2019 年度　卒業論文

ライブストリーミングに対応した分散ハッシュテーブルの検討

学籍番号: T18I917F
沈 嘉秋

指導教員: 萩原 威志 助教

新潟大学工学部情報工学科

Year 2019 Graduation thesis

Title

Student number: T18I917F
Yoshiaki Shin

Advising professor: Assistant Professor Takeshi Hagiwara

Department of Information Engineering, Faculty of Engineering,
Niigata University

概要

分散ハッシュテーブルはファイル共有サービス等に応用され、すでに普及している一方で、近年はブロックチェーンの関連技術としても注目を集めている。分散ハッシュテーブルの中でもKademlia はその実装の容易さとノードの出入りに対する耐性の強さから実用的なサービスへの応用が可能であり、多くのサービスで採用されている。近年、ライブストリーミングサービスの需要が高まっている一方で配信プラットフォームの事業者への依存が強くサービス利用者の立場は弱いものとなっている。そこで本論文では分散的なライブストリーミングサービスの基盤となるシステムを Kademlia を元に実装し、その評価を行った。Kademlia 上で単純にライブストリーミングの実装を行う場合、非効率的な通信が発生するが、本論文の手法では、Kademlia ネットワークを更に構造化することで非効率的な通信を削減することが可能となった。結果、分散的なライブストリーミングサービスの実装への足がかりを作ることが出来た。

キーワード　P2P, 分散ハッシュテーブル

abstract

While distributed hash tables have been widely used in file-sharing services, they have recently attracted attention as a blockchain-related technology. Among the distributed hash tables, Kademlia can be applied to practical services because of its ease of implementation and resistance to incoming and outgoing nodes, and it is used in many services. In recent years, while the demand for live streaming services has been increasing, there has been a strong dependence on providers of distribution platforms, and the position of service users has become weak. In this paper, we evaluate a system based on Kademlia, which is the basis of distributed live streaming services. When live streaming is simply implemented on Kademlia, inefficient communication occurs, but this paper can reduce inefficient communication by further structuring the Kademlia network. As a result, we were able to create a foothold for implementing a distributed live streaming service.

keywords P2P, distributed hash table

目次

第 1 章

はじめに

1.1 背景

近年、P2P ネットワーク技術がブロックチェーンなどによって再び注目を集めている。最近ではブロックチェーン流行以前に流行していた P2P ネットワーク技術を利用した分散型ファイル共有サービスと組み合わせて開発されたサービスも見られる。[1] 分散型ファイル共有サービスは分散ハッシュテーブルという技術を用いて実装されることがあり、分散型ファイル共有サービスの中でも利用者の特に多い BitTorrent では Kademlia という分散ハッシュテーブルを用いて開発されている。分散型ファイル共有サービスはこれまで、膨大なサーバリソースを持つ大企業などでなければ実現できなかった、大規模かつ、高速なファイル共有を実現した。

近年 Youtube Live や Twitch といった動画ライブストリーミングサービスが家庭用ネットワークやモバイルネットワークの環境改善に伴い普及してきている。しかし、これらのサービスは膨大なサーバリソースを持つ大企業などでなければ実現できず、プラットフォーム事業者への依存が強く、サービス利用者の立場は弱いものとなっており不健全な状態にある。

1.2 目的

分散ハッシュテーブルの中でも Kademlia はその実装の容易さとノードの出入りに対する耐性の強さから実用的なサービスへの応用が可能であるが、その一方で Kademlia 上で単純にライブストリーミングの実装を行う場合、非効率的な通信が発生する。

そこで本論文では、分散型の動画ライブストリーミングサービスの基盤となるシステムを、Kademlia をベースに改良し実装する。成果物が Web ブラウザと Node.js で動作するライブラリとなるように開発を行う。

成果物ができるだけ多くのプラットフォームで動作するようにするために P2P 通信箇所に WebRTC [*1] を用いる。WebRTC は Web ブラウザなどといったフロントエンド環境とサーバーサイド環境の両方に対応した低遅延通信のための規格である。また WebRTC は近年では、スマー

