

# TruCluster Server

---

## クラスタ概要

Part Number: AA-RM87D-TE

**2002 年 11 月**

ソフトウェア・バージョン: TruCluster Server バージョン 5.1B

オペレーティング・システム: Tru64 UNIX バージョン 5.1B

本書は、TruCluster Server のユーザと管理者を対象にしてその機能をまとめたものです。本書では、TruCluster Server バージョン 5.1B の構成要素および機能を説明するとともに、バージョン 5.1A 以降の新機能や変更点についても説明しています。

---

© 2002 Hewlett-Packard Company

本書の著作権は日本ヒューレット・パッカート株式会社が保有しており、本書中の解説および図、表は日本ヒューレット・パッカートの文書による許可なしに、その全体または一部を、いかなる場合にも再版あるいは複製することを禁じます。

日本ヒューレット・パッカートは、弊社または弊社の指定する会社から納入された機器以外の機器で対象ソフトウェアを使用した場合、その性能あるいは信頼性について一切責任を負いかねます。

本書に記載されている事項は、予告なく変更されることがありますので、あらかじめご承知おきください。万一、本書の記述に誤りがあった場合でも、弊社は一切その責任を負いかねます。

本書で解説するソフトウェア(対象ソフトウェア)は、所定のライセンス契約が締結された場合に限り、その使用あるいは複製が許可されます。

COMPAQ, Compaq ロゴ, Digital ロゴは U.S. Patent and Trademark Office に登録されています。Alpha, AlphaServer, NonStop, TruCluster, および Tru64 は米国 Compaq Computer Corporation の商標です。

Microsoft, Windows および Windows NT は米国 Microsoft 社の登録商標です。Intel は米国 Intel 社の登録商標です。Motif, OSF/1, UNIX, The Open Group および X/Open は、The Open Group の米国ならびに他の国における商標です。

このドキュメントに記載されているその他の会社名および製品名は、各社の商標または登録商標です。

---

# 目次

## まえがき

## 1 TruCluster Server の概要

1.1	バージョン 5.1B で新しく追加された機能と変更された機能 .	1-1
1.2	TruCluster Server の機能 .....	1-3

## 2 クラスタ単位のファイル・システム , ストレージ , デバイス名

2.1	サポートされているファイル・システム .....	2-3
2.2	クラスタ・ファイル・システム .....	2-7
2.3	デバイス要求ディスパッチャ .....	2-11
2.4	CFS とデバイス要求ディスパッチャに関する FAQ .....	2-12
2.4.1	CFS , 入出力 , およびクラスタ・インターコネクト .....	2-13
2.4.2	AdvFS 要求ブロック・キャッシング .....	2-14
2.4.3	デバイス要求ディスパッチャとファイルのオープン .....	2-14
2.4.4	CFS サーバの再配置 .....	2-14
2.5	コンテキスト依存シンボリック・リンク .....	2-15
2.6	デバイス名 .....	2-18
2.7	ワールドワイド ID .....	2-21
2.8	クラスタと LSM .....	2-22

## 3 接続マネージャ

3.1	クォーラムとポート .....	3-2
3.1.1	クラスタのメンバとは .....	3-2
3.1.2	ノード・ポート .....	3-2
3.1.3	クォーラム・ディスク・ポート .....	3-3

3.1.4	期待ポート .....	3-3
3.1.5	現在のポート .....	3-4
3.2	クラスタ・クォーラムの計算 .....	3-5
3.3	クォーラム・ディスクの使用 .....	3-7
 <b>4 高可用性アプリケーション</b>		
 <b>5 CAA (Cluster Application Availability)</b>		
5.1	CAA の概要 .....	5-1
5.2	CAA のアーキテクチャ .....	5-3
5.3	リソース .....	5-7
5.4	リソース・プロファイル .....	5-9
5.4.1	アプリケーション・リソース・プロファイル .....	5-9
5.4.2	非アプリケーション・リソース・プロファイル .....	5-12
5.5	処理スクリプト .....	5-13
 <b>6 クラスタ別名</b>		
6.1	クラスタ別名の概要 .....	6-1
6.2	クラスタ別名の構成要素 .....	6-3
6.3	省略時のクラスタ別名 .....	6-5
6.4	クラスタごとの別名の数 .....	6-6
6.5	別名 IP アドレスの位置 .....	6-7
6.6	別名 IP アドレスへのルーティング .....	6-9
6.6.1	別名へのルートの公開 .....	6-10
6.6.2	共通サブネットでの別名のルーティング .....	6-10
6.6.3	仮想サブネットでの別名のルーティング .....	6-12
6.6.4	共通サブネットおよび仮想サブネット上の別名に対する ルーティングのまとめ .....	6-12
6.6.5	別名宛て接続要求およびパケットの受け付けとリダイレク ション .....	6-13

6.6.6	ルーティングの例 .....	6-16
6.7	in_single および in_multi サービス .....	6-18
6.8	別名の属性 .....	6-21
6.9	サービス・ポートの属性 .....	6-24
6.10	vMAC サポート .....	6-26
6.11	NFS とクラスタ別名 .....	6-27
6.11.1	ネットワークからのパケットの受信 .....	6-28
6.11.2	マウント要求 .....	6-28
6.11.3	NFS over TCP .....	6-28
6.11.4	NFS over UDP .....	6-30
6.12	RPC サービスとクラスタ別名 .....	6-32
6.13	ifconfig 別名とクラスタ別名 .....	6-33
<b>7</b>	<b>クラスタ・インターコネクト</b>	
7.1	クラスタ・インターコネクトを経由するストレージ・トラフィックの制御 .....	7-4
7.2	クラスタ・インターコネクトを経由するアプリケーション・トラフィックの制御 .....	7-5
7.3	クラスタ・インターコネクトを経由するクラスタ別名トラフィックの制御 .....	7-6
7.4	クラスタ・インターコネクトのトラフィックに与えるクラスタ規模の影響 .....	7-7
7.5	クラスタ・インターコネクトの選択 .....	7-8
7.6	Memory Channel インターコネクト .....	7-9
7.7	LAN インターコネクト .....	7-12
<b>8</b>	<b>分散ロック・マネージャ</b>	
<b>9</b>	<b>クラスタのインストレーションと管理</b>	
9.1	インストレーション .....	9-1

9.2	管理 .....	9-3
用語集		
索引		
図		
2-1	クラスタでのストレージ・ソフトウェア階層 .....	2-2
2-2	クラスタのハードウェア・ビュー .....	2-6
2-3	すべてのクラスタ・メンバからファイル・システムを使用できるようにする CFS .....	2-9
2-4	CDSL のパス名解決 .....	2-17
3-1	クォーラム・ディスクのない, 2 メンバの deli クラスタ .....	3-8
3-2	メンバが 1 つ失われても持ちこたえる, クォーラム・ディスクがある 2 メンバの deli クラスタ .....	3-10
5-1	CAA によるアプリケーションのフェイルオーバー .....	5-3
5-2	CAA のアーキテクチャ .....	5-7
6-1	クラスタ別名がある場合とない場合の, クライアントからのクラスタの見え方 .....	6-2
6-2	クラスタ別名の機能概要 .....	6-5
6-3	2 つの別名を使用するクラスタ .....	6-6
6-4	別名ルーティングの例 .....	6-17
6-5	省略時のクラスタ別名を通してアクセスされる in_single サービス .....	6-20
6-6	省略時のクラスタ別名を通してアクセスされる in_multi サービス .....	6-21
6-7	NFS over TCP .....	6-29
6-8	NFS over UDP .....	6-31
7-1	Memory Channel の論理図 .....	7-10

## 表

1-1	TruCluster Server バージョン 5.1B 製品の機能 .....	1-4
2-1	クラスタでサポートされているファイル・システム .....	2-3
2-2	新しいデバイス名の例 .....	2-19
6-1	共通サブネットおよび仮想サブネット上の別名に対するルーティングのまとめ .....	6-13
7-1	Memory Channel インターコネクトと LAN インターコネクトの特性比較 .....	7-3





---

## まえがき

本書では、HP TruCluster Server バージョン 5.1B 製品で利用できる機能について、その概要を説明します。

本書の内容は、TruCluster Server の『*QuickSpec*』に代わるものではありません。『*QuickSpec*』は、本製品の正式な説明書です。製品の最新情報については、『*QuickSpec*』を参照してください。

### 対象読者

本書は、TruCluster Server の機能の概要に関心があるユーザや管理者を対象としています。HP Tru64 UNIX オペレーティング・システムと TruCluster Server 製品の知識が必要です。

### 新しい機能と変更された機能

本書には、バージョン 5.1A のリリース以降に、次のような変更が行われています。

- バージョン 5.1A の『クラスタ LAN インターコネクト』に記述されていた必要な情報は、本書、『クラスタ・ハードウェア構成ガイド』、『クラスタ・インストレーション・ガイド』、および『クラスタ管理ガイド』のそれぞれのマニュアルに統合されました。このため、『クラスタ LAN インターコネクト』は TruCluster Server マニュアル・セットの一部ではなくなりました。
- 1.1 節では、TruCluster Server バージョン 5.1B 製品で利用できる新機能について説明しています。
- 第 5 章では、このリリースでの CAA (Cluster Application Availability) の新機能の概要を説明しています。この概要では、CAA によりアプリケーション・リソースを分散する方法、ユーザ定義属性を使用してアプリケーション・プロファイルを拡張する方法などについて説明します。
- 第 6 章では、制限を緩和した `gated` コマンドについて説明しています。制限を緩和したことにより、クラスタ・メンバは `gated` コマンド

や `routed` コマンドの実行，または静的ルーティングの使用ができるようになりました。

- 第 7 章の内容は大幅に書き換えられて，バージョン 5.1A の『クラスタ LAN インターコネクト』に記述されていたクラスタ・インターコネクトの選択に関する情報が追加されました。
- 第 9 章では，TruCluster Server バージョン 5.1A からローリング・アップグレードを行うとき，複数のクラスタ・メンバのローリングを並列に行う方法を説明しています。

## 本書の構成

本書は，以下のように構成されています。

- |       |  |
|-------|--|
| 第 1 章 | TruCluster Server 機能の概要を説明します。   |
| 第 2 章 | サポートしているファイル・システム，クラスタ・ファイル・システム (CFS)，コンテキスト依存シンボリック・リンク (CDSL)，ストレージ，デバイス命名規則について説明します。                            |
| 第 3 章 | 接続マネージャの概要と，クォーラムとポートによるクラスタの形成と保守における，その役割について説明します。  |
| 第 4 章 | クラスタ内の高可用性アプリケーションの基本的なタイプを 3 つ (シングル・インスタンス，マルチ・インスタンス，分散型) 定義します。  |
| 第 5 章 | シングル・インスタンス・アプリケーションをクラスタ単位で管理する CAA (Cluster Application Availability) サブシステムの概要を説明します。                             |
| 第 6 章 | TCP (Transmission Control Protocol) および UDP (User Datagram Protocol) に対してクラスタをシングル・システムのように見せるクラスタ別名サブシステムの概要を説明します。 |
| 第 7 章 | LAN と Memory Channel のクラスタ・インターコネクトについて概要を説明します。   |
| 第 8 章 | 分散ロック・マネージャ (DLM) について説明します。DLM には，クラスタ内で連携するプロセスが共用リソースへのアクセスの同期をとるための機能があります。                                      |
| 第 9 章 | クラスタのインストレーションおよび管理の概要を説明するとともに，ローリング・アップグレードについても説明します。   |
| 用語集   | TruCluster Server のドキュメント全体を通して使用される共通の用語を定義します。   |

## 関連資料

次のドキュメントおよびマニュアルでは、TruCluster Server 製品の詳細情報が説明されています。

- TruCluster Server 『*QuickSpec*』 — TruCluster Server バージョン 5.1B 製品の正式な説明書です。『*QuickSpec*』の最新版は、次の URL から入手できます。

[http://www.tru64unix.compaq.com/docs/pub\\_page/spds.html](http://www.tru64unix.compaq.com/docs/pub_page/spds.html)

- 『クラスタ・リリース・ノート』 — TruCluster Server の新機能の概要を紹介するとともに、既知の問題とその回避策について説明しています。
- 『クラスタ・ハードウェア構成ガイド』 — クラスタ・メンバのシステムの設定方法と、クラスタ共用ストレージの構成方法について説明しています。
- 『クラスタ・インストレーション・ガイド』 — TruCluster Server ソフトウェアのインストール方法について説明しています。
- 『クラスタ高可用性アプリケーション・ガイド』 — 既存のアプリケーションを TruCluster Server クラスタで実行する方法と、クラスタ・ベースのアプリケーションの作成方法について説明しています。
- 『クラスタ管理ガイド』 — クラスタ特有の管理作業について説明しています。

TruCluster Server の最新ドキュメントは、次の URL から入手することができます。

[http://www.tru64unix.compaq.com/docs/pub\\_page/cluster\\_list.html](http://www.tru64unix.compaq.com/docs/pub_page/cluster_list.html)

## 本書で使用する表記法

本書では、次の表記法を使用します。

#	番号記号は root としてログインした場合のシステム・プロンプトを表します。
% <b>cat</b>	対話式の例における太字(ボールド体)は、ユーザが入力する文字を示します。
<i>file</i>	イタリック体(斜体)は、変数値、プレースホルダ、および関数の引数名を示します。
cat(1)	リファレンス・ページの参照には、該当するセクション番号をカッコ内に示します。たとえば、cat(1) は、cat コマンドについての情報が、リファレンス・ページのセクション 1 に記載されていることを示します。

## TruCluster Server の概要

TruCluster Server バージョン 5.1B を使用すると、HP Tru64 UNIX オペレーティング・システム・ソフトウェア、複数の AlphaServer システムおよびストレージ・デバイスを高度に統合して、単一の仮想システムとして動作させることができます。クラスタのメンバは、リソース、データ・ストレージ、クラスタ単位のファイル・システムを単一のセキュリティと管理ドメイン下で共用できるだけでなく、クライアントに対するクラスタのサービスを停止することなく個別にブートまたはシャットダウンできます。

TruCluster Server 環境は、必要に応じて、簡単なものにも、機能が豊富なものにもすることができます。必要に合わせて、2 ~ 8 ノードのクラスタを構成できます。このクラスタでは、トランザクション処理システム、ネットワーク・クライアント/サーバ・アプリケーションのサーバ、高可用性を必要とするデータ共有アプリケーション、TruCluster Server のアプリケーション・プログラミング・インタフェース (API) の利点をフルに活用する分散型並列処理アプリケーションなどの高可用性アプリケーションが実行できます。

TruCluster Server にはインターネット・プロトコル群 (TCP/IP) のクラスタ別名が含まれているため、クラスタはネットワーク・クライアントおよびピア・システムからは単一のシステムとして認識されます。

この章では、次の事項について説明します。

- 本リリースで新しく追加された機能と変更された機能 (1.1 節)
- TruCluster Server の機能を包括的に説明する表 (1.2 節)

### 1.1 バージョン 5.1B で新しく追加された機能と変更された機能

以下は、TruCluster Server バージョン 5.1B で新しく追加された機能と変更された機能です。各項目には、その機能に関して説明している TruCluster Server のマニュアル、本書中の節、またはリファレンス・ページも合わせて記載しています。

- クラスタ・ファイル・システム (CFS) 負荷分散 — CFS 負荷分散デーモン (cfstd) は、ファイル・システムの使用量を監視し分析します。メンバがクラスタに参加した場合やクラスタへの参加を中止した場合、ストレージとの接続が変更された場合、または CFS メモリの使用量に対応して、ファイル・システムを自動的に再配置するように cfstd を設定することができます。CFS 負荷分散についての詳細は、『クラスタ管理ガイド』、cfstd.conf(4)、および cfstd(8) を参照してください。
- mount -o コマンドによるファイル・システムの分散 — mount コマンドの server=name オプションを使用すると、起動時にどのクラスタ・メンバにどのファイル・システムを割り当てるかを指定できます。詳細は、『クラスタ管理ガイド』、mount(2)、および mount(8) を参照してください。
- AdvFS (Advanced File System) ファイル・システムまたはドメインの強制アンマウント — TruCluster Server バージョン 5.1B には、クラスタ・メンバが使用している AdvFS ドメイン全体を強制的にアンマウントできる、cfsmgr -U コマンドが用意されています。詳細は、『クラスタ管理ガイド』および cfsmgr(8) を参照してください。
- CFS の停止 — Tru64 UNIX バージョン 5.1B には、メタデータの一貫性を保ったまま AdvFS ドメインをフリーズ状態にする、freezefs コマンドが用意されています。詳細は、『クラスタ管理ガイド』、freezefs(8)、および thawfs(8) を参照してください。
- 並列ローリング・アップグレード — この機能を使用すると、一度に複数のクラスタ・メンバをアップグレードすることができるため、クラスタのアップグレードに要する時間を短くできます。並列ローリング・アップグレードについての詳細は、9.1 節、『クラスタ・インストレーション・ガイド』、および clu\_upgrade(8) を参照してください。
- gated に関する制限の緩和 — TruCluster Server の以前のリリースでは、クラスタ・メンバは gated ルーティング・デーモンを起動している必要がありました。バージョン 5.1B ではその制限を緩和して、クラスタ・メンバで gated の実行、routed の実行、または静的ルーティングの使用ができるようになりました。gated に関する制限の緩和についての詳細は、『クラスタ管理ガイド』を参照してください。
- CAA アプリケーション・リソースの分散 — CAA によるアプリケーション・リソースの配置の再評価は、定期的にまたはいつでも必要な

ときに手動で行うことができます。この機能と新しい `caa_balance` コマンドについての詳細は、『クラスタ高可用性アプリケーション・ガイド』を参照してください。

- CAA ユーザ定義属性の作成 — アプリケーション・リソース・プロファイルの形式は、ユーザ定義属性で拡張できます。詳細は、『クラスタ高可用性アプリケーション・ガイド』を参照してください。
- CAA 処理スクリプトの出力先のリダイレクト — 処理スクリプトの出力をリダイレクトできるので、CAA コマンドがいつ実行されたかを表示することができます。詳細は、『クラスタ高可用性アプリケーション・ガイド』を参照してください。
- 可用性の測定 — すべてのユーザがアプリケーション・リソースの可用性の測定を指定および表示できます。この機能と新しい `caa_report` コマンドについての詳細は、『クラスタ管理ガイド』を参照してください。
- `esmd` による CAA の監視 — Essential Services Monitor デーモン (`esmd`) は、必要に応じて `caad` デーモンを自動的に再起動します。詳細は、『クラスタ管理ガイド』を参照してください。
- インターネット・プロトコル・バージョン 6 (IPv6) アドレスの使用 — クラスタ・メンバは IPv6 アドレスを使用し公開することができます。ただし、クラスタ内で使用する IPv6 アドレスには次の制限があります。
  - IPv6 アドレスはクラスタ別名に関連付けることはできません。
  - IPv6 アドレスはクラスタ・インターコネクト・アドレスには使用できません。`clu_create` と `clu_add_member` は、クラスタ・インターコネクト・アドレスとしては IPv6 アドレスを受け付けません。

## 1.2 TruCluster Server の機能

TruCluster Server バージョン 5.1B の機能を表 1-1 に示します。

表 1-1: TruCluster Server バージョン 5.1B 製品の機能

機能	説明
クラスタ単位のネームスペース	<p>クラスタ・ファイル・システム (CFS) は、クラスタ単位の単一のネームスペースをサポートし、統一された方法でクラスタ内の全ファイル・システムへアクセスできるようにする。システムごとの構成と、共用 CFS ルート (/), /usr, /var ファイル・システム内のデータ・ファイルの保持には、コンテキスト依存シンボリック・リンク (CDSL) が使用される。</p> <p>CFS についての詳細は、2.2 節を参照。CDSL についての詳細は、2.5 節を参照。</p>
ディスクおよびテープ・ストレージへのクラスタ単位のアクセス	<p>デバイス要求ディスパッチャ機能により、テープ・デバイス、文字型ディスク・デバイス、およびブロック型ディスク・デバイスに対して、可用性の高い、クラスタ単位のアクセスが可能。クラスタのディスク入出力およびテープ入出力は、すべてデバイス要求ディスパッチャを通して渡される。</p> <p>デバイス要求ディスパッチャについての詳細は、2.3 節を参照。</p>
クラスタ単位の LSM (Logical Storage Manager)	<p>LSM のセマンティクスが、クラスタ環境に拡張されている。</p> <p>クラスタ環境での LSM についての詳細は、2.8 節を参照。</p>
接続マネージャ	<p>接続マネージャは、すべてのクラスタ・メンバ間の通信を確保し、クラスタの形成と継続されている操作を制御する。接続マネージャはクォーラムに必要なポートを計算し、メンバをクラスタへ追加する時期やクラスタから削除する時期を決定する。</p> <p>接続マネージャについての詳細は、第 3 章を参照。</p>
CAA (Cluster Application Availability)	<p>CAA 機能により、リソースのモニタリングとアプリケーションの再起動が可能となる。この機能では、TruCluster Available Server Software および TruCluster Production Server Software 製品のユーザ定義サービスで提供される可用性と同様のアプリケーションの可用性を実現できる。</p> <p>クラスタ内で実行できるアプリケーションのタイプの定義については、第 4 章を参照。シングル・インスタンス・アプリケーションの可用性を高めるための CAA の役割については、第 5 章を参照。</p>



表 1-1: TruCluster Server バージョン 5.1B 製品の機能 (続き)

機能	説明
クラスタ別名	<p>クラスタ別名サブシステムを使用すると、TCP アプリケーションおよび UDP アプリケーションで、クラスタを単一のシステムとしてアドレス指定できるようになる。クラスタを作成すると、すべてのクラスタ・メンバを宛先とする、省略時のクラスタ別名が定義される。クラスタ・メンバの一部またはすべてを宛先とする別名を、サイトで追加定義することもできる。</p> <p>クラスタ別名についての詳細は、第 6 章を参照。</p>
クラスタ別名を使用した、高可用性 NFS (ネットワーク・ファイル・システム) サーバ	<p>出荷時には、クラスタは高可用性 NFS (ネットワーク・ファイル・システム) サーバである。CFS を使用することにより、TruCluster Server クラスタがエクスポートするファイル・システムは、クライアントに対する可用性が高くなる。省略時の設定では、クライアントは、クラスタがエクスポートしたファイル・システムをマウントする際に、省略時のクラスタ別名を NFS サーバ名として使用する。ただし、クライアントは、<code>/etc/exports.aliases</code> ファイルに定義されている別名を使用することもできる。</p> <p>詳細については、6.11 節を参照。</p>
クラスタ別名を使用した、高可用性インターネット・サービス	<p>クラスタは、<code>telnet</code> や <code>ftp</code> など多くのインターネット機能を高可用性サービスとして、出荷時からサポートしている。これらのサービスは、<code>/etc/clua_services</code> ファイルに <code>in_multi</code> サービスとして指定されており、これらサービスに送られてくるパケットや要求は、クラスタ別名サブシステムが、そのときに対応可能なクラスタ・メンバへ振り分ける。こうした処理はクラスタ別名サブシステムが行うように設計されており、サービス側は何もする必要がない。</p>

表 1-1: TruCluster Server バージョン 5.1B 製品の機能 (続き)

機能	説明
マルチ・クラスタ・インターコネクト	<p>TruCluster Server バージョン 5.1B ではクラスタのインターコネクトに Memory Channel またはローカル・エリア・ネットワーク (LAN) ハードウェアを使用することができる。</p> <p>Memory Channel インターコネクトは、クラスタの要件に合うように設計された高速のインターコネクトである。</p> <p>TruCluster Server には、Memory Channel アプリケーション・プログラミング・インタフェース (API) ライブラリがある。これは、TruCluster Production Server Software 製品で提供されている API と同じである。</p> <p>LAN クラスタ・インターコネクトは、新しいクラスタを構成したり既存の TruCluster ASE クラスタをアップグレードする際に、Memory Channel の代わりとして使用できる。</p> <p>LAN および Memory Channel のインターコネクトについての詳細は、第 7 章を参照。Memory Channel API についての説明は、『クラスタ高可用性アプリケーション・ガイド』を参照。</p>
分散ロック・マネージャ (DLM)	<p>TruCluster Server は、DLM とその API をサポートしている。この API は、TruCluster Production Server Software 製品で提供されている API と同じである。</p> <p>DLM についての説明は、第 8 章を参照。DLM API についての説明は、『クラスタ高可用性アプリケーション・ガイド』を参照。</p>
単一のシステム管理	<p>クラスタでは CFS を使用しているため、すべてのメンバ・システムの構成ファイルを管理することができる。SysMan のグラフィカル管理ユーティリティ群によってクラスタ環境を統合して表示することができ、単一のメンバを管理することも、クラスタ全体を管理することもできる。</p> <p>クラスタのインストレーションおよび管理の概要は、第 9 章を参照。</p>
ローリング・アップグレード	<p>TruCluster Server バージョン 5.1B は、ローリング・アップグレードをサポートしている。ユーザは、クラスタの TruCluster Server バージョン 5.1A から バージョン 5.1B へのローリング・アップグレードを行うことができる。詳細は、第 9 章を参照。</p>

表 1-1: TruCluster Server バージョン 5.1B 製品の機能 (続き)

機能	説明
ファイル・システムのパーティショニング	<p>CFS では、AdvFS ファイル・システムをマウントして単一のクラスタ・メンバだけがアクセスできるようにすることができ。この機能を、ファイル・システムのパーティショニングと呼ぶ。</p> <p>TruCluster Server にはファイル・システムのパーティショニング機能があり、TruCluster Production Server Software や TruCluster Available Server Software バージョン 1.5 またはバージョン 1.6 からの移行が容易である。ファイル・システムのパーティショニングは、ファイル・システムのアクセスを単一メンバに制限する汎用機能としては作られていない。</p> <p>詳細は、『クラスタ管理ガイド』および mount(8) を参照。</p>
単一のセキュリティ・ドメイン	<p>クラスタは CFS を使用しているため、/etc/passwd や /etc/group などのセキュリティ管理ファイルは 1 つだけ存在する。1 つのメンバで認証されたユーザは、すべてのメンバにアクセスできる。また、1 つのメンバ上の特定のファイルにアクセス可能なユーザは、どのメンバからでもそのファイルにアクセスできる。アクセス制御リスト (ACL) は、すべてのメンバから同じように使用できる。</p> <p>詳細は、Tru64 UNIX 『セキュリティ管理ガイド』を参照。</p>
拡張プロセス ID (PID)	<p>PID は、32 ビットをすべて使用する値まで拡張される。また PID は、クラスタ全体で一意である。各クラスタ・メンバには、PID として割り当てられる番号の範囲がある。</p>



---

## クラスタ単位のファイル・システム，ストレージ，デバイス名

構成および管理の観点では，TruCluster Server の最も重要な機能は，ファイルおよびディレクトリに対するネームスペースをクラスタ単位に 1 つだけ作成することです。このネームスペースにより，すべてのファイル・システムが，どのクラスタ・メンバからも同じように見えます。また，ほとんどの構成ファイルは 1 つだけ存在します。一部の例外を除き，クラスタのディレクトリ構造は，スタンドアロン・システムの場合と同じです。

クラスタ単位のネームスペースは，クラスタ・ファイル・システム (CFS) やデバイス要求ディスパッチャなどの，TruCluster Server テクノロジーによって実現されています。CFS やデバイス要求ディスパッチャについては，この章で後述します。

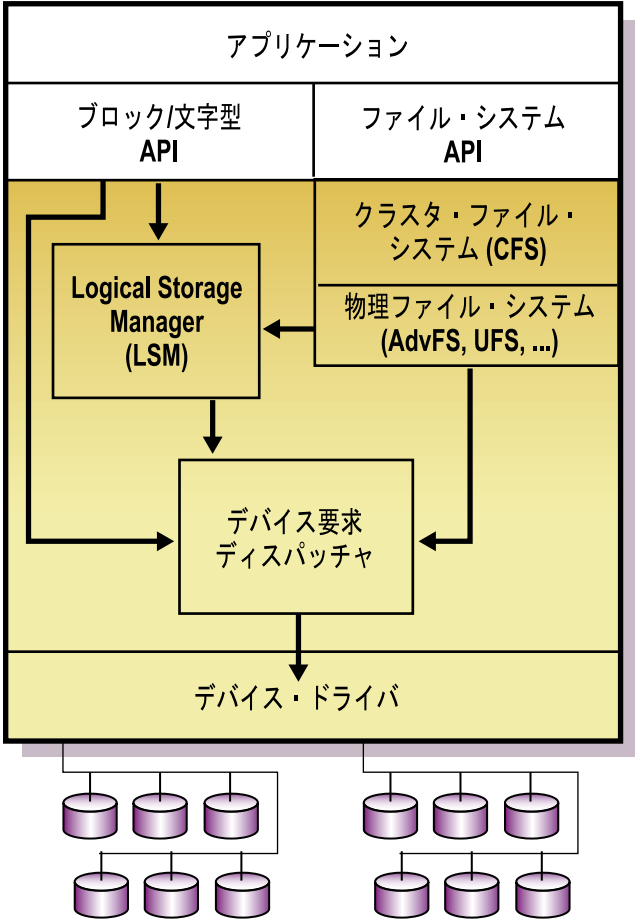
この章では，次の事項について説明します。

- サポートされているファイル・システム (2.1 節)
- クラスタ・ファイル・システム (CFS) (2.2 節)
- デバイス要求ディスパッチャ (2.3 節)
- CFS とデバイス要求ディスパッチャについての FAQ (2.4 節)
- コンテキスト依存シンボリック・リンク (CDSL) (2.5 節)
- デバイス名 (2.6 節)
- ワールドワイド ID (2.7 節)
- クラスタと LSM (Logical Storage Manager) (2.8 節)

クラスタ内でストレージ・ソフトウェアがどのように動作するかを理解するには，まず，図 2-1 を参照してください。この図は，クラスタ内で階層を形成しているストレージ・ソフトウェアをわかりやすく示しています。デバイス要求ディスパッチャは，物理デバイスへのすべての入出力を制御しています。すべてのクラスタ入出力は，このサブシステムを通ります。ま

た，CFS は AdvFS (Advanced File System) などの既存のファイル・システムより上の層にあります。

図 2-1: クラスタでのストレージ・ソフトウェア階層



ZK-1547U-AI

## 2.1 サポートされているファイル・システム

表 2-1 に、サポートされているファイル・システムの要約を示します。

表 2-1: クラスタでサポートされているファイル・システム

タイプ	サポート方法	障害の特性
AdvFS (Advanced File System)	読み取り/書き込み	ファイル・ドメインのサービスは、そのファイル・システムの置かれているストレージに接続しているかしていないかを基にして選ばれるメンバが行う。メンバに障害が発生すると、CFS はそのドメインに対して新しいサーバを選択する。パスに障害が発生すると、CFS はそのストレージへの代替デバイス要求ディスクパッチャ・パスを使用する。
CD-ROM ファイル・システム (CDFS)	読み取り専用	デバイスに直接接続されたメンバが、CD-ROM デバイスに対する読み取り専用アクセスのサービスを行う。TruCluster Server は共用バス上の CD-ROM デバイスをサポートしていないため、デバイスが接続されているメンバに障害が発生すると、他のメンバがサービスを引き継いだ場合でも、CD-ROM デバイスはアクセス不能となる。障害が発生していたメンバがクラスタに復帰すると、そのデバイスが再度アクセス可能となる。
DVD-ROM ファイル・システム (DVDFS)	読み取り専用	デバイスに直接接続されたメンバが、DVD-ROM デバイスに対する読み取り専用アクセスのサービスを行う。TruCluster Server は共用バス上の DVD-ROM デバイスをサポートしていないため、デバイスが接続されているメンバに障害が発生すると、他のメンバがサービスを引き継いだ場合でも、DVD-ROM デバイスはアクセス不能となる。障害が発生していたメンバがクラスタに復帰すると、そのデバイスが再度アクセス可能となる。
FFM (File-on-File Mounting) ファイル・システム	読み取り/書き込み (ローカル)	ローカル・メンバだけが、マウントおよびアクセス可能。

表 2-1: クラスタでサポートされているファイル・システム (続き)

タイプ	サポート方法	障害の特性
メモリ・ファイル・システム (MFS)	読み取り/書き込み (ローカル)	クラスタ・メンバは、MFS ファイル・システムを読み取り専用または読み取り/書き込みでマウントできる。ファイル・システムにはそのメンバだけがアクセス可能。リモート・アクセスは不可。フェイルオーバーはない。
名前付きパイプ	読み取り/書き込み (ローカル)	読み取り側と書き込み側は、同じメンバ上に存在しなければならない。
ネットワーク・ファイル・システム (NFS) サーバ	読み取り/書き込み	外部のクライアントは、クラスタによって NFS エクスポートされたファイル・システムをマウントする際に、省略時のクラスタ別名または <code>/etc/exports.aliaes</code> で指定されている別名をホスト名として使用する。ファイル・システムのフェイルオーバーおよび回復は、外部の NFS クライアントには見えない。
NFS クライアント	読み取り/書き込み	クラスタ・メンバは、クラスタ外の NFS ファイル・システムのマウントを行うことができる。クラスタ・メンバに障害が発生すると、そのファイル・システムは自動的にアンマウントされる。クラスタが <code>automount</code> または <code>autofs</code> を使用して動作している場合、ファイル・システムの再マウントは自動的に行われる。それ以外の場合は、ファイル・システムの再マウントは手動で行う必要がある。
PC-NFS サーバ	読み取り/書き込み	PC クライアントは、クラスタによって NFS エクスポートされたファイル・システムをマウントする際に、ホスト名として省略時のクラスタ別名または <code>/etc/exports.aliaes</code> で指定されている別名を使用する。ファイル・システムのフェイルオーバーと回復は、外部の NFS クライアントには見えない。



表 2-1: クラスタでサポートされているファイル・システム (続き)

