NAIST-IS-DD0761019

博士論文

時刻属性に基づいた
階層化オーバーレイネットワークの研究

洞井 晋一

2010年2月4日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である。

洞井 晋一

審査委員：

砂原 秀樹 教授	（主指導教員）
山口 英 教授	（副指導教員）
藤川 和利 准教授	（副指導教員）
寺西 裕一 准教授	（大阪大学）

時刻属性に基づいた
階層化オーバーレイネットワークの研究*

洞井 晋一

内容梗概

インターネット上で提供されているサービスはサーバを利用した一対多の通信方式が主流であるが，多対多通信を実現する P2P 型の通信方式が注目されてきている．P2P 型の通信を実現するオーバーレイネットワークの研究分野では分散ハッシュテーブルに関する研究が盛んであり，特にトポロジに制約を加えた構造化オーバーレイネットワークは理論的に高速かつ確実な検索を可能としているため，信頼性の高いサービスを提供できるものとして注目されている．しかし，P2P 型のサービスは利用者の計算機によるサービス提供であるため，頻繁にオーバーレイネットワークへの参加と離脱を繰り返す計算機が存在する．こうした挙動はオーバーレイネットワークの信頼性を本質的に低める要因となるため，現実的にサービスとして利用している事例は少ない．これに加え，集中管理型のサービスと異なり情報が分散されるため，複数の属性を指定した情報の検索が困難である．このため，オーバーレイネットワークの利用は特定の情報を提供するサービスに留まり，汎用的に利用できる情報基盤の構築が難しい．

本論文では時刻属性が多くの情報に付随することに着目し，情報の共有に時刻属性の利用を想定した．従来の手法では時刻属性を用いて共有することで，負荷の不均衡や検索クエリの増大が問題となった．そこで，時間経過に伴って集約されていくように Value をノードへ配置し，範囲検索によるクエリの増大を防いだ．また，提案手法ではオーバーレイネットワークをノードの参加時間に応じて階層

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文, NAIST-IS-DD0761019, 2010 年 2 月 4 日.

化し，Value を時間が経過するに従って安定した階層へと集約する．この集約の効果によって Churn の激しい環境下であっても検索成功率を高いまま維持することができる．

提案手法が動作するアプリケーションを実装し，オーバーレイネットワークのエミュレーションによる実験の結果，提案手法は従来の手法と比べ検索の成功率が約 15%向上した．また，従来のものよりネットワーク上を流れるメッセージ数が約 10%増加したが，性能には問題の無い範囲であり，Value の内容や検索の頻度によっては，ネットワーク上を流れるトラフィックを削減できる．

これらのオーバーレイネットワーク技術に対する貢献によって，実際の P2P 型アプリケーションを想定とした汎用的な情報基盤の構築が可能であることを示した．また，一般の利用者が参加するようなノードの離脱が激しい環境であっても安定したサービスの提供が行えることが可能であることを示し，P2P 型アプリケーションの実現と普及に大きな貢献を果たした．

キーワード

P2P ネットワーク, 多対多通信, 範囲検索

A Study on Hierarchical Overlay Network based on Time Attribute of Information*

Shinichi Doi

Abstract

Data distribution services have been developed with point-to-multipoint communication. Then, the Internet technology offers a multipoint-to-multipoint communication. However, It is too difficult to construct a multipoint-to-multipoint application, because most services on the Internet based on a client/server model. Lots of overlay network, which is multipoint-to-multipoint technology on the Internet, based on DHT (Distributed Hash Table) have been developed in recent years. There are some overlay networks called "Structured Overlay Network" which have a topology restriction. However, we think that DHTs have the following two drawbacks. DHTs do not think about the heterogeneity of the characteristics, which is contains Churn behavior, of nodes, and do not support complex queries.

The purpose of this dissertation is to describe the design and evaluation of our overlay netowork, an id assignment method for a hierarchical overlay network. Our solution forms a multi-ring topology using a session time of a node. The reason why we consider a session time of a node is that heavy users tend to use a high-performance PCs. Key-values are gradually gathered in a high layer ring as time passes, and our solution supports multi-attribute queries with these key-values.

*Doctoral Dissertation, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0761019, February 4, 2010.

The experiment result, which is used Weibull distribution as node session time, shows that our hierarchical overlay network achieve approximately 35% success rate increase. And the number of messages is approximately 10% increase, however it have no practical impact on real networking.

Keywords:

P2P Network, Multipoint-to-multipoint communication, Range query

目 次

図 目 次

表 目 次

第1章　序論

世界では至る所から情報が生成され続けている．情報は人に影響を与えたときに価値があると判断されるため，生成された情報は影響を与える人の数と，与えた行動によって価値が定まるといえる．多人数に影響を与える情報は価値が高く，また人の行動に大きく影響する情報も価値が高い．こうした情報には単一で価値の高いものと，集合体になることによって価値が高くなる情報の二種類がある．例えば気象情報を扱うセンシングデバイスの場合，1つのセンサはその周辺の利用者にしか影響を与えないため，異常気象などの災害時以外は情報の価値が低い．しかし，それらのセンサから得られる情報を集めることで，気候の変動や今後の予測など，多人数の行動に大きく影響する価値の高い情報が生まれる．また，人の考えや行動といった情報なども，集合体となることで社会動向や今後の予測など価値のある情報になり得る．

こうした情報は，生成された場所から利用される人まで伝達されなければ人の行動に影響せず，情報としての価値が発生しない．情報化社会では情報の生成や利用方法が注目されるが，それらを伝達する手段やアーキテクチャも重要である．情報の伝達にはアーキテクチャとして一対多の通信方法と，多対多の通信方法がある．一対多の通信では単一の情報を多人数に伝え，多対多の通信では多量の情報を多人数に伝える．従来のメディアでは主に一対多の通信方式によって情報伝達を行ってきた．これは，技術的な制約と費用対効果によるところが大きいが，成熟した情報化社会では莫大な情報の生成と消費が考えられるため，一対多だけではなく多対多通信の実現が必要である．本章ではインターネットを用いた多対多の通信方式について，現実的な計算機資源とサービス提供を考慮した上で議論を行い，P2P 方式のサービスについて問題点を明らかにする．また，本論文の研究目標となるオーバーレイネットワークを用いた可用性の向上について述べる．

1.1. インターネットと多対多通信

インターネットは中心を持たないネットワークとして誕生した背景がある．単一障害点の発生による可用性の低下を避けるために，経路の冗長化とパケット方式によるネットワークを生成することで，インターネットは地球を覆い尽くさんばかりの拡がりを見せた．インターネットに接続した計算機（ノード）は IP アドレスと呼ばれる世界で固有のアドレスが割り当てられ，ノード間同士の通信を全世界規模で保証している．これは電波を用いた放送メディアとは異なり，多対多の通信を実現できうる技術である．

しかし，インターネット上で情報提供を行うサービスは，サーバを用いたクライアント・サーバ型が主流である．クライアント・サーバ型は 1 台のサーバに対して多くの利用者が接続する，一対多の通信である．インターネットは本来的に中心を持たないネットワークとして設計されているが，情報提供の根幹となるサービスのアーキテクチャはクライアント・サーバ型を利用した中央集権型として設計されている．これは，サービスの提供に費用対効果を考慮した結果であるといえる．クライアント・サーバ型ではサービス提供者が高性能な計算機を用意し，インターネットに接続するだけで，世界中の利用者にサービスを提供できるため費用対効果が高い．

このような一対多の通信に適したクライアント・サーバ型のアーキテクチャを用い，多対多の情報伝達を行うには様々な不都合が生じる．一つにはサーバの稼働率がサービスの可用性に大きく影響することである．クライアント・サーバ型ではサーバが稼働していなければサービスを提供できない．インターネットは，提供者と利用者間の相互通信を可能にしているにも関わらず，情報の提供者がサーバを通じて利用者へと情報を伝達する場合，サーバが動作していなければ実際に情報が伝達されることはない．顕著な例としては電子メールサービスが挙げられる．電子メールサービスはサーバにメールを送信し，サーバが受信者の利用するサーバへ電子メールを転送する．受信者はサーバに新着メールが無いかどうかを確認することで，メールの受信を行う．障害などでメールのサーバが停止すると，送信者と受信者がインターネットに接続し，相互に通信が可能な状況であっても電子メールを送受信することはできない．

このような可用性の低下を補う方法としてはサーバなどのシステムを冗長化する方式が多く用いられている．あらかじめサーバとなる計算機を複数用意し，並列にサービス提供すれば全ての計算機が停止しない限りサービスは停止しない．計算機を複数台用意すれば現実的な確率で全ての計算機の停止することが無く，可用性の高いサービスを構築することが可能である．しかし，冗長化は初期コストや運用コストなどの増加に繋がり，収益の出にくい公共サービスや公益性の高いサービスなどでは冗長化に伴う費用を賄えないことがある．また，インターネットがノード間での通信を保証しているにも関わらず，ノード間の通信における単一障害点としてサーバを設け，それを回避するために冗長化するという方式はサービス提供にかかるコストをいたずらに増加させているといえる．情報化社会で利用される莫大な量の情報交換に対し，こうした本質的な欠陥を是正することは社会全体として大きな利益となり得ることが予想できる．

1.2. P2P 方式による多対多通信とその問題点

本来のインターネットが可能としているノード同士の通信を行うことで，サーバを利用せずに多対多通信を行う P2P 方式のサービスが注目されている．P2P 方式のサービスでは，情報の利用者が情報の提供者からインターネットを通じて直接情報を得る．利用者は情報の提供と利用を同時に行うため，利用者の計算機はクライアントとサーバの両方の役割を果たすこととなる．情報提供者の計算機が障害などで停止した場合，基本的には停止した計算機が提供していた情報は取得できなくなるが，他の計算機が提供している情報は取得することが可能であり，サービスそのものが停止するわけではない．サーバの冗長化と同様にサービス提供を行う計算機が冗長化されるため，可用性の向上を期待することができる．また，利用者の計算機を用いて冗長化を図るため費用が安く，大きな利益の見込めないサービスや公益性の高いサービスなどであっても実現することが可能となる．

オーバーレイネットワークを用いて多対多通信を実現するためには，利用者に情報の検索機能を提供する必要がある．情報の提供者がインターネットに接続されているだけでは，利用者側から発見することができない．情報の提供者を別の

利用者から発見可能にすることがオーバーレイネットワークの機能として求められる．P2P 方式ではクライアント・サーバ型のように中央となる計算機を持たないため，オーバーレイネットワークを構築するノードの協調動作によって情報の検索機能を提供しなければならない．検索の機能は Key-Value Store（KVS）の実現として表すことができる．KVS では，共有する情報の属性などを Key，情報そのものまたは情報へのポインタを Value として，Key の検索に対して Value の取得を行う．利用者は Key の指定による情報の検索を行い，Value を取得することによって情報の提供者，または情報そのものを発見する．

一方で，オーバーレイネットワークでは可用性の向上と引き替えに，この Key を指定した Value の取得という単純な機能の提供ですら整合性を保つことが困難となり，サービスとしての信頼性を保証することが難しい．P2P 型サービスの黎明期にはノードが自由にクエリの送信を行い，構造を持たない非構造化オーバーレイネットワークと呼ばれる手法が用いられた．非構造化オーバーレイネットワークでは Value の検索にクエリを多くのノードへ送信する手法（フラッディング）が用いられたが，このフラッディングによる検索手法ではクエリの増大によるトラフィックの増加や計算機にかかる負荷の増加に加え，検索結果の整合性を保つことが難しいばかりか，検索が終了したかどうかの判別すら難しく，サービスの信頼性を保証することができないといった欠点がある．

検索結果に対する保証は，検索クエリに対する応答ノードを明確に定めることで実現できる．そこで，オーバーレイネットワーク全体が構造を持つように設計された構造化オーバーレイネットワークと呼ばれる手法が提案された．構造化オーバーレイネットワークでは分散ハッシュテーブル（DHT:Distributed Hash Table）と呼ばれる手法を用いて複数の計算機上で KVS を実現する．DHT では Key とノードを一次元の ID に変換し，同じ ID 空間へ写像することでノードに Value を割り当てる．ノードと Key-Value の数が十分に大きく，各計算機にかかる負荷を Value の割当量とみなせばノード間の均等な負荷分散を期待できる．また，オーバーレイネットワーク全体で共有する Value について Key によって割り当てノードを明確に定めることにより，検索に対する応答の保証ができる上，クエリの送信や転送の少ない検索を実現できる．このため，DHT では単一の属性を指定し

た完全一致検索に関しては，検索結果の成功率と整合性を保証することができる．

一方で，複数の属性を指定することや完全一致ではない検索は，その成功率や整合性を DHT によって保証することができない．複数の属性を指定した検索を実現するためには検索を可能とする属性の数だけ ID 空間を増やす必要があるが，構造が複雑になるためアプリケーションの実装や運用が困難であり，また，ノードの参加や離脱（Churn）に起因する検索の成功率低下にもつながる．一次元 ID 空間を用いた DHT で複数属性を指定した複雑な検索を行うためには ID 空間上で共有されている Key について情報を取得し，その検索結果の中から他の属性による検索を行う必要があるが，このとき Key として選択する情報の属性は全ての情報が持ち合わせている属性でなければならない．

これらに加えて DHT では負荷の均等な分散を目指しているため，サーバの冗長化という分野では効果的であるが実際的な P2P 方式のサービスに用いることが難しい．DHT で用いられる ID の変換方法には SHA-1[12] や MD5[34] といった暗号学的ハッシュ関数は Key のノードへの割り当てを均等に分散する．そのため，計算機の性能やネットワークの帯域，接続時間といった特性は考慮されずに負荷を分散する．P2P 方式によるサービス提供は利用者の計算機による構成を想定しているため，計算機の特性を考慮しない負荷分散はシステム全体の性能低下に繋がる恐れがある．特に，churn を繰り返すような不安定なノードであっても他の計算機と同じように Value を割り当てるため，ノードの離脱によって Value が失われ検索の成功率や再現性の低下に繋がる．P2P 方式の問題を以下の通りまとめる．

安定したサービスの提供　P2P 方式では利用者の計算機という不安定なリソースを利用したサービス提供であるため，安定したサービスの提供が難しい．

複雑な検索機能の実現　DHT を用いることで単一属性による完全一致検索の提供は実現されているが，複数属性を指定した検索や，範囲を指定した検索などの一般的な検索機能は提供が困難である．

1.3. 研究方針

本論文では 1.2 節にて述べた問題点に対して，以下の方針による解決を図る．

時刻属性を利用した検索機能　既存の DHT では単一属性による検索機能しか提供されておらず，複数属性の検索機能は提供することが難しい．そのため，検索に用いる属性には汎用的に利用できる属性でなければ検索機能によって実現できる多対多通信のサービスが著しく制限されてしまう．情報提供者が現実世界から取得した情報を提供するのであれば，その提供された情報には現実世界の時刻属性の付随を期待することができる．そのため，本論文では検索機能として提供する単一の属性として情報の時刻属性を扱い，現実世界の情報を利用したサービスでの応用を想定する．時刻を利用した検索機能を提供することにより，情報の検索時に具体的な時刻や時間範囲を指定した検索だけでなく，検索結果を時刻順にソートして表示するといった暗黙的な時刻指定など，利用者による時刻属性の指定は十分に想定することができる．しかし，暗号学的ハッシュ関数を基礎とした DHT では時刻属性を Key にしたオーバーレイネットワークを構築することが難しい．DHT は時刻属性の連続量を破壊するため，時刻の範囲を指定した検索を行えない．また，時刻属性はその連続量の最小値と最大値の定義が難しく，既存の範囲検索手法では実現が難しい．本論文では従来の暗号学的ハッシュ関数に代わり，時刻属性の Key を配置する新しい関数を考案する．新しいハッシュ関数は Key の現在時刻までの経過時間に着目し，動的に Value を再配置することで Key の始点と終点を定めることなく範囲検索を実現できる．

参加時間に基づいた階層化　既存のオーバーレイネットワークを計測した研究により，オーバーレイネットワークを構成する計算機が離脱する確率と，その計算機がこれまで参加した時間には相関があると分かっている．また，実際のサービスではサービス提供者などが計算機をオーバーレイネットワークへ投入し，サービスの安定動作を行うことを想定できる．そこで，参加時間の長い計算機に優先して Key-Value を割り当てることで，Key-Value を喪失する確率が下がり，検索時の成功率と検索結果の再現性を向上させることができる．本論文ではオーバー

レイネットワークを階層化し，更に前述した時刻を指定する関数を利用することで，安定した階層に Key-Value を割り当てる手法を考案した．情報を段階的に安定した階層へと集約することで，Churn による影響を下位層に限定し，検索成功率を向上させることができる．

1.4. 論文の構成

以下に本論文の構成について述べる．2 章では多対多通信サービスに用いられる既存のオーバーレイネットワーク技術を紹介する．既存のオーバーレイネットワーク技術は，トポロジに関する研究や，情報の配置に関する研究，セキュリティに関する研究，アプリケーションの開発などが挙げられる．本論文では特に情報の配置手法を改善することで多対多通信を行う上での可用性向上を目指すため，既存の情報の配置と検索の手法について紹介し，問題点を考察する．3 章では本論文で提案する情報の時刻属性に基づいた構造化オーバーレイネットワーク技術について説明する．特に，情報の配置と検索の方法と，階層化の方式について述べる．4 章では提案手法の実装方法について述べる．オーバーレイネットワークはアプリケーションとの結びつきが強く，汎用的な情報基盤の構築には容易な拡張性を確保することが必要である．実装に際してそれらを考慮した工夫点などを述べる．5 章では提案手法の評価として，実験による検索成功率の結果について述べる．アプリケーションによって利用者の挙動は異なることが想定できるため，利用者の生存時間分布として 3 つの分布を用いた結果を述べる．また，アプリケーションへの適用例として，オーバーレイネットワーク上で高画素写真の共有を可能にするアプリケーションの紹介と，その実装方法について述べる．6 章では多対多通信の考察として，今後の普及に対する考察とそれに向けた課題について述べる．最後に 7 章にて本論文の結論を述べる．

第2章　オーバーレイネットワーク技術

本章では既存のオーバーレイネットワーク技術を紹介し，P2P 型サービスの問題点の本質について考察した上で，特に時刻を Key とした KVS の実現について議論する．

2.1. 構造化オーバーレイネットワーク

P2P 型通信を行うサービスでは，利用者の計算機がクライアントとサーバの両方の役割を果たす．情報はオーバーレイネットワークを構成する計算機（ノード）間で共有され，あるノードが情報の検索をすれば他のノードが協調して検索結果を返す．この検索機能は Key Value Store（KVS）として表すことができる．ノードは共有する情報（Value）のキーワードなど（Key）を用いて共有する（Put 動作）．また，情報の検索は Key を指定することで Value を得る（Get 動作）．オーバーレイネットワークを構成するノードは，主に以下の 2 つの部分によって構成し，KVS を実現する．

- ノードリスト
  オーバーレイネットワークに参加している他のノードに関する情報を格納する場所
- データストア
  KVS を実現するために，Key-Value を保存する場所

利用者はデータストアに Key-Value を追加することで Put 動作を行う．初期のオーバーレイネットワークは，構造に制限が無かったため，ノードは自由にネッ

トワークの構造を構築した．初期に構築された Gnutella[3] や Freenet[13] はその代表である．これらのオーバーレイネットワークは後に「非構造化オーバーレイネットワーク」と呼ばれた．非構造化オーバーレイネットワークでは Get 動作を行うためにノードリストに記載されたノードへクエリを送信し，クエリを受信したノードが更にクエリを転送する「フラッディング」と呼ばれる手法が用いられた．しかし，クエリのフラッディングによる Key-Value の検索はクエリや検索結果によるトラフィックの増加や計算機にかかる負荷の増加に加えて，検索結果の保証を困難にする問題がある．あるノードがデータストアに Key-Value を追加していたとしても，他のノードがそのノードまでクエリを到達できなければ Key-Value を取得することができない．これはオーバーレイネットワークに参加するノードが個々に自由な動作を行い，全体として構造を持たないために引き起こされる問題である．こうした欠点を補うため，ノードの接続関係を明確にし，構造的なネットワークを構築する「構造化オーバーレイネットワーク」と呼ばれる手法が登場した．以下にその手法と特徴，問題点について述べる．

2.1.1 分散ハッシュテーブル

構造化オーバーレイネットワーク上で KVS を実現するために最も多く用いられている手法が分散ハッシュテーブル（DHT:Distributed Hash Table）と呼ばれる手法である．DHT ではデータストアの Key にハッシュ値を用いる．ハッシュ値は暗号などに用いられる SHA-1 や MD5 などのハッシュ関数を利用し，情報の検索対象となるもの（ファイル名，キーワードなど）から算出する．また，ノードリストも同様にハッシュ値を用いる．ノードのハッシュ値（ID) はノードにアクセスするための情報（IP アドレスとポート番号など）からハッシュ関数を用いて算出する．Key-Value は，Key を ID として各オーバーレイネットワークが規定する ID 間の距離計算を行い，距離の近いノードがデータストアに保存する．情報を検索する場合は，検索対象となるファイル名やキーワードをハッシュ関数を用いて ID に変換し，ID に近いノードを探索することで Key-Value を得ることができる．ID 間の距離計算方法や，検索方式によって様々な方式が提案されており，リング構造の Chord[38] を始めとしてツリー構造の Pastry[35], Tapestry[42],

Kademlia[28] や N 次元トーラス形状の CAN[33]，Butterfly 型の Viceroy[26], de Bruijin Graph[14] を構築する Koorde[24] などがある．その中でも代表的な Chord と Kademlia について以下に詳細を述べる．Chord は分散コンピューティングに広く用いられているコンシステント・ハッシュ法 [25] を DHT に拡張したものであり，Kademlia は Churn の激しい環境化であっても動作する DHT として知られている．また，DHT の問題点についても説明する．

Chord

Chord では ID の距離が対称ではなく，ID_1 から ID_2 までの距離 d は以下の式 2.1 ように表すことができる．尚，ID は便宜上 1 未満の正数を用いるが，実際の実装では SHA-1 などに用いられる 160bit の空間が用いられる．

$$d = \begin{cases} (ID_2 - ID_1 + 1) \bmod 1 & \text{if } ID_1 \neq ID_2 \\ 1 & \text{if } ID_1 = ID_2 \end{cases} \tag{2.1}$$

図 2.1 に例を示す．Chord では図のように ID 空間の始点と終点を接続し，時計回りの円として表される．図中の A から E はノードを表し，黒四角は Value を表す．Value の Key は 0.121 である．このときノード間の距離は時計回り方向への距離として定まる．また，Chord ではその ID と最も近い ID のノードを successor と呼び，最も遠い ID のノードを predicessor と呼ぶ．Value を保持するノードは Key から最も近いノードである successor が保持する．図 2.1 では，0.121 の ID とノード B との距離は 0.144 であり，ノード A との距離は 0.767 となる．ノード B より近い ID のノードはいないため，ノード B が 0.121 の successor となり，Value を保持するノードとなる．. また，各ノードはノードリストに自身の ID を Key としたときの successor と predicessor を記載し，ノード全体をリング状に接続する．Chord では Key を検索するときにノードの ID から Key までの距離に応じて successor か predicessor に検索クエリを送信すれば担当となるノードまで検索クエリを必ず到達させることができる．しかし，ノードの数が増えれば担当している ID のノードまで検索クエリを中継するノードが増えるため，検索にかかる時間がが非常に長くなる．一回の検索クエリが通過するノード数をホップ数として，

リング上にノードが均等に存在しているとすれば，全体のノード数を N として最大のホップ数は $\frac{N}{2}$ となり，平均は $\frac{N}{4}$ となる．これは参加ノードである N に正比例するため，仮に 100 万ノードの参加を考えれば現実的な時間で検索が終わらない可能性がある．

そこで Chord では式 2.2 に示す Key の successor をノードリストに加える．Chord ではこのノード群を特に finger table と呼ぶ．

$$Key = (ID + \frac{1}{2^n}) \bmod 1 \tag{2.2}$$

ここで，n は 1 以上の整数であり，Key の predicessor が finger table を作成するノードの ID になるまで繰り返す．また，ID は finger table を作成するノードの ID である．finger table に記載されるノードの例を図 2.2 に示す．図 2.2 の例では，ノード A の finger table に距離が 0.5 離れた Key を保持するノード D，0.25 離れた Key を保持するノード C，0.125 離れた Key を保持するノード B が記載される．ノードが Key を探索するにはノードリストを参照し，Key の successor にクエリを送信する．クエリを受け取ったノードはノードリスト内から Key の successor を取得する．successor が自身の ID であれば，データストア内から Key-Value を検索し，検索結果をクエリ元のノードに送信する．自身の ID でなければ successor のノードにクエリを転送するか，successor のノード情報をクエリ元のノードに送信する．この finger table を用いれば，ホップ数が $O(\log_2 N)$ となるため，参加するノード数が増えてもホップ数はほとんど変わらずに検索機能を提供できる．

Kademlia

Kademlia では ID 間の距離は排他的論理和となり，ID_1 と ID_2 の距離は式 2.3 のように対称的に定義される．尚，ID には 160bit の空間が想定されている．

$$d = ID_1 \otimes ID_2 \tag{2.3}$$

3bit の場合の例を図 2.3 に示す．ID が 101 の場合，距離が最も近いノード C が Value を保持する．各ノードは図 2.4 に示す，k-bucket と呼ばれるノードリストを用いてノードの探索を行う．k-bucket の各行はその行の距離に該当するノード