[*1] https://webrtc.org/

トフォンやパーソナルコンピュータに搭載されている Web ブラウザで動作する唯一の P2P 通信規格でもある。

完成したライブラリを用いたベンチマークプログラムと一般的な Kademlia を用いたベンチマークプログラムを Node.js 上で実行しその性能や性質の比較を行い有用性などの検証を行う。完成したライブラリを用いて Web ブラウザ上で動作する分散ライブ動画配信アプリのサンプルを開発し動作確認する。

第 2 章

知識

2.1 分散ハッシュテーブル

分散ハッシュテーブルとは、あるデータとそのハッシュ値をペアとしたハッシュテーブルを P2P ネットワーク上で複数のノードによって分散的に実装する技術である。複数のノードにデータを分散配置を行うため適切な構造化を行う必要がある。構造化には様々な手法が存在し、Chord や Kademlia といったさまざまな実装が存在する。分散ハッシュテーブルの実装の優劣はデータの探索効率、Churn 耐性[*1]、実装の容易さなどによって付けられる。

2.2 Kademlia

Kademlia とは分散ハッシュテーブルの一種である。高い Churn 耐性を持つため、実用的な P2P アプリケーションにて多く利用されている。Kademlia はノード数 N のシステムにおいてデータを探索する際に $O(\log(n))$ 回ノードへの通信を行う。

[*1] ノードの出入りに対する耐性

2.2.1　採用例

サービス名	使用箇所
Torrent	magnetURL という機能を用いてファイルをダウンロードする際に 目的のファイルを持っているノードを探索するのに Kademlia を用いている。 Torrent はアクティブユーザと転送量という点で見ると 世界で最も成功した P2P のシステムであり、 そのシステムに Kademlia はおおいに貢献していると言える。
Ethereum	Node Discovery Protocol v4 というノードの探索プロトコルに用いられている.
IPFS	IPFS とは複数のノードが協調して一つの大きなストレージ または HTTP の置き換えとして機能することを目的としているシステムある。 IPFS は Kademlia をベースとして開発されている

2.2.2　アルゴリズム

ノード ID とキー

Kademlia における個々のノードには固有のノード ID が割り振られる。このノード ID は 160bit と定義されている。ノード ID の決定方法は、ランダムな値に sha1 というハッシュ関数を適用し、160bit の値を取り出すのが一般的である。また、ハッシュテーブルに保存するバリューと対になるキーも 160bit と定義されている。

経路表

分散ハッシュテーブルのアルゴリズムによってノードを管理する経路表の形は様々である。例えば、Chord という分散ハッシュテーブルの場合は環状の経路表を持っている。Kademlia の場合は k-buckets という 160 個の k-bucekt からなる経路表を持っている。一つの k-bucket には K 個 (たいていの実装例では 20 個) のノードが登録でき、自身のノードとの距離に応じた k-bucket にそれぞれのノードが登録されていく。ノード間の距離は 2 進数のノード ID 同士を XOR で掛け合わした結果を 10 進数に戻した値を用いる。

プロトコル

Kademlia には 4 種類の通信問い合わせがある。名称と内容についてまとめる。

名称	内容
PING	対象のノードがオンラインかどうかを問い合わせる。
STORE	対象ノードに key,value の組を保持させる。 保持させる際のルールは、 自身の k-buckets から最も key に xor の距離が近いノードを選択し、 そのノードに key,value を与える。 受け取ったノードは更に自身の k-buckets から受け取った key に 最も近いノードを選択し、 key,value を与える。この動作を何度も繰り返す。 最終的にはネットワーク上で最も key に距離が近いノードが 目的の key,value を持つ。
FIND_NODE	自身の k-buckets のうち最も key に xor 距離の近いノードに 自身のノード ID と距離が近い上位 K 個のノードの情報を送らせる。
FIND_VALUE	自身の k-buckets のうち最も key に xor 距離の近いノードに key と対応する value を持っているか問い合わせる。 持っている場合はその value を、 持っていない場合は問い合わせられたノード自身の k-buckets のうち、 key に最も近いノードの情報を返す。

ノードの管理

Kademlia の Churn 耐性の高さはこのノードの管理方法にある。Kademlia のノード管理は上記の 4 つのプロトコルの通信を行うついでに行われる。そのため、ノードの離脱の際の処理が必要なく、Churn を考慮することなくネットワークを維持することができる。