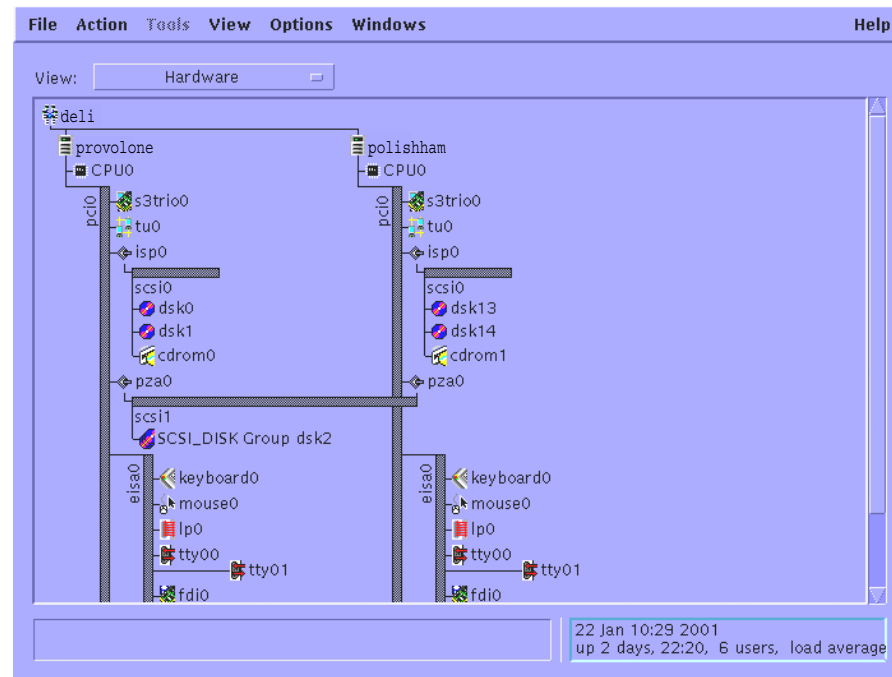
タイプ	サポート方法	障害の特性
/proc ファイル・システム	読み取り/書き込み (ローカル)	各クラスタ・メンバには、そのメンバ だけがアクセスできる固有の /proc ファイル・システムがある。
UNIX ファイル・シス テム (UFS)	読み取り専用 (クラスタ単位) 読み取り/書き込み (ローカル)	読み取り専用で明示的にマウントされ た UFS ファイル・システムは、その ファイル・システムの置かれているス トレージに接続しているメンバが選ば れてサービスし、クラスタ全体で読み 取り専用のアクセスを行うことができ る。メンバに障害が発生すると、CFS はそのファイル・システムに対して新 しいサーバを選択する。パスに障害が 発生すると、CFS はそのストレージへ の代替デバイス要求ディスパッチャ・ パスを使用する。  読み取り/書き込みのサポートは MFS の読み取り/書き込みサポートと同じ である。クラスタ・メンバは、UFS ファイル・システムを読み取り/書き 込みでマウントすることができる。そ のファイル・システムへはそのメンバ だけがアクセスできる (そのメンバだ けが読み取りと書き込みができる)。 リモート・アクセスは不可。フェイ ルオーバはない。

TruCluster Server では単一システムの管理機能をクラスタに拡張しているため、Tru64 UNIX システムの管理方法が理解できていれば、TruCluster Server クラスタの管理方法も理解できていることになります。TruCluster Server には、クラスタの全メンバが共用できる単一のルート (/) ファイル・システムなど、ファイルおよびディレクトリ用のクラスタ単位のネームスペースがあります。同様に、ストレージ・デバイス用のクラスタ単位のネームスペースもあります。各ストレージ・デバイスには、クラスタ内で一意のデバイス名が 1 つあります。

SysMan のグラフィカル管理ユーティリティ群によってクラスタ環境を統合して表示することができ、個々のメンバを管理することやクラスタ

全体を管理することもできます。図 2-2 に、メンバが 2 つ (provolone と polishham) で名前が deli というクラスタの SysMan Station ハードウェア・ビューを示します。

図 2-2: クラスタのハードウェア・ビュー



ZK-1700U-AI

TruCluster Server は、Tru64 UNIX バージョン 4.0 シリーズのオペレーティング・システム向けの TruCluster 製品で提供されていた、次の可用性および性能に関する機能を備えています。

- TruCluster Server を使用すると、TruCluster Available Server Software および TruCluster Production Server 製品と同様に、クラスタのどのメンバからでもディスク・データにアクセスできる、可用性の高いサービスを実現できます。

Tru64 UNIX 上で実行されるアプリケーションはすべて、可用性の高いシングル・インスタンス・アプリケーションとしてクラスタ内で実行できます。必要なリソースや現在のメンバ自体が利用できなくなると、アプリケーションは自動的に他のクラスタ・メンバに再配置 (フェイルオーバー) されます。

- TruCluster Server を使用すると、TruCluster Production Server Software 製品と同様に、分散型アプリケーションの構成要素を並列に実行できます。これにより、クラスタ特有の同期メカニズムと性能の最適化という利点を生かしつつ、可用性を高めることもできます。

## 2.2 クラスタ・ファイル・システム

クラスタ・ファイル・システム (CFS) では、すべてのファイルがすべてのクラスタ・メンバから見えて、アクセスできます。どのクラスタ・メンバから同じように見えます。ファイルがすべてのクラスタ・メンバに接続されたデバイスに格納されているか、1つのメンバのプライベートなデバイスに格納されているかは、関係ありません。CFS は、クラスタ・メンバ間でキャッシュの整合性を保つことにより、クラスタ内にマウントされているファイル・システムの見え方が常にすべてのメンバで同じであることを保証します。

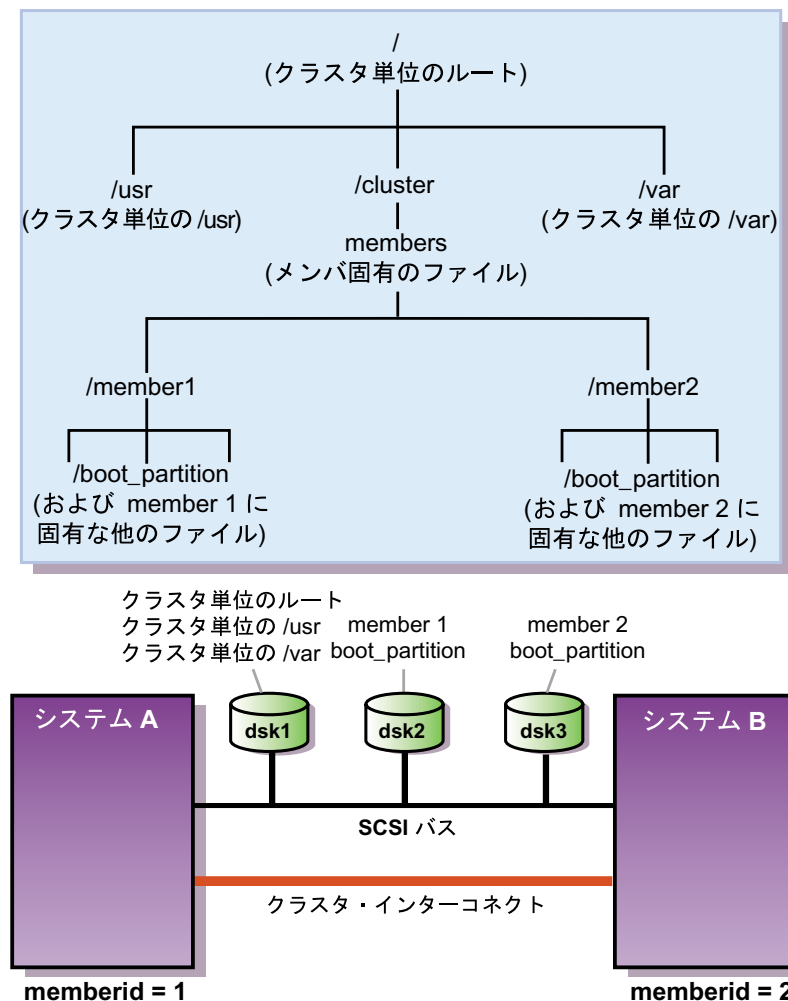
CFS から見ると、1つのクラスタ・メンバがクラスタ全体に対して、各ファイル・システムや AdvFS ドメインについてのサービスを行っているように見えます。どのクラスタ・メンバも、クラスタ内のどのデバイス上にあるファイル・システムであっても、サービスを行うことができます。クラスタのブート時にマウントされたファイル・システムについては、そのファイル・システムに最初にアクセスしたクラスタ・メンバがサービスを行います。つまり、1つのクラスタ・メンバのプライベートなバスに接続されているデバイスのファイル・システムについては、そのメンバがサービスを行うことになります。

このクライアント/サーバ・モデルでは、1つのクラスタ・メンバが、あるドメインではクライアントになり、他のドメインではサーバになります。また、メンバの役割がクライアントかサーバかを切り替えることもできます。たとえば、`/usr/sbin/cfsmgr` コマンドをオプションなしで入力すると、ドメインおよびファイル・システムの名前、それぞれがマウントされている位置、それぞれのサーバの名前、サーバの状態が返されます。この情報を使用して、ファイル・システムを他の CFS サーバに再配置し、クラスタ内の負荷をバランスさせることができます。

CFS は、X/Open および POSIX のファイル・システム・アクセスに関するセマンティクスに従っているため、ファイル管理インタフェースおよびユーティリティは、スタンドアロン・システムと同じように動作します。

図 2-3 は、共用バス上のディスクにあるファイル・システムと、それらを結合したクラスタ・ディレクトリ構造の関係を示しています。各メンバは固有のブート・パーティションからブートしますが、その後、そのファイル・システムを、クラスタ単位のネームスペースにある `boot_partition` マウント・ポイントにマウントします。この図は、クラスタ内のファイル・システムの見え方が、どのクラスタ・メンバでも同じになることを示す一例です。物理的な構成は、他にも多数考えられます。また、実際のクラスタには、ルート (`/`)、`/usr`、`/var` などの重要なファイル・システムをミラーリングするためのストレージが追加されています。

図 2-3: すべてのクラスタ・メンバからファイル・システムを使用できるようにする CFS



ZK-1439U-AI

CFS では、次の性能強化が行われています。

- 直接入出力: ファイル (フラグ `O_DIRECTIO` を使ってオープンする) への直接入出力が使用可能な場合は、ディスク・ストレージに対する読み取り/書き込み要求が、AdvFS および CFS のキャッシングをバイパスし、直接メモリ・アクセスを使用して実行されます。これにより、キャッシングとファイル領域の同期化を行うデータベース・アプリケーションの入出力の性能が向上します。CFS サーバのローカルなアプリケーション

と同様にリモート CFS クライアントも、直接入出力用にオープンされたファイル・システムに対する読み取り/書き込みを直接的に行うことができます。どのメンバの入出力要求でも、ファイルに対する直接入出力はクラスタ・インターコネクトを経由して CFS サーバへは転送されません。

- **ダイレクト・アクセス・キャッシュド・リード:** AdvFS ファイル・システムでは、サイズが 64 K バイト以上の場合の読み取り性能が向上しました。CFS では、ダイレクト・アクセス・キャッシュド・リード機能により、同時に複数のクラスタ・メンバがストレージから直接読み取りを行うことができるようになりました。ダイレクト・アクセス・キャッシュド・リードでは、読み取り要求を出したクラスタ・メンバがそのファイル・システムのあるストレージに直接接続されている場合には、クラスタ・インターコネクトを経由して CFS サーバへアクセスしに行くのではなく、そのストレージに直接アクセスします。この機能拡張により、サーバがメタデータとログを更新するという今までのファイル・システムのモデルをそのまま活かしながら、CFS クライアントからストレージに対して直接入出力できるようにすることで、クラスタ・インターコネクトの負荷を削減しました。

64 K バイト以上のファイルを読み書きするアプリケーションでは、そのファイルからデータを読み込む際にダイレクト・アクセス・キャッシュド・リードを使用します。たとえば、次のようなアプリケーションは、ダイレクト・アクセス・キャッシュド・リードの効果が期待できます。

- Web サーバやプロキシ・サーバのように、ほとんどが読み取りだけを行うマルチ・インスタンス・アプリケーション。これらのアプリケーションでは、複数のクラスタ・ノードから同時に直接アクセスすることができます。
- バックアップ・アプリケーション。アプリケーションがどのノードで動作しても、ファイル・システムの内容はクラスタ・インターコネクトを経由しません。
- **ファイル・システムのマウント。** mount コマンドがあるメンバ上で発行されたときにそのメンバが対象となるストレージに接続されていなければ、そのストレージに接続されているメンバがファイル・システムの CFS サーバとして選ばれます。

詳細は、『クラスタ管理ガイド』を参照してください。

## 2.3 デバイス要求ディスパッチャ

TruCluster Server クラスタでは、デバイス要求ディスパッチャ・サブシステムが、物理デバイスへの入出力をすべて制御します。すべてのクラスタ入出力は、このサブシステムを経由します。このサブシステムでは、単一システムによるオープンだけが認められているので、デバイスをオープンできるのは一度に1つのプログラムだけとなります。デバイス要求ディスパッチャにより、物理ディスクおよびテープ・ストレージは、そのストレージがクラスタ内で物理的に存在する位置に関係なく、全クラスタ・メンバから使用できるようになります。デバイス要求ディスパッチャは、新しいデバイス命名モデルを使用して、クラスタ全体でデバイス名の一貫性を保ちます。これにより、非常に柔軟にハードウェアを構成できるようになります。メンバは、ディスクがつながっているバスに直接接続されていなくても、そのディスクのストレージにアクセスできます。

デバイス要求ディスパッチャは、必要に応じてクライアント/サーバ・モデルを使用します。CFS はファイル・システムや AdvFS サーバ・ドメインのサービスを行いますが、デバイス要求ディスパッチャは、ディスク、テープ、CD-ROM ドライブなどのデバイスに対するサービスを行います。CFS のクライアント/サーバ・モデルでは、1つのクラスタ・メンバがクラスタ全体に対して各ファイル・システムまたは AdvFS ドメインのサービスを行いますが、デバイス要求ディスパッチャでは、同時に動作する多数のサーバを使用することができます。

デバイス要求ディスパッチャ・モデルでは、クラスタ内のデバイスはサーバの1つに接続されている入出力デバイスか、ダイレクト・アクセス入出力デバイスです。サーバの1つに接続されているデバイス(テープ・デバイスなど)は、1つのメンバ、つまりそのデバイスのサーバからのアクセスだけをサポートします。ダイレクト・アクセス入出力デバイスは、複数のクラスタ・メンバからの同時アクセスをサポートします。共用バス上のダイレクト・アクセス入出力デバイスには、そのバス上にあるすべてのクラスタ・メンバがサービスを行うことができます。

`drdmgr` コマンドを使用すると、デバイスのデバイス要求ディスパッチャの状態を調べることができます。次の例では、`dsk6` デバイスが共用バス上にあり、3つのクラスタ・メンバがサービスを行っています。

```
# drdmgr dsk6
```

```
View of Data from member polishham as of 2000-07-26:10:52:40
```

```
Device Name: dsk6
Device Type: Direct Access IO Disk
Device Status: OK
Number of Servers: 3
  Server Name: polishham
  Server State: Server
  Server Name: pepicelli
  Server State: Server
  Server Name: provolone
  Server State: Server
Access Member Name: polishham
Open Partition Mask: 0x4 < c >
Statistics for Client Member: polishham
  Number of Read Operations: 737
  Number of Write Operations: 643
  Number of Bytes Read: 21176320
  Number of Bytes Written: 6184960
```

デバイス要求ディスクパッチャは、文字型ディスク・デバイスとブロック型ディスク・デバイスの両方へのクラスタ単位のアクセスをサポートしています。TruCluster Server 構成内の raw ディスク・デバイス・パーティションへのアクセスは、Tru64 UNIX のスタンドアロン・システムと同じ方法で行います。つまり、`/dev/rdisk` ディレクトリ内のデバイス・スペシャル・ファイルの名前を使用します。

---

#### 注意

---

TruCluster Server バージョン 5.0 より前は、ストレージへのこのようなレベルの物理アクセスを可能にするためには、クラスタ管理者が特別な DRD (Distributed Raw Disk) サービスを定義しなければなりませんでした。このアクセス方法は、TruCluster Server バージョン 5.0 からはクラスタ・アーキテクチャ内に組み込まれており、すべてのクラスタ・メンバが自動的に利用できます。

---

## 2.4 CFS とデバイス要求ディスクパッチャに関する FAQ

ここでは、CFS とデバイス要求ディスクパッチャに対する FAQ (Frequently Asked Questions: よく尋ねられる質問) のうち、次の内容に関する FAQ にお答えします。

- CFS、入出力、およびクラスタ・インターコネクト (2.4.1 項)



- AdvFS 要求ブロックのキャッシング (2.4.2 項)
- デバイス要求ディスパッチャとファイルのオープン (2.4.3 項)
- CFS サーバの再配置 (2.4.4 項)

### 2.4.1 CFS , 入出力 , およびクラスタ・インターコネクト

質問: ダイレクト・アクセスが可能な入出力ディスクの接続されている共用バスで入出力を行った場合、その入出力はクラスタ・インターコネクトを通らなければなりませんか。

答え: raw 入出力の場合、そのデバイスに直接接続されているノードは、デバイス要求ディスパッチャを介してそのデバイスにある raw パーティションへ直接アクセスします (デバイスのサーバになっているノードは `drdmgr` コマンドで調べられます)。

ファイル・システムではなく、直接接続されているブロック・ストレージへブロック入出力を行うと、その入出力は、そのデバイス・スペシャル・ファイルの CFS サーバが処理します。

通常のファイル入出力では、読み取りを行うファイルのサイズが 64 K バイトより小さい場合、そのファイル・システムの CFS サーバが処理します。CFS クライアントがそのファイル・システムの CFS サーバでない場合、要求はクラスタ・インターコネクトを通して、そのファイル・システムの CFS サーバになっているノードへ送られ、その CFS サーバ・ノードのデバイス要求ディスパッチャに渡されます。ある CFS サーバ・ノードへ送られた要求が、そのノードとは別のノードに存在するデバイス要求ディスパッチャへ転送されることはありません (非同期書き込みの場合は、いったんメモリに書き込まれた後、後書きによって、サーバへ一括して送られます)。

読み取るファイルが 64 K バイト以上であれば、CFS クライアントはダイレクト・アクセス・キャッシュド・リードを使って、そのファイルをストレージから直接読み取ることができます。

また、プログラムで `O_DIRECTIO` を指定してファイルをオープンすると、その読み取り/書き込み要求に対するディスク・ストレージへの入出力は、AdvFS や CFS でキャッシングされることなく、直接メモリ・アクセスによって実行されます。ファイルに対する直接入出力は、どのメンバがその入出力要求を出したかに関係なく、クラスタ・インターコネクトを通りません。

ダイレクト・アクセス・キャッシュド・リードについての詳細は、2.2 節で説明しています。open(2) も参照してください。

まとめると、入出力は次の場合に、ストレージに対して直接行われます。

- 直接接続されているストレージに対する raw 入出力
- CFS クライアントが CFS サーバと同じノードにある
- O\_DIRECTIO でオープンしたファイルへのファイル入出力
- 64 K バイト以上のファイルの読み込み

## 2.4.2 AdvFS 要求ブロック・キャッシング

質問: AdvFS ファイル・システムの要求ブロックは CFS クライアント・ノードでキャッシングされるのですか。

答え: はい。CFS クライアントはデータをキャッシングし、後書きします。

## 2.4.3 デバイス要求ディスパッチャとファイルのオープン

質問: プログラムでファイルをオープンした場合、デバイス要求ディスパッチャはどの過程で処理に関わるのですか。

答え: open() では CFS しか関係しません。read() と write() では CFS とデバイス要求ディスパッチャが関係してきます。その際、デバイス要求ディスパッチャが関係するのは、read() の場合は CFS の読み込むキャッシュに必要な内容を置かなければならないときであり、また、write() の場合は CFS の書き込むキャッシュを空にしなければいけないときです。

## 2.4.4 CFS サーバの再配置

質問: CFS サーバの再配置はいつ行くと効果的ですか。

答え: cfsmgr コマンドの出力を使って、どのメンバが最も多くの入出力を処理しているかを調べてください。通常、再配置の目的は 1 つのノードにすべてのファイル・システムの処理が集中するのを避けることです。CFS はメモリを大量に使用しますので、すべてのファイル・システムが同じメンバで処理される事態になると、処理が遅くなることがあります。再配置の方法としては、しばらくの間入出力を監視し、どのメンバをどのファイル・システムの CFS サーバとすればよいかを決定した後、システムを適正

なホストへ自動的に再配置する簡単なブート・スクリプトを (たとえば, /sbin/init.d/ に) 作成して適用するのが最も簡単です。

たとえば, 2 つのメンバ (M1 と M2) からなるクラスタと 6 個のファイル・システム (A, B, C, D, E, F) を考えてみてください。入出力を監視した結果, M1 に A, D, および E を, また, M2 に B, C, および F をそれぞれ担当させた方がよいと決定したとします。この場合, M1 が A, D, および E を自分のところに再配置し, M2 が B, C, および F を自分のところに再配置するブート・スクリプトを作成します。

入出力をクラスタ・メンバの間で分散させる場合は, デバイス要求ディスクパッチャではなく, CFS のレベルで分散させてください。つまり, 入出力をクラスタ・メンバの間で分散させる場合は, drdmgr コマンドではなく, cfsmgr コマンドを使用してください。

## 2.5 コンテキスト依存シンボリック・リンク

単一のネームスペースを使用することによりシステム管理が大幅に簡略化されますが, 構成ファイルやディレクトリをすべてのクラスタ・メンバと共有したくはありません。たとえば, メンバのファイル /etc/sysconfigtab にはシステムのカーネル構成要素の構成に関する情報が含まれており, その構成を使用するのはそのシステムのみでなければなりません。このため, クラスタでは, 各メンバが /etc/sysconfigtab という名前のファイルの読み取り/書き込みを行うと, 実際にはメンバ固有の sysconfigtab ファイルの読み取り/書き込みを行うというメカニズムを使用する必要があります。

次のような特徴を持つネームスペースを作成するために, TruCluster Server は, Tru64 UNIX バージョン 5.0 で導入されたコンテキスト依存シンボリック・リンク (CDSL) という特殊な形式のシンボリック・リンクを使用します。CDSL を使用すると, 名前がクラスタ単位のファイルまたはディレクトリを表しているか, メンバ固有のファイルまたはディレクトリを表しているかには関係なく, 同じ名前がファイルまたはディレクトリにアクセスできます。CDSL では, 従来の命名規則を守りながら, 各メンバがメンバに固有のシステム構成ファイルを読み書きできるようにしています。

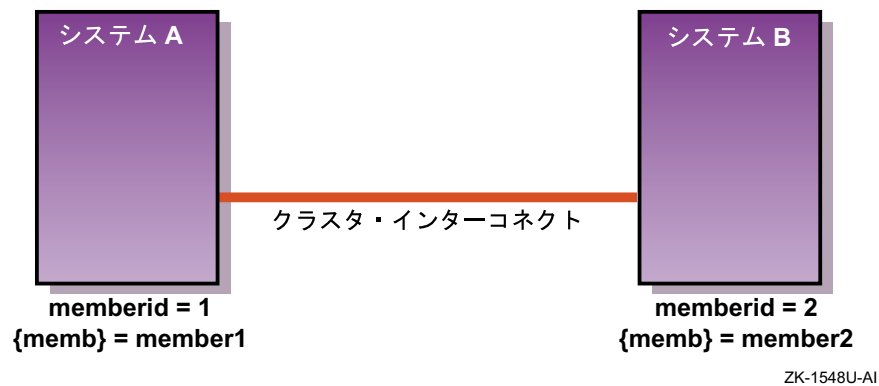
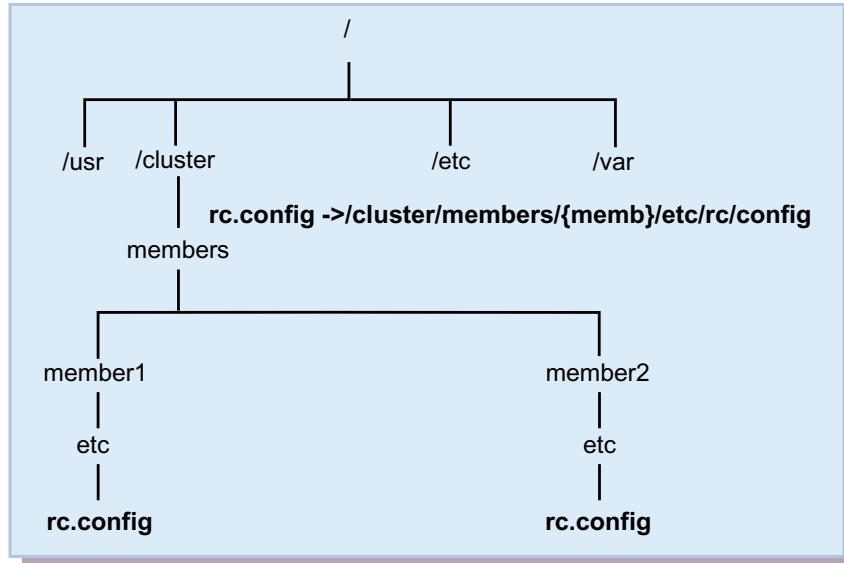
CDSL には, パス名の解決時にだけ決定される値を持つ変数が含まれています。{memb} 変数は, クラスタ内にあるメンバ固有のファイルへのアクセスに使用されます。次の例は, /etc/rc.config への CDSL を示しています。

```
/etc/rc.config -> ../cluster/members/{memb}/etc/rc.config
```

CDSL のパス名を解決する際、カーネルは {memb} 変数を member $n$  ( $n$  は現在のメンバのメンバID) という文字列に置き換えます。つまり、メンバID が2のクラスタ・メンバでは、パス名 /cluster/members/{memb}/etc/rc.config は、/cluster/members/member2/etc/rc.config として解決されます。図 2-4 は、{memb} と、CDSL のパス名解決の関係を示しています。

異なるクラスタ・メンバがそれぞれ異なるデータのセットで、アプリケーションのインスタンスを複数実行する場合には、CDSL が便利です。アプリケーションが CDSL を使用してメンバ固有のデータ・セットおよびログ・ファイルを保守する方法については、『クラスタ高可用性アプリケーション・ガイド』を参照してください。

図 2-4: CDSL のパス名解決



ファイルまたはディレクトリを移動する前に、移動先の名前が CDSL と一致しないことを確認してください。移動先のファイル名が CDSL と一致する場合には、メンバ固有のファイルを保守するために特別な配慮が必要になります。たとえば、次の例に示す /etc/rc.config ファイルがあるとします。

```
/etc/rc.config -> ../cluster/members/{memb}/etc/rc.config
```

ファイルを /etc/rc.config に移動すると、シンボリック・リンクが実際のファイルに置き換わります。したがって、/etc/rc.config は、

/cluster/members/{memb}/etc/rc.config へのシンボリック・リンクではなくなります。

mkcdsl コマンドを使用すると、システム管理者は CDSL の作成と CDSL のインベントリ・ファイルの更新ができます。cdslinvchk コマンドは、現在の CDSL インベントリを検証します。これらのコマンドについての詳細は、mkcdsl(8) および cdslinvchk(8) を参照してください。

CDSL についての詳細は、Tru64 UNIX の『システム管理ガイド』、hier(5)、ln(1)、symlink(2) を参照してください。

## 2.6 デバイス名

ここでは、Tru64 UNIX バージョン 5.0 で導入されたデバイス命名モデルについて簡単に説明します。このデバイス命名モデルについての詳細は、Tru64 UNIX の『ハードウェア管理ガイド』を参照してください。

デバイス名はクラスタ単位で一貫性があり、次のような特徴があります。

- ブートしても名前が変わらない。
- ディスクまたはテープをクラスタ内の別の位置に移動しても、デバイス名が変わらない。

Tru64 UNIX バージョン 5.0 のリリースより前は、ディスク・デバイスにはディスクの入出力パスがコード化され、付与されていました。このパスには多くのデータが組み込まれており、少なくとも次のような情報が含まれていました。ディスクが接続されているコントローラへのアクセスに使用するデバイス・ドライバ、ドライバが管理するシステムのコントローラのインスタンス、コントローラごとのデバイス・ユニット ID などの情報です。

たとえば、rz デバイス・ドライバは SCSI と ATAPI/IDE の両方のデバイス・コントローラへのアクセスに使用されていました。これらのコントローラに接続されたディスクの名前は、rzn です。n は、接続されたディスクへのコントローラとユニット ID の両方を表していました。たとえば、2 番目の SCSI/ATAPI/IDE コントローラ上での SCSI ID=3 のディスクは、rz11 でした。そのディスクを 3 番目のコントローラへ移動すると、rz19 となっていました。

Tru64 UNIX バージョン 5.0 では新しいデバイス命名モデルを使用するようになりました。このモデルのデバイス名は簡単に、デバイスの種類を示す名前とインスタンス番号から構成されています。これらの 2 つの要素で、dsk0

などの、デバイスのベース名が形成されます。デバイスの新しい名前のインスタンス番号は古い名前のユニット番号に対応しません。オペレーティング・システムは、デバイスを検出すると、インスタンス番号を0(ゼロ)から順に割り当てます。また、最新のディスクには、ディスクを別々のものとして識別するIDがあります。この機能をサポートするディスクの場合、Tru64 UNIX バージョン 5.0 はこのIDを追跡し、それを使ってディスクとデバイス名をマップするテーブルを保守します。そのため、これらのディスクの物理的接続を変更しても、そのディスクのデバイス名は変わりません。この機能により、システム管理者がシステム上でディスクを構成するときの柔軟性が大きくなります。

TruCluster 環境では、クラスタ内の各ディスクには単一の名前が与えられるので、新しいデバイス命名モデルの柔軟性は特に有益です。

注意

Tru64 UNIX は、古いスタイルのデバイス名を互換性のオプションとしてサポートしていますが、TruCluster Server は新しいスタイルのデバイス名のみをサポートしています。古いスタイルのデバイス名 (/dev の構造) に依存するアプリケーションは、新しいデバイス命名モデルを使用するように変更する必要があります。

表 2-2 に、新しいデバイス名の例を示します。

表 2-2: 新しいデバイス名の例

古い名前	新しい名前	説明
/dev/rz4c	/dev/disk/dsk4c	オペレーティング・システムが5番目に認識したディスクのcパーティション
/dev/rz19c	/dev/disk/dsk5c	オペレーティング・システムが6番目に認識したディスクのcパーティション

デバイス名スペシャル・ファイルに割り当てられるサフィックスは、デバイスのタイプによって次のように変わります。

- ディスク — 一般に、ディスク・デバイス・ファイル名は、ベース名と1文字のサフィックス(a ~ z)で構成されます(たとえば、/dev/disk/dsk0a)。ディスクでは、a ~ hを使用してパーティションを識別します。省略時には、フロッピー・ディスクおよびCD-ROMデ

バイスでは文字 a および c のみを使用します (たとえば, floppy0a , cdrom1c)。

raw デバイス名用として, 同じデバイス名がディレクトリ /dev/rdisk にもあります。

- テープ — テープ・デバイス・ファイル名は, ベース名とサフィックスで構成されます。このサフィックスは, 文字 \_d の後に 1 桁の数字を付加したものです (たとえば, tape0\_d0)。このサフィックスは, /etc/ddb.dbase ファイルのデバイスのエントリに応じた, テープ・デバイスの密度を表します。例を次に示します。

デバイス	密度
tape0	省略時の密度
tape0c	省略時の密度 (圧縮あり)
tape0_d0	/etc/ddb.dbase 内のエントリ 0 に対応する密度
tape0_d1	/etc/ddb.dbase 内のエントリ 1 に対応する密度

テープ用デバイス・スペシャル・ファイルの新しい命名規則では, 古い名前のサフィックスに対して, 次を示すような新しい名前のサフィックスが対応します。

古いサフィックス	新しいサフィックス
l (低)	_d0
m (中)	_d2
h (高)	_d1
a (代替)	_d3

テープのデバイス名は 2 セットあり, どちらも新しい命名規則に準拠しています。—巻き戻しのあるデバイスは /dev/tape ディレクトリにあり, 巻き戻しのないデバイスは /dev/ntape ディレクトリにあります。/etc/ddb.dbase ファイルの内容を見ると, どちらのデバイス・スペシャル・ファイルを使用するかを判断できます。

Tru64 UNIX には, デバイス名を識別するためのユーティリティがあります。たとえば, 次の hwmgr コマンドを使用すると, クラスタ内のデバイスおよびデバイスの階層に関する情報がそれぞれ表示されます。



```
# hwmgr -view devices -cluster
# hwmgr -view hierarchy -cluster
```

hwmgr コマンドを使用して、メンバのハードウェア構成をリストし、バス・ターゲット LUN 名と /dev/disk/dskn 名の対応を表示することができます。hwmgr コマンドについての詳細は、hwmgr(8) を参照してください。

---

#### 注意

---

LSM (Logical Storage Manager) の命名規則は変更されていません。

---

## 2.7 ワールドワイド ID

Tru64 UNIX は、新しいデバイス名をディスクのワールドワイド ID (WWID) に対応付けます。ディスクの WWID は一意であり、WWID をサポートするデバイスの製造元で設定されます。このため、2 台のディスクが同じ WWID を持つことはありません。WWID を使用してディスクを識別することには、2 つの意味があります。オペレーティング・システムがいったんディスクを認識すると、ディスクの /dev/disk/dsk 名は SCSI アドレスが変わっても変わりません。

ディスクを認識するこの機能により、Tru64 UNIX は、複数の SCSI アダプタを通してアクセスできるディスクの場合に、ディスクへのマルチパスをサポートできます。TruCluster Server 環境内でディスクを移動しても、ディスクのデバイス名と、ディスクへのアクセス方法は変わりません。

---

#### 注意

---

RAID アレイ・コントローラ下のディスクの名前は、コントローラ・モジュールの WWID と、自身のバス、ターゲット、LUN の位置の両方に対応付けられます。この場合、ディスクを移動すると、そのデバイス名が変わります。ただし、hwmgr ユーティリティを使用して、このようなディスクを以前のデバイス名に対応付けることはできます。

---

次の hwmgr コマンドは、クラスタの WWID を表示します。

```
# hwmgr -get attr -a name -cluster
```

## 2.8 クラスタと LSM

LSM (Logical Storage Manager) を使用すると、どのクラスタ・メンバからでもすべての LSM ボリュームへ共用でアクセスできます。LSM は、物理ディスク・デバイス、論理エンティティ、およびその両者を結び付けるマッピングで構成されます。LSM はボリュームと呼ばれる仮想ディスクを、UNIX の物理ディスク上に構築します。LSM は物理ディスクとアプリケーションの間にボリュームを透過的に配置し、アプリケーションが物理ディスクではなくボリュームを操作するようにします。たとえば、物理ディスク上ではなく、LSM ボリューム上にファイル・システムを作成します。

図 2-1 ですでに示したように、LSM はデバイス要求ディスパッチャの上に階層化されています。単一のシステムで LSM を使用するのと同じように、クラスタ内で LSM が使用できます。クラスタ構成と非クラスタ構成のどちらにも同じ LSM ソフトウェア・サブセットが使用され、どのクラスタ・メンバからでも構成を変更できます。LSM は、クラスタ単位で構成の一貫性を保ちます。

クラスタ単位の基本ファイル・システムに対する LSM のサポート状況は次のとおりです。

- サポート: ルート (/), /usr, /var の各ファイル・システム、および、メンバのスワップ・パーティション
- 未サポート: クォーラム・ディスクとメンバのブート・ディスク

TruCluster Server 環境での LSM に特有の構成および使用方法については、Tru64 UNIX 『*Logical Storage Manager*』を参照してください。LSM の実装に関する Tru64 UNIX と TruCluster Server の違いについては、『*クラスタ管理ガイド*』を参照してください。

## 接続マネージャ

クラスタ化されたシステムでは、さまざまなデータとシステム・リソース (ディスクおよびファイルへのアクセスなど) を共有します。リソースの完全性を維持するための調整を行うには、クラスタにはメンバシップの明確な基準が必要で、その基準を満たしていないシステムはクラスタに参加できないようにしなければなりません。