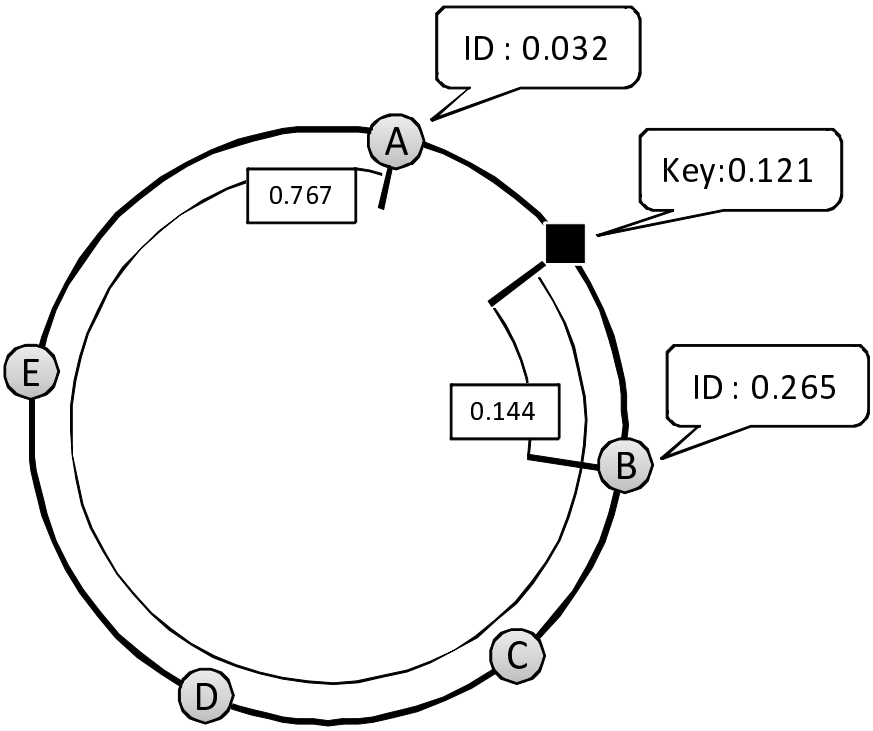

図 2.1 Chord の Key-ID 間距離

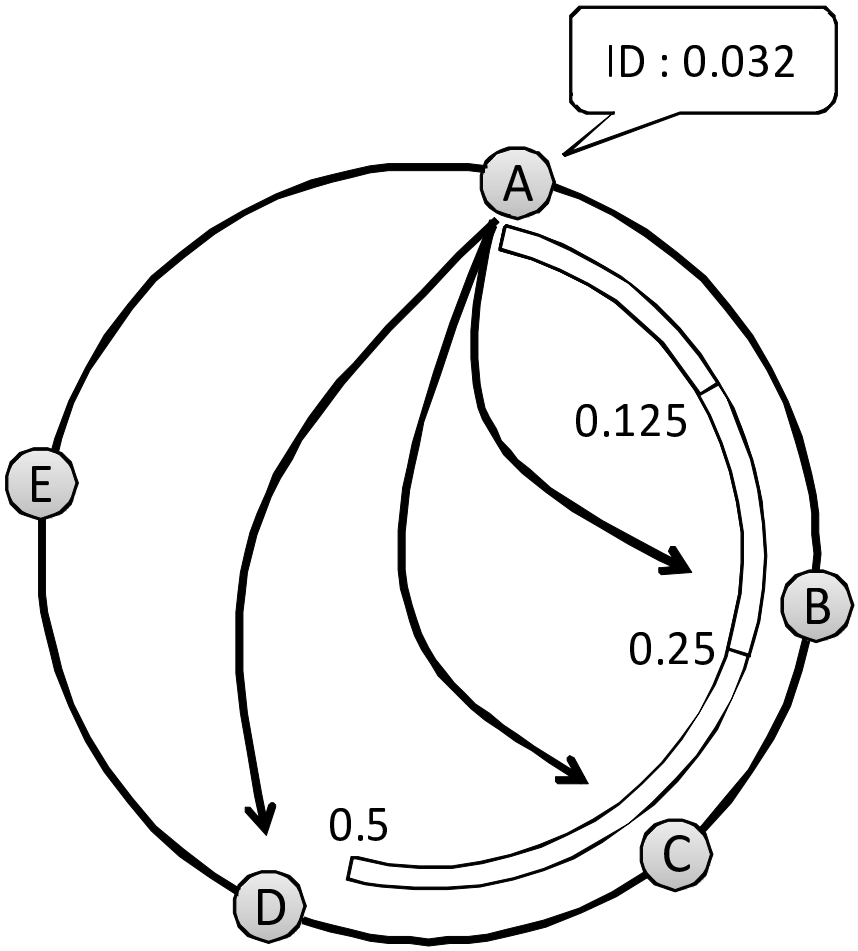

図 2.2 Chord の Finger Table

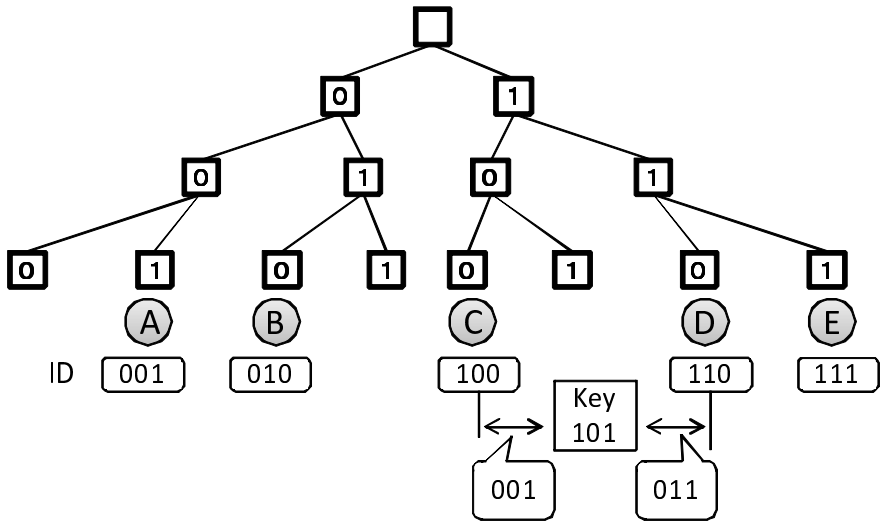

図 2.3 Kademlia の概要図

を参加時刻の古い順に整列したリストとなる．これは文献 [36] に示されているように，参加時間の長いノードほどその後も参加している確率が高いためである．ここで，リストの長さは最長で k とする．Key の担当ノードを探索するには，まず Key と ID の距離を計算し，k-bucket から同じ距離の行に記載されたノードを k 個取得する．また，取得したノードは ID に近い順にソートする．これらの k 個のノードに対して Key の問い合わせを行うことで，Key に最も近い ID のノードを発見することができる．

Kademlia の特徴は距離が対称になる点にある．例えば図 2.3 のノード C と D の距離はどちらのノードから計算しても 010 である．この性質のため，クエリを受け取ったノードがクエリの送信元ノードをノードリストに追加することができ，メッセージの転送などにノードリスト内の情報を有効に使うことができる．また，メッセージのやりとりがされている限り，ノードリスト内のノード情報を最新に保つことが容易になる．

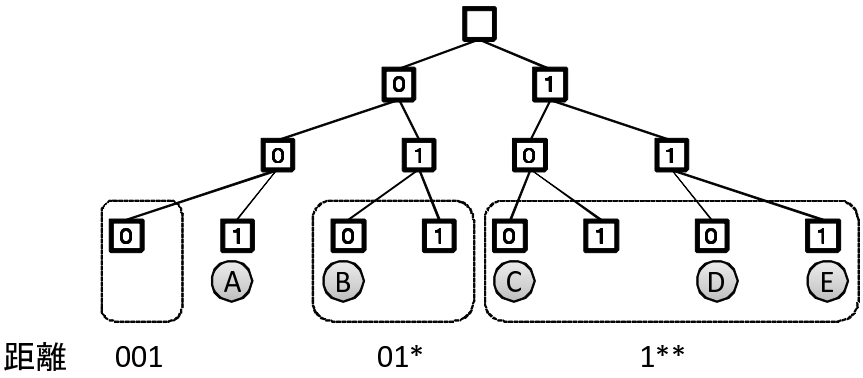

図 2.4 k-buket の例

問題点

これら DHT に共通する大きな問題点は，ハッシュ関数が Key 間の関連性を破壊するため，Key の完全一致検索しかできない点 [22] である．地理位置情報や時刻情報のような連続量の利用を考えれば，Key の完全一致検索よりも範囲を指定した検索を用いる方が自然である．例えば地理位置情報として，緯度と経度を正確に指定できることは少ない．また，時刻情報も秒数などの単位まで指定することは考えにくく，2009 年の 10 月といった検索は，暗黙的に 2009 年 10 月 1 日 0 時 0 分から 2009 年 11 月 1 日 0 時 0 分といった指定である．こうした Key の範囲検索は DHT では困難となる．例えば時刻情報を Key として利用した場合を考える．2009 年 10 月 10 日の Key と 2009 年 10 月 11 日の Key はハッシュ関数を用いることで全く異なる ID となるため，別のノードに割り当てられる可能性が高い．そのため，2009 年 10 月 1 日から 2009 年 11 月 1 日までの Key を範囲検索した場合，全ての日付となる 31 個のクエリを送信する必要がある．また，これは日付を Key の最小単位に用いているため 31 個のクエリで良いが，1 時間を最小単位として用いるのであれば 744 個のクエリを送信しなければならない．クエリの増加は構造化しないオーバーレイネットワークと同様に検索結果の保証を行うことが難しくなるため，構造化オーバーレイネットワークの利点を活かすことができなくなる．

2 つめの問題点として，ノードとなる計算機の特性を考慮できないことがある．

Value が割り当てられるノードはハッシュ関数によってのみ定まるため，ノードの特性は考慮されない．特に，ノードは離脱することで割り当てられた Key-Value を喪失し，検索の成功率が低下するが，Value の割り当てにノードの離脱頻度などは考慮されない．そのため，Churn の激しい環境下では検索の成功率が著しく低下し，サービスの可用性として大きな問題となる．

2.1.2 範囲検索機能の提供

DHT を用いた範囲検索の手法としては，前述したように範囲検索に用いられる最小単位をあらかじめ定めておく手法がある．Gupta[20] の手法では，クエリに用いられる最小単位からハッシュ値を算出し，既存の DHT に Put しておくことで範囲検索を実現した．しかし，最小単位の定め方が難しく，大きくすれば負荷の分散が行えず，小さすぎると範囲検索に用いるクエリ数が多くなる．そのため，暗号学的ハッシュ関数を用いないことで範囲検索を行う方式が多く，Value の配置に連続性を破壊しない関数が用いられる．範囲検索ができるオーバーレイネットワークの 1 つとして，Skip Graph[10] について以下に述べる．

Skip Graph

Skip Graph[10] は skip lists[32] を用いた分散型のデータ構造である．図 2.5 に Skip Graph の例を示す．この例でも Chord と同様に 0 から 1.0 の値を ID として用いる．図ではノード C から見た Skip Graph の構造を示す．Skip Graph では隣接ノードが階層毎に決まる．Level 0 では全てのノードが ID の順に整列し，ノード C はノード B とノード D が隣接ノードとなる．これは Chord と同様である．Level 1 では Level 0 のノードから一定の確率によってノードが選出される．ここでは 50%の確率で Level 1 のノードとなる．図の例ではノード A とノード D が隣接ノードとなる．同様に Level を上げ，ノードの数を減らすことで，最終的にはノード C しか存在しなくなる．図における 0.97 の Key を探索する場合，ノード C は最も高い Level から順に Key の担当となるノードを探索し，Level 0 になるまで続ける．図の例では Level 3 でノード D を発見し，Level 0 でノード E を発見す

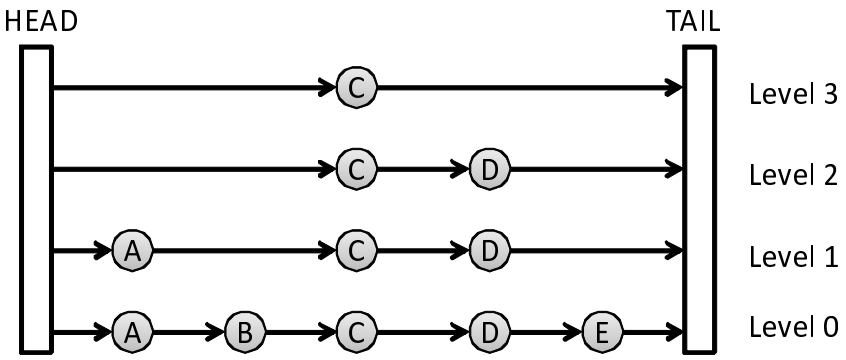

図 2.5 Skip Graph の概要

ることができる．各階層が Chord の finger table のエントリのように振る舞うため，高速な検索を実現することができる．

Skip Graph では Value の配置に暗号学的ハッシュ関数を用いないため，Key の範囲検索が可能である．しかし，Skip Graph では負荷が均等にならないことが指摘されている [19]．SkipNet[21] では暗号学的ハッシュ関数を用いて負荷の分散方法を提示しているが，この方法では範囲検索ができない．

範囲検索と負荷分散の関係

範囲検索を可能としているものは多々あり，Chord と同様のリング構造を多重化する Mercury[11] やオーバーレイネットワーク上に B 木を構築する BATON[23] などがある．また，検索の Key として地理位置情報の利用を行うセンサーネットワークの分野でも範囲検索の提供は必要となるため，Mill[27] や Skip Graph を用いた LL-net[43] などが提案されている．地理位置のような 2 次元空間での検索は CAN を用いて空間充填曲線を利用した方式 [8], Chord を用いて空間充填曲線を利用した方式 [37], Chord と 4 分木を用いた方式 [40] などがある．どの手法でも範囲検索を実現するために用いている方法は Key の連続性を破壊しないハッシュ関数を用いる手法である．しかし，Key は偏ることが想定できるため，暗号学的ハッシュ関数を用いないのであれば負荷分散を考慮しなければならない．本来，範囲検索を効率的に実現するということは，少ないクエリによって範囲検索の結

果を得られるということである．そのため，Value がノードに集中している場合，範囲検索のクエリ数が少なくなり，範囲検索は効率的に実現可能となる．しかし，ノードへ Value を集中させることは，クエリをノードへ集中させることに繋がり負荷分散を実現することができない．逆に，負荷分散は多くのノードへ Value を分散させることで実現するため，範囲検索に使うクエリ数は増加することになる．このように範囲検索と負荷分散は二律背反の状態にあり，両方を実現することは困難である．

Mercury や BATON では負荷の増加した部分に対して構造を再構築することで負荷の分散を行う．負荷の偏りが発生した時にノード間による協調動作によって構造を再構築し，動的に負荷を分散することで汎用的に連続量を扱うことができる．しかし，構造の再構築は Churn の激しい環境下では困難となる上，負荷の偏りが頻繁に発生する状況では構造の再構築に伴って検索成功率が著しく低下する．HotRod[30] では複製の配置を増やすことで全体の負荷を均一にすることが提案されている．しかし，複製の配置は Value となる情報が大きくなれば，トラフィックやノードにかかる負荷が大きくなるため好ましくない．以上のように，汎用的に利用できる範囲検索は未だに実現が難しい．

2.2. P2P 型アプリケーション

本節ではオーバーレイネットワーク技術を利用した P2P 型アプリケーションの利用実態についてその特徴を述べ，P2P 型アプリケーションが本質として持っている問題点について議論する．

2.2.1 利用者参加型アプリケーション

オーバーレイネットワークは基本的に利用者参加型のアプリケーションにおいて，その真価を発揮する．一部ではオーバーレイネットワークをサーバの冗長化として利用することが考えられており，例えば Amazon の Dynamo[15] ではリング状のオーバーレイネットワークが構築されている．しかし，サーバの冗長化を行う分野ではノードの離脱や参加といった不規則な挙動へ対応する必要がほとん

どない．また，ノードの数が数百から数千の規模であれば例えばChordで用いられているfinger tableのような最適化は現実的には必要なく，全てのノード情報をノードリストへと追加することができる．Dynamoでは全ノードがフルメッシュにリンクを持つ手法が用いられているように，ノード数が少なく，かつ離脱や参加が頻繁でないのであればオーバーレイネットワークの特徴を十分に活かすことが出来ない．一方で利用者参加型のアプリケーションでは利用者の挙動によってノードが参加や離脱を行い，不規則に挙動する．また，サービスによっては数万から数百万のノードが参加するため，リンクを最適化するなどの手法が必要となる．

2.2.2 ノードの参加時間分布

ノードは利用者がアプリケーションを利用する際にオーバーレイネットワークへと参加し，利用後にアプリケーションを終了することでオーバーレイネットワークから離脱する．この参加から離脱するまでの時間をノードの参加時間と呼ぶ．参加時間は利用者のアプリケーション利用方法に依存し，アプリケーションの種類やサービスの提供方法にも依存する．例えばSkype[5]のように，常に起動しておくことが求められるようなアプリケーションでは，利用者の計算機が起動している間はオーバーレイネットワークへの参加を期待できるが，BitTorrent[1]のようにダウンロードすることが目的となるサービスでは，ダウンロードを終えるとアプリケーションを終了するような利己的な利用者を考慮しなければならない．

論文[36]では既存のP2P型アプリケーションとしてGnutellaとNapsterに参加しているノードの参加時間を調査している．調査の結果によれば大半のノードは参加時間が短く，一部のノードだけがオーバーレイネットワークに長く参加している．また，論文[39]では既存のP2P型アプリケーションとして，Gnutella, Kad, BitTorrentを調査し，セッション時間の分布傾向を報告している．論文[39]では全てのP2P型アプリケーションにおいて，参加時間を横軸，ノード数のCCDF（Complementary Cumulative Distribution Function）を縦軸に表したときに対数正規分布，またはWeibull分布に近似することが示されている．

これらの調査から，アプリケーションによってノードの挙動は異なるものの，参加時間の分布が対数的に減少していくという挙動は共通するものと予想でき

る．どのようなアプリケーションやサービスをであっても，利用者の大半が数分から数時間程度のノードとなることは避けがたく，数日から数週間の長い参加を行うノードは非常に少ない．現実的に商用などのサービスを行うためには安定したサービス提供を行う必要があるが，大半のノードは挙動が安定しないため，安定したサービスとして成立しえないことが予想される．

2.3. 想定環境

本節では本論文の想定環境について述べる．多対多通信を実現するオーバーレイネットワークに関する研究は数千から数百万といった非常に多くのノードによる構築が想定されている．これは，ノード数の少ない多対多通信はサーバを中継ノードとするような，クライアント・サーバ型を問題なく用いることができるためである．

2.3.1 一般利用者の計算機とサービス提供者の高性能計算機の利用

本論文で想定するオーバーレイネットワークを構成するノードは一般利用者の計算機資源と，サービスの提供者が用意する高性能計算機とする．一般利用者の計算機資源によってのみ構成するオーバーレイネットワークでは，サービスの品質やセキュリティに対する保証，著作権等の法的な問題に対処することが難しい．また，オーバーレイネットワークを構築するプログラムの配布や，参加のために必要な初期ノード情報の配布などを考えるとサービス提供者が高性能計算機を用意し，サーバとして稼働させることは十分に想定できる．また，オーバーレイネットワークはプログラムを配布する形でサービスを提供せざるを得ないため，改竄されたプログラムによる不正ノードの参加を考慮すれば，不正ノードを除外するといった面でもサーバのようなサービス提供者の用意する高性能計算機が必要となる．

P2P 型のサービスがクライアント・サーバ型のサービスと違う点は，利用者の計算機資源を有効に利用するかどうかである．近年の半導体技術の進歩によって，一般利用者の計算機資源は利用用途に対して過剰性能であることが多い．それら

の性能を引き出すことで，クライアント・サーバ型では実現できないようなサービスを実現できる余地がある．例えば利用者の計算機能力を，蛋白質構造の解析 [2] や地球外知的生命体の探査 [4] に利用するものがあり，利用者の計算機性能は大きな可能性を秘めている．また，BitTorrent[1] ではファイルをダウンロードするために，すでにダウンロードした利用者から並列ダウンロードすることでサーバの負荷軽減と高速なダウンロードを提供している．これは利用者の上り回線を有効に利用していると言える．計算機資源を有効に活用し，クライアント・サーバ型では実現出来ないサービスを提供し，参加するサーバにかかる負荷や費用を下げることがオーバーレイネットワークを構築する最も有益な点である．以上のことから，一般利用者の計算機資源と，サービス提供者の用意する高性能計算機と利用者の計算機資源によってオーバーレイネットワークを構築することがサービスを提供する現実的な環境であると想定する．

2.3.2 ノードの頻繁な参加と離脱

オーバーレイネットワークに参加するノードは，随時参加し，通知無しに離脱するものとする．ノードとなる計算機は様々な理由によってオーバーレイネットワークから参加と離脱を行うことが想定される．サービス提供者の用意するサーバは，故障や障害，またはメンテナンスによって計算機がオーバーレイネットワークから離脱することがある．一般利用者の計算機は利用するときにだけ起動させることが多く，利用が終わるとオーバーレイネットワークから離脱する．また，OS にはスリープ機能やサスペンド機能が搭載されているものが多くなり，ラップトップ型の計算機だけでなくデスクトップ型の計算機でもスリープやサスペンドの状態にすることが多い．スリープやサスペンドの状態になった場合，ノードとしての機能を果たさなくなるため，オーバーレイネットワーク上から離脱したと見なされる．スリープやサスペンド機能による離脱は，その頻度が多くなるといった問題点と，離脱に際して特別な処理を行うことができない．また，障害や無線 LAN の圏外になるなど，突然オーバーレイネットワークから離脱する場面は多い．何らかの処理を行った上で離脱することは既存の技術としても比較的容易であるため，本論文では離脱の際に特別な処理を行わないものとする．

また，参加時間の長いノードほど離脱する確率は低くなると想定する．これは Kademlia に実装されている k-bucket の実装や，文献 [36] によっても示されている．また，既存のオーバーレイネットワークを計測した論文 [39] によれば，生存時間の分布が Weibull 分布に近くなることが指摘されている．アプリケーションの種類によっては分布の種類が異なるが，利用者の大半は数時間以下の利用になることは十分に想定できるため，それ以上参加しているノードは，サービスのヘビーユーザやサービスの提供者による計算機であると予想でき，離脱の確率は低くなると想定できる．

2.3.3 単一属性による検索

現実的なサービスでは利用者は複数の属性を指定した情報の検索を行う．ファイル共有のようなサービスであってもファイルの種類や作成時刻，ファイル名に含まれる文字，ファイルの作成者などの複数の属性である．また，センシングデータのような情報であればセンサの種類，地理位置情報，時刻，センサの精度などの属性が挙げられる．しかし，オーバーレイネットワークに用いられる Key 空間は 1 次元であることが多く，一つの属性による検索しか扱えないものが多い．次元数を増やすことはノードの隣接ノードを増やすことに繋がり，特に Churn の激しい環境ではトポロジの維持を困難にするためである．一次元の ID 空間を用いて複数属性を指定した検索を行うには最初に ID 空間上での検索を行い，その検索結果の中から他の属性による検索を行えばよい．例えばセンシングデータの場合では地理位置情報による ID 空間を作成し，地理位置による検索を行った後で取得したデータの中から他のセンサの種類や時刻といった情報を用いた検索を行う．もしくは，地理位置情報によってノードを発見し，そのノードへクエリとしてセンサの種類や時間を送信し，結果を取得する．しかし，特定の利用を想定せずに，汎用的な情報基盤を目指すのであれば最初の検索に用いる Key はどの情報も属性として保持しており，更に利用者が検索に用いる必要がある．

2.3.4 負荷分散

本論文では負荷はKey-Valueの割当量によって定まると考える．一般的に計算機にかかる負荷は，計算能力やストレージ，ネットワーク帯域の占有率などを指す．Valueは非常に小さな情報であるため，多量のValueを保持してもストレージを圧迫するほどの問題は起きにくい．しかし，構造化オーバーレイネットワークでValueの割当量が多いということは，検索クエリの宛先ノードになる確率が高いということにもつながる．そのため，Valueの割当量が多いほどクエリの到達する確率も高くなり，ノードの計算能力とネットワーク帯域を占有するため，負荷が高くなると考えられる．

DHTや多くのオーバーレイネットワークで考えられている負荷分散はノードへ均等にValueを割り当てることである．しかし，ノードへ均等に負荷を割り当てる行為は数千から数百万のノードが参加する環境には適さない．本論文が想定しているように，非常に多くのノードが参加するオーバーレイネットワークにおいてノードの性能が等しくなることはあり得ないため，サーバとなる高性能計算機や一般の計算機が参加しているネットワークでは負荷を均一にすることは望ましくない．一方で，ノードの性能に見合った適切な負荷分散を行うことは困難である．オーバーレイネットワークを構成する計算機はそのサービスの内容やサービス提供者の投入する計算機の性能・数によって大きく異なる．こうした計算機にかかる負荷は計算機上で動作するノードの数の増減によって調整することができるが，多くのノードを一台の計算機上で動作させることは，離脱したときを考慮すると好ましくない．ノード数による調整は行うとしても，性能の高いと思われる計算機に負荷を割り当てる性質がオーバーレイネットワークには求められる．また，予測可能な負荷の割り当て量にすることで，利用者がノード数の増減によって負荷の調整を行い易くすることも求められる．

以上をまとめると，想定とするオーバーレイネットワークでは負荷をノードに均等に割り当てる手法ではなく，性能の高いと思われる計算機に対して負荷を多く割り当てる性質と，それを予測できることによる利用者による負荷の調整の容易さであるといえる．

2.4. 既存研究に関する議論

本節では既存研究に関して本論文での想定環境を考慮したときの問題点について議論する．

2.4.1 時刻属性の Key への利用

既存のオーバーレイネットワークを用いて時刻属性を Key とするような KVS の構築を考える．暗号学的ハッシュ関数を用いた DHT では時刻属性の連続性を無視して Key をノードへと割り当てるため時刻の範囲を指定した検索が困難になる．また，Key の連続性を考慮した範囲検索の可能なオーバーレイネットワークでは Key の取り得る範囲として始点と終点を定める必要があり，時刻のように始点と終点を定めることが困難な属性には向いていない．例えば Key の始点を 1990 年，終点を 2010 年に定めた場合，1990 年より以前の Key や 2010 年以降の Key を使用できず，更に現在時刻よりも未来の Key は利用されないため，Key-Value の割り当てが不均衡になる．また，Key を 0 時から 24 時といった形で周期的に定義した場合，Key の出現頻度が一定であれば Key-Value の分散を期待できるが，1 日を越えた範囲検索は全ノードへクエリを送信しなければならない．このように Key として時刻属性の周期を用いた場合，Key-Value の割り当てを均等にすることと，範囲検索に必要なクエリ数を減らすことは二律背反の状態にあり，双方を満たすことができない．オーバーレイネットワークを構成している計算機の数が少なく，クエリ数が多くとも検索結果を保証できるのであれば負荷分散の観点から Key の周期を短くすればよい．しかし，想定環境で述べたように参加する計算機の少ないオーバーレイネットワークは考えにくい．また，Key の周期を長くすることで範囲検索時のクエリ数を削減できるが，Key-Value の割り当てがノード毎で大きく異なってしまう．また，Mercury や BATON のように負荷の偏りを検知することで動的に再構築する方法では時間の経過に従って頻繁に構造を再構築する必要があるため，時間の連続量には向いていない．論文 [9] では時刻や位置情報が階層構造を持つことに着目し，オーバーレイネットワークの構造に取り入れることが提案されている．しかし，具体的に時刻属性の特性をどのように扱

うかは議論されていない.

一方で利用者による時刻を用いた情報の検索を考えたとき，人は古い情報に対する検索ほど正確な時刻を覚えておらず，検索に用いる時刻の範囲が広くなることが想定できる．人間の記憶は古い情報ほど劣化していくため，時刻属性も同様に劣化し，曖昧になっていく．例えば何かの事件が発生した場合，その日は事件の発生した正確な時刻まで記憶しているが，時間が経つにつれて何日の事件，何月の事件，何年の事件といった形で劣化していく．このような新しい情報と古い情報での検索の特性が異なることも留意すべきである.

