オペレーティング・システム 48－7
（1990. 9. 7）

V 分散システム

下島 健彦
日本電気(株)

スタンフォード大学で開発されているV分散システムについて紹介する．Vシステムは，分散カーネルとサーバの集合により構成される．分散カーネルがネットワーク透過な通信を提供し，それ以外のOS機能を通信によって起動されるサーバで実現することで，ネットワーク透過なOSを実現している．ネットワーク・プロトコルとして，VMTPと呼ばれるRPCを効率よくサポートするプロトコルを採用していること，通信の速度を重視した数々の最適化を行なっていること，プロセスのグループに対する通信(マルチキャスト)を提供していること，分散した名前管理を行なっていることなどが特徴である．

The V Distributed System

Takehiko Shimojima
NEC Corporation

This paper is an introduction of the V distributed operating system. The V consists of a distributed kernel and user level servers. The kernel provides network transparent interprocess communication (IPC) facility, process and memory management and device drivers, while servers provide other operating system functions invoked through IPC. The V uses the VMTP transport protocol which is optimized for request-response behavior, makes several optimizations for fast IPC, provides multicast to a group of processes, uses decenterized naming management.

1 概要

V分散システムはCheriton教授の指導のもとに1980年代初めからスタンフォード大学で開発されている分散システムで,現在も拡張や新しいアイデアの実験環境として開発が続けられている.著者は1988年夏から1年間スタンフォード大学に留学する機会を得,Vシステムプロジェクトに参加した.本論文ではその経験を元に,Vシステムの紹介を行なう.

1.1 設計思想

文献 [3] に述べられているように,Vシステムでは次の三つの設計思想が掲げられている.

1. 高速な通信の重要性
   分散システムは高速な通信を基礎として,(そのコスト意識の上に) 設計されるべきである.また,通信はスループットだけでなく,レスポンス性能を重視したトランザクション・オリエンテッドなものが必要である.

2. プロトコルの重要性
   ネットワーク・プロトコルだけでなく,I/O,ネーミング,認証などのプロトコルがシステムを規定する.大学での研究の目的は,そういったプロトコルを開発することにあり,システム作りは,開発したプロトコルを検証するためのものと考えている.

3. 小さい分散カーネルとサーバによるシステム構成
   通常のOS機能は,比較的小さな分散カーネルとプロセスレベルのサーバにより実現できる.分散カーネルはネットワーク透過な通信とプロセスの走行環境であるプロセス管理,メモリ管理を実現する.

   VシステムではOSの機能をサーバで実現し,サーバへの処理要求をネットワーク透過な通信によって行なうことで,OS機能をネットワーク透過にしている.

1.2 アプリケーション分野とハードウェア環境

分散カーネルは特定のアプリケーション分野に依存せず,汎用に設計されている.例えばプロセス・スケジュリングもプリエンプティブ固定プライオリティの単純なものが提供されており,リアルタイムシステムもTSSのような環境もサポートできるようになっている.

サーバやコマンドを含んだVシステム全体としては,主にプログラム開発などを行なうワークステーションのようなアプリケーションが想定されており,エディタ,パラレルmake,デバッガなどのコマンド群が用意されている.さらに,Vシステムをファクトリ・オートメーション (FA) に応用した例もあるとのことである.

ハードウェア環境としては現在,Sun2, Sun3, μVAX-II, DEC3100(RISCワークステーション), DEC Firefly(密結合型マルチプロセッサ), VMP [5] (Cheriton教授のもとで開発されているソフトウェア制御のキャッシュを持った密結合型マルチプロセッサ) 等が一つのネットワーク (イーサネット) に混在できるヘテロジニアスな環境がサポートされている.ユーザインタフェースとしてはビットマップとマウスが想定されている.

