WebAssemblyのメモリにおける増加量とリクエスト数の比による必要量の推定を用いたインスタンスの再起動

中川 翔太[1]　小山 智之[1]　串田 高幸[1]

概要：WebAssembly (Wasm) の利用はブラウザにとどまらず，既存のソフトウェアを拡張するプラグインとしても利用される．プロキシではユーザー定義のパケットフィルターの実装する際に Wasm が利用できる．Wasm には 1 度使用した最大量のメモリを確保し続ける特徴があるため，Wasm のインスタンスの実行時間に応じて，余分にメモリを確保し続ける期間が長くなる．Wasm インスタンスが余剰させたメモリを解放するためには，再起動を実施する必要があるが，同時に Wasm インスタンス内に保持するキャッシュも破棄されてしまう．そのため，Wasm インスタンスの保持するキャッシュが少ないときを見計らって再起動をすることにより，キャッシュヒット率の低下を抑えつつ，メモリの余剰を削減することで，Wasm インスタンスのメモリ使用量を縮小する．

1. はじめに

背景

WebAssembly (Wasm) は Web ブラウザ上で実行することを目的に設計された，高速で軽量な低レベルのバイトコードである [1]．Wasm の実行単位を Wasm インスタンスと呼び，それぞれの Wasm インスタンスは独立した関数・メモリ・グローバル変数のテーブルを持っている．

Wasm の特徴の 1 つにメモリモデルがある．本稿における用語の定義を，図 1 に示す．Wasm インスタンスが確保した全体のメモリ量を確保量と呼ぶ．確保量の内，実際に Wasm インスタンスが利用しているメモリ容量を必要量と呼ぶ．確保量の内，その時点で Wasm インスタンスが利用していないメモリ容量を余剰量と呼ぶ．図 1 のメモリモデルの説明の例では，初めのメモリの必要量が 3MB である．そして，メモリの必要量が 8MB に増加した後，必要量は 3MB に減少している．しかし，Wasm インスタンスのメモリモデルでは必要量に応じて Wasm インスタンスのメモリ量は増加するが減少しないため，余剰量が 5MB 発生する．

Wasm はサーバーサイドでも利用される．本稿では，プロキシ上での Wasm の利用に焦点を当てている．プロキシ上での Wasm を利用することによって，ユーザーによって定義された独自のフィルターが利用可能になる．具体例は，図 2 の認証情報のキャッシュ機構を備えたフィルターである．プロキシ内部で Wasm インスタンスが実行されており，インメモリで認証情報をキャッシュする．したがって，キャッシュの量に比例してメモリの必要量が増加する．その際に余剰量がない場合は確保量も増加する．Wasm インスタンスはクライアントからの HTTP リクエストヘッダに付加された認証トークンがキャッシュヒットした場合，認可サーバーによる認証・認可を省略する．キャッシュミスの場合は，認可サーバーへ HTTP リクエストを送り，その結果を新規にキャッシュする．これにより，認可サーバーへのリクエスト件数を削減する．本稿では，認証情報のキャッシュのデータサイズは 1 件あたり 1KB とした．このキャッシュ件数は，新たにログインしたユーザーがページへアクセスを試みた際に増加し，Time To Live (TTL) に伴って破棄される．