経路表の更新

ノードは 4 つのプロトコルのいずれかのメッセージを受け取った際に送信元が該当する k-bucket の中にあった場合そのノードを k-bucket の末尾に移す。送信元が該当する k-bucket の中に存在しないせず、k-bucket がすでに満杯な場合、その k-bucket 中の先頭のノードがオンラインかどうかを PING で確認する。オンラインなら先頭のノードを残し、そうでなければ送信元の新しいノードを k-bucket に追加する。こうすることで長時間オンラインになっているノードが優先的に k-bucket に残るため、ネットワークの安定性が増す。

ノードの新規参加

新規参加するノードは、まず接続先のノードに対して自身のノード ID を key として FIND_NODE を行う。問い合わせを受けたノードは送信元の key に近い最大 K 個のノードの情報を送信元に STORE する。そうすることで、新規参加するノードはまず最大 K 個のノードに接

続される。このあと、さらに自身の k-buckets のうち最も自身のノード ID に xor 距離が近いノードに対し自身のノード ID を key とした FIND_NODE を繰り返すことで接続先のノードを増やすことができる。

ノードの離脱

何もしない

2.3 WebRTC

WebRTC (Web Real-Time Communication)[2] とは World Wide Web Consortium (W3C) が提唱するリアルタイムコミュニケーション用の API の定義で，プラグイン無しでウェブブラウザ間のボイスチャット，ビデオチャット，ファイル共有ができる．ブラウザでリアルタイムなコミュニケーションを可能にする WebRTC は Google によってオープンソース化されており，現在は，W3C によってブラウザ対応 API の標準化が進められている WebRTC はブラウザ向けの API として誕生したが，現在では，Android や iOS といったネイティブ環境で実装するためのライブラリが公開されている．WebRTC には NAT 越えを実現するために Interactive Connectivity Establishment (ICE) という仕組みを採用している。

2.3.1 NAT 越え

NAT（Network Address Translation）とは，インターネットプロトコルによって構築されたコンピュータネットワークにおいて，パケットヘッダに含まれる IP アドレスを，別の IP アドレスに変換する技術である．プライベートネットワーク環境下でプライベート IP アドレスを持つホストから，グローバル IP アドレスを持つゲートウェイを通して，インターネットにアクセスする際に，プライベート IP アドレスをグローバル IP アドレスに変換するために利用されることが多い．モバイルネットワークにおいてはキャリアグレード NAT が用いられている．そのため，スマートフォン間で P2P 通信を行うためには，NAT 越えを行う必要がある．本研究では WebRTC を用いて NAT 越えを行う．WebRTC では ICE の情報をやり取りすることで，NAT 越えを行っている．ICE とは通信可能性のある通信経路に関する情報を示し，文字列で表現される．次のような複数の経路を候補とする．・P2P による直接通信・STUN による，NAT 通過のためのポートマッピング・TURN による，リレーサーバーを介した中継通信 STUN (Session Traversal Utilities for NATs) とは，音声，映像，文章などの双方向リアルタイム IP 通信を行うアプリケーションにおいて，NAT 越えの方法の 1 つとして使われる標準化されたインターネットプロトコルである．STUN プロトコルは，アプリケーションが NAT の存在と種類を発見し，リモートホストへの UDP (User Datagram Protocol) 接続に NAT が割り当てたグローバル IP アドレスとポート番号とを得ることを許す．STUN プロトコルが動作するには，インターネット上に STUN サーバが存在する必要がある．STUN プロトコルは，RFC(Request for Comments) 5389 に定めら

れる．TURN (Traversal Using Relay around NAT) とは，マルチメディアアプリケーションがNAT やファイアウォールを超えて通信することを補助するためのインターネットプロトコルである．TURN が一番役立つのは，TCP， UDP を使って対象型 NAT 装置により隠蔽（マスカレード）されたプライベートネットワークに接続されたクライアントを利用する場合である．TURN は RFC 5766 で標準化されており，IPv6 用のアップデートは RFC 6156 である．TURN が使う URI スキームは RFC 7065 に記述されている．本研究では Google が無料公開している STUN を用いている．TURN は自ら TURN 用のサーバを構築する必要があるうえ，全ての通信は TURN サーバを経由するため，TURN 上でのデータの転送量が非常に多くなるので本研究では TURN サーバを利用していない．

