# 分散プロセスを一括管理するプロセスマネージャの実装

1P－11

石川真二　河野健二　益田隆司

(株) 日立製作所 公共情報事業部　東京大学大学院 理学系研究科 情報科学専攻　東京大学大学院 理学系研究科 情報科学専攻

## 1　はじめに

分散アプリケーションを運用・管理する場合，分散したプロセスの一貫性を保ちつつ，それらを個別に運用・管理しなければならない．本研究では，分散したプロセスの一括管理を可能にする分散プロセスマネージャを提案する．本方式の特徴は，分散アプリケーションの開発段階で必要となるプロセス管理の自動化を行なっている点にある．アプリケーションの開発段階では，プロセスの起動・終了などを頻繁に行なうだけでなく，プロセス間で送受信されるメッセージの内容に応じて緻密なプロセス管理が必要となる．提案方式では，プロセスの状態およびメッセージ送受信の制御・監視を可能にし，スクリプトによる柔軟なプロセス管理を実現している．

## 2　分散アプリケーションの開発段階におけるプロセス管理

分散アプリケーションの開発段階では，デバッグ，動作試験，耐久試験などのために，複数のプロセスを頻繁に起動させたり停止させたりする必要がある．また，プロセス間で送受信するメッセージの内容を監視し，意図した通りにメッセージがやりとりされていることを確認したり，プロセスの異常動作の検出を行なったりする．

例えば，アプリケーションの動作確認を行なうために，プロセスの起動時のパラメータを変更しながら動作確認を行なう場合を考えよう．このような場合，プロセスの実行状態や送受信されるメッセージを監視して動作の確認を行ない，プロセスが異常終了したり不正なメッセージが送受信された場合はログを残してすべてのプロセスを終了させる．その後，ふたたび別のパラメータでプロセスを起動しなおしてから動作チェックを継続する．

このようなプロセスの状態やメッセージの監視を行なうには，そのためのコードをアプリケーションに追加しなければならず，分散アプリケーションの開発を一層繁雑なものにしてしまう．さらに，アプリケーションの開発が完了した段階では，こうした監視のためのコードは不要になる場合が多い．

本稿で述べる手法では，プロセスの制御・監視およびメッセージの送受信の制御・監視をアプリケーションとは別にスクリプトとして記述できるようにする．このスクリプトでは，プロセス間の同期をとりながら分散アプリケーション全体で一貫性を保つことを可能にしており，複数の計算機上で動作する分散プロセスの一括管理を可能にしている．

## 3　分散プロセスマネージャの設計と実装

### 3.1　システムの構成

本論文で示すシステムは図1で示すように (1) プロセスの状態の制御・監視を行なうアクション・プロセス，(2) アクション・プロセス間の同期を取る同期プロセス，(3) メッセージの制御・監視を行なうプロキシ・プロセスで構成される．スクリプトには実行するコマンド，期待する実行結果，メッセージの制御・監視用スクリプトなどを記述する．本システムはスクリプトにしたがって制御・監視を自動実行する．

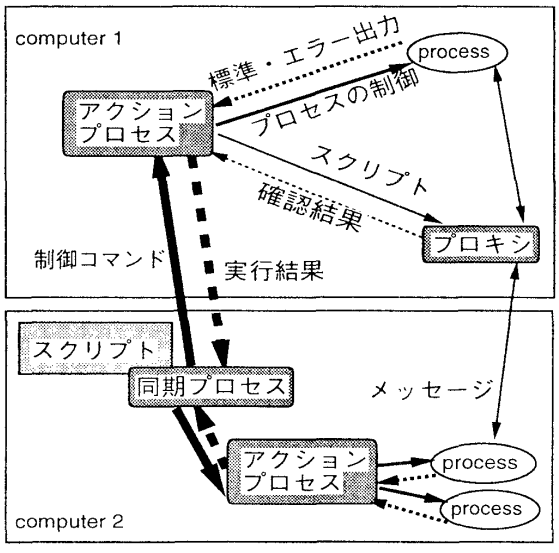


図 1: システム構成図

Implementation of Process Manager for Centralized Management of Distributed Processes.

Sinji Ishikawa, Kenji Kono and Takashi Masuda.

Hitachi Co.,Ltd., Dept. of Information Science, University of Tokyo, Dept. of Information Science, University of Tokyo.

