

Tru64 UNIX

システムの構成とチューニング

Part Number: AA-RM7AC-TE

2003 年 1 月

ソフトウェア・バージョン: Tru64 UNIX Version 5.1B 以上

本書では、Tru64 UNIX をチューニングして、オペレーティング・システムの性能を改善する方法を説明し、Oracle、ネットワーク・ファイル・システム (NFS)、Web サーバ・アプリケーションで推奨されるチューニングを示します。また、Tru64 UNIX オペレーティング・システムの各コンポーネントの推奨チューニングについて説明します。

© 2003 日本ヒューレット・パッカート株式会社

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。

Microsoft[®]、Windows NT[®] は米国 Microsoft 社の登録商標です。Intel[®]、Pentium[®]、Intel Inside[®] は米国 Intel 社の登録商標です。UNIX[®]、The Open Group[™] 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。

Oracle[®] は、Oracle Corporation の登録商標です。Oracle9i[™] および Oracle8i[™] は、Oracle Corporation の商標です。

原典: System Configuration and Tuning (AA-RH9GC-TE)
 Copyright © 2002 Hewlett-Packard Company

目次

まえがき

Part 1 システム・チューニングの概要

1 システム・チューニングの概要

1.1	ハードウェア構成	1-1
1.1.1	ハードウェア構成の概要	1-2
1.2	性能に関する用語と概念	1-3
1.3	ディスク・ストレージ・リソース	1-5
1.3.1	RAID 技術	1-5
1.3.2	SCSI の概念	1-8
1.3.2.1	データ・パス	1-8
1.3.2.2	SCSI バス速度	1-9
1.3.2.3	転送方式	1-10
1.3.2.4	UltraSCSI バス・セグメントの拡張	1-12
1.3.2.5	SCSI バスの長さとターミネーション	1-12
1.3.3	ファイバ・チャネル	1-13
1.3.3.1	ファイバ・チャネル・トポロジ	1-14
1.3.3.1.1	ポイント・ツー・ポイント・トポロジ	1-14
1.3.3.1.2	ファブリック・トポロジ	1-15
1.3.3.1.3	調停ループ・トポロジ	1-16
1.3.3.2	ファイバ・チャネル・トポロジの比較	1-17
1.3.3.3	ゾーニング	1-19
1.3.3.3.1	スイッチ・ゾーニングとセレクトティブ・ストレージ・プレゼンテーション	1-19
1.3.3.3.2	ゾーニングの種類	1-20

1.3.3.3.3	ゾーニングの例	1-21
1.3.3.4	スイッチのカスケード接続	1-22
1.4	ネットワーク・リソース	1-24
1.4.1	ネットワーク・サブシステム	1-25
1.4.2	冗長ネットワークの使用	1-26
1.4.3	NetRAIN	1-26
1.4.4	ルーティング	1-27
1.4.5	LAG インタフェース	1-28
1.5	ファイル・システム・リソース	1-29
1.5.1	AdvFS の使用	1-29
1.5.1.1	UBC の使用	1-29
1.5.2	NFS の使用	1-30
1.6	メモリ・リソース	1-31
1.6.1	ページングとスワッピング	1-32
1.6.2	データのキャッシング	1-33
1.7	CPU リソース	1-33
1.8	作業負荷に適したリソース・モデルの明確化	1-36
1.9	一般的なチューニング対象サブシステム	1-37

2 システム情報と性能情報の取得

2.1	性能問題の解決のための方法論的アプローチ	2-2
2.2	システム・イベントについての情報の取得	2-3
2.2.1	イベント・マネージャの使用	2-4
2.2.2	DECevent の使用	2-5
2.2.3	Compaq Analyze の使用	2-5
2.2.4	システム課金とディスク・クォータの使用	2-6
2.3	情報収集用の基本的なツール	2-7
2.3.1	hwmgr ユーティリティを使用したハードウェア情報の収集	2-7

2.3.2	collect ユーティリティを使用したシステム情報の収集 ...	2-9
2.3.2.1	collect をシステム・リブート時に自動的に起動するための構成	2-11
2.3.2.2	collect データ・ファイルのプロット	2-12
2.3.3	sys_check ユーティリティを使用した構成のチェック	2-13
2.4	情報収集用の補助的なツール	2-14
2.4.1	lockinfo ユーティリティを使用したロック統計情報の収集	2-15
2.4.2	sched_stat ユーティリティを使用した CPU 利用率とプロセス統計情報の収集	2-16
2.4.3	nfsstat ユーティリティを使用したネットワークと NFS 統計情報の収集	2-17
2.4.4	tcpdump ユーティリティを使用した情報の収集	2-18
2.4.5	netstat コマンドを使用したネットワーク統計情報のモニタリング	2-20
2.4.5.1	入出力エラーと衝突	2-21
2.4.5.2	デバイス・ドライバのエラー	2-22
2.4.5.3	メモリの利用率	2-22
2.4.5.4	ソケット接続	2-23
2.4.5.5	廃棄されたパケットまたは紛失したパケット	2-24
2.4.5.6	再送, 順序の不正なパケット, および不正チェックサム	2-25
2.4.5.7	ルーティングの統計情報	2-27
2.4.5.8	プロトコルの統計情報	2-27
2.4.6	ps axlmp を使用した NFS サーバ側の情報収集	2-29
2.4.7	nfsiod を使用した NFS クライアント側の情報収集	2-29
2.4.8	nfswatch コマンドを使用した NFS サーバの受信ネットワーク・トラフィックのモニタリング	2-30
2.5	その他の性能モニタリング・ツール	2-31
2.6	プロファイリング情報とデバッグ情報の収集	2-32

3 カーネル・サブシステム属性の表示と変更

3.1	オペレーティング・システムでサポートする属性	3-1
3.2	属性値の表示	3-2
3.3	属性値の変更	3-3
3.3.1	現在値	3-4
3.3.2	永久値	3-5

Part 2 アプリケーション・タイプ別のチューニング

4 Oracle のチューニング

4.1	Oracle 統計情報のモニタリング	4-1
4.2	gettimeofday() 関数の性能改善	4-2
4.3	IPC 通信プロトコルの選択と有効化	4-3
4.4	推奨するチューニング方法	4-4
4.4.1	仮想メモリ属性の変更	4-5
4.4.1.1	共用メモリを無効にする	4-5
4.4.1.2	共用メモリの割り当て	4-6
4.4.1.2.1	rad_gh_regions 属性を変更する	4-7
4.4.1.2.2	gh_chuncks 属性を変更する	4-8
4.4.1.3	UBC で使用する物理メモリの割合を変更する	4-8
4.4.1.4	UBC が借りるメモリの割合を変更する	4-9
4.4.1.5	UBC が 1 つのファイルに対して使用できるメモリの割合を変更する	4-9
4.4.1.6	UBC しきい値を変更する	4-9
4.4.1.7	ダーティ・ページの割合を変更する	4-10
4.4.1.8	スワップ割り当てモードを変更する	4-10
4.4.2	Advanced File System 属性の変更	4-11
4.4.3	仮想ファイル・システムの属性の変更	4-12
4.4.4	プロセス間通信属性の変更	4-13

4.4.4.1	System V の共用領域を変更する	4-13
4.4.4.2	System V 共用メモリ領域の最大サイズを変更する ...	4-13
4.4.4.3	System V 共用メモリ領域の最小サイズを変更する ...	4-14
4.4.4.4	一時期に使用できる共用メモリ領域の数を変更する ..	4-14
4.4.4.5	一時期に接続できる共用メモリ領域の数を変更する ..	4-14
4.4.5	インターネット属性の変更	4-14
4.4.5.1	UDP ソケット用送信バッファ・サイズを変更する ...	4-15
4.4.5.2	UDP ソケット用受信バッファ・サイズを変更する ...	4-15
4.4.5.3	システムが同時に送信接続を確立できる回数を変更する	4-15
4.4.6	プロセス属性の変更	4-15
4.4.6.1	プロセスごとのスタック・サイズを変更する	4-16
4.4.6.2	ユーザ・プロセスのスタック・サイズの最大サイズを変更する	4-16
4.4.6.3	プロセスごとのデータ・サイズを変更する	4-16
4.4.6.4	プロセスごとのデータ・サイズの最大サイズを変更する	4-17
4.4.6.5	プロセスごとのアドレス・サイズを変更する	4-17
4.4.6.6	プロセスごとのアドレス・サイズの最大サイズを変更する	4-18
4.4.6.7	プロセスの最大数を変更する	4-18
4.4.6.8	スレッドの最大数を変更する	4-18
4.4.6.9	システム・テーブルに割り当てる領域を変更する	4-18
4.4.7	リアルタイム属性の変更	4-19
4.4.8	Reliable Datagram 属性の変更	4-19
4.4.8.1	RDG 内のオブジェクトの最大数を変更する	4-20
4.4.8.2	RDG メッセージの最大サイズを変更する	4-20
4.4.8.3	RDG 内のメッセージの最大数を変更する	4-20
4.4.8.4	RDG テーブル内のセッションの最大数を変更する ...	4-20

4.4.8.5	メッセージ・パケット用に固定されるページの最大数 を変更する	4-20
4.4.9	メモリ・チャネル属性の変更	4-21
5 ネットワーク・ファイル・システムのチューニング		
5.1	NFS 統計情報のモニタリング	5-2
5.2	NFS 性能低下の検出	5-3
5.3	性能上の利点と欠点	5-4
5.4	NFS の構成	5-4
5.4.1	サーバ・スレッドの構成	5-4
5.4.2	クライアント・スレッドの構成	5-5
5.4.3	キャッシュ・タイムアウトの限界値の変更	5-6
5.5	NFS の再送	5-6
5.5.1	ネットワークのタイムアウトを減らす	5-7
5.6	NFS サーバのチューニング	5-7
5.6.1	NFS サーバ側の属性の変更	5-10
5.6.1.1	書き込みの集積	5-11
5.6.1.1.1	クライアントの書き込み要求に対する NFS サーバ の応答速度を改善する	5-12
5.6.1.2	サーバが書き込みを遅らせる時間を秒単位で指定する	5-13
5.6.1.3	NFS 送受信バッファのサイズを大きくする	5-15
5.7	NFS クライアントのチューニング	5-15
5.7.1	NFS クライアント側の属性の変更	5-16
5.7.1.1	読み取り性能を改善する	5-17
5.7.1.2	クライアントが再送を開始するまでの時間を制御する	5-17
5.7.1.3	ディレクトリ名の検索用キャッシュ (DNLC)	5-18
5.7.1.4	ネガティブ名キャッシュの検索 (NNC)	5-18
5.7.1.5	NFS クライアント間でのファイルの一貫性を指定する	5-19
5.7.1.6	NFS クライアントがファイル属性をフェッチする際の 動きを変える	5-20

6 インターネット・サーバのチューニング

6.1	インターネット・サーバの性能改善	6-1
6.1.1	ハードウェアの構成	6-2
6.1.2	メモリとスワップ領域の構成	6-2
6.1.3	IP アドレスのロギング	6-3
6.1.4	ネットワーク統計情報のモニタリング	6-4
6.1.5	ソケット統計情報のモニタリング	6-5
6.1.6	仮想メモリ統計情報のモニタリング	6-6
6.1.7	構成情報の収集	6-7
6.2	基本的な推奨チューニング	6-7
6.2.1	インターネット属性の変更	6-8
6.2.1.1	TCP ハッシュ・テーブルのサイズを大きくする	6-8
6.2.1.2	PMTU 検出を無効にする	6-9
6.2.1.3	送信接続ポート数を増やす	6-9
6.2.2	プロセス属性の変更	6-9
6.2.2.1	システム・テーブルとデータ構造体のサイズを大きく する	6-10
6.2.2.2	ユーザごとのプロセス数を増やす	6-11
6.2.2.3	ユーザごとのスレッド数を増やす	6-11
6.2.2.4	ユーザ・プロセスのデータ・セグメント・サイズの限 界値を大きくする	6-11
6.2.2.5	ユーザ・プロセスのアドレス空間の限界値を大きくす る	6-12
6.2.3	ソケット属性の変更	6-12
6.2.3.1	保留中 TCP 接続の最大数を増やす	6-12
6.2.3.2	保留中 TCP 接続の最小数を増やす	6-13
6.2.3.3	mbuf クラスタの圧縮を有効にする	6-13
6.3	高度な推奨チューニング	6-14

6.3.1	汎用属性の変更	6-14
6.3.2	インターネット属性の変更	6-15
6.3.2.1	TCP ハッシュ・テーブルの数を増やす	6-15
6.3.2.2	ハッシュ・パケットの数を増やす	6-16
6.3.2.3	TCP パーシャル接続タイムアウト限界値を変更する .	6-16
6.3.2.4	TCP 再送の速度を遅くする	6-17
6.3.2.5	TCP 持続機能を有効にする	6-17
6.3.2.6	TCP 接続コンテキスト・タイムアウトの頻度を高くする	6-18
6.3.2.7	送信接続ポートの範囲を変更する	6-19
6.3.2.8	IP 受信キューの数を増やす	6-20
6.3.2.9	IP 受信キューの最大長を大きくする	6-20
6.3.3	ネットワーク属性の変更	6-21
6.3.3.1	送信キューの最大長を大きくする	6-21
6.3.3.2	スクリーニング・キャッシュのミスを減らす	6-21
6.3.3.3	スクリーニング・バッファのドロップを減らす	6-22
6.3.4	ソケット属性の変更	6-23
6.3.5	仮想メモリ属性の変更	6-23

7 アプリケーション性能の管理

7.1	アプリケーションの性能改善	7-1
7.1.1	オペレーティング・システムの最新のパッチをインストールする	7-2
7.1.2	最新バージョンのコンパイラを使用する	7-2
7.1.3	並列処理を使用する	7-2
7.1.4	アプリケーションを最適化する	7-2
7.1.5	シェアード・ライブラリを使用する	7-3
7.1.6	アプリケーションが必要とするメモリ量を減らす	7-3
7.1.7	メモリ・ロックを制御する	7-4

Part 3 コンポーネント別のチューニング

8 システム・リソース割り当ての管理

8.1	プロセスの限界値のチューニング	8-1
8.1.1	システム・テーブルとデータ構造体を大きくする	8-2
8.1.2	プロセスの最大数を増やす	8-3
8.1.3	スレッドの最大数を増やす	8-4
8.2	プログラム・サイズの限界値のチューニング	8-5
8.2.1	ユーザ・プロセス・スタックのサイズを大きくする	8-6
8.2.2	ユーザ・プロセスのデータ・セグメント・サイズを大きく する	8-6
8.3	アドレス空間の限界値のチューニング	8-7
8.4	プロセス間通信の限界値のチューニング	8-8
8.4.1	System V メッセージの最大サイズを大きくする	8-10
8.4.2	System V のメッセージ・キューの最大サイズを大きくす る	8-10
8.4.3	System V のキュー上のメッセージ最大数を増やす	8-11
8.4.4	System V の共用メモリ領域の最大サイズを大きくする ..	8-12
8.4.5	1 つのプロセスにアタッチできる共用メモリ領域の最大数 を増やす	8-12
8.4.6	共用ページ・テーブルの共用を変更する	8-13
8.5	オープン・ファイルの限界値のチューニング	8-14
8.5.1	オープン・ファイルの最大数を増やす	8-14
8.5.2	オープン・ファイル記述子の最大数を増やす	8-15
8.6	Aurema ARMTech Suite	8-18

9 ディスク・ストレージ性能の管理

9.1	ディスク入出力負荷の分散に関するガイドライン	9-1
9.2	ディスク入出力の分散状況のモニタリング	9-3

9.2.1	iostat コマンドによるディスク使用状況の表示	9-4
9.3	LSM を使用したストレージの管理	9-6
9.3.1	LSM の機能	9-6
9.4	ハードウェア RAID サブシステムの性能の管理	9-7
9.4.1	ハードウェア RAID 機能	9-8
9.4.2	ハードウェア RAID 製品	9-10
9.4.3	ハードウェア RAID 構成のガイドライン	9-11
9.4.3.1	ストレージ・セットのディスクをバス間に分散させる	9-12
9.4.3.2	同じデータ容量のディスクを使用する	9-12
9.4.3.3	正しいハードウェア RAID ストライプ・サイズを選択 する	9-12
9.4.3.4	ストライプ・セットをミラーリングする	9-14
9.4.3.5	ライトバック・キャッシュを使用する	9-14
9.4.3.6	デュアル冗長コントローラを使用する	9-15
9.4.3.7	破損したディスクをスペア・ディスクと交換する	9-15
9.5	CAM 性能の管理	9-15
10	ネットワーク性能の管理	
10.1	ネットワーク情報の収集	10-1
10.1.1	sysconfig コマンドを用いてソケット・リッスン・キューの 統計情報をチェックする	10-4
10.2	ネットワーク・サブシステムのチューニング	10-4
10.2.1	TCP 制御ブロックの検索速度を速くする	10-7
10.2.2	TCP ハッシュ・テーブルの数を増やす	10-8
10.2.3	TCP ソケット・リッスン・キューの限界値をチューニング する	10-8
10.2.4	送信接続ポート数を増やす	10-10
10.2.5	送信接続ポートの範囲を変更する	10-11
10.2.6	PMTU の検出を無効にする	10-11
10.2.7	IP 入力キューの数を増やす	10-12

10.2.8	mbuf クラスタの圧縮を有効にする	10-13
10.2.9	TCP の keepalive 機能を有効にする	10-14
10.2.10	IP アドレスの検索速度を速くする	10-15
10.2.11	TCP のパーシャル接続タイムアウトの限界値を小さくする	10-16
10.2.12	TCP 接続コンテキストのタイムアウト限界値を小さくする	10-17
10.2.13	TCP 再送レートを下げる	10-18
10.2.14	TCP データの肯定応答の遅延を無効にする	10-19
10.2.15	TCP セグメントの最大サイズを大きくする	10-19
10.2.16	UDP ソケットの送受信バッファを大きくする	10-20
10.2.17	ソケット・バッファの最大サイズを大きくする	10-21
10.2.18	入力パケットのドロップを防止する	10-21
 11 ファイル・システム性能の管理		
11.1	キャッシュのチューニング	11-1
11.1.1	キャッシュ統計情報のモニタリング	11-2
11.1.2	namei キャッシュのチューニング	11-2
11.1.3	UBC のチューニング	11-5
11.1.4	メタデータ・バッファ・キャッシュのチューニング	11-8
11.1.5	AdvFS アクセス構造体のチューニング	11-10
11.2	AdvFS のチューニング	11-11
11.2.1	AdvFS の構成のガイドライン	11-11
11.2.1.1	RAID1 または RAID5 を用いてデータを格納する	11-13
11.2.1.2	同期書き込み要求を強制するか、永続的なアトミック書き込みデータ・ロギングを有効にする	11-13
11.2.1.3	直接入出力を有効にする	11-15
11.2.1.4	AdvFS を使用してファイルを分散させる	11-16
11.2.1.5	データをストライピングする	11-17

11.2.1.6	ファイル・ドメインの断片化を解消する	11-19
11.2.1.7	入出力転送サイズを小さくする	11-20
11.2.1.8	トランザクション・ログを移動する	11-21
11.2.2	AdvFS 統計情報のモニタリング	11-22
11.2.2.1	AdvFS の性能の統計情報を表示する	11-22
11.2.2.2	AdvFS ファイル・ドメイン内のディスクを表示する .	11-24
11.2.2.3	AdvFS ファイル・ドメインを表示する	11-25
11.2.2.4	AdvFS ファイル情報を表示する	11-25
11.2.2.5	ファイル・ドメイン内の AdvFS ファイルセットを表示 する	11-26
11.2.3	AdvFS キューのチューニング	11-27
11.3	UFS のチューニング	11-31
11.3.1	UFS の構成のガイドライン	11-31
11.3.1.1	ファイル・システムのフラグメント・サイズとブロッ ク・サイズを変更する	11-32
11.3.1.2	i ノードの密度を低くする	11-32
11.3.1.3	回転遅延時間を設定する	11-33
11.3.1.4	クラスタとして結合するブロックの数を増やす	11-33
11.3.1.5	MFS を使用する	11-33
11.3.1.6	UFS のディスク・クォータを使用する	11-34
11.3.1.7	UFS および MFS のマウントの数を増やす	11-34
11.3.2	UFS 統計情報のモニタリング	11-35
11.3.2.1	UFS 情報を表示する	11-35
11.3.2.2	UFS クラスタをモニタリングする	11-36
11.3.2.3	メタデータ・バッファ・キャッシュを表示する	11-37
11.3.3	UFS の性能のチューニング	11-38
11.3.3.1	UFS Smooth Sync と入出力スロットリングを調整す る	11-38
11.3.3.2	UFS クラスタの書き込みを遅らせる	11-41

11.3.3.3	クラスタ内のブロックの数を増やす	11-42
11.3.3.4	ファイル・システムの断片化を解消する	11-43
11.4	NFS のチューニング	11-44
12	メモリ性能の管理	
12.1	仮想メモリの動作	12-1
12.1.1	物理ページの維持管理	12-3
12.1.2	ファイル・システム・バッファ・キャッシュのメモリ割り当て	12-3
12.1.2.1	メタデータ・バッファ・キャッシュのメモリ割り当て	12-4
12.1.2.2	ユニファイド・バッファ・キャッシュのメモリ割り当て	12-4
12.1.3	プロセスのメモリ割り当て	12-6
12.1.3.1	プロセスの仮想アドレス空間の割り当て	12-6
12.1.3.2	仮想アドレスから物理アドレスへの変換	12-7
12.1.3.3	ページ・フォールト	12-8
12.1.4	ページの再生	12-10
12.1.4.1	変更ページの事前書き出し	12-13
12.1.4.2	ページングによるメモリの再生	12-14
12.1.4.3	スワッピングによるメモリの再生	12-16
12.2	性能の高いスワップ領域の構成	12-18
12.3	メモリ統計情報のモニタリング	12-19
12.3.1	vmstat コマンドを使ってメモリを表示する	12-21
12.3.2	ps コマンドを使ってメモリを表示する	12-25
12.3.3	swapon コマンドを使ってスワップ領域の使用状況を表示する	12-28
12.3.4	dbx デバッガを使って UBC を表示する	12-29
12.4	プロセス用のメモリを増やすためのチューニング	12-29
12.4.1	同時に実行するプロセスの数を少なくする	12-30

12.4.2	カーネルの静的サイズを小さくする	12-30
12.4.3	カーネルの malloc 割り当て用に予約するメモリを増やす	12-31
12.5	ページング動作およびスワッピング動作の変更	12-31
12.5.1	ページングしきい値を大きくする	12-32
12.5.2	スワッピング・レート进行管理する	12-33
12.5.3	タスク・スワッピングを積極的に行うようにする	12-35
12.5.4	スワッピングを避けるために常駐セットのサイズを制限する	12-35
12.5.5	変更ページの事前書き出し进行管理する	12-38
12.5.6	ページ・イン・クラスタおよびページ・アウト・クラスタのサイズ进行管理する	12-40
12.5.7	スワップ・パーティションでの入出力要求进行管理する	12-41
12.6	共用メモリ用の物理メモリの予約	12-42
12.6.1	粒度ヒントを使用するようにカーネルをチューニングする	12-42
12.6.2	粒度ヒントを使用するようにアプリケーションを変更する	12-44
12.7	ビッグ・ページによる性能改善	12-46
12.7.1	ビッグ・ページを使用する	12-46
12.7.2	メモリ・オブジェクトにビッグ・ページを使用するための条件を指定する	12-48

13 CPU 性能の管理

13.1	CPU 性能に関する情報のモニタリング	13-1
13.1.1	uptime コマンドを使って平均負荷をモニタリングする ..	13-3
13.1.2	kdbx デバッガ lockstat 拡張を使って CPU 使用状況をチェックする	13-4
13.1.3	kdbx デバッガ lockstat 拡張を使ってロックの使用状況をチェックする	13-5
13.2	CPU 性能の改善	13-6
13.2.1	プロセッサを追加する	13-6
13.2.2	クラス・スケジューラを使用する	13-7

13.2.2.1	クラス・スケジューラの概要	13-9
13.2.2.1.1	関連ユーティリティ	13-10
13.2.2.1.2	クラス・スケジューラの起動	13-10
13.2.2.2	クラス・スケジューリングのプランニング	13-11
13.2.2.3	クラス・スケジューラの構成	13-12
13.2.2.4	クラスの作成と管理	13-14
13.2.2.4.1	クラスを作成する	13-14
13.2.2.4.2	クラス内での識別子タイプを管理する	13-15
13.2.2.4.3	クラス・スケジューラを有効にする	13-16
13.2.2.4.4	クラスへメンバを追加する	13-16
13.2.2.4.5	クラスからメンバを削除する	13-17
13.2.2.4.6	その他のクラス管理オプション	13-17
13.2.2.5	runclass コマンドの使用	13-18
13.2.2.6	クラス・スケジューリング・グラフィカル・インタ フェースの使用	13-18
13.2.2.7	データベースの作成と変更	13-20
13.2.3	ジョブの優先順位を設定する	13-23
13.2.4	ジョブをオフピークの時間帯にスケジューリングする	13-23
13.2.5	advfsd デーモンを停止する	13-23
13.2.6	ハードウェア RAID を使って CPU の入出力オーバーヘッド を軽減する	13-24

用語集

索引

図

1-1	ハードウェア構成図の作成	1-3
1-2	ポイント・ツー・ポイント・トポロジ	1-15
1-3	ファブリック・トポロジ	1-16

1-4	調停ループ・トポロジ	1-17
1-5	単純なゾーン構成	1-22
1-6	回復力のあるメッシュ・ファブリック (カスケード接続された スイッチを 4 台使用)	1-23
1-7	単一インタフェースの構成	1-25
1-8	複数のインタフェース	1-27
1-9	物理メモリの用途	1-32
1-10	メモリ・ハードウェア間での命令とデータの移動	1-35
11-1	データのストライピング	11-18
11-2	AdvFS の入出力キュー	11-28
12-1	UBC のメモリ割り当て	12-5
12-2	ファイル・システム動作の負荷が高く、ページング動作がない 場合のメモリ割り当て	12-5
12-3	ファイル・システム動作の負荷が低く、ページング動作の負荷 が高い場合のメモリ割り当て	12-6
12-4	プロセス仮想アドレス空間の用途	12-7
12-5	仮想アドレスから物理アドレスへの変換	12-8
12-6	ページングおよびスワッピングの属性	12-12
12-7	ページング動作	12-16

表

1-1	RAID レベルごとの性能と可用性の比較	1-7
1-2	SCSI バス速度	1-10
1-3	SCSI バスおよびセグメントの長さ	1-13
1-4	ファイバ・チャネル・ファブリックとファイバ・チャネル調停 ループの比較	1-18
1-5	スイッチがサポートしているゾーニングの種類	1-21
1-6	メモリ管理ハードウェア・リソース	1-34
1-7	リソース・モデルと、それに対する構成ソリューション	1-36
2-1	継続的な性能モニタリングのツール	2-31

2-2	アプリケーションのプロファイリング・ツールとデバッグ・ ツール	2-33
4-1	性能の低い Oracle アプリケーションを検出するツール	4-2
5-1	NFS の性能低下を検出するツール	5-2
5-2	発生しうる NFS の問題と解決策	5-3
5-3	NFS のチューニング・ガイドライン	5-4
5-4	NFS サーバのチューニング・ガイドライン	5-10
5-5	ネットワーク・カードのタイプの確認	5-14
6-1	ネットワーク統計情報モニタリング・ツール	6-4
7-1	アプリケーションの性能改善のガイドライン	7-1
8-1	maxusers 属性の省略時の値	8-3
8-2	IPC の限界値のチューニング・ガイドライン	8-9
9-1	ディスク入出力の分散状況のモニタリング・ツール	9-3
9-2	ハードウェア RAID サブシステムの構成のガイドライン	9-11
10-1	ネットワークのモニタリング・ツール	10-2
10-2	ネットワークのチューニング・ガイドライン	10-5
11-1	キャッシュ情報を表示するツール	11-2
11-2	namei キャッシュ関係の属性の値を変更する時期	11-5
11-3	UBC 関連の属性の値を変更する時期	11-8
11-4	AdvFS の構成のガイドライン	11-12
11-5	AdvFS 情報を表示するツール	11-22
11-6	UFS の構成のガイドライン	11-31
11-7	UFS 情報を表示するツール	11-35
11-8	UFS のチューニング・ガイドライン	11-38
12-1	vm_page_free_target 属性の省略時の値	12-12
12-2	仮想メモリおよび UBC を表示するためのツール	12-20
12-3	メモリ・リソースのチューニング・ガイドライン	12-30
13-1	CPU のモニタリング・ツール	13-1
13-2	主な CPU 性能改善のガイドライン	13-6



まえがき

本書では、HP Tru64 UNIX の性能と可用性を高めるためのチューニング方法を説明しています。また、本書には、Oracle、NFS (Network File System) と Web サーバ・アプリケーション、および Tru64 UNIX オペレーティング・システムのコンポーネントについてのチューニング方法も説明しています。

Tru64 UNIX のシステム管理には、グラフィカル・ユーザ・インタフェース (GUI) を使用することをお勧めします。この GUI は SysMan によって表示されます。SysMan は、共通デスクトップ環境 (CDE) ソフトウェアがシステムにロードされる時点で自動的にロードされるアプリケーションです。SysMan アプリケーションは、CDE のフロントパネルからアクセスできるアプリケーション・マネージャから使用できます。

本書の対象読者

本書は、Tru64 UNIX オペレーティング・システムの管理責任者を対象としています。管理者は、オペレーティング・システムの概要、コマンド、ユーティリティ以外に、アプリケーションとユーザについてもよく理解していなければなりません。これらを理解しておくことは、システムをうまくチューニングして性能を向上するために必要です。

新しい機能および変更された機能

Tru64 UNIX の本バージョンでは、マニュアルに次のような追加と変更があります。

- 本書は、3 つの部分で構成されています。
 - システム・チューニングの概要 — 新しいハードウェア構成の情報と図が含まれます。
 - アプリケーション・タイプ別のチューニング — 3 つのタイプの異なるアプリケーション、Oracle、Network File Systems およびインターネット・サーバについて 3 とおりのチューニング方法を示しています。ここでは、アプリケーション・タイプ別に、チューニング属性とシステム・モニタリング・ツールを詳細に説明しています。

- コンポーネント別のチューニング – システム・リソース、ディスク・ストレージ、ネットワーク、ファイル・システム、メモリおよび CPU についてのチューニング方法が含まれています。ここには、各コンポーネントのチューニング属性の詳しい説明も含まれます。
 - 本書では、システムの性能を改善するために使用できるカーネル・サブシステム属性についてのみ説明しています。リファレンス・ページには、すべてのカーネル・サブシステム属性が記載されています。詳細は、`sys_attrs(5)` を参照してください。
 - カーネル・サブシステム属性では、ダッシュと下線の組み合わせの代わりに、下線のみを使用します。
 - AdvFS (Advanced File System) バッファ・キャッシュは使用されなくなりました。AdvFS のユーザおよびアプリケーション・データとメタデータは、現在ではユニファイド・バッファ・キャッシュ (UBC) にキャッシュされます。詳細は 11.1.4 項を参照してください。
- AdvFS バッファ・キャッシュに関連するカーネル・サブシステム属性を説明していた箇所は、削除されました。
- UNIX ファイル・システム (UFS) では、`smooth sync` 機能と入出力スロットリングを使用して UFS の性能を改善しています。詳細は 11.3.3.1 項 を参照してください。

本書の構成

本書は、以下の 13 章と用語集から構成されています。

- | | |
|-------|---|
| 第 1 部 | システム・チューニングの概要 |
| 第 1 章 | Tru64 UNIX のハードウェア構成と重要な用語と概念について説明しています。 |
| 第 2 章 | システム情報の収集方法と性能問題の診断方法について説明しています。 |
| 第 3 章 | カーネル・サブシステムへアクセスし、変更する方法について説明しています。 |
| 第 2 部 | アプリケーション・タイプ別のチューニング |
| 第 4 章 | Oracle データベース・アプリケーションを構成し、チューニングする方法について説明しています。 |
| 第 5 章 | Network File System アプリケーションを構成し、チューニングする方法について説明しています。 |

第 6 章	Web サーバ・アプリケーションを構成し、チューニングする方法について説明しています。
第 7 章	アプリケーション性能を管理する方法について説明しています。
第 3 部	コンポーネント別のチューニング
第 8 章	システム・リソースの割り当てを管理する方法について説明しています。
第 9 章	ディスク・ストレージの性能をモニタリングし、チューニングする方法について説明しています。
第 10 章	ネットワークの性能をモニタリングし、チューニングする方法について説明しています。
第 11 章	ファイル・システムの性能を管理する方法について説明しています。
第 12 章	メモリの性能をモニタリングし、チューニングする方法について説明しています。
第 13 章	CPU の性能をモニタリングし、チューニングする方法について説明しています。

関連資料

システムの管理および監視については『システム管理ガイド』で説明しています。Tru64 UNIX オペレーティング・システムにおけるプログラミング・ツールについては『プログラミング・ガイド』で説明しています。またこのマニュアルでは、アプリケーション・プログラムのコードを最適化する方法、構築プロセスの結果を最適化する方法についても説明しています。『*Asynchronous Transfer Mode*』では、ATM (Asynchronous Transfer Mode) のチューニングについて説明しています。

以下の Tru64 UNIX のマニュアルも参考になります。

- 『*Tru64 UNIX 概要*』
- 『ネットワーク管理ガイド：接続編』
- 『*Logical Storage Manager*』
- 『*AdvFS 管理ガイド*』
- 『*Tru64 UNIX Version 5.1B QuickSpecs*』
- 『*TruCluster Server Version 5.1B QuickSpecs*』
- 『ハードウェア管理ガイド』

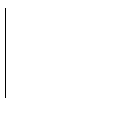
本書の表記法

本書では、以下の表記法を使用しています。

#	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。
% cat	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。
[] { }	構文定義では、大カッコはオプションの項目を示し、中カッコは必須項目を示します。大カッコまたは中カッコの中の項目を縦線で区切っている場合は、そこに併記されている項目の中から1つの項目を選択することを示します。
⋮	垂直の反復記号は、実際には存在する例の一部が省略されていることを示します。
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。

Part 1

システム・チューニングの概要



システム・チューニングの概要

Tru64 UNIX は、システムの性能をモニタリングするためのさまざまなツールを備えています。本書では、システムの性能を改善するための推奨チューニング方法について説明します。どのようなシステム属性でも、変更を行う場合は、事前に以下の内容を理解しておく必要があります。

この章ではシステム・チューニングの概要を説明します。以下の項目があります。

- ハードウェア構成 (1.1 節)
- 性能に関する用語と概念 (1.2 節)
- ディスク・ストレージ・リソース (1.3 節)
- ネットワーク・リソース (1.4 節)
- ファイル・システム・リソース (1.5 節)
- メモリ・リソース (1.6 節)
- CPU リソース (1.7 節)
- 作業負荷に適したリソース・モデルの明確化 (1.8 節)
- 一般的なチューニング対象カーネル・サブシステム (1.9 節)

1.1 ハードウェア構成

構成には、オペレーティング・システムとアプリケーション・ソフトウェアだけでなく、システム、ディスク・ストレージ、ネットワーク・ハードウェアも含まれます。どのように構成するかによって、CPU パワー、メモリ・リソース、入出力性能、および記憶容量が変化します。本書で説明する構成ガイドラインに従って、作業負荷、性能、および可用性の要件に合った構成を選択してください。

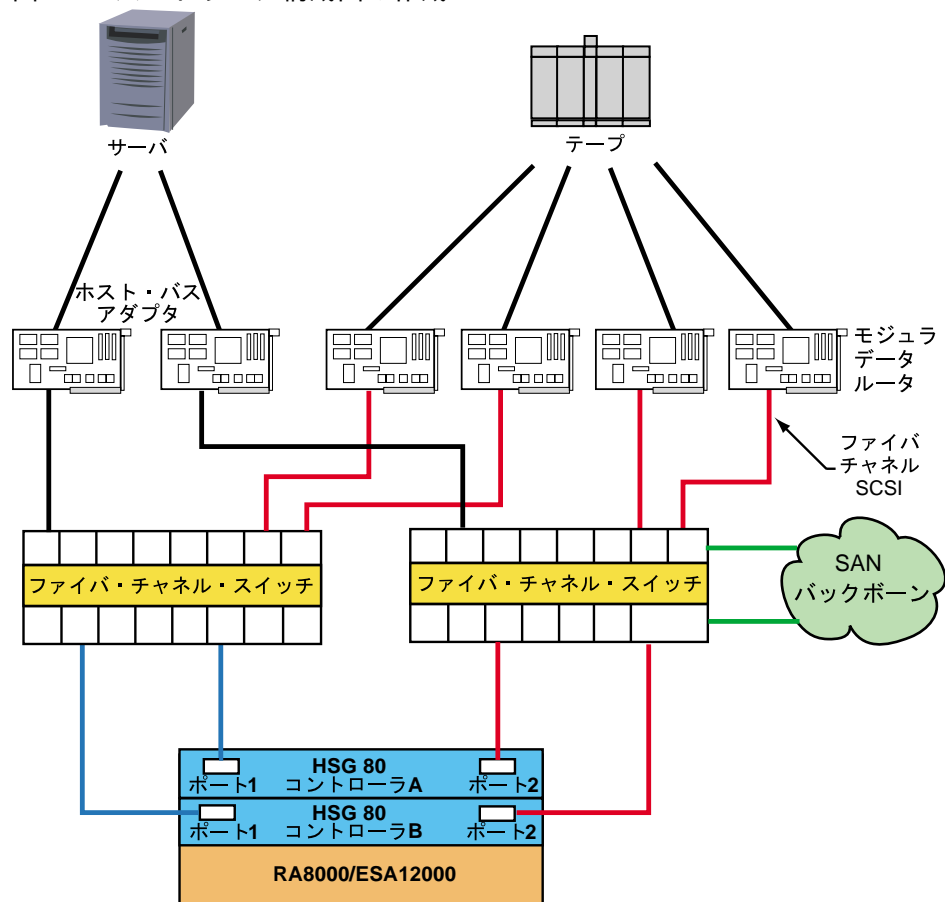
システムの構成後は、オペレーティング・システムをチューニングして性能を改善できます。通常のチューニングでは、カーネル・サブシステムの動作と性能に影響する属性の省略時の値を変更して、カーネルを変更します。

以降の節では、CPU、メモリ、および入出力構成がどのように性能に影響するかを理解する上で役立つ情報を説明します。ハードウェアやオペレーティング・システムの性能の特性については、『*Tru64 UNIX Version 5.1B QuickSpecs*』および『*Tru64 UNIX 概要*』を参照してください。

1.1.1 ハードウェア構成の概要

ご使用のシステムのハードウェア環境を理解するために、ハードウェア構成図を作成することをお勧めします。図 1-1 に、ハードウェア構成の例を示します。この例では、CPU、ホスト・バス・アダプタ、ネットワーク・インタフェース・カード、ファイバ・チャネル・スイッチとファイバ・チャネル接続、およびストレージ・アレイの数など、システムのハードウェア構成要素のうち性能に影響を与える主要な構成要素を図で示しています。

図 1-1: ハードウェア構成図の作成



ZK-1962U-AI

1.2 性能に関する用語と概念

システムの性能は、システムのリソースの利用効率に従って変化します。リソースとは、ユーザやアプリケーションが利用できるハードウェアおよびソフトウェアの構成要素です。システムは、アプリケーションやユーザによる通常の作業負荷で、十分に動作しなければなりません。

作業負荷は時が経つに従って変化するため（たとえば、アプリケーションの実行を追加するなど）、システムはスケーラブルでなければなりません。スケーラブルとは、ハードウェア・リソースを追加することで、期待どおりの効果を得ることができる、システムの能力のことです。スケーラビリティ

ティは、性能を大幅に悪化させることなく、作業負荷の増加を吸収できる、システムの能力も指します。

構成分野によっては、性能上の問題はボトルネックと呼ばれます。ボトルネックは、作業負荷の増大によって能力 (システム・リソースの論理上の最大スループット) よりも多くのリソースが必要となった場合に発生します。

多くの場合、性能は2つの速度で表されます。帯域幅は、入出力サブシステムや構成要素がデータ・バイトを転送できる速度です。帯域幅は、転送速度とも呼ばれます。帯域幅は、大規模なシーケンシャル・データ転送を行うアプリケーションで特に重要です。

スループットは、入出力サブシステムや構成要素が入出力操作を行える速度です。スループットは、小規模な入出力操作をたくさん行うアプリケーションで特に重要です。

性能は、待ち時間でも計測されます。待ち時間は、特定の動作が完了するまでの時間です。待ち時間は、遅延とも呼ばれます。高性能のシステムは、待ち時間が短い必要があります。入出力待ち時間はミリ秒単位、メモリ待ち時間はナノ秒単位で計測されます。メモリ待ち時間は、メモリのバンク構成と、メモリ量により異なります。

ディスクの性能は通常、ディスク・アクセス時間で表されます。ディスク・アクセス時間は、シーク時間 (ディスクのヘッドが、特定のディスク・トラックへ移動するための時間) と回転待ち時間 (ディスクが特定のディスク・セクタまで回転するための時間) を合わせたものです。

データ転送には、ファイル・システムのデータの転送と、**raw** 入出力 (ファイル・システムを含まないディスクまたはディスク・パーティションへの入出力) があります。raw 入出力では、バッファとキャッシュがバイパスされるため、場合によってはファイル・システム入出力よりも性能が良いことがあります。raw 入出力は、オペレーティング・システムやデータベース・アプリケーション・ソフトウェアでよく使用されます。

データ転送には、アクセス・パターンが複数あります。シーケンシャル・アクセス・パターンは、ディスク上の連続する (隣接する) ブロックからデータの読み取り/書き込みを行うアクセス・パターンです。ランダム・アクセス・パターンは、ディスク上の異なる位置 (通常は、隣接していない位置) のブロックからデータの読み取り/書き込みを行うアクセス・パターンです。

1.3 ディスク・ストレージ・リソース

ディスク・ストレージの構成は幅が広いので、どの構成がアプリケーションやユーザの性能や可用性の要件に合っているかを判断しなければなりません。

ディスク・ストレージの構成では、個々のディスクを従来の個別ディスク・パーティションに分割した構成とすることもできます。ただし、LSM (Logical Storage Manager) を使用して、大規模なディスク・ストレージを管理することもできます。LSM を使用すると、ストレージの共用プールの設定が可能で、RAID サポートなどの高性能、高可用性の機能も利用できます。

ストレージの構成には、ハードウェア **RAID** サブシステムを含めることもできます。このサブシステムを使用すると、1 つの入出力バスに接続できるディスク数が大幅に増え、RAID サポートやライトバック・キャッシュなど、多くの高性能機能や高可用性機能が利用できます。ハードウェア RAID サブシステムにはさまざまなタイプがあり、各種の環境に適合できます。

ホスト・バス・アダプタ、RAID コントローラ、およびディスクにはさまざまな性能のものがあり、ファイバ・チャネルと、さまざまなバージョンのパラレル **SCSI** (Small Computer SYstem Interface) をサポートします。ファイバ・チャネルと SCSI はデバイスおよび相互接続の技術であり、高性能、高可用性、および構成の柔軟さの点で進化し続けています。ディスク・ストレージ・リソースについての詳細は、以下の項を参照してください。

- RAID 機能についての詳細は、1.3.1 項を参照してください。
- SCSI についての詳細は、1.3.2 項を参照してください。
- ファイバ・チャネルについての詳細は、1.3.3 項を参照してください。
- ストレージの構成についての詳細は、第 9 章を参照してください。

1.3.1 RAID 技術

ストレージの構成で **RAID** (redundant array of independent disks) 技術を使用すると、性能とデータの可用性を高くできます。LSM (Logical Storage Manager) を使用するか、ハードウェア・ベースの RAID サブシステムを使用すると、RAID 機能を利用できます。

主な RAID レベルには、次の 4 つがあります。

- **RAID0**— データまたはディスクのストライピングとも呼ばれます。RAID0 では、データがブロックに分割され、ブロックがアレイ内の複数

のディスクに分散されます。これにより、スループットが向上します。ストライピングでは、ディスク・データの可用性は改善されません。

- **RAID1**— データまたはディスクのミラーリングとも呼ばれます。RAID1 では、データのコピーがアレイ内の別のディスク上に保持されます。別のディスク上にデータを複製することにより、データの可用性が高くなり、ディスク読み込みの性能が向上します。RAID0 に RAID1 を組み合わせて、ストライピングしたデータやディスクをミラーリングすることができます。
- **RAID3**— パリティ RAID の一種です。RAID3 では、データ・ブロックが分割され、そのデータがディスク・アレイ上に分散されます。これによりデータの並列アクセスが可能となり、帯域幅が向上します。RAID3 では、独立したディスク上に冗長パリティ情報を置くことによって、データの可用性も改善されます。このパリティ情報は、アレイ内のディスクの障害時に、データを再生するために使用されます。
- **RAID5**— パリティ RAID の一種です。RAID5 では、データ・ブロックがアレイ内のディスクに分散されます。RAID5 では独立したデータ・アクセスが可能で、入出力操作を同時に処理できます。これにより、スループットが向上します。RAID5 では、ディスクのアレイに冗長パリティ情報を分散することによって、データの可用性が改善されます。このパリティ情報は、アレイ内のディスクの障害時に、データを再生するために使用されます。

さらに、高性能 RAID コントローラでは、動的パリティ **RAID** (アダプティブ **RAID3/5** と呼ばれる) をサポートしています。この RAID は RAID3 と RAID5 の長所を組み合わせたもので、各種のアプリケーションでのディスク入出力の性能を改善します。動的パリティ RAID では、必要な作業負荷に応じて、データ転送を多用するアルゴリズムと入出力操作を多用するアルゴリズムの間で動的な調整を行います。

RAID の性能は、RAID サブシステム内のデバイスの状態に依存します。この状態には、次の3 種類があります。

- 安定状態 (障害なし)
- 障害状態 (1 つ以上のディスクに障害あり)
- 回復状態 (サブシステムが障害から回復中)

表 1-1 では、各 RAID レベルでの性能と、可用性のレベルを比較しています。

表 1-1: RAID レベルごとの性能と可用性の比較

RAID レベル	性能	可用性のレベル
RAID0 (ストライピング)	入出力負荷が平均化し、スループットが向上する。	単独のディスクより低い。
RAID1 (ミラーリング)	読み込みの性能は向上するが、書き込みの性能は低下する。	最高。
RAID0+1	入出力負荷が平均化し、スループットが向上する。ただし、書き込みの性能は低下する。	最高。
RAID3	帯域幅が改善されるが、複数のディスクで障害が発生した場合は、性能が低下することがある。	単独のディスクより高い。
RAID5	スループットが改善されるが、複数のディスクで障害が発生した場合は、性能が低下することがある。	単独のディスクより高い。
動的パリティ RAID (RAID3/5)	帯域幅とスループットが改善されるが、複数のディスクで障害が発生した場合は、性能が低下することがある。	単独のディスクより高い。

RAID 構成を選択するときには、考慮しなければならない事項が多数あります。

- すべての RAID 製品がすべての RAID レベルをサポートしているわけではありません。
たとえば、動的パリティ RAID をサポートしているのは、高性能 RAID コントローラだけです。
- RAID 製品ごとに、性能が異なります。
たとえば、RAID コントローラだけがライトバック・キャッシュをサポートし、CPU の入出力オーバーヘッドを軽減します。
- RAID 構成の種類によっては、他の構成より費用効率が良くなる場合があります。
一般に、ハードウェア RAID サブシステムよりも LSM の方が、費用効率の良い RAID 機能を利用できます。また、パリティ RAID では、RAID1 (ミラーリング) よりも低価格でデータの可用性が改善されます。これは、 n 個のディスクのミラーリングでは、 $2n$ 個のディスクが必要となるためです。
- データ回復の速度は、RAID 構成に依存します。

たとえば、ディスクの障害時、パリティ情報を使用するよりもミラーリングされたコピーを使用する方が、速くデータを再生できます。また、パリティ RAID を使用している場合は、障害のあるディスクが増えるたびに、入出力性能が低下します。

RAID 構成についての詳細は、第 9 章を参照してください。

1.3.2 SCSI の概念

最も一般的な SCSI タイプは、パラレル **SCSI** です。これは、SCSI の一種であり、幅広い性能オプションや構成オプションをサポートしています。この SCSI には、データ・バス (ナローまたはワイド)、バス速度 (Slow, Fast, Ultra, Ultra2, Ultra3)、および転送方式 (シングルエンドまたはディファレンシャル) によるバリエーションがあります。これらの SCSI 属性によって、バスの帯域幅と、SCSI バスの最大長が決まります。

シリアル **SCSI** は、次世代の SCSI です。シリアル SCSI は、パラレル SCSI の速度、距離、接続性 (バス上のデバイス数) の制限を緩和し、ホット・スワップやフォールト・トレランスなどの高可用性機能も提供します。

ファイバ・チャネルは、シリアル SCSI の一種です。ファイバ・チャネルは、複数のプロトコル (SCSI, IPI, FIPS60, TCP/IP, HIPPI など) をサポートする高性能入出力バスで、インテリジェント・スイッチのネットワークをベースとしています。リンク速度は、全二重モードで最大 100 MB/秒です。

以降の項では、パラレル SCSI の概念を詳細に説明します。

1.3.2.1 データ・バス

ディスク、ホスト・バス・アダプタ、入出力コントローラ、およびストレージ筐体は、固有のデータ・バスをサポートします。データ・バスとバス速度によって、バスの実際の帯域幅が決まります。データ・バスには、次の 2 種類があります。

- ナロー・データ・バス

8 ビット・データ・バスを規定します。このモードの性能には限界があります。SCSI バス仕様では、ナロー SCSI バス上のデバイス数を、8 台までに制限しています。

- ワイド・データ・バス

Slow SCSI , Fast SCSI , UltraSCSI , Ultra2 , Ultra3 用の 16 ビット・データ・パスを規定します。このモードでは、バス上で並列に転送されるデータ量が多くなります。SCSI バス仕様では、ワイド・バス上のデバイス数を、16 台までに制限しています。

ワイド・データ・パスを使用するディスクおよびホスト・バス・アダプタは、ナロー・データ・パスを使用するディスクおよびアダプタのほぼ 2 倍の帯域幅があります。ワイド・デバイスでは、大規模なデータ転送での入出力性能を大幅に改善できます。

最近の大部分のディスクは、ワイド・データ・パスをサポートしています。旧式のディスクには、ワイド・データ・パスをサポートするバージョンと、ナロー・データ・パスをサポートするバージョンがあります。データ・パスの異なる (または転送方式の異なる) デバイスは、同じバス上で直接接続することはできません。データ・パスの異なるデバイスを接続するには、SCSI 信号変換器 (たとえば、DWZZA や DWZZB) か、**UltraSCSI** エクステンダ (たとえば、DWZZC や DWZZH (SCSI ハブ)) を使用しなければなりません。

1.3.2.2 SCSI バス速度

SCSI バス速度は 1 秒あたりの転送数であり、転送速度または帯域幅ともいいます。バス速度が速いほど、性能も良くなります。バス速度とデータ・パス (ナローまたはワイド) によって、実際のバス帯域幅 (1 秒あたりの転送バイト数) が決まります。

すべてのデバイスが、すべてのバス速度をサポートしているわけではありません。ホスト・バス・アダプタのバス速度を設定するには、アダプタのタイプに応じて、コンソール・コマンドか LFU (Loadable Firmware Update) ユーティリティを使用します。SCSI デバイスのサポート状況については、『*TruCluster Server Software Version 5.1B QuickSpecs*』を参照してください。

表 1-2 に、利用できるバス速度を示します。

表 1-2: SCSI バス速度

バス速度	最大転送速度 (100 万転送/秒)	最大バイト転送速度: ナロー (MB/秒)	最大バイト転送速度: ワイド (MB/秒)
Ultra3	80	80	160
Ultra2	40	40	80
UltraSCSI	20	20	40
Fast SCSI	10	10	20
Slow	5	5	10

弊社の Ultra3 は、Ultra3 SCSI 仕様を実装した Ultra 160/m と互換性があり、これに準拠しています。

Fast SCSI は **Fast10** とも呼ばれ、SCSI-2 仕様の拡張です。Fast SCSI には高速同期転送オプションがあり、入出力デバイスは同期モードで最高の転送速度になります。

UltraSCSI は **Fast20** とも呼ばれ、Fast SCSI の性能や構成上の欠点の多くに対処した、SCSI-2 の高性能な拡張バージョンです。Fast SCSI のバス速度に比べて、UltraSCSI では帯域幅と構成距離が 2 倍になりますが、費用は変わりません。また、UltraSCSI では処理時間が短縮され、データ分析が速く正確になります。すべての UltraSCSI の構成要素は、通常の SCSI-2 の構成要素と互換性があります。

1.3.2.3 転送方式

バスの転送方式とは、SCSI 仕様の電氣的な実装方式を指します。サポートされている転送方式は、次のとおりです。

- シングルエンド (SE) SCSI

通常、同一キャビネット内にあるデバイスの接続に使用します。シングルエンド SCSI では、通常、短いケーブルを使用する必要があります。

シングルエンド SCSI バスでは、データ転送にデータ線 1 本とグラウンド線 1 本を使用します。シングルエンドの受信側は、信号線のみを入力としてチェックします。信号線上でバスの受信側に到着した転送信号は、信号の反射により少し歪みます。バスの長さや負荷が、この歪みの程度を左右します。このため、シングルエンド転送方式は安価ですが、ディファレンシャル転送方式よりもノイズに弱いので、ケーブルを短くする必要があります。

- **ディファレンシャル SCSI**

最大 25 メートル離れたデバイスを接続できます。

ディファレンシャル SCSI バスでは、信号の転送に信号線を 2 本使用します。この 2 本の信号線は、ディファレンシャル・ドライバが駆動します。ディファレンシャル・ドライバは 1 つの信号 (+SIGNAL) を 1 本の信号線に置き、180 度位相のずれた信号 (-SIGNAL) をもう 1 本の信号線に置きます。ディファレンシャルのレシーバは、この 2 つの入力信号が異なる場合のみ、信号出力を生成します。信号の反射は両方の信号線で事実上同じであるため、2 本の信号線の相違のみに注目するレシーバには信号の反射が見えなくなります。ディファレンシャル転送方式は、シングルエンド SCSI よりもノイズの影響を受けにくいため、長いケーブルを使用できます。

SE デバイスとディファレンシャル・デバイスを同じバスに直接接続することはできません。SE デバイスとディファレンシャル・デバイスを同じバスに接続するためには、UltraSCSI エクステンダを使用する必要があります。

- **LVD (Low Voltage Differential) SCSI**

ディファレンシャル SCSI と同じですが、使用する電圧が低く、SE ドライブと LVD SCSI ドライバを同じ SCSI バスに接続できる点が異なります。

SE デバイスと LVD SCSI デバイスを同じ SCSI バスに接続すると、その SCSI バス (その SCSI チャンネル) 上のすべてのデバイスの性能が SE SCSI の動作に制限されます (40 MB/秒)。SCSI の規定では、本来の LVD SCSI バスとそれに応じた性能を維持するには、LVD SCSI チャンネルには LVD SCSI ドライブのみを接続する必要があります。ただし、1 つのアレイ・コントローラで、SE 専用チャンネルと LVD 専用チャンネルを使用することはできます。

Ultra2 および Ultra3 デバイスは、LVD の電気条件で動作します。Ultra2 および Ultra3 デバイスが同じ Ultra3 SCSI バスに接続されている場合、Ultra2 デバイスはデータを最大 80 MB/秒で転送し、Ultra3 デバイスはデータを最大 160 MB/秒で転送します。SCSI バスが Ultra2 のみをサポートしている場合、LVD デバイスはすべて、最大転送速度が 80 MB/秒になります。

1.3.2.4 UltraSCSI バス・セグメントの拡張

UltraSCSI デバイスには、シングルエンドとディファレンシャルがあります。UltraSCSI のバス速度は高速なため、シングルエンド UltraSCSI の信号は、シングルエンドの Fast SCSI の信号と同じ距離まで強度と完全性を維持することができません。このため、UltraSCSI ではバス・セグメントとバス・エクステンダを使用して、システムとストレージ間の距離が長くても構成できるようにしています。

UltraSCSI バス・エクステンダは、SCSI プロトコルに影響することなく、2 つのバス・セグメントを連結します。バス・セグメントは、コンダクタ(ケーブルまたはバックプレーン)とコネクタからなる、切り離せない電氣的バスとして定義されています。各 UltraSCSI バス・セグメントには、バス・セグメントの両端に 1 つずつ、2 つのターミネータがなければなりません。このため、UltraSCSI のバス・セグメントは、Fast SCSI のバス全体に相当します。SCSI ドメインは、すべてのバス・セグメント上の SCSI デバイスの集まりです。Fast SCSI バスの場合と同様に、UltraSCSI のバス・セグメントは、同じタイプ(シングルエンドまたはディファレンシャル)のデバイスのみをサポートします。

UltraSCSI の構成要素を使用すると、UltraSCSI ドメインは Fast SCSI バスよりも遠くまで延長できますが、限界があります。また、バス・エクステンダを使用すると UltraSCSI ドメインを直線ではなくツリー状にできるため、バスの長さを考慮する代わりに、UltraSCSI ドメインの直径を考慮しなければなりません。

1.3.2.5 SCSI バスの長さでターミネーション

SCSI バスのケーブルの長さには制限があります。ケーブルの最大長は、バス速度と転送方式(シングルエンドまたはディファレンシャル)に依存します。物理バスまたは UltraSCSI バス・セグメントのケーブルのトータル長は、ターミネートされた端から端までで計算します。

さらに、各 SCSI バスまたはバス・セグメントは、各終端でのみターミネートしなければなりません。不適切なバス・ターミネーションや、不適切なバス長は、バスの誤動作の原因となります。

転送方式とデータ・バスが同じデバイス(たとえば、ワイドでディファレンシャル)を使用している場合、バスは 1 つの物理バス(UltraSCSI の場合は、複数のバス・フラグメント)で構成できます。転送方式の異なるデバイスを

使用している場合は、シングルエンドとディファレンシャルの両方の物理バスまたはバス・セグメントが必要になります。各バスまたはバス・セグメントは両端でのみターミネートし、バス長の規則に従わなければなりません。

表 1-3 に、各種のバス速度と転送方式での最大バス長を示します。

表 1-3: SCSI バスおよびセグメントの長さ

バス速度	転送方式	バスまたはセグメントの最大長
Slow	シングルエンド	6 メートル
Fast	シングルエンド	3 メートル
Fast	ディファレンシャル	25 メートル
Ultra	ディファレンシャル	25 メートル
Ultra	シングルエンド	1.5 メートル (デバイス間の距離が 1 メートル未満のデジiser・チェーン構成)
Ultra	シングルエンド	4 メートル (デバイス間の距離が 1 メートルを超えるデジiser・チェーン構成)
Ultra	シングルエンド	20 メートル (デバイスがバス・セグメントの端だけにある、ポイント・ツー・ポイント構成)

物理バスのトータル長には、各システムやディスク・ストレージのシェルフ内にあるケーブルの長さも含めなければなりません。この長さは、デバイスによって異なります。たとえば、BA350、BA353、または BA356 ストレージのシェルフ内部のケーブルの長さは、約 1.0 メートルです。

1.3.3 ファイバ・チャネル

ファイバ・チャネルは、同一の物理インタフェース上で、複数のプロトコルをサポートします。ファイバ・チャネルは、基本的にはプロトコルから独立したトランスポート・メディアです。したがって、使用目的には依存しません。

Tru64 UNIX では、SCSI 用ファイバ・チャネル・プロトコル (FCP) を使用して、ファイバ・チャネルを物理インタフェースとして使用します。

ファイバ・チャネルではシリアル転送を行うため、以下の機能を備えることで、パラレル SCSI に存在する制限をなくしています。

- 100 MB/秒、200 MB/秒、および400 MB/秒 のデータ転送速度

- 複数のプロトコルのサポート
- 優れたスケーラビリティ
- 改善された信頼性および可能性

ファイバ・チャンネルでは、高速なデータ転送速度を達成するために、極めて高速な転送クロック周波数が使用されます。光ファイバ・ケーブルを使用した場合は、最大 40 km の距離を高速で送信できます。40 km がトランスミッタとレシーバ間の最大の距離です。短い距離では、銅ケーブルも使用できます。

以降の項では、ファイバ・チャンネルについて詳しく説明します。

- ファイバ・チャンネル・トポロジ (1.3.3.1 項)
- ファイバ・チャンネル・トポロジの比較 (1.3.3.2 項)
- ゾーニング (1.3.3.3 項)

1.3.3.1 ファイバ・チャンネル・トポロジ

ファイバ・チャンネルでは、以下の 3 種類の相互接続トポロジがサポートされています。

- ポイント・ツー・ポイント (1.3.3.1.1 項)
- ファブリック (1.3.3.1.2 項)
- 調停ループ (1.3.3.1.3 項)

注意

調停ループとファブリックを相互接続することもできますが、ハイブリッド構成は現在サポートされていません。このため、本書では、このようなハイブリッド構成は説明していません。

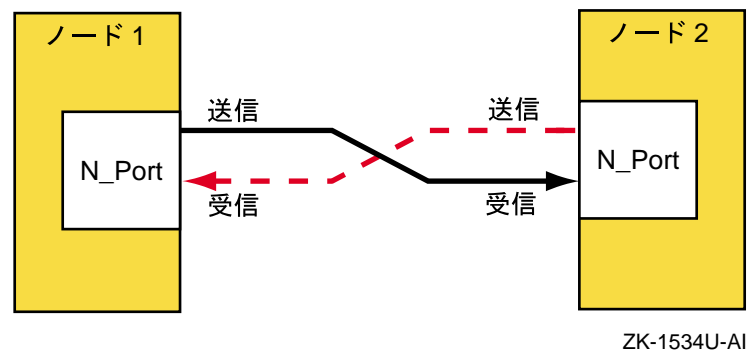
1.3.3.1.1 ポイント・ツー・ポイント・トポロジ

ポイント・ツー・ポイント・トポロジは、最も簡単なファイバ・チャンネル・トポロジです。ポイント・ツー・ポイント・トポロジでは、1 つの **N_Port** と、他の N_Port を単一のリンクで接続します。

片方の N_Port から送信されたフレームがすべて、送信したときと同じ順序で他方の N_Port に受信されるため、フレームをルーティングする必要がありません。

図 1-2 に、ポイント・ツー・ポイント・トポロジの例を示します。

図 1-2: ポイント・ツー・ポイント・トポロジ



1.3.3.1.2 ファブリック・トポロジ

ファブリック・トポロジは、ポイント・ツー・ポイント・トポロジよりも高い接続性を備えています。ファブリック・トポロジでは、最大 2^{24} ポートまで接続できます。

ファブリックは、フレーム・ヘッダ内のデスティネーション・アドレスを検査して、フレームをデスティネーション・ノードにルーティングします。

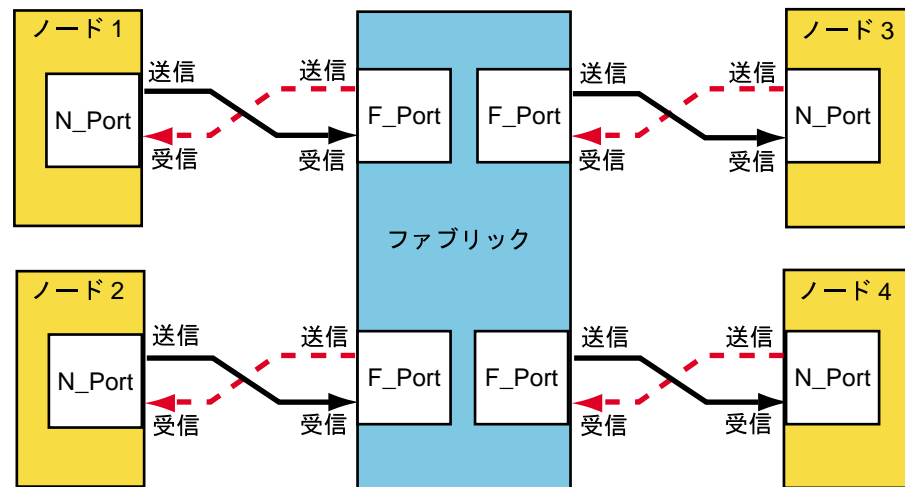
ファブリックは、1 台のスイッチで構成すること、または相互接続された数台のスイッチ (最大 3 台のスイッチの相互接続をサポートします) で構成することもできます。各スイッチには 2 つ以上のファブリック・ポート (**F_Port**) があり、それらのポートはファブリックのスイッチ機能によって内部で接続されています。このファブリックのスイッチ機能により、スイッチ内の F_Port 間でフレームがルーティングされます。2 台のスイッチ間の通信は、2 つの拡張ポート (**E_Port**) 間でルーティングされます。

N_Port を F_Port に接続すると、ファブリックは、ファブリックに接続された N_Port にファイバ・チャンネル・アドレスを割り当てます。また、ファブリックは、ファブリック内で、フレームがデスティネーションに到達するための経路を選択する必要もあります。