接続マネージャは、クラスタのメンバが相互に通信できるかどうかを監視し、クラスタのメンバシップに関する規則を強制する分散型のカーネル構成要素です。接続マネージャには次のような機能があります。

- クラスタの形成、クラスタへのメンバの追加、およびクラスタからのメンバの削除
- クラスタのどのメンバが稼働中であるかの追跡
- すべてのクラスタ・メンバで一貫したクラスタ・メンバシップ・リストの保守
- イベント・マネージャ (EVM) イベントを使用した、メンバシップ変更のタイムリな通知
- クラスタ分断の検出と処理

接続マネージャのインスタンスは、クラスタの各メンバ上で動作します。これらのインスタンスは、クラスタのメンバシップ・リストのような情報を共有することによって、メンバ間の相互接続を維持します。接続マネージャは、三相コミット・プロトコルを使って、すべてのメンバから見えるクラスタのビューの一貫性を維持します。

この章では、次の事項について説明します。

- クォーラム、ポート、およびクラスタ・メンバシップの説明 (3.1 節)
- 接続マネージャによるクォーラムの計算方法の説明 (3.2 節)
- クォーラム・ディスクの使用時期と使用方法 (3.3 節)

## 3.1 クォーラムとポート

接続マネージャは、投票メカニズムを使用することによって、通信障害の発生時でもデータの一貫性を保証します。このメカニズムでは、ポート (投票数) が過半数に達したときに限り、クラスタ内でのプロセス動作と入出力操作を許可します。このようにクラスタ内に過半数のポートが存在する状態を、クラスタがクォーラム (定足数) を維持しているといいます。

接続マネージャはクォーラムを計算し、その結果に基づいて、システムがクラスタ・メンバになることを許可します。このような投票メカニズムは、期待ポート、現在のポート、ノード・ポート、クォーラム・ディスク・ポートなど多くの要素に依存します。この節では、これらの要素の概念について説明します。

### 3.1.1 クラスタのメンバとは

クラスタのメンバシップを管理できるのは接続マネージャだけです。コマンド `clu_create` または `clu_add_member` を使ってクラスタのメンバになるようにノードを構成しても、そのノードはすぐにはクラスタのメンバになりません。メンバになるのは、クラスタ化カーネルによってリブートされ、接続マネージャによってクラスタの形成またはクラスタへの参加を許可されてからです。クラスタのメンバと、クラスタのメンバになるように構成されたノードとの違いは、クォーラムとポートについて考えるとき常に重要です。

ノードがクラスタを形成したか、クラスタに参加すると、そのノードは接続マネージャによって無期限に (ただし `clu_delete_member` を使ってクラスタから削除されるまで) クラスタのメンバとみなされます。クラスタ内でのハードウェアの故障または切断などによる通信の切断によって、既存クラスタが複数のクラスタに分断されることがあります。この状況をクラスタ分断といいます。クラスタ分断が生じると、ノードは所属先のクラスタを特定できなくなる可能性があります。ただし 3.2 節で説明するように、接続マネージャは、これらのクラスタの 1 つしか稼働させません。

### 3.1.2 ノード・ポート

ノード・ポートは、クラスタの 1 つのメンバがクォーラムに投じるポートの定数です。メンバには 1 または 0 (ゼロ) ノード・ポートを割り当てることができます。1 ポートを持つメンバは、クラスタの投票メンバとみなされます。0 (ゼロ) ポートを持つメンバは非投票メンバとみなされます。

投票メンバはクラスタを形成できます。非投票メンバは既存クラスタへの参加しかできません。

メンバのポートは、最初、メンバ固有の `etc/sysconfigtab` ファイル内にある、`clubase` サブシステムの `cluster_node_votes` カーネル属性によって決定されます。

### 3.1.3 クォーラム・ディスク・ポート

一部のクラスタ構成では、3.3 節の説明に従ってクォーラム・ディスクを構成することにより、クラスタの可用性を高めることができます。クォーラム・ディスク・ポートは、クォーラム・ディスクがクォーラムに投じるポートの定数です。クォーラム・ディスクには1または0ポートを割り当てることができます。

クォーラム・ディスク・ポートは、各メンバの `etc/sysconfigtab` ファイル内にある `clubase` サブシステムのカーネル属性 `cluster_qdisk_votes` によって初期化されます。

クォーラム・ディスクを構成すると、そのポートは、接続マネージャが強制する次の規則に従って、クラスタの形成で特殊な役割を果たします。

- ブート・ノードは、クォーラムを満たすまでクラスタを形成できない。
- ブート・ノードは、クォーラム・ディスクの所有権とそのポートを要求するために、クラスタのメンバでなければならない。

つまり、ブート・ノードがクォーラムを満たそうとしてクォーラム・ディスク・ポートを要求すると、これらの規則によってジレンマに陥ります。ブート・ノードはいつになってもクラスタを形成できません。

このジレンマから抜け出すために、接続マネージャは、ブート・ノードがクォーラムにクォーラム・ディスク・ポートを一時的に投じることができるようにします。これによって、ブート・ノードはクォーラムを満たして、クラスタを形成できます。ブート・ノードは、クラスタを形成した後、クォーラム・ディスクの所有権を要求します。この時点で、そのクォーラム・ディスクのポートは仮のポートから正式なポートになります。

### 3.1.4 期待ポート

期待ポートは、構成済みのポートがすべて使用可能なときに接続マネージャが期待するポートの数です。つまり期待ポートは、クラスタ内で構成済みの

ノード・ポートとクォーラム・ディスク・ポートとの合計です (クォーラム・ディスク・ポートはクォーラム・ディスクが構成されている場合のみ加算します)。各メンバは、それぞれ自身の期待ポートをクラスタに渡します。すべてのメンバの期待ポートの数が一致する必要があります。

接続マネージャは、クラスタのブート・メンバのノードの期待ポートを参照して、クラスタ全体の期待ポートを独自に内部で決定します。このクラスタ全体の期待ポートのことをクラスタ期待ポートといいます。接続マネージャは、自分のクラスタ期待ポートの値を使って、クラスタがクォーラムを維持するために必要なポート数 (3.2 節を参照) を決定します。

クラスタ期待ポートの現在の値を表示するには、`clu_quorum` コマンドまたは `clu_get_info-full` コマンドを使用します。

`clu_create` および `clu_add_member` コマンドは、新しい投票メンバまたはクォーラム・ディスクがクラスタ内に構成されると、各メンバの期待ポートを自動的に調整します。`clu_delete_member` コマンドは、メンバが削除されると期待ポートの数を自動的に減らします。同じように、`clu_quorum` コマンドは、クォーラム・ディスクが追加または削除されたときや、ノード・ポートがメンバに割り当てられたりメンバから削除されたときに、各メンバの期待ポートを調整します。これらのコマンドにより、メンバ固有の期待ポート値はどのクラスタ・メンバでも同じで、すべてのノード・ポートとクォーラム・ディスク・ポート (クォーラム・ディスクが構成されている場合) の合計になります。

メンバの期待ポートは、メンバ固有の `etc/sysconfigtab` ファイル内の、`clubase` サブシステムの `cluster_expected_votes` カーネル属性によって初期化されます。メンバの期待ポートを表示するには、`clu_quorum` コマンドを使用します。

### 3.1.5 現在のポート

現在のポートは、クラスタ内で実際に見えるポートの数です。期待ポートの数がクラスタ内で構成されたポートの数と一致する場合、現在のポートは、現在オンラインのメンバと構成済みクォーラム・ディスクが投じたポートの合計です。

## 3.2 クラスタ・クォーラムの計算

接続マネージャは、クォーラム・アルゴリズムを使って、あるノードがクラスタに参加し、クラスタ全体のリソースに安全にアクセスして、有用な作業を実行できるかどうかを調べます。このアルゴリズムは動的です。つまり、クラスタのクォーラム・ボートの計算はクラスタのイベントによって開始され、その計算結果はクラスタの存在期間に渡って変化します。ここでは、接続マネージャのクォーラム・アルゴリズムの仕組みについて説明します。

クォーラム・アルゴリズムは次のように機能します。

1. 接続マネージャは、クラスタのクォーラム・ボートの計算に使用するクラスタ・メンバの集合を選択します。この集合には相互通信可能なメンバのみが含まれます。構成されているがブートされていないノード、ダウンしているメンバ、ハードウェアの障害（クラスタのインターコネク・ケーブルの切断やクラスタ・インターコネク・アダプタの故障）によって通信不能になったメンバなどは含まれません。
2. クラスタが形成され、ノードがブートされてクラスタに参加するたびに、接続マネージャは次の値から最大値を選択し、その値を使ってクラスタ期待ボートを計算します。
  - 手順 1 で選択したメンバの集合から取得したメンバの期待ボートのうちの最大値
  - 手順 1 で選択したメンバの集合から取得したノード・ボートとクォーラム・ディスク・ボート（クォーラム・ディスクが構成されている場合）との合計
  - 前回のクラスタ期待ボート値

たとえば、クォーラム・ディスクがない 3 メンバ構成のクラスタがあるとします。すべてのメンバは稼働中であり、相互に接続されています。各メンバのノード・ボートは 1 に設定され、期待ボートは 3 に設定されています。クラスタ期待ボートは現在 3 です。

その後、4 番目の投票メンバがクラスタに参加するとします。この新しいメンバがブートしてクラスタに参加した場合、接続マネージャはクラスタ期待ボートを計算して、4 に設定します。この値はクラスタ内のノード・ボートの合計です。

クラスタ期待ボートの現在の値を表示するには、`clu_quorum` コマンドまたは `clu_get_info-full` コマンドを使用します。

3. 接続マネージャは、クラスタ期待ボートを再計算するたびに (または `clu_quorum -e` コマンドでクラスタ期待ボートをリセットするたびに)、クォーラム・ボートを計算します。

クォーラム・ボートは、クラスタ期待ボートの値に基づいて動的に計算されるクラスタ単位の値です。この値によって、ノードに対し、クラスタの形成、クラスタへの参加、またはクラスタへの残留を許可するかどうかが決まります。接続マネージャは、次の数式を使ってクラスタのクォーラム・ボートを計算します。

クォーラム・ボート = (クラスタ期待ボート + 2) / 2 (小数点以下切り捨て)

たとえば、手順 2 で説明した 3 メンバ構成のクラスタの場合、クラスタ期待ボートが 3 なので、クォーラム・ボートは、 $(3+2)/2$  で計算して小数点以下を切り捨てた結果、2 になります。4 番目のメンバが正常に追加された場合、クォーラム・ボートは、 $(4+2)/2$  で計算して小数点以下を切り捨てた結果、3 になります。

---

#### 注意

---

期待ボートは (結果的にクォーラム・ボートも) クラスタ構成に基づきます。どのノードが稼働しているかダウンしているかには関係ありません。メンバがシャットダウンされたか、他の何らかの理由でダウンしたとき、接続マネージャはクォーラム・ボートの値を減らしません。メンバが削除されたか、`clu_quorum -e` コマンドが実行されたときだけ、接続マネージャは稼働中のクラスタのクォーラム・ボート値を減らします。

4. クラスタのメンバが認識可能なボート数が変化したとき (ノードがクラスタに参加するか、既存メンバがクラスタから削除されるか、通信エラーが報告されているとき)、そのメンバは常に現在のボートとクォーラム・ボートとを比較します。

メンバは、次の条件に基づいた動作を実行します。



- 現在のポートの値がクォーラム・ポートの値以上であれば、そのメンバは実行を続行するか、再開 (中断状態だった場合) します。
- 現在のポートの値がクォーラム・ポートより小さい場合、入出力はすべて中断され、クラスタ・インターコネクト・インタフェースを除くネットワーク・インタフェースはすべてオフになります。クラスタ単位のリソースにアクセスするコマンドは、そのメンバ上では動作せず、ハングしたように見えます。

この状態は、十分なポートが追加され (つまり、十分なメンバがクラスタに参加するか通信上の問題が修復されて)、現在のポートがクォーラム以上になるまで続きます。

クォーラム・ロスがクラスタ単位のイベントのように見えますが、現在のポートとクォーラム・ポートの比較は、メンバごとに行われます。

そのメンバがどのようにクォーラムを失ったかにもよりますが、クォーラムを失ったメンバがクォーラムを満たせるだけのポートを別のメンバに割り当て、そのメンバをブートし、クォーラムを回復させることによって、この状況を解決できる場合があります。ただし、クラスタのすべてのメンバがクォーラムを失った場合は、それらのメンバがクォーラムを満たせるだけのポートを新規メンバに割り当て、その新規メンバをブートするか、クラスタ全体をリブートするか、あるいは『クラスタ管理ガイド』に記載されているトラブルシューティングに関する手順を実行するしかありません。

### 3.3 クォーラム・ディスクの使用

2 メンバのクラスタ構成で、期待ポートが 2 で各メンバにメンバ・ポートが 1 票ずつある場合、メンバを 1 つ失うとクラスタはクォーラムを失い、すべてのアプリケーションが中断されます。このような構成では、可用性は高くありません。

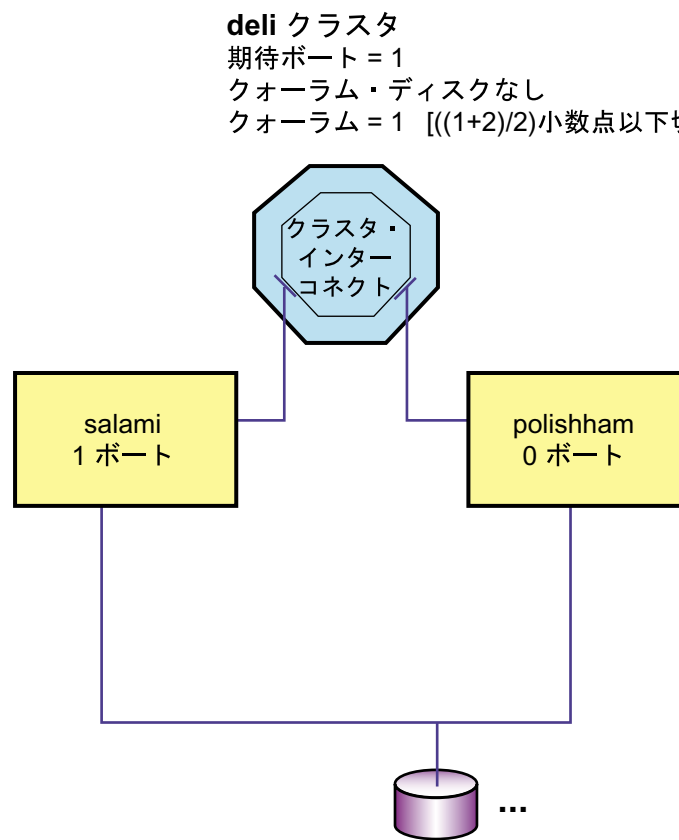
より現実的な (しかし、実質的により良いわけではない) 2 メンバ構成で、1 つのメンバにポートを 1 票割り当て、もう一方のメンバには 0 (ゼロ) 票を割り当てているとします。この場合、期待ポートは 1 になります。このクラスタは 2 番目のメンバ (0 票を持つメンバ) を失っても継続して動作できます。ただし、最初のメンバ (1 票を持つメンバ) を失うことはできません。

このような構成の可用性を高めるために、共用バス上のディスクをクォーラム・ディスクとして指定できます。クォーラム・ディスクは、期待ポートの

合計にポートを 1 つ追加するための、仮想的なクラスタ・メンバの働きをします。2 メンバのクラスタにクォーラム・ディスクが構成されている場合、クォーラム・ディスクやメンバの 1 つに障害が発生してもクラスタは持ちこたえて、動作を続けます。

たとえば、図 3-1 に示すように、クォーラム・ディスクのない、2 メンバの deli クラスタがあるとします。

図 3-1: クォーラム・ディスクのない、2 メンバの deli クラスタ



ZK-1569U-AI

1 つのメンバのノード・ポートは 1 票で、もう一方のメンバのノード・ポートは 0 (ゼロ) 票で、クラスタ期待ポートは 1 です。接続マネージャは、次のようにクォーラム・ポートを計算します。

$$\begin{aligned}\text{クォーラム・ポート} &= (\text{クラスタ期待ポート} + 2) / 2 && (\text{小数点以下切り捨て}) \\ &= (1 + 2) / 2 \\ &= 1\end{aligned}$$

### 3-8 接続マネージャ

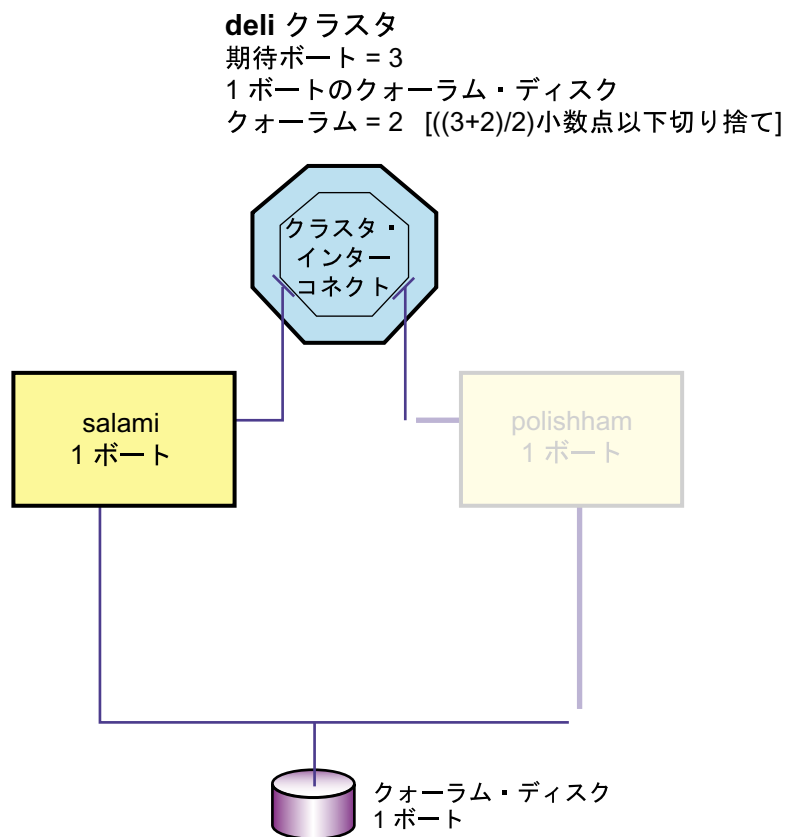
salami メンバの障害やシャットダウン時には、polishham メンバがクォーラムを失います。クラスタの動作は中断されます。

ただし、クラスタにクォーラム・ディスクがあり (クラスタ期待ボートの合計にボートを 1 票追加する)、メンバ polishham にも 1 票与えられると、期待ボートは 3 になり、クォーラム・ボートは 2 のままです。

$$\begin{aligned}\text{クォーラム・ボート} &= (\text{クラスタ期待ボート} + 2) / 2 && (\text{小数点以下切り捨て}) \\ &= (3 + 2) / 2 \\ &= 2\end{aligned}$$

メンバの 1 つ、またはクォーラム・ディスクがクラスタから失われても、クラスタがクォーラムを失わないだけの現在のボートが残ります。図 3-2 のクラスタは、動作を続けます。

図 3-2: メンバが 1 つ失われても持ちこたえる , クォーラム・ディスクがある 2 メンバの deli クラスタ



ZK-1575U-AI

`clu_create` ユーティリティを使用すると、クラスタの作成時にクォーラム・ディスクを指定し、それにポートを 1 票割り当てることができます。また、`clu_quorum` ユーティリティを使用すると、後でクォーラム・ディスクを追加することができます。たとえば、`clu_delete_member` コマンドを実行した結果として、可用性の低下した 2 メンバ構成のクラスタになった場合などです。

クォーラム・ディスクを構成するには、`clu_quorum -d add` コマンドを使用します。たとえば、次のコマンドは、`/dev/disk/dsk11` をポート 1 票を持つクォーラム・ディスクとして定義します。

```
# clu_quorum -d add dsk11 1
Collecting quorum data for Member(s): 1 2
```

```

    Initializing cnx partition on quorum disk : dsk11h

    Successful quorum disk creation
# clu_quorum
Collecting quorum data for Member(s): 1 2

Quorum Data for Cluster: deli as of Thu Mar 9 09:59:18 EDT 2000

Cluster Common Quorum Data
Quorum disk: dsk11h
    .
    .
    .

```

クォーラム・ディスクの使用には、次の制限が適用されます。

- クラスタは、クォーラム・ディスクを1台だけ持つことができる。
- クォーラム・ディスクは、すべてのクラスタ・メンバに直接接続されている共用バス上に置く。

この位置にない場合、クォーラム・ディスクに直接接続されていないメンバは、直接接続されているメンバよりも早くクォーラムを失うことがあります。

- クォーラム・ディスクには、データを置いてはならない。

`clu_quorum` コマンドは、クォーラム・ディスクの初期化時に既存のデータを上書きします。実行中のクラスタからクォーラム・ディスクに置かれたデータ(またはファイル・システムのメタデータ)の完全性は、メンバに障害が発生すると保証されなくなります。

メンバのブート・ディスクとクラスタ単位のルート(/)を含むディスクは、クォーラム・ディスクとしては使用しないでください。

- クォーラム・ディスクは、小さくても構わない。  
クラスタ・サブシステムでは、ディスクを1MBだけ使用します。
- クォーラム・ディスクには、ポートが1票あるか、ポートがない(0(ゼロ)票)かのどちらかである。

通常、クォーラム・ディスクには常にポートを割り当てなければなりません。1メンバ・クラスタのような、テスト構成または一時的な構成では、既存のクォーラム・ディスクにポートを割り当てないこともできます(このような構成では、投票クォーラム・ディスクが第2の故障点になることがあります)。

- クォーラム・ディスクでは , LSM (Logical Storage Manager) は使用できない。

---

## 高可用性アプリケーション

クラスタ上のアプリケーションは、次の3つの基本タイプに分類できます。

### シングル・インスタンス・アプリケーション

シングル・インスタンス・アプリケーションは、ある時点では1つのクラスタ・メンバだけで動作します。このタイプのアプリケーションの可用性を高めるには、現在のクラスタ・メンバでアプリケーションが実行できなくなった場合に他のメンバでそのアプリケーションの実行を開始するというメカニズムがクラスタになければなりません。シングル・インスタンス・アプリケーション用の TruCluster Server 高可用性メカニズムは、CAA (Cluster Application Availability) サブシステムで実現しています。CAA についての詳細は、第5章を参照してください。

『クラスタ高可用性アプリケーション・ガイド』には、TruCluster Software バージョン 1.x シリーズの製品から TruCluster Server バージョン 5.1B にアプリケーションを移行するための詳しい説明があります。

### マルチ・インスタンス・アプリケーション

マルチ・インスタンス・アプリケーションは、同時に複数のクラスタ・メンバで実行できます。マルチ・インスタンス・アプリケーションは、その定義上、高可用性アプリケーションです。これは、1つのクラスタ・メンバに障害が発生しても、他のメンバで実行されているアプリケーションのインスタンスには影響しないためです。クラスタ別名を使ってクラスタがアプリケーションに透過的にアクセスする方法については、第6章を参照してください。

## 分散型アプリケーション

分散型アプリケーションは、クラスタ上で実行するように設計されたアプリケーションで、目的によって異なるメンバ上で動作します。これらのアプリケーションは、Memory Channel、分散ロック・マネージャ (DLM)、クラスタ別名のアプリケーション・プログラミング・インタフェース (API) を使用して、アプリケーションとクラスタ・リソースを統合します。

TruCluster Server を使用すると、分散型アプリケーションの構成要素を並列に実行し、可用性を高めると同時にクラスタ特有の同期メカニズムと性能の最適化という利点を得ることができます。

分散型アプリケーションの作成に使用できるサブシステムおよびインタフェースについての詳細は、第 6 章、7.6 節、第 8 章、および『クラスタ高可用性アプリケーション・ガイド』を参照してください。

---

### 注意

---

6.7 節に `in_single` および `in_multi` サービス属性の説明があります。この属性によって、クラスタ別名サブシステムが、ある別名を通して到着するサービス関連パケットを、その別名に結びつけられているメンバの 1 つにすべて受けさせるか、または、その別名に対応するすべてのメンバに分散させるかが決まります。これらの用語はシングル・インスタンスとマルチ・インスタンスに似ていますが、`in_single` と `in_multi` 属性の方は、クラスタ別名サブシステムがサービス・ポートに到着する接続要求とパケットをどのように振り分けるかに影響します。これらの用語を一般用語 (シングル・インスタンスとマルチ・インスタンス) と混同しないでください。一般用語の方は、1 つのクラスタでインスタンスを何個動作させられるかを表す場合に使います。『クラスタ管理ガイド』にあるクラスタ別名サブシステムを説明した章に、クラスタ別名サブシステムと CAA (Cluster Application Availability) サブシステムの違いが説明されています。

---



---

## CAA (Cluster Application Availability)

この章では、次の事項について説明します。

- CAA (Cluster Application Availability) サブシステムの概要 (5.1 節)
- CAA アーキテクチャの説明 (5.2 節)
- CAA リソースの概要 (5.3 節)
- リソース・プロファイルとその使用方法 (5.4 節)
- CAA コマンドがアプリケーションおよびその他のリソース管理に使用する処理スクリプト (5.5 節)

### 5.1 CAA の概要

CAA (Cluster Application Availability) サブシステムは、シングル・インスタンス・アプリケーションの可用性を高め、他のタイプのリソース (ネットワーク・インタフェース、テープ・デバイス、メディア・チェンジャ・デバイスなど) の状態を監視します。シングル・インスタンス・アプリケーションは、クラスタ上の 1 つのメンバで動作し、同時に複数のメンバで動作することはできません。Tru64 UNIX で動作可能なすべてのシングル・インスタンス・アプリケーションは、CAA のあるクラスタでは可用性を高めることができます。たとえば、クラスタでは、BIND (named)、DHCP (joind)、ネットワーク・ロック (rpc.lockd および rpc.statd) のデーモンが、CAA によって管理されています。

CAA の制御下で動作する各アプリケーションには、アプリケーションのリソース要件と、そのアプリケーションを別のクラスタ・メンバに再配置できる状況を示す、リソース・プロファイルがあります。CAA はクラスタ・メンバとリソースを監視して、各アプリケーションが、そのリソース要件を満たすメンバで動作することを保証します。リソース・プロファイルは、コマンド行インタフェースまたはグラフィカル・ユーザ・インタフェース (GUI) のいずれかを使用して作成および管理することができます。

必須のリソース，または現在のメンバ自体が動作できなくなった場合，CAA は自動的にアプリケーションを他のクラスタ・メンバに再配置します。この機能はアプリケーション自体の変更を必要とせず，どのようなシングル・インスタンス・アプリケーションでも使用できます。また，CAA では，リソースを監視して，リソース障害のためにオフラインにされていたアプリケーション・リソースを再起動できるようにすることができます。

さらに，CAA では，アプリケーション・リソースの配置の再評価を定期的にスケジュールされた時刻に行うことも，または管理者が手動によるアプリケーションの負荷分散を行うときにもできます。負荷分散は，負荷を考慮して行うのではなく，CAA の標準配置決定メカニズムを使用して行います。最適なクラスタ・メンバに配置されなかったアプリケーションは，最も好ましいクラスタ・メンバに再配置されます。

---

#### 注意

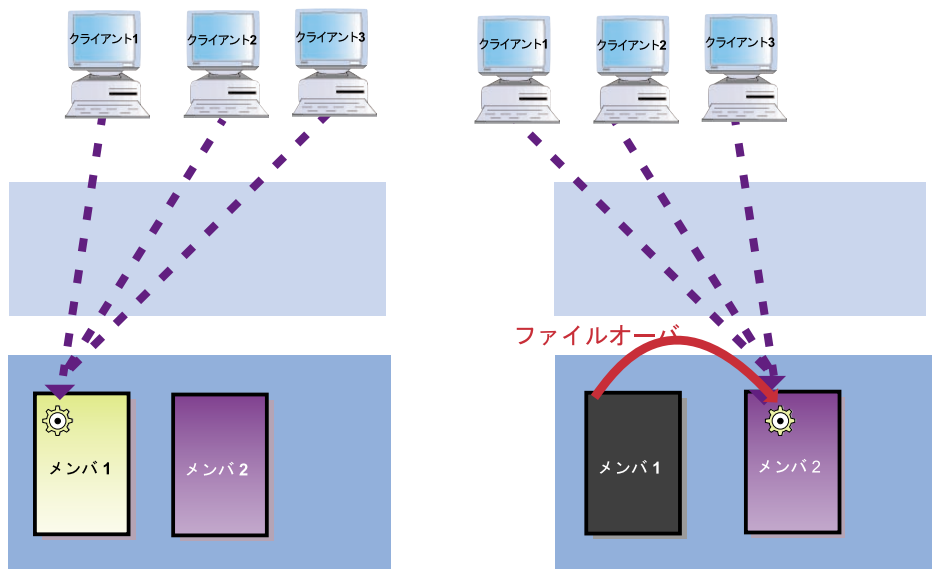
---

CAA によるリソースの監視やアプリケーションの再起動機能は，以前の TruCluster 製品でのユーザ定義サービスのために ASE (Available Server Environment) によって提供されていたタイプのアプリケーションの可用性に対して強化された機能です。

---

メンバの障害により 2 番目のメンバにアプリケーションがフェイルオーバーされる方法を，図 5-1 に示します。クライアントがクラスタ別名を使用してアプリケーションにアクセスしている場合，クラスタ別名サブシステムは，接続要求を自動的に 2 番目のメンバに転送します。

図 5-1: CAA によるアプリケーションのフェイルオーバー  
フェイルオーバー前:                      フェイルオーバー後:



⚙️ シングル・インスタンス・サービス

ZK-1446U-AI

## 5.2 CAA のアーキテクチャ

CAA サブシステムには、次の構成要素があります。

### リソース

リソースとは、エンド・ユーザまたは別のソフトウェア構成要素に対してサービスを提供するクラスタのソフトウェアまたはハードウェア構成要素であり、クライアントに対するサービスを高可用にするために CAA が使用します。CAA では、次のタイプのリソースをサポートしています。つまり、アプリケーション、ネットワーク・インターフェース、テープ・ドライブ、メディア・チェンジャです。

### リソース・マネージャ

リソース・マネージャは、CAA サブシステムのすべての構成要素、接続マネージャおよびイベント・マネージャ (EVM) と通信を行います。

リソース・マネージャは、クラスタ・メンバ上で実行されるすべての CAA デーモンからなります。各 CAA デーモン (caad) は、必須のリソース、アプリケーション自体、またはクラスタ・メンバに障害が発生した場合に、アプリケーション・リソースの起動、停止、再配置、再起動を行います。各クラスタ・メンバは、CAA デーモンを実行します。これらのデーモンは独立していますが、相互に通信を行い、リソースの状態に関する情報を共有します。いずれかの caad デーモンで障害が発生した場合、Essential Services Monitor デーモンである esmd が障害の発生した caad デーモンを再起動するので、リソースの管理を継続することができます。

リソース・マネージャは、リソース・モニタを使用して、特定のタイプのリソースの状態の監視も行います。

リソース・モニタ	リソース・モニタは、 <code>/var/cluster/caa/monitors</code> にあるシェアード・ライブラリであり、ブート時に、リソース・マネージャ caad によってロードされます。各タイプのリソース (アプリケーション、テープ、メディア・チェンジャ) に対して 1 つのリソース・モニタがあります。
リソース・プロファイル	<p>リソース・プロファイルには、リソース・マネージャとリソース・モニタがアプリケーションの再配置の管理とリソースの監視に使用する情報が含まれています。</p> <p>リソース・プロファイルには、リソース、(アプリケーション・リソースに対する) 依存関係、および CAA によるリソースの管理方法を定義する、キーワードと値からなる属性が含まれています。caad_register でリソースを登録すると、リソース・マネージャがリソース・プロファイルを使用できるようになります。</p>

リソース・プロファイルは、`caa_profile` コマンドおよび SysMan で作成することも、任意のテキスト・エディタで作成することもできます。テキスト・エディタで作成または変更したプロファイルは、`caa_profile -validate` によって検証し、正しい構文であることを確認する必要があります。構文エラー以外のエラーは、登録時に検出されます。この 2 段階の検証により、現在、オフライン状態や、これから作成するリソースとの依存関係を使用して、プロファイルを作成することが可能になります。

リソース・プロファイルは、  
`/var/cluster/caa/profile` ディレクトリにあります。リソース・プロファイルのファイル名は、`resource_name.cap` という形式です。

## 処理スクリプト

処理スクリプトは、CAA がアプリケーションの起動、停止、およびチェックに使用するコマンドのセットです。アプリケーションの処理スクリプト名は、アプリケーションのリソース・プロファイル内で定義されます。

処理スクリプトは、コマンド行インタフェース、SysMan、またはテキスト・エディタで作成や更新ができます。

処理スクリプトでは、CAA で利用できる変数を使用できます。すべてのプロファイル属性の値は、処理スクリプトで使用できます。理由コードの変数は、処理スクリプトが実行される理由を示します。処理スクリプトが実行される環境のロケール変数の多くは、処理スクリプトで使用できます。また、ユーザ定義属性も処理スクリプトで使用できます。

オプションで、処理スクリプトの標準出力をスクリプトを起動するコマンドの標準出力にリダイレクトすることができます。省略時の設定では、この標準出力のリダイレクトは行われません。

処理スクリプトは、`/var/cluster/caa/script` ディレクトリにあります。処理スクリプトのファイル名は、`resource_name.scr` という形式です。

#### コマンド行インタ フェース

CAA サブシステムには、リソースの管理と監視を行う、`caa_profile`、`caa_register`、`caa_unregister`、`caa_start`、`caa_stop`、`caa_relocate`、`caa_balance`、`caa_report`、`caa_stat` コマンドがあります。CAA の全リファレンス・ページのリストについては、`caa(4)` を参照してください。

コマンド行インタフェースは、リソース・プロファイル、処理スクリプト、リソース・マネージャと連携して動作します。

#### グラフィカル・ユー ザ・インタフェース

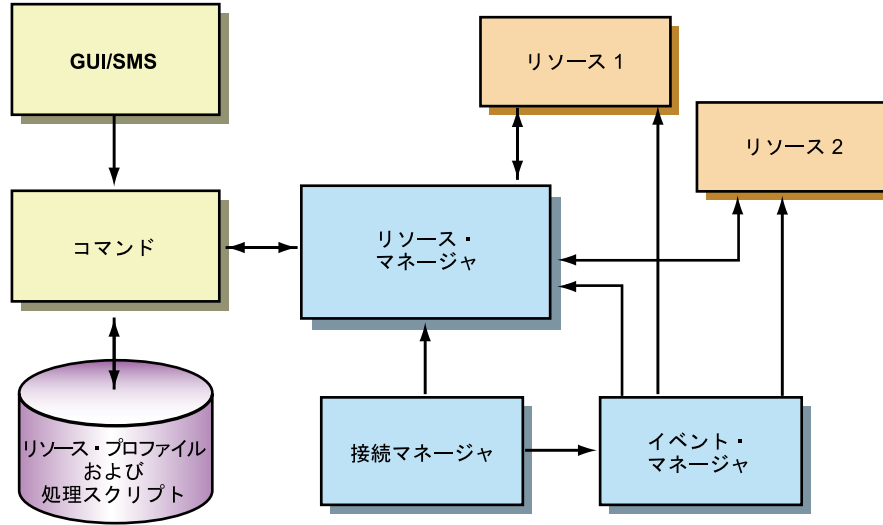
SysMan Menu および SysMan Station には、クラスター、クラスター・メンバ、CAA アプリケーションのシステム管理作業を実行するための、グラフィカル・ユーザ・インタフェース (GUI) があります。GUI を使用して CAA アプリケーションのシステム管理作業を実行する方法についての詳細は、`sysman(8)` と、SysMan Menu および SysMan Station のオンライン・ヘルプを参照してください。

