情報処理学会論文誌 Vol. 52 No. 2 507–519 (Feb. 2011)

# 広域分散システムによるデジタルコンテンツレポジトリの実現

三島和宏[†1] 朝枝仁[†1] 村井純[†2]

本研究では，将来に向けたインターネットストリーミングサービスのさらなる展開に対応するため，膨大なデジタルコンテンツ情報を汎用的，かつ，統一的に扱う，デジタルコンテンツ情報のレポジトリの実現を提案する．コンテンツレポジトリは，広域分散システムによって構成されたストレージアプリケーション，それを管理するアクセスアプリケーション，ローカルストレージにより実現され，特定のアプリケーションやポータルサイトに依存しないコンテンツ情報の提供を可能とする．これにより，汎用的なコンテンツ情報管理基盤を実現する．提案手法の評価として，PlanetLab 上で動作するプロトタイプアプリケーションとして実装し，性能評価し，有効性を確認した．

# Digital Content Repository Organized with Distributed Systems

Kazuhiro Mishima,[†1] Hitoshi Asaeda[†1] and Jun Murai[†2]

This paper describes a distributed digital content information repository system. Digital content delivery environment has changed significantly due to the advancement of computer and network. In this situation, there are no integrated systems to search contents which have been delivered over the Internet. We proposed a distributed digital content information repository system that achieves scalability and flexibility of searching digital contents. In this paper, we evaluated the actual implementation on the PlanetLab to measure the performance of our proposed system. The experimental results have shown the effectiveness of our distributed content information repository.

## 1. まえがき

安価な高性能 PC の流通や高速インターネットの発展により，YouTube[1] や Yahoo Video[2]，ニコニコ動画[3] などのポータルサイトを経由する「サーバ型」のコンテンツ配信，PPLive[4] や KeyholeTV[5] など専用のアプリケーションを送受信者の計算機にインストールしてコンテンツデータを受信者同士が転送し合う「P2P 型」のコンテンツ配信など，様々なデジタルコンテンツの配信サービスが普及してきた．

しかし，現在のストリーミングサービスでは，デジタルコンテンツがポータルサイトや P2P アプリケーション内で管理/配信される形式であるため，インターネット上で受信可能なすべてのデジタルコンテンツの情報を包括的に管理する手段がない．つまり，コンテンツの内容やメディアフォーマット，使用帯域や転送プロトコルなどの属性を示す「コンテンツ情報」もコンテンツデータとともにポータルサイトやアプリケーション固有のものとして管理されるため，インターネット全体のコンテンツ情報を得るためには，ユーザはそれぞれのアプリケーションやポータルサイトごとにコンテンツ情報を検索しなければ情報を得ることができない．この制約は，インターネット全体でさらに増加していくであろう膨大なコンテンツ情報の管理において，送信者・受信者にかかわらずユーザの利便性を失わせる．また，現状の環境であれば，企業や個人などがコンテンツ配信を行う場合，必ず特定のポータルサイトにコンテンツを登録して配信を行うか，もしくはすべてのユーザに対してコンテンツ情報を通知しなければユーザはその存在を伝える術がない．この機能的な制約は，将来に向けたコンテンツ配信技術の発展や普及の妨げにもなる．

コンテンツ情報は，コンテンツ内容やメディアに関する情報だけでなく，開始時刻や終了時刻といった時間軸情報，システムが行うべき処理に関する情報，コンテンツ鍵など暗号化を要する情報，受信者を限定するポリシ情報といった多岐にわたる情報が格納される．このため，様々な属性を持ち，場合によっては受信されるべきユーザを限定するようなケースも考えられ，コンテンツ情報の持つ属性を考慮した管理が必要となる．

本研究では，デジタルコンテンツ情報を汎用的，かつ，統一的に扱うために，特定のアプリケーションやポータルサイトに依存しないデジタルコンテンツ情報のレポジトリ（以下，

†1 慶應義塾大学大学院政策・メディア研究科
Graduate School of Media and Governance, Keio University
†2 慶應義塾大学環境情報学部
Faculty of Environment and Information Studies, Keio University

“コンテンツレポジトリ”）を提案する．コンテンツレポジトリは，標準メタデータ形式で表現されたデジタルコンテンツ情報をインターネット上に配置された複数台のサーバに登録・管理する広域分散システムとして実現され，ユーザはこの分散システムを 1 つの大規模データベースのようにアクセスし必要なコンテンツ情報を取得する．コンテンツレポジトリは，コンテンツ情報のみを管理するレポジトリであるため，膨大な数のコンテンツ情報を一元管理できるだけでなく，インターネット基盤の特徴である自律的かつ非集中的なネットワーク構成により，機能拡張性と柔軟性を両立させた高性能な分散システムを実現する．コンテンツレポジトリが普及した場合，ポータルサイトやアプリケーションがコンテンツレポジトリを参照してコンテンツ配信することが可能となり，ユーザにとっての利便性が向上するだけでなく，新しいストリーミングサービスの展開が期待できる．

以下では，2 章でデジタルコンテンツ情報の検索/通知手法の概要と既存手法における問題点ならびにコンテンツレポジトリに対する技術要求について述べ，3 章で本研究で提案するコンテンツレポジトリの設計，4 章でコンテンツレポジトリのプロトタイプ実装ならびにその評価，5 章で関連研究についてそれぞれ述べる．最後に 6 章で本研究のまとめと今後の課題について述べる．

## 2. コンテンツ情報の管理

### 2.1 現 状 分 析

ユーザが受信可能なすべてのコンテンツ情報を取得する方法として，コンテンツ送信者がユーザに対してコンテンツ情報を通知する方法と，ユーザが自らコンテンツ情報を検索する方法がある．

前者においては，マルチキャスト環境下における標準的なコンテンツ情報の通知手法として，Session Announcement Protocol（SAP）[6] が広く利用されている．SAP は，Session Description Protocol（SDP）[7] で記述されたコンテンツ情報を定期的に所定のマルチキャストグループに対して送信することで，そのグループに参加したユーザがコンテンツ情報を取得できる仕組みを提供する．しかし，SAP にはネットワーク全体に定期的にメッセージを送り続けるという特性から，そのスケーラビリティの低さ，コンテンツ数に比例して配信にかかる時間が非常に長くなるなどの問題点，またプロトコル的な脆弱性が指摘されている[8),9)]．

また，広く普及しているメールや RSS を用いてコンテンツ送信者や代理サイトがコンテンツ情報をユーザに通知する手法も考えられるが，これらはユーザがあらかじめコンテンツ情報の通知依頼（Subscribe）をしておく必要があり，ユーザが認識していない送信者やサイトからの情報を得ることができず，送信者が多岐にわたる場合にユーザがコンテンツ情報をくまなく発見することは困難である．

後者のコンテンツ情報検索手法に関しては，Google[10)] などのサーチエンジンによるコンテンツ情報検索が想定できる．サーチエンジンでは，前者の通知型手法とは異なり，ユーザが能動的にコンテンツ情報を探索するモデルである．サーチエンジンは，多種多様な情報の検索が可能である一方，クローラと呼ばれる機能による情報収集完了後に初めてユーザが検索を行えるようになる[11)]．しかしクローラによる情報収集は時間を要するため，クローラの情報収集から更新するまでの時間とコンテンツ配信時間にはタイムラグが生じ，結果として，コンテンツ配信が開始されているのにコンテンツ情報は発見/検索できない，あるいは，コンテンツ配信は終わっているのにサーチエンジンからはコンテンツ情報が見えてしまう，などの不整合が生じてしまう．これは，特にライブなどのリアルタイムで配信されるコンテンツ情報の検索に関して顕著な問題となる．

