P2Pのファイル共有を目的とした通信時に通信帯域が圧迫されるのを分散する方法

秋山 佑太[1,a)]　串田 高幸[1]

概要：P2P はノード同士が通信を行うアーキテクチャでノード同士が通信をするという特徴的な通信方式である. P2P でデータをやり取りするときに単体の peer が性能不足や回線速度の不安定によってデータの送受信ができなくなる可能性がある. 本研究では単体の peer の処理が行えなかったときに処理を分散し, 単体の peer の処理を早く処理させ, データを送る時間を少なくさせて高速化を図る. 分散の方法はピュア P2P を使い, 通信中に接続が悪く負荷がかかっているときに一時的に別の peer に接続させ, 通信している情報を別の peer に逃がし, 回線などが安定してきたときに再度接続させ通信を行うという方法を提案する. 本実験では 5MB から 40MB までで 5MB 刻みのデータ量の値で測定し既存のピュア P2P の通信時間をグラフ化, P2P の通信時にどのような特性があるのかを測定した. その結果, 5MB から 40MB までで 5MB 刻みのデータ量の値ではデータ量が増えると通信する時間が比例的に通信時間が増えていて, 5MB 増えるごとに平均約 0.026 秒増えていることが分かった.

1. はじめに

背景

P2P とは, peer to peer の略称でコンピュータ同士で対等に通信を行うアーキテクチャである.P2P の対置されるのもとしてクライアントサーバという方式がある. 一般的にはクライアントサーバは複数のクライアントに対し 1 つのサーバで構成されている [1]. 逆に P2P ではネットワークに接続されたコンピュータ同士がサーバを経由せずに直接通信を行う方式である. 2 つの大きな違いはクライアントサーバでは 1 つのサーバに対してクライアントの数が多くなるとサーバにアクセスが集中してしまい, 回線やサーバに負荷が集中してしまう. このため負荷が多くサーバが負荷に耐えられなくなってしまうとサーバがダウンしてしまい, クライアントがサーバに接続できず通信ができない状況が起こることがある. しかし, P2P はコンピュータ同士が個々で通信を行うため, 接続数が膨大になっても 1 つにアクセス集中することがなく, 負荷がかかりにくいという特徴がある [2]. 特徴としてブロックチェーン技術が使われていて, 分散型のデータ管理として使われているのが P2P ネットワークである. P2P では一般的なサーバー型とは異なり, 特定のサーバーやクライアントを持たずにノード同士が直接通信することができる. そのためユーザー同士での情報共有や決済が可能となっている. 一般的な決済システムではサーバー側で情報を集約して管理をするため, サーバー側に障害が起きた場合, サーバー側で保持している情報が使えなくなってしまう. しかし, P2P のブロックチェーンではユーザー側に情報が保存されているため複数のノードに障害が起きたとしてもシステムを維持して運用することが可能である [3]. そのため, ユーザー側の情報の保護も可能になる. 保護ができるため, さまざまな暗号通貨に使われており, 例としてビットコインに利用されている [4][5].

サーバーを返さない通信方式のため匿名のデータ送受信が可能である. そのため, ソフトウェアにセキュリティホール [6] が存在した場合, ウィルスが P2P のネットワークで感染がおこる脆弱性が起きることがある. 他にも, P2P の問題として音楽ファイルの違法アップロードがあげられる [7]. 誰でも簡単にファイルを共有でき, サーバーがないため特定する方法がない. よって違法アップロードが行われている. これが P2P の大きな問題の 1 つである.

また P2P には 3 種類存在し, 図 1 のようなピュア型, ハイブリット型, スーパーノード型が存在する [8]. ピュア型は P2P のシステムに参加しているノードの情報をすべて管理しているノードは存在せず, 必要としている情報を所有しているやりとりを行いたいノードを探すためにノードからノードというように隣接しているノードに対して必要な情報を所有しているか確認し, 目的の情報を持っている

[1] 東京工科大学コンピュータサイエンス学部
〒 192–0982 東京都八王子市片倉町 1404-1
[a)] C0118006

やりとりを行いたいノードを見つけるまで確認を行うという方式である. ハイブリット型はピュア型にサーバを追加した型でコンテンツデータのやりとりはノード同士で行うが, コンテンツデータの発見やノードの発見の役割をサーバに行わせるという方式である. スーパーノード型はノードの中でも処理能力が高いかつ通信が安定しているノードがネットワーク上から選出され, 接続されているノードの情報を管理する方式である. 個人間の情報交換の手段として P2P アプリケーションが普及しブログ, LINE, 情報交換アプリに使われている [9].