CAA の GUI は、コマンド行インタフェースを呼び出して、リソース・プロファイル、処理スクリプト、リソース・マネージャと連携して動作します。

接続マネージャとイベント・マネージャは CAA サブシステムの一部ではありませんが、CAA サブシステムではこれらの機能を広い範囲で使用しています。

図 5-2 は、CAA のアーキテクチャを図示しています。

図 5-2: CAA のアーキテクチャ



ZK-1585U-AI

## 5.3 リソース

リソースとは、エンド・ユーザまたは他のソフトウェア構成要素にサービスを行う、クラスタのハードウェア構成要素またはソフトウェア構成要素です。リソースは、クライアントに対するサービスを高可用にするために CAA が使用する構成要素です。CAA では、次のタイプのリソースをサポートしています。

### アプリケーション

実行可能プログラム。アプリケーション・リソースでは、別のアプリケーション・リソースを含め、他のリソースに対する依存関係を持つことができます。アプリケーション・リソースを定義するリソース・プロファイルでは、これらの依存関係は必須の `REQUIRED_RESOURCES`、またはオプションの `OPTIONAL_RESOURCES` として定義されます。

リソースを必須リソースとして定義し、その必須リソースが利用できなくなった場合、CAA はそのアプリケーションを停止します。そして CAA は、その必須リソースを持つ他のメンバでアプリケーションを再起動しようとします。他のメンバがダウンしていたり、配置ポリシーによってそのメンバからのアプリ

ケーションの起動が禁止されているために、CAA が他のメンバでアプリケーションを再起動できない場合、アプリケーションは停止されます。CAA は、すべての必須リソースが利用できるようになるまで、そのアプリケーションを再起動しません。

オプションのリソースは、必須リソース、およびアプリケーションを起動する最適なシステムを判断するための再配置ポリシーと組み合わせて使用されます。オプション・リソースが使用できなくなっても、アプリケーションはフェイルオーバーされません。

## ネットワーク

ネットワーク・インタフェース。クラスタのメンバはすべて、任意のメンバに接続されている任意のネットワークに、間接的にアクセスできます。他のクラスタ・メンバ上で利用できるネットワーク接続を長時間に渡って利用するアプリケーションでは、クラスタ・インターコネクトへのトラフィックが増加する可能性があり、アプリケーションとクラスタの両方の性能を低下させることがあります。ネットワーク・リソースをアプリケーションの必須リソースとして定義することは、特定のネットワークに直接接続されているメンバでアプリケーションを実行させたい場合に便利です。

ネットワーク・リソースをアプリケーションの必須リソースとして定義していて、ネットワーク・インタフェース・アダプタに障害が発生した場合、CAA はリソースの再配置ができなければ、アプリケーションを再配置または停止します。

ネットワーク・リソースをアプリケーションのオプション・リソースとして指定すると、CAA はサブネットに直接接続されているメンバ上でアプリケーションを起動します。サブネット・アダプタに障害が発生すると、アプリケーションは、間接的にネットワークにアクセスする状態に戻ります。



テープまたはチェンジャ

テープ・ドライブまたはメディア・チェンジャ。  
テープ・リソースまたはメディア・チェンジャ・リソースをアプリケーションの必須リソースとして定義すると、そのアプリケーションは必ず、テープ・デバイスまたはチェンジャに直接接続されているクラスタ・メンバで実行されます。デバイスに障害が発生すると、CAA はアプリケーションを再配置しようとし、再配置ができなければ、アプリケーションを停止します。

テープ・リソースまたはメディア・チェンジャ・リソースをアプリケーションのオプションのリソースとして定義すると、CAA はそのアプリケーションを直接接続されているメンバで実行しようとしませんが、デバイスに直接接続されていないメンバでそのアプリケーションを実行することもあります。性能を最大限に向上させるには、テープ・デバイスに直接接続されているメンバで実行することが望まれます。

## 5.4 リソース・プロファイル

各リソースにはリソース・プロファイルがあります。このプロファイルでは、リソースの定義、その依存関係のリスト、CAA によるリソースの管理方法を定義しています。リソース・プロファイルは、キーワードと値のペアのリストが格納されたテキスト・ファイルで、`caa(4)` に説明があります。省略時の設定では、すべてのリソース・プロファイルは `/var/cluster/caa/profile` ディレクトリに置かれています。

CAA でリソースの監視と管理を行うには、`caa_register` コマンドを使用して、リソース・プロファイルを登録する必要があります。

以降の各項で、リソース・プロファイルの2つのタイプについて説明します。

- アプリケーション・リソース・プロファイル (5.4.1 項)
- 非アプリケーション・リソース・プロファイル (5.4.2 項)

### 5.4.1 アプリケーション・リソース・プロファイル

アプリケーション・リソースの場合、リソース・プロファイルにはアプリケーションのタイプ、名前、チェック間隔、監視のしきい値、リソースの

依存関係 (必須リソース), オプションのリソース, ホスト・メンバ・リスト, 配置ポリシ, 再起動試行, フェイルオーバー遅延, 自動起動およびアクティブ配置に関する設定値, 負荷分散を行う時間や, 処理スクリプトの名前が含まれます。いくつかのキーワードはオプションです。たとえば, 次の `named.cap` というリソース・ファイルのサンプルでは, アクティブ配置を設定していません。これは, メンバがブートしてクラスタに組み込まれた際に, アプリケーションの配置は再評価されないということです。

```
# cat named.cap
TYPE = application
NAME = named
DESCRIPTION = BIND Server
CHECK_INTERVAL =
FAILURE_THRESHOLD = 0
FAILURE_INTERVAL = 0
REQUIRED_RESOURCES =
OPTIONAL_RESOURCES =
HOSTING_MEMBERS =
PLACEMENT = balanced
RESTART_ATTEMPTS =
FAILOVER_DELAY =
AUTO_START =
ACTION_SCRIPT = named.scr
```

プロファイルおよびキーワードの各タイプについての詳細は, `caa(4)` を参照してください。また, アプリケーション・リソース・プロファイルの内容および作成方法についての詳細は, 『クラスタ高可用性アプリケーション・ガイド』 および `caa_profile(8)` を参照してください。

アプリケーション・プロファイルは, ユーザ定義属性を使用して拡張できます。これらの属性値は, アプリケーション・プロファイルまたは CAA コマンドのコマンド行で定義できます。これらの属性値は, 処理スクリプトで使用し, リソースの起動, 停止, チェック時における処理スクリプトの実行をカスタマイズできます。ユーザ定義属性はアプリケーション・タイプ定義ファイル内のすべてのアプリケーション・リソースに対して定義できます。詳細は, 『クラスタ高可用性アプリケーション・ガイド』 を参照してください。

この項の以降の部分では, 配置ポリシ, ホスト・メンバ, アクティブ配置, 障害しきい値, および障害間隔について, その概要を説明します。処理スクリプトについては, 5.5 節で説明します。

アプリケーションの配置ポリシーは、どこでアプリケーションを起動するかを決定します。サポートされているポリシーは、balanced、favored、restricted です。

balanced

CAA は、現在稼働しているアプリケーション・リソースが最も少ないメンバ上で、アプリケーション・リソースの起動または再起動を行います。オプションのリソースによる配置が最初に考慮されます。次に、実行中のアプリケーションが最も少ないホストが選択されます。この基準で優先されるクラスタ・メンバがない場合は、利用可能な任意のメンバが選ばれます。

favored

CAA は、リソース・ファイルの `HOSTING_MEMBERS` 属性のメンバ・リストを参照します。このリストにあって、必須リソースを満足するクラスタ・メンバだけが配置に適するメンバとして考慮されます。オプションのリソースによる配置が最初に考慮されます。オプションのリソースに基づいてメンバを選択できない場合、ホスト・メンバの順番により、そのアプリケーション・リソースを実行するメンバが決められます。ホスト・メンバ・リストにあるどのメンバも利用できない場合には、CAA は、実行しているアプリケーション・リソースが最少のメンバ上にそのアプリケーション・リソースを配置します。

favored 配置ポリシーを選択した場合は、ホスト・メンバ・リストを指定しなければなりません。

restricted

このポリシーは favored 配置ポリシーと似ていますが、利用できるメンバがホスト・メンバ・リスト中にない場合に、CAA がアプリケーション・リソースの起動や再起動を行わない点が異なります。restricted 配置ポリシーでは、手動でリソースを再配置しないかぎり、リストに載っていないメンバ上でリソースが実行されることはありません。

restricted 配置ポリシーを選択した場合は、ホスト・メンバ・リストを指定しなければなりません。

ホスト・メンバとは、アプリケーションの (a) 起動時に考慮するメンバ、または (b) 再配置時に考慮するメンバを優先順位の順に並べたものです。ホスト・メンバ・リストは、favored 配置ポリシーまたは restricted 配置ポリシーの場合にのみ使用されます。

アクティブ配置では、新規クラスタ・メンバがクラスタに追加されるか、またはリポートされると、CAA はアプリケーションの配置を再評価します。より高い優先度のクラスタ・メンバがクラスタに参加したときに、アクティブ配置がオンになっている場合、アプリケーションは現在のメンバ上で停止して、より優先度の高いメンバ上で再起動します。

時刻に基づいてアプリケーションをフェールバックさせたい場合は、アクティブ配置の代わりに REBALANCE プロファイル属性を使用できます。アプリケーションは、クラスタ・メンバがクラスタに復帰したときではなく、指定した時間に優先メンバに再配置されます。

障害しきい値および障害間隔時間は、繰り返し障害の発生しているアプリケーションを停止させるために、一緒に使用されます。障害間隔時間内にアプリケーションが何度も失敗すると、そのアプリケーションは再起動されません。これらの値は、アプリケーションのチェックが失敗した場合にのみ考慮され、最初の起動の際には考慮されません。

再起動試行回数は、1 つのクラスタ・メンバ上で、アプリケーションの起動試行が失敗したとみなされるまでに実行するアプリケーションの起動または再起動の最大試行回数を定義します。

#### 5.4.2 非アプリケーション・リソース・プロファイル

現在サポートされているリソースの他のタイプ (ネットワーク、テープ、メディア・チェンジャ) には、監視するリソースを定義し、障害しきい値および障害間隔時間を指定するリソース・プロファイルがあります。障害間隔時間の間に、非アプリケーション・リソースに何度も障害が発生すると、そのリソースの監視は停止します。

テープ・リソースおよびメディア・チェンジャ・リソースについては、デバイス名により監視するテープを定義し、ネットワーク・リソースについては、サブネットを定義する必要があります。

リソース・プロファイルの内容と作成についての詳細は、『クラスタ高可用性アプリケーション・ガイド』, `caa_profile(8)`, および `caa(4)` を参照してください。

## 5.5 処理スクリプト

処理スクリプトは、CAA がアプリケーションの起動、停止、チェックを行うために使用するコマンドのセットです。アプリケーション・リソースだけが処理スクリプトを持つことができます。処理スクリプトの名前は、アプリケーションのリソース・プロファイルに `ACTION_SCRIPT` 値として指定されています。

省略時の設定では、処理スクリプトは `/var/cluster/caa/script` ディレクトリに置かれていますが、クラスタ・ファイル・システム上の別のディレクトリに置くこともできます。処理スクリプトのファイル名は、`resource_name.scr` という形式をしています。

『クラスタ高可用性アプリケーション・ガイド』に、処理スクリプトの例があります。

機能としては、処理スクリプトは、ASE (Available Server Environment) スクリプト、および `/sbin/init.d` ディレクトリにあるシステム初期化スクリプトに似ています。

処理スクリプトには、アプリケーション・リソースの起動または停止が必要な場合に CAA コマンドから実行されるエントリ・ポイントが複数あります。`start` エントリ・ポイントは、`caa_start` と `caa_relocate` がアプリケーションの起動のために使用し、`stop` エントリ・ポイントは `caa_stop` と `caa_relocate` がアプリケーションの停止のために使用します。`check` エントリ・ポイントは、リソース・マネージャがアプリケーションが実行中であることを確認するために使用されます。

アプリケーション・リソース・プロファイルには、各処理スクリプトに対応するタイムアウト値が定義されています。処理スクリプトの実行がこの時間内に終わらなかった場合、CAA は起動失敗と見なし、アプリケーションを別のメンバで起動するか、完全な障害とします。

オプションで、処理スクリプトの標準出力をスクリプトを起動する CAA コマンドの標準出力にリダイレクトすることができます。

`caa_profile` コマンドと SysMan の一連のアプリケーションはどちらも、リソース・プロファイルを作成する際に簡単な処理スクリプトを作成するために使用できます。アプリケーションの起動、停止、およびチェック処理をカスタマイズするために、これらの処理スクリプトを編集しなければならないことがあります。

# 6

## クラスタ別名

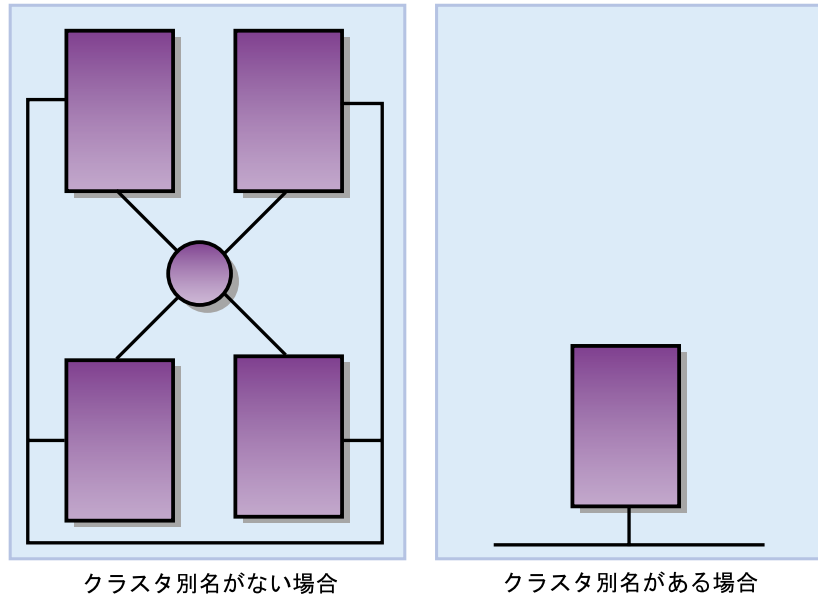
この章では、次の事項について説明します。

- クラスタ別名の一般的な概要 (6.1 節)
- クラスタ別名サブシステムの構成要素 (6.2 節)
- 省略時のクラスタ別名 (6.3 節)
- 1 つのクラスタで持つことができる別名の数 (6.4 節)
- 別名 IP アドレスの位置 (6.5 節)
- 別名 IP アドレスへのルーティング (6.6 節)
- `in_single` および `in_multi` サービス (6.7 節)
- 別名の属性 (6.8 節)
- サービス・ポートの属性 (6.9 節)
- vMAC サポート (6.10 節)
- NFS とクラスタ別名 (6.11 節)
- RPC サービスとクラスタ別名 (6.12 節)
- `ifconfig` 別名とクラスタ別名 (6.13 節)

### 6.1 クラスタ別名の概要

クラスタ別名は、クラスタ内のシステムの一部またはすべてを、TCP (Transmission Control Protocol) および UDP (User Datagram Protocol) アプリケーションからは 1 つのシステムとして見えるようにするための IP アドレスです。図 6-1 は、クラスタ別名があるクラスタ・システムと、クラスタ別名がないクラスタ・システムが、ネットワーク・クライアントからどのように見えるかを示しています。

図 6-1: クラスタ別名がある場合とない場合の、クライアントからのクラスタの見え方



ZK-1471U-AI

クラスタ別名を使用すると、クライアントは、サービスを受けるために特定のクラスタ・メンバに接続する必要がなくなります。クライアントがさまざまなサービスを1つのホストに要求できるのと同じように、クライアントはさまざまなサービスをクラスタ別名に要求できます。たとえば、単独のホストの場合と同じように、`telnet` や `rlogin` をクラスタ別名に対して実行できます。

1つのクラスタが、複数のクラスタ別名を持つこともできます。1つは、省略時のクラスタ別名です。この別名はクラスタのインストール時に作成され、この別名宛のパケットはすべてのメンバが受信できます。クラスタ管理者は、必要に応じて別名を追加できます。

クラスタ別名は、分散型で仮想的な、クラスタ単位のネットワーク・インタフェースと考えてください。その意味で、クラスタ別名は、1つの物理ネットワーク・インタフェースが複数のIPアドレスに対して応答する `ifconfig` 別名と、概念的に似ています。

クラスタ内の各システムは、所属したい別名に明示的に参加します。別名に参加したシステムは、その別名のメンバになります。クラスタ別名が仮想的



なネットワーク・インタフェース上のアドレスのようなのだとすると、別名への参加は、その別名インタフェースに対して `ifconfig up` コマンドを発行するようなものです。これで、そのメンバはその別名宛のパケットを受信できるようになります。

クライアントは、別名の IP アドレスに、TCP 接続要求または UDP メッセージを送信します。クラスタは、その要求またはメッセージを、その別名の現在のメンバであるクラスタ・ノードへ透過的にルーティングします。クラスタ内のホップでは、ネットワーク・ルーティングではなくクラスタ・インターコネクトが使用されます。

別名内に使用できないメンバがある場合、クラスタはそのメンバへのパケットの送信をやめ、その別名内のアクティブなメンバにパケットをルーティングします。別名内にアクティブなメンバが 1 つあれば、その別名は使用できます。

## 6.2 クラスタ別名の構成要素

クラスタ別名サブシステムの主な構成要素は、次のとおりです。

- クラスタ別名サブシステム `clua` のカーネル部分。ブート時にロードされる、構成可能なカーネル・サブシステムです。
- ユーザ・レベル・デーモンの `aliasd`。カーネルはこのデーモンと通信を行い、クラスタ別名に対するルーティングを管理します。別名デーモンは、クラスタ別名に対するルーティング構成を透過的に処理し、クラスタ別名へのホスト・ルート（および仮想サブネット上の別名アドレスに対するネットワーク・ルート）が必要であれば、自動的にそのメンバの `/etc/gated.conf.membern` ファイルに追加します。デーモンは、このファイルを `gated` の構成ファイルとして使用し（メンバの `/cluster/members/{memb}/etc/gated.conf` ファイルは使用しません）、`gated` を起動します。

`cluamgr` コマンドには、デーモンの動作を変更するオプションがあります。各クラスタ・メンバは、`aliasd` を実行します。

---

### 注意

---

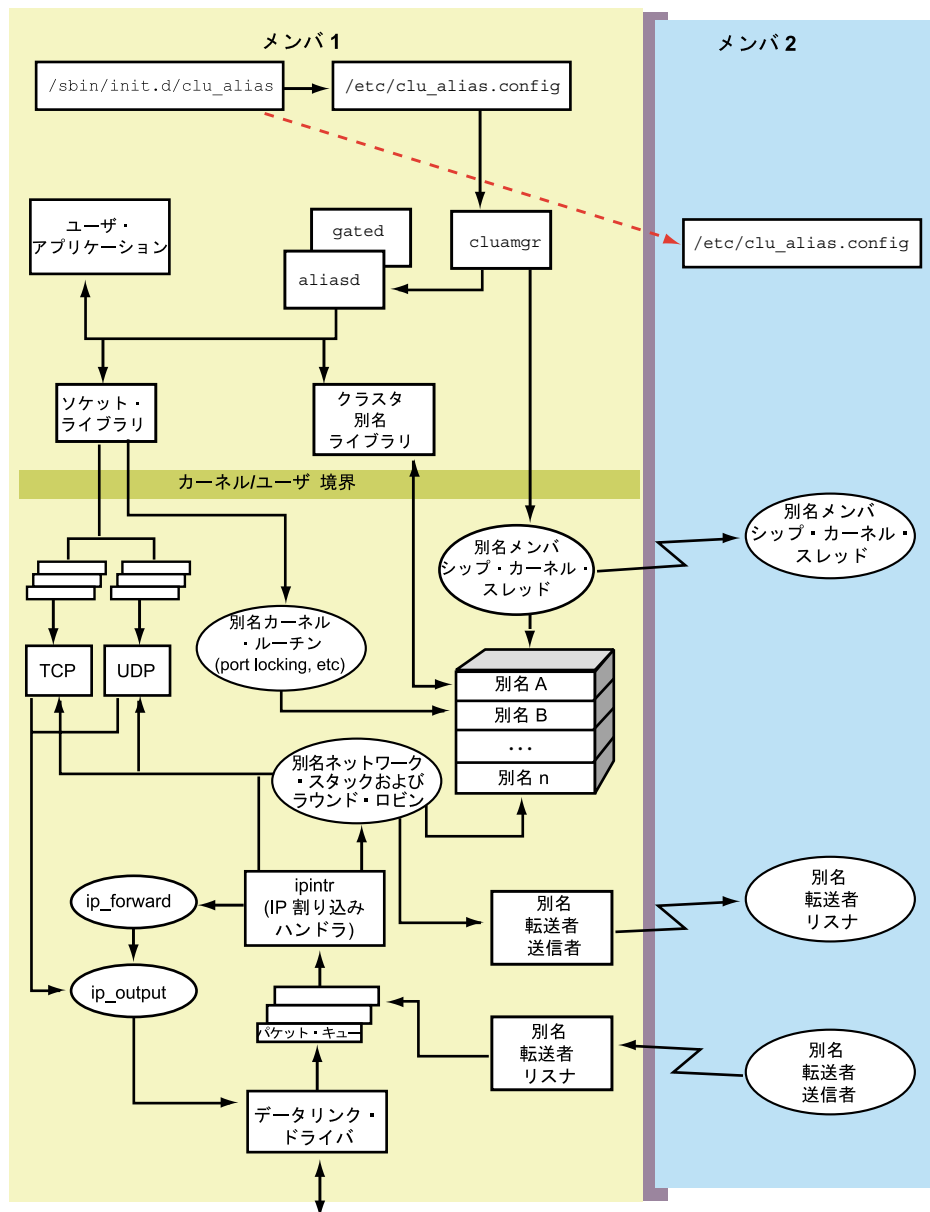
`aliasd` デーモンは、ルーティング情報プロトコル (RIP) のみをサポートします。

---

- コマンド行とグラフィカル・ユーザ・インタフェース (GUI) の両方で別名と別名属性を管理する，管理インタフェース。コマンド行インタフェースは，`cluamgr` コマンドです。GUI は，`SysMan Menu` からアクセスします。
- メンバ固有の別名構成ファイル，`/etc/clu_alias.config`。そのメンバの，省略時のクラスタ別名などの別名を構成する，`cluamgr` コマンドが含まれています。
- クラスタ単位のアプリケーション構成ファイル，`/etc/clua_services`。サービスで使用するポートに，別名に関連する属性を割り当てます。`/etc/clua_services` ファイルは，`/etc/services` ファイルのクラスタ別名拡張です。`clua_services` ファイルは，`services` 構文を拡張して，別名に関連する属性をポートに割り当てます。
- 省略時のクラスタ別名以外で NFS クライアントの使用できる別名が入っている，クラスタ単位のファイル `/etc/exports.aliases`。省略時の設定では，省略時の別名に宛てられた NFS の要求だけがクラスタで受け付けられます。このファイルを使うことにより，NFS サーバ名の別名を追加して使用することができます。このような機能は，たとえば，エクスポートされたファイル・システムが置かれているストレージにクラスタ・メンバの一部が直接接続していない場合に有効です。このような場合，そのストレージに直接接続しているメンバだけを対象にして別名を作成し，そのクラスタから NFS サービスを要求するときにはそのクラスタ別名を使うように NFS クライアント・システムのユーザに通知すればよいことになります。
- アプリケーション・プログラミング・インタフェース (API)，`libclua`。

図 6-2 は，クラスタ別名サブシステムの構成要素の機能概要を示しています。

図 6-2: クラスタ別名の機能概要



ZK-1551U-AI

### 6.3 省略時のクラスタ別名

省略時のクラスタ別名は、特別な別名です。クラスタには、インストール時に名前が設定されます。この名前は、`cluster_name` 属性の値と

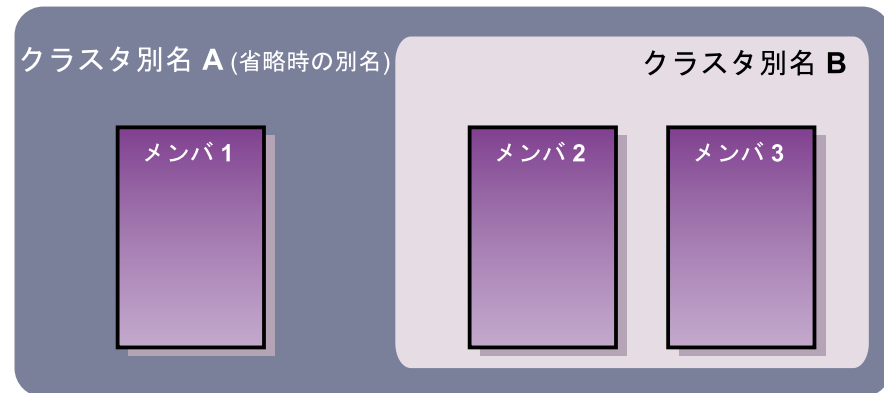
して `/etc/sysconfigtab` に格納されます。インストール・プロセスは、`/etc/hosts` にエントリを追加します。このエントリは、クラスタ名と、ユーザ指定の省略時のクラスタ別名の IP アドレスを対応付けます。たとえば、クラスタの名前が `deli` で別名の IP アドレスが `16.140.112.209` である場合、インストール・プロセスは次のエントリを `/etc/hosts` に追加します。

```
16.140.112.209    deli.zk3.dec.com    deli
```

各クラスタ・メンバは、省略時のクラスタ別名のメンバです。クラスタ・メンバを省略時のクラスタ別名に参加させるコマンド `cluamgr` は、各メンバの `/etc/clu_alias.config` ファイルにあります。ブート時に各クラスタ・メンバがその `clu_alias.config` ファイル内のコマンドを起動するため、すべてのクラスタ・メンバは自動的に省略時のクラスタ別名に参加します。

図 6-3 では、2 つのクラスタ別名を持つ 3 ノードのクラスタを示しています。省略時のクラスタ別名である別名 A にはすべてのメンバが属していますが、別名 B に属しているメンバは 2 つだけです。

図 6-3: 2 つの別名を使用するクラスタ



ZK-1443U-AI

## 6.4 クラスタごとの別名の数

クラスタをインストールしたのち、そのクラスタに必要な数だけの別名を定義することができます。clua サブシステムの `max_aliasid` 属性の値は省略時で 8 で、最大値は 102,400 です。この上限は、実際にはメモリ容量で制限されますが、この範囲にあれば実用上おそらく問題にはなりません。

多くのクラスタでは、省略時のクラスタ別名により、クラスタのクライアントに対して十分なアクセスを提供できます。別名の追加がクラスタにとって有益かどうかは、クラスタの対称性(ストレージとネットワーク)や、すべてのメンバでクライアントからのすべてのサービス要求を処理させたいかどうかによって異なります。別名の追加は、次のような場合に便利です。

- 異機種のクラスタ・メンバがあるために、一部のクラスタ・メンバでサービスした方が最高のサービスを行えるデバイスまたはアプリケーションがある場合。
- 要求およびパケットの内部転送を減らすために、サービスを一部のクラスタ・メンバに限定したい場合。
- クラスタによってエクスポートされたファイル・システムが置かれているストレージに、そのクラスタのメンバの一部が接続していない場合。この場合、このストレージに直接接続しているメンバだけを対象にした別名を使うことにより、クラスタ・インターコネクトを経由するトラフィックを減らすことができる。

省略時の別名をしばらく使った後で別名を追加した方がよいかどうか判断されるようお勧めします。多くの場合、省略時の別名だけで十分です。『クラスタ管理ガイド』に2つの別名を使って負荷分散を行うケースが説明されていますので参考にしてください。

## 6.5 別名 IP アドレスの位置

クラスタ別名の IP アドレスは(省略時のクラスタ別名の IP アドレスも含む)、クラスタのクライアントがアクセスできるネットワーク上になければなりません。つまり、クライアントからの要求をこのサブネットにルーティングできる必要があります。このため、クラスタ別名の IP アドレスを、クラスタ・インターコネクト(クラスタが内部通信用に使用するサブネット)に置くことはできません。

クラスタ別名のアドレスは、次の2種類のサブネットのどちらかにあります。

- |         |   |
|---------|---|
| 共通サブネット | 1 つ以上のクラスタが物理的なネットワーク・インターフェースで接続されたサブネット。<br><br>クラスタ別名に共通サブネットを使用すると、クラスタが接続されているローカル・エリア・ネットワークが1 つだけであり、そのネットワークが単一 |
|---------|---|

の IP アドレス・ドメインとして管理されている場合にうまく動作します。

共通サブネットでのクラスタ別名のルーティングは、プロキシ ARP (Address Resolution Protocol) のサポートに基づいて行われます。各別名に対して、1 つのクラスタ・メンバがその別名に対するプロキシ ARP マスタとして動作します。

#### 仮想サブネット

クラスタ別名のアドレスがどの物理インタフェースにも対応しないサブネットにある場合、そのクラスタ別名は仮想サブネット内にあります。仮想サブネットは、`gated` (ゲートウェイ・ルーティング・デーモン) によって、物理的なネットワークから見るすることができます。

`cluamgr` コマンドを使用して `virtual` オプションを別名アドレスに割り当てる場合、このコマンドが実行されるクラスタ・メンバは、別名へのホスト・ルートとネットワーク・ルートの両方を公開します。

同じ LAN 上の複数のクラスタは、同一の仮想サブネットを使用できます。

---

#### 警告

---

仮想サブネットは、その中に実際のシステムがあってはなりません。

---

サブネット・タイプの選択は、主に、クラスタが接続されている既存のサブネット (つまり、共通サブネット) にクラスタ別名用として利用できるアドレスが十分にあるかどうかで決まります。既存のサブネットでアドレスが利用できないようであれば、仮想サブネットの作成を検討してください。クラスタが複数のサブネットに接続されている場合に、仮想サブネットを構成すると、接続されているすべてのサブネットから一様に到達できるという利点がある、ということはあまり考慮する必要はありません。ただし、この利点は、実質よりもスタイルの問題です。クラスタ別名アドレスにどのタイプの

サブネットを使用するかについては、実際のところ大きな違いはないので、サイトに一番適したサブネットを使用してください。

サブネットは、そのタイプにかかわらず、クライアントからのパケットが別名アドレスにルーティングされるように構成しなければなりません。これらの別名アドレスが、クライアントから到達できない仮想サブネットまたは共通サブネット上にある場合、クラスタ別名を使用するサービスにはアクセスできません。

クラスタ別名アドレスは、ブロードキャスト・アドレスやマルチキャスト・アドレスであってはなりません。また、クラスタ別名アドレスは、クラスタ・インターコネクトで使うサブネット内のアドレスであってなりません。クラスタ別名に割り当てた IP アドレスが RFC 1918 で定義されている専用アドレス空間にある場合でも、`cluamgr -r resvok` コマンドを使って、別名サブシステムにその別名アドレスに至るルートを公開させなければなりません (`resvok` フラグの使用と、`/etc/rc.config.common` にエントリを追加してルートの公開がリブート後も保持されるようにする方法については、`cluamgr(8)` を参照してください)。

## 6.6 別名 IP アドレスへのルーティング

ここでは、別名に至るルートの公開方法と別名宛てパケットのルーティング方法について説明します。

- 別名へのルートの公開 ( 6.6.1 項)
- 共通サブネットでの別名のルーティング ( 6.6.2 項)
- 仮想サブネットでの別名のルーティング ( 6.6.3 項)
- 共通サブネットおよび仮想サブネット上の別名に対するルーティングのまとめ ( 6.6.4 項)
- 別名宛て接続要求およびパケットの受け付けとリダイレクション ( 6.6.5 項)
- ルーティングの例 ( 6.6.6 項)

ここで使用する次の用語は用語集に定義されています。これらの用語をよく知らない場合は、用語集でその意味を調べてから先に進んでください。

- ホスト・ルート
- ネットワーク・ルート

- プロキシ ARP

### 6.6.1 別名へのルートの公開

別名ルータは、クラスタの別名アドレスをネットワーク上に公開してその別名への着信パケットを受信する、クラスタ・メンバです。省略時の設定により、クラスタ・メンバはすべて、ブート時にそのクラスタ別名の別名ルータとして構成されます。どのクラスタ・メンバでも、別名へのホスト・ルートやネットワーク・ルートを公開するように構成することができます。

---

#### 注意

---

省略時の設定では、クラスタ・メンバは汎用ルータとして構成されないため、クラスタ別名宛のトラフィックしかルーティングしません。あるサイトでクラスタ・メンバを1つ以上構成して別名宛以外のトラフィックをルーティングさせるかどうかは、そのサイトのネットワーク管理者の仕事です。

---

クラスタ・メンバは、別名のメンバでなくても、その別名にルーティングすることができます。次の例では、クラスタ・メンバは `alias1` と `alias2` を指定するとともに `alias2` へ参加しています。その結果、クラスタ・メンバは `alias1` または `alias2` 宛てパケットをルーティングしますが、`alias2` 宛の要求およびパケットしか受信しません。

```
/usr/sbin/cluamgr -a alias=alias1  
/usr/sbin/cluamgr -a alias=alias2,join
```

### 6.6.2 共通サブネットでの別名のルーティング

省略時の設定では、クラスタ別名を指定またはクラスタ別名に参加したクラスタ・メンバは、`gated` を使って、その別名へ至るホスト・ルートを公開します。`aliasd` デーモンは `/etc/gated.conf.membern` を自動的に構成し、`/etc/clu_config.alias` にある情報に基づいてブート時にホスト・ルートを公開できるようにします。

あるクラスタ別名に対する ARP (Address Resolution Protocol) 要求には、同時に1つの別名メンバしか応答できません。応答できるメンバは、その別名のプロキシ ARP マスタです。このシステムに障害が起こると、別のメンバが選ばれてプロキシ ARP マスタの役割を引き継ぎます。



