オーバレイ構築ツールキット Overlay Weaver

首藤一幸[†,††] 田中良夫[†] 関口智嗣[†]

我々は，ネットワークオーバレイのアルゴリズム研究基盤として，オーバレイ構築ツールキット Overlay Weaver を開発・配布している．これを用いることで，計算機 1 台上で繰り返し動作試験を行いながら，少ないコード量で structured オーバレイのアルゴリズムを実装できる．また，アルゴリズムの差し替え，および，計算機 1 台上での数千ノード規模のエミュレーションが可能であるため，アルゴリズム間の公正な比較を行うことができる．同時に，こうして実装したアルゴリズムはそのまま実ネットワーク上で動作するため，アルゴリズム研究の成果が直接アプリケーションに結び付く．

本ツールキットでは，アルゴリズムの実装を容易なものとするために，各アルゴリズムに共通するルーティング処理をアルゴリズム実装から分離した．本論文では，この分離を可能とする両者間のインタフェースを提案する．この分離によって，いくつかのアルゴリズムを，それぞれたかだか数百ステップで実装できた．ルーティング処理の側も，iterative および recursive の両ルーティング様式を差し替えられるようになった．

Overlay Weaver: An Overlay Construction Toolkit

Kazuyuki Shudo,[†,††] Yoshio Tanaka[†] and Satoshi Sekiguchi[†]

We have been developing an overlay construction toolkit called Overlay Weaver as the groundwork for future research on algorithms. Algorithm designers can implement structured overlay algorithms in just hundreds of lines of code with the toolkit and improve these rapidly by iterative testing them on a single computer. The toolkit enables designers to make fair and large-scale comparisons between of new and existing algorithms, since the implemented algorithms are pluggable and the emulator provided by the toolkit can host thousands of virtual nodes. Furthermore, the implemented algorithms can work on a real network in addition to the emulator. The toolkit enables algorithms developed through research to be used in applications directly.

We decomposed the routing layer of software which constructs structured overlay into a routing process and a routing algorithm. In this paper, we propose a programming interface between them. Decomposition enables an algorithm to be implemented in just hundreds of lines of code. The routing process also became pluggable by decomposition and the toolkit can provide both iterative and recursive routing.

1. はじめに

大規模なインターネット上分散アプリケーションでは，ノード数が数万，数百万と増えてもなお性能と耐故障性を保つために，自律的・非集中的にアプリケーションレベルのネットワークを構成することが不可欠となっている．ここで構成されるネットワークは，下位ネットワークの物理的トポロジとは独立したトポロジを持つためネットワークオーバレイと呼ばれ，検索やマルチキャストなどの機能を提供する．

† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology（AIST）
†† 現在，ウタゴエ（株）
Presently with Utagoe, Ltd.

structured オーバレイに限定しても，これまで多くのアルゴリズム[1)~5)] が提案されてきた．また，それらを実装したライブラリ[6)~9)] やアプリケーション[10)~12)] も現れている．

アルゴリズムの設計・評価においては，可能であれば数百万という多数のノードを想定した実験が欠かせない．大規模な実験環境を用意することは容易ではないため，通常，アルゴリズムの動作と有用性はシミュレーションで確認される．その後，インターネット上の実環境で評価，応用するためには，実環境で動作するソフトウェアを，別途，実装する必要がある．このため，提案時には実環境で動作する実装が伴わないアルゴリズムも多い．そこで提案されたアルゴリズムを実環境向けに実装する開発者にとっては，シミュレーション時と寸分違わぬアルゴリズムとパラメータにし

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年9月

ない限り，自らの実装がどういう挙動を示すかは，実のところ，明らかではない．オーバレイの挙動は，アルゴリズムその他のパラメータ，例えば定期的な処理の時間間隔に依存するためである．

我々は，この，アルゴリズム研究から応用までの距離を短縮するべく，オーバレイ構築ツールキット *Overlay Weaver* を開発・配布している[13),14)]．structured オーバレイのアルゴリズムを設計，実装する際に本ツールキットを用いることで，一度の実装で，インターネット上実環境での動作と計算機 1 台上での数千ノード規模のエミュレーションの双方が可能となる．これにより，大規模な動作試験をエミュレータ上で繰り返し行いながらアルゴリズムを設計，実装して品質を向上させ，その結果得られた実装をそのまま実環境で用いることが可能となる．

我々はまた，structured オーバレイ上のルーティング処理をアルゴリズム実装から切り離すことを可能とする，両者間のインタフェースを提案する．ルーティング処理の実装はツールキット側が提供することで，アルゴリズムの実装が容易になった．これによって，Chord[1)]，Pastry[2)]，Tapestry[3)]，Kademlia[4)] の各アルゴリズムをそれぞれたかだか数百ステップで実装できた．同時に，ルーティング処理部の切り替えも可能となり，アルゴリズム実装には一切の変更なしで iterative ルーティングと recursive ルーティング[6),15)] からどちらかを実行時に選択できるようになった．

以降では，まず 2 章で関連研究を挙げる．3 章では，アルゴリズム実装とルーティング処理を切り離す方法を示し，両者間のインタフェースを提案する．続く 4 章では本ツールキットを構成する各コンポーネントとツールを説明し，それらがアルゴリズムの設計と実装，試験，比較をどうサポートするかを述べる．5 章では，各アルゴリズムを少ないコード量で実装できること，多ノードでの動作試験やアルゴリズム間比較が可能であることを示す．