2.3.2 シグナリング

WebRTC では，SDP(Session Description Protocol) と ICE Candidate の二つの情報を端末間で交換することによって P2P 通信が開始される．この SDP 等を交換する作業をシグナリングと言う．シグナリングを行うためには SDP 等を交換する必要がある．シグナリングには Trickle Ice と Vanilla Ice の 2 つの方法がある。本研究では Vanilla Ice というシグナリング手法を用いる．Vanilla Ice は，実装が容易である，シグナリングサーバとの通信回数が少ないというメリットがある一方，P2P 接続の完了にかかる時間が Trickle Ice より長くなる傾向がある．Vanilla Ice の手順について説明する．Vanilla Ice の概要図を図 2.1 に示す．

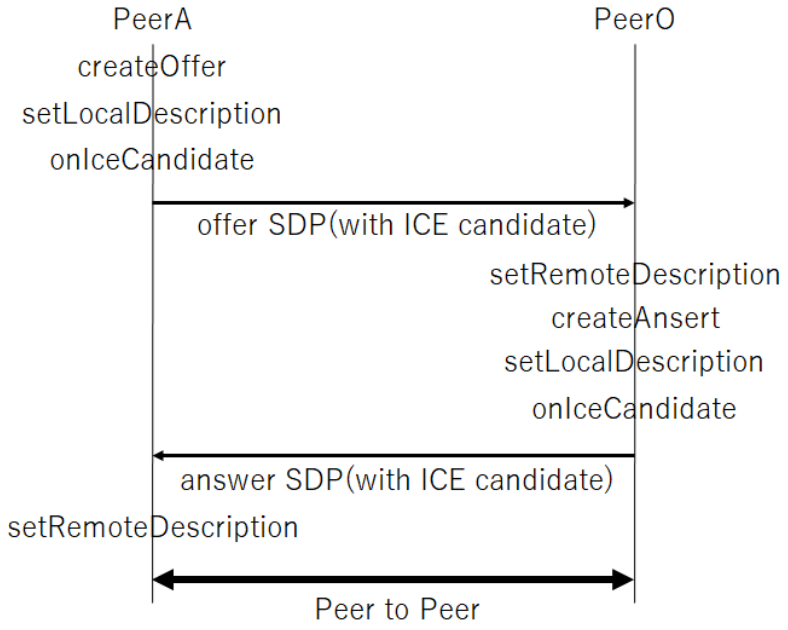

図 2.1　シグナリング

PeerA が createOffer を行い offer 側の SDP の作成準備を行う．次に setLocalDescription で

SDP を作成し，ICE candidate のリストアップを行う．ICE candidate のリストアップが完了すると，ICE candidate を offer 側の SDP の中に含ませて，PeerO へ送る．PeerO は PeerA の offer 側の SDP を setRemoteDescription で受け取り，createAnswer で answer 側の SDP の作成準備を行う．setLocalDescription で SDP を作成し，ICE candidate のリストアップを行う．ICE candidate のリストアップが完了すると ICE candidate を answer 側の SDP の中に含ませて，PeerA へ送る．PeerA は PeerO の answer 側の SDP を setRemoteDescription で受け取り P2P 接続が完了する．

第 3 章

本論

3.1　手法

本論文では、Kademlia をベースに効率的にストリーム形式のデータを共有できるシステムである LayeredKad を提案する。Kademlia はデータを共有する際に、そのデータをネットワーク全体に確率的に分散させる。そのため Kademlia で連続的なストリーム型のデータを共有する場合、データのチャンクをネットワーク全体から連続的に探索する必要がある。ライブ映像のようなチャンクの生成周期が非常に短く、大量のチャンクデータが存在するストリームデータを Kademlia で扱う場合、ネットワーク全体に対して連続的に負荷がかかることになり、非常に効率が悪いことが予測できる。そこで LayeredKad では共有するデータごとに別の Kademlia ネットワークを作成しネットワーク自体のスコープをデータごとに区切ることで、ストリームデータを効率よく共有できるようにする。本手法では、共有するデータの情報をメタファイルに保存し、Kademlia ネットワーク上で共有する。このメタファイルを共有するネットワークをメインネットワークとする。メインネットワーク上でそのメタファイルを所有するユーザ同士でさらに別の Kademlia ネットワークを構築し、メタファイルの情報に従いデータの共有を行う。この実体ファイルを共有するネットワークをサブネットワークとする。メインネットワークとサブネットワークの関係を図 3.1 に示す。

