

エッジコンピューティングのための 軽量なワークフローエンジンの設計と実装

佐藤 琢斗[†] 廣津 登志夫[‡]

法政大学情報科学部

1. はじめに

エッジコンピューティングは、従来のクラウドコンピューティングによる負荷の集中やレイテンシの問題を解決することを目的として、エッジと呼ばれるセンサ端末および端末の近傍にある計算ノードでデータ収集・加工の処理を行う。そこでのサービスは多数のジョブからなるワークフローがクラウド・エッジ・端末など多様な環境をまたがって稼働するため、ワークフローエンジンと呼ばれる複数ジョブの実行管理システムの利用が欠かせない。一方、エッジコンピューティングの処理は情報の秘匿、処理性能、オフローディングなどの実行に関する多様な制約や、ノードが持つソフトウェアやプロセッサアーキテクチャなどの実行環境によってノードに対する実行配置の要求が複雑になる。既存のワークフローエンジン[1][2]ではこれらの課題を解決するようなワークフローを構築することは難しい。そこで本研究ではエッジコンピューティングの特性を考慮した軽量なワークフローエンジンを提案する。

2. 関連研究

既存のワークフローエンジン[1][2]ではジョブを実行するワーカーノードとジョブおよびワーカーノードを管理するマスターノードから構成される。そして、マスターノードが起点となり各ワーカーノードに対してジョブの実行指令を送ることでジョブが実行される。しかしエッジコンピューティングでは、端末から送信された生のデータは情報の秘匿やオフローディングのためにエッジ外に出さずその内部で処理をさせたいという要求がある。そのためジョブを特定のワーカーノード内でのみ実行するようにし、端末から受信した生のデータをマスターノード含め他ノードへ流出しないように実行配置を定義・管理できる仕組みが必要となる。既存の構成ではジョブを実行するためにマスターノードや他ワーカーノードとデータを中継する必要があるため、これらの要求を満たすことができない。

3. 設計

エッジコンピューティングのワークフロー制御に必要となるジョブの記述とその記述に基づいた実行制御アーキテクチャについて説明する。

3.1 ワークフローの記述

ワークフローは複数のステップから構成される。

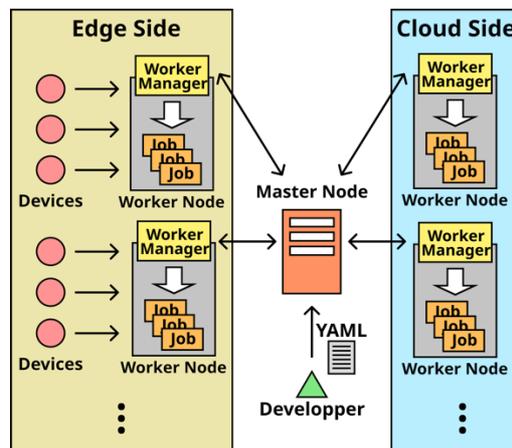


図1 ワークフローエンジンの構成図

ステップとは、実行したいジョブとジョブの実行制約を定義したものである。これには以下を定義する項目を設ける。

- そのステップで実行されるジョブ
- ステップ実行順序の依存関係
- ステップ実行失敗時に実行するステップ
- エッジ・クラウド指定
 - ステップの実行環境
- ラベル
 - ステップ実行ノードを指定するラベル

これによってエッジ、クラウド、特定のデータ基盤など、ジョブの実行配置先を指定することが可能になる。また、ジョブの定義では1つのジョブに対してコンテナイメージやスクリプトといった複数のアプリケーションを対応づけられるようにする。そして、特定のソフトウェアがインストール済みかどうかやプロセッサアーキテクチャなど、そのアプリケーションを使用可能な環境を指定する項目を設ける。これによってジョブを多様な環境で実行できるように定義することができる。ワークフローエンジンはこれらの定義を元にジョブの実行配置をスケジューリングする。また、ワークフローを開始する条件を指定するためにトリガという項目を設ける。この項目では日時などのスケジュールや、HTTP リクエストを指定することができる。

3.2 アーキテクチャ

システムの構成は端末、エッジノードのあるエッジサイドと、クラウドノードのあるクラウドサイド

```

name: object-detector
trigger:
  type: http
  path: /detect
jobs:
  - name: generate-image-id
    images:
      - type: docker
        arch: amd64
        image: tockn/generate-image-id:latest
      - type: shell
        arch: arm
        image: ./generate-data-id.sh
  - name: object-detect
    images:
      - type: docker
        arch: amd64
        image: tockn/object-detector:latest
  - name: store-result
    images:
      - type: docker
        arch: amd64
        image: tockn/store-detection-result:latest
  - name: store-image
    images:
      - type: docker
        arch: amd64
        image: tockn/store-image:latest
steps:
  - name: generate-image-id-step
    jobName: generate-image-id
    place: edge
  - name: store-image-step
    jobName: store-image
    labels:
      - db
    after: generate-image-id-step
  - name: object-detect-step
    jobName: object-detect
    place: any
    after: generate-image-id-step
  - name: store-result-step
    jobName: store-result
    labels:
      - db
    after: object-detect-step

```

図2 YAML で実際に定義したワークフロー

に大別される (図 1)。クラウド、エッジサイドにあるノードが、実際にジョブが稼働するワーカーノードである。これらのワーカーノードとワークフローはマスターノード (図 1 中央) により制御される。ワークフローは開発者が記述したものがマスターノードおよび全ワーカーノードに登録される。そして、マスターノードはワーカーノードから定期的送信されてくる資源情報を元に、ジョブを実行すべきワーカーノードを決定する API を提供する。ワーカーノードはこの API を用いることで、各ジョブを適切に実行することができる。

提案するワークフローエンジンではワークフローのトリガを HTTP リクエストに設定することで、マスターノードだけでなく端末が直接ワーカーノードに対してワークフローの開始指令を送ることができる。具体的には、端末がエッジのワーカーノードに対してワークフローのトリガとなる HTTP リクエストを送信し、リクエストを受けたワーカーノードは保持しているワークフロー情報を元にエッジ指定されているジョブを直接実行する。これによってエッジ外へデータを中継することなくジョブを実行することができる。エッジ指定されていないジョブ

STEP	generate-image-id-step	store-image-step	object-detect-step	store-result-step
IS PENDING				
IS READY	edge-worker	cloud-db-worker	cloud-large-worker, cloud-db-worker	cloud-db-worker
IS RUNNING				

図3 複数ワークフロー実行終了時のワーカーの状態

に関してはワーカーノードがマスターノードへ問い合わせるべきワーカーノードを決定し、ワーカーノード同士でジョブ実行リクエストを送り合うことで実行される。

4. 実装および評価

ワークフローエンジンは Go 言語を用いて実装し、マスターノードがワークフローやワーカーノードの情報を保存するためのデータベースとして MongoDB を用いた。また、ワークフローは YAML により記述することとし、定義した属性値に関して解析を行うパーサを用意した。

実装したワークフローエンジンを用いて学内バスの運行管理システムを例としてサービスの記述を行った (図 2)。このサービスでは端末で取得されたカメラ画像や GPS 等のセンサ情報から混雑度やバスの位置情報を提供する。ここでは、取得したデータの一意な ID を発行する処理をオフローディングのためにエッジノード内で行う。また、負荷の高い処理である物体検出は複数のワーカーノードが実行できるようにし、取得したデータや検出結果はクラウド上にあるデータ収集基盤に保存する。

評価として、実際にシステムを稼働させ、期待した動作と結果が得られることを検証する。評価を行う環境としてマスターノードの cloud-master、エッジノードの edge-worker、計算性能の高い cloud-large-worker、特定のデータ基盤を想定し db のラベルを付与した cloud-db-worker の計 4 台用意した。ワークフローは図 2 で示したものをを用いた。51KB の画像データを 2 秒間隔で計 10 回送信しワークフローを複数回実行した。実行終了時にマスタから得られた各ワーカーノードの状態を図 3 に示す。各ステップの IS READY にあるのが実行配置されたノード名である。ワークフローの定義を考慮し、適切にジョブが配置されたことがわかる。

5. まとめ

複数の環境に対応したジョブをエッジや特定のデータ基盤で実行配置が可能になるワークフローエンジンの設計と実装を行った。その結果、エッジコンピューティングの特性を考慮してワークフローが実行されることを示すことができた。

参考文献

[1] Treasure Data Inc. Digidag - simple, open source, multicloud workflow engine. <https://www.digidag.io/>.
 [2] S. Marru, L. Gunathilake, et al. "Apache Airavata: A Framework for Distributed Applications and Computational Workflows." 2011 ACM Workshop on Gateway Computing Environments (GCE'11), pp.21-28, 2011. DOI: 10.1145/2110486.2110490.