2. 関 連 研 究

複数のアルゴリズムをサポートしているオーバレイ構築ソフトウェアには，本ツールキットの他に MACEDON[16),17)] と Mace[18)]，P2[19)] がある．

MACEDON では，C，C++に似た独自の専用言語でオーバレイのアルゴリズムを記述し，それを MACEDON のコンパイラで処理することで動作する C++言語のコードを得る．本ツールキットと同様に実環境での通信は UDP，TCP で行う他，“partial support”[16)] ながらネットワークシミュレータ ns 用のコードも生成できる．アルゴリズムは，分散ハッシュ表（DHT）としては Chord と Pastry の実装を提供している．

アルゴリズムの記述に必要となるコードの分量は，MACEDON と本ツールキットで最大 1.5 倍程度の違いであり，大きな差はない（5.1 節）．通信などのライブラリ提供に加えて，MACEDON は専用の言語を導入することで必要なコード量を減らしており，一方，本ツールキットはルーティング処理をアルゴリズムから分離することで同等の効果を得ている．

MACEDON はエミュレータ（4.2 節）を持たず，多ノードでの試験は計算機 2〜50 台以上からなるインターネットエミュレータ ModelNet 上で行われている[16)]．エミュレートされたノード数は最大でも 1000 にとどまっており，そのノード数ではオーバレイの構築（経路表が安定する様子の観察）しか行われていない．これに対して本ツールキットは，計算機 1 台で 4000 ノードのエミュレーションを達成している．

Mace[18)] は MACEDON の後継プロジェクトである． ただし，Mace release 0.7.1-20050919 で実装されている structured オーバレイは Pastry と，それを利用した Scribe および SplitStream に限られており，様々なアルゴリズムをサポートできることはまだ実証されていない．

P2 では，Prolog に似た宣言的オーバレイ記述言語 OverLog で，経路表といったノード状態の更新処理を記述する．オーバレイ記述はコンパイルされ，それを分散データフローエンジンが実行する．オーバレイのアルゴリズム記述は簡潔であり，例えば Chord の実装は，47 のルールと表の宣言 13 行，ステップ数で表すと 142 で済んでいる．本ツールキットではアルゴリズムは Java 言語で手続的に記述されるのに対して，P2 では専用言語で宣言的に記述される点が相違点である．提案間もない専用言語であるため，アルゴリズム記述の容易さや保守性はまだ明らかではない．また，P2 release 0.6 で実装されているアルゴリズムは Narada と Chord，そのうち structured オーバレイは Chord のみであり，Mace 同様，多くのアルゴリズムの記述可能性は未検証である．実験の規模は，計算機 110 台を用いた 500 ノードにとどまっている．

MACEDON，Mace，P2 のどれも，アルゴリズム記述は iterative/recursive ルーティングのどちらかに特化せざるを得ない．それに対して本ツールキットでは，アルゴリズム記述には変更なしで両様式を差し替えて動作させることができる．

structured オーバレイのアルゴリズムを実装しているライブラリとしては，他にも Bamboo[6),7)] ，

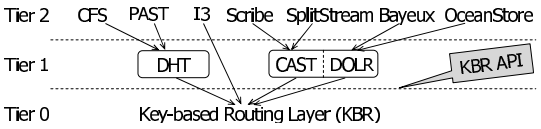


図 1 Key-based routing (KBR)（論文 22) の Figure 1）
Fig. 1 Key-based routing (KBR)（Figure 1 from Dabek et al.[22]）.

Chimera および Tapestry[3),8)]，Khashmir[9)] などが入手可能となっている．それらはどれも単一のアルゴリズムを実装しており，複数のアルゴリズムを差し替えて使えるようにはなっていない．

p2psim[20)] は peer-to-peer プロトコルのシミュレータである．アルゴリズムとしては，Chord，Accordion，Koorde，Kelips，Tapestry，Kademlia の実装を提供している．これらはすべて DHT，つまり structured オーバレイである．Accordion の提案[21)] では Chord との比較のために p2psim が使われており，最大で 3000 ノードのシミュレーションが行われている．

シミュレーションには，再現可能なシミュレーションを行うことができる，という利点がある．本ツールキットが行うような並行システムのエミュレーションは非決定的であり，結果に再現性はない．一方で，p2psim はシミュレータであるため，p2psim 用のアルゴリズム実装を実環境で動作させることはできない．また，アルゴリズムを記述する際は遠隔呼び出し（RPC）を含む通信を直接書くことになるため必要となるコード量は少なくなく，本ツールキットや MACEDON の数倍となっている（5.1 節）．

3. ルーティング共通処理とアルゴリズムの分離

Dabek らは，あらゆる structured オーバレイに共通するルーティング処理を key-based routing（KBR）と呼んだ[22)]．**図 1** の通り，第 0 層と位置付けられる KBR，つまりルーティング層の上には，より高位のサービスである DHT やマルチキャスト，エニーキャスト，メッセージ配送などを構築できる．

本ツールキットもこの抽象化に従い，ルーティング層と，その上の DHT サービス，マルチキャストサービスを分離している．この抽象化によって各層は交換可能となり，アプリケーションを変更することなく，様々なルーティングアルゴリズムを差し替えて利用することが可能となっている．