ファブリックが複数のスイッチで構成されている場合、ファブリックは、代替経路を決定して、フレームがデスティネーションに確実に到達するようにできます。

図 1-3 に、ファブリック・トポロジの例を示します。

図 1-3: ファブリック・トポロジ



ZK-1536U-AI

1.3.3.1.3 調停ループ・トポロジ

調停ループ・トポロジでは、フレームはノード間のリンクによって形成されたループに沿って伝送されます。ノードやノードのケーブルに障害が発生した場合、ノードの電源が切断された場合、または保守のためにノードを切り離れた場合には、ハブによりそれらのノードがバイパスされ、ループの接続が維持されます。ハブは、プロトコルからは見えません。ハブにはファイバ・チャネルの調停ループ・アドレスが割り当てられないため、ハブはファイバ・チャネルの調停ループ・ポートによってアドレス指定することはできません。

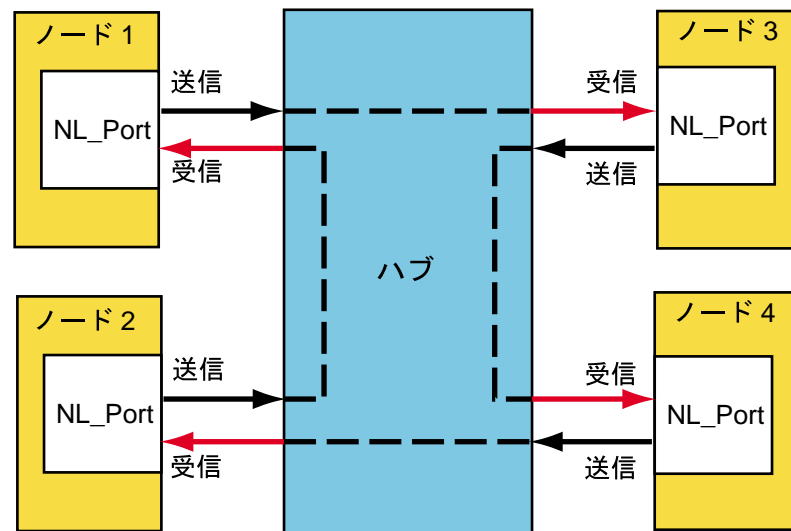
ノードは、ループの制御を得る (マスタになる) ために調停を行います。1 台のノードがマスタになると、各ノードは、ビットマスクにビットを設定することにより、独自の AL_PA (Arbitrated Loop Physical Address) を選択します。AL_PA は、ループ上にあるノードをアドレス指定するために使用されます。AL_PA は動的なので、ループが初期化されたり、ノードが追加または削除されたり、またはループの構成が変化するような状況が発生すると、

AL_PA が変わります。 ノードは、データを送信する準備が整うと、自身を識別する AL_PA を含むファイバ・チャネル基本信号を送信します。

調停ループ・トポロジでは、ノード・ポートを **NL_Port** (ノード・ループ・ポート) と呼び、ファブリック・ポートを **FL_Port** (ファブリック・ループ・ポート) と呼びます。

図 1-4 に、調停ループ・トポロジの例を示します。

図 1-4: 調停ループ・トポロジ



ZK-1535U-AI

1.3.3.2 ファイバ・チャネル・トポロジの比較

ここでは、ファブリック・トポロジと調停ループ・トポロジを比較対比して、これらのトポロジを選択する理由について説明します。

ファブリック (スイッチ) ・トポロジと比べると、調停ループ・トポロジは低コスト、低性能であり、ファブリックの代わりとして選択できる方式です。調停ループ・トポロジは、比較的高価なスイッチに代わり、インテリジェントではなく、管理不要なことが多い安価なハブを使用することにより、ファイバ・チャネルのコストを削減します。ハブは、物理ループを論理スター型に変形させた状態で動作します。ケーブル、関連するコネクタ、および可能なケーブル長は、ファブリック・トポロジの場合と似ています。調停ループでは、論理上、ループ内に最大 127 のノードがサポートされま

す。調停ループのノードは、自己構成ノードであり、ファイバ・チャネル・アドレス・スイッチは不要です。

調停ループは、帯域幅を犠牲にしてコストを削減します。ループ内のすべてのノードが帯域幅 (ループごとに 100 MB/秒) を共有するため、ノードとケーブルを追加するに従って帯域幅がわずかに減少します。ループ上のノードは、他のノード間のトラフィックも含め、ループ上のすべてのトラフィックを扱います。ハブにポート・バイパス機能を追加すると、ループ上のノードの追加、削除を管理することができます。たとえば、ポート・バイパス・ロジックが問題を検出すると、ハブは、人手による介入なしでそのノードをループから除くことができます。このように、ノード障害、ケーブル切断、およびネットワーク再構成などによるダウン時間を回避することで、データの可用性が維持されます。ただし、ノードの追加や削除、エラーなどにより生じたトラフィックにより、ループが一時的に途絶えることがあります。

ファブリック・トポロジは比較的高価ですが、接続性が高まり、性能も向上します。スイッチにより、ファブリックへ全二重 100 (200) MB/秒のポイント・ツー・ポイント接続を行えます。また、ファブリック内のノードが自ノード宛のデータのみを処理することに加え、それぞれのノードがファブリック内の他ノードの再構成や障害回復から切り離されるため、性能とスケラビリティが向上します。スイッチでは、ファイバ・チャネル・ファブリックの全体的な構造に関する管理情報を得ることもできます。これは、通常、調停ループのハブではできません。

表 1-4 は、ファブリック・トポロジと調停ループ・トポロジの比較表です。

表 1-4: ファイバ・チャネル・ファブリックとファイバ・チャネル調停ループの比較

調停ループを使用する場合	ファブリックを使用する場合
メンバ数が 2 までのクラスタ	メンバ数が 3 以上のクラスタ
ソリューションの総コストが低いことと、簡便性が主な要件であるアプリケーション	再構成や修復のための一時的なトラフィックの中断が問題となる、マルチノードのクラスタ構成
調停ループ構成における帯域幅の共有が制限要因とならないようなアプリケーション	調停ループ・トポロジにおける帯域の共有が不適切な広帯域アプリケーション
拡張や規模拡大の予定がない構成	拡張の予定があり、性能のスケラビリティが必要なクラスタ構成

1.3.3.3 ゾーニング

ここでは、ゾーニングの概要について簡単に説明します。

ゾーンとは、ファブリックに接続されたファイバ・チャネル・デバイスの論理的なサブセットです。ゾーニングにより、リソースを分割して、管理やアクセス制御を行うことができます。構成によっては、1 台のスイッチで、複数のクラスタ、さらには複数のオペレーティング・システムに対してサービスを行うようにすることで、ハードウェア・リソースを有効に活用することができます。ゾーニングでは、ファブリックが複数のゾーンに分割されるので、各ゾーンは実質的に仮想ファブリックになります。

ゾーニングは、以下のような場合に使用します。

- 動作環境や用途が異なるシステムの間を仕切りたい場合。たとえば、2 つのクラスタで 1 台のスイッチを利用する場合。
- ファブリックの使用済みの部分から独立したテスト領域を作成したい場合。
- 未使用ポートの数を減らして、スイッチの利用効率を改善したい場合。

注意

最初のゾーニングは、ホスト・バス・アダプタとストレージをスイッチに接続する前に行う必要があります。ただし、ゾーニングを構成した後は、動的に変更することができます。

1.3.3.3.1 スイッチ・ゾーニングとセレクトティブ・ストレージ・プレゼンテーション

HSG80 コントローラのスイッチ・ゾーニング機能とセレクトティブ・ストレージ・プレゼンテーション (**SSP**) 機能は、類似の機能を持っています。

スイッチ・ゾーニングは、どのサーバが相互に通信可能であるか、および各ストレージ・コントローラのホスト・ポートと通信可能であることを制御します。SSP は、どのサーバが各ストレージ・ユニットにアクセスできるかを制御します。

スイッチ・ゾーニングでは、ストレージ・システムのレベルでアクセスを制御しますが、SSP ではストレージ・ユニットのレベルでアクセスを制御します。

以下の構成では、ゾーニング、またはセレクトティブ・ストレージ・プレゼンテーションが必要です。

- ストレージ・エリア・ネットワーク (SAN) に TruCluster Server クラスタと、その他のスタンドアロン・システム (UNIX または UNIX 以外)、またはその他のクラスタが存在する場合。
- 同じ SAN に Tru64 UNIX と、Windows NT または Windows 2000 が存在する場合は必須 (Windows NT や Windows 2000 は、別のスイッチ・ゾーンに存在しなければなりません)。
- RA8000, ESA12000, MA6000, MA8000, または EMA12000 への接続が 65 以上存在する SAN 構成の場合。

ストレージへのアクセスを制御するためには、セレクトティブ・ストレージ・プレゼンテーションが好ましい方式です (ゾーニングが不要になります)。

1.3.3.3.2 ゾーニングの種類

ゾーニングには、ソフト・ゾーニングとハード・ゾーニングの 2 種類があります。

- ソフト・ゾーニングは、シンプル・ネーム・サーバ (**SNS**) に基づいてゾーンが強制される、ソフトウェアによる方式です。ゾーンは、ノードまたはポートのワールド・ワイド名 (**WWN**)、またはドメインとポート番号のいずれかで定義されます。ドメインとポート番号は、D、P の形式で表されます。D はドメインを表し、P はスイッチの物理ポート番号を表します。

ホスト・システムは、ファブリックに接続されているすべてのアダプタおよびストレージ・コントローラのリストを要求します。ネーム・サービスにより、要求元のホスト・バス・アダプタと同じゾーンにあるすべてのポートのリストが与えられます。

ソフト・ゾーニングは、すべてのホストがソフト・ゾーニングに対応している場合にのみ使用できます。ソフト・ゾーニングを許可するように設定されていないホストがあると、使用できません。たとえば、ホスト

がゾーン外のコントローラにアクセスしようとする、スイッチはそのアクセスを阻止できません。

Tru64 UNIX は、ソフト・ゾーニングに対応しており、ゾーン外のデバイスにアクセスしようとしません。

WWN を使用してゾーンを定義している場合に、KGPSA ホスト・バス・アダプタを交換すると、ノード WWN が変わるので、ゾーン構成と SSP を変更しなければなりません。

- ハード・ゾーニングでは、ファイバ・チャネルのフレームをハード的にブロックすることにより、すべてのファブリック・スイッチにおいて物理レベルでゾーンが強制されます。ハードウェア・ゾーンの定義は、D、P の形式で表されます。D はドメインを表し、P はスイッチの物理ポート番号を表します。たとえば、スイッチ 1、ポート 2 の場合は、1, 2 のように表されます。

ホストがゾーン外のポートにアクセスしようとする、スイッチのハードウェアがそのアクセスをブロックします。

ゾーン内でケーブルを別のポートに移した場合は必ず、ゾーン構成を修正しなければなりません。

どのようなゾーンでも確実にゾーン外へアクセスしないようにするには、ハードウェア・ゾーニングを使用するか、またはソフトウェア・ゾーニングをサポートしていることが明記されているオペレーティング・システムを使用してください。

表 1-5 に、サポートされている各ファイバ・チャネル・スイッチが対応しているゾーニングの種類を示します。

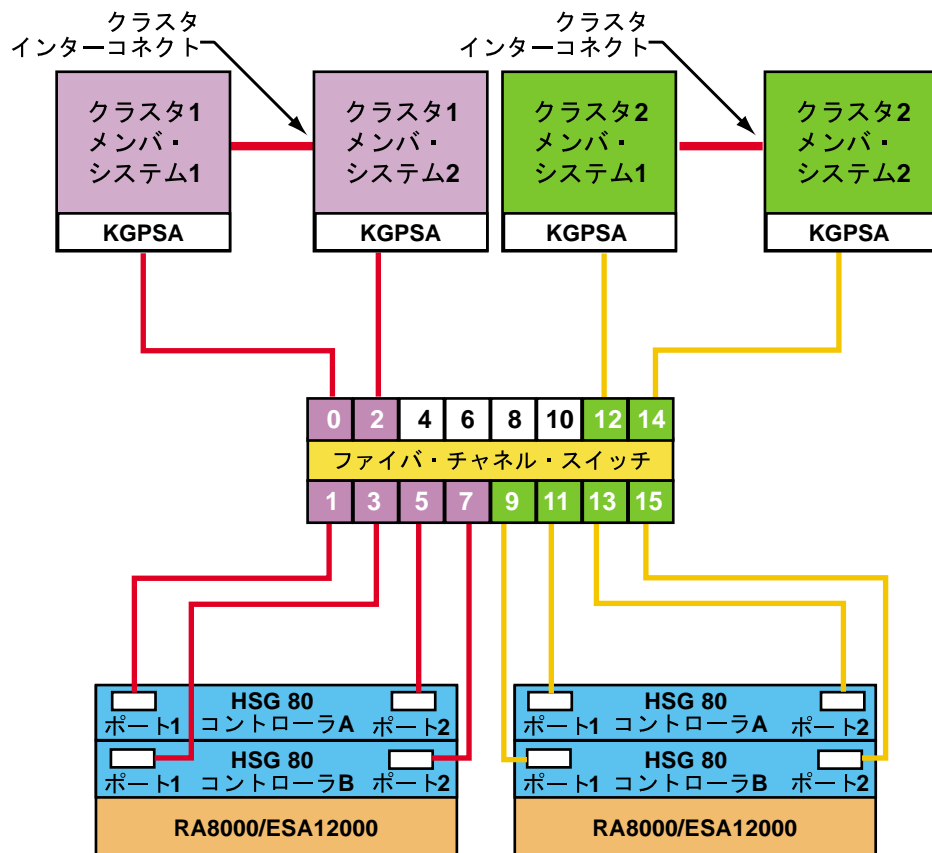
表 1-5: スイッチがサポートしているゾーニングの種類

スイッチの種類	サポートされるゾーニングの種類
DS-DSGGA	ソフト
DS-DSGGB	ソフトおよびハード
DS-DSGGC	ソフトおよびハード

1.3.3.3.3 ゾーニングの例

図 1-5 に、ゾーニングを使用した構成例を示します。この構成は、2 つの独立したゾーンからなり、各ゾーンには独立したクラスタがあります。

図 1-5: 単純なゾーン構成



ZK-1709U-AI

ゾーニングの設定方法については、スイッチに付属の SAN スイッチ・ゾーニングに関するドキュメントを参照してください。

詳細については、『クラスタ・ハードウェア構成ガイド』を参照してください。

1.3.3.4 スイッチのカスケード接続

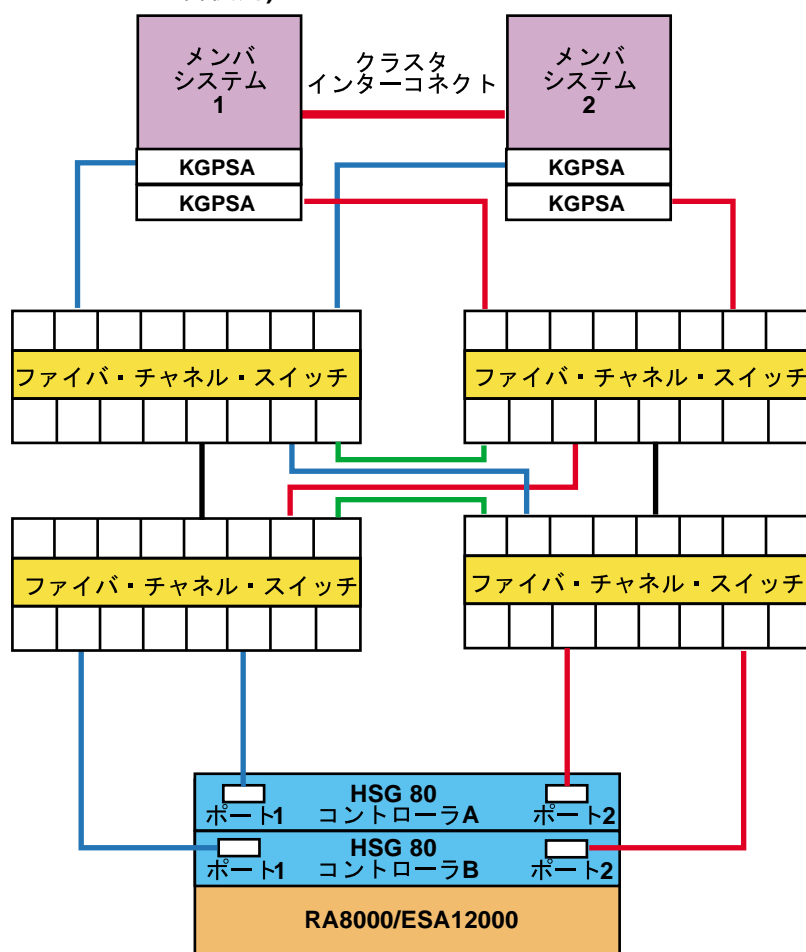
複数のスイッチを相互に接続して、スイッチのネットワーク、つまりスイッチのカスケード接続を形成することができます。

スイッチがカスケード接続された構成では、そのスイッチまでのネットワーク (スイッチも含む) に障害が発生しても、SAN 接続ノードへのデー

タ・パスが失われません。この構成をメッシュ、またはメッシュ・ファブリックと呼びます。

図 1-6 に、4 台のスイッチを相互にカスケード接続した、回復力の強いメッシュ・ファブリックの例を示します。この構成は、複数のデータ・バスに障害が発生しても耐久性があり、シングル・ポイント障害を発生させない (NSPOF) 構成です。

図 1-6: 回復力のあるメッシュ・ファブリック (カスケード接続されたスイッチを 4 台使用)



ZK-1794U-AI

注意

ISL (スイッチ間リンク) が失われると、通信は別のスイッチを経由して別のコントローラと同じポートヘルレーティングされます。これは、最大許容ホップ数 2 を満たしています。

ファイバ・チャネルについての詳細は、『クラスタ・ハードウェア構成ガイド』を参照してください。

1.4 ネットワーク・リソース

システムでは、性能の異なるさまざまな種類のネットワークとネットワーク・アダプタがサポートされます。たとえば、ATM (Asynchronous Transfer Mode) の高性能ネットワークは、ATM ネットワークが提供する高速かつ低遅延という特徴 (交換式、全二重のネットワーク・インフラストラクチャ) を必要とするアプリケーションに適しています。

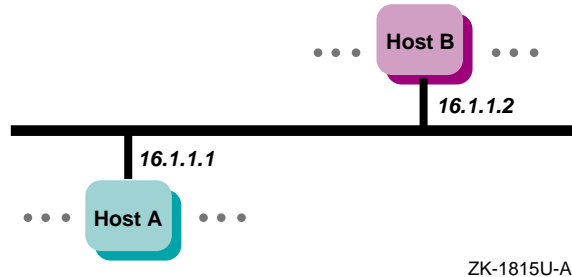
さらに、複数のネットワーク・アダプタを構成するか、または NetRAIN を使用して、ネットワーク・アクセスを増強し、ネットワークの可用性を高めることができます。

システムは、ネットワーク・インタフェース・カード (**NIC**) (ネットワーク・インタフェースまたはネットワーク・アダプタと呼ばれることもあります) を経由してネットワークに接続されます。エンド・システムまたはホストでは、インタフェースについて以下のような選択肢があります。

- サブネット内に単一のインタフェース
- サブネット内に複数のインタフェース
- 自動フェイルオーバー機能を持つ複数のインタフェース (NetRAIN)
- 複数の集約インタフェース (リンク・アグリゲーション)

ルータは、通常、複数のインタフェースを備え、それぞれが異なるサブネットに接続されます。図 1-7 に、ホスト A とホスト B という 2 つのホストがあり、それぞれが単一のネットワーク・インタフェースでサブネットに接続されているネットワークを示します。

図 1-7: 単一インターフェースの構成



以降の項で、システムの性能改善のために重要なネットワーク・リソースについて説明します。

1.4.1 ネットワーク・サブシステム

ネットワーク・サブシステムによって使用されるリソースの大部分は、動的に割り当てられ、調整されます。しかし、性能を改善するためのチューニング・ガイドラインがいくつかあり、特に、Webサーバ、プロキシ・サーバ、ファイアウォール・サーバ、ゲートウェイ・サーバなどのインターネット・サーバのようなシステムで有効です。

ネットワークの性能が影響されるのは、リソースの供給がリソースの需要に追いつかない場合です。以下のような2つの条件で、このような状況が発生することがあります。

- 1つ以上のハードウェアまたはソフトウェア・ネットワーク構成要素に問題がある
- すべてが正常に動作しているように見えるが、作業負荷(ネットワーク・トラフィック)が利用可能なリソースの容量を常に超過している

上記の問題はいずれも、ネットワークのチューニングとは関係ありません。ネットワーク上の問題の場合は、その問題を切り分けて解消しなければなりません。ネットワーク・トラフィックが多い場合(たとえば、システムが100%の能力で動作しているときに、Webサーバのヒット率が最大値に達した場合は、ネットワークを再設計して負荷を再分散するか、ネットワーク・クライアントの数を減らすか、またはネットワーク負荷を処理するシステムの数を増やさなければなりません。

1.4.2 冗長ネットワークの使用

ネットワークの接続は、ネットワーク・インタフェースの障害やネットワーク自体の障害のために失われることがあります。ネットワーク接続に冗長性を持たせることで、ネットワーク接続の可用性を高めることができます。1つの接続が利用不能になった場合でも、別の接続を使用してネットワーク・アクセスを継続することができます。複数のネットワークを使用できるかどうかは、アプリケーション、ネットワーク構成、およびネットワーク・プロトコルに依存します。

また、**NetRAIN (redundant array of independent network adapters)**を使用すると、同じ LAN セグメント上の複数のインタフェースを単一のインタフェースとして構成し、ネットワーク・アダプタおよびネットワーク接続に対してフェイルオーバーをサポートするようにできます。1つのインタフェースが常にアクティブになり、その他のインタフェースはアイドルの状態になります。アクティブなインタフェースが故障すると、アイドル状態のインタフェースが 10 秒以内にオンラインに切り替わります。

NetRAIN は、イーサネットおよび FDDI のみをサポートします。NetRAIN についての詳細は、1.4.3 項を参照してください。

NetRAIN についての詳細は、nr(7) を参照してください。ネットワーク構成については、『ネットワーク管理ガイド：接続編』を参照してください。ネットワーク性能の改善方法については、第 10 章を参照してください。

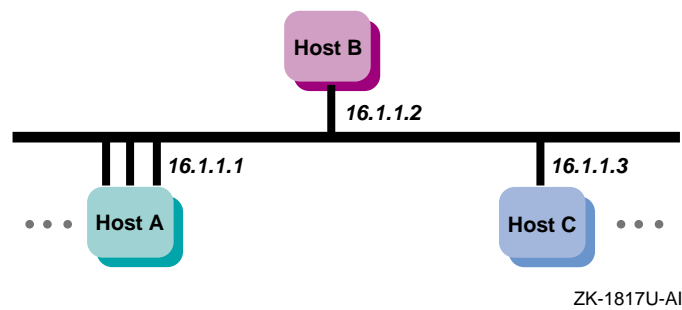
1.4.3 NetRAIN

NetRAIN (Redundant Array of Independent Network Adaptors) インタフェースは、ある種のネットワーク接続障害から保護するメカニズムを提供します。

NetRAIN では、同じローカル・エリア・ネットワーク (LAN) ・セグメント上の複数のネットワーク・インタフェースを、NetRAIN セットと呼ばれる、単一の仮想インタフェースに統合します。セット内の 1 つのネットワーク・インタフェースは常にアクティブであり、その他のインタフェースはアイドルの状態になります。アクティブなインタフェースに障害が発生すると、アイドル状態のセット・メンバのうちの 1 つのインタフェースが、同じ IP アドレスを使用して、フェイルオーバー時間 (調整可能) 内にオンラインに切り替わります。図 1-8 に、NetRAIN セットの一部として 3 つのインタ

フェースを持つホスト A を示します。この NetRAIN 仮想インタフェースには、アドレス 16.1.1.1 が割り当てられています。

図 1-8: 複数のインタフェース



NetRAIN は、ネットワーク障害の可能性を検出して報告するツール NIFF (Network Interface Failure Finder) を使用して、セット内のネットワーク・インタフェースの状態をモニタリングします。このツールは、NetRAIN と独立して使用することができます。NIFF についての詳細は、NIFF(7) を参照してください。

1.4.4 ルーティング

ネットワークに接続されたすべてのシステム (ホストおよびルータ) は、別のネットワークにある他のシステムと通信できるように、ネットワーク・ルーティングをサポートするように構成しなければなりません。経路とは、あるシステムから別のシステムへパケットがネットワークを通過する際にたどる道筋です。このようにして、別のネットワークにある他のシステムと通信することができるようになります。経路は、各システムのルーティング・テーブルまたはルーティング・データベースに保存されます。各経路のエントリは、以下の項目から成り立ちます。

- デスティネーション・アドレス (ネットワークまたはホスト)
- システムからデスティネーションまでたどる際の次のホップ・アドレス
- 経路がインタフェース経由の場合は、ネットワーク上でのシステム
のアドレス
- ネットワーク・インタフェース (たとえば、tu0 や fta0)
- メトリクス (たとえば、ホップ・カウントや MTU)

インターネット制御メッセージ・プロトコル (ICMP) のリダイレクト・メッセージにより、ルーティング・テーブルにその他の経路が追加されることがあります。これらは、ローカル・ネットワーク内の別のルータにトラフィックを転送するようにホストに伝えるための、ルータからホストに送られるメッセージです。

1.4.5 LAG インタフェース

リンク・アグリゲーション (LAG) インタフェースを使用すると、複数のネットワーク・アダプタを備えたシステムの可用性が高まり、フォルト・トレランス、負荷分散が実現されます。また、リンク・アグリゲーション、つまりトランッキングにより、管理者は1つ以上の物理イーサネット NIC を組み合わせて、1本の論理リンクを作成することができます。このリンク・アグリゲーション・グループは、上位層のソフトウェアからは単一の論理インタフェースとして見えます。この単一の論理リンクでは、単一のインタフェースよりも速いデータ転送速度でトラフィックを伝送できます。これは、リンク・アグリゲーション・グループを構成するすべての物理ポートにトラフィックが分散されるためです。

リンク・アグリゲーションを使用すると、以下の利点があります。

- ネットワーク帯域幅の増加 — 帯域幅は、リンク・アグリゲーション・グループに追加されるポートまたは NIC の数と種類に応じて増加します。
- フォルト・トレランス — リンク・アグリゲーション・グループ内のポートに障害が発生すると、ソフトウェアにより障害が検出され、トラフィックは他の利用可能なポートに切り替えて転送されます。この機能は、DEGPA (alt) および DE60x (ee) デバイスのみで利用できます。
- 負荷分散 — リンク・アグリゲーション・グループは、受信および送信両方のトラフィックの負荷分散を行います。パケットの送信時には、負荷分散アルゴリズムを使用して、パケットを送信する接続ポートを決定します。

リンク・アグリゲーション・グループの仮想インタフェースは、サーバ間、およびサーバとスイッチ間のポイント・ツー・ポイント接続で使用できます。詳細は、『ネットワーク管理ガイド：接続編』を参照してください。

1.5 ファイル・システム・リソース

AdvFS (Advanced File System) およびネットワーク・ファイル・システム (NFS) では、ファイル・システムのチューニングが重要です。一般的に、ファイル・システムのチューニングを行うと、入出力多用型のユーザ・アプリケーションの性能が改善されます。以降の項では、AdvFS、UNIX ファイル・システム (UFS)、および NFS に関するファイル・システム・リソースについて説明します。

1.5.1 AdvFS の使用

AdvFS (Advanced File System) ファイル・システムは従来の UNIX ファイル・システム (UFS) と異なります。AdvFS では、システムをシャットダウンせずにいつでもシステムの構成を変更することができます。AdvFS Utilities を使用する AdvFS では、マルチボリューム・ファイル・システムがサポートされるため、システム要件の変化に従って、簡単にストレージを追加または削除できます。さらに、LSM (Logical Storage Manager) ボリュームとストレージ・エリア・ネットワーク (SAN) を AdvFS ストレージとして使用できます。

これに対し、UFS モデルは柔軟性がありません。各ディスク (またはディスク・パーティション) に 1 つのファイル・システムがあります。UFS のディレクトリ階層構造は、物理ストレージ層と強く結びついています。ファイル・システムがいっぱいになった場合、この結びつきのため、一部のファイルを別のディスクに移動すると、それらのファイルの完全パス名が変わります。論理的なディレクトリをディレクトリ・サブツリーに分割して、それらのサブツリーを別のディスクに移動する作業は、慎重に行う必要があります。広範囲にわたって計画を立てたとしても、UFS モデルではディレクトリ構造の調節に限界があります。

1.5.1.1 UBC の使用

データが頻繁に再利用される場合は、キャッシングを行うと性能が向上します。AdvFS では、ユニファイド・バッファ・キャッシュ (**UBC**) という動的メモリ・キャッシュを使用して、ファイルのメタデータおよびユーザ・データを管理します。

キャッシングに UBC を使用することで、AdvFS は、メモリが利用できる限り、ファイル・データをメモリに保持することができます。別のシステム・リソースが、ファイル・システムのキャッシュとして使用されているメ

モリの一部を必要とする場合、UBC はファイル・システムが使用しているメモリの一部を再生して、メモリを要求しているリソースに対してそのメモリを再割り当てします。

AdvFS は、UBC を使用してキャッシングを制御するので、キャッシュのチューニングは UBC のチューニング・パラメータを使用していきます。UBC のチューニング・パラメータは以下のとおりです。

- UBC が同時に使用できる物理メモリの最大割合を変更する変数
- UBC がディスクへのページの書き込みを開始する前に、ダーティでなければならないページの割合
- 1 つのファイルをキャッシュするために使用可能な、UBC に割り当てられているメモリの最大量

これらのパラメータを変更する際のガイドラインについては、第 11 章を参照してください。

1.5.2 NFS の使用

ネットワーク・ファイル・システム (NFS) を使用すると、ネットワークを通じて透過的にファイルにアクセスできます。NFS は、小規模で単純なネットワークから、大規模で複雑なネットワークまで、各種のネットワーク・トポロジをサポートします。NFS は、ユニファイド・バッファ・キャッシュ (UBC) を、仮想メモリ・サブシステムおよびローカル・ファイル・システムと共有します。

NFS 要求の処理で CPU 時間と実時間の大部分が消費されるため、NFS ではファイル・システムのチューニングが重要です。理想的には、UBC ヒット率が高くなければなりません。UBC のヒット率を高くするには、メモリを増やしたり、他のファイル・システムのキャッシュのサイズを小さくする必要があります。

NFS は単純な、状態のない (stateless) プロトコルを使っているため、各クライアントの要求は完全に自己完備型である必要があります。また、サーバは各要求を完全に処理してから、クライアントに肯定応答を送信する必要があります。

NFS のサービスだけを行っているシステムで性能を改善するのは、汎用タイムシェアリング用に使用しているシステムでのチューニングとは異なります。これは、NFS サーバは少数の小規模なユーザ・レベル・プログラ

ムだけを実行しており、システム・リソースはほとんど使用しないからです。ページングやスワッピングも最低限しか発生しないので、メモリ・リソースは、ファイル・システムのデータのキャッシングに重点を置いて割り当てなければなりません。

NFS のチューニングについての詳細は、第 5 章および第 10 章を参照してください。

1.6 メモリ・リソース

システムの性能を確保するためには、メモリ・リソースが十分にあることが重要です。CPU やメモリを多用するアプリケーションを動作させる構成では、64 ビット・アーキテクチャを使用し、マルチプロセッシングで、メモリ容量が最低でも 2 GB の大規模メモリ (VLM) システムが必要です。大規模データベース (VLDB) システムは、複雑なストレージ構成も使用する VLM システムです。

物理メモリの総量は、システムに実装されているメモリ・ボードの容量によって決まります。仮想メモリ (vm) サブシステムは物理メモリを、ページという 8 KB の単位で管理します。ページは、次の領域に分配されます。

- 静的固定メモリ

ブート時に割り当てられ、オペレーティング・システムのデータやテキスト、システム・テーブル用に使用されます。UNIX ファイル・システム (UFS) や CD-ROM ファイル・システム (CDFS) の最近アクセスされたメタデータを保持するメタデータ・バッファ・キャッシュも、静的固定メモリを使用します。

- 動的固定メモリ

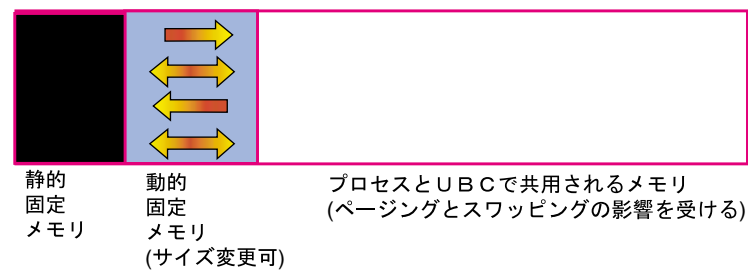
動的固定メモリは、システム・ハッシュ・テーブルなどの、動的に割り当てられるデータ構造体用に使用されます。ユーザ・プロセスも、`mlock` 関数などの仮想メモリ・ロック・インタフェースを使用して、アドレス空間に動的固定メモリを割り当てます。動的固定メモリの量は、要求に従って変化します。vm サブシステムの属性 `vm_syswiredpercent` は、ユーザ・プロセスが固定できるメモリの最大容量 (省略時は、物理メモリの 80 パーセント) を指定します。

- プロセスおよびデータ・キャッシュ用の物理メモリ

固定されていない物理メモリを、ページング可能メモリといいます。このメモリは、プロセスが最近アクセスした 可変メモリ (変更可能な仮想アドレス空間) と、ファイル・バック・メモリ (プログラム・テキストまたはシェアード・ライブラリ用に使用されるメモリ) 用に使われます。ページング可能メモリは、AdvFS のメタデータやファイル・データその他、最近アクセスされた UFS ファイル・システム・データ (読み取り書き込み用、およびマッピングされたファイル領域のページ・フォールト用) のキャッシュにも使用されます。仮想メモリ・サブシステムは、プロセスおよびファイル・システムの要求に応じて、物理ページを割り当てます。

図 1-9 に、物理メモリの区分を示します。

図 1-9: 物理メモリの用途



ZK-1359U-AIJ

1.6.1 ページングとスワッピング

物理メモリは、すべてのアクティブなプロセスが使用するリソースです。多くの場合、システム上のアクティブなプロセスすべてに十分な物理メモリを割り当てることはできません。使用可能な物理メモリを増やすために、`vm` サブシステムは使用可能な物理メモリの量をモニタリングして、スワップ・デバイスという 2 次メモリ・デバイスにページを移動することがあります。スワップ・デバイスは、ディスクの構成可能なセクション内にあるブロック・デバイスです。カーネルは、プロセスがページを参照すると、要求に応じて (on demand) スワップ・デバイスからページを取り出します。このメモリ管理方式をページングといいます。

負荷が重い場合、プロセス全体がスワップ・デバイスに移動されることがあります。スワッパと呼ばれるプロセスが、物理メモリとスワップ・デバイス間の移動を管理します。このメモリ管理方式をスワッピングといいます。

ページングおよびスワッピングに関連する属性のチューニング方法については、第 12 章を参照してください。

1.6.2 データのキャッシング

カーネルは、最近アクセスされたデータをメモリにキャッシュ（一時的に保管）します。データは何度も再使用され、ディスクからデータを取り出すよりもメモリから取り出すほうが高速なので、データをキャッシュすると効率が上がります。カーネルがデータを取り出すときには、まずデータがキャッシュされているかどうかをチェックします。データがキャッシュされていれば、すぐにそのデータを返します。キャッシュされていなければ、ディスクからデータを取り出してキャッシュします。キャッシュされたデータが後で再使用されると、ファイル・システムの性能が改善されます。

キャッシュされるデータには、ファイルに関する情報、ユーザまたはアプリケーションのデータ、メタデータ（ファイルなどのオブジェクトを記述するデータ）などがあります。キャッシュされるデータのタイプを、以下のリストに示します。

- ファイル名と、それに対応する `vnode` は `namei` キャッシュにキャッシュされます（11.1.2 項）。
- UFS のユーザおよびアプリケーション・データと、AdvFS のユーザおよびアプリケーション・データとメタデータは、ユニファイド・バッファ・キャッシュ (UBC) にキャッシュされます（11.1.3 項）。
- UFS のメタデータは、メタデータ・バッファ・キャッシュにキャッシュされます（11.1.4 項）。
- AdvFS のオープン・ファイル情報はアクセス構造体にキャッシュされます（11.1.5 項）。

1.7 CPU リソース

CPU には、さまざまなプロセッサ速度やオンボード・キャッシュ・サイズのものがあります。さらに、シングル CPU システムか、マルチプロセッサ・システム（2 つ以上のプロセッサが共通の物理メモリを共有する）かを選択できます。大規模なデータベース環境のように CPU を多用する環境では、作業負荷に対応するためにマルチプロセッシング・システムが必要になります。

マルチプロセッシング・システムの一例としては、シンメトリック・マルチプロセッシング (SMP) システムがあります。このシステムでは複数の CPU

が、共通メモリ上の同じオペレーティング・システムを実行し、命令を同時に実行します。

プログラムの実行中、オペレーティング・システムはデータや命令を、CPU キャッシュ、物理メモリ、およびディスク・スワップ領域の間で移動します。データや命令のアクセス速度は、それらが置かれている場所によって異なります。表 1-6 では、各種のハードウェア・リソースについて説明します。

表 1-6: メモリ管理ハードウェア・リソース

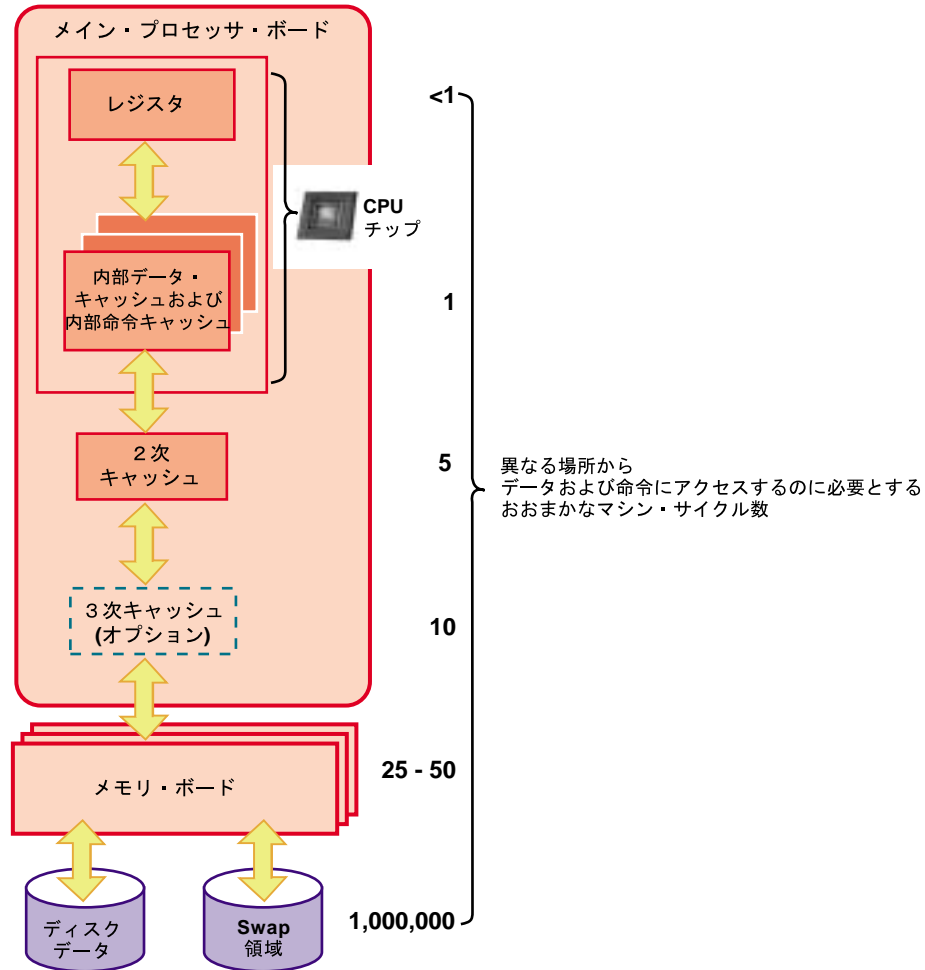
リソース	説明
CPU チップ・キャッシュ	CPU チップ内に実装された各種の内部キャッシュ。サイズはプロセッサにより異なり、最大で 64 KB である。このキャッシュには、変換索引バッファ (TLB)、高速内部仮想物理間変換キャッシュ、高速内部命令キャッシュ、高速内部データ・キャッシュなどがある。
2 次キャッシュ	2 次直接マップ物理データ・キャッシュは CPU の外部にあるが、通常はメイン・プロセッサ・ボード上に実装される。2 次キャッシュのブロック・サイズは、プロセッサのタイプによって異なり、32 ~ 256 バイトである。2 次キャッシュのサイズは、128 KB ~ 8 MB である。
3 次キャッシュ	3 次キャッシュは、一部の Alpha CPU では利用できない。その他は、2 次キャッシュと同じ。
物理メモリ	物理メモリの実際の容量は、さまざまである。
スワップ領域	スワップ領域は、1 つ以上のディスクまたはディスク・パーティション (ブロック型特殊デバイス) からなる。

ハードウェア・ロジックと **PAL (Privileged Architecture Library)** コードによって、CPU キャッシュ、2 次キャッシュ、3 次キャッシュ、物理メモリ間でのアドレスおよびデータの移動の大半が制御されます。この移動は、オペレーティング・システムからは見えません。

ディスク入出力の速度は遅いため、キャッシュと物理メモリ間の移動は、ディスクと物理メモリ間の移動よりかなり高速です。アプリケーションでは、可能な限りディスク入出力操作を避け、キャッシュを利用してください。

図 1-10 では、プログラムの実行中に命令やデータがハードウェア構成要素間をどのように移動するかと、各ハードウェア構成要素からデータや命令をアクセスするのに必要なマシン・サイクルを示します。

図 1-10: メモリ・ハードウェア間での命令とデータの移動



ZK-1362U-AIJ

CPU , 2 次キャッシュ , 3 次キャッシュについての詳細は , 『 *Alpha Architecture Reference Manual* 』 を参照してください。

CPU の性能を最適化する方法は , いくつかあります。プロセスを再スケジューリングしたり , クラス・スケジューラを使用してタスクやアプリケーションに割り当てる CPU 時間の割合を指定することができます。この方法により , CPU 時間の多くを重要なプロセス用に予約し , 比較的重要でないプロセスの CPU 使用を制限することができます。詳細については , 13.2.2 項を参照してください。

1.8 作業負荷に適したリソース・モデルの明確化

構成を計画したりチューニングをする前に、作業負荷のリソース・モデルを明確にしなければなりません。つまり、アプリケーションがメモリと CPU のどちらを多用するかや、アプリケーションがディスク入出力やネットワーク入出力をどのように実行するかを調べなければなりません。この情報は、作業負荷に適した構成やチューニング・ガイドラインを選択する上で役立ちます。

たとえば、データベース・サーバが大規模なシーケンシャル・データ転送を行っている場合は、高い帯域幅の構成を選択します。アプリケーションが頻繁にディスク書き込み操作を行う場合は、RAID1 (ミラー化) 構成を選択すべきではありません。

表 1-7 を使って、作業負荷のリソース・モデルを判断し、各モデルで可能な構成ソリューションを把握してください。

表 1-7: リソース・モデルと、それに対する構成ソリューション

リソース・モデル	構成ソリューション
CPU 多用	マルチプロセッシング・システム、高速の CPU、またはハードウェア RAID システム
メモリ多用	VLM システムまたは大規模なオンボード CPU キャッシュ
大量のディスク・ストレージが必要	入出力能力の高いシステム、LSM、またはハードウェア RAID サブシステム
短いディスク遅延時間が必要	半導体ディスク、高速ディスク、RAID アレイ、またはファイバ・チャネル
高スループットが必要	半導体ディスク、高性能 SCSI アダプタ、ストライピング、RAID5、または動的パリティ RAID (アダプティブ RAID3/5)
高帯域幅が必要	半導体ディスク、高性能アダプタ、ワイド・デバイス、RAID3、または動的パリティ RAID
大規模なシーケンシャル・データ転送を頻繁に実行	高性能ディスク、ワイド・デバイス、ストライピング、パリティ RAID
小規模なデータ転送を頻繁に実行	RAID5
読み取り転送を主に発行	ミラーリング、RAID5、またはストライピング
書き込み転送を主に発行	Prestoserve またはライトバック・キャッシュ

表 1-7: リソース・モデルと、それに対する構成ソリューション (続き)

リソース・モデル	構成ソリューション
頻繁にネットワーク操作を実行	複数のネットワーク・アダプタ, NetRAIN, または高性能アダプタ
アプリケーションの高可用性が必要	クラスタ
データの高可用性が必要	ミラーリング (特に, 異なるバスにまたがったミラーリング) またはパリティ RAID
ネットワーク入出力多用	複数のネットワーク・アダプタまたは NetRAIN

1.9 一般的なチューニング対象サブシステム

本書では多くのサブシステム属性のチューニング方法を説明します。システムに固有の、性能問題に影響を与える属性だけをチューニングすることをお勧めします。通常チューニングの対象となるのは、次の5つのサブシステムです。

- 仮想メモリ (vm)

- new_wire_method (4.4.1.1 項)
 - rad_gh_regions (4.4.1.2 項)
 - gh_chunks (4.4.1.2.2 項)
 - ubc_maxpercent (4.4.1.3 項)
 - ubc_borrowpercent (4.4.1.4 項)
 - vm_ubcseqstartpercent (4.4.1.6 項)
 - vm_ubcdirtypercent (4.4.1.7 項)
 - vm_swap_eager (4.4.1.8 項)

- プロセス間通信 (ipc)

- ssm_threshold (4.4.4.1 項)
 - shm_max (4.4.4.2 項)
 - shm_min (4.4.4.3 項)
 - shm_mni (4.4.4.4 項)
 - shm_seg (4.4.4.5 項)

- プロセス (proc)

- per_proc_stack_size (4.4.6.1 項)
 - max_per_proc_stack_size (4.4.6.2 項)
 - per_proc_data_size (4.4.6.3 項)
 - max_per_proc_data_size (4.4.6.4 項 と 6.2.2.4 項)

per_proc_address_space (4.4.6.5 項)
max_per_proc_address_space (4.4.6.6 項と 6.2.2.5 項)
max_proc_per_user (4.4.6.7 項と 6.2.2.2 項)
max_threads_per_user (4.4.6.8 項と 6.2.2.3 項)
maxusers (4.4.6.9 項と 6.2.2.1 項)

- インターネット (inet)

udp_sendspace (4.4.5.1 項)
udp_recvspace (4.4.5.2 項)
udp_unserreserved (4.4.5.3 項)
tcblhashsize (6.2.1.1 項)
pmtu_enabled (6.2.1.2 項)
ipport_userreserved (6.2.1.3 項)

- ソケット (socket)

somaxconn (6.2.3.1 項)
sominconn (6.2.3.2 項)
sbcompress-threshold (6.2.3.3 項)

本書では、アプリケーションのタイプ別、およびコンポーネント別にシステムをチューニングする方法を説明します。システムのチューニングを行う前に、システムのハードウェア構成を理解する必要があります (詳細は、1.1 節を参照)。最も一般的なチューニング対象サブシステムを本書で紹介しますが、性能問題に関係する属性だけをチューニングしてください。

システム性能を改善するために変更すべき属性を詳細に説明している章は、以下のとおりです。

- アプリケーション・タイプ別のチューニング (Part 2)

Oracle のチューニング (第 4 章)
ネットワーク・ファイル・システム のチューニング (第 5 章)
インターネット・サーバのチューニング (第 6 章)

- コンポーネント別のチューニング (Part 3)

システム・リソース割り当ての管理 (第 8 章)
ディスク・ストレージ性能の管理 (第 9 章)
ネットワーク性能の管理 (第 10 章)
ファイル・システム性能の管理 (第 11 章)
メモリ性能の管理 (第 12 章)
CPU 性能の管理 (第 13 章)

サブシステム属性の詳細は、sys_attrs(5) を参照してください。

システム情報と性能情報の取得

性能上の問題点を見つけたり，どの分野で性能が低下しているかを把握するには，性能について広範囲の情報を収集しなければなりません。

症状が明白な場合や，性能上の問題点が明示的に通知される場合もあります。たとえば，アプリケーションが完了するまでの時間が長かったり，システムのリソース不足を示すメッセージがコンソールに表示される場合があります。このような場合以外は，問題点や性能不足は明白ではなく，システム性能をモニタリングして初めて検出できます。

システム性能情報を取得するために使用できる各種のコマンドとユーティリティがあります。統計情報はさまざまな条件のもとで取得することが重要です。データのセットを比較することが，性能問題の診断に役に立ちます。

たとえば，アプリケーションがシステム性能にどのように影響を与えているのか調べるには，アプリケーションを実行していない状態で性能の統計情報を取得し，その後，アプリケーションを実行してから，同じ統計情報を取得します。異なるデータのセットを比較すれば，アプリケーションのメモリ，CPU，またはディスク入出力の消費量がわかります。

さらに，正確な性能情報を得るには，アプリケーション処理の複数の段階で情報を収集しなければなりません。たとえば，ある段階では入出力を多用し，別の段階ではCPUを多用するアプリケーションが考えられます。

この章では，以下の内容について説明します。

- 性能問題の解決のための方法論的アプローチ (2.1 節)
- システム・イベント情報の取得 (2.2 節)
- 情報収集用の基本的なツールの使用 (2.3 節)
- 情報収集用の補助的なツールの使用 (2.4 節)
- 継続的な性能モニタリング (2.5 節)

性能上の問題点を見つけたり，どの分野で性能が低下しているかを把握したら，適切な解決方法を見つけることができます。システム性能の改善のために，アプリケーション別にチューニングを行う方法は，Part 2を参照してください。構成要素別にチューニングを行う方法は，Part 3を参照してください。

2.1 性能問題の解決のための方法論的アプローチ

性能問題を診断するための推奨手順は5つあります。最初に，性能と可用性に関する用語と概念を，十分に理解しておく必要があります。詳細は，第1章を参照してください。

また，アプリケーションがシステム・リソースをどのように使用しているかについても，理解している必要があります。すべての構成とチューニングのガイドラインが，すべてのタイプの作業負荷に適するわけではないからです。たとえば，アプリケーションはメモリ多用型なのか，CPU 多用型なのかを調べる必要があります。あるいは，ディスク操作が多いのか，ネットワーク操作が多いのかを調べる必要があります。構成に適したリソース・モデルについて詳細は，1.8 節を参照してください。

性能問題を診断するには，以下の手順を実行します。

1. 最初に，システムのハードウェア構成を理解する必要があります。ハードウェア構成要素を調べて，管理するには，`hwmgr` ユーティリティを使用します。詳細は，1.1 節または 2.3.1 項を参照してください。
2. システム性能のチューニングに使用するオペレーティング・システムのパラメータとカーネル属性を分析してから，`sys_check` ユーティリティを実行します。このツールは性能問題を診断するために使用できます。詳細は，2.3.3 項を参照してください。
3. ソフトウェアの構成エラーを確認し，認識しておきます。`sys_check` を使用して，性能問題を診断することもできます。システム・イベントの情報の取得方法は，2.2 節を参照してください。
4. どのようなタイプのアプリケーションを使用しているか確認し，アプリケーションを，Oracle，ネットワーク・ファイル・システム，またはインターネット・サーバに分類します。システムをアプリケーション別にチューニングする場合は，以下の章を参照してください。
 - Oracle のチューニング (第 4 章)
 - ネットワーク・ファイル・システムのチューニング (第 5 章)

- インターネット・サーバのチューニング (第 6 章)
- 5. ボトルネック, つまり性能低下の原因となっているシステム・リソースを調べます。以下の情報をプロットして, 性能問題を判断します。
 - CPU — アイドル・システム時間とユーザ時間
 - メモリ — プロセス, UBC および固定メモリによって使用されているアクティブ・ページと非アクティブ・ページの合計。
 - ディスク入出力 — 秒当たりのトランザクション数と秒当たりのブロック数

`collect` コマンドを使用して, システムに負荷がかかっている時間, または性能問題が現れている時間の性能データを取得します。性能情報を取得した後, `collgui` グラフィカル・インタフェースを使用して, データをプロットします。 `collgui` の使用方法は, 2.3.2.2 項を参照してください。作業負荷に適したリソース・モデルについては, 1.8 節を参照してください。

2.2 システム・イベントについての情報の取得

システム・イベントを継続的にモニタリングし, 重大な問題が発生した場合に警告を発するルーチンを設定してください。定期的にイベントやログ・ファイルを調べると, 性能や可用性に影響が現われる前に問題点を修正できます。また, 性能上の問題の診断が容易になります。

システム・イベントのログは, システム・イベント・ロギング機能と, バイナリ・イベント・ロギング機能によって取得されます。システム・イベント・ロギング機能では, `syslog` 関数を使用して, ASCII 形式でイベントのログを取得します。 `syslogd` デーモンは, 各種のカーネル, コマンド, ユーティリティ, アプリケーション・プログラムで取得されたログ・メッセージを収集します。そしてこのデーモンは, `/etc/syslog.conf` イベント・ロギング構成ファイルで指定されているとおりに, メッセージをローカル・ファイルに書き込むか, メッセージをリモート・システムに転送します。これらの ASCII ログ・ファイルを定期的にモニタリングして, 性能情報を入手してください。

バイナリ・イベント・ロギング機能では, カーネル内のハードウェア・イベントとソフトウェア・イベントを検出し, 詳細情報のログをバイナリ形式のレコードで記録します。バイナリ・イベント・ロギング機能では, `binlogd` デーモンを使用して, 各種のイベント・ログ・レコードを収集します。そ

してこのデーモンは、省略時の構成ファイル `/etc/binlog.conf` で指定されているとおりに、レコードをローカル・ファイルに書き込むか、リモート・システムに転送します。

バイナリ・イベント・ログ・ファイルは、次の方法で調査できます。

- イベント・マネージャ (EVM) は、バイナリ・ログ・ファイルを使用して、イベント通知を必要とする関係者へイベント情報を通知し、すぐにまたは後でアクションをとれるようにします。EVM についての詳細は、2.2.1 項を参照してください。
- DECEvent ユーティリティは、規則に基づいて変換と通知を行うユーティリティであり、バイナリ・エラー・ログ・イベントのイベント変換を行います。EVM は DECEvent の変換機能 `dia` を使用して、バイナリ・エラー・ログ・イベントを、人が読める形式に変換します。Compaq Analyze は、EV6 シリーズのいくつかのプロセッサで、同じような役割を果たします。

DECEvent についての詳細は、2.2.2 項 または `dia(8)` を参照してください。

Compaq Analyze についての詳細は、2.2.3 項 または `ca(8)` を参照してください。

また、クラッシュ・ダンプ・サポートをシステムに構成してください。性能上の重大な問題によってシステム・クラッシュが引き起こされることがあります。クラッシュ・ダンプ分析ツールを使用すると、性能上の問題点を診断するのに役立ちます。

イベント・ログとクラッシュ・ダンプについての詳細は、『システム管理ガイド』を参照してください。

2.2.1 イベント・マネージャの使用

イベント・マネージャ (EVM) を使用すると、イベント情報を取得して、この情報を必要とする関係者へ通知し、すぐにまたは後でアクションをとることができます。イベント・マネージャには、次の機能があります。

- カーネル・レベルおよびユーザ・レベルのプロセスおよび構成要素がイベントを通知できるようにする。
- プログラムやユーザなどのイベントの受信者が、選択したイベントの発生時に通知を受け取れるようにする。

- バイナリ・ロガー・デーモンなどの、既存のイベント・チャンネルをサポートする。
- ユーザがイベントを参照できるように、グラフィカル・ユーザ・インタフェース (GUI) を提供する。
- イベントを通知したり受け取ったりするルーチンをプログラマが作成できるように、アプリケーション・プログラミング・インタフェース (API) ライブラリを提供する。
- EVM 環境を構成し管理するためのコマンド行ユーティリティ (管理者用) と、イベントの通知や取り出しを行うためのコマンド行ユーティリティ (ユーザ用) をサポートする。

EVM についての詳細は、『システム管理ガイド』を参照してください。

2.2.2 DECevent の使用

DECevent ユーティリティは、バイナリ・イベント・ロギング機能を使用して継続的にシステム・イベントをモニタリングし、イベントをデコードして、システム装置が取得したイベントの数と重大度を追跡します。DECevent はシステム・イベントを分析し、障害が発生した装置を切り分け、発生する可能性のある問題についての警告を行う通知メカニズム (メールなど) を提供します。

DECevent の分析および通知機能を使用するには、ライセンスを登録しなければなりません。または、これらの機能がサービス契約に含まれている場合もあります。DECevent を使って、バイナリ・ログ・ファイルを ASCII 形式へ変換するだけなら、ライセンスは不要です。

詳細は、『*DECevent Translation and Reporting Utility*』を参照してください。

2.2.3 Compaq Analyze の使用

Compaq Analyze は、単一のエラー/障害イベントの分析、多重イベントの分析および複合分析を行う障害分析ユーティリティです。Compaq Analyze は、従来のバイナリ・エラー・ログの他に、他のエラー/障害データ・ソースを使用してシステム分析を行います。

Compaq Analyze は、アクティブなエラー・ログをモニタリングし、発生したイベントをリアルタイムに処理することにより、自動分析をバックグ

ラウンドで行います。エラー・ログ・ファイルのイベントは、分析規則と照らし合わせてチェックされます。エラー・ログ・ファイルに規則に指定された条件を満たすイベントが 1 つ以上あると、分析エンジンがエラー・データを収集して、問題の説明や必要な修正作業などを記載した問題レポートを作成します。問題レポートは、いったん作成されると、通知の設定に従って配信されます。

最新の Alpha EV6 プロセッサでは、Compaq Analyze のみがサポートされ、DECevent はサポートされていないので注意してください。

Compaq Analyze とその他の WEBES (Web Based Enterprise Service Suite) ツール、およびドキュメントの最新バージョンは、次の場所からダウンロードできます。

<http://www.compaq.com/support/svctools/webes>

Web サイトからキットをダウンロードして、`/var/tmp/webes` に保存します。キットは、次のようなコマンドを用いて展開します。

```
# tar -xvf <tar file name>
```

次のコマンドを使用して、Compaq Web Based Enterprise Service Suite をインストールします。

```
# setld -l /var/temp/webes/kit
```

インストレーションの際には、省略時のオプションを選択して構いません。ただし、オプションの WEBES ツールをすべてインストールする必要はありません。EVM が使用するのは Compaq Analyze のみです。詳細は、別冊の Compaq Analyze のマニュアルと、ca(8) を参照してください。

2.2.4 システム課金とディスク・クォータの使用

各ユーザが消費しているリソースの情報を取得できるように、システム課金を設定してください。課金によって、CPU の使用量および接続時間、生成したプロセス数、メモリおよびディスクの使用量、入出力動作の回数、印刷動作の回数を追跡できます。

ディスク使用量を追跡して制御するには、AdvFS (Advanced File System) および UNIX ファイル・システム (UFS) のディスク・クォータを構成しなければなりません。ディスク・クォータを使用すると、ユーザが利用できるディスク・スペースを制限でき、ディスク・スペースの使用量を監視できます。

システム課金と UFS ディスクのクォータについては、『システム管理ガイド』を参照してください。AdvFS クォータについては、『AdvFS 管理ガイド』を参照してください。

2.3 情報収集用の基本的なツール

以下のユーティリティが、情報収集用の基本的なツールです。

- hwmgr ユーティリティ (2.3.1 項)
- collect ユーティリティ (2.3.2 項)
- sys_check ユーティリティ (2.3.3 項)

2.3.1 hwmgr ユーティリティを使用したハードウェア情報の収集

ハードウェア管理に使用する基本的なツールは、hwmgr コマンド行インタフェース (CLI) です。SysMan タスクのようなその他のインタフェースでは、hwmgr の機能の一部しか使用できません。hwmgr コマンドを使用すると、不慣れなシステムに接続して、構成要素の階層に関する情報が取得できます。そして特定の構成要素に対する属性を設定できます。

view コマンドを使用して、システム内のハードウェアの階層を表示します。このコマンドを使用すると、デバイスを制御しているアダプタと、アダプタがインストールされているバス上の位置を検出できます。以下の例は、クラスタを構成していない小規模システムのハードウェア構成要素階層を示しています。

```
# hwmgr view hierarchy

HWID: Hardware component hierarchy
-----
1: platform AlphaServer 800 5/500
2:   cpu CPU0
4:   bus pci0
5:     scsi_adapter isp0
6:     scsi_bus scsi0
18:    disk bus-0-targ-0-lun-0 dsk0
19:    disk bus-0-targ-4-lun-0 cdrom0
20:      graphics_controller trio0
8:      bus eisa0
9: serial_port tty00
10: serial_port tty01
11: parallel_port lp0
12: keyboard PCXAL
13: pointer PCXAS
14: fdi_controller fdi0
15:  disk fdi0-unit-0 floppy0
16:    network tu0
17:    network tul
output truncated
```

階層上で複数のエントリとして表示される構成要素もあります。たとえば、ディスクが2つのアダプタで共用される SCSI バス上にある場合、階層上には同一デバイスに対して2つのエントリが表示されます。同様のハードウェア構成要素の階層は、SysMan Station GUI を使用しても表示できます。SysMan Menu を実行するための方法は、『システム管理ガイド』を参照してください。13.2.2.6 項には、グラフィカル・インタフェースの使用方を示しています。有効なデータ・エントリについての詳細は、オンライン・ヘルプを参照してください。

階層内の特定の構成要素を参照するには、`grep` コマンドを使用します。次の例は、CPU のハードウェア構成要素についての出力を示しています。

```
# hwmgr view hierarchy | grep "cpu"
2:      cpu qbb-0 CPU0
3:      cpu qbb-0 CPU1
4:      cpu qbb-0 CPU2
5:      cpu qbb-0 CPU3
7:      cpu qbb-1 CPU5
8:      cpu qbb-1 CPU6
9:      cpu qbb-1 CPU7
10:     cpu qbb-2 CPU8
11:     cpu qbb-2 CPU9
12:     cpu qbb-2 CPU10
13:     cpu qbb-2 CPU11
```

階層表示コマンドは、システム階層に入っている現在登録済みのハードウェア構成要素を表示します。フラグ・ステータスを持った構成要素は、コマンド出力の中の以下のコードで区別できます。

- (!) 警告
- (X) クリティカル
- (-) 非アクティブ

これらのコードについての説明は、`hwmgr(8)` を参照してください。

システムに接続されているすべての SCSI デバイス (ディスクとテープ) を参照するには、次のコマンドを実行します。

```
# hwmgr show scsi
```

ホストから認識できる RAID アレイ・コントローラを参照するには、次のコマンドを実行します。

```
# hwmgr show scsi | grep scp
```

	SCSI		DEVICE	DEVICE	DRIVER	NUM	DEVICE	FIRST
HWDID:	DEVICEID	HOSTNAME	TYPE	SUBTYPE	OWNER	PATH	FILE	VALID PATH
266:	30	wf99	disk	none	0	20	scp0	[2/0/7]
274:	38	wf99	disk	none	0	20	scp1	[2/1/7]
282:	46	wf99	disk	none	0	20	scp2	[2/2/7]


```

290:  54    wf99    disk    none    0    20    scp3  [2/3/7]
298:  62    wf99    disk    none    0    20    scp4  [2/4/7]
306:  70    wf99    disk    none    0    20    scp5  [2/5/7]
314:  78    wf99    disk    none    0    20    scp6  [2/6/7]
322:  86    wf99    disk    none    0    20    scp7  [2/7/7]
330:  94    wf99    disk    none    0    20    scp8  [2/8/7]
338: 102    wf99    disk    none    0    20    scp9  [2/9/7]
346: 110    wf99    disk    none    0    20    scp10 [2/10/7]
354: 118    wf99    disk    none    0    20    scp11 [2/11/7]

```

この例で `scp` は、サービス制御ポートを表わし、RAID アレイ (HSG) が管理と診断のために提供するアドレスを示します。

`hwmgr` コマンドの詳細は、『ハードウェア管理ガイド』または `hwmgr(8)` を参照してください。

2.3.2 collect ユーティリティを使用したシステム情報の収集

`collect` ユーティリティは、特定のオペレーティング・システムのデータを記録または表示する、システム・モニタリング・ツールです。これは、ファイル・システム、メモリ、ディスク、プロセス・データ、CPU、ネットワーク、メッセージ・キュー、LSM、その他の重要なシステム性能情報も収集します。`collect` ユーティリティのオーバーヘッドはわずかで、高い信頼性があります。データ収集とプレイバックを柔軟に切り替える機能もあります。収集したデータは、ターミナルに表示するか、または圧縮または非圧縮のいずれかのデータ・ファイルとして保存します。そのデータ・ファイルは、コマンド行からの読み取りと操作が可能です。

`collect` ユーティリティは、信頼できる統計情報を提供するために、ページ・ロック関数の `plock()` を使用してメモリにロックし、省略時の設定では、システムによってスワップ・アウトされることはありません。また、優先度関数 `nice()` を使用して、優先度も上げています。それでも、通常の負荷のもとではシステムに影響を与えることはありません。極端に負荷が高い場合でも、システムには最小限の影響しか与えません。

`collect` ユーティリティは、`collgui` グラフィカル・ユーザ・インタフェース、またはコマンド行から起動します。グラフィカル・ユーザ・インタフェースを使用している場合には、コマンド行から `cfilt` を実行し、`collgui` とユーザ・スクリプトで使用される `collect` のデータをフィルタ処理してください。詳細は、`collect(8)` を参照してください。

