync や NFS を用いる方式と提案方式の比較を行なう。

実験 1 の概要

図 6 に実験 1 の構成図を示す。更新したコンテンツをコンテンツ作成者側から 3 方式を用いて 3 台のサーバに分配し、クライアントからのリクエストに対して複数のサーバ間で同じコンテンツが提供できているかをサーバのログにより確認する。rsync および NFS を用いた場合、WWW サーバには一般的な Apache-1.3.20 [6] を用い、提案方式を用いた場合、リバースプロキシに chamomile を用いた。オリジンサーバに Apache を用いた。Apache が生成する ETag は他ホスト間のコンテンツの比較に向いていないので、この実験では更新の確認にファイルサイズを用いる。

コンテンツ作成者側では同一名でサイズが異なる 2 つのファイルを用意する。これらを一定間隔で各サーバに対して交互に供給する。図 7 は時間

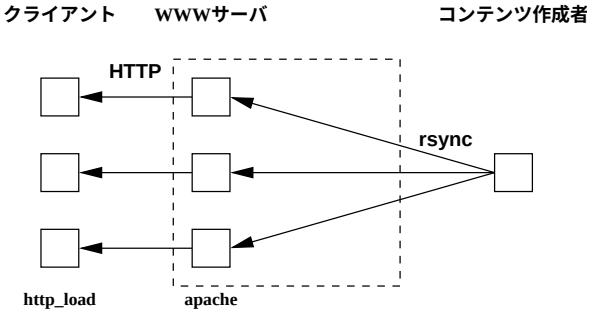

図 8: rsync によるコンテンツの更新

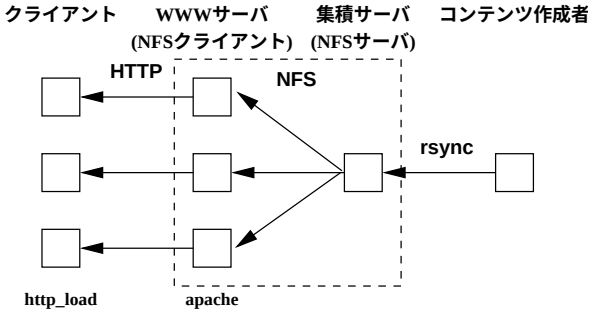

図 9: NFS によるコンテンツの更新

経過とサーバで提供されるコンテンツの関係を示したものである。このファイルに対してクライアント側からリクエストを発行する。1 秒間に発行するリクエストの件数 (リクエストレート) を変化させた。クライアントには http_load を用いた。クライアントからのリクエストにより得られるサーバのログを調べると、理想的には図 7 のように、時間経過とともに複数のサーバでコンテンツが更新されることがファイルサイズの違いによりわかるはずである。一定間隔で 2 種類のコンテンツを全てのサーバで交互に更新し、コンテンツの公開時刻を調べることによりコンテンツ不確定時間差を算出する。分配する対象のサーバの台数、ファイルサイズ、更新間隔、クライアント側からのリクエストなどの条件は同じである。このようにして、コンテンツ不確定時間差を提案方式と他方式で比較する。

rsync を用いる場合は、図 8 のようにコンテンツ作成者が、rsync を実行して直接サーバに同時に分配する。更新したコンテンツの流れは図の矢印のようになる。NFS を用いる場合、図 9 のようにコンテンツ作成者と WWW サーバの間にコンテンツ集積サーバを設ける。コンテンツ集積サーバを NFS サーバ、WWW サーバを NFS クライアントとして動作させ、コンテンツ集積サーバのファイルシステムを共有する。NFS サーバの更新を即座に反映させるため、NFS クライアントでは NFS のキャッシュ機能は無効にしてマウントした。コンテンツ作成者側から rsync で転送し、コンテンツ集積サーバのコンテンツを更新する。提案方式によるコンテンツの更新ではコンテンツ作成者が FTP によりコンテンツ公開制御サーバに対してコンテンツを転送する。

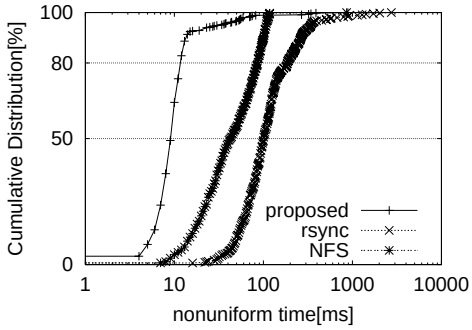

図 10: コンテンツ不確定時間

結果・評価

図 10 はクライアントから各サーバホストへのリクエストレートを毎秒 300 件 (リクエスト間隔 3.3ms) とした時の 3 方式のコンテンツ不確定時間の累積度数分布を示したものである。コンテンツ不確定時間を 80%tile 値で比較すると、rsync は 200ms, NFS は 100ms, 提案方式では 10ms となった。提案方式はコンテンツ不確定時間を最小限に抑えることに成功している。