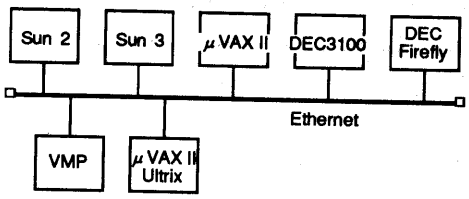

図 1: ハードウェア構成

2 分散カーネル

分散カーネルはネットワーク透過なプロセス間通信(IPC)とプロセス管理, メモリ管理, デバイスドライバなどを実現している. このうち, IPC以外は全てサーバ(カーネルサーバと呼ばれる)の形で実現されている. カーネルサーバについては2.7で述べる.

2.1 VのIPC

Vシステムでは図2のような同期型のIPCが提供されている. VのIPCの特徴として次の点があげられる.

- RPCに適した単純なプリミティブ.
- 固定長メッセージ(32バイト)+可変長データ・セグメント.
- RPCに適した通信プロトコルVMTP.
- 通信のためのカーネルの最適化.
- マルチキャストのサポート.

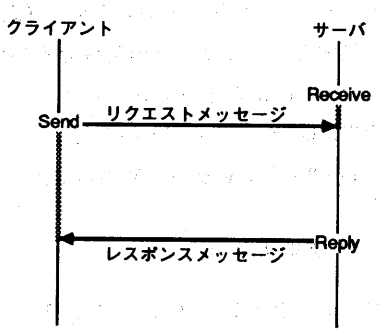

図2: VシステムのIPC

2.2 IPCプリミティブ

IPCプリミティブはSend-Receive-Replyを基本としている. 特にSendは要求の送信とレスポンスの受信というRPCでの通常ケースを1プリミティブで行なうため, オーバーヘッドが少い. また, クライアントが指定した送信バッファにレスポンスを直接書き戻すことでバッファ管理も単純化している.

2.3 固定長メッセージ

IPCは32バイトの固定長メッセージを基本とすることでバッファ管理を単純化している. 32バイト以上のデータの送受信が必要な場合は可変長のデータセグメントを付加する. 実際にVシステムでメッセージ・トラフィックパターンを調査した結果 [9], 5割が32バイト以下のデータであることが報告されている.

2.4 VMTP

VMTP [4, 10] はVプロジェクトで開発されたトランスポート層のプロトコルで次のような特徴を持つ.

- コネクションの設定 / 切断がなく, リクエスト / レスポンス主体の通信に向いている.
- レスポンスがリクエストのAckを兼ね, リクエストが直前のレスポンスのAckを兼ねることで最小2パケットでRPCを実現.
- ヘッダ中に固定長メッセージが入る.
- マルチキャスト, フォワーディング, ストリーム, プライオリティなどの豊富な機能.
- 選択的再送, レートベースのフロー制御など, 高速, 低エラー率のネットワークへの対応.
- Idempotentなレスポンスの最適化.

再実行しても同じ(あるいは意味のある)結果の得られる操作をIdempotentな操作という. Idempotentな操作では, 通信が失敗した場合などに, レスポンスを再送せず, クライアントが要求を再度行なえばよい. これに対応しVMTPはサーバ側で再送のためのレスポンスの保存を省略する.

2.5 通信のためのカーネルの最適化

VのIPCの基本はブロック型のメッセージ送信であるため, プロセスは通常一度に一つのメッセージ送信しか行なわない. カーネルではプロセス管理構造体中にVMTPヘッダを含むことで, バッファ

管理を簡単化している．また，ヘッダの一部のフィールドはプロセス初期化時に値を設定してしまう．

2.6 マルチキャスト

Vシステムの IPC の特徴の一つにマルチキャストがある．V の通常の IPC は，通信のエンド・ポイントとしてプロセスを指定する．V ではプロセスの他にプロセス・グループを通信のエンド・ポイントとして指定できる．プロセスはプロセス・グループを生成，削除したり，参加，離脱することができる．プロセス・グループに対する Send はそのときプロセス・グループに参加しているプロセスにマルチキャストされ，最初の Reply メッセージを受け取ると待ちを解除される．2 個目以降の Reply は GetReply で受け取る．

