3K−3

分散システムのための拡張可能サーバ

風間 一洋
NTTソフトウェア研究所

1 はじめに

近年、コンピュータをネットワークで接続した分散環境は一般的に使用されるようになってきた。半面、分散システムの修正や機能追加は大規模になるほど大変な作業になり、異機種が混在するネットワークでは移植の必要性があるなど、分散システム開発は非常に困難である。従来のサーバシステムは、サーバ・クライアント間の通信はプロトコルという形で厳密に定められていて、サーバはあらかじめ決められた手続きしか実行できなかった。

本稿では、分散プログラムのプロトタイピング環境として、特定の環境に依存せずに、動的な変更・拡張が可能であるサーバと通信プロトコルを提案し、分散システムに適用した場合の有効性を述べる。

2 拡張可能サーバ

サーバはプログラミング言語を実行するインタープリタであり、ネットワーク経由でプログラムの送信・受信や実行ができる。プログラムを送信することでサーバの機能の変更や追加が可能である。この処理形態は、ページ記述言語 POSTSCRIPT[1] やネットワークファイルシステム NeFS[4] 、分散 lisp 環境[2]などで採用されている。

拡張可能性

分散システムの拡張(変更)可能な部分を、次の4種類に分類する。

1. データ
2. プログラム
3. 例外処理(エラー処理など)
4. インタープリタ

この順序で変更可能な段階が進むほど拡張可能性が高いと定義する。分散システムが拡張可能なためには、データとプログラムの変更が可能であることが最低限要求される。この機能の究極な例は、**自己反映計算モデル** (reflective computation model)[3]である。

機種独立性

拡張可能サーバを記述する言語は、分散システムの動作を記述するための中間言語の役割を果たし、異なる機種で動作するサーバの上でも同じプログラムを動かすことが可能になる。

An Extensible Server for Distributed Systems
Kazuhiro KAZAMA (kazama@nttlab.ntt.jp)
NTT Software Laboratories

インタープリタ

サーバの拡張性は、そのインタープリタの構成にある。これはスタック指向のインタープリタであり、取り扱う全てのオブジェクトはデータであるリテラルと実行可能なエグゼキュータブルの2種類に分類される。リテラルの場合には、オペランドスタックにプッシュされ、エグゼキュータブルの場合には、実行スタックにプッシュしてオペランドスタックの値を取り出して実行した後に、結果を再びオペランドスタックにプッシュする。変数や手続きはキーと値の組で辞書に格納し、実行時にキーを使って辞書スタックの再上位の辞書から検索する。インタプリタの基本的な変数や手続きを定義してある辞書は、辞書スタックの再下位にある。

インタープリタの読み込み・実行ループはメタインタープリタの形で実現され、サーバ起動時にこの手続きの名前オブジェクトを実行スタックにプッシュする。これはサーバの動作に必要な初期化、通信設定、エラー回復などの処理もおこなう。この実行中にエラーが検出されると、エラーごとに決められた名前オブジェクトを実行スタックにプッシュする。デフォルトのエラー処理が提供されているが、プログラムの中でエラーを検出してエラーの種類を判断し処理することもできる。

拡張方法

本インタープリタはこのように拡張性と簡潔性を兼ね備えている。インタープリタの核は組み込みオペレータを必要最小限に制限しているために非常にコンパクトである。実際のプログラムに必要な手続きは、新たに手続きを定義していくことで機能を拡張できる。

また組み込みオペレータやインタープリタのメインループ、エラー処理も、直接実行せずに名前オブジェクトをキーにして辞書スタックを検索して発見した手続きを実行するために、プログラマが再定義することが可能である。

デバッグ

人間がサーバに直接アクセスして対話型式で処理ができるので、エラー発生時の各スタックのスナップショットやメモリ使用状況、各変数の値などの情報を自由に参照できるように、エラーハンドラを変更できる。

3 拡張可能サーバプロトコル

拡張可能サーバの通信プロトコルは、OSI 参照モデルに基づいて表1のように階層構成になっている。現在プロセス間通信をするために、UNIX のソケットを使っている。