### 3.2 プロセスの管理

プロセス管理のための機能として，アクション・プロセスはプロセスの生成，終了，シグナルの発行などを行なう．また，それらの機能の実行結果としてプロセスの終了コードや標準出力および標準エラー出力を取得する．

同期プロセスはスクリプトに従って，逐次これらアクション・プロセスに制御コマンドの形で指示を与えていく．その実行結果を同期プロセスが受けとり，評価することでプロセス間の同期をとる．

スクリプトには柔軟な記述が可能で，複数のプロセスを並行して実行させ，その中の任意のプロセスの正常終了を待って別のプロセスを実行させる，といった記述も可能である．

### 3.3 メッセージの送受信の管理と制御

メッセージの送受信の管理と制御として，プロキシ・プロセスはメッセージの内容の確認と変更の機能を提供している．プロキシ・プロセスは同期プロセスからアクション・プロセスを経由してスクリプトを受けとる．このスクリプトはプロキシ・プロセスの動作について記述したもので，プロキシ・プロセスの動作を柔軟に指示することが可能である．

メッセージの内容の確認・変更を行なうにはポート番号のみを変更し，プロキシ・プロセスを経由して通信を行なう．メッセージの送受信を行なうプロセスがプロキシ・プロセスに接続すると，スクリプトで指定された送受信対象プロセスと接続を行なう．それ以後，プロキシ・プロセスはメッセージを橋渡ししながら，必要に応じてその内容をログとして記録する．スクリプトに変更対象となるメッセージを定義しておくと，プロキシ・プロセスはそのメッセージを受信する度に，スクリプトで指定した内容に改変して送信する．

## 4 実験

プロセス間のメッセージ送受信の制御・監視を行うことにより，そのためのオーバーヘッドが発生する．このオーバーヘッドが大きいと，制御・監視する分散アプリケーションの動作に影響を及ぼす場合があり，アプリケーション開発段階でのプロセス管理に不適切である．現段階での実装を用いて，オーバーヘッドの大きさを計測する実験を行なった．

実験環境はサーバに JCC JU1 (UltraSPARC 200MHz, 主記憶 160Mbyte, Solaris 2.5.1) を，クライアントに Sun Ultra1 (UltraSPARC 167MHz, 主記憶 128Mbyte, Solaris 2.5) を用いた．ネットワークは 10Mbps のイーサネットである．

実験では，ftp クライアント - サーバ間において 10M バイトのファイルを転送させ，その実行時間を測定した．実験結果を図 2に示す．分散プロセスマネージャを利用しない場合 (非経由)，転送したファイルの内容をログとして保存しなかった場合 (経由 log 無し)，転送したデータをログとして保存した場合 (経由 log 有り) について測定した．ログはサーバ上のローカルファイルシステムに書き出される．

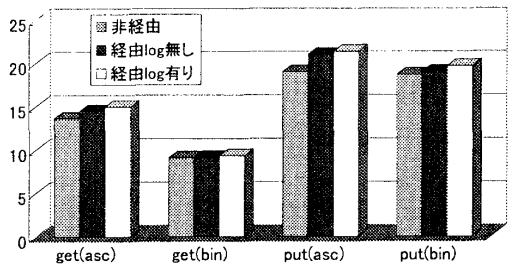


図 2: 10MB のファイルの ftp 転送の実行時間 (秒)

図 2の縦軸は実行時間 (秒) で，get コマンド，put コマンドについてはそれぞれ ascii モードと binary モードで実行した．どのコマンド・モードにおいても同様のオーバーヘッドであり，最大で put コマンド ascii モード時に非経由 19 秒，経由 (log 有り)21.33 秒，12.3% のオーバーヘッドであった．このグラフより，転送したデータのログを残したとしてもそのオーバーヘッドは全体の 10% 程度であり，このプロセス間のメッセージ送受信の制御・監視機能は分散アプリケーションに大きな影響を与えずに実行できるといえる．

## 5 おわりに

分散アプリケーションを運用・管理する場合，分散したプロセスの一貫性を保ちつつ，それらを個別に運用・管理しなければならない．この問題を解決する一つの案としてプロセス管理を自動実行する分散プロセスマネージャを実装した．また，さまざまな分散アプリケーションに汎用的に用いることのできる，プロセス間のメッセージ送受信の制御・監視機能を実装し，より高度なプロセス管理を可能とした．今後はより大規模な分散プロセスにも対応していけるように，耐故障性なども考慮した実装を進める．

## 参考文献

[1] John K.Ousterhout 著, 西中 芳幸 / 石曽根 信 訳, "Tcl and the Tk Toolkit", Addison-Wesley Publishing Company, ソフトバンク株式会社, 1994.

[2] W.Richard Stevens, "UNIX Network Programing", Prentice-Hall, Inc. 1990.