2ZC-03

Mercureプロトコルに基づくリアルタイム通信ライブラリの実装と検証

野口　尚裕†　　Martin J. Dürst†

† 青山学院大学 理工学部　〒252–5258 神奈川県 相模原市 淵野辺
E-mail: †noguchi@sw.it.aoyama.ac.jp, ††duerst@it.aoyama.ac.jp

1 はじめに

近年，Web上でリアルタイム通信の重要度は増している．実装はWebSocketが主流である．リアルタイム通信の中でも，双方向からデータ更新するチャットアプリのような場面は増加しているが，単にサーバ側からの更新を受信する株価や天気情報などWebSocket以外の技術が適した場面も想定される．

本研究では，Mercureプロトコルのライブラリをプログラミング言語Rubyで実装し，検証する．ライブラリは単独でも，Ruby on Railsの一部でも使用可能である．検証では，新実装とRailsのWebSocket実装であるAction Cableを比較する．CPU利用率や通信量，クライアント数に応じた処理速度をトピック数，送信者と受信者の比率など様々な状況下で計測し，それぞれの技術と実装の適切な利用場面について考察する．

2 基礎技術

2.1 Representational State Transfer

Representational State Transfer（REST）は， Webアーキテクチャスタイルであり，現代のWebで主流である．HTTP通信はリクエストレスポンスが基本である．そのため，HTTPの拡張で実現されたAjaxやLong Pollingなどのリアルタイム通信技術はクライアント主体の通信である．Appelqvistらの研究[1]にあるように，現在もWebにおけるリアルタイム通信技術の議論はなされている．

2.2 Publish/Subscribe メッセージングモデル

Publish/Subscribe（Pub/Sub）メッセージングモデルはリアルタイム通信を実現するためのモデルである．クライアントはデータ送信側をパブリッシャ，データ受信側をサブスクライバと呼ぶ．Pub/Subにおいて，サブスクライバは通信開始時にトピックと呼ばれるデータを受信する論理的な通信経路を指定し，サブスクライブする．パブリッシャがデータを送信したトピックのサブスクライバにのみデータが送られる．サブスクライバとパブリッシャは相互に通信する必要はなく疎結合であり，スケールしやすい．

2.3 WebSocket

WebSocketは通信開始時にクライアントからサーバに対し，HTTPリクエストを行い，確立した通信経路をWebSocketにアップグレードする．その後はサーバ，クライアント双方向からの通信が可能で，対称的なプロトコルである．

2.4 Server-Sent Events

Server-Sent Events（SSE）[1]はHTTPを利用したサーバプッシュ型のリアルタイム通信技術である．クライアントからサーバに対し，HTTPリクエストを行い，サーバ側は任意のタイミングでクライアントにイベント送信することでリアルタイム通信を実現する．SSEは接続確立後は，サーバからクライアントへの一方向の通信であり，サーバ主体の通信である．

2.5 Mercure

Mercure[2]はHTTPを使用し，Pub/Subを実現するリアルタイム通信プロトコルの1つである．サブスクライブにSSEを使用し，パブリッシュはHTTP POSTリクエストで実現する．

3 Mercure ライブラリの設計と実装

本研究では，Mercureプロトコルを使用したライブラリを実装した．ライブラリはプログラミング言語Rubyを使用し，Rack[3]アプリケーションとして実装した．全体の構成としてはPublisher，Server，Hubクラスがある．Publisherクラスはデータ更新をするクラスである．Serverクラスはクライアントからの HTTP リクエストを受信し，データを適切なハンドラに渡すクラスである．Hubクラスはサブスクライバの登録，管理をするクラスである．

4 実　　験

Action Cable[4]とMercureライブラリを比較・検証し，様々な状況下で適切なリアルタイム通信技術を考察する．まず，サーバ負荷を計測する．指標はサーバの最大CPU利用率と送受信通信量を使用する．次に，サブスクライバ数を増加させ，パブリッシャからデータ送信し，各サブスクライバが受信するまでの時間の計測し，平均値，中央値，最小値，最大値を算出することで，ライブラリの処理時間を検証する．加えて，トピックやパブリッシャ数を増やした状況下の実験をする．

1：https://html.spec.whatwg.org/multipage/server-sent-events.html
2：https://mercure.rocks/docs
3：https://github.com/rack/rack
4：https://guides.rubyonrails.org/action_cable_overview.html

Copyright ©2023 Information Processing Society of Japan.
All Rights Reserved.

表 1　100 クライアントがサブスクラブした際のサーバ負荷

クライアント数	最大 CPU 利用率 (%)	通信量 (KB)	
		送信	受信
Action Cable	0.173	40.351	58.989
Mercure	0.032	17.428	23.038

表 2　パブリッシュ時のサーバ負荷

クライアント数	最大 CPU 利用率 (%)	通信量 (KB)	
		送信	受信
Action Cable	0.037	11.434	5.569
Mercure	0.005	11.174	10.557

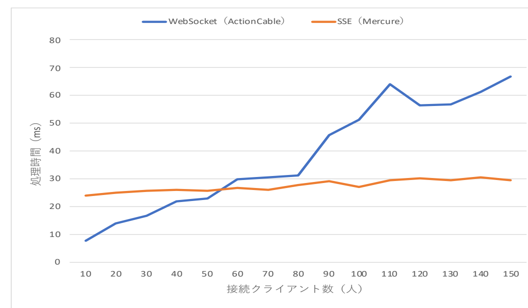

図 1　Topic 数 1，Publisher 数 1 の状況下での処理速度平均値

4.1　実験環境

本実験はクライアント，サーバ共に 1 つのマシン上で実施し，ネットワーク遅延はないものとなる．クライアント側は Node.js を使用し，Action Cable への接続には actioncable-nodejs，Mercure への接続には EventSource を使用している．サーバ側は Action Cable，Mercure を使用したアプリケーションを作成した．外部の影響を最小限にするため，Docker コンテナ内に作成した．また，サーバ負荷の実験では cAdvisor[5] を使用してメトリクス情報を取得した．本実験に使用したマシンは OS が Ubuntu 22.04.1 LTS，CPU が Intel Xeon Gold 5220 CPU @ 2.20GHz，メモリが 128MB である．

4.2　サーバ負荷

トピック数 1 の状況下で 100 クライアントがサブスクライブした時とそのトピックにパブリッシュした時のサーバ負荷を測定した．表 1 に見えるように，サブスクライブ時，Mercure は Action Cable と比較して CPU 利用率，通信量は小さくなった．

表 2 に見えるように，パブリッシュ時では，Mercure の CPU 利用率は Action Cable と比較して小さいが，受信の通信量は Mercure が ActionCable より大きくなった．

4.3　処理時間

クライアント数を増加させ，処理時間の平均値，中央値，最小値，最大値を計測し，それぞれ 3 回分の平均値を計算した．

はじめにトピック数 1，パブリッシャ数 1 の状況下で処理時間を計測した．図 1 に見えるように ActionCable の平均値はクライアント数の増加と共に，右肩上がりの傾向が見られた．Mercure はクライアント数に関わらず，横ばいの傾向が見られた．中央値，最大値も平均値と同様の傾向が見られ，最小値は Action Cable が Mercure よりも平均的に約 15ms 小さかった．以上のことから，Action Cable はクライアント数が増えるにつれ，処理時間の最大と最小の差が大きくなることがわかった．一方，Mercure は処理時間の最大と最小の差は小さく，クライアント数の影響が小さかった．

次に，トピック数 10 の状況下で処理時間を計測した．平均値や中央値，最大値は図 1 と同様の傾向になったが，最小値は Action Cable と Mercure で大きな差が見られなかった．Mercure はトピック数に関わらず一定の最小値であったが，Action Cable は約 10ms 大きくなったことで両者の差がなかった．

次に，トピック数 10，パブリッシャ数 10 の状況下で処理時間を計測した．この条件では，最小値は Mercure が Action Cable よりも約 15ms 小さかった．

5：`https://github.com/google/cadvisor`

5　考　　察

実験 4.2 の結果から，サブスクライブ時において，WebSocket は通信開始時にハンドシェイクが必要である．SSE は通常の HTTP リクエストと同様であることから，Action Cable は Mercure と比較して CPU 利用率や通信量が大きくなったと考える．また，パブリッシュ時において，WebSocket ではオーバーヘッドが小さいバイナリデータの送信が可能である．Mercure はテキストデータを扱うため，Action Cable の方が Mercure より受信の通信量が小さいと考えられる．実験 4.3 において，Action Cable はパブリッシャ数 1 の状況下では処理時間の最小値は Mercure より小さいが，クライアント数が増加すると，Mercure より平均値が大きくなる．このことから，Mercure での送信データを含んだイベント作成処理は，Action Cable の WebSocket 作成処理より高速であると考えられる．

このことから，パブリッシャとサブスクライバの関係で，Mercure は株価や災害情報の受信のような 1 対多，Action Cable はチャットアプリなどのような多対多の場面に適切であると考える．

6　まとめと今後の課題

本稿では，リアルタイム通信技術の適切な利用場面を考察した．Mercure のような SSE は厳密なリアルタイム性が必要な株価や災害情報の受信など 1 対多の場面，Action Cable で使用される WebSocket は頻繁にクライアントから更新され，厳密なリアルタイム性が必要ないチャットアプリなど多対多の場面に適していることがわかった．

今後の課題として Mercure ライブラリのフレームワーク化と本研究で使用した 2 つの技術に加えて，HTTP/2 や HTTP/3 との比較が考えられる．

文　　献

[1] Rasmus Appelqvist and Oliver Örnmyr. Performance comparison of xhr polling, long polling, server sent events and websockets, 2017.

Copyright ©2023 Information Processing Society of Japan.
All Rights Reserved.