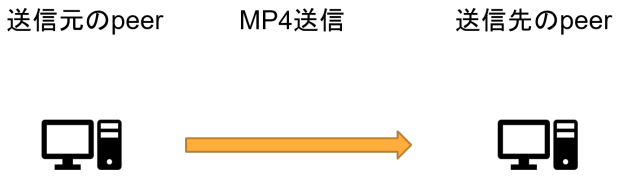

図 1　シンプルなピュア P2P

P2P の用途として主にデータ共有とストリーミングの 2 つが挙げられる.

P2P のデータ共有の特徴としてユーザー同士のデータ共有が簡単に行える.

また, サーバーにデータが保存されているわけではないので障害が起きたときに複数のノードに保存されている. そのため, 障害に強くファイル共有が簡単に行える [10].

P2P ストリーミングの特徴は, 一般的にストリーミング配信はサーバー側でファイルサイズが大きい動画データを配信し, データを複数のノードに向けて送信量を調節しながら送信できる. しかし, P2P ストリーミングではノード同士が直接通信を行う方式のため, サーバーを必要としない. よって従来のサーバーでのストリーミングではサーバーに集中していたものがノードに分散できるため, サーバーの負荷を軽減できるというものである [11].

データ共有において P2P は簡単で障害が少ないため必要不可欠なものである. これまでデータ共有を行う場合, 複数のノードが同じファイルを持つことによってダウンロード時に効率化を求めていた. 本研究では, ファイル共有の分散手法をふまえ, 新たな分散手法を提案する.

課題

現在 P2P はストリーミングやリアルタイムの放送型のコンテンツ配信などに使われていたり, ダウンロードの蓄積型コンテンツ配信, 情報共有（コミュニティ, ファイル共有）がある. そのファイル共有を目的とした際にデータ転送を断続的におこなうため, web ブラウジングに使用できる通信帯域が圧迫される状況が起きる. 仮に 200M ビット秒の回線を保有するサービスがあったとして, P2P ファイル共有ソフトを 24 時間使い続けたと仮定して, 1 日に使用するデータ量は 800G を超える. そのため通信帯域が圧迫されてしまうのを解決するために分散システムの仕組みを必要としている.

そこで送信先のノードが通信が出来ない状況時に送信元の待機時間と通信帯域が圧迫状態になってしまうため, 圧迫された状況を改善する必要がある.

例えば図 2 のように送信元の peer が通信できない状況になってしまった場合, 送信元の peer はデータを送信できなくなってしまう. よって送信を行う時間がかかってしまい, 通信時間と通信帯域が圧迫されてしまう状況になってしまう.

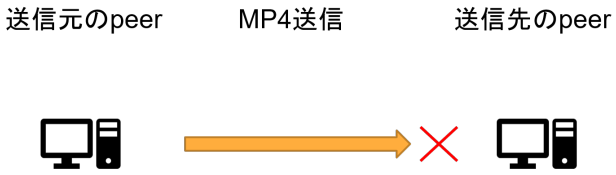

図 2　既存手法における問題点

各章の概要

第 2 章では関連研究について説明する. 第 3 章ではこのソフトの提案について説明する. 第 4 章では実装と実験環境について説明する. 第 5 章では評価と分析について説明する. 第 6 章では現状の課題とその解決方法を議論する. 最後に本研究の提案とそれに伴い得られた成果を示す.