既存のコンテンツ情報管理手法にはいずれにおいても課題が残されており，有効な手法がない．また，コンテンツ情報の持つ属性を効果的に取り扱うことのできる手法も存在しない．以上の分析より，前者のコンテンツ情報通知手法と後者のコンテンツ情報検索手法の利点をあわせ持ち，かつ欠点を補い合うコンテンツレポジトリの実現が必要不可欠である．

### 2.2 システムの技術的な要件

上記分析に基づき，本研究で着目するコンテンツレポジトリシステムの持つべき技術的な要件は以下のとおりである．

#### 2.2.1 要件 1：大量なコンテンツ情報の管理

インターネット全体に存在するコンテンツ情報の管理を想定したコンテンツレポジトリでは，数億エントリ規模の大量のデータ処理を想定する必要がある．このため，システム全体としての規模を考慮し，多くのノードが自律分散協調的に動作することで機能拡張性を高める必要がある．

#### 2.2.2 要件 2：コンテンツ情報の迅速な登録と取得

コンテンツ送信者が迅速にコンテンツ情報の登録を行えなければならない．また，コンテンツ情報の更新や削除が必要になった場合も，それを迅速に行えるシステムでなければならない．また，ユーザは登録されたコンテンツ情報を迅速に取得/検索できるシステムでなければならない．

#### 2.2.3　要件 3：多属性なコンテンツ情報に対する考慮

コンテンツ情報には，コンテンツ名や送信者情報などのコンテンツが必ず持つ情報と，コンテンツの形式や認証に関わる情報などの任意の情報があり，様々な種類の情報（属性）が含まれる．これらの情報を考慮した情報管理が必要である．また，将来においても属性の変化などに対応するため，データ構造の変化に対しても拡張性の高いシステムである必要がある．

#### 2.2.4　要件 4：コンテンツ情報のスコープに対する考慮

コンテンツ情報には，そのコンテンツの配信範囲（スコープ）を定義して，スコープ内にあるユーザだけがコンテンツの受信が可能となる必要がある．スコープには，そのサイトローカルなユーザを対象としたもの（ローカルコンテンツ情報）と，それ以外のすべてのユーザを対象としたもの（グローバルコンテンツ情報）があり，これらのスコープを考慮したコンテンツ情報の提供が必要である．

#### 2.2.5　要件 5：コンテンツ情報に対するアクセスコントロール

コンテンツ情報には，受信ユーザを限定して情報提供したいものが存在する．そこで，コンテンツ情報のスコープ管理だけでなく，コンテンツ情報に対するアクセスコントロールによって，コンテンツ情報を提供するユーザを限定できる手法が必要である．

#### 2.2.6　要件 6：コンテンツ情報の保護

アクセスコントロールされるコンテンツ情報は，提供したいユーザを限定するものであるため，想定しないユーザによる参照の防止という観点からコンテンツ情報の保護が行われる必要がある．

## 3. コンテンツレポジトリシステム

本章では，前章までに述べたコンテンツ情報管理の現状分析ならびに技術要件をふまえ，本研究で提案するコンテンツレポジトリの各機能について検討し，その設計・機能について述べる．

### 3.1　システム概要

本研究で提案するコンテンツレポジトリシステムは，インターネット上に配置された複数のサーバから構成される広域分散システム上に構築され，自律分散協調型のシステムとなる．このシステムにより，送信者がコンテンツ情報の通知（登録）をコンテンツレポジトリに対して行い，それをユーザは大規模データベースのようにアクセスし，最新のコンテンツ情報を取得することが可能となる．

2.2 節にあげた要件により，本研究で提案するシステムは，非常に大規模な情報の管理（**要件 1**）と情報提供の迅速さ（**要件 2**）を両立させる仕組みが必要となる．本システムを構成するデータストレージの選択にあたって，リレーショナルデータベース（RDB）に関しては，その規模性やパフォーマンスに対する懸念[12)–14)] から，RDB 単体ではこれらの要件を満たすことができない．また，通常の分散ハッシュテーブル（DHT）では，**要件 4** にあるスコープの概念は満たせず，また**要件 5** のアクセスコントロール要件に関しても特別な機能を要求する．そこで，本研究では，1) 大規模な情報管理と迅速な情報提供のために，複数のノードから構成される構造化オーバレイネットワークを用いた “グローバルストレージ”，2) コンテンツ情報を提供するスコープを考慮するために，RDB を用いた “ローカルストレージ” の 2 種類のデータストレージを併用したコンテンツレポジトリシステムを提案する．

コンテンツレポジトリシステムでは，グローバルコンテンツ情報の管理にグローバルストレージを，ローカルコンテンツ情報の管理にローカルストレージを用いる．ローカルコンテンツ情報はその DB が存在するノード以外からは参照ができない構造をとり，ローカルコンテンツを取得できるユーザだけにそのノードへのアクセス権を与えることで，**要件 4** にあるスコープ管理を実現できる．また，情報の参照がノード内で完結できるため，ローカルコンテンツ情報の参照にかかる処理を迅速に行うことが可能となる．

また，ローカルストレージにはサイトローカルな管理情報，つまりグローバルストレージを構成する近隣接続ノード情報，自ノードからコンテンツレポジトリシステムに登録されたコンテンツ情報の複製が含まれる．レポジトリシステムに対して登録されたコンテンツ情報の複製は，グローバルコンテンツ，ローカルコンテンツにかかわらず管理情報として格納され，情報の変更や削除（3.3.4 項），TTL によるデータ管理（3.4 節）といった処理が行われる際に参照される．

また，分散ハッシュテーブルは大規模かつ迅速なデータ処理が可能であるが，基本機能ではハッシュ関数による ID 空間全体がフラットかつランダムな情報空間として実現されるため，格納データに対する識別を行わないという問題がある．文献 15) が示すとおり，**要件 5** を満たすためには，分散ハッシュテーブル上に登録されるコンテンツ情報に対する特別なアクセスコントロール手法が必要となる．そこで，3.6 節に示す情報共有ラベルによるアクセスコントロール機能を提供する．

アクセスコントロールに加え，**要件 6** で示したとおり，コンテンツ情報に対する保護も必要である．そこで，3.7 節に示す閾値暗号を用いた情報分散による情報保護機能を提供する．

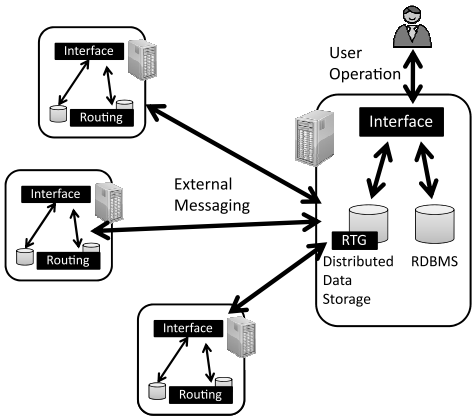


**図 1**　ノード間連携の概要
Fig. 1　Overview of node relationship.

本システムのコンポーネント構成や具体的な動作は，3.3 節に示す．

### 3.2　コンテンツレポジトリシステムのコンポーネント

本システムは，複数ノードの協調動作により成り立つ．**図 1** にノード間の連携について示す．また，本システムは，**図 2** に示す複数のコンポーネントから構成される．