次の例は、標準的な 10 秒間隔で完全なデータ収集を実行し、出力をターミナルに表示する方法を示しています。

```
# /usr/sbin/collect
```

このコマンドは、`vmstat(1)`、`iostat(1)`、`netstat(1)`、および `volstat(8)` のような、出力モニタリング・コマンドと似ています。

サブシステムをデータ収集の対象に含めるには、`-s` オプションを使用します。サブシステムをデータ収集の対象から外すには、`-e` (除外) オプションを使用します。

以下の出力では、ファイル・システム・サブシステムのデータ収集だけを指定しています。

```
# /usr/sbin/collect -sf
# FileSystem Statistics
# FS      Filesystem      Capacity  Free
0         root_domain#root      128       30
1         usr_domain#usr    700       147
3         usr_domain#var    700       147
```

オプション文字は、以下のサブシステムを表わしています。

- `p` — プロセス・データを指定
- `m` — メモリ・データを指定
- `d` — ディスク・データを指定
- `l` — LSM ボリューム・データを指定
- `n` — ネットワーク・データを指定
- `c` — CPU データを指定
- `f` — ファイル・システム・データを指定
- `tyy` — ターミナル・データを指定

プロセス・データを収集する場合は、`-s` (ソート) オプションと `-nX` (数) オプションを使用して、CPU 使用率でデータをソートし、`X` 個のプロセスだけを保存します。`Plist` オプションを使用すると、特定のプロセス群を対象にできます。ただし、`list` は、コンマで区切られたプロセス ID のリストです。

モニタリング対象のシステムに接続されているディスクが多い場合 (100 以上) は、`-D` オプションを使用して、ディスクの特定のサブセットをモニタリングするようにします。

collect ユーティリティに `-p` オプションを指定すると、複数のバイナリ・データ・ファイルを読み込み、サンプル番号を単調に増加させるような形で、1つのストリームとして出力します。複数のバイナリ入力ファイルを、1つのバイナリ・データ・ファイルに結合することもできます。その場合、`-p` オプションに入力ファイルを指定し、`-f` オプションに出力ファイルを指定します。

collect ユーティリティは入力ファイルを、コマンド行で指定する任意の順序で結合します。そのため、結合された出力ファイルを将来処理したいのであれば、入力ファイルは完全に時間軸に対応させる必要があります。異なるシステムの、異なる時刻に作成された、異なるサブシステムのサブセットのデータを収集した、複数のバイナリ入力ファイルを結合することもできます。このユーティリティでは、`-e`、`-s`、`-P`、および `-D` のようなフィルタ処理用オプションを使用できます。

詳細は、`collect(8)` を参照してください。

2.3.2.1 collect をシステム・リブート時に自動的に起動するための構成

collect を構成して、システム・リブート時に自動的に起動するようにできます。こうすると、サブシステムやプロセスの継続的モニタリングを行う場合に、特に役に立ちます。これは障害と性能問題の診断に有効です。各システムで `rcmgr` コマンドに `set` 操作を指定して使用し、次のように、`/etc/rc.config*` ファイル内に以下の値を構成します。

```
% rcmgr set COLLECT_AUTORUN 1
```

1 の値を使用すると、collect はシステム・リブート時に自動的に起動されます。0 の値 (省略時の値) を使用すると、collect はリブート時には起動されません。

```
% rcmgr set COLLECT_ARGS " -ol -i 10:60 \  
-f /var/adm/collect.dated/collect -H d0:5,1w "
```

ヌル値を指定すると、collect は以下に示す省略時の値で起動されます。

```
-i60, 120 -f /var/adm/collect.dated -W 1h -M 10, 15
```

collect の直接出力は、NFS にマウントされているファイル・システムではなく、ローカル・ファイル・システムに書き出す必要があります。これはシステムやネットワークの問題の発生時に重要な診断データを失うことを防止するためです。また、ファイル・システムが応答しないことで collect の出力がブロックされて、その結果システム問題が発生することを防ぐためです。

詳細は、rcmgr(8) を参照してください。

2.3.2.2 collect データ・ファイルのプロット

collect コマンドの collgui グラフィカル・インタフェース、または collect コマンドの cflit フィルタのいずれかを使用して、collect データ・ファイルを Excel にエクスポートします。

注意

collgui を実行するには、Perl と Perl/TK が必要です。これは次の URL で collect の FTP サイトから、無償でダウンロードできます。 <ftp://ftp.digital.com/pub/DEC/collect>

collgui グラフィカル・インタフェースを使用して、情報をプロットするには、以下の手順を実行します。

1. collgui をデバッグ・モードで実行します。

```
>> collgui -d "collect datafile"
```
2. サブシステムを選択し、[Display] をクリックします。
3. collgui を起動したシェルに戻ります。collgui によって、`/var/tmp` ディレクトリが作成されたことがわかります。ファイル名は、`collgui.xxxx` です。ここで、`xxxx` は整数です。データ・ファイル (`collgui.xxxx`) は、Excel にエクスポートすることができます。これを Windows システムへコピーします。
4. Windows システムで Excel を起動し、`collgui.xxxx` を開きます。「ファイルの種類」フィールドでは、「すべてのファイル (*.*)」を選択する必要があります。
5. Excel 2000 で、「テキストファイルウィザード」が開きます。
6. Excel 2000 で、「データのファイル形式」で [コンマやタブなどの区切り文字によってフィールドごとに区切られたデータ] を選択し、[次へ] を選択します。
7. Excel 2000 で、「区切り文字」として [タブ] と [スペース] を選択します。そして、[次へ] を選択します。
8. 「データのプレビュー」ペインで、シフト・キーを使用してインポートする列を選択し、[完了] を選択します。

9. ワークシートに列が表示されます。

cfilt collect フィルタを使用して情報をプロットするには、以下の手順を実行します。

1. cfilt を使用して、データ・ファイルを作成します。たとえば、データの処理に使用されている system time, physical memory, そして「Single CPU」フィールドに待ち時間, user+system time を表示するように選択するには、次のコマンドを実行します。

```
cfilt -f "collect datafile" 'sin:WAIT:USER+SYS' 'pro:System#:RSS#'  
> /var/tmp/collgui.xxxx
```

collgui データ・ファイルを Windows システムにコピーします。

2. 上述の collgui での手順中、手順 4～9 を実行します。

詳細は、次の Web アドレスを参照してください。

http://www.tru64unix.compaq.com/collect/collect_faq.html

2.3.3 sys_check ユーティリティを使用した構成のチェック

sys_check ユーティリティは、システム性能のチューニングに使用する、オペレーティング・システムのパラメータとカーネル属性の分析を実行します。このユーティリティは、メモリ・リソースと CPU リソースをチェックし、SMP システムとカーネル・プロファイルの性能データとロック統計情報を作成し、警告とチューニングのガイドラインを出力します。

sys_check ユーティリティは、システム構成を説明する HTML ファイルを作成するので、問題を診断するために使用することができます。sys_check で作成されるレポートには、現在の設定で問題があれば、警告が表示されます。システム状態を完全に概観し、制御するには、sys_check ユーティリティは、イベント管理およびシステム・モニタリング・ツールとともに使用する必要があります。

高度なチューニング・ガイドラインを適用する前に、sys_check ユーティリティが出力した構成およびチューニングのガイドラインを適用することを検討してください。

注意

`sys_check` ユーティリティの動作中は、システムの性能が低下することがあります。性能への影響を最小限に抑えるには、負荷が少ない時間帯にこのユーティリティを実行してください。

`sys_check` ユーティリティは、SysMan のグラフィカル・ユーザ・インタフェースとコマンド行のどちらでも起動できます。`sys_check` にコマンド行オプションを指定しなかった場合は、基本的なシステム分析が行われ、構成およびチューニングのガイドラインを示す HTML ファイルが作成されます。コマンド行で指定できるオプションを、次に示します。

- `-all` オプションでは、セキュリティ情報および `setld` インベントリ確認を含む、すべてのサブシステム情報が出力されます。
- `-perf` オプションでは、性能データだけが出力され、構成データは出力されません。完了するまでに、5 ~ 10 分の時間がかかります。
- `-escalate` オプションでは、障害を報告する場合に必要なエスカレーション・ファイルが作成されます。

詳細は、`sys_check(8)` を参照してください。

2.4 情報収集用の補助的なツール

以下に示すユーティリティが、性能情報収集用の補助的なツールです。

システム情報の収集

- `lockinfo` ユーティリティ (2.4.1 項)
- `sched_stat` ユーティリティ (2.4.2 項)

ネットワーク情報の収集

- `nfsstat` ユーティリティ (2.4.3 項)
- `tcpdump` ユーティリティ (2.4.4 項)
- `netstat` コマンド (2.4.5 項)
- `ps axlmp` コマンド (2.4.6 項)
- `nfsiod` デモン (2.4.7 項)

- `nfswatch` コマンド (2.4.8 項)

2.4.1 `lockinfo` ユーティリティを使用したロック統計情報の収集

`lockinfo` ユーティリティを使用すると、カーネル SMP ロックのロック統計情報の収集と表示が行われます。このユーティリティはデータを収集するために、`/dev/lockdev` 擬似ドライバを使用します。ロック統計情報は、`generic` サブシステムの `lockmode` 属性が、2 (省略時の値)、3、または 4 のときに収集されます。

`lockinfo` で統計情報を収集するには、以下の手順を実行します。

1. システムに作業負荷を発生させ、それが安定状態になるまで待ちます。
2. `sleep` コマンドを指定し、適当な秒数を `cmd_args` に指定して `lockinfo` を起動します。これにより、`lockinfo` は `sleep` コマンドを実行している間の統計情報を収集します。
3. 最初の一連の結果に基づき、`lockinfo` を再び実行します。今度は、システムの性能問題の原因になる可能性のある (たとえばミス率の高い) 結果を表示するために、いずれかのロック・クラスの特定の情報を要求します。

次の例は、60 秒間だけ各プロセッサのロック統計情報を収集する方法を示しています。

```
# lockinfo -percpu sleep 60
hostname:      sysname.node.corp.com
lockmode:      4 (SMP DEBUG with kernel mode preemption enabled)
processors:    4
start time:    Wed Jun  9 14:45:08 1999
end time:      Wed Jun  9 14:46:08 1999
command:       sleep 60
```

	tries	reads	trmax	misses	percent misses	sleeps	waitmax seconds	waitsum seconds
bsBuf.bufLock (S)								
0	1400786	0	45745	47030	3.4	0	0.00007	0.15526
1	1415828	0	45367	47538	3.4	0	0.00006	0.15732
2	1399462	0	33076	48507	3.5	0	0.00005	0.15907
3	1398336	0	31753	48867	3.5	0	0.00005	0.15934

ALL	5614412	0	45745	191942	3.4	0	0.00007	0.63099
lock.l_lock (S)								
0	1360769	0	40985	18460	1.4	0	0.00005	0.04041
1	1375384	0	20720	18581	1.4	0	0.00005	0.04124
2	1375122	0	20657	18831	1.4	0	0.00009	0.04198

```

ALL      5483049      0      40985      74688      1.4      0      0.00009  0.16526
...inifaddr_lock  (C)
0          0          0          1          0          0.0      0      0.00000  0.00000
1          1          1          1          0          0.0      0      0.00000  0.00000
2          0          0          1          0          0.0      0      0.00000  0.00000
3          0          0          1          0          0.0      0      0.00000  0.00000

```

```

-----
ALL          1          1          1          0          0.0      0      0.00000  0.00000

total simple_locks = 28100338      percent unknown = 0.0
total rws_locks = 1466      percent reads = 100.0
total complex_locks = 2716146      percent reads = 33.2      percent unknown = 0.0

```

ロック問題は、特定のリソースに対して多くの競合があることです。入出力に関連するロックに競合が存在し、特定のアプリケーションが同一のファイルやディレクトリで競合する多くのプロセスを生成している場合、アプリケーションやデータベース・ストレージの設計を再調整することが必要です。

System V セマフォを使用しているアプリケーションは、単一のセマフォ・セットで非常に多数のセマフォを作成していると、カーネルはセマフォの各セットごとにロックを使用するので、時としてロック競合問題を起こします。この場合、アプリケーションでは、使用するセマフォ・セットの数を増やし、各セマフォ・セットのセマフォの数を減らすようにすれば、性能が改善します。

詳細は、`lockinfo(8)` を参照してください。

2.4.2 sched_stat ユーティリティを使用した CPU 利用率とプロセス統計情報の収集

`sched_stat` ユーティリティは、システムの負荷が各 CPU に正しく分散されているか、どのジョブが各 CPU で十分な実行時間を与えられているか、またはいないか、そしてこれらのジョブに対してキャッシュが有効に使用されているか、というような項目を調べるのに役に立ちます。`sched_stat` は、SMP および NUMA プラットフォームの CPU 利用率とプロセス・スケジューリングを表示します。

`sched_stat` を使用して統計情報を収集するには、以下の手順を実行します。

1. システムの作業負荷を発生させ、それが安定状態になるまで待ちます。
2. `sleep` コマンドを指定し、適当な秒数を `cmd_args` に指定して `sched_stat` を起動します。これにより、`sched_stat` は `sleep` コマンドを実行したときの統計情報を収集します。

たとえば、次のコマンドを実行すると、`sched_stat` は 60 秒間の統計情報を収集し、レポートをプリントします。

```
# /usr/sbin/sched_stat sleep 60
```

コマンド行にオプションを指定すると、指定したオプションの統計情報だけがレポートされます。コマンド行にオプションを指定しないと、`-R` を除くすべてのオプションを指定したものとして扱われます。詳細は、`sched_stat(8)` を参照してください。

2.4.3 `nfsstat` ユーティリティを使用したネットワークと NFS 統計情報の収集

クライアントおよびサーバについて NFS およびリモート・プロシージャ・コール (RPC) の統計情報を表示 (または初期化) するには、次のように入力します。統計情報には、再送を必要としたパケット数 (`retrans`)、応答のトランザクション ID が要求のトランザクション ID と一致しなかった回数 (`badxid`) などがあります。

```
# /usr/ucb/nfsstat
```

以下のような情報が表示されます。

```
Server rpc:
calls      badcalls  nullrecv   badlen     xdrcall
38903      0             0           0           0

Server nfs:
calls      badcalls
38903      0

Server nfs V2:
null      getattr  setattr  root      lookup    readlink  read
5 0%      3345 8%  61 0%      0 0%      5902 15%  250 0%    1497 3%
wrcache  write    create    remove    rename    link      symlink
0 0%      1400 3%  549 1%     1049 2%   352 0%    250 0%    250 0%
mkdir     rmdir    readdir   statfs
171 0%    172 0%    689 1%     1751 4%

Server nfs V3:
null      getattr  setattr  lookup    access    readlink  read
0 0%      1333 3%  1019 2%   5196 13%  238 0%    400 1%    2816 7%
write     create    mkdir     symlink    mknod     remove    rmdir
2560 6%    752 1%    140 0%    400 1%    0 0%      1352 3%   140 0%
rename    link      readdir   readdir+   fsstat    fsinfo    pathconf
200 0%    200 0%    936 2%    0 0%      3504 9%   3 0%      0 0%
commit
21 0%

Client rpc:
calls      badcalls  retrans    badxid     timeout    wait      newcred
27989      1           0           0           1           0           0
badverfs   timers
0           4
```

```
Client nfs:
calls      badcalls  nclget      nclsleep
27988      0           27988       0

Client nfs V2:
null       getattr     setattr     root        lookup      readlink    read
0 0%      3414 12%   61 0%      0 0%      5973 21%   257 0%    1503 5%
wrcache    write       create      remove      rename      link        symlink
0 0%      1400 5%    549 1%     1049 3%    352 1%     250 0%    250 0%
mkdir      rmdir       readdir     statfs
171 0%    171 0%    713 2%     1756 6%

Client nfs V3:
null       getattr     setattr     lookup      access      readlink    read
0 0%      666 2%    9 0%      2598 9%    137 0%     200 0%    1408 5%
write      create      mkdir       symlink     mknod       remove      rmdir
1280 4%    376 1%    70 0%     200 0%     0 0%       676 2%    70 0%
rename     link        readdir     readdir+    fsstat      fsinfo      pathconf
100 0%     100 0%    468 1%     0 0%       1750 6%    1 0%      0 0%
commit
10 0%
```

呼び出しのタイムアウト率 (1 パーセントを超えないようにしてください) は、NFS の統計情報を見る上で最も重要な事項です。呼び出しのタイムアウト率が 1 パーセントを超えると、性能にかなりの悪影響があります。タイムアウトを避けるようにシステムをチューニングする方法は、第 10 章を参照してください。

NFS および RPC の情報を一定間隔 (秒) で表示するには、次のように入力します。

```
# /usr/ucb/nfsstat -s -i number
```

次の例は、10 秒間隔で NFS および RPC 情報を表示します。

```
# /usr/ucb/nfsstat -s -i 10
```

nfsstat で試験的にモニタリングする場合は、試験を開始する前に NFS カウンタを 0 にリセットしてください。NFS カウンタをリセットして 0 にするには、次のように入力します。

```
# /usr/ucb/nfsstat -z
```

コマンドのオプションと出力についての詳細は、nfsstat(8) を参照してください。

2.4.4 tcpdump ユーティリティを使用した情報の収集

tcpdump ユーティリティは、ネットワーク・インタフェース上のパケット・ヘッダをモニタリングし、表示します。リッスンしているインタフェース、パケット転送の方向、あるいはプロトコル・トラフィックのタイプを表示対象として指定できます。

tcpdump を使用すると、特定のネットワーク・サービスに関連するネットワーク・トラフィックをモニタリングでき、パケットの送信元を調べることができます。これにより、要求が受信されたか、または受信に対し肯定応答があったかを調べることができます。また、ネットワーク性能が低い場合には、ネットワーク要求の送信元を調べることができます。

このコマンドを使用するには、次の例のように、カーネルに packetfilter オプションが構成されている必要があります。

```
# pfconfig +p +c tu0
```

netstat -ni コマンドを使用すると、次のように、構成されているネットワーク・インタフェースが表示されます。

```
# netstat -ni
Name  Mtu  Network      Address          Ipkts Ierrs  Opkts Oerrs  Coll
tu0    1500  <Link>        00:00f8:22:f8:05  486404139      010939748  632583736
tu0    1500  16.140.48/24  16.140.48.156    486404139      010939748  632583736
tu0    1500  DLI          none             486404139      010939748  632583736
sl0*   296   <Link>        0 0 0 0 0
lo0    4096  <Link>        1001631086 0 1001631086 0 0
lo0    4096  127/8        127.0.0.1       1001631086 0 1001631086 0 0
```

netstat コマンドの出力を使用して、tcpdump で指定するインタフェースを決定します。たとえば、次のとおりです。

```
# tcpdump -mvi tu0 -Nts1500
tcpdump: listening on tu0
Using kernel BPF filter

k-1.fc77a110 > foo.pmap-v2: 56 call getport prog "nfs" V3 prot UDP port 0 \
(ttl 30, id 20054)
foo.fc77a110 > k-1.pmap-v2: 28 reply getport 2049 (ttl 30, id 36169)
k-1.fd77a110 > foo.pmap-v2: 56 call getport prog "mount" V3 prot UDP port 0 \
(ttl 30, id 20057)
foo.fd77a110 > k-1.pmap-v2: 28 reply getport 1030 (ttl 30, id 36170)
k-1.fe77a110 > foo.mount-v3: 112 call mount "/pns2" (ttl 30, id 20062)
foo.fe77a110 > k-1.mount-v3: 68 reply mount OSF/1 fh 19,17/1.4154.1027680688/4154.\
1027680688 (DF) (ttl 30, id 36171)
k-1.b81097eb > fubar.nfs-v3: 136 call fsinfo OSF/1 fh 19,17/1.4154.1027680688/4154.\
1027680688 (ttl 30, id 20067)
```

-s snaplen オプションを指定すると、省略時の値の 68 バイトではなく、各パケットのデータの snaplen バイト分が表示されます。省略時の値は、IP、ICP、TCP、および UDP に対しては十分ですが、NFS および RPC で十分な結果を必要とする場合には、500 ~ 1500 バイトをお勧めします。

詳細は、tcpdump(8) および packetfilter(7) を参照してください。

2.4.5 netstat コマンドを使用したネットワーク統計情報のモニタリング

ネットワークの統計情報を調べるには、`netstat` コマンドを使用します。検出する問題には、次のようなものがあります。

- `netstat -i command` コマンドで、入力エラー (`Ierrs`)、出力エラー (`Oerrs`)、または衝突 (`Coll`) が大量に表示される場合は、ネットワークに問題があることを示しています。たとえば、ケーブルが正しく接続されていないか、またはイーサネットが飽和状態になっている場合があります (2.4.5.1 項を参照)。
- `netstat -is` コマンドを使用して、ネットワーク・デバイス・ドライバのエラーを調べます (2.4.5.2 項を参照)。
- `netstat -m` コマンドを使用して、ネットワークで使っているメモリが、システムにインストールされているメモリの総計に対して多すぎないか調べます。

`netstat -m` コマンドの出力が、メモリへのいくつかの要求が待たされるか拒否されていることを示している場合は、物理メモリを一時的に使い果たしているか、またはカーネルの `malloc` 空きリストが空になっています (2.4.5.3 項を参照)。
- 各ソケットでネットワーク接続が行われます。システムで割り当てられているソケット数が極端に多い場合は、`netstat -an` コマンドを用いてネットワーク接続の状態を調べてください (2.4.5.4 項を参照)。

インターネット・サーバの場合は、大多数の接続は `TIME_WAIT` 状態になっています。
- `netstat -p ip` コマンドを使用して、チェックサムの間違い、長さの問題、過度のリダイレクション、リソースの問題によるパケットの紛失をチェックします (2.4.5.5 項を参照)。
- `netstat -p tcp` コマンドを使用して、再送、パケットの順序の乱れ、チェックサムの間違いをチェックします (2.4.5.6 項を参照)。
- `netstat -p udp` コマンドを使用すると、不正チェックサムとフル・パケットがチェックされます (2.4.5.6 項を参照)。
- `netstat -rs` コマンドを使用して、ルーティングの統計情報を取得します (2.4.5.7 項を参照)。

- `netstat -s` コマンドを使用して、IP、ICMP、IGMP、TCP、および UDP プロトコル層に関連する統計情報を表示できます (2.4.5.8 項を参照)。

`netstat` で出力される情報のほとんどは、チューニングの可能性を調べるために使用されるのではなく、ネットワークのハードウェアまたはソフトウェアの障害を診断するために使用されます。障害の診断方法についての詳細は、『ネットワーク管理ガイド：接続編』を参照してください。

各種のコマンド・オプションで生成される出力についての詳細は、`netstat(1)` を参照してください。

2.4.5.1 入出力エラーと衝突

ネットワーク上の衝突は、ネットワーク上では普通に発生します。2 つ以上のイーサネット・ステーションが、ネットワーク上で同時に送信を試みると、衝突が発生します。ステーションが、別のステーションで既にネットワークが使われているために、ネットワークを使用できない場合には、そのステーションはネットワークに対するアクセスを少しの時間待ちます。ステーションがネットワークにアクセスできない場合、常に衝突が発生します。大部分のネットワーク・インタフェース・カード (NIC) は最大 15 回の再送を試み、その後出力パケットを廃棄し、`excessive collisions` エラーを発行します。

`netstat -i` コマンドの出力を使って、入力エラー (`Ierrs`)、出力エラー (`Oerrs`)、あるいは衝突 (`Coll`) をチェックします。これらのフィールドの値を、送信パケットの総数と比較します。値が大きいとネットワークに問題がある可能性があります。たとえば、ケーブルが正しく接続されていないか、イーサネットが飽和状態になっています。10% 未満の衝突率では、トラフィックの多いイーサネットでは問題とは言えません。しかし、衝突率が 20% を越えていると、問題があります。`netstat -i` コマンドの例は、次のとおりです。

```
# netstat -i
Name Mtu Network Address Ipkts Ierrs Opkts Oerrs Coll
tu0 1500 Link 00:00:aa:11:0a:c1 0 0 43427 43427 0
tu0 1500 DLI none 0 0 43427 43427 0
tu1 1500 Link bb:00:03:01:6c:4d 963447 138 902543 1118 80006
tu1 1500 DLI none 963447 138 902543 1118 80006
tu1 1500 o-net plume 963447 138 902543 1118 80006
.
.
.
```

2.4.5.2 デバイス・ドライバのエラー

`netstat -is` コマンドを使用して、次のように、ネットワーク・デバイス・ドライバのエラーをチェックします。

```
# netstat -is
tu0 Ethernet counters at Tue Aug  3 13:57:35 2002
    191 seconds since last zeroed
    14624204 bytes received
    4749029 bytes sent
    34784 data blocks received
    11017 data blocks sent
    2197154 multicast bytes received
    17086 multicast blocks received
    1894 multicast bytes sent
    17 multicast blocks sent
    932 blocks sent, initially deferred
    347 blocks sent, single collision
    666 blocks sent, multiple collisions
    0 send failures
    0 collision detect check failure
    1 receive failures, reasons include: Frame too long
    0 unrecognized frame destination
    0 data overruns
    0 system buffer unavailable
    0 user buffer unavailable
```

この例では、システムは 11,017 ブロックを送信したことが示されています。これらのブロックのうち、1,013 (347 + 666) ブロックは衝突を起こし、送信したブロックのおよそ 10% が衝突したことを示しています。10% 未満の衝突率はトラフィックの多いイーサネットでは問題とは言えません。しかし、衝突率が 20% を越えていると、問題があります。

また、以下のフィールドは、0 もしくは小さな 1 桁の数字である必要があります。

- send failures
- receive failures
- data overruns
- system buffer unavailable
- user buffer unavailable

2.4.5.3 メモリの利用率

`netstat -m` は、mbuf クラスタで使われるメモリを含む、ネットワーク関連のメモリ構造体の統計情報を表示します。このコマンドを使用すると、ネットワークがシステムにインストールされているメモリの総容量に対して過度の割合でメモリを使用しているかどうかを調べることができます。

netstat -m コマンドによって、メモリ (mbuf) クラスタに対するいくつかの要求が、遅延されている、または拒否されていることが示されると、システムには物理メモリが一時的に不足していることがわかります。次の例は、128 MB のメモリを備え、mbuf クラスタの圧縮を有効にしていないファイアウォール・サーバの例です。

```
# netstat -m
2521 Kbytes for small data mbufs (peak usage 9462 Kbytes)
78262 Kbytes for mbuf clusters (peak usage 97924 Kbytes)
8730 Kbytes for sockets (peak usage 14120 Kbytes)
9202 Kbytes for protocol control blocks (peak usage 14551
    2 Kbytes for routing table (peak usage 2 Kbytes)
    2 Kbytes for socket names (peak usage 4 Kbytes)
    4 Kbytes for packet headers (peak usage 32 Kbytes)
39773 requests for mbufs denied
    0 calls to protocol drain routines
98727 Kbytes allocated to network
```

この例では、39,773 回メモリ要求が拒否されていることが示されています。この値は 0 である必要があるので、問題があることを示しています。またこの例では、78 MB のメモリが mbuf クラスタに割り当てられており、98 MB のメモリがネットワーク・サブシステムで使用されていることが示されています。

ソケット・サブシステムの属性 sbcompress_threshold を 600 に増やすと、ネットワーク・サブシステムに割り当てられているメモリは即座に 18 MB に下がります。カーネル・ソケット・バッファ・インタフェースで圧縮を行うと、メモリが効率良く使用されるようになるからです。sbcompress_threshold 属性の詳細は、6.2.3.3 項を参照してください。

2.4.5.4 ソケット接続

ソケットごとに、ネットワーク接続が行われます。システムが過度の数ソケットを割り当てている場合には、netstat -an コマンドを使用して、既存のネットワーク接続の状態を調べます。以下の例は、プロトコル制御ブロック・テーブルと各状態の TCP 接続の数を示しています。

```
# netstat -an | grep tcp | awk '{print $6}' | sort | uniq -c
    1 CLOSE_WAIT
   58 ESTABLISHED
   12 FIN_WAIT_1
    8 FIN_WAIT_2
   17 LISTEN
    1 SYN_RCVD
15749 TIME_WAIT
#
```

インターネット・サーバでは、通常、接続の大部分は TIME_WAIT 状態になっています。FIN_WAIT_1 および FIN_WAIT_2 フィールドのエントリの

数が、総接続数 (各フィールドの合計) のうちの大きな割合を示していたら、`keepalive` を有効にします (6.3.2.5 項を参照)。

`SYN_RCVD` フィールド内のエントリの数、総接続数のうちの大きな割合を示していたら、サーバが過負荷か、TCP SYN アタックを受けていることを意味します。

この例で、およそ 16,000 のソケットが使われ、16 MB のメモリが必要になっていることに注意してください。メモリとスワップ領域の構成方法は、6.1.2 項を参照してください。

2.4.5.5 廃棄されたパケットまたは紛失したパケット

`netstat -p ip` コマンドを使用し IP プロトコルについて、不正チェックサム、長さ異常、過度のリダイレクト、およびリソースの問題が原因のパケットの紛失をチェックします。出力の `lost packets due to resource problems` フィールドに 0 ではない数があるかどうかを調べてください。`netstat -p ip` コマンドの例は、次のとおりです。

```
# netstat -p ip
ip:
    259201001 total packets received
    0 bad header checksums
    0 with size smaller than minimum
    0 with data size < data length
    0 with header length < data size
    0 with data length < header length
    25794050 fragments received
    0 fragments dropped (duplicate or out of space)
    802 fragments dropped after timeout
    0 packets forwarded
    67381376 packets not forwardable
        67381376 link-level broadcasts
    0 packets denied access
    0 redirects sent
    0 packets with unknown or unsupported protocol
    170988694 packets consumed here
    160039654 total packets generated here
    0 lost packets due to resource problems
    4964271 total packets reassembled ok
    2678389 output packets fragmented ok
    14229303 output fragments created
    0 packets with special flags set
```

`netstat -id` コマンドを使用して、廃棄された出力パケットのモニタリングを行います。出力の `Drop` 欄に 0 ではない数があるかどうかを調べてください。インタフェースの「`Drop`」欄に 0 ではない数がある場合は、`ifqmaxlen` カーネル変数の値を増やして、パケットの紛失を防ぎます。この属性の詳細は、6.3.2.9 項を参照してください。

以下の例は、tu1 ネットワーク・インタフェースに 4,221 個の廃棄されたパケットがあることを示しています。

```
# netstat -id
Name Mtu Network Address          Ipkts Ierrs Opkts Oerrs Coll Drop
tu0 1500 Link 00:00:f8:06:0a:b1      0      0 98129 98129  0      0
tu0 1500 DLI none                   0      0 98129 98129  0      0
tu1 1500 Link aa:00:04:00:6a:4e 892390 785 814280 68031 93848 4221
tu1 1500 DLI none                   892390 785 814280 68031 93848 4221
tu1 1500 orange flume                   892390 785 814280 68031 93848 4221
.
.
.
```

上のコマンドの出力では、tu0 インタフェースの Opkts および Oerrs フィールドが同じ数値になっており、イーサネット・ケーブルが接続されていないことを示しています。また、tu1 インタフェースの Oerrs フィールドの値は 68,031 で、高い割合であることを示しています。netstat -is コマンドを使用して、詳細なエラー情報を取得してください。

2.4.5.6 再送、順序の不正なパケット、および不正チェックサム

netstat -p tcp コマンドを使用して、TCP プロトコルの再送、順序の不正なパケット、および不正チェックサムをチェックします。netstat -p udp コマンドを使用して、UDP プロトコルの不正チェックサムとソケット・フルを探します。これらのコマンドの出力のいくつかのフィールドの値を、送信パケットあるいは受信パケットの総数と比較することにより、ネットワークの性能問題を調べることができます。

たとえば、再送パケットと重複肯定応答の割合は、2% 以下なら正常です。不正チェックサムの割合は、1% 以下なら正常です。

また、embryonic connections dropped フィールドのエントリが大きな数字を示している場合には、リッスン・キューが小さすぎるか、サーバの性能が低すぎるため、クライアントが要求をキャンセルしていることを意味します。その他の重要な調査対象フィールドは、completely duplicate packets, out-of-order packets, およびdiscarded です。netstat -p tcp コマンドの例は、次のとおりです。

```
# netstat -p tcp
tcp:
    66776579 packets sent
        58018945 data packets (1773864027 bytes)
        54447 data packets (132256902 bytes) retransmitted
        5079202 ack-only packets (3354381 delayed)
        29 URG only packets
        7266 window probe packets
        2322828 window update packets
        1294022 control packets
    40166265 packets received
```

```

29455895 acks (for 1767211650 bytes)
719524 duplicate acks
0 acks for unsent data
19788741 packets (2952573297 bytes) received in-sequence
123726 completely duplicate packets (9224858 bytes)
2181 packets with some dup. data (67344 bytes duped)
472000 out-of-order packets (85613803 bytes)
1478 packets (926739 bytes) of data after window
43 window probes
201331 window update packets
1373 packets received after close
118 discarded for bad checksums
0 discarded for bad header offset fields
0 discarded because packet too short
448388 connection requests
431873 connection accepts
765040 connections established (including accepts)
896693 connections closed (including 14570 drops)
86298 embryonic connections dropped
25467050 segments updated rtt (of 25608120 attempts)
106020 retransmit timeouts
    145 connections dropped by rexmit timeout
6329 persist timeouts
37653 keepalive timeouts
    15536 keepalive probes sent
    16874 connections dropped by keepalive

```

上のコマンド出力では、送信された 58,018,945 個のデータ・パケットのうち、54,447 パケットが再送されています。これは 2% 以下の正常な範囲内にあります。

また、ここでは、29,455,895 の肯定応答のうち、719,524 が重複しており、2% 以下の正常な範囲を少し越えています。

netstat -p udp コマンドの重要なフィールドには、incomplete headers、bad data length fields、bad checksums、および full sockets フィールドがあり、これらは小さな値である必要があります。no port フィールドは存在しないポート（たとえば、rwhod または routed のブロードキャスト・パケット）を宛て先として受信したパケットの数です。このパケットは廃棄されます。このフィールドが大きな値でも正常であり、性能問題は意味しません。netstat -p udp コマンドの実行例は、次のとおりです。

```

# netstat -p udp
udp:
    144965408 packets sent
    217573986 packets received
    0 incomplete headers
    0 bad data length fields
    0 bad checksums
    5359 full sockets
    28001087 for no port (27996512 broadcasts, 0 multicasts)
    0 input packets missed pcb cache

```

この例では、full sockets フィールドに 5,359 の値が表示されています。
これは、UDP ソケット・バッファが小さすぎることを意味しています。

2.4.5.7 ルーティングの統計情報

netstat -rs コマンドを使用すると、ルーティングの統計情報が取得できます。bad routing redirects フィールドの値は小さい必要があります。値が大きいとネットワークに重大な問題があることを意味します。netstat -rs コマンドの実行例は、次のとおりです。

```
# netstat -rs
routing:
    0 bad routing redirects
    0 dynamically created routes
    0 new gateways due to redirects
    1082 destinations found unreachable
    0 uses of a wildcard route
```

2.4.5.8 プロトコルの統計情報

netstat -s コマンドを使用すると、IP、ICMP、IGMP、TCP、および UDP プロトコル層に関連する統計情報が同時に表示できます。netstat -s コマンドの実行例は、次のとおりです。

```
# netstat -s
ip:
    377583120 total packets received
    0 bad header checksums
    7 with size smaller than minimum
    0 with data size < data length
    0 with header length < data size
    0 with data length < header length
    12975385 fragments received
    0 fragments dropped (dup or out of space)
    3997 fragments dropped after timeout
    523667 packets forwarded
    108432573 packets not forwardable
    0 packets denied access
    0 redirects sent
    0 packets with unknown or unsupported protocol
    259208056 packets consumed here
    213176626 total packets generated here
    581 lost packets due to resource problems
    3556589 total packets reassembled ok
    4484231 output packets fragmented ok
    18923658 output fragments created
    0 packets with special flags set

icmp:
    4575 calls to icmp_error
    0 errors not generated because old ip message was too short
    0 errors not generated because old message was icmp
Output histogram:
    echo reply: 586585
    destination unreachable: 4575
    time stamp reply: 1
    0 messages with bad code fields
    0 messages < minimum length
```

```

0 bad checksums
0 messages with bad length
Input histogram:
    echo reply: 612979
    destination unreachable: 147286
    source quench: 10
    echo: 586585
    router advertisement: 91
    time exceeded: 231
    time stamp: 1
    time stamp reply: 1
    address mask request: 7
586586 message responses generated

igmp:
0 messages received
0 messages received with too few bytes
0 messages received with bad checksum
0 membership queries received
0 membership queries received with invalid field(s)
0 membership reports received
0 membership reports received with invalid field(s)
0 membership reports received for groups to which we belong
0 membership reports sent

tcp:
66818923 packets sent
    58058082 data packets (1804507309 bytes)
    54448 data packets (132259102 bytes) retransmitted
    5081656 ack-only packets (3356297 delayed)
    29 URG only packets
    7271 window probe packets
    2323163 window update packets
    1294434 control packets
40195436 packets received
    29477231 acks (for 1797854515 bytes)
    719829 duplicate acks
    0 acks for unsent data
    19803825 packets (2954660057 bytes) received in-sequence
    123763 completely duplicate packets (9225546 bytes)
    2181 packets with some dup. data (67344 bytes duped)
    472188 out-of-order packets (85660891 bytes)
    1479 packets (926739 bytes) of data after window
    43 window probes
    201512 window update packets
    1377 packets received after close
    118 discarded for bad checksums
    0 discarded for bad header offset fields
    0 discarded because packet too short
448558 connection requests
431981 connection accepts
765275 connections established (including accepts)
896982 connections closed (including 14571 drops)
86330 embryonic connections dropped
25482179 segments updated rtt (of 25623298 attempts)
106040 retransmit timeouts
    145 connections dropped by rexmit timeout
6329 persist timeouts
37659 keepalive timeouts
    15537 keepalive probes sent
    16876 connections dropped by keepalive

udp:
145045792 packets sent
217665429 packets received
0 incomplete headers
0 bad data length fields
0 bad checksums

```

```
5359 full sockets
28004209 for no port (27999634 broadcasts, 0 multicasts)
0 input packets missed pcb cache
```

2.4.6 ps axlmp を使用した NFS サーバ側の情報収集

NFS サーバ・システムでは、`nfsd` デーモンによって、クライアントからの入出力要求をサービスする入出力スレッドが複数生成されます。サーバでは標準的である複数の並列要求を処理するためには、十分な数のスレッドが構成されることが必要です。省略時の値である、8つのUDPスレッドと8つのTCPスレッドは、小規模なクライアントに少数のディレクトリをエクスポートするワークステーションでは、十分な値です。負荷の大きなNFSサーバでは、TCPとUDPに分散される、最大128のサーバ・スレッドを構成できます。サーバ上でNFSサーバ・スレッドのモニタリングを行い、NFSの負荷を処理するためにスレッドを増やす必要があるかどうかを判断します。

サーバ・システムのアイドル入出力スレッドを表示するには、次のコマンドを実行します。

```
# /usr/ucb/ps axlmp 0 | grep -v grep | grep -c nfs_udp
# /usr/ucb/ps axlmp 0 | grep -v grep | grep -c nfs_tcp
```

スリープ状態とアイドル状態のUDPスレッドとTCPスレッドの数のカウントが表示されます。スリープ状態のUDPスレッドとTCPスレッドの数が0、もしくは小さな値の場合、スレッドの数を増やしてNFSの性能を改善します。詳細は、5.4.1項または`nfsd(8)`を参照してください。

2.4.7 nfsiod を使用した NFS クライアント側の情報収集

NFS クライアント・システムでは、`nfsiod` デーモンによって、サーバへの非同期入出力要求を処理する入出力スレッドが複数生成されます。入出力スレッドによって、NFSの読み書き両方の性能が改善できます。最適の入出力スレッドの数は、多数の要因(クライアントの書き込み頻度、同時にアクセスするファイルの数、そしてNFSサーバの特性など)によって決まります。小規模なクライアントの場合、省略時の値である7スレッドで十分です。NFSクライアントの負荷が大きい大規模サーバの場合、クライアント・スレッドの数を増やす必要があります。サーバでNFSクライアント・スレッドのモニタリングを行い、NFSの負荷を処理するためにスレッドを増やす必要があるかどうかを判断します。

クライアント・システムのアイドル入出力スレッドを表示するには、次のコマンドを実行します。

```
# ps axlm | grep -v grep | grep -c nfsiod
```

スリープ状態とアイドル状態の I/O スレッドの数が表示されます。スリープ状態とアイドル状態の I/O スレッドの数が 0、もしくは小さな値の場合、スレッドの数を増やして NFS の性能を改善します。詳細は、5.4.2 項または nfsiod(8) を参照してください。

2.4.8 nfswatch コマンドを使用した NFS サーバの受信ネットワーク・トラフィックのモニタリング

nfswatch プログラムは、NFS ファイル・サーバのすべての受信ネットワーク・トラフィックをモニタリングし、それを複数のカテゴリに分類します。各カテゴリ内の受信パケットの数と割合は、持続的に更新される画面に表示されます。nfswatch コマンドは通常オプション無しで実行され、役に立つ情報を生成します。nfswatch プログラムの実行例は、次のとおりです。

```
# /usr/sbin/nfswatch
Interval packets:      628 (network)      626 (to host)      0 (dropped)
Total packets:        6309 (network)    6307 (to host)    0 (dropped)
Monitoring packets from interface ee0

      int  pct  total  unt  pct  total
ND Read      0   0%    0    TCP Packets      0   0%    0
ND Write     0   0%    0    UDP Packets     139  10%   443
NFS Read      2   0%    2    ICMP Packets      1   0%    1
NFS Write     0   0%    0    Routing Control    81   6%   204
NFS Mount     2   0%    3    Address Resolution 109   8%   280
YP/NIS/NIS+   0   0%    0    Reverse Addr Resol  15   1%    43
RPC Authorization 7   1%   12    Ethernet/FDDI Bdcst 406  30%  1152
Other RPC Packets 4   0%   10    Other Packets    1087  80%  3352

      18 NFS Procedures [10 not displayed]      more->
Procedure      int  pct  total  completed  avg(msec)  std dev  max resp
CREATE          0   0%    0
GETATTR         1  50%    1          1      3.90          3.90
GETROOT         0   0%    0
LINK            0   0%    0
LOOKUP          0   0%    0
MKDIR           0   0%    0
NULLPROC        0   0%    0
READ            0   0%    0
```

注意

nfswatch コマンドは、NFS バージョン 2.0 でマウントされたファイル・システムについてのみ、データをモニタリングし表示します。

カーネルには `packetfilter` オプションを構成する必要があります。カーネルの構成後、スーパーユーザが `pfconfig` コマンドを使用して無差別モード操作を有効にすれば、どのユーザも `nfswatch` を起動できるようになります。詳細は、`packetfilter(7)` を参照してください。

2.5 その他の性能モニタリング・ツール

システムの性能を継続的にモニタリングするルーチンを設定することが出来ます。一部のモニタリング・ツールは、重大な問題が発生した場合に警告を行います (たとえば、電子メールで通知する)。正確な性能情報を得るには、オーバヘッドの少ないモニタリング・ツールを選択することが重要です。

表 2-1 には、性能を継続的にモニタリングするためのツールを示しています。

表 2-1: 継続的な性能モニタリングのツール

名前	説明
Performance Visualizer	<p>並列処理システムの重要な構成要素すべての性能を、グラフィカルに表示する。Performance Visualizer を使用すると、クラスタ内のすべてのメンバ・システムの性能をモニタリングできる。</p> <p>Performance Visualizer は複数のシステムの性能を同時にモニタリングするため、並列処理アプリケーションがすべてのシステムに与えている影響を調べたり、システム全体でそのアプリケーションのバランスがとれていることを確認できる。問題の原因が究明できたら、アプリケーションのコードを変更し、Performance Visualizer を使用してその変更による影響を評価できる。Performance Visualizer は Tru64 UNIX のレイヤード・プロダクトであり、オペレーティング・システムとは別のライセンスが必要。</p> <p>Performance Visualizer を使用すると、負荷の高いシステムや、使用率の低いリソース、アクティブ・ユーザ、およびビジー状態のプロセスを見つけることもできる。並列処理システムのすべてのホストを調べるか、個々のホストを調べるかを指定できる。詳細は、Performance Visualizer のドキュメントを参照。</p>

表 2-1: 継続的な性能モニタリングのツール (続き)

名前	説明
monitor	稼働中のシステムの各種の性能データを収集し、その情報を表示するか、バイナリ・ファイルに保存する。monitor ユーティリティは、Tru64 UNIX のフリーウェア CD-ROM で提供。詳細は、 http://www.tru64unix.com-paq.com/demos/osscc-v51a/html/monitor.htm を参照。
top	CPU リソースの使用量の多いプロセスのリストなど、システム状態について継続的に報告する。top コマンドは、Tru64 UNIX のフリーウェア CD-ROM で提供。詳細は、 ftp://eecs.nwu.edu/pub/top を参照。
xload	システムの平均負荷をヒストグラムで表示し、定期的に更新する。詳細は、xload(1X) を参照。
volstat	LSM 制御下のボリューム、ブレックス、サブディスク、およびディスクのアクセス状況を表示する。volstat ユーティリティは、ブート時または統計情報のリセット時以降の LSM オブジェクトの動作レベルを反映した統計情報を報告する。詳細は、9.3 節を参照。
volwatch	LSM のディスク、ボリューム、およびブレックスの障害をモニタリングし、障害が発生した場合はメールを送信する。詳細は、9.3 節を参照。

2.6 プロファイリング情報とデバッグ情報の収集

プロファイリングを使用すると、アプリケーション・コードの中で実行時間の多くを占める部分を検出できます。このツールは、カーネルのプロファイリングやデバッグにも使用できます。性能を改善するには、プログラム内で CPU 時間をたくさん消費している部分のコードの効率化に注目してください。

表 2-2 で、アプリケーションの情報収集で使用するコマンドを説明します。これらのツールの詳細は、『プログラミング・ガイド』と『*Kernel Debugging*』を参照してください。

また、アプリケーション・プロファイラ、プロファイリング、最適化、および性能分析の概要が、`prof_intro(1)` に記載されています。

表 2-2: アプリケーションのプロファイリング・ツールとデバッグ・ツール

名前	用途	説明
atom	アプリケーションのプロファイルを作成する。	パッケージ化されたツールのセット (<code>third</code> , <code>hiprof</code> , および <code>pixie</code>) で、アプリケーションのプロファイリングとデバッグ用の測定を行うために使用される。atom ツールキットには、コマンド・インタフェースと測定ルーチンの集合も付いており、アプリケーション測定用のカスタマイズ・ツールを作成できる。詳細は、『プログラミング・ガイド』と <code>atom(1)</code> を参照。
third	メモリへのアクセスをチェックしてアプリケーションのメモリ・リークを検出する。	atom ツールを使用して、実行可能オブジェクトと共有オブジェクトにコードを追加して、C あるいは C++ で作成されたプログラムの実行時のメモリ・アクセス・チェックとメモリ・リーク検出を行う。Third Degree ツールは、参照先のライブラリを含む、プログラム全体の測定を行う。詳細は、 <code>third(1)</code> を参照。
hiprof	アプリケーションのプロシージャ実行時間のプロファイルを作成する。	atom ベースのプログラム・プロファイリング・ツール。指定したプロシージャでの実行時間を示すフラット・プロファイルと、指定したプロシージャおよびそれから呼び出されるすべてのプロシージャの実行時間を示す階層プロファイルを作成する。 hiprof ツールは、プログラム・カウンタ (PC) のサンプリングではなく、コードの計測によって統計情報を収集する。通常、出力ファイルのフィルタおよびマージ処理とプロファイル・レポートのフォーマットिंगには、 <code>gprof</code> コマンドが使用される。詳細は、 <code>hiprof(1)</code> を参照。

表 2-2: アプリケーションのプロファイリング・ツールとデバッグ・ツール (続き)

名前	用途	説明
pixie	アプリケーション内の基本ブロックのプロファイルを作成する。	<p>プログラム内の各命令の実行回数を示すプロファイルを作成する。この情報は、テーブル形式で報告することもできるが、後で、C コンパイラに <code>-feedback</code>、<code>-om</code> または <code>-cord</code> オプションを指定して、最適化の自動指示に使用することもできる (cc(1) 参照)。</p> <p>pixie プロファイラは実行可能プログラムを読み込み、基本ブロックに分割して、各基本ブロックの実行回数をカウントするコードを付加した同等のプログラムを出力する。</p> <p>また、pixie ユーティリティは、各基本ブロックのアドレスが格納されたファイルも生成する。pixie が生成したプログラムを実行すると、基本ブロックのカウントを含むファイルが生成される。prof および pixstats コマンドは、これらのファイルの分析に使用できる。詳細は、pixie(1) を参照。</p>
prof	プロファイル・データを分析する。	<p>プロファイル・データを分析して、コードの中で CPU 時間が最もかかっている部分と経過時間が長い場所を示す統計情報を作成する (たとえば、ルーチン・レベル、基本ブロック・レベル、命令レベル)。</p> <p>prof コマンドは、プロファイリング・ツールの kprofile、uprofile、または pixie で生成された 1 つまたは複数のデータ・ファイルを入力として使用する。また、prof コマンドは、cc などのコンパイラに <code>-p</code> スイッチを指定してリンクされたプログラムが生成したプロファイル・データ・ファイルにも使用できる。</p> <p>prof が生成した情報を使用すると、ソース・コードを最適化するために、どの部分に注目すればよいか判断できる。詳細は、prof(1) を参照。</p>

表 2-2: アプリケーションのプロファイリング・ツールとデバッグ・ツール (続き)

名前	用途	説明
gprof	プロファイル・データを分析し、アプリケーション内のプロシージャ呼び出し情報とプログラム・カウンタを統計的にサンプリングした情報を表示する。	<p>プロファイル・データを分析し、プロシージャ呼び出し情報の収集とプログラム・カウンタ (PC) の統計的なサンプリングにより、どのルーチンが最も多く呼び出されたか、またルーチンの呼び出し元が何かを判断できるようにする。</p> <p>gprof ツールはルーチンの CPU 使用状況のフラット・プロファイルを生成する。プログラムのグラフィカルな実行プロファイルを作成するために、このツールでは、<code>cc -pg</code> コマンドでコンパイルしたプログラムが生成した PC サンプリング・プロファイルのデータか、または <code>atom -tool hiprof</code> コマンドで変更したプログラムが生成した計測プロファイルのデータを使用する。詳細は、<code>gprof(1)</code> を参照。</p>
uprofile	アプリケーション内のユーザ・コードのプロファイルを作成する。	<p>Alpha チップの性能カウンタを用いてユーザ・コードのプロファイルを作成する。</p> <p>uprofile ツールを使用すると、プログラムの実行可能部分のみをプロファイリングできる。uprofile ツールは、シェアード・ライブラリの情報は収集しない。このツールで収集したデータは、<code>prof</code> コマンドで処理できる。詳細は、『<i>Kernel Debugging</i>』または <code>uprofile(1)</code> を参照。</p>
kprofile	動作中のカーネルのプログラム・カウンタ・プロファイルを作成する。	<p>Alpha チップの性能カウンタを用いて、動作中のカーネルのプロファイリングを行う。このツールで収集した性能データは、<code>prof</code> コマンドで分析する。詳細は、<code>kprofile(1)</code> を参照。</p>

表 2-2: アプリケーションのプロファイリング・ツールとデバッグ・ツール (続き)

名前	用途	説明
Visual Threads	マルチスレッド・アプリケーション内でのボトルネックと性能上の問題を把握する。	このツールでは、マルチスレッド・アプリケーションの分析と改良ができる。Visual Threads を使用すると、ボトルネックと性能の問題を把握し、スレッド関連のロジックの潜在的な問題をデバッグすることができる。Visual Threads は規則に基づいた分析と、統計機能および視覚化の技法を使用する。Visual Threads は、Developers' Toolkit for Tru64 UNIX の一部としてライセンスされている。
dbx	動作中のカーネル、プログラム、とクラッシュ・ダンプのデバッグ、およびカーネル変数の調査と一時的な変更をする。	C, Fortran, Pascal, アセンブラ言語、およびマシン語のソース・レベル・デバッグに使用する。dbx を使用すると、クラッシュ・ダンプの分析、プログラム・オブジェクト内の問題点のトレース (ソース・コード・レベルまたはマシン・コード・レベル)、プログラムの実行の制御、プログラム・ロジックや制御フローのトレース、およびメモリ領域のモニタリングができる。 カーネルのデバッグ、デバッグ情報のないイメージのデバッグ、メモリ内容の調査、マルチスレッドのデバッグ、ユーザ・コードとアプリケーションの分析、カーネル・データ構造体の値およびフォーマットの表示、一部のカーネル変数の値の一時的な変更を行うには、dbx を使用する。詳細は、dbx(8) を参照。

表 2-2: アプリケーションのプロファイリング・ツールとデバッグ・ツール (続き)

名前	用途	説明
kdbx	動作中のカーネルおよびクラッシュ・ダンプのデバッグをする。	<p>動作中のカーネルやクラッシュ・ダンプを調査できる。kdbx デバッガは dbx デバッガのフロントエンドであり、カーネル・コードをデバッグし、カーネル・データを読み取り可能な形式で表示するために使用する。このデバッガは拡張およびカスタマイズが可能で、カーネル・デバッグのニーズに合ったコマンドを作成することができる。</p> <p>拡張機能を使用して、リソースの使用状況 (たとえば、CPU の使用状況) をチェックすることもできる。詳細は、kdbx(8) を参照。</p>
ladebug	カーネルおよびアプリケーションのデバッグ	<p>プログラムとカーネルをデバッグし、プログラムの実行時エラーの検出手助けする。ladebug シンボリック・デバッガは dbx の代替デバッガであり、コマンド行とグラフィカル・ユーザ・インタフェースの両方で、マルチスレッド・プログラムのデバッグができる。詳細は、『<i>Ladebug Debugger Manual</i>』および ladebug(1) を参照。</p>
lsOf	オープン・ファイルを表示する。	<p>実行中のプロセスが現在オープンしているファイルに関する情報を表示する。lsOf は、または Tru64 UNIX のフリーウェア CD-ROM から入手可能。</p>



カーネル・サブシステム属性の表示と変更

オペレーティング・システムには、カーネルの基本機能や拡張機能を実現するさまざまなサブシステムが含まれています。カーネル・サブシステム属性は、サブシステムの動作を制御したり、サブシステムの統計情報を維持するカーネル変数を設定するために使用されます。属性には、ブート時に省略時の値が設定されます。一部の属性の省略時の値がご使用のシステムで適切でない場合は、最適な性能を得るためにこれらの値を変更する必要があります。

カーネル・サブシステム属性の表示や変更を行うには、以下の手順に従ってください。

1. オペレーティング・システムでの属性のサポート状況を調べる (3.1 節)
2. 属性値を表示する (3.2 節)
3. 属性値を変更する (3.3 節)

3.1 オペレーティング・システムでサポートする属性

使用しているオペレーティング・システムで、特定のカーネル・サブシステム属性がサポートされているかどうかを調べるには、以下のいずれかの方法を使用します。

- `sysconfig -q subsystem [attribute]` コマンドを使用します。

属性を指定しない場合、システムは、`sysconfig` コマンドまたは `sysconfigdb` コマンドで変更できるすべてのサブシステム属性を表示します。サブシステムが構成されていない場合、オペレーティング・システムは次のようなメッセージを表示します。

```
framework error: subsystem 'inet' not found
```

属性を指定すると、その属性に関する情報だけが表示されます。たとえば、次のようになります。

```
# sysconfig -q inet tcbhashsize
inet:
tcbhashsize = 32
```

属性がサポートされていないか、または `sysconfig` でアクセスできない場合、オペレーティング・システムは、次のようなメッセージを表示します。

```
inet:
tcbhashsize = unknown attribute
```

- `dbx p (print)` コマンドを使用します。属性にアクセスできない場合、オペレーティング・システムは、その属性が「定義されていないか、またはアクティブでない (is not defined or active)」というメッセージを表示します。たとえば、次のようになります。

```
# dbx -k /vmunix
dbx version 5.1
Type 'help' for help.

(dbx) p tcp_keepalive_default
"tcp_keepalive_default" is not defined or not active
(dbx)
```

詳細は、`sysconfig(8)` および `dbx(1)` を参照してください。

3.2 属性値の表示

カーネル・サブシステム属性の現在の値やその他の説明情報を表示するには、さまざまな方法があります。以下の方法のいずれかを使用します。

- カーネル・チューナ (`dxkerneltuner`) の GUI (グラフィカル・ユーザ・インタフェース) を使って、属性の永久値、現在値 (稼働中)、最小値、および最大値を表示します。この場合、共通デスクトップ環境 (CDE) の「アプリケーション・マネージャ」ウィンドウから GUI にアクセスします。「システム管理」アイコンを選択し、次に「モニタリング/チューニング」アイコンを選択します。その後、属性を表示したいサブシステムを選択します。
- `sysconfig -q subsystem [attribute]` コマンドを使って、指定した属性の現在の (稼働中の) 値を表示します。または、属性を指定しないで、指定したサブシステムのすべての属性を表示します。

```
sysconfig -q subsystem [attribute]
```

たとえば、次のようになります。

```
# sysconfig -q vm ubc_maxpercent
vm:
ubc_maxpercent = 100
```


- `sysconfig -Q subsystem [attribute]` コマンドを使って、指定したサブシステムの属性の最大値と最小値を表示します。または、特定の属性を指定して、その属性の情報だけを表示します。

このコマンドの出力には、その属性に対して実行できる操作に関する情報も含まれています。以下に、`sysconfig` 属性の一部を示します。

- C - サブシステムの初期ロード時に属性が変更されます。つまり、この属性はブート時に永続的な変更が行われます。
- R - システムの稼働中に属性をチューニングできます。つまり、システムが現在使用している値を変更できます。
- Q - この属性の現在値を表示できます (照会できます)。

たとえば、次のようになります。

```
# sysconfig -Q vfs bufcache
vfs:
bufcache -      type=INT op=CQ min_val=0 max_val=50
```

このコマンドの出力は、`bufcache` 属性の最小値が 0 で、最大値が 50 であることを示しています。この出力では、現在の (稼働中の) 値は変更できないことも示しています。

以下に、属性を表示する代わりに、`dbx p (print)` コマンドを使用してカーネル変数の現在値を表示する例を示します。たとえば、次のようになります。

```
# dbx -k /vmunix
dbx version 5.1
Type 'help' for help.

(dbx) p ipport_userreserved
5000
(dbx)
```

詳細は、`dxkerneltuner(8)`、`sysconfig(8)`、および `dbx(1)` を参照してください。

3.3 属性値の変更

属性値を変更する方法はいくつかあります。使用方法は、使用しているオペレーティング・システムのバージョンによって異なります。また、属性の現在の (稼働中の) 値を変更するのか、または属性の永久値を変更するのかによっても異なります。属性値を変更するためには、`root` としてログインしなければなりません。

以下の項で、現在値と永久値を変更する方法を説明します。

3.3.1 現在値

属性によっては、現在の (稼働中の) 値を変更できるものがあります。これにより、属性値を変更することでシステム性能が改善できるかどうかを、システムをブートし直すことなく確認することができます。すべての属性が、システムの稼働中にチューニングできるわけではありません。また、一時的に変更した内容は、システムをリブートすると失われます。ある属性が、システムの稼働中に変更できるかどうかを調べるには、`sysconfig -Q` コマンドを使用してください。

現在の (稼働中の) 値を変更するには、以下の方法のいずれかを使用します。

- カーネル・チューナ (`dxkerneltuner`) の GUI を使って、属性の現在値を変更します。この場合、共通デスクトップ環境 (CDE) の「アプリケーション・マネージャ」ウィンドウから GUI にアクセスします。「システム管理」アイコンを選択し、次に「モニタリング/チューニング」アイコンを選択します。その後、属性を変更したいサブシステムを選択し、「現在の値」フィールドに新しい値を入力します。
- `sysconfig -r` コマンドを使って、属性の現在値を変更します (コマンドでその属性がサポートされている場合)。次のコマンド構文を使用します。

```
sysconfig -r subsystem attribute=value
```

たとえば、次のようになります。

```
# sysconfig -r inet tcp_keepinit=30
tcp_keepinit: reconfigured
```

以下に、属性を変更する代わりに、`dbx assign` コマンドを使用してカーネル変数の現在値を変更する例を示します。ただし、`dbx assign` コマンドによって変更した内容は、システムをリブートすると失われます。次のコマンド構文を使用します。

```
dbx assign attribute=value
```

たとえば、次のようになります。

```
# dbx -k /vmunix
dbx version 5.1
Type 'help' for help.

(dbx) assign ipport_userreserved=60000
60000
(dbx)
```

詳細は、`dxkerneltuner(8)`、`sysconfig(8)`、および `dbx(1)` を参照してください。

3.3.2 永久値

属性の永久値 (ブート時の値) を変更するには、`sysconfigtab` ファイルに、サブシステム名、属性名、および属性の値を記述する必要があります。ただし、`sysconfigtab` ファイルの変更は、手作業では行わないでください。このファイルを変更するには、以下のいずれかの方法を使用します。

- カーネル・チューナ (`dxkerneltuner`) の GUI を使って、属性の永久値を変更します。この場合、共通デスクトップ環境 (CDE) の「アプリケーション・マネージャ」ウィンドウから GUI にアクセスします。「システム管理」アイコンを選択し、次に「モニタリング/チューニング」アイコンを選択します。その後、属性を変更したいサブシステムを選択し、「ブート時の値」フィールドに新しい値を入力します。
- `sysconfigdb` コマンドを使って、属性の永久値を変更します。次の、コマンド構文を使用します。

```
sysconfigdb -a -f stanza_file subsystem
```

stanza_file はサブシステム名と属性およびその値のリストが記述された特別な形式のファイルです。このファイルの内容は、`sysconfigtab` ファイルにマージされます。詳細は、`stanza(4)` を参照してください。

新しい属性値を有効にするには、`sysconfig -r` コマンドを実行する (システムの稼働中にチューニングできる属性の場合) か、システムをリブートしてください。

また、`dbx patch` コマンドを使って変数の値を変更したり、ディスク上の `/vmunix` イメージの値を変更することもできます。

詳細は、`dxkerneltuner(8)`、`sysconfig(8)`、および `sysconfigdb(8)` を参照してください。システム構成ファイルの変更についての詳細は、『システム管理ガイド』を参照してください。



Part 2

アプリケーション・タイプ別のチューニング



Oracle のチューニング

この章では Oracle 8.1.7.x/9i データベースの性能を改善する方法を説明します。さらに、いくつかのモニタリング・ツールと推奨するチューニング方法(以下のものを含む)も説明します。

- Oracle 統計情報のモニタリング (4.1 節)
- `gettimeofday()` 関数の性能改善 (4.2 節)
- IPC 通信プロトコルの選択と有効化 (4.3 節)

注意

本書では、Oracle バージョン 8.1.7、あるいはそれ以降をご使用であることを想定しています。このバージョン要件は重要です。Oracle バージョン 8.1.7、あるいはそれ以降では、ファイル・システム層の UBC (Unified Buffer Cache) を迂回するために、AdvFS の直接入出力機能を使用しているからです。

4.1 Oracle 統計情報のモニタリング

ここではシステム性能情報を収集するために使用するコマンドとユーティリティがいくつかあります。統計情報はさまざまな条件のもとで収集することが重要です。データ・セットを比較すれば、性能問題の診断に役に立ちます。

表 4-1では、Oracle アプリケーションを実行しているシステムをモニタリングするためのツールが示されています。

表 4-1: 性能の低い Oracle アプリケーションを検出するツール

ツール	説明	参照先
collect	特定のオペレーティング・システム・データを記録し、表示する。特定のサブシステムについての重要なシステム性能情報も収集する。	2.3.2 項
lockinfo	カーネル SMP ロックのロック統計情報を収集し、表示する。データの収集には、/dev/lockdev 擬似ドライバを使用する。	2.4.1 項
sched_stat	システム負荷がどのように各 CPU に分散されているか、どのジョブが各 CPU で十分に実行時間を与えられているか、あるいはいないか、そしてこれらのジョブにはキャッシュが有効に働いているのかといった項目を調べるのに有効。	2.4.2 項

詳細は、collect(8)、lockinfo(8) および sched_stat(8) を参照してください。

4.2 gettimeofday() 関数の性能改善

Oracle Server は実行時に多くの関数の時間を計測します。これは INIT.ORA パラメータ timed_statistics が TRUE に設定されていると、特に多くなります。

時間計測関数は オペレーティング・システム・カーネルのシステム・コールになるので、呼び出したプロセスは CPU を放棄することになり、Oracle の性能が低下します。Tru64 UNIX には、プロセスにオペレーティング・システムのリアルタイム・クロックへ直接アクセスを許す機能があります。

この機能を使用すると、負荷の高いシステムでの性能が改善します。負荷の低いシステムでも性能が改善しますが、目立つほどではありません。

注意

この機能は、Oracle バージョン 7.3、またはそれ以降でサポートされています。

この機能を有効にするには、以下のコマンドを実行します。

```
# mknod /dev/timedev c 150
# chmod 644 /dev/timedev
```


クラスタで実行中の場合は、各クラスタ・メンバでこれらのコマンドを実行します。 /dev/timedev 特殊ファイルはシステムのリブートを行っても存続します。

この機能を Oracle で使用するためには、インスタンスを再起動する必要があります。 /dev/timedev ファイルの存在は、インスタンスが起動されるときだけチェックされるからです。 クラスタ内のすべてのインスタンス (つまり、すべてのノード) でこの機能を有効にすることをお勧めします。