2.4.2 Churnによる影響

Churn がオーバーレイネットワークに与える影響は以下の二つがある.

- オーバーレイネットワーク上の経路の喪失
- Put された Key-Value の喪失

例えば Chord の場合，ノードの離脱によってリング構造が途切れ，メッセージの転送に失敗する場合がある．また，Put された Key-Value をノードが保持したまま離脱した場合，オーバーレイネットワークが構造を再構築したとしても Put を行ったノードがもう一度 Put しない限り，Get によって情報を得ることはできない.

DHT や多くのオーバーレイネットワークで考えられている負荷分散はノードへ均等に Key-Value を割り当てることである．しかし，負荷の均等な割り当ては Churn の発生しやすいノードに対しても他のノードと同様に Key-Value を割り当てるため，ノードの離脱によって Key-Value を喪失し，ユーザが Key-Value の検索に失敗する原因になる．Kademlia では k-bucket をノードの参加時間順に並べ変えることで Churn に強いオーバーレイネットワークを構築しているが，これは経路の喪失を防いでいるのみであって Put された Key-Value の喪失を防いでいるわけではない.

既存の耐 Churn の手法は主に Key-Value の複製を Put することで対処している．しかし，複製の配置はノードにかかる負荷を増やす恐れがある．1 つの Key-Value

に対して複製を 10 個配置すると，1 ノードあたりの Key-Value は 10 倍に増える．また，複製の配置を許可することはノードの ID と Key から算出した ID が最も近いノードでなくても Key-Value を保持できるため Key-Value の改竄などが容易になる．

2.5. まとめ

既存のオーバーレイネットワーク技術では，検索結果の保証ができる構造化オーバーレイネットワークが注目されている．しかし，暗号学的ハッシュ関数を用いた DHT によって構築されたオーバーレイネットワークでは，Key の完全一致検索しか提供できないため，現実的に利用できる Key は限定される．これに対して範囲検索を行える手法がいくつか提案されているが，負荷を均等に分散しつつ範囲検索のクエリ数を削減することは困難である．

本論文では多対多通信を実現するために，オーバーレイネットワークの研究分野に対して以下の研究目標を設定する．

時刻属性による検索　情報の時間属性を利用することで，様々な情報を同時に共有することが可能となる．既存のオーバーレイネットワークでは，Key の始点と終点を定める必要があり，時刻属性を用いた場合に負荷分散と範囲検索を両立することが出来なくなる．

検索成功率の向上　Churn の激しい環境では，ノードの参加や離脱の特性を考慮しない Value の配置は検索成功率の低下を招く．実際のサービス展開にとって，検索成功率の向上は必須である．

第3章　提案手法

本節では提案手法について述べる．尚，本節ではID空間として0から1の値を用いる．実際にはChordなどと同様に160bitのID空間を用い，0から1のID空間を0から2^{160}のID空間に相互変換する．ノードのIDはChordなどの手法と同様，IPアドレスなどを適切なハッシュ関数を用いてIDへ変換を行う．

3.1. 概要

時刻属性をKeyとした際に問題となるのは，ハッシュ関数の出力するIDである．暗号学的ハッシュ関数の場合はKey間の関連性を破壊し，連続量をそのまま保存する方式はKeyの始点と終点の定め方によって負荷の偏りを生じさせる．そのため，ハッシュ関数を改良することによって時刻属性をKeyとして扱い，オーバーレイネットワークのトポロジなどは特に変更が必要無い．本論文で提案する手法はChordと同様のリングトポロジを用い，ハッシュ関数の改良とリングの階層化によって既存の問題点を解決する．

時刻属性をKeyとしたKVSを実現するために，本論文ではハッシュ値であるKeyの算出に新たな関数$H_T(t)$を提案する．$H_T(t)$では時刻属性であるtを引数に取りIDを返す．$H_T(t)$はtの現在時刻までの経過時間によってIDが異なる．出力されるIDは経過時間が長くなるほどID間の距離が短くなるため，特に古い情報を検索するときに時間の範囲を拡げてもID空間上での範囲が拡がらず，クエリの数を抑えることができる．

提案手法ではChordと同様のリング状オーバーレイネットワークを構築し，ノードの生存している確率を基にした階層化を行う．階層はLで表し，Lが大きいほど上位の階層，Lが小さいほど下位の階層とし，上位の階層ほど生存している確

率が高いノードによるオーバーレイネットワークとする．上位階層のノードへ優先的にKey-Valueを割り当てることで，検索クエリの成功率を向上させることができる．しかし，最も生存確率の高い最上位層のノードにのみKey-Valueを割り当てることは，そのノードが処理をしなければならないクエリ数が増えるため，負荷分散の点から好ましくない．そのため，ノードにはKey-Valueを割り当てられる上限があるものとし，オーバーレイネットワークの階層化とKey-Valueの割り当てを行う必要がある．また，ノードが参加や離脱を繰り返した場合に構造が分断する恐れや，リングが多重に生成される恐れがあるため，階層間での整合性などを取る必要がある．そこで，ある階層に属するノードはそれより下位の階層にも必ず属するように階層化を行う．このような階層化を行うことでリングの多重化や分断を防ぐことが期待出来る．一方でこのような階層化は階層の数が多くなるにつれて上位の階層は属する階層が多くなり，オーバーレイネットワークを維持するための動作などが増える．そのため，階層数は必要以上に増えないように設計しなければならない．

図3.1に概要図を示す．図はAからHまでの8ノードによるオーバーレイネットワークを表している．ここでTは時刻を表す．図のノードAは4階層目（$L = 4$）に参加していることを表しているが上位の階層に属しているノードはそれより下位の階層にも属しているため，$L = 3, 2, 1$にも参加している．IDは0から1の値をそれぞれの階層で時計回りに定義しており，ノードAは$ID = 0.25$である．図中のコンテンツは$T = 165$からIDとLを算出し，オーバーレイネットワークに参加しているノードに割り当てられる．例では$ID = 0.735, L = 3$であるため，ノードBがコンテンツを保持する．検索クエリは検索対象となる時刻からIDとLを算出し，適切なノードへとクエリを送信する．検索クエリには時刻の範囲を指定でき，例では$T = 150$から$T = 200$の間のコンテンツを検索している．それぞれをハッシュ関数を用いてIDとLに変換し，$L = 3$の$ID = 0.700$から$ID = 0.815$の範囲となり，管理しているのはノードBであるため，クエリはノードBへ転送される．ノードBはクエリを受け取ると検索結果を検索元のノードへと返信し，検索を終える．

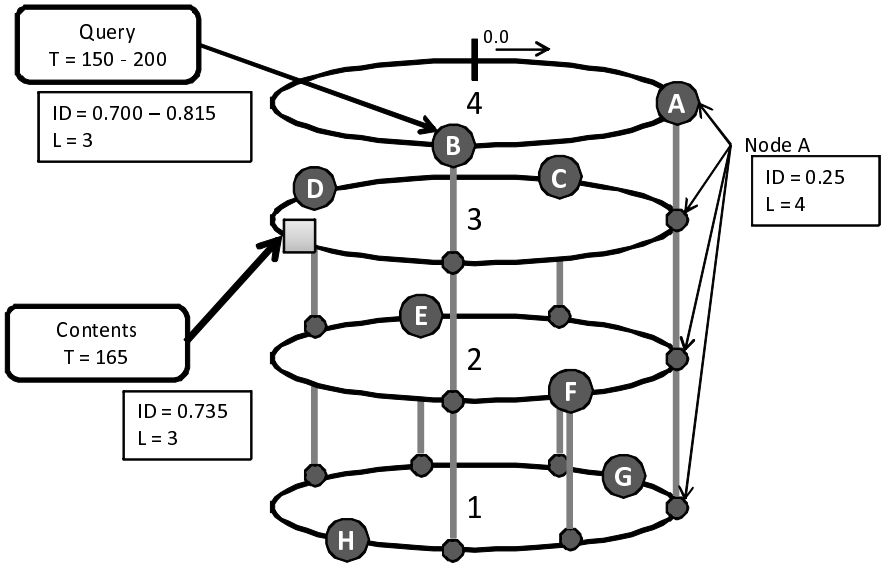

図 3.1 階層化オーバーレイネットワーク概要図

3.2. 階層化オーバーレイネットワークの構築

提案するオーバーレイネットワークに参加するノードは以下の情報を識別子とする.

- リング上の識別子 (ID)
- 階層 (L)

ノードはそれぞれルーティングテーブルを持ち，識別子とそのノードと通信するために必要な IP アドレスなどの情報を保持する．ID は Chord と同様に IP アドレス等をハッシュ関数を用いて変換する．属する階層を表す L は今後の生存時間を元に，より長く生存しているノードを上位のノードとし，Key-Value を優先的に上位の階層へと割り当てることで検索の成功率向上を目指す.

3.2.1 参加時間を利用した階層化

本論文で提案する階層化オーバーレイネットワークでは生存確率と計算機の性能を考慮して階層化を行う必要がある．そこで，提案手法ではノードの参加時間を利用した階層化を行う．これには以下の仮定を用いる．

1. 参加時間の長いノードほど生存確率が高い
2. 参加時間に比例して Key-Value を割り当てる

1 の仮定は論文 [36] により明らかにされており，Kademlia などのオーバーレイネットワークでも利用しているオーバーレイネットワークの性質である．論文 [39] では否定的な見方がされているが，1 週間や 1ヶ月のような長期的な期間では有効であると想定をすることが可能である．また，本論文が想定しているようなサービス提供者による計算機を考慮すれば，参加時間による生存確率の推定は有益である．2 の仮定は長時間参加しているノードほど高性能な計算機であるという想定に基づくものである．オーバーレイネットワークに長時間参加するノードはサービスのヘビーユーザの計算機である可能性が高く，ヘビーユーザは高性能の計算機を利用すると想定している．以上のように 1 と 2 を仮定すれば，生存確率と計算機性能を考慮した階層化に参加時間を用いることができるが，階層 L に変換するためには以下の問題点に留意する必要がある．

参加時間に対する階層 L の増加　時間はサービスの始まりから単調に増加するが，それに比例する形で階層の数が単調に増加してはならない．提案手法の階層化では，ノードの参加と離脱による分断や多重化を考慮し，上位のノードが下位の階層にも属する．そのため，参加時間に比例して階層数が単調に増加すると階層毎にオーバーレイネットワークを維持する必要があり，ノードの動作やメッセージ数が増え，ノードの負荷やトラフィックを圧迫する恐れがある．

Key-Value の割り当て　ノードとなる計算機に割り当てられる Key-Value の数は，サービスや計算機の利用状況などによっても異なるため，どのような環境下でも最適に分配することは非常に困難である．そのため，利用者は計算機上で動

作させるノードの数を増減させることで計算機にかかる負荷を調整する．ノードの参加時間を用いることは生存時間分布によって階層のノード数が変化することになり，Key-Value の割り当てを予測することが困難になる．

3.2.2 対数を用いた階層化手法

3.2.1 で述べた階層化の問題点を解決する手法として，対数を用いた以下の手法を提案する．対数を用いることで，3.2.1 に述べた問題点がノードの生存時間分布が特定の分布を示す場合に解決できる他，2 を底とした対数を用いることでシフト演算とマスク処理のみで実装できるなどの利点がある．以下に対数を用いた手法を説明する．

参加時刻は現在時刻 Now を用いて，次のように階層 L へ変換する．

$$L \quad = \quad \lfloor \log_2(Now - t) \rfloor \tag{3.1}$$

参加時刻である t は ID と共に識別子として他のノードへと通知し，実際の L は各ノードが計算機の時計から取得した Now を用いて演算する．そのため，ノードの階層 L が変化したときに他のノードへそれを通知する必要はない．L を算出する関数に対数を用いることで，参加時間の増加に比例して L が大きくなることはなく，数ヶ月や数年の運用であってもオーバーレイネットワークの階層が多くなることはない．例えば参加時間が 1ヶ月（30 日）のノードは秒数にすると 2,592,000 秒となるが，L は 21 である．同様に参加時間が 1 年（365 日）のノードは秒数にすると 31,536,000 秒となるが，L は 24 であり，1ヶ月の場合と比べて 3 階層の違いしかない．この例では階層が 20 以上作成されることになるが，参加時間の下限となる時刻をあらかじめ設定すれば階層の数を調整する．

また，式 3.1 を用いることで，参加時間の長さ（$T = Now - t$）に対してノード数が対数的に減少（$-\log_2 T$）すると仮定すれば，ノードに割り当てられる Key-Value を参加時間の長さに正比例させることができるため，計算機に割り当てる Key-Value の数を調整することが容易になる．以下にそれを示し，また各階層のオーバーレイネットワークについて説明する．

階層 L と割り当てられる Key-Value の関係　L に属するノード数を N_L とし，3.3 に示す手法で Key-Value を Put した場合，1 ノードあたりの担当範囲 W は下記の通りとなる．

$$W = \frac{2^L}{N_L} + \frac{2^{L-1}}{N_{L-1}} + \cdots + \frac{2^0}{N_0} \tag{3.2}$$

ここで，参加時間 (T) とノード数 $(N(T))$ の関係が $N(T) = -\log_2 T$ であれば，各階層のノード数は単調に減少するため，1 ノードあたりの担当範囲は下記の通りとなる．

$$\begin{aligned} W &= \frac{2^L}{N_L} + \frac{2^{L-1}}{2N_L} + \cdots + \frac{2^0}{LN_L} && (3.3)\\ &= \frac{1}{N_L}\sum_{i=0}^{L}\frac{2^{L-i}}{i+1} && (3.4)\\ & && (3.5) \end{aligned}$$

ここで，$W < W_s$ となる以下の式を仮定する．

$$\begin{aligned} W_s &= \frac{1}{N_L}\sum_{i=1}^{L} 2^{L-i} && (3.6)\\ &= \frac{1}{N_L}(2^{L-i+1} - 1) && (3.7) \end{aligned}$$

この W_s の L による変化は

$$O(2^L) = O(T) \tag{3.8}$$

となるため，W_s はノードの参加時間に正比例する形で大きくなる．$W < W_s$ であるから，W も参加時間に正比例するかそれより小さな増分になる．

各階層のオーバーレイネットワーク　各階層は Chord と同様のリングトポロジとなるため，successor, predicessor はそれぞれ L 個のノードとなる．successor は ID の管理範囲に密接に関係があるため，定期的にメッセージを送受信することで生存の確認を行い，ID の管理範囲が変化していないかを常に監視する．また，階層 l の successor が $l+1$ の successor と同じ場合，l より下の階層については生存確認を行わない．ノードはリングの多重化を防ぐために最も参加時間の長いノード

をルーティングテーブルに必ず保持する．このノードはサービス提供者の用意する計算機になることを想定しているため，参加する際にサービス提供者などが用意する初期ノード（ブートストラップサーバ）から受け取ることにより対処する．

3.3. Key-Value の Put

階層化オーバーレイネットワークに情報を Put する手法を以下に述べる．時刻は始点と終点を定めることが困難であるため，以下のように周期的な ID を定める．尚，ID は 0 から 1 の値を取り，T は定数，t はコンテンツの時刻情報を示す．

$$H_T(t) \quad = \quad \frac{t \mod T}{T} \tag{3.9}$$

図 3.2 に式 (3.9) による時刻 t と ID の関係図を表す．図 3.2 の左の図は右の図に比べて T が小さいため，直線の傾きが小さくなっている．このため，同じ t の範囲に対する ID の範囲は左図の方が広くなり，右の図に比べて範囲検索に必要なクエリ数は多い．一方で右の図は時間の経過に伴って増える Key-Value の割当量が少なく，ID によって割当量が大きく異なる．このように T を大きくすることで範囲検索時のクエリ数を削減できるが，負荷の偏りを減らすには T を小さくしなければならない．

ここで，2.4.1 で述べたように利用者は古い情報を検索するときほど広い時刻範囲を検索すると考慮すれば，古い情報ほど範囲検索時のクエリ数を削減することが望ましい．そこで，式 (3.9) に用いた T に代わり，式 (3.1) を用いて $H_T(t)$ を以下のように定義する．

$$H_T(t) \quad = \quad \frac{t \mod 2^L}{2^L} \tag{3.10}$$

尚，L はルーティングテーブル内の最も古いノードの階層 L_{max} により制限され，$L \leq L_{max}$ となる．図 3.3 に経過時間と $H_T(t)$ の出力する ID の関係を表した図を示す．図では簡単のため式 (3.10) 中の t の代わりに経過時間を用いているため，経過時間と共に ID が変化するが，実際には t を用いることで L が変化しなければ ID は変わらない．図のように経過時間が長いほど傾きが大きくなるため，ID

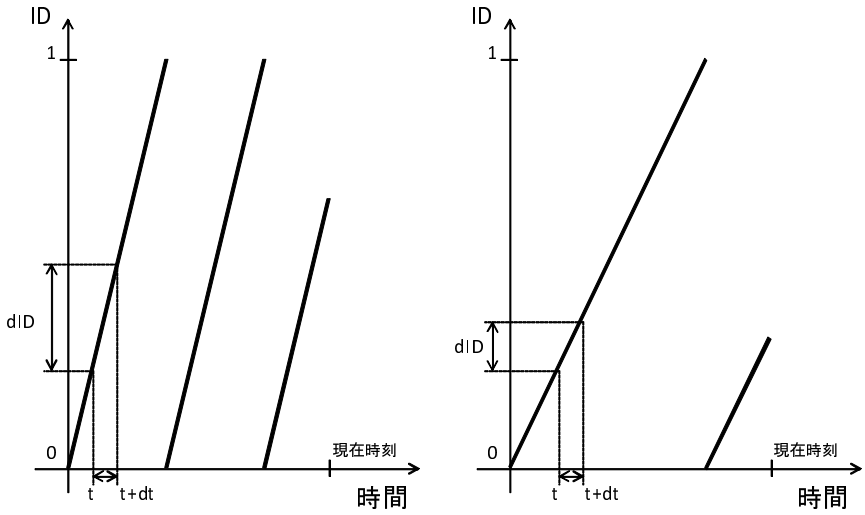

図 3.2 T による ID 範囲の変化

の変化量が小さくなる．t_1 から dt までの時間範囲は ID に変化すると dID_1 とり，古い時刻である t_2 から同じ dt までの時刻範囲は ID に変換すると dID_2 となる．$dID_1 \leq dID_2$ であることから，古い時刻へは少ないクエリ数で範囲検索を行うことができる．

これに加えて，情報が配置されるノードは階層が L 以上のノードに限定する．$H_T(t)$ は古い情報ほど ID の変化が小さく，同じノードへ配置される Key-Value の数が多くなる．そのため，ノードの離脱によって失われる Key-Value は古い情報ほど多くなる．一方でオーバーレイネットワーク上の L が大きなノード群は参加時間の長いノードで構成されているため離脱する確率が低い．そこで，古い情報ほど上位の階層へと Key-Value を配置することで Churn による Key-Value の喪失を防ぐことができる．

3.4. Key-Value の Get

ここでは共有されている Key-Value から指定された時刻属性の Key-Value を取得する手法について説明する．

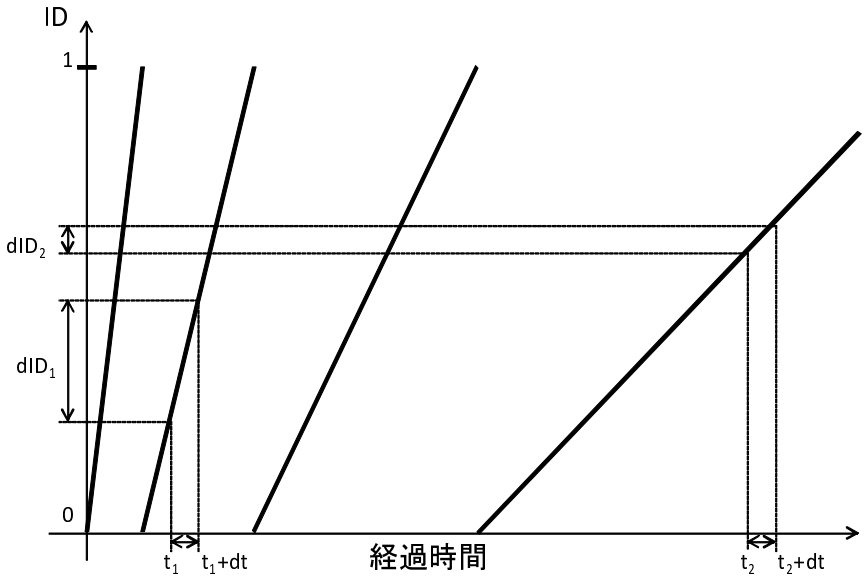

図 3.3 階層化オーバーレイネットワーク概要図

3.4.1 Key-Value の単一検索

利用者が時刻 (t) を指定した検索を行う場合を想定する．t に完全一致する Key-Value の取得方法は従来の DHT の手法と同様である．まず，t から $H_T(t)$ を用いて ID を算出する．これをクエリ ID とする．ノードはノードリスト内からクエリ ID に最も近い ID のノードを探索し，そのノードへクエリを転送する．クエリを受信したノードは同様にノードリスト内から最も近い ID のノードを探索し，転送を繰り返すことでオーバーレイネットワーク上でクエリ ID に最も近いノードへ転送される．クエリを受信したクエリ ID に最も近いノードはデータストア内から Key-Value を探索し，クエリの送信元へ検索結果を送信する．この場合，利用者が正確な時刻を指定しなければデータストア内から Key-Value の探索を出来ないため，現実には次に述べる時刻の範囲を指定した検索の利用が考えられる．

3.4.2 Key-Value の時刻を指定した範囲検索

利用者が始点の時刻 (t_s) と終点の時刻 (t_e) を指定した範囲検索を考える．ここで，式 (3.1) を用いて t_s と t_e から得られる階層の L をそれぞれ L_s, L_e とする．なお，$t_s > t_e$ とし，$L_s < L_e$ である．時刻情報は連続量であるが，$H_T(t)$ を用いて変換した ID は L が変化するときに不連続となる．t_s から t_e の間に不連続点があるかどうかを調べるには，L_s と L_e を比較すればよい．$L_s = L_e$ の場合は同じ階層への検索なので不連続点は考慮しなくてよい．検索する範囲は $L = L_s$ の $H_T(t_s)$ から $H_T(t_e)$ の範囲になる．

$L_s + 1 = L_e$ の場合は t_s と t_e の間に不連続点がある．不連続点となる時刻 t_{si} は以下の式から算出できる．

$$t_{si} = Now - 2^{L_e} \tag{3.11}$$

式 (3.11) では L_e の階層になる最も新しい時刻までの経過時間 2^{L_e} を現在時刻から引くことで不連続点を算出する．この不連続点の時刻を用いて検索する範囲は $L = L_s$ の $H_T(t_s)$ から $H_T(t_{si})$ と，$L = L_s + 1$ の $H_T(t_{si})$ から $H_T(t_e)$ の範囲になる．

$L_s + 1 < L_e$ となるような 3 階層以上にまたがる検索は，$L_s + 1$ の階層の全てのノードへ検索クエリを送信する必要がある．また，上位層のノードが下位層にも属するという前提があるため，$L_S + 1$ よりも上位の階層へはクエリを送信する必要がない．そのため，検索する範囲は $L = L_s$ の $H_T(t_s)$ から $H_T(t_{si})$ と，$L = L_s + 1$ の 0 から 1 の範囲になる．

第4章　実用アプリケーションを考慮したオーバーレイネットワークの実装

本章では提案手法を用いたオーバーレイネットワークの実装について述べる．実際のP2Pアプリケーションによる利用を想定し，オーバーレイネットワーク部分のみのツールとして実装した．P2Pアプリケーションの開発者はツールを用いることで容易にP2P型のアプリケーションを作成できる．実装にはJavaを用いJava VMの動作する環境であればどの計算機でも実行することが可能である．

4.1. 設計方針

オーバーレイネットワークを構成するノードを実装することで，提案手法が動作するツールを実装する．アプリケーションはP2PNodeクラスをインスタンス化することでオーバーレイネットワークに参加し，他のノードとメッセージを交換しながら構造化オーバーレイネットワークを構築する．P2PNodeクラスは次のような設計方針の元に実装した．

4.1.1 ノードの離脱

オーバーレイネットワークではノードは参加と離脱を自由に行えるが，これはオーバーレイネットワークの構造を不安定にする要因となる．これに対して，ノードが離脱する際に特殊な動作（Leave動作）を行うことで不安定な状態を避けることができる．例えば離脱することを周りのノードに伝える動作やKey-Valueの

譲渡などである．しかし，OS にはスリープ機能やサスペンド機能を備えた物が多く，特にラップトップ型の計算機ではそれらの利用頻度が高い．スリープ機能やサスペンド機能が利用された場合，他のノードからは Leave 動作なしに離脱したと見なされる．また，無線 LAN などの電波が圏外になった場合などネットワーク到達性が失われることも多く，これらの場合も Leave 動作なしに離脱したと見なされる．このようなノードの離脱は他のノードがどのような状態にあるときでも発生しうるため，Leave 動作を期待してオーバーレイネットワークを構築することは難しい．そこで，実装するツールでは一般的なインターネットプロトコルの実装に見られるような Soft State な実装を用いる．ノードは一定時間通信が無ければそのノードは離脱したと見なし，オーバーレイネットワークを再構築するなどの動作を行う．ノード間の通信や他のノードから依頼されたタスクの処理などに際してこの Soft State を考慮した実装を行う．

4.1.2 P2P 通信の確保

オーバーレイネットワークを構成するノードはクライアントとサーバの両方の動作を行うため，双方向に通信を行える必要がある．しかし，実際のネットワークでは双方向に通信を行うことが難しい場合が多い．最も顕著な例が NAT[17] によるプライベートアドレスを割り当てられた計算機である．NAT 内のノードはプライベートアドレスが割り当てられるため，NAT 内のノードとは双方向に通信出来ても NAT の外側のノードには片方向からしか接続することができない．また，近年では IPv4 アドレス [31] の枯渇に伴い，IPv6 アドレス [16] の普及が始まっている．IPv6 アドレスのみを保持した計算機と IPv4 アドレスのみを保持した計算機も直接 P2P 通信を行うことはできない．キャリアグレード NAT と呼ばれる ISP のプライベートアドレス提供が始まれば IPv6 アドレスと IPv4 のプライベートアドレスを保持した計算機を想定できる．NAT のみを考慮するのであれば，UDP ホールパンチングや UPnP[7] の IGD 仕様を利用したポート開放の手法を利用できるが，キャリアグレード NAT 環境下では有効でない可能性が高く，また IPv6 アドレスなどには対処できない．NAT 下のノード同士を TCP 接続させる手法 [18] もあるが，やはり IPv6 アドレスのことは考慮されていない．IPv6 ノードと IPv4

