Unix上でのRPCサーバの移送方法とその実現について

1D－4　堀 正弘　伊藤 光恭
NTT ソフトウェア研究所

1 はじめに

1.1 マイグレーション

プロセスマイグレーション(プロセス移送)は、ネットワーク全体のパフォーマンスや信頼性の向上に有効な、負荷分散機能や障害処理機能を実現するために使用される。しかし現存するネットワーク分散環境の多くは、移送機能を持たないUNIXやMS-DOS、MS-Windowsといったosをプラットフォームにしているため、負荷分散機能の実験を行う環境でさえ構築するのは困難である。

そこで、これらの実験を容易にし、さらにUNIX上の開発ツールを活用できるようにするために、UNIXマシン(Sun-OS)上でRPCサーバの移送を実現する方法を考察し、その一部を実装した。

1.2 プロセス間通信

Unixのプロセス間通信の代表的な方法として、ソケットとRPCがある。前者は比較的低レベルのシステムコールが用意されており、プログラマによるきめ細かな管理が必要とされる。一方後者は抽象度が高いため、一般にプログラミングが容易で、保守性の高いプログラムを得ることができる。

ソケットで通信を行うプロセスの移送については、[4]で発表されている。そこで今回はRPCベースでのプロセス移送について検討、実装した。

2 実装

2.1 プロセス状態の保存

UNIXプロセスを移送する方法が[3]に提案されている。この方法では、通信路を除くほとんどのプロセス実行状態が保存/復帰されるが、データ量が増えるためホスト間の通信量も大きくなる。今回は、移送対象をRPCサーバに限定し、ホスト間の通信量を抑えるために、移送前後で保存されるべき変数名をプログラマが特定のヘッダファイルに記述する方法を採った。このヘッダファイルをコンパイル前に前処理し、変数の保存/復帰ルーチンをソースコードに埋め込む。

指定された変数は、移送が起動された後、ファイルに書き出され、移送先ホストに遠隔コピーされる。このデータファイルは、移送先でプログラムが再び立ち上げられたときに、初期化部分で読み出され、移送前の値が復帰される。

2.2 通信路の保存

C言語用に用意されているRPCライブラリは、ソケットのライブラリに比べて抽象度が高いため、低レベルの処理を行うことは難しい。今回の実装では、クライアントの要求に対してサーバが遠隔手続きを処理している間には、移送を起動しないことにした。

サーバが遠隔手続きを実行している間に発行された移送要求はペンディングされる。クライアントの要求に対する遠隔手続き処理が完了し、応答をサーバが返送した直後に実際の移送処理が実行される。

従って、移送の処理中には、通信路中にデータが存在しなくなるので、通信路の管理処理を大幅に簡素化することができる。

2.3 構造

各ホストには移送サーバデーモン(以下MD：Migration Server Daemon)が常駐し、RPCサーバプロセスへの移送開始通知やファイル転送等の処理を行なう。RPCサーバおよびクライアントには移送システム専用のライブラリがリンクされ、MDと通信しつつ移送を実現している。

2.4 手順

図1に移送システムでの移送の実行の様子を示す。

(1) 初期状態では、クライアントとサーバとが通常のRPCで通信している。

(2) MDがwell-knownのポートを介して移送開始の要求を受け取ると、MDは、移送されるRPCサーバプロセス(以下SP)にユーザ定義シグナルを送る。シグナルを受け取ったSPは内部フラグを立ててRPC要求を待つ。これはRPCのサービス中に移送要求が発生した場合とそうでない場合との処理を共通にするためである。
サーバはRPC要求が発生した時点で指定された内部変数をファイルに書き出す。但し、SPが遠隔関数を実行している間は、その関数が終了するまで待ち、終了し次第ファイルへの書き出しを開始する。

(3) MDは、移送開始要求で指定された移送先ホストに存在するMDとソケットで通信路を確立し、RPCサーバのプログラムファイルと、(2)で生成されたデータファイルを遠隔コピーする。

(4) 遠隔コピーが完了すると、移送先ホストのMDは移送元ホストのMDに通知した後、プログラムファイルを起動する。プログラムファイルはデータファイルを読み込んで、移送前のサーバの状態を復帰し、クライアントからのRPC要求を待つ。

(5) 以上の処理が完了した後、不要になったデータファイルを削除する。

一方、(2)でMDからのシグナルを受け取った後のSPがクライアントからのRPC要求を受信すると、以下の処理を実行する。

(a) 受け取ったRPC要求に対し、クライアントに向けて不当応答を返す。

'An Implementation of Unix RPC Server Migrations'
Masahiro HORI, Mitsutaka Ito, NTT Software Laboratories
3-9-11 Midori-cho Musashino-shi Tokyo 180 Japan