詳細は、gettimeofday(2) を参照してください。

4.3 IPC 通信プロトコルの選択と有効化

Oracle では、DLM/IPQ インスタンス間通信で UDP あるいは RDG (Reliable Datagram) のいずれも使用できますが、UDP ではなく、RDG を使用することをお勧めします。

注意

Oracle 8.1.7 は、通信での RDG の使用をサポートしていますが、それを有効にしないで、UDP を継続使用してください。 Oracle 9i でも通信プロトコルとして UDP を使用する必要がある場合があります。

IPC 用に異なるプロトコルを有効または無効にするには、以下のコマンドを使用します。

Oracle 8i または 9i で NUMA サポートを無効にするには、次のコマンドを実行します。

```
# cd $ORACLE_HOME/rdbms/lib
# make -f ins_rdbms.mk numa_off
# make -f ins_rdbms.mkioracle
```

Oracle で NUMA サポートを有効にすることは、現在は、Oracle 8.1.7/OPS と 9.0.1/RAC インストレーションではサポートされていません。 RAC または OPS を使用する場合は、NUMA を無効にする必要があります。

Oracle 9i RAC または 8.1.7 Oracle Parallel Server を使用可能にするには、以下のコマンドを実行します。

- すべての場合

```
# cd $ORACLE_HOME/rdbms/lib
```

- Oracle 8*i* の場合

```
# make -f ins_rdbms.mk ops_on
# make -f ins_rdbms.mk ioracle
```

- Oracle 9*i* の場合

```
# make -f ins_rdbms.mk rac_on
# make -f ins_rdbms.mk ioracle
```

- IPC で UDP プロトコルを使用させる場合 (Oracle 8*i* OPS の省略時の設定)

```
# make -f ins_rdbms.mk ipc_udp
# make -f ins_rdbms.mk ioracle
```

- IPC で RDG (reliable datagram) プロトコルを使用させる場合 (Oracle 9*i* の省略時の設定)

```
# make -f ins_rdbms.mk rac_on
# make -f ins_rdbms.mk ioracle
```

4.4 推奨するチューニング方法

Oracle 8.1.7.x/9*i* データベースの性能に影響を与える多くのサブシステム属性があります。この節では、以下のサブシステムについて、いくつかの属性に対する主要な推奨チューニング方法を説明します。

- 仮想メモリ (4.4.1 項)
- Advanced File System (4.4.2 項)
- 仮想ファイル・システム (4.4.3 項)
- プロセス間通信 (4.4.4 項)
- インターネット (4.4.5 項)
- プロセス (4.4.6 項)
- リアルタイム (4.4.7 項)
- Reliable Datagram (4.4.8 項)
- メモリ・チャネル (4.4.9 項)

注意

カーネル・サブシステム属性によっては、値を変更して、稼働中のシステムに適用できるものがあります。それ以外の属性は、新しい値を有効にするためには、システムをリブートする必要

があります。属性が稼働中にチューニングできるかどうかについては、3.3.1 項を参照してください。

詳細は、`sys_attrs(5)` を参照してください。

4.4.1 仮想メモリ属性の変更

以下の `vm` サブシステム属性をチューニングすることで、Oracle 8.1.7.x/9i データベースの性能を改善できます。

- `new_wire_method` (4.4.1.1 項)
- `rad_gh_regions` (4.4.1.2 項)
- `gh_chunks` (4.4.1.2.2 項)
- `ubc_maxpercent` (4.4.1.3 項)
- `ubc_borrowpercent` (4.4.1.4 項)
- `vm_abcseqstartpercent` (4.4.1.6 項)
- `vm_abcdirtypercent` (4.4.1.7 項)
- `vm_swap_eager` (4.4.1.8 項)

詳細は `sys_attrs_vm(5)` を参照し、カーネル・サブシステムの変更の詳細については第 3 章を参照してください。

4.4.1.1 共用メモリを無効にする

Oracle 8.1.7.x/9i の共用メモリを設定するときには、粒度ヒント共用メモリ (しばしばラージ・ページと呼ばれます) を使用することをお勧めします。ただし、`ipc` サブシステムの `ssm_threshold` 属性で制御される SSM (セグメント化共用メモリ) もサポートされており、それが省略時の設定で有効になっています。

SSM を使用するには、`new_wire_method` を 0 に設定します。たとえば、`ssm_threshold` が省略時の値である 8 MB になっていたら、`new_wire_method` に 0 を設定します。

AIO (非同期入出力) とページ固定メカニズムの間には相互作用があるため、システム時間が長くなることがあります。この問題を解決

するには、`new_wire_method` 属性を 0 にすることをお勧めします。
`new_wire_method` 属性の省略時の値は 1 です。

このチューニング属性を無効にしても、性能への悪影響はありません。ただし、`sys_attrs_vm(5)` では、サポート担当者から、またはパッチ・キットのドキュメントで指示がない限り、省略時の値 (1, オン) を変更しないように指示しています。

`new_wire_method` と `ssm_threshold` の相互作用を避ける手段の 1 つは、データベースで使用する固定メモリとして粒度ヒント・メモリ (`gh_regions` または `rad_gh_regions`) を使用することです。

`gh_chunks` と `rad_gh_regions` の詳細は、4.4.1.2.2 項と 4.4.1.2.1 項を参照してください。

4.4.1.2 共用メモリの割り当て

Oracle 8.1.7.x/9i に共用メモリを割り当てる方法には、2 つのオプションがあります。従来のオプションは、SSM (セグメント化共用メモリ) を使用する方法です。これは、`ipc` サブシステムのチューニング属性 `ssm_threshold` によって制御されます。`ssm_threshold` 属性は省略時の設定で有効になっています。

`gh_chunks` のような粒度ヒント (しばしば、ラージ・ページと呼ばれます) を使用することが、Oracle 8.1.7.x/9i を実行する GS80/160/320 システムで共用メモリの割り当てを行う場合の望ましい方法です (粒度ヒント属性については、`sys_attrs_vm(5)` を参照)。このメモリは他の目的には使用できません。そして、使用する必要がなくなってもシステムに戻せませんし、再利用もできません。データベースが SGA (システム・グローバル領域) として使用するか、または共用メモリの割り当てに `shmget()` または `nshmget()` を使用しているアプリケーションだけが、使用できます。

`vmstat -P` コマンドを使用すると、システムに構成されている粒度ヒント・メモリの容量がわかります。予約されているメモリ容量を調べるための 2 つの属性は、`rad_gh_regions` と `gh_chunks` です。`rad_gh_regions` と `gh_chunks` のいずれを使用するのは、使用しているシステムによって変わります。

- NUMA 対応のシステム (GS80, GS160, GS320) では、`rad_gh_regions` 属性を設定する必要があります。`gh_chunks` 属性は、NUMA 非対応のシステムでも NUMA 対応のシステムでも適

用できますが、`gh_chunks` 属性はシステムでサポートされる最初の RAD (Resource Affinity Domain) にしか効果がありません (詳細は、4.4.1.2.1 項を参照)。

- ES、DS、および GS140 プラットフォームのような NUMA 非対応のシステムで、粒度ヒント・メモリ割り当てを使用する場合には、`gh_chunks` 属性を設定することをお勧めします (詳細は、4.4.1.2.2 項を参照)。

4.4.1.2.1 `rad_gh_regions` 属性を変更する

GS80/GS160/GS320 プラットフォームで `rad_gh_regions` 属性を設定するには、対応する `rad_gh_regions` 属性によって RAD/QBB ごとのメモリ容量を MB 単位で指定します。たとえば、QBB0 に 2 GB を割り当てるには、`rad_gh_regions` 属性を 2048 MB に変更します。

`rad_gh_regions` の設定を決めるには、予定している Oracle の SGA のサイズを、GS システムに構成されている QBBs/RAD の数で割ります。各 `rad_gh_regions[x]` (x は 0 ~ 7 の QBB ID) には、必要な値を MB 単位で指定します。たとえば、8 QBB に 64 GB のメイン・メモリを持つ GS320 プラットフォームの場合、Oracle SGA が 16 GB のサイズであれば、`rad_gh_regions[0]` ~ `rad_gh_regions[7]` の値を、少なくとも 2048 MB に変更します。

`rad_gh_regions[*]` の合計は、少なくとも Oracle SGA のサイズになるように設定することをお勧めします。`rad_gh_regions` に大きな値を割り当てて、Oracle SGA のサイズをシステムをリブートせずに変更したいと考えるかもしれません。`rad_gh_regions` を変更するには、システムのリブートが必要で、`rad_gh_regions` は動的なシステム・チューニング属性ではありません。

共用メモリは、すべての使用可能な RAD のメモリに分散させるため、一般的にはストライプ・モードで割り当てます。シーケンシャル割り当て方式に変更すると、個々の RAD にホットスポットが生じるため、性能に悪影響を与える可能性があります。

`rad_gh_regions` に 0 以外の値を設定する場合は、`ssm_threshold` には 0 を設定して、セグメント化共用メモリ (ssm) を無効にする必要があります。`rad_gh_regions` の省略時の設定は、0 すなわち無効です。

粒度ヒントのメモリ割り当てについての詳細は、4.4.1.2.2 項を参照してください。

4.4.1.2.2 gh_chunks 属性を変更する

gh_chunks 属性には、共用メモリとして使用するために、ブート時に予約された 4MB 単位のメモリの数を指定します。このメモリは他の目的には使用できません。そして、使用しなくなっても、システムに戻せませんし、再利用もできません。

gh_chunks を使用しても最大で 7% の性能改善が得られるだけです。したがって、実装の複雑さを考えると、多くのシステムでの最善のオプションではありません。

ただし、gh_chunks を使用することで効果があるシステムはあります。特に、多くのクライアントがデータベースへ接続したり、切断する Oracle 8.1.7.x/9i 環境には効果があります。この種の環境では、gh_chunks を使用すると大きな性能改善が得られます。gh_chunks には、最小で Oracle SGA サイズを 4 MB で割った値を設定することをお勧めします。gh_chunks の値は、Oracle SGA サイズを 4 MB で割って計算し、この値を 4 MB 単位で表現します。たとえば、Oracle SGA サイズが 16 GB の場合、16 GB を 4 MB で割ると、結果は 4000 MB になります。したがって、gh_chunks に 1000 を設定することになります。gh_chunks に 0 以外の値を設定する場合は、ssm_threshold には 0 を設定して、ssm を無効にする必要があります。gh_chunks の省略時の値は、0 すなわち無効です。

4.4.1.3 UBC で使用する物理メモリの割合を変更する

ubc_maxpercent 属性には、UBC が一時期に使用する物理メモリの最大の割合を指定します。

Oracle 8.1.7.x/9i は AdvFS の直接入出力を使用します。したがって、人為的に UBC (Unified Buffer Cache) を制限する必要はありません。ファイル・システムでの (二重の) キャッシングを避けるため、UBC が一時期に使用する物理メモリを少なくすることをお勧めします。ただし、ubc_maxpercent に小さな値を設定すると、カーネルでの競合が発生し性能に悪影響を与える可能性があります。

UBC が一時期に使用できる物理メモリの割合は、少なくとも 70% に増やすことをお勧めします。35% より小さい値は設定してはいけません。

4.4.1.4 UBC が借りるメモリの割合を変更する

`ubc_borrowpercent` 属性には、メモリの割合を指定します。この割合を超えたメモリは、UBC が仮想メモリ・サブシステムから借りていることになります。UBC が借りているメモリをすべて返すまで、ページングは発生しません。

`ubc_borrowpercent` の省略時の値は 20% です。これは大部分のシステムに対して、適正な値です。しかし、対話型ユーザのいないデータベース・サーバを運用している場合には、バックアップ性能を向上させるために、10% 程度に落とすことができます。

4.4.1.5 UBC が 1 つのファイルに対して使用できるメモリの割合を変更する

`vm_abcseqpercent` 属性には、1 つのファイルのキャッシュ用に使用できる UBC メモリの最大の割合を指定します。UBC がこの制限をチェックする時期を制御する方法は、4.4.1.6 項を参照してください。`vm_abcseqpercent` の省略時の値は 10% です。これは大部分のシステムに対して、適正な値です。しかし、運用しているシステムがデータベース・サーバだけの環境では、バックアップ性能を向上させるために、5% 程度に落とすことができます。

4.4.1.6 UBC しきい値を変更する

`vm_abcseqstartpercent` 属性には、UBC が各ファイル・オブジェクトをキャッシュしている UBC ページの割合のチェックを開始する時期を決定するしきい値 (UBC の現在のサイズに対する割合) を指定します。ファイルのキャッシュされているページの割合が `vm_abcseqpercent` の値を超えていると、UBC はファイルの UBC LRU ページを仮想メモリに返却します。`vm_abcseqpercent` 属性の詳細は、4.4.1.5 項を参照してください。

注意

`vm_abcseqstartpercent` 属性は、`ubc_maxpercent` 属性に対する割合と定義されています。`ubc_maxpercent` は使用可能なメモリに対する割合です。この定義の変更は、`ubc_maxpercent` が省略時の値 (100%) に設定されたままだと問題はありません。しかし、`ubc_maxpercent` の値が下げられている場合には、影響があります。たとえば、`vm_abcseqstartpercent` の値には 25 を設定する必要があります。

4.4.1.7 ダーティ・ページの割合を変更する

`vm_ubcdirtypercent` 属性は、UBC がディスクへの書き込みを開始するためにダーティ (変更済み) になっていなければならないページの割合です。大部分のシステムに対して、省略時の値の 10% は効果があります。しかし、ファイル・システム/UBC が頻繁にアクセスされ、ファイル・システムのページを UBC に保持しておくことが効果的なシステムでは、値を 90% に増加させます。

4.4.1.8 スワップ割り当てモードを変更する

`vm_swap_eager` 属性には、システムのスワップ割り当てモードを指定して使用可能なスワップ領域の使用方法を制御します。スワップ割り当てモードには、即時モード (1) と延期モード (0) があります。どちらのモードにしても、性能への影響はありません。

スワップ領域割り当てモードを以下に示します。

- 即時モード — このモードでは、プロセスが最初に可変メモリを割り当てたときにスワップ領域が予約されます。即時モードは省略時のスワップ領域割り当てモードで、**eager** モードとも呼ばれます。

即時モードでは、システムが不必要に多くのスワップ領域をプロセス用に予約することがあります。ただし、即時モードでは、プロセスがスワップ領域を必要としたときに必ずスワップ領域を利用できます。即時モードは、物理メモリの量を超えてメモリを使用するシステム (ページングが発生するシステム) での使用をお勧めします。

- 延期モード — このモードでは、仮想メモリ・サブシステムが、変更された仮想ページをスワップ領域に書き込む必要が発生した場合だけ、スワップ領域が予約されます。このモードでは、可変メモリ用のスワップ領域の予約が、実際に必要となるまで延期されます。延期モードは、**lazy** モードとも呼ばれます。

延期モードでは、必要となるスワップ領域が即時モードよりも少なく、またシステムの動作が速くなります。これは、延期モードでは、スワップ領域の予約作業が少なくてすむためです。ただし、延期モードではスワップ領域が事前に予約されないため、プロセスでスワップ領域が必要になったときにスワップ領域が利用できないことがあります。この場合、プロセスは非同期に強制終了させられます。延期モードは、大規

模メモリ・システムや、物理メモリの量を超えてメモリを使用しない
(ページングが発生しない) システムでお勧めします。

`vm_swap_eager` 属性を省略時の値の 1 に設定すると、システムは eager スワップ割り当てモードになります。このモードの場合、すべてのプロセス用の可変仮想メモリを合計し、少なくとも 10% をスワップ領域のサイズに追加します。eager スワップ割り当てモードは、物理メモリの量を超えてメモリを使用する信頼性の高いシステムで使います。

`vm_swap_eager` 属性を 0 に設定すると、システムは lazy スワップ割り当てモードになります。このモードの場合、すべてのプロセス用の可変仮想メモリを合計し、物理メモリの半分を減算した値をスワップ領域サイズとします。lazy モードは、物理メモリの量を超えてメモリを使用しないシステムで使います。`vm_swap_eager` が 0 に設定されており、システムでスワップ領域を使い果たした場合は、スワップ領域を割り当てようとするプロセスは強制終了させられます。この状況でプロセスの削除を防ぐ手段はありません。

Oracle 8.1.7.x/9i データベース・サーバ環境では、十分なメモリがあっても `vm_swap_eager` を 1 に設定することをお勧めします。ただし、作業負荷がよく把握されており、スワッピングが行われなだけの十分なメモリがシステムに構成されている場合には、`vm_swap_eager` を 0 に設定できます。eager スワップ割り当てモードの省略時の値は 1 です。

4.4.2 Advanced File System 属性の変更

`advfs` サブシステムの `AdvfsSyncMmapPages` 属性をチューニングすると、Oracle 8.1.7.x/9i データベースの性能が改善できることがあります。

`AdvfsSyncMmapPages` 属性には、変更された (ダーティな) メモリ・マップされたページを `sync()` システム・コールの実行時にディスクにフラッシュするかどうかを制御する値を指定します。値が 1 の場合、ダーティなメモリ・マップされたページはディスクに非同期に書き込まれます。値が 0 の場合、ダーティなメモリ・マップされたページは、`sync` システム・コールの際にディスクに書き込まれません。

パラメータを 0 に設定すると、AdvFS は、メモリ・マップされたファイルのページをフラッシュしなくなります。省略時の値である 1 を設定すると、メモリ・マップされたページは `sync()` システム・コールの実行時にディスクに非同期にフラッシュされます。

`mmap()` を使用してページとファイルをメモリへマップする大部分のアプリケーションでは、`fsync()` コールで独自の同期化を行っているので、AdvFS が同じ動作をもう一度行う必要はありません。この設定は、メモリに留まる必要があるページを AdvFS がフラッシュするのも禁止します。

詳細は、`sys_attrs_advfs(5)` を参照し、カーネル・サブシステム属性の変更についての詳細は、第 3 章 を参照してください。

4.4.3 仮想ファイル・システムの属性の変更

vfs サブシステムの `fifo_do_adaptive` 属性をチューニングすると、Oracle 8.1.7.x/9i データベースの性能が改善できることがあります。

`fifo_do_adaptive` 属性には、パイプヘー括書き込みをし、単一呼び出しでデータを受け取り側に渡すパイプ処理を、有効にする (1) か、無効にする (0) 値を指定します。`fifo_do_adaptive` 属性は、システムがデータベース・サーバの場合、省略時の値をそのまま使用するのが適切でない可能性があるチューニング・パラメータの 1 つです。

省略時の設定の 1 は、FIFO ルーチンの代替アルゴリズムを有効にします。これにより最適なワーキング・セット・サイズが作成され、より少ないデータ転送操作が行われることになりますが、サイズは大きくなります。省略時の設定は、一定のサイズ、またはほぼ一定のサイズのデータを転送するアプリケーションでは、効果的です。省略時の設定は、ランダムなサイズのデータを転送するような類のアプリケーションでは効果がありません。特に、FIFO ルーチンによって最適な転送サイズが決定されるような転送を行うアプリケーションでは、効果がありません。

省略時の設定は、対になるプロセスが同期を取って動作するようなアプリケーションでは効果がありません。たとえば、`procA` が `procB` に転送し、`procB` からの応答を待つような場合です。`fifo_do_adaptive` パラメータを無効にすると、アプリケーションによっては性能が低下し、またアプリケーションによっては性能が改善されます。性能の変化はパイプの使われ方に依存します。Oracle 環境では、このパラメータの値を 0 に設定することをお勧めします。

詳細については、`sys_attrs_vfs(5)` を参照し、カーネル・サブシステム属性の変更についての詳細は、第 3 章 を参照してください。

4.4.4 プロセス間通信属性の変更

以下の ipc サブシステム属性をチューニングすると、Oracle 8.1.7.x/9i データベースの性能が改善できることがあります。

- `ssm_threshold` (4.4.4.1 項)
- `shm_max` (4.4.4.2 項)
- `shm_min` (4.4.4.3 項)
- `shm_mni` (4.4.4.4 項)
- `shm_seg` (4.4.4.5 項)

詳細は、`sys_attrs_ipc(5)` を参照し、カーネル・サブシステム属性の変更については、第 3 章 を参照してください。

4.4.4.1 System V の共用領域を変更する

`ssm_threshold` 属性には、共用ページ・テーブルとして使用する System V の共用領域の最小のサイズを、バイト単位で指定します。`ssm_threshold` 属性は、使用する SSM (セグメント化共用メモリ) の実装タイプを制御します。

省略時の値は 8 MB です。ラージ・ページを使用しているときは、`ssm_threshold` を無効にします。すなわち、`rad_gh_regions` または `gh_chunks` を設定している場合です (4.4.1.2 項を参照)。`rad_gh_regions` または `gh_chunks` を使用していない限り、GS シリーズ以外のプラットフォームでは、`ssm_threshold` は省略時の値のままにしておくことをお勧めします。`rad_gh_regions` または `gh_chunks` を使用している場合は、`ssm_threshold` は 0 (無効) に設定します。

4.4.4.2 System V 共用メモリ領域の最大サイズを変更する

`shm_max` 属性には、単一の System V 共用メモリ領域の最大サイズをバイト単位で指定します。Oracle は、SGA が `shm_max` に設定されている値より大きい場合、複数の共用メモリ領域を連結します。単一の共用メモリ領域 (SSM) のサイズは 2 GB より大きくできますが、同一システム上で共用領域を使用するアプリケーションが複数ある場合、2GB より大きな共用メモリ・セグメントでは、問題を起こす可能性があります。

互換性の問題を起こさないために、個々の共用メモリ・セグメントの最大サイズを 2 GB にすることをお勧めします。推奨値は、2 GB - 8 MB =

2,139,095,040 バイトです。システムで実行するアプリケーションが Oracle だけの場合、shm_max のサイズは、4 GB - 16 MB = 4,278,190,080 バイトまで大きくできます。省略時の値は 4,194,304 バイト (512 ページ) です。

4.4.4.3 System V 共用メモリ領域の最小サイズを変更する

shm_min 属性には、単一の System V 共用メモリ領域の最小サイズをバイト単位で指定します。

推奨値は 1 つの領域で、省略時の値は 1 つの領域です。

4.4.4.4 一時期に使用できる共用メモリ領域の数を変更する

shm_mni 属性には、システムで一時期に使用できる共用メモリ領域の最大数を指定します。

推奨値は 256 領域 (省略時の値は 100 領域) です。システムは、数をそれに最も近い 2 の累乗に切り上げた値にしますので、省略時の値は実際には 128 領域になります。

4.4.4.5 一時期に接続できる共用メモリ領域の数を変更する

shm_seg 属性には、一時期に 1 つのプロセスに接続することができる System V 共用メモリ領域の最大数を指定します。

推奨値は、128 領域です。

4.4.5 インターネット属性の変更

以下の inet サブシステム属性をチューニングすることで、Oracle 8.1.7.x/9i データベースの性能が改善できることがあります。

- udp_sendspace (4.4.5.1 項)
- udp_recvspace (4.4.5.2 項)
- ipport_userreserved (4.4.5.3 項)

詳細は、sys_attrs_inet(5) を参照し、カーネル・サブシステムの変更については、第 3 章を参照してください。

さらに、インターネット・サーバのチューニング方法の詳細は、第 6 章を参照してください。ギガビット・イーサネットの性能に特有の推奨チューニングについては、次の URL を参照してください。

4.4.5.1 UDP ソケット用送信バッファ・サイズを変更する

`udp_sendspace` 属性には、UDP ソケット用の省略時の設定の送信バッファ・サイズをバイト単位で指定します。アプリケーションでギガビット・イーサネットを使用している場合、またはネットワークの負荷が大きい場合には、`udp_sendspace` 属性には省略時の値よりも大きな値を設定します。

推奨値は 65536 バイト以上です。ユーザ・セッションや照会の大きさと数に応じて増やします。

4.4.5.2 UDP ソケット用受信バッファ・サイズを変更する

`udp_recvspace` 属性には、UDP ソケット用の省略時の設定の受信バッファ・サイズをバイト単位で指定します。アプリケーションでギガビット・イーサネットを使用している場合、またはネットワークの負荷が大きい場合には、`udp_recvspace` 属性には省略時の値よりも大きな値を設定します。

推奨値は 65536 バイト以上です。ユーザ・セッションや照会の大きさと数に応じて増やします。

4.4.5.3 システムが同時に送信接続を確立できる回数を変更する

`ipport_userreserved` 属性には、システムが他システムへ同時に送信接続を確立できる回数を指定します。

送信ポートの数は、`ipport_userreserved` 属性の値から、`ipport_userreserved_min` 属性の値を引いた数です。省略時の値は 5000 バイトです。したがって、送信ポートの数の省略時の値は 3976 です。大規模 Oracle インストレーションでの推奨値は、最大値の 65535 (バイト) です。

4.4.6 プロセス属性の変更

以下の `proc` サブシステム属性をチューニングすることで、Oracle 8.1.7.x/9i データベースの性能が改善できることがあります。

- `per_proc_stack_size` (4.4.6.1 項)
- `max_per_proc_stack_size` (4.4.6.2 項)
- `per_proc_data_size` (4.4.6.3 項)

- `max_per_proc_data_size` (4.4.6.4 項)
- `per_proc_address_space` (4.4.6.5 項)
- `max_per_proc_address_space` (4.4.6.6 項)
- `max_proc_per_user` (4.4.6.7 項)
- `max_threads_per_user` (4.4.6.8 項)
- `maxusers` (4.4.6.9 項)

詳細は、`sys_attrs_proc(5)` を参照し、カーネル・サブシステム属性の変更については、第 3 章を参照してください。

4.4.6.1 プロセスごとのスタック・サイズを変更する

`per_proc_stack_size` 属性には、プロセスごとのスタック・サイズをバイト単位で指定します。省略時の値の 8 MB は、大部分の Oracle 環境には十分な大きさです。しかし、非常に大規模なインストレーションおよびデータ・ウェアハウスのような環境では、値を増やす必要があります。

推奨値は、33,554,432 (32 MB) です。

4.4.6.2 ユーザ・プロセスのスタック・サイズの最大サイズを変更する

`max_per_proc_stack_size` 属性には、ユーザ・プロセス・スタックの最大サイズをバイト単位で指定します。省略時の値の 32 MB は、大部分の Oracle 環境には十分な大きさです。しかし、非常に大規模なインストレーションおよびデータ・ウェアハウスのような環境では、値を増やす必要があります。

推奨値は、536,870,912 (512 MB) です。Oracle 環境に応じて、最大スタック・サイズは、最大値の 896 MB、あるいはそれ以下にします。この制限は、Oracle が性能上の理由で、PGA と SGA を 0x38000000 と 0x58000000 に固定しているためです。このパラメータで 896 MB より大きな値を使用すると、Oracle は固定 PGA と固定 SGA を破壊します。

4.4.6.3 プロセスごとのデータ・サイズを変更する

`per_proc_data_size` 属性には、プロセスごとのデータ・サイズをバイト単位で指定します。この値にはシステムにインストールされている物理メモリ量を設定します。この値には、実際に使用可能なメモリより大きな値が指定できます。しかし、そうすると、単一プロセスがシステムのメイン・メ

メモリを越える大きさになる可能性があり、過度のスワッピングやページングが発生します (12.5 節を参照)。

`per_proc_data_size` 属性は使用可能なメモリの範囲にすることをお勧めします。 `per_proc_data_size` を、使用可能な物理メモリに、構成されているスワップ領域を加えた値より大きくすることは、決してしないでください。推奨値は、インストールされている物理メモリと同じ大きさです。ただし、最大値は 4,398,046,511,104 バイトです。

4.4.6.4 プロセスごとのデータ・サイズの最大サイズを変更する

`max_per_proc_data_size` 属性には、プロセスごとのデータ・セグメントの最大サイズをバイト単位で指定します。この値にはシステムにインストールされている物理メモリ量を設定します。この値には、実際に使用可能なメモリより大きな値が指定できます。しかし、そうすると、単一プロセスがシステムのメイン・メモリを越える大きさになる可能性があり、過度のスワッピングやページングが発生します (12.5 節を参照)。

`max_per_proc_data_size` 属性は使用可能なメモリの範囲にすることをお勧めします。 `max_per_proc_data_size` を、使用可能な物理メモリに、構成されているスワップ領域を加えた値より大きくすることは、決してしないでください。推奨値は、インストールされている物理メモリと同じ大きさです。ただし、最大値は 4,398,046,511,104 バイトです。

4.4.6.5 プロセスごとのアドレス・サイズを変更する

`per_proc_address_space` 属性には、プロセスごとのアドレス・サイズをバイト単位で指定します。この値にはシステムにインストールされている物理メモリ量の値を設定します。この値には、実際に使用可能なメモリより大きな値が指定できます。しかし、そうすると、単一プロセスがシステムのメイン・メモリを越える大きさになる可能性があり、過度のスワッピングやページングが発生します (12.5 節を参照)。

`per_proc_address_space` 属性は使用可能なメモリの範囲にすることをお勧めします。 `per_proc_address_space` を、使用可能な物理メモリに、構成されているスワップ領域を加えた値より大きくすることは、決してしないでください。推奨値は、インストールされている物理メモリ量と同じ大きさです。ただし、最大値は 4,398,046,511,104 バイトです。

4.4.6.6 プロセスごとのアドレス・サイズの最大サイズを変更する

`max_per_proc_address_space` 属性には、ユーザ・プロセスのアドレス・スペースの最大値をバイト単位で指定します。この値にはシステムにインストールされている物理メモリ量を設定します。この値には、実際に使用可能なメモリより大きな値が指定できます。しかし、そうすると、単一プロセスがシステムのメイン・メモリを越える大きさになる可能性があり、過度のスワッピングやページングが発生します (12.5 節を参照)。

`max_per_proc_address_space` 属性は使用可能なメモリの範囲にすることをお勧めします。`max_per_proc_address_space` を、使用可能な物理メモリに、構成されているスワップ領域を加えた値より大きくすることは、決してしないでください。推奨値は、インストールされている物理メモリ量です。ただし、最大値は 4,398,046,511,104 バイトです。

4.4.6.7 プロセスの最大数を変更する

`max_proc_per_user` 属性には、ユーザが作成できるプロセス (タスク) の最大数を指定します (スーパーユーザには適用されません)。`max_proc_per_user` 属性による制限を解除するには、この属性値として 0 を設定します。

推奨値は 1024 (プロセス) です。アプリケーションがユーザごとに 1024 タスク以上を必要としているなら、それに応じて増やしてください。

4.4.6.8 スレッドの最大数を変更する

`max_threads_per_user` 属性には、ユーザが作成できるスレッドの上限を指定します (スーパーユーザには適用されません)。`max_threads_per_user` 属性による制限を解除するには、この属性値として 0 を設定します。

推奨値は 4096 スレッドです。アプリケーションがユーザごとに 1024 タスク以上を必要としているなら、それに応じて増やしてください。

4.4.6.9 システム・テーブルに割り当てる領域を変更する

`maxusers` 属性には、システム・リソースに負担を与えることなく、システムがサポートできる同時ユーザ数を指定します。システムのアルゴリズムは、最大ユーザ数を使って各種のシステム・データ構造体のサイズを決定したり、システム・プロセス・テーブルのようなシステム・テーブルに割り当てる値を決定します。

ES40 クラス以上のシステムに対しては、この属性値に 8192 (ユーザ)、または最大値の 16,384 (ユーザ) までの値を設定することをお勧めします。省略時の値はシステム依存です。

4.4.7 リアルタイム属性の変更

rt サブシステムの aio_task_max_num 属性をチューニングすることで、Oracle 8.1.7.x/9i データベースの性能が改善できることがあります。

aio_task_max_num 属性には、指定した数のタスクで利用できる固定物理メモリの量を制限することで、物理メモリに固定化される AIO 要求の数を間接的に制御する制限を指定します。固定物理メモリの 1 ページが aio_task_max_num で指定される数のタスクで使用できます。

推奨値は、DBWR I/O 操作と DB_FILE_MULTIBLOCK_READ_COUNT パラメータの値のいずれよりも大きい必要があります。DBWR I/O 操作の最大数は、_DB_WRITER_MAX_WRITES 初期化パラメータに指定しない限り、省略時の値の 8192 です。省略時の値は、102 (102 タスクに対して、1 ページの固定メモリ) です。

このチューニング属性値の簡単な計算方法は、次のとおりです。

$$(\text{DB_WRITER_MAX_WRITES (省略時の値は 8192)} \times \text{DB_WRITER_PROCESSES}) + (\text{PARALLEL_MAX_SERVERS} \times \text{DB_FILE_MULTIBLOCK_READ_COUNT}) + 10$$

詳細は、sys_attrs_rt(5) を参照し、カーネル・サブシステム属性の変更に 대해서는、第 3 章を参照してください。

4.4.8 Reliable Datagram 属性の変更

以下の rdg サブシステム属性をチューニングすることで、Oracle 8.1.7.x/9i データベースの性能が改善できることがあります。

- max_objs (4.4.8.1 項)
- msg_size (4.4.8.2 項)
- max_async_req (4.4.8.3 項)
- max_sessions (4.4.8.4 項)
- rdg_max_auto_msg_wires (4.4.8.5 項)

詳細は、`sys_attrs_rdg(5)` を参照し、カーネル・サブシステム属性の変更については、第 3 章を参照してください。

4.4.8.1 RDG 内のオブジェクトの最大数を変更する

`max_objs` 属性には、RDG の終点とバッファ・リストにあるオブジェクトの最大数を指定します。

推奨値は、最小でノード当たりの Oracle プロセスの数の 5 倍で、最大では 10240、または Oracle プロセスの数に 70 を掛けた値です。

4.4.8.2 RDG メッセージの最大サイズを変更する

`msg_size` 属性には、RDG メッセージの最大サイズをバイト単位で指定します。

推奨値は、データベースの `DB_BLOCK_SIZE` パラメータの最大値以上です。Oracle では、Oracle9i が各テーブル領域で異なるブロック・サイズをサポートしているので、値を 32768 にすることを推奨しています。

4.4.8.3 RDG 内のメッセージの最大数を変更する

`max_async_req` 属性には、RDG の送信キューと受信キュー内に保持される非同期メッセージの最大数を指定します。

推奨値は最小で 100 です。256 の値を指定すれば、性能が改善されます。

4.4.8.4 RDG テーブル内のセッションの最大数を変更する

`max_async_req` 属性には、1 つの RDG コンテキスト・テーブル内のセッションの最大数を指定します。

推奨値は、最小で Oracle のプロセス数に 2 を加えたものです。

4.4.8.5 メッセージ・パケット用に固定されるページの最大数を変更する

`max_async_req` 属性には、メッセージ・パケット用にメモリに自動的に固定されるページの最大数を指定します。

`max_async_req` 属性の値は 0 にすることをお勧めします。

4.4.9 メモリ・チャネル属性の変更

次の `rm` サブシステムの `rm_check_for_ipl` 属性をチューニングすることで、Oracle 8.1.7.x/9i データベースの性能が改善できることがあります。

`rm_check_for_ipl` 属性には、CPU のプロセッサ優先順位レベル (`spl`) をトレース・バッファに格納する時期を示すビットマスクを指定します。

この属性の値は、省略時の値の 63 にすることをお勧めします。

詳細は、`sys_attrs_rm(5)` を参照し、カーネル・サブシステム属性の変更に
ついては、第 3 章を参照してください。



ネットワーク・ファイル・システムのチューニング

ネットワーク・ファイル・システム (NFS) を使用すると、ネットワークを通じて透過的にファイルにアクセスできます。NFS は、小規模で単純なネットワークから、大規模で複雑なネットワークまで、各種のネットワーク・トポロジをサポートします。NFS は、ユニファイド・バッファ・キャッシュ (UBC) を、仮想メモリ・サブシステムおよびローカル・ファイル・サブシステムと共有します。

NFS は、ネットワークに過度の負荷をかけることがあります。NFS の性能が良くない場合は大半が、ネットワーク・インフラストラクチャの問題です。NFS クライアントの再送メッセージの数が多くないか、ネットワークの入出力エラーがないか、負荷に耐えられないルータがないかを調べてください。ネットワーク上でパケットの紛失があると、NFS の性能が大幅に低下します。パケットの紛失は、サーバの輻輳、送信中のパケットの破損 (電氣的接続の問題や、ノイズの多い環境、ノイズの多いイーサネット・インタフェースが原因)、転送処理の放棄が早過ぎるルータが原因で発生します。

NFS の性能を評価する場合、NFS ファイル上でファイル・ロック・メカニズムが使用されているときは NFS の性能が良くないことを考慮してください。ロックを行うとファイルがクライアント側にキャッシュされなくなるからです。

NFS のサービスだけを行っているシステムで性能を改善する方法は、汎用タイムシェアリング用に使用しているシステムのチューニングとは異なります。というのは、NFS サーバは少数の小規模なユーザ・レベル・プログラムだけを実行しており、システム・リソースはほとんど使用しないからです。ページングやスワッピングも最低限しか発生しないので、メモリ・リソースは、ファイル・システムのデータのキャッシングに重点を置いて割り当てる必要があります。

NFS 要求の処理で CPU 時間と実時間の大部分が消費されるため、NFS ではファイル・システムのチューニングが重要です。理想的には、UBC ヒッ

ト率が高くなければなりません。UBC のヒット率を高くするには、メモリを増やしたり、他のファイル・システムのキャッシュのサイズを小さくする必要があります (11.1.3 項を参照)。一般的には、ファイル・システムのチューニングを行うと、入出力多用型のユーザ・アプリケーションの性能が改善されます。また、ファイル・データを保存するには、vnode が必要です。AdvFS を使用している場合は、ファイル・データを保存するために、アクセス構造体も必要です。ファイル・システムについての詳細は、第 11 章を参照してください。

この章では、NFS の性能を改善する方法を説明します。以下のトピックを含め、各種の構成ガイドラインを紹介し、いくつかのモニタリング・ツールについて説明します。

- NFS 統計情報のモニタリング (5.1 節)
- 性能低下の検出 (5.2 節)
- 性能上の利点と欠点 (5.3 節)
- NFS の構成 (5.4 節)
- NFS の再送 (5.5 節)
- NFS サーバのチューニング (5.6 節)
- NFS クライアントのチューニング (5.7 節)

5.1 NFS 統計情報のモニタリング

この節では、NFS 性能情報を収集するのに使用するユーティリティの参照先を紹介します。統計情報はさまざまな条件のもとで収集することが重要です。データ・セットを比較すると、性能問題を診断するのに役に立ちます。

表 5-1 は、NFS 性能の低下を検出するのに使用するツールを説明しています。

表 5-1: NFS の性能低下を検出するツール

ツール	説明	参照先
nfsstat	ネットワーク・ファイル・システムの統計情報の表示	2.4.3 項
tcpdump	ネットワーク・インタフェース上のパケット・ヘッダのモニタリングと表示	2.4.4 項
netstat	ネットワーク統計情報の表示	2.4.5 項

5-2 ネットワーク・ファイル・システムのチューニング

表 5-1: NFS の性能低下を検出するツール(続き)

ツール	説明	参照先
ps axlm	システム上のアイドル入出力スレッドの表示	サーバ側は 2.4.6 項，クライアント側は 2.4.7 項
nfswatch	NFS ファイル・サーバの受信ネットワーク・トラフィックのモニタリングとそのカテゴリ分類	2.4.8 項
dbx print nfs_sv_active_hist	アクティブな NFS サーバ・スレッドのヒストグラムの表示	3.1 節
dbx print nchstats	namei キャッシュのヒット率の調査	2.6 節
dbx print bio_stats	メタデータ・バッファ・キャッシュのヒット率の調査	3.1 節

5.2 NFS 性能低下の検出

表 5-2は，NFS の性能低下の問題とその解決策をいくつか示しています。

表 5-2: 発生しうる NFS の問題と解決策

問題	解決策
NFS サーバのスレッドがビジー。	サーバを再構成して，実行するスレッドの数を多くする。 2.4.6 項を参照。
ファイル・システムのキャッシュに重点を置いてメモリ・リソースが割り当てられていない。	UBC に割り当てるメモリの量を増やす。 11.1.3 項を参照。 AdvFS を使用している場合は，AdvFS アクセス構造体用に予約するメモリを増やす。 11.1.5 項を参照。
システム・リソース割り当てが十分ではない。	maxusers 属性の値に 1 秒あたりに発生するサーバ NFS 動作の数を設定する。 8.1 節を参照。
UFS メタデータ・バッファ・キャッシュのヒット率が低い。	メタデータ・バッファ・キャッシュのサイズを増やす。 11.1.4 項を参照。 namei キャッシュのサイズを増やす。 11.1.2 項を参照。
CPU アイドル時間が少ない。	AdvFS の代わりに，UFS を使う。 11.3 節を参照。

5.3 性能上の利点と欠点

表 5-3 は、NFS の構成ガイドラインとそれぞれの性能上の利点と欠点を示します。

表 5-3: NFS のチューニング・ガイドライン

利点	ガイドライン	欠点
入出力のブロック動作が効率的になる	NFS サーバ上に適切な数のスレッドを構成する (5.4.1 項)	なし
入出力のブロック動作が効率的になる	クライアント・システム上に適切な数のスレッドを構成する (2.4.7 項)	なし
遅いネットワーク、または輻輳したネットワークの性能が改善される	クライアント・システムのネットワーク・タイムアウトを減らす (5.4.3 項)	理論上性能は低下
読み取り専用のファイル・システムのネットワーク性能を改善し、クライアントが変更を迅速に検出できるようになる	クライアント・システム上のキャッシュのタイムアウト限界値を変更する (5.4.3 項)	サーバへのネットワーク・トラフィックが増加する

以降の節で、これらのガイドラインを詳細に説明します。

5.4 NFS の構成

この節では、NFS (ネットワーク・ファイル・システム) 構成の特定の領域について説明します。ネットワーク構成の詳細は、『ネットワーク管理ガイド：接続編』を参照してください。

5.4.1 サーバ・スレッドの構成

`nfsd` デーモンは NFS サーバ上で実行され、クライアント・システムからの NFS 要求を処理します。このデーモンは、クライアント・システムからの NFS 要求を処理する多数のサーバ・スレッドを生成します。サーバとして動作するには、少なくとも 1 つのサーバ・スレッドがマシン上で実行されていなければなりません。スレッドの数により、並列動作の数が決まります。この数は、8 の倍数でなければなりません。

頻繁に使用される NFS サーバの性能を改善するには、16 個または 32 個のスレッドを構成します。この構成では、入出力動作のブロックが最も効率的になります。UDP スレッドと TCP スレッド用に合計で 16～128 のスレッドを構成します。

次のコマンドを実行して、UDP スレッドと TCP スレッドの数をモニタリングします。

```
# ps axlm | grep -v grep | grep -c nfs_udp
# ps axlm | grep -v grep | grep -c nfs_tcp
```

上記のコマンドを実行すると、スリープ状態のスレッドとアイドル状態のスレッドの数が表示されます。この数が繰り返し 0 になるようなら、`nfsd` スレッドを追加する必要があります。詳細は、2.4.7 項または `nfsd(8)` を参照してください。

5.4.2 クライアント・スレッドの構成

クライアント・システムは `nfsiod` デーモンを使用して、バッファ・キャッシュの先読みや書き込み動作の遅延などの、非同期入出力動作を処理します。`nfsiod` デーモンは、複数の入出力スレッドを生成して、サーバへの非同期入出力要求を処理します。入出力スレッドは、NFS の読み取りと書き込みの両方の性能を改善します。

最適な入出力スレッド数は、多数の要因 (クライアントの書き込み頻度、同時にアクセスされるファイルの数、NFS サーバの動作など) によって決まります。スレッドの数は、8 の倍数から 1 を引いた数でなければなりません (たとえば、7、15、31 などが最適です)。

NFS サーバは、書き込みを完全な UFS クラスタとして集積してから、入出力を開始します。スレッド数に 1 を加えた値は、クライアントが同時に保持できる未処理の書き込みの数です。スレッドがちょうど 7 個または 15 個ある場合は、入出力動作のブロックが最も効率的になります。書き込みの集積が有効でクライアントにスレッドがない場合は、性能が低下することがあります。書き込みの集積を無効にするには、`dbx patch` コマンドを使用して、`nfs_write_gather` カーネル変数に 0 を設定します。`dbx` コマンドについての詳細は、3.2 節を参照してください。

クライアント上のアイドル状態の入出力スレッドを表示するには、次のコマンドを実行します。

```
# ps axlm | grep -v grep | grep -c nfsiod
```

スリープ状態のスレッドがほとんどない場合、スレッドの数を増やすと NFS の性能が改善される可能性があります。詳細は、2.4.6 項または `nfsiod(8)` を参照してください。

5.4.3 キャッシュ・タイムアウトの限界値の変更

読み取り専用のファイル・システムと低速のネットワーク・リンクでは、NFS クライアント・システム上のキャッシュ・タイムアウトの限界値を変更すると、性能を改善できる可能性があります。このタイムアウトは、他のホストが変更したファイルまたはディレクトリの変更内容が反映される早さに影響します。サーバ・システムなどの他のホスト上のユーザとファイルを共有していない場合は、タイムアウト値を大きくすると性能が少し改善され、生成するネットワーク・トラフィックの量が少なくなります。

詳細は、`mount(8)` と、`acregmin`、`acregmax`、`acdirmin`、`acdirmax`、`actimeo` オプションに関する説明を参照してください。

5.5 NFS の再送

過度の再送があると、クライアントは要求を再送する前にサーバからの応答を待つ必要があるので、性能が低下する原因になります。過度の再送の原因は、以下のとおりです。

- 過負荷なサーバでバッファが不足してパケットが紛失する
- イーサネット・トランシーバが不適切でビジー状態のときにパケットが紛失する
- ノイズの多い同軸ケーブルで発生するような物理的なネットワーク・エラー

クライアント・コンピュータ上の NFS 再送率を測定するには、`nfsstat -c` コマンドを使用します。それにより、ネットワークの再送率がわかります。詳細は、`nfsstat(8)` を参照してください。

メディアの負荷が低いか中間程度の場合、クライアント要求に対する NFS の平均応答時間は 15 ミリ秒程度です。大部分のクライアントは、およそ 1 秒後に要求を再送します。性能が 10% 悪くなっても良ければ、応答時間を 1.5 ミリ秒増加できます。こうすると、NFS の再送率は 0.15% になって、許容範囲に収まります。計算は、次のとおりです。

```
.0015 秒/要求
----- = 0.0015 再送/要求
```

1.0 秒/再送

最悪の場合の NFS 要求 (イーサネット上の 8 KB の読み取り, または書き込み) には 7 パケット (1 つの要求と 6 つの断片化した応答) が必要なので, ネットワークのエラー率は, 0.02% 以下です。計算は, 次のとおりです。

$$\frac{0.15 \%}{7} = 0.02 \%$$

ネットワークのエラー率を測定するには, `netstat -i` コマンドを使用します。この率が許容できないほど高い場合は, 個々のコンピュータが過度のエラーを起こしていないか調べます。問題が広範囲に及んでいる場合は, 採用しているケーブルを調べます。たとえば, ノイズの多い非標準の同軸ケーブルに問題があるのなら, ツイスト・ペアのイーサネットに変更します。詳細は, `netstat(1)` を参照してください。

5.5.1 ネットワークのタイムアウトを減らす

低速のネットワーク・リンク, 輻輳しているネットワーク, ワイド・エリア・ネットワーク (WAN) では, NFS の性能は良くありません。特に, クライアント・システムでのネットワーク・タイムアウトで, NFS の性能が大幅に低下することがあります。このような状況が発生しているかどうかは, `nfsstat` コマンドを使用し, 呼び出しのタイムアウト率を調べることで判断できます。タイムアウトが呼び出し総数の 1 パーセントを超えると, NFS の性能が大幅に低下することがあります。タイムアウトおよび呼び出しの統計情報の `nfsstat` 出力の例は, 2.4.3 項を参照してください。

また, `netstat -s` コマンドを使用して, タイムアウトの問題があるか調べることもできます。`netstat` で出力された `ip` セクションの `fragments dropped after timeout` フィールドが 0 以外の値の場合は, 問題が発生している可能性があります。`netstat` コマンドの出力例は, 2.4.5 項を参照してください。

フラグメントのドロップがクライアント・システムで問題になっている場合は, `-rsize=1024` オプションと `-wsize=1024` オプションを指定して `mount` コマンドを実行し, NFS の読み書きバッファのサイズを 1 KB にします。

5.6 NFS サーバのチューニング

Tru64 UNIX はメモリ内のバッファ・キャッシュを使用して, できる限り, ディスク操作を行わないようにしています。このメモリは, 比較的低速な

ディスク入出力に対して、クライアントの待ち時間を減らすのに効果的です。ディスク操作のステージングやスケジューリングを行えば、ディスク入出力がさらに効率的になります。

ディスク・デバイス・ドライバが、ディスクのアーム位置に応じて、複数の要求を一度に処理するようにスケジュールできれば、性能が改善できます。繰り返される要求をキャッシュで処理すれば、ディスク入出力の総数は減ります。NFS の読み込み操作が多くある場合、サーバにメモリを追加すれば、サーバの性能は向上します。バッファ・キャッシュは、メモリ・サイズに比例して確保されるからです。

サーバに性能の問題があると、リモート書き込み要求があった場合に、システムのバッファ・キャッシュの効率が悪くなります。NFS は単純な、状態のない (stateless) プロトコルを使っているため、各クライアントの要求は完結して自己完備型である必要があります。サーバは各要求を完全に処理してから、クライアントに肯定応答を送信します。サーバがクラッシュしたり、肯定応答が紛失すると、クライアントは要求の再送を行います。そのため、以下の状況が発生します。

- データが安定したストレージへ安全に書き込まれるまで、サーバはクライアントの要求に肯定応答を行えない。
- クライアントはサーバによって安全に格納された変更のあったデータの量を完全に認識している。
- サーバがクラッシュすると、データが失われるため、サーバは変更のあったデータを揮発性ストレージでキャッシュできない。

NFS バージョン 2 では、書き込み操作は同期型になりました。すなわち、サーバは、書き込み要求を受信すると、データおよびそれを後で検索するために必要な情報を書き込んでから、応答します。Tru64 UNIX では、書き込みの集積 (write gathering) と呼ばれる方法を使って、この入出力負荷を減らしていますが、それでも性能への影響は大きく残っています。

NFS バージョン 3 では、書き込み要求は通常非同期型になり、書き込み操作の性能へ与える影響は最小になりました。サーバは書き込み要求を受け取ると、データ受信の肯定応答を行います。後でクライアントが、キャッシュに残っているデータの書き込みを要求するコミット要求を送信すると、サーバは、すべてのデータを安定したストレージに格納した段階で応答を行います。このプロトコルには、書き込み確認が含まれるので、クライアントは書き込み操作とコミット操作の間にサーバがクラッシュしてリブートした場合

は確認できます。サーバがクラッシュしてリブートしていた場合には、クライアントはコミットされなかった書き込み要求を再送し、サーバに正しいデータが保存されていることが保証できます。

システム・バッファ・キャッシュは、データを変更する NFS 要求の性能を改善するためには使えません。サーバが変更されたデータを揮発性メモリに書き込むだけだと、サーバがクラッシュした場合には、データの完全性が危険にさらされます。クライアントには安全に格納されたことになっていても、データが揮発性メモリにしか格納されていなかった場合、クラッシュが発生して、そのデータは失われる可能性があるからです。多くのクライアントのデータを 1 台のサーバに格納しているので、多くのクライアントに影響を及ぼします。しかし、変更結果を常に同期をとってディスクへ書き込んでおけば、データが失われることなく、サーバのクラッシュから回復できます。

NFS の操作はディスクに同期型でコミットするので、データの完全性が保証されることになり、サーバはシステム障害が発生しても復旧できます。しかし、この操作はディスクの速度に依存し、キャッシュ操作の場合のメモリ速度では実行できないので、性能は低下します。また、この操作はシリアルに行われるので、ディスクのアーム位置のスケジュールによる最適化も行えません。キャッシュの変更もディスクに同期をとって書き込まれるため、ディスクへの書き込みのトラフィックも最適化できません。

NFS サーバは少数の小規模なユーザ・レベル・プログラムしか実行しないので、システム・リソースは殆ど使用しません。NFS 要求の処理で CPU 時間と実時間の大部分が消費されるので、ファイル・システムのチューニングは重要です。ファイル・システムのチューニングについての詳細は、第 11 章を参照してください。

また、NFS を TCP (Transmission Control Protocol) 上で動作させている場合は、多くのアクティブなクライアントがいるときには、TCP のチューニングでも性能が改善される可能性があります。ネットワーク・サブシステムのチューニングについての詳細は、10.2 節を参照してください。NFS を UDP (User Datagram Protocol) 上で動作させている場合は、ネットワーク・サブシステムのチューニングは通常は必要ありません。

NFS だけをサービスしているシステムでは、表 5-4 のガイドラインに従ってチューニングを行ってください。

表 5-4: NFS サーバのチューニング・ガイドライン

ガイドライン	参照先
<code>maxusers</code> 属性の値に、想定される毎秒当たりのサーバ NFS 操作の回数を設定する。	8.1 節
<code>namei</code> キャッシュのサイズを大きくする。	11.1.2 項
AdvFS を使っている場合は、AdvFS アクセス構造体用に予約されているメモリを増やす。	11.1.5 項
UFS を使っている場合は、メタデータ・バッファ・キャッシュのサイズを大きくする。	11.1.4 項

5.6.1 NFS サーバ側の属性の変更

以下のネットワーク・ファイル・システム (nfs) サブシステム属性をチューニングすることで、ネットワーク・ファイル・システム・サーバの性能を改善できることがあります。

- 書き込みの集積 (5.6.1.1 項)
 - `nfs_write_gather` (バージョン 2.0)
 - `nfs_ufs_lbolt` (バージョン 2.0)
 - `nfs3_write_gather` (バージョン 3.0)
 - `nfs3_ufs_lbolt` (バージョン 3.0)
- サーバが書き込みを遅らせる時間を指定する (5.6.1.2 項)
 - `nfs_slow_ticks`
 - `nfs_fast_ticks`
 - `nfs_unkn__ticks`
- NFS の送受信バッファのサイズを大きくする (5.6.1.3 項)
 - `nfs_tcpsendspace`
 - `nfs_tcprecvspace`

注意

nfs カーネル・サブシステムのパラメータには、`dbx` を使ったアクセスだけが可能です。 `/sbin/sysconfig` コマンド、または `dxkerneltuner` GUI では、これに相当するシステム属性は

アクセスできません。 dbx についての詳細は、 3.2 節を参照してください。

詳細は、`sys_attrs_inet(5)` を参照し、カーネル・サブシステムの変更については、第 3 章を参照してください。

5.6.1.1 書き込みの集積

クライアントから要求されたデータのディスクへの書き込みを延期する、書き込みの集積機能によって、サーバの性能が改善される可能性があります。メタデータの更新は、書き込み要求の頻度よりも少ない頻度で行われます。書き込みの集積機能では、要求処理サイクルを少し遅らせ、同一のディスク・ブロックへの書き込み要求がサーバに到着するのを待ちます。しかし、サーバの CPU 時間を解放することによる全体的な利点が、多くの場合、この処理に必要なオーバーヘッドを上回ります。

書き込みの集積機能は、より少ない回数で、より大きいデータの書き込みを可能にするので、帯域幅を改善します(たとえば、シークや回転待ちが少なくなります)。NFS V3 クライアントには、非同期書き込みをサポートするものもあり、その場合、書き込みの集積機能の利点は明白ではありません。しかし、NFS V2 クライアントやある種の NFS V3 クライアントのように、非同期書き込みをサポートしていないクライアントでは、回復時に同期書き込みが必要であり、このとき書き込みの集積機能は有効になっています。この場合、システム性能は改善されます。

省略時の設定では、書き込みの集積機能は有効になっています。この機能の利点を最大に活用するために、クライアントで `nfsiods` をチューニングすれば、大規模なサーバのスケーラビリティの改善にも効果があります。

`nfs_write_gather` を無効にした場合、適用されない `nfs` 変数があります。たとえば、`nfs_ufs_lbolt` は、`nfs_write_gather` が有効になっているときだけ、オンまたはオフにできます。以下の条件は、NFS V2 と NFS V3 で共通です。

変数は以下の条件で変更できます。

1. `nfs_write_gather` がオン (省略時の設定) → `nfs_ufs_lbolt` がオン
→ サーバが書き込みを遅らせる時間を指定できる (5.6.1.2 項を参照)。

2. `nfs_write_gather` がオン → `nfs_ufs_lbolt` がオフ → `nfs_*_ticks` を変更しても、何の影響もない。
3. `nfs_write_gather` がオフ → `nfs_ufs_lbolt` がオフ → `nfs_*_ticks` を変更しても、何の影響もない。

PC や `biod` をサポートしないクライアント、あるいは書き込みを稀にしか要求しないクライアントのような単スレッドのダム・クライアントをサービスするときは、サーバからの応答が遅れても待つので、書き込みの集積を行うとクライアントの動作が遅くなります。これは、書き込みを遅らせるという、サーバ側に追加された待ちのオーバーヘッドが原因です。クライアントの性能を改善するには、`nfs_write_gather` を無効にします。すなわち、`dbx -k /vmunix` を使用して、0 を設定します。dbx についての詳細は、3.2 節または 5.6.1.1.1 項を参照してください。

5.6.1.1.1 クライアントの書き込み要求に対する NFS サーバの応答速度を改善する

`nfs_ufs_lbolt` パラメータを 0 に変更すると、以下のいずれかの条件のときに、クライアントの書き込み要求に対する NFS サーバの応答速度が著しく改善されることがあります。

- NFS サーバで書き込まれるストレージ・デバイスに非揮発性 (バッテリー・バックアップ付き) のキャッシュがある。
- NFS クライアントの大部分が (PC のように) 別の要求を送信する前に、前の要求に対する応答を待つシステムである。

`nfs_ufs_lbolt` の設定は、NFS V2 プロトコルが使われているときだけ有効です。NFS V2 の場合、NFS サーバでの同期型書き込み要求の性能改善は、書き込みの集積機能に依存します。書き込みの集積機能の特徴の 1 つは、クライアントへの書き込みの応答を返すのを遅らせて一定の間隔内の同一ファイルに対する後続の書き込み要求を待ち受けることです。遅延間隔内で処理されるすべての要求がストレージに安全に格納されたとき、サーバは関連する応答を同時に発行します。

サーバがクライアントからの追加の要求を待つ間隔は、シーク操作に必要な時間よりは短く、デバイスのキャッシュにフラッシュする時間よりは長い時間です。したがって、デバイス・キャッシュが非揮発性の場合 (データはメディアへの転送が完了する前に、安全にストレージに格納されています)、サーバが追加の要求を待っている時間は、効率的ではありません。さらに、

間隔が遅れることにより、同時に 1 つしか要求を発行せず、その応答を待つクライアント・システムでは、性能が低下します。

以下の例は、`dbx assign` コマンドを使用して、実行中のカーネルの `nfs_ufs_lbolt` パラメータを変更し、また `dbx patch` コマンドを使用して、新しい設定をディスク上の `/vmunix` ファイルにも確実に反映させる方法を示しています。

```
# dbx -k /vmunix
dbx version 5.1
Type 'help' for help.

(dbx) print nfs_ufs_lbolt = 1
(dbx) assign nfs_ufs_lbolt = 0
(dbx) patch nfs_ufs_lbolt = 0
```

`nfs_ufs_lbolt` パラメータは、NFS V2 を UFS で使用する場合にのみ使用するわけではありません。このパラメータを 0 に設定すると、AdvFS や CFS (クラスタ・ファイル・システム) での NFS V2 の性能も改善されます。しかし、クラスタ環境では、欠点があります。NFS サーバと NFS 用の CFS サーバは、同じメンバ・システムであるとは限りません。これらのサーバが異なるメンバ・システムにあり、`nfs_ufs_lbolt` が 0 に設定されている場合は、TCP マウントを通した NFS 書き込み要求に対する複数の応答は、クラスタ内の 2 つのサーバ間の 1 つの RPC にまとめられません。この場合、RPC の数が増え、クラスタの性能が低下します。

`nfs3_ufs_lbolt` に 0 を設定すると、NFS V2 ではなく、NFS V3 を使用した要求について、`nfs_ufs_lbolt` を設定したときと同様に遅延間隔が無くなります。NFS V3 では、クライアントの要求に対して書き込みの集積機能はほとんど利用しないので、`nfs3_ufs_lbolt` に 0 を設定しても、NFS V3 の性能は目立つほどには改善しません。

`dbx` についての詳細は、3.2 節と 5.6.1.1.1 項を参照してください。

5.6.1.2 サーバが書き込みを遅らせる時間を秒単位で指定する

`nfs` サブシステム変数の、`nfs_slow_ticks`、`nfs_fast_ticks` および `nfs_unkn_ticks` は、書き込みの集積機能用であり、サーバが書き込みを遅らせる時間を秒単位で指定するために使用します。書き込みの集積機能ではこれらの変数を使用して書き込みを遅らせ、同一のファイルに対する書き込み要求の数を増やすようにします。

書き込みの集積機能が無効になっていたり、`nfs_ufs_lbolt` がオフになっている場合は、`nfs` 変数は有効ではありません。 詳細は、 5.6.1.1 項を参照してください。

使用中のネットワーク・カードを調べるには、以下のいずれかのコマンドを実行します。

- 対象のインタフェースの `ifnet` 構造体をトレースし、対応する `if_type` を次のコマンドを使ってチェックします。

```
# dbx -k/vmunix
```

- `hwmgr` コマンドを使って、使用中のネットワーク・カードを表示します。たとえば、以下のように入力します。

```
# hwmgr -get attr -cat network -a name -a sub_category 42:
name = tu0
sub_category = Ethernet
```

`hwmgr` ユーティリティは、ネットワーク・ドライバから返される `if_type` の値に基づいて動作します。 `if_type` の値には、以下の3つがあります。

- `IFT_ETHER` (イーサネット)
- `IFT_FDDI` (FDDI)
- `IFT_ISO88025` (トークン・リングその他)

`hwmgr` の詳細は、 2.3.1 項を参照してください。

3 つの `nfs` 変数のどれを使用するか指定するには、NFS クライアント/サーバ通信で使用しているネットワーク・カードを調べて、表 5-5 の値と照合します。

表 5-5: ネットワーク・カードのタイプの確認

ネットワーク・カードのタイプ	変数	省略時の値 (ミリ秒)
FDDI	<code>nfs_fast_ticks</code>	8
Ethernet	<code>nfs_slow_ticks</code>	5
その他	<code>nfs_unkn_ticks</code>	8

ギガビット・イーサネットのような新しい高速なネットワーク・カードに対しては、`nfs_slow_ticks` (`IFT_ETHER`) のサイズを小さくすると性能が向上することがあります。

5.6.1.3 NFS 送受信バッファのサイズを大きくする

`nfs_tcpsendspace` および `nfs_tcprecvspace` 変数は、NFS での TCP ソケットの送受信バッファの省略時のサイズを指定します。ギガビット・イーサネットのような高速のネットワーク・アダプタを使用している場合は、これらの変数の値を大きくするとシステムの性能が改善することがあります。

次のコマンドを使用してこれらの変数の値を変更します。

```
# dbx -k/vmunix
```

`nfs_tcpsendspace` と `nfs_tcprecvspace` の省略時の値は 98304 バイトです。NFS V3 では、I/O 転送サイズが 64 KB のとき、ほとんどのネットワーク・アダプタでの推奨値は省略時の値と同じになります。NFS Version 2.0 では、I/O 転送サイズが 8 KB のとき、省略時の値が推奨値になります。

`tcpdump` を使用して、リモート・システムの NFS が 65536 バイトより大きなサイズの TCP ウィンドウをサポートしているかどうかを確認します。TCP ウィンドウの省略時のサイズは 65536 バイトです。省略時の設定では RFC 1323 をサポートしており、ウィンドウ・スケーリングによって大きなサイズのウィンドウを設定することができます。ただし、リモート・システムの NFS が RFC 1323 をサポートしていないときは、接続時に送信する SYN パケットが拒否されます。

サーバの `nfs_tcpsendspace` ウィンドウ・サイズを大きく設定すると、パケットの送信が速くなり、性能が向上します。クライアント側では、クライアント・システムにギガビット・イーサネットが実装されているときには同じ利点を享受することができます。

ウィンドウ・スケーリングの詳細は、『ネットワーク・プログラミング・ガイド』を参照してください。

5.7 NFS クライアントのチューニング

クライアントにディスクまたはメモリを追加すると、2 つの理由で性能が向上します。1 つはアクセス時間を改善できるため、2 つにはサーバやネットワークの全体の負荷が削減できるためです。クライアントでは、共用されないファイル(ルート、スワップ領域、一時ファイルなど)に対してローカル・ディスクを使うことで、ネットワーク・ファイル・システム(NFS)の性能問題を避けることができます。ディスクレス・クライアントの場合、メモリを追加すると、性能は著しく向上します。クライアントのスワッピングやペー

ジングの頻度が下がるからです。ローカル・リソースを追加することで、サーバやネットワークの負荷を減少させることができます。

クライアントの性能は、メモリやディスクを追加することで簡単に向上させることができますが、この向上策は、オペレーティング・システムの保守に必要な管理作業を考えると、コスト効果的には優れていません。たとえば、重要なデータをローカル・ディスクに保存している場合、ディスクのバックアップが必要になります。データが共用されている場合は、他のシステムのアクセスも考える必要があります。サーバにリソースを追加した場合は、クライアントにリソースを追加した場合よりも、追加の管理コストは低くなります。