ノードの通信にはNAT-PT[41] の利用が考えられるが，単一障害点になることや実現の難しさが問題点として挙げられる．こうした直接通信を行えない計算期間の通信としては文献 [29] があるが，下位ネットワークの大きな変更を要求するためオーバーレイネットワークのツールとしては変更範囲が大きくなりすぎる．

以上のことを考慮し，P2P 通信の方式として以下の 3 つを留意して実装を行った．

- 双方向から接続することができる
- 片方向から接続することができる
- 双方向から接続することができない

同じアドレス体系を持つ計算機間では双方向から接続することができる．しかし，グローバルアドレスとプライベートアドレス間では片方向からの接続しかできない．また，異なるアドレス体系では双方向から接続することができない．このような 3 つの接続関係を考慮した上で P2P 通信を行う．

4.1.3 アプリケーションへの拡張容易性

オーバーレイネットワークはアプリケーションへ提供する通信機能の 1 つに過ぎない．そのため，アプリケーションに対して容易に機能拡張できる必要がある．KVS を実現するだけであれば，アプリケーションに対して Put と Get の操作を提供するだけでよいが，実際にはノードリストやデータストアを直接操作したり，独自のメッセージを参加ノードに送信するなど様々な拡張が想定できる．このような拡張容易性を考慮するために，以下の点を考慮した．

簡潔な構造

拡張性を重視するあまりソースコードの抽象度が高くなり，開発者が記述しなければならない箇所を増やすことがある．また，安全性を重視することで開発者への制限が大きくなることもある．こうした構造の複雑化はアプリケーション全体の把握を困難にし，デバッグの非効率化やソースコードの可読性低下などの原因になる．構造を簡潔にすることで機能拡張を容易にする．

イベントドリブン方式による実装

オーバーレイネットワークに参加しているノードは非同期に並列して動作する．そのため，ノードに対して行われる動作の要求は他のノードによって非同期に引き起こされる．そこで，実装にはイベントドリブン方式を用い，イベントの発生とそれに対するハンドラの登録によってノードの挙動を記述した．イベントの発生やハンドラの登録をアプリケーションが後から追加できるため，開発者はノードで発生する全てのイベントをアプリケーション上で利用することができる．

4.2. 概要

実装したオーバーレイネットワークの概要を示す．

4.2.1 タスクを中心としたノード構成

P2PNode クラスはタスクを中心として動作する．タスクは投入と共にスレッドを生成し，同時並列に動作する．概要を図 4.1 に示す．図 4.1 に示すようにノードはタスクを実行することでリソースを操作する．オーバーレイネットワークの維持に欠かせない動作などはノード自身がタスクとして定期的に投入する．例えば隣接ノードの生存確認を行うタスクは，一定の間隔でノードがタスクとして投入する．また，ネットワーク越しにタスクを投入することで他のノードとの通信が行われる．開発者は独自にタスクを作成し，P2PNode オブジェクトに追加できるため，P2PNode の発生させるイベントなどを利用してタスクを作成することができる．

4.2.2 メッセージによるノードの遠隔操作

構造化オーバーレイネットワークはノード間の協調動作によって構造を生成する．そのため，ノードは他のノードに動作の依頼が出来ればよい．提案するツールではメッセージを送信することで他のノードへのタスク投入を行う．メッセー

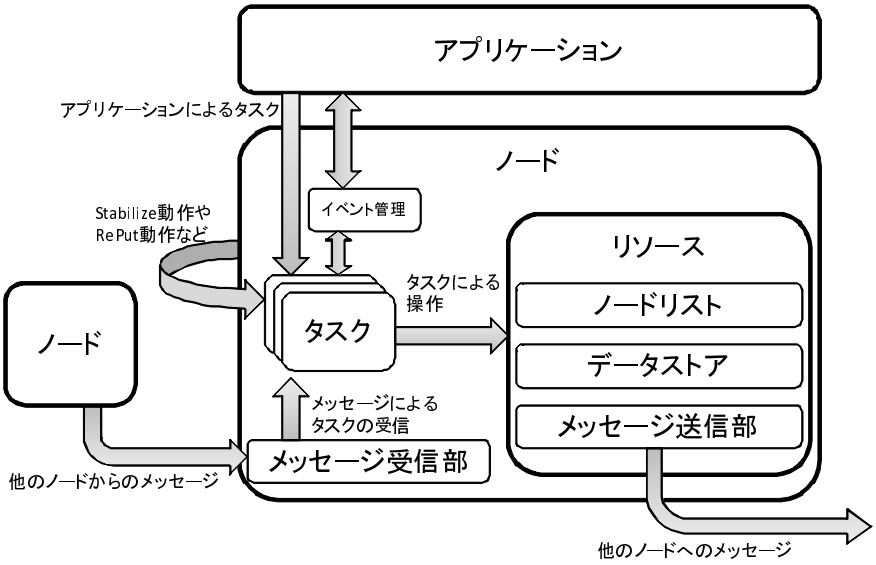

図 4.1 タスクを中心としたノード構成

ジはタスクの内容とタスクに必要なオブジェクトによって構成し，P2PMessage クラスとして実装した．P2PMessage クラスは引数に P2PNode クラスを取る invoke メソッドのみを持ったクラスである．P2PMessage クラスの継承クラスを作成することでメッセージを作成し，他のノードへタスクを投入する．送信元ノードは P2PMessage オブジェクトを直列化し，宛先ノードへと送信する．宛先ノードは P2PMessage オブジェクトを受け取ると invoke メソッドの引数に自身の P2PNode オブジェクトを引き渡し，実行する．つまり，メッセージの invoke メソッドに書かれたソースコードは，送信先のノード内で実行されることになる．

メッセージの例としてノードの生存確認に用いる PingMessage と PongMessage の例をソースコード 4.1 に示す．PingMessage クラスは送信元の情報を srcNode として内部に保持したクラスである．生存確認を行うために，まず PingMessage オブジェクトを直列化し，相手ノードへと送信する．PingMessage クラスを受信したノードは，PignMessage クラス内の invoke メソッドを実行する．invoke メソッド内には PongMessage の作成と，srcNode への送信が記述されているため，メッセージを受け取ったノードは PongMessage を PingMessage の送信元ノードへと送信する．PongMessage を受け取ったノードは，PongMessage 内の invoke メソッ

ドを実行し，ノードリスト内のノード情報を更新することで生存が確認できたことを記録する．

このように開発者はP2PMessageクラスを継承したクラスを作成し，invokeメソッド内に送信先のノードで行う動作を記述することでノード間の通信を実現する．invokeメソッド内ではP2PNodeオブジェクトを通してリソースを操作できるため，開発者はほとんど制限を受けることなく様々な動作を記述することができる．また，Javaの直列化機能を使ったため，送信先のJava VMで読み込まれていないクラスファイルは復元できない．そのため，開発者が作成した任意のコードが全く関係のないノード上で動作することはない．

ソースコード 4.1 Ping/Pongメッセージ例

```
1  public class PingMessage extends P2PMessage {
2
3    private NodeId srcNode;
4
5    public PingMessage(NodeId srcNode){
6      this.srcNode = srcNode;
7    }
8
9    public void invoke(P2PNode node){
10     PongMessage pong = new PongMessage(node.getId());
11     node.getSender().sendMessage(pong, srcNode);
12   }
13 }
14
15 public class PongMessage extends P2PMessage {
16
17   private NodeId srcNode;
18
19   public PongMessage(NodeId srcNode){
20     this.srcNode = srcNode;
21   }
22
23   public void invoke(P2PNode node){
24     node.getNodeIdList().update(srcNode);
25   }
26 }
```

4.2.3 P2PEventManager クラス

タスク間での通信をイベントドリブン方式による P2PEventManager クラスで実現する．図 4.2 に概要を示す．図中の P2PEvent, P2PEventListener, P2PEventControler, P2PEventManager はそれぞれ実装したクラスとインタフェースの名前である．尚，＜＞内は Java の Generics 機能の記述である．開発者は P2PEvent クラスの継承クラスを作成することでイベントを記述し，P2PEventListener インタフェースの実装によってハンドラを記述する．ハンドラは P2PEventControler に登録されることで，イベントの発生と共に実行される．タスクが P2PEventControler クラスの action メソッドを呼び出すことでイベントが発生する．また，P2PEventManager クラスで P2PEventControler オブジェクトをまとめ，イベントとハンドラを一括管理する．P2PEventManager クラスは内部に P2PEventControler オブジェクトのリストを保持し，P2PEventControler の登録や削除，イベントの発生とリスナーの登録と削除を行う．開発者はオブジェクトの P2PEventManager オブジェクトを取得することで，イベントの登録や発生を任意に行うことができる．

この方式のイベントとハンドラの登録は，オブジェクトの発生させるイベントの種類が分からないため，実装のバグがあったとしても動作するまで分からない問題点がある．しかし，ハンドラを追加するだけの継承クラスを作成する必要がなく，無名インナクラスを利用することで簡潔にハンドラを記述できる利点がある．

ソースコード 4.2 にイベントドリブン方式を用いた Hello World プログラムの例を示す．ソースコード 4.2 ではまず最初に node インスタンスの P2PEventManager に HelloEvent の P2PEventControler を追加する．HelloEvent クラスは別に実装する．次にイベントのハンドラとなる P2PEventListener を登録する．この例では無名インナクラスを用いて作成する．P2PEventManager に対して action メソッドを呼び出し，HelloEvent が発生する．HelloEvent が発生すると event メソッドが呼ばれ，標準出力に「Hello World!」の文字列が表示される．

ソースコード 4.2 イベントドリブン方式による Hello World の例

```
1    node.getManager().addControler(HelloEvent.class);
2    node.getManager().add(new P2PEventListener<HelloEvent>() {
3      public void event(HelloEvent event){
4        System.out.println("Hello␣World!");
```

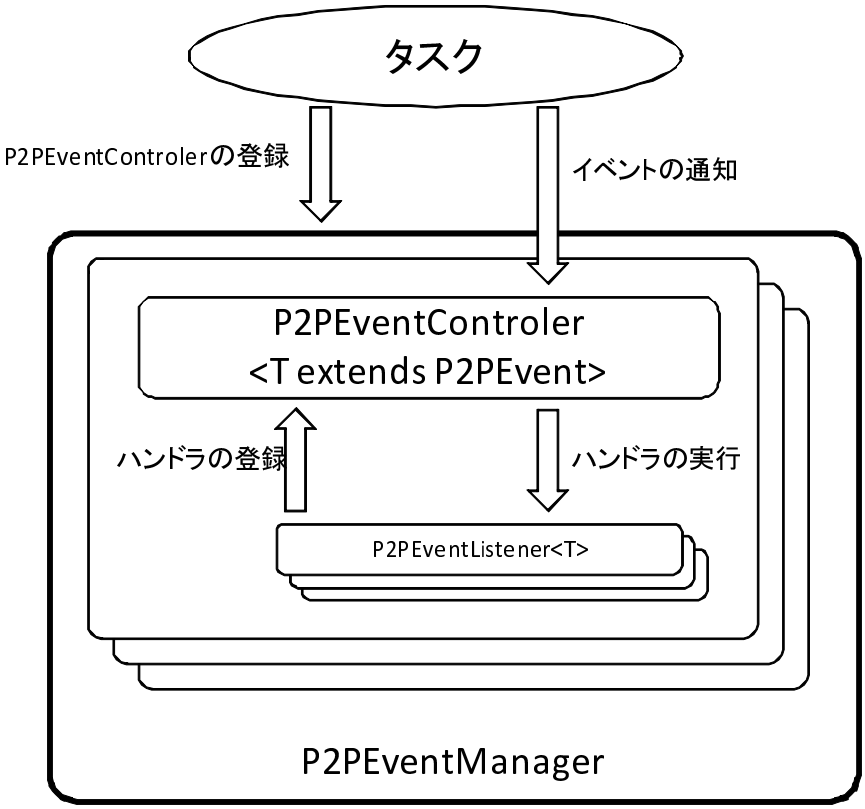

図 4.2 イベントドリブン方式の概要図

```
5       }
6     }
7     HelloEvent event = new HelloEvent();
8     node.getManager().action(event);
```

4.2.4 リング構造の安定化

ノードは構造を維持するためのタスク（Stabilize タスク）を定期的に実行する．Stabilize タスクは以下の通りの手順となる．

1. 自身から最も近いノードに対して Stabilize メッセージを送る．
2. Stabilize メッセージを受け取ったノードは，ノードリストに送信元のノードを追加し，送信元ノードから最も近いノード情報を返信する．
3. 返信を受け取ったノードは一定時間後に 1 へ戻る．

Stabilize メッセージの送受信に失敗した場合は，ノードリスト内の該当ノードを削除する．Stabilize タスクが定期的に実行されるため，実装するオーバーレイネットワークでは参加や離脱に際し特別な動作を定めない．ノードがオーバーレイネットワークへ参加するには，ブートストラップサーバなどから参加しているノードの情報を受け取り，1 つでもノードリスト内に他のノードの記載があれば参加していると見なす．参加したノードはすぐに Stabilize タスクを起動するため，時間経過と共にリング上の隣接ノードを発見することができる．

4.3. メッセージの送受信

メッセージの送信には MessageSender クラスを利用し，メッセージの受信には MessageReceiver クラスを用いる．MessageReceiver は抽象クラスであり，実際の受信を行うネットワーク毎に継承クラスを記述する．標準で TCP と UDP に対応し，それぞれに IPv4,IPv6 のアドレスを指定できる．ノードは受信可能なアドレスの情報などを NetworkInterface クラスのオブジェクトとして作成する．NetworkInterface クラス内にはそのノードへメッセージを送信するために必要な情報が記

されている．例えばTCP の場合ではIP アドレスとPort 番号である．ノードは自身の NodeId オブジェクトのメンバとして NetworkInterface オブジェクトを追加する．他のノードに NodeId オブジェクトを送信しておくことで，他のノードのメッセージ送信を可能にする．また，NetworkInterface を実装することで，TCP やUDP でなくとも様々な通信方法を実装することが可能となる．

4.3.1 ノード間リンクの回収機構

他のノードと通信を行うには NodeId を用いて宛先ノードへのリンクを作成する．メッセージの送信後，リンクはコネクションのプール（Connection GC）へと格納される．別のタスクが同じノードと通信を行うときは，Connection GC 内にあるリンクを取得してメッセージを送信する．複数のタスクが同時にリンクを取得する可能性があるため，Connection GC 内のリンクは取得されると削除し，メッセージの送信が終わると Connection GC 内に再びリンクを格納する．リンクの取得に失敗した場合は NetworkInterface を用いてリンクを新しく作成する．リンクを Connection GC に格納する際に，既にリンクが格納されている場合は格納されたリンクを削除して，新しいリンクを格納する．

また，Connection GC はガーベジコレクションのように自動的に利用されなくなったリンクの回収を行う．リンクは Connection GC へ格納する際に利用した時刻を記載する．Connection GC を管理するタスクは利用した時刻から期限切れのリンクを探索し，そのリンクをクローズした上で削除する．このようなリンクの回収機構を導入することで，リンクのクローズや再利用を開発者は考慮する必要がない．特に TCP ではリンクの接続とクローズを頻繁に行うとポートの枯渇が起きる．オーバーレイネットワークでは ID 空間上の隣接ノードや距離の近いノードと頻繁にメッセージ交換を行うことが多いため，リンクの回収機構を用意することで効率的にメッセージ交換を行うことを期待できる．

4.3.2 ReverseLink と RelayLink による P2P 通信の実現

4.1.2 節に述べたように，実際の利用では直接通信できないノード間の通信を考慮しなければならない．そこで，メッセージの送受信に用いるリンクとして ReverseLink と RelayLink を作成する．図 4.3 に概要図を示す．尚，図では通常のリンクを P2PLink, メッセージを P2PMessage として表す.

Reverse Link

ReverseLink クラスは片方向からのみ接続できる場合に利用する．これは特に NAT 内のノードが TCP 接続することを想定している．図 4.3 では B と C の関係がそれにあたる．C は B に接続を行えるが，B は C に接続ができない．この場合でも，C から B へ接続したリンクを利用して B から C へメッセージを転送することは可能である．そこで，ノードは ReverseLink を C から B へ接続し，C は接続したリンクを利用した MessageReceiver を作成し，ReverseLink からのメッセージを待機する．B は接続された ReverseLink から作成した NetworkInterface オブジェクトを送信元ノードの NodeId に登録し，メッセージの送信時に利用する．尚，ノード B が離脱した場合に備えるため，待ち受けにはタイムアウトを設定する．C はタイムアウトの時間より短い間隔で B に対して PingMessage を送信し，PongMessage は ReverseLink を利用するためリンクを維持できる．

原則として，接続可能なノードの中で最も ID の距離が近いノードから PingMessage に対する応答が無かった場合にのみ ReverseLink を作成する．ReverseLink の作成を規制しなければノードが作成する ReverseLink の数が増え，PingMessage などの動作によってノードが過負荷状態になる可能性がある．相手ノードから ReverseLink の要求があった場合などは作成するが，それ以外の場合は作成しない.

RelayLink

RelayLink は 2 つのノードが双方向に接続できない場合に，両方のノードが通信できるノードを中継させることでメッセージを送信するリンクである．図 4.3

ではAとCの関係がそれにあたり，Bが中継することでAからCにメッセージを転送する．

ノードはお互いに送受信を行えるノードの中で，最もIDの距離が近いノードを中継ノードと定め，NodeIDに中継ノードのIDが記載されたNetworkInterfaceオブジェクトを追加する．送信元のノードはNodeIdに記載された情報を元にメッセージの送信を試み，他のどの方法を用いてもメッセージを送信出来ない場合にRelayLinkを使ってメッセージを送信する．図4.3の例ではBを中継ノードとしたRelayLinkをCが作成し，CのNodeIdに追加した上でBに新しいNodeIdを伝える．Aは何らかの方法でCの新しいNodeIdを取得する．多くのオーバーレイネットワークでは，通信できないノードを発見した場合，ID空間上で近くのノードを代替ノードとすることが多いため，代替ノードにあらかじめ通知しておくことでNodeIdの取得を期待できる．尚，実装したリング型のオーバーレイネットワークではChordのsuccessorにあたるノードを中継ノードと定めた．Cの新しいNodeIdを得たAはRelayLinkを用いてメッセージを送信する．RelayLinkでは，元のメッセージを内包したRelayMessageを中継ノードへ送信することでメッセージの中継を行う．RelayMessageではinvokeメソッド内に宛先ノードへ内包したメッセージを送信する動作が書かれているため，送信先の中継ノードでinvokeメソッドが実行されれば内包したメッセージは宛先ノードへ送信される．ソースコード4.3にRelayMessageの概要を示す．中継ノードはRelayMessageのinvokeメソッドを実行し，元のメッセージを宛先ノードへと送信する．図4.3の例ではRelayMessageがBへ送信され，BからCへ元のメッセージが送信される．CはBに対して前述のReverseLinkを作成しているため，BはCに対してメッセージを送信することができる．

ソースコード 4.3 RelayMessage例

```
1 public class RelayMessage extends P2PMessage {
2
3   private NodeId dstNode;
4
5   private P2PMessage originalMessage;
6
7   public RelayMessage(NodeId dstNode, P2PMessage originalMessage){
8     this.dstNode = dstNode;
```

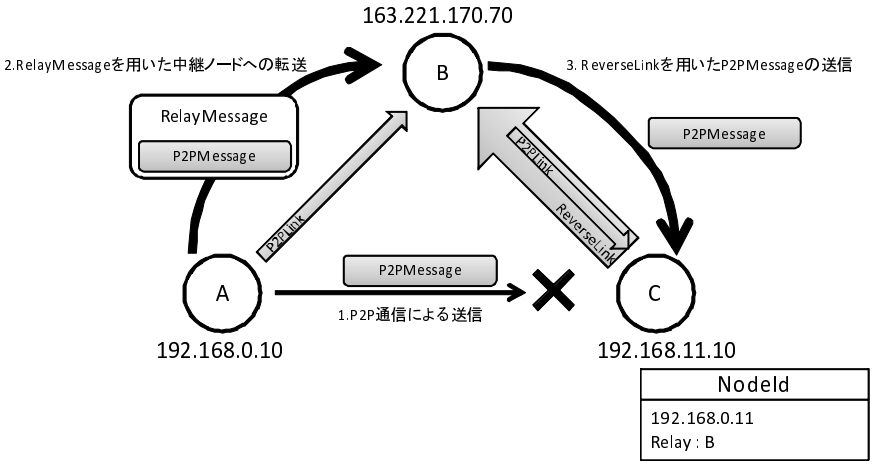

図 4.3 ReverseLink と RelayLink による P2P 通信

```
9       this.originalMessage = originalMessage;
10    }
11
12    public void invoke(P2PNode node){
13      node.getSender.sendMessage(dstNode, originalMessage);
14    }
15  }
```

4.3.3 範囲検索による情報の取得

DHT を用いた KVS では単一の Key に対して完全一致する Value を探しだす検索機能を提供すればよい．しかし，提案手法は時刻を Key として扱うため，Key の範囲を指定した検索を実装する必要がある．以下に問題点とその対処法を述べる．

範囲検索の問題点

単一の検索では Key に対する割り当てノードが一意に定まるため，1 つのクエリを送信し，それに対する結果を得られればよい．しかし，範囲検索では Key の範囲を指定するため，Key に対する割り当てノードが複数になる場合が多く，そ

れら全てのノードに対してクエリを送信しなければならない．4.1.1 で述べたようにノードはいつ離脱するか分からないため，クエリの結果を待機するような設計は望ましくない．また，全てのクエリに対する応答を得るまで，検索結果をアプリケーションが利用出来ないことは避けるべきである．単一の検索では検索の結果も単一のノードから取得するため，検索の結果を取得すればすぐにアプリケーションへ通知すればよい．これに対して範囲検索では，複数のノードにクエリを送信するため，検索結果を得るには全てのノードからの検索結果を待機しなければならない．しかし，アプリケーションによっては完全な検索結果でなくとも結果を取得したい場面がある．例えばファイル共有のアプリケーションの場合，時刻による範囲検索によって得られたファイル情報は全てが揃わなくとも利用者に提示する方が利便性が高い．そのため，従来の KVS における GET のように，引数に対する返り値として検索結果を引き渡すことは範囲検索では適切ではない．

ResultBox による範囲検索の実装

検索結果を格納する ResultBox クラスとイベントドリブン方式による通知によって範囲検索の機能を実装した．図 4.4 に概要図を示す．範囲検索を行うタスクはまずクエリを作成する．クエリには検索の時刻の範囲と，3.4.2 の手法によって算出された検索対象の ID 範囲が記載される．図 4.4 の例では簡単のため階層を省略し，0.3 から 0.8 の ID 範囲とする．タスクはこのクエリをまず ResultBox に登録し，検索結果を取得したときの動作をハンドラとして登録する．タスクはクエリを送信するノードをノードリストから取得し，全てのノードにクエリを送信してタスクを終える．このとき，送信先のノード毎にクエリの ID 範囲を書き換える．図 4.5 に例を示す．図 4.5 では図 4.4 と同様に 0.3 から 0.8 の ID 範囲にクエリを送信する．0.3 から 0.8 の ID 範囲を担当する可能性があるノードには全てクエリを送信するため，ノードの ID が 0.2,0.4,0.7 のノードがクエリを送信するノードとなる．送信するクエリにはそれぞれクエリを送信した ID 範囲が記載される．ID が 0.4 のノードには 0.4 から 0.7 の範囲についてクエリを送信したことが記載される．ID が 0.4 のノードはクエリを受け取ると，0.4 から 0.7 の範囲について同様の範囲検索を行う．元のノードは 0.4 から 0.7 に含まれる 0.5 や 0.6 の

ノードを知らないが，0.4 のノードがクエリを再送信することで ID 範囲に含まれる全てのノードにクエリを送信することができる．

クエリを受け取ったノードはクエリに対する応答を Result メッセージとして作成し，送信元ノードへと返信する．Result メッセージには結果の Key-Value を表す Data の他にノード ID と successor の ID を記載する．これによって，オーバーレイネットワークの ID 空間上でどの部分の検索結果であるかを明示することができる．Result メッセージを受け取ったノードは，ResultBox 内に Data を格納し，登録された Query の ID を減算する．図 4.4 の例では 0.3 から 0.8 の ID 範囲について 0.7 から 0.9 の検索が終わったため，減算して 0.3 から 0.7 が残りの ID 範囲となる．クエリの送信先でクエリが別のノードへと送信されていたとしても，Result メッセージの ID 範囲を確認することで範囲検索が終了したかどうかを確認することができる．また，この ID 範囲の距離はクエリしたノード数と関係するため，元の ID 範囲の距離と残りの ID 範囲の距離を比較することで，検索結果が終了するまでの目安をパーセンテージとして表すことができる．この ID 範囲の距離が 0 になったときに全ての ID 範囲から結果を受け取ったことになるため，登録されたハンドラを呼び出し検索終了後の処理を行う．また，結果を受け取ったときに発生するイベントや，残りのパーセンテージによって発生するイベントなどを定義することで，アプリケーションの多様な要求に応えることが可能である．