ツールキットの一方の利用者であるアプリケーション開発者にとっては，多くの選択肢から目的に合ったルーティングアルゴリズムを選択できることが望ましい．また，もう一方であるアルゴリズム研究者にとっては，自身が設計するアルゴリズムを実装しやすいことに加えて，比較対象となる他のアルゴリズムが数多く提供されていることが望ましい．そのためには，ツールキットにはアルゴリズムの実装が容易であることが求められる．しかし，ルーティング層の実装は，アルゴリズムの本質的な部分だけでなく，ノード間の通信プロトコルの設計，実装から行う必要があり，容易とは言えない．

そこで我々は，ルーティング層から各種のルーティングアルゴリズムに共通する処理をくくり出し，各アルゴリズムに固有の処理との間のプログラミングインタフェースを見出した．共通処理はツールキット側が提供することでアルゴリズムの実装が容易になり，様々なアルゴリズムをたかだか数百ステップで実装することが可能となった（5.1 節）．

図 2 に，本ツールキットのランタイムを構成するコンポーネント群を示す．図中で，Dabek らの提案（図 1）[22)] におけるルーティング層に相当するのは，Routing Driver と，そこから直接利用される Routing Algorithm，Messaging Service である．このうち，*Routing Driver* が共通処理を行うコンポーネントである．*Routing Algorithm* は各種ルーティングアルゴリズムの実装であり，Routing Driver はここから必要な情報を得てルーティング処理を行う．

共通処理のくくり出しによって，アルゴリズム実装だけでなく共通処理の側も，複数の実装を用意して実行時に選択することが可能となった．具体的には，本ツールキットは Routing Driver の実装として iterative ルーティングと recursive ルーティング[6),15)] の 2 種類を提供している（3.3 節）．そのどちらかを，アルゴリズム実装（Routing Algorithm）には一切の変更なしで，実行時に選択することが可能となった．

3.1 アルゴリズム側のインタフェース

Routing Driver から見た Routing Algorithm 側のプログラミングインタフェースを説明する．本ツールキットの実装言語である Java 言語の形式で示す．

```
RoutingContext initialRoutingContext(
ID target)
```

ルーティング中，ノードからノードへ伝播する状態，文脈の初期値を返す．文脈とは例えば Koorde であれば，論文 5) の Figure 3 中の i（仮想ノードの ID）と $kshift$ である．ここで引数 `target` はルーティングのターゲットである．

```
IDAddressPair[] closestNodes(ID target,
int maxNumber, RoutingContext context)
```

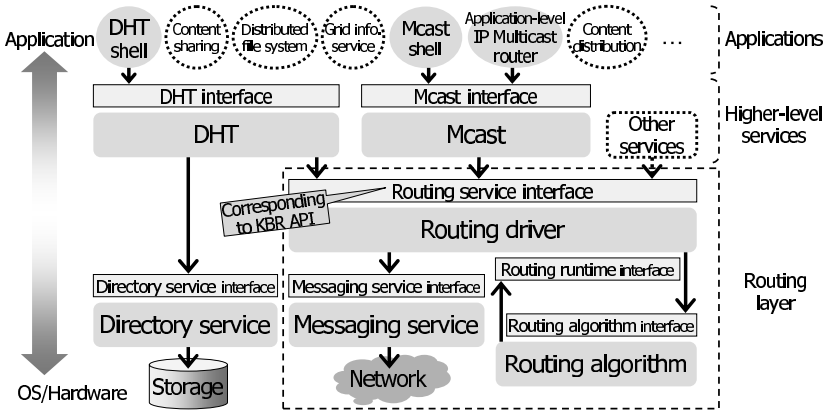


図 2 Overlay Weaver のランタイムを構成するコンポーネント群
Fig. 2 Components organizing runtime of Overlay Weaver.

引数 `target` で表される ID に対して最も近い ID を持つノードを，最大 `maxNumber` ノード返す．Routing Driver は，返されたノード群を，先頭から順にルーティングの次ホップ候補として扱う．返り値には，この処理が実行されているノード自身を含めてもよい．

引数 `context` は，ノード間を伝播する文脈を必要としないアルゴリズムでは null が与えられる．アルゴリズム実装は，次ホップに伝播させる文脈を，返り値である `IDAddressPair` 型の値，つまり次ホップ候補に付加して返すことができる．

ここで，`ID` はルーティングのキーやノードの ID を表す型であり，`IDAddressPair` は ID と通信用アドレスの組である．通信用アドレスは，例えば UDP や TCP では IP アドレスとポート番号の組となる．

```
IDAddressPair[] adjustRoot(ID target)
```

ルーティングの最終段階で，ルーティングのターゲットに最も近い ID を持つノードに対して呼び出される．引数 `target` がターゲットである．自身が root，つまりルーティングの目的地である場合には `null` または空の配列を返す．そうでない場合は，root ノードの候補を返す．

```
void join(IDAddressPair[] route)
```

あるノードがオーバレイに参加しようとする際，Routing Driver はそのノードの ID をターゲットとしてルーティングを行う．このメソッドは，ルーティングが完了した後，参加しようとしているノードに対して呼び出される．引数 `route` としては，参加しようとしているノード自身から（参加前の）root ノードまでの経路が与えられる．

```
void join(IDAddressPair joiningNode,
IDAddressPair lastHop, boolean isRootNode)
```

このメソッドは，あるノードがオーバレイに参加しようとする際のルーティングにおいて，経路上の全ノードに対して呼び出される．引数 `joiningNode` は参加しようとしているノード，`lastHop` は経路を 1 つ遡ったノードである．引数 `isRootNode` としては，root ノードに対して呼び出される際には `true`，そうでない場合には `false` が与えられる．

```
void touch(IDAddressPair from)
```