---

### 注意

---

プロキシ ARP は、共通サブネット (物理的なネットワーク) 上に構成されている別名アドレスだけに適用されます。

---

クラスタ別名アドレスが複数定義されている場合は、`rpri` 別名属性を使用して、別名アドレスごとに異なるクラスタ・メンバに最も高いルーティング優先順位を与えれば、受信負荷を少しだけ軽減することができます。別名が 1 つだけでしかも ARP ベースのルーティングを行っている場合は、一度に 1 つのメンバだけが別名ルータとして動作します (6.8 節にルータ優先順位の属性 `rpri` の説明があります)。

ただし、別名を知っているそれぞれのクラスタ・メンバは `gated` を使って、各クラスタ別名へのホスト・ルートを自分のネットワーク・インタフェースに設定します。ARP で使うインタフェース・ルートよりホスト・ルートが優先されるため、ルート・デーモンを実行しているクラスタと同じサブネット内であれば、どのクライアントも、ホスト・ルートを見ることができます。どのクラスタ・メンバがどの順番でブートされるかはその時々で異なるため、クライアントが異なれば、最初に見るクラスタ・メンバのホスト・ルートも異なってきます。したがって、それぞれのクライアントが、クラスタ別名へのルートとして異なるクラスタ・メンバを使うこともあります (これは、ホスト・ルート情報がすべてのクライアントに配布されているとは限らないためです。クライアントがクラスタより前にブートされている場合は、公開した最初のクラスタ・メンバを調べ、そこから得たホスト・ルートを使用します)。 `routed` や `gated` のようなルート・デーモンを実行していないクライアント・システムでは、ARP プロトコルを使ってクラスタ別名を調べ、そこへ向けて送信します。これはプロキシ ARP マスタを通してルーティングされます。

つまり、クライアントが別名へのルートを解決するために、ホスト・ルートではなく ARP を使用している場合、ARP に応答するクラスタ・メンバは、着信パケットが最初に送られて、トンネルするシステムになります。クライアントがホスト・ルートを使用する場合 (ルータは通常そうします)、`rpri` 設定は、着信パケットがたどるパスには影響を及ぼしません。

### 6.6.3 仮想サブネットでの別名のルーティング

省略時の設定では、クラスタ別名を指定またはクラスタ別名に参加したクラスタ・メンバは、`gated` を使って、その別名へ至るホスト・ルートを公開します。別名デーモン `aliasd` は、各クラスタ・メンバに対して `/etc/gated.conf.membern` ファイルを作成します。別名構成処理では、仮想サブネット内の各別名アドレスをホスト・ルートとして公開するために、この構成ファイルをあらかじめ変更しています。手動による変更は不要です。各メンバの別名デーモンが、そのメンバの `/etc/gated.conf.membern` ファイルを自動的に変更し、そのメンバ上の各ネットワーク・インタフェースを介して、各クラスタ別名ホスト・アドレスへのルートを公開します。

`cluamgr` コマンドで別名アドレスに `virtual` オプションを指定すると、クラスタ・メンバはその別名の存在する仮想サブセットへのネットワーク・ルートを公開します。ネットワークに仮想サブネットの位置を確実に知らせるためには、少なくとも 1 つ、可能ならばすべてのメンバが、各仮想サブネット内の少なくとも 1 つの別名に対して `virtual=t` オプションを指定して `cluamgr` を実行するようにしてください。

共通サブネットのルートの公開と同様に、ルーティングの負荷を複数のクラスタ・メンバに分散させることができます。ただし、どの程度分散できるかは、クライアントがどの順番でルートの公開情報を見るかによって異なります。

### 6.6.4 共通サブネットおよび仮想サブネット上の別名に対するルーティングのまとめ

表 6-1 に、省略時のルーティング・デーモン構成 (`gated` を制御する `aliasd`) で、共通サブネットおよび仮想サブネット上のクラスタ別名に対して公開されるルートのタイプを、まとめて示します。

表 6-1: 共通サブネットおよび仮想サブネット上の別名に対するルーティングのまとめ

サブネットのタイプ	アドレスのドメイン	プロキシ ARP	gated 経由のホスト・ルート	gated 経由のネットワーク・ルート
共通	クラスタが接続しているサブネット	a	b	×
仮想	物理的に接続していないサブネット (クラスタの「背後」に存在して見える)	×	b	c

<sup>a</sup>別名を指定したかまたは別名に参加したクラスタ別名に対して、そのサブネット上のどのインタフェースでも可能。

<sup>b</sup>別名を指定したかまたは別名に参加したクラスタ・メンバの各インタフェースごとに可能。

<sup>c</sup>別名を指定したかまたは別名に参加したクラスタ・メンバのうち、その仮想サブネット上の少なくとも 1 つのクラスタ別名に対して `cluamgr` のオプション `virtual=t` を指定したクラスタ・メンバの各インタフェースごとに可能。

#### 注意

クラスタ・メンバに対する省略時のルーティング・デーモン構成は `gated` デーモンで、そのメンバの `/etc/gated.conf.membern` デーモン構成ファイルを使用します。クラスタ・メンバは、クラスタ別名へのホスト・ルート (`virtual` が指定された場合には仮想ネットワークのネットワーク・ルートも) を自動的に公開します。標準の `gated` 構成ファイル (`/etc/gated.conf`) を使用する場合、別のルーティング・デーモン (`routed`) を使用する場合、またはいずれのルーティング・デーモンも使用しない場合、クライアントがクラスタ別名にアクセスできるように静的ルートを指定する必要があります。

詳細は、『クラスタ管理ガイド』を参照してください。

### 6.6.5 別名宛て接続要求およびパケットの受け付けとリダイレクション

通常のルーティングであれば、クラスタ別名に宛てられたパケットは必ず 1 つのクラスタ・メンバに到着します (1 つのパケットを複数のクラスタ・メンバが処理することはありません)。そのメンバは、次に示す情報および方法で、どの別名メンバがパケットの受信と処理を行うかを決めます。

- どの別名メンバが処理できる状態にあるか (別名サブシステムは `bind()` および `listen()` の呼び出しを監視している)

- ポート番号
- パケットのタイプ
- 加重ラウンドロビン・アルゴリズム

次の表で、パケットがクラスタ内でどのようにリダイレクトされるかを説明します。

新規の TCP/IP 接続	パケットを参照し、ターゲット・ポートに合ったメンバのリストを作成する。加重ラウンドロビン・アルゴリズムを使用して、受信待ちメンバのリストからメンバを選択する。選択したメンバにパケットを転送する。
既存の TCP/IP 接続	どの別名メンバがこの接続を所有しているかを調べる。そのメンバにパケットを転送する。
UDP	パケットを参照し、ターゲット・ポートに合ったメンバのリストを作成する。加重ラウンドロビン・アルゴリズムを使用して、受信待ちメンバのリストからメンバを選択する。そのメンバにパケットを転送する。 (クラスタ別名サブシステムで NFS over UDP がどのように処理されるかについての詳細は、6.11 節を参照。)
ICMP (ICMP パケットには、クラスタ別名コンテキストで処理しなければならないものもある)	パケットを参照し、そのパケットを処理するか他のメンバに転送するかを判断する。

一例として、マルチ・インスタンス・サービスに対する TCP 接続要求のルーティングを考えてみます。クライアント・システムがクラスタ別名 IP アドレスに対して TCP 接続を行い、クライアントとクラスタ・メンバ間にルータがあるとします。

- クラスタ内の 1 つのノードだけがクラスタ別名への ARP 要求に応答しますが、その別名のすべてのメンバはその別名へのホスト・ルートを公開します。ルータは RIP プロトコルを見て処理するので、ホスト・ルータを見て、おそらく ARP 要求に対してそれを優先的に使用します。ルータの別名アドレス用のルート・テーブルにあるクラスタ・ノードは、ARP 要求に応答するノードかもしれませんが、応答しないノード

ドかもしれませんが、ルータだけが (RIP と仮定して) そのクラスタ別名の単一のルート・テーブル・エントリを持ちます。Member1 が、ルータが持っている問題のクラスタ別名用のルート・テーブル・エントリにあると仮定します。

- Member2 に宛てたクライアントからの要求は、ルータのルート・テーブル・エントリのために、Member1 を経由します。

Member2 からのパケットは、Member1 を経由せず、クライアントへの最も効率的なルートで、直接ネットワーク・インタフェース・カード (NIC) から送られます。クラスタがサーバである場合のほとんどは、データ転送のほとんどが出力です。つまり、入力方向の短いクライアント要求はトンネルされ (クライアント → R → M1 → M2)、長いデータ応答は直接クライアントへ送られます (M2 → R → クライアント)。

入力トラフィックは余分のホップを伴いますが、出力トラフィックには余分のホップは必要ありません (完全に形成された mbuf パケットは、外部に転送されたのちに再度戻されるのではなく、Member2 上のネットワーク・スタックへのクラスタ・インターコネクトを経由してトンネルされるため、余分のホップによるオーバーヘッドは完全なネットワーク・ホップより少なくなります)。大量の入力データのあるアプリケーションは、入力要求が短くて、出力データ転送が多いアプリケーションよりも、余分なホップの影響がはっきり見えます。

割り当てにクラスタ別名を使用して負荷分散を行うが、入力の余分なホップさえ回避する必要がある特別なアプリケーションは、バインディング・エージェントとしてクラスタ別名を使用し、初期接続を受け付けるノードに再接続することができます (クラスタ対応アプリケーションについての詳細は『クラスタ高可用性アプリケーション・ガイド』を参照してください)。

- 別個の TCP 接続はそれぞれそのポート上で待機している任意のクラスタ・ノードに分散される可能性があります。サービスの特定のインスタンスが、複数の接続に渡って保持される必要のあるクライアント・コンテキストを保持する (つまり、アプリケーションは接続ごとに等べきでない) 場合、アプリケーションは、それらの接続に対して一意のポート番号を使用するか、またはシングル・インスタンス・サービスとして構成すべきです。単一のポート番号を使用するクラスタ単位のサービスは、クラスタ別名のメンバである任意のクラスタ・ノードから同じサービスを提供することができると考えられます。

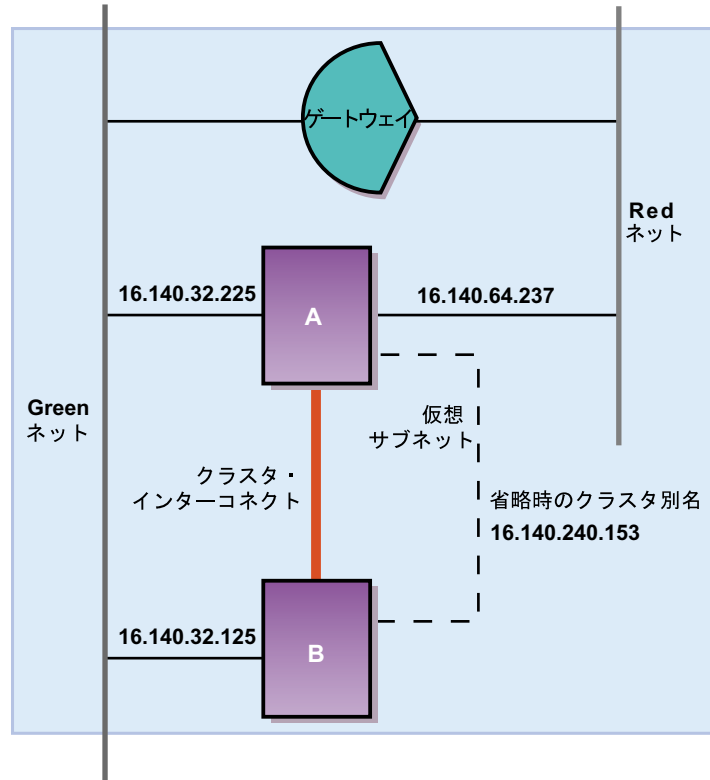
ルータもクライアントも、どのクラスタ・ノードがその接続に割り当てられたかを知る方法がないため (接続で使用するアドレスは変更されません。つまり、クラスタ別名アドレスです)、入力トラフィックは、ルータの別名用のルート・テーブルにあるメンバ・ノードを継続して使用します。メンバ・ノードへの接続をバインドする新和性情報は、クラスタ内でのみ知られています。

- Member1 に障害が発生して、別のメンバがプロキシ ARP マスタになると、新しいマスタはクラスタ別名 IP アドレスに関連付けられている MAC アドレスをルータにブロードキャストします。無償の ARP メッセージは、プロキシ ARP マスタシップを引き受けたメンバによって送信されます。仮想 MAC (vMAC) オプションを使用することもできます。このオプションを使用すると、クラスタは各クラスタ別名について一意の MAC アドレスを作成して使用することができます。vMAC アドレスは、必要に応じて、マスタからマスタへ渡されます (vMAC は、ルータが無償の ARP メッセージに応答しない場合にだけ必要です)。vMAC についての詳細は 6.10 節を参照してください。

#### 6.6.6 ルーティングの例

図 6-4 は、3 つのネットワーク、2 つの公用共通ネットワーク、1 つの専用仮想ネットワークのインタフェースを備えたクラスタを示しています。省略時のクラスタ別名の IP アドレスは、仮想サブネット上にあります。

図 6-4: 別名ルーティングの例



ZK-1472U-AI

適切な `cluamgr` を使って別名を構成すれば、各ノードの `gated` デモンが、接続しているすべてのネットワークで次の情報を公開します。

- そのアドレスへのホスト・ルート。ローカル・ノードで活用するための情報。
- 仮想ネットワークへのネットワーク・ルート。これらのネットワークより遠くにあるノードはこの情報を基に仮想サブネットの位置を知ることができる。

次の例では、`cluamgr` コマンドをホスト A とホスト B で実行して、仮想サブネット上のクラスタ別名へのルートを公開しています。

- `gated` に別名のホスト・ルートを公開させる場合  

```
# cluamgr -a alias=16.140.240.153
```
- `gated` に別名のホスト・ルートとネットワーク・ルート (16.140.240.0) を公開させる場合

```
# cluamgr -a alias=16.140.240.153,virtual=t
```

- `gated` に別名のホスト・ルートとネットワーク・ルート (16.140.240.0) を公開させ、その別名宛の接続要求とパケットを受信する場合

```
# cluamgr -a alias=16.140.240.153,virtual=t,join
```

## 6.7 in\_single および in\_multi サービス

クラスタ別名を通してアクセスされるサービス・ポートは、**in\_single** または **in\_multi** のいずれかとして定義されます。これらのサービス・ポート属性は、アプリケーションへのネットワーク要求のルーティングを決定しますが、アプリケーションが複数のメンバで同時に実行できるかどうかは決定しません。クラスタ別名サブシステムから見ると、次のような処理が行われます。

- サービスのポートを **in\_single** として指定した場合、1つの別名メンバだけが、そのサービスへの接続要求やパケットを受信します。そのメンバが利用できなくなると、クラスタ別名サブシステムは別名メンバの中から別の適格なメンバを選択して、すべての接続要求やパケットを受信させます。
- サービスのポートを **in\_multi** として指定した場合、別名サブシステムは接続要求とパケットを、別名内の適格なメンバすべてに分散します。

省略時の設定では、クラスタ別名サブシステムは、すべてのサービスを **in\_single** として処理します。クラスタ別名サブシステムがサービスのポートを **in\_multi** として処理するようにするには、`/etc/clua_services` 内か、`clua_registerservice()` 呼び出しで、ポートを **in\_multi** として登録しなければなりません。サービス・ポートの属性についての詳細は、6.9 節を参照してください。

ポートが **in\_multi** として指定されているサービスでは、クラスタ別名の利点を生かして、別名メンバ間に着信 TCP 接続要求と UDP パケットを分散させることができます。別名サブシステムは、別名メンバ間に要求およびパケットを分散させる加重ラウンドロビン・アルゴリズムによって、負荷のバランスを調整します。クライアントの要求に 응답できない別名メンバがある場合、クラスタ別名ソフトウェアは要求およびパケットを他の別名メンバに透過的に分散させます (アルゴリズムの重み付けの部分は、別名の各メンバがその別名に割り当てる選択の重み `selw` によって制御されます。選択の重みについての詳細は 6.8 節を参照してください)。



---

### 注意

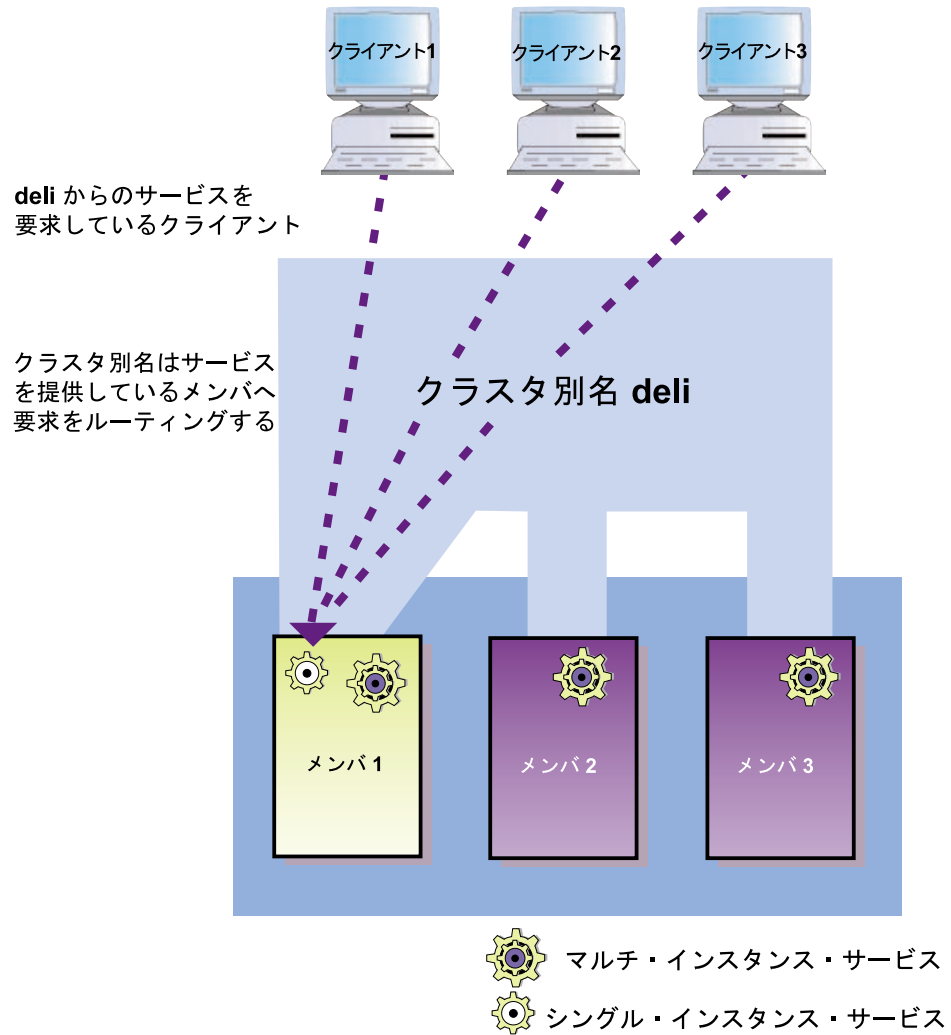
---

クラスタ別名と CAA は、補足し合う異なる機能を備えた独立したサブシステムです。CAA はアプリケーション制御ツールであり、クラスタ別名はルーティング・ツールです。CAA はアプリケーションが実行される場所を決定し、クラスタ別名はその場所へ到達する方法を決定します。CAA を使用してクラスタ内のルーティングを制御することはできません。また、クラスタ別名を使用して、クラスタ内のどこでアプリケーションを実行するかを制御することはできません。『クラスタ管理ガイド』に、クラスタ別名と CAA の相違点についての詳しい説明があります。

---

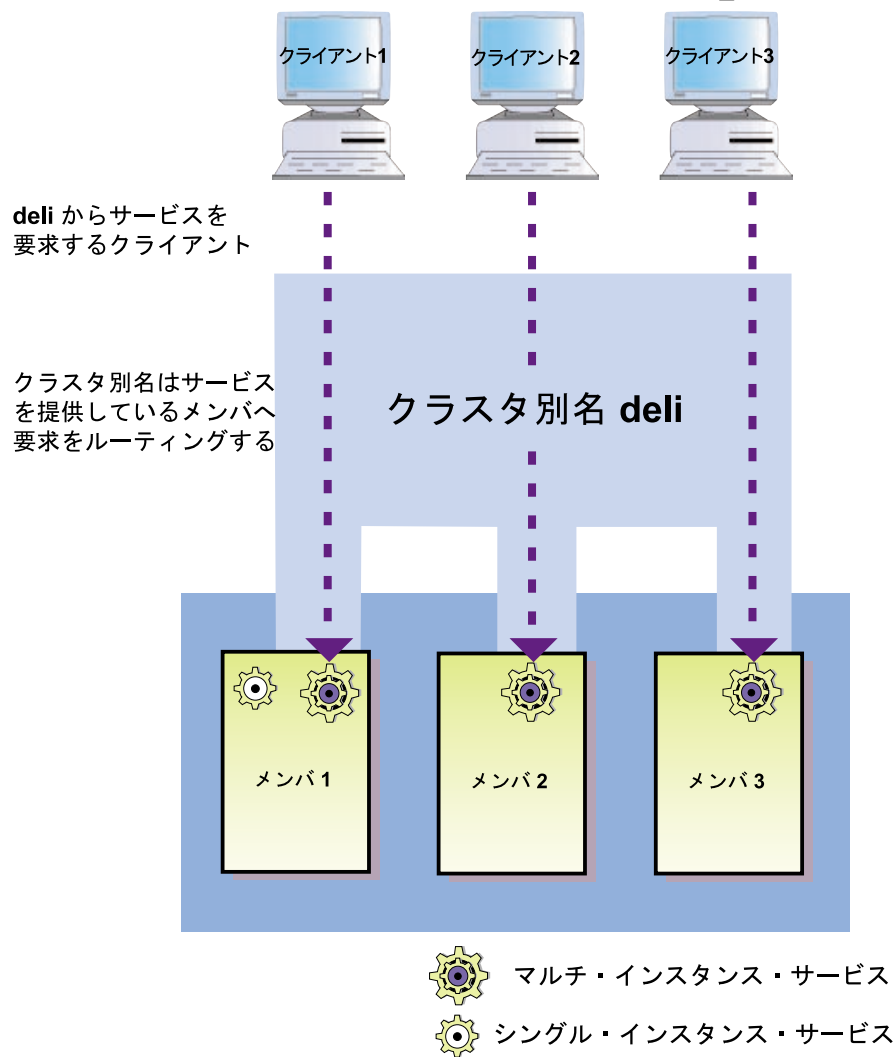
次の 2 つの図は、クラスタ別名サブシステムが `in_single` サービスおよび `in_multi` サービスのクライアント要求を分配する方法を示しています。`in_single` サービスの場合 (図 6-5)、要求はすべて、現在サービスを実行している別名メンバに送られます。`in_multi` サービスの場合 (図 6-6)、要求はすべての別名メンバに分散されます。

図 6-5: 省略時のクラスタ別名を通してアクセスされる in\_single サービス



ZK-1444U-AI

図 6-6: 省略時のクラスタ別名を通してアクセスされる in\_multi サービス



ZK-1445U-AI

## 6.8 別名の属性

ここでは、別名の属性について一般的に説明し、ルータの優先順位 (rpri)、選択の優先順位 (selp)、および選択の重み (selw) について詳しく説明します。

別名とその属性は、cluamgr コマンドと SysMan Menu によって管理されます。SysMan Menu は、必要に応じて cluamgr を呼び出します。

別名の属性は、メンバに固有です。クラスタ・メンバによって、別名の見え方が異なります。たとえば、クラスタ・メンバによっては、別名ヘルレーティングできても、その別名のメンバでないことがあります。また、他のクラスタ・メンバは、その別名ヘルレーティングが可能で、同時にその別名宛の要求またはメッセージの最終受信者となることもあります。同様に、別名の属性は別名にも固有です。クラスタ・メンバは、同時に2つの別名のメンバとなり、それぞれの別名に異なる選択の重みを割り当てることができます。そうすることでメンバ・システムは、選択の重みがより大きい方の別名に宛てられた接続を、もう一方の別名より多くの割合で受けることができます。

次の属性は、接続要求およびパケットの、別名のメンバへのルーティングと分配の方法を制御します。この説明は、`cluamgr(8)` の説明を書き換えたものです。このリファレンス・ページには、この他の別名属性についても説明があります。

ルータの優先順位	<p>ルータの優先順位 (<code>rpri</code>) は、共通サブネット上の別名に対する、プロキシ ARP ルータの選択を制御します (このオプションは、アドレスが仮想サブネット内にある別名に対しては関係ありません)。</p> <p>共通サブネット内の各別名について、その別名に対するルータ優先順位が最も高いクラスタ・メンバが、ARP 要求に応答します。このオプションは、どのメンバが別名のホストやネットワーク・ルートをブロードキャストするかを制御するわけではありません。</p> <p>クラスタに複数のクラスタ別名がある場合、ルータの優先順位を使用して、別名のプロキシ ARP 応答オーバーヘッドをクラスタ・メンバ間に分散させることができます。</p>
選択の優先順位	<p>選択の優先順位 (<code>selp</code>) によって、新しい接続要求を受け付ける別名のメンバのサブセットが決まります。</p> <p>選択の優先順位によって、別名のメンバが階層化されます。接続要求は、選択の優先順位の値が最も高いメンバ間に分散されます。別名にメンバが3つあり、2つが <code>selp=10</code> で1つが <code>selp=5</code> である場合、<code>selp=10</code> のメンバがどちらか使用できる</p>

かぎり、`selp=5` のメンバに接続要求やメッセージが渡されることはありません。

選択の優先順位の値を使用すると、特定のクラスタ別名のメンバに対して、フェイルオーバーの順序を設定できます。

選択の優先順位は `in_multi` サービスとして登録されているアプリケーションにだけ適用されます (選択の優先順位は、ラウンドロビン・プロセスでノードのセットを束縛し、ラウンドロビン・プロセスだけがマルチ・インスタンス・アプリケーションに適用されます)。 `in_single` サービスの接続要求の分散に対しては何の影響も及ぼしません。

## 選択の重み

選択の重み (`selw`) を使うことで、別名のどのメンバに接続を最も多く受けさせるかを、簡単でしかも固定的に制御することができます。

選択の重みは、`selp` 値が同じ次の別名メンバに接続が渡されるまでにそのメンバに渡される接続の数 (平均) を示します。 `selp` の値により、要求やメッセージを受け取るメンバを選択する順序が決まります。 `selw` の値により、選択された後に 1 つのメンバが受け取る要求またはメッセージの数が決まります。

ノード A がノード B より処理能力があって、ノード B より 50 % 多く接続を処理できる場合、たとえば、ノード A 上の別名に `selw=3` を、また、ノード B 上の別名に `selw=2` をそれぞれ割り当てます。

選択の重みは、`in_multi` サービスとして登録されたアプリケーションにのみ適用されます。

`in_single` サービスのトラフィックはすべて、そのサービスを実行しているクラスタ・メンバに渡されなければなりません。

選択の重みとルーティングの優先順位は、負荷分散に関する 2 つの問題を解決します。選択の重みは、クラスタ内でのアプリケーションのオーバーヘッド

の負荷分散を行い、ルータの優先順位は、クラスタ内でのプロキシ ARP 応答のオーバーヘッドの負荷分散を行います。

一般には、省略時の優先順位で十分な性能が得られます。選択の重みが役立つのは、大規模なシステムと小規模なシステムが混在するクラスタ内でアプリケーションの負荷分散を行う場合です。

## 6.9 サービス・ポートの属性

`/etc/clua_services` ファイルはすべてのクラスタ・メンバによって読まれる共用ファイルです。このファイルの概念と構文は、`/etc/services` ファイルに似ています。`clua_services` ファイルにより、別名に関連する属性と、サービスに使用するポート番号とを対応させることができます (アプリケーションのソース・コードでは、`clua_registerservice()` 関数が同じ目的で使用されます)。ポート割り当てが固定のサービスは、`/etc/clua_services` 内にエントリを置くことができます。

以下の属性を、サービスのポートに対応させることができます。

### **in\_single**

クラスタ別名から見ると、一時点では 1 つのクラスタ・メンバのみで実行され、アクティブなサービスの障害時には、他のメンバ上のそのサービスのインスタンスにフェイルオーバーするサービス。ここでいう「アクティブ」は、クラスタ別名宛のメッセージのみに関連します。サービスのインスタンスはすべて、`in_nolocal` 属性が設定されていないかぎり、ノードのローカル IP アドレスに対しては常にアクティブです。各サービスがアプリケーションのポートにバインドされるたびに、最初のインスタンスがその別名に対してアクティブになり、その他は非アクティブになります。アクティブなサービスに障害が発生すると、非アクティブなサービス・デーモンのいずれかがアクティブとしてマークされます。

`clua_services` ファイル内で `in_multi` と明示されていないポートや、`clua_registerservice()` 関数の呼び出しで `in_multi` として登録されていないポートは、`in_single` として処理されます。

**in\_multi**

2 つ以上のクラスタ・メンバで同時に実行できるサービスを示します。UDP を使用するサービスの場合、各パケットは別の別名メンバに転送されることがあります。TCP を使用するサービスの場合、各接続はいずれかの別名メンバにバインドされます。ただし、同じクライアントからのそのサービスへの別の接続は、別の別名メンバ上で確立されることがあります。

`in_multi` サービスは、`/etc/clua_services` ファイル内か、`clua_registerservice()` 関数によって、明示的に登録しなければなりません。

**in\_noalias**

別名アドレスへの接続要求を受け付けないポートであることを示します。

**in\_nolocal**

別名でないアドレスへの接続要求を受け付けないポートであることを示します。TCP の場合、ポートは接続を受け付けません。UDP の場合、ポートはメッセージをドロップします。

**out\_alias**

ポートがデスティネーションとして使用されるときに、ソース・アドレスとして省略時のクラスタ別名を使用することを示します。通常、発信接続（または UDP メッセージ）では、クライアントが実行されているクラスタ・メンバのローカル IP アドレスを使用します。クラスタからの発信トラフィックのソース・アドレスとして、クラスタ別名アドレスを使用した方が便利な場合があります（たとえば、認証が簡単になる）。`telnet` や `login` などの標準のインターネット・サービスには、省略時のクラスタ別名を発信パケットのソース・アドレスとして使用するものがあります。

`out_alias` 属性は、接続（UDP ではなく TCP）がクラスタから開始された場合にのみ適用されます。つまり、クラスタがクライアントとなります。クラスタ・メンバ上で実行しているプロセスが発信接続を開始し、クラスタの `/etc/clua_services` ファイ

ル内でデスティネーション・ポート (クラスタ内になり側の接続を表すポート) に `out_alias` のフラグが付けられている場合、接続のソース・アドレスとして省略時の別名を使用します。

発信トラフィックが UDP 送信である場合にも、同じことが言えます。これは、各送信を小さな接続と見なすことができるためです。

### **static**

このポートが、動的ポートとして割り当てられないことを示します。このオプションは、ブート時に必ず起動される既知のマルチ・インスタンス・ネットワーク・サービスが使う 512 ~ 1024 のポートに割り当てられます。

`out_alias` 属性を除き、これらの属性は、どのクラスタ別名を通してアクセスされるサービスにも適用されます。クラスタからの接続のみに適用される `out_alias` 属性は、省略時のクラスタ別名に固有の属性です。

`in_multi`、`in_single`、`in_noalias` 属性は、互いに排他的です。`in_nolocal` と `in_noalias` 属性は、互いに排他的です。これらの属性の使用方法についての詳細は、`clua_services(4)` および `clua_registerservice(3)` を参照してください。

## **6.10 vMAC サポート**

クラスタ別名 IP アドレスが共通サブネット内で構成されている場合、そのサブネット内のクラスタ・メンバが別名のプロキシ ARP マスタとして動作し、その別名に宛てられたローカルな ARP 要求に応答します。別名の他のメンバがプロキシ ARP マスタを引き継いだ場合は、その新しいマスタが無償の (gratuitous) ARP パケットをブロードキャストし、その別名の IP アドレスに関連付けられているハードウェアのメディア・アクセス制御 (MAC) アドレスを他のシステムに通知します。他のローカル・システムは、この新しいクラスタ別名と MAC との関連を反映するように ARP テーブルを更新します。

ただし、このブロードキャスト・パケットは、無償の ARP パケットを理解しないシステムにとっては問題となります。このようなシステムでは、ARP テーブルの通常のタイムアウト間隔が経過するまで、クラスタ別名と MAC の関連付けに変更があったことを認識しないまま、無効となった MAC アドレス



へ別名のトラフィックを送り続けます。この問題の解決方法は、各クラスタ別名の仮想ハードウェア・アドレス (vMAC アドレス) を提供することです。

仮想 MAC アドレスは、各別名 IP アドレスに対して自動的に作成される一意のハードウェア・アドレスです。別名 vMAC アドレスは、必要に応じて、クラスタ別名プロキシ ARP マスタの移動に合わせてノードからノードへ移動します。どのクラスタ・マスタが別名に対してプロキシ ARP としてサービスを行っているかには関係なく、その別名の vMAC アドレスは変わりません。

クラスタ別名に対する vMAC サポートを有効にする方法については、『クラスタ管理ガイド』を参照してください。

## 6.11 NFS とクラスタ別名

クラスタが NFS サーバとして構成されている場合、NFS クライアントの出す要求は、省略時のクラスタ別名または `/etc/exports.aliases` の中に指定されている別名へ送る必要があります。宛先としてクラスタ・メンバを個別に指定して NFS マウントを要求すると拒否されます。

出荷された時点で NFS クライアントが使用できる別名は、省略時のクラスタ別名だけです。これ以外の別名を NFS クライアントが使用できるようにする場合は、その別名を `exports.aliases` ファイルで指定します。この機能は、クラスタの一部のメンバがエクスポートされたファイル・システムのあるストレージに直接接続していない場合に有効です。その場合、ファイル・システムに直接接続しているメンバだけを別名に登録することにより、NFS 要求のサービスに必要な内部のルーティング・ホップ数を減らすことができます。

以下に、TCP と UDP NFS トラフィックの観点から、クラスタ別名サブシステムと NFS がどのように関わるかを説明します (できれば、ネットワーク・トランスポートに UDP を使用するようお勧めします)。説明では、クライアントがクラスタに対して行う最初のアクションとして、クラスタのエクスポートしたファイル・システムをマウントするものとします。また、すべてのクラスタ・メンバが共通ネットワークとストレージの両方に接続しているものとします。

- ネットワークからのパケットの受信 (6.11.1 項)
- マウント要求 (6.11.2 項)
- NFS over TCP (6.11.3 項)

- NFS over UDP ( 6.11.4 項)

### 6.11.1 ネットワークからのパケットの受信

TCP または UDP を使用して NFS の要求を送る場合、その要求は省略時のクラスタ別名、または、`/etc/exports.aliases` の中に名前の指定されている別名へ送られます。