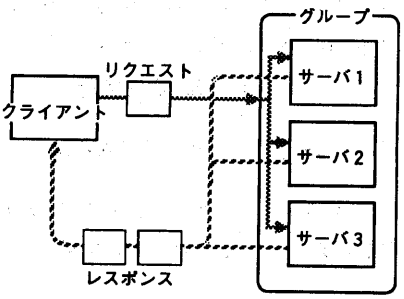

図 3: マルチキャスト

プロセス・グループに対するマルチキャストの典型的な使用例は名前管理である．V では専用のネームサーバを持たず，資源の各サーバが分散して名前を管理する．文字列により名前を管理する全サーバは，CSNH(Character String Name Handling) サーバグループと言う静的に定義されたプロセス・グループに属している．クライアントが文字列名を CSNH サーバグループにマルチキャストすると，その名前に対応する資源を管理しているサーバが Reply することで，サーバが特定される．指定した文字列名が存在しないことは，マルチキャストが (何回かのリトライの後で) タイムアウトすることで知ることができる．

2.7 カーネルサーバ

分散カーネルは IPC 以外にプロセス管理，メモリ管理，デバイスドライバなどの機能を提供する．これらは Send で要求を出すサーバの形で実現されている．プロセスのスケジュリングはプライオリティ・ベースの単純なもので，タイムスライスなどはカーネル外のスケジュラ・プロセスが実現する．マルチプロセッサに対してはプロセッサ毎にレディキューをもつスタティックバインドを採用している．

V のプロセスはライト・ウェイト・プロセスであり，複数のプロセスが 1 仮想空間を共有できる．空間を共有するプロセスをチームと呼ぶ．通常チームはプログラムに対応している．

3 サーバ

ファイルや認証管理などの機能はプロセスレベルのサーバとして実現されている．サーバ実現上共通性のある I/O とネーミングは各サーバに共通のプロトコルが規定されており，サーバの扱いを統一化すると共に，プロトコルを扱うサーバ側ライブラリを提供することでサーバの実現を容易にしている．ここでは I/O とネーミングのプロトコルの特徴を簡単に紹介し，サーバの例としてチームサーバ，Internet サーバ，V サーバについて述べる．

3.1 I/O プロトコル

V システムでは UIO と呼ばれる I/O プロトコルが規定されている．詳細は [2] を参照していただくとして，ここでは 2 ～ 3 の特徴を述べる．

アプリケーションに対しては getc, putc, fseek などの UNIX ライクなインタフェースが提供されている．このアプリケーション・インタフェースはクライアント側ライブラリによってクライアント - サーバ間プロトコルに対応付けられる．ライブラリ内ではキャッシュによってサーバへの通信回数を減らし，効率をあげている．また，ファイル中のリード / ライト・ポインタをクライアント側ライブラリで管理している．これによりリードはサー

バの状態を変えない(リードポインタが進まない) Idempotent なものになる. Idempotent な通信に対しては VMTP がリプライを再送用に保存しないという最適化を行なうため, 効率のよいI/O が実現される. このようにできる限り多くの処理をクライアント側ライブラリで行なうことにより, アプリケーション・プログラマには効率化の詳細を意識させずに, 共有資源であるサーバへの負荷の集中を避けている.

3.2 ネーミングプロトコル

V システムのネーミングは資源を管理する各サーバが資源の名前も管理する分散管理方式である [7]. これはネームサーバによる集中管理よりも効率的であり耐故障性も高い. 2.6で述べたように, 資源を文字列名で管理する各サーバは CSNH サーバグループに属している. クライアントは文字列名を CSNH サーバグループにマルチキャストすることでサーバを特定するが, 通信の回数を減らすために名前のプレフィクスとサーバの対応のキャッシュを持っている. サーバがクラッシュして再起動された場合にはサーバの pid が変わりキャッシュの一貫性が崩れる. これに対しては, とりあえずキャッシュの値を使用して通信してみて, 値が古くなっていたらマルチキャストして正しいサーバを探すと同時にキャッシュを更新する. このような考え方を問題指向共有メモリ [1] と呼んでいる.