2. 関連研究

P2P のファイル共有システムで分散型の効率的でファイルを複製する研究では, ファイルを複製することによりファイルを共有する際に効率を向上させることができるが複製することによりオーバーヘッドが高くなってしまう問題がある. そこでファイルの重要度が変動したりするのに対し動的に対応することでファイルの複製を過剰に行わず, 最適な量のレプリカを作ることができるものである [12]. この研究ではファイルの複製しているため, 単純に処理が複数になってしまう.

ワイヤレス P2P のファイル共有で通信障害が起きたノードを瞬時に置き換えて適格なノードを配置するという実装をした研究では, 普通の P2P と比べて障害回復時間とファイル転送時間が改善した [13]. だが, 転送を可能としていないので, 一部のノードが持っていないと時間がかかってしまう.

ハイブリット P2P のファイル共有で検索装置の強化を行い少ないレプリカを見つけ出す研究では, ハイブリット P2P を使い少ないレプリカを探し出すという提案をした [14]. この研究では少ないレプリカを検索するときの応答時間と低帯域幅のオーバーヘッドが課題である.

P2P で送信に失敗した情報を集めその情報を生かして,

失敗するノードを避ける研究では, 失敗した情報を集め失敗する確率を求めて効率化させるという提案をした [15]. この研究では分析する情報が少なくて対応できる時間が少なく効率化を求めることができないことが課題である.

3. 提案

本研究ではピュア P2P でファイル送信時の分散処理の手法を提案する. 提案では P2P でファイル通信を行う時, 送信先の peer が接続できなくなってしまった状況で送信元の peer が待たなくてはならない.

これに対して, ピュア P2P で peer の状況を監視させ接続ができないと判断した場合に転送の peer に送信元から送信先に送るファイルを送信元から転送の peer に切り替える. 通信ができていない状況を確認するために, 送信元の peer が送信先の peer にデータが送れない状況を確認させ, 10 秒以上返答がなかった場合に転送 peer に切り替える. peer 自体にハッシュ値を ID に持たせる. 探す場合は送信先のハッシュ値を持っている peer を見つけ送信先のハッシュ値を持っている peer を転送 peer にする. よって転送をさせる peer は送信先の peer のハッシュ値を持っているため, 瞬時に送信先の peer に転送させることが可能である.

これにより送信先の peer の接続ができない状況でも送信元の peer は転送の peer にファイルを送信できるため, 時間の短縮をすることができる. その構成を以下の図 3 に示す.

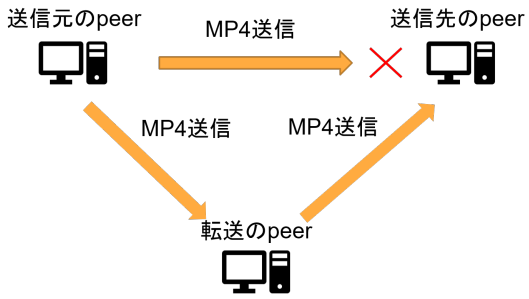

図 3　ファイル共有の方法

図 4 は通信の流れを表している. まず, 送信元の peer が送信先の peer に 1024 バイトの分割データを送信する. この時は送信先の peer の接続が継続しているとする. 次に送信先の peer からの反応がなかった場合, 送信先の peer に 1024 バイトの分割データを送信できないため転送の peer に送信先を切り替える. 最後に送信元の peer の通信が改善された場合, 送信元の peer が転送の peer に送った 1024 バイトの分割データを転送の peer から送信先の peer に送信する.

図 4 ように送信先の peer がダウンした状況である場合, 送信元 peer は送信先の peer の接続が戻るまで待たなくてはならない. その時間と通信帯域の圧迫を解消できるかが重要であり, 送信元 peer が送信先の peer を待たずともデータをアップロードできて待つ時間がかからないというのがこの提案の利点である. また, ファイルの複製をしないため, 処理が簡単である.