省略時の設定では、クラスタ・メンバは、自分が指定したかまたは参加している各別名へのホスト・ルートを公開します。クライアントは、通常、最初に取得した別名へのホスト・ルートをキャッシュに置きます。そのため、そのルートが利用できる限り、クライアントは同じ別名宛のパケットをすべて同じクラスタ・メンバへ送ることになります。

着信方向では、すべてのパケットがこのメンバを通りますが、応答の送信方向では必ずしも通りません。そのため、応答パケットを伝送路に送り出すクラスタ・メンバはソース・アドレスとしてそのクラスタ別名アドレスを挿入します。その結果、クライアントは、パケットを送り出したその宛先別名から応答パケットを受信することができます。

### 6.11.2 マウント要求

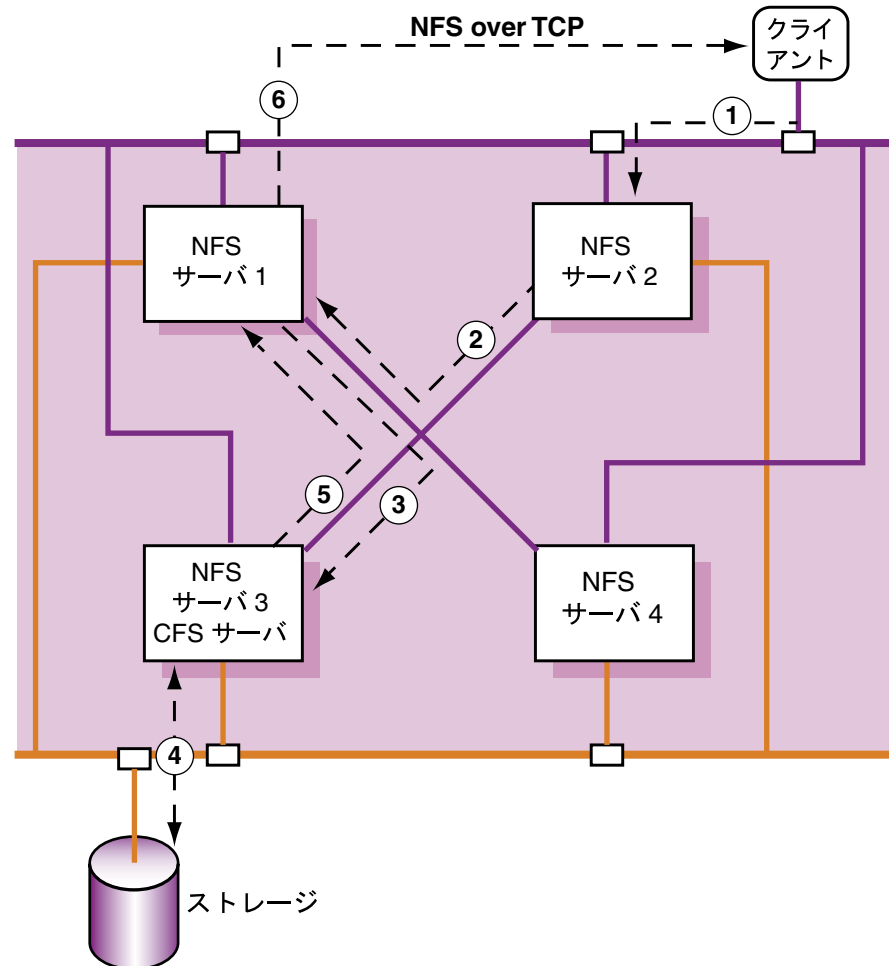
マウント・デーモン `mountd` は、UDP および TCP のマウント要求を受け付けて処理するマルチ・インスタンス・サービスです。どの `mountd` インスタンス・サービスにマウント要求を処理させるかは、クラスタ別名サブシステムが決めます。マウント要求を処理するノードと NFS パケットを受信して処理するノードが同じノードになることもありますが、基本的には両者に関係はありません。

### 6.11.3 NFS over TCP

TCP の接続要求は、別名のラウンドロビン・アルゴリズムと別名の選択重みを基にして割り当てられます (接続要求の中にファイル・システムの情報がないため、クラスタは、その時点でファイル・システムの CFS サーバになっているメンバにその要求を送れません)。クライアントからそのファイル・システムへ向けて送られてくるそれ以降の TCP NFS パケットは、その接続に割り当てられているメンバが引き続き処理します。この関係は、ファイル・システムが再配置されても変わりません。

図 6-7 とその後にある流れの説明を読みながら，NFS TCP 接続のパスをたどってください。

図 6-7: NFS over TCP



ZK-1801U-AI

1. クライアントにルートがキャッシュされているので，NFS の要求は，ほとんどの場合，マウント要求のときと同じメンバを通ります。
2. TCP の接続が最初の要求ですでに確立されているので，伝送路からパケットを受信したメンバは，自動的にそのパケットをその接続を処理する NFS サーバのメンバへ回します (伝送路からパケットを受信したノードでは，CFS サーバの検索は行われません)。

---

注意

---

以下の手順では、NFS サーバとファイル・システムの CFS サーバが異なる場所にあると想定しています。

---

3. NFS サーバは、インターコネクトを介して、CFS サーバになっているメンバへ CFS 要求を送ります。
4. CFS サーバがストレージの入出力を処理します。
5. CFS サーバは、インターコネクトを介して、NFS サーバ・メンバへ結果を返します。
6. NFS サーバ・メンバがクライアントに応答します (ソース・アドレスとして別名アドレスを使用)。

#### 6.11.4 NFS over UDP

UDP NFS パケットは、ファイル・システムのサービスを行っているクラスタ・メンバへリダイレクトされます。そのファイル・システムに対する UDP NFS トラフィックはすべて、そのメンバ上で処理されます。別のクラスタ・メンバがそのファイル・システムの CFS サーバになると、UDP パケットは新しいサーバにトンネルされます。ファイル・システムに対する CFS サーバの位置が変われば、UDP パケットは必ず、その新しい位置へ送られます。

---

注意

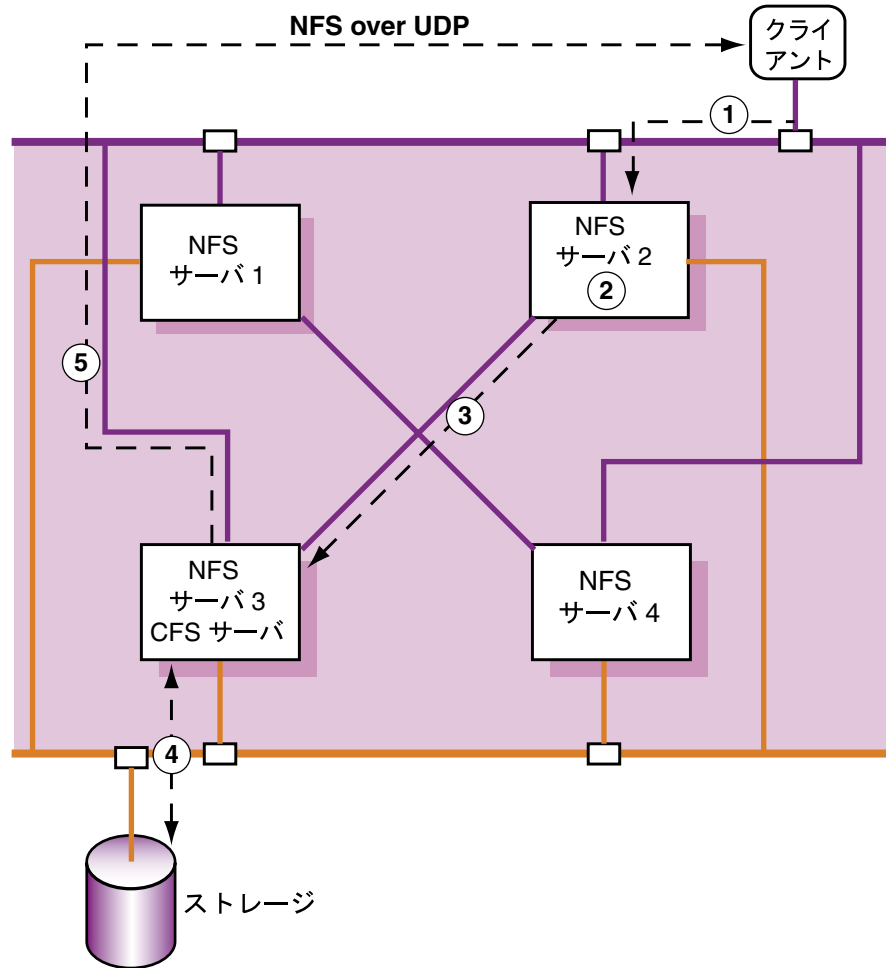
---

PC など一部のクライアントは、NFS サーバを探す際に UDP 要求をブロードキャストします。クラスタは、省略時のクラスタ別名の IP アドレスを返して、これらの要求に応答します。これにより、その後の NFS クライアントの要求は省略時のクラスタ別名へ確実に送られます。

---

図 6-8 とその後にある流れの説明を読みながら、NFS over UDP で NFS 要求の通るパスをたどってください。

図 6-8: NFS over UDP



ZK-1800U-AI

1. クライアントにルートがキャッシュされているので、NFS の要求は、ほとんどの場合、マウント要求のときと同じメンバを通ります。
2. NFS UDP パケットに NFS の要求に必要なデータがすべて入っているので、伝送路からパケットを受信したメンバ上のクラスタ別名ソフトウェアは、どのファイル・システムにそのファイルがあるかを調べ、さらに、CFS コールアウトを実行して、どのメンバがそのファイル・システムの CFS サーバになっているかを知ることができます。

---

#### 注意

---

以下の手順では、CFS サーバがそのパケットの宛先となっているクラスタ別名のメンバであると想定しています。

---

3. パケットは、そのファイル・システムの CFS サーバでもある NFS サーバへトンネルされます。CFS サーバは、要求を自分のところで処理できます (TCP より UDP の方の性能がすぐれています。これは、UDP パケットがインターコネクトを 1 度しか通らないためです。また、NFS サーバが同時に CFS サーバにもなっているため、CFS の入出力処理で余分な通信が発生することはありません)。
4. NFS サーバと CFS サーバの両方になっているメンバが入出力を処理します。
5. NFS サーバと CFS サーバの両方になっているメンバがクライアントに直接応答します (ソース・アドレスとして別名アドレスを使用)。

---

#### 注意

---

そのファイル・システムの CFS サーバが別名のメンバでなければ、受信ノードは、TCP 接続の要求を処理する場合と同様に、そのパケットをラウンドロビンで次へ回します。この場合、UDP の性能は TCP の場合より悪くなります。それは、TCP の場合、クライアントから入ってくるすべてのパケットがそのコネクションをサービスするノードへトンネルされ、結果として、同じクライアントからの同じファイルへの入出力がすべて同じノードで処理されるからです。これに対して UDP の場合は、CFS サーバがその別名のメンバでなければ、同じファイルのそれぞれの入出力要求が、異なるクラスタ・メンバで処理されることになります。この場合、CFS クライアントはデータをキャッシュすることができないため、お互いのキャッシュ内容を無効にしあって、CFS サーバ・ノードへそのたびに書き込みに行く結果になります。

---

## 6.12 RPC サービスとクラスタ別名

RPC サービスでは、`clusvc_getcommport()` 関数または `clusvc_getresvcommport()` 関数を呼び出して、ポートへのバインドを

行います (ポート番号 0 ~ 1023 の予約済み (特権) ポートへのバインドを行う場合は、`clusvc_getresvcommport()` を使用します)。どちらの関数も `clua_registerservice()` を呼び出して、ポートに `CLUASRV_MULTI` (`in_multi`) 属性を自動的に設定します。

`clusvc_getcommport()` および `clusvc_getresvcommport()` 関数は、次のような状況で使います。

- RPC サービスが、既知のポートを使用しない (既知のポートを使用するサービスでは、`/etc/clua_services` 内に `in_multi` エントリを置くことができる)。
- 1 つのクラスタ上で、RPC サービスの複数のインスタンスを実行できる。
- RPC サービスへの要求は、クラスタ別名宛てに行われる。これにより、サービスのインスタンス間で負荷のバランスがとれる。

これらの 2 つの関数により、クラスタ別名でアクセスされる RPC アプリケーションを複数のクラスタ・メンバ上で実行できるようになります。この関数により、RPC アプリケーションの各インスタンスが同じ共通ポートを使用するだけでなく、アプリケーションがマルチ・インスタンスの別名アプリケーションであることを `portmap` デーモンに通知します。

これらの関数を使用せずにポートにバインドした場合でも、アプリケーションのインスタンスを複数実行することはできます。ただし、クラスタ別名宛の要求を受信するのは、そのインスタンスだけです。

## 6.13 ifconfig 別名とクラスタ別名

TruCluster Server バージョン 5.0 より前の TruCluster 製品では、`asemgr` コマンドを使用して、アプリケーションのフェイルオーバを制御していました。`asemgr` コマンドは `ifconfig` コマンドを実行して、必要に応じて IP 別名を作成していました。クラスタ別名サブシステムはクラスタ単位で別名の作成と管理を行うため、アプリケーションのフェイルオーバ時に `ifconfig` を使用して IP 別名を明示的に確立したり削除したりする必要はなくなりました。

クラスタ別名アドレスは、IP アドレスとサービスが 1 対 1 になるようには設計されていません。クラスタ別名は、クラスタ全体を表すアドレス (またはクラスタのサブセットを選んで特定の別名を定義したもの) であり、その設計概念の中心となるのは、複数のコピーを同時に走らせることのできるアプリケーションです (マルチ・インスタンス・サービス)。シングル・インスタン

ス・サービスが必要な場合は、CAA で最適に構成することにより、サービスのフェイルオーバを一層簡単に管理することができます。

ASE サービスの知識があれば、`ifconfig alias` を使い、CAA スクリプトでアプリケーションに固有なインタフェースの別名アドレスを定義することができます。こうして定義した別名はクラスタ別名に依存していないため、競合は起こりません。

省略時のクラスタ別名を使用する利点は、クラスタ内でアプリケーションを移してもアプリケーション・アドレスを移行させる必要がないということです。これは、すべてのアプリケーションが同じアドレス (省略時のクラスタ別名) を使っているということだけでなく、そのクラスタ別名のコードによって、アプリケーションがクラスタ内のどこで動作しているかが常に分かるからです。さらに、アプリケーションが (広範囲にわたる複数メンバで同時に) マルチ・インスタンスを動作させた場合、クライアントは、同じクラスタ別名を使用することにより、複数のメンバが関係することを意識しないですべてのインスタンスに透過的にアクセスすることができます。

ASE スタイルのサービス・インタフェースごとの別名 (スクリプトで定義し、スクリプトでサービスと一緒に移行したもの) を使用する利点の 1 つは、トラフィックが、常に、サービスを実行しているノードへ直接ルーティングされるということです (クラスタ別名の場合、トラフィックは、かなりの頻度でクラスタ内を 1 ホップかかって転送されます)。サービスごとにアドレスを定義してそれを手動で移行するほどこのホップを重要視するかどうかは、アプリケーションのスループット要求で決まります。



---

## クラスタ・インターコネクト

クラスタには、すべてのクラスタ・メンバを接続するための専用クラスタ・インターコネクトが必要です。このインターコネクトは、クラスタ・メンバ間の専用通信チャネルとして使用されます。クラスタ・インターコネクトのハードウェアには Memory Channel または専用の LAN を使用できますが、同時にはいずれか一方しか使えません。

この章では、クラスタ・インターコネクトの目的、その使用、そのトラフィック制御メソッド、および使用するインターコネクトの種類を決める方法について説明します。具体的には、次の事項について説明します。

- クラスタ・インターコネクトを経由するストレージ・トラフィックの制御 (7.1 節)
- クラスタ・インターコネクトを経由するアプリケーション・トラフィックの制御 (7.2 節)
- クラスタ・インターコネクトを経由するクラスタ別名トラフィックの制御 (7.3 節)
- クラスタ・サイズとクラスタ・インターコネクト・トラフィックの影響の理解 (7.4 節)
- クラスタ・インターコネクトの選択 (7.5 節)
- Memory Channel インターコネクトの理解 (7.6 節)
- LAN インターコネクトの理解 (7.7 節)

一般に、クラスタ・インターコネクトは次のような高レベルの機能に対して使用します。

- ヘルス・メッセージ、ステータス・メッセージ、および同期メッセージ  
接続マネージャはこれらのメッセージを使用して、クラスタとそのメンバの状態を監視するとともに、クラスタ内でメンバ数とアプリケーションの配置を調整します。このタイプのメッセージ・トラフィックはメンバ数が変化するとき (たとえば、メンバがクラスタに追加されたり、削

除されたりするとき)に増加しますが、クラスタが安定状態にあるときはほとんど発生しません(詳細は、7.4 節を参照)。

- 分散ロック・マネージャ (DLM) のメッセージ

TruCluster Server は DLM を使用して共用リソースのアクセスを調整します。ユーザ・アプリケーションも、DLM API (アプリケーション・プログラミング・インタフェース) ライブラリを介して、この調整機能を利用することができます。このようなロック機能の調整に必要なメッセージは、クラスタ・インターコネクトを使って送信されます。アプリケーションがこの機能を頻繁に使用しても、クラスタ・ソフトウェア自身が生成する DLM トラフィックはわずかです。

- リモート・ファイル・システムへのアクセス

TruCluster Server ソフトウェアでは、すべてのクラスタ・メンバがストレージ・デバイスを等しく利用できるようになっています。つまり、あるメンバの専用ストレージ・バスに存在するストレージでも、すべてのクラスタ・メンバからアクセスすることができます。他のクラスタ・メンバが行うこのストレージのファイル・システムに対する読み書きは、クラスタ・インターコネクトを経由して送信されます。共用ストレージにあるファイルへの入出力要求(特に読み取り)は、可能であれば、クラスタ・インターコネクトを迂回してストレージに直接送られます。クラスタにファイル・システムとストレージをどのように構成するかで、クラスタ・インターコネクトのスループットが大きく異なってきます(詳細は、7.1 節を参照)。

- アプリケーション固有のトラフィック

クラスタ・インターコネクトには、各メンバの仮想ネットワーク・インタフェース (ics0) に対応した TCP/IP アドレスがあります。ユーザ・アプリケーションは、このアドレスを使用してインターコネクト上の通信を行います。このトラフィックがインターコネクトに与える負荷は、アプリケーションの組み合わせによって異なります(詳細は、7.2 節を参照)。

- クラスタ別名ルーティング

クラスタ別名には1つの TCP/IP アドレスが割り当てられます。クライアントは、このアドレスを使うことにより、クラスタ・メンバ全体またはそのサブセットを指定することができます。ただし、別名サブシステムでは、各 TCP/IP 接続は、ある特定のクラスタ・メンバで動作している1つのプロセスに対して行われます。たとえば、あるクラスタ別名に

対して複数のネットワーク・ファイル・システム (NFS) 操作を同時に行うと、その処理はクラスタ・メンバ間で分散されますが、クラスタ別名に対する個々の NFS 操作は、ある 1 つのメンバにある NFS デーモンがサービスします。クラスタ・インターコネクトは、クラスタ別名宛の TCP/IP パケットをその接続をサービスしている特定メンバへ送る場合に使われます。クラスタ別名に必要なインターコネクトの帯域幅は、その別名がどの程度使われるかに依存します (詳細は、7.3 節を参照)。

このような高レベルでの使われ方を見れば、クラスタ・インターコネクトの通信負荷がクラスタのストレージ構成とクラスタで実行するアプリケーションの組み合わせによって大きく影響されることがわかります。

表 7-1 に、LAN インターコネクトと Memory Channel インターコネクトを対象にした、コスト、性能、規模、メンバ間の距離、Memory Channel アプリケーション・プログラミング・インタフェース (API) ライブラリのサポート、および冗長性の比較を示します。以降の各節では、クラスタ・インターコネクトの帯域幅を管理する方法と、いくつかの要因に基づいて適切なインターコネクトを選択する方法を説明します。

表 7-1: Memory Channel インターコネクトと LAN インターコネクトの特性比較

Memory Channel	LAN
コスト高。	一般的に低コスト。
高帯域幅、低遅延。	100 Mb/秒の場合:帯域幅は中で、遅延は中～高。1000 Mb/秒の場合:高帯域幅で、遅延は中～高。
最高 8 メンバで、Memory Channel ハブの容量により制約される。	現在は最高 8 メンバ。将来、拡張予定。
メンバ間の距離は、銅線の場合、最長 20 m (65.6 フィート)。仮想ハブ・モードの光ファイバ・ケーブルの場合、最長 2000 m (1.2 マイル)。物理ハブを使った光ファイバ・ケーブルの場合、最長 6000 m (3.7 マイル)。	ネットワーク・セグメントの距離は、使用するイーサネット・ハードウェアの機能とそのオプションによって決まる。『クラスタ・ハードウェア構成ガイド』で説明されている LAN インターコネクト・ハードウェアの要件と構成の指針を使用する場合は、 <a href="http://www.compaq.com/quickspecs">http://www.compaq.com/quickspecs</a> にある個別のネットワーク・アダプタの『QuickSpec』を参照。

表 7-1: Memory Channel インターコネクトと LAN インターコネクトの特性比較 (続き)

Memory Channel	LAN
Memory Channel API (アプリケーション・プログラミング・インタフェース) ライブラリを使用可能。	Memory Channel API ライブラリは使用不可。アプリケーションによっては、TruCluster Server バージョン 5.1B から新しく使えるようになったメンバ間の汎用シグナル送信機能 (クラスタ単位の kill) を使うだけで、メンバ間の通信を十分に行える。
マルチレール (フェイルオーバー・ペア) 冗長 Memory Channel 構成。	各メンバ上で複数のネットワーク・アダプタを NetRAIN (redundant array of independent network adapters) の仮想インタフェースとして構成するとともにその接続を複数のスイッチに分散することで、冗長性を実現する。

## 7.1 クラスタ・インターコネクトを経由するストレージ・トラフィックの制御

クラスタ・ファイル・システム (CFS) では、ファイル・システムのアクセスをクラスタ全体にわたって調整します。この調整機能は、ファイル・システムごとに CFS サーバの役割を 1 つのクラスタ・メンバに担当させることで実現します。CFS サーバは、他のクラスタ・メンバに代わって、そのファイル・システムに対するすべてのアクセス、つまり読み取りと書き込みを行います。

TruCluster Server バージョン 5.1A から、ファイル・システムの読み取りアクセスが、CFS サーバを迂回してディスクに対して直接行えるようになります。この機能を利用すると、クラスタ・インターコネクトを経由しません。クラスタにあるすべてのディスクがどのクラスタ・メンバからでも同じようにアクセスできれば、この機能を使うことにより、読み取り操作に必要なクラスタ・インターコネクトの帯域幅を大幅に減らすことができます。ただし、読み取りアクセスがインターコネクトを迂回できても、他のメンバの提供するファイル・システムへ書き込む際は (この場合は、間接入出力アクセスになる)、すべてインターコネクトを通ります。ファイル・システムに大量のデータを書き込むアプリケーションがあれば、できる限りそのアプリケーションをそのファイル・システムの CFS サーバと同じメンバに配置して、このトラフィックを減らすことをお勧めします。そうすれば、インターコネクトを経由するファイル・システムの入出力は、リモートに対する書き込みだけになります。アプリケーションの組み合わせ、CFS サーバの配置、およびリモートに対する書き込みのデータ量を分析すれば、そのクラスタに最適なインターコネクトを決定できます。

Oracle Parallel Server (OPS) のようなアプリケーションでは、ディスクの書き込み要求をその対象ディスクへ直接送ることで、CFS サーバに至るクラスタ・インターコネクトの利用を省くことができます。この直接入出力は、ファイルをオープンするときにアプリケーションで `O_DIRECTIO` フラグを指定することによって有効にされます。直接入出力が有効にされると、アプリケーションがクラスタ全体にわたってそのファイルに対する自身の書き込みを調整することが、CFS に対して宣言されます。この機能を使うアプリケーションが増えれば、指定されたファイルに対するクラスタ全体の書き込みスループットが増大するだけでなく、クラスタ・インターコネクトを経由するリモート書き込みのトラフィックも減少します。

この方法は、OPS のように他の方法ではデータ・キャッシュ、事前読み取り、および非同期書き込みによる性能改善効果を期待できないアプリケーションに対してだけ有効です。アプリケーションの開発者はこのフラグの利用を慎重に行ってください。アプリケーションがこのフラグを設定してファイルをオープンすると、オペレーティング・システムは、そのファイルがクローズされるまで、通常書き込み同期機能をそのファイルに対して適用しなくなります。この場合、アプリケーションで独自のキャッシュ管理、ロック管理、および非同期入出力管理などを行わなければ、重大な性能低下やデータ破壊が発生します。

CFS およびデバイス要求ディスパッチャによるクラスタ・インターコネクトの使用と、直接入出力による最適化についての詳細は、『クラスタ管理ガイド』を参照してください。

## 7.2 クラスタ・インターコネクトを経由するアプリケーション・トラフィックの制御

クラスタにあるコンピュータ資源の利用方法はアプリケーションごとに異なります。あるクラスタでは、各メンバが同じストレージおよび管理環境を共用して、それぞれが独立したコンピューティング・サービスを提供します（たとえば、ユーザが自分のプログラムをある 1 つのシステムで実行するタイムシェアリング中心のクラスタ）。一方、OPS のような他のアプリケーションでは、分散処理を使って、クラスタ・メンバのすべてのコンピュータ資源をクラスタ単位の単一アプリケーションに集中させます。この場合は、以下に示すように分散型アプリケーションの構成要素どうしがどのように通信するかを理解しておく必要があります。

- 共用ディスク・ファイルを使って情報を交換するかどうか。

- インターコネクトを経由したプロセス間の直接通信で通信するかどうか。
- 単位時間あたりの通信の頻度とデータ量はどのくらいか。
- 伝送遅延から見て、アプリケーションには何が必要か。

これらの質問に答えることができれば、アプリケーションに必要な条件をインターコネクト・オプションの特性として表すことができます。たとえば、調整メッセージのやりとりに 1 秒あたり 10,000 バイトしか必要としないアプリケーションでは、クラスタの規模が大きくなっても、LAN インターコネクトに余分な負荷をかけることなく、そのコンピュータ資源を利用することができます。一方、OPSのように高いデータ転送速度と低い伝送遅延を必要とする分散型アプリケーションでは、クラスタの規模が小さくても、インターコネクトとして Memory Channel を使うことにメリットがあります。

### 7.3 クラスタ・インターコネクトを経由するクラスタ別名トラフィックの制御

インターコネクトの種類を決定するときには、クラスタ別名を使うアプリケーションの組み合わせ、クラスタ別名を使ってクラスタへ送信されるデータ量、およびクラスタのネットワーク・トポロジ（たとえば、メンバが外部ネットワークに対称に接続されているか非対称に接続されているか）が重要な検討項目となります。

クラスタ別名の通常の利用方法（たとえば、telnet, ftp, Web ホスティング）では、インターコネクトの通信要求はわずかです。これらのアプリケーションでは、一般に、クラスタ別名に宛てて送信されるデータ量より、クラスタからクライアントに返されるデータ量の方が大きく上回ります。サーバ・プロセスに宛てて送られてくるデータ・パケットだけが、インターコネクトを経由する必要があります。すべてのメンバが外部に接続されていれば、クラスタから出ていくパケットは、すべて外部ネットワークに直接送信され、インターコネクトを経由する必要がありません。したがって、このようなアプリケーションに必要なインターコネクトの帯域幅は、一般的に狭くて済みます。

一方、よく使われるネットワーク・ファイル・システム (NFS) では、クラスタ・インターコネクトに多くの帯域幅を必要とします。サービスを行っているディスクからデータを読み取る場合は、読み取り要求だけがインターコネクトを経由するので、インターコネクトのトラフィック量はそれほど増えません。しかし、NFS を使ってディスクにデータを書き込む場合は、イン

ターコネクットのトラフィック量がかなり増えます。この場合、インターコネクットを経由しなくてはならない入力データはディスク・ブロックから構成されています。NFS サービスを提供するクラスタでは、さまざまなインターコネクット構成案を検討して、ディスクの書き込みが発生しそうな単位時間あたりの平均データ量とそのインターコネクット構成で提供される帯域幅とを比較する必要があります。

TruCluster Server バージョン 5.1B で、NFS の書き込みによる影響を緩和する機能が導入されました。つまり、一部のクラスタ・メンバに別のクラスタ別名を割り当てて NFS のサービスを提供させることができるようになりました。この機能を使用すれば、一部の特定クラスタ・メンバだけを NFS サーバとして認識させることができるので、対応するサーバ・プロセスにインターコネクットを経由して送信しなければならない入方向の平均パケット数が減ります。4 メンバ・クラスタで負荷をランダムに分散すると、平均してディスク書き込みの 75% がインターコネクットを経由します。しかし、このうちの 2 つのメンバに別のクラスタ別名を割り当て、NFS のサービスを提供させると、インターコネクットを経由するディスク書き込みの平均的な数の割合が 50% に低下します。

クラスタ別名の使用方法と調整方法については、『クラスタ管理ガイド』を参照してください。

## 7.4 クラスタ・インターコネクットのトラフィックに与えるクラスタ規模の影響

最適なインターコネクットを決定するときは、クラスタ・メンバの数、つまり規模を考慮するだけでは不十分です。クラスタにおける利用方法がインターコネクットにどのような負荷をかけるかも考える必要があります。規模の大きいクラスタでは、特定のアプリケーションの組み合わせに対して高いデータ転送容量を用意する傾向がありますが、クラスタのストレージ構成方法やアプリケーションの特性をよく検討して、適切なインターコネクットを決定した方が賢明です。しかし、クラスタの規模も、ある意味ではインターコネクットの帯域幅要件に影響を与えます。アプリケーションの処理がクラスタ全体にわたってランダムに（しかも管理されずに）分散されており、CFS サーバも均等に分散されていると仮定すると、クラスタ・インターコネクットを経由するディスク書き込みの割合は、クラスタの規模が大きくなるにつれて増えていきます。たとえば、2 メンバのクラスタでは、平均して 50% の書き込みがインターコネクットを経由しますが、4 メンバのクラスタでは、これが 75% に増

加します。7.1 節では、ファイル・システムに対する書き込みの大部分をそのファイル・システムを管理している CFS サーバが行うようなシステムを推奨しています。この推奨構成は、インターコネクトを経由する書き込み回数を最小限に抑えることができるので、使用するインターコネクトの種類に関係なく有効です。この推奨事項を満足する度合いが高いほど、ディスク書き込みに必要なインターコネクトの帯域幅は少なくなります。

ただし、クラスタの規模 (メンバ数と使用するディスク数の両方) がインターコネクトのトラフィックに直接影響する場合があります。それは、クラスタ・メンバ数の変化です。特にメンバがクラスタから外れる場合が問題で、残るメンバは他のクラスタ・メンバに調整メッセージを渡さなければなりません。Memory Channel インターコネクトは遅延が少ないので、Memory Channel を使用するクラスタではこの変化が迅速に通知されます。どちらのインターコネクトを使用するか判断するときは、メンバ数の変化がどの程度の頻度で発生するか (たとえば、クラスタ・メンバが定期的にリブートされるか) も判断基準に入れてください。

## 7.5 クラスタ・インターコネクトの選択

前の節で紹介した推奨事項の他に、次に述べる規則と制約事項もクラスタ・インターコネクトの選択に関係があります。

- すべてのクラスタ・メンバを、LAN インターコネクトまたは Memory Channel インターコネクトのいずれかを使うように構成する必要があります。クラスタ内で種類の異なるインターコネクトを混在させることはできません。
- Memory Channel API ライブラリを使うアプリケーションには Memory Channel が必要です。LAN インターコネクトを使うクラスタに Memory Channel API のアプリケーションだけが使う Memory Channel を構成することもできます。Memory Channel API を使うと、クラスタ・インターコネクトに TCP/IP トラフィックが少しだけ余分に発生します。
- クラスタ内に 1 台でも AlphaServer DS10L システムを構成する場合は、ファースト・イーサネット (100Base-T) LAN インターコネクトが必要です。AlphaServer DS10L システムは、10/100 Mb/秒のイーサネット・ポート × 2、64 ビット PCI (Peripheral Component Interconnect) 拡張スロット × 1、内部固定 IDE (Integrated Device Electronic) ディスク × 1 の構成で出荷されます。AlphaServer DS10L をクラスタに構成するときは、(クラスタ・ルート、メンバ・ブート・ディスク、およ



びオプションのクォーラム・ディスクが存在する) 共用ストレージに PCI 拡張スロットを、また外部ネットワークにイーサネット・ポートをそれぞれ使い、残りの 10/100 Mb/秒 イーサネット・ポートを LAN インターコネクで使うことをお勧めします。

- Memory Channel インターコネクを LAN インターコネクに置き換える場合 (およびその逆の場合) は、クラスタをしばらくダウンさせる必要があります。同様に、ファースト・イーサネット LAN インターコネクをギガビット・イーサネット LAN インターコネクに置き換える場合 (およびその逆の場合) は、クラスタをダウンさせる必要があります。
- LSM (Logical Storage Manager) を使えば、ストレージの透過的なミラーリングと高可用性アクセスを実現できます。しかし、地理的に分散したクラスタでは、LSM はデータ複製技術として適切ではありません。耐災害性システムでは、LAN または Memory Channel ベースのインターコネクと LSM を使った構成はサポートされていません。代わりに、StorageWorks Data Replication Manager (DRM) ソリューションを使った構成がサポートされています。

## 7.6 Memory Channel インターコネク

Memory Channel インターコネクは、クラスタのニーズに合わせて特別に設計されています。Memory Channel インターコネクでは、ブロードキャストとクラスタ・メンバ間のポイント・ツー・ポイント接続の両方が可能です。

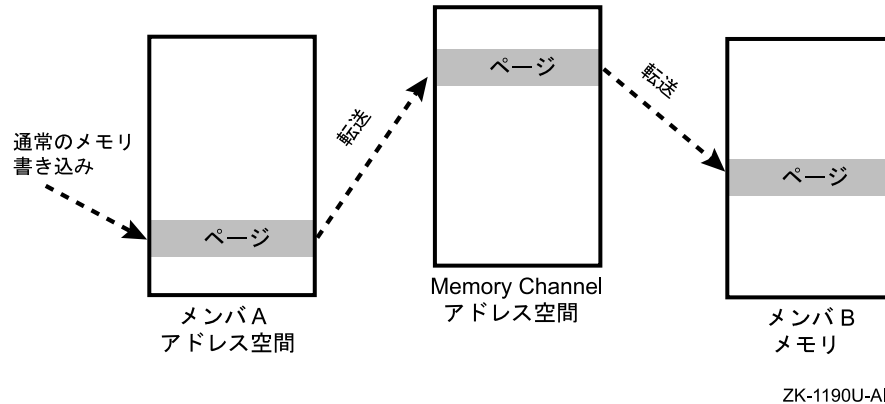
Memory Channel インターコネクには、次のような機能があります。

- クラスタ・メンバは、他のクラスタ・メンバとの間で高性能のメモリ・マップ接続を設定できます。その後、他のクラスタ・メンバは、Memory Channel インターコネクからの転送を直接メモリにマッピングできます。このため、クラスタ・メンバは書き込み専用のウィンドウを、他のクラスタ・システムのメモリ内に確保できます。この接続による通常のメモリ転送は、非常に短い待ち時間 (3 ~ 5 マイクロ秒) で完了します。
- エラー・チェックが組み込まれていて、実質的に検出されないエラーが無いため、ソフトウェアのエラー検出メカニズム (チェックサムなど) を省略できます。エラーが検出される頻度は非常に低く、接続 1 つに対して、1 年に 1 エラー程度です。