本研究で提案するシステムは，1) グローバルストレージに格納される情報を保持する Storage Layer，2) Storage Layer から情報を検索する Routing Layer，3) Routing Layer だけでは実現できない機能を実現する Processing Layer，4) グローバルストレージに格納した情報の管理を行う Local Management Layer，5) Routing Layer に対して検索クエリを発行する Access Layer，6) ユーザからの要求を処理する Interface Layer の複数レイヤ，ならびにローカルストレージとしての RDB によって構成される．処理すべき機能を各レイヤで分割し，それらが連携動作する．すべてのコンポーネントは，同一ノード上で動作する．本システムは，機能をレイヤ分割した実装となっている．これにより，各レイヤには任意のアルゴリズムを導入することが可能となり，本システムに対して容易に拡張を加えることができる．各機能を区分化し，レイヤとしてモジュール化することにより，同様の機能を持つ他のアルゴリズムなどを既存のアルゴリズムの代替機能として容易に置き換えることが可能となる．さらに，ネットワークを通じて複数のサーバが連携し，各ノード上で動作するコンポーネントが協調して大規模なリポジトリシステムを構成する．コンテンツレポジトリを利用するユーザは，各ノードの Intarface Layer を通じて本システムにアクセスする．

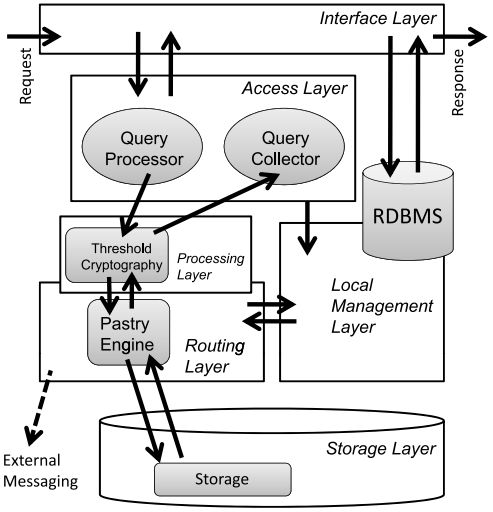


**図 2**　コンテンツレポジトリシステムのコンポーネント概要
Fig. 2　Component overview of content repository system.

#### 3.2.1　Access Layer

Access Layer では，ユーザからのリクエストの受け付け，ユーザから受け付けたクエリ解析処理，下位の Routing，Local Management の各 Layer の管理を行う．ユーザからのリクエストは，SQL 文に準拠した形式のデータを SOAP を用いた通信で受け付ける．SOAP や SQL 文といった標準的なプロトコルを用いることで，Web インタフェースなど（Interface layer）を別途実装することができ，ユーザの操作性を向上させることができる．

Access Layer には，2 つの処理モジュール（Query Processor，Query Collector）が内包されている．

**Query Processor**：Access Layer に含まれる Query Processor は，ユーザから受け付けたリクエストを解析し，Routing Layer に対して必要な処理を実行する．実際の処理については，後述する．

**Query Collector**：Query Collector は，Query Processor で実行された処理結果を解析し，Access Layer に結果の返答を行う．Routing Layer で実行したクエリを，ユーザから提示されたリクエストに基づく結合処理を Query Collector で実行する．

#### 3.2.2 Routing Layer

Routing Layer は，グローバルストレージの実現のために，ルーチングアルゴリズムをルーチングエンジンとして実装し，そのアルゴリズムに基づくデータ検索機能を提供する．Routing Layer では，ルーチングエンジンをモジュール化しており，検索時の用途にあわせ，いずれのアルゴリズムを実装・使用することができ，さらに必要に応じてその機能を組み合わせて利用することも可能である．ここでは，1) ルーチングアルゴリズムの持つ検索性能（Pastry では標準アルゴリズムで N ノードに対して $O(\log N)$ の検索性能を持つ）を最大限利用すること，2) プロトタイプ実装の容易さ，の観点から，分散ハッシュテーブルのアルゴリズムの 1 つである Pastry[16] の基本アルゴリズムを Pastry Engine として実装している．Routing Layer のルーチングアルゴリズムを用いることで**要件 1** ならびに**要件 2** を満たす大規模なデータストレージを構築することができる．

今回実装している Pastry を代表とする分散ハッシュテーブルは，ある Key とその Key に対する Value を格納することを想定した機構であり，かつ Key の検索においては，完全一致検索に制約されるという課題があり**要件 3** を満たすことができない．そこで，Access Layer にキーワード分割による部分一致検索機能を実装し，Routing Layer と連携することで検索機能を提供する．本システムでは，プロトタイプの実装の容易さを考慮し，分散ハッシュテーブルに対する様々な部分一致検索機能の中でも，基本的な機能であるキーワード分割による検索機能を実装している．これ以外の検索機能の検討ならびに本システムへの実装については今後の課題（後述）としてあげられる．図 1 ならびに図 2 中の “External Messaging” は，Pastry による対外ノードとの通信を示すものである．

#### 3.2.3 Processing Layer

Processing Layer では，Routing Layer に実装される Routing Engine だけでは実現できない機能を上位レイヤとして介在することで実装するためのレイヤである．本システムでは，**要件 5** や**要件 6** でアクセスコントロールや情報の保護の機能を要求事項としてあげている．これらの機能は，既存の分散ハッシュテーブルの基本アルゴリズムでは，解決することができない．そこで，Processing Layer に必要な機能を実装することで，Routing Layer 以下の機能を変更することなくこのような機能を実現できる．本システムでは，3.6 節や 3.7 節に示す機能を，このレイヤに実装している．

#### 3.2.4 Local Management Layer

Local Management Layer では，自ノードのストレージ（Storage Layer）に対して登録されたデータの管理を行う．データ管理にあたっては，登録管理情報を RDB に保持し，データの変更や削除の際に効率化を図る．また，自ノードに対して登録されたデータと他ノードから分散配置されたデータの判別し，誤ったデータ削除を行わないように，データ削除を行う際はこの Layer で管理される登録データ一覧を用いる．

#### 3.2.5 Storage Layer

Storage Layer では，Routing Layer におけるグローバルストレージに関わるデータ（本システムでは Pastry Engine によって自ノードにて登録されたデータ，ならびに他ノードから登録されたデータ）が保存される．すべてのデータは，Routing Layer を通じて登録・変更・削除処理が行われ，透過的に 1 つのストレージ内で管理される．

### 3.3 動作概要

本節では，本研究で提案するコンテンツレポジトリの動作概要について示す．

#### 3.3.1 データの例

本論文では，データの登録・更新・検索・削除に関わる処理の説明のため，明示的に以下のリクエストを処理するものとする．本システムでのコンテンツ情報の記述には，標準化された書式である Session Description Protocol（SDP）[7] を用いることを想定している．

下記に，例として「SDP Seminar」というコンテンツの登録・検索・削除に関する SQL 文を記述する．この例に利用されるデータのアトリビュートは，s（Session Name）：コンテンツのタイトルを示す文字列，i（Session Information）：コンテンツの詳細情報を示す文字列，c（Connection Data）：接続先を示す文字列，start_t（Start Timing）：開始時刻を示す整数列，end_t（End Timing）：終了時刻を示す整数列である．

```
INSERT INTO storage (s, i, c, start_t, end_t) VALUES ('SDP Seminar', 'A Seminar on the session description protocol', 'IN IP4 224.2.17.12/127', '2873397496', '2873404696');
SELECT c WHERE s = 'SDP Seminar' AND i = 'session';
```