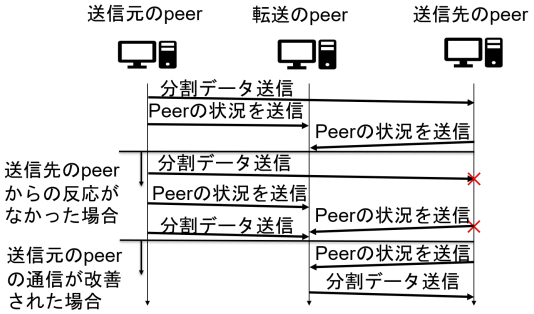

図 4　ファイル送信時のシーケンス図

図 5 は処理フローを示している. ここで送るデータとは 1024 バイトの分割データである. 送信元の peer の処理を示しており, 初めに送信先の peer にデータを送る処理を行う. 次に送信先の peer が通信できるか確認する. できた場合送信先にデータが送信されて処理は終了となる. 通信が出来なかった場合, 送信先が転送先の peer に切り替わり転送先の peer に送信される. 次に送信先の peer が通信できるようになった場合, 転送先 peer から送信先 peer にデータが送信され処理は終了となる. 通信が出来なかった場合送信先が通信されるまで待機し, 前の処理に戻る.

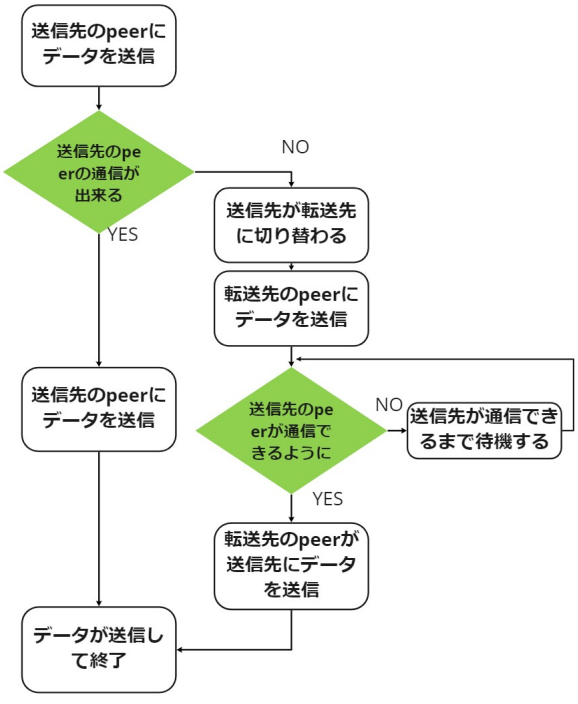

図 5　通信処理のフロー

4. 実装と実験環境

4.1 実装

ノードは送信先 peer と送信元 peer の 2 つに分けられる.peer では, 一般的なファイル共有を目的としたノードとして Ubuntu で仮想マシンを作成しノードとして使うため tesu を実装して配置した.peer の管理を行うため, tesu を実装し配置した. 図 6 に tesu のソフトウェアの構成について示す.

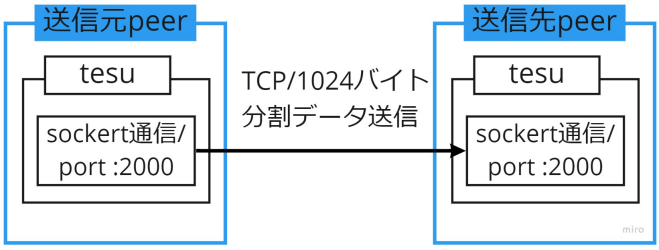

図 6 tesu の構成図

送信先の peer と送信元の peer のコードは Python のソケット通信を使用し, 作成した. 送信元の peer は送信先の peer が実行されている状態ではないと動かないため, 先に送信先の peer のコードを起動させる. なぜならば, 先に起動させておかないと通信ができたいためである. その後送信元の peer を起動し, 通信を行う.

送信するデータは TCP で 1024 バイトのデータを分割して送る. 使用するポートは 2000 番を使用した. ポート番号の重複を避けるため 2000 番で実験を行った. IP アドレスは送信先 peer の IP アドレスを入れて実装を行った.

送信元の peer のソースコードを, ソースコード 1 に示す.