通信プロトコルの一部としてシステム自身を記述する

アプリケーション層	ユーザプログラム
プレゼンテーション層	言語仕様
セッション層	UNIXのソケット

図 1: 通信プロトコルの階層構成

プログラミング言語を用いていることは、次の利点がある。

1. サーバの基本機能を用いて必要な機能を作成できる
2. クライアント側からサーバの機能追加・バグ修正ができる
3. 条件判断、繰り返しなどの制御をサーバ側で実行できる

3.1 通信の最適化

拡張可能サーバの通信内容は、実行に必要な手続き定義部分と手続き実行部分の2種類に分類できる。拡張可能サーバのプロトコルは、従来使われてきたプロトコルより柔軟度が高いが、手続き定義部分の情報転送量が多くなりやすい欠点がある。この欠点を通信回数と通信量に関してカバーするために、次のような通信最適化手法を適用する。

通信回数の減少

1. サーバ側での条件判断、繰り返しなどの制御構造の使用

通信量の減少

1. 未定義の手続きの検出・送信
2. サーバに送信する手続きの共用
3. 手続きのサーバのファイルシステムへの書き込みと再使用

4 RPC (Remote Procedure Call)

分散システムを構築によく使われるプロトコルとして、RPCがある。現在使用されているいくつかのRPCの実装では、遠隔の手続きをプログラム番号とバージョン番号を使って呼び出す。もしサーバに手続きが存在しなかったり、サポートされていないバージョンの場合は、手続きは実行されない。本サーバではクライアントからサーバに手続きを送信できるので、この場合にも実行できる。さらにサーバ側を停止したり変更したりせずに、RPC手続きをクライアント側から変更できる利点がある。

これを手続きを呼び出す前にサーバ側に問い合わせて判断する方法では、通信のオーバヘッドが生じる。本稿では図2のように手続きの未定義はインタープリタによってバージョンの不一致は手続き自体がバージョン番号で判断し、返答を返す代わりにエラー発生をクライアントに知らせる方法を提案する。クライアントのスタブがこのエラーを検出した時だけ、必要な手続きを送信して再実行する。この処理はスタブ内部で実行されるので、プログラマは意識する必要がない。

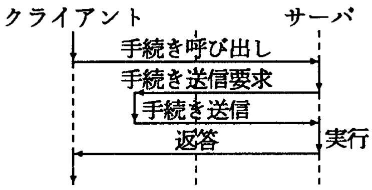

図 2: 拡張可能サーバのRPC

5 おわりに

拡張可能サーバは、インターネットワーク上で動作する分散システムを構築する基本システムとして作成した。今後は、次の点に注目して研究をおこなう予定である。

マルチエージェントシステムへの適用

今回はクライアント・サーバモデルを取り上げたが、これはサーバだけが拡張可能な異種システムの間の通信であり、結果としてシステム単体としての拡張性についてしか検討しなかった。今後はマルチエージェントシステムに適用して、拡張可能エージェントによって構成された分散システム全体としての拡張性について検討する。

分散プログラム開発環境の充実

今後はサーバをネットワーク上の各マシン上に配置して、簡単に分散システムのプロトタイピングができるようにする。特にサーバとの対話型の開発とサーバ間のプログラム転送を利用して、分散したシステムを統一的に開発できるような環境を作成する。

拡張可能性の表現の検討

システムに対して変更可能な部分が多いほどよいわけではなく、それをプログラマに対してどのような形で提供するかが重要である。さらに簡潔性、拡張性に優れてデバッグの容易な言語仕様を検討する。

参考文献

1) Adobe Systems Inc., : *PostScript Language Reference Manual*, Addison Wesley(1985).
2) 天海良治, 島原美樹: 分散記号処理プログラミングワークベンチ, 日本ソフトウェア科学会第7回大会論文集, 東京大学 工学部 (10 1990), 日本ソフトウェア科学会, 197–200.
3) Maes, P.: COMPUTATIONAL REFRECTION, Technical report, Vrije Universiteit Brussel, Belgium(1987).
4) Sun Microsystems Inc., : Network Extensible File System Protocol Specification, Draft(1990).