4.4. P2PNode クラスを構成するクラス

P2PNode クラス内にある他の主要なクラスについて説明する．

4.4.1 NodeIdList クラス

NodeIdList クラスはノードリストを表すクラスである．NodeIdList クラスは以下の要素によって構成される．

- ノード自身の NodeId
- 最古ノード

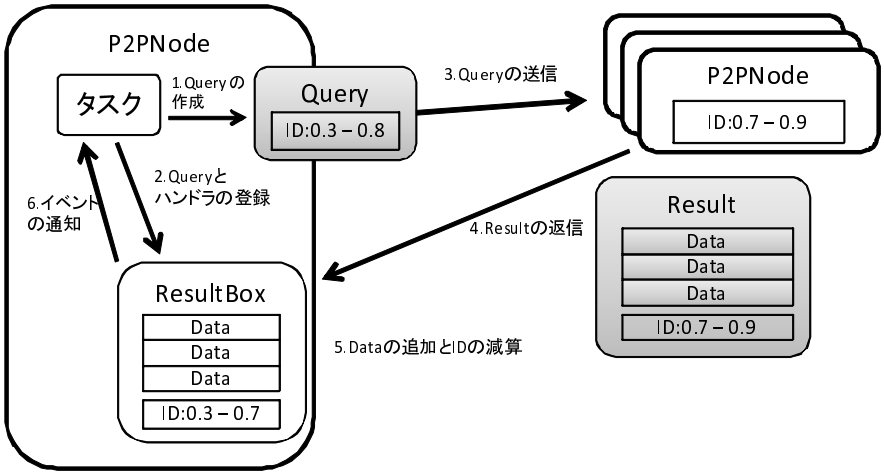

図 4.4 ResultBox とイベントドリブン方式による範囲検索

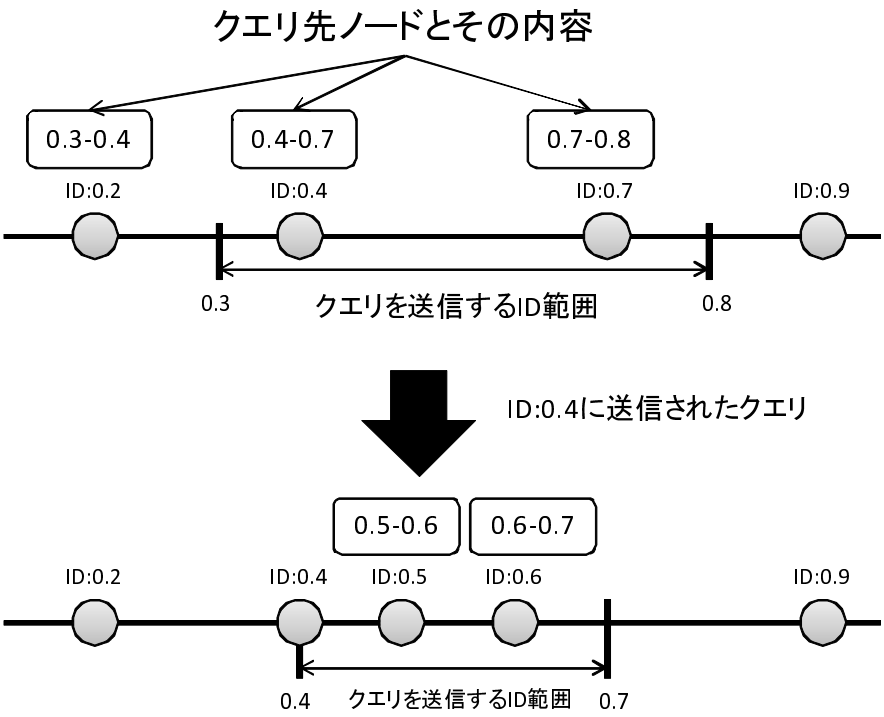

図 4.5 0.3 から 0.8 の ID 範囲に対する検索の例

- ID リスト

NodeId はこの NodeIdList を保持しているノード自身の ID を表し，最古ノードはオーバーレイネットワークに参加している最も古いノードの ID を表す．ID リストは自身の NodeId を含めた既知の NodeId を全て記載する．NodeId クラスは以下のエントリによって構成する．

- ID
- 参加時刻
- NetworkInterface リスト
- 更新時刻

ID は各ノードを識別するためのものである．特に規定していないため，DHT で一般的な 160bit の Byte 列を利用できるが，実装では浮動点少数型の Double を用いている．参加時刻はそのノードがオーバーレイネットワークに参加した時刻を記載する．この時刻を元にノードの階層を算出する．NetworkInterface は 4.3 節で述べたメッセージを実際に送信するときに利用される．最後の更新時刻はこのノードからメッセージを受け取った最後の時刻を表す．生存確認を行う隣接ノードに関して，ノードはこの更新時刻を確認することで生存しているかどうかを判断する．ID リストは Skip List を用いて ID 順にソートし，任意の ID から最も近いノードの ID を高速に探索することができる．

4.4.2 DataStore クラス

DataStore クラスでは共有する Key-Value の情報を管理する．KVS における Key-Value は P2PData クラスによって表す．P2PData クラスはエントリとして，以下の属性を持つ．

- 時刻属性の Key
- 作成した NodeId

- 期限切れ時刻

KVS の Key として利用するのが時刻属性の Key である．また，作成した NodeId を保持する．オーバーレイネットワークで共有される情報は，元の情報のメタデータであることが多く，作成元のノードに問い合わせることで元のデータを取得することができる．そのため，作成元の NodeId を P2PData クラスの内部に保持しておく．期限切れ時刻は P2PData オブジェクトを受信したノードが P2PData オブジェクトを削除するかどうかの判断に利用する．P2PData クラスを継承したクラスの実装によりアプリケーションは共有する情報を作成する．単に情報を共有するだけであれば，P2PData クラスの継承クラスを作成し，DataStore に追加するだけで共有される．DataStore クラスでは以下のリストによって P2PData クラスのオブジェクトを保持する．

- myDataStore
  DataStore を保持するノードによって追加された P2PData を保存するリストが myDataStore となる．myDataStore 内の P2PData は明示的な処理がない限り削除されることはない．また，myDataStore 内の P2PData は定期的にその時刻属性から ID を計算し，担当となるノードへ P2PData の更新を行う．myDataStore から削除された P2PData は更新が行われなくなることで，担当のノードから削除される．

- extDataStore
  他のノードから受信した P2PData を保存するリストが extDataStore になる．myDataStore と違い，満了時刻を過ぎても更新されなかった場合は廃棄される．

- forwardingQueue
  本来ならば別のノードへと届けなければいけない P2PData をなんらかの理由によって受信した場合，その P2PData は forwardingQueue に追加される．forwardingQueue に追加された P2PData は myDataStore 内の P2PData を更新する際にまとめて該当するノードへと送信される．

また，標準で定期的に動作するタスクが DataStore の myDataStore 内にある P2PData オブジェクトを監視し，追加された P2PData オブジェクトがあれば PutDataMessage オブジェクトを作成し，担当となるノードへ送信する．ソースコード 4.4 に PutDataMessage クラスの例を示す．PutDataMessage クラスは送信先のノードで P2PData オブジェクトを保持する NodeId を取得し，その NodeId が送信先のノードであれば，DataStore に P2PData を追加する．別のノードであった場合はそのノードへと PutDataMessage を転送する．

ソースコード 4.4 PutDataMessage 例

```
1  public class PutDataMessage extends P2PMessage {
2
3    private P2PData data;
4
5    public RelayMessage(P2PData data){
6      this.data = data;
7    }
8
9    public void invoke(P2PNode node){
10     NodeId dstNode = node.getNodeIdList().getDataHolder(data);
11     if(dstNode.equals(node.getId()){
12       node.getDataStore().add(data);
13     } else {
14       node.getSender.sendMessage(dstNode, this);
15     }
16   }
17 }
```

第5章　評価

本章では提案手法が多対多通信を行う上で有益となることを示す．本論文では時刻属性を KVS の Key として利用することで，より多くの情報を共有することが可能であると提案した．本章では検索成功率とクエリ数を，従来の DHT，階層化しないオーバーレイネットワークと比較して，提案手法の評価を行う．また，階層化の手法に関して提案手法以外に 2 つの手法によって同様の実験を行い，提案手法を評価する．これらに加えて，時刻を Key に用いた実際のアプリケーションとして，写真を共有するアプリケーションの実例を示す．実際のアプリケーションへの拡張が容易であることと，実装したエミュレータが簡単な拡張で様々な機能を追加できることを示すことでエミュレータの評価とする．

5.1. 階層化・非階層化・DHT での比較評価

以下の 3 つのオーバーレイネットワークを比較することで提案手法を評価する．

- 階層化したもの（階層化と式 (3.10) による *ID*）
- 階層化しないもの（式 (3.10) による *ID*）
- DHT（リングトポロジと SHA-1 ハッシュ関数による *ID*）

また，ノードの生存時間分布は以下のものを用いた．

- 対数を利用した分布
- Weibull 分布
- 正規分布

生存時間が t であるノードの分布 ($f(t)$) を以下のように定義したものが対数を用いた分布である．

$$f(t) = -\log t \tag{5.1}$$

この分布では提案手法を適用した場合に各階層のノード数が等倍となり，ノードに割り当てられる Value の範囲が生存時間に正比例する．ノードへの負荷を生存時間に正比例して割り当てるのであれば，この対数を利用した分布が理想的な分布となる．また，Weibull 分布は論文 [39] にて指摘されている分布であり，既存のオーバーレイネットワークを計測した結果から得られる分布である．利用者の参加するサービスとしてオーバーレイネットワークを構築すれば，1 分や 2 分のような非常に短い間隔で参加離脱を繰り返すノードよりも，少し長めの間隔で離脱するノードが多くなると予想できるため，Weibull 分布は現実の利用に近いと想定できる．最後に一般的に用いられる正規分布を用いる．これは利用者の挙動としては考えにくいが，想定とは外れた分布の場合の動作として評価する．ノードは作成されたときに生存時間が定められ，以降は生存時間毎に参加と離脱を繰り返す．実験時間を 0 から 1 としたときのノード数の分布を図 5.1 に示す．離脱したノードは同じ生存時間を用いて参加し，ノードの生存時間分布や参加ノード数は変化しない．

以上のようなオーバーレイネットワークを擬似的に構築し，検索の成功率と検索クエリ数，ノードの保持している Value の数，オーバーレイネットワーク上で交換されたメッセージの総数を評価した．4 章に述べたツールを利用し，Chord と同様のリング構造を持つ評価用オーバーレイネットワークを実装した．実装では Chord にある finger table は作成せず，一度でも送受信を行ったノード情報を全てルーティングテーブルへ追加することで検索の高速化を行った．これは，提案手法を比較する対象は Chord ではなく DHT 全般であるため，Chord のようにルーティングテーブルを制限する方式よりも，制限しない方式の方が Churn の影響を受けにくいためである．実装した評価用オーバーレイネットワークは時刻の Key から ID の算出を行う部分を自由に選択できるため，暗号学的ハッシュ関数である SHA-1 を用いた DHT と，提案手法の比較を行うことができる．また，各ノードの階層を参加時刻を元に算出する方法と，常に同じ値を返す方法を用いる

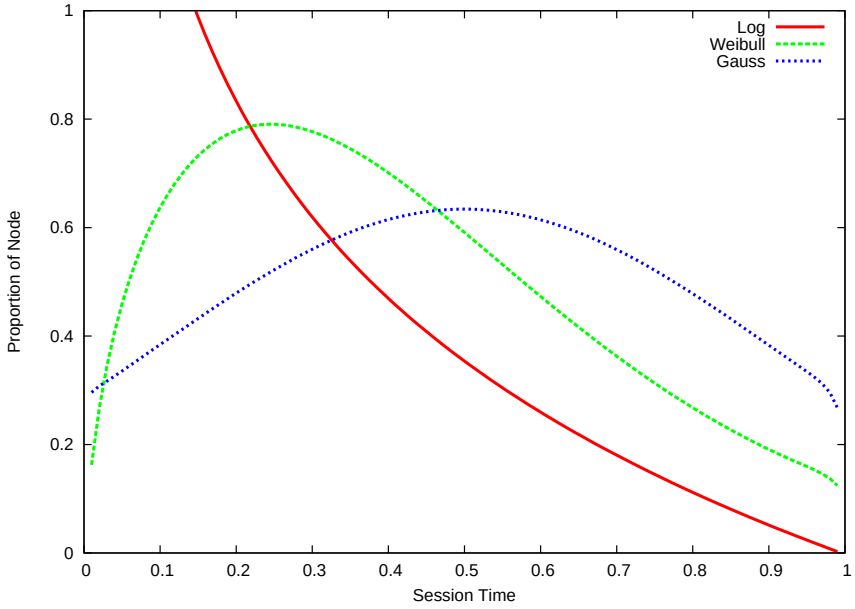

図 5.1 実験に用いたノードの生存時間分布

ことで，階層化の有無に関しても比較を行った．

評価方法はランダムな時刻を起点とした範囲検索を 10 秒間隔で行い，実際に共有されている Value のうち検索内容に該当する Value すべてを取得できた場合に成功，残りの場合を検索失敗とした．ノードからクエリを送信し，検索結果を得た後に全ての P2PNode オブジェクトから検索内容に該当する Value を取得し，検索結果と比較して検索の成功を判断する．クエリが指定する時間の範囲は 60 秒，120 秒，180 秒と 3 つのパターンを用意し，それぞれのクエリについて独立して実験を行った．また，DHT を用いた場合は範囲検索を行うことができないが，Value を 1 秒間隔で量子化し，クエリも同様に量子化することによって代替した．つまり，60 秒の範囲検索の場合は 60 クエリを送信することになる．

5.1.1 実験結果

評価実験では 500 ノードによるオーバーレイネットワークを実験環境上で 2 時間動作させた．図 5.1 における最大の生存時間が 2 時間に相当する．また，計算

表 5.1 実験に用いた計算機

Sun Fire V40z	1 台
CPU	AMD Opteron 852 (2.6GHz) x4
Memory	16GB
OS	Solaris 10
Sun Java Workstation W2100z	9 台
CPU	AMD Opteron 252 (2.6GHz) x2
Memory	2GB
OS	Solaris 10

機 10 台を用いて同じ実験を行い，それらの平均値を実験結果として評価した．表 5.1 に実験に用いた計算機の概要を示す．また，表 5.2 に実験結果の平均値を示す．

対数を用いた分布の実験結果

図 5.2, 5.3, 5.4, に検索成功率の結果を，図 5.5 にクエリ数の実験結果を示す．これらのグラフは横軸にクエリの検索対象となる時刻を用いているため，横軸が大きいほど古いデータに対する検索である．図 5.2 に示す通り，提案手法を用いることで特に古い Value に対する成功率に顕著な差が出ているといえる．表 5.2 から，階層化したことによって約 15%向上し，DHT と比較すれば約 44%向上したことになる．図 5.2,5.3,5.4 を比較すれば検索の時間範囲を拡大しても提案手法の検索成功率はそれほど低下せず，他の方式と比べて高い成功率を維持していることが分かる．また，図 5.5 と照らし合わせると，提案手法はクエリ数を減らしつつ成功率が向上したことが分かる．表 5.2 から，階層化することで約 22 のクエリを削減できたことがわかる．なお，DHT は量子化を行うことで範囲検索を行ったため，クエリの数は一定であるが，検索範囲を広げればクエリ数は線形に大きくなる．実験の結果では 60 秒の範囲検索によって約 94 のクエリを確認した．これは，クエリの転送などによってクエリ数が増加したためである．

図 5.6 にノードの生存時間と保持している Value の数を，図 5.7 にメッセージ数の時間変化を示す．図 5.6 は横軸にノードの生存時間を用いているため，ノー

ドは生存時間が長くなるほど Value が多くなることが確認できる．しかし，生存時間の長いノードほど高性能な計算機であることを仮定しているため，想定通りに Value が割り当てられているといえる．また，図 5.6 のように Value が集約されたため，クエリ数を減らしつつ成功率を向上できたといえる．一方で，図 5.7 に示すようにメッセージ数が階層化したものだけ増加している．これは階層化の数だけ生存確認を行う必要があるためである．表 5.2 から増えたメッセージ数は約 100 であることが確認でき，階層化によって 11.4%，DHT と比較して 10.0%の増加となった．

Weibull 分布による実験結果

5.1.1 と同様に，図 5.8, 5.9, 5.10, 5.11,5.12,5.13 にそれぞれ実験結果を示す．Weibull 分布を用いた場合でも，5.1.1 の実験とそれほど差はない．提案手法が性能を向上させていることが分かる．

正規分布による実験結果

最後に正規分布を用いた場合について図 5.14, 5.15, 5.16, 5.17,5.18,5.19 にそれぞれ実験結果を示す．正規分布を用いた場合，他の分布に比べて Churn の頻度が低いため，全体的に検索の成功率が高い．また，ノードの保持する Value の数も生存時間に対してそれほど増えてはいない．これは，上位層にノードが多く割り当てられたためと考えられる．しかし，正規分布であっても従来の手法と比較して提案手法の検索は成功率が高いことが分かる．

5.1.2 実験結果の考察

実験の結果から，提案手法は従来手法に比べて少ないクエリ数で成功率の高い検索機能を提供できたといえる．また，Value が集約して配置されていることも確認できたため，初期クエリとして時刻を指定し，Value の絞り込みを行えば複雑な検索機能であっても少ないクエリ数で提供することが可能である．ノードの

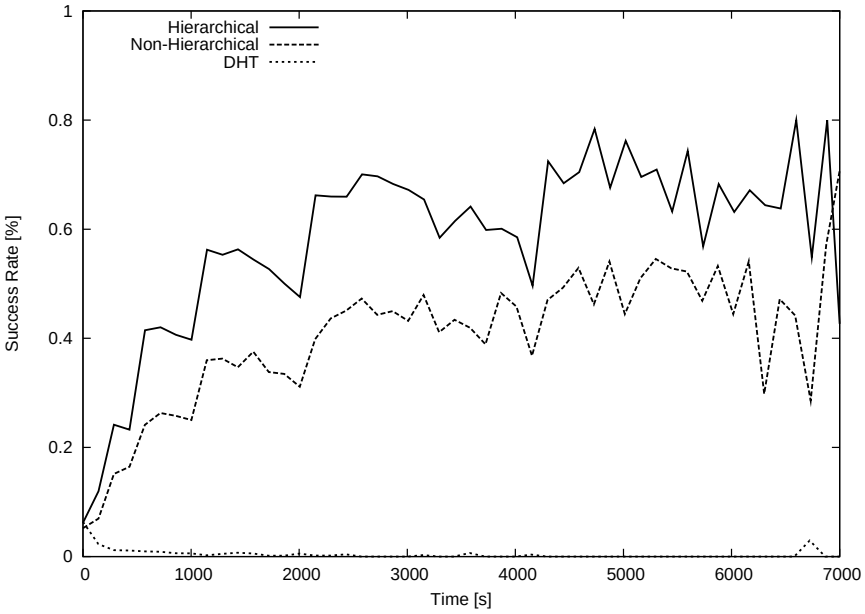

図 5.2 対数を用いた分布の場合の検索成功率 (60 秒の範囲検索)

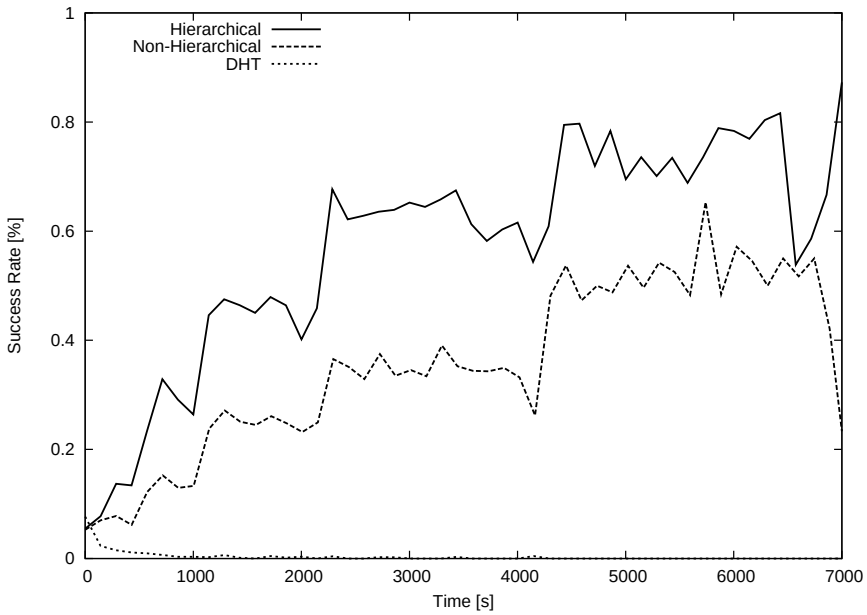

図 5.3 対数を用いた分布の場合の検索成功率 (120 秒の範囲検索)

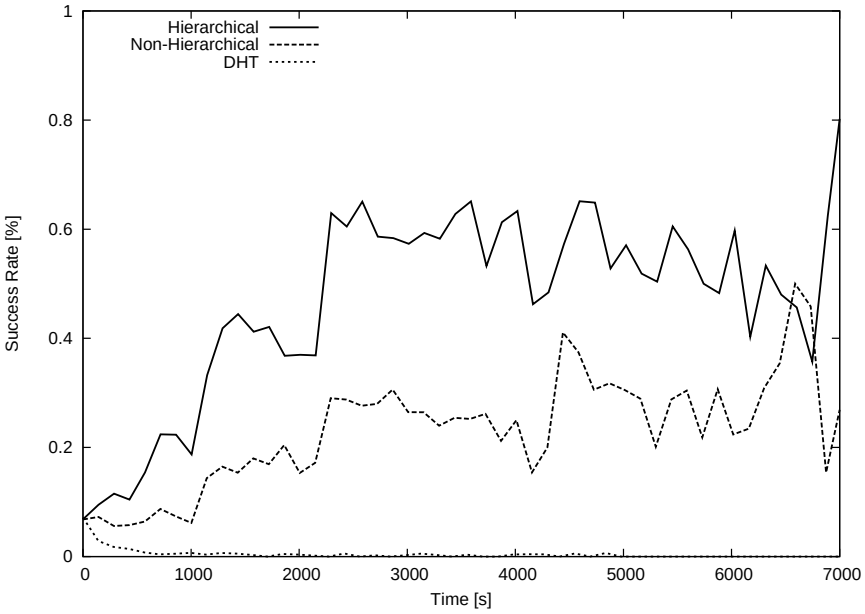

図 5.4 対数を用いた分布の場合の検索成功率 (180 秒の範囲検索)

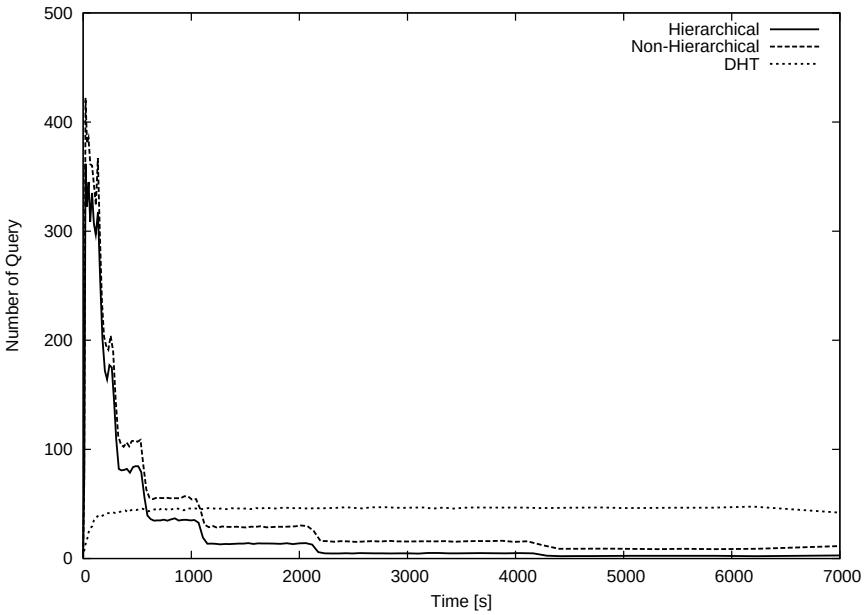

図 5.5 対数を用いた分布の場合のクエリ数

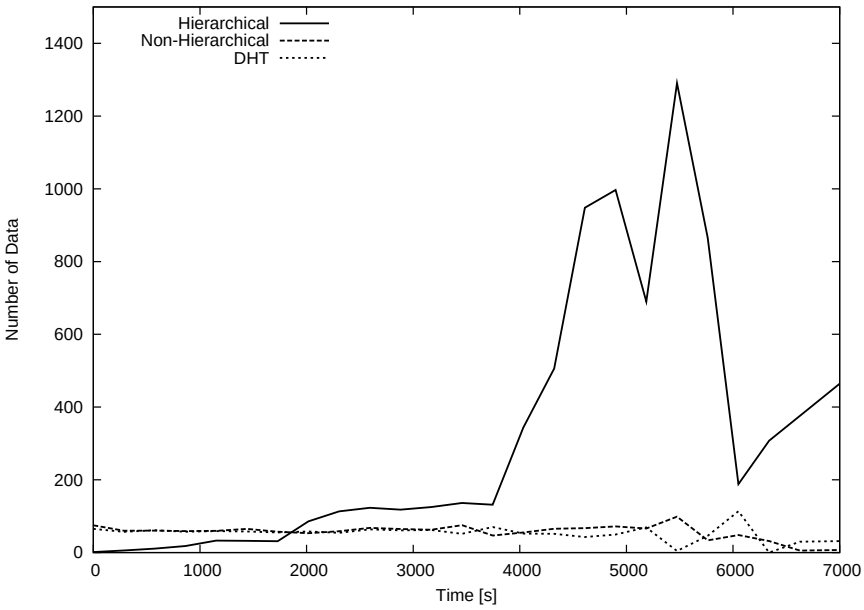

図 5.6 ノードの生存時間と Value 数の関係

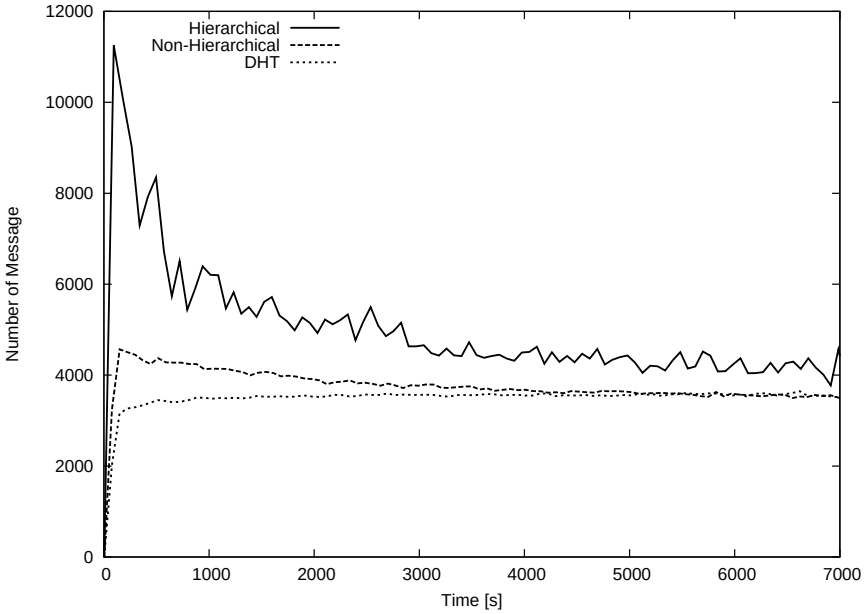

図 5.7 メッセージ数の時間経過による変化

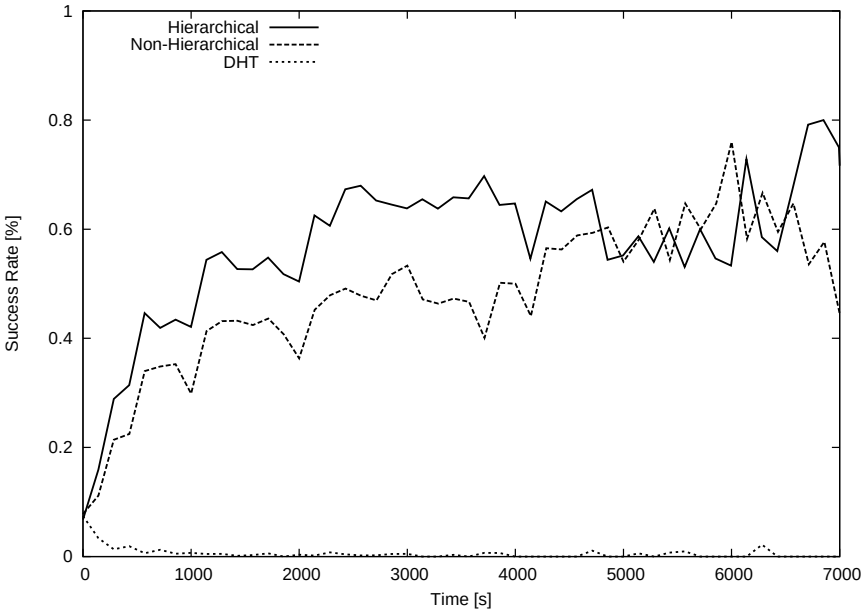

図 5.8 Weibull 分布の場合の検索成功率 (60 秒の範囲検索)