ソースコード 1 送信元の peer

```
1  import time
2  import socket
3
4  def main():
5      soc = socket.socket(socket.AF_INET, socket.
           SOCK_STREAM)
6      soc.connect(("192.168.100.196", 2000))
7      data = b'a' * 1024 * 1024 * 5
8      t = time.time()
9      soc.send(data)
10     print(t)
11
12 if __name__ == '__main__':
13     main()
```

送信先の peer のソースコードを, ソースコード 2 に示す.

ソースコード 2 送信先の peer

```
1  import time
2  import socket
3
4  def main():
5      soc = socket.socket(socket.AF_INET, socket.
           SOCK_STREAM)
6      soc.bind(("0.0.0.0", 2000))
7      soc.listen(1)
8      soc, addr = soc.accept()
9      data_sum = ''
10     while True:
11         data = soc.recv(1024)
12         data_sum = data_sum + str(data)
13         if not data:
14             break
15     t = time.time()
16     print(t)
17
18 if __name__ == '__main__':
19     main()
```

5. 実験環境

検証環境は peer は 2 台で転送 peer は 1 台で行う.P2P では peer の数が多ければ多いほど良いが, 本研究では 1 台の peer で通信障害が起きたと想定するため, 3 台で十分であると考えた.peer は VM で Ubuntu18.04 をインストールしたもので, それを 3 台の仮想マシンで Python でソフトウェアを動かし使用した. 転送の peer のハードウェアは peer と同じため同じ条件で作成した. オープンソースでは実装の追加は困難だと考え, ソフトウェアを自作した. 実験を行うにあたって作成する必要があるプログラムは 3 つである.

- 送信プログラム
- 受信プログラム
- 転送プログラム

送信先 peer は ESXi 上に VM を作成し, Ubuntu18.04 で socket プログラムを動作させた. 送信元 peer も同様に ESXi 上にＶＭを作成し, Ubuntu18.04 で動作させた. 転送先 peer も同様に ESXi 上にＶＭを作成し, Ubuntu18.04 で動作させた. 図 7 は実験環境の構成図である.

5.0.1 比較の実験手順

この実験では一般的なピュア P2P での送受信にかかる時間を図る実験である. 5MB から 40MB までで 5MB 刻みのデータ量の値で, 試行回数は 1 つにつき 4 回行い, そのとった値の平均値を求めてグラフ化する.

(1) ＶＭ上に送信先 peer と送信元 peer を作成する.
(2) 送信先 peer のプログラムを起動し, 次に送信元 peer のプログラムを起動する.
(3) 通信を確認出来たらデータの送受信が終わるまで待つ.
(4) 送受信が終了後, 時間を記録し, グラフ化する.

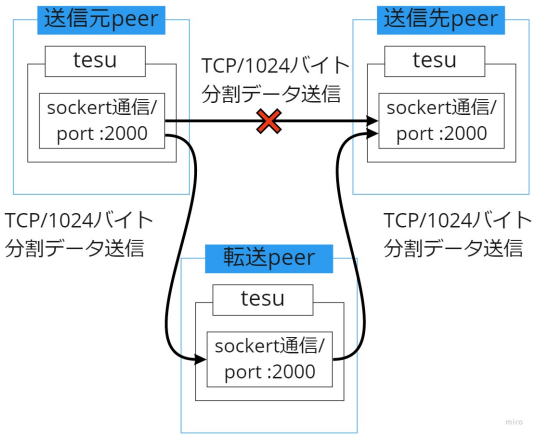

図 7　実験の構成図

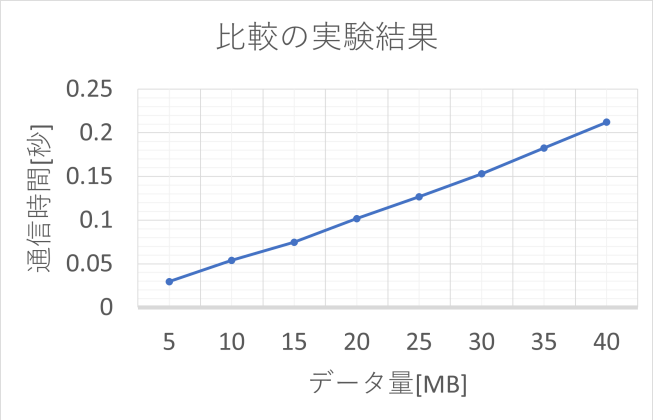

図 8　比較元の実験結果

5.0.2　提案の実験手順

この実験では本研究で提案した手法で送受信にかかる時間を図る実験である．5MB から 40MB までで 5MB 刻みのデータ量の値で, 試行回数は 1 つにつき 4 回行い, そのとった値の平均値を求めてグラフ化する．