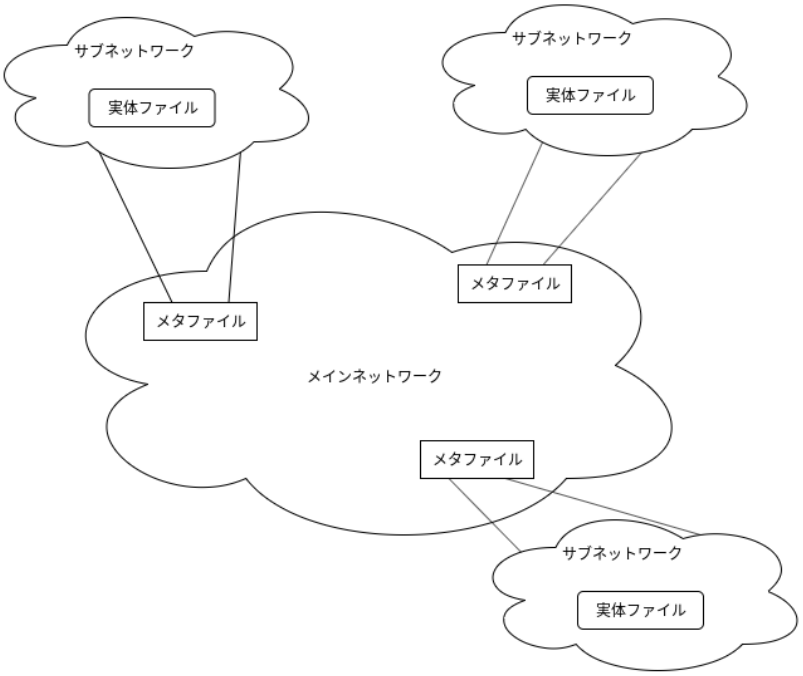

図 3.1　メインネットワークとサブネットワーク

3.1.1　Network

MainNet(メインネットワーク)

Kademlia をベースとしたネットワーク。メタデータのやり取りを行う。

SubNet(サブネットワーク)

Kademlia をベースとしたネットワーク。メタデータの指し示すデータのやり取りを行う。構造としては、一つの MainNet 上に複数の SubNet が存在することになる。

3.1.2　Actor

ノードは Seeder,Navigator,User の 3 つの Actor になりうる。一つのノードは複数の Actor に同時になりうる。

Seeder

任意のデータをもとにメタデータを作成し、メタデータを MainNet 上に Store する。MainNet 上でメタデータを Store されたノードは Navigator となる。また、Seeder も Navigator として振る舞う

User

メタデータの URL をもとに MainNet 上でメタデータを探索する。メタデータの所有者である Navigator を仲介として Seeder と接続する。Seeder と User のピアは SubNet という Kademlia ネットワーク上で管理される。SubNet と接続している User を Observer と呼ぶ

Navigator

User からメタデータの FindValue 命令を受け取ったら、Navigator と接続されたノード (Seeder もしくは、Observer) と FindValue を行った User との接続を仲介する。

データを共有する流れを示す。

3.1.3 メタデータ

LayeredKad では静的、動的の両方のデータ形式の共有に対応するためにメタデータとして共有するデータの情報の共有を行う。メタデータの基本構造は次のようになっている。

```
type Meta = {
  type: "static" | "stream";
  name: string;
  payload: { [key: string]: any };
};
```

StaticMeta

静的なデータを扱うメタデータ

```
type StaticMeta = Meta & {
  type: "static";
  payload: { keys: string[] };
};
```

StreamMeta

ストリーム (ライブ映像など) を扱うメタデータ

```
type StreamMetaPayload = {
  first: string;
  width?: number;
  height?: number;
  cycle: number;
};
type StreamMeta = Meta & {
```

```
  type: "stream";
  payload: StreamMetaPayload;
};
```

3.2 実装

TypeScript を用いて Node.js と Chrome 上で動作するように実装を行った。P2P 通信部部には WebRTC を用いている。

通常の Kademlia と本手法の LayeredKad の性能比較を行うために、Node.js 上で動作するベンチマーカーの実装を行った。なお、ベンチマーカーは実装の都合上 P2P 通信部分は WebRTC ではなく、UDP を使用している。

