マルチメディアチェックポイントプロトコルの Java による実装 *

4J−01

東京電機大学 理工学部 情報システム工学科†
平賀 研吾 桧垣 博章‡ §

1 背景と目的

コンピュータネットワーク技術の進歩により、異なるコンピュータに存在する複数のプロセスが互いに通信しあうネットワークアプリケーションを実行する分散システムが開発され、広く普及してきた。分散システムにおいて、故障の発生によって一部のプロセスが実行を停止した場合でもユーザに提供するサービスに矛盾を生じさせない耐故障性を実現するひとつの手法としてチェックポイントリカバリがある [1]。一方、多地点遠隔会議システムやバーチャルユニバーシティなどのマルチメディアネットワークシステムの開発が進められている。従来のチェックポイントリカバリ手法は、メッセージ通信イベントが瞬時に終了することを前提としている。従来の手法では、送受信イベントの継続中にチェックポイントを設定することができない。ところが、マルチメディアメッセージは従来のメッセージよりもサイズが大きく、その送受信イベントは一定時間継続する。そのため、新たな一貫性の定義とそれを用いた、プロトコルの設計が必要であった。論文 [2] では、ひとつのマルチメディアメッセージが、複数のパケットに分割されて送受信されることに着目し、閉区間 $[0,1]$ で評価する新たな一貫性を定義した。また、この一貫性を用いた QoS ベースのチェックポイントプロトコルを設計した。本論文ではこのプロトコルを Java[3] により実装したプロトタイプの作成について述べる。なお、システムの設計には UML[4] を用いている。

2 チェックポイントプロトコル

本論文で実装するマルチメディアチェックポイントプロトコルの手順を以下に示す。

[チェックポイントプロトコル]

1) コーディネータープロセス p_c がアプリケーションで要求される一貫性を指定し、チェックポイント設定要求 *Req* を各プロセスに送信する。
2) *Req* を受け取った各プロセスは、仮チェックポイントを設定し、p_c に確認応答 *Ack* を返信する。各プロセスは、p_c が一貫性を計算するのに必要な情報を *Ack* に付加する。
3) すべてのプロセスから *Ack* を受信したならば、p_c は一貫性を評価する。これが要求される一貫性より高いならば *Done* メッセージを、要求を満たしていなければ *Cancel* メッセージを各プロセスに送信し、チェックポイントプロトコルを終了する。

3 システム設計

3.1 各プロセスの基本構成

対象システムの構成は *Application* プロセスと一貫性の管理を行う *Coordinator* プロセスからなる。それぞれのプロセスのクラス構成は、以下の通りである。

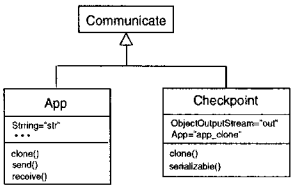

図 1: *Application* プロセス

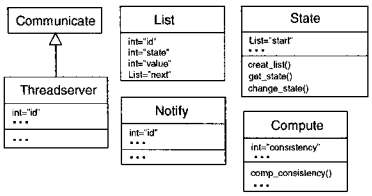

図 2: *Coordinator* プロセス

- *Application* プロセス (図 1)
  - *App* ··· アプリケーションプログラムを実行する
  - *Checkpoint* ··· *Coordinator* と制御メッセージを交換する。また、チェックポイントの設定を行なう。
- *Coordinator* プロセス (図 2)
  - *Notify* ··· *Req* メッセージの同期送信を管理する。
  - *State* ··· メッセージ送信の同期管理に使うリストの生成、管理を行なう。
  - *Threadserver* ··· *Application* プロセスの *Checkpoint* と制御メッセージを交換する。
  - *Compute* ··· 一貫性を計算する。

今回、システムを構成するプロセスの多くが、他のプロセスとの間で通信を行なう。そこで、通信に関する変数、メソッドからなるクラス *Communicate* を作成し、各クラスが継承することで、通信を可能にしている。 *Communicate* の構成を以下に示す。

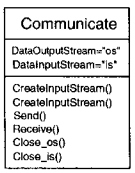

図 3: *Communicate* クラス

*Implementation of Multimedia Checkpoint Protocol in Java
†Tokyo Denki University
‡Kengo Hiraga, Hiroaki Higaki
§{hira, hig}@higlab.k.dendai.ac.jp

- *CreateOutputStream* ··· 出力ストリームの作成
- *CreateInputStream* ··· 入力ストリームの作成
- *Send* ··· パケットの送信
- *Receive* ··· パケットの受信
- *Close_os* ··· 出力ストリームのクローズ
- *Close_is* ··· 入力ストリームのクローズ

Send メソッドは *DataOutputStream* のメソッドである *writeUTF* でデータを出力ストリームに書き込み、*Receive* メソッドは *DataInputStream* のメソッドの *readUTF* で入力ストリームに書き込まれたデータを読み出す。