ネームスペースはシステムグローバルな木構造になっている.

3.3 チームサーバ

チームサーバはチーム(プログラム)のロード, 実行, モニタなどの制御を行なう. チームサーバには次のような機能がある.

- チームのロード, プライオリティの設定
- チームの終了
- チームのラウンド・ロビン・スケジュリング
- デフォルト例外ハンドラの提供. (実際には例外を起こしたプロセスに対しデバッガを起動する.)
- プロセスのリモート実行の際のホスト選択のための負荷情報の管理

チームのロード要求はリモートなチームサーバに対してもローカルな場合と同様に行なえる. これによりネットワーク透過なプログラムのリモート起動が実現される. チームロードのクライアント側ライブラリでは, その時点での各ホストの負荷をチームサーバに問い合わせ, もっとも負荷の低いサーバにチームをロードするという機能が実現されている.

3.4 Internet サーバ

V システムはホスト間プロトコルとして VMTP を採用している. Internet サーバは VMTP が実装されていない(V 以外の OS が動いている) ホストとの通信を行なうサーバである. このサーバは

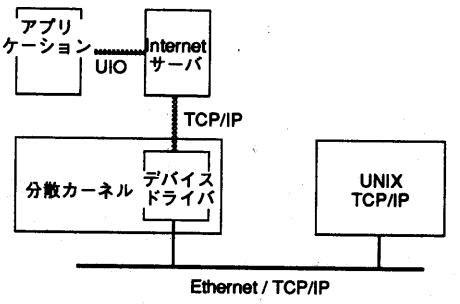

図 4: Internet サーバ

基本的にはプロトコル変換機であり, V のアプリケーションからは V の I/O プロトコルでアクセスできる. Internet サーバはこれを指定されたプロトコルに変換し, ネットワーク・デバイスと直接入出力することで, VMTP の実装されていないホストと通信する. 現在プロトコルとしては TCP/IP がサポートされているが, Internet サーバの構造はプロトコル独立なフレームワークとプロトコル

依存部分とが分離されており,新たなプロトコルの追加が容易になるように考慮されている.

3.5 Vサーバ

Vサーバは他のOS(現在はUNIXのみサポートされている)のファイルシステムやプロセスを,Vのファイルやプロセスのように扱うためのサーバである.Vシステムには,VのI/Oプロトコルでアクセスする独自のファイルサーバがある.システム中には複数のファイルサーバが存在できるようになっている.Vサーバは他のOS上で動作するサーバで,Vのネーミング・プロトコルとI/Oプロトコルを認識し,それを他のOSのファイルI/Oに変換する.これにより,Vのアプリケーションからは Vのファイルサーバをアクセスしているのと同一のプロトコルで実際には他のOSのファイルがアクセスできる.Vのプロセッサ間プロトコルはVMTPなので,Vサーバが動作するマシンはVMTPが実装されている必要がある.

また,Vのプロセスとの間にパイプを作り,他のOSのプログラムを他のOS上のプロセスとして実行することもできる.

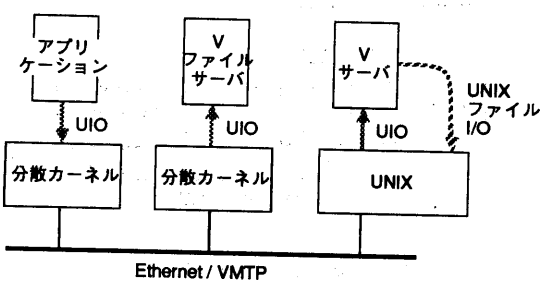

図 5: Vサーバ

3.6 多重化ファイルサーバ

著者のVプロジェクトでの研究テーマであった多重化ファイルサーバについて簡単に述べる.