3.3 仮説

本論文では、データ通信量と、タスクの処理時間の 2 つの観点から、LayeredKad の評価を行う。現実的なユースケースを想定し、幾つかのファイルがネットワーク上に共有され、それぞれのファイルに関心があるユーザがそれぞれのグループを形成して共有しあっているケースを想定し仮説を立てる。

3.3.1 データ通信量

LayeredKad では共有するデータごとにネットワークを分割しており、データの保存や探索がネットワーク全体に影響しないので、Kademlia とノード数が同じであれば、ネットワーク全体における、データ通信量が少なくなることが予想される。ファイル共有サービスにおいてファイル共有速度の最も大きなボトルネックはネットワークの通信速度であるので、データ通信量が少ないことはメリットである。

3.3.2 タスクの処理時間

LayeredKad は Kademlia をさらに構造化したシステムであるため、単純に一つのファイルを共有するようなケースでは、当然、Kademlia の方が高速である。しかし、今回のケースでは Kademlia はすべてのファイルについての処理においてもネットワーク全体で処理を行う必要があるのに対して、LayeredKad ではユーザグループ（SubNet）内で完結するため、処理効率がよく、タスクの処理時間も短くなると考えられる。

3.4　結果

3.4.1　データ通信量

実験

データ通信量の計測を行うために、Kademlia と LayeredKad でデータ通信量の計測を行うベンチマークプログラムを作成した。ベンチマークプログラムは表 3.1 の環境で実行した。