3.2 チェックポイント

チェックポイントは、正常動作しているプロセスの実行状態を安定記憶に格納することによって設定される。また、リカバリの際には、プロセスがチェックポイントにおける実行状態を復元できなければならない。Java の *Object* クラスに、オブジェクトの現在の複製を作成するメソッド *clone* がある。さらに、Java には、オブジェクトを復元可能なバイト列に変換する直列化を行なうメソッドがあり、ファイルに保存することが可能である。そこで、*clone* メソッドで *Application* プロセスのなかでリカバリに必要な *App* オブジェクトを複製を作成し、更に直列化を行なってファイルに保存する。リカバリの際には、保存したファイルから直列化されたオブジェクトを復元する。

3.3 *Req* メッセージの同期送信と一貫性の計算

Coordinator は 各 *Application* プロセスと接続するために、スレッド *Threadserver* を生成し接続を行う。そのため、各 *Threadserver* が 同じタイミングで *Req* メッセージの送信を行なう必要がある。 そこで、*ThreadServer* の同期管理を *Notify* クラスと *State* クラスを作成しリスト構造を用いることで実現する。リスト構造は *List* クラスを用いて実現する。*List* クラスのメンバを以下に示す。

- *id* ··· 識別番号
- *state* ··· 実行状態を記録
- *value* ··· 一貫性を記録
- *next* ··· 次の List の位置を記録

List は *Notify* と *Threadserver* が生成される際に *State* のメソッド *creat_list()* により生成される。各プロセスは同時に発行される識別番号 *id* を用いて、リストへアクセスを行なうことができる。リストでの管理によるメッセージ送信の同期管理方法を以下に示す。

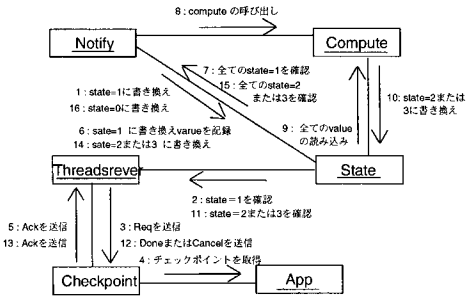

図 4: チェックポイント取得手順

1) *Notify* は自分のリストの *state* の値を *State* のメソッド *change_state* により 1 に書き換え、*Req* メッセージ送信を許可する。
2) *Threadserver* は *get_state* によりフラグが立ったことを確認すると、直ちに、*Application* へ *Req* メッセージを送信する。
3) 各 *Application* から *Ack* を受信すると、*Threadsrever* は *List* の要素 *value* に *Ack* と共に送信された値を書き込む。そして、*Req* 送信とチェックポイント取得が終了したことを示すため、自分のリストの *state* の値を *change_state* で 1 に書き換える。

全ての *Threadserver* が送信とチェックポイント取得が終了したことを、全リストの *state* から確認した *Notify* は *Compute* を呼出し、一貫性の計算を行なう。一貫性の計算から、チェックポイントの確定までを以下に示す。

4) *Notify* から呼び出された *Compute* クラスは、全リストの *value* から一貫性の計算を行なう。もし、計算結果が要求する一貫性を満足するのであれば、*Notify* のリストの *state* を 2 に、満足しない場合には 3 に書き換える。
5) *Threadserver* は 2 の場合には *Done* メッセージを送信し、各 *Application* へチェックポイントの終了を、3 の場合には *Cancel* メッセージを送信し、今取得したチェックポイントを破棄することを通知する。
6) メッセージを受信した *Application* はメッセージごとの処理を行い、*Ack* を送信する。*Ack* を受信した、*Threadserver* は自分のリストの *state* の値を *change_state* で *Notify* のリストの *state* と同じ値に書き換える。すべての *Threadserver* が送信を終了したことを、全リストの *state* から確認した *Notify* はすべての *state* の値を *change_state* で 0 に書き換え、チェックポイントプロトコルを終了する。

4 まとめと今後の課題

マルチメディアチェックポイントプロトコルを Java によって実装した。今後は、マルチメディアアプリケーションを対象として性能評価を行う。

参考文献

[1] Elnozahy,E.N., Johnson,D.B.,Wang,Y.M. "A Survey of Rollback-Recovery Protocols in Message-Passing Systems." Technical Report of Carnegie-Mellon University (1996).

[2] 平賀, 長谷部, 桧垣, "Consistent Global Checkpoint in Multimedia Network Systems," 第 8 回情報処理学会マルチメディア通信と分散処理ワークショップ論文集, pp. 253–258 (2000).

[3] "Java™ 2 SDK, Standard Edition Documentation," http://www.sun.com/j2se/1.3/docs.

[4] "Unified Modeling Language Resource Center," http://www.rational.com/uml.