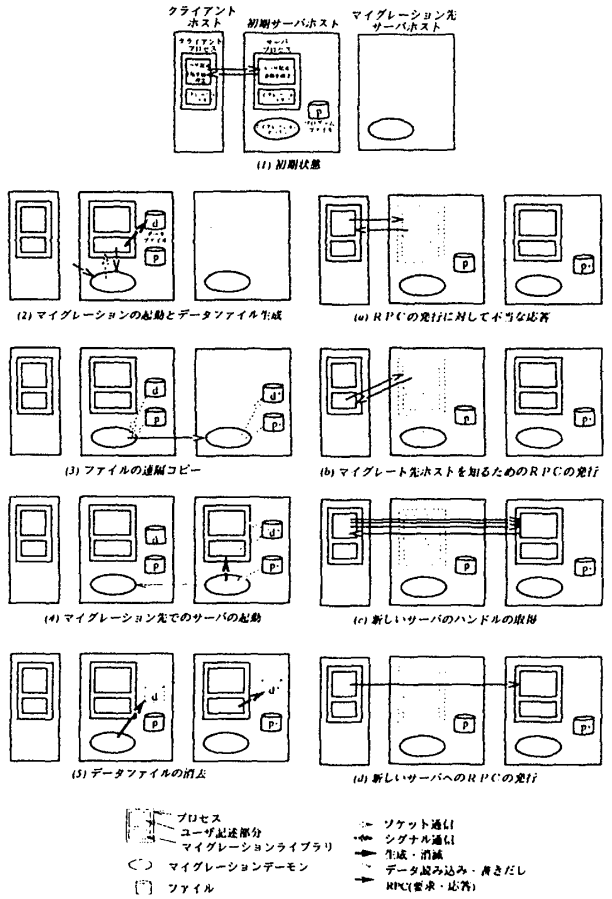

図 1: 移送の手順

(b) この不当応答を受信したクライアントは、SP の移送先ホスト名を得るための RPC 要求を、改めて送出する。

(c) クライアントは移送元ホストに対する RPC ハンドルを廃棄し、移送先ホストに対するハンドルを新たに生成する。この時点で移送先の新 SP が起動していないときは、ハンドルの生成要求に対して NULL が返されるので、正常にハンドルを取得できるまでリトライする。

(d) 取得したハンドルを用いて、(2) で送出した本来の RPC を発行する。

実行シーケンスの一例を図 2 に示す。

3 評価

移送するプログラムは C で記述された rdict[?]dict という辞書プログラムのサーバで、MD 他一部を c++ で記述している。実行環境は、SparkLT (CPU Super Spark 33MHz Memory 64Mbyte) を使用し、コンパイラには gcc を用いた。

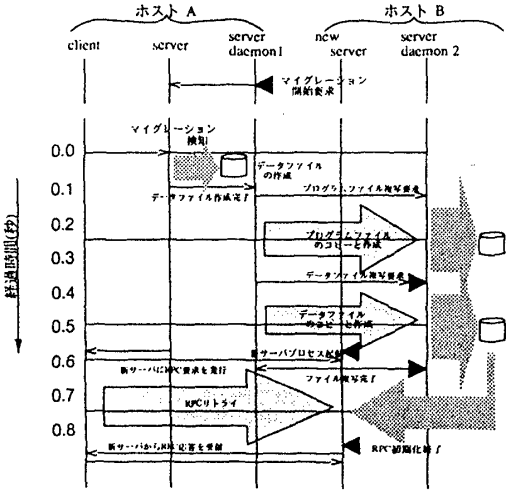

図 2: 実行シーケンスの一例

3.1 プログラムサイズ

実行プログラムサイズの増分は、約 16kbyte 、サーバで約 33kbyte である。増加分は本来のプログラムサイズにはほとんど関係無いため、プログラムの規模が大きくなる程移送システムの影響は小さくなる。

3.2 実行時間

移送の全ての手順が終了し、クライアントが新しいホストから RPC 応答を受け取るまでに 0.93 秒を要した。このうちデータファイル作成に要する 0.08 秒とプログラムおよびデータファイルの遠隔コピーに要する 0.41 秒は、プログラムファイルの大きさと保存されるデータ量とに依存する。仮にこれらの時間が RPC サーバのプログラムサイズに比例するとすると、移送に要する時間は、プログラムサイズが 130kbyte 程度で約 1.4 秒、1Mbyte 程度になると約 9 秒となり移送にかかるコストは非常に大きくなってしまう。従って負荷分散機能を実現する場合には、対象プログラムの大きさによって移送のタイミングを制御するなどの戦略が重要になる。

4 おわりに

今後は、処理速度の向上、保存したい変数宣言の簡略化、負荷分散などへの応用等の課題を解決していく予定である。

参考文献

[1] 前川守, 所真理雄, 清水謙多郎, "分散オペレーティングシステム UNIX の次にくるもの", 共立出版株式会社, 1991.

[2] Douglas E. Comer,avid L.Stevens, "Internetworking with TCP/IP volume III client-server programming and applications bsd socket version", Prentice-Hall International, Inc.

[3] 森山茂男, 多田好克, "利用者レベルで実現したプロセス移送ライブラリ", 情報処理学会研究会 OS 研究会報告, 91-OS-51, pp 41-47, 1991.

[4] 白木原敏雄, 金井達徳, "通信を行うプロセスの移送機能の設計と実装", 情報処理学会論文誌, vol.34. No.6, pp 1457-1467, Jun. 1993.