表 3.1　スペック

CPU	Intel Core i7-7500U 2.70GHz 2Core 4Threads
RAM	8GB
OS	Ubuntu 19.10

本ベンチマークプログラムでは、ノードのペアを 1 グループとして、ファイル共有を行わせる。ベンチマークプログラムのシナリオのイメージを図 3.2 に示す

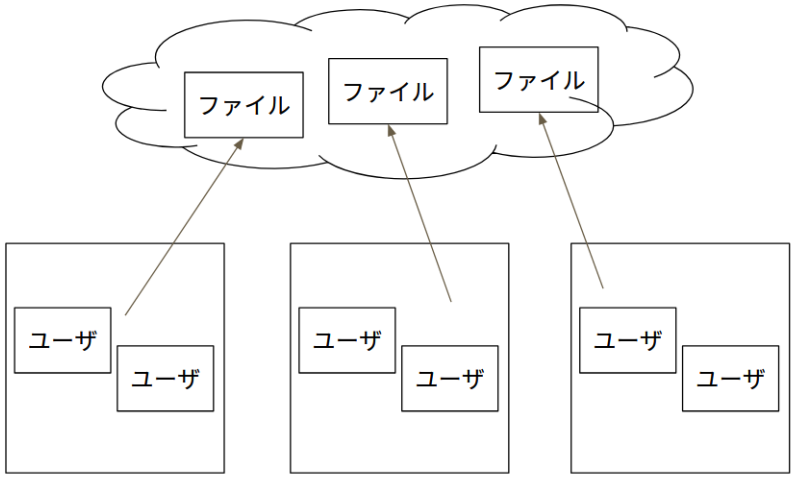

図 3.2　通信量ベンチマークのイメージ

ノード数を変数パラメータとして、パラメータを増加させて LayeredKad と Kademlia のデータ通信回数を記録し、比較を行う。

実験結果

表 3.2　実験結果

ノード数	Layered Kad	Kademlia
10	10	380
30	16	729
40	32	1026
50	33	1278

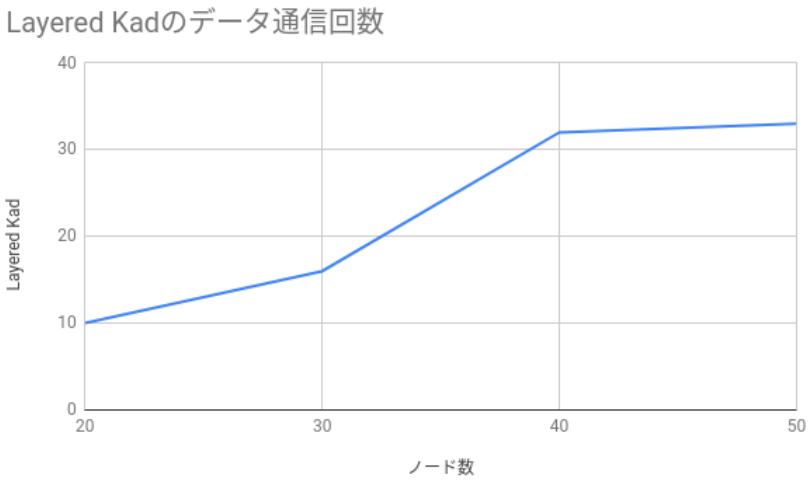

図 3.3　layeredKad の通信量

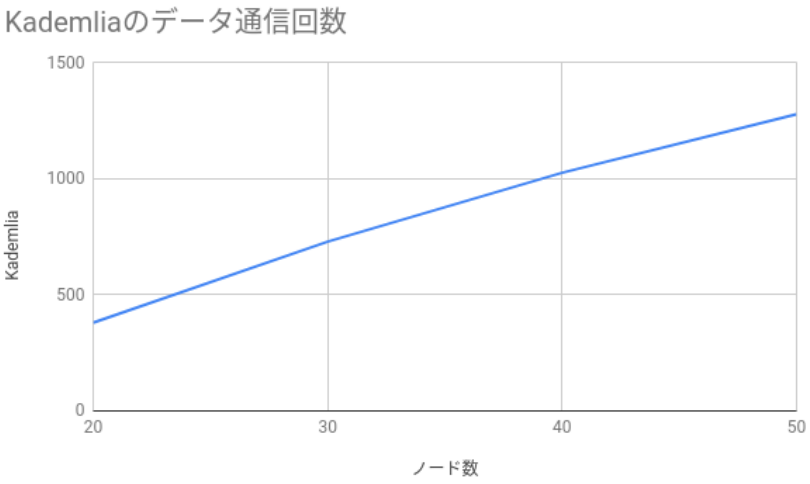

図 3.4　kademlia の通信量

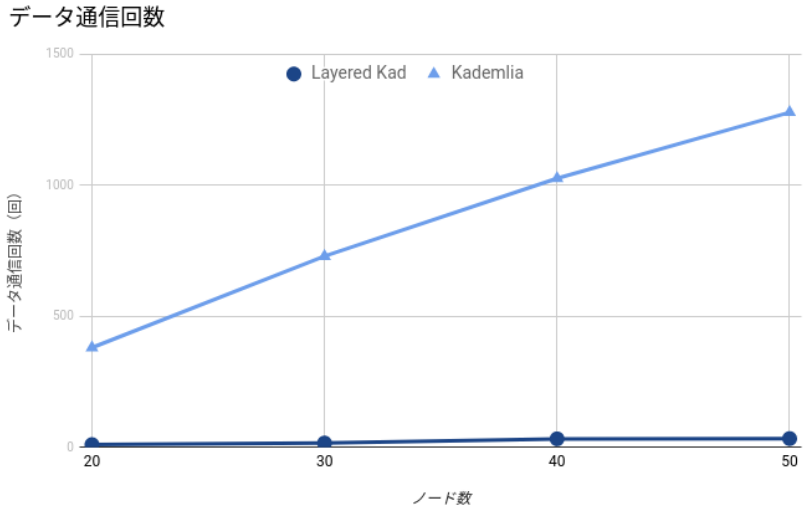

図 3.5　通信量の比較

3.4.2　タスク処理時間

実験

タスク処理時間の計測を行うために、Kademlia と LayeredKad でタスク処理時間の計測を行うベンチマークプログラムを作成した。ベンチマークプログラムは表 3.3 の環境で実行した。

表 3.3　スペック

CPU	Ryzen Threadripper 2950X 3.50Ghz 16Core 32Threads
RAM	?GB
OS	Ubuntu 18.04

ベンチマークプログラムのシナリオはデータ通信量のベンチマーカーと同じであるが、タスク処理時間をより正確に計測するために、1 ノードに付き、CPU のスレッド 1 つを割り当てるようにベンチマークプログラムのマルチスレッド化を行っている。本実験では、ノード数は 16 ノードで固定とし、共有するデータ数のチャンキング数を変数として実験を行う。

