サーバレス環境におけるエッジコンピューティングの高速化

kino-ma, 親 nyatsume

1 概要

従来のオンプレミスやクラウド型のコンピューティングに代わって，アプリケーションの実行にサーバレスアーキテクチャ (Function as a Service, FaaS) を採用できるようになった．サーバレスアーキテクチャでは，コンテナなどの隔離環境内でプログラムを実行し，クライアントに対してその結果を返却する．サーバレスアーキテクチャを採用することで，世界各地に分散したエッジコンピューティングも行いやすくなる．しかしサーバレスアーキテクチャの隔離環境としてコンテナを採用すると，コンテナの構築処理によるオーバヘッドが発生し，自動運転車など応答速度を要求するような用途に向かない．そこで本研究では，単純なコンテナ実行に代わる実行方式を考察し，応答速度・実行速度を両立しうる方式としてコンテナと WebAssembly を組み合わせるものを提案する．

2 背景

従来のクラウド式のサーバ構築では，開発者はプラットフォーム事業者が提供する仮想環境内を借り受けて，実行に必要な環境を 1 から構築することになる．反対に，AWS Lambda [1] や Google Cloud Functions [2] に代表される FaaS では，開発者はリクエストを処理するプログラムのみを記述し，その実行や結果の返却などはプラットフォームが担う．それにより，開発者から見ると環境構築の手間が省けること，必要に応じて自動でスケールされること，従来型のサーバと違ってアイドル時間へのコストがないことなど，アプリケーション開発におけるメリットが多い [3].

FaaS の活用先のひとつに，IoT アプリケーションなどでのエッジコンピューティングがある．クライアントから送られてくるデータを，自動でスケールするシステムで処理することができ，一つのサーバで処理するのに比べて応答速度やコストの面で優れていると考えられる．

3 問題

現在利用されているサーバレスアーキテクチャでは，システムが各アプリケーション用のコンテナ環境を作成し，実行が終了次第コンテナを停止する [4]．これによって，ファイルシステムやメモリなど資源を他のアプリケーションから隔離している．しかし，コンテナ環境を作成するのにはオーバーヘッドが存在し，全体の応答速度に影響する [5]．一度作成したコンテナ環境のキャッシュを取り，二度目以降の起動時間を短縮できる場合もあるが，一度目の起動については遅延が避けられない．また，そのキャッシュをしたがって，自動運転車など応答速度に厳しいアプリケーションで活用するためには課題が残る．

4 目的

本研究では，応答速度が求められるアプリケーションにおいてサーバレスアーキテクチャを利用するために，以下の要求を満たすことを目的とする．

4.1 要求

応答速度が求められるアプリケーションをサーバレスアーキテクチャで実現するために要求される事項は，以下の二つがある．

- 応答時間がより短いこと
- 他のアプリケーションの動作に影響を及ぼさないこと
- 既存のアプリケーションからの移行ハードルが低いこと

二点目は，FaaS プラットフォームにおいては物理的に一つのマシンで複数の FaaS アプリケーションを動作させることが考えられることから導かれる．

三点目は，すでに従来のクラウド型のプラットフォームでエッジコンピューティングシステムを構築している場合，FaaS に移行することの利点が薄まってしまい，事業者側が FaaS システムの恩恵を受けられることが減りうることから，求められる．

4.2 要件

また，上記の要求を満たすための要件を，以下の通り定義した．

応答速度: アプリケーション実行に関する各処理ステップの実行速度が短いこと．想定されるステップとして，具体的には以下がある．

- リクエスト送信〜アプリケーション起動
- アプリケーション起動〜処理終了
- 処理終了〜レスポンス到着

他アプリケーションに影響しない: 実行環境が隔離されていること．コンテナ環境のように，各アプリケーションが利用できるコンピュータ資源が制限または隔離されており，他のアプリケーションの資源と互いに影響を及ぼさないこと．本研究では，以下の資源を隔離対象とする．

- ファイルシステム
- メモリ
- CPU

メモリおよび CPU が隔離されていることにより，一つのアプリケーションが計算リソースを独占することがなくなり，アプリケーション間で不平等な実行速度になることを防げる．

移行ハードル: 既存のアプリケーションロジックを書き換えることなく，プログラムを使い回すことができること

5 関連研究

コンテナの代わりに WebAssembly でサーバレスアーキテクチャを実現する研究 [6]．プロトタイプ実装では初回の起動時間でコンテナより優れており，二度目以降の実行では大きく遅れをとる形になった．

6 提案手法

WebAssemby をインタープリタで実行し，一定時間が経過したらより高速な実行方式に切り替える方式を提案する．WebAssembly はデフォルトでサンドボックス環境であるため，要件の**隔離されていること** を満たせる．また，WebAssembly は主要なプログラミング言語が多くサポートしており，移行ハードルに関する要件も満たしていると考えられる．

WebAssembly から切り替えるより高速な実行形式としては，以下が考えられる．この二つのうちから，開発者の任意で形式を選択し，提案システムがそれを実行する．

- wasmer llvm
- 事前に用意したコンテナ

wasmer llvm では，LLVM を使用して WebAssembly を効率的なネイティブバイナリに変換する．独自のネイティブコンテナを持つ開発者は，オプションとしてそれを使用することができる．

7 実装

サーバレスアーキテクチャを実現するオープンソースソフトウェアである Apache OpenWhisk のアーキテクチャを参考にし，Controller, Invoker の二つを実装する．Controller はクライアントから HTTP リクエストを受信し，それを適切な形に加工して Invoker に渡す．Invoker は，実行対象となるアプリケーションの WebAssembly 関数を与えられた引数とともに呼び出し，その出力を Controller に返却する．

8 評価

提案手法のセクションで述べたように，要件のうち後者二つについては WebAssembly によって満たされる．したがって，のこりの **処理時間**の要件を満たしていることを確認するため，以下の項目を計測し，その結果の比較を行う．

各処理ステップの実行時間の計測. アプリケーション実行に係る以下の各処理ステップについて，開始から終了までの時間を計測する．これにより，要求のうち「**応答速度がより短いこと**」の項目を確かめる．

- リクエスト送信〜アプリケーション起動
- アプリケーション起動〜処理終了
- 処理終了〜レスポンス到着

また，それぞれについて以下のような種類のタスクを割り振り，それらがどのような分野に適しているかを考察する．

- 一回きりと，２回以上の比較
- 軽いタスク〜重いタスク

9 結論

コンテナを使用したサーバレスアーキテクチャの応答速度の課題を解決するため，WebAssembly とコンテナを組み合わせたシステムの提案をおこなった．定義した三つの要件のうち二つを満たすと考えられ，残りの一つは実装後に確認を行う．

10 今後の予定

- 11 月末: 先行研究のより詳しい調査
- 12 月上旬: Controller, Invoker の概形の実装 (設計) を行う
- 12 月中旬: Controller の HTTP サーバ部分，Invoker 接続部分を実装する
- 並行: Invoker の呼び出し部分の実装を行う
- 12 月下旬: 実装物の動作を検証し，期待通りの動作を確認する
- 1 月上旬: 評価内容をまとめ，考察を行う
- 1 月中旬: 考察結果をまとめ，結論を導く

11 参考文献

[1] 2021, https://aws.amazon.com/jp/lambda/

[2] 2021, https://cloud.google.com/functions

[3] 2021, https://www.hpe.com/us/en/insights/articles/serverless-computing-explained-1807.html

[4] 2021, https://www.ibm.com/cloud/architecture/architectures/open-cloud-platform/

[5] Pelle, István, et al. “Towards latency sensitive cloud native applications: A performance study on aws.” 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). IEEE, 2019.

[6] Hall, Adam, and Umakishore Ramachandran. “An execution model for serverless functions at the edge.” Proceedings of the International Conference on Internet of Things Design and Implementation. 2019.

[7] Du, Dong, et al. “Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting.” /Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems/. 2020.