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

Here is abstract Here is abstract Here is abstract Here is abstract Here is abstract Here is abstract Here is abstract Here is abstract Here is abstract Here is abstract Here is abstract Here is abstract Here is abstract Here is abstract

keywords hoge, hoge

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 [*2] 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

[*1] https://www.bittorrent.com/btt/

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

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

第 3 章

本論

第 4 章

おわりに

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] Stephen M. Blackburn, Robin Garner, Chris Hoffmann, Asjad M. Khang, Kathryn S. McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. The dacapo benchmarks: Java benchmarking development and analysis. *SIGPLAN Not.*, 41(10):169–190, October 2006.

[2] Philipp Lengauer and Hanspeter Mössenböck. The taming of the shrew: Increasing performance by automatic parameter tuning for java garbage collectors. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, pages 111–122, New York, NY, USA, 2014. ACM.

[3] Oracle. Java platform, standard edition hotspot virtual machine ガベージ・コレクション・チューニング・ガイド. https://docs.oracle.com/javase/jp/8/docs/technotes/guides/vm/gctuning/index.html (2018 年 1 月 24 日閲覧).

[4] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[5] Jeremy Singer, Gavin Brown, Ian Watson, and John Cavazos. Intelligent selection of application-specific garbage collectors. In *Proceedings of the 6th International Symposium on Memory Management*, ISMM '07, pages 91–102, New York, NY, USA, 2007. ACM.

[6] Jeremy Singer, George Kovoor, Gavin Brown, and Mikel Luján. Garbage collection autotuning for java mapreduce on multi-cores. *SIGPLAN Not.*, 46(11):109–118, June 2011.

[7] Sunil Soman, Chandra Krintz, and David F. Bacon. Dynamic selection of application-specific garbage collectors. In *Proceedings of the 4th International Symposium on Memory Management*, ISMM '04, pages 49–60, New York, NY, USA, 2004. ACM.

[8] 相川光 中村成洋. **ガベージコレクションのアルゴリズムと実装**. 秀和システム, 2010.