以降の節では、`nfs` サブシステム属性を変更して、`nfs` の性能を改善する方法を説明します。

5.7.1 NFS クライアント側の属性の変更

以下の `nfs` サブシステム属性をチューニングすることで、NFS サーバの性能が改善できることがあります。

- 読み取り性能を改善する (5.7.1.1 項)
 - `nfs3_readahead`
 - `nfs3_maxreadahead`
- クライアントが再送を開始するまでの時間を制御する (5.7.1.2 項)
 - `nfs3_jukebox_delay`
- ディレクトリ名の検索 (5.7.1.3 項, 5.7.1.4 項)
 - `nfs_dnlc`
 - `nfs_nnc`
- NFS クライアント間でのファイルの一貫性を指定する (5.7.1.5 項)
 - `nfs_cto`
- NFS クライアントがファイル属性をフェッチする際の動きを変える (5.7.1.6 項)
 - `nfs_quicker_attr`
- NFS 送受信バッファのサイズを大きくする (5.6.1.3 項)
 - `nfs_tcpndata`

- nfs_tcprecvspace

注意

nfs カーネル・サブシステムのパラメータには、dbx を使ったアクセスだけが可能です。 /sbin/sysconfig コマンド、または dxkerneltuner GUI では、これに相当するシステム属性はアクセスできません。

詳細は、sys_attrs_inet(5) を参照し、カーネル・サブシステム属性の変更については、第 3 章を参照してください。

5.7.1.1 読み取り性能を改善する

NFS バージョン 3.0 のクライアントが、長いシーケンシャルな読み込みまたは部分ブロックの書き込みを完了して、アイドル状態のクライアントが出現した場合、NFS V3 のクライアントは、先読みを試みます。nfs3_readahead 変数には、クライアントが先読みを行うページ数を指定しますが、最大ページ数を越えることはできません。nfs3_maxreadahead 変数には、クライアントが先読みする最大のページ数を指定します。nfs3_readahead の省略時の値は 2 で、nfs3_maxreadahead の省略時の値は 8 です。

先読み機能は、読み込み性能の改善に効果があります。新しい高速なネットワーク・インタフェースを備えたシステムの場合は、両方の変数と、実行する nfsiod の数をチューニングすると、ネットワーク・インタフェースを飽和状態で動作させることができます。これによって、ハードウェアの能力を最大限に引き出すことが可能になります。この変数をチューニングすると、新しいギガビット・ネットワーク・カードでは、読み取り性能を 2 倍に向上させることができます。

5.7.1.2 クライアントが再送を開始するまでの時間を制御する

nfs3_jukebox_delay は、クライアントが再送を開始するまでの時間を秒単位で制御するクライアント側の変数です。負荷の高いサーバでのトランザクション処理では、nfs3_jukebox_delay の値を大きくして、不必要なクライアントの要求の再送を減らすことができます。nfs3_jukebox_delay の省略時の値は 10 秒です。

`nfs3_jukebox_delay` 変数は、ストレージ HSM メカニズムとは関係がありません。この名前は、サーバから送信されるエラー・メッセージ `NFS3ERR_JUKEBOX` にちなんで付けられました。JUKEBOX という用語は、NFS の歴史的な出来事に由来しており、ファイルが一時的にアクセスできないことを意味しています。これによって、クライアントはサーバの状態を認識し、要求を繰り返し再送する代わりに、ファイルへのアクセスを意識的に遅らせるかどうかの判断ができるようになります。

5.7.1.3 ディレクトリ名の検索用キャッシュ (DNLC)

`nfs_dnlc` 変数には、ディレクトリ名の検索用キャッシュを指定します。省略時の設定では、クライアントは最新のファイル・システム・ディレクトリの検索操作の結果のキャッシュを保持しています。サーバへの検索要求の数が減ることになるので、クライアントの性能は改善されます。

ディレクトリ名の検索キャッシュを無効にするには、`mount` コマンドに `-noac` オプションを指定します。`-noac` が `mount` 時に指定されていない場合は、`nfs_dnlc` 変数をオフにして、`dnlc` を無効にすることができます。

サーバがディレクトリ内のファイルを頻繁に変更している場合には、`nfs_dnlc` を無効にすると効果があります。こうすると、ファイルのオープンで古いファイルのハンドルを使うのをやめて、新たに検索呼び出しを行うように強制できます。

5.7.1.4 ネガティブ名キャッシュの検索 (NNC)

`nfs_nnc` 変数には、ネガティブ名キャッシュを指定します。クライアントの検索操作では、キャッシュ検索に失敗した場合、失敗した `vfs` 層のキャッシングを記録するために、ネガティブ名キャッシュも保持します。これによって、将来、回線を通して同じサーバに対する不必要な検索を行わないようにします。

省略時の設定では、`nfs_nnc` は有効になっています。NFS ディレクトリの検索要求を行わない、または検索要求の数の少ないアプリケーションでは、`nfs_nnc` を無効にするとアプリケーションの性能が改善される可能性があります。

サーバがディレクトリ内のファイルを頻繁に変更している場合には、`nfs_dnlc` を無効にすると効果があります。こうすると、ファイルのオー

プンで古いファイルのハンドルを使うのをやめて、新たに検索呼び出しを行うように強制できます。

5.7.1.5 NFS クライアント間でのファイルの一貫性を指定する

`nfs_cto` 変数には、CTO (closed-to-open) 処理を指定します。ファイルがクローズされ、そのファイルに関連するすべての変更されたデータがサーバにフラッシュされるとき、またはファイルをオープンするとき、クライアントはクライアント内のキャッシュを確認する要求を、サーバに送信します。この操作によって、複数の NFS クライアント間でファイルの一貫性が保たれます。

マウント時に `-nocto` オプションを指定すると、クライアントはクローズ時にフラッシュを実行しません。そのため、同一ファイルの、複数のクライアントに格納されているコピーが異なったものになる可能性があります。`mount` コマンドの実行例は、次のとおりです。

```
# mount -nocto fubar:/abc/local
```

省略時の設定では、`nfs_cto` はオンになっています。この変数をオンにすることの利点は、複数のクライアントからアクセスされるファイルの一貫性が失われる問題を回避できることです。最初のクライアントがファイルへの書き込みを行いクローズすると、次のクライアントがそのファイルをオープンしたときに、そのクライアントにあるデータが最新のものであることが保証されます。

`nfs_cto` をオフにする利点は、特定のファイル・システムへのアクセスが、1 つのクライアントだけから実行される場合にあります。この場合は、`nfs_cto` を無効にすると性能が改善される可能性があります。

クライアントは CTO による一貫性をマウント時に検査します。まず、クライアントは `nfs_cto` 変数を検査し、`mount` の `-nocto` オプションの設定を検査します。その後、クライアントは `nfs_cto` がオンになっているか、または `mount` の `-nocto` オプションが設定されていないかを検査して、CTO による一貫性の検査をします。

マウント時に `-nocto` オプションを使用して、`nfs_cto` がオフになっている場合は、`dbx -k` を使って `nfs_cto` の設定を行うと、クライアントはマウントされているファイル・システムの CTO による一貫性の検査を行うようになります。

5.7.1.6 NFS クライアントがファイル属性をフェッチする際の動きを変える

`nfs_quicker_attr` 変数を 0 以外の値に設定すると、ファイル属性をフェッチする (たとえば、`ls -l` または `stat(2)`) 際の NFS クライアントの動作が変わります。省略時の設定では、NFS はファイル I/O の完了を待ってから `fsync()` 関数を実行して最新の属性情報を入手します。ただし、このことによって、ディスクより高速のインタフェースを通してファイルを書き込む際に遅れが生じてしまいます。管理された環境では、この変数を設定すると、キャッシュされた属性をフェッチするようになります。

`nfs_quicker_attr` 変数の変更は、テストまたはデバッグ環境で、大きなファイルの書き込みの状況を観察する際に、(たとえば、`ls -l` を使用して) ファイル属性を何度もフェッチするときに役に立ちます。

インターネット・サーバのチューニング

この章では、Tru64 UNIX をチューニングしてインターネット・サーバの性能を改善する方法を説明します。以下の項目で、各種の構成ガイドラインを示し、モニタリング・ツールをいくつか説明して、基本的な推奨チューニングと高度な推奨チューニングを示します。

- インターネット・サーバの性能改善 (6.1 節)
- 基本的な推奨チューニング (6.2 節)
- 高度な推奨チューニング (6.3 節)

すべての推奨チューニングがすべての構成に適用できるわけではありません。また、わずかな性能改善しか得られない場合もあります。したがって、これらの推奨チューニングを適用する前に、現在の構成と処理量を完全に把握し、本書を注意深く読む必要があります。

注意

Tru64 UNIX バージョン 5.0 以上で変更された属性名があります。

6.1 インターネット・サーバの性能改善

この節では、インターネット・サーバの性能を改善する方法を説明します。ここでは、以下のトピックに着目して、各種の構成ガイドラインを示し、いくつかのモニタリング・ツールについて説明します。

- ハードウェアの構成 (6.1.1 項)
- メモリとスワップ領域の構成 (6.1.2 項)
- IP アドレスのロギング (6.1.3 項)
- ネットワーク統計情報のモニタリング (6.1.4 項)
- ソケット統計情報のモニタリング (6.1.5 項)
- 仮想メモリ統計情報のモニタリング (6.1.6 項)

- 構成情報の収集 (6.1.7 項)

6.1.1 ハードウェアの構成

インターネット・サーバの性能を改善するには、ハードウェア構成に関する以下のガイドラインが役立ちます。

- システム、ディスク、アダプタ、およびコントローラのファームウェアのバージョンが最新であることを確認してください。
- 処理量をこなせるだけのメモリ容量とスワップ領域を備えていることを確認してください。詳細は、6.1.2 項を参照してください。
- インターネット・サーバの構成では、ディスク、アダプタ、およびコントローラなどに、高性能なストレージ・ハードウェアを使用してください。
- 高性能と高可用性を実現するために、LSM (Logical Storage Manager) またはハードウェア RAID ストレージ構成を使用してください。
- ハードウェア RAID 構成でライトバック・キャッシュを使用すると、インターネット・サーバの性能が著しく改善されます。
- /tmp ディレクトリと /var/tmp ディレクトリは異なるファイル・システムに配置してください。また、可能であれば異なるディスクに配置してください。最高の性能を得るには、ディスク上のディレクトリは、ライトバック・キャッシュを有効にした RAID コントローラの管理下に置きます。

6.1.2 メモリとスワップ領域の構成

サーバの処理量をこなせるだけのメモリ容量とスワップ領域を用意してください。メモリ容量やスワップ領域が不足すると、性能上の問題が発生することがあります。メモリとスワップ領域を構成するには、以下の手順に従ってください。

1. 処理量をこなすために必要な物理メモリ量を調べます。
2. スワップ領域の割り当てモードを、即時モードと延期モードのどちらにするかを選択します。
3. スワップ領域の必要量を調べます。
4. ディスク入出力を効率的に分散できるように、スワップ領域を構成します。

6-2 インターネット・サーバのチューニング

インターネット・サーバでは、システムやアプリケーションの動作に必要なメモリの他に、確立されている接続ごとに以下のメモリ・リソースが必要になります。

- カーネルのソケット構造体
- インターネット・プロトコル制御ブロック (inpcb) 構造体
- TCP 制御ブロック構造体
- パケットを受信するたびに消費されるソケット・バッファ領域

これらのメモリ・リソースは、接続終端ごとに 1 KB 必要になります (ソケット・バッファ領域は含みません)。したがって、10,000 の接続に対応するためには、10 MB のメモリが必要になります。

厳しいピーク時の負荷に耐えられるだけのメモリをサーバが備えていることを確認してください。負荷が高い日にサーバが必要とするメモリ量の 10 倍のメモリを構成しておけば、時折発生する一時的な高負荷にも対応することができます。

接続をサービスするために必要なだけのメモリ・リソースさえあれば、サーバの TCP 接続の処理量を制限するものではありません。しかし、メモリ量が十分でない場合は、十分な数の既存の接続が解放されるまで、サーバは新しい接続要求を拒否します。ネットワーク・サブシステムで現在使用されているメモリをモニタリングするためには、`netstat -m` コマンドを使用します。`netstat` コマンドの詳細は、6.1.4 項を参照してください。

6.1.3 IP アドレスのロギング

インターネット・サーバでクライアント・ホスト名のログを記録している場合、アプリケーション・ソフトウェアがクライアント・ホスト名を得るためにシステムに DNS 逆引きルックアップを要求することがあります。DNS 逆引きルックアップは時間のかかる処理なので、クライアントの数が多い高負荷のサーバでは性能上の問題が発生する可能性があります。

クライアント・ホスト名の代わりに、クライアント IP (インターネット・プロトコル) アドレスをログに記録するように、インターネット・ソフトウェアを変更することができます。IP アドレスでログを記録するようにすれば、重要な情報を失うことなく、インターネット・サーバの効率が著しく改善されます。

クライアント・ホスト名のロギングを無効にする方法については、インターネット・サーバのソフトウェア・ベンダが提供するドキュメントを参照してください。たとえば、Apache HTTP サーバ・ソフトウェアを変更するための情報は、次の URL の Apache HTTP サーバのドキュメント・サイトで入手できます。

<http://httpd.apache.org/docs/>

6.1.4 ネットワーク統計情報のモニタリング

`netstat` コマンドは、各プロトコルのネットワーク経路やアクティブ・ソケットの情報などのネットワーク統計情報を表示します。このコマンドは、受信パケット数と送信パケット数、パケット衝突の回数、ネットワーク操作で使用されたメモリに関する情報、IP、ICMP、TCP、UDP のプロトコル層に関連する統計情報などの累積統計情報も表示します。

`netstat` コマンドを使用して調べることができるネットワーク統計情報を表 6-1 に示します。

表 6-1: ネットワーク統計情報モニタリング・ツール

ツール	説明	参照先
<code>netstat -i</code>	過度の入力エラー (Ierrs)、出力エラー (Oerrs)、あるいは衝突 (Coll) を表示する。これは、ネットワークに問題がある可能性があることを示す。	2.4.5.1 項
<code>netstat -is</code>	ネットワーク・デバイス・ドライバのエラーを検査する。	2.4.5.2 項
<code>netstat -m</code>	ネットワークが使用しているメモリがシステムにインストールされているメモリ総量に対して多すぎないか調べる。	2.4.5.3 項
<code>netstat -an</code>	既存のネットワーク接続の状態を調べる。	2.4.5.4 項
<code>netstat -p ip</code>	不正チェックサム、長さの問題、過度のリダイレクト、およびリソース問題が原因のパケット紛失を検査する。	2.4.5.5 項
<code>netstat -p tcp</code>	再送、パケットの順序の乱れ、および不正チェックサムを検査する。	2.4.5.6 項
<code>netstat -p udp</code>	不正チェックサムとソケット・フルを検査する。	2.4.5.6 項
<code>netstat -rs</code>	ルーティング統計情報を表示する。	2.4.5.7 項

表 6-1: ネットワーク統計情報モニタリング・ツール (続き)

ツール	説明	参照先
netstat -s	IP, ICMP, IGMP, TCP, およ び UDP プロトコル層に関する統 計情報を表示する。	2.4.5.8 項
sysconfig -q socket	現在の属性値を表示する。表示された 値によってキューがオーバーフローしてい ることがわかったときには、ソケットの リッスン・キュー限界値を増加させる。	6.1.5 項
vmstat	仮想メモリの利用状況のデー タを表示する。	6.1.6 項

詳細は、netstat(1) を参照してください。

6.1.5 ソケット統計情報のモニタリング

次の 3 つの socket サブシステム属性で、ソケット・リッスン・キューのイベントをモニタリングします。

- sobacklog_hiwat 属性によって、すべてのサーバ・ソケットに対する保留中の要求の最大数をカウントします。
- sobacklog_drops 属性によって、ソケットに対する SYN_RCVD 接続がキューに入れられた数がソケットのバックログ限界値に等しくなったために、システムが受信 SYN パケットをドロップした回数をカウントします。
- somaxconn_drops 属性によって、ソケットに対する SYN_RCVD 接続がキューに入れられた数がソケットのバックログ長の上限 (somaxconn 属性) に等しくなったために、システムが受信 SYN パケットをドロップした回数をカウントします。

これらの属性のブート時の初期値は 0 です。sysconfig -q socket コマンドを使って、現在の属性値を表示します。キューがオーバーフローしていることを示す値が表示された場合は、ソケット・リッスン・キューの限界値を大きくする必要があります。sysconfig -q socket コマンドの例を、次に示します。

```
# sysconfig -q socket
socket:
pftimerbindcpu = 0
sbcompress_threshold = 0
sb_max = 1048576
sobacklog_drops = 0
```

```
sobacklog_hiwat = 21
somaxconn = 65535
somaxconn_drops = 0
sominconn = 65535
mbuf_ext_lock_count = 64
umc_min_len = 1024
umc = 0
```

sominconn 属性の値は，somaxconn 属性の値と同じにすることを勧めします。このようにした場合は，somaxconn_drops の値は，sobacklog_drops の値と等しくなります。

ただし，sominconn 属性の値が 0 (これが省略時の値) で，また 1 つまたは複数のサーバ・アプリケーションが listen システム・コールのバックログ引数に不十分な値を指定している場合は，sobacklog_drops の値は somaxconn_drops カウンタが増加する速度よりも速い速度で増加します。このような状況が発生する場合は，sominconn 属性の値を大きくしてください。sominconn 属性の詳細は，6.2.3.2 項を参照してください。

6.1.6 仮想メモリ統計情報のモニタリング

vmstat コマンドは，仮想メモリの使用状況に関するデータを表示します。このコマンドは，過度のページングが発生してインターネット・サーバの性能が低下していないか調べるのに役立ちます。vmstat コマンドの例を，次に示します。

```
# vmstat 1
Virtual Memory Statistics: (pagesize = 8192)
procs  memory          pages          intr          cpu
r  w  u  act  free wire fault cow zero react pin pout  in  sy  cs  us  sy  id
7 526 59 80K 758 45K 402M 94M 132M 1M 74M 139K 757 42K 1K 38 14 48
7 526 59 81K 278 45K 939 15 896 0 11 0 824 2K 1K 85 11 4
6 528 59 81K 285 45K 595 67 411 0 10 0 983 5K 2K 81 17 2
7 526 59 81K 353 45K 560 31 446 0 17 0 781 2K 1K 87 10 3
7 526 59 81K 353 45K 406 0 406 0 0 0 1K 4K 2K 85 13 2
7 527 59 81K 288 45K 406 0 406 0 0 0 1K 7K 4K 81 18 1
9 524 59 81K 350 45K 640 72 420 0 13 0 999 3K 2K 85 13 2
.
.
.
```

「memory」欄の値は，8 KB のページ単位で示されます。未使用ページ・リスト (free) のサイズをチェックしてください。未使用ページの数をアクティブ・ページ (act) および固定ページ (wire) の値と比較します。未使用ページ，アクティブ・ページ，および固定ページの合計は，システムの物理メモリの容量に近くなければなりません。free の値は小さくても構いませんが，この値が常に小さく (128 ページ未満)，ページングとスワッピングが過度に発生している場合は，物理メモリが不足している可能性があります。

また、ページ・アウト (pout) 欄もチェックしてください。ページ・アウトの数が常に大きい場合は、メモリが不足している可能性があります。また、スワップ領域が不足しているか、スワップ領域が不適切に構成されている可能性があります。swapon -s コマンドを使って、スワップ・デバイスの構成を表示し、iostat コマンドを使ってどのスワップ・ディスクが頻繁に使用されているかを調べてください。

詳細は、vmstat(1)、swapon(8)、および iostat(1) を参照してください。

6.1.7 構成情報の収集

sys_check スクリプトは、構成情報を収集して HTML 形式に変換してファイルに出力する ksh スクリプトです。このスクリプトは、構成上の問題を検出すると警告を発し、カーネル・サブシステムの属性の設定値を調べ、推奨される属性チューニングを提示します。詳細は、2.3.3 項を参照してください。

sys_check を使用するときには、最新版を使用してください。最新版は、次の Web ページから入手できます。

http://www.tru64unix.compaq.com/sys_check/sys_check.html

6.2 基本的な推奨チューニング

インターネット・サーバの性能に影響を与えるカーネル・サブシステムの属性には、多くのものがあります。ここで言うインターネット・サーバとは、Web サーバ、ftp サーバ、メール・サーバおよびメール中継サーバ、プロキシ・サーバ、キャッシュ・サーバ、ゲートウェイ・システム、およびファイアウォール・システムなどを指します。この節では、以下のサブシステムのいくつかの属性について、基本的な推奨チューニングを説明します。

- インターネット (6.2.1 項)
- プロセス (6.2.2 項)
- ソケット (6.2.3 項)

注意

カーネル・サブシステム属性の一部は、稼働中のシステムで値を変更してその値を適用することができます。その他の属性は、新しい値を有効にするためにはシステムをリブートする必要があります。

ます。ある属性がシステムの稼働中にチューニングできるかどうかを確認する方法については、3.3.1 項を参照してください。

基本的な推奨チューニングを適用すれば、大半のインターネット・サーバ構成で、最大の性能改善が得られます。これらのチューニングを実施してもまだ性能が不十分な場合は、6.3 節で説明している、その他のカーネル・サブシステム属性を変更することで性能を改善できることがあります。

また、CPI (Compaq Continuous Profiling Infrastructure、以前は DCPI と呼ばれていた) ツールを使って、CPU 時間を大量に消費しているシステム・コンポーネントに関する詳細情報を得ることができます。CPI は Advanced Development Kit として提供されています。詳細は、次の URL の Web サイトを参照してください。

<http://www.tru64unix.compaq.com/dcp>

6.2.1 インターネット属性の変更

以下の `inet` インターネット・サブシステムの属性をチューニングすることで、インターネット・サーバの性能を改善することができます。

- `tcbhashsize` (6.2.1.1 項)
- `pmtu_enabled` (6.2.1.2 項)
- `ipport_userreserved` (6.2.1.3 項)

詳細は、`sys_attrs_inet(5)` を参照し、カーネル・サブシステム属性の変更については第 3 章を参照してください。

6.2.1.1 TCP ハッシュ・テーブルのサイズを大きくする

`tcbhashsize` 属性は、TCP (Transmission Control Protocol) の `inpcb` ハッシュ・テーブル内のバケット数を指定します。カーネルは TCP パケットを受信するたびに、接続ブロックを検索する必要があるため、テーブルのサイズを大きくすると検索の速度が速くなり、性能が改善されます。

ただし、ハッシュ・テーブルのサイズを大きくすると、固定メモリが少し増加します。また、SMP システムでは、これが TCP ハッシュ・テーブルでのボトルネックになる可能性があります。

省略時の値は 512 バケットです。推奨値は 16384 です。

6.2.1.2 PMTU 検出を無効にする

サーバ間で転送されるパケットは、ルータや、イーサネット・ネットワークのようなパケットの小さなネットワークでのデータ転送を容易にするために、同一サイズのユニットに分割されます。

`pmu_enabled` 属性が有効になっていると、オペレーティング・システムはサーバ間の最大の共通 PMTU (Path maximum transmission unit) 値を調べ、それをユニット・サイズとして使用します。サーバに接続するクライアント・ネットワークごとに、ルーティング・テーブル・エントリも作成されます。

主にリモート・トラフィックを処理し、ルーティング・テーブルが 1000 エントリより多くなってインターネット・サーバの性能が低下している場合、PMTU 検出を無効にすることで、ルーティング・テーブルのサイズを小さくし、サーバの効率を改善することができます。ただし、サーバが主としてローカル・トラフィックを処理しており、リモート・トラフィックが少ない場合は、PMTU 検出を無効にすると帯域幅が狭くなります。 `netstat -r` コマンドを使って、ルーティング・テーブルの内容を表示してください。

省略時の値は、1 (PMTU は有効) です。推奨値は、0 (PMTU は無効) です。

6.2.1.3 送信接続ポート数を増やす

TCP または UDP のアプリケーションが送信接続を確立する際に、カーネルは接続ごとに、予約されていないポート番号を動的に割り当てます。

カーネルは、`ipport_userreserved_min` から `ipport_userreserved` までの範囲で、ポート番号を選択します。

省略時の属性値を使用すると、送信ポートの範囲は、ポート 1024 ~ 5000 になるため、同時送信接続数は 3976 (5000 マイナス 1024) に制限されます。

プロキシ・サーバ、キャッシュ・サーバ、ゲートウェイ・システム、またはファイアウォール・システムの同時接続数が 4000 を超える場合は、`ipport_userreserved` 属性の値を変更したほうがよい可能性があります。省略時の値は 5000 で、これは最小値です。推奨値は 65535 で、これは最大値です。65535 より大きい値や 5000 より小さい値は指定しないでください。

6.2.2 プロセス属性の変更

以下の `proc` プロセス・サブシステムの属性をチューニングすることで、インターネット・サーバの性能を改善することができます。

- `maxusers` (6.2.2.1 項)
- `max_proc_per_user` (6.2.2.2 項)
- `max_threads_per_user` (6.2.2.3 項)
- `max_per_proc_data_size` (6.2.2.4 項)
- `max_per_proc_address_space` (6.2.2.5 項)

これらの属性はシステム・リソースの限界値を設定します。インターネット・サーバがリソースの限界に近づいていると思われる場合は、これらの属性の値を増加させてください。ただし、これらの属性の値を増加させると、システムのメモリ消費量が増加します。

詳細は、`sys_attrs_proc(5)` を参照し、カーネル・サブシステム属性の変更については第 3 章を参照してください。

6.2.2.1 システム・テーブルとデータ構造体のサイズを大きくする

システムのアルゴリズムでは、各種のデータ構造体とシステム・テーブルのサイズを決めるために、`maxusers` 属性を使用します。`maxusers` 属性の値を大きくすると、より多くのシステム・リソースをプロセスが使用できるようになります。ただし、固定メモリの量が増加します。

システムでリソースの不足が発生していて(たとえば、「Out of processes」, 「No more processes」, または「pid table is full」といったメッセージが表示される)、メモリが十分にある場合は、`maxusers` 属性の値を大きくしてください。

`maxusers` 属性の適切な値を決めるためには、性能が改善されるまで、省略時の値を 2 倍にしていきます。たとえば、1 GB のメモリがある場合は、`maxusers` 属性の値を 512 に増加させます。2 GB のメモリがある場合は、値を 1024 に増加させます。インターネット・サーバ、Web サーバ、プロキシ・サーバ、キャッシュ・サーバ、ファイアウォール・システム、またはゲートウェイ・システムの場合は、`maxusers` 属性の値を 2048 に増加させます。

省略時の値は、システムに実装されている物理メモリの量に従って、16 ~ 2048 の間で変化します。2048 より大きな値にすることは、お勧めできません。

システム管理者は、`maxusers` 属性を、次のコマンドで変更できます。

```
# sysconfig -r proc maxusers=N
```

*N*に新しい値を指定します。このコマンドによって、`pid` テーブルが自動的に拡張されます。他のシステム・テーブルのサイズ変更は、`/etc/sysconfigtab` ファイルの `maxusers` 属性に新しい値を指定してシステムをリブートするまで行われません。

6.2.2.2 ユーザごとのプロセス数を増やす

`max_proc_per_user` 属性は、各ユーザ (スーパーユーザを除く) に割り当てることができるプロセスの最大数を指定します。

システムでプロセス不足の状況が発生している場合は、この属性の値を大きくしてください。マルチプロセスのインターネット・サーバ (たとえば、`IPlanet`、`Apache`、`CERN`、または `Zeus` を稼働させているサーバ) の場合も、この属性の値を大きくしたほうがよい可能性があります。この値を大きくすると、固定メモリの量も増加することに注意してください。

省略時の値は 64 です。推奨値は 2000 です。選択する値は、システムで起動できるプロセスの最大数を超えてはなりません。インターネット・サーバの場合は、これらのプロセスには、`CGI` プロセスが含まれます。この属性の値として 0 を指定すると、ユーザごとのプロセス数には制限がなくなります。

6.2.2.3 ユーザごとのスレッド数を増やす

`max_threads_per_user` 属性は、各ユーザ (スーパーユーザを除く) に割り当てることができるスレッドの最大数を指定します。

システムでスレッド不足の状況が発生している場合は、この属性の値を大きくしてください。マルチスレッドのインターネット・サーバ (たとえば、`Netscape FastTrack` または `Netscape Enterprise` を稼働させているサーバ) の場合は、この属性の値を大きくしたほうがよい可能性があります。

省略時の値は 256 です。推奨値は 4096 です。選択する値は、システムで起動できるスレッドの最大数を超えてはなりません。

6.2.2.4 ユーザ・プロセスのデータ・セグメント・サイズの限界値を大きくする

`max_per_proc_data_size` 属性は、データ・セグメント・サイズの上限の限界値を指定します。大きなプログラムや大きなメモリを使用するプロセスには、この属性値を大きくしないと実行できないものがあります。

「`Out of process memory`」というメッセージが表示される場合は、この限界値を大きくしてください。

省略時の値は 1073741824 (1 GB) です。推奨値は 10737418240 (10 GB) です。システムのメモリが 10 GB より大きい場合は、この値をもっと大きくすることができます。

6.2.2.5 ユーザ・プロセスのアドレス空間の限界値を大きくする

`max_per_proc_address_space` 属性は、ユーザ・プロセスのアドレス空間の上限の限界値 (仮想メモリのバイト数) を指定します。大きなプログラムや大きなメモリを使用するプロセスには、この属性値を大きくしないと実行できないものがあります。ただし、アドレス空間の限界値を大きくすると、メモリの消費量も少し増加します。

省略時の値は、Tru64 UNIX バージョン 5.0 以降が稼働しているシステムでは 4,294,967,296 バイト (4 GB) です。

推奨値は 10,737,418,240 (10 GB) です。システムのメモリが 10 GB より大きい場合は、この値をもっと大きくすることができます。

6.2.3 ソケット属性の変更

以下のソケット属性をチューニングすることで、インターネット・サーバの性能を改善することができます。

- `somaxconn` (6.2.3.1 項)
- `sominconn` (6.2.3.2 項)
- `sbcompress_threshold` (6.2.3.3 項)

詳細は、`sys_attrs(5)` を参照し、カーネル・サブシステム属性の変更については第 3 章を参照してください。

6.2.3.1 保留中 TCP 接続の最大数を増やす

`somaxconn` 属性は、各サーバのソケット (たとえば、HTTP サーバのソケット) の保留中 TCP 接続の最大数 (ソケット・リッスン・キューの限界値) を指定します。TCP 接続の保留は、インターネットでのパケットの紛失やサービス拒否アタックによって発生します。負荷の高いインターネット・サーバでは、保留中 TCP 接続の数が増えることがあります。リッスン・キューの接続限界値が小さすぎると、受信接続要求がドロップされることがあります。

省略時の値は 1024 です。推奨値は 65535 で、これは最大値です。最大値より大きい値を指定したときの動作は予測できないため、最大値より大きい値は、指定しないでください。

6.2.3.2 保留中 TCP 接続の最小数を増やす

`sominconn` 属性は、各サーバのソケットの保留中 TCP 接続 (backlog) の最小数を指定します。この属性は、システムが同時に処理できる SYN パケットの最大数を制御し、これ以上の要求は破棄されます。クライアントがソケット・リッスン・キューをエラーの TCP SYN パケットでいっぱいになると、他のユーザがキューにアクセスできなくなるため、ネットワーク性能が低下します。

`sominconn` 属性の値は、アプリケーションに固有の backlog 値より優先されます。このアプリケーションに固有の値は、サーバ・ソフトウェアによっては、小さ過ぎる場合があります。アプリケーションのソース・コードがない場合は、`sominconn` 属性を使って、接続保留の限界値にアプリケーションに適した値を設定してください。

省略時の値は 0 です。推奨値は 65535 で、これは最大値です。`sominconn` 属性の値を、`somaxconn` 属性の値と同じにすることをお勧めします。`somaxconn` 属性の詳細は、6.2.3.1 項を参照してください。

6.2.3.3 mbuf クラスタの圧縮を有効にする

`sbcompress_threshold` 属性は、mbuf クラスタをソケット層で圧縮するかどうかを制御します。省略時の設定では、mbuf クラスタは圧縮されないため、プロキシ・サーバやキャッシュ・サーバがすべての使用可能な mbuf クラスタを消費してしまう場合があります。この問題は、イーサネットの代わりに FDDI を使用しているときによく発生します。mbuf クラスタをモニタリングする方法の詳細は、2.4.5.3 項を参照してください。

mbuf クラスタの圧縮を有効にするには、`sbcompress_threshold` 属性を変更し、値を指定します。パケット・サイズがこの値より小さい場合、そのパケットは既存の mbuf クラスタにコピーされます。

省略時の値は 0 (mbuf 圧縮は無効) です。プロキシ・サーバ、キャッシュ・サーバ、ゲートウェイ・システム、またはファイアウォール・システムの場合、推奨値は 600 バイトです。

6.3 高度な推奨チューニング

この節では、以下のサブシステムのいくつかの属性について、高度な推奨チューニングを説明します。

- 汎用 (6.3.1 項)
- インターネット (6.3.2 項)
- ネットワーク (6.3.3 項)
- ソケット (6.3.4 項)
- 仮想メモリ (6.3.5 項)

これらのチューニングは、十分な物理メモリを備えた、主としてインターネット・サーバとして使用されるシステムに対してのみ有効です。インターネット・サーバ以外で属性の推奨値を使用すると、システム性能が低下することがあります。

インターネット・サーバの構成はシステムごとに異なっており、推奨値はすべての構成に最適なものではないため、属性値の変更は注意深く行う必要があります。属性の説明を読み、使用しているシステムの構成に適した値を確認してください。属性の値を変更しても性能が改善できない場合は、省略時の値に戻してください。

6.3.1 汎用属性の変更

`kmemreserve_percent` 汎用 (generic) サブシステム属性をチューニングすることで、インターネット・サーバの性能を改善できます。この属性は、ページ・サイズ (8 KB) 以下のカーネルのメモリ割り当て用に予約される物理メモリの割合を増加させます。`kmemreserve_percent` の値を大きくすると、システムに大きなネットワークの負荷がかかっているときにドロップされるパケットの数が減るため、ネットワーク・スループットが改善されます。ただし、この値を大きくすると、メモリを消費します。

`kmemreserve_percent` 属性値の増加は、`netstat` コマンドの出力でパケットのドロップが示される場合、または `vmstat -M` コマンドの出力で「fail_nowait」見出しの下にパケットのドロップが示される場合に行ってください。これは、システムに大きなネットワークの負荷がかかっているときに発生します。

省略時の値は 0 (予約する物理メモリの割合は、使用可能なメモリの 0.4% と 256 KB のうち小さい方) です。この値を少しずつ (最大、75 まで) 増やし、`vmstat -M` コマンドで「fail_nowait」見出しの下のエントリ数が 0 になるまで増やしていきます。

6.3.2 インターネット属性の変更

インターネット・サーバの性能は、以下の `inet` インターネット・サブシステム属性をチューニングすることで、改善できます。

- `tcbhashnum` (6.3.2.1 項)
- `inifaddr_hsize` (6.3.2.2 項)
- `tcp_keepinit` (6.3.2.3 項)
- `tcp_rexmit_interval_min` (6.3.2.4 項)
- `tcp_keepalive_default` (6.3.2.5 項)
- `tcp_msl` (6.3.2.6 項)
- `ipport_userreserved_min` (6.3.2.7 項)
- `ipqs` (6.3.2.8 項)
- `ipqmaxlen` (6.3.2.9 項)

カーネル・サブシステム属性の変更については、`sys_attrs_inet(5)` と第 3 章を参照してください。

6.3.2.1 TCP ハッシュ・テーブルの数を増やす

`tcbhashnum` 属性は、TCP ハッシュ・テーブルの数を指定します。ハッシュ・テーブルの数を増やすと負荷が分散されるため、性能が改善できます。ただし、システムの固定メモリが少し増加します。

省略時の値は、1 ハッシュ・テーブルですが、この値は最小値です。負荷の高いインターネット・サーバの SMP システムでは、推奨値は 16 です。最大値は 64 です。

ハッシュ・テーブルの数を増やしたら、ハッシュ・テーブルのサイズを小さくしてください。詳細は、6.2.1.1 項を参照してください。また、この属性の値は、`ipqs` 属性と同じ値にすることをお勧めします。`ipqs` 属性の詳細は、6.3.2.8 項を参照してください。

6.3.2.2 ハッシュ・パケットの数を増やす

`inifaddr_hsize` 属性は、カーネル・インタフェース別名テーブル (`in_ifaddr`) 内のハッシュ・パケットの数を指定します。

システムが、多くのサーバ・ドメイン名 (それぞれが固有の IP アドレスに割り当てられている) をサービスしている場合、受信したパケットを正しいサーバ・アドレスと照合する処理では、IP アドレスを高速に検索するために、ハッシュ・テーブルを使用します。これらの IP アドレスは、通常、`ifconfig alias` コマンド、または `ifconfig aliaslist` コマンドを使って設定されます。テーブル内のハッシュ・パケットの数を増やすと、多数の IP 別名アドレスを使っているシステムの性能が改善されます。

省略時の値は、32 ハッシュ・パケットです。インタフェース IP 別名を使用していなかったり、250 未満の別名しか使用していない大部分のインターネット・サーバでの推奨値は、32 です。500 を超えるインタフェース IP 別名を使用している場合は、推奨値は 512 で、これが最大値です。

最適な性能を得るためには、この属性の値は、最も近い 2 のべき乗値まで切り下げる必要があります。

6.3.2.3 TCP パーシャル接続タイムアウト限界値を変更する

`tcp_keepinit` 属性は、部分的に確立された TCP 接続が、ソケット・リッスン・キューに留まる最長時間 (タイムアウトになるまでの時間) を指定します。この属性の値は、0.5 秒単位です。パーシャル接続は、ソケット・リッスン・キューのスロットを消費し、キューを `SYN_RCVD` 状態の接続で満たします。

省略時の値は、150 単位 (75 秒) です。 `somaxconn_drops` 属性の値が増加することがそれほどなければ、TCP パーシャル接続タイムアウト限界値を変更する必要はありません。イベント・カウンタの詳細は、6.1.5 項を参照してください。

ソケット・キューの限界値として最大値を設定している場合は、通常はこの属性の省略時の値で十分です。 `somaxconn_drops` 属性の値が頻繁に増加し、ソケット・キューの限界値を大きくしてもリッスン・キューがいっぱいになるのを避けることができない場合は、この属性の値を小さくして、パーシャル接続がもっと早くタイムアウトになるようにします。

また、クライアントによってソケット・リッスン・キューに TCP SYN パケットが過度に詰め込まれると、他のユーザがキューにアクセスできなく

なるため、ネットワーク性能が低下します。この問題を避けるためには、ソケット・リッスン・キューの限界値を最大値まで大きくします。それでもシステムが SYN パケットをドロップする場合は、`tcp_keepinit` 属性の値を 30 (15 秒) に下げてください。システムがパケットをドロップしているかどうかを確認するには、イベント・カウンタの `sobacklog_drops` と `somaxconn_drops` の値をモニタリングします。

この属性の値を小さくし過ぎないでください。小さくし過ぎると、遅いネットワーク・パスや多くのパケットがドロップするネットワーク・パスでは、クライアントとの接続処理の打ち切りが早過ぎてしまうためです。この値は、20 単位 (10 秒) より小さくしないでください。

6.3.2.4 TCP 再送の速度を遅くする

`tcp_rexmit_interval_min` 属性は、最初の TCP 再送の時間間隔の最小値を指定します。一部の WAN (wide area network) では、省略時の値では小さ過ぎるため、再送タイムアウトが早く発生し、それによってパケットを重複して送信したり、TCP 混雑回避アルゴリズムが誤って起動されることがあります。

この属性の値を大きくすると、TCP 再送速度が遅くなるため、混雑を減らし性能を改善することができます。

省略時の値は 2 単位 (1 秒) です。すべての接続が、長い再送時間を必要とするわけではありません。通常はこの属性の省略時の値で十分です。しかし、一部の WAN では、省略時の再送時間では小さ過ぎることがあります。

再送が発生しているかどうかを確認するには、`netstat -p tcp` コマンドの出力で「再送されたデータ・パケット (data packets retransmitted)」を調べます。

この属性の値を大きくして、TCP 再送速度を遅くすることができます。この属性は、0.5 秒単位で指定します。

この属性の省略時の値は、TCP アルゴリズムを熟知していないかぎり、変更しないでください。1 単位より小さい値は指定しないでください。

6.3.2.5 TCP 持続機能を有効にする

持続 (keepalive) 機能を使用すると、接続済みのソケット上でメッセージを定期的に転送する際に、アクティブな接続を持続させ、アクティブでない接続

をタイムアウトにすることができます。明確に終了していないソケットは、持続時間が経過すると、抹消されます。持続機能が有効になっていない場合、これらのソケットはシステムをリブートするまで存続し続けます。

アプリケーションでは、`setsockopt` 関数の `SO_KEEPALIVE` オプションを設定することで、ソケットの持続機能を有効にします。省略時の値は 0 (持続機能は無効) です。持続機能を有効にしていないプログラムの場合、またはアプリケーションのソース・コードに手を加えることができない場合に持続機能を有効にしたいときには、この属性を 1 にしてください。この属性を設定した後は、すべての接続で持続機能が有効になります。既存の接続は、以前の持続機能の設定内容のままになります。

この属性を変更した後にシステムをリブートしていないときには、既存のソケットの動作は、アプリケーションを再起動するまで以前の動作のままになります。

持続機能を有効にすると、ソケットごとに以下の TCP オプションも構成できます。

- `tcp_keepidle` 属性で、持続プローブを送信するまでのアイドル時間を 0.5 秒単位で指定できます。この属性の省略時の値は、2 時間です。
- `tcp_keepintvl` 属性で、持続プローブの再送間隔を 0.5 秒単位で指定できます。この属性の省略時の値は、75 秒です。
- `tcp_keepcnt` 属性で、接続を打ち切る前に送信される持続プローブの最大数を指定できます。この属性の省略時の値は、8 プローブです。
- `tcp_keepinit` 属性で、最初の接続要求がタイムアウトになるまでの時間を、0.5 秒単位で指定できます。この属性の省略時の値は、75 秒です。

6.3.2.6 TCP 接続コンテキスト・タイムアウトの頻度を高くする

`tcp_msl` 属性は、TCP セグメントの最大存続期間と `TIME_WAIT` 状態のタイムアウト値を決定します。TCP プロトコルには、MSL (最大セグメント存続期間) という概念があります。TCP 接続が `TIME_WAIT` 状態になると、MSL の 2 倍の時間だけこの状態に留まる必要があります。そうしないと、その後の接続で検出されないデータ・エラーが発生する可能性があります。

この属性の値を小さくすることで、接続終了時に TCP 接続コンテキストを、より早くタイムアウトさせることができます。ただし、そのようにすると、データが壊れる可能性が高くなります。

省略時の値は 60 単位です (これは 30 秒で、TCP 接続が 60 秒間、すなわち MSL の倍の時間だけ、TIME_WAIT 状態に留まることを意味します)。この属性値は、0.5 秒単位で設定します。推奨値は省略時の値です。これ以外の値を指定すると、データが壊れる可能性があります。

TCP の仕様では MSL は 120 秒と規定されていますが、大部分の TCP の実装では、120 より小さい値を使用しています。詳細は、*Internet FAQ Consortium* Web サイトを参照してください。RFC793 については、次の URL を参照してください。

<http://www.faqs.org/rfcs/rfc793.html>

RFC1122 については、以下の URL を参照してください。

<http://www.faqs.org/rfcs/rfc1122.html>

場合によっては、TIME_WAIT 状態の省略時のタイムアウト値が長すぎる場合があります。そのため、この属性の値を小さくすると、接続リソースを省略時の動作より早く解放させることができます。

この属性の値は、ネットワークおよび TCP プロトコルの設計と動作について熟知していないかぎり、小さくしないでください。

6.3.2.7 送信接続ポートの範囲を変更する

TCP または UDP のアプリケーションが送信接続を確立する際に、カーネルは接続ごとに、予約されていないポート番号を動的に割り当てます。

カーネルは、`ipport_userreserved_min` から `ipport_userreserved` までの範囲から、ポート番号を選択します。

システムが特定の範囲のポートを必要とする場合は、この属性の値を変更することができます。

省略時の値は 1024 です。最大値は 65535 です。この属性に、65535 より大きい値や 1024 より小さい値は指定しないでください。

6.3.2.8 IP 受信キューの数を増やす

SMP システムでは、IP 受信キューの数を増やすと、受信キューのロック争奪が減少し、負荷を分散させることができます。ipqs 属性は、IP 受信キューの数を指定します。

省略時の値は、最小値である 1 キューです。負荷が高いインターネット・サーバの SMP システムでは、推奨値は 16 です。最大値は 64 です。

この属性の値は、tcblhashnum 属性と同じ値にすることをお勧めします。tcblhashnum 属性の詳細は、6.2.1.1 項を参照してください。

6.3.2.9 IP 受信キューの最大長を大きくする

ネットワークの負荷が高い場合、IP 受信キューがいっぱいになって、受信パケットがドロップすることがあります。ipqmaxlen 属性は、IP 受信キュー (ipintrq) の最大長をバイト単位で指定します。この長さを超えた受信パケットはドロップします。

システムで受信パケットがドロップしている場合は、ipqmaxlen 属性の値を大きくしたほうがよい可能性があります。受信パケットがドロップしているかどうかを確認するには、以下のように、dbx コマンドを使って、ipintrq カーネル構造体を調べてください。

```
# dbx -k /vmunix
(dbx) print ipintrq
struct {
    ifq_head = (nil)
    ifq_tail = (nil)
    ifq_len = 0
    ifq_maxlen = 512
    ifq_drops = 128
    ifq_slock = struct {
        sl_data = 0
        sl_info = 0
        sl_cpuid = 0
        sl_lifms = 0
    }
}
```

ifq_drops フィールドが 0 でない場合、システムは IP 受信パケットをドロップしています。

省略時の値は、1024 です。最小値が省略時の値です。最大値は 65535 です。システムが受信パケットをドロップしている場合、推奨値は 2048 です。送信キューを制御する ifqmaxlen 属性の値を大きくしたほうがよい場合もあります。ifqmaxlen 属性の詳細は、6.3.3.1 項を参照してください。

6.3.3 ネットワーク属性の変更

以下の `net` ネットワーク・サブシステムの属性をチューニングすることで、インターネット・サーバの性能を改善できます。

- `ifqmaxlen` (6.3.3.1 項)
- `screen_cachedepth` (6.3.3.2 項)
- `screen_cachewidth` (6.3.3.2 項)
- `screen_maxpend` (6.3.3.3 項)

詳細は、`sys_attrs_net(5)` を参照し、カーネル・サブシステム属性の変更については第 3 章を参照してください。

6.3.3.1 送信キューの最大長を大きくする

ネットワークの負荷が高い場合、インタフェースの送信キューがいっぱいになって、送信パケットがドロップされることがあります。`ifqmaxlen` 属性は、パケットがドロップされることなく、ネットワーク・アダプタの送信キューに入れることができるパケットの数を指定します。

`netstat -id` コマンドを使って、送信パケットがドロップされているかどうかを確認することができます。このコマンドの出力で、インタフェースの「Drop」欄が 0 でない場合、システムで送信パケットのドロップが発生しているので、この属性の値を大きくしたほうがよい可能性があります。

省略時の値は、1024 です。最小値は、省略時の値です。最大値は、65535 です。システムが送信パケットをドロップしている場合は、推奨値は 2048 です。

6.3.3.2 スクリーニング・キャッシュのミス減らす

スクリーニング・ルータ、または `screend` 機能を実行しているスクリーニング・ファイアウォールとして動作しているコンピュータで、多くのパス・スルー接続が同時に確立されている場合、スクリーニング・キャッシュのミスが発生している可能性があります。

スクリーニング・キャッシュのミスは、カーネルのスクリーニング・テーブルで、アドレス/ポートの組とプロトコルに従って、キャッシュ・エントリのないパケットを、スクリーニングしようとしたときに発生します。この場合、テーブルによってパケットがキューにつながれ、`screend` デーモンがそ

の packets を調べます。これは、通常、接続の最初の packets の場合と、キャッシュが多くのエントリを保持するには小さすぎる場合に発生します。

スクリーニング・キャッシュがミスしているかどうかを確認するには、次のように、dbx を使ってスクリーニング・キャッシュのヒットとミス調べます。

```
(dbx) p screen_cachemiss
616738
(dbx) p screen_cachehits
11080198
```

ヒットに比べてミスの割合が高い場合は、screen_cachedepth 属性と screen_cachewidth 属性の値を大きくしたほうがよい可能性があります。

screen_cachedepth 属性の省略時の値は 8 で、これは最小値です。スクリーニング・キャッシュのミス率が高い場合、推奨値は 16 で、これは最大値です。

screen_cachewidth 属性の省略時の値は 8 で、これは最小値です。スクリーニング・キャッシュのミス率が高い場合、推奨値は 2048 で、これは最大値です。

screen_cachedepth を大きくする前に、まず screen_cachewidth を大きくすることをお勧めします。また、これらの属性をチューニングしても、必ずしもスクリーニング・キャッシュのミスは 0 にならないことに注意してください。変更した内容を有効にするには、リブートが必要です。

これらの値を大きくするとメモリの消費量が少し増えます。

6.3.3.3 スクリーニング・バッファのドロップを減らす

スクリーニング・ルータ、または screend 機能を実行しているスクリーニング・ファイアウォールとして動作しているコンピュータで、ネットワークの負荷が高い場合、スクリーニング・バッファのドロップが発生している可能性があります。

screenstat コマンドを使って、現在の状態を表示してください。例を以下に示します。

```
# /usr/sbin/screenstat
total packets screened: 11696910
total accepted: 11470734
total rejected: 225453
packets dropped:
    because buffer was full:      34723
    because user was out of sync: 0
    because too old:              0
total dropped: 34723
```

「バッファがいっぱいのため (because buffer was full)」にドロップされているパケットの数が多い場合、`screen_maxpend` 属性の値を大きくしたほうがよい可能性があります。省略時の値は 32 で、これは最小値です。スクリーニング・バッファがいっぱいになったことを示す値が大きい場合、推奨値は 8192 です。最大値は 16384 です。

この値を大きくするとメモリの消費量が少し増えます。この属性を変更するには、システムのリブートが必要です。

6.3.4 ソケット属性の変更

インターネット・サーバの性能は、`sb_max` ソケット (socket) サブシステム属性をチューニングすることで、改善できます。また、`socket` サブシステムの属性の `sobacklog_hiwat`、`sobacklog_drops`、および `somaxconn_drops` は、ソケット・リッスン・キューに関するイベントを追跡します。これらの値をモニタリングすることにより、キューがオーバーフローしているかどうかを調べることができます。これらの属性の詳細は、6.1.5 項を参照してください。

`sb_max` 属性は、ソケット・バッファの最大サイズを指定します。ソケット・バッファの最大サイズを大きくすると、バッファ・サイズが大きいほうが効率が良くなるアプリケーションを使っている場合、性能が改善されます。

省略時の値は 1048576 バイトです。省略時の値より大きなソケット・バッファを必要とするアプリケーションでは、この属性の値を大きくしてください。

カーネル・サブシステム属性の変更については、`sys_attrs(5)` と第 3 章を参照してください。

6.3.5 仮想メモリ属性の変更

インターネット・サーバの性能は、`vm` 仮想メモリ・サブシステム属性 `ubc_maxpercent`、`ubc_minpercent` および `ubc_borrowpercent` をチューニングすることで、改善できます。

負荷の高いインターネット・サーバでは、通常、仮想メモリの消費量は平均的で、多数のファイルが使用されます。プロセスと、ファイル・システムのデータをキャッシュする UBC (Unified Buffer Cache) は、カーネルが固定していない物理メモリを共有します。

UBC に割り当てるメモリが多すぎると、過度のページングとスワッピングが発生し、システム全体の性能が低下します。逆に、UBC に割り当てるメモリが少なすぎると、ファイル・システムの性能が低下します。

`ubc_minpercent` 属性は、UBC だけが利用できるメモリの最小割合を指定します。残りのメモリは、プロセスと共有されます。

`ubc_maxpercent` 属性は、UBC が利用できるメモリの最大割合を指定します。`ubc_borrowpercent` 属性は、UBC の借用しきい値を指定します。

`ubc_borrowpercent` 属性の値と `ubc_maxpercent` 属性の値の間で、UBC に割り当てられているメモリはプロセスから借りていると見なされます。ページングが始まると、これらの借りているページが最初に再生され、UBC に割り当てられたメモリは `ubc_borrowpercent` 属性の値になるまで減らされます。

`ubc_minpercent` の省略時の値は 10% です。`ubc_maxpercent` の省略時の値は 100% です。`ubc_borrowpercent` の省略時の値は 20% です。通常のインターネット・サーバでは、各属性の省略時の値で十分です。また、ディスクに対してファイル・システム入出力が頻繁に実行され、システムに十分な空きページがある場合も、省略時の値を使用してください。

`vmstat` コマンドを使って、空きページの数などの仮想メモリ情報を表示してください。

空きページの数が少ない場合は、UBC が利用できるメモリを減らし、プロセスが利用できるメモリを増やしたほうがよい可能性があります。UBC ミスの回数が増えたとしても、プロセスのワーキング・セットをメモリ内に維持する必要があります。

`ubc_maxpercent` 属性の値を、省略時の値から 10% 単位で減らしてみてください。

`ubc_borrowpercent` 属性の値を減らして借用メモリのしきい値を減らすと、メモリが少ない場合にシステムの応答時間が改善されます。ただし、これにより、UBC の性能は低下します。

アプリケーション性能の管理

アプリケーションの性能を改善することで、Tru64 UNIX 全体の性能が改善できます。この章では、アプリケーションの性能のガイドラインについて説明します。詳細は、表 7-1を参照してください。

7.1 アプリケーションの性能改善

良くできたアプリケーションは、CPU、メモリ、および入出力リソースを効率的に使用します。表 7-1は、アプリケーションの性能を改善する際のガイドラインを示しています。

表 7-1: アプリケーションの性能改善のガイドライン

ガイドライン	性能上の利点	欠点
オペレーティング・システムの最新のパッチをインストールする (7.1.1 項)。	最新の最適化が反映される。	なし
最新バージョンのコンパイラを使用する (7.1.2 項)。	最新の最適化が反映される。	なし
並列処理を使用する (7.1.3 項)。	SMP の性能が改善される。	なし
アプリケーションを最適化する (7.1.4 項)。	効率的なコードが生成される。	なし
シェアード・ライブラリを使用する (7.1.5 項)。	メモリが解放される。	実行時間が長くなることがある。
アプリケーションが必要とするメモリ量を減らす (7.1.6 項)。	メモリが解放される。	プログラムの実行状態が最適でなくなることがある。
リアルタイム・プログラムの初期化処理でメモリをロックする (7.1.7 項)。	必要に応じてメモリがロックまたはアンロックされる。	プロセスおよび UBC に使用できるメモリが減る。

以降の項では、上記の改善のガイドラインについて詳しく説明します。

7.1.1 オペレーティング・システムの最新のパッチをインストールする

常にオペレーティング・システムの最新のパッチをインストールします。パッチには、性能改善が含まれていることがよくあります。

/etc/motd ファイルを調べて、適用済みのパッチを把握してください。パッチのインストールについての詳細は、カスタマ・サービスにお問い合わせください。

7.1.2 最新バージョンのコンパイラを使用する

常に最新バージョンのコンパイラを使用して、アプリケーション・プログラムを構築します。通常、新しいバージョンでは、より高度な最適化が行われます。

システム上のソフトウェアをチェックして、最新バージョンのコンパイラを使用していることを確認してください。

7.1.3 並列処理を使用する

並列処理を強化するために、Fortran や C で開発しているアプリケーション開発者は、KAP (Kuch & Associates Preprocessor) の使用を検討してください。これを使用すると、SMP の性能が大きく改善される可能性があります。KAP の詳細については、『プログラミング・ガイド』を参照してください。

7.1.4 アプリケーションを最適化する

アプリケーション・プログラムの最適化では、構築プロセスの変更やソース・コードの変更が必要になることがあります。コンパイラおよびリンカのさまざまな最適化レベルを用いて、より効率的なユーザ・コードを生成できます。最適化についての詳細は、『プログラミング・ガイド』を参照してください。

32 ビットのシステムから Tru64 UNIX へアプリケーションを移植する場合と、新しいアプリケーションを開発する場合のどちらでも、アプリケーションのデバッグとテストを十分に行うまでは最適化しないでください。C で記述されたアプリケーションを移植する場合は、lint コマンドに -Q オプションを指定して使用するか、C コンパイラに -check オプションを指定してコンパイルすると、解決する必要がある移植上の問題があるか調べることができます。

7.1.5 シェアード・ライブラリを使用する

シェアード・ライブラリを使用すると、メモリおよびディスク・スペースの必要量が減少します。複数のプログラムが1つのシェアード・ライブラリにリンクされている場合、各プロセスが使用する物理メモリの容量は大幅に削減されます。

ただし、シェアード・ライブラリを使用すると、最初の実行での実行速度が、静的ライブラリを使用した場合よりも遅くなることがあります。

7.1.6 アプリケーションが必要とするメモリ量を減らす

アプリケーションのメモリ使用量を削減することで、他のプロセスやファイル・システムのキャッシュのためのメモリ・リソースを増やせることがあります。コーディングの際に以下の項目を考慮すると、アプリケーションのメモリ使用量を削減できます。

- アプリケーションのインストレーション処理で示されるガイドラインに基づいてアプリケーションを構成し、チューニングします。たとえば、アプリケーションに必要な可変メモリの削減、並列/多重処理属性の設定、共用グローバル領域とプライベート・キャッシュのサイズの変更、オープン/マップされたファイルの最大数の設定を行います。
- アプリケーション内では、`read` 関数や `write` 関数の代わりに `mmap` 関数を使用することができます。`read` および `write` システム・コールはバッファ・メモリのページと UBC メモリのページを必要としますが、`mmap` は1ページのメモリしか必要としません。
- 使用頻度の高いデータ構造体の間の、データ・キャッシュの衝突を探します。これは、メモリ内に割り当てられている2つのデータ構造体の間の距離が、1次(内部)データ・キャッシュのサイズと等しい場合に発生します。データ構造体が小さい場合は、メモリ内に続けて配置することで衝突を防止できます。このようにするためには、`malloc` を何度も呼び出さないで1度だけ呼び出すようにします。
- アプリケーションが、大量のデータを短い時間だけ使用する場合、メモリを静的に宣言せずに、`malloc` 関数を使ってデータを動的に割り当てます。動的に割り当てたメモリの使用を終えたときには、メモリを解放して、プログラム内でその後に発生する他のデータ構造体で使えるようにします。メモリ・リソースが限られている場合は、デー

タを動的に割り当てると、アプリケーションのメモリ使用量が減り、実質的に性能が改善されます。

- アプリケーションが `malloc` 関数を頻繁に用いている場合は、関数の制御変数を用いてメモリ割り当てをチューニングすると、処理速度が改善され、メモリ使用量を削減できます。メモリ割り当てのチューニングについての詳細は、`malloc(3)` を参照してください。
- アプリケーションが 32 ビットのアドレス空間に納まっていて、多くのポインタを含む構造体を用いて動的メモリを大量に割り当てている場合は、`-xtaso` オプションを使用するとメモリ使用量を削減できます。`-xtaso` オプションは、すべてのバージョンの C コンパイラ (`-newc`, `-migrate`, `-oldc` バージョン) でサポートされています。`-xtaso` オプションを使用するには、ポインタの割り当てサイズを制御する C 言語のプリグマでソース・コードを変更します。詳細は、`cc(1)` を参照してください。

プロセスのメモリ割り当てについての詳細は、『プログラミング・ガイド』を参照してください。

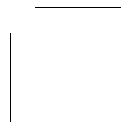
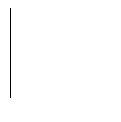
7.1.7 メモリ・ロックを制御する

リアルタイム・アプリケーションの開発者は、メモリ・ロックをプログラムの初期化処理に必須の処理と考えなければなりません。リアルタイム・アプリケーションの多くは、実行中はロックされたままですが、一部のアプリケーションでは、実行状況に応じてメモリをロック/アンロックした方が良いこともあります。メモリ・ロック関数を用いると、関数呼び出しの時点からアプリケーションの存在期間全体を通じてプロセス全体をロックできます。ロックされたメモリのページはページングの対象にできなくなり、プロセスはスワップ・アウトできなくなります。

メモリ・ロックは、プロセスのアドレス空間に適用されます。プロセスのアドレス空間にマップされたページだけをメモリにロックできます。プロセスが終了すると、ページはアドレス空間から削除され、ロックは解除されます。

プロセスのアドレス空間全体をロックするには、`mlockall` 関数を使用します。ロックされたメモリは、プロセスが終了するか、アプリケーションが `munlockall` 関数を呼び出すまでロックされたままです。`ps` コマンドを使用すると、プロセスがメモリにロックされてスワップ・アウトできない状態かどうかわかります。12.3.2 項を参照してください。

メモリ・ロックは、子プロセス (fork 関数で生成) には継承されません。また、プロセスに対応するすべてのメモリ・ロックは、`exec` 関数を呼び出したとき、またはプロセスが終了したときにアンロックされます。詳細は、『*Guide to Realtime Programming*』と `mlockall(3)` を参照してください。



Part 3

コンポーネント別のチューニング



システム・リソース割り当ての管理

Tru64 UNIX オペレーティング・システムは、ブート時にリソースの限界値を設定します。この限界値によって、システム・テーブル、仮想アドレス空間、および他のシステム・リソースのサイズが制御されます。

大半の構成では、省略時のシステム・リソース限界値で十分です。ただし、システムのメモリ容量が大きい場合や、大量のリソースを必要とするプログラムやメモリ・サイズの大きいアプリケーションを実行する場合は、サブシステム属性を変更して、システムの限界値を大きくしなければならいこともあります。

この章では、システム・リソースの割り当てと、次のシステム・ワイドの限界値を大きくする方法について説明します。

- プロセスの限界値 (8.1 節)
- プログラム・サイズの限界値 (8.2 節)
- アドレス空間の限界値 (8.3 節)
- プロセス間通信 (IPC) の限界値 (8.4 節)
- オープン・ファイルの限界値 (8.5 節)
- Aurema ARMTech Suite (8.6 節)

システム・ワイドの限界値を変更する代わりに、`setrlimit` 関数を使用して、特定のプロセスとその子プロセスが使用するシステム・リソースを制御できます。詳細は、`setrlimit(2)` を参照してください。

8.1 プロセスの限界値のチューニング

Tru64 UNIX は、大半の構成に適した、プロセスの限界値を使用します。ただし、メモリを多用するアプリケーションの場合や、大規模メモリ (LVM) システムやインターネット・サーバ (Web サーバ、プロキシ・サーバ、ファイアウォール・サーバ、ゲートウェイ・サーバなど) がある場合は、プロセスの限界値を大きくしなければならないこともあります。プロセスの限界値を大

きくするとシステムの固定メモリの量も増えるため、限界値を大きくするのは、システムが必要とするリソースが不足する場合だけにしてください。

以降の項では、限界値を大きくする方法について説明します。

- システム・テーブルとデータ構造体 (8.1.1 項)
- プロセスの最大数 (8.1.2 項)
- スレッドの最大数 (8.1.3 項)

proc サブシステム属性の詳細は、`sys_attrs_proc(5)`を参照してください。

8.1.1 システム・テーブルとデータ構造体を大きくする

システムのアルゴリズムでは、proc サブシステムの `maxusers` 属性によって、システム・プロセス・テーブルなどの各種のシステム・データ構造体およびシステム・テーブルのサイズが決定されます。このテーブルによって、同時に実行できるアクティブ・プロセスの数が決まります。

注意

`maxusers` 属性の値は、システムの限界値を設定するサブシステム属性 (`max_proc_per_user` , `max_threads_per_user` , `min_free_vnodes` , および `name_cache_hash_size` 属性) の省略時の値を設定するために使用されます。

性能上の利点と欠点

`maxusers` の値を大きくすると、プロセスが利用できるシステム・リソースが増えます。ただし、ユーザ用のリソースを増やすと、固定メモリの量も増えます。

`maxusers` 属性は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

大規模メモリ・システムやインターネット・サーバがある場合や、システムがリソース不足になっている場合は、`maxusers` 属性の値を大きくします。リソース不足は、`No more processes` , `Out of processes` , または `pid table is full` というメッセージで示されます。

推奨値

省略時の `maxusers` 属性の値は、システムのメモリ容量によって異なります。表 8-1 に、各メモリ容量での `maxusers` 属性の省略時の値を示します。

表 8-1: `maxusers` 属性の省略時の値

メモリ・サイズ	<code>maxusers</code> の値
256 MB 以下	128
257 ~ 512 MB	256
513 ~ 1024 MB	512
1025 ~ 2048 MB	1024
2049 ~ 4096 MB	2048
4097 MB 以上	2048

`maxusers` 属性の最適値を判断するには、性能が改善されるまで、省略時の値を 2 倍にしていきます。インターネット・サーバがある場合は、`maxusers` 属性の値を 2048 に増やします。`maxusers` 属性の値は、2048 より大きくしないようにしてください。