#### 3.3.2 データ登録処理

レポジトリに対するデータ登録は，以下のように行われる．

(1) Access Layer でクエリを受け付ける．ローカルコンテンツ情報はローカルストレージに情報を格納し，グローバルコンテンツ情報はグローバルストレージへ情報を格納する（以降は，グローバルコンテンツ情報の登録に関するフローである）．

(2) Query Processor は，主キーの登録を Pastry Engine に対して行う．この際，登録に利用される検索キーは，“アトリビュート：そのアトリビュートの値” のハッシュ値，登録データは，登録される値（INSERT 句の VALUES の値）をそれぞれ利用する．

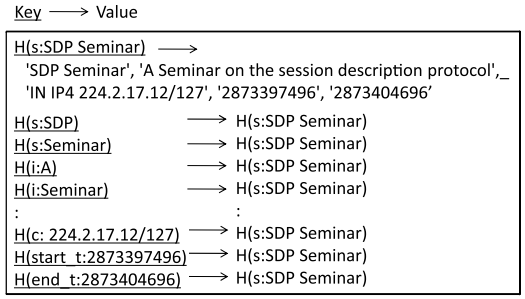


**図 3**　登録されるデータ
Fig. 3　Registration information examples.

（3）　Query Processor は与えられたクエリを解析する．Pastry Engine へ送られるデータはスペースごとに文字列を分割し，“アトリビュート：分割された文字列” を検索キー，主キーのハッシュ値（主キーへのポインタ情報）を登録データとして登録処理を実行する．登録の際に重複するデータは 1 つにまとめて登録処理を行う．

（4）　削除や管理のために主キーのハッシュ値を検索キー，登録したデータのすべての検索キーをカンマで区切りで示したものを登録データとして Local Management Layer に登録する．

3.3.1 項にあるデータを Query Processor で登録する場合のデータ一覧を**図 3** に示す．

#### 3.3.3　データ検索処理

レポジトリに対するデータ検索は，以下のように行われる．

（1）　Access Layer でクエリを受け付ける．ローカルコンテンツ情報はローカルストレージから情報を検索し，グローバルコンテンツ情報はグローバルストレージから情報を検索する（以降は，グローバルコンテンツ情報の検索に関するフローである）．

（2）　Query Processor は，主キー “アトリビュート：そのアトリビュートの値” のハッシュ値を検索キーとして検索する．Pastry Engine へ送られるデータはスペースごとに文字列を分割し，“アトリビュート：分割された文字列” を検索キーとして検索する．

（3）　見つかったすべてのデータを Query Collector で集約し，データが含まれているものはそのまま，元データへのハッシュ値が含まれているものは再度 Pastry Engine で検索を行い実際のデータを得る．重複するデータの除外と結合処理を行った後，Access Layer を通じてユーザに検索結果を返答する．

#### 3.3.4　データ削除処理

レポジトリに対するデータ削除は，以下のように行われる．

（1）　Access Layer でクエリを受け付ける．

（2）　ローカルコンテンツ情報はローカルストレージの情報を削除する（以降は，グローバルコンテンツ情報の削除に関するフローである）．

（3）　Local Management Layer にデータ送信する．

（4）　Local Management Layer に登録された管理用データを検索し，削除対象の検索キー一覧を取得する．

（5）　得られた検索キーに対して削除処理を実行する．

### 3.4　TTL によるデータの管理

本システムのグローバルストレージに登録されるデータは，TTL によるデータ管理が行われる．これは，不必要なデータがストレージ上に残り続けることを防ぐためであり，特定の処理により TTL の延長を行うことができる．TTL によるデータ管理機能は，Storage Layer に実装される．TTL の値は設定により変更可能である．デフォルトの値は 86,400 秒（1 日）に設定され，データの登録時に TTL を指定した場合は，その TTL 値となる．

TTL 値を延長する処理には，2 種類ある．(1) 登録処理の再実行：同一の検索キー，登録データで再度登録処理が実行された場合，TTL が延長される．(2) 検索の実行：1 度検索が行われた登録データは，自動的に TTL が延長される．これにより，ユーザに参照されているデータは自動的に削除されにくくなる．

TTL を超えたデータは，Storage Layer の機能により自動的に削除される．削除を行うデータは，自ノードのストレージに含まれるもののみで，隣接ノードや他のノードに含まれるデータに対して削除処理は行わない．TTL によるデータの消失を防ぐため，Local Management Layer では，自ノードに登録されたデータを定期的に再登録（更新）することで，登録されたデータの TTL を自動延長を行う．

### 3.5　検索パフォーマンス向上

本研究では，グローバルストレージにおいて，検索の柔軟性を実現するためにキーワード分割によるデータ検索を行う．キーワード分割を行うことで，検索性能は向上するものの，登録されるデータ数は増加する．このため，データ検索時に多くのクエリを発生させる結果となる．そこで，本研究では，Access Layer ならびに Storage Layer において，パフォーマンス向上を目的としたデータ処理を行うことで，これらの問題の改善を図る．

### 3.5.1 Dynamic Indexing

Access Layer では，ユーザから受け付けたリクエストと実際に得られた検索結果をもとに元データに対するポインタ情報として新たなデータ（Index）を Routing Layer を通じてストレージに対して登録することで検索時のパフォーマンスを向上させる．これを Dynamic Indexing と呼ぶ．

**図 4** に Dynamic Indexing の処理を示す．図中の *UserA* は先にデータ検索を行うユーザ，*UserB* は後にデータ検索を行うユーザを示す．*UserB* は，同一ノード以外から検索を行う場合も同様の流れとなる．

(1)　Access Layer でユーザのクエリを受け付ける．
(2)　クエリに基づきデータの検索を行う．
(3)　得られた結果とユーザのクエリをもとに新たなインデックスを Routing Layer を通じて作成する（この際に登録されるインデックスの持つ登録データには，得られた結果の検索キーを格納する）．
(4)　Routing Layer は，Storage Layer に新たなインデックスを格納する．

これにより，すでに行われた検索キーワードの組合せが後ほどユーザからリクエストされた場合（*UserB* による検索の場合），そのデータへのポインタ（元のデータに対する参照リンク）を直接取得できるようになり，検索クエリ数や検索処理にかかる時間，検索にかかるトラフィックを削減することが可能となる．

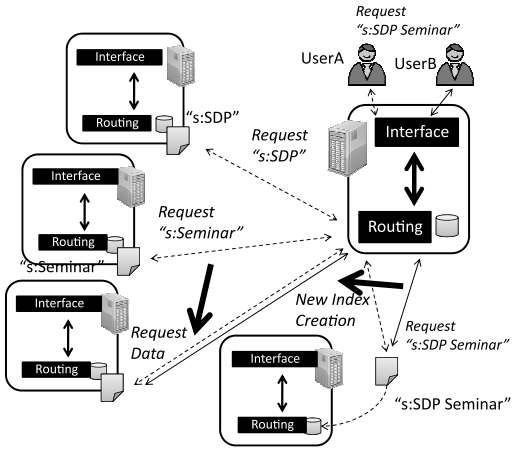


**図 4**　Dynamic Indexing によるデータ処理
Fig. 4　Data flow of Dynamic Indexing.

### 3.5.2 ローカルキャッシュ

Storage Layer におけるパフォーマンス向上処理として，ローカルキャッシュを行う．ローカルキャッシュは，1 度行われた検索結果を含むデータをローカルノードの Storage Layer に対してコピーを行う手法である．ローカルキャッシュで新たに登録されるデータは，参照情報として TTL は短く設定される．