実験結果

表 3.4　実験結果

チャンク数	Layered Kad	Kademlia
1	2.098	0.636
4	2.728	2.032
5	2.469	2.435
6	2.581	2.679

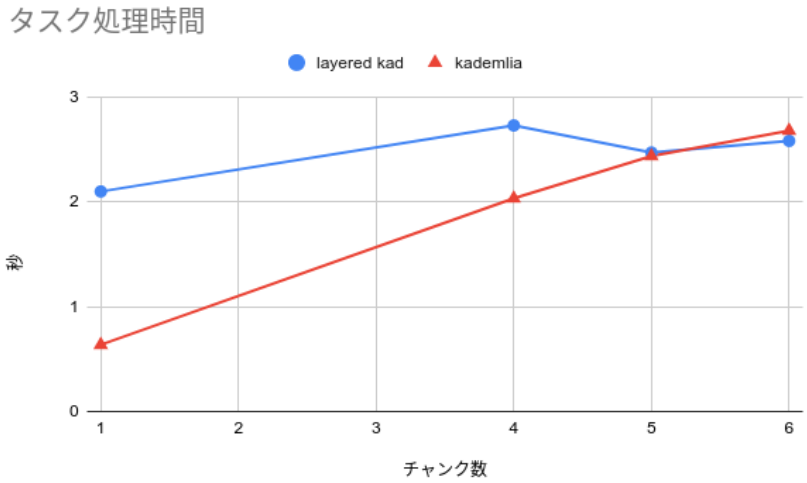

図 3.6　タスク処理時間の比較

3.5　考察

3.5.1　通信量

実験結果は、仮説の通りの結果となった。ただし、この結果はベンチマークのシナリオが LayeredKad が共有するデータ毎に別のネットワークを構築するという性質に有利な設定であるため、これほどの性能差が出たと考えられる。

3.5.2　タスク処理時間

LayeredKad ではデータのチャンキング数が増加してもタスク処理時間は概ね変化しないのに対して、Kademlia ではチャンキング数の増加に対してタスク処理時間が比例増大していることがわかる。LayeredKad では、いくらチャンク数が増えたとしてもサブネットワークのペアとなるノードとの一対一の通信回数がチャンク数分増えるだけなのに対して、Kademlia はチャンク数分、ネットワーク全体に対して問い合わせるため、その分処理時間が増大していると考えられる。

第 4 章

おわりに

4.1 結論

本論文では、Kademlia をベースに効率的に静的なデータやストリーム形式のデータを共有できるシステムの開発を行った。ネットワークを共有するデータ毎に分割し、非効率的なデータ通信を削減するための仕組みである LayeredKad を考案し、実装した。静的なデータや動的なデータなどを柔軟に扱えるようにするために、メタデータを共有する仕組みを考案した。メタデータを共有するネットワークをメインネットワークとし、メタデータの実体データを共有するネットワークをサブネットワークとし、ネットワークの分割を実現した。

LayeredKad の性能を評価するため、ユーザが実利用する際のシチュエーションに沿ったシナリオ上で、LayeredKad と Kademlia のベンチマークを行い性能の評価を行った。ベンチマークはネットワークのボトルネックとなるデータ通信と、タスク処理時間の二項目に着目して行った。データ通信のベンチマークでは、LayeredKad はそのネットワーク分割能力によって Kademlia よりデータ通信量が小さいごとを実証できた。タスク処理時間のベンチマークでは、Kademlia がチャンク数の増加に対して処理時間が比例増大したのに対して、LayeredKad ではチャンク数の増加が処理時間に影響しないことがわかった。

4.2 課題

Kademlia はネットワーク全体に一定のルールでデータを保存しているが、LayeredKad では実体のデータはデータを利用するユーザ間でのみデータが保存されている。そのためデータの冗長性という面では Kademlia に劣っている。また、Kademlia は様々なサービスに応用され長期間の安定した動作が確認されているが、本論文では LayeredKad の長期間の安定動作については検証しておらず、未知の問題が存在する可能性がある。

謝辞

本研究を進めるにあたり, ご指導を頂いた指導教員の萩原助教授に感謝致します. 日頃の議論において助言や知識を頂いた萩原研究室の皆様に感謝します.

参考文献

[1] bittorrent. btt. https://www.bittorrent.com/btt (2020 年 1 月 16 日 閲覧).