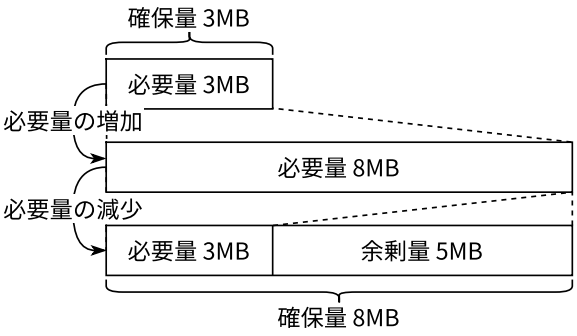

図 1　Wasm のメモリモデルの特徴

例えばアップストリームのサービスが EC サイトであった場合，セールの開始直後にリクエスト数が急増する．リ

[1] 東京工科大学大学院 バイオ・情報メディア研究科 コンピュータサイエンス専攻
〒 192–0982 東京都八王子市片倉町 1404–1

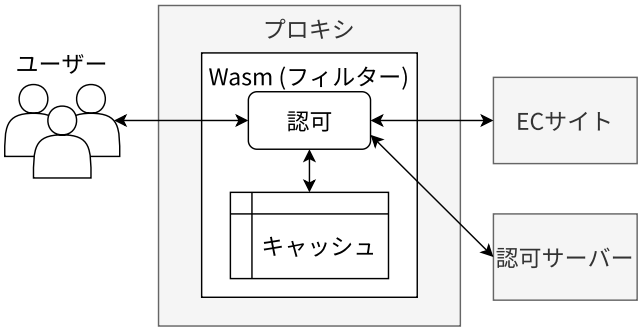

図 2　アーキテクチャ

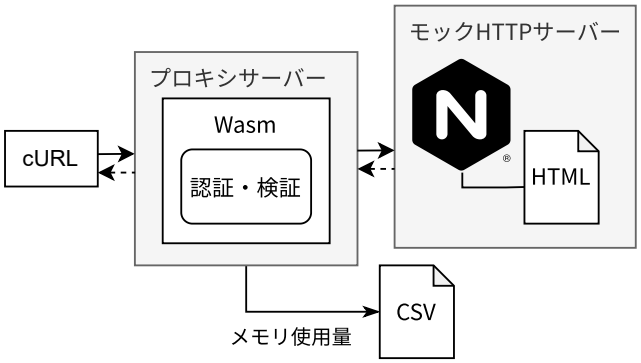

図 3　基礎実験の構成

クエスト数の増減に伴って，Wasm のメモリモデルのためプロキシ上の Wasm のメモリに余剰が生じる．

Web アプリケーションのためのプロキシを不特定多数のユーザーへ提供しており，Wasm による認証情報のキャッシュを行っている場合を考える．この場合，Wasm インスタンスのメモリに余剰が生じると，Wasm インスタンスが動作するサーバーへ同時に配置可能な Wasm インスタンスの数が減少する．

メモリの余剰量を解放する手段の 1 つに，Wasm インスタンス再起動がある．Wasm インスタンスを再起動した場合，Wasm インスタンスのメモリの状態も全て初期化される．そのため，余剰量を解放することができる一方で，インメモリでキャッシュしていた認証情報も消失する．認証情報のキャッシュが消失すると，これまでキャッシュの利用により省略可能であった，認可サーバーへリクエストを再度おこなう必要が生じる．例えば認証・認可に Software as a Service (SaaS) を利用している場合，リクエストに対してレートリミットが設定されている．また，認可サーバーをセルフホストしている場合でも，リクエスト数の増加に対応するためにスケールをさせる必要が生じ，メンテナンスコストが増大する．そのため認証・認可のリクエスト数を削減する必要がある．

基礎実験

リクエストの急激な増加に伴い Wasm のインメモリ上のキャッシュサイズが増加した後キャッシュサイズが減少した場合に，Wasm インスタンスの余剰量が増加することを確認する．基礎実験の構成を図 3 に示す．

ユーザーからの HTTP リクエストの送信を模する際には，cURL を利用する．この基礎実験で利用したプロキシ上の Wasm インスタンスは，`/login` のエントリーポイントへのアクセスがあると，1[KB] のトークンを発行し，自らのメモリ上にキャッシュする．`/logout` のエントリーポイントへアクセスがあると，そのリクエストヘッダに付与さているトークンを参照し，一致したものをキャッシュから破棄する．また，Wasm インスタンスのメモリの確保量についてタイムスタンプをつけ，CSV 形式で保存する．

図 4 に 2022 年 9 月 1 日に実施した基礎実験の結果を示す．cURL を利用し，`/login` エントリーポイントへ毎秒 1 回，合計 1024 回アクセスしたのち，毎秒 1 回 `/logout` エントリーポイントへ合計 1024 回アクセスした．したがって，キャッシュされるトークン数の推移は図 4 のトークン数のようになる．一方で青線を見ると，トークン数の増加に伴ってのメモリ使用量 [KB] 増加する一方で，トークン数の減少にのメモリ使用量 [KB] が追従しておらず，それまでにの確保量最大値を維持していることが確認できる．

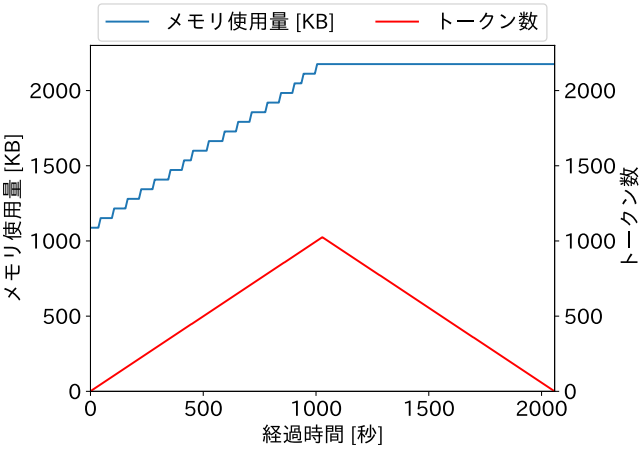

図 4　Wasm インスタンスのメモリ確保量が減少しないことの確認

課題

本稿で想定する Wasm インスタンスは実装の特徴として，確保するメモリの容量に上限が設けられておらず，アクティブユーザー数の増加量だけ認証トークンをキャッシュするため，際限なくメモリの確保量を増大させる．この場合の Wasm インスタンスの再起動条件の決め方には課題がある．再起動条件の決め方により，Wasm インスタンスのメモリの余剰量と，キャッシュヒット率が変化する．そのため，余剰量を削減しつつ，キャッシュヒット率が維持できる再起動条件が必要である．

例えば Wasm インスタンスが起動してから一定の経過時間で再起動を実施する場合，その経過時間の大小により，

2 つの観点がある．1 つ目はメモリの余剰量を保持している期間である．この期間が短いほど，ホストマシン上にデプロイされている複数の Wasm インスタンスの余剰量の総和が小さくなる．そのため，ホストマシンの利用可能なメモリの量が増加し，同時に起動をしておける Wasm インスタンスの数が増加する．2 つ目は有効なキャッシュの量である．有効なキャッシュの量が多いほど，キャッシュヒット率の低下が抑制される．結果，認可サーバーへのリクエストの件数が削減可能である．

Wasm インスタンスの再起動の決定に用いる経過時間が相対的に小さいとき，メモリの余剰量を保持している期間は短くなり，有効なキャッシュの量が減少する．Wasm インスタンスの再起動の決定に用いる経過時間が相対的に大きいとき，メモリの余剰量を保持している期間が長くなり，有効なキャッシュの量が増加する．

Wasm インスタンスを起動してからの経過時間をもとに再起動を実施する手法では，Wasm インスタンスのメモリの確保量の内訳を考慮できないため，2 つの観点の両立が出来ない．そのため，有効なキャッシュの量が多く，必要量の割合が大きい場合でも再起動が実施され，不必要な認可サーバーへのリクエストが生じる．メモリの余剰量の占める割合が大きい場合に直ちに再起動が実施されず，余剰量を保持している期間が長くなる．

各章の概要

2 章では，関連研究を紹介する．3 章では，提案方式とそのユースケース・シナリオについて説明する．4 章では，提案方式に基づいたソフトウェアの実装と，その実験方法について説明をする．5 章では，提案方式とそのソフトウェアの評価方法及び，得られるデータの分析手法について説明をする．6 章では，提案方式について議論をする．7 章では，まとめを述べる．

2. 関連研究

Halvorsen らはサービスメッシュにおけるデータ転送プロセス（データプレーン）へトークンベースの認証と認可を組み込むアプローチを提案している [2]．プロキシに Envoy を利用しており，フィルター機構に Wasm を利用して認証と認可を行っている．この研究では，評価としてレスポンスタイムを計測しているが，メモリ使用量の計測は行われていない．そのため加えて Wasm インスタンスのメモリ使用量について検討する必要がある．

Hockley らは，Wasm とネイティブでプロキシのキャッシュシミュレーターを作成している [3]．この研究では，ネイティブと Wasm それぞれの実行時間を比較を比較しているが，メモリ使用量については言及されていない．そのため，Wasm とネイティブのメモリ使用の傾向についても比較・分析をする必要がある．

3. 提案方式

提案方式

本提案の目的は，認証情報をインメモリでキャッシュする Wasm インスタンスについて，キャッシュヒット率を維持しつつ，メモリの余剰量を削減することである．

Wasm インスタンスの処理を呼び出しているプロキシ本体が観測可能かつ，再起動判定の参考にするメトリクスは，以下の 3 つに限られる．

- Wasm インスタンスのメモリ確保量
- Wasm インスタンスの処理リクエスト数
- Wasm インスタンスの送信 HTTP リクエスト数

また，オリジンサーバーへ HTTP リクエストを送信する役割はプロキシ本体の役割であるため，Wasm インスタンスの送信 HTTP リクエスト数にはオリジンサーバーへの HTTP リクエストは含まれない．

認可情報をキャッシュする Wasm インスタンスにおいて，メモリ余剰量が発生するのは，認可サーバーへ HTTP リクエストを送信し，そのレスポンスをキャッシュしたときである．

Wasm インスタンスのメモリ必要量の増加量を，メモリ余剰量の増加量が上回った時点で，その Wasm インスタンスを再起動する．

Wasm インスタンスの再起動を決定する際に，メモリ必要量及びメモリ余剰量を参照する．しかし，直接 Wasm インスタンスから取得可能なメモリに関するメトリクスはメモリ確保量であり，その内訳であるメモリ必要量・メモリ余剰量ではない．そのため，初めに Wasm インスタンスから取得可能な 3 つのメトリクスからメモリ必要量・メモリ余剰量を見積もる必要がある．本稿では，Wasm インスタンスのメモリの用途を，認可サーバーへ送信した HTTP リクエストに対するレスポンスのキャッシュとしている．また，キャッシュには TTL が存在し，一定時間で破棄される．したがって，Wasm インスタンスのメモリ必要量とメモリ余剰量は，Wasm インスタンスからの外向けの HTTP リクエスト数とキャッシュの TTL によって決定される．

本提案の全体像を図 5, 6 に示す．全体の流れは，Wasm インスタンスが起動してから，計測期間を設け，TTL の見積もり，基準値決定をした後に，再起動判定のループの手順になっている．

初めに Wasm インスタンスの起動から 15 分間の計測期間では，再起動の判定に用いるメトリクスの収集する．TTL の最大値を IBM の認証情報のキャッシュのデフォルトの値 [*1] を参考に 10 分とした．15 分は，10 分をウィンド

[*1] IBM, Authentication cache settings, https://www.ibm.com/docs/en/was/8.5.5?topic=domains-authentication-cache-settings

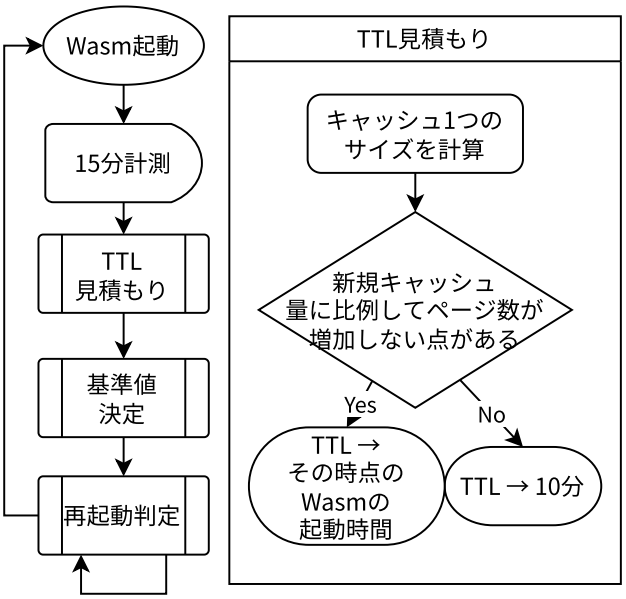

図 5 提案手法における全体の流れと TTL 見積もり

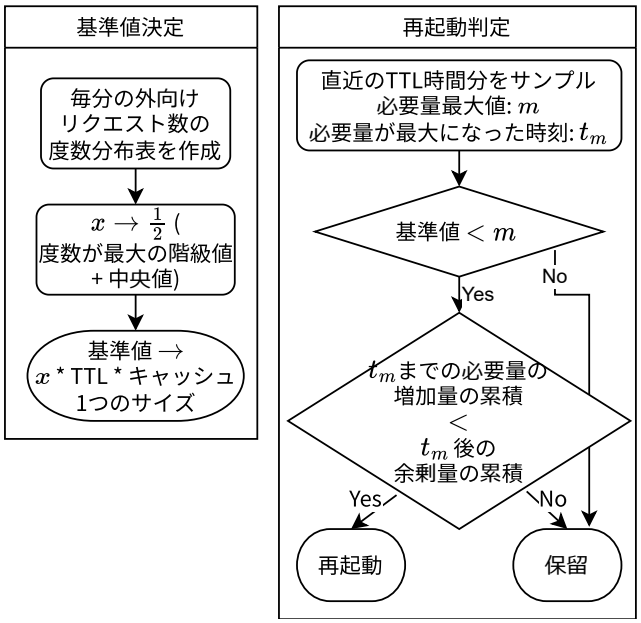

図 6 基準値決定と再起動判定

ウサイズとしたときに，半ウィンドウ分だけ拡大した期間である．TTL 見積もりでは，Wasm インスタンスの 16KB のページ単位のメモリを確保と認可サーバーへの外向けのリクエスト数の比を求める．そして，ページの確保が外向けリクエスト数よりも少なくなった場合に，古いキャッシュが破棄されたと見なし，その期間を TTL と判断する．10 分は上限値かつデフォルト値である．

次に求める基準値は，再起動を繰り返しても同程度のメモリ確保量になると見積ったときの値である．全体の中央値と，1 ページ分のリクエスト数毎の階級に分けたときの度数が最大の階級の階級値の平均を求める．実際に基準値として用いる際にはメモリ確保量と比較をする．そのため，基準値をリクエスト数からメモリ確保量ベースの値へ変換している．再起動は，基準値以上のメモリ確保量がある場合でかつ，メモリの必要量が減少に転じ，メモリの余剰量が増加したときに実施する．これにより，有効なキャッシュの損失を抑えつつ，メモリの余剰量の削減が実現できる．初回の再起動判定では，都合確実に再起動と判定される．2 回目以降の再起動判定では，既に求まった基準値を利用する．

ユースケース・シナリオ

Content Delivery Network (CDN) ベンダーが提供するプロキシサービスを図 7 に示す．プロキシサーバーの管理者は，それぞれプロキシサーバーへプロキシのプロセスをアップストリームの EC サイトの数だけデプロイする．各 EC サイトの開発者 ABC は，自らの EC サイトに対応するプロキシに対して Wasm による認証・認可のためのフィルターをそれぞれ WasmABC として作成し，プロキシ上に配置する．プロキシへ配置された Wasm はユーザーからの HTTP リクエストに対応して，認可サーバー X や認可サーバー Y への問い合わせを行い，そのレスポンスと自らのメモリ中へキャッシュする．ユーザーからの HTTP リクエストは，プロキシを経由して各 EC サイトへ送信される．この Wasm インスタンス実装の特徴として，確保するメモリの容量に上限が設けられておらず，アクティブユーザー数の増加量だけ認証トークンをキャッシュする．そのため際限なくメモリの確保量を増大させる．この際にプロキシへ HTTP リクエストが届くと，プロキシは Wasm インスタンスに対して HTTP リクエストを渡す．HTTP リクエストをプロキシによって渡された Wasm は，HTTP のヘッダ情報をもとにキャッシュによる認可，または認可サーバーへの問い合わせによる認可を行う．そして，プロキシは HTTP リクエストが認可されると，アップストリームの EC サイトへルーティングする．

4. 実装と実験方法

実装

ソフトウェアは，Rust 言語で実装する．実装するコンポーネントは，バックエンドの EC サイトへ接続するプロキシと，そのプロキシから利用する Wasm モジュールである．Wasm ランタイムには，Wasmtime を利用する．Wasmtime は，Wasm ランタイムの内リファレンス実装に位置し，BytecodeAliance によって開発が進められている．

図 8 にソフトウェアの実装の概要を示す．大きく分けて，プロキシとデーモンの 2 つのソフトウェアを作成する．

プロキシでは，アップストリームの EC サイトへのプロキシをする際，Wasm インスタンスの関数に HTTP リクエストのヘッダを引数として呼び出し，認証・認可のプロセスを実施する．また，キャッシュの TTL にしたがって

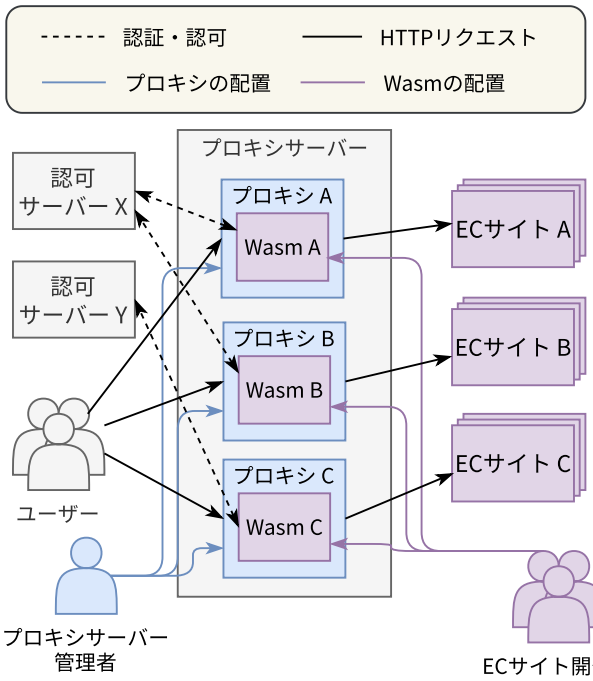

図 7　ユースケース・シナリオ

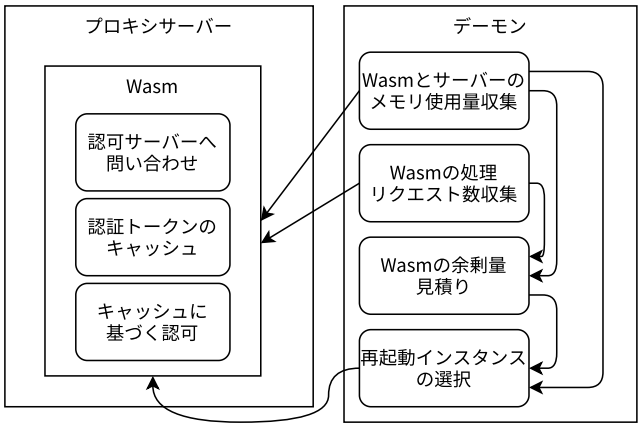

図 8　ソフトウェアの実装の概要

キャッシュのエントリーを削除する必要がある．そのため，呼び出しもとのプロキシは一定間隔で Wasm の関数を呼び出し TTL の切れたキャッシュを削除する．Wasm インスタンスでは，Authorization ヘッダ情報をもとにキャッシュを確認する．キャッシュヒットした場合は，そのキャッシュの情報で認可を行う．キャッシュミスの場合は，認可サーバーへ問い合わせをしその結果をキャッシュする．

デーモンでは，Wasm インスタンスの管理を行う．はじめに，Wasm インスタンスの再起動の判断のために必要なメトリクスの収集を行う．収集するメトリクスは，Wasm インスタンスのメモリ使用量，Wasm インスタンスの処理した HTTP リクエスト数，プロキシサーバー全体のメモリ使用量である．次に収集したメトリクスをもとにして提案方式によって，Wasm インスタンスの余剰量の見積もりと，再起動させる Wasm インスタンスの選択を行う．再起動させる Wasm インスタンスが決定すると，デーモンはプロキシプロセスへ Wasm インスタンスを再起動させる指示をする．そして，再起動の指示を受け取ったプロセスプロセスが Wasm インスタンスの再起動を行う．

実験環境

実験環境は，図 9 に示す 5 つのサーバーで構成する．Elasticsearch には，EC サイトのアクセスログが保存されている．アクセスサーバーは，Elasticsearch へ時刻を指定してアクセスログのクエリを投げ，その結果からリクエストシナリオを作成する．そして，作成したシナリオをもとに EC サイトのモックサーバー宛にプロキシサーバーを経由してリクエストを送信する．認可サーバーはプロキシサーバーからのリクエストに応じて MongoDB に予め保存されたユーザーとトークンのリストをもとに認可の結果を返す．プロキシサーバーでは，Rust 言語で記述されたプロキシのプロセスが動作しており，そのプロセス内で Wasm インスタンスが読み込まれている．プロキシプロセス内で収集した次のメトリクス及びログをメトリクスサーバー上の MongoDB へ保存する．

(1) 受信した HTTP リクエストのログ
(2) Wasm インスタンスのメモリ確保料
(3) Wasm インスタンスが認可サーバーへ送信した HTTP リクエストのログ

Wasm インスタンスマネージャーは，メトリクスサーバー上の MongoDB へログの件数とメモリ確保量の推移についてのクエリを送信し，その結果をもとに提案手法により Wasm インスタンスの再起動の是非を判定する．アップストリームを模する EC サイトのモックサーバーには NGINX を利用する．

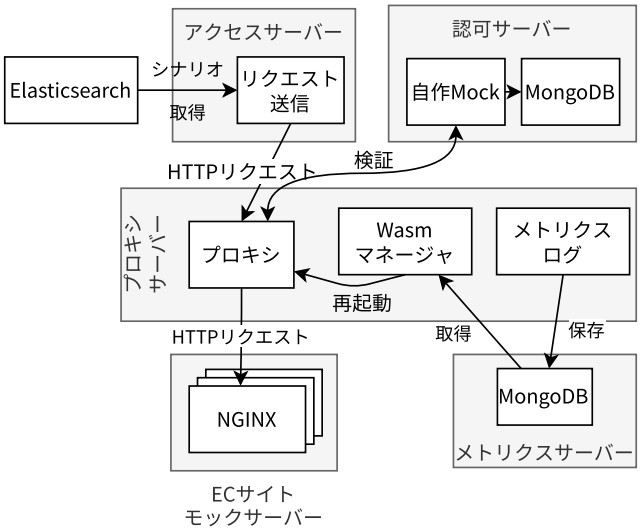

図 9　実験環境

5. 評価手法と分析手法

評価手法では，Wasm インスタンス再起動を実施しなかった場合と 提案手法である Wasm インスタンスの起動時間の上限値を定めそれに従って再起動させた場合を比較する．同様の HTTP リクエストのシナリオを適用し，Wasm インスタンスとキャッシュヒット率の推移で評価する．

評価で利用する HTTP リクエストのシナリオには，Chodak らによる EC サイトのログを参考に作成する [4,5].

6. 議論

本研究の提案手法では，初めて対象の Wasm インスタンスを起動してから最初の 15 分間を計測期間としている．そこで収集された HTTP リクエストの数とメモリ確保量から再起動判定のための閾値を求めている．Wasm インスタンスの実装で起動時に一括でメモリを確保する．Wasm インスタンスの内部でキャッシュの保存と破棄が行われた場合，見かけの必要量と確保量は常に一定になり，提案手法による再起動が一切起こらない．また，アリーナアロケーターが利用された場合，メモリの解放がアリーナと呼ばれる単位で一括で実施されるため，キャッシュの TTL とメモリの消費傾向が一致しなくなる [6].

Wasm によるプロキシを拡張する実装方針の 1 つに，Wasm インスタンスごとに固定長でメモリを割り当てる方針がある．この方針の場合，メモリ使用量に対してハードリミットの設定のみを行えばよく，Wasm インスタンスの再起動を検討する必要がない．したがって，1 インスタンスが利用可能なメモリは常に公平でかつ，運用も単純化されるメリットがある．その一方で，どの Wasm インスタンスからも利用されないメモリ量が生じ，1 つのサーバーへ収容可能なインスタンスの数を増加させられない．

7. おわりに

プロキシのフィルタを拡張する用途で Wasm を利用する際に，Wasm インスタンスの収容可能な数を増加させること研究の目的とした．Wasm インスタンス 1 つあたりのメモリ確保量の平均を縮小するためには，Wasm インスタンスの再起動をする必要がある．課題は，再起動のトレードオフとして，インメモリの認証情報のキャッシュが消失し，キャッシュヒット率が一時的に低下することである．

本稿では，キャッシュヒット率の低下を抑えつつ，Wasm インスタンスのメモリ確保量を縮小するための再起動のポリシーを提案した．この提案では，キャッシュの TTL と Wasm インスタンスがクライアントとなって外部のサーバーへ送信したリクエスト数に着目し， インメモリ上の有効なキャッシュの量を見積もった．そして，Wasm インスタンスのインメモリ上の有効なキャッシュの量が少なくなったタイミングで，対象の Wasm インスタンスを再起動をする．これにより，キャッシュの損失を抑えつつ，メモリ余剰量の削減を図る．

参考文献

[1] Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A. and Bastien, J.: Bringing the web up to speed with WebAssembly, *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 185–200 (2017).

[2] Halvorsen, S.: Authentication in the mesh with WebAssembly, Master's thesis (2021).

[3] Hockley, D. and Williamson, C.: Benchmarking Runtime Scripting Performance in WebAssembly (2022).

[4] Chodak, G., Suchacka, G. and Chawla, Y.: HTTP-level e-commerce data based on server access logs for an online store, *Computer Networks*, Vol. 183, p. 107589 (2020).

[5] Chodak, G., Suchacka, G. and Chawla, Y.: EClog: HTTP-level e-commerce data based on server access logs for an online store (2020).

[6] Bonwick, J. et al.: The slab allocator: An object-caching kernel memory allocator., *USENIX summer*, Vol. 16, Boston, MA, USA (1994).