ローカルキャッシュによりコピーされたデータは，どのノードのストレージにあったデータであっても，ローカルノードに保存されるため，次に同様の検索が行われる際には，自ノードが保持するデータとして認識される．これにより，データの取得は，ローカルノードのみで完結し，結果として検索処理にかかる時間や検索にかかるトラフィックを削減することが可能となる．

また，検索されやすいキーワードの場合，何度も検索が行われる．TTL は，検索が行われるたびに延長されるため，ローカルキャッシュで保存されるデータについてもよく検索される結果については，より長く保持され，検索の効率を高めることができる．

### 3.6 情報共有ラベルを用いたアクセスコントロール

3.1 節に示したように，グローバルストレージには，**要件 5** を満たすために，グローバルストレージに登録されるコンテンツ情報に対するアクセスコントロール手法が必要となる．そこで，情報格納時に利用する Key を SHA-1 でハッシュ化する処理を応用し，情報共有タグ *label* を用いたアクセスコントロール手法を実現する．Key は登録時に SHA-1 でハッシュ化されるため，Key 内部に情報共有タグを含めることで，タグを知りえないユーザは Key を生成できなくなる．これにより，事前に共有された情報共有タグを知りえたユーザの間のみで情報の交換が行えるようになる．情報の登録にあたって用いられる Key は，式 (1) に示すとおり，その分割データにアクセスする際に利用する情報共有タグ *label* を付与して生成する．

### 3.7 閾値暗号を用いた情報の保護

**要件 6** から，アクセスコントロールが行われるコンテンツ情報に対する情報保護が必要である．本システムでの情報の保護には，閾値暗号[17] を用いたデータ暗号化を行う．

閾値暗号とは，秘密分散法の 1 つであり，保護の対象となるあるオリジナルデータ $S$ を $n$ 個の分割データ（share）に分割し，それらのデータの中から $t$ 個の分割データを収集することでオリジナルデータ $S$ を復元する手法である．この手法では，分割データ 1 つ 1 つは

意味を持たず，$t-1$ 個の分割データを収集しただけではオリジナルデータ $S$ をいっさい復元できないという特徴がある．また，オリジナルデータ $S$ を復元する場合，$t$ 個の分割データを収集すればよく，$n$ 個のうち，$n-t$ 個までの分割データであれば紛失しても復元が可能である．

閾値暗号は，1) 暗号方式としての安全性：証明可能安全性を持ち，一般的な情報の暗号化アルゴリズムと比較し，強力な暗号をデータ分割によってより低コストで実現できる，2) データ損失に対する強度：グローバルストレージ網に障害があっても必要な数の一部分割データを収集すれば情報を復元できる，といった 2 つの観点から，自律分散協調環境を想定した本システムで用いる情報保護手法として適すると考えられる．

また，単純にデータを暗号化する手法では，グローバルストレージに登録された情報そのものが破損すると元データの復元ができなくなる．レプリケーションなどを利用する場合でも，破損したデータが網内に存在しうる限り，正常データには到達できない．このため，本システムでは，データの単純な暗号化ではなく，閾値暗号を用いる．

分割データの登録にあたって用いられる Key は，式 (1) に示すとおり，オリジナルキー *org_key* とカウンタ値を示す情報 *counter* を付与して生成する．

$$Key = H(org\_key \,||\, label \,||\, counter) \tag{1}$$

## 4. 性能評価

本研究では，提案した手法の有効性を確認するために，プロトタイプアプリケーションを実装し，評価を行った．本章では，プロトタイプ実装の詳細について述べるとともに，その評価結果について述べる．

### 4.1 プロトタイプ実装

本研究の評価のために，3.2 節に示したコンテンツレポジトリを実現する各コンポーネントをプロトタイプとして実装した．プロトタイプ実装では，Interface Layer と Access Layer と Local Management Layer を実現するアプリケーション（アクセスアプリケーション），Routing Layer と Storage Layer を実現するアプリケーション（ストレージアプリケーション）をそれぞれ異なるコンポーネントとして実装し，相互の通信に SOAP を用いることで，自律分散協調的に相互連携する．

アクセスアプリケーションの実装は，Perl 言語で行い，コンポーネント間通信のために SOAP::Lite，ユーザからのクエリ解析のために SQL::Statement をライブラリとして用いた．ストレージアプリケーションの実装は，C 言語で行い，コンポーネント通信のために libcsoap をライブラリとして用いた．文献 18) にあるとおり，Garbage Collection によるオーバヘッドが Memory Allocation によるオーバヘッドより 5 倍のメモリを要することから，ストレージアプリケーションの実装は，既存のアプリケーションやプロトタイプ実装に用いられやすい Java 言語をあえて利用せず，C 言語での実装を行っている．

また，本システムでは，Routing Layer から Storage Layer に対してデータを格納したり，取得したりする処理を複数回実行する．これを逐次実行すると処理終了までに非常に時間を要する結果となる．そこで，プロトタイプ実装では，並列処理を行える処理では，スレッドによる並列処理を採用し，処理時間の高速化を図っている．

### 4.2 プロトタイプ評価

本論文では，提案するシステムの性能を評価するために，データ処理に関わる処理時間を測定した．性能測定には，PlanetLab[19] を用い，プロトタイプ実装を実際に動作させることで行った．PlanetLab を用いることにより，(1) 地理的な分散環境での評価，(2) 実ノードでの実装評価を両立させる評価を実施できる[⋆1]．

次節よりあげる評価は，PlanetLab 上のノードにストレージアプリケーションを展開し，基盤となる Routing Layer と Storage Layer を構成し行った．評価に用いた世界に分散する PlanetLab ノードのうち比較的安定に動作する 80 ノード（アメリカ，ヨーロッパ，日本，アジアのノードから選定）である．そのうえで，日本国内の PlanetLab ノードのうちの 1 ノードにその他の Layer を含むすべての機能を展開し，評価に関わる処理はそのノードより実行させた．本評価では，Interface Layer から Access Layer に対して，3.3.1 項に示した SQL 文のクエリが実行されたものとする．グローバルストレージへ格納されるデータは以下のとおりである．

> (Key) コンテンツ情報の記述に基づくコンテンツ ID（150 ビットの SHA1 ハッシュ値）
> (Value) s:'SDP Seminar', i:'A Seminar on the session description protocol', c:'IN IP4 224.2.17.12/127', start_t:'2873397496', end_t:'2873404696'

#### 4.2.1 各コンポーネントでの処理時間

本評価では，各コンポーネントでの処理にかかる時間を PlanetLab 上に展開したプロトタイプ実装を用いて測定した．**表 1** にその結果をまとめた．示した表の時間は，(1) Interface/Access Layer，(2) Routing/Storage Layer について，登録処理と検索処理における

⋆1 PlanetLab を用いた評価では，実環境での評価ができる一方，ノードごとの Load Average やメモリ使用量などが大きく異なることがあるため，評価値に変動が生じる可能性がある．

**表 1**　各コンポーネントでの処理時間（秒）
Table 1　Processing time in each components (sec.).

| コンポーネント | 登録 | 検索 |
|---|---|---|
| Int./Acc. Layer | 0.08 | 0.17 |
| Rtg./Str. Layer | 1.09 | 0.90 |
| 計 | 1.17 | 1.07 |