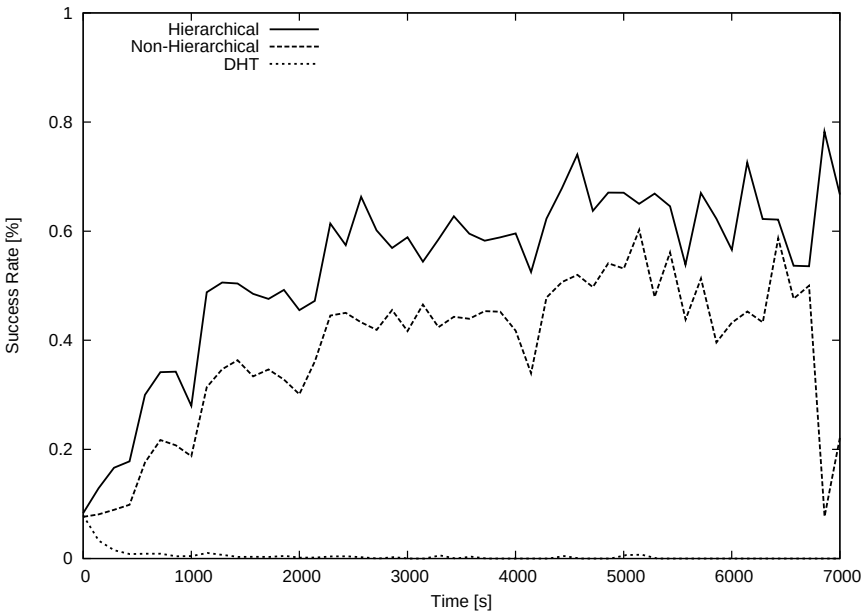

図 5.9 Weibull 分布の場合の検索成功率 (120 秒の範囲検索)

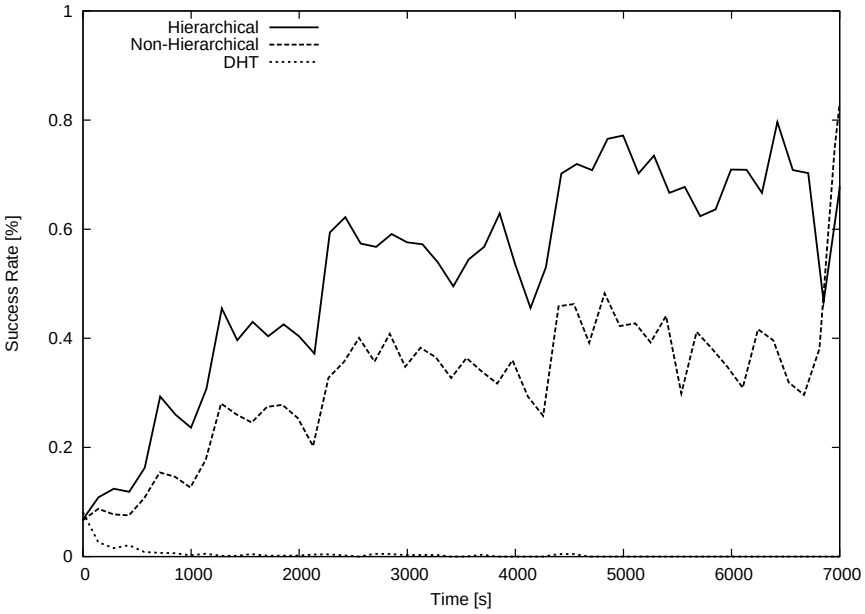

図 5.10 Weibull 分布の場合の検索成功率 (180 秒の範囲検索)

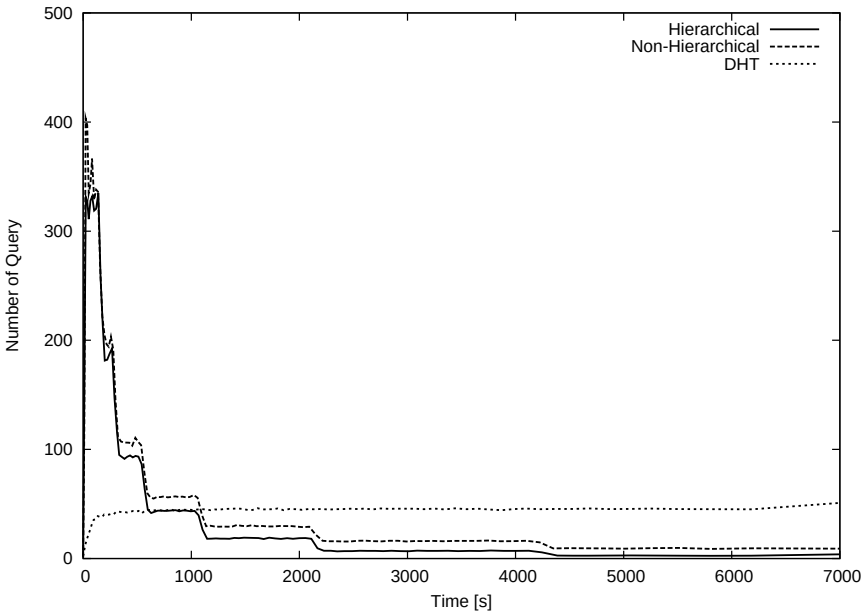

図 5.11 Weibull 分布の場合のクエリ数

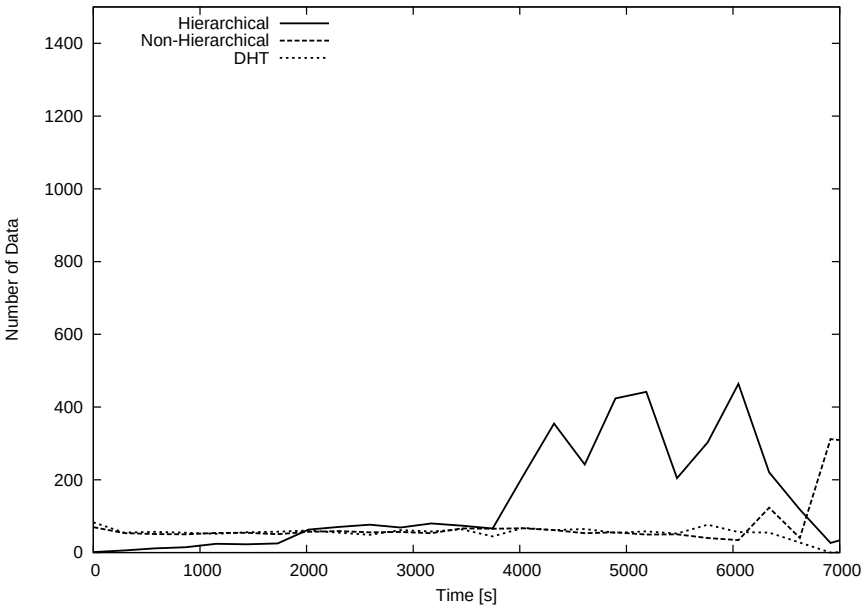

図 5.12 ノードの生存時間と Value 数の関係

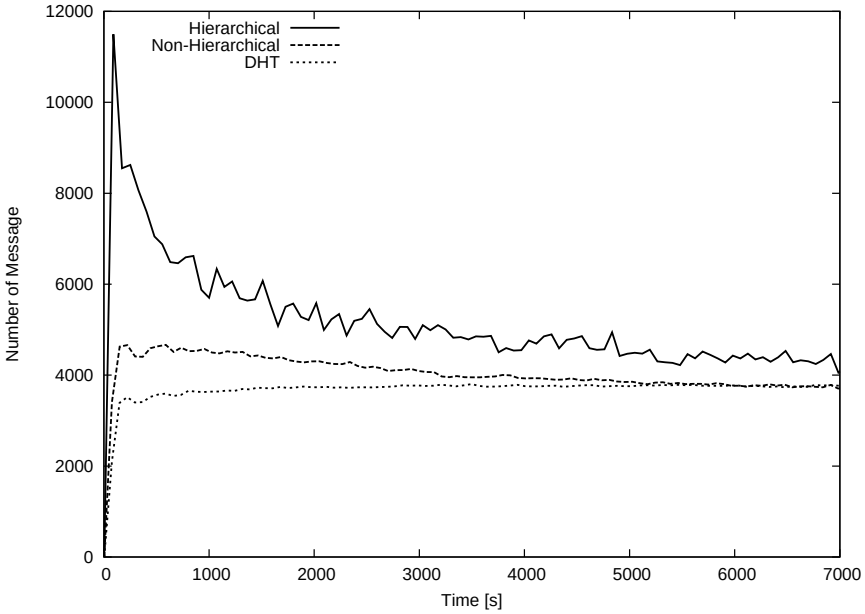

図 5.13 メッセージ数の時間経過による変化

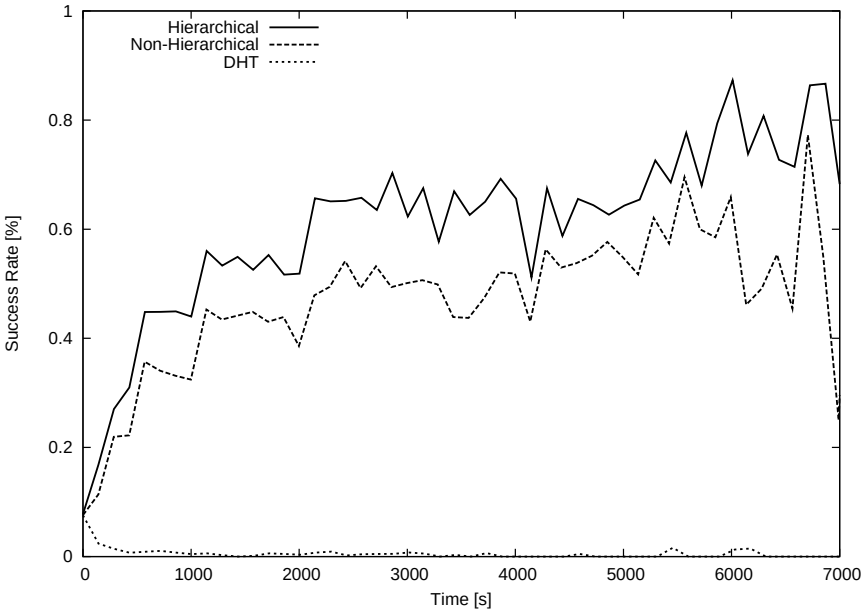

図 5.14 正規分布の場合の検索成功率 (60 秒の範囲検索)

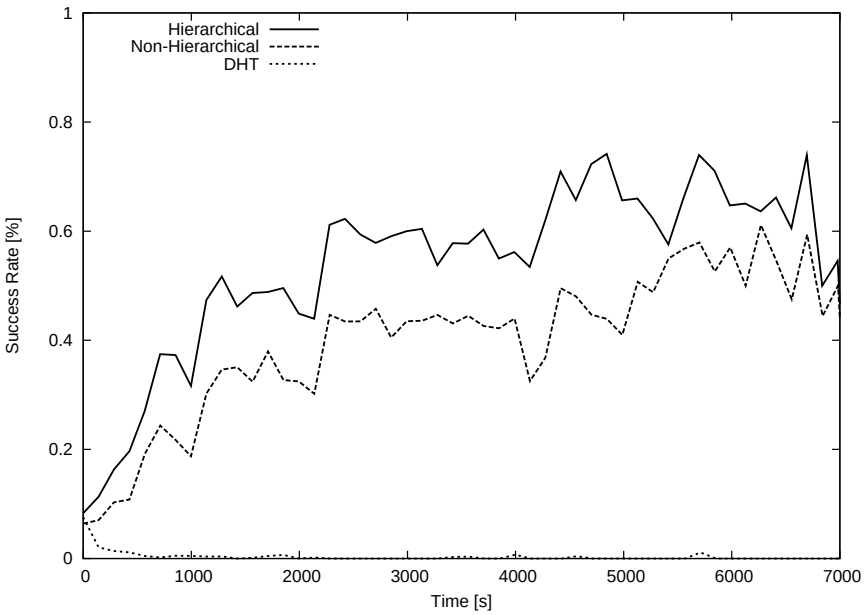

図 5.15 正規分布の場合の検索成功率 (120 秒の範囲検索)

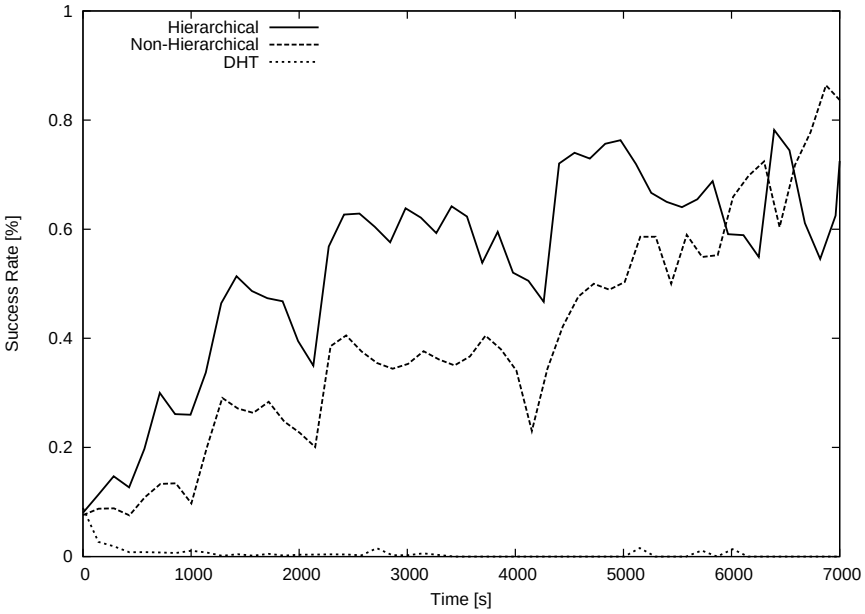

図 5.16 正規分布の場合の検索成功率 (180 秒の範囲検索)

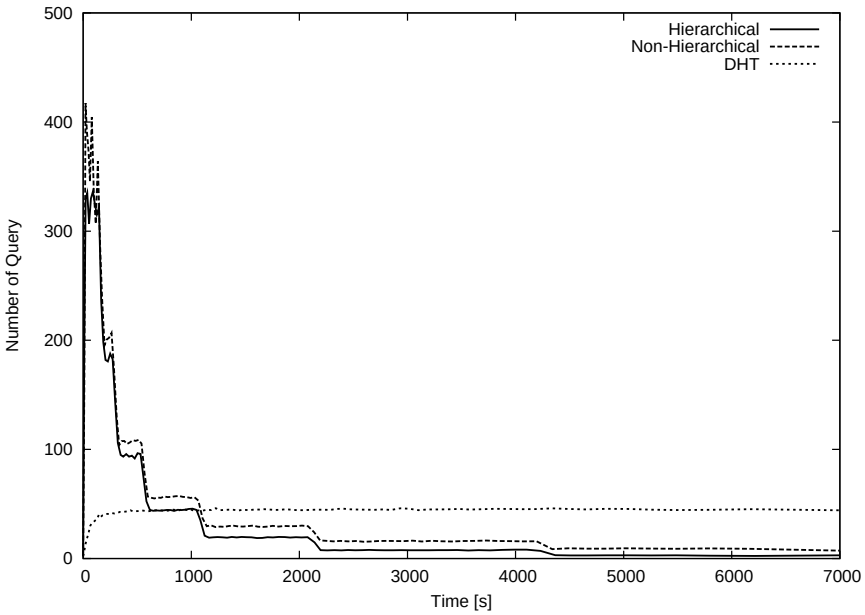

図 5.17 正規分布の場合のクエリ数

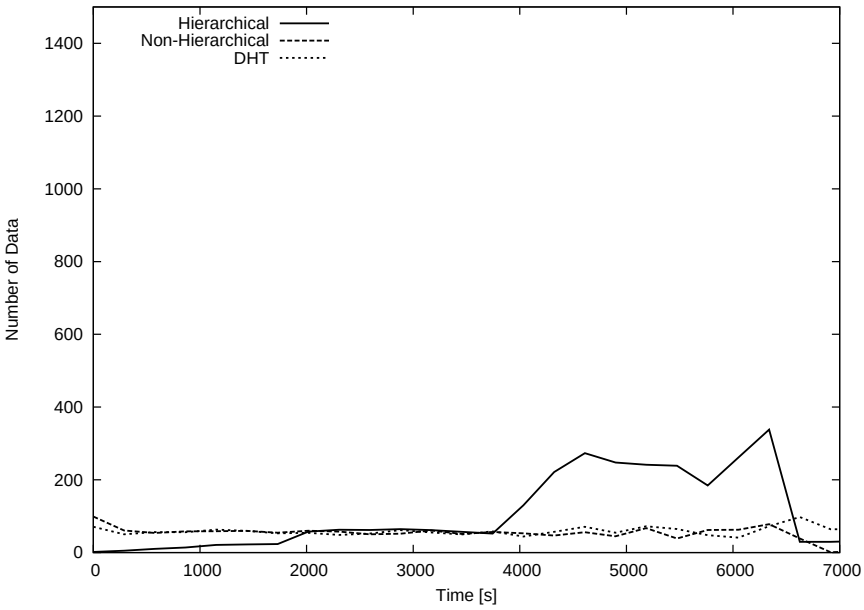

図 5.18 ノードの生存時間と Value 数の関係

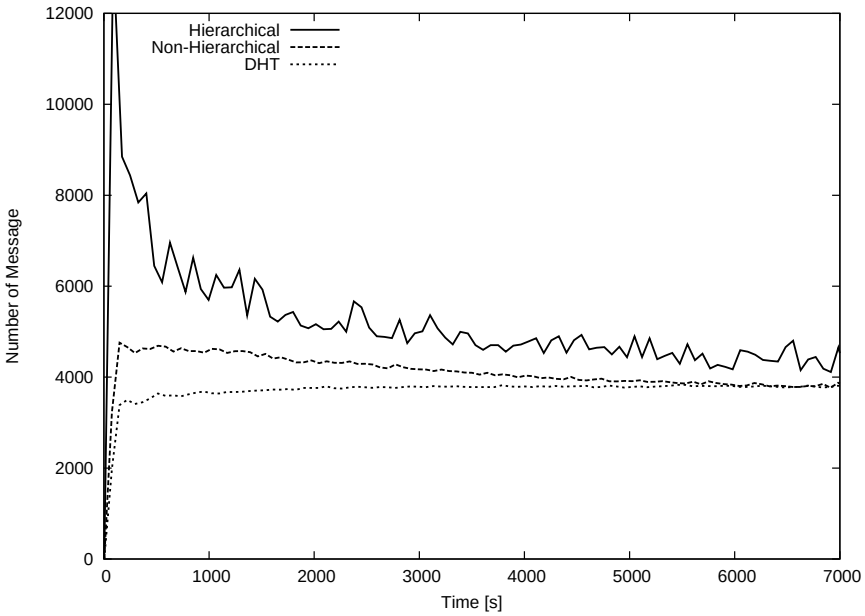

図 5.19 メッセージ数の時間経過による変化

表 5.2 検索結果の平均値

対数を用いた分布			
実験方式	階層化	非階層化	DHT
成功率 (%) 60 秒	45.0	29.8	1.1
成功率 (%) 120 秒	39.5	22.5	1.2
成功率 (%) 180 秒	33.9	15.6	1.3
クエリ数	59.6	78.5	43.4
メッセージ数	5010.5	3770.6	3503.0

Weibull 分布			
実験方式	階層化	非階層化	DHT
成功率 (%) 60 秒	45.6	36.0	1.4
成功率 (%) 120 秒	40.2	28.1	1.4
成功率 (%) 180 秒	35.8	22.0	1.4
クエリ数	64.6	78.0	42.5
メッセージ数	5177.9	4047.4	3687.5

正規分布			
実験方式	階層化	非階層化	DHT
成功率 (%) 60 秒	47.0	36.6	1.4
成功率 (%) 120 秒	40.3	27.9	1.2
成功率 (%) 180 秒	37.8	23.2	1.5
クエリ数	64.5	78.6	42.3
メッセージ数	5195.0	4113.1	3713.5

生存時間の分布にもほとんど影響することなく，階層化と集約による効果を実験結果から得ることができた．そのため，実際にサービスとして利用した場合に，想定と異なる生存時間分布であったり，分布が変化するような状況であっても階層化と集約による検索成功率の向上を期待することができる．

一方で，メッセージ数が従来の手法に比べて約 10%増加した．これは階層化したことによって，各階層での安定化動作を行う必要が出たため，ノードの生存確認を行うメッセージ（Stabilize メッセージ）が増加したことが原因である．しかし，Stabilize メッセージは単純なメッセージであるため，ネットワーク上を流れるトラフィックは小さく，ノードの動作もメッセージの送信元に返信するだけであり負荷も非常に小さい．例えば Stabilize メッセージの代わりに ICMP を用いることを考え 1 メッセージあたり 32 バイトと想定すれば，100 メッセージの増加は 3.2 キロバイト程度となり，現状のインターネット環境ではほとんど無視することができる．このことから，安定化動作による Stabilize メッセージの増加はオーバーレイネットワークおよび実ネットワークの性能にほとんど影響せず，大きな欠点ではないと考えられる．また，どの分布においても時間経過に伴ってメッセージ数の差異が小さくなっていることがわかる．これは，Value を配置する際に同一宛先への Value はまとめて送信されるため，時間経過によって階層化が進めばまとめられる Value の数が多くなり，メッセージ数が削減するためである．複数の Value を圧縮するなどすれば，実際にネットワーク上を流れるトラフィックは逆転することも考えられる．

また，提案手法は上位層へ負荷を集中させるため，上位層のノードが限界に達すればオーバーレイネットワーク全体の性能に影響する．本論文ではサービス提供者が高性能計算機を投入することを前提としているため，上位層に配置されたノードの限界は従来のサーバの限界として扱うことができ，1 サーバあたり 1 万台程度のクライアント接続により限界に達すると想定できる．こうした場合はサービス提供者が新たな計算機を投入することで対処できる．従来の DHT であれば負荷の集中する Key があったとしても，その Key を担当するノードの ID を得ることや，事前にそれを知ることも困難であるため，計算機の投入によって負荷を軽減することが難しい．このことから，上位層へ負荷を集中させることは提案手

法の利点と考えることも出来る.

5.2. 階層化手法の比較評価

本節では式 3.1 に挙げた手法に加え，以下の 2 つの手法を用いて階層化を行い，比較評価を行った.

定数に分割した階層化　階層の数を L とし，最も古いノードの参加時間 T_{max} としたとき，参加時間を $\frac{T_{max}}{L}$ 毎に分割することで階層化を行う．実験では提案手法を用いたときに最大となる L を用いて階層化を行った.

平方根を用いた階層化　式 3.1 に用いた対数の代わりに平方根を利用し，$L = \lfloor\sqrt{Now - t}\rfloor$ とした階層化を行う．なお，実験では提案手法と同じ階層数になるように調整を加えた上で階層化を行った.

5.2.1 実験結果

提案手法と定数，平方根の手法について実験を行った．実験結果を以下に示す.

対数を用いた分布の実験結果について，図 5.20 に検索成功率を, 図 5.21 にクエリ数を示す．また，Weibull 分布を用いた場合について図 5.22，図 5.23 に実験結果を示し，正規分布を用いた場合について図 5.24，図 5.25 に実験結果を示す．図では log が提案手法を表し，int が定数による階層化，sqrt が平方根を用いた階層化を示している．実験の結果から，全ての分布において対数を用いた提案手法の検索成功率が他の手法に比べて優れていることが分かる．一方でクエリ数に関しては他の手法の方が少ないが，古い時刻になるにつれてその差はほとんど無いことが分かる.

