出世魚プロジェクト：出世魚のUNIXへの移植

5G－9

岩崎孝司† 白光毅§ Yu Ge§ 牧之内顕文§
龍忠光‡
† 富士通九州通信システム § 九州大学情報工学科 ‡ 富士通研究所

1 はじめに

WAKASHIは九州大学工学部情報工学科牧之内研究室で提案されたオブジェクト指向データベースシステム「出世魚」の最下層部である分散共有永続ヒープ（distributed shared persistent heap,DSPH)ををを提供するストレージシステムである。

DSPHとは、図1のように複数のワークステーションをネットワークで結合した分散環境において、タスクの仮想メモリ空間に二次記憶場のファイルと１対１に対応した領域を設け、これをDSPHと呼ぶ。DSPH領域では、応用プログラムでファイル入出力を行なわなくてもデータの保存・再利用ができるだけでなく、通常のヒープ領域において揮発データを扱う場合と同様にファイル上の永続データを扱うことができる。またローカルファイルだけでなくリモートファイルのマッピングも可能である。主記憶・二次記憶のデータ一貫性はWAKASHIサーバが統一的に管理する。

本稿ではWAKASHIの分散型対応について説明する。

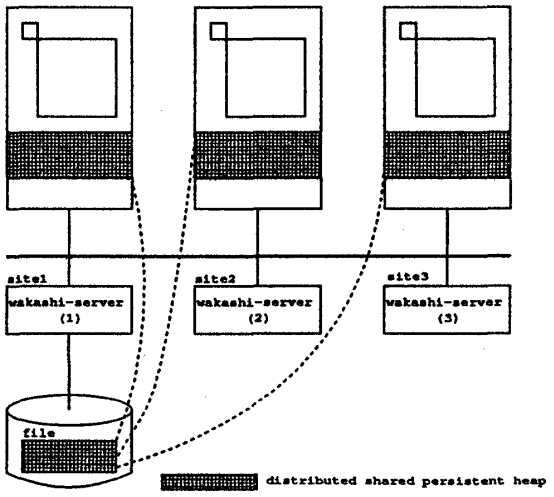

図 1: 分散共有永続ヒープの装置構成

SYUSSEUO project:Porting SYUSSEUO system to UNIX

†Takashi Iwasaki, Fujitsu Kyushu Communication Systems Ltd.

§Guangyi Bai & Yu Ge & Akifumi Makinouchi, Department of Computer Science and Communication Engineering, Kyushu University

‡Tadamitu Ryu, Fujitsu Laboratories

2 WAKASHIの分散型への対応

WAKASHIの分散型へのエンハンスを行なう際に考慮された点は以下の２つである。

1. 集中型WAKASHIの性能を改善する

2. 新しいサービス（トランザクション／リカバリ）の導入を容易にする。

1の要件を満たすためには前回の集中型WAKASHIで問題となった通信部分を可換にしておくこと、すなわちレイヤの分離が必要と思われる。同様に２の要件を満たすためには外部に拡張モジュールの付加が可能な構造でなければならない。

2.1 レイヤ分離

レイヤを図２に示すように２つに分離した。

2.1.1 WAKASHIシステムレイヤ

このレイヤの中では従来からのUserAdapter-Providerの枠を保って、自らは動作の決定を行なわない分散型のWAKASHIを構成する。

このレイヤにおいては、Behaviorが動作を決定する。BehaviorはWAKASHIシステムで発生した、様々なイベントをBehaviorとつながっている拡張機能モジュールに伝え、またそれらの拡張機能モジュールから要求を受けとる。

2.1.2 通信インフラレイヤ

一般通信インフラレイヤは、従来ソフト方式に依存して組み込まれてきたソケット／RPC（Remote Procedure Call）といった仕組みをWAKASHI本体から分離したものである。