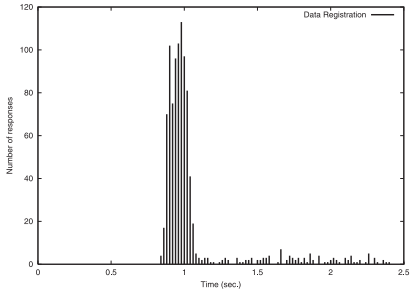


**図 5**　登録処理にかかる処理時間
Fig. 5　Processing time of registration.

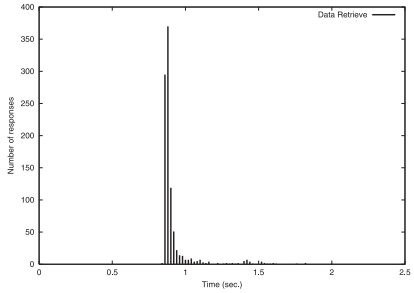


**図 6**　検索処理にかかる処理時間
Fig. 6　Processing time of retrieve.

(a) クエリ処理にかかる時間，(b) グローバルストレージへのアクセスにかかる時間である．(a) については単一ノード内での処理時間となるため非常に短時間で処理が終了している．

#### 4.2.2　データの登録・検索に関わる処理時間

本評価では，データの登録・検索に要する処理時間を計測した．計測は，(1) クエリ文の提供から登録終了までに要する時間，(2) クエリ文の提供から検索結果の取得までに要する時間をそれぞれ求めた．本評価は，それぞれを 1,000 回試行し，その結果を調査した．

**図 5** に，登録処理を行った場合の処理時間の度数分布を示す．登録処理にかかる時間の平均値は 1.17 秒，最小値は 0.86 秒であった．**図 6** に，検索処理を行った場合の処理時間の度数分布を示す．検索処理にかかる時間の平均値は 1.07 秒，最小値は 0.86 秒であった．

いずれの処理も，分散ノードに対するデータの処理（格納，取得）が主要な処理である．登録処理では，同時に 11 のクエリが，検索処理では同時に 3 のクエリが処理されたものである．

#### 4.2.3　データ数の変化による登録・検索処理時間

本評価では，グローバルストレージに含まれるデータ数の変化によるデータの登録・検索に要する処理時間の変化について調査した．計測は，特定の登録件数において，(1) クエリ文の提供から登録を実行しそれに要する時間，(2) クエリ文の提供から検索を実行しそれに要する時間をそれぞれ計測した．本評価では，それぞれをクエリの試行を 100 回行い，その結果を調査した．

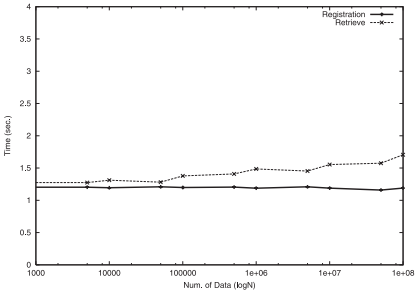


**図 7**　処理時間の変化（データ数）
Fig. 7　Processing time (Number of data).

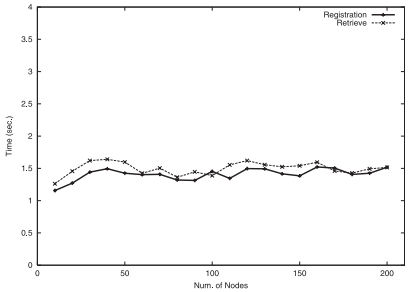


**図 8**　処理時間の変化（ノード数）
Fig. 8　Processing time (Number of node).

**図 7** に，処理時間の変化を示す．横軸を登録データの総数，縦軸を処理時間とする．登録データ数は数が多いため横軸は対数軸としている．

登録データを増加させた場合，登録処理に関わる時間の変動はほぼ見られなかった．しかし，検索処理に関わる時間はデータ数 100,000,000 の増加に対して 0.4 秒程度のわずかな線形の増加が確認された．変動の幅が小さいことから，さらなる対応については，実装コストの面を考慮した検討が必要である．

#### 4.2.4　ノード数の変化による登録・検索処理時間

本評価では，グローバルストレージに参加するノード数の変化によるデータの登録・検索に要する処理時間の変化について調査した．計測は，参加ノード数を徐々に増加させ，特定のノード数となったところで，(1) クエリ文の提供から登録を実行しそれに要する時間，(2) クエリ文の提供から検索を実行しそれに要する時間をそれぞれ計測した．本評価では，それぞれをクエリの試行を 100 回行い，その結果を調査した．

**図 8** に，処理時間の変化を示す．横軸を参加ノード数，縦軸を処理時間とする．

参加ノード数を増加させた場合，いずれの処理についても処理時間はノード数によって大きな変化が出ないことが分かった．処理にかかる時間の最大値と最小値の差はいずれも 0.4 秒程度であった．ノード数変動に比例しない形で処理時間が変動する理由としては，他の評価が比較的安定した PlanetLab を用いているのに対し，本評価では，評価用ノードの確保

のため，Load Average が高いなど安定しないがノードが稼働しているものも少なからず含まれたためであると考えられる．

#### 4.2.5　クエリの内容による登録・検索処理時間

本評価では，グローバルストレージに対して行われたクエリの内容の変化によるデータの登録・検索に要する処理時間の変化について調査した．計測は，クエリに含まれる WHERE 句の個数を増加させ，(1) クエリ文の提供から登録を実行しそれに要する時間，(2) クエリ文の提供から検索を実行しそれに要する時間をそれぞれ計測した．本評価では，それぞれをクエリの試行を 100 回行い，その結果を調査した．ここで利用したクエリを以下に示す．

INSERT INTO storage (s, i, c, start_t, end_t, u, e, m, a) VALUES ('SDP Seminar', 'A Seminar on the session description protocol', 'IN IP4 224.2.17.12/127', '2873397496', '2873404696', 'http://www.example.com/seminars/sdp.pdf ', 'j.doe@example.com', 'audio 49170 RTP/AVP 0, video 51372 RTP/AVP 99 ', 'rtpmap:99 h263-1998/90000');
SELECT c WHERE s = 'SDP Seminar' AND i LIKE '%Seminar%' AND i LIKE '%session description protocol%' AND c LIKE '%IP4%'; (WHERE 句: 5)

**図 9** に，処理時間の変化を示す．横軸を WHERE 句数，縦軸を処理時間とする．

クエリの内容を変更した場合，本来検索処理の回数は増加する．しかし，実装にスレッドによる並列処理を利用し，クエリを同時に処理できるため，処理時間の変動幅は非常に小さなものにとどめられている．

#### 4.2.6　クエリ発生間隔による登録・検索処理時間

本評価では，グローバルストレージに対するクエリの発生状況によるデータの登録・検索に要する処理時間の変化について調査した．計測は，クエリを発生させる回数を 30 回とし，各クエリの間の時間間隔を徐々に増加させ，それぞれの (1) クエリ文の提供から登録を実行しそれに要する時間，(2) クエリ文の提供から検索を実行しそれに要する時間をそれぞれ計測した．

**図 10** に，処理時間の変化を示す．横軸を時間，縦軸を処理時間とする．

クエリの発生間隔を変動した場合でも，処理時間の変動に大きな差は見られなかった．このことから，処理間隔が短い場合でも，長い場合でも，処理パフォーマンスに対する影響は少ないということがいえる．

#### 4.2.7　パフォーマンス向上手法の評価

本評価では，本システムにて採用したパフォーマンス向上処理にかかる定量評価として，(1) Dynamic Indexing，(2) ローカルキャッシュを用いた場合のデータの登録・検索に要する処理時間を計測した．本評価は，評価 2 と同様の実験を行い，処理時間の変化について調査した．