`maxusers` の値を大きくする場合は、`max_vnodes` 属性の値もそれに比例して大きくしてください (8.5.1 項を参照)。

`maxusers` 属性は、省略時の値より小さくしないでください。

カーネル・サブシステム属性の変更については、第 3 章 を参照してください。

8.1.2 プロセスの最大数を増やす

`proc` サブシステムの `max_proc_per_user` 属性は、一時点で各ユーザ (スーパーユーザを除く) に割り当てることができるプロセスの最大数を指定します。

性能上の利点と欠点

`max_proc_per_user` の値を大きくすると、プロセスが利用できるシステム・リソースが増えます。

チューニングするかどうかの判断

システムがプロセス不足になっている場合や、大規模メモリ (VLM) システムやインターネット・サーバがある場合は、`max_proc_per_user` 属性の値を大きくしてください。

`max_proc_per_user` 属性の値を変更したときには、システムのリブートが必要です。

推奨値

`max_proc_per_user` 属性の省略時の値は、`maxusers` 属性によって決まります。プロセスの最大数を大きくする場合は、`maxusers` 属性の値を大きくします (8.1.1 項 参照)。または、`max_proc_per_user` 属性の値に、システム上で同時に実行されるプロセスの最大数以上の値を指定することもできます。Web サーバがある場合は、このプロセスには CGI プロセスも含まれます。

インターネット・サーバがある場合は、`max_proc_per_user` 属性の値を 512 に増やします。

`max_proc_per_user` 属性の値に 0 を指定すると、プロセス数は制限されません。

カーネル・サブシステム属性の変更については、第 3 章を参照してください。

8.1.3 スレッドの最大数を増やす

`proc` サブシステムの `max_threads_per_user` 属性は、一時点で各ユーザ (スーパユーザを除く) に割り当てることができるスレッドの最大数を指定します。

性能上の利点と欠点

`max_threads_per_user` の値を大きくすると、プロセスが利用できるシステム・リソースが増えます。

`max_proc_per_user` 属性を変更したときには、システムのリブートが必要です。

チューニングするかどうかの判断

システムがスレッド不足になっている場合や、VLM システムやインターネット・サーバがある場合は、`max_threads_per_user` 属性の値を大きくしてください。

推奨値

`max_threads_per_user` 属性の省略時の値は、`maxusers` 属性によって決まります。スレッドの最大数を大きくしたいときには、`maxusers` 属性の値を変更します (8.1.1 項 参照)。または、`max_threads_per_user` 属性の値に、システム上で同時に実行されるスレッドの最大数以上の値を指定することもできます。たとえば、`max_threads_per_user` 属性の値を 512 に増やすことができます。

メモリが十分にある負荷の高いサーバや、インターネット・サーバでは、`max_threads_per_user` 属性の値を 4096 に増やします。

`max_threads_per_user` 属性の値が 0 の場合、スレッド数の限界値がなくなります。

`max_threads_per_user` 属性の値に 0 を指定すると、スレッド数は制限されません。

カーネル・サブシステム属性の変更については、第 3 章 を参照してください。

8.2 プログラム・サイズの限界値のチューニング

大規模なアプリケーションを実行する場合は、プログラム・サイズの限界値を制御する `proc` サブシステムの属性値を大きくしなければならないことがあります。大規模なプログラムや大規模メモリ・プロセスは、この属性の省略時の値を変更しないと動作しないことがあります。

以降の項では、次の作業を実行する方法について説明します。

- ユーザ・プロセス・スタックの最大サイズを大きくする (8.2.1 項)。
- ユーザ・プロセスのデータ・セグメントの最大サイズを大きくする (8.2.2 項)。

`proc` サブシステム属性についての詳細は、`sys_attrs_proc(5)` を参照してください。

8.2.1 ユーザ・プロセス・スタックのサイズを大きくする

proc サブシステムの `per_proc_stack_size` 属性および `max_per_proc_stack_size` 属性は、ユーザ・プロセス・スタックの省略時のサイズと最大サイズを指定します。大規模なプログラムや大規模メモリ・プロセスは、これらの属性の省略時の値を変更しないと動作しないことがあります。

性能上の利点と欠点

ユーザ・プロセス・スタックの省略時のサイズと最大サイズを大きくすると、大規模なアプリケーションを実行できるようになります。

`per_proc_stack_size` 属性と `max_per_proc_stack_size` 属性を変更したときには、システムのリブートが必要です。

チューニングするかどうかの判断

大規模なプログラムまたは大規模メモリ・プロセスを実行している場合や、`Cannot grow stack` というメッセージが表示された場合は、ユーザ・プロセス・スタックの省略時のサイズと最大サイズを大きくします。

推奨値

`per_proc_stack_size` 属性の省略時の値は、8,388,608 バイトです。
`max_per_proc_stack_size` 属性の省略時の値は、33,554,432 バイトです。アドレス空間の限界値よりも十分に小さい値を選択してください。詳細は 8.3 節を参照してください。

カーネル・サブシステム属性の変更については、第 3 章を参照してください。

8.2.2 ユーザ・プロセスのデータ・セグメント・サイズを大きくする

proc サブシステムの `per_proc_data_size` 属性および `max_per_proc_data_size` 属性は、ユーザ・プロセスのデータ・セグメント・サイズの省略時の値および最大値を指定します。大規模なプログラムや大規模メモリ・プロセスでは、これらの属性の省略時の値を変更しないと動作しないことがあります。

性能上の利点と欠点

ユーザ・プロセスのデータ・セグメント・サイズの省略時の値および最大値を大きくすると、大規模なアプリケーションを実行できるようになります。

`per_proc_data_size` 属性と `max_per_proc_data_size` 属性を変更したときには、システムのリブートが必要です。

チューニングするかどうかの判断

大規模なプログラムや大規模メモリ・プロセスを実行する場合、`Out of process memory` というメッセージが表示された場合、またはシステムがインターネット・サーバの場合には、`per_proc_data_size` 属性および `max_per_proc_data_size` 属性の値を大きくしなければならないことがあります。

推奨値

`per_proc_data_size` の省略時の値は、134,217,728 バイトです。
`max_per_proc_data_size` の省略時の値は、1 GB (1,073,741,824 バイト) です。アドレス空間の限界値よりも十分に小さい値を選択してください。8.3 節を参照してください。

インターネット・サーバの場合は、`max_per_proc_data_size` 属性の値を大きくして、10 GB (10,737,418,240 バイト) にします。

カーネル・サブシステム属性の変更については、第3章を参照してください。

8.3 アドレス空間の限界値のチューニング

`proc` サブシステムの属性 `per_proc_address_space` および `max_per_proc_address_space` は、ユーザ・プロセスのアドレス空間の大きさ (有効な仮想領域の数) について、省略時の値および最大値を指定します。

性能上の利点と欠点

アドレス空間の限界値を大きくすると、大規模なプログラムを実行できるようになり、メモリを多用するアプリケーションの性能が向上します。ただし、メモリの使用量が少し増加します。

`per_proc_address_space` 属性と `max_per_proc_address_space` 属性を変更したときには、システムのリブートが必要です。

チューニングするかどうかの判断

メモリを多用するプロセスを実行する場合や、システムがインターネット・サーバの場合は、アドレス空間の限界値を大きくしてください。

推奨値

`per_proc_address_space` 属性および `max_per_proc_address_space` 属性の省略時の値は、4 GB (4,294,967,296 バイト) です。

インターネット・サーバの場合は、`max_per_proc_address_space` 属性の値を大きくして、10 GB (10,737,418,240 バイト) にします。

カーネル属性の変更については、第 3 章を参照してください。

8.4 プロセス間通信の限界値のチューニング

プロセス間通信 (IPC) とは、複数のプロセス間で情報を交換することです。IPC の例としては、メッセージ、共用メモリ、セマフォ、パイプ、シグナル、プロセスのトレース、およびネットワークを通して他プロセスと通信するプロセスがあります。

以降の項では、以下の方法について説明します。

- System V メッセージの最大サイズを大きくする (8.4.1 項)
- System V メッセージ・キューの最大サイズを大きくする (8.4.2 項)
- System V キューのメッセージの最大サイズを大きくする (8.4.3 項)
- System V 共用メモリ領域の最大サイズを大きくする (8.4.4 項)
- プロセスにアタッチできる共用メモリ領域の最大数を増やす (8.4.5 項)
- 共用ページ・テーブルの共用を変更する (8.4.6 項)

Tru64 UNIX オペレーティング・システムには、次のプロセス間通信機能があります。

- パイプ — パイプについては、『*Guide to Realtime Programming*』を参照してください。
- シグナル — 『*Guide to Realtime Programming*』を参照してください。
- ソケット — 『ネットワーク・プログラミング・ガイド』を参照してください。

- ストリーム — 『*Programmer's Guide: STREAMS*』を参照してください。
- XTI (X/Open Transport Interface) — 『ネットワーク・プログラミング・ガイド』を参照してください。

メモリを多用するプロセスを実行している場合は、一部の `ipc` サブシステム属性の値を大きくしなければならないことがあります。

表 8-2 に、IPC の限界値を大きくするためのガイドラインと、性能上の利点および欠点のリストを示します。

表 8-2: IPC の限界値のチューニング・ガイドライン

ガイドライン	性能上の利点	欠点
System V メッセージの最大サイズを大きくする (8.4.1 項)。	System V のメッセージ・サイズを大きくして効果のあるアプリケーションでは、性能が改善される可能性がある。	メモリを少し消費する。
System V のメッセージ・キューの最大バイト数を大きくする (8.4.2 項)。	System V のメッセージ・キューを大きくして効果のあるアプリケーションでは、性能が改善される可能性がある。	メモリを少し消費する。
System V キュー上の処理待ちメッセージの最大数を増やす (8.4.3 項)。	処理待ちメッセージの数を多くできて効果のあるアプリケーションでは、性能が改善される可能性がある。	メモリを少し消費する。
System V の共用メモリ領域の最大サイズを大きくする (8.4.4 項)。	System V の共用メモリ領域を大きくして効果のある、メモリを多用するアプリケーションでは、性能が改善される可能性がある。	メモリを消費する。
プロセスにアタッチできる共用メモリ領域の最大数を増やす (8.4.5 項)。	多数の共用メモリ領域にアタッチしているアプリケーションの性能が改善されることがある。	メモリを消費することがある。
共用ページ・テーブルの限界値を変更する (8.4.6 項)。	メモリを多用するシステムや、VLM システムが、効率的に動作するようになる。	メモリを消費することがある。

以降の項では、一部の System V 属性のチューニング方法について説明します。その他の IPC サブシステム属性については、`sys_attrs_ipc(5)` を参照してください。

8.4.1 System V メッセージの最大サイズを大きくする

ipc サブシステムの `msg_max` 属性は、アプリケーションが受信できる System V メッセージの最大サイズを指定します。

性能上の利点と欠点

`msg_max` 属性の値を大きくすると、System V のメッセージ・サイズを省略時の値よりも大きくして効果のあるアプリケーションでは、性能が改善される可能性があります。ただし、この値を大きくすると、メモリを消費します。

`msg_max` 属性を変更したときには、システムのリブートが必要です。

チューニングするかどうかの判断

System V の省略時のメッセージ最大サイズを 8192 バイトより大きい値にして効果のあるアプリケーションの場合は、`msg_max` 属性の値を大きくします。

推奨値

`msg_max` 属性の省略時の値は、8192 バイト (1 ページ) です。

カーネル・サブシステム属性の変更については、第 3 章を参照してください。

8.4.2 System V のメッセージ・キューの最大サイズを大きくする

ipc サブシステムの `msg_mnb` 属性は、System V のメッセージ・キューに保持できる最大バイト数を指定します。

キュー内のバイト数が `msg_mnb` 属性で指定された限界値を超えている場合、プロセスはそのキューへメッセージを送信できません。この限界値に達した場合、プロセスはスリープし、この状態が解除されるまで待ちます。

性能上の利点と欠点

`msg_mnb` 属性の値を大きくすると、System V のメッセージ・キューを省略時の値よりも大きくして効果のあるアプリケーションでは、性能が改善される可能性があります。ただし、この値を大きくすると、メモリを消費します。

`msg_mnb` 属性を変更したときには、システムのリブートが必要です。

チューニングするかどうかの判断

IPC 機能の使用状況は、`ipcs -a` コマンド (`ipcs(1)` を参照) で確認できます。キュー内の現在のバイト数とメッセージ・ヘッダを参照すると、待ちを減らすために System V のメッセージ・キューをチューニングする必要があるかどうかを判断できます。

推奨値

`msg_mnb` 属性の省略時の値は、16,384 バイトです。

カーネル・サブシステム属性の変更については、第 3 章を参照してください。

8.4.3 System V のキュー上のメッセージ最大数を増やす

`ipc` サブシステムの `msg_tql` 属性は、System V のメッセージ・キュー上に存在できるメッセージの最大数 (システム内に存在できる処理待ちメッセージの総数) を指定します。

性能上の利点と欠点

`msg_tql` 属性の値を大きくすると、処理待ちメッセージの数を省略時の値よりも大きくして効果のあるアプリケーションの性能が改善される可能性があります。ただし、この属性の値を大きくすると、メモリを消費します。

`msg_tql` 属性の値を変更したときには、システムのリブートが必要です。

チューニングするかどうかの判断

処理待ちメッセージの最大数を 40 よりも大きい値にして効果のあるアプリケーションの場合は、`msg_tql` の値を大きくしてください。

IPC 機能の使用状況は、`ipcs -a` コマンド (`ipcs(1)` を参照) で確認できます。キュー内の現在のバイト数とメッセージ・ヘッダを参照すると、待ちを減らすために System V のメッセージ・キューをチューニングする必要があるかどうかを判断できます。

推奨値

`msg_tql` の省略時の値は 40 です。

カーネル・サブシステム属性の変更については、第 3 章を参照してください。

8.4.4 System V の共用メモリ領域の最大サイズを大きくする

ipc サブシステムの `shm_max` 属性は、System V の共用メモリ領域 1 個の最大サイズを指定します。

性能上の利点と欠点

`shm_max` 属性の値を大きくすると、System V の共用メモリ領域を大きくして効果のある、メモリを多用するアプリケーションの性能が改善される可能性があります。ただし、`shm_max` 属性の値を大きくすると、メモリの使用量が増加します。

`shm_max` 属性を変更したときには、システムのリブートが必要です。

チューニングするかどうかの判断

メモリを多用するアプリケーションで、System V の共用メモリ領域を省略時の値の 512 ページよりも大きくして効果がある場合は、`shm_max` 属性の値を大きくしてください。

推奨値

`shm_max` 属性の省略時の値は、4,194,304 バイト (512 ページ) です。

カーネル・サブシステム属性の変更については、第 3 章を参照してください。

8.4.5 1 つのプロセスにアタッチできる共用メモリ領域の最大数を増やす

ipc サブシステムの `shm_seg` 属性は、1 つのプロセスに同時にアタッチできる System V 共用メモリ領域の最大数を指定します。

設計段階で、共用メモリの代わりにスレッドを使用すると性能が向上するかどうかを考慮してください。

性能上の利点と欠点

1 つのプロセスにアタッチできる System V 共用メモリ領域の数を増やすと、多数の共用メモリ領域をアタッチするアプリケーションの性能が改善される可能性があります。

`shm_seg` 属性の値を大きくすると、プロセスが多数の共用メモリ領域をアタッチしたときに、メモリを消費します。

`shm_seg` 属性の値を変更したときには、システムのリブートが必要です。

チューニングするかどうかの判断

プロセスが限界値を超えて共用メモリ領域をアタッチしようとした (`shmat` 関数が `EMFILE` エラーを返却する) 場合は、`shm_seg` 属性の値を大きくしてください。

推奨値

`shm_seg` の省略時の値は 32 です。

カーネル・サブシステム属性の変更については、第 3 章を参照してください。

8.4.6 共用ページ・テーブルの共用を変更する

`shmget` 関数で作成する System V 共用メモリ・セグメントのサイズが `ipc` サブシステムの `ssm_threshold` 属性の値以上の場合、第 3 レベルのページ・テーブル共用が発生します。

性能上の利点と欠点

共用ページ・テーブルの限界値を大きくすると、共用ページ・テーブルの使用は、8 MB を超える共用メモリ・セグメントを作成するアプリケーションに制限されます。ただし、この値を大きくすると、メモリの使用量が増えます。

アプリケーションが共用ページ・テーブルを使用できない場合は、ページ・テーブル共用を無効にできます。

`ssm_threshold` 属性は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

ページ・テーブル共用の使用を、8 MB を超える共用メモリ・セグメントを作成するアプリケーションに制限したい場合は、`ssm_threshold` 属性の値を大きくしてください。

境界合わせの制限のためにアプリケーションが共用ページ・テーブルを使用できない場合は、ページ・テーブル共用を無効にできます。

推奨値

`ssm_threshold` 属性の省略時の値は、8 MB (8,388,608 バイト) です。

`ssm_threshold` 属性に 0 を設定すると、セグメント化共用メモリの使用が無効になります。

カーネル・サブシステム属性の変更については、第 3 章を参照してください。

8.5 オープン・ファイルの限界値のチューニング

以降の項では、次の作業の実行方法について説明します。

- オープン・ファイルの最大数を増やす (8.5.1 項)。
- オープン・ファイル記述子の最大数を増やす (8.5.2 項)。

`proc` サブシステム属性の詳細は、`sys_attrs_proc(5)` を参照してください。

8.5.1 オープン・ファイルの最大数を増やす

オープン・ファイル用のカーネル・データ構造体のことを **vnode** といいます。この構造体は、すべてのファイル・システムで使用されます。vnode の数で、オープン・ファイルの数が決まります。vnode の割り当てと割り当て解除は、オペレーティング・システムが動的に行います。

`vfs` サブシステムの `max_vnodes` 属性は、常にシステム内のオープン・ファイルの最大数以上である、vnode キャッシュのサイズを指定します。オープン・ファイルの最大数を増やすには、この属性を省略時の値より大きくしてください。この作業は、`proc` サブシステムの `maxusers` 属性値を大きくしても実行できます。詳細は 8.1 節を参照してください。

性能上の利点と欠点

vnode キャッシュのサイズを大きくすると、多数のオープン・ファイルを必要とするアプリケーションの性能が改善されます。ただし、このサイズを大きくすると、メモリを消費します。

`max_vnodes` 属性の値は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

アプリケーションが多数のオープン・ファイルを必要とする場合や、vnode 不足を示すメッセージが表示された場合は、`max_vnodes` 属性を省略時の値より大きくします。

推奨値

`max_vnodes` 属性の省略時の値は、メモリの 5 パーセントです。

カーネル・サブシステム属性の変更については、第 3 章を参照してください。

8.5.2 オープン・ファイル記述子の最大数を増やす

すべてのプロセスまたは特定のプロセスについて、オープン・ファイル記述子の最大数を増やすことができます。proc サブシステムの `open_max_soft` 属性および `open_max_hard` 属性は、プロセスごとのオープン・ファイル記述子のシステム・ワイドの最大数を制御します。

オープン・ファイル記述子の限界値により、割り当ての暴走 (エラーのために抜け出せないループ内での割り当てなど) で、利用可能なファイル記述子がすべて使われてしまうのを防ぐことができます。プロセスが `open_max_soft` 限界値に達した場合、警告メッセージが表示されます。プロセスが `open_max_hard` 限界値に達した場合、プロセスは停止させられます。

性能上の利点と欠点

多数のファイルをオープンするアプリケーションの性能が改善されます。

`open_max_soft` 属性および `open_max_hard` 属性を変更したときには、システムのリブートが必要です。

チューニングするかどうかの判断

アプリケーションが多数のオープン・ファイルを必要とする場合は、`open_max_soft` 属性および `open_max_hard` 属性の値を大きくすることによって、オープン・ファイル記述子の限界値を大きくできます。ただし、オープン・ファイル記述子の限界値を大きくすると、割り当ての暴走の原因となることがあります。

推奨値

`open_max_soft` 属性および `open_max_hard` 属性の省略時の値は、4,096 です。この値は、`/etc/sysconfigtab` ファイルに設定できる、システム・ワイドな値の最大値です。

多数のファイルをオープンするアプリケーションがある場合は、システム・ワイドな限界値を大きくするのではなく、そのアプリケーションのオープン・

ファイル限界値だけを大きくすることができます。特定のアプリケーションで拡張 (64 KB) ファイル記述子を使用可能にするには、次の手順に従います。

1. `setsysinfo` システム・コールの `SSI_FD_NEWMAX` 操作パラメータに 1 を設定します。これにより `utask` ビットが設定され、最大 65,536 (64 KB) のオープン・ファイル記述子が使用可能になり、プロセスのハード・ファイル限界値が 64 KB に引き上げられます。この設定は、すべての子プロセスに引き継がれます。詳細は、`setsysinfo(2)` を参照してください。
2. プロセスのファイル記述子のソフト限界値に、4,096 (省略時の値) を超える値を設定します。この設定を行うには、次のコードに示すように、`setrlimit` 関数を使用します。

```
# include <sys/resource.h>
struct rlimit *rlp;

rlp->rlim_cur = 6000;
rlp->rlim_max = 6000;
setrlimit(RLIMIT_NOFILE, rlp);
```

この設定は、すべての子プロセスに引き継がれます。詳細は、`setrlimit(2)` を参照してください。

3. このステップは、`select` 関数の `fd_set` パラメータを使用するアプリケーションの場合だけ必要です。このパラメータは、入出力記述子セット (および、`FD_CLR` マクロ、`FD_ISSET` マクロ、`FD_SET` マクロ、または `FD_ZERO` マクロ) を指し、入出力記述子セットを変更できます。この条件を満たしている場合は、次の 2 つの手順 (ファイル記述子の最大値の静的定義を有効にする手順または動的定義を有効にする手順) のいずれかを使用できます。

- 静的な定義:

`FD_SETSIZE` に最大値 65,536 を指定して、`<sys/select.h>` ヘッダ・ファイル内の `FD_SETSIZE` の省略時の値 4,096 を無効にします。この値は次のように、コード内で `<sys/time.h>` ヘッダ・ファイル (`<sys/select.h>` ヘッダ・ファイルをインクルードする) をインクルードする前に指定しなければなりません。

```
# define FD_SETSIZE 65536
# include <sys/time.h>
```

この設定は、子プロセスには継承されません。このため、64 KB ファイル記述子を必要とする各子プロセスのコードに、`FD_SETSIZE` を明示的に設定しなければなりません。

- 動的な定義:

静的に定義される `fd_set` 構造体を使用する代わりに、`fd_set` ポインタを `malloc` 関数とともに使用することができます。この方法では、ファイル記述子の最大数が将来変更になっても、互換性が保たれます。例を次に示します。

```
fd_set *fdp;

fdp = (fd_set *) malloc(
    (fds_howmany(max_fds, FD_NFDBITS)) * sizeof(fd_mask));

max_fds の値は、操作するファイル記述子の数です。ファイル記述子のソフト限界値を、この値に指定することをお勧めします。他のキーワードはすべて、<sys/select.h> ヘッド・ファイルで定義されています。この方法を選択した場合のコードを次に示します。

#include <sys/time.h>
#include <sys/resource.h>

my_program()
{
    fd_set *fdp;
    struct rlimit rlim;
    int max_fds;

    getrlimit(RLIMIT_NOFILE, &rlim);
    max_fds = rlim.rlim_cur;

    fdp = (fd_set *) malloc(
        (fds_howmany(max_fds, FD_NFDBITS)) * sizeof(fd_mask));

    FD_SET(2, fdp);

    for (;;) {
        switch(select(max_fds, (fd_set *)0, fdp, (fd_set *)0,
            struct timeval *)0)) {
            ...
        }
    }
}
```

また、`vfs` サブシステムの `max_vnodes` 属性には、記述子を多数必要とするアプリケーションに合わせて大きな値を設定しなければなりません。`max_vnodes` 属性は `vnode` キャッシュのサイズを指定し、省略時はシステム・メモリの 5 パーセントが設定されます。詳細は、8.5.1 項を参照してください。

64 KB までのファイル記述子のサポートをアプリケーションで無効にするには、`setsysinfo` システム・コールの `SSI_FD_NEWMAX` 操作のパラメータに 0 を指定します。これによって `utask` ビットが無効になり、ハード・ファイル限界値がオープン記述子の省略時の最大値である 4,096 に戻ります。

す。ただし、プロセスが 4,096 を超えるファイル記述子を使用した場合、`setsysinfo` システム・コールは `EINVAL` エラーを返します。また、呼び出しプロセスのハードまたはソフト限界値が 4,096 を超えていた場合、呼び出しが正常終了すると、この限界値は 4 KB になります。この設定は、すべての子プロセスに継承されます。

8.6 Aurema ARMTech Suite

Tru64 UNIX では、Aurema の ARMTech (Active Resource Management Technology) ソフトウェア製品スイートをサポートしています。このソフトウェアは、高度なリソース管理機能を提供しています。ARMTech スイートを使用すると、Tru64 UNIX のシステム管理者は、ユーザとグループの他に、Web サイトやアプリケーションなど、オペレーティング・システムの多くの構成要素を管理することができます。

`armtech(5)` と Aurema の Web サイトを参照してください。

<http://www.aurema.com>

ディスク・ストレージ性能の管理

ディスク・ストレージを管理する方法には、さまざまな方法があります。性能と可用性のニーズに応じて、静的ディスク・パーティション、LSM (Logical Storage Manager)、ハードウェア RAID、またはこれらの方法の組み合わせが使用できます。

ディスク・ストレージの構成は、システムの性能に大きく影響します。これは、ディスクの入出力はファイル・システムの動作で使用され、また、ページングやスワッピングのために仮想メモリ・サブシステムでも使用されるためです。

この章で説明する構成やチューニング・ガイドラインに従ってチューニングすると、ディスク入出力の性能が改善できます。内容は次のとおりです。

- 入出力負荷の分散による、全体的なディスク入出力性能の改善 (9.1 節)
- ディスク入出力の分散のモニタリング (9.2 節)
- LSM 性能の管理 (9.3 節)
- ハードウェア RAID サブシステムの性能の管理 (9.4 節)
- CAM (Common Access Method) の性能の管理 (9.5 節)

すべてのディスク・ストレージの構成に対してすべてのガイドラインがあてはまるとは限りません。ガイドラインを適用する前に、1.8 節で説明している作業負荷のリソース・モデルと、ガイドラインの利点と欠点を理解しておいてください。

9.1 ディスク入出力負荷の分散に関するガイドライン

ディスク入出力の負荷を複数のデバイスに分散させると、単一のディスク、コントローラ、バスなどがボトルネックになるという事態を避けることができます。また、入出力の同時動作も可能になります。

たとえば、16 GB のディスク・ストレージを使用するとき、4 GB のディスク 4 台よりも 1 GB のディスク 16 台の方が性能が良くなります。これ

は、使用するドライブ (ディスク) が多いと同時に実行できる動作も多くなるためです。ランダムな入出力動作では、4 台ではなく 16 台のディスクを同時にシークさせることができます。大量のシーケンシャル・データ転送では、4 つのデータ・ストリームではなく 16 個のデータ・ストリームを同時に動作させることができます。

ディスク入出力の負荷を分散させる際には、次のガイドラインに従ってください。

- データまたはディスクをストライピングする

RAID0 (データまたはディスクのストライピング) を使用すると、データを効率的にディスク間に分散させることができます。ストライピングの利点についての詳細は、11.2.1.5 項を参照してください。ストライピングしたディスク・アレイのディスク数が増えると、可用性が低下することに注意してください。

データをストライピングするには、LSM (9.3 節 参照) を使用します。ディスクをストライピングするには、ハードウェア RAID サブシステム (9.4 節 参照) を使用します。

データまたはディスクのストライピングに対する代替方法として、AdvFS (Advanced File System) を使用して、ファイル・ドメイン内の複数のディスクにわたって個別のファイルをストライピングするという方法があります。ただし、ディスクとその中のファイルを同時にストライピングしないでください。詳細は、11.2 節を参照してください。

- RAID5 を使用する

RAID5 では、アレイ内の複数のディスクにディスク・データとパリティ・データを分散させて、データの可用性を高め、読み取りの性能を向上させます。ただし、RAID5 を使用すると、障害のない状態で書き込みの性能が低下し、障害のある状態で読み取りと書き込みの性能が低下します。RAID5 は主に読み取りを多用する構成で使用します。ミラーリングに対するコスト効率の良い代替方法として、RAID5 を使用して、アクセス頻度の低いデータの可用性を高めることもできます。

RAID5 構成を実現するには、LSM (9.3 節 参照) またはハードウェア RAID サブシステム (9.4 節 参照) を使用します。

- 頻繁にアクセスされるファイル・システムを複数のディスク (できれば異なるバスおよびコントローラ) に分散させる

頻繁にアクセスされるファイル・システムを複数のディスク (できれば異なるバスおよびコントローラ) に分散させます。実行可能ファイルや一時的ファイルを含むディレクトリ, すなわち `/var`, `/usr`, `/tmp` などは頻繁にアクセスされます。できれば, `/usr` と `/tmp` は異なるディスクに置いてください。

AdvFS の `balance` コマンドを使用すると, AdvFS ファイル・ドメイン内のディスク間で, 使用される領域のバランスをとることができます。詳細は, 11.2.1.4 項を参照してください。

- スワップ入出力を複数のデバイスへ分散させる

ページングおよびスワッピングの効率を良くし, アダプタ, バス, ディスクのいずれもボトルネックにならないようにするには, スワップ領域を複数のディスクに分散させます。同じディスクに複数のスワップ・パーティションを置かないでください。

また, LSM を用いてスワップ領域をミラーリングすることもできます。詳細は, 9.3 節を参照してください。

スワップ・デバイスを構成して性能を高める方法の詳細については, 12.2 節を参照してください。

ディスク入出力の分散状況をモニタリングする方法については, 9.2 節を参照してください。

9.2 ディスク入出力の分散状況のモニタリング

表 9-1 では, ディスク入出力が分散されているかどうかを調べるために使用できるコマンドについて説明しています。

表 9-1: ディスク入出力の分散状況のモニタリング・ツール

ツール	説明	参照先
<code>showfdmn</code>	AdvFS ファイル・ドメインに関する情報を表示し, AdvFS ボリューム間でファイルが均等に分散されているかどうかを調べる。	11.2.2.3 項

表 9-1: ディスク入出力の分散状況のモニタリング・ツール (続き)

ツール	説明	参照先
advfsstat	AdvFS のファイル・ドメインおよびファイル・セットの使用状況に関する情報を表示して、ファイル・システム入出力が均等に分散されているかどうかを判断するために使用できる、AdvFS のファイル・ドメインおよびファイル・セットの統計情報を提供する。	11.2.2.1 項
swapon	スワップ領域の構成と使用状況に関する情報を表示する。スワップ・パーティションごとに、割り当てられたスワップ領域の総容量、使用されているスワップ領域の容量、空きスワップ領域の容量が表示される。	12.3.3 項
volstat	LSM オブジェクトについて、性能統計情報を表示し、入出力の作業負荷の特性を調べて理解するために使用できる、LSM ボリュームおよびディスクの使用状況に関する情報を提供する。この情報には読み取り/書き込み比率、平均転送サイズ、ディスク入出力が均等に分散されているか、などの情報が含まれる。	9.3 節または『 <i>Logical Storage Manager</i> 』
iostat	ディスク入出力の統計情報を表示し、どのディスクが最もよく使用されているかについての情報を提供する。	9.2.1 項

9.2.1 iostat コマンドによるディスク使用状況の表示

最良の性能を得るには、ディスク間でディスク入出力を均等に分散させなければなりません。iostat コマンドを使用すると、どのディスクが最も頻繁に使用されているかを調べることができます。このコマンドは、端末および CPU の統計情報の他に、ディスク入出力に関する統計情報も表示します。

iostat コマンドの例を次に示します。出力は 1 秒間隔で表示されます。

```
# /usr/ucb/iostat 1
      tty      floppy0      dsk0      dsk1      cdrom0      cpu
tin tout    bps tps    bps tps    bps tps    bps tps    us ni sy id
  1   73      0  0    23  2    37  3      0  0     5  0 17 79
  0   58      0  0    47  5   204 25      0  0     8  0 14 77
  0   58      0  0     8  1    62  1      0  0    27  0 27 46
```

iostat コマンドの出力では、次のような情報が表示されます。

- iostat コマンド出力の最初の行は、ブート時からの平均です。その後には、一定の時間間隔で最新の情報が表示されます。
- 各ディスクの欄 (dskn) には、1 秒あたりの転送 KB 数 (bps) と、1 秒あたりの転送回数 (tps) が表示されます。
- システムの欄 (cpu) には、ユーザ状態において省略時の優先順位または高い優先順位 (preferred) で CPU が動作した時間 (us)、ユーザ・モードにおいて低い優先順位 (less favored) でプロセスが実行された時間 (ni)、システム・モードの時間 (sy)、アイドル・モードの時間 (id) の割合が表示されます。この情報により、ディスク入出力がどの程度 CPU 時間に影響しているかがわかります。ユーザ・モードには、CPU がライブラリ・ルーチンを実行した時間も含まれます。システム・モードには、CPU がシステム・コールを処理した時間も含まれます。

iostat コマンドは、次のような場合に役に立ちます。

- 使用頻度の最も高いディスクと最も低いディスクを見つける場合。この情報は、ファイル・システムとスワップ領域をどのように分散させるかを判断する際に役に立ちます。swapon -s コマンドを使用すると、どのディスクがスワップ領域として使用されているかが分かります。
- システムがディスク多用型であるかどうかを調べる場合。iostat コマンドの出力で、ディスクのアクティビティが多く、システムのアイドル時間の割合が高いことが示されている場合は、そのシステムはディスク多用型である可能性があります。ディスク入出力の負荷バランスの調整、ディスクの断片化の解消、ハードウェアのアップグレードなどを行わなければならない可能性があります。
- アプリケーションが効率的に作成されているかどうかを調べる場合。ディスクの転送回数 (「tps」フィールド) が多いが、データの読み取りと書き込みの量 (「bps」フィールド) が少ない場合は、アプリケーションがディスク入出力をどのように行っているか調べてください。アプリケーションが大量の入出力操作を行って少量のデータだけを処理してい

る可能性があります。このような動作が必要でなければ、アプリケーションを作り直してください。

9.3 LSM を使用したストレージの管理

LSM (Logical Storage Manager) を使用すると、柔軟なストレージ管理、ディスク入出力性能の改善、データの高可用性を、少しのオーバーヘッドで実現できます。どのようなタイプのシステムでも LSM による効果がありますが、大量のディスクを備えた構成や、定期的に記憶域を追加していく構成には特に適しています。

LSM を使用すると、複数のディスクからなる記憶域のプールを設定できます。これらのディスク・グループから、仮想ディスク (LSM ボリューム) を作成し、それをディスク・パーティションと同じように使用することができます。UFS または AdvFS ファイル・システムをボリューム上に作成したり、ボリュームを raw デバイスとして使用したり、RAID ストレージ・セット上にボリュームを作成することができます。

LSM ボリュームと物理ディスクの間には直接的な関係がないので、複数のディスクをまたいでファイル・システムや raw 入出力を作成することができます。ディスクをディスク・グループに追加/削除したり、入出力の負荷のバランスを調整したり、他のストレージ管理作業を実行することは簡単にできます。

さらに、RAID 技術を使用すると、LSM の性能と可用性が向上します。LSM はソフトウェア **RAID** と呼ばれることがよくあります。LSM 構成は、ハードウェア RAID サブシステムよりコスト効率が良く、より単純です。LSM RAID 機能を使用するためには別途ライセンスが必要なので注意してください。

9.3.1 LSM の機能

LSM には、次のような、ライセンスを必要としない基本的なディスク管理機能があります。

- ディスク連結機能により、複数のディスクから 1 つの大きなボリュームを作成できます。
- 負荷バランス調整機能により、データを透過的に複数のディスクに分散できます。

- 構成データベースの負荷バランス調整機能により、人手を介さずに、最適な数の LSM 構成データベースを適切な位置に自動的に配置できます。
- `volstat` コマンドを使用すると、LSM の詳細な性能情報を表示できます。

以下の LSM 機能は、ライセンスを必要とします。

- RAID0 (ストライピング) は、アレイ内のディスクにデータを分散させます。ストライピングは、大量のデータを高速に転送したい場合に効果があります。また、複数ユーザのアプリケーションからの入出力負荷を複数のディスクに均等に分散させることができます。LSM ストライピングでは、CPU に負担を少しかけるだけで、入出力の性能が大きく改善されます。
- RAID1 (ミラーリング) は、データのコピーを複数のディスクに保存し、ディスクが 1 台故障しただけでデータが使用不可能にはならないようにします。
- RAID5 (パリティ RAID) は、パリティを持つことでデータの可用性を高め、データとパリティをアレイ内のディスクに分散させます。
- ルート・ファイル・システムとスワップ領域をミラーリングすることで、可用性を改善できます。
- ホット・スペア・サポートを使用すると、ミラーリングまたは RAID5 のオブジェクトで入出力の障害が発生した場合に自動的に対処することができます。これは、影響を受けるオブジェクトをスペア・ディスクまたは他の空き領域に再配置することで行います。
- ダーティ・リージョン・ロギング (DRL) は、システムに障害が発生した後のミラー・ボリュームの回復時間を短縮します。
- グラフィカル・ユーザ・インタフェース (GUI) を使用すると、ディスク管理が簡単に実行でき、詳細な性能情報を得ることができます。

LSMの性能を最良の状態にするには、『*Logical Storage Manager*』で説明している構成とチューニングのガイドラインに従ってください。

9.4 ハードウェア RAID サブシステムの性能の管理

ハードウェア RAID サブシステムは、高性能で高可用性の RAID 機能を実現し、CPU のディスク入出力オーバーヘッドを軽減し、1 本 (場合によっては複

数本)の入出力バスに多数のディスクを接続可能にします。ハードウェア RAID サブシステムには、性能や可用性の機能が異なる種々のタイプがありますが、いずれのタイプにも RAID コントローラ、筐体に組み込まれたディスク、ケーブル配線、ディスク管理ソフトウェアが含まれます。

RAID ストレージ・ソリューションには、低価格のバックプレーン RAID アレイ・コントローラから、クラスタ機能を備え、広範囲にわたる性能と可用性機能(ライトバック・キャッシュや構成要素の完全な冗長性など)を持つ RAID アレイ・コントローラまであります。

ハードウェア RAID サブシステムでは、RCU (RAID Configuration Utility) や SWCC (StorageWorks Command Console) ユーティリティなどのディスク管理ソフトウェアを使用して、RAID デバイスを管理します。メニュー方式のインタフェースを使用して、RAID のレベルを選択できます。

ハードウェア RAID を使用して複数のディスクを 1 台のストレージ・セットとして連結し、システムがそれを 1 台のユニットとして認識するようにします。ストレージ・セットは、ディスクの単純なセット、ストライプ・セット、ミラー・セット、RAID セットのいずれかで構成できます。ストレージ・セット上に LSM ボリューム、AdvFS ファイル・ドメイン、または UFS ファイル・システムを作成したり、ストレージ・セットを raw デバイスとして使用することができます。

以降の項では、次の RAID ハードウェアのトピックについて説明します。

- ハードウェア RAID 機能 (9.4.1 項)
- ハードウェア RAID 製品 (9.4.2 項)
- ハードウェア RAID の構成に関するガイドライン (9.4.3 項)

構成情報についての詳細は、ハードウェア RAID 製品のマニュアルを参照してください。

9.4.1 ハードウェア RAID 機能

ハードウェア RAID ストレージ・ソリューションには、低価格のバックプレーン RAID アレイ・コントローラから、クラスタ機能を備え、広範囲にわたる性能と可用性機能を持つ RAID アレイ・コントローラまであります。ハードウェア RAID サブシステムは、すべて以下の機能を備えています。

- CPU のディスク入出力オーバーヘッドを軽減する RAID コントローラ

- ディスク・ストレージ容量の増強

ハードウェア RAID サブシステムを使用すると、大量のディスクを 1 本 (場合によっては複数本) の入出力バスに接続できます。一般的なストレージ構成では、入出力バス・スロットにインストールされたホスト・バス・アダプタにバスを接続して、ディスク・ストレージ・シェルフをシステムに接続します。ただし、SCSI バスに接続できるディスクの数は限られており、システムの入出力バス・スロットの数も限られています。

これに対して、ハードウェア RAID サブシステムには複数の内部 SCSI バスがあり、1 本の入出力バス・スロットを用いてシステムに接続できます。

- 読み取りキャッシュ

読み取りキャッシュは、ホストが要求すると予想されるデータを保持しておくことで、入出力の読み取り性能を改善します。システムが読み取りキャッシュ内にすでにあるデータを要求した場合 (キャッシュ・ヒット)、データはただちに渡され、ディスクからは読み取られません。データが変更されると、ディスクと読み取りキャッシュの両方に書き込まれます (ライトスルー・キャッシュ)。

- ライトバック・キャッシュ

ハードウェア RAID サブシステムはライトバック・キャッシュを (標準またはオプションの機能として) サポートしています。これにより、入出力の書き込み性能が向上し、同時にデータの完全性も維持されます。ライトバック・キャッシュは多数の小規模な書き込みの待ち時間を短縮し、書き込みがただちに行われるように見せることで、インターネット・サーバの性能を改善します。書き込みをほとんど行わないアプリケーションでは、ライトバック・キャッシュのメリットはありません。

ライトバック・キャッシュを使用すると、ディスクに書き込まれるデータは一時的にキャッシュに保存され、統合されてから定期的にディスクに書き込まれる (フラッシュされる) ため最高の効率を実現できます。複数のホストからの連続したデータ・ブロックの書き込みが単一のユニットに統合されるため、入出力の待ち時間が短縮されます。

ライトバック・キャッシュは、データの損失や破損を防ぐために無停電電源装置 (UPS) でバックアップしなければなりません。

- RAID サポート

ハードウェア RAID サブシステムは、RAID0 (ディスクのストライピング)、RAID1 (ディスクのミラーリング)、RAID5 をサポートしています。高性能の RAID アレイ・サブシステムでは、RAID3 と動的パリティ RAID もサポートしています。RAID レベルについての詳細は、1.3.1 項を参照してください。

- 非 RAID ディスク・アレイ機能または「単なるディスクの集まり」(JBOD: just a bunch of disks)
- 構成要素のホット・スワッピングとホット・スペアリング
ホット・スワップ機能を使用すると、システムを稼働させたまま、故障した構成要素を交換できます。ホット・スペア機能を使用すると、障害が発生した場合に、あらかじめインストールしておいた構成要素に自動的に切り替えることができます。
- 管理とモニタリングを簡単にするグラフィカル・ユーザ・インタフェース (GUI)

9.4.2 ハードウェア RAID 製品

ハードウェア RAID サブシステムには何種類かのタイプがあり、性能や可用性のレベルに応じてさまざまなコストで実現できるようになっています。HP では次のようなハードウェア RAID システムをサポートしています。

- バックプレーン RAID アレイ・ストレージ・サブシステム
RAID Array 230/Plus ストレージ・コントローラを利用しているような、エントリ・レベルのサブシステムは、低コストのハードウェア RAID ソリューションであり、小～中規模の部門およびワークグループ用に設計されています。
バックプレーン RAID アレイ・ストレージ・コントローラは、PCI バス・スロットにインストールされており、ホスト・バス・アダプタと RAID コントローラの両方の働きをします。
バックプレーン RAID アレイ・サブシステムには、RAID 機能 (0, 1, 0+1, 5)、オプションのライトバック・キャッシュ、およびホット・スワップ機能があります。
- 高性能の RAID アレイ・サブシステム

このサブシステムには RAID Array 450 サブシステムなどがあり、広範囲の性能と可用性機能を備えており、クライアント/サーバ、データ・センター、中～大規模の部門用に設計されています。

HSZ80 コントローラのような高性能の RAID アレイ・コントローラは、ultrawide のディファレンシャル SCSI バスと、入出力バス・スロットにインストールされた高性能のホスト・バス・アダプタによってシステムに接続されます。

高性能の RAID アレイ・サブシステムは、RAID 機能 (0, 1, 0+1, 3, 5, 動的パリティ RAID)、デュアル冗長コントローラのサポート、スケラビリティ、ストレージ・セットのパーティショニング、標準 UPS ライトバック・キャッシュ、ホット・スワップ可能な構成要素を備えています。

- ESA (Enterprise Storage Arrays)/MSA (Modular storage array)

RAID Array 12000 などの設定済みの高性能ハードウェア RAID サブシステムであり、最高の性能、可用性、RAID サブシステムのディスク容量を備えています。これらは、高度なトランザクション指向アプリケーションや高帯域幅の意思決定サポート・アプリケーションで使用されます。

ESA は、動的パリティ RAID を含め主要な RAID レベルすべて、ホット・スワップが可能な完全冗長構成要素、標準 UPS ライトバック・キャッシュ、集中型ストレージ管理をサポートしています。

ハードウェア RAID サブシステム機能についての詳細は、『*Logical Storage Manager Version 5.1B QuickSpecs*』を参照してください。

9.4.3 ハードウェア RAID 構成のガイドライン

表 9-2 に、ハードウェア RAID サブシステムの構成ガイドラインを示し、性能上の利点と欠点をリストします。

表 9-2: ハードウェア RAID サブシステムの構成のガイドライン

ガイドライン	性能上の利点	欠点
ストレージ・セット内のディスクを複数のバスに均等に分散させる (9.4.3.1 項)。	性能が向上しボトルネックを防げる。	なし
各ストレージ・セットで、同じデータ容量のディスクを使用する (9.4.3.2 項)。	ストレージ管理が簡単になる。	なし

表 9-2: ハードウェア RAID サブシステムの構成のガイドライン (続き)

ガイドライン	性能上の利点	欠点
適切なストライプ・サイズを使用する (9.4.3.3 項)。	性能が向上する。	なし
ストライプ・セットをミラーリングする (9.4.3.4 項)。	可用性が向上し、ディスクの入出力負荷が分散される。	構成の複雑さが増し、書き込み性能が低下するおそれがある。
ライトバック・キャッシュを使用する (9.4.3.5 項)。	特に RAID5 ストレージ・セットの書き込み性能が向上する。	ハードウェアのコスト
デュアル冗長 RAID コントローラを使用する (9.4.3.6 項)。	性能と可用性が向上し、入出力バスのボトルネックを防止できる。	ハードウェアのコスト
スペア・ディスクをインストールする (9.4.3.7 項)。	可用性が向上する。	ディスクのコスト
障害が発生したディスクをすぐに交換する (9.4.3.7 項)。	性能が向上する。	なし

以降の項では、これらのガイドラインについて説明します。構成についての詳細は、RAID サブシステムのマニュアルを参照してください。

9.4.3.1 ストレージ・セットのディスクをバス間に分散させる

ストレージ・セットのディスクを複数のバスに均等に分散させると、性能が向上し、ボトルネックを防ぐことができます。

また、ミラーリングした各セットの最初のメンバは、必ず別々のバスに置くようにしてください。

9.4.3.2 同じデータ容量のディスクを使用する

同じ容量のディスクをストレージ・セット内で使用します。これにより、ストレージ管理が簡単になり、無駄なディスク・スペースが削減できます。

9.4.3.3 正しいハードウェア RAID ストライプ・サイズを選択する

最高の性能が得られるようにストライプ (チャンク) サイズを選択するには、まず、アプリケーションがディスク入出力をどのように実行するかを把握しなければなりません。システムのリソース・モデルを把握する方法についての詳細は、1.8 節を参照してください。

次に、ストライプ・サイズについてのガイドラインを示します。

- ストライプ・サイズが、平均的な入出力サイズに比べて大きい場合、ストライプ・セット内の各ディスクは個別のデータ転送に応えることができます。その後、入出力動作を並列に実行できるため、シーケンシャル書き込みの性能とスループットが向上します。これにより、大量の入出力動作を実行する環境（トランザクション処理、オフィス・オートメーション、ファイル・サービス環境など）、および複数のランダムな読み書きを実行する環境での性能を改善することができます。
- ストライプ・サイズが、平均的な入出力サイズに比べて小さい場合、1つの入出力動作を複数のディスクで同時に処理できるため、帯域幅が拡大されシーケンシャル・ファイルの処理が改善されます。これは、画像処理やデータ収集の環境に有効です。ただし、ストライプ・サイズを小さくしすぎて、大規模なシーケンシャル・データ転送の性能が低下しないように注意してください。

たとえば、ストライプ・サイズを8 KB にすると、小規模なデータ転送はメンバ・ディスクに均等に分散されますが、64 KB のデータ転送は少なくとも 8 回のデータ転送に分割されます。

また、正しいストライプ・サイズを選択する際には、次のガイドラインが役に立ちます。

- raw ディスクの入出力操作

アプリケーションが、ファイル・システムではなく raw デバイスへの入出力を行っている場合は、1 回のデータ転送がメンバ・ディスク間に均等に分散されるようなストライプ・サイズを使用してください。たとえば、典型的な入出力サイズが 1 MB で、アレイ内に 4 つのディスクがある場合、ストライプ・サイズを 256 KB にします。これにより、データは 4 台のメンバ・ディスクに均等に分散され、それぞれのディスクが 256 KB のデータ転送を 1 回ずつ、並行して行うことになります。

- 小規模ファイル・システムの入出力操作

小規模ファイル・システムの入出力操作では、典型的な入出力サイズの倍数（たとえば、入出力サイズの 4、5 倍）のストライプ・サイズを使用します。これにより、入出力が複数のディスクに分割されないようになります。

- ブロックの特定の範囲に対する入出力

ブロックの特定範囲がボトルネックにならないようなストライプ・サイズを選択します。たとえば、アプリケーションが特定の 8 KB ブロックを頻繁に使用する場合、8 KB より少し大きいまたは小さいストライプ・サイズを使用するか、または 8 KB の倍数のサイズを使用して、データが強制的に他のディスクに入るようにします。

9.4.3.4 ストライプ・セットをミラーリングする

ストライプ・ディスクでは、ディスクの入出力負荷が分散されるため、入出力性能が向上します。ただし、ストライピングを行うと、ディスクの 1 台に障害が発生しただけで、ストライプ・セット全体が使用できなくなるため、可用性が低下します。ストライプ・セットの可用性を高めるには、ストライプ・セットをミラーリングします。

9.4.3.5 ライトバック・キャッシュを使用する

RAID サブシステムでは、標準機能またはオプション機能として、不揮発性 (バッテリー・バックアップ付き) のライトバック・キャッシュをサポートしています。これにより、ディスクの入出力性能を改善でき、データの完全性も確保できます。ライトバック・キャッシュにより、頻繁に書き込みを行うシステムや、RAID5 ストレージ・セットの性能が改善されます。書き込みをほとんど行わないアプリケーションでは、ライトバック・キャッシュの効果はあまりありません。

ライトバック・キャッシュを使用すると、ディスクに書き込まれるデータは一時的にキャッシュに保存されてから、定期的にディスクに書き込まれる (フラッシュされる) ため効率を最大限に高めることができます。複数のホストからの連続したデータ・ブロックの書き込みが単一のユニットに統合されるため、入出力の待ち時間が短縮されます。

ライトバック・キャッシュを使用すると、書き込みがただちに行われるように見えるため、特にインターネット・サーバの性能が向上します。障害発生後の回復時に、RAID コントローラは、ライトバック・キャッシュ内でまだ書き込まれていないデータを検出し、そのデータをディスクに書き込み、その後で通常のコントローラ動作を開始します。

ライトバック・キャッシュは、データの紛失や破損を防ぐために無停電電源装置 (UPS) でバックアップする必要があります。

HSZ40, HSZ50, HSZ70, HSZ80 のいずれかの RAID コントローラでライトバック・キャッシュを使用している場合は、次のガイドラインで性能が改善されることがあります。

- `CACHE_POLICY` に B を設定する
- `CACHE_FLUSH_TIMER` を最小の 45 (秒) にする
- 各ユニットに対してライトバック・キャッシュ (`WRITEBACK_CACHE`) を有効にし、`MAXIMUM_CACHED_TRANSFER_SIZE` の値を最小の 256 にする

ライトバック・キャッシュの使用についての詳細は、RAID サブシステムのマニュアルを参照してください。

9.4.3.6 デュアル冗長コントローラを使用する

RAID システムでデュアル冗長コントローラがサポートされていれば、デュアル冗長コントローラ構成を使用して、2 つのコントローラにディスクを分散させてバランスをとることができます。これにより性能が改善され、可用性も高くなり、入出力バスのボトルネックも防止できます。

9.4.3.7 破損したディスクをスペア・ディスクと交換する

あらかじめ、スペア・ディスクを別のコントローラ・ポートとストレージ・シェルフにインストールします。これによりデータの可用性が維持され、ディスクに障害が発生してもすぐに回復できます。

9.5 CAM 性能の管理

CAM (Common Access Method) は、オペレーティング・システムとハードウェアの間のインタフェースです。CAM は、入出力の実行に使用するバッファのプールを管理しています。各バッファの物理メモリ・サイズは、約 1 KB です。これらのプールをモニタリングして、必要に応じてチューニングしてください。

次のような `io` サブシステム属性を変更すると、CAM 性能が向上する可能性があります。

- `cam_ccb_pool_size`— ブート時点での、バッファ・プール空きリストの初期サイズ。省略時の値は 200 です。
- `cam_ccb_low_water`— プールの空きリストにおいて、カーネルからバッファの追加割り当てが発生するバッファ数。CAM はこれだけの数

のバッファを予約することで、暴走したプロセスをカーネルがシャットダウンするためのメモリを確保します。省略時の値は 100 です。

- `cam_ccb_increment`— バッファ・プールの空きリストに追加される、または削除されるバッファ数。バッファは、要求をただちに処理するために必要に応じて割り当てられますが、解放は、スパイクから保護するために慎重に行われます。省略時の値は 50 です。

システムの入出力パターンが、入出力動作が断続的に多発する傾向にある場合 (入出力スパイク) は、`cam_ccb_pool_size` および `cam_ccb_increment` 属性の値を大きくすると、性能が改善されることがあります。

`dbx` を用いて、`raw` ディスク入出力に使用されるバッファ構造体プールの統計情報を保持している `ccmn_bp_head` データ構造体を調べると、CAM の性能に関する問題を診断できることがあります。表示される情報は、バッファ構造体プールの現在のサイズ (`num_bp`) と、バッファ待ちの回数 (`bp_wait_cnt`) です。

例:

```
# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) print ccmn_bp_head
struct {
    num_bp = 50
    bp_list = 0xffffffff81f1be00
    bp_wait_cnt = 0
}
(dbx)
```

`bp_wait_cnt` フィールドの値が 0 でない場合は、CAM のバッファ・プール領域を使いきってしまった状態が発生しています。この状態が続く場合、ここで説明した CAM サブシステム属性をいくつか変更することで、問題を解決できる可能性があります。

ネットワーク性能の管理

クライアントとサーバとの間の通信で使用しているネットワークは、通常は性能上の問題を起こすことはありません。しかし、注意すべき 2 つの状況があります。ネットワーク遅延と、高い再送率です。イーサネットが過度に使用されると、クライアントで要求送信のための空きスロットを待ち、大きな遅れが出る現象が発生します。イーサネットで、使用率が 50 % を超えると、非常に大きなネットワーク遅延が発生しやすくなります。

ネットワーク・トポロジが大きな遅延を招くこともあります。クライアントがよく使用するサーバに到達するまでにいくつものゲートウェイを経由する必要がある場合、クライアントからの要求は遅延することになります。ネットワーク・トポロジを再構築して負荷を均等に分散させることで問題が解決することがあります。

この章では、Tru64 UNIX のネットワーク・サブシステムの性能を管理する方法について説明します。以降の各節では、次のような内容を説明しています。

- ネットワーク・サブシステムのモニタリング方法 (10.1 節)
- ネットワーク・サブシステムのチューニング方法 (10.2 節)

10.1 ネットワーク情報の収集

表 10-1 では、ネットワークの動作状況を収集するために使用できるコマンドを説明します。

表 10-1: ネットワークのモニタリング・ツール

ツール	説明	参照先
netstat	ネットワークの統計情報と、プロトコルごとのアクティブなソケットのリスト、ネットワーク経路に関する情報、ネットワーク・インタフェースに関する累積統計情報(受信および送信のパケット数、パケット衝突の回数など)を表示する。また、ネットワーク動作に使用されるメモリに関する情報も表示する。	2.4.5 項
sobacklog_hiwat 属性	任意のサーバ・ソケットに対して保留された要求の最大数を報告する。システム内の任意のサーバ・ソケットに対して保留された要求の最大数を表示できる。	10.1.1 項
sobacklog_drops 属性	ソケットのバックログの限界値を超えたために失われたバックログの数を報告する。キューに入れられた、ソケットへの SYN_RCVD 接続の数がソケットのバックログ限界値と等しくなったために、受信した SYN パケットをシステムがドロップした回数を表示できる。	10.1.1 項
somaxconn_drops 属性	somaxconn 属性の値を超えたドロップの数を報告する。キューに入れられた、ソケットへの SYN_RCVD 接続の数がバックログ長の上限 (somaxconn 属性) と等しくなったために、受信した SYN パケットをシステムがドロップした回数を表示できる。	10.1.1 項

表 10-1: ネットワークのモニタリング・ツール (続き)

ツール	説明	参照先
ping	ネットワーク上で特定のシステムに到達できるかどうかを調べる。 ICMP (Internet Control Message Protocol) のエコー要求をホストに送信し、ホストが稼働していて到達可能かどうか、また IP ルータに到達可能かどうかを調べる。 ルーティングに直接また間接的に関係する問題など、ネットワークに関する問題を切り分けるために使用する。	ping(8) を参照
tcpdump	ネットワーク・インタフェース・パケットをモニタリングする。 リッスンするインタフェース、パケットの転送方向、表示するプロトコル・トラフィックのタイプを指定できる。 tcpdump コマンドでは、特定のネットワーク・サービスに対応するネットワーク・トラフィックをモニタリングしたり、パケットのソースを確認したりすることができる。また、要求が受信または肯定応答されたかどうかを調べたり、あるいはネットワークの性能が低速である場合にネットワーク要求のソースを調べたりすることができる。 このコマンドを使用するには、packetfilter オプションを用いてカーネルを構成しておかなければならない。	2.4.4 項を参照
traceroute	ネットワーク・ホストへのパケット経路を表示し、ネットワーク・パケットがゲートウェイ間で転送される経路を追跡する。	traceroute(8) を参照

10.1.1 sysconfig コマンドを用いてソケット・リッスン・キューの統計情報をチェックする

`sysconfig -q socket` コマンドを用いて以下の属性の値を表示することにより、ソケット・リッスン・キューの限界値を大きくする必要があるかどうかを判断できます。

- `sobacklog_hiwat` — システム内のすべてのサーバ・ソケットについて、保留される要求の最大数をモニタリングできます。初期値は 0 です。
- `sobacklog_drops` — キュー内にあるソケットへの `SYN_RCVD` 接続の数がソケットのバックログ限界値と等しくなったために、受信した `SYN` パケットをシステムがドロップした回数をモニタリングできます。初期値は 0 です。
- `somaxconn_drops` — キュー内にあるソケットへの `SYN_RCVD` 接続の数がバックログ長の上限 (`somaxconn` 属性) と等しくなったために、受信した `SYN` パケットをシステムがドロップした回数をモニタリングできます。初期値は 0 です。

`sominconn` 属性の値と `somaxconn` 属性の値を同じにすることをお勧めします。その場合、`somaxconn_drops` の値は `sobacklog_drops` の値と同じになります。

ただし、`sominconn` 属性の値が 0 (省略時) であり、1 つまたは複数のサーバ・アプリケーションが `listen` システム・コールへのバックログ引数として不適切な値を用いている場合、`sobacklog_drops` の値は、`somaxconn_drops` カウンタが増加する速さより速く増加することがあります。このような場合は、`sominconn` 属性の値を増やしてください。

ソケット・リッスン・キューの限界値のチューニングについての詳細は、10.2.3 項を参照してください。

10.2 ネットワーク・サブシステムのチューニング

ネットワーク・サブシステムが使用するリソースの多くは、動的に割り当てられて調整されます。ただし、性能を改善するために使用できるチューニングのガイドラインもあります。特に、システムが Web サーバ、プロキシ・サーバ、ファイアウォール、ゲートウェイ・サーバなどのインターネット・サーバである場合に適用できます。

ネットワークの性能は、リソース要求に対応できるだけのリソースが確保できない場合に影響を受けます。そのような状況は、次の2通りの場合に発生します。

- 1つまたは複数のハードウェアまたはソフトウェアのネットワーク構成要素に問題が発生した場合
- 作業負荷(ネットワーク・トラフィック)が使用可能なリソースの許容量を定常的に超過しているが、すべて順調に動作しているように見える場合

この問題は、どちらもネットワークのチューニングの問題ではありません。ネットワークに問題がある場合は、その問題を究明して解決しなければなりません。ネットワーク・トラフィックが高い場合(たとえば、システムが100% ビジーのときに、Web サーバのヒット・レートが最大値に達している場合)、ネットワークの設計を変更して負荷をさらに分散させるか、ネットワーク・クライアントの数を減らすか、ネットワークの負荷を処理するシステムの数を増やさなければなりません。

ネットワークの問題を解決する方法についての詳細は、『ネットワーク・プログラミング・ガイド』および『ネットワーク管理ガイド：接続編』を参照してください。

表 10-2 に、ネットワーク・サブシステムのチューニングのガイドラインと、性能上の利点と欠点を示します。

表 10-2: ネットワークのチューニング・ガイドライン

ガイドライン	性能上の利点	欠点
TCP 制御ブロックを検索するためにカーネルが使用するハッシュ・テーブルのサイズを増やす (10.2.1 項)。	TCP 制御ブロックの検索速度が速くなり、接続速度が向上する。	固定メモリの量が少し増加する。
TCP ハッシュ・テーブルの数を増やす (10.2.2 項)。	SMP システムでのハッシュ・テーブルのロック争奪が減少する。	固定メモリの量が少し増加する。
ソケット・リッスン・キューでのパーシャル TCP 接続の限界値を大きくする (10.2.3 項)。	大量の接続を処理するシステムで、スループットと応答速度が向上する。	保留された接続がキュー内にとどまった場合に、メモリを消費する。
送信接続ポートの数を増やす (10.2.4 項)。	同時に作成できる送信接続の数が増える。	なし

表 10-2: ネットワークのチューニング・ガイドライン (続き)

ガイドライン	性能上の利点	欠点
送信接続ポートの範囲を変更する (10.2.5 項)。	特定の範囲のポートが使用できるようになる。	なし
PMTU 検出を無効にする (10.2.6 項)。	多数のクライアントからのリモート・トラフィックを処理するサーバの効率が改善される。	LAN トラフィックに対するサーバの効率が低下することがある。
IP 入力キューの数を増やす (10.2.7 項)。	SMP システムでの IP 入力キューのロック争奪が減少する。	なし
mbuf クラスタの圧縮を有効にする (10.2.8 項)。	ネットワークのメモリ割当て効率改善される。	なし
TCP の keepalive 機能を有効にする (10.2.9 項)。	アクティブでないソケット接続をタイムアウトにできる。	なし
カーネル・インタフェース別名テーブルのサイズを大きくする (10.2.10 項)。	多数のドメイン名のサービスを行うシステムでの IP アドレス検索速度が向上する。	固定メモリの量が少し増加する。
パーシャル TCP 接続のタイムアウトを速くする (10.2.11 項)。	クライアントがソケット・リッスン・キューを満杯にすることがなくなる。	時間制限が短くなるため、成立する可能性があった接続が早期に切断されることがある。
接続の最後で、TCP 接続コンテキストのタイムアウトを速くする (10.2.12 項)。	接続リソースが早く解放される。	タイムアウト限界値を小さくすると、データが破損する可能性が高くなる。このガイドラインを適用する際は注意が必要。
TCP 再送レートを下げる (10.2.13 項)。	早期の再送がなくなり、混雑が減少する。	長い再送時間がすべての状況に適しているとは限らない。
TCP データの即時肯定応答を有効にする (10.2.14 項)。	一部の接続でネットワークの性能が向上する。	ネットワークの帯域幅に悪影響を及ぼすことがある。
TCP セグメントの最大サイズを大きくする (10.2.15 項)。	1 つのパケットで送信できるデータの量が増える。	ルータの境界で断片化が発生することがある。

表 10-2: ネットワークのチューニング・ガイドライン (続き)

ガイドライン	性能上の利点	欠点
送信および受信ソケット・バッファのサイズを大きくする (10.2.3 項)。	1 つのソケットにバッファリングされる TCP パケットが増える。	バッファ領域が使用されている場合に、使用可能なメモリが減少することがある。
UDP ソケットの送信および受信バッファのサイズを大きくする (10.2.16 項)。	UDP パケットのドロップ防止に効果がある。	バッファ領域が使用されている場合に、使用可能なメモリが減少することがある。
UBC に十分なメモリを割り当てる (11.1.3 項)。	ディスクの入出力性能が向上する。	プロセスが使用できる物理メモリが減少することがある。
ソケット・バッファの最大サイズを大きくする (10.2.17 項)。	ソケット・バッファのサイズを大きくできる。	メモリ・リソースを消費する。
入力パケットのドロップを防止する (10.2.18 項)。	高いネットワーク負荷に対応できる。	なし

以降の項では、上記チューニングのガイドラインについて詳しく説明します。

カーネル・サブシステム属性を変更する方法は、第 3 章を参照してください。

10.2.1 TCP 制御ブロックの検索速度を速くする

カーネルが伝送制御プロトコル (TCP) の制御ブロックを検索するために使用する、ハッシュ・テーブルのサイズを変更できます。inet サブシステムの `tcbhashsize` 属性は、カーネルの TCP 接続テーブル内のハッシュ・パケットの数 (`inpcb` ハッシュ・テーブル内のパケット数) を指定します。