- 高性能の相互に排他的なロック (スピンロックを使用) により, 連携動作するアプリケーション間でのリソースの同期を制御します。

図 7-1 は, Memory Channel 転送の概略フローを示しています。

図 7-1: Memory Channel の論理図



各メンバ・システムの PCI スロットには, Memory Channel アダプタを取り付ける必要があります。これらのアダプタは, リンク・ケーブルで接続します。クラスタ内に 3 つ以上のメンバがある場合は, Memory Channel ハブも必要です。

Memory Channel の構成を冗長性のあるマルチレールにすると, 信頼性と可用性がさらに高くなります。マルチレールにするには, 各クラスタ・メンバに 2 つ目の Memory Channel アダプタと, そのアダプタを接続するためのリンク・ケーブルが必要です。クラスタ内に 3 つ以上のメンバがある場合は, 2 つ目の Memory Channel ハブも必要です。

Memory Channel のマルチレール・モデルは, 物理レールと論理レールという概念で動作します。物理レールは, ケーブルと Memory Channel アダプタが接続された Memory Channel ハブと, 各ノードのアダプタ用の Memory Channel ドライバとして定義されます。論理レールは, 1 つまたは 2 つの物理レールからなります。

1 つのクラスタでは, 1 つ以上 (最大 4 つ) の論理レールを使用できます。論理レールは, 次のようなスタイルで構成されます。

- シングルレール
- フェイルオーバー・ペア

クラスタがシングルレールのスタイルで構成されている場合、物理レールと論理レールの間には 1 対 1 の関係があります。この構成には、フェイルオーバー機能がありません。物理レールに障害が発生すると、論理レールにも障害が発生します。この構成は、高可用性アプリケーションではなく、Memory Channel アプリケーション・プログラミング・インタフェース (API) ライブラリを使用する高性能のコンピューティング・アプリケーションで主に使用されます。

クラスタがフェイルオーバー・ペアのスタイルで構成されている場合、論理レールは、1 つがアクティブでもう 1 つが非アクティブの、2 つの物理レールからなります。アクティブな物理レールに障害が発生すると、フェイルオーバーが発生して、非アクティブの物理レールが使用され、論理レールはフェイルオーバーの後もアクティブなままです。このフェイルオーバーは、ユーザには見えません。フェイルオーバー・ペアのスタイルは、マルチレール構成での省略時の設定です。

構成済みで利用可能な二次 Memory Channel インターコネクトがすべてのメンバ・システム上にあり、一次インターコネクトで次のいずれかの状態が発生すると、クラスタは 1 つの Memory Channel インターコネクトから別のインターコネクトへフェイルオーバーします。

- 1 分間に 10 個を超えるエラーが記録された場合。
- リンク・ケーブルが切断された場合。
- ハブがオフになった場合。

フェイルオーバーが完了すると、二次 Memory Channel インターコネクトが一次インターコネクトになります。当初一次であったインターコネクトの問題を修正するまでは、新たなインターコネクト・フェイルオーバーは発生しません。

1 分間に発生した Memory Channel エラーが 10 個を超えるメンバ・システムがある場合、Memory Channel のエラー回復処理では、二次 Memory Channel インターコネクトがメンバ上に構成されているかどうかを次のように調べます。

- 二次 Memory Channel インターコネクトがすべてのメンバ・システム上にある場合、エラーを検出したメンバ・システムは一次 Memory Channel インターコネクトを不良とマークし、二次 Memory Channel インターコネクトにフェイルオーバーするように、すべてのメンバ・システム (自分自身を含む) に指示します。

- 構成済みで利用可能な二次 Memory Channel インターコネク트가ないメンバ・システムがある場合，エラーを検出したメンバ・システムは，Memory Channel のハードウェア・エラー限界値を超えたことを示すメッセージを表示して，パニック状態になります。

クラスタ内に Memory Channel インターコネク트를構成する方法についての詳細は，『クラスタ・ハードウェア構成ガイド』を参照してください。

Memory Channel の API ライブラリは，Memory Channel API クラスタ・メンバ間の効率的なメモリ共用を実現します。このメモリ共用には，自動エラー処理，ロック処理，UNIX スタイルの保護機能が備わっています。Memory Channel API ライブラリについての詳細は，『クラスタ高可用性アプリケーション・ガイド』を参照してください。

## 7.7 LAN インターコネクト

標準的な 100 Mb/秒 や 1000 Mb/秒の LAN で動作するイーサネット・アダプタ，スイッチ，またはハブであれば，LAN インターコネクトでもまず問題なく動作します。

---

### 注意

---

FDDI (Fiber Distributed Data Interface)，LANE (ATM LAN Emulation)，10 Mb/秒のイーサネットは，LAN インターコネクトではサポートされていません。

---

クラスタ LAN インターコネクトのイーサネット・ハードウェアには，次の条件が必要です。

- LAN インターコネクトは，クラスタ・メンバ専用でなければなりません。あるクラスタ・メンバの LAN インターコネクト・アダプタで送信されたパケットは，別のメンバの LAN インターコネクト・アダプタでのみ受信できます。
- LAN インターコネクトは，2 つのクラスタ・メンバ間を全二重で直接接続するか，またはスイッチやハブ (どちらか一方のみ) を使って接続します。3 つ以上のメンバで構成されるクラスタ，およびメンバがクラスタ・インターコネクト・デバイスとして NetRAIN (redundant array of independent network adapters) の仮想インタフェースを使うクラスタでは，スイッチが 1 つ以上必要です。

## 注意

LAN インターコネクトのほとんどの構成ではハブとスイッチは置き換え可能ですが、性能とスケーラビリティの点でスイッチの使用をお勧めします。ハブは半二重モードで動作するため、LAN インターコネクトでハブを使用するとクラスタの性能が制限される可能性があります。さらに、ハブには LAN インターコネクトの二重化冗長構成に必要な機能がありません。そのため、3 つ以上のメンバで構成されるクラスタに対しては、LAN インターコネクトにハブではなくスイッチを使用した方が、スケーラビリティははるかに向上します。

- 100 Mb/秒または 1000 Mb/秒の全二重動作に対しては、アダプタ・ポートとスイッチ・ポートの互換性があるように構成する必要があります。

DE60<sub>x</sub> ファミリのアダプタ (コンソール名の形式が `eiX0`) または DEGPA-*xx* アダプタと一緒にスイッチを使用する場合は、自動折衝をサポートするスイッチを使用してください。自動折衝が適切に行われない DE50<sub>x</sub> ファミリのネットワーク・アダプタ (コンソール名の形式が `ewX0`) と一緒にスイッチを使用する場合は、自動折衝の機能を無効にできるものを使用する必要があります。

- 完全に冗長性のある LAN インターコネクトで 2 つのスイッチを 2 本のクロス・ケーブルを使って接続すると、2 つ目のリンクによってルーティング・ループが形成されます。そのため、このループが原因で発生するパケット転送の問題を回避できるようにスイッチを構成する必要があります。以下に示すように、スイッチ間の並列リンクをサポートする方法は 3 つありますが、一般的なスイッチであれば、少なくともその 1 つを備えています。それらを有効な順に説明すると、以下のとおりです。

リンク集約	2 つのスイッチ間における複数の物理リンクを単一のリンクとして扱い、パケットのトラフィックを複数の物理リンク間で分散する。
-------	---

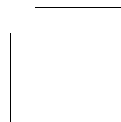
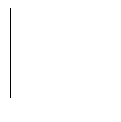
リンク復元	2 つのスイッチ間における複数の物理リンクのうち、1 つのリンクをアクティブ・リンクとし、残りのリンクをスタンバイ・リンクにして、それらの間でフェイルオーバーを行う。
-------	---

スパニング・ツリー・プロトコル

分散ルーティング・アルゴリズムを使うことで、スイッチどうしが協調しあってルーティング・ループを検出し削除する。

- スイッチ間に並列リンクがある場合は、そこに使われているスイッチ・ポートを通してルーティング・ループが形成されます。スパニング・ツリー・プロトコル (STP) は、こうしたループを避けるために有効です。しかし、メンバが単一のアダプタを使うか NetRAIN デバイスに含まれている複数のアダプタを使うかに関係なく、クラスタ・メンバに接続されているすべてのイーサネット・スイッチ・ポートで STP を無効にしておく必要があります。そうしなければ、クラスタ・メンバへブロードキャスト・メッセージが大量に送りつけられます。その結果、クラスタはサービスを提供できなくなります。
- どのクラスタ・メンバも、他のすべてのメンバとポイント・ツー・ポイントで接続している必要があります。あるメンバで LAN インターコネクต์に使われているイーサネット・アダプタに障害が起こると、そのメンバは他のどのメンバとも通信できなくなります。あるメンバから来るインターコネクต์・トラフィックを別のサブネットに転送するようにクラスタ・インターコネクต์を構成することはできません。つまり、スイッチの機能をメンバ・システムに行わせることはできません。
- 2 つのクラスタ・メンバ間で使用できるスイッチは最大で 2 つです。たとえば、通信衛星のアップリンクやワイド・エリア・ネットワーク (WAN) を LAN インターコネクットの 2 つの構成要素間のパスとして使うと、許容できないほど大きな遅延が発生するので、そのような接続は絶対に行わないでください。
- LAN インターコネクต์では、Tru64 UNIX の機能 (lagconfig コマンドを含む) を使ったイーサネット・アダプタのリンク集約を行えません。
- 管理を単純にするために、すべてのクラスタ・メンバで LAN インターコネクットのネットワーク・アダプタを対称に構成してください。他のネットワーク・アダプタについても、すべてのメンバで同じタイプのアダプタを同じ相対位置にインストールすれば、どのクラスタ・メンバでもアダプタの名前が同じようになります。相互接続した 2 つ以上のスイッチと NetRAIN 仮想インタフェースとをメンバの相互接続デバイスとして使った、完全に冗長性のある LAN インターコネクต์構成では、すべてのメンバで、それぞれの NetRAIN にリストされている先頭の

ネットワーク・アダプタを最初のスイッチに，2 番目のネットワーク・アダプタを 2 番目のスイッチに，という具合に，同じ順序で接続してください。こうすることで，監視や保守を行う際にアダプタを簡単に識別することができます。また，クラスタが最初にブートする時，どのメンバーのアクティブ・アダプタも同じスイッチに接続されます。LAN インターコネクトのすべてのアクティブ・アダプタが同じスイッチに接続するようにしておけば，障害が発生しても，クラスタのネットワークが分断がされないように保護することができます。詳細は，『クラスタ・ハードウェア構成ガイド』を参照してください。





## 分散ロック・マネージャ

分散ロック・マネージャ (DLM) には、クラスタ内で連携して動作するプロセスが、共用リソース (raw ディスク・デバイスやプログラムなど) へのアクセスの同期をとるための機能があります。DLM によって共用リソースへのアクセスの同期を効率的にとるには、クラスタ内でリソースを共用するプロセスすべてが、DLM 機能を使用してそのリソースへのアクセスを制御しなければなりません。たとえば、分散型データベース・アプリケーションはロック・マネージャ・サービスを使用して、データベースに関連する共用ディスクへのアクセスの同期をとります。

アプリケーションは、名前付き共用リソースに対し、ロックを取得します。リソース名は、一次元の名前とツリー構造の名前のどちらにもできます。リソース・ツリーを使用すると、共用リソースの構造を反映する、ロックとサブロックの階層構造を作成することもできます。DLM には、次のような機能があります。

- 相互排他的データ・アクセス、制限付き共用データ・アクセス、完全な共用データ・アクセスが可能。
- ロックを保持しているプロセスに対して、そのロックによって他のプロセスのリソースへのアクセスがブロックされる場合に通知が行われる。
- リソースへのロック要求をキューイングしたプロセスに対して、その要求が許可されたときに通知が行われる。
- ロックのモードを、緩やかな制限のモードと厳しい制限のモードの間で切り替えることができる。
- ロックに関する情報を返す。

DLM は、分散型の集中ツリー構造に設計されています。DLM は、各クラスタ・メンバにロック情報を複製しません。ロック・ツリーを管理するクラスタ・メンバが、そのツリーに関するすべての情報を保持します。ロックを保持しているメンバは、そのリソースに対し自分がロックを行っていることだけを

認識します。どのメンバ・システムも、任意のロック・ツリーのマスタとして動作できます。これにより、ロック管理の全体的な負荷が分散されます。

DLM は分散ディレクトリ・サービスを使用して、リソース・ツリーのディレクトリ・ノードの位置をすばやく見つけます。ディレクトリ・テーブルは、ルート・リソース名と、リソースのマネージャであるクラスタ・メンバを対応付けます。このディレクトリ・テーブルは、すべてのクラスタ・メンバで同じです。

DLM は、メンバの障害に対処するように設計されています。ロックを保持しているメンバに障害が発生すると、そのロックは解放されます。メンバ・システムに障害が発生すると、そのメンバが直前にマスタとなっていたロックに対して新しいロック・マスタが選ばれ、関連するロック情報がすべて渡されます。

## クラスタのインストールと管理

この章では、クラスタのインストール (9.1 節) と管理 (9.2 節) についての概要を説明します。

### 9.1 インストール

TruCluster Server バージョン 5.1B では、次のような 3 つのアップグレード・パスをサポートしています。

1. TruCluster Server バージョン 5.1B のフル・インストール。
2. TruCluster Server バージョン 5.1A からのローリング・アップグレード。

ローリング・アップグレードとは、クラスタの操作中にクラスタのソフトウェア・アップグレードを実行することです。クラスタが Tru64 UNIX ベース・オペレーティング・システム、クラスタ、WLS (ワールドワイド言語サポート) ソフトウェアの混在バージョン環境を透過的に維持しながら、一度に 1 つのメンバがローリングされて、元に戻されます。

クラスタのアップグレードに要する時間を短くするために、複数メンバのローリングを並列で処理することができます。同時にローリングするメンバの数を制限する条件は、ローリングされないメンバ (メンバの状態が UP) にクォーラムを維持するのに十分な数のポートがあることです。並列処理でローリングすれば、一度に 1 つのメンバをローリングして 4 メンバ構成のクラスタをアップグレードするのに要する時間とほぼ同じ時間で、8 メンバ構成のクラスタをアップグレードできます。

ローリング・アップグレード手順は、次の 3 つの主要な作業で使われます。

- a. Tru64 UNIX ベース・オペレーティング・システムおよびクラスタ・ソフトウェアの以前のバージョンから現在のバージョンへのローリング
- b. クラスタへのパッチ・キットのローリング

c. バージョン 5.1B クラスタへの NHD (New Hardware Delivery) キットのローリング

パッチ・キットまたは NHD キットのローリングでは、ベース・オペレーティング・システムおよびクラスタ・ソフトウェアの新しいリリースへのローリングと同じ手順を使用します。clu\_upgrade コマンドは、ローリング・アップグレードを制御します。clu\_upgrade コマンドの詳細については、clu\_upgrade(8) を参照してください。

3. TruCluster Production Server Software または TruCluster Available Server Software バージョン 1.5 またはバージョン 1.6 製品からの 3 つのアップグレード手順。

このうちの 2 つのオプションでは、旧クラスタ (rz\* 形式のデバイス名) から新クラスタ (dsk\* 形式のデバイス名) へのストレージの移行を容易にするために特別に設計されたスクリプトを使用します。共用ストレージがほとんどないか全くない TruCluster Memory Channel Software 製品のアップグレード・パスについては、『クラスタ・インストレーション・ガイド』でも説明しています。

TruCluster Server バージョン 5.x クラスタを作成する際の主な違いは、Tru64 UNIX をクラスタ内の 1 つのシステムにだけインストールすることです。CFS はクラスタ単位の共用ファイル・システムを作成するため、クラスタが作成されれば、クラスタ内に追加でブートされたメンバは、これらのファイルにアクセスできるようになります。TruCluster Server バージョン 5.0 より前のリリースでは、ベース・オペレーティング・システムをすべてのクラスタ・メンバにインストールしなければならず、クラスタ単位のファイル・システムはありませんでした。

TruCluster Server では、最初のクラスタ作成、メンバの追加、メンバの削除は、3 つの対話型のインストレーション・スクリプト、clu\_create、clu\_add\_member、clu\_delete\_member によって行います。これらのスクリプトには、オンライン・ヘルプがあります。また、これらのスクリプトは、/cluster/admin ディレクトリにログ・ファイルを書き込みます。

次のリストは、新規に TruCluster Server クラスタをインストールして形成するために必要な手順の概要です。

1. 『クラスタ・ハードウェア構成ガイド』の情報をを使用して、システムおよびストレージの、ハードウェアおよびファームウェアを構成します。

2. AdvFS ファイル・システムを使用して、最初のクラスタ・メンバになるシステムのローカル・ディスクに Tru64 UNIX をインストールします。
3. ネットワークやタイム・サービスを含め Tru64 UNIX システムを構成します。クラスタで使用するアプリケーションをロードして構成します。
4. TruCluster Server のライセンスおよびソフトウェアをロードします。

---

#### 注意

---

各クラスタ・メンバには、Tru64 UNIX と TruCluster Server の両方のライセンスがなければなりません。

入手したパッチ・キットまたは NHD キットがあれば、クラスタ・ソフトウェアをロードした後、`clu_create` コマンドを実行する前にインストールします。`clu_create` コマンドを実行する前にそれらをインストールすれば、後になってそれらをクラスタにローリングする必要がなくなります。

5. `clu_create` コマンドを実行して、最初のクラスタ・メンバのブート・ディスクを作成し、クラスタ単位のルート (/), /usr, /var の AdvFS ファイル・システムを作成して配置します。
6. Tru64 UNIX システムを停止させ、最初のメンバのクラスタ・ブート・パーティションを含むディスクからブートします。システムが起動すると、シングル・メンバ・クラスタが形成され、クラスタ単位のルート (/), /usr, /var の各ファイル・システムがマウントされます。
7. シングル・メンバ・クラスタにログインし、`clu_add_member` コマンドを実行してクラスタにメンバを追加します。新しいメンバは、次のメンバを追加する前にブートしてください。

TruCluster Server のインストールについての詳細は、『クラスタ・インストール・ガイド』を参照してください。

## 9.2 管理

クラスタ単位のファイルのネームスペースがあるため、クラスタ管理が大幅に簡略化されます。クラスタには、ほとんどのシステム構成ファイルのコピーが 1 つだけあります。たとえば、`/etc/group` ファイル 1 つと

/etc/passwd ファイル 1 つで、クラスタが 1 つのセキュリティ・ドメインとして管理されます。

ファイルへのユーザ・アクセスは、ユーザがどのノードにログインして、どのノードがそのファイルのサービスを行っているかということとは独立しています。ファイル許可とアクセス制御リスト (ACL) は、クラスタ全体で共通です。

監査ログは、共通の位置にあります。各メンバのホスト名がログ・ファイルに付加され、監査イベントを追跡する際に混乱しないようになっています。

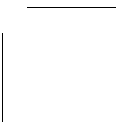
多くの場合、TruCluster Server 環境の次のような側面を管理しなければならないことがあるため、1 つのシステムではなくクラスタを管理していることが明らかに分かります。次に示すリストでは、各管理領域の後に、クラスタを管理またはモニタリングするために使用する、クラスタ固有のコマンドが示されています (SysMan Menu および SysMan Station の GUI で、コマンド行のほとんどの機能を実行できます。ただし、クラスタのインストールではクラスタ・インストレーション・コマンドを使用しなければなりません)。

- 最初のクラスタ・メンバのインストール、メンバの追加と削除、およびクラスタ構成の照会をサポートする、クラスタ作成 (clu\_create , clu\_add\_member , clu\_delete\_member , clu\_check\_config)
- 高可用性アプリケーションの定義と管理を可能にする CAA (Cluster Application Availability) (caa\_profile , caa\_register , caa\_unregister , caa\_start , caa\_stop , caa\_relocate , caa\_stat)
- ネットワークから単一システムに見えるようにする、クラスタ別名 (cluamgr)
- 有効なクラスタ構成と、そのクラスタ内でのメンバシップを決定し、それによりクラスタ・リソースへのアクセスを制御するクラスタ・クォーラムおよびポート (clu\_quorum)
- CFS サーバのオプションの負荷分散 (cfsmgr)
- デバイス要求ディスパッチャ・サブシステムのオプションの負荷分散 (drdmgr)

上記の項目に加え、SSI (Single System Image) モデルには、コマンド・レベルの例外もあります (SSI とは、可能な場合には、ユーザからクラスタが 1 つのコンピュータ・システムに見えるようにすることです)。たとえば、

`wall` コマンドを実行すると、メッセージはコマンドが実行されたクラスタ・メンバにログインしているユーザにのみ送信されます。すべてのクラスタ・メンバにログインしているすべてのユーザにメッセージを送信するには、`wall -c` コマンドを実行します。`shutdown` コマンドについても、同じことが言えます。個々のメンバのシャットダウンとクラスタ全体のシャットダウンのどちらもできます。

TruCluster Server クラスタの構成と管理についての詳細は、『クラスタ管理ガイド』を参照してください。





---

## 用語集

この用語集では、TruCluster Server 環境で共通に使用されている用語について説明します。

### C

#### **CAA (cluster application availability)**

クラスタ・アプリケーションの可用性。シングル・インスタンス・アプリケーションの可用性を高めるとともに他のタイプのリソース (ネットワーク・インタフェースなど) の状態をモニタリングするサブシステム。クラスタ内で CAA を使用することにより、Tru64 UNIX で動作するどのアプリケーションでもそのシングル・インスタンスの可用性が高くなる。

#### **CDSL (context-dependent symbolic link)**

コンテキスト依存シンボリック・リンク。ターゲット・パス名にたとえば {memb} のような環境変数 (実行時に解決される) が含まれる、特別な形式のシンボリック・リンク。クラスタでは、CDSL によって、システムごとの構成と、共用 CFS ルート (/), /usr, /var ファイル・システム内のデータ・ファイルを維持できる。

#### **CFS (cluster file system)**

クラスタ・ファイル・システム。物理ファイル・システムの上位にあって、クラスタ内にマウントされているすべてのファイル・システムへ (デバイス要求ディスパッチャの支援を受けて) クラスタ単位でアクセスできるようにする仮想ファイル・システム。CFS がすべてのクラスタ・メンバ間でキャッシュの一貫性を維持する。キャッシュの一貫性が保たれるので、すべてのメンバが、クラスタに直接接続されているファイル・システムを矛盾のないかたちで同じように扱うことができる。

#### **cluster application availability (CAA)**

*CAA (cluster application availability)* を参照

## D

### **DLM (distributed lock manager)**

分散ロック・マネージャ。クラスタ内の連携するプロセス間で、共用リソースへのアクセスの同期をとるソフトウェア構成要素。

## E

### **EVM (event manager)**

イベント・マネージャ。カーネル・レベルおよびユーザ・レベルのプロセスと構成要素がイベントをポストできるようにするとともに、選択されたイベントが発生した時点でプロセスに通知されるようにする機能。この機能には、イベント・ビューア、API、コマンド行ユーティリティがある。詳細は、EVM(5) を参照。

## I

### **in\_multi サービス (in\_multi service)**

クラスタ別名サブシステムが接続要求およびパケットをその別名のすべての適格なメンバへ送るように指示する、サービス・ポート上の指定。

### **in\_noalias サービス (in\_noalias service)**

当該サービス・ポートで入力方向の別名メッセージを受けないようにクラスタ別名サブシステムに指示する、サービス・ポート上の指定。

### **in\_nolocal サービス (in\_nolocal service)**

当該サービス・ポートで非別名アドレスへの接続要求を受け付けないようにクラスタ別名サブシステムに指示する、サービス・ポート上の指定。

### **in\_single サービス (in\_single service)**

1 つの別名メンバだけがそのサービスの接続要求およびパケットを受けるようにクラスタ別名サブシステムに指示する、サービス・ポート上の指定。

## L

### **Logical Storage Manager (LSM)**

*LSM (logical storage manager)* を参照

### **LSM (logical storage manager)**

Logical Storage Manager。データ紛失からの保護を行ったり、ディスクの入出力性能を改善したり、ディスク構成をカスタマイズしたりする、ディスク・ストレージ管理ツール。

LSM を使用すると、システム管理者は、ディスク上にあるデータにアクセスするユーザやアプリケーションを中断することなく、ディスク管理機能を実行できる。

### **LSM ディスク・グループ (LSM disk group)**

共通の構成を共有する LSM (Logical Storage Manager) ディスクのグループ。LSM ディスク・グループの構成情報は、LSM ディスク・グループに関連する LSM ディスク、LSM ボリューム、LSM プレックス、LSM サブディスクなどのオブジェクトを記述するレコード群で構成されている。各 LSM ディスク・グループには管理者が割り当てた名前があり、その名前を使用して LSM ディスク・グループを参照することができる。

### **LSM プレックス (LSM plex)**

LSM ボリュームの論理データ・アドレス空間のコピー。ミラーと呼ばれることもある。LSM ボリュームには、LSM プレックスを 8 個まで対応させることができる。読み取りはどの LSM プレックスからでも行えるが、書き込みはすべての LSM プレックスに対して行われる。

### **LSM ボリューム (LSM volume)**

UNIX ファイル・システム、データベース、およびその他のアプリケーションの使用データを含むスペシャル・デバイス。LSM は、アプリケーションと物理ディスクの間に LSM ボリュームを透過的に配置する。この後、アプリケーションは物理ディスクではなく、LSM ボリュームで動作するようになる。たとえば、ファイル・システムは物理ディスクではなく、LSM ボリューム上に作成される。

LSM ボリュームには、ディスク・パーティション・スペシャル・デバイスと使用上の互換性がある、ブロック・インタフェースと raw インタフェースがある。LSM ボリュームは仮想デバイスであるため、管理コマンドを使用して、ミラーリングや、複数ディスク・ドライブ上への分散、別のストレージを使用するための移動、ストライピングを行うことができる。LSM ボリュームの構成は、その LSM ボリュームを使用しているアプリケーションやファイル・システムを中断することなく、LSM ユーティリティを使用して変更できる。

### **LUN (logical unit number)**

論理ユニット番号。ターゲットを通してアドレス指定が可能な、物理周辺デバイスまたは仮想周辺デバイス。LUN はターゲットのバス接続を使用して、SCSI バス上で通信を行う。

## **M**

### **MB/秒**

メガバイト/秒

### **Mb/秒**

メガビット/秒。

### **Memory Channel インターコネクト (Memory Channel interconnect)**

クラスタ・メンバ間での、高速で信頼性の高い通信を実現する PCI (peripheral component interconnect) インターコネクト。物理的には、このインターコネクトは、各メンバ・システムの PCI スロットに取り付けられている Memory Channel アダプタ、アダプタを接続する 1 本以上の Memory Channel ケーブル、オプションの Memory Channel ハブで構成される。

## **O**

### **out\_alias サービス (out\_alias service)**

当該ポートが宛先になっていれば省略時のクラスタ別名をソース・アドレスとして使用するようにクラスタ別名サブシステムに指示する、サービス・ポート上の指定。

## **P**

### **PCI (peripheral component interconnect)**

peripheral component interconnect。同期型で非対称の入出力チャネルである業界標準の拡張入出力バス。

### **peripheral component interconnect (PCI)**

*PCI (peripheral component interconnect)* を参照

## R

### **RAID (redundant array of inexpensive disks)**

冗長性を持つ低価格ディスク・アレイ。ディスク・データを編成して性能と信頼性を高める技術。RAID には次の 3 つの特徴がある。

- 単一の論理デバイスまたは複数の論理デバイスとしてユーザが認識する物理ディスク群である。
- ディスク・データは、定義されている方法で、物理的なドライブ群内に分散される。
- ディスク容量に冗長性を持たせているため、1 台のドライブに障害が発生してもデータを回復できる。

### **redundant array of inexpensive disks (RAID)**

*RAID (redundant array of inexpensive disks)* を参照

### **RIP (routing information protocol)**

ルーティング情報プロトコル。ゲートウェイと他のホストとの間でルーティング情報を交換するために使用するプロトコル。プロトコルの仕様は、PFC 1058 で定義され、RFC 1388 および 1723 で更新されている。

## S

### **SCSI (small computer system interface)**

小型コンピュータ・システム・インタフェース。ディスクやその他の周辺デバイスをコンピュータ・システムに接続するための、米国規格協会 (ANSI) による標準インタフェース。SCSI ベースのデバイスは、同じバス上の複数のデバイスとともに、一列に構成できる。

### **SCSI-2**

オリジナルの SCSI 標準の拡張であり、同じバス上に複数のシステムを配置する機能やホット・スワップをサポートしている。SCSI-2 標準は、ANSI 標準の X3.T9.2/86-109 である。

### **SCSI ID**

SCSI バス上でデバイスを識別する一意のアドレス (0 ~ 15)。

### **SCSI アダプタ (SCSI adapter)**

一般にホスト・バス・アダプタ (HBA) と呼ばれるストレージ・アダプタ。このアダプタは、入出力バスと SCSI バスとを接続する。

**SCSI コントローラ (SCSI controller)**

SCSI アダプタ (SCSI adapter) を参照

**SCSI デバイス (SCSI device)**

SCSI バスに接続できる SCSI アダプタ，周辺機器コントローラ，インテリジェント周辺機器。

**SCSI バス (SCSI bus)**

SCSI プロトコルの伝送および信号の要件をサポートするバス。

**SCSI バス速度 (SCSI bus speed)**

SCSI バスのデータ転送速度。SCSI バス速度には，低速 (slow) (5 MB/秒まで)，高速 (fast) (10 MB/秒まで)，高速ワイド (fast wide) (20 MB/秒まで)，UltraSCSI (40 MB/秒まで) がある。

**SRM (system reference manual)**

オペレーティング・システムからコンソール・ファームウェアへの外部インタフェース。ファームウェアが Alpha SRM (System Reference Manual) に準拠していることを前提とする。

**static サービス (static service)**

当該ポートがダイナミック・ポートとして割り当てられないようにクラスター別名サブシステムに指示する，サービス・ポート上の指定。

## U

**UltraSCSI**

細いケーブルと小型のコネクタを使用し，25 メートル (約 82 フィート) でのバス速度を最大 40 MB/秒とするディファレンシャル SCSI バス標準。

**UltraSCSI ハブ (UltraSCSI hub)**

複数のコネクタを備えた専用の信号変換器。UltraSCSI ハブは，ホスト・バス・アダプタからのディファレンシャル入力 SCSI 信号をシングルエンド信号に変換した後，RAID アレイ・コントローラへの出力コネクション用として，シングルエンド信号をディファレンシャル信号に戻す。UltraSCSI ハブを使用すると，UltraSCSI デバイスを放射状に接続でき，ホストとストレージの分離が促進される。

## V

### **vMAC (virtual media access control)**

クラスタ別名のコンテキストにおいて、各別名 IP アドレスに対して自動的に作成される一意のハードウェア・アドレス。別名 vMAC (virtual Media Access Control) アドレスは、必要に応じて、ノードからノードへ、クラスタ別名プロキシ ARP マスタに従う。どのクラスタ・メンバが別名に対してプロキシ ARP マスタとしてサービスを行っているのかには関係なく、別名の vMAC アドレスは変わらない。

## W

### **WWID (worldwide ID)**

ワールドワイド ID。製造者がディスクに割り当てた一意の識別子。

## Y

### **Y ケーブル (Y cable)**

2 本のケーブルを結合して 1 台のデバイスに接続したり、アダプタや RAID コントローラの外部で共用 SCSI バスをターミネートするケーブル。

## あ

### **アダプタ (adapter)**

バスのプロトコルおよびハードウェア・インタフェースの種類を、別のバスのプロトコルおよびハードウェア・インタフェースに変換するデバイス。

### **アドレス・スイッチ (address switches)**

一部のディスク・ドライブにある電氣的スイッチ。このスイッチによって、ドライブの SCSI アドレス設定が決まる。

### **イベント・マネージャ (EVM: Event Manager)**

*EVM (event manager)* を参照

## か

### **仮想サブネット (virtual subnet)**

クラスタ別名に関連した用語で、物理的に接続していないサブネット。クラスタ別名 IP アドレスは、共用サブネットまたは仮想サブネットのどちらかにある。

### 仮想ハブ・モード (virtual hub mode)

Memory Channel アダプタの接続に Memory Channel ハブを使用しない，Memory Channel インターコネクト構成。仮想ハブ・モードは，メンバ・システムが2つのクラスタだけでサポートされる。仮想ハブ・モードで Memory Channel インターコネクトを設定するには，Memory Channel リンク・ケーブルを使用して，1つのメンバ・システムの Memory Channel アダプタを，他方のメンバ・システムの対応する Memory Channel アダプタに接続する。

### 可用性 (availability)

中断を最小限に抑えるか，中断することなくコンピューティング・サービス (アプリケーションなど) をクライアントから利用できるようにする，コンピューティング・システムの特徴。

高可用性 (*highly available*) も参照

### 期待ボート (expected votes)

クラスタ・メンバが保持しているノード・ボートの合計に，クォーラム・ディスク (定義されている場合) のボートを足した値。

### 共通サブネット (common subnet)

クラスタ別名に関連した用語で，実際に存在する物理サブネット。クラスタ別名の IP アドレスは，共通サブネットまたは仮想サブネット内にある。

### 共用バス (shared bus)

複数のメンバ・システムに接続され，オプションで1つ以上のストレージ・デバイスにも接続されているバス。

### 共用ストレージ (shared storage)

共用バスに接続されているディスク。

### クォーラム (quorum)

メンバがクラスタ単位の共用リソースにアクセスでき，有用な作業を実行できる，クラスタの状態。クラスタ内のノード・ボートとクォーラム・ディスク・ボートの合計が，必要なクォーラム・ボート数以上であると接続マネージャが判断した場合，そのクラスタはクォーラムを持つ。

クォーラム・ボート (*quorum votes*) も参照

### クォーラム・アルゴリズム (quorum algorithm)

接続マネージャが使用する数学的メソッド。このメソッドにより，特定のメンバがクラスタに参加し，クラスタ単位のリソースへ安全にアクセスして，役に立つ作業を実行できるかどうか判断される。



#### クォーラム・ディスク (quorum disk)

h パーティションにクラスタ状態とクォーラム情報が格納されているディスク。各クラスタでは、クォーラム・ディスクを最大で 1 つ使用できる。クォーラム・ディスクには、クォーラムの計算に使用されるポートが割り当てられている。

#### クォーラム・ディスク・ポート (quorum disk votes)

クォーラム・ディスクがクォーラムに投じるポートの数。

#### クォーラム・ポート (quorum votes)

クラスタの形成または維持に必要なポートの数。クォーラム・ポートを計算する式は次のとおり。

クォーラム・ポート = (クラスタ期待ポート + 2) / 2 (小数点以下切り捨て)