さらに、リクエストレートを変化させた時のコンテンツ不確定時間の 80%tile 値を図 11 に示す。どのリクエストレートにおいても、提案方式ではコンテンツ不確定時間を抑制していることがわかる。図 11中の直線は、リクエストレートとリクエスト間隔の関係をプロットしたものである。この直線は検出できる不確定時間の限界値を示している。このため、それぞれの結果は検出限界値より大きくなっている。リクエストレートが小さい時は 3 方式のコンテンツ不確定時間は大きく変わらない。しかしリクエストレートが大きくなると、

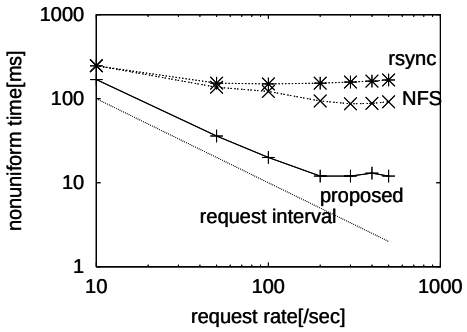

図 11: コンテンツ不確定時間の 80%tile 値とリクエストレートの関係

rsync および NFS は直線から離れ、ほぼ一定値に収束している。rsync および NFS によるコンテンツ不確定時間はそれぞれ 200ms 程度、100ms 程度であった。一方、提案方式は直線からの距離が短く、より限界に近い性能を示し、コンテンツ不確定時間は約 15ms であった。コンテンツ不確定時間は収束しており、これ以上リクエストレートを大きくしても、コンテンツ不確定時間はサーバが過負荷にならない限り変化しないと思われる。提案方式により、コンテンツ不確定時間は他方式と比べて、1/10 程度に抑制していることがわかる。

5.3 実験 2: 甲子園インターネット中継システムにおける運用

提案方式を 2001 年 8 月 8 日から 8 月 22 日に開催された高校野球 WWW 中継において運用した。当 WWW サイトは試合状況を伝えるスコアボードや選手情報などユーザにとって情報の新鮮度に対する要求が極めて高い。さらにコンテンツの更新頻度が高く、ライブ中継的要素が強い。

例年の大会期間中総アクセス数は数億件に達しており、大規模 WWW システムである。更新頻度が高くアクセスが大量であり、更新に対して特に配慮が必要である。提案方式はこのような WWW システムに向いているコンテンツ更新機構である。

システムの概要

システム全体の構成図を図 12 に、運用条件、環境を表 1、および表 2 に示す。コンテンツ公開制御サーバは 1 台、リバースプロキシの台数は 3 台である。コンテンツ公開制御サーバには dnotify、リバースプロキシには chamomile を用いた。オリジンサーバの WWW サーバには Apache-1.3.20 を用いた。クラスタ内の時刻は NTP を用いて同期した。

表 1: 運用環境

リバースプロキシ	3 台
コンテンツ公開制御サーバ	1 台
平均ファイルサイズ	2.3kB
コンテンツ最短更新間隔	5 秒

表 2: 運用した各サーバの仕様

コンテンツ公開制御サーバ	
OS	Solaris 8 (SunOS 5.8)
model	SuperMicro SuperServer6010H
CPU	Intel PentiumIII 1GHz x 2
Memory	1GB

リバースプロキシ	
OS	Solaris 7 (SunOS 5.7)
model	Sun Enterprise 250
CPU	UltraSPARC-II 400MHz x 2
Memory	2GB

結果・評価

全リバースプロキシのログを集計し、同期成功率を算出した。運用期間中の典型的な一日の 8 月 17 日と決勝戦があった 8 月 22 日の結果を表 3 に示す。22 日は 1 試合あたりのアクセス数が期間中の最大値で負荷は高かった。

コンテンツ分配段階の所要時間の分布を図 13 に示す。80%tile 値で比較すると 17 日は 40ms、22 日は 100ms となった。22 日の値が大きくなったのは 22 日の方が高負荷であったためだと思われる。

両日の同期成功率は 17 日はほぼ 100%、22 日は 100%である。提案方式は公開時刻を指定することで、クラスタ内でコンテンツ格納処理に要する時間差を吸収し、一貫性を保った。

また、サーバの負荷の状況を調べた。トラヒックの大小をリバースプロキシに対する負荷の尺度とみなせる。図 14 に 22 日のあるサーバ 1 台におけるコンテンツ分配段階の所要時間と、その時のクライアントからのリクエスト処理に伴うトラヒックの関係を示す。トラヒックが 15Mbps 程度まで