(1) ＶＭ上に送信先 peer と送信元 peer と転送 peer を作成する．
(2) 送信先 peer のプログラムを起動し, 次に送信元 peer のプログラムと転送 peer のプログラムを起動する．
(3) 通信を確認出来たら送信先 peer の通信を落とす．
(4) 転送 peer にデータが転送される．
(5) 転送終了後, 送信先 peer の通信を復帰させる．
(6) 転送 peer から送信先 peer へデータを送信する．
(7) 転送 peer から送信先 peer へのデータの送信が終了後, 時間を記録し, グラフ化する．

6.　評価と分析

6.1　評価

6.1.1　比較の実験

実験の結果を図 8 に示す．横軸は peer 同士が通信するデータ量をあらわし, 縦軸は peer 同士が通信した時間を秒数であらわした．この実験ではデータ量が増えると通信する時間が比例的に通信時間が増えていて, 5MB 増えるごとに平均約 0.026 秒増えていることが分かる．

6.1.2　提案の実験

本稿ではこの実験に関しては転送 peer の実装をできなかったため実験を行うことができなかった．足りなかった状態として転送するコードが作成できていなかった．しかし, 比較の実験と提案の実験比較した場合, 提案の実験の数値が比較の実験の数値の 2 倍になることが見込まれる．これは通信状態が変動なく一定値で動いた場合である．

7.　議論

既存の P2P では問題なく送受信が行われている．しかし, 通信するデータ量が増えると送信先や送信元で通信問題や物理的問題が生じたときに既存の P2P では普及するまでの待機時間があるため, 復帰するまでに時間がかかってしまう．そのため大量のデータの場合, 待機時間がかからないように分散させる必要がある．

tesu の構築では, P2P でデータの分散を行い処理の分散を行うプログラムを作成した．大量のデータの場合, 別のノードに負荷を分散させることによって待機時間の短縮と通信帯域が圧迫されるのを防ぐことができる．しかし, 実際の P2P では別のノードを探すのに時間がかかってしまい, 現実的ではあるが検討が必要である．そのため, ハッシュ値を使い, ノードごとに情報を振り分けることによって探す時間を短縮できる．

実験の結果からデータ量に対して時間が比例していくことが分かったため, データ量が多くなると時間がかかることが分かる．よって, データ量が多くなると通信時間と通信帯域が圧迫される時間が長くなることが分かる．そのため, この提案では送信元 peer はダウンした送信先の peer から別の peer にデータを送ることによって, 待機時間と通信帯域の圧迫が軽減されることになる．しかし, 別の peer を見つける際に送信先の情報を持っている peer を探す必要があるため, 無限に探し続けてしまう危険性がある．そうなってしまった場合, peer を探すのに通信帯域を使ってしまい, 探している意味がなくなってしまう．そのため, 探し続ける peer には上限を付ける必要がある．また, 別の peer を探し出しても, その peer もダウンしてしまったらそれ以上に時間がかかってしまうため, 探す peer に通信状態がいいものを選ぶ必要がある．そのため, peer を探すときに ping を使うことによってより通信状況がいい peer を選ぶことが可能である．

8.　おわりに

本研究では P2P の通信時に送信先の peer が通信できなくなった場合, 送信するために待機時間と通信帯域が圧迫されるのを課題とした．P2P でデータをやり取りするとき

に単体の peer が性能不足や回線速度の不安定によってデータの送受信ができなくなる可能性がある. 単体の peer の処理が行えなかったときに処理を分散し, 単体の peer の処理を早く処理させ, データを送る時間を少なくさせて高速化を図る必要がある. 分散の方法はピュア P2P を使い, 通信中に接続が悪く負荷がかかっているときに一時的に別の peer に接続させ, 通信している情報を別の peer に逃がし, 回線などが安定してきたときに再度接続させ通信を行うという方法を提案した.