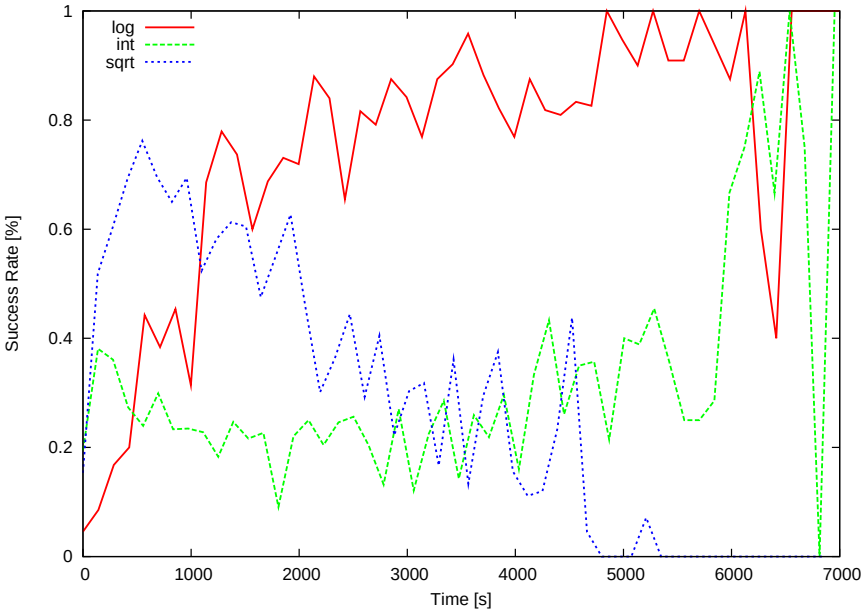

図 5.20 対数を用いた分布の場合の検索成功率 (階層化手法による差異)

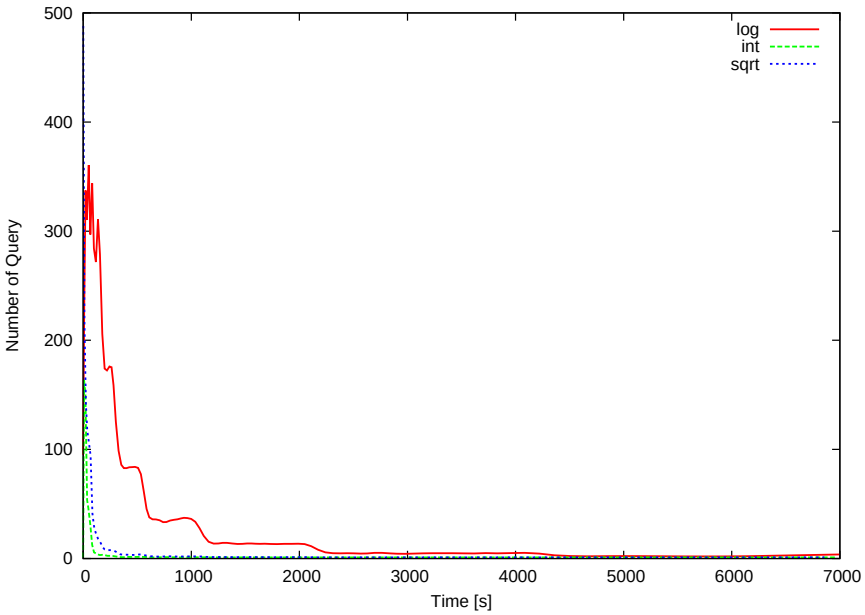

図 5.21 対数を用いた分布の場合のクエリ数 (階層化手法による差異)

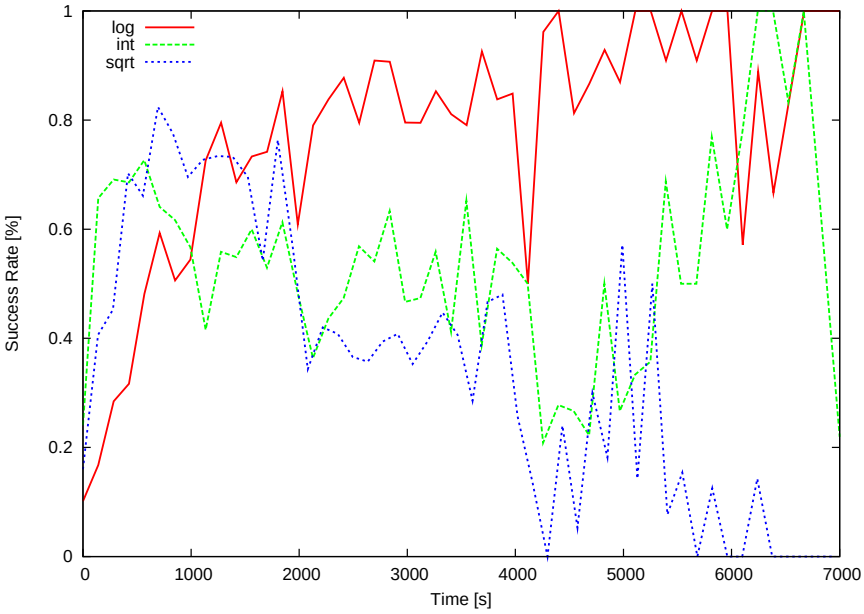

図 5.22 Weibull 分布を用いた場合の検索成功率 (階層化手法による差異)

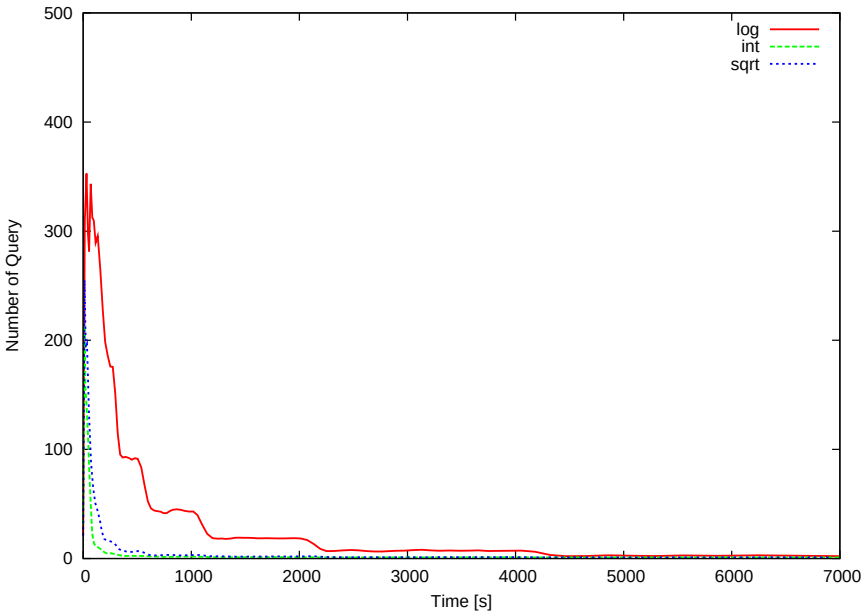

図 5.23 Weibull 分布を用いた場合のクエリ数（階層化手法による差異）

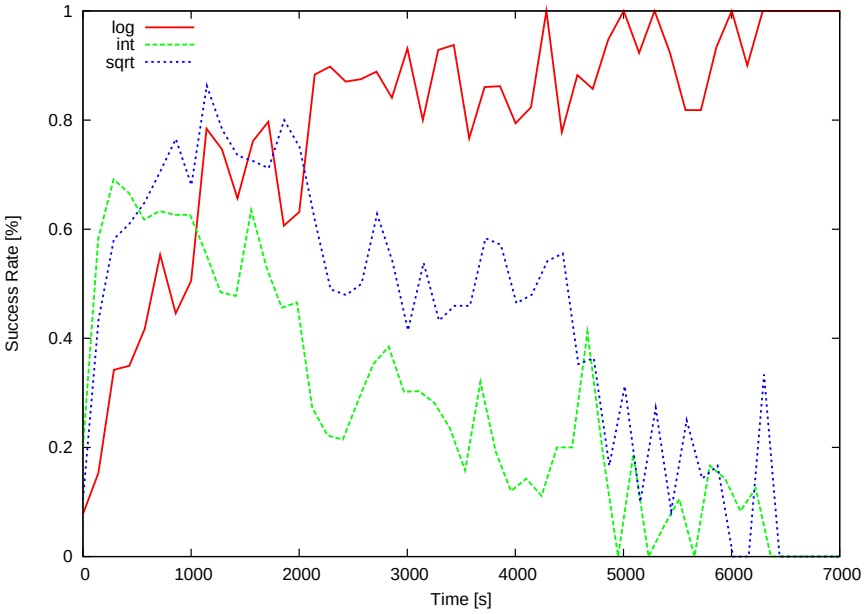

図 5.24　正規分布を用いた場合の検索成功率 (階層化手法による差異)

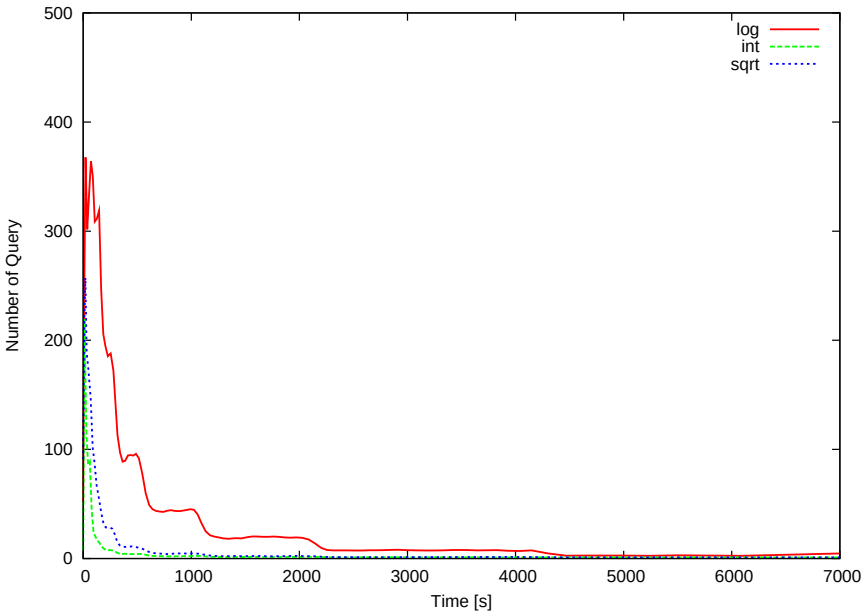

図 5.25　正規分布を用いた場合のクエリ数（階層化手法による差異）

5.2.2 実験結果の考察

実験の結果から，対数を用いた手法でなければ検索成功率を向上できないことが分かった．これは，Key-Value の Put に用いる対数を用いた関数との親和性に関係している．Key-Value は時間経過に従って配置されるノードが変化し，近い時刻の Key-Value と共に集合体へと変化していくが，この集合体への変化は関数内の対数によって決まるため，経過時間に対して対数的に集合体へと変化していく．これに対してノードの階層も経過時間に対して対数的に変化していくため，Key-Value が集合体へと変化する時期とノードの階層が変化する時期が同期することで安定したノードへ Key-Value の集合体を配置することが可能となる．そのため，定数や平方根を用いた場合では階層が変化する時期が早すぎ，安定したノードへ集合体を配置することができない．また，逆に変化する時期を遅くしすぎた場合には Churn の激しいノードを上位層に招くことになり，検索成功率が低下する．このため，対数を用いた手法による階層化と Key-Value の配置手法によって提案手法は検索成功率を向上させていることが分かる．

5.3. アプリケーションへの応用

本節では時刻を Key に指定したオーバーレイネットワークの応用例として，写真を共有する P2P アプリケーション「PSP2P」について述べる．

5.3.1 概要

アプリケーション開発の背景

デジタルスチルカメラ（以下,デジタルカメラ）の低価格化やパソコンの普及に伴い, 高画素なデジタルカメラが一般に広く普及してきている．現在ではコンパクトなデジタルカメラでさえ 1000 万画素を超えるものが登場している他, 一眼レフタイプのデジタルカメラでは 2000 万画素を超えるものも登場している．このようなデジタルカメラで撮影した高画素写真は,JPEG 形式であっても数メガバイトのファイルサイズになり,RAW 形式では数十メガバイトにも達する．また, デジタ

ルカメラが従来のフィルムカメラと大きく異なる点として, 膨大な量の写真を撮影できる点が挙げられる. 従来のフィルムカメラはフィルムの枚数によって撮影できる写真の枚数が制限されていた. しかし, デジタルカメラの場合は写真をカードメディア等に保存するため, カードメディアが大容量化を続けた近年ではかなりの枚数を撮影しても容量が不足するようことは少なくなってきている. さらに, フィルムの場合は現像しなければ写真を見ることができないが, デジタルカメラで撮影した写真はパソコン等での閲覧を行えるため現像する必要がない. こうした手軽さから, デジタルカメラによる写真の枚数は増加し, 膨大な量の写真データが世界には存在していると想像できる.

こうした膨大な量の写真データをインターネットを用いて交換・共有することで, 様々なサービスを展開することが考えられている. 一般的なインターネット上のサービスと同様に, 写真を共有するサービスもサーバを用いた構成が多い. 利用者はサーバに撮影した写真をアップロードし, 閲覧者はサーバにある写真を閲覧する. しかし, サーバ側の負荷やネットワーク速度などを理由に, アップロードできる写真の枚数は制限されている場合が多い. また, ユーザが多量の写真をアップロードするためには非常に長い時間を要する. こうしたインターネットの性能による写真データ共有における制約は, 前述した写真の枚数が増加している現状と合っていない.

また, 多数の写真を共有したとき, どのようにして目的の写真を検索するかといった問題もある. ファイルや文章データの場合, 一般的にはファイル名や文章中の単語等から検索を行う. しかし, 写真データのファイル名には有意な名前が付けられているとは考えにくく, また多数の写真データ１つ１つに対してユーザが名前を付けるとも考えにくい. ウェブサービス上で展開されている写真共有サービスでは, ユーザによってつけられたタグ等による検索が行われているが, 写真の枚数が膨大な量になったとき, それらに対して有意なタグが付与されることも考えにくい.

撮影時刻を用いた写真の検索

写真を共有する上で以下の３つの問題点が残されている.

- ユーザが写真に新たな情報を加えることなく検索機能を実現する
- ユーザは多量の高画素写真を閲覧できる
- 写真のアップロードにかかる時間を短縮する

この3つの問題点に対して，次のような解決策を試みるアプリケーション（PSP2P:Photo Sharing P2P Network）を開発した．

- 写真の撮影時刻を元に検索機能を提供する
- 高画素の写真は写真の提供者から P2P で直接ダウンロードする
- ノード間ではサムネイル画像を共有することによりアップロードの時間を短縮する

前述の通り写真データのファイル名を検索に用いることは考えにくい．そこで, 写真にあらかじめ付与されているメタデータを用いて検索機能を提供する．一般的な写真データには Exif(Exchangeable Image File Format）形式によるメタデータが付与されている．近年ではデジタルカメラだけではなく，携帯電話などで撮影した写真データであっても Exif 形式のメタデータが付与されることが多い．Exif 形式のメタデータは撮影したカメラに関する情報や，撮影時の情報としてシャッター速度・絞り・焦点距離などの記載が定められている．これに加えてカメラの内蔵時計を元にした撮影時刻の記載が定められているため，ほとんどの写真データは Exif 形式のメタデータを解析することで写真の撮影時刻を特定することが可能である．撮影した時刻は万人に対して共通に認識できる指標であるといえるだけでなく，「同時刻帯に何が起きていたか」ということを検索することも可能にする，有用な情報であるといえる．

図 5.26 に PSP2P による写真共有の概要を示す．写真を提供するユーザはまず写真のメタデータとサムネイル画像をオーバーレイネットワークに登録する (1). ユーザはこのオーバーレイネットワークに対して撮影時刻を指定した検索のクエリを送信 (2) し, 検索結果のサムネイル画像とメタデータを取得する．この結果の中から取得したい画像を選出し, 高画素の写真データを写真を提供する利用者のノードから直接ダウンロードする (4).

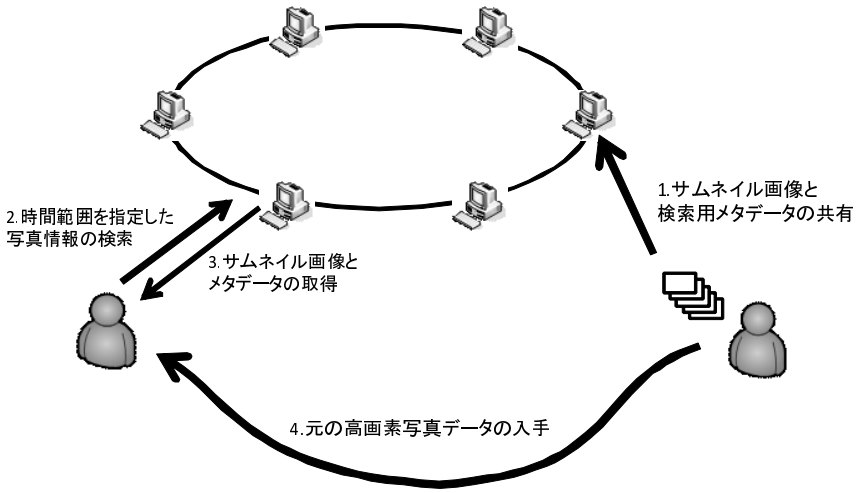

図 5.26 PSP2P による写真共有の概要

このように PSP2P では写真の検索時に高画素の写真データが必要でない点に着目し, サムネイル画像とメタデータのみを検索の対象とすることで, 写真のアップロード時間を短縮した. 写真のメタデータやサムネイル画像は数百 Kbyte 程度と非常に容量が少ないため, 写真のアップロードにはほとんど時間がかからない. 高画素な写真データは直接ノードからダウンロードすることにより, 必要な写真データを必要な利用者だけに提供することができる. また, 撮影された時に付与される Exif 情報を元にオーバーレイネットワーク上での Key-Value を作成するため, 写真の提供者はアップロード時に特別な操作やタグを付けるといった行為無しに写真を共有することができる.

5.3.2 PSP2P の実装

PSP2P は 4 章で述べたツールを用い，オーバーレイネットワークの機能拡張によって実現した．PSP2P の実装にはユーザインタフェースとして GUI や HTTP サーバなどが含まれるが，写真の共有を行うためにオーバーレイネットワークに施した拡張は P2PData クラスを継承した PhotoData クラスの実装と，PhotoData オブジェクトを生成する PhotoDataCreator クラスの実装である．以下に拡張の

方法について述べる．また，付録 A にユーザインタフェースに関する実装を記載する．

PhotoData クラスの実装

PhotoData クラスは P2PData クラスを拡張して実装した．PhotoData クラスには以下のエントリが含まれる．

- File:originalFile
  元の写真ファイルを示す File オブジェクト

- File:thumbFile
  サムネイル画像のファイルを示す File オブジェクト

写真の撮影時刻は P2PData クラスのエントリとして記述される．PhotoData クラスはコンストラクタに元の写真ファイルを示す File オブジェクトを指定することでインスタンスを作成する．元の写真ファイルを読み込み，Exif メタデータを解析することで写真の撮影された時刻を取得する．時刻は P2PData クラスの時刻情報に記載する．元の写真ファイルは画素数の多い物になると 5MB を上回るものもあるため，P2PData オブジェクトの内部に持つべきではない．また，サムネイル画像は 1 つのファイルであれば数百 KB 程度であるが，写真の枚数が増えるとメモリを圧迫する原因になる．そのため，PhotoData クラスのエントリはファイルへのポインタを示す File オブジェクトを利用する．また，File オブジェクトは Java の直列化を行っても転送されない上に，File へのパス情報などを他のノードに伝えることは好ましくない．そこで，PhotoData クラスでは readObject メソッドと writeObject メソッドをオーバーライドし，サムネイル画像のファイルを宛先ノードに転送する．宛先ノードではファイルを受け取るとあらかじめ定めた Cache ディレクトリにファイルを格納し，そのファイル情報を thumbFile として上書きする．

PhotoDataCreator クラス

利用者が GUI などでディレクトリを指定し，PSP2P はそのディレクトリ内にある写真を共有する．PhotoDataCreator クラスは指定されたディレクトリ内を定期的に確認し，写真ファイルを用いて PhotoData オブジェクトを生成する．生成した PhotoData オブジェクトはデータストアに追加することで，オーバーレイネットワーク上で検索可能になる．また，一度に作成できる PhotoData オブジェクトには上限を設ける．これは一度に PhotoData オブジェクトを大量に生成すると，オーバーレイネットワークへの Put 動作が一度に大量に発生するためである．ファイルサイズの小さな Value であれば大量に Put 動作を行っても問題がないが，PhotoData オブジェクトの Put 動作ではサムネイル画像を送信する必要があるため，一度に大量に Put 動作を行うことで計算機の負荷を高める可能性があるためである．

第6章　階層化オーバーレイネットワークによるサービスに関する考察

インターネット上で提供されるサービスの内，P2P 方式のアプリケーションはクライアント・サーバ型と比べて非常に少ない．VoIP を無料で提供している Skype[5] や，サイズの大きなファイルのダウンロードを支援する BitTorrent[1] などは著名であり，他にもストリーミングによるライブ配信を行う UG Live / Ocean Grid[6] などはP2P 型のアプリケーションとしてサービスの提供を行っている．しかし，ファイル共有ソフトとして開発された P2P アプリケーションでは常に著作権の侵害に関わる問題や情報漏洩の問題が発生していることもあり，一般的なサービスへ利用されている事例は少ない．本章では 3 章で提案したオーバーレイネットワークによって実現できる P2P 方式のアプリケーションについて議論し，現状の P2P 方式のアプリケーションにおける問題点を解決する．

6.1. 時刻属性を利用したアプリケーション

3 章で提案したオーバーレイネットワークによって，時刻属性による情報の共有が可能である．時刻属性は現実世界に強い関連のある属性であるため，現実世界での情報を共有するアプリケーションに利用することができる．例えば PSP2P では現実の世界をデジタルデータ化した写真データの共有を行った．これが誰かの書いたイラストデータであれば，時刻属性による検索はほとんど利用されないと考えられる．同様に映像のような情報も，現実世界を撮影した映像であれば時間による検索対象となるが，空想の映像作品では時刻属性による検索を期待でき

ない．他にもセンシングデバイスの生成するセンサデータや，利用者の行動履歴，ニュースや日記などの記事などが挙げられる．

時刻属性を利用して情報を共有する利点は，写真や映像，行動履歴やニュースといった全く別の情報であっても時間という同じ意味論での共有が行える点である．ニュースなどの時刻で検索することで，ニュースに関する写真や映像などを取得できる他，同じ時間帯の別のニュースなども取得することができる．また，P2P 型のアプリケーションはサービスを利用するために利用者にアプリケーションをインストールしてもらう必要がある．また，数百万程度の利用者が集まらなければクライアント・サーバ型に比べた利点を見いだせない．そのため，P2P 型アプリケーションの普及には 1 つのアプリケーションで様々な用途に利用可能な汎用的基盤の構築が必要である．こうした汎用的基盤の作成に現実世界で共有の要素である時刻属性は十分に利用できると考察できる．

6.2. 負荷の公平な分散

3 章で述べた階層化オーバーレイネットワークでは，以下の環境を想定とした負荷の分散を行った．

1. ノード数が参加時間に対して対数的に減少（$-\log_2 T$）
2. 参加時間に対して比例した Key-Value の配置

5 章の実験結果より，ノード数の参加時間分布は想定と違ったものであっても提案手法が他の手法に比べて優れた検索成功率であった．また，ノードに対して割り当てる Key-Value の数は利用するサービスによって異なることが予想されるため，どのようなサービスに対しても最適な Key-Value の割り当てが出来るわけではない．そのため，オーバーレイネットワークを実現するアプリケーションでは，負荷の調整を可能とする必要がある．

6.2.1 仮想ノードを用いた負荷の調整

Chord などの Consistent-hashing 法では計算機上に仮想ノードを作成することで負荷の割り当てを可変にすることができる．Consistent-hashing 法などの暗号学的ハッシュ関数を均等な負荷分散に用いている手法ではノード数が少ない場合に負荷が偏るという特徴がある．これは，暗号学的ハッシュ関数は ID 空間を均等に分割するものではないため，特にノード数が少ない場合，割り当てられる ID の領域に大きな差異が生じるためである．この負荷の偏りはノード数が増すことで次第に緩和されていくが，少なくとも 100 ノード程度が参加しないと負荷に大きな偏りを生じさせる可能性がある．そこで，1 台の計算機上に複数のノードを作成することで，実際の計算機間での負荷を均等に分散することができる．この 1 台の計算機に割り当てられた複数のノードを仮想ノードと呼ぶ．

また，1 台の計算機に仮想ノードを多く割り当てることで負荷を多く割り当てることにも利用することができる．計算機性能に対して割り当てられた Key-Value の数が少ないのであれば，仮想ノードを作成することで割り当てられる Key-Value の数が増し，計算機性能や利用者の要求に見合った負荷を割り当てることができるようになる．しかし，この手法は計算機が障害などで離脱した場合，一度に多くのノードがオーバーレイネットワークから離脱することになるため，Put された Key-Value を喪失するだけでなくオーバーレイネットワークの構造そのものを破壊しかねない．また，サービスの利用者が増え，共有するべき Key-Value の数が増したときに負荷を減らす目的で仮想ノードを削除したときも同様の問題が起こり得る．

提案手法では，参加時間の長いノードに対して Key-Value の割り当て量が増すという本質的な特性がある．そのため，仮想ノードを投入することによる負荷の増加が大きく，少ない仮想ノードで負荷の調整を行うことが期待できる．従来の手法よりも少ない仮想ノードで負荷の調整が行えるため，前述したノードの離脱時に構造を破壊することが無いため，オーバーレイネットワークに与える影響を少なくすることができる．

6.2.2 階層の上限と下限による負荷の制限

提案手法では時間の経過に伴って割り当てられるKey-Valueの割り当て量が増加するため，ノードが過負荷状態になる可能性がある．これに対して，ノードは参加する階層を制限することで負荷の制限を行うことが出来る．ノードはあらかじめ階層の上限値を定め，ノードのIDと共に他のノードへ伝えておくことで，ノードは上限値よりも上位の階層へと属することがなくなる．また，逆にサーバのような高性能の計算機は，下限値を設けることで初期の段階から上位層へ参加し，Valueの割り当て量を増やすことができる．なお，この上限値と下限値は参加者の意図を反映させないために，自動的に設定されるような仕組みやサービス提供者のみが設定できるような枠組みが必要である．こうした枠組みはブートストラップノードなど，サービス提供者の用意する計算機を用いることで実現可能である．

6.2.3 対数の底を利用した負荷の調整

提案手法では式3.10を用いることで上位層のノードに対して優先的にKey-Valueを割り当てている．しかし，式3.10以上に上位層にKey-Valueを割り当てたい場合などがある．サービス提供者の用意する計算機が多ければ，上位層により多くのKey-Valueを割り当てることで検索の成功率をより向上させることができる．また，逆にサービス提供者の用意する計算機が少ないため，検索の成功率を犠牲にしても負荷を減らしたい場合なども考えられる．こうした場面では，式3.10で用いている対数の底を調整することで割り当てられるKey-Valueの量を調整することが可能となる．対数の底を調整できるように改良した関数を以下に示す．

$$H_T(t,b) = \frac{t \mod (b-1)b^{L(t,b)}}{(b-1)b^{L(t,b)}} \tag{6.1}$$

$$L(t,b) = \lfloor \log_b(Now - t) \rfloor \tag{6.2}$$

式6.1と式6.2では引数として時刻と底を表すbを取り，時刻tのKey-ValueをPutするIDと階層Lが定まる．提案手法は$b = 2$の場合であるともいえる．bに

は1より大きな任意の数を用いることができるため，割り当てに対して細かな調整を可能とする．

対数の底を調整することによる負荷の調整は割り当てられるKey-Valueを細かく調整することができるが，サービスの開始時にbを指定する必要があるため，サービスの提供中に変更することが難しい．サービス提供者はサービスの性質や投入する計算機の性能や数などを考慮し，あらかじめbの値を設定する必要がある．

6.2.4 まとめ

提案手法を用いて負荷の調整は，以下の手順によって行うことができる．

対数の底を調整したサービス開始　サービスの開始時に，投入する計算機の性能や数などから適切な対数の底を定め，オーバーレイネットワークを構築する．

仮想ノードによる各ノードでの調整　性能に対して割り当てられたKey-Valueが少ない場合は仮想ノードを作成することで負荷を割り当てることが可能となる．

上限値と下限値による負荷の制限　階層の上限値や下限値を各ノードが設定することで，負荷の増加によるノードの離脱や，新しく投入した計算機に対する負荷の割り当てなどを行う．

6.3. 多属性を指定した複雑な検索機能の提供

情報の検索には様々な属性を指定することが考えられ，また正規表現などの複雑な検索がサービスには求められることが多い．一般的なP2P方式のサービスでは情報が分散していることからそうした複雑な検索機能の提供は困難であった．しかし，提案手法で示した手法では情報が分散せずに集約する特性があるため，集約された情報に対しては従来のサーバに対する検索のように複雑な検索を可能にすることができる．