性能上の利点と欠点

カーネルは、受信するすべての TCP パケットについて接続ブロックを検索しなければならないため、テーブルのサイズを大きくすると検索速度が上がり、性能が改善されます。これにより、固定メモリの量が少し増加します。

`tcbhashsize` 属性は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

インターネット・サーバがある場合は、カーネルの TCP 接続テーブル内のハッシュ・パケットの数を増やします。

推奨値

`tcblhashsize` 属性の省略時の値は 512 です。インターネット・サーバの場合は、`tcblhashsize` 属性の値を 16384 にしてください。

10.2.2 TCP ハッシュ・テーブルの数を増やす

カーネルは、受信するすべての伝送制御プロトコル (TCP) パケットについて接続ブロックを検索しなければならないため、SMP システムの TCP ハッシュ・テーブルがボトルネックになるおそれがあります。テーブルの数を増やすと負荷が分散されるため、性能を改善できます。`inet` サブシステムの `tcblhashnum` 属性は、TCP ハッシュ・テーブルの数を指定します。

性能上の利点と欠点

SMP システムでは、カーネルが TCP 制御ブロックの検索に使用するハッシュ・テーブルの数を増やすと、ハッシュ・テーブルのロック争奪を削減できます。これにより、固定メモリの量が少し増加します。

`tcblhashnum` 属性を変更したときには、システムのリブートが必要です。

チューニングするかどうかの判断

SMP システムのインターネット・サーバの場合は、TCP ハッシュ・テーブルの数を増やします。

推奨値

`tcblhashnum` 属性の最小値および省略時の値は 1 です。最大値は 64 です。負荷の高いインターネット・サーバの SMP システムでは、`tcblhashnum` 属性の値を 16 に増やします。この属性の値を大きくする場合は、ハッシュ・テーブルのサイズも大きくしてください。詳細は 10.2.1 項を参照してください。

`tcblhashnum` 属性の値を `inet` サブシステムの `ipqs` 属性と等しくすることをお勧めします。詳細は 10.2.7 項を参照してください。

10.2.3 TCP ソケット・リッスン・キューの限界値をチューニングする

ソケット・リッスン・キューの限界値を大きくすることで、性能が改善されることがあります (TCP の場合のみ)。`socket` サブシステムの `somaxconn` 属性は、各サーバ・ソケットに対して保留される TCP 接続の最大数 (ソケット・リッスン・キューの限界値) を指定します。リッスン・キューの接続限

界値が小さ過ぎると、受信接続要求が失われることがあります。TCP 接続の保留は、インターネットでのパケット紛失や、サービス・アタックの拒否によって生じることがあるので注意してください。

socket サブシステムの `sominconn` 属性は、各サーバ・ソケットに対して保留される TCP 接続 (バックログ) 数の最小値を指定します。この属性は、新たな要求を破棄しないで、同時に処理できる SYN パケットの数を制御します。`sominconn` 属性の値は、アプリケーションに固有のバックログの値より優先されます。アプリケーションに固有のバックログの値は、一部のサーバ・ソフトウェアに対しては小さ過ぎることがあります。

性能上の利点と欠点

ドロップを少なくして、スループットと応答速度を改善するには、`somaxconn` 属性の値を増やします。

アプリケーションを再コンパイルすることなく性能を改善したい場合、またはインターネット・サーバの場合は、`sominconn` 属性の値を大きくします。この属性の値を大きくすると、エラーの TCP SYN パケットでクライアントのソケット・リッスン・キューが満杯になるのを防止できます。

`somaxconn` および `sominconn` 属性の値は、システムをリブートすることなく変更できます。ただし、すでにオープンしているソケットでは、アプリケーションを再起動するまで、以前のソケット限界値が使用されます。

チューニングするかどうかの判断

インターネット・サーバを使用している場合や、負荷の高いシステムで保留される接続が多く、また大量の接続を生成するアプリケーションを実行している場合は、ソケット・リッスン・キューの限界値を大きくします。

`sobacklog_hiwat`、`sobacklog_drops`、`somaxconn_drops` 属性をモニタリングして、ソケット・キューがオーバフローしていないか調べます。オーバフローしている場合は、ソケット・リッスン・キューの限界値を大きくする必要があります。詳細は、10.1.1 項を参照してください。

推奨値

`somaxconn` 属性の省略時の値は 1024 です。インターネット・サーバの場合は、`somaxconn` 属性の値を最大値の 65535 にします。

sominconn 属性の省略時の値は 0 です。アプリケーションを再コンパイルすることなく性能を改善したい場合や、インターネット・サーバの場合は、sominconn 属性の値を最大値の 65535 にします。

クライアントのソケット・リッスン・キューがエラーの TCP SYN パケットで満杯になっている場合は、他のユーザを効率的にキューからブロックして、sominconn 属性の値を 65535 に増やします。システムで受信 SYN パケットのドロップが続く場合は、inet サブシステムの tcp_keepinit 属性を 30 (15 秒) に減らします。

sominconn 属性の値は somaxconn 属性の値と同じにしてください。

10.2.4 送信接続ポート数を増やす

TCP または UDP アプリケーションが送信接続を作成する場合、カーネルは予約されていないポート番号を各接続に動的に割り当てます。カーネルは、inet サブシステムの ipport_userreserved_min 属性の値から ipport_userreserved 属性の値までの範囲からポート番号を選択します。省略時の属性値を使用する場合、同時に接続できる送信接続の数は 3976 に制限されます。

性能上の利点

ポートの数を増やすと、TCP および UDP アプリケーションに対するポート数が増えます。

ipport_userreserved 属性の値は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

システムに多数の送信ポートが必要な場合は、ipport_userreserved 属性の値を大きくします。

推奨値

ipport_userreserved 属性の省略時の値は 5000 です。この場合、省略時のポート数は 3976 (5000 から 1024 を引いた値) になります。

システムがプロキシ・サーバ (Squid キャッシュ・サーバやファイアウォール・システムなど) で、同時接続が 4000 を超える場合は、ipport_userreserved 属性の値を最大値の 65000 まで増やします。

`ipport_userreserved` 属性の値を 5000 未満に減らしたり、65000 より大きい値に増やさないでください。

また、送信接続ポートの範囲を変更することもできます。詳細は 10.2.5 項を参照してください。

10.2.5 送信接続ポートの範囲を変更する

TCP または UDP アプリケーションが送信接続を作成する場合、カーネルは予約されていないポート番号を各接続に動的に割り当てます。カーネルは、`inet` サブシステムの `ipport_userreserved_min` 属性の値から `ipport_userreserved` 属性の値までの範囲からポート番号を選択します。これらの属性の省略時の値を使用した場合、送信ポートの範囲は 1024 ~ 5000 になります。

性能上の利点と欠点

送信接続の範囲を変更すると、TCP および UDP アプリケーションには指定した範囲のポートが割り当てられます。

`ipport_userreserved_min` および `ipport_userreserved` 属性は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

特定の範囲の送信ポートを使用したい場合は、`ipport_userreserved_min` 属性および `ipport_userreserved` 属性の値を変更します。

推奨値

`ipport_userreserved_min` 属性の省略時の値は 1024 です。

`ipport_userreserved` の省略時の値は 5000 です。属性の最大値は、どちらも 65000 です。

`ipport_userreserved` 属性の値は 5000 未満にしないでください。また、`ipport_userreserved_min` 属性の値は 1024 未満にしないでください。

10.2.6 PMTU の検出を無効にする

サーバ間で伝送されるパケットは、ルータやパケット・サイズの小さいネットワーク（イーサネット・ネットワークなど）を介して伝送しやすくするために、特定のサイズのユニットに分割されます。`inet` サブシステムの

`pmtu_enabled` 属性が有効 (1 に設定。省略時の動作) の場合、システムはサーバ間の PMTU (Path Maximum Transmission Unit) 値の共通の最大値を調べ、それをユニットのサイズとして使用します。また、システムは、サーバに接続しようとしたクライアント・ネットワークごとにルーティング・テーブルのエントリを作成します。

性能上の利点と欠点

サーバが多数のリモート・クライアント間のトラフィックを処理する場合、PMTU 検出を無効にしておくと、カーネルのルーティング・テーブルのサイズが小さくなり、サーバの効率が良くなります。ただし、ローカル・トラフィックとリモート・トラフィックを処理するサーバでは、PMTU 検出を無効にすると帯域幅が低下することがあります。

`pmtu_enabled` 属性の値は、システムをリブートすることなく変更できません。

チューニングするかどうかの判断

パスを分断する FDDI ツー・イーサネットのブリッジがあるような場合は、PMTU 検出を使用すると、あるサイズよりも大きなパケットが見えなくなり、データ転送ができなくなることがあります。このような場合は、PMTU 検出を無効にします。また、多数のリモート・クライアント間のトラフィックを処理するサーバがある場合や、性能の低いインターネット・サーバを使用していて、ルーティング・テーブルのエントリ数が 1000 を超えている場合にも、PMTU 検出を無効にしてください。

10.2.7 IP 入力キューの数を増やす

`inet` サブシステムの `ipqs` 属性は、IP 入力キューの数を指定します。

性能上の利点と欠点

IP 入力キューの数を増やすと、負荷が分散されるため、キューのロック争奪が減少します。

`ipqs` 属性を変更したときには、システムのリブートが必要です。

チューニングするかどうかの判断

インターネット・サーバとして使用している SMP システムでは、IP 入力キューの数を増やします。

推奨値

インターネット・サーバとして使用している SMP システムでは、`ipqs` 属性の値を 16 に増やします。最大値は 64 です。

`ipqs` 属性の値は、`inet` サブシステムの `tcbhashnum` 属性の値と同じにすることを勧めます。詳細は、10.2.2 項を参照してください。

10.2.8 mbuf クラスタの圧縮を有効にする

`socket` サブシステムの `sbcompress_threshold` 属性は、`mbuf` クラスタがソケット・レイヤで圧縮されるかどうかを制御します。特に指定しなければ、`mbuf` クラスタは圧縮されません (`sbcompress_threshold` の設定が 0)。

性能上の利点

`mbuf` クラスタを圧縮すると、使用可能な `mbuf` クラスタをプロキシ・サーバが使い果たしてしまうのを防止できます。

`sbcompress_threshold` 属性は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

プロキシ・サーバを使用している場合は、`mbuf` クラスタの圧縮を有効にします。プロキシ・サーバのシステムでは、イーサネットの代わりに FDDI を使用している場合に、使用可能な `mbuf` クラスタを使い果たしてしまうことがよくあります。

`mbuf` クラスタとして使用しているメモリを調べるには、`netstat -m` コマンドを実行します。次の例は、128 MB のメモリを備え、`mbuf` クラスタの圧縮を有効にしていないファイアウォール・サーバの例です。

```
# netstat -m
2521 Kbytes for small data mbufs (peak usage 9462 Kbytes)
78262 Kbytes for mbuf clusters (peak usage 97924 Kbytes)
8730 Kbytes for sockets (peak usage 14120 Kbytes)
9202 Kbytes for protocol control blocks (peak usage 14551
  2 Kbytes for routing table (peak usage 2 Kbytes)
  2 Kbytes for socket names (peak usage 4 Kbytes)
  4 Kbytes for packet headers (peak usage 32 Kbytes)
```

```
39773 requests for mbufs denied
      0 calls to protocol drain routines
98727 Kbytes allocated to network
```

上記の例では、39773 個のメモリ要求が拒否されたことを示しています。この値は 0 でなければならないので、これは問題があることを示しています。また、この例では、78 MB のメモリが `mbuf` クラスタに割り当てられ、98 MB のメモリがネットワーク・サブシステムによって消費されていることもわかります。

推奨値

`mbuf` クラスタの圧縮を有効にするには、`socket` サブシステムの `sbcompress_threshold` 属性の省略時の値を変更します。パケット・サイズがこの値より小さい場合、パケットは既存の `mbuf` クラスタにコピーされます。プロキシ・サーバの場合は、この属性に 600 を指定します。

`sbcompress_threshold` 属性の値を大きくして 600 にすると、ネットワーク・サブシステムに割り当てられているメモリはただちに 18 MB まで減少します。これは、カーネル・ソケット・バッファのインタフェースで圧縮が行われ、メモリの使用効率が改善されるためです。

10.2.9 TCP の `keepalive` 機能を有効にする

`keepalive` 機能を有効にすると、接続をアクティブの状態に保つために、接続されたソケット上でメッセージが定期的に伝送されます。正常に終了していないソケットは、`keepalive` の時間間隔が経過した時点で消去されます。`keepalive` を有効にしていない場合、このようなソケットはシステムをリブートするまで消えません。

アプリケーションでは、`setsockopt` 関数の `SO_KEEPAVIVE` オプションを設定することで、ソケットの `keepalive` 機能を有効にします。自分で `keepalive` を設定しないプログラムについて指定を変更する場合、またはアプリケーションのソースにアクセスできない場合は、`inet` サブシステムの `tcp_keepalive_default` 属性を用いて `keepalive` 機能を有効にします。

性能上の利点

`keepalive` 機能は、`keepalive` の時間間隔が経過した時点で、正常に終了していないソケットを消去します。

`tcp_keepalive_default` 属性は、システムをリブートすることなく変更できます。ただし、すでに存在しているソケットは、アプリケーションを再起動するまで以前の動作を続けます。

チューニングするかどうかの判断

`keepalive` 機能が必要で、ソース・コードにアクセスできない場合に `keepalive` を有効にします。

推奨値

自分で `keepalive` を設定しないプログラムについて指定を変更する場合、またはアプリケーションのソース・コードにアクセスできない場合は、`inet` サブシステムの `tcp_keepalive_default` 属性に 1 を設定して、すべてのソケットの `keepalive` を有効にします。

`keepalive` を有効にしている場合は、ソケットに対して次の `inet` サブシステム属性も構成できます。

- `tcp_keepidle` 属性は、`keepalive` プローブを送信するまでのアイドル時間の長さを指定します (0.5 秒単位で指定)。省略時の間隔は 2 時間です。
- `tcp_keepintvl` 属性は、`keepalive` プローブを再送する時間間隔 (0.5 秒単位) を指定します。省略時の間隔は 75 秒です。
- `tcp_keepcnt` 属性は、接続を切るまでに送信する `keepalive` プローブの最大数を指定します。省略時の値は 8 プローブです。
- `tcp_keepinit` 属性は、初期接続がタイムアウトになるまでの最大時間を 0.5 秒単位で指定します。省略時の値は 75 秒です。

アプリケーションでは、`setsockopt()` コールを用いて、ソケットごとに属性の値を変更できます。

10.2.10 IP アドレスの検索速度を速くする

`inet` サブシステムの `inifaddr_hsize` 属性は、カーネル・インタフェース別名テーブル (`in_ifaddr`) 内のハッシュ・バケット数を指定します。

システムが多数のサーバ・ドメイン名に対するサーバとして使用され、ドメイン名がそれぞれ一意の IP アドレスにバインドされている場合、受信したパ

ケットを正しいサーバ・アドレスと照合する処理では、ハッシュ・テーブルを使用して IP アドレスの検索処理を高速化しています。

性能上の利点と欠点

テーブル内のハッシュ・バケットの数を増やすと、大量の別名を使用するシステムの性能を改善できます。

`inifaddr_hsize` 属性は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

大量の別名を使用しているシステムでは、カーネル・インタフェース別名テーブル内のハッシュ・バケット数を増やします。

推奨値

`inet` サブシステムの `inifaddr_hsize` 属性の省略時の値は 32 です。
また、最大値は 512 です。

最良の性能を得るために、`inifaddr_hsize` 属性の値は、最も近い 2 の累乗に切り下げられます。インタフェース IP の別名を 500 個以上使用している場合は、最大値の 512 にします。別名が 250 個より少ない場合は、省略時の 32 を使用します。

10.2.11 TCP のパーシャル接続タイムアウトの限界値を小さくする

`inet` サブシステムの `tcp_keepinit` 属性は、部分的に確立された TCP 接続がタイムアウトになるまでにソケット・リッスン・キューにとどまる時間を指定します。パーシャル接続では、リッスン・キューのロットが消費され、`SYN_RCVD` 状態の接続がキューに入れられます。

性能上の利点と欠点

`tcp_keepinit` 属性の値を小さくすると、パーシャル接続が早くタイムアウトになります。

`tcp_keepinit` 属性の値は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

`somaxconn_drops` 属性の値が頻繁に増加しない限り、TCP パーシャル接続のタイムアウト限界値を変更する必要はありません。頻繁に増加する場合は、`tcp_keepinit` 属性の値を小さくします。

推奨値

`tcp_keepinit` 属性の値は 0.5 秒単位で指定します。省略時の値は 150 単位 (75 秒) です。`sominconn` 属性の値が 65535 の場合、`tcp_keepinit` 属性には省略時の値を使用してください。

`tcp_keepinit` 属性に設定する値が小さすぎないようにしてください。この値が小さすぎると、低速のネットワーク・パスや多数のパケットが失われるネットワーク・パスで、クライアントとの接続が早期に切断されてしまうおそれがあります。20 単位 (10 秒) 未満の値は設定しないようにしてください。

10.2.12 TCP 接続コンテキストのタイムアウト限界値を小さくする

TCP プロトコルには、MSL (Maximum Segment Lifetime) という概念があります。TCP 接続は、`TIME_WAIT` 状態になったときには、MSL の値の 2 倍だけこの状態にとどまらなければなりません。そうしないと、その後の接続でデータ未検出のエラーが発生する可能性があります。`inet` サブシステムの `tcp_msl` 属性によって、TCP セグメントの最大存在期間と `TIME_WAIT` 状態のタイムアウト値が決まります。

性能上の利点と欠点

`tcp_msl` 属性の値を小さくすると、接続の終わりで TCP 接続コンテキストのタイムアウトが早くなります。ただし、これによりデータが破損する可能性が高くなります。

`tcp_msl` 属性は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

通常は、TCP 接続コンテキストのタイムアウト限界値を変更する必要はありません。

推奨値

`tcp_msl` 属性の値は、0.5 秒単位で設定します。省略時の値は 60 単位 (30 秒) です。これは、TCP 接続が 60 秒間 (MSL の値の 2 倍) だけ `TIME_WAIT` 状態にとどまるということです。状況によっては、`TIME_WAIT` 状態のタイムアウト値として、省略時の値 (60 秒) では長すぎるため、`tcp_msl` 属性の値を小さくして、接続リソースが省略時よりも早く解放されるようにすることがあります。

ネットワークの設計および動作と TCP プロトコルについて完全に理解するまでは、`tcp_msl` 属性の値を小さくしないでください。省略時の値を使用することをお勧めします。それ以外の値を使用すると、データが破損するおそれがあります。

10.2.13 TCP 再送レートを下げる

`inet` サブシステムの `tcp_rexmit_interval_min` 属性は、最初の TCP 再送を行う前に待つ時間の最小値です。

性能上の利点と欠点

`tcp_rexmit_interval_min` 属性の値を大きくすると、TCP 再送レートは低くなり、混雑が減って性能が改善されます。

`tcp_rexmit_interval_min` 属性は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

すべての接続で再送時間を長くする必要はありません。通常、省略時の値で十分です。ただし、広域ネットワーク (WAN) の中には、省略時の再送間隔では短すぎて、再送のタイムアウトが早く発生しすぎることがあります。これにより、パケットの伝送が重複したり、TCP の混雑制御アルゴリズムが誤って起動されるおそれがあります。

再送をチェックするには、`netstat -p tcp` コマンドを用いて `data packets retransmitted` という出力を調べます。`netstat` コマンドの詳細は、2.4.5 項を参照してください。

推奨値

`tcp_rexmit_interval_min` 属性は、0.5 秒単位で指定します。省略時の値は 2 単位 (1 秒) です。

1 単位より小さい値は指定しないでください。TCP アルゴリズムを完全に理解するまでは、この属性を変更しないでください。

10.2.14 TCP データの肯定応答の遅延を無効にする

特に指定しない限り、システムは TCP データの肯定応答を遅延させます。`inet` サブシステムの `tcpnodelack` 属性によって、システムが TCP データの肯定応答を遅延させるかどうかが決まります。

性能上の利点と欠点

TCP データの遅延を無効にすると、性能が改善されることがあります。ただし、ネットワークの帯域幅に悪影響を及ぼすことがあります。

`tcpnodelack` 属性は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

通常、`tcpnodelack` 属性の省略時の値で十分です。ただし、接続によっては (ループバックなど)、この遅延により性能が低下することもあります。`tcpdump` コマンドを用いて、遅れが大きすぎないかチェックしてください。

推奨値

`tcpnodelack` の省略時の値は 0 です。TCP 肯定応答の遅延を無効にするには、`tcpnodelack` 属性の値を 1 にしてください。

10.2.15 TCP セグメントの最大サイズを大きくする

`inet` サブシステムの `tcp_mssdflt` 属性は、TCP セグメントの最大サイズを指定します。

性能上の利点と欠点

TCP セグメントの最大サイズを大きくすると、1 つのソケットで送信できるデータの量が増えますが、ルータの境界で断片化が発生することがあります。

`tcp_mssdflt` 属性は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

通常は、TCP セグメントの最大サイズを変更する必要はありません。

推奨値

`tcp_mssdflt` 属性の省略時の値は 536 です。この値は 1460 まで大きくすることができます。

10.2.16 UDP ソケットの送受信バッファを大きくする

`inet` サブシステムの `udp_sendspace` 属性は、Internet UDP (User Datagram Protocol) ソケットの省略時の送信バッファ・サイズを指定します。`inet` サブシステムの `udp_recvspace` 属性は、UDP ソケットの省略時の受信バッファ・サイズを指定します。

性能上の利点と欠点

UDP 送信および受信ソケット・バッファを大きくすると、1つのソケットにバッファリングできる UDP パケットの数が多くなります。ただし、値を大きくすると、アプリケーションがバッファを使用する(データの送受信)際に使用されるメモリも多くなります。

注意

UDP 属性は、ネットワーク・ファイル・システム (NFS) の性能には影響しません。

`udp_sendspace` および `udp_recvspace` 属性は、システムをリブートすることなく変更できます。ただし、新しい UDP ソケット・バッファの値を使用するには、アプリケーションを再起動しなければなりません。

チューニングするかどうかの判断

`netstat -p udp` コマンドを使用して、ソケット・フルをチェックします。出力に `full sockets` がたくさん表示される場合は、`udp_recvspace` 属性の値を大きくしてください。

推奨値

`udp_sendspace` の省略時の値は 9 KB (9216 バイト) です。 `udp_recvspace` の省略時の値は 40 KB (42240 バイト) です。 これらの属性の値は、64 KB まで大きくすることができます。

10.2.17 ソケット・バッファの最大サイズを大きくする

`socket` サブシステムの `sb_max` 属性は、ソケット・バッファの最大サイズを指定します。

性能上の利点と欠点

ソケット・バッファの最大サイズを大きくすると、バッファ・サイズを大きくして効果のあるアプリケーションの場合は、性能が改善できることがあります。

`sb_max` 属性の値は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

大きなソケット・バッファが必要な場合は、ソケット・バッファ・サイズの最大値を大きくします。

推奨値

`sb_max` 属性の省略時の値は 128 KB です。 送信および受信ソケット・バッファのサイズを大きくする前に、この値を大きくしてください。

10.2.18 入力パケットのドロップを防止する

ネットワークの負荷が高いために IP の入力キューがオーバーフローすると、入力パケットがドロップします。

`inet` サブシステムの `ipqmaxlen` 属性は、入力パケットがドロップすることなく使用できる IP 入力キュー (`ipintrq`) の最大長 (バイト) を指定します。 `ifqmaxlen` 属性は、パケットがドロップすることなくネットワーク・アダプタのキューに入れることができる出力パケットの数を指定します。

性能上の利点と欠点

IP 入力キューを大きくすると、パケットのドロップを防止できます。

ipqmaxlen および ifqmaxlen 属性の値は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

システムでパケットのドロップが発生している場合は、ipqmaxlen および ifqmaxlen 属性の値を大きくします。入力パケットのドロップをチェックするには、dbx を用いて ipintrq カーネル構造体を調べます。ifq_drops フィールドが 0 でない場合は、入力パケットのドロップが発生しています。次に例を示します。

```
# dbx -k /vmunix
(dbx)print ipintrq
struct {
    ifq_head = (nil)
    ifq_tail = (nil)
    ifq_len = 0
    ifq_maxlen = 512
    ifq_drops = 128
    .
    .
    .
}
```

netstat -id コマンドを用いて、出力パケットのドロップをモニタリングします。出力の中で、インタフェースの Drop 欄にゼロ以外の値がないか調べます。次の例は、ネットワーク・インタフェースの tul で 579 個の出力パケットがドロップしていることを示しています。

```
# netstat -id
```

Name	Mtu	Network	Address	Ipkts	Ierrs	Opkts	Oerrs	Coll	Drop
fta0	4352	link	08:00:2b:b1:26:59	41586	0	39450	0	0	0
fta0	4352	DLI	none	41586	0	39450	0	0	0
fta0	4352	10	fratbert	41586	0	39450	0	0	0
tul	1500	link	00:00:f8:23:11:c8	2135983	0	163454	13	3376	579
tul	1500	DLI	none	2135983	0	163454	13	3376	579
tul	1500	red-net	ratbert	2135983	0	163454	13	3376	579
.

さらに、netstat -p ip を使用して、lost packets due to resource problems フィールドまたは no memory or interface queue was full フィールドにゼロ以外の値がないか調べます。次に例を示します。

```
# netstat -p ip
ip:
    259201001 total packets received
    0 bad header checksums
    0 with size smaller than minimum
    0 with data size < data length
    0 with header length < data size
    0 with data length < header length
    25794050 fragments received
```

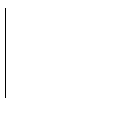


```
0 fragments dropped (duplicate or out of space)
802 fragments dropped after timeout
0 packets forwarded
67381376 packets not forwardable
    67381376 link-level broadcasts
0 packets denied access
0 redirects sent
0 packets with unknown or unsupported protocol
170988694 packets consumed here
160039654 total packets generated here
0 lost packets due to resource problems
4964271 total packets reassembled ok
2678389 output packets fragmented ok
14229303 output fragments created
0 packets with special flags set
```

推奨値

ipqmaxlen および ifqmaxlen 属性の省略時の値および最小値は 1024 です。最大値は 65535 です。多くの構成では、省略時の値で十分です。パケットのドロップが発生している場合に限り、値を増やしてください。

システムでパケットのドロップが発生している場合は、ipqmaxlen および ifqmaxlen 属性の値を、パケットがドロップしなくなるまで増やします。たとえば、省略時の値から 2000 に増やします。



ファイル・システム性能の管理

ファイル・システム性能をチューニングするには、アプリケーションとユーザがどのようにディスクの入出力を実行するかを理解していなければなりません (1.8 節で説明)。また、使用するファイル・システムがプロセスとの間でメモリを共有する方法 (第 12 章で説明) について理解していなければなりません。この情報を利用すると、この章で説明するカーネル・サブシステム属性の値を変更することで、ファイル・システムの性能を改善できます。

この章では、次のような項目をチューニングする方法について説明します。

- ファイル・システムが使用するキャッシュ (11.1 節)
- Advanced File System (AdvFS) (11.2 節)
- UNIX ファイル・システム (UFS) (11.3 節)
- ネットワーク・ファイル・システム (NFS) (11.4 節 と 第 5 章)

11.1 キャッシュのチューニング

カーネルは、最近アクセスされたデータをメモリにキャッシュ (一時的に保管) します。データは何度も再使用され、ディスクからデータを取得するよりもメモリから取得するほうが高速なので、データをキャッシュすると効率が上がります。カーネルがデータを必要としているときには、まずデータがキャッシュされているかどうかをチェックします。データがキャッシュされていれば、すぐにそのデータが返されます。キャッシュされていなければ、ディスクからデータが取得されキャッシュされます。データがキャッシュされた後で再使用される場合に、ファイル・システムの性能が向上します。

キャッシュ内に見つかったデータをキャッシュ・ヒットといいます。キャッシュされたデータの効率は、キャッシュのヒット率で計測します。キャッシュ内に見つからなかったデータはキャッシュ・ミスといいます。

キャッシュされるデータには、ファイルに関する情報、ユーザまたはアプリケーションのデータ、メタデータ (ファイルなどのオブジェクトを記述す

るデータ) などがあります。キャッシュされるデータのタイプを、以下のリストに示します。

- ファイル名と、それに対応する vnode は namei キャッシュにキャッシュされます (11.1.2 項)。
- UFS のユーザおよびアプリケーションのデータと、AdvFS のユーザおよびアプリケーションのデータとメタデータは、ユニファイド・バッファ・キャッシュ (UBC) にキャッシュされます (11.1.3 項)。
- UFS ファイルのメタデータは、メタデータ・バッファ・キャッシュにキャッシュされます (11.1.4 項)。
- AdvFS のオープン・ファイル情報はアクセス構造体にキャッシュされます (11.1.5 項)。

11.1.1 キャッシュ統計情報のモニタリング

表 11-1 では、キャッシュ情報の表示と管理に使用するコマンドを説明します。

表 11-1: キャッシュ情報を表示するツール

ツール	説明	参照先
(dbx) print processor number	namei キャッシュの統計 情報を表示する。	11.1.2 項
vmstat	仮想メモリの統計情報 を表示する。	11.1.3 項 および 12.3.1 項
(dbx) print bio_stats	メタデータ・バッファ・ キャッシュの統計情報 を表示する。	11.3.2.3 項

11.1.2 namei キャッシュのチューニング

仮想ファイル・システム (VFS) は、下位ファイル・システム・レイヤから抽出した統一カーネル・インタフェースをアプリケーションに提供します。その結果、ファイルのアクセスでのファイル・システムの種類の違いは、ユーザには見えなくなります。

VFS は **vnode** という構造体を用いて、マウントされたファイル・システム内の各オープン・ファイルに関する情報を保持します。アプリケーションがファイルの読み取りまたは書き込み要求を発行すると、VFS は vnode 情報を

用いて要求を変換し、それを適切なファイル・システムに渡します。たとえば、アプリケーションがファイルに対する `read()` システム・コールを発行すると、VFS は `vnode` 情報を用いてシステム・コールを、ファイルが置かれているファイル・システムに対して適切な呼び出しに変換します。UFS の場合は `ufs_read()`、AdvFS の場合は `advfs_read()`、NFS でマウントしたファイル・システムにファイルがある場合は `nfs_read()` コールに変換し、その要求を適切なファイル・システムに渡します。

VFS は最近アクセスされたファイル名とそれに対応する `vnode` を **namei** キャッシュにキャッシュします。ファイル・システムが再使用され、ファイル名とそれに対応する `vnode` が `namei` キャッシュにある場合に、ファイル・システムの性能が向上します。

以下のリストでは、`namei` キャッシュに関連する `vfs` サブシステム属性を説明します。

関連する属性

- `vnode_deallocation_enable` — システムの要求に応じて `vnode` を動的に割り当てるかどうかを指定します。

値 0 または 1
省略時の値: 1 (有効)

無効にすると、オペレーティング・システムは静的 `vnode` プールを使用します。性能を最大にするためには、`vnode` の動的割り当てを無効にしないでください。

- `name_cache_hash_size` — `namei` キャッシュのハッシュ・チェーン・テーブルのサイズをスロット数で指定します。

省略時の値: $2 * (148 + 10 * \text{maxusers}) * 11 / 10 / 15$

- `vnode_age` — 解放された `vnode` を再利用可能とするまでの経過時間を秒数で指定します。

値: 0 ~ 2,147,483,647
省略時の値: 120 秒

- `namei_cache_valid_time` — `namei` キャッシュが破棄されるまでにキャッシュ内にとどまる時間を秒数で指定します。

値: 0 ~ 2,147,483,647

省略時の値: 32MB 以上のシステムでは 1200 (秒) , 24 MB システムでは 30 (秒)

注意

namei キャッシュに関連する属性の値を増やす場合は、ファイルとディレクトリの情報をキャッシュするファイル・システム属性を増やすことも検討してください。AdvFS を使用する場合は、11.1.5 項を参照してください。UFS を使用する場合は、11.1.4 項を参照してください。

チューニングするかどうかの判断

namei キャッシュの統計情報を調べると、namei キャッシュに関連する属性の値を変更すべきかどうか判断できます。namei キャッシュの統計情報を調べるには、dbx print コマンドを入力してプロセッサ番号を指定し、nchstats データ構造体を調べます。たとえば、次のコマンドを入力します。

```
# /usr/ucb/dbx -k /vmunix /dev/mem  
(dbx) print processor_ptr[0].nchstats
```

次のような情報が表示されます。

```
struct {  
    ncs_goodhits = 18984  
    ncs_neghits = 358  
    ncs_badhits = 113  
    ncs_falsehits = 23  
    ncs_miss = 699  
    ncs_long = 21  
    ncs_badtimehits = 33  
    ncs_collisions = 2  
    ncs_unequaldups = 0  
    ncs_newentry = 697  
    ncs_newnegentry = 419  
    ncs_gnn_hit = 1653  
    ncs_gnn_miss = 12  
    ncs_gnn_badhits = 12  
    ncs_gnn_collision = 4  
    ncs_pad = {  
        [0] 0  
    }  
}
```

dbx print の出力に基づいて、namei キャッシュ関連の属性値をいつ変更すればよいかを、表 11-2 に示します。

表 11-2: namei キャッシュ関係の属性の値を変更する時期

条件	増やす値
$\frac{\text{ncs_goodhits} + \text{ncs_neghits}}{\text{ncs_goodhits} + \text{ncs_neghits} + \text{ncs_miss} + \text{ncs_falsehits}}$ の値が 80 パーセント未満	maxusers 属性または name_cache_hash_size 属性の値
ncs_badtimehits が ncs_goodhits の 0.1 パーセントよりも大きい	namei_cache_valid_time 属性および vnode_age 属性の値

name_cache_hash_size 属性、namei_cache_valid_time 属性、vnode_deallocation_enable 属性のいずれかを変更したときには、システムのリブートが必要です。vnode_age 属性の値は、システムをリブートすることなく変更できます。サブシステム属性の変更についての詳細は第 3 章を参照してください。

11.1.3 UBC のチューニング

ユニファイド・バッファ・キャッシュ (UBC) は、非固定メモリをプロセスと共有し、UFS のユーザおよびアプリケーション・データと、AdvFS のユーザおよびアプリケーション・データとメタデータをキャッシュします。ファイル・システムの性能は、データおよびメタデータが再使用されて UBC 内にあった場合に向上します。

関連する属性

UBC に関連する vm サブシステム属性を、以下のリストに示します。

- vm_ubcdirtypercent 属性 — UBC がディスクへの書き込みを開始するときの、ダーティ (変更されている) ページの割合の最小値を指定します。
値: 0 ~ 100
省略時の値: 10 パーセント
- ubc_maxdirtywrites 属性 — UBC 内のダーティ (変更されている) ページの数が vm_ubcdirtypercent 属性を超えたときに、vm サブシステムが実行する入出力操作の頻度 (1 秒あたりの回数) を指定します。
値: 0 ~ 2,147,483,647

省略時の値: 5 (1 秒あたりの回数)

- `ubc_maxpercent` 属性 — UBC が同時に使用できる物理メモリの最大割合を指定します。

値: 0 ~ 100

省略時の値: 100 パーセント

- `ubc_borrowpercent` 属性 — メモリの使用量がここで指定した割合を上回ると、UBC は `vm` サブシステムからメモリを借用していると思われる。UBC が借用ページをすべて返却するまで、ページングは発生しません。

値: 0 ~ 100

省略時の値: 20 パーセント

この値を増やすと、メモリが窮迫した場合にシステムのレスポンスが低下することがあります (たとえば、大規模なプロセスのワーキング・セット)。

- `ubc_minpercent` 属性 — UBC が使用できるメモリの最小割合を指定します。残りのメモリはプロセスと共有されます。

値: 0 ~ 100

省略時の値: 20 パーセント

この値を増やすと、UBC が使用できるメモリが大規模なプログラムですべて使用されることがなくなります。
入出力サーバでは、UBC が確実に十分なメモリを使用できるように、この値を増やすことを検討してください。

- `vm_ubcpagesteal` 属性 — ファイルの拡張に備えて用意するページの最小数を指定します。使用可能なページ数がこの値を下回ると、UBC はさらにページを流用して、ファイルの拡張要求に備えます。

値: 0 ~ 2,147,483,647

省略時の値: 24 (ファイル・ページ)

- `vm_ubcseqpercent` 属性 — 1 つのファイルをキャッシュするために使用する、UBC に割り当てられたメモリの最大量を指定します。

値: 0 ~ 100

省略時の値: UBC に割り当てられたメモリの 10 パーセント
アプリケーションが大規模なファイルを書き込む場合は、この値を増やすことを検討してください。

- `vm_abcseqstartpercent` 属性 — UBC がシーケンシャル・ファイル・アクセスを認識し、UBC LRU ページを流用してファイル用のページ要求を満たそうとし始めるしきい値を指定します。この値は、物理メモリの割合で表わした UBC のサイズです。

値: 0 ~ 100
省略時の値: 50 パーセント

アプリケーションが大規模なファイルを書き込む場合は、この値を大きくすることを検討してください。

注意

`abc_maxpercent` と `abc_minpercent` の値が近い場合、ファイル・システムの性能が低下するおそれがあります。

チューニングするかどうかの判断

UBC に割り当てられているメモリ容量が不十分な場合、ファイル・システムの性能が低下することがあります。UBC とプロセスがメモリを共有しているため、UBC 関連の属性値を変更すると、システムでページングが発生する原因になることがあります。`vmstat` コマンドの値を使用して、仮想メモリの統計情報を表示することができます。この統計情報は、UBC 関連の属性の値を変更する必要があるかどうかを判断する際に役に立ちます。`vmstat` の出力に基づいて、UBC 関連の属性の値をいつ変更すれば良いかを、表 11-3 に示します。

表 11-3: UBC 関連の属性の値を変更する時期

vmstat の出力にこの状況が頻 繁に表示される場合	処置
ページングが行われているが ページ・アウトがほとんど (ま たはまったく) ない	ubc_borrowpercent 属性の値を大き くする。
ページングおよびスワッピング	ubc_maxpercent 属性の値を小 さくする。
ページング	ubc_maxpercent 属性の値が vm_ubseqstartpercent 属性の値 より大きくなるようにし (省略時の設 定), また vm_ubcseqpercent 属性の値 が参照されているファイルより大き くなるように指定して, システムが空 リストのページを使用しないで UBC の ページを再使用するように強制する。
ページ・アウト	ubc_minpercent 属性の値を増やす

vmstat コマンドについての詳細は , 12.3.1 項 を参照してください。UBC
メモリ割り当てについての詳細は , 12.1.2.2 項 を参照してください。

この節で説明した UBC パラメータはすべて , システムをリブートするこ
となく変更できます。システム属性の変更についての詳細は , 第 3 章 を
参照してください。

注意

ランダムな入出力を大量に実行するアプリケーションの性能は ,
UBC を大きくしても改善されません。これは , ランダム入出力で
は , 次にアクセスする位置が予測できないためです。

11.1.4 メタデータ・バッファ・キャッシュのチューニング

カーネルはブート時に , メモリの一部をメタデータ・バッファ・キャッシュ
用に固定します。UFS ファイルのメタデータ (スーパーブロック , i ノード ,
間接ブロック , ディレクトリ・ブロック , シリンダ・グループ・サマリな
ど) は , メタデータ・バッファ・キャッシュにキャッシュされます。メタ
データが再使用されてメタデータ・バッファ・キャッシュ内にあった場合
に , ファイル・システムの性能が向上します。

関連する属性

メタデータ・バッファ・キャッシュに関連する `vfs` サブシステム属性を、以下のリストに示します。

- `bufcache` — カーネルがメタデータ・バッファ・キャッシュ用に固定するメモリのサイズを割合で指定します。

値: 0 ~ 50

省略時の値: 32 MB 以上のシステムでは 3 パーセント, 24 MB のシステムでは 2 パーセント

- `buffer_hash_size` — メタデータ・バッファ・キャッシュのハッシュ・チェーン・テーブルのサイズをスロット数で指定します。

値: 0 ~ 524,287

省略時の値: 2048 (スロット)

この値を増やすとバッファが分散され、平均的なチェーンの長さが短くなり、UFS の性能が向上しますが、プロセスおよび UBC が使用できるメモリの量は減少します。

`buffer_hash_size` 属性と `bufcache` 属性の値を変更したときには、システムのリブートが必要です。カーネル・サブシステム属性の変更についての詳細は、第 3 章 を参照してください。

チューニングするかどうかの判断

キャッシュのミス率が高い (キャッシュのヒット率が低い) 場合は、`bufcache` 属性のサイズを増やすことを検討してください。

キャッシュのミス率が高いかどうかを判断するには、`dbx print` コマンドを使用して `bio_stats` データ構造体を表示します。ミス率 (ブロックのミス数を、ブロック・ミスとブロック・ヒットの和で割った値) が 3 パーセントを上回る場合は、`bufcache` 属性の値を増やすことを検討してください。`bio_stats` データ構造体の表示についての詳細は、11.3.2.3 項 を参照してください。

`bufcache` 属性の値を増やすと、プロセスおよび UBC で使用できるメモリの量が減少するので注意してください。

11.1.5 AdvFS アクセス構造体のチューニング

ブート時にシステムは、カーネルによって固定されていない物理メモリの一部を、AdvFS アクセス構造体用に予約します。AdvFS はオープン・ファイルに関する情報と、オープンされていたが現在はクローズされているファイルに関する情報を、AdvFS 構造体にキャッシュします。ファイル情報が再使用され、アクセス構造体の中にある場合に、ファイル・システムの性能が向上します。

AdvFS アクセス構造体は、カーネル構成およびシステムの負荷に応じて動的に割り当てと割り当て解除が行われます。

関連する属性

- `AdvfsAccessMaxPercent` — AdvFS アクセス構造体に割り当てることができる、ページング可能メモリの最大容量を割合 (パーセント) で指定します。

値: 5 ~ 95

省略時の値: 25 パーセント

`AdvfsAccessMaxPercent` 属性の値は、システムをリブートすることなく変更できます。カーネル・サブシステム属性の変更についての詳細は、第 3 章 を参照してください。

チューニングするかどうかの判断

ユーザまたはアプリケーションが AdvFS ファイルを再使用する (たとえばプロキシ・サーバなど) 場合は、`AdvfsAccessMaxPercent` 属性の値を増やして、AdvFS アクセス構造体に割り当てるメモリを増やすことを検討してください。`AdvfsAccessMaxPercent` 属性の値を増やすと、プロセスに使用できるメモリの量が減り、ページングとスワッピングが頻繁に発生することがあるので注意してください。`vmstat` コマンドを使用すると、仮想メモリの統計情報を表示して、ページングとスワッピングが頻繁に発生しているかどうかを調べることができます。`vmstat` コマンドについての詳細は、12.3.1 項 を参照してください。

次のような場合は、AdvFS アクセス構造体用に予約されているメモリの量を減らすことを検討してください。

- AdvFS を使用していない場合

- 業務処理で同じファイルを何度もオープン、クローズ、再オープンしない場合
- 大容量メモリを使用している場合 (オープン・ファイルの数は、UBC メモリ使用量とプロセス・メモリ使用量ほどシステム・メモリの容量に比例して増えないため)。

11.2 AdvFS のチューニング

ここでは、AdvFS (Advanced File System) のキューをチューニングする方法、AdvFS 構成のガイドライン、AdvFS の情報を表示するために使用できるコマンドについて説明します。

AdvFS の機能と、AdvFS のセットアップおよび管理についての詳細は、『*AdvFS 管理ガイド*』を参照してください。

11.2.1 AdvFS の構成のガイドライン

ファイル・ドメイン内のボリュームでの入出力競合の量は、ファイルセットの性能に対して最も重要な要因となります。これは、大規模で負荷の高いファイル・ドメインで発生することがあります。ファイルセットのセットアップ方法を決めるには、まず次の項目を調べます。

- 頻繁にアクセスされるデータ
- まれにしかアクセスされないデータ
- 特定タイプのデータ (一時的なデータやデータベースのデータなど)
- 特定のアクセス・パターンを持つデータ (作成、削除、読み取り、書き込みなど)

次に、上記の情報と次に示すガイドラインを用いてファイルセットとファイル・ドメインを構成します。

- 類似したタイプのファイルを含むファイルセットを同じファイル・ドメイン内に構成して、ディスクの断片化を減らし性能を改善します。たとえば、cron の出力、ニュース、メール、Web キャッシュ・サーバの出力などの小さな一時ファイルを、大規模なデータベース・ファイルと同じドメイン内に置かないようにしてください。
- 多数のファイルの作成または削除操作を実行するアプリケーションでは、複数のファイルセットを構成して、これらのファイルセットにファイルを分散させます。これにより、個別のディレクトリ、ルー

ト・タグ・ディレクトリ、クォータ・ファイル、フラグ・ファイルの競合は少なくなります。

- アプリケーションが、異なる入出力アクセス・パターン (作成、削除、読み取り、書き込みパターンなど) で使用するファイルセットを同じファイル・ドメイン内に構成します。これにより、入出力負荷のバランスがとれることがあります。
- 複数のファイルセットがあるマルチボリューム・ファイル・ドメイン内での入出力の競合を削減するには、複数のドメインを構成して、これらのドメインにファイルセットを分散させます。これにより、各ボリュームとドメインではトランザクション・ログを使用するファイルセットの数が少なくなります。
- 小さいファイルが大量にあるファイルセットでは、`vdump` および `vrestore` コマンドに悪影響を与えることがあります。複数のファイルセットを使用すると、`vdump` コマンドは各ファイルセットで同時に実行できるようになり、`vrestore` コマンドでファイルセットを回復するために要する時間が短くなります。

表 11-4 に、その他の AdvFS 構成ガイドラインと、性能上の利点と欠点を示します。AdvFS についての詳細は『*AdvFS 管理ガイド*』を参照してください。

表 11-4: AdvFS の構成のガイドライン

利点	ガイドライン	欠点
データの損失を防げる。	LSM または RAID を用いて、RAID1 (ミラー・データ) または RAID5 でデータを格納する (11.2.1.1 項)。	LSM または RAID が必要。
データの損失を防げる。	同期書き込みを強制するか、ファイルでのアトミック書き込みデータ・ロギングを有効にする (11.2.1.2 項)。	ファイル・システムの性能が低下することがある。
データを 1 度だけ読み書きするアプリケーションの性能が向上する。	直接入出力を有効にする (11.2.1.3 項)。	同じデータにくり返しアクセスするアプリケーションの性能が低下する。
性能が改善される。	AdvFS を使用してファイル・ドメイン内にファイルを分散させる (11.2.1.4 項)。	なし。

表 11-4: AdvFS の構成のガイドライン (続き)

利点	ガイドライン	欠点
性能が改善される。	データをストライピングする (11.2.1.5 項)。	AdvFS を使用している場合はなし。 または、LSM が RAID が必要。
性能が改善される。	ファイル・ドメインの断片化を解消する (11.2.1.6 項)。	なし。
性能が改善される。	入出力転送サイズを小さくする (11.2.1.7 項)。	なし。
性能が改善される。	トランザクション・ログを、高速なディスクまたは混雑していないディスクに移動する (11.2.1.8 項)。	ディスクの増設が必要になる。

以降の項で、これらのガイドラインを詳しく説明します。

11.2.1.1 RAID1 または RAID5 を用いてデータを格納する

LSM またはハードウェア RAID を使用すると、RAID1 または RAID5 のデータ・ストレージを構成できます。

RAID1 の構成では、LSM やハードウェア RAID は、ファイル・ドメインまたはトランザクション・ログ・データのミラー (コピー) を複数のディスクに保存します。ディスクに障害が発生すると、LSM やハードウェア RAID はミラーを用いてデータを使用可能にします。

RAID5 の構成では、LSM やハードウェア RAID はパリティ情報とデータを保存します。ディスクに障害が発生すると、LSM やハードウェア RAID は、残ったディスク上のパリティ情報とデータを使用して、失われたデータを再構築します。

LSM についての詳細は『*Logical Storage Manager*』を参照してください。ハードウェア RAID についての詳細は、ストレージ・デバイスのマニュアルを参照してください。

11.2.1.2 同期書き込み要求を強制するか、永続的なアトミック書き込みデータ・ロギングを有効にする

AdvFS は 8 KB 単位でディスクにデータを書き込みます。特に指定しなければ、AdvFS の非同期書き込み要求は UBC にキャッシュされ、write システ

ム・コールは正常終了の値を返します。データは後でディスクに書き込まれます (非同期)。AdvFS では、書き込み中または書き込み直後にクラッシュが発生した場合、データのすべてまたは一部が実際にディスクに書き込まれたかどうかは保証されません。たとえば、8 KB 単位のデータを 2 つ書き込む途中でシステムがクラッシュした場合、全書き込みの一部 (16 KB 未満) だけが正常に行われた可能性があります。これにより、データが部分的に書き込まれ、データの不整合が発生することがあります。

AdvFS を構成して、指定されたファイルへの書き込み要求を強制的に同期的に実行し、`write` システム・コールが正常な値を返す前に、確実にデータがディスクに書き込まれるようにすることができます。

指定したファイルへの永続的なアトミック書き込みデータ・ロギングを有効にすると、データはディスクに書き込まれる前にトランザクション・ログに書き込まれます。`write` システム・コールの途中または直後にシステムがクラッシュした場合、ログ・ファイル内のデータを使用して、回復時に `write` システム・コールを再構築できます。

1 つのファイルに対して、同期書き込みの強制と永続的なアトミック書き込みデータ・ロギングの両方を有効にすることはできません。ただし、ファイルのアトミック書き込みデータ・ロギングを有効にして、そのファイルを `O_SYNC` オプション付きでオープンすることはできます。このように指定すると、同期書き込みが行われる他、`write` システム・コールから戻る前にクラッシュが発生した場合の部分的な書き込みも防止できます。

同期書き込み要求を強制するには、次のように入力します。

```
# chfile -l on filename
```

永続的なアトミック書き込みデータ・ロギングを有効にしたファイルは、`mmap` システム・コールを使用してメモリにマップすることはできません。また、直接入出力を有効にすることもできません (11.2.1.3 項を参照)。永続的なアトミック書き込みデータ・ロギングを有効にするには、次のように入力します。

```
# chfile -L on filename
```

永続的なアトミック書き込みデータ・ロギングが設定されたファイルは、書き込みが 8192 バイト以下で実行されたときのみ、アトミックに書き込まれます。書き込みサイズが 8192 バイトより大きい場合には、8192 バイト以下の複数のセグメントに分割され、それぞれのセグメントがアトミックに書き込まれます。

NFS マウントされた AdvFS ファイルでアトミック書き込みデータ・ロギングを有効にする場合は、次の点を確認してください。

- NFS プロパティ・リスト・デーモン `proplistd` が NFS サーバ上で実行されており、`proplist` オプション付きの `mount` コマンドで、ファイルセットがクライアントにマウントされていること。
- ファイル内でのオフセットが 8 KB のページ境界に合っていること。これは、NFS が 8 KB のページ境界に合わせて入出力を行うためです。この場合、8 KB のページ境界から始まる 8192 バイトのセグメントだけが自動的に書き込まれます。

詳細は、`chfile(8)` と『*AdvFS 管理ガイド*』を参照してください。

11.2.1.3 直接入出力を有効にする

直接入出力を有効にすると、以前にアクセスしたデータを何度も再使用しないアプリケーションでの、ディスク入出力のスループットが大幅に向上します。直接入出力を有効にする際に考慮が必要な点を、以下のリストに示します。

- データは UBC にキャッシュされず、読み取りと書き込みは同期的に行われます。非同期入出力 (AIO) 関数 (`aio_read` および `aio_write`) を使用すると、要求の完了を待たずに 1 つ以上の同期直接入出力要求を発行することで、アプリケーションに非同期のような動作をさせることもできます。
- 直接入出力は任意のバイト・サイズの入出力要求をサポートしていますが、要求されたバイト転送がディスクのセクタ境界に合っていて、下位のディスク・セクタのサイズの偶数倍であれば、性能は最高になります。

データ・ロギング用にすでにオープンされているファイルや、メモリにマッピングされているファイルでは、直接入出力を有効にすることはできません。 `F_GETCACHEDPOLICY` 引数を指定して `fcntl` システム・コールを呼び出すと、オープン・ファイルで直接入出力が有効になっているかどうか判断できます。

特定のファイルに対して直接入出力を有効にするには、`open` システム・コールを使用して、`O_DIRECTIO` ファイル・アクセス・フラグをセットします。直接入出力を指定してファイルをオープンすると、すべてのユーザがそのファイルをクローズするまでこのモードが有効になります。

詳細は、`fcntl(2)`、`open(2)`、『*AdvFS 管理ガイド*』、『*プログラミング・ガイド*』を参照してください。

11.2.1.4 AdvFS を使用してファイルを分散させる

マルチボリューム・ドメイン内のファイルが均等に分散されていない場合、性能が低下することがあります。マルチボリューム・ファイル・ドメイン内のボリュームに対して均等にスペースを分散させると、ドメイン内のボリュームの中で、使用中のスペースの割合のバランスをとることができます。ファイルは、ドメイン内の各ボリュームで、使用中のスペースの割合ができるだけ均等になるまで、ボリューム間で移動できます

ファイルのバランシングが必要かどうかを判断するには、次のように入力します。

```
# showfdmn file_domain_name
```

次のような情報が表示されます。

	Id	Date Created	LogPgs	Version	Domain Name		
3437d34d.000ca710	Sun Oct 5 10:50:05 2001	512	3	usr_domain			
Vol	512-Blks	Free	% Used	Cmode	Rblks	Wblks	Vol Name
1L	1488716	549232	63%	on	128	128	/dev/disk/dsk0g
2	262144	262000	0%	on	128	128	/dev/disk/dsk4a

	1750860	811232	54%				

% Used フィールドには、現在、ファイルまたはメタデータ (ファイルセット・データ構造体) に割り当てられているボリューム・スペースの割合が表示されます。上記の例では、usr_domain ファイル・ドメインはバランスが取れていません。ボリューム 1 には 63% の使用中のスペースがありますが、ボリューム 2 の使用中のスペースは 0% です (追加されたばかり)。

マルチボリューム・ファイル・ドメイン内の複数のボリュームに、使用中のスペースを均等に分散させるには、次のように入力します。

```
# balance file_domain_name
```

balance コマンドは、ユーザおよびアプリケーションからは見えず、データの可用性には影響せず、ファイルの分割も行いません。したがって、大きなファイルがあるファイル・ドメインでは、小さなファイルからなるファイル・ドメインほど均等にファイルをバランスさせることができないため、大きなファイルを、マルチボリューム・ファイル・ドメイン内の同じボリュームに手作業で移動することが必要になる場合があります。

ファイルを移動する必要があるかどうかを判断するには、次のように入力します。

```
# showfile -x file_name
```

次のような情報が表示されます。

```
Id Vol PgSz Pages XtntType Segs SegSz I/O Perf File
8.8002 1 16 11 simple ** ** async 18% src

    extentMap: 1
    pageOff    pageCnt    vol    volBlock    blockCnt
      0         1         1    187296         16
      1         1         1    187328         16
      2         1         1    187264         16
      3         1         1    187184         16
      4         1         1    187216         16
      5         1         1    187312         16
      6         1         1    187280         16
      7         1         1    187248         16
      8         1         1    187344         16
      9         1         1    187200         16
     10         1         1    187232         16
    extentCnt: 11
```

上記の例のファイルは、別のボリュームに移動する候補としては良い例です。エクステントが 11 個あり、Perf フィールドに示されるように、性能効率が 18 パーセントだからです。割合が高いほど、効率が良いということです。

ファイルをファイル・ドメイン内の別のボリュームに移動するには、次のように入力します。

```
# migrate [-p pageoffset] [-n pagecount] [-s volumeindex_from] \
[-d volumeindex_to] file_name
```

ファイルの移動元を指定することもできますし、ファイル・ドメイン内の最適の場所をシステムに選択させることもできます。ファイル全体と特定のページのどちらでも、別のボリュームに移動させることができます。

ファイルを移動した後に balance ユーティリティを使用すると、ファイルが別のボリュームに移動されることがあるので、注意してください。

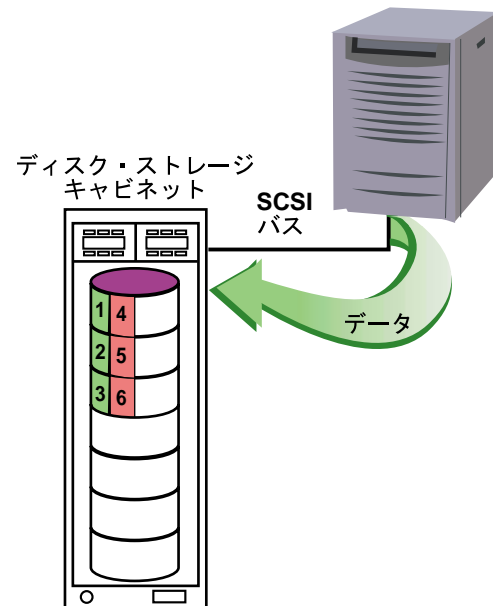
詳細は showfdmn(8)、migrate(8)、balance(8) を参照してください。

11.2.1.5 データをストライピングする

AdvFS、LSM、ハードウェア RAID を使用すると、データのストライピング（分散）が実現できます。ストライピングされたデータは、同じサイズのユニットに分割されてから 2 台以上のディスクに書き込まれ、データのストライプとして構成されたデータです。2 台以上のユニットがあり、ディスクが複数の SCSI バス上にある場合、データは同時に書き込まれます。

384 KB のデータの書き込み要求が、6 個の 64 KB のデータ・ユニットに分割されて、2 つの完全なストライプとして 3 台のディスクに書き込まれる動作を、図 11-1 に示します。

図 11-1: データのストライピング



ZK-1687U-J-AI

データをストライピングするときには、1 種類の方式のみを使用します。場合によっては、複数のストライピング方式を用いて性能を改善できる場合がありますが、それは次のような場合に限られます。

- ほとんどの入出力要求が大きい (1 MB 以上) 場合
- 異なるコントローラ上の複数の RAID セットに対してデータがストライピングされる場合
- LSM や AdvFS のストライプ・サイズが、フル・ハードウェア RAID のストライプ・サイズの倍数である場合

AdvFS を用いたデータのストライピングについての詳細は `stripe(8)` を参照してください。LSM を用いたデータのストライピングについての詳細は、『*Logical Storage Manager*』を参照してください。ハードウェア RAID を使用したデータのストライピングについては、ストレージ・デバイスのマニュアルを参照してください。

11.2.1.6 ファイル・ドメインの断片化を解消する

エクステントは、AdvFS がファイルに割り当てる連続したディスク・スペースです。エクステントは、1 つ以上の 8 KB ページからなります。ファイルに記憶域が追加されると、エクステントとしてグループ化されます。ファイル内のすべてのデータが連続したブロックに格納された場合、このファイルのファイル・エクステントは 1 個です。ただし、ファイルが大きくなると、新しいデータを格納できるだけの連続ブロックがディスク上にない場合があります。この場合、ファイルは連続していないブロック、また複数のファイル・エクステントに分散させなければなりません。

ファイルの入出力は、エクステントの数が少ないほど効率的になります。ファイルが多数の小さなエクステントで構成されている場合、AdvFS はファイルを読み書きするために、より多くの入出力処理を必要とします。ディスクが断片化されていると、エクステントが多くなり、ファイルにアクセスするときに多数のディスク・アドレスを調べなければならないため、読み書きの性能が低下します。

ファイル・ドメインの断片化情報を表示するには、次のように入力します。

```
# defragment -vn file_domain_name
```

次のような情報が表示されます。

```
defragment: Gathering data for 'staff_dmn'
Current domain data:
  Extents:                263675
  Files w/ extents:       152693
  Avg exts per file w/exts: 1.73
  Aggregate I/O perf:     70%
  Free space fragments:   85574
                           <100K  <1M   <10M  >10M
  Free space:    34%   45%   19%   2%
  Fragments:    76197  8930   440    7
```

各ファイルのエクステントは少ない方が理想的です。

defragment コマンドはデータの可用性には影響せず、ユーザおよびアプリケーションからは見えませんが、処理に時間がかかり、ディスク・スペースを必要とします。ファイル・システムの動作が少ないときに定期的なファイル・システム保守作業の一環として、または断片化が多くなって問題が生じた場合に、defragment コマンドを実行してください。

ファイル・ドメインに 8 KB 未満のファイルが含まれているか、ファイル・ドメインがメール・サーバで使用されているか、または読み取り専用である場合、断片化を解消しても性能はあまり向上しません。

また、`showfile` コマンドを使用すると、ファイルの断片化を調べることができます。詳細は 11.2.2.4 項と `defragment(8)` を参照してください。

11.2.1.7 入出力転送サイズを小さくする

AdvFS は、デバイス・ドライバにとって最も効率の良いサイズで、ディスクに対するデータ転送を行います。この値はデバイス・ドライバによって提示され、推奨転送サイズと呼ばれます。AdvFS は推奨転送サイズを使用して以下の動作を実行します。

- 連続した小さい入出力転送を、推奨転送サイズの大きな入出力 1 つに統合します。これにより、入出力要求の数が減少し、スループットが向上します。
- シーケンシャルに読み取られるファイルに対して、アプリケーションが後で読み取ることを予測して、推奨転送サイズになるまで次のページを先取りまたは先読みします。

一般に、デバイス・ドライバが提供する入出力転送サイズは、最も効率的なサイズです。ただし、場合によっては AdvFS の入出力サイズを小さくした方が良いことがあります。たとえば、AdvFS ファイルセットで LSM ボリュームを使用している場合は、推奨転送サイズでは大きすぎることがあります。この場合、読み取るファイル用にバッファが使用されるために、キャッシュが非常に希薄になることがあります。このようなおそれがある場合、読み取り転送サイズを小さくすると、問題が緩和されることがあります。

`mmap` ページ・フォールトの効率が悪いシステムや、メモリが制限されているシステムでは、読み取り転送サイズを制限して、先取りされるデータの量を制限します。ただし、このようにすると、このディスクからの読み取りすべてで入出力の統合が制限されます。

ディスクに対する入出力転送サイズを表示するには、次のように入力します。

```
# chvol -l block_special_device_name domain
```

読み取り入出力転送サイズを変更するには、次のように入力します。

```
# chvol -r blocks block_special_device_name domain
```

書き込み入出力転送サイズを変更するには、次のように入力します。

```
# chvol -w blocks block_special_device_name domain
```

詳細は、`chvol(8)` を参照してください。

各デバイス・ドライバには、入出力転送サイズの最小値と最大値があります。サポートされていない値を使用すると、デバイス・ドライバは自動的に値を制限して、サポートしている最大または最小の入出力転送サイズに変更します。サポートされている入出力転送サイズについての詳細は、デバイス・ドライバのマニュアルを参照してください。

11.2.1.8 トランザクション・ログを移動する

AdvFS のトランザクション・ログは、高速なディスクとバス、またはアクセス頻度の低いディスクとバスに配置してください。 そうしないと、性能が低下することがあります。

ボリューム情報を表示するには、次のように入力します。

```
# showfdmn file_domain_name
```

次のような情報が表示されます。

Id		Date Created		LogPgs	Domain Name		
35ab99b6.000e65d2		Tue Jul 14	13:47:34 2002	512	staff_dmn		
Vol	512-Blks	Free	% Used	Cmode	Rblks	Wblks	Vol Name
3L	262144	154512	41%	on	256	256	/dev/rz13a
4	786432	452656	42%	on	256	256	/dev/rz13b
-----		-----	-----				
1048576		607168	42%				

showfdmn コマンドの表示では、トランザクション・ログが置かれているボリュームの横に文字 L が表示されます。

トランザクション・ログが置かれているディスクが低速または負荷が高い場合、次のようにします。

- トランザクション・ログを別のディスクに移す。

switchlog コマンドを使用してトランザクション・ログを移動します。

- 大きなマルチボリューム・ファイル・ドメインを、複数の小さなファイル・ドメインに分割する。これにより、トランザクション・ログの入出力アクセスが複数のログに分散されます。

マルチボリューム・ドメインを複数の小さなドメインに分割するには、小さなドメインを作成してから、大きなドメインの一部を小さなドメインにコピーします。AdvFS の vdump および vrestore コマンドを使用すると、大きなドメインで使用されていたディスクを、複数の小さなドメインの構築に使用できます。

詳細は、showfdmn(8)、switchlog(8)、vdump(8)、vrestore(8)を参照してください。

11.2.2 AdvFS 統計情報のモニタリング

表 11-5 では、AdvFS 情報を表示するためのコマンドについて説明しています。

表 11-5: AdvFS 情報を表示するツール

ツール	説明	参照先
advfsstat	AdvFS の性能の統計情報を表示する。	11.2.2.1 項
advscan	ファイル・ドメイン内のディスクを表示する。	11.2.2.2 項
showfdmn	AdvFS ファイル・ドメインとボリュームに関する情報を表示する。	11.2.2.3 項
showfsets	ファイル・ドメインに関する AdvFS ファイルセット情報を表示する。	11.2.2.5 項
showfile	AdvFS ファイルセット内のファイルに関する情報を表示する。	11.2.2.4 項