ネットワーク上でファイルサーバを共有し,ファイルが数ヶ所のホストに集中すると,ファイルサーバの障害がシステム全体に及ぼす被害は大きくなる.そこで,ファイルサーバの耐故障性が重要な課題になる.

分散システムでは,同じファイルを持つファイルサーバを異なるホストで複数動作させ,任意のサーバからリードすることで,耐故障性が得られる.分散システムでは一般的に,故障に対するホストどとの独立性が高いので,ディスクの二重化などに比べより高い耐故障性が実現できる可能性がある.

多重化サーバのデータを同時に更新するために,UIOのリード/ライト・プロトコルを“多重系全サーバへライト/任意サーバからリード”に拡張した.多重系を構成するサーバは同じプロセスグループに属するようにして,クライアントがそのグループにライト要求をマルチキャストすることで,全サーバライトは効率よく実現できる.リードはオープン時にグループに名前の参照要求をマルチキャストし,最初にリプライを返したサーバのファイルをオープンすることで任意サーバを選択した.これはオープン時に最も負荷の低いサーバが選ばれることになり,リード処理の負荷を動的にバランスさせるという副作用も持っている.リードエラーの場合,再度別のサーバをオープンし,同じブロックを再リードすることで,処理を続行する.この時,リード操作のIdempotencyがファイルサーバにまたがって維持されるため,サーバの切り替えは意識されない.以上のようなI/Oプロトコルの拡張は全てライブラリ内に閉じており,アプリケーションプログラムには意識させずにファイルの多重化が実現できた.

拡張したプロトコルのオーバーヘッドをプロトタイプを作成して評価したところ,多重度1～3の範囲ではリード/ライト/リードオープン処理にはほとんどオーバーヘッドがないことがわかった.ライトオープンでは多重度2で2.6倍,3で2.8倍程度の時間がかかった.イーサネットでのマルチキャストが物理的にも1パケットで実現されていることが,低いオーバーヘッドに寄与していると思われた.

耐故障サーバの実現のためには, 復旧したサーバのデータの同期, ネットワークのパーティションへの対応などの問題が残されている.

4 ユーザ環境

Vシステムは全体としてはワークステーション群を一つのTSSマシンに見せる分散OSと見ることができる. ブート直後やユーザが使用していない状態のワークステーションでは, 分散カーネル上にチームサーバ, 例外サーバ, execサーバ, ウィンドウ管理サーバが動いている [6].

4.1 Execサーバ

ExecサーバはUNIXのshell風のコマンド・インタプリタで, ヒストリ, alias, 入出力のリダイレクト, パイプなどの機能やコマンドの1ラインエディットの機能がある.

コマンドのサーチ順序は環境変数PATHで指定できる. コマンドがこのサーチパスにない場合には, カレント・ディレクトリを提供するサーバでリモート実行される. カレント・ディレクトリがVサーバが提供するUNIXファイルの場合, まずVのコマンドが探され, 次にUNIXコマンドが探されリモート実行される. つまりユーザから見るとVのコマンドもUNIXコマンドも同様に利用できることになる. 最近ではV上のUNIXバイナリのシミュレータが開発され, UNIXコマンドはV上で直接動作するようになったようである [8].

通常, コマンドはログインしたホストで実行されるが, コマンドライン中にホストを指定することで, コマンドのリモート実行が行なえる. さらに, ホスト名としてanyを指定すると, その時点で誰もログインしていなくて, メモリとCPUの使用量の一番低いものがホストとして選択され, リモート実行される.

現在Vシステムは Sun2, Sun3, μVAX, DEC3100などのアーキテクチャをサポートしている. Vコマンドは CPU毎に, Sunなら.m68k, VAXは.vaxといった別のサフィックスが付けられている. Execサーバはコマンドを実行するホストによって自動的にサフィックスを選択するので, ユーザはホストのCPUタイプを意識する必要がない. もちろんanyを指定して負荷の軽い任意のホストを選ぶ時にも自動的に適切なバイナリが選択される.