このレイヤには通信リソースの個々のクラスと、上位レイヤから使用される、このレイヤにおけるクライアント／サーバがある。

2.2 外部モジュールの付加

2.2.1 Behavior

Behaviorは一般目的で作られたレイヤに特定の動作をもたらす、いわゆるエンティティの役割を果たす。

Behaviorはレイヤに応じたコールバックをWAKASHIシステムに提供することでWAKASHIの内部で生

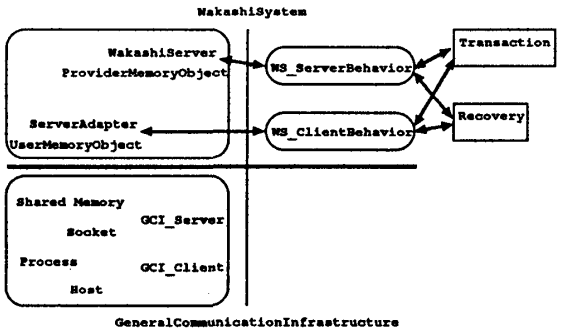

図 2: 分散型 WAKASHI の構成

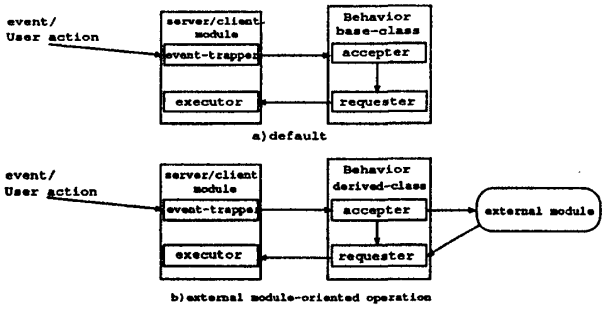

図 3: Behavior による動作決定

じる様々なイベントを取り扱い、またコールバックに対するアクションとして API コールを WAKASHI システムから提供してもらうことで、Behavior は外部モジュールと WAKASHI の協調動作を仲介する。WAKASHI システムでは発生した様々なイベントを Behavior とつながっている拡張機能モジュールに伝え、またそれらの拡張機能から要求を受ける。外部からの要求、外部への要求を Behavior で行なうことにより、基本機能に外部からの影響を及ぼさない様に、切り離すことができる。

また、Behavior の入れ換えによりミニマム動作 WAKASHI、トランザクション拡張の WAKASHI 等々のバリエーションを持たせることが出来る。

2.2.2 Behavior を使用した動作

Behavior を使って外部から WAKASHI（のクライアント／サーバ）の動作を決定するために

- クライアント／サーバ本体に event-trapper, executor を設ける
- Behavior に accepter,requester を設ける

ことが必要である。クライアント／サーバに設けられる event-trapper は「ユーザプログラムまたは任意のイベントから呼び出される」従来のメソッドインタフェースに相当する。しかしその処理内容は Behavior の accepter を必要なパラメータとともに呼び出すだけである。executer は従来のメソッドの処理内容そのものである。ただし、自オブジェクトからではなく、Behavior から任意の時点で呼び出されうる。accepter は Behavior に設けられたクライアント／サーバ本体の event-trapper と 1 対 1 に対応する Behavior のインターフェイスである。Behavior の規定クラスでは、単に同じ Behavior 内の requester を呼び出すだけだが、導出クラスにおいては適当な外部モジュールにその動作を任せることができる。

3 WAKASHI の性能改善

前回実装した集中型 WAKASHI においては 1 ページにアクセスする度に read/write 権の取得を行なう際、約 0.1 秒程度かかっていたため複数のページをバースト的にアクセスするアプリケーションにおいては実用に耐えなかった。原因は stream 型ソケットを使用し、ページフォルト毎にソケットの open/close を行なっていたためであり、通信方式の変更を行なうことでこれを解決しようとした。候補としては

- stream 型ソケット open 状態のまま持続して使用 (a)
- datagram 型ソケット (b)
- 共有メモリ通信 (c)

があった。(a) と (b) はほぼ同じ程度の性能であり、(c) より 100 倍程度速かったが、プロセス構成が変わらないことを考慮して、結果として (b) を採用することとした。

4 おわりに

以上述べてきたような拡張可能性と性能改善を目的とした分散型 WAKASHI のアーキテクチャであるが、これが実際にトランザクションやリカバリといった機能拡張に耐え、また実用上問題の無い程度まで全体のパフォーマンスをあげることができるかどうかは今後の実装ならびに性能評価を待たなければならない。