以降の項では、これらのコマンドを詳しく説明します。

11.2.2.1 AdvFS の性能の統計情報を表示する

ファイル・ドメインについての詳細な情報 (UBC および namei キャッシュの使用状況、ファイルセットの vnode 動作、ロック、ビットファイル・メタデータ・テーブル (BMT) の統計情報、ボリュームの入出力性能など) を表示するには、advfsstat コマンドを使用します。

次のコマンドを実行すると、ボリューム入出力キューの統計情報が表示されます。

```
# advfsstat -v 3 [-i number_of_seconds] file_domain
```

次のような情報が、1 ディスク・ブロック単位 (512 バイト) で表示されます。

rd	wr	rg	arg	wg	awg	blk	ubcr	flsh	wlz	sms	rlz	con	dev
0	0	0	0	0	0	1M	0	10K	303K	51K	33K	33K	44K

-i オプションを使用すると、特定の時間間隔 (秒単位) で情報を表示できます。

上記の例では、次のような内容が表示されています。

- rd (読み取り) および wr (書き込み) 要求

読み取り要求の数と書き込み要求の数を比較します。読み取り要求は、読み取りが完了するまでブロックされますが、非同期書き込み要求では呼び出し元のスレッドがブロックされないため、マルチスレッドのスループットが向上します。

- rg および arg (統合された読み取り) と wg および awg (統合された書き込み)

統合された読み取りおよび書き込みの値は、デバイス・ドライバへの単一の入出力として統合された読み取りおよび書き込みの数を表します。統合された読み取りおよび書き込みの数が、読み取りおよび書き込み数に比べて少ない場合、AdvFS は入出力を統合していない可能性があります。

- blk (ブロック・キュー), ubcr (ubc リクエスト・キュー), flsh (フラッシュ・キュー), wlz (ウェイト・キュー), sms (smooth sync キュー), rlz (レディ・キュー), con (コンソール・キュー), dev (デバイス・キュー)。AdvFS の入出力キューについての詳細は、11.2.3 項を参照してください。

性能が低下し、flsh キュー、blk キュー、または ubcr キュー上の入出力要求の数が増加し続け、dev キュー上の要求の数が一定している場合は、アプリケーションがこのデバイスの入出力に拘束されている可能性があります。ドメインにディスクを追加するか、LSM またはハードウェア RAID でストライピングを行うと、この問題を解決できる可能性があります。

指定したドメインまたはファイルセットに関して、ファイルの作成、読み取り、書き込み、およびその他の動作の回数を表示するには、次のように入力します。

```
# advfsstat [-i number_of_seconds] -f 2 file_domain file_set
```

たとえば、次のように表示されます。

lkup	crt	geta	read	writ	fsnc	dsnc	rm	mv	rdir	mkd	rmd	link
0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	10	0	0	0	0	2	0	2	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0
24	8	51	0	9	0	0	3	0	0	4	0	0
1201	324	2985	0	601	0	0	300	0	0	0	0	0
1275	296	3225	0	655	0	0	281	0	0	0	0	0
1217	305	3014	0	596	0	0	317	0	0	0	0	0
1249	304	3166	0	643	0	0	292	0	0	0	0	0
1175	289	2985	0	601	0	0	299	0	0	0	0	0
779	148	1743	0	260	0	0	182	0	47	0	4	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0

詳細は、advfsstat(8) を参照してください。

11.2.2.2 AdvFS ファイル・ドメイン内のディスクを表示する

次のような場合には、advscan コマンドを使用します。

- AdvFS ドメインのすべてのデバイスと LSM ディスク・グループを検索する場合。
- /etc/fdmns ディレクトリ、/etc/fdmns の下のディレクトリ・ドメイン、または /etc/fdmns の下のドメイン・ディレクトリからのリンクを削除したときに、/etc/fdmns ディレクトリのすべてまたは一部を再構築する場合。
- デバイスを移動してデバイス番号が変更された場合。

デバイス上、または LSM ディスク・グループ内の AdvFS ボリュームを表示するには、次のように入力します。

```
# advscan device | LSM_disk_group
```

次のような情報が表示されます。

```
Scanning disks dsk0 dsk5
Found domains:
usr_domain
    Domain Id      2e09be37.0002eb40
    Created        Thu Jun 26 09:54:15 2002
    Domain volumes 2
    /etc/fdmns links 2
Actual partitions found:
                    dsk0c
                    dsk5c
```

失われたドメインをデバイス上に再度作成するには、次のように入力します。

```
# advscan -r device
```

次のような情報が表示されます。

```
Scanning disks dsk6
Found domains: *unknown*
    Domain Id      2f2421ba.0008c1c0
    Created        Mon Jan 20 13:38:02 2002
    Domain volumes 1
```

```

        /etc/fdmns links          0
        Actual partitions found:
                                dsk6a*
*unknown*
        Domain Id          2f535f8c.000b6860
        Created            Tue Feb 25 09:38:20 2002
        Domain volumes     1
        /etc/fdmns links   0
        Actual partitions found:
                                dsk6b*

Creating /etc/fdmns/domain_dsk6a/
        linking dsk6a
Creating /etc/fdmns/domain_dsk6b/
        linking dsk6b

```

詳細は，advscan(8) を参照してください。

11.2.2.3 AdvFS ファイル・ドメインを表示する

ファイル・ドメインに関する情報を表示するには，次のように入力します。情報には，トランザクション・ログの作成日，サイズ，位置，ドメイン内の各ボリュームに関する情報（サイズ，空きブロック数，同時に読み書きできるブロックの最大数，デバイス特殊ファイルなど）などがあります。

```
# showfdmn file_domain
```

次のような情報が表示されます。

	Id	Date Created	LogPgs	Version	Domain Name
	34f0ce64.0004f2e0	Wed Mar 17 15:19:48 2002	512	4	root_domain

Vol	512-Blks	Free	% Used	Cmode	Rblks	Wblks	Vol Name
1L	262144	94896	64%	on	256	256	/dev/disk/dsk0a

マルチボリューム・ドメインで showfdmn コマンドを使用すると，ボリューム全体のサイズ，空きブロックの総数，現在割り当てられているボリューム・スペース全体の割合も表示されます。

コマンドの出力についての詳細は，showfdmn(8) を参照してください。

11.2.2.4 AdvFS ファイル情報を表示する

AdvFS ファイルセット内のファイル（およびディレクトリ）についての詳細情報を表示するには，次のように入力します。

```
# showfile filename...
```

または

```
# showfile *
```

* を指定すると、現在の作業ディレクトリ内にある全ファイルの AdvFS 特性が表示されます。

次のような情報が表示されます。

Id	Vol	PgSz	Pages	XtntType	Segs	SegSz	I/O	Perf	File
23c1.8001	1	16	1	simple	**	**	ftx	100%	OV
58ba.8004	1	16	1	simple	**	**	ftx	100%	TT_DB
**	**	**	**	symlink	**	**	**	**	adm
239f.8001	1	16	1	simple	**	**	ftx	100%	advfs
**	**	**	**	symlink	**	**	**	**	archive
9.8001	1	16	2	simple	**	**	ftx	100%	bin (index)
**	**	**	**	symlink	**	**	**	**	bsd
**	**	**	**	symlink	**	**	**	**	dict
288.8001	1	16	1	simple	**	**	ftx	100%	doc
28a.8001	1	16	1	simple	**	**	ftx	100%	dt
**	**	**	**	symlink	**	**	**	**	man
5ad4.8001	1	16	1	simple	**	**	ftx	100%	net
**	**	**	**	symlink	**	**	**	**	news
3e1.8001	1	16	1	simple	**	**	ftx	100%	opt
**	**	**	**	symlink	**	**	**	**	preserve
**	**	**	**	advfs	**	**	**	**	quota.group
**	**	**	**	advfs	**	**	**	**	quota.user
b.8001	1	16	2	simple	**	**	ftx	100%	sbin (index)
**	**	**	**	symlink	**	**	**	**	sde
61d.8001	1	16	1	simple	**	**	ftx	100%	tcb
**	**	**	**	symlink	**	**	**	**	tmp
**	**	**	**	symlink	**	**	**	**	ucb
6df8.8001	1	16	1	simple	**	**	ftx	100%	users

このコマンドの出力についての詳細は、showfile(8) を参照してください。

11.2.2.5 ファイル・ドメイン内の AdvFS ファイルセットを表示する

ファイルセット名、ファイルの総数、使用中のブロック数、クォータの状態、クローンの状態など、ファイル・ドメイン内のファイルセットに関する情報を表示するには、次のように入力します。

```
# showfsets file_domain
```

次のような情報が表示されます。

```
usr
    Id          : 3d0f7cf8.000daec4.1.8001
    Files       : 30469,  SLim=          0,  HLim=          0
    Blocks (512): 1586588,  SLim=          0,  HLim=          0
    Quota Status: user=off group=off
    Object Safety: off
    Fragging    : on
    DMAPI       : off
```

上記の例は、dmn1 というファイル・ドメインに、ファイルセットが 1 つとクローン・ファイルセットが 1 つあることを示しています。

詳細は showfsets(8) を参照してください。

11.2.3 AdvFS キューのチューニング

各 AdvFS ボリュームでは、入出力要求は次のいずれかのキューに送られます。

- ブロック・キュー/**UBC** キュー/フラッシュ・キュー

ブロック・キュー、UBC キュー、およびフラッシュ・キューは、読み取りおよび同期書き込み要求をキャッシュするキューです。同期書き込み要求では、ディスクへの書き込みが完了した後に、アプリケーションが続行可能になります。

ブロック・キューは、主に読み取りおよびカーネルの同期書き込みに使用されます。UBC リクエスト・キューは、ページをディスクへフラッシュする UBC リクエストを処理するために使用されます。フラッシュ・キューは、主にバッファの書き込み要求に使用されます。これには `fsync()`、`sync()`、同期書き込みがあります。ブロック・キューと UBC リクエスト・キューのバッファはフラッシュ・キューよりも優先順位が少し高いので、カーネルの要求は迅速に処理され、ディスクへの書き込みを待っているバッファが多数ある場合でもブロックされません。

ブロック・キュー、UBC リクエスト・キュー、またはフラッシュ・キューのバッファにあるデータの読み取りや変更を行うプロセスは、データがディスクに書き込まれるまで待たなければなりません。最終的にデバイス・キューに移動されるまでは、いつでも変更が可能であるレイジー・キューのバッファとは、この点が異なります。

- レイジー・キュー

レイジー・キューは、非同期の書き込み要求をキャッシュする論理的なキューの並びです。非同期入出力要求は、レイジー・キューに入られるとタイム・スタンプが割り当てられます。このタイム・スタンプは、要求をまとめて大きな入出力に統合して、バッファを定期的にディスクにフラッシュするために使用します。プロセスは、レイジー・キューにあるバッファ内のデータをいつでも変更して、新たに入出力が発生しないようにすることができます。レイジー・キュー内のキューについては、図 11-2 の後で説明しています。

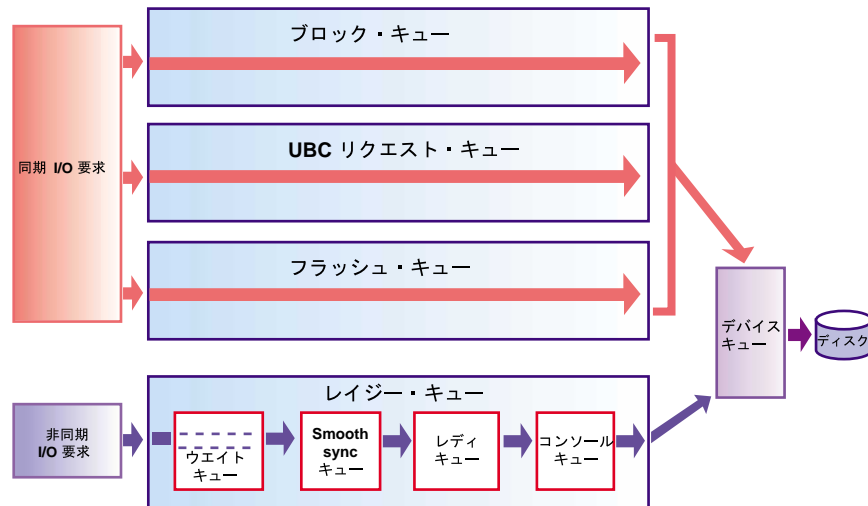
これらの 4 つのキュー (ブロック、UBC リクエスト、フラッシュ、レイジー) では、バッファがデバイス・キューに移されます。バッファがデバイス・キューに移されるときに、論理的に連続した入出力は、より大きな入出力に統合されます。これにより、実行しなければならない実際の入出力

の数は少なくなります。デバイス・キュー上のバッファは、入出力が完了するまで変更できません。

バッファをデバイス・キューに移すアルゴリズムでは、ブロック・キュー、UBC リクエスト・キュー、フラッシュ・キューの順でバッファをキューから取り出します。また、この3つはいずれもレイジー・キューより優先されます。デバイス・キューのサイズは、デバイスおよびドライバのリソースによって制限されます。デバイス・キューにロードするアルゴリズムでは、ドライバからのフィードバックによって、デバイス・キューがいつ満杯になったか分かります。この時点でデバイスは飽和状態になっているため、これ以上バッファをデバイス・キューへ移動しても、デバイスのスループットが低下するだけです。同期入出力動作にかかる時間は、最終的には、デバイス・キューのサイズと、混み具合によって決まります。

図 11-2 は、AdvFS 入出力キューでの同期および非同期入出力要求の移動の状況を示しています。

図 11-2: AdvFS の入出力キュー



ZK-1764U-AI

AdvFS のレイジー・キューについて、次に詳しく説明します。

- ウェイト・キュー — AdvFS トランザクション・ログの書き込み完了を待っている非同期入出力要求は、最初にウェイト・キューに入ります。各ファイル・ドメインには、ファイル・ドメイン内の全ファイルセットのファイルセット動作を追跡し、クラッシュが発生しても AdvFS

メタデータの整合性を確保できるようにするトランザクション・ログがあります。

AdvFS は、先書きロギングを使用します。先書きロギングを使用する場合、メタデータの変更時には、実際のメタデータが書き込まれる前にトランザクション・ログの書き込みが完了していなければなりません。これにより、AdvFS はいつでもトランザクション・ログを使用して、ファイル・システムのメタデータを矛盾のない状態にすることができます。トランザクション・ログの書き込みが完了すると、入出力要求はウェイト・キューから smooth sync キューに移動できます。

- smooth sync キュー — 非同期入出力要求は、特に指定しなければ最低 30 秒間 smooth sync キューにとどまります。指定した時間の間、smooth sync キューに要求がとどまるようにすると、入出力の集中を防止でき、キャッシュのヒット率が向上し、要求の統合が進みます。要求は、smooth sync キューの中で一定期間を過ぎるとレディ・キューに移されます。
- レディ・キュー — 非同期入出力は、レディ・キューの中でソートされます。キューが指定されたサイズになると、要求はコンソール・キューに移されます。
- コンソール・キュー — 非同期入出力要求は、コンソール・キュー内でインタリーブされ、デバイス・キューに移されます。

関連する属性

次のリストでは、AdvFS キューに関連する `vfs` サブシステム属性について説明します。

- `smoothsync_age` — 変更されたページが、`smoothsync` メカニズムでディスクにフラッシュできる状態になるまでに待たされる時間を秒数で指定します。

値: 0 ~ 60
省略時の値: 30 秒

値を 0 にすると、データはキャッシュされていた時間に関係なく、30 秒おきにレディ・キューに送られます。
この値を増やすと、システムがクラッシュした場合にデータが失われる可能性が高くなりますが、ダーティ・ページがキャッシュ

されている時間が長くなるので、正味の入出力負荷は軽減され
ず (性能が改善される)。

`smoothsync_age` 属性は、システムがマルチユーザ・モードでブートされると有効になり、システムがマルチユーザ・モードからシングルユーザ・モードに切り替わったときに無効になります。 `smoothsync_age` 属性の値を永続的に変更するには、`/etc/inittab` ファイルの以下の行を編集します。

```
smSync:23:wait:/sbin/sysconfig -r vfs smoothsync_age=30 > /dev/null 2>&1
smSyncS:5s:wait:/sbin/sysconfig -r vfs smoothsync_age=0 > /dev/null 2>&1
```

`smSync2` マウント・オプションを使用すると、代替の `smoothsync` ポリシを指定して、さらに正味の入出力負荷を軽減することができます。省略時のポリシは、ページへの変更が継続されているかどうかにかかわらず、`smoothsync_age` で指定した時間が経過したダーティ・ページをフラッシュすることです。 `smSync2` マウント・オプションを使用してファイル・システムをマウントした場合、非メモリ・マップ・モードの変更ページは、`smoothsync_age` で指定された時間の間ダーティでかつアイドルでなければ、ディスクには書き込まれません。

メモリ・マップ・モードの AdvFS ファイルは、`smoothsync_age` に従ってフラッシュされない可能性があることに注意してください。

- `AdvfsSyncMmapPages` — `mmap` ページのフラッシュを独自に管理するアプリケーションに対して、`smoothsync` を無効にするかどうかを指定します。

値: 0 または 1
省略時の値: 1 (有効)

詳細は、`mmap(2)` および `msync(2)` を参照してください。

- `AdvfsReadyQLim` — レディ・キューのサイズを指定します。

値: 0 ~ 32 K (ブロック)
省略時の値: 16 K (ブロック)

`AdvfsSyncMmapPages` 属性、`smoothsync_age` 属性、および `AdvfsReadyQLim` 属性の値は、システムをリブートすることなく変更できます。カーネル・サブシステム属性の変更については、第 3 章 を参照してください。

チューニングするかどうかの判断

データを再使用する場合は、次の値を増やすことを検討してください。

- 入出力要求が smoothsync キューにとどまる時間を長くして、キャッシュのヒット率を高めます。ただし、このようにすると、システム・クラッシュが起きたときにデータが失われる可能性が高くなります。

advfsstat -S コマンドを使用して、AdvFS smoothsync キューのキャッシュ統計情報を表示します。

- レディ・キューの大きさを大きくして、入出力要求が単一の大きな入出力に統合される可能性を高めて、キャッシュのヒット率を高めます。ただし、smoothsync が有効なときにはあまり影響がなく、受け付けた要求をレディ・キュー内でソートするオーバーヘッドが増えることがあります。

11.3 UFS のチューニング

ここでは、UFS 構成とチューニングのガイドライン、および UFS 情報を表示するためのコマンドについて説明します。

11.3.1 UFS の構成のガイドライン

表 11-6 では、UFS の構成のガイドラインと、性能上の利点と欠点について説明します。

表 11-6: UFS の構成のガイドライン

利点	ガイドライン	欠点
小さいファイルの性能が改善される。	ファイル・システムのフラグメント・サイズをブロック・サイズと等しくする (11.3.1.1 項)。	小さいファイルでは、ディスク・スペースが浪費される。
大規模なファイルの性能が改善される。	省略時のファイル・システム・フラグメント・サイズとして 1 KB を使用する (11.3.1.1 項)。	大規模なファイルでは、オーバーヘッドが増加する。
ディスク・スペースが解放され、大規模なファイルの性能が改善される。	ファイル・システムの i ノードの密度を低くする (11.3.1.2 項)。	作成できるファイルの数が少なくなる。
先読みキャッシュを備えていないディスクで性能が改善される。	回転遅延時間を設定する (11.3.1.3 項)。	なし。

表 11-6: UFS の構成のガイドライン (続き)

利点	ガイドライン	欠点
ディスクの入出力動作の回数が少なくなる。	クラスタとして結合するブロックの数を増やす (11.3.1.4 項)。	なし。
性能が改善される。	メモリ・ファイル・システム (MFS) を使用する (11.3.1.5 項)。	揮発性のキャッシュであるため、データの完全性が保証されない。
ディスク・スペースの使用量を制御できる。	ディスク・クォータを使用する (11.3.1.6 項)。	リブート時間が少し長くなることがある。
より多くのファイル・システムをマウントできる。	UFS および MFS のマウントの最大数を大きくする (11.3.1.7 項)。	追加メモリ・リソースが必要。

以降の項で、これらのガイドラインを詳しく説明します。

11.3.1.1 ファイル・システムのフラグメント・サイズとブロック・サイズを変更する

UFS ファイル・システムのブロック・サイズは 8 KB です。省略時のフラグメント・サイズは 1 KB です。newfs コマンドでは、作成時のフラグメント・サイズを、1024 KB、2048 KB、4096 KB、8192 KB のいずれかにできます。

省略時のフラグメント・サイズでは、ディスク・スペースが効率的に使用されますが、96 KB 未満のファイルではオーバーヘッドが増加します。ファイル・システム内の平均的なファイルが 96 KB より小さい場合、ファイル・システムのフラグメント・サイズを省略時のブロック・サイズ (8 KB) と同じにすると、ディスクのアクセス速度が改善され、システムのオーバーヘッドが小さくなる可能性があります。

詳細は newfs(8) を参照してください。

11.3.1.2 i ノードの密度を低くする

i ノードには、ファイル・システム内の各ファイルの情報が記述されています。ファイル・システムの最大ファイル数は、i ノードの数とファイル・システムのサイズによって決まります。システムは、ファイル・システム内のデータ・スペース 4 KB (4096 バイト) ごとに、i ノードを 1 つ作成します。

ファイル・システムに大きなファイルが多数含まれ、4 KB のスペースの単位ではファイルを作成しないことが分かっている場合は、ファイル・シ

システム上の i ノードの密度を低くすることができます。これによりファイル・データ用にディスク・スペースが解放されますが、作成できるファイルの数は少なくなります。

i ノードの密度を低くするには、`newfs -i` コマンドを使用してファイル・システムを作成するときに、各 i ノードに割り当てられるディスク・スペースの量を指定します。詳細は、`newfs(8)` を参照してください。

11.3.1.3 回転遅延時間を設定する

UFS の `rotdelay` パラメータは、転送完了の割り込みを処理して同じディスク上で新しい転送を開始するためにかかる時間をミリ秒単位で指定します。この値は、ファイル内の連続したブロックの間の回転スペースの量を判断する際に使用します。特に指定しなければ、`rotdelay` パラメータは 0 に設定され、ブロックを連続して割り当てることができます。この設定は、先読みキャッシュを備えていないディスクで `rotdelay` を設定する場合に役立ちます。キャッシュを備えているディスクでは、`rotdelay` を 0 に設定します。

`rotdelay` の値を変更するには、`tunefs` コマンドまたは `newfs` コマンドを使用します。詳細は、`newfs(8)` および `tunefs(8)` を参照してください。

11.3.1.4 クラスタとして結合するブロックの数を増やす

UFS の `maxcontig` パラメータの値は、単一のクラスタ (ファイル・ブロック・グループ) として結合できるブロックの数を指定します。`maxcontig` の省略時の値は 8 です。ファイル・システムは、`maxcontig` の値にブロック・サイズ (8 KB) を掛けた値のサイズで、入出力動作を実行します。

複数のバッファをチェーンして 1 回で転送するデバイス・ドライバでは、`maxcontig` の値を最大チェーン数として使用しなければなりません。チェーンすることにより、ディスクの入出力動作の回数が少なくなります。

`maxcontig` の値を変更するには、`tunefs` コマンドまたは `newfs` コマンドを使用します。詳細は、`newfs(8)` および `tunefs(8)` を参照してください。

11.3.1.5 MFS を使用する

メモリ・ファイル・システム (MFS) は、メモリ内のみに存在する UFS ファイル・システムです。永続的なデータやファイル構造がディスクに書き込まれることはありません。MFS を使用すると、読み書きの性能を改善できま

す。ただし、MFS は揮発性のキャッシュなので、MFS の内容は、リブートや、アンマウント動作、電源障害の後には失われます。

ディスクにデータが書き込まれないため、MFS は非常に高速なファイル・システムであり、作成後にファイル・システムにロードされる一時ファイルや読み取り専用のファイルの格納に使用できます。たとえば、ソフトウェアの構築 (障害の発生時には再起動し直されるもの) を行っている場合は、構築中に作成される一時ファイルのキャッシュに MFS を使用すると、構築時間を短縮できます。

詳細は、`mfs(8)` を参照してください。

11.3.1.6 UFS のディスク・クォータを使用する

UFS のディスク・クォータ (UFS のファイル・システム・クォータ) を設定すると、ユーザ・アカウントおよびグループに対して、UFS ファイル・システムの限界値を指定できます。ファイル・システムにクォータを適用すると、ユーザ・アカウントまたはユーザ・グループに割り当てることのできるブロックおよび i ノード (ファイル) の数の限界値を設定できます。各ファイル・システム上のユーザまたはユーザ・グループそれぞれに対して、別のクォータを設定できます。

ホーム・ディレクトリが配置されているファイル・システムにクォータを設定してください。これは、ホーム・ディレクトリが置かれているファイル・システムのサイズは、他のファイル・システムより大きくなるためです。

`/tmp` ファイル・システムにはクォータを設定しないでください。

AdvFS クォータとは異なり、UFS クォータを使用すると、リブート時間が少し長くなることがあります。AdvFS クォータについては、『*AdvFS 管理ガイド*』を参照してください。UFS クォータについては、『*システム管理ガイド*』を参照してください。

11.3.1.7 UFS および MFS のマウントの数を増やす

マウント構造体は、マウント要求時に動的に割り当てられ、その後、アンマウント要求時に割り当て解除されます。

関連する属性

`max_ufs_mounts` 属性は、システム上の UFS および MFS のマウントの最大数を指定します。

値: 0 ~ 2,147,483,647

省略時の値: 1000 (ファイル・システム・マウント)

max_ufs_mounts 属性は、システムをリブートすることなく変更できません。カーネル・サブシステム属性の変更についての詳細は、第 3 章を参照してください。

チューニングするかどうかの判断

システムのマウント数が省略時の限界値である 1000 マウントよりも多い場合は、UFS および MFS マウントを増やしてください。

UFS および MFS マウントの最大数を増やすと、マウントできるファイル・システムの数が多くなります。ただし、最大マウント数を多くすると、追加のマウント用にメモリ・リソースが必要になります。

11.3.2 UFS 統計情報のモニタリング

表 11-7 では、UFS の情報を表示できるコマンドについて説明します。

表 11-7: UFS 情報を表示するツール

ツール	説明	参照先
dumpfs	UFS 情報を表示する。	11.3.2.1 項
(dbx) print ufs_clusterstats	UFS クラスタの統計情報を表示する。	11.3.2.2 項
(dbx) print bio_stats	メタデータ・バッファ・キャッシュの統計情報を表示する。	11.3.2.3 項

11.3.2.1 UFS 情報を表示する

ファイル・システムを指定して UFS 情報 (スーパーブロック、シリンダ・グループ情報など) を表示するには、次のように入力します。

```
# dumpfs filesystem | /devices/disk/device_name
```

次のような情報が表示されます。

```
magic    11954    format  dynamic time  Tue Sep 14 15:46:52 2002
nbfree   21490   ndir     9          nifree  99541   nffree   60
ncg      65       ncyl     1027      size    409600  blocks   396062
bsize    8192     shift    13        mask    0xffffe000
fsize    1024    shift    10        mask    0xfffffc00
frag      8        shift    3         fsbtodb  1
cpg       16      bpg      798      fpg      6384    ipg      1536
```

```
minfree 10%      optim   time      maxcontig 8      maxbpg 2048
rotdelay 0ms     headswitch 0us  trackseek 0us  rps      60
```

先頭の数行には、チューニングに関連する情報が含まれています。特に重要なフィールドを、次に示します。

- `bsize` — ファイル・システムの、バイト単位のブロック・サイズ (8 KB)。
- `fsize` — ファイル・システムの、バイト単位のフラグメント・サイズ。入出力の性能を最適化するために、フラグメント・サイズを変更することができます。
- `minfree` — 通常のユーザが使用できないスペースの割合 (最小空きスペースのしきい値)。
- `maxcontig` — 回転遅延を強制しないで、連続して配置されるブロックの最大数。つまり、結合されて単一の読み取り要求になるブロックの数。
- `maxbpg` — 1 つのファイルに、1 つのシリンダ・グループから割り当てることができるブロック数の最大値。この数を超えると、強制的に他のシリンダ・グループからの割り当てが開始されます。`maxbpg` の値を大きくすると、大規模なファイルの性能が改善されます。
- `rotdelay` — 転送完了の割り込みを処理し、同じディスク上で新しい転送を開始するためにかかる時間の予測値 (ミリ秒)。ファイル内の連続するブロック間に、どれだけの回転遅延スペースを置くかを決定するために使用されます。`rotdelay` が 0 の場合、ブロックは連続して割り当てられます。

11.3.2.2 UFS クラスタをモニタリングする

システムでクラスタの読み書き転送がどのように実行されているかを表示するには、`dbx print` コマンドを使用して `ufs_clusterstats` データ構造体を調べます。以下に例を示します。

```
# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) print ufs_clusterstats
```

次のような情報が表示されます。

```
struct {
    full_cluster_transfers = 3130
    part_cluster_transfers = 9786
    non_cluster_transfers = 16833
    sum_cluster_transfers = {
```

```

        [0] 0
        [1] 24644
        [2] 1128
        [3] 463
        [4] 202
        [5] 55
        [6] 117
        [7] 36
        [8] 123
        [9] 0
        .
        .
        .
    [33]

    }
}
(dbx)

```

上記の例では、単一ブロックの転送が 24644 回、2 ブロックの転送が 1128 回、3 ブロックの転送が 463 回、となっています。

dbx print コマンドを使用すると、ufs_clusterstats_read と ufs_clusterstats_write データ構造体を指定して、クラスタの読み取りと書き込みを別々に調べることもできます。

11.3.2.3 メタデータ・バッファ・キャッシュを表示する

メタデータ・バッファ・キャッシュの統計情報(スーパーブロック、i ノード、間接ブロック、ディレクトリ・ブロック、シリンダ・グループ要約)を表示するには、dbx print コマンドを使用して、bio_stats データ構造体を調べます。以下に例を示します。

```

# /usr/ucb/dbx -k /vmunix /dev/mem
(dbx) print bio_stats

```

次のような情報が表示されます。

```

struct {
    getblk_hits = 4590388
    getblk_misses = 17569
    getblk_research = 0
    getblk_dupbuf = 0
    getnewbuf_calls = 17590
    getnewbuf_buflocked = 0
    vflushbuf_lockskips = 0
    mntflushbuf_misses = 0
    mntinvalbuf_misses = 0
    vinalbuf_misses = 0
    allocbuf_buflocked = 0
    ufssync_misses = 0
}

```

ブロック・ミスの数 (getblk_misses) をブロック・ミスとブロック・ヒット (getblk_hits) の和で割った値が、3 パーセントを超えないようにしてください。ブロック・ミスの値が大きい場合は、bufcache 属性の値を大きくしてください。bufcache 属性の値を増やす方法についての詳細は、11.1.4 項 を参照してください。

11.3.3 UFS の性能のチューニング

表 11-8 では、UFS のチューニング・ガイドラインと、性能上の利点と欠点について説明します。

表 11-8: UFS のチューニング・ガイドライン

利点	ガイドライン	欠点
性能が改善される。	非同期 UFS 入出力要求に対して UFS smoothsync および入出力スロットリングを調整する (11.3.3.1 項)。	なし。
CPU 時間が解放され、入出力動作の回数が少なくなる。	UFS クラスタの書き込みを遅らせる (11.3.3.2 項)。	入出力スロットリングを使用しない場合、バッファのフラッシュ時に、リアルタイム処理の性能が低下することがある。
ディスクの入出力動作の回数が少なくなる。	クラスタ用に結合したブロックの数を増やす (11.3.3.3 項)。	バッファ・データに必要なメモリ量が多くなる可能性がある。
読み書きの性能が改善される。	ファイル・システムの断片化を解消する (11.3.3.4 項)。	ダウン時間が必要。

以降の項で、これらのガイドラインを詳しく説明します。

11.3.3.1 UFS Smooth Sync と入出力スロットリングを調整する

UFS では、smoothsync と入出力スロットリングを用いて UFS の性能を改善し、システムの入出力負荷が重過ぎてシステムが停止状態になるのを最小限に抑えます。

smoothsync 機能を使用すると、ダーティ・ページは、指定された時間が経過してからディスクに書き出されます。これにより、頻繁に変更されるページ

がキャッシュ内で見つかることが多くなり、入出力の負荷が軽減されます。また、ダーティ・ページが大量にデバイス・キューにロックされたために発生するスパイクが、最小限に抑えられます。これは、update デーモンにフラッシュされた場合とは逆に、十分な時間が経過した後にページがデバイス・キューに入れられるためです。

さらに、入出力スロットリングはダーティ・ページをデバイス・キューにロックする場合にも対応できます。これは、任意の時点でデバイス・キューに入れることができる、遅延入出力要求の数を強制的に制限します。これにより、新しいプログラムのメモリへの読み込みやロードなどで、デバイス・キューに追加された同期要求に対する応答性が良くなります。また、ページは、デバイス・キューに置かれるまで使用できるため、ダーティ・バッファの参照でのプロセスの遅延の頻度や時間が減少します。

関連する属性

smoothsync とスロットリングに影響する `vfs` サブシステム属性は、次のとおりです。

- `smoothsync_age` — 変更ページが `smoothsync` メカニズムによってディスクにフラッシュできるようになるまでの時間を秒数で指定します。

値: 0 ~ 60
省略時の値: 30 秒

0 を設定すると `smoothsync` は無効となり、ダーティ・ページのフラッシュは 30 秒間隔で update デーモンによって制御されるようになります。

チューニングするかどうかの判断

この値を大きくすると、システムがクラッシュした場合にデータが失われる可能性も高くなりますが、ダーティ・ページがキャッシュにとどまる時間が長くなるため、正味の入出力負荷は軽減されます (性能が改善されます)。

`smoothsync_age` 属性は、システムがマルチユーザ・モードでブートされたときに有効となり、システムがマルチユーザからシングルユーザ・モードに切り替ったときに無効になります。 `smoothsync_age` 属性の値を変更するには、`/etc/inittab` ファイルの以下の行を編集します。

```
smoothsync:23:wait:/sbin/sysconfig -r vfs smoothsync_age=30 > /dev/null 2>&1
smoothsync:Ss:wait:/sbin/sysconfig -r vfs smoothsync_age=0 > /dev/null 2>&1
```

msync2 マウント・オプションを使用すると、代替の smoothsync ポリシを指定して、さらに正味の入出力負荷を軽減することができます。省略時のポリシは、ページへの変更が継続されているかどうかにかかわらず、smoothsync_age で指定した時間が経過したダーティ・ページをフラッシュすることです。msync2 マウント・オプションを使用して UFS をマウントした場合、変更ページは、smoothsync_age で指定された時間の間ダーティでかつアイドルでなければ、ディスクには書き込まれません。メモリ・マップされたページでは、msync2 の設定とは無関係に、常にこの省略時のポリシが使用されるので注意してください。

- io_throttle_shift — 入出力デバイス・キューに同時に入れることができる遅延 UFS 入出力要求の最大数を制限するための値を指定します。

省略時の値: 1 (2 秒)。ただし、io_throttle_shift 属性は、マウント・オプションの throttle を使ってマウントしたファイル・システムだけに適用されます。

入出力デバイス・キュー上の要求の数が増えるほど、要求を処理してページとデバイスが使用できるようになるまでの時間が長くなります。入出力デバイス・キューに同時に入れることができる遅延入出力要求の数は、io_throttle_shift 属性を設定することで絞る (制御する) ことができます。算出されたスロットルの値は、io_throttle_shift 属性の値と、算出されたデバイス入出力完了レートに基づいています。入出力デバイス・キューの処理に必要な時間は、スロットルの値に比例します。io_throttle_shift 属性の値とデバイス・キューの処理に必要な時間の対応は、次のようになります。

io_throttle_shift 属性の値	デバイス・キューの処理に要する時間 (秒数)
-4	0.0625
-3	0.125
-2	0.25
-1	0.5
0	1
1	2
2	4
3	8
4	16

入出力デバイスへのアクセス遅延が特に問題となる環境では、`io_throttle_shift` 属性の値を減らすことを検討してください。

- `io_maxmzthruput` — 入出力スループットを最大化するかどうか、またはダーティ・ページの可用性を最大化するかどうかを指定します。入出力スループットを最大化すると、デバイスのビジー状態が続くように積極的に動きますが、それは `io_throttle_shift` 属性で指定した値の範囲にとどまります。ダーティ・ページの可用性を最大化すると、ダーティ・ページを待つための停止時間が削減されます。

値: 0 (無効) または 1 (有効)

省略時の値: 1 (有効)。ただし、`io_throttle_maxmzthruput` 属性は、`throttle` マウント・オプションを用いてマウントしたファイル・システムにのみ適用されます。

チューニングするかどうかの判断

頻繁に使用されるダーティ・ページへのアクセス遅延が問題となる環境や、入出力が少数の入出力多用アプリケーションに限られており、全体的な性能を良くするには、入出力デバイスをビジー状態に保つことよりも、特定のページ・セットへのアクセスが重要になるような環境では、`io_maxmzthruput` 属性を無効にすることを検討してください。

`smoothsync_age`、`io_throttle_static`、`io_throttle_maxmzthruput` の各属性は、システムをリブートすることなく変更できます。

11.3.3.2 UFS クラスタの書き込みを遅らせる

特に指定しなければ、UFS ページのクラスタには非同期書き込みが行われます。他の変更されたデータとメタデータ・ページの書き込みと同じように、UFS ページのクラスタが遅れて書き込まれるように構成することもできます。

関連する属性

`delay_wbuffers` — UFS ページのクラスタの書き込みが非同期で行われるか、または遅延されるかを指定します。

値: 0 または 1

省略時の値: 0 (非同期)

UBC ダーティ・ページの割合が `delay_wbuffers_percent` 属性の値に達すると、クラスタは `delay_wbuffers` 属性の値にかかわらず非同期で書き込まれます。

以前に書き込んだページに頻繁に書き込みを行うアプリケーションの場合は、UFS ページのクラスタの書き込みを遅らせてください。これにより、入出力要求の総数が少なくなります。ただし、入出力スロットリングを使用していない場合は、同期の際にシステムの入出力負荷が重くなるため、リアルタイム処理の性能に悪影響を与えることがあります。

UFS ページのクラスタの書き込みを遅らせるには、`dbx patch` コマンドを使用して `delay_wbuffers` カーネル変数を 1 (有効) に設定します。

`dbx` の使用についての詳細は、3.2 節を参照してください。

11.3.3.3 クラスタ内のブロックの数を増やす

UFS は連続するブロックを連結してクラスタとし、入出力動作を減らします。クラスタ内のブロックの数は指定できます。

関連する属性

`cluster_maxcontig` — 単一の入出力動作として連結されるブロックの数を指定します。

省略時の値: 32 ブロック

特定のファイル・システムの回転遅延時間の値が 0 (省略時の値) の場合、UFS はブロック数が n 個までのクラスタを作成しようとします。ここで、 n は、`cluster_maxcontig` 属性の値、デバイス・ジオメトリの値のうち、小さい方になります。

特定のファイル・システムの回転遅延時間が 0 以外の値であれば、 n は、`cluster_maxcontig` 属性の値、デバイス・ジオメトリの値、`maxcontig` ファイル・システム属性の値のうち、小さい方の値になります。

チューニングするかどうかの判断

アプリケーションがサイズの大きなクラスタを使用できる場合は、クラスタとして連結するブロック数を増やします。

`newfs` コマンドを使用すると、ファイル・システムの回転遅延時間と `maxcontig` 属性の値を設定できます。 `dbx` コマンドを使用すると、 `cluster_maxcontig` 属性の値を設定できます。

11.3.3.4 ファイル・システムの断片化を解消する

連続していないファイル・エクステントでファイルが構成されている場合、そのファイルは断片化していると考えられます。断片化の度合いが大きいファイルでは、ファイルのアクセスに必要な入出力動作が多くなるため、UFS の読み書きの性能が低下します。

作業のタイミング

UFS ファイル・システムの断片化を解消すると、ファイル・システムの性能が改善されます。ただし、断片化の解消には時間がかかります。

ファイル・システム内のファイルが断片化しているかどうかは、システムのクラスタ化の効率を調べることで判断できます。判断するには、`dbx print` コマンドを使用して、`ufs_clusterstats` データ構造体を調べます。詳細は、11.3.2.2 項を参照してください。

UFS ブロックをクラスタ化すると、通常は効率がよくなります。UFS クラスタ化のカーネル構造体で、クラスタ化の効率が良くないという数値が示された場合は、ファイル・システム内のファイルが断片化している可能性があります。

推奨手順

UFS ファイル・システムの断片化を解消するには、次の手順に従います。

1. テープまたは他のパーティションに、ファイル・システムをバックアップします。
2. 同じパーティションまたは別のパーティションに、新しいファイル・システムを作成します。
3. ファイル・システムをリストアします。

データのバックアップおよびリストアと、UFS ファイル・システムの作成については、『システム管理ガイド』を参照してください。

11.4 NFS のチューニング

ネットワーク・ファイル・システム (NFS) は、ユニファイド・バッファ・キャッシュ (UBC) を、仮想メモリ・サブシステムおよびローカル・ファイル・サブシステムと共有します。NFS は、ネットワークに過度の負荷をかけることがあります。NFS の性能が良くない場合は大半が、ネットワーク・インフラストラクチャの問題です。NFS クライアントの再送メッセージの数が多くないか、ネットワークの入出力エラーがないか、負荷に耐えられないルータがないかを調べてください。

ネットワーク上でパケットの紛失があると、NFS の性能が大幅に低下します。パケットの紛失は、サーバの輻輳、送信中のパケットの破損 (電氣的接続の問題や、ノイズの多い環境、ノイズの多いイーサネット・インタフェースが原因)、転送処理の放棄が早過ぎるルータが原因で発生します。

ネットワーク・ファイル・システムのチューニング方法は、第 5 章を参照してください。

メモリ性能の管理

メモリ・リソースを最適化することで、Tru64 UNIX の性能を改善できる場合があります。通常は、ページングやスワッピングをなくすか減らすのが、性能を改善する最適な方法です。そのためには、メモリ・リソースを追加します。

この章では、次の項目について説明します。

- オペレーティング・システムがプロセスおよびファイル・システム・キャッシュに仮想メモリを割り当てる方法、およびメモリが再生 (reclaim) される方法 (12.1 節)。
- 性能の高いスワップ領域を構成する方法 (12.2 節)
- メモリ使用状況の情報を表示する方法 (12.3 節)。
- プロセス用のメモリ・リソースを増やすために変更できるカーネル・サブシステム属性 (12.4 節)。
- ページングおよびスワッピング動作の変更 (12.5 節)。
- 共用メモリ用に物理メモリを予約する方法 (12.6 節)。
- ビッグ・ページを使用して、メモリを多用するアプリケーションの性能を改善する方法 (12.7 節)

12.1 仮想メモリの動作

オペレーティング・システムは、ページと呼ばれる 8 KB 単位で物理メモリを割り当てます。仮想メモリ・サブシステムはシステム内のすべての物理ページを維持管理し、ページを次の 3 つの領域に効率的に分配します。

- 静的固定メモリ

ブート時に割り当てられ、オペレーティング・システムのデータやテキスト、システム・テーブル用に使用されます。UNIX ファイル・システム (UFS) や CD-ROM ファイル・システム (CDFS) の最近アクセスされ

たメタデータを保持するメタデータ・バッファ・キャッシュにも、静的固定メモリが使用されます。

静的固定メモリの量は、サブシステムを削除するか、メタデータ・バッファ・キャッシュのサイズを小さくしないかぎり、減らすことはできません (12.1.2.1 項を参照)。

- 動的固定メモリ

動的固定メモリはブート時に割り当てられ、システム・ハッシュ・テーブルなどの、動的に割り当てられるデータ構造体用に使用されます。ユーザ・プロセスも、`mlock` 関数などの仮想メモリ・ロック・インタフェースを使用して、アドレス空間に動的固定メモリを割り当てます。動的固定メモリの量は、要求に従って変化します。vm サブシステムの `vm_syswiredpercent` 属性は、ユーザ・プロセスが固定できるメモリ量の最大値 (省略時は、物理メモリの 80 パーセント) を指定します。

- プロセスおよびデータ・キャッシュ用の物理メモリ

固定されていない物理メモリを、ページング可能メモリといいます。このメモリは、プロセスが最近アクセスした可変メモリ (変更可能な仮想アドレス空間) と、ファイル・バック・メモリ (プログラム・テキストやシェアード・ライブラリ用に使用されるメモリ) 用に使用されます。ページング可能メモリは、AdvFS のメタデータやファイル・データの他、最近アクセスされた UFS ファイル・システム・データ (読み取り書き込み用、およびマッピングされたファイル領域でのページ・フォールト用) のキャッシュにも使用されます。

仮想メモリ・サブシステムは、プロセスおよびファイル・システムの要求に応じて、物理ページを割り当てます。限られた量の物理メモリをプロセスとファイル・システムが取り合うため、仮想メモリ・サブシステムは最も古いページの内容を定期的にスワップ領域またはスワップ・ディスクに書き込んで、そのページを再生します (ページング)。負荷が高い場合は、メモリを解放するためにプロセス全体が中断されることがあります (スワッピング)。この章で説明しているように、各種の vm サブシステム属性をチューニングすることで仮想メモリの動作を制御できます。

どのチューニング・ガイドラインに従えば処理性能を改善できるかを判断するには、メモリの動作を理解していなければなりません。以降の項では、仮想メモリ・サブシステムが下記の処理をどのように行うかを説明します。

- 物理ページの維持管理 (12.1.1 項)

- ファイル・システム・バッファ・キャッシュへのメモリの割り当て (12.1.2 項)
- プロセスへのメモリの割り当て (12.1.3 項)
- ページの再生 (12.1.4 項)

12.1.1 物理ページの維持管理

仮想メモリ・サブシステムは、システム内のすべての物理メモリ・ページを維持管理します。ページのリストを使用して、各ページの位置と古さが管理されます。最も古いページが、最初に再生 (再利用) されます。各物理ページはいつでも、次のリストのいずれかに置かれます。

- 固定リスト — 固定されていて、再生できないページ
- 空きリスト — クリーンな状態で、使用されていないページ
ページの再生は、空きリストのサイズが限界値 (チューニング可能) まで減ったときに開始されます。
- アクティブ・リスト — プロセスまたはユニファイド・バッファ・キャッシュ (UBC) で現在使用されているページ
どのアクティブ・ページを最初に再生 (再利用) するかを決定するために、ページ・スティーラ・デーモンがアクティブ・リスト上で最も古いページを見つけます。プロセスが使用している最も古いページは、非アクティブ・ページと見なされます。UBC が使用している最も古いページは、**UBC LRU** (Unified Buffer Cache least-recently used) ページと呼ばれます。

ページ・リスト上にあるページの数を知るには、`vmstat` コマンドを使用します。アクティブ・リスト (`vmstat` の出力の `act` フィールド) 上のページには、非アクティブ・ページと UBC LRU ページも含まれます。

12.1.2 ファイル・システム・バッファ・キャッシュのメモリ割り当て

オペレーティング・システムは、ファイル・システムのユーザ・データとメタデータを格納するために、キャッシュを使用します。キャッシュが後で再使用された場合は、ディスク入出力動作を行わなくてすむため、性能が向上します。これは、ディスク入出力動作よりも、メモリからのデータ取り出しの方が高速なためです。

以降の項では、次のファイル・システムのキャッシュについて説明します。

- メタデータ・バッファ・キャッシュ (12.1.2.1 項)
- ユニファイド・バッファ・キャッシュ (12.1.2.2 項)

12.1.2.1 メタデータ・バッファ・キャッシュのメモリ割り当て

カーネルはブート時に、固定メモリをメタデータ・バッファ・キャッシュに割り当てます。このキャッシュは、最近アクセスされた UFS および CDFS メタデータ (ファイル・ヘッダ情報、スーパーブロック、i ノード、間接ブロック、ディレクトリ・ブロック、およびシリンダ・グループの要約を含む) を保存することによって、オペレーティング・システムとディスクの間のレイヤとして動作します。データが後で再使用され、ディスク動作が省かれると、性能が向上します。

メタデータ・バッファ・キャッシュでは、`bcopy` ルーチンを使用して、データがメモリから出し入れされます。メタデータ・バッファ・キャッシュのメモリは、ページ再生の対象になりません。

メタデータ・バッファ・キャッシュのサイズは、`vfs` サブシステムの `bufcache` 属性で指定されます。`bufcache` 属性のチューニングについては、11.1.4 項を参照してください。

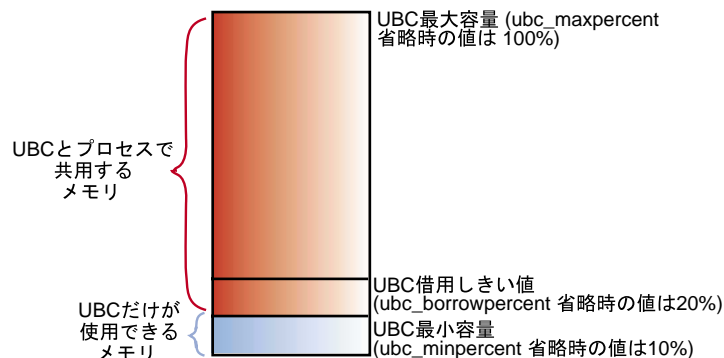
12.1.2.2 ユニファイド・バッファ・キャッシュのメモリ割り当て

固定されていない物理メモリは、プロセスとユニファイド・バッファ・キャッシュ (UBC) から利用できます。プロセスと UBC は、このメモリを取り合います。

UBC は、最近アクセスされたファイル・システム・データ (従来のファイル操作による読み込みおよび書き込み) を保存したり、マッピングされているファイル・セクションでのページ・フォールトを捕捉することによって、オペレーティング・システムとディスクの間のレイヤとして機能します。UFS はユーザおよびアプリケーションのデータを UBC にキャッシュします。AdvFS はユーザおよびアプリケーションのデータとメタデータを UBC にキャッシュします。データとメタデータが再使用され、UBC の中にある場合に、ファイル・システムの性能が向上します。

図 12-1 に、メモリ・サブシステムが UBC およびプロセスに物理メモリを割り当てる方法を示します。

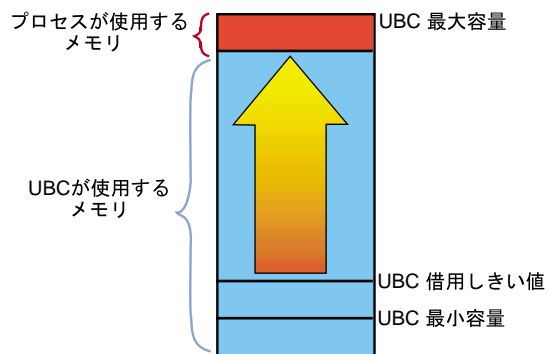
図 12-1: UBC のメモリ割り当て



ZK-1360U-AIJ

UBC およびプロセスに割り当てられているメモリの量は、ファイル・システムおよびプロセスからの要求に従って変化します。たとえば、ファイル・システム動作の負荷が高く、プロセスからの要求が少ない場合は、大半のページが UBC に割り当てられます (図 12-2 を参照)。

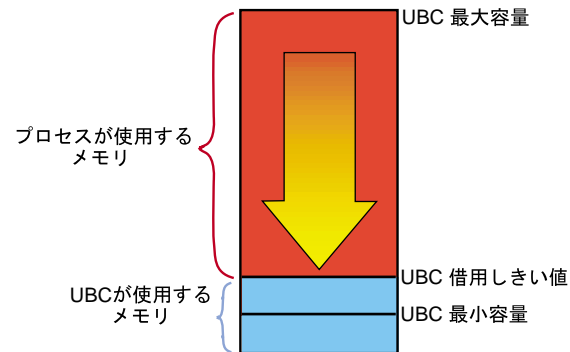
図 12-2: ファイル・システム動作の負荷が高く、ページング動作がない場合のメモリ割り当て



ZK-1426U-AIJ

逆に、プロセス動作の負荷が高い場合 (たとえば、大規模な実行可能プログラムのワーキング・セットが大きく拡大した場合など) は、`ubc_borrowpercent` 属性の値以下になるまで、メモリ・サブシステムが UBC 借用ページを再生します (図 12-3 を参照)。

図 12-3: ファイル・システム動作の負荷が低く、ページング動作の負荷が高い場合のメモリ割り当て



ZK-1427U-AIJ

UBC のサイズは、`vfs` サブシステムの UBC 関連属性の値によって指定されます。UBC 関連属性のチューニングについては、11.1.3 項を参照してください。

12.1.3 プロセスのメモリ割り当て

固定されていない物理メモリは、プロセスおよび UBC で利用できます。プロセスと UBC は、このメモリを取り合います。仮想メモリ・サブシステムは要求に応じてプロセスと UBC にメモリ・リソースを割り当てます。利用可能な空きページをこの要求で使い尽くした場合、仮想メモリ・サブシステムは最も古いページを再生します。

以降の項では、仮想メモリ・サブシステムがプロセスにメモリを割り当てる方法について説明します。

12.1.3.1 プロセスの仮想アドレス空間の割り当て

`fork` システム・コールは、新しいプロセスを作成します。ユーザがプロセスを起動すると、`fork` システム・コールは次の処理を行います。

1. カーネルがプロセスの維持管理に使用するデータ構造体セットと、リソース限界値のセットなどからなる、UNIX プロセス本体を作成します。詳細は、`fork(2)` を参照してください。
2. プロセス用に、連続したブロックの仮想アドレス空間を作成します。仮想アドレス空間は、実際の物理メモリへマッピングするためにプロセスが使用する仮想ページの配列です。仮想アドレス空間は、可変メモリ

(プロセスの実行中に変更されるデータ要素や構造体を保持するメモリ)とファイル・バック・メモリ(プログラム・テキストやシェアード・ライブラリに使用されるメモリ)用に使用されます。

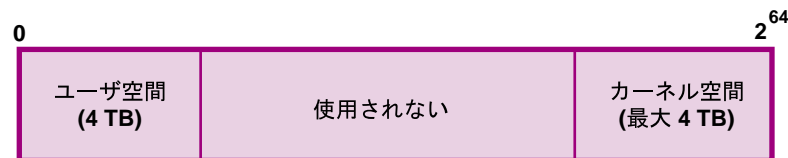
物理メモリの量は限られているため、プロセスの仮想アドレス空間全体を一度に物理メモリに置くことはできません。ただし、仮想アドレス空間の一部(プロセスのワーキング・セット)が物理メモリにマッピングされていれば、プロセスは実行できます。可変メモリとファイル・バック・メモリのページは、必要なときだけメモリに読み込まれます(ページ・イン)。メモリ要求が増えてページを再生しなければならない場合、可変メモリのページはページ・アウトされ、その内容はスワップ領域に移動されます。一方、ファイル・バック・メモリのページは、単に解放されるだけです。

3. スレッド(実行の流れ)を1つ以上作成します。省略時は、1 プロセスにつき1 スレッドです。マルチプロセッシング・システムでは、複数のプロセス・スレッドをサポートしています。

仮想メモリ・サブシステムは、各プロセスに大量の仮想アドレス空間を割り当てますが、プロセスはこの空間の一部しか使用しません。ユーザ領域には、4 TB だけが割り当てられます。ユーザ領域は一般的にプライベートで、非共用の物理ページにマッピングされます。さらに4 TB の仮想アドレス空間が、カーネル空間用に使用されます。カーネル空間は通常、共用物理ページにマッピングされます。残りの空間は、何の用途にも使用されません。

図 12-4 に、プロセスの仮想アドレス空間の用途を示します。

図 12-4: プロセス仮想アドレス空間の用途



ZK-1363U-AIJ

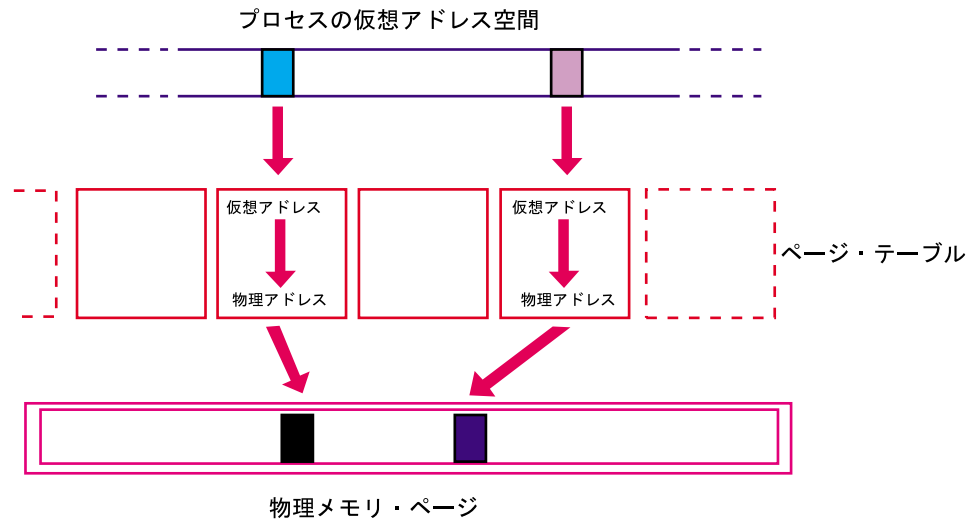
12.1.3.2 仮想アドレスから物理アドレスへの変換

仮想ページがアクセスされると、仮想メモリ・サブシステムは物理ページを見つけて、仮想アドレスを物理アドレスに変換しなければなりません。各プロセスには、仮想アドレスから物理アドレスへの現在の変換エントリからなる配列である、ページ・テーブルがあります。ページ・テーブル

のエントリは、仮想アドレスに順番に対応しており（つまり、仮想アドレス 1 がページ・テーブル・エントリ 1 に対応する）、物理ページへのポインタと保護情報を持っています。

図 12-5 に、仮想アドレスから物理アドレスへの変換を示します。

図 12-5: 仮想アドレスから物理アドレスへの変換



ZK-1358U-AIJ

プロセスの常駐セットとは、物理アドレスにマッピングされたことのあるすべての仮想アドレス（つまり、プロセスの動作中にアクセスされたすべてのページ）のセットのことです。常駐セットのページは、複数のプロセスで共用されている可能性があります。

プロセスのワーキング・セットとは、現在、物理アドレスにマッピングされている仮想アドレスのセットのことです。ワーキング・セットは常駐セットのサブセットで、ある時点でのプロセスの常駐セットのスナップショットです。

12.1.3.3 ページ・フォールト

可変メモリ（ファイル・バック・メモリではない）の仮想アドレスが要求された場合、仮想メモリ・サブシステムは物理ページを見つけ、そのページをプロセスから使用できるようにしなければなりません。この速度は、ページがメモリとディスクのどちらにあるかによって異なります（図 1-10 を参照）。

要求されたアドレスが現在使用中の場合（つまり、そのアドレスがアクティブ・ページ・リストにある場合）は、ページ・テーブルにそのアドレスのエ

ントリがあります。この場合、PAL コードが物理アドレスを変換索引バッファ (TLB) にロードし、TLB から CPU にアドレスが渡されます。この処理はメモリ動作なので高速です。

要求されたアドレスがページ・テーブル内でアクティブでない場合、PAL の検索コードはページ・フォールトを発行します。ページ・フォールトにより、仮想メモリ・サブシステムはページを見つけ、ページ・テーブルで仮想アドレスから物理アドレスへの変換ができるようにします。

ページ・フォールトには、次の 4 種類があります。

1. 要求された仮想アドレスが、初めてアクセスされるアドレスの場合は、ゼロフィル・オンデマンド・ページ・フォールトが発生します。仮想メモリ・サブシステムは、次の処理を実行します。
 - a. 使用可能な物理メモリ・ページを割り当てます。
 - b. ページをゼロで埋めます。
 - c. ページ・テーブルに、仮想アドレスから物理アドレスへの変換を設定します。
2. 要求された仮想アドレスが、すでにアクセスされたことがあり、メモリ・サブシステムの内部データ構造体にある場合は、ショート・ページ・フォールトが発生します。たとえば、物理アドレスがハッシュ・キュー・リストまたはページ・キュー・リストにある場合、仮想メモリ・サブシステムはそのアドレスを CPU に渡し、仮想アドレスから物理アドレスへの変換をページ・テーブルに設定します。この処理はメモリ動作なので高速です。
3. 要求された仮想アドレスがすでにアクセスされたことがあり、物理ページが再生されている場合、ページの内容は空きページ・リストまたはスワップ領域にあります。ページが空きページ・リストにある場合は、ハッシュ・キューおよび空きリストから取り除かれ、再利用されます。この動作は高速で、ディスク入出力を伴いません。

ページがスワップ領域にある場合は、ページ・イン・ページ・フォールトが発生します。仮想メモリ・サブシステムはページの内容をスワップ領域から物理メモリにコピーし、仮想アドレスから物理アドレスへの変換をページ・テーブルに設定します。この処理にはディスク入出力動作が必要なため、メモリ動作よりも時間がかかります。

4. プロセスが読み取り専用仮想ページを変更しようとした場合、コピー・オン・ライト・ページ・フォールトが発生します。仮想メモリ・サブシステムは使用可能な物理メモリ・ページを割り当て、読み取り専用ページの内容を新しいページにコピーし、ページ・テーブルに変換を設定します。

仮想メモリ・サブシステムでは、プロセスの実行時間を改善し、ページ・フォールトの回数を減らすために、次の技法を使用します。

- 追加ページのマッピング

仮想メモリ・サブシステムでは、どのページが次に必要となるかを予測します。どのページが最近使用されたかや、使用可能なページの数、その他の要因をチェックするアルゴリズムを使用して、サブシステムは要求されたアドレスを含むページとともに追加ページをマッピングします。

- ページ・カラリング

可能であれば、仮想メモリ・サブシステムはプロセスの常駐セット全体の 2 次キャッシュへのマッピングを試み、テキストとデータがキャッシュにある状態で処理全体を実行します。

vm サブシステムの `private_cache_percent` 属性は、可変メモリ用に予約する 2 次キャッシュの割合を指定します。この属性は、ベンチマークにのみ使用されます。省略時は、キャッシュの 50 パーセントが可変メモリ用に予約され、50 パーセントがファイル・バック・メモリ (共用) 用に予約されます。可変メモリのキャッシュを大きくするには、`private_cache_percent` 属性の値を大きくします。

12.1.4 ページの再生

メモリ・リソースの量は限られているため、仮想メモリ・サブシステムは定期的にページを再生しなければなりません。空きページ・リストには、プロセスおよび UBC から利用できるクリーン・ページが置かれています。メモリに対する要求が増えると、このリストが使い尽くされてしまうことがあります。ページ数が限界値 (チューニング可能) より少なくなると、仮想メモリ・サブシステムは、プロセスおよび UBC が使用している最も古いページを再生することによって、空きリストを補充します。

ページを再生するために、仮想メモリ・サブシステムは次の処理を実行します。

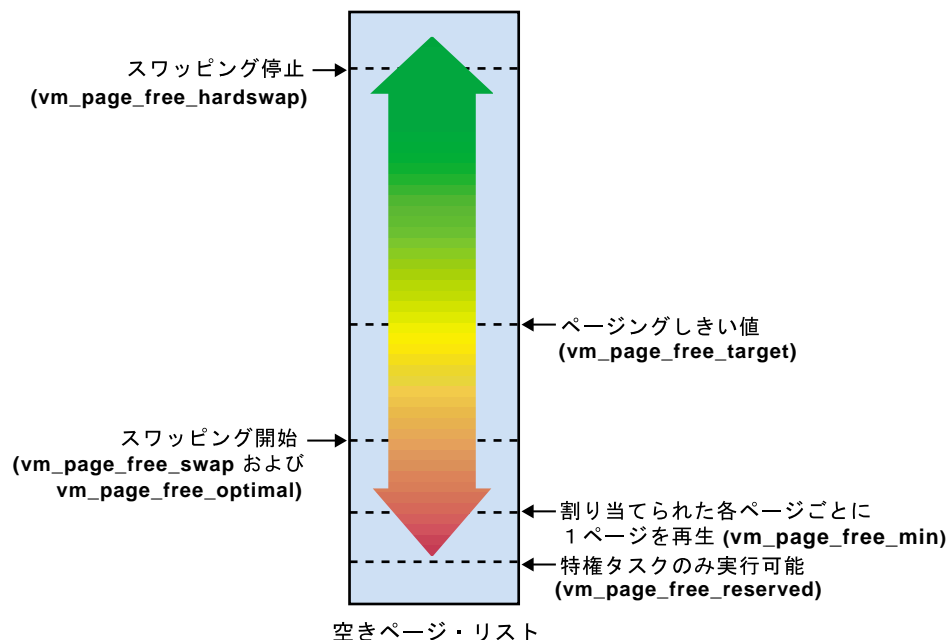
1. メモリ不足を防止するために、変更されたページをスワップ領域に事前に書き出します。詳細は、12.1.4.1 項を参照してください。
2. メモリ要求を満たすことができなかった場合は、次の手順でページングを開始します。
 - a. UBC が借用しているページを再生し、空きページ・リストに入れます。
 - b. アクティブ・リスト内で最も古い非アクティブ・ページおよび UBC LRU ページを再生し、その変更されたページの内容をスワップ領域またはスワップ・ディスクに移動して、空きリストにクリーン・ページを置きます。
 - c. 必要であれば、アクティブ・リスト内のページをもっと積極的に再生 (再利用) します。

ページングによるメモリの再生についての詳細は、12.1.4.2 項を参照してください。

3. メモリ要求を満たすことができなかった場合は、スワッピングを開始します。仮想メモリ・サブシステムは、一時的にプロセスを中断し、常駐セット全体をスワップ領域に移動して、多数のページを解放します。スワッピングについては、12.1.4.3 項を参照してