3N-6

AWS（自律型WEBサービス）ミドルウェアフレームワーク制御

二宮良太†，平本真道†，安齋太一朗†, 大谷真†

湘南工科大学†

1. はじめに

AWS(自律型 Web サービス)ミドルウェアのフレームワーク制御の目的は、動的協調したビジネスプロセスモデルに従って、商取引のためのユーザアプリケーションを駆動することである。一般に、商取引は並行実行される。しかし、従来のフレームワーク制御では、並行実行する商取引の数が多いときにスレッド数が過多になる問題があった。本研究ではフレームワーク制御にスレッドプール制御および中断制御を新たに追加することで、この問題を解決した。

2. AWS ミドルウェア

AWS ミドルウェア[1]は、独立に定義されたビジネスプロセスモデル（BPM）を持つシステム間で互いのモデルを実行時に動的協調（変形・整合化）することで、商取引が出来るようにすることを目的としている。フレームワーク制御は AWS ミドルウェアの一部で、動的協調された BPM にそってアプリケーションの実行を制御する。取引を実行するアプリケーション（以降 UserAP と呼ぶ）は、システムが持つ入出力オペレーションに対応するメソッド（AP メソッドという）の集まりとして記述される。フレームワーク制御は、BPM の状態変化に応じて対応する AP メソッドを自動的に実行して、下位のメッセージング層を使って相手システムとの間でビジネスメッセージの送受信を行う。

3. 従来のフレームワーク制御の問題点

一般に商取引は並行実行される。したがって、UserAP は複数同時実行される。すなわち、UserAP オブジェクトが複数生成され同時実行される。従来の AWS ミドルウェアでは、生成された 1 つの UserAP 並行取引に 1 つスレッドを割り当てることで実現していた。しかし、同時実行される UserAP オブジェクトの数は一般に多数で、かつ、1 つの取引の終了まで（すなわち 1 つの UserAP オブジェクトの処理完了まで）はかなり長い時間かかることも多い。このため、以下の 2 つの問題点があった。

問題点 1：UserAP オブジェクトの数が増加するに従って、生成されるスレッドの数も増加して、オーバーヘッドが増大していく。したがって、スレッドプールを適用することで、スレッド数に上限を持たせる必要がある。

問題点 2：UserAP オブジェクトは、実行中に（すなわち取引実行の途中で）相手からのメッセージを長時間待つことがある。場合によっては数日や数週間待つことも珍しくない。このため、スレッドプールを適用するだけでは UserAP オブジェクトが長時間待ちになると、スレッドが長時間開放されなくなり、新たな UserAP オブジェクトがスケジュールできなくなる（沈み込んでしまう）。したがって、長時間待ちになった UserAP オブジェクトを一旦実行中断して、一定時間経過後に再開始する処理（以降中断制御）が必要である。

本研究では、上記を解決するフレームワーク制御の機能（並行取引機能）を開発した。

4. 並行取引機能

4.1. AWS アプリケーション

図 1 は AWS アプリケーションを構成するクラスを示したものである。これらのうちユーザが作成するのは Main と UserAP である。AWSFramework と Manager はフレームワーク制御の一部であり、VLSession は下位のメッセージング層のインターフェースクラスである。従来と同様に、UserAP は AWSFramework を継承して作成する。これによって、その中の AP メソッドが AWSFramework によって自動実行される。Manager は今回新たに追加したクラスである。従来のフレームワーク制御では、Main が直接スレッドを生成して、UserAP を実行していたが、並行取引では、Main が Manager の execute メソッドを利用して UserAP を起動する。

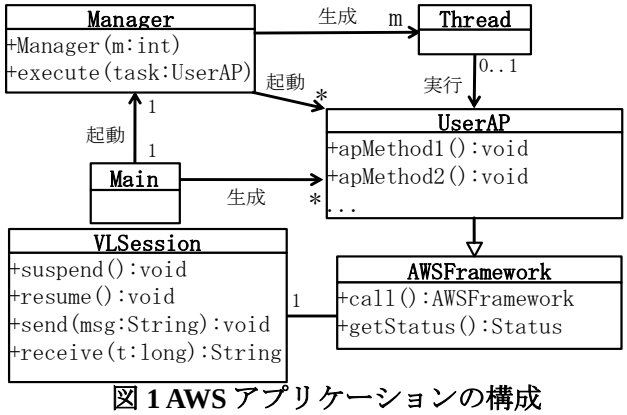

図 1 AWS アプリケーションの構成

Main のプログラム例を図 2 に示す。Manager オブジェクト生成時にスレッドプール内の最大

Framework for AWS Middleware

†Ryota Ninomiya, Masamichi Hiramoto, Taichiro Anzai, Makoto Oya, Shonan Institute of Technology

スレッド数 m を設定する。while 文の中は並行取引で UserAP オブジェクトが必要になった時点で、次々と UserAP オブジェクトを生成して execute で起動する。

```
Manager mgr = new Manager(m);
(new Thread(mgr)).start();
while (1){
   mgr.execute(new UserAp());
}
```

図 2 Main のプログラム例

4.2. スレッドプール

スレッドプールは Java の ExecutorService を用いて実装した。具体的には execute メソッドで ExecutorService の submit メソッドを実行するようにした。これにより、execute を実行すると UserAP オブジェクトは Ready 状態（実行待機状態）になる。その後スレッドに空きがあれば実行が始まる。一連の取引が終了したらスレッドを開放して、UserAP オブジェクトは消滅する。図 3 はこの様子を表したものである。

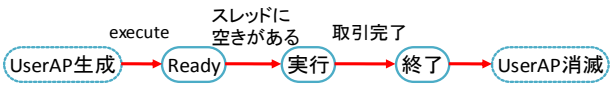

図 3 UserAP の状態遷移

4.3. 中断制御

3.の問題点 2 で述べたように UserAP は実行途中で長時間受信待ちになることがあり、図 3 の方法ではスレッドが長時間開放されなくなるため、実行開始を待っている UserAP オブジェクトが長い間実行状態にならなくなる（つまり取引が開始しない）。受信に使う VLSession の receive メソッドでは最大待ち時間を指定できる。これを利用して最大待ち時間を超えた場合には、その UserAP オブジェクトから強制的にスレッドを待機状態にすることで、他の UserAP オブジェクトのスレッドもスケジュールできるようにした。具体的な制御方法を図 4 に示す。最大待ち時間 s_n を超えると、待機状態にし、待機時間 t_n 後に receive を再実行することを繰り返す。s_n と t_n は、デフォルトとしては Exponential Backoff[2]を適用して、$s_n=2 \cdot s_n\text{-}1$，$t_n=2 \cdot t_n\text{-}1$ で求めることとした（s_0 と t_0 は UserAP で指定する）。

```
for (n=0; n<N; n++){
   receive(最大待ち時間=sn);
   if (受信成功) break;
   待機(待機時間=tn);
}
```

図 4 擬似コード

5. 並行取引状態の処理方式

5.1. UserAP オブジェクトの状態遷移

4.3 を実現するために UserAP オブジェクトは図 5 のように状態遷移させるようにした。UserAP が s_n 時間受信中になると、UserAP が待機状態へと変わる。この UserAP が t_n 時間待機状態になった後、再度 Ready 状態とする。

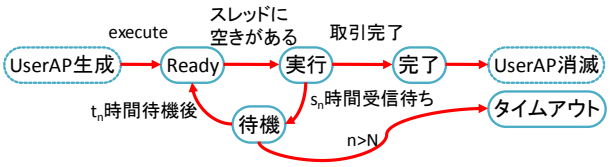

図 5 UserAP の状態遷移（改良後）

5.2. 中断待機処理

状態遷移においては、UserAP オブジェクトを残したまま待機状態にする必要がある（図 6）。まず、待機状態であることを保持して、UserAP オブジェクトは Manager にリターンする。Manager はリターンした UserAP オブジェクトが待機状態なのか、正常に完了したのかを調べる。待機状態なら受信待機キューに追加する。受信待機キューに入った UserAP オブジェクトは待機時間後に execute で再開始させる。

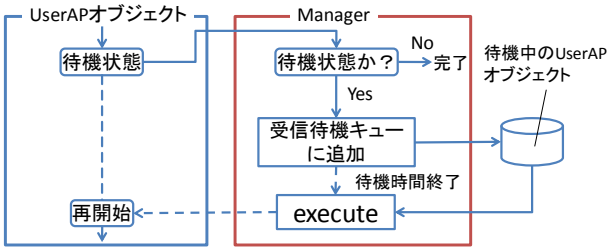

図 6 中断待機処理

6. まとめ

今回新たにフレームワーク制御に Manager クラスを追加することで多数の並行取引を処理することが可能になった。また、中断制御によって、UserAP オブジェクトがスケジュールできなくなる問題を解決することができた。本研究は科研費(21500110)の助成を受けたものである。

7. 参考文献

[1] Oya, M., Ito, M. and Kimura, T.: *Middleware for the Autonomous Web Services(AWS)*, IFIP I3E, Software Services for e-World, Springer, pp.5-16, 2010

[2] IEEE, *IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2007