`from` から何らかのルーティング用メッセージを受信した際に呼び出される．

```
void forget(IDAddressPair node)
```

`node` を経路表から外すべき際に呼び出される．現実装では，`node` との通信に連続して一定回数失敗した場合に，このメソッドが呼び出される．

```
BigInteger distance(ID to, ID from)
```

引数として与えられた 2 つの ID 間の距離を返す．

厳密には，このメソッドが Routing Algorithm 側インタフェースの一部である必要はない．Routing Driver から呼び出されることはなく，アルゴリズムの実装内部で用いられるだけだからである．

ここで `BigInteger` は多倍長整数型である．

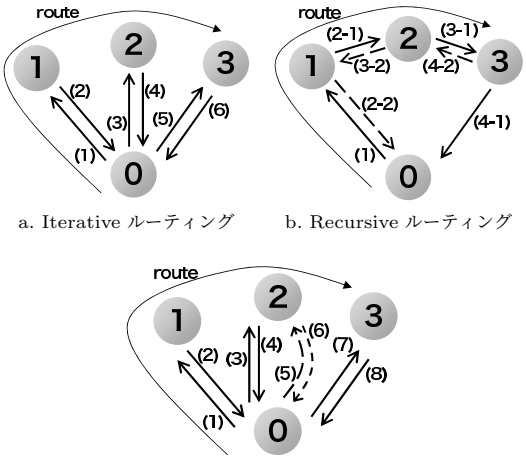
3.2 各アルゴリズムによるインタフェースの実装

本ツールキットが実装を提供しているルーティングアルゴリズムは Chord と Pastry，Tapestry，Kademlia である．**表 1** は，これらの各アルゴリズムが Routing Algorithm 側インタフェースが規定するどのメソッドに対して有効な実装を提供しているかを示したものである．表中の `join`(1)，(2) はそれぞれ，3.1 節に 2 度現れる `join` に，現れる順番通りに対応している．

`adjustRoot` は Chord が必要とする（‡1）．他のアルゴリズムでは `closestNode` を呼び出して次ホップを順に辿り最も近いノードにたどり着けば，それがルーティングの目的地，つまり root ノードとなる．しかし

表 1 アルゴリズム側インタフェースが規定するメソッドと各アルゴリズムの対応
Table 1 Correspondence between methods specified by Routing Algorithm interface and routing algorithms.

	Chord	Pastry	Tapestry	Kademlia
`closestNodes`	×	×	×	×
`adjustRoot`	× ‡1			
`join`(1)	×			
`join`(2)		×	×	
`touch`		×	×	×
`forget`	×	×	×	×
`distance`	×	×	‡2	×



a. Iterative ルーティング　b. Recursive ルーティング

c. Iterative ルーティング（Overlay Weaver の場合）

図 3 ルーティングの様式
Fig. 3 Routing styles.

Chord では，最も近いノードではなくその successor が root ノードである．そのため，最も近いノードが求まった後で，root ノードを求めるための補正が必要となる．`adjustRoot` メソッドはこの補正を行う．

Tapestry はルーティングに ID 間距離を用いず，定められた手順に従って次ホップを決める．このため，Tapestry が `distance` を実装する必要はない（‡2）．Pastry も Tapestry と同様に，routing table を参照してのルーティングには距離を用いない．しかし，ルーティングの最終段階で用いる leaf set を保守するために，`distance` を呼び出してノードをソートする．

3.3 ルーティング共通処理

Routing Driver，すなわちルーティング共通処理の実装として，本ツールキットは iterative ルーティングと recursive ルーティング[6),15)] の 2 種類を提供している．そのどちらかを実行時に選択できる．

図 3 は両様式で行われる通信を図示したものである．円はノードを表し，直線の矢印は破線を含めてメッセージの送受信を表す．括弧内の数字は，通信が起こる順を表す．図の通り，iterative ルーティングではルーティングを始めたノードが経路上の各ノードに対して順に問い合わせを行い，recursive ルーティングではルーティング要求が経路に沿って転送されていく．

図 3 中の破線は，一般的な iterative/recursive ルーティングでは行われるとは限らない通信を表す．図 3 c の（5）と（6）は `adjustRoot` （3.1 節）の問い合わせと返答を表す．本ツールキットが実装を提供しているアルゴリズムのうちこれを必要とするのは Chord のみであり，他のアルゴリズムでは（5），（6）の通信は起こらない．

図 3 b 中の破線は，ルーティング要求メッセージに対する返答であり，到達確認メッセージを表す．これは，信頼性のない通信路でも確実にルーティングを完遂するために送受信している．ただし，経路上の各ノードは，到達確認の返信よりもルーティング要求の転送を優先して行う．このため，ルーティングに要する時間に対して通信遅延が支配的である場合には，iterative よりも recursive ルーティングの方がルーティングに要する時間が短いことが期待できる．

1 回のルーティングに要するメッセージの数は次の通りである．Chord のように `adjustRoot` を必要とするアルゴリズムを除き，図 3 より，経路が n ホップの場合 iterative ルーティングで $2n$，recursive ルーティングで $2(n+1)$ である．`adjustRoot` を必要とするアルゴリズムで iterative ルーティングを行う場合には，$2(n+1)$ となる．

4. ツールキットの構成

本章では，本ツールキットを構成するランタイムと各種ツールについて説明する．また，それらがルーティングアルゴリズムの設計と実装，試験，そしてアルゴリズム間の比較をどうサポートするかを述べる．

本ツールキットは，図 2 に示す各コンポーネントと，次に挙げるツール群から構成される．

- 分散環境エミュレータ
- シナリオ生成器
- メッセージカウンタ
- メッセージング可視化ツール

Routing Driver と Routing Algorithm 以外のコンポーネントも，複数種類の実装を実行時に差し替えて利用することが可能である．例えば Messaging Service としては次の 3 種類を提供している．

- UDP 実装
- TCP 実装

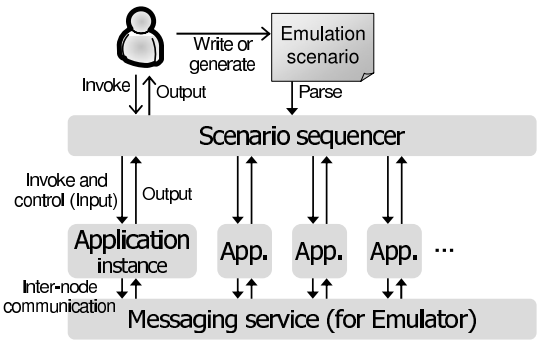


図 4 エミュレータの構成
Fig. 4 Structure of Emulator.

- 単一プロセス内のスレッド間で通信をエミュレートする実装（エミュレータ用）

4.1 高レベルサービスとサンプルアプリケーション

3 章で説明したルーティング層の上に，アプリケーションが直接利用する高レベルサービスが実装される（図 2）．本ツールキットは高レベルサービスとして，DHT に加えて Mcast を提供している．Mcast はマルチキャスト機能を提供する高レベルサービスである．これを用いることで，ID で識別されるグループへの参加，離脱，また，グループに参加している全ノードへのマルチキャストを行うことができる．

サンプルアプリケーションとして，それぞれ DHT シェル，Mcast シェルを提供している．これらを用いることで，両サービスを起動して，キャラクタ端末もしくはネットワーク経由で直接制御することができる．これらシェルをエミュレータ（4.2 節）と組み合わせて利用することで，ルーティングアルゴリズムの動作試験や比較を行うことができる．5 章では，こういった比較が可能であることを示す．

4.2 エミュレータ

本ツールキットは分散環境エミュレータを提供しており，数千ノードを計算機 1 台上でエミュレートできる．これを用いることで，ルーティングアルゴリズムを設計，実装する際に動作試験をエミュレータ上で繰り返し行いながら品質を向上させることができる．また，同一のシナリオを異なるアルゴリズムで実行することでアルゴリズム間の公正な比較を行うことができる．なおかつ，その結果得られた実装をそのまま実際の分散環境で動作させることができるため，アルゴリズム研究の成果が直接アプリケーションに結び付く．

エミュレータの構成を**図 4** に示す． 起動されたアプリケーションインスタンスには仮想ホスト名が割り当てられ，仮想ホスト間の通信をエミュレータ用 Messaging Service が行う．

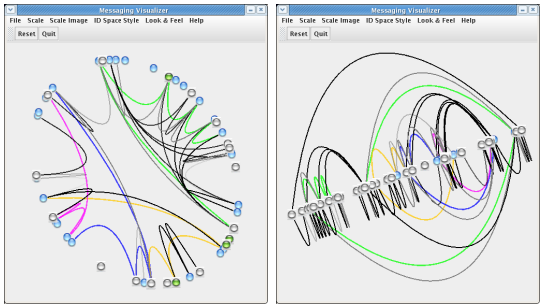


図 5 メッセージング可視化ツール
Fig. 5 Messaging Visualizer.

シナリオファイルにはまず，起動するクラスの指定，起動する際の引数，起動の指示を記述する．Scenario Sequencer はそれを読み込み，アプリケーションをスレッドとして起動する．シナリオには続いて，アプリケーションに対して与える指示を記述できる．1 つの指示は，指示内容と相手先，指示を与える時刻から成る．指示は標準入力を通して与えられるため，アプリケーションが受け付ける文字列ならどういった指示でも与えることが可能である．

4.3 メッセージカウンタ

Messaging Service には，メッセージ送受信をネットワーク越しに逐一報告させることができる．メッセージカウンタは，この報告を受信してメッセージ数を数えるツールである．

4.4 メッセージング可視化ツール

Messaging Visualizer は，メッセージ送受信の様子を実行中に可視化するツールである．可視化結果を**図 5** に示す．ノードは，円環や直線などの上に，その ID に応じた位置に描画される．通信は，発生した時点で送信元と送信先ノードの間に弧が描かれる．Mcast がマルチキャストのために構築する配送木も描画される．

このツールを用いることでアルゴリズムの動作を直観的に把握できる．動作試験に有用であるだけでなく，アルゴリズムの実演説明に用いることもできる．

4.5 各アルゴリズムの実装とパラメータ

本ツールキットが提供する Chord，Pastry，Tapestry，Kademlia 実装について，その詳細と，5 章の実験で用いるパラメータを述べる．

Chord

stabilization を行う通常のアルゴリズムに加えて，論文 1) の Figure 6 に示されている，オーバレイ参加時に経路表を完成させてしまうアルゴリズムについても実装を提供している．こちらのアルゴリズムは，以降「Chord (Figure 6)」と表記する．

手続き stabilize を呼び出す間隔は，はじめは 10 秒だが，経路表に変更がなかった場合には徐々に延ばしていき，最長 120 秒まで広げる．fix_fingers の呼び出し間隔は，finger table の埋まり具合いに応じて 5 秒から 600 秒の間の値をとる．

Pastry

ID は 4 ビットを 1 つの digit とした（$b = 4$）．Pastry，Tapestry 実装ともに，この値は実行時に指定できる．leaf set のサイズは 8 とした（$|L| = 8$）．

経路表維持のためには，Pastry の最初の提案[2] とは異なるが，論文 23) の Section 2.2 に記述されている periodic routing table maintenance を行う．その間隔は 60 秒とした．

Tapestry

ID は，論文 3) を通して行われている通り，16 進数の 1 桁を 1 つの digit として扱う（$\beta = 16$）．Pastry と同じパラメータである．

現実装は backup links を用意しない（$c = 1$）．また，backpointer を保持しない．

Kademlia

k-bucket の長さは論文 4) の通り 20 とした（$k = 20$）．ルーティング中に保持するターゲット ID への最近接ノードの数も論文 4) の通り 20 としたが，問い合わせを受けたノードが返す最近接ノードの数は 20 ではなく 5 とした．

並行クエリ（3.3 節）の並行性（concurrency）は，論文の通り 3 とした（$\alpha = 3$）．

5. 評　　価

本研究の目標は，アルゴリズムの設計と実装を容易にしてその成果を直接アプリケーションに結び付けることである．そこで，次の 4 点について本ツールキットを評価する．

- アルゴリズムの実装が容易であること
- 多ノードのエミュレーションでアルゴリズム実装の動作を確認できること
- アルゴリズム間の比較を公正に行えること
- アプリケーションが実ネットワークで動作すること

5.1 アルゴリズム実装のコード量

アルゴリズム実装の容易さを示すために，**図 6** に，現実装が提供しているアルゴリズム実装のステップ数を MACEDON[16),17)]，Mace[18)]，p2psim[20)] と比較して示す．ここでステップ数とは，空行とコメントを除いた後のコードの行数である．

本ツールキット上では，どのアルゴリズムもたかだか数百ステップで実装できている．最もステップ数が多い Pastry でも 900 ステップ弱である．これは，ルーティングに共通する処理をアルゴリズム実装から分離したこと（3 章）の成果である．

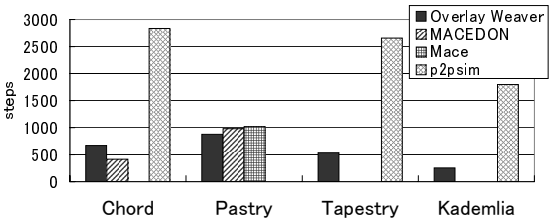


図 6　各アルゴリズム実装のステップ数
Fig. 6　Lines of code for each routing algorithm.

オーバレイのアルゴリズムを記述するための専用言語を用意している MACEDON と比較しても，Chord でたかだか約 1.6 倍のステップ数であり，Pastry のステップ数は本ツールキットの方が少ない．p2psim ではアルゴリズム実装自身が通信や RPC といった低レベルの処理を行うため，コードの量が多くなっている．MACEDON の Chord 実装は，本ツールキットとは異なり，論文 1) に記述されているアルゴリズムのサブセットであることを追記しておく．ID は 160 ビットではなく 32 ビットであり，保持できる successor はただ 1 つである．

5.2 4000 ノードのエミュレーション

エミュレータ上で大規模な動作試験が可能であることを示すために，計算機 1 台で 4000 ノードをエミュレートした結果を示す．この実験の目的はアルゴリズムの比較やパラメータの評価それ自体ではなく，本ツールキットを用いることで公正な比較や評価を行い得ることを示すことである．

実験は，3.4 GHz Pentium 4（Prescott）と 1 GB のメモリを搭載した PC で行った．OS は Linux 2.6.15 である．本ツールキットの実行に用いた Java 実行系は，Java 2 Standard Edition（J2SE） 5.0 Update 6 である．

図 7，**図 8** に，4000 ノードのエミュレーションを行った際の，1 秒および 1 ノードあたりのメッセージ数を示す．メッセージ数は 10 分間の平均値である．エミュレーションシナリオの内容は，6 秒おきに 4000 ノードがオーバレイに参加した後，100 秒間の休みをおいて，2 秒おきに DHT への put を 4000 回繰り返し，再び 100 秒間の休みをおいて，2 秒おきに DHT からの get を 4000 回行うというものである．put と get を行うノードは，シナリオを生成器を用いて生成する際に乱数で決めている．

アルゴリズムを評価，比較するためには，調べたい

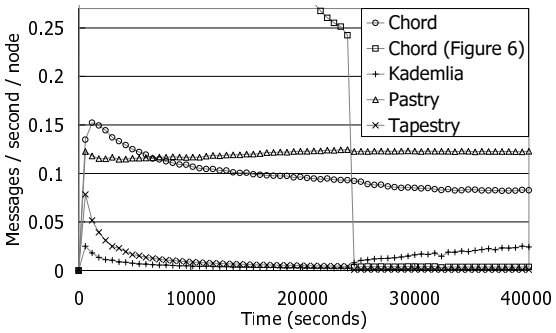


図 7 4000 ノードのエミュレーションシナリオでの 1 秒・1 ノードあたりのメッセージ数（Iterative ルーティングの場合）
Fig. 7 Number of messages per second per node passed between emulated 4000 nodes (iterative routing).

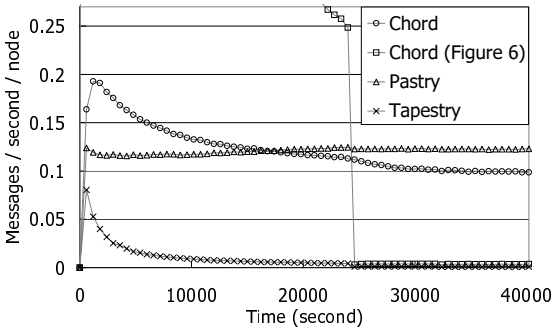


図 8 4000 ノードのエミュレーションシナリオでの 1 秒・1 ノードあたりのメッセージ数（Recursive ルーティングの場合）
Fig. 8 Number of messages per second per node passed between emulated 4000 nodes (recursive routing).

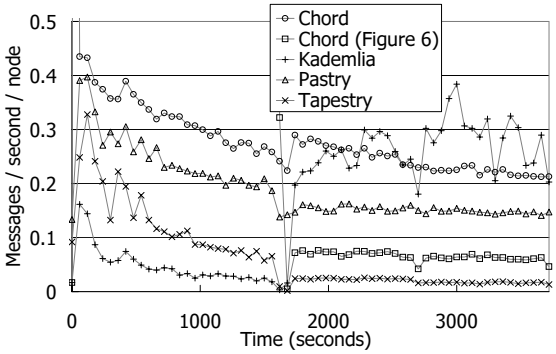


図 9 実機 197 台での 1 秒・1 ノードあたりのメッセージ数（Iterative ルーティングの場合）
Fig. 9 Number of messages per second per node passed between real 197 computers (iterative routing).

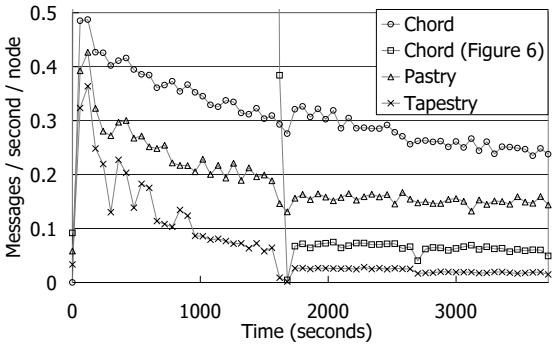


図 10 実機 197 台での 1 秒・1 ノードあたりのメッセージ数（Recursive ルーティングの場合）
Fig. 10 Number of messages per second per node passed between real 197 computers (recursive routing).

内容についてのログを出力させて，エミュレート後に集計すればよい．ノード単独で算出できる値，例えば DHT でのデータ保持量や経路表の収束具合いであれば，この方法で済む．一方，複数ノードが関わる値については，時刻情報を付けてログを出力しておいて後で集計するか，メッセージカウンタ（4.3 節）を使って集計すればよい．図 7 と図 8 を作成するためのデータ採取はメッセージカウンタを用いて行った．

図 7 と図 8 からは，各アルゴリズムの以下の挙動を読み取ることができる．

- Chord と Pastry はオーバレイ保守のために定期的な通信を行うため，Kademlia，Tapestry 実装と比較して，定常状態でのメッセージ数が多い．
- Chord (Figure 6) は，オーバレイ参加時に，経路表を完成させるために多くのメッセージを送受信する．
- Kademlia は，通信相手ノードを経路表に加えるか否かを判断するために，経路表中のノードに対して PING，ACK の送受信を行う．このため，約 24000 秒経過以降の put/get 時に，put/get に必要となるより多くのメッセージを送受信している．

ありふれた PC を用いて 4000 ノードをエミュレートし，アルゴリズムの挙動を調べることができた．また，アルゴリズムの公正な比較を行うためのデータ採取方法を述べ，そのうち，メッセージカウンタを用いる方法を行って結果を示した．

5.3 実ネットワークでの動作確認

本ツールキットが実ネットワークで動作することを示す．実験は，AIST スーパークラスタ（ASC）の一部である 3.06 GHz Xeon 搭載 PC 196 台に加えて，5.2 節で用いた PC 1 台の合計 197 台で行った．PC のネットワークインタフェースはすべて Gigabit イーサネットであり，ASC の 196 台は同一のスイッチに接続されている．ASC と残る 1 台の間には 3 台のルータがあるが，帯域幅は 1 Gbps が確保されている．196 台の OS は Linux 2.4.24 であり，Java 実行系は J2SE

5.0 Update 6 である．DHT シェルを各 PC に 1 つずつ起動し，ネットワーク越しに制御した．

図 9 と**図 10** は 1 秒および 1 ノードあたりのメッセージ数を示したものである．メッセージ数は 1 分間の平均値である．集計にはメッセージカウンタを用いた．制御シナリオは，8 秒おきに 197 ノードがオーバレイに参加した後，100 秒間の休みをおいて，2 秒おきに DHT への put を 500 回繰り返し，30 秒間の休みをおいて，2 秒おきに DHT からの get を 500 回行うというものである．put，get を行うノードはやはり乱数で決めた．

約 200 台という規模の実ネットワークで本ツールキットが動作することを確認できた．

6. ま と め

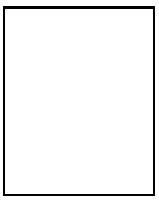
本論文では，ネットワークオーバレイのアルゴリズム研究基盤として開発しているツールキット Overlay Weaver の設計を述べた．これを用いることで，既存のよく知られたルーティングアルゴリズムをたかだか数百ステップで実装できること，また，計算機 1 台上で 4000 ノードという規模のエミュレーションを行ってアルゴリズムの挙動，性質を調べられることを示した．約 200 台の実環境での動作も示した．

同時に，アルゴリズムの実装をルーティング処理の実装から分離する両者間のインタフェースを提案した．アルゴリズム実装の容易さはこの分離の成果である．

参 考 文 献

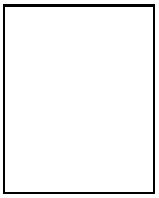
1) Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. and Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications, *Proc. ACM SIGCOMM 2001*, pp. 149–160 (2001).
2) Rowstron, A. and Druschel, P.: Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems, *Proc. Middleware 2001* (2001).
3) Zhao, B. Y. et al.: Tapestry: A Resilient Global-Scale Overlay for Service Deployment, *Journal on selected area in communications*, Vol. 22, No. 1, pp. 41–53 (2004).
4) Maymounkov, P. and Mazières, D.: Kademlia: A Peer-to-peer Information System Based on the XOR Metric, *Proc. IPTPS'02* (2002).
5) Kaashoek, M. F. and Karger, D. R.: Koorde: A simple degree-optimal distributed hash table, *Proc. IPTPS'03* (2003).
6) Rhea, S., Geels, D., Roscoe, T. and Kubiatowicz, J.: Handling Churn in a DHT, *Proc. USENIX '04* (2004).
7) The Bamboo Distributed Hash Table. http://www.bamboo-dht.org/.
8) Chimera and Tapestry. http://p2p.cs.ucsb.edu/chimera/.
9) Khashmir. http://khashmir.sourceforge.net/.
10) Azureus: Java BitTorrent Client. http://azureus.sourceforge.net/.
11) The Official BitTorrent Home Page. http://www.bittorrent.com/.
12) eMule-Project.net - Official eMule Homepage. http://www.emule-project.net/.
13) Overlay Weaver: An Overlay Construction Toolkit. http://overlayweaver.sourceforge.net/.
14) 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, 先進的計算基盤システムシンポジウム（SACSIS2006）論文集, pp. 183–191 (2006).
15) 首藤一幸, 加藤大志, 門林雄基, 土井裕介: 構造化オーバレイにおける反復探索と再帰探索の比較, 情報処理学会研究報告, 2006-OS-103-2 (2006).
16) Rodriguez, A., Killian, C., Bhat, S. and Kostić, D.: MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks, *Proc. NSDI'04*, pp. 267–280 (2004).
17) The MACEDON project. http://macedon.ucsd.edu/.
18) The Mace project. http://mace.ucsd.edu/.
19) Loo, B. T., Condie, T., Hellerstein, J. M., Roscoe, P. M. T. and Stoica, I.: Implementing Declarative Overlays, *Proc. SOSP '05* (2005).
20) p2psim: a simulator for peer-to-peer protocols. http://pdos.csail.mit.edu/p2psim/.
21) Li, J., Stribling, J., Morris, R. and Kaashoek, M. F.: Bandwidth-efficient management of DHT routing tables, *Proc. NSDI'05* (2005).
22) Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J. and Stoica, I.: Towards a Common API for Structured Peer-to-Peer Overlays, *Proc. IPTPS'03* (2003).
23) Castro, M., Costa, M. and Rowstron, A.: Debunking some myths about structured and unstructured overlays, *Proc. NSDI'05* (2005).

(平成 18 年 1 月 26 日受付)
(平成 18 年 5 月 23 日採録)



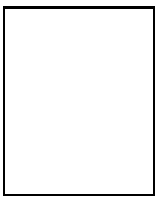
首藤　一幸（正会員）

1996 年早稲田大学理工学部情報学科卒業．1998 年同大学メディアネットワークセンター助手．2001 年同大学大学院理工学研究科情報科学専攻博士後期課程修了．同年産業技術総合研究所入所．2006 年ウタゴエ (株) 取締役最高技術責任者．博士（情報科学）．分散処理方式，プログラミング言語処理系，情報セキュリティ等に興味を持つ．SACSIS2006 最優秀論文賞．日本ソフトウェア科学会，IEEE–CS，ACM 各会員．



田中　良夫（正会員）

1965 年生．1995 年慶応義塾大学大学院理工学研究科後期博士課程単位取得退学．1996 年技術研究組合新情報処理開発機構入所．2000 年通産省電子技術総合研究所入所．2001 年 4 月より独立行政法人産業技術総合研究所．現在同所グリッド研究センター主幹研究員．博士 (工学)．グリッドにおけるプログラミングミドルウェア，グリッドセキュリティ，およびテストベッド構築に関する研究に従事．IC’99，HPCS2005，SACSIS2006 論文賞．ACM 会員．



関口　智嗣（正会員）

1959 年生．1982 年東京大学理学部情報科学科卒業．1984 年筑波大学大学院理工学研究科修了．同年電子技術総合研究所入所．以来，データ駆動型スーパーコンピュータ SIGMA-1 の開発等の研究に従事．2001 年独立行政法人産業技術総合研究所に改組．2002 年 1 月より同所グリッド研究センターセンター長．並列数値アルゴリズム，計算機性能評価技術，グリッドコンピューティングに興味を持つ．市村賞，情報処理学会論文賞受賞．グリッド協議会会長．日本応用数理学会，ソフトウェア科学会，SIAM，IEEE，つくばサイエンスアカデミー各会員．