本実験では 5MB から 40MB までで 5MB 刻みのデータ量の値で測定し既存のピュア P2P の通信時間をグラフ化, P2P の通信時にどのような特性があるのかを測定した. その結果, 5MB から 40MB までで 5MB 刻みのデータ量の値ではデータ量が増えると通信する時間が比例的に通信時間が増えていて, 5MB 増えるごとに平均約 0.026 秒増えていることが分かった.

参考文献

[1] Risson, J. and Moors, T.: Survey of research towards robust peer-to-peer networks: Search methods, *Computer networks*, Vol. 50, No. 17, pp. 3485–3521 (2006).

[2] Valduriez, P. and Pacitti, E.: Data management in large-scale p2p systems, *International Conference on High Performance Computing for Computational Science*, Springer, pp. 104–118 (2004).

[3] He, Y., Li, H., Cheng, X., Liu, Y., Yang, C. and Sun, L.: A blockchain based truthful incentive mechanism for distributed P2P applications, *IEEE Access*, Vol. 6, pp. 27324–27335 (2018).

[4] Delgado-Segura, S., Pérez-Solà, C., Herrera-Joancomartí, J., Navarro-Arribas, G. and Borrell, J.: Cryptocurrency networks: A new P2P paradigm, *Mobile Information Systems*, Vol. 2018 (2018).

[5] Gribble, S. D., Halevy, A. Y., Ives, Z. G., Rodrig, M. and Suciu, D.: What can database do for peer-to-peer?, *WebDB*, Vol. 1, pp. 31–36 (2001).

[6] bt Abd Halim, N. S., Rahman, M. A., Azad, S. and Kabir, M. N.: Blockchain security hole: Issues and solutions, *International Conference of Reliable Information and Communication Technology*, Springer, pp. 739–746 (2017).

[7] McGill, D. A.: New Year, New Catch-22: Why the RIAA's Proposed Partnership with ISPS Will Not Significantly Decrease the Prevalence of P2P Music File Sharing, *Loy. LA Ent. L. Rev.*, Vol. 29, p. 353 (2008).

[8] Shi, L., Zhou, J. and Huang, Q.: A Chord-based supernode selection algorithm for load balancing in hybrid P2P networks, *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, IEEE, pp. 2090–2094 (2013).

[9] Xiaohe, L.: On P2P file-sharing: A major problem–a Chinese perspective, *Journal of Business Ethics*, Vol. 63, No. 1, pp. 63–73 (2006).

[10] Saroiu, S., Gummadi, P. K. and Gribble, S. D.: Measurement study of peer-to-peer file sharing systems, *Multimedia Computing and Networking 2002*, Vol. 4673, International Society for Optics and Photonics, pp. 156–170 (2001).

[11] Liu, Y., Guo, Y. and Liang, C.: A survey on peer-to-peer video streaming systems, *Peer-to-peer Networking and Applications*, Vol. 1, No. 1, pp. 18–28 (2008).

[12] Shen, H.: An efficient and adaptive decentralized file replication algorithm in P2P file sharing systems, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 21, No. 6, pp. 827–840 (2009).

[13] Li, X., Ji, H., Yu, F. R. and Zheng, R.: A FCM-Based peer grouping scheme for node failure recovery in wireless P2P file sharing, *2009 IEEE International Conference on Communications*, IEEE, pp. 1–5 (2009).

[14] Loo, B. T., Hellerstein, J. M., Huebsch, R., Shenker, S. and Stoica, I.: Enhancing p2p file-sharing with an internet-scale query processor, *Departmental Papers (CIS)*, p. 307 (2004).

[15] Marozzo, F., Talia, D. and Trunfio, P.: P2P-MapReduce: Parallel data processing in dynamic Cloud environments, *Journal of Computer and System Sciences*, Vol. 78, No. 5, pp. 1382–1402 (2012).