4.2 コマンド

現在Vシステムでは200個程度のコマンドが提供されている. このうち約100個はUNIXから移植されたもので, cc, CC, emacs, mail, rcs, TEX, LATEX などが含まれる. VサーバによってUNIXをVのファイルサーバとして使っている場合, UNIXコマンドもリモート実行して利用できるが, Vコマンドに移植することで, コマンドがVのワークステーションでローカルに実行でき, UNIXマシンの負荷が分散できるため, 使用頻度の高いものは積極的にVに移植されている. 分散機能を生かしたコマンドとしてはパラレルmakeや実験的なものとして並列smalltalk, 並列シミュレータなどが開発されていた.

4.3 OSの開発環境

Vシステムはファイル管理, プログラムロードなど通常のOS機能がプロセスレベルのサーバとして実現されている. このため, このようなモジュールの開発も, 対象のサーバをデバッガで対話的にデバッグしたり, システム全体を動作させたままで, 再コンパイルして置き換えたりが可能であり, システムプログラムの開発効率は高い. また, 分散カーネルにはデバッガが組み込まれており, カーネル内で例外が起こると自動的にこのデバッガが起動される.

Vプロジェクトでは, 開発環境として使用している安定したシステムと, 実験中の不安定なシステムとを同じネットワーク中に接続しているが, パケット中のVMTPのバージョン番号を変えることで不安定なシステムが開発環境に干渉を与えないようにしている.

5 おわりに

Vシステムは1986年にV6.0がリリースされた後も，積極的な拡張，書き直しがされている．現在ではV6.0とはかなり機能や内部構造が変化しているが，現在のところ次版のリリースの予定は定かではない．

また，ここに触れなかった話題として

- 共有メモリマルチプロセッサへの対応 [5].
- ファイル・キャッシュ・プロトコル [11].
- 分散アトミック・トランザクション・プロトコル.

などがある．いずれも研究開発中またはある程度成果がまとまった段階である．

参考文献

[1] David R. Cheriton. Problem-oriented shared memory: A decentralized approach to distributed system design. In *the 6th International conference on Distributed Computer Systems*, pages 190–197. IEEE Computer Society, May 1986.

[2] David R. Cheriton. UIO: A uniform I/O interface for distributed systems. *ACM Transactions on Computer Systems*, 5(1):12–46, February 1987.

[3] David R. Cheriton. The V distributed system. *Communications of the ACM*, 31(3):314–333, March 1988.

[4] David R. Cheriton. Versatile message transaction protocol (vmtp). Technical Report RFC 1045, SRI Network Information Center, February 1988.

[5] David R. Cheriton, Hendrik A. Goosen, and Patrick D. Boyle. Multi-level shared caching techniques for scalability in vmp-mc. In *the 16th Annual International Symposium on Computer Architecture*, May 1989.

[6] David R. Cheriton and Keith A. Lantz. *V-System 6.0 Reference Manual.* Stanford University, July 1986.

[7] David R. Cheriton and Timothy P. Mann. Decentralizing a global naming service for improved performance and fault tolerance. *ACM Transactions on Computer Systems*, 7(2):147–183, May 1989.

[8] David R. Cheriton, Gregory R. Whitehead, and Edward W. Sznyter. Binary emulation of Unix using the V kernel. In *USENIX Summer Conference*, pages 73–85. USENIX, June 1990.

[9] David R. Cheriton and Carey L. Williamson. Network measurement of the VMTP request-response protocol in the V distributed system. In *SIGMETRICS 87*. ACM, 1987.

[10] David R. Cheriton and Carey L. Williamson. VMTP as the transport layer for high-performance distributed systems. *IEEE Communications*, pages 37–44, June 1989.

[11] Cary G. Gray and David R. Cheriton. Leases: An efficient fault-tolerant mechanism for diftributed file cache consistency. In *the 12th ACM Symposium on Operating System Principles*, pages 202–210. ACM SIGOPS, December 1989.