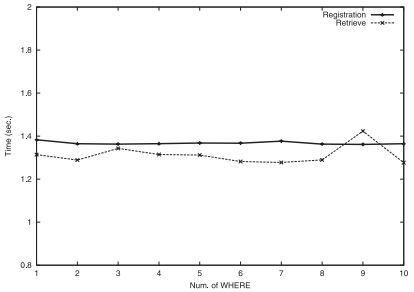


**図 9**　処理時間の変化（クエリ内容）
Fig. 9　Processing time (Query).

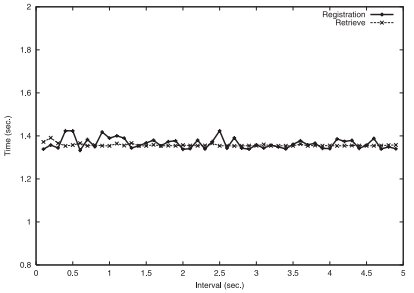


**図 10**　処理時間の変化（クエリ発生間隔）
Fig. 10　Processing time (Interval).

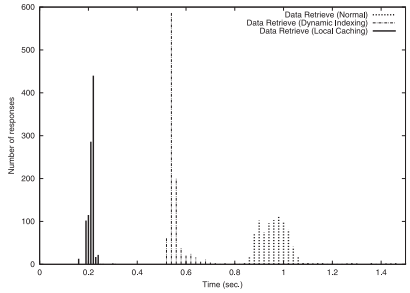


**図 11**　検索処理にかかる各処理時間の度数分布
Fig. 11　Processing time of retrieve operations.

**図 11** に，(a) 各手法を用いない処理，(b) Dynamic Indexing を用いた処理，(c) ローカルキャッシュを用いた処理をそれぞれ行った場合の処理時間の度数分布を示す．(a) の場合，処理時間の平均値は 1.07 秒，最小値は 0.86 秒，(b) の場合，平均値は 0.58 秒，最小値は 0.54 秒，(c) の場合，平均値は 0.22 秒，最小値は 0.17 秒となった．

各手法を用いない処理と比較して，Dynamic Indexing を用いた処理では，検索処理にかかる時間が短縮されている．これは，新たに設定されたインデックスの参照により，処理にかかる時間が短縮されるためである．Pastry Engine において，1 つのクエリを実行する際にかかるトランザクション数は，29 である．3.3.1 項にあげた例では，キーワードの検索に

3 クエリ，実データの問合せに 2 クエリがそれぞれ発生するため，総トランザクション数は 145 となる．これに対して，Dynamic Indexing では，検索に 1 クエリ，実データの問合せに 1 クエリで完結するため，総トランザクション数も 58 に削減される．

ローカルキャッシュを用いた処理では，1 度検索が行われた情報はローカルの Storage Layer に対して保存されるため，外部のノードに対しての問合せが発生しない．ローカルストレージに対する情報検索処理は非常に短時間に完結することができるため，検索にかかる処理時間の大幅な短縮が図られることが結果から分かった．

## 5. 関 連 研 究

文献 8) では，SAP の持つ問題を解決するため，リレーショナルデータベースを持つノードを各インターネットサイトに設置し，登録されたコンテンツ情報を Hop-by-Hop でノード間で配信し全ノードに浸透させ，セッション情報を通知する手法が提案されている．しかし，Hop-by-Hop での情報伝達にはノード間の状況によって伝達時間が長くなるという問題や，ノード間の接続関係を静的に設定する必要があるといった操作上の問題が存在する．

広域に分散したデータストレージを実現する手法に，構造化オーバレイネットワークを用いた手法がある．CAN[20]，Chord[21]，Pastry[22]，Kademlia[23] などのルーチングアルゴリズムを用いることで，$O(\log N)$（$N$ は全ノード数）の探索性能を持つストレージを構築することができる．しかし，KBR における最大の制約事項は，データ検索が全文一致検索のみしかサポートされないという点にある[24),25]．これにより，広域分散ストレージより情報を取得しようとしたユーザは，検索に用いる Key を完全な形で知っている必要がある．たとえば，“The first contents” という Key で登録された情報を取得する場合には，“contents” など一部のキーワードだけを部分一致させて検索し，情報を取得することができない．同様に，ある値の範囲を求めるような範囲検索や，それぞれの検索手法を組み合わせた複合検索も行うことができない．

これらの問題に対していくつかの解決手法が提案されている．ここでは，既存研究の中で，特に完全一致検索のみの制約の排除に向けた関連研究の考察を行う．

文献 26) では，“The Keyword” のようなキーワードセットを “The” と “Keyword” などに分割を行い，これらをハッシュテーブル上に分散させることによって部分一致検索を実現する手法を提案している．この手法では，部分一致検索についてのみ考慮しており，複数の検索手法に対応していない．また，シミュレータ上での評価にとどまっており，実装が存在しない．

これらの手法は，いずれもアルゴリズムの拡張を行うことで，部分一致検索や範囲検索といった検索をそれぞれ実現するものである．しかし，複数の検索を柔軟に行うことはできない．CANDy[25] では，キーワード分割による部分一致検索と Bloom Filter を用いた範囲検索を問合せに応じて組み合わせることにより，柔軟な検索を実現する手法を提案している．CANDy では，検索を行うための Key を格納する “Index DHT” と実際のデータを含む “Resource DHT” にそれぞれ必要な情報を登録する．検索を行う際には，“User Agent” を用い，与えられた “Query” に基づき，“Index DHT” を検索し目的の情報の Key を取得し，さらに “Resource DHT” に対して問合せを行うことで実際のデータを取得する．CANDy は，本論文で提案するシステムに似た構造を持っているが，手法の提案と実現性について言及するのみとなっており，具体的な実装や評価に関する記述もないため，その有効性を定量的に判断することは困難である．

## 6. む　す　び

本研究では，デジタルコンテンツに関する情報を記述したコンテンツ情報を集約し，その検索を実現するデジタル情報のコンテンツレポジトリを提案した．コンテンツ情報の管理には，多くの機能要件があり，これらを満たすシステムの実現は，将来のコンテンツ配信環境の発展において重要な要素である．本研究では，グローバルストレージと RDB の組合せによる複合的なデータストレージの採用により，これらの機能要件を満たした．本研究では，機能拡張性と情報管理の迅速性を満たす必要のあるグローバルコンテンツ情報の管理に，広域分散システムによって構成されたストレージアプリケーションとそれを管理するアクセスアプリケーションを用い，機能拡張性のある情報管理基盤を実現し，さらに従来の広域分散システムにおける課題であった検索の柔軟性についても考慮したシステムを実現した．これらの手法の有効性を検証するために，PlanetLab 上で動作するプロトタイプアプリケーションを実装し，性能評価を行った．これによって，機能要件にあげた目的を満たすシステムを実現した．この実現により，ポータルサイトやアプリケーションがコンテンツレポジトリを参照したコンテンツ配信により，ユーザにとっての利便性が向上するだけでなく，新しいストリーミングサービスの展開が期待できる．

今後の課題として，検索手法の柔軟性向上とパフォーマンス向上があげられる．本論文では，プロトタイプ実装の容易さの観点から基本的なキーワード分割による部分検索を取り上げているが，n-gram 法によってバイグラムやトリグラムを生成し検索に用いることや，Ringed Bloom Filter[27] を用いることでさらに柔軟な検索が可能となる．分散ハッシュテー