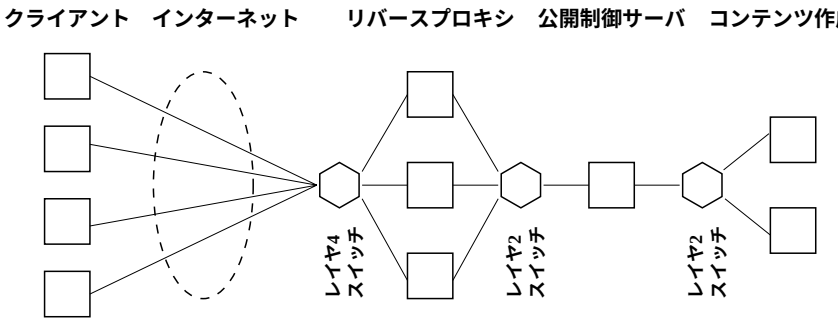

図 12: 実験 2 の構成図

表 3: 8/17, 22 の諸元

日付	8/17	8/22
アクセス数	4900 万件	2600 万件
瞬間最大〃	900 件/秒・台	1200 件/秒・台
更新回数	1 万件	1 万件
同期成功率	99.98%	100.00%

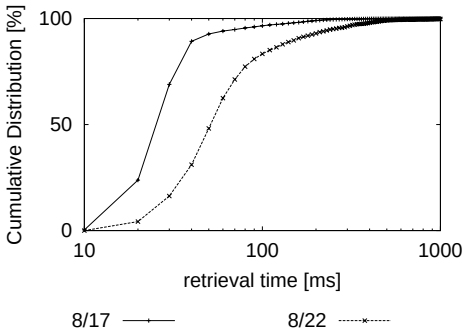

図 13: コンテンツ分配段階所要時間分布

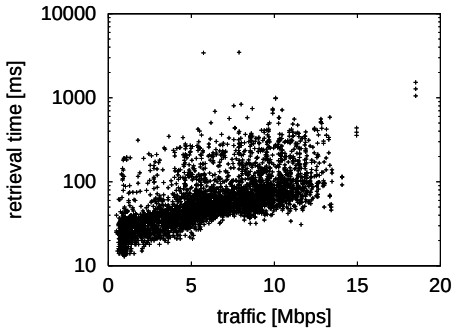

図 14: コンテンツ分配段階所要時間とトラヒックの関係

増加するとコンテンツ取得段階の時間が長くなるものの、大部分が 1 秒以内に収まった。

6　今後の課題

提案方式は、互いにネットワーク的に離れた位置にリバースプロキシを設置してもコンテンツの一貫性を保ち同時に公開することができる。これは、コンテンツの格納段階と公開段階に分離しているので、遅延によるコンテンツの転送時間の差が大きくても公開に影響しないからである。しかし、提案方式はコンテンツ制御サーバ・リバースプロキシ間の通信の失敗を補うような機能は含まれていない。このような機能を加え、信頼性を向上させると、CDN などの分散型コンテンツ配送システム等、システムの適応範囲が広がる。

提案方式では、更新したコンテンツの公開時刻の決定はコンテンツ公開制御サーバとリバースプロキシとの通信によって決定する。コンテンツ作成者がコンテンツの公開が有効になる時刻を決定することは考慮していない。コンテンツ作成者がコンテンツの公開時間を指定できる機能を付加するとより利便性が増す。

7　おわりに

本論文では、サーバクラスタ内でコンテンツを更新するときに用いられる既存の方式の問題を示した。提案方式は、コンテンツの公開を制御しコンテンツ一斉公開を実現する。

提案方式では、コンテンツの更新処理を格納段階と公開段階に分離し、全ての WWW サーバの格納処理が完了すると、公開段階に移行する。公開段階ではコンテンツ公開制御サーバが公開時刻を全 WWW サーバに通知する。これにより、クラスタ内で公開時刻を一致させる。このように格納段

階と公開段階に分離すると、更新に要する時間差を吸収することが可能となり、コンテンツ更新時のクラスタ内のコンテンツの一貫性を保持できる。

さらに他方式との比較や運用実験を行なうことにより、提案方式を評価し有効性を示した。提案方式は、コンテンツ不確定時間を大幅に軽減していることがわかった。また、高校野球の WWW 中継サイトにおいて運用を行ない、現実のシステムで高負荷な状況において、コンテンツの一貫性を保ちつつユーザにサービスを提供することができた。同期成功率を算出した結果、運用においてクラスタ内で更新時にコンテンツの一貫性を保つことができ、アクセスが大量に到来しコンテンツの更新が頻繁な WWW サイトにおいて本システムが有効であることを示した。

謝辞

甲子園の中継実験なくして、この研究はありえません。貴重な実験の機会を与えて下さった関係組織の皆様、およびアクセス頂いた視聴者の皆様に深く感謝いたします。また、本論文に対して助言を頂きました、奈良先端科学技術大学院大学情報ネットワーク講座の藤田裕介氏、藤部修平氏に感謝いたします。

参考文献

[1] "rsync", (URL:http://rsync.samba.org/)

[2] Postel, J.,andJ.Reynolds : "File Transfer Protocol(FTP)", RFC 959

[3] R.Fielding et al. : "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616

[4] R.Braden : "Requirements for Internet Hosts – Application and Support, RFC1123

[5] "chamomile", (URL:http://iplab.aist-nara.ac.jp/∼eiji-ka/chamomile/index.html)

[6] "APACHE HTTP SERVER PROJECT", (URL: http://www.apache.org)