#### クォーラム・ロス (quorum loss)

クラスタ単位の共用リソースにアクセスできるメンバがない、クラスタの状態。クラスタ内のメンバおよびクォーラム・ディスクのポートが、必要なクォーラム・ポート数より少ないと接続マネージャが判断した場合、クラスタはクォーラム・ロス (クォーラムの喪失) 状態になる。

クォーラム・ポート (*quorum votes*) も参照

#### クライアント (client)

他のコンピュータ (サーバと呼ばれる) のリソースを使用するコンピュータ・システム。

#### クラスタ (cluster)

アプリケーションやデータの高可用性を実現するために、ストレージや他のリソースを共用するサーバをゆるやかに結合したもの。クラスタは通信媒体、メンバ・システム、周辺デバイス、およびアプリケーションから構成される。メンバ・システム同士は、クラスタ・インターコネクトで通信する。

#### クラスタ・インターコネクト (cluster interconnect)

クラスタ・メンバがクラスタ内通信用に使用する専用のインターコネクト。

#### クラスタ期待ポート (cluster expected votes)

期待ポート (*expected votes*) を参照

#### クラスタ・ファイル・システム (CFS: cluster file system)

CFS (*cluster file system*) を参照

#### クラスタ分断 (cluster partition)

既存のクラスタのノードが複数の独立したクラスタに分断された異常状態。

#### クラスタ別名 (cluster alias)

クラスタ内の全メンバ、またはメンバのサブセットを指定するために使用される IP アドレス。クラスタ別名により、クラスタ内のシステムの一部またはすべてが、外部からは単一のシステムのように見える。

#### クラスタ・メンバ (cluster member)

クラスタ内にある、基本的なコンピューティング・リソース。メンバ・システムは、クラスタ・インターコネクトと物理的に接続されていなければならない。

一般的には、TruCluster Server ソフトウェアで構成され、クラスタの形成やクラスタへの参加が可能なシステム。接続マネージャから見ると、シングル・メンバ・クラスタを形成するか、既存のクラスタへのメンバシップが与えられているシステム。接続マネージャは、クラスタ・メンバ間の通信状態に基づいて、クラスタ・メンバシップを動的に判断する。クラスタの共用リソースには、アクティブなクラスタ・メンバのみがアクセスできる。

#### 現在のボート (current votes)

特定のメンバから見た、現在のクラスタ・メンバとクォーラム・ディスクによるボートの数。

#### 高可用性 (highly available)

TruCluster Server ソフトウェアでは、単一のハードウェアまたはソフトウェアの障害に耐えられる能力。

ハードウェアとソフトウェアで、単一の障害 (システム障害、ディスク障害、または SCSI ケーブルの切断など) を保護している場合、そのクラスタは可用性が高いと考えられる。

サービスが依存しているハードウェアが単一の障害に対して保護されており、障害発生時にフェイルオーバーするようにサービスが構成されている場合、そのサービスは可用性が高いと考えられる。

#### 高速 SCSI (fast SCSI)

SCSI-2 のオプション・モード。最大転送速度は、10 MB/s。

小型コンピュータ・システム・インタフェース (**SCSI: small computer system interface**)

*SCSI (small computer system interface)* を参照

コンテキスト依存シンボリック・リンク (**CDSL: context-dependent symbolic link**)

*CDSL (context-dependent symbolic link)* を参照

## さ

サブセット (**subset**)

Tru64 UNIX の `setld` ソフトウェア・インストール・ユーティリティと互換性のある、インストール可能なソフトウェア・モジュール。

冗長性 (**redundant**)

ハードウェアを二重化して予備のリソースを確保し、構成要素の障害時に使用できるようにすることを表す用語。

省略時のクラスタ別名 (**default cluster alias**)

クラスタのインストール時に作成される特別なクラスタ別名。すべてのクラスタ・メンバは、省略時の設定により、省略時のクラスタ別名のメンバである。

処理スクリプト (**action script**)

アプリケーションの起動方法、停止方法、チェック方法を制御するために CAA が使用するシェル・スクリプト。省略時の設定では、処理スクリプトは、`/var/cluster/caa/script` ディレクトリにある。処理スクリプトのファイル名は、`resource_name.scr` という形式である。

シングル・インスタンス・アプリケーション (**single-instance application**)

一時点では 1 つのクラスタ・メンバだけで実行されるアプリケーション。CAA (Cluster Application Availability) サブシステムを使用すると、シングル・インスタンス・アプリケーションの最初のスタートアップとフェイルオーバー特性が制御されるため、可用性が高くなる。

シングルエンド SCSI バス (**single-ended SCSI bus**)

データ線 1 本とグランド線 1 本を使用してデバイスを接続する信号バス。この伝送方式は経済的であるが、ディファレンシャル SCSI バスよりもノイズの影響を受けやすい。

**シングル・サーバ・デバイス (single-server device)**

1 つのメンバからしかアクセスできないデバイス。

ダイレクト・アクセス入出力デバイス (*direct-access I/O device*) も参照

**シングルレール (single rail)**

物理レールと論理レールに 1 対 1 の関係がある Memory Channel の論理レール構成。この構成には、フェイルオーバ・プロパティはない。物理レールに障害が発生すると、論理レールにも障害が発生する。

**信号変換器 (signal converter)**

シングルエンド SCSI バスとディファレンシャル SCSI バスの間の変換を行うデバイス。

**接続マネージャ (connection manager)**

クラスタへのシステムの参加状況を調整し、システムがクラスタに参加したりクラスタから外れたときのクラスタの完全性を維持する、クラスタ・ソフトウェア構成要素。

**選択の重み (selection weight)**

選択の優先順位が同じ次の別名メンバに接続を担当させるまで、このメンバが (平均で) 担当する接続数。

**選択の優先順位 (selection priority)**

別名のどのメンバが新しい接続要求を受信するかの順番を決定するためにクラスタ別名に割り当てられる優先順位。選択の優先順位は、別名のメンバ間に階層を確立する。接続要求は、最も高い優先順位値を共有するメンバ間に分散される。

**専用ストレージ (private storage)**

専用バス上のストレージ・デバイス。ストレージ・デバイスには、ハード・ディスク、フロッピー・ディスク、テープ・ドライブ、その他のデバイスがある。

**専用バス (private bus)**

専用ストレージをローカル・システムに接続するバス。

**専用ポート (dedicated port)**

ロックされたポート (*locked port*) を参照

た

**ターミネータ (terminator)**

SCSI バスをターミネートするためのレジスタ・アレイ・デバイス。SCSI バスは、バスの両端でターミネートする必要がある。

**ダイレクト・アクセス入出力デバイス (direct-access I/O device)**

複数のクラスタ・メンバから同時にアクセスできる入出力デバイス。  
シングル・サーバ・デバイス (*single-server device*) も参照

**ダイレクト・アクセス・キャッシュド・リード (direct-access cached reads)**

AdvFS ファイル・システムの性能強化機能。CFS は、ダイレクト・アクセス・キャッシュド・リードを使用することにより、同時に複数のクラスタ・メンバに代わって、ストレージから直接読みとることができる。

**ディファレンシャル SCSI バス (differential SCSI bus)**

信号のレベルが、2 本の信号線の電圧差によって決まる SCSI バス。

**デバイス要求ディスパッチャ (device request dispatcher)**

クラスタ内のストレージ・デバイスに対するすべての入出力を制御するカーネル・サブシステム。デバイス要求ディスパッチャは、文字型ディスク・デバイスおよびブロック型ディスク・デバイスの両方に対するクラスタ単位のアクセスをサポートする。

注意: デバイス要求ディスパッチャと、TruCluster Production Server 製品の DRD (Distributed Raw Disk) サービスを混同しないこと。デバイス要求ディスパッチャは、完全にカーネルに統合されており、クラスタ・メンバからストレージにアクセスできるようにするための特別なサービスは必要としない。

**投票メンバ (voting member)**

ボートを持つクラスタ・メンバ。  
非投票メンバ (*nonvoting member*) も参照

**トライリンク・コネクタ (trilink connector)**

2 本のケーブルを 1 つのデバイスに接続したり、アダプタの外部または RAID コントローラの外部で共用 SCSI バスをターミネートできるようにするコネクタ。

### トンネリング (tunneling)

クラスタ別名に関連した用語で、受信後に mbuf チェーンをクラスタ・メンバ間で移動すること。

## な

### ネットワーク・ルート (network route)

クラスタ別名のルーティングに関する用語であり、クラスタ別名 IP アドレスが 1 つ以上存在する仮想サブネットへの公開ルートのこと (公開は RIP で行われる)。

### ノード・ボート (node votes)

メンバがクォーラムに投じるボートの定数。

## は

### パーソナリティ・モジュール (personality module)

ディファレンシャル SCSI バスと、ストレージ・シェルフのシングルエンド SCSI バスとの間のインタフェースとなる、ストレージ・シェルフ上のモジュール。モジュール上のスイッチにより、SCSI バスのターミネーションが有効になり、ストレージ・シェルフの SCSI バス ID が制御できる。

### 配置ポリシー (placement policy)

CAA の制御下にあるアプリケーションをどこで実行するかを決めるポリシー。サポートされているポリシーには、balanced、favored、restricted がある。

### バス (bus)

個別の並列回路で構成された、フラット・ケーブル、ツイスト・ワイヤ・ケーブル、またはバックプレーン。バスはコンピュータ・システムの構成要素を接続して、アドレス、データ、制御情報のための通信バスを形成する。

### 非投票メンバ (nonvoting member)

ボート数が 0 (ゼロ) のクラスタ・メンバ。

投票メンバ (*voting member*) も参照

### 標準ハブ・モード (standard hub mode)

Memory Channel ハブを使用して Memory Channel アダプタを接続する、Memory Channel インターコネクト構成。標準モードで Memory Channel インターコネクトを設定するには、リンク・ケーブルを使用して、各 Memory

Channel アダプタを Memory Channel ハブに取り付けられているラインカードに接続する。

#### ファイル・システムのパーティショニング (file system partitioning)

AdvFS ファイル・システムを、1 つのクラスタ・メンバからだけアクセスできるようにマウントすること。

ファイル・システムのパーティショニングは、TruCluster Production Server Software および TruCluster Available Server Software のバージョン 1.5 またはバージョン 1.6 からの移行を容易にするために提供されており、ファイル・システムのアクセスを単一のクラスタ・メンバに制限する汎用的な方法としては作られていない。

#### フェイルオーバー (failover)

サービスを提供する責任の移動。フェイルオーバーは、ハードウェアまたはソフトウェアの障害により、サービスが他のメンバ・システムで再起動されたときに発生する。

#### フェイルオーバー・ペア (failover pair)

片方がアクティブでもう片方が非アクティブの 2 つの物理レールで構成される Memory Channel の論理レール。アクティブ状態の物理レールに障害が発生すると、フェイルオーバーによって、非アクティブ状態の物理レールが使用される。

#### 物理レール (physical rail)

Memory Channel ハブおよびそのケーブルと、Memory Channel アダプタおよび各ノード上のアダプタ用の Memory Channel ドライバ。

論理レール (*logical rail*) も参照

#### プロキシ ARP (proxy ARP)

共通サブネット上にアドレスがあるクラスタ別名宛の要求を処理するためにクラスタ・メンバが使用するメカニズム。アドレス解決プロトコル (ARP) は、IP アドレスとイーサネット・アドレスを動的にマッピングする。ARP 要求には、ターゲット・ホスト上のインタフェースの IP アドレスが含まれる。この IP アドレスを認識するホストは、そのインタフェースのイーサネット・アドレスで応答しなければならない。他のホストはすべて、ARP 要求を無視しなければならない。

プロキシ ARP は、本質的には、システムまたはルータが、ARP 要求内の IP アドレスと一致するインタフェースを持つシステムのふりをするという

ことである。プロキシ ARP システムは、自分のイーサネット・アドレスを返すことで ARP 要求に応答する。そしてそのシステムは、実際のターゲット・システムにパケットをルーティングする。プロキシ ARP は、サブネット化するときや、ルータの使用がまだ構成されていないホストがあるトポロジにルータを追加する場合に便利である。

別名に対するプロキシ ARP マスタとして動作しているクラスタ・メンバが、自分のネットワーク・インタフェースの 1 つを使用して、クラスタ別名 IP アドレスの ARP 要求に応答する。

#### 分散型アプリケーション (**distributed application**)

目的によって異なるメンバを使用する、クラスタ上での実行用に設計されたアプリケーション。これらのアプリケーションは、Memory Channel、分散ロック・マネージャ (DLM)、およびクラスタ別名のアプリケーション・プログラミング・インタフェースを使用して、アプリケーションとクラスタ・リソースを統合する。

#### 分散ロック・マネージャ (**DLM: distributed lock manager**)

*DLM (distributed lock manager)* を参照

#### 別名ルータ (**alias router**)

クラスタの別名アドレスをネットワーク上で使用できるようにし、その別名に対する着信パケットを受信するクラスタ・メンバ。省略時の設定により、クラスタ・メンバはすべてブート時に別名ルータとして構成される。

#### ボート (**votes**)

*クォーラム (quorum)* を参照

#### ホスト・ルート (**host route**)

クラスタ別名のルーティングに関する用語であり、クラスタ・ノード上のローカル IP アドレスを経由したクラスタ別名 IP アドレスへの公開ルートのこと (公開は RIP で行われる)。サブネット・マスクは「オール 1」にして使用される。

#### ホット・スワップ (**hot swap**)

共用バスをアクティブにしたままそのバスに接続されているデバイスを交換できる機能。



## ま

### マルチ・インスタンス・アプリケーション (multi-instance application)

複数のクラスタ・メンバで同時に実行できるアプリケーション。定義により、マルチ・インスタンス・アプリケーションは、1つのクラスタ・メンバに障害が発生しても、他のメンバで実行されているアプリケーションのインスタンスには影響しないため、可用性が高いアプリケーションである。

### メンバ (member)

クラスタ・メンバ (*cluster member*) を参照

### メンバ ID (member ID)

クラスタ・メンバ・システム識別する整数 (1 ~ 63)。各メンバには、インストール・プロシージャ中に割り当てられた一意のメンバ ID がある。

## ら

### リソース (resource)

エンド・ユーザまたは他のソフトウェア構成要素に対してサービスを行う、クラスタのハードウェア構成要素またはソフトウェア構成要素。リソースには、ディスク、テープ、ファイル・システム、ネットワーク・インタフェース、アプリケーション・ソフトウェアなどがある。

### リソース・プロファイル (resource profile)

アプリケーションのリソース要件が入っているファイル。このファイルには、リソースのモニタリングや、アプリケーションのフェイルオーバーの制御のために CAA が使用するキーワードと値のペアが含まれている。CAA の制御下にあるアプリケーションごとにリソース・プロファイルがある。リソース・プロファイルは、`/var/cluster/caa/profile` ディレクトリにある。リソース・プロファイルのファイル名は、`resource_name.cap` という形式である。

### リソース・マネージャ (resource manager)

クラスタ・メンバ上で実行されているすべての CAA デーモン。これらのデーモンは独立しているが、相互に通信して、リソースの状態に関する情報を共有している。

リソース・マネージャは、CAA サブシステムのすべての構成要素に加え、接続マネージャおよびイベント・マネージャ (EVM) とも通信する。また、

リソース・マネージャは、特定のタイプのリソースの状態を監視するリソース・モニタも使用する。

**リソース・モニタ (resource monitor)**

ブート時にリソース・マネージャによってロードされるモニタ。各タイプのリソース (アプリケーション、ネットワーク、テープ、メディア・チェンジャ) に対して 1 つのリソース・モニタがある。

**ルータの優先順位 (router priority)**

共通サブネット上のクラスタ別名に対してプロキシ ARP ルータの行う選択を制御する方法。共通サブネット内の各別名について、その別名に対するルータ優先順位の最も高いクラスタ・メンバが、プロキシ ARP 要求の別名に応答する。

**ルーティング情報プロトコル (RIP: routing information protocol)**

*RIP (routing information protocol)* を参照

**ローカル・バス (local bus)**

専用バス (*private bus*) を参照

**ローリング・アップグレード (rolling upgrade)**

クラスタの操作中にクラスタのソフトウェア・アップグレードを実行すること。クラスタがベース・オペレーティング・システム、クラスタ、WLS (ワールドワイド言語サポート) の混在バージョン環境を透過的に維持しながら、一度に 1 つのメンバ、または複数のメンバが並列に、ローリングされて、元に戻される。クライアントは、ローリング・アップグレードが実行されているときでもサービスにアクセスできる。

**ロック・ファイル (lock file)**

1 つまたは複数の他のファイルに対する動作が、制限または禁止されていることを示すファイル。ロック・ファイルの存在を指示として使用したり、ロック・ファイルに制限の性質を示す情報を記述しておくことができる。

**ロックされたポート (locked port)**

クラスタ内の単一のノードによって専用に使われるクラスタ単位のポート・スペースにあるポート。

**論理レール (logical rail)**

1 つまたは複数の Memory Channel 物理レール。論理レールは、シングルレールまたは フェイルオーバ・ペアとして構成される。

論理ユニット番号 (**LUM: logical unit number**)

*LUN (logical unit number)* を参照

わ

ワールドワイド ID (**worldwide ID**)

*WWID (worldwide ID)* を参照



## A

<b>ACL</b> .....	9-4
<b>Advanced File System</b> ( AdvFS を参照 )	
<b>AdvFS</b>	
CFS クライアントでのキャッシング グ .....	2-14
上層の CFS.....	2-1
読み取り/書き込みでサポート	2-3t
<b>aliasd デーモン</b>	
gated 構成ファイルの自動維 持 .....	6-10, 6-12
RIP のサポート .....	6-3
定義 .....	6-3
<b>AlphaServer</b>	
DS10L.....	7-8
<b>API</b>	
DLM.....	7-2
Memory Channel .....	7-4t, 7-8
<b>ATM LANE</b> .....	7-12

## B

<b>balanced</b> 配置ポリシ .....	5-11
-----------------------------	------

## C

<b>CAA</b> .....	5-1
------------------	-----

caad デーモン.....	5-3
クラスタ別名との比較 .....	6-18
処理スクリプト .....	5-5
配置ポリシ.....	5-11
リソース・プロファイル .....	5-4
リソース・マネージャ .....	5-3
リソース・モニタ.....	5-4
<b>caad</b> デーモン .....	5-3
<b>CD-ROM</b> .....	2-3t
<b>CD-ROM</b> ファイル・システム ( CDFS を参照 )	
<b>CDFS</b>	
読み取り専用でサポート .....	2-3t
<b>CDSL</b> .....	2-15
<b>CFS</b> .....	2-7
cfsmgr コマンド .....	2-7
FAQ.....	2-12
X/Open および POSIX のセマンティ クスに従う .....	2-7
強化 .....	2-9
クライアントによる AdvFS プロッ クのキャッシング .....	2-14
クラスタ・インターコネクトの使 用 .....	7-2, 7-4
サーバの負荷を分散する .....	2-14
層が AdvFS の上.....	2-1
直接接続されているストレージへの 入出力.....	2-14

読み取りアクセス ..... 7-4

**clu\_add\_member** コマンド

期待ポートの自動調整 ..... 3-4

**clu\_alias.config** ファイル ..... 6-4

**clu\_create** コマンド

期待ポートの自動調整 ..... 3-4

**clu\_delete** コマンド

期待ポートの自動調整 ..... 3-4

**clu\_quorum** コマンド

期待ポートの値の表示 ..... 3-4

期待ポートの自動調整 ..... 3-4

クォーラム・ディスクの定義 3-10

**clu\_upgrade** コマンド ..... 9-2

**clua\_registerservice()** 関数... 6-18

**clua\_services** ファイル ..... 6-4,  
6-18, 6-24

**cluamgr** コマンド ..... 6-4

別名の指定と参加の例 ..... 6-10

**cluster\_expected\_votes** 属性 .. 3-4

**cluster\_node\_votes** 属性 ..... 3-3

**cluster\_qdisk\_votes** 属性 ..... 3-3

**clusvc\_getcommport()** 関数 .. 6-32

**clusvc\_getresvcommport()** 関  
数 ..... 6-32

## D

**DLM** ..... 8-1

クラスタ・インターコネクトの使  
用 ..... 7-2

**drdmgr** コマンド

例 ..... 2-11

**DVD-ROM** ..... 2-3t

**DVDFS**

読み取り専用でサポート ..... 2-3t

## E

**/etc/clua\_services** ..... 6-4, 6-18

**/etc/services** との比較 ..... 6-24

**/etc/exports.aliases** ..... 6-4

**/etc/gated.conf.member<n>...** 6-3

aliasd による作成と維持 ..... 6-12

**EVM** ..... 5-6

**exports.aliases** ファイル ..... 6-4

## F

**favored** 配置ポリシー ..... 5-11

**FDDI** ..... 7-12

**FFM**

ローカル使用のみサポート... 2-3t

**File-on-File Mounting** ファイル・  
システム  
( FFM を参照 )

## G

**gated.conf.member<n>** ファイ  
ル ..... 6-3

**gated.conf file** ..... 6-3

**gated** デーモン ..... 6-3

プロキシ ARP に対するホスト・  
ルートおよびネットワーク・ルー  
トの優先 ..... 6-11

## H

**hwmgr** コマンド ..... 2-21

## I

**ifconfig** 別名

クラスタ別名との概念上の類似性 .....	6-2
クラスタ別名との比較 .....	6-33
<b>in_multi</b>	
サービス .....	6-18
属性 .....	6-24
<b>in_noalias</b> 属性 .....	6-25
<b>in_nolocal</b> 属性 .....	6-25
<b>in_single</b>	
サービス .....	6-18
属性 .....	6-24

## L

<b>lagconfig</b> コマンド	
LAN インターコネクトでは非サポート .....	7-14
<b>LAN</b> インターコネクト	
Memory Channel との比較 ....	7-3
規則と制約 .....	7-12
ハードウェア要件 .....	7-12
<b>Logical Storage Manager</b>	
(LSM を参照)	
<b>LSM</b>	
クラスタのサポート .....	2-22
サイト間のデータの複製は非サポート .....	7-9

## M

<b>Memory Channel</b> .....	7-9
LAN との比較 .....	7-3
遅延と分散処理の利点 .....	7-8
物理ルール .....	7-10

論理ルール .....	7-10
<b>Memory Channel</b> インターコネクトと LAN インターコネクトの比較 .....	7-3
<b>MFS</b>	
ローカル使用でのみ読み取り/書き込みサポート .....	2-3t

## N

<b>New Hardware Delivery</b>	
(NHD を参照)	
<b>NFS</b> .....	7-2, 7-6
exports.aliaes ファイル .....	6-4
クラスタ別名との関わり .....	6-27
他の別名の使用 .....	6-4
読み取り/書き込みでサポートされるクライアント .....	2-4t
読み取り/書き込みでサポートされるサーバ .....	2-4t
<b>NHD</b> .....	9-2

## O

<b>O_DIRECTIO</b> フラグ ....	2-9, 2-13
<b>OPS</b>	
直接入出力によるディスクの書き込み .....	7-5
<b>out_alias</b> 属性 .....	6-25

## P

<b>PC-NFS</b>	
読み取り/書き込みでサポート .....	2-4t

**PID** ..... 1-7t  
**portmap** デーモン ..... 6-33  
**/proc** ファイル・システム  
ローカル使用のみサポート... 2-4t

## R

**raw** 入出力..... 2-13  
**restricted** 配置ポリシ ..... 5-11  
**RIP**  
aliasd のサポート ..... 6-3  
**RPC** サービス  
クラスタ別名との相互作用... 6-32  
**rpri** 属性..... 6-11, 6-22

## S

**selp** 属性..... 6-22  
**selw** 属性 ..... 6-23  
**services**  
RPC..... 6-32  
**services** ファイル ..... 6-4  
**static** 属性 ..... 6-26  
**STP** ..... 7-14

## U

**UFS**  
クラスタ単位で読み取り専用でサ  
ポート..... 2-5t  
ローカルでのみ読み取り/書き込み  
サポート ..... 2-5t  
**UNIX** ファイル・システム  
( UFS を参照 )

## V

**vMAC**..... 6-26  
**VPN**..... 7-12

## W

**WAN** ..... 7-14

## あ

アクセス制御リスト  
( ACL を参照 )  
アダプタ  
DE50x ..... 7-13  
DE60x ..... 7-13  
DEGPA-xx..... 7-13  
イーサネット ..... 7-12  
アプリケーション  
CAA による高可用性..... 5-1  
クラスタ・インターコネクトの使  
用 ..... 7-2, 7-5  
クラスタ別名による要求のルーティ  
ング ..... 6-18  
タイプ ..... 4-1

## い

イベント・マネージャ  
( EVM を参照 )  
インストレーション ..... 9-1  
手順の概要..... 9-2  
イーサネット・アダプタ ..... 7-12

## か

書き込みアクセス



CFS.....	7-4
仮想サブネット	
使用する場合 .....	6-8
定義済み.....	6-8
ネットワーク・ルートの公開 .....	6-12
ホスト・ルートの公開 .....	6-12
監査ログ .....	9-4
完全な冗長 LAN インターコネク	
ト.....	7-13
管理.....	9-3

## き

期待ポート	
クラスタ.....	3-3, 3-5
計算 .....	3-5
メンバ固有.....	3-3, 3-5
期待ポートの調整 .....	3-4
規模	
クラスタ.....	7-3t, 7-7
共通サブネット	
使用する場合 .....	6-8
定義済み.....	6-7
プロキシ ARP.....	6-10
ホスト・ルートの公開 .....	6-10
距離の比較	
Memory Channel インターコネク	
と LAN インターコネク	7-3t

## く

クォーラム .....	3-5
アルゴリズム .....	3-5
計算 .....	3-5

ロス .....	3-6
クォーラム・ディスク	
LSM .....	3-12
構成 .....	3-11
使用 .....	3-7
ポート .....	3-3
ポートの数.....	3-11
クォーラム・ポート.....	3-6
クラスタ・アプリケーションの可用性	
(CAA を参照)	
クラスタ・インターコネク	
規則と制約事項 .....	7-8
選択 .....	7-1
定義された.....	7-1
クラスタの規模	
クラスタ・インターコネクの帯域	
幅要件.....	7-7
クラスタ・ファイル・システム	
(CFS を参照)	
クラスタ別名.....	6-1
aliasd デーモン .....	6-3
ARP 要求 .....	6-10
CAA との比較.....	6-18
ifconfig 別名との比較 .....	6-33
IP アドレスの位置の制限 .....	6-9
NFS 要求の処理 .....	6-27
vMAC サポート .....	6-26
仮想サブネット .....	6-8
共通サブネット .....	6-7
クライアントのルーティング・テー	
ブル .....	6-11
クラスタ・インターコネクの使	
用 .....	7-2, 7-6
サブシステムの構成要素 .....	6-3

指定と参加.....	6-10
省略時の.....	6-2, 6-5
パケットのリダイレクション	6-13
必要な追加の決定.....	6-6
ルーティング.....	6-9
クラスタ別名のサービス属性	
in_multi.....	6-24
in_noalias.....	6-25
in_nolocal.....	6-25
in_single.....	6-24
out_alias.....	6-25
static.....	6-26
クラスタ別名の属性	
選択の重み.....	6-23
選択の優先順位.....	6-22
ルータの優先順位.....	6-22
クラスタ別名へのルーティング..	6-9
クラスタ・メンバ数の変化 .	7-1, 7-8
クロス・ケーブル.....	7-13

## け

現在のポート.....	3-4, 3-6
ケーブル接続の比較	
Memory Channel インターコネクト	
と LAN インターコネクト .	7-3t

## こ

構成	
完全な冗長 LAN.....	7-13
制約.....	7-12
対称性の重要性.....	7-14
コスト比較	
Memory Channel インターコネクト	
と LAN インターコネクト .	7-3t

コンテキスト依存シンボリック・リンク	
( CDSL を参照 )	

## さ

サブネット	
仮想.....	6-8
共通.....	6-7
サービス	
in_single および in_multi.....	6-18

## し

冗長 <b>LAN</b> インターコネクト ...	7-13
冗長性	
Memory Channel と LAN インター	
コネクトの比較.....	7-4t
省略時のクラスタ別名.....	6-5
追加別名の使用.....	6-2
処理スクリプト.....	5-5, 5-13
シングル・インスタンス・アプリケーション	
.....	4-1
シングル・システム管理.....	9-3

## す

スイッチ	
完全な冗長 LAN インターコネクト	
での複数使用.....	7-13
ハブとの比較.....	7-13n
要件.....	7-12
スクリプト	
処理.....	5-5
スケーラビリティ	
スイッチの利点.....	7-13n

スパニング・ツリー・プロトコル  
( STP を参照 )

## せ

接続マネージャ ..... 3-1  
    クラスタ・インターコネクトの使  
    用 ..... 7-1, 7-7  
選択の重みの属性 ..... 6-23  
選択の優先順位の属性 ..... 6-22  
全二重モード ..... 7-13n

## た

帯域幅の比較  
    Memory Channel インターコネクト  
    と LAN インターコネクト . 7-3t  
ダイレクト・アクセス・キャッシュ  
    ド・リード ..... 2-10  
単一のシステム管理 ..... 1-6

## ち

直接入出力 ..... 2-9  
    OPS ..... 7-5

## て

ディスクの書き込み  
    CFS ..... 7-2, 7-4  
    OPS ..... 7-5  
デバイス名  
    drdmgr コマンド ..... 2-11  
    新しいスタイルの例 ..... 2-19

新しい命名モデル ..... 2-18  
クラスタ単位での一貫性 ..... 2-18  
クラスタでの新しいスタイルのサ  
    ポート ..... 2-18  
決定 ..... 2-18  
識別 ..... 2-20  
ディスク ..... 2-19  
テープ ..... 2-20  
デバイス要求ディスパッチャ ... 2-11  
    drdmgr コマンド ..... 2-11  
    FAQ ..... 2-12  
デーモン  
    aliasd ..... 6-3  
    caad ..... 5-3

## と

同期  
    書き込み ..... 7-5  
トラフィック  
    アプリケーション ..... 7-5  
    クラスタ別名 ..... 7-6  
    ストレージ ..... 7-4

## な

名前付きパイプ  
    ローカル使用のみサポート ... 2-4t

## に

入出力  
    raw ..... 2-13

クラスタ・インターコネクトの使用 ..... 7-2, 7-4  
ストレージに直接入出力する場合 ..... 2-14  
ブロック ..... 2-13

## ね

ネットワーク・ファイル・システム  
( NFS を参照 )  
ネットワーク・ルート ..... 6-12

## は

配置ポリシー  
balanced ..... 5-11  
favored ..... 5-11  
restricted ..... 5-11  
パイプ  
ローカル使用のみサポートされる名前付きパイプ ..... 2-4t  
パッチ ..... 9-1  
ハブ  
スイッチとの比較 ..... 7-13n  
要件 ..... 7-12  
半二重モード ..... 7-13n  
パーティショニング  
ファイル・システム ..... 1-6t

## ふ

ファイル  
64 K バイト以上の読み取り .. 2-13  
O\_DIRECTIO を指定したオープン ..... 2-13  
ファイル・システム

クラスタでのサポート ..... 2-3  
ファイル・システムのパーティショニング ..... 1-6t  
負荷分散  
CFS サーバ ..... 2-14  
クラスタ別名の複数使用 ..... 6-7  
別名属性の使用法 ..... 6-23  
復元リンク ..... 7-13  
物理ルール ..... 7-10  
プロキシ **ARP**  
ARP 要求に対する応答 ..... 6-10  
ホスト・ルートおよびネットワーク・ルートの優先 ..... 6-10  
無償 ARP パケットと vMAC . 6-26  
プロセス **ID**  
( PID を参照 )  
ブロック入出力 ..... 2-13  
ブロードキャスト・アドレス .... 6-9  
分散型アプリケーション ..... 4-1  
分散処理 ..... 7-5  
分散ロック・マネージャ  
( DLM を参照 )  
分断 ..... 3-2  
ブート・パーティション  
クラスタ・メンバのブート・デバイス ..... 2-7

## へ

別名  
( クラスタ別名 , ifconfig 別名 を参照 )  
別名への参加  
定義 ..... 6-2  
と別名指定 ..... 6-10

## ほ

---

- ホスト・ルート
  - 仮想サブネット上の別名 ..... 6-12
  - 共通サブネット上の別名 ..... 6-11
- ポート
  - 期待 ..... 3-3
  - クォーラム ..... 3-6
  - クォーラム・ディスク ..... 3-3
  - 現在の ..... 3-6
  - 調整 ..... 3-4
  - ノード ..... 3-2

## ま

---

- マルチ・インスタンス・アプリケーション ..... 4-1
- マルチキャスト・アドレス ..... 6-9

## め

---

- メッセージ
  - クラスタ・インターコネクトの使用 ..... 7-1
- メディア・アクセス制御 ..... 6-26
- メモリ・ファイル・システム
  - ( MFS を参照 )
- メンバ ..... 3-2

## よ

---

- 読み取りアクセス
  - CFS ..... 7-4

## り

---

- リソース
  - CAA ..... 5-7
- リソース・プロファイル ..... 5-4
- リソース・マネージャ ..... 5-3
- リソース・モニタ ..... 5-4
- リモート・ファイル・システム
  - クラスタ・インターコネクトを経由したアクセス ..... 7-2
- リモート・プロシージャ・コール
  - ( RPC を参照 )
- リンク集約 ..... 7-13
- リンク復元 ..... 7-13

## る

---

- ルータの優先順位の属性 ..... 6-22
- ルーティング情報プロトコル
  - ( RIP を参照 )

## ろ

---

- ログ
  - 監査 ..... 9-4
- 論理ルール ..... 7-10
- シングルルール ..... 7-10
- フェイルオーバー・ペア ..... 7-10
- ローリング・アップグレード ... 1-6t, 9-1

## わ

---

- ワールドワイド ID ..... 2-21



# マニュアルに対するご意見

## TruCluster Server

### クラスタ概要

AA-RM87D-TE

弊社のマニュアルに関して、ご意見、ご要望、または内容の不明確な部分など、お気づきの点がございましたら、下記にご記入の上、弊社社員にお渡しくださるようお願い申し上げます。

マニュアルの採点：

	大変良い	良い	普通	良くない
正確さ(説明どおりに動作するか)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
情報量(十分か)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
分かり易さ	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
マニュアルの構成	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
図(役立つか)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
例(役立つか)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
索引(項目の検索性)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ページ・レイアウト(情報の検索性)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

内容の不明確な部分がありましたら、以下にご記入ください：

ペー ジ


その他お気づきの点がございましたら、以下にご記入ください：


ご使用のソフトウェアのバージョン： \_\_\_\_\_

貴社名/部課名 \_\_\_\_\_

御名前 \_\_\_\_\_

記入日 \_\_\_\_\_

(注) 当用紙を受け取った弊社社員は、すみやかに下記にお送りください。

ビジネスクリティカルシステム統括本部 **BCS** 技術本部 **Alpha** ソフトウェア技術部