ブルでの部分検索は，検索に利用されるトラフィックや通信処理がさらに増えるため，登録処理や検索処理におけるオーバヘッドが高まる．そこで，今回提案したDynamic Indexingやローカルキャッシュ以外の手法を盛り込んでいくことが必要であると考えている．また，大量のデータの検索時にわずかながら線形に処理時間が増大することが発見されている．これについても開発コストとパフォーマンス改善の2面のバランスに応じた対応を検討したい．

また，本研究で提案・実装したシステムの公開と展開を進めることも今後の課題としてあげられる．今回実装したオーバレイツールキットならびにアクセスエージェントそれぞれをパッケージ化し順次公開を進めており[28]，その公開により，さらに多くの実環境での評価とシステムの改善を行うことが可能であると考えている．

**謝辞**　本研究の一部は，情報通信研究機構（NICT）の委託研究「ダイナミックネットワーク技術の研究開発」の助成を受けて実施したものである．

## 参 考 文 献

1) YouTubeLLC: YouTube. http://www.youtube.com/（参照 2010/01）
2) Yahoo! Inc.: Yahoo! Video. http://video.yahoo.com/（参照 2010/01）
3) 株式会社ニワンゴ：ニコニコ動画．http://www.nicovideo.jp/（参照 2010/01）
4) Spoto, S., Gaeta, R., Grangetto, M. and Sereno, M.: Analysis of PPLive through active and passive measurements, *IPDPS '09: Proc. 2009 IEEE International Symposium on Parallel & Distributed Processing*, Washington, DC, USA, IEEE Computer Society, pp.1–7 (2009).
5) コグニティブリサーチラボ株式会社：KeyHoleTV, http://www.v2p.jp/video/（参照 2010/01）．
6) Handley, M., Perkins, C. and Whelan, E.: Session Announcement Protocol, RFC 2974 (Experimental) (2000).
7) Handley, M., Jacobson, V. and Perkins, C.: SDP: Session Description Protocol, RFC 4566 (Proposed Standard) (2006).
8) Hitoshi, A., Wacharapol, P. and Soh, Y.: Channel Reflector: An Interdomain Channel Directory System, *IEICE Trans. Communications*, Vol.89, No.9, pp.2860–2867 (2006).
9) Asaeda, H. and Roca, V.: Requirements for IP Multicast Session Announcement, Internet draft, IETF (2009). draft-ietf-mboned-session-announcement-req-02
10) Google: Google, http://www.google.com/（参照 2010/01）．
11) Brin, S. and Page, L.: The anatomy of a large-scale hypertextual Web search engine, *Comput. Netw. ISDN Syst.*, Vol.30, No.1-7, pp.107–117 (1998).
12) Risson, J. and Moors, T.: Survey of Research towards Robust Peer-to-Peer Networks: Search Methods, RFC 4981 (Informational) (2007).
13) Loo, B.T., Hellerstein, J.M., Huebsch, R., Shenker, S. and Stoica, I.: Enhancing P2P file-sharing with an internet-scale query processor, *VLDB '04: Proc. 30th International Conference on Very Large Data Bases*, VLDB Endowment, pp.432–443 (2004).
14) Stonebraker, M., Aoki, P.M., Litwin, W., Pfeffer, A., Sah, A., Sidell, J., Staelin, C. and Yu, A.: Mariposa: A wide-area distributed database system, *The VLDB Journal*, Vol.5, No.1, pp.048–063 (1996).
15) Dahan, S. and Sato, M.: Survey of Six Myths and Oversights about Distributed Hash Tables' Security, *ICDCSW '07: Proc. 27th International Conference on Distributed Computing Systems Workshops*, Washington, DC, USA, IEEE Computer Society, p.26 (2007).
16) Rowstron, A. and Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, Lecture Notes in Computer Science, Vol.2218, pp.329–350 (2001).
17) Shamir, A.: How to share a secret, *Comm. ACM*, Vol.22, No.11, pp.612–613 (1979).
18) Hertz, M. and Berger, E.D.: Quantifying the performance of garbage collection vs. explicit memory management, *SIGPLAN Not.*, Vol.40, No.10, pp.313–326 (2005).
19) Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M. and Bowman, M.: PlanetLab: An overlay testbed for broad-coverage services, *SIGCOMM Computer Communication Review*, Vol.33, No.3, pp.3–12 (2003).
20) Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Schenker, S.: A scalable content-addressable network, *SIGCOMM '01: Proc. 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, New York, NY, USA, ACM, pp.161–172 (2001).
21) Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F. and Balakrishnan, H.: Chord: A scalable peer-to-peer lookup protocol for internet applications, *IEEE/ACM Trans. Netw.*, Vol.11, No.1, pp.17–32 (2003).
22) Rowstron, A. and Druschel, P.: Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems, *Middleware 2001*, Lecture Notes in Computer Science, Vol.2218, pp.329–350, Springer Berlin/Heidelberg (2001).
23) Maymounkov, P. and Mazières, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric, *Peer-to-Peer Systems*, Lecture Notes in Computer Science, Vol.2429, pp.53–65, Springer Berlin/Heidelberg (2002).
24) Harren, M., Hellerstein, J.M., Huebsch, R., Loo, B.T., Shenker, S. and Stoica, I.: Complex Queries in DHT-based Peer-to-Peer Networks, *IPTPS '01: Revised Papers from the 1st International Workshop on Peer-to-Peer Systems*, London, UK, pp.242–259, Springer-Verlag (2002).

25) Bauer, D., Hurley, P., Pletka, R. and Waldvogel, M.: Bringing efficient advanced queries to distributed hash tables, *29th Annual IEEE International Conference on Local Computer Networks*, pp.6–14 (2004).
26) Reynolds, P. and Vahdat, A.: Efficient peer-to-peer keyword searching, *Middleware '03: Proc. ACM/IFIP/USENIX 2003 International Conference on Middleware*, New York, NY, USA, pp.21–40, Springer-Verlag New York, Inc. (2003).
27) 清　雄一，松崎和賢，本位田真一：Ringed Bloom Filter による分散ハッシュテーブルのトラフィック量削減，情報処理学会論文誌，Vol.48, No.7, pp.2267–2277 (2007).
28) Channel Reflector Project, http://www.channel-reflector.org/（参照 2010/11）.

(平成 22 年 5 月 31 日受付)
(平成 22 年 11 月 5 日採録)

**三島　和宏**（学生会員）
平成 16 年慶應義塾大学環境情報学部卒業．平成 18 年慶應義塾大学大学院政策・メディア研究科修士課程修了．現在，同博士課程在籍中．分散環境における適応型メディアコンテンツ配信とメディアガイド配信に関する研究に従事．電子情報通信学会，映像情報メディア学会各学生会員．

**朝枝　仁**（正会員）
平成 3 年慶應義塾大学理工学部卒業．日本アイ・ビー・エム（株），フランス国立情報科学制御研究所（INRIA）リサーチエンジニアスペシャリストを経て，平成 17 年慶應義塾大学大学院政策・メディア研究科助教．平成 20 年より同大学院特別研究准教授．インターネット・マルチキャスト，ルーティング・アーキテクチャの研究に従事．ACM，IEEE，電子情報通信学会各会員．博士（政策・メディア）．

**村井　純**（正会員）
昭和 59 年慶應義塾大学大学院理工学研究科博士課程数理工学専攻修了．東京工業大学総合情報処理センター助手，東京大学大型計算機センター助手，慶應義塾大学環境情報学部助教授を経て，1997 年より同教授．2009 年より同環境情報学部長．博士（工学）．