これは他のオーバーレイネットワークの手法と比べても特に有益となる特性であると考察できる．一般的なオーバーレイネットワークでは負荷の均等な分散が考えられているため，情報を集約することによる複雑な検索機能の提供は考えられていない．しかし，構造を複雑にすることで実現される複雑な検索機能はノードの Churn 動作による性能劣化を避けにくいことや，限定された用途に対する検索機能である場合が多い．本論文で提案する手法は，情報の集約による複雑な検索機能の提供を可能としており，これは情報の対象を時刻属性に関連のある現実世界のものに限定しているが，社会の中で広範囲に利用できる属性であると考察することができる．

6.4. オーバーレイネットワーク技術への貢献

本論文では P2P 型のアプリケーションを実現するために，オーバーレイネットワーク技術に対して以下の貢献を果たした．

時刻属性を利用した汎用的基盤の実現　P2P 型アプリケーションでは情報を分散して管理するため，正規表現やワイルドカードを利用するような複雑な検索を実現することが困難となる．そのため，検索に用いる属性は 1 つに絞り込むことが必要であり，従来のオーバーレイネットワークでは用途によってその属性を定めてきた．しかし，P2P 型のアプリケーションがクライアント・サーバ型よりも優れた性能を示すためには，非常に多くの利用者を募る必要があり，単一のアプリケーションでそれだけの利用者を集めることは難しい．多くの利用者を募るためには汎用的に扱えるオーバーレイネットワークの基盤を用意する必要があるが，既存の DHT ではそれが困難であった．本論文では検索に時刻属性の利用を可能としたため，現実世界から発生する情報に関しては汎用的に扱うことを可能にした．この貢献によって，クエリのフラッディングを用いることなく時刻属性を初期クエリとした複雑な検索を実現することができる．一方で，本論文の提案では時刻属性が検索時に有効となる現実世界の情報に限られるため，時刻属性には関わりのない情報の検索を行うのであれば他の属性を考慮する必要がある．

Churn の影響を受けにくい階層化構造の実現　非常に多くの利用者によってオーバーレイネットワークを構築した場合，ノードの Churn 動作による影響を考慮しなければならない．特にサービス提供者が投入するサーバなどの高性能計算機を考慮すれば，従来の DHT などのように負荷を均等に分散することよりも，ノードの特性に応じて負荷を割り当てる方が現実的であると想定できる．本論文ではノードの参加時間に着目し階層化構造を実現することで，Churn 動作による影響を限定的なものとし，一般の利用者が参加するオーバーレイネットワークを想定しても安定してサービスの提供が可能であることを示した．特にオーバーレイネットワークの研究分野では Key-Value の複製を用いた Churn への対策が多く，提案した Key-Value の配置手法に対する改善は既存の Churn 対策手法と組み合わせることが可能である．そのため，特に Key-Value の配置手法を階層化構造を用いて改良しただけで，Churn による影響を限定的なものにできたことは大きな貢献であるといえる．

6.5. 今後の展望

6.5.1 検索率向上に向けた課題

提案手法によって検索結果の成功率を向上させることはできたが，インターネットサービスの主流であるサーバ・クライアントな環境ではほぼ 100%に近い成功率を実現している．より信頼性の高いサービスを提供できる基盤になるためには検索率の更なる向上が必要である．提案手法は Value の配置に改良を加えた手法であるため，他の Churn 対策手法をそのまま応用することが可能である．例として Value の複製を別のノードへ置く手法がある．これは Value を担当しているノードの離脱時に次の担当となるノードへあらかじめ Value を配置しておく手法である．多くの複製を配置することはトラフィックの増大と Value 管理の難しさが問題となるが，提案手法は上位の階層ほどノードが少なく，また Churn の起きる確率も少ないため，より少ない数の複製を置くだけで再現率の大幅な向上を期待できる．こうした手法を確立し，その効果を評価することは現実的な運用をする上での今後の課題である．

6.5.2 費用対効果における課題

半導体技術の発展による計算機の性能向上と，世界を覆い尽くさんとする電子ネットワーク網の整備によって，サービスの提供者はサーバとなる計算機をインターネットに接続するだけで世界的なサービスを提供することが可能になった．例え非常に需要の少ないサービスであったとしても，インターネットを用いて世界規模に展開することでサーバの運用やネットワーク接続にかかる費用を賄うことが可能になる．サーバレスなP2P型アプリケーションはイニシャルコストやランニングコストを低く抑えることができるが，需要を見越したサービスであればサーバの設置や運用に関わる費用は特に問題とならない．

一方で，公共サービスなどのように収益を上げることは難しいが公益性の高いサービスがある．それらのサービスではサーバの設置に関わる費用や，可用性を確保するために必要な冗長化のコストなどを賄えない場合がある．また，サービス提供における費用対効果が低いサービスがある．写真を共有するサービスがその例である．クライアント・サーバ型で提供されている写真共有サービスでは無料で利用出来るストレージに限りがあり，有料でストレージを確保する必要がある．PSP2Pでは利用者の計算機資源を利用することによって，高画素の写真データであっても利用者間での共有を可能にした．安定したサービスの提供を行うには高性能計算機をオーバーレイネットワークに投入する必要があるが，従来のクライアント・サーバ型と比べればPSP2Pは費用対効果が高いと言える．このように，P2P型のアプリケーションを開発するには，費用対効果を考えなければ実際に利用することは難しい．

6.5.3 多対多通信の情報基盤構築に向けた課題

P2P型のアプリケーションによって多対多通信の情報基盤を構築するためには汎用的に利用できるオーバーレイネットワークとキラーアプリケーションの開発，そして不正な利用者の排除が必要である．

クライアント・サーバ型のサービスは利用者が複数のサービスを同時に利用することができる．利用者はクライアントとなる計算機上でサービスを受けるため

のプログラムを動作させればよい．また，近年ではHTTPによるWebサービス提供が多数を占めるため，プログラムとしてブラウザを起動するだけで多くのサービスを利用することができる．一方でP2P型のアプリケーションはプログラムを配布する形で提供されるため，利用者のインストールにかかる手間などが利用の障壁となっている．そこで，P2P型サービスのブラウザとして，汎用的に利用出来るオーバーレイネットワークのプログラムと，利用者がそのオーバーレイネットワークを利用する動機となるキラーアプリケーションが必要である．汎用的に利用出来るオーバーレイネットワークの基盤が出来れば，クライアント・サーバ型では提供が困難な様々なサービスの投入が期待でき，前述したコストに見合わないが公益性の高いサービスなども期待できる．

一方で，汎用的に利用できるオーバーレイネットワークでは，違法性の高いサービス提供が行われる可能性が少なくない．それらのサービスはP2P型アプリケーションの社会的な印象を悪くし，利用者やサービス提供者が敬遠する原因にもなり得る．また，P2P型アプリケーションは利用者の計算機がサービスの提供者にもなるため，サービスの利用者が知らぬ間に違法性の高いサービス提供に荷担してしまう可能性もある．多対多通信のサービスをオーバーレイネットワークを用いて広く普及させるためには，こうしたサービスの不正利用者などを排除する仕組みを構築する必要がある．

第7章　結論

インターネットの普及によって，多対多通信のアプリケーションは実現可能となった．しかし，クライアント・サーバ型では単一障害点の回避のためにサーバを冗長化する必要があり，需要の低いサービス提供は困難であった．P2P 型によるサービス提供はこれらのサービス提供にかかるコストを下げ，公益性の高いサービス提供を可能にする．

P2P 型のサービスは，構造化オーバーレイネットワークの構築によって信頼性の高いサービスを提供することができる．しかし，構造化オーバーレイネットワークではノードの Churn によって検索の成功率が低下する．利用者の計算機が参加するようなオーバーレイネットワークではノードの参加と離脱は頻繁に起こることが想定できるため，激しい Churn による検索率の低下は防がなければならない．本論文ではオーバーレイネットワークの研究分野において以下の研究目標を定めた．

時刻属性による検索　オーバーレイネットワークの検索に用いる単一の Key に時刻属性を用いることで，様々な情報をオーバーレイネットワーク上で共有可能にする．既存のオーバーレイネットワークでは連続量に対して始点と終点を定める必要があるため，時間のような連続量を扱うことが困難である．

検索成功率の向上　利用者の参加するオーバーレイネットワークではノードの Churn によって検索成功率が低下する．特に範囲検索では対象となる情報が多くなるため，Churn の影響を受けやすく，検索成功率が低下しやすい．

研究目標に対して本論文では，新しいハッシュ関数の提案とノードの時刻属性による階層化によって時刻属性を用いた検索の成功率を大きく向上させることができた．提案したハッシュ関数は時刻属性が過去になるほど似た ID を生成するため，Key に対する Value が同じノードへと集約される．また，Churn による影響を階層によって分離し，検索成功率の向上と負荷の公平な分散を実現した．

提案手法を実装にあたって特に P2P 通信の確保とアプリケーション拡張容易性を設計方針とした．IPv4 と IPv6 アドレスが混在する環境下では P2P 通信の確保が困難になるため，コネクションを逆向きに利用する手法と，メッセージをリレーする手法を取り入れることによって，想定するどのような環境でも P2P 通信を確保した．また，簡潔な構造とイベントドリブン方式によるアプリケーションを実装したため，容易にアプリケーションを開発することが可能である．

実験の結果，理想的な対数を用いた分布では DHT と比べて検索成功率が約 41%の向上，現実的な Weibull 分布を用いた場合でも DHT と比べて約 35%の向上が見られた．また，階層化によるオーバーヘッドも約 100 メッセージであり，実ネットワークでの性能にはほとんど影響しないことも確認した．また，アプリケーションとして写真の共有を行う P2P ネットワークを開発した．P2PData クラスを拡張することで他の情報であっても共有することが可能である．

本論文では時刻による情報の検索を実現し，検索成功率も大きく向上した．検索成功率がサービスとして許容できる範囲内であるならば実際のサービス展開に利用できる．一方で限りなく 100%に近い成功率を求めるサービスのために，他の手法との併用を行う必要がある．提案手法は情報の配置のみによって検索成功率を向上させたため，他の手法を取り入れることで 100%の成功率を実現できると想定できる．一方で，多対多通信の情報基盤を構築するためには，汎用的なオーバーレイネットワーク基盤とそこで動作するキラーアプリケーションの開発が必要である．オーバーレイネットワークのアプリケーション開発を容易にすることでキラーアプリケーションの登場を期待する．

謝辞

五年間に渡る奈良先端科学技術大学院大学での研究活動は多くの人の様々な支援があったからこそ続けることが出来ました．様々な形で支えてくださった全ての方に心より感謝いたします.

奈良先端科学技術大学院大学情報科学研究科インターネット・アーキテクチャ講座の砂原秀樹教授には研究のみならず研究者としての心構えや，社会との接し方など様々な助言を頂きました．また，研究への助言やビジョンなど，様々な議論を重ねることで大きく成長したと考えています．大変感謝致します.

奈良先端科学技術大学院大学情報科学研究科インターネット工学講座の山口英教授には中間発表など研究の節目の際に有用な助言・指導を頂きました．深く感謝いたします.

奈良先端科学技術大学院大学情報科学研究科インターネット・アーキテクチャ講座の藤川和利准教授には，研究に対する姿勢から論文の詳細な部分まで大変お世話になりました．心より感謝致します.

大阪大学情報科学研究科の寺西裕一准教授には他大学でありながら研究会や勉強会などの際に研究や就職に関してなど様々な助言を頂きました．本当にありがとうございました.

奈良先端科学技術大学院大学情報科学研究科の松浦知史助教には研究に対する助言や指導だけでなく，日々の生活に至るまで大変お世話になりました．忙しい中論文の確認をして頂くなど，多大な苦労をかけた学生であったと自認しています．本当にありがとうございました.

猪俣敦夫准教授，和泉順子助教，垣内正年助教，寺田直美助教，秘書の呂さんを始めとした奈良先端科学技術大学院大学インターネット・アーキテクチャ講座の学生ならびに教員の皆様には五年間にわたって大変お世話になりました．また，

共同研究者の新井イスマイルさん，島田秀輝さんには博士前期課程の頃からお世話になり，様々な社会を生きる術を学びました．心より感謝致します．

また，遠く北海道の地から奈良まで来て頂いただけでなく，生活の面でも多大な苦労をかけ，博士課程として在学する私の我が儘に付き合わせてしまったにも関わらず，私の生活と心の支えとなってくれた妻の洋美に深く感謝します．本当にありがとう．

最後に，博士後期課程への進学に対して理解して頂き，研究活動並びに生活と，何より美味しいお米を遠方の地より提供してくれた家族に感謝致します．

参考文献

[1] BitTorrent. http://www.bittorrent.com.

[2] Folding@home. http://folding.stanford.edu/.

[3] Gnutella(プ ロ ト コ ル に つ い て). http://rfc-gnutella.sourceforge.net/index.html.

[4] SETI@home. http://www.seti.org.

[5] Skype. http://www.skype.com.

[6] UG Live / Ocean Gridについて. http://www.utagoe.com/jp/technology/live/index.html.

[7] UPnP Forum. http://www.upnp.org.

[8] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. Proc. Second IEEE Int ' l Conf. on Peer to Peer Computing, 2002.

[9] A. Asiki, K. Doka, D. Tsoumakos, and N. Koziris. Support for Concept Hierarchies in DHTs. In *Proceedings of the 2008 Eighth International Conference on Peer-to-Peer Computing*, pp. 121–124. Citeseer, 2008.

[10] J. Aspnes and G. Shah. Skip graphs. *ACM Transactions on Algorithms (TALG)*, Vol. 3, No. 4, p. 37, 2007.

[11] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 353–366, New York, NY, USA, 2004. ACM Press.

[12] R.H. Brown and A. Prabhakar. FIPS180-1: Secure Hash Standard (SHA). *Federal Information Processing Standards Publication (FIPS)*, 1993.

[13] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, pp. 46–66, 2001.

[14] NG De Bruijn. A combinatorial problem. *Kibern. Sb., Nov. Ser.*, Vol. 6, pp. 33–40, 1969.

[15] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, Vol. 41, No. 6, p. 220, 2007.

[16] S. Deering and R. Hinden. RFC2460: Internet Protocol, Version 6 (IPv6) Specification. *RFC Editor United States*, 1998.

[17] K. Egevang and P. Francis. RFC1631: The IP Network Address Translator (NAT). *RFC Editor United States*, 1994.

[18] J.L. Eppinger. TCP connections for P2P apps: A software approach to solving the NAT problem. Technical report, Citeseer, 2005.

[19] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, p. 455. VLDB Endowment, 2004.

[20] A. Gupta, D. Agrawal, and A. El Abbadi. Approximate range selection queries in peer-to-peer systems. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research*. Citeseer, 2003.

[21] Nicholas J.A. Harvey, Michael B.Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. In *4th USENIX Symposium on Internet Technologies and Systems*, pp. 113–126, 2003.

[22] R. Huebsch, J.M. Hellerstein, N. Lanham, B.T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, p. 332. VLDB Endowment, 2003.

[23] HV Jagadish, B.C. Ooi, and Q.H. Vu. BATON: A balanced tree structure for peer-to-peer networks. In *Proceedings of the 31st international conference on Very large data bases*, pp. 661–672. VLDB Endowment, 2005.

[24] M.F. Kaashoek and D.R. Karger. Koorde: A simple degree-optimal distributed hash table. *Lecture Notes in Computer Science*, pp. 98–107, 2003.

[25] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 654–663. ACM New York, NY, USA, 1997.

[26] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pp. 183–192. ACM New York, NY, USA, 2002.

[27] Satoshi Matsuura, Kazutoshi Fujikawa, and Hideki Sunahara. Mill: an information management and retrieval method considering geographical location on ubiquitous environment. In *Applications and the Internet Workshops, 2006. SAINT Workshops 2006 International Symposium on Applications and the Internet*, p. 4, 2006.

[28] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS*, pp. 53–65. Springer Berlin / Heidelberg, 2002.

[29] T.S.E. Ng, I. Stoica, and H. Zhang. A Waypoint Service Approach to Connect Heterogeneous Internet Address Spaces [*].

[30] T. Pitoura, N. Ntarmos, and P. Triantafillou. Replication, load balancing and efficient range query processing in DHTs. *Lecture Notes in Computer Science*, Vol. 3896, p. 131, 2006.

[31] J. Postel. RFC 791: Internet protocol, 1981.

[32] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. 1990.

[33] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 161–172, New York, NY, USA, 2001. ACM Press.

[34] R. Rivest. RFC1321: The MD5 message-digest algorithm. *RFC Editor United States*, 1992.

[35] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, p. 329. Springer Berlin / Heidelberg, 2001.

[36] S. Saroiu, P.K. Gummadi, S.D. Gribble, et al. A measurement study of peer-to-peer file sharing systems. In *proceedings of Multimedia Computing and Networking*, Vol. 2002, p. 152. Citeseer, 2002.

[37] C. Schmidt and M. Parashar. Enabling flexible queries with guarantees in P2P systems. In *IEEE Internet Computing*, Vol. 8, pp. 19–26, 2004.

[38] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, Vol. 11, No. 1, pp. 17–32, 2003.

[39] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pp. 189–202. ACM New York, NY, USA, 2006.

[40] E. Tanin, A. Harwood, and H. Samet. Indexing distributed complex data for complex queries. In *Proceedings of the 2004 annual national conference on Digital government research*, p. 10. Digital Government Society of North America, 2004.

[41] G. Tsirtsis and P. Srisuresh. Network address translation-protocol translation (NAT-PT), 2000.

[42] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, Berkeley, CA, USA, 2001.

[43] 金子雄, 春本要, 福村真哉, 下條真司, 西尾章治郎. ユビキタス環境における端末の位置情報に基づく P2P ネットワーク. 情報処理学会論文誌:データベース, Vol. Vol.46, No. No.SIG18(TOD28), pp. pp.1–15.

付録

A. PSP2P のユーザインタフェース

PSP2P ではユーザとのやり取りに HTTP を利用する. 各ノードが HTTP サーバを動作させ，利用者はブラウザを用いて写真の閲覧を行う.

PSP2P ではノードを起動することで HTTP サーバが動作する．図 1 にトップページを示す. トップページではまず検索クエリを作成するフォームがあり, その下に公開している写真のディレクトリが表示される. さらにその下には現在キャッシュされている写真の一覧がディレクトリ表示される. ディレクトリをクリックすることでディレクトリ内に存在する写真一覧を閲覧することができ, また階層化されたディレクトリ等も表示される.

図 2 に検索の結果が表示される HTML ページを示す. 撮影時間を指定したクエリをトップページから作成することで, その時間に撮影された写真を得ることができる．Web ページ上には, 検索の結果見つかった写真が含まれているディレクトリの名前と, 検索によって見つかった写真のサムネイルが一覧表示される．写真の表示枚数は制限することができ, その場合は「Next」と書かれたリンクが挿入される. また, サムネイル画像をクリックすることで高画素写真を閲覧するページへとリンクされる.

図 3 に高画素写真を閲覧するためのページを示す. このページでは中央に大きく高画素の写真が表示される. この写真は HTML の img タグで挿入されており, アドレスにはこの写真を公開している IP アドレスを始めとした写真への URL が記載されているため, ブラウザで閲覧しているユーザは公開しているノードから直接取得していることになる. また, ページの右側には 3 種類のリンクを用意している. 一番上のリンクは検索の結果の中で, 時間順に次の写真へのリンクであ

図 1 HTTP サーバのトップページ

る. 時間的に次の写真が表示されるため, 写真の撮影者等に無関係に表示される. 次のリンクは検索の結果の中で,URL 順にしたときの次の写真へのリンクである. 時間順に写真を閲覧するのではなく, 公開しているユーザ毎に写真を閲覧する場合もあるため, このリンクを用意している. 一番下のリンクはノードのキャッシュ内にある時間的に次の写真へのリンクである. 検索から時間が経っている場合など, 新しい写真が追加されている場合があるため, このリンクを用意している.

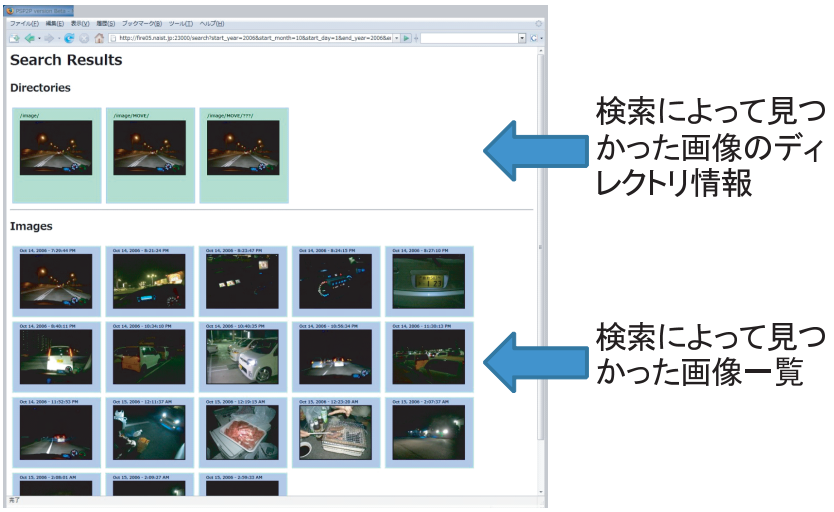

図 2　検索結果の表示画面

図 3　高画素写真の閲覧画面

著者研究業績

論文誌

1. 洞井晋一，松浦知史，藤川和利，砂原秀樹，“時間と位置を考慮したセンサオーバーレイネットワークの提案と評価”，情報処理学会論文誌，Vol.49，No.2，pp.590-602，2008 年 2 月

2. 洞井晋一，松浦知史，藤川和利，砂原秀樹，“時間に基づく階層化と Value の集約配置手法による耐 Churn オーバーレイネットワーク”，情報処理学会論文誌，Vol.51， No.4，2010 年 4 月掲載予定

国際会議

1. Shinichi Doi, Satoshi Matsuura, Kazutoshi Fujikawa, Hideki Sunahara, “Overlay Network Considering the Time and Location of Data Generation”, The 2007 International Symposium on Applications and the Internet SAINT2007 DAS-P2P 2007, Jan 2007

受賞

1. 洞井 晋一, 松浦 知史, 藤川 和利, 砂原 秀樹: “P2P ネットワークを利用したフォトペイントツール”, 野口賞 (デモンストレーション), マルチメディア，分散，協調とモバイル (DICOMO2009) シンポジウム

2. 洞井 晋一, 松浦 知史, 藤川 和利, 砂原 秀樹: “PSP2P:P2P ネットワークを用いた高画素写真の共有”, 優秀プレゼンテーション賞, マルチメディア，分散，協調とモバイル (DICOMO2008) シンポジウム

国内研究会

1. 洞井 晋一, 松浦 知史, 藤川 和利, 砂原 秀樹: “P2P ネットワークを利用したフォトペイントツール”, マルチメディア，分散，協調とモバイル (DICOMO2009) シンポジウム,pp.1855–1862,Jul,2009

2. 洞井 晋一, 松浦 知史, 藤川 和利, 砂原 秀樹: “PSP2P:P2P ネットワークによる写真共有”, 楽天研究開発シンポジウム 2008（ポスターセッション）, Nov, 2008

3. 洞井 晋一, 松浦 知史, 藤川 和利, 砂原 秀樹: “PSP2P:P2P ネットワークを用いた高画素写真の共有”, マルチメディア，分散，協調とモバイル (DICOMO2008) シンポジウム,pp.658–667,Jul,2008

4. 洞井晋一, 松浦知史, 藤川和利, 砂原秀樹: “既存の気象情報を利用した仮想センサの生成”, 情報処理学会 マルチメディア 分散 協調とモバイル（DICOMO2007）シンポジウム論文集,pp.1502–1509,Jul,2007

5. 洞井晋一, 松浦知史, 藤川和利, 砂原秀樹: ”センシングデータを考慮したP2P ネットワーク上での並列ダウンロード性能に関する一考察”, 並列/分散/協調処理に関するサマー・ワークショップ システム評価研究会,Aug,2007

6. 洞井 晋一, 松浦 知史, 砂原 秀樹: “移動センサを含めた環境情報を共有する基盤構築”, マルチメディア，分散，協調とモバイル (DICOMO2006) シンポジウム,pp.801–804,Jul,2006

7. 洞井晋一, 松浦知史, 藤川和利, 砂原秀樹: “時間と位置を考慮したオーバーレイネットワークの提案”, 第４８回プログラミングシンポジウム報告集,pp.65–76,Jan,2007

8. 川原 貴裕, 松浦 知史, 洞井 晋一, 藤川 和利, 砂原 秀樹: “組み込みセンサ用のプラグイン管理機構の提案”, 電子情報通信学会技術研究報告（IA2008） Vol. 108 No.74,pp.17–22,May,2008

9. 藤樫淳平, 池部実, 洞井晋一, 藤川和利, 砂原秀樹: “P2P システムにおける実ネットワークトポロジを考慮したオーバレイネットワークトポロジ構築に関する一考察”, マルチメディア，分散，協調とモバイル (DICOMO2008) シンポジウム,pp.925–931,Jul,2008

10. 水谷 后宏, 洞井 晋一, 松浦 知史, 藤川 和利, 砂原 秀樹: “プラグイン型ルーティングアルゴリズムを用いた複数オーバーレイネットワークの並列検証機構”, IPSJ SIG technical reports,pp.9–16,Jan,2009

11. 嶋津 裕己, 洞井 晋一, 和泉 順子, 砂原 秀樹: “ランドマーク情報を用いた歩行者ナビゲーション補助システムの提案”, 電子情報通信学会 総合大会 2009,Mar,2009

12. 山内 正人, 洞井 晋一, 松浦 知史, 砂原 秀樹: “センサに対する地理位置情報の自動設定手法”, 情報処理学会 マルチメディア 分散 協調とモバイル（DICOMO2007）シンポジウム論文集,pp.60–65,Jul,2007

13. 松浦 知史, 洞井 晋一, 落合 秀也, 石塚 宏紀, 江崎 浩, 砂原 秀樹: “Live E! : センサストリーム制御機構および地理位置に基づくオーバレイネットワークを利用したセンサ情報共有基盤の構築”, 情報処理学会 マルチメディア，分散，協調とモバイル (DICOMO2007) シンポジウム論文集,pp.1161–1167,Jul,2007

14. 川原 貴裕, 松浦 知史, 洞井 晋一, 藤川 和利, 砂原 秀樹: “多様なセンサを考慮した組込機器用ミドルウェアの開発”, 情報処理学会研究報告 2008-SLDM-134 / 2008-EMB-8,pp.209–214,Mar,2008

15. 山内 正人, 洞井 晋一, 松浦 知史, 藤川 和利, 砂原 秀樹: “情報爆発時代におけるセンシングデータ”, 情報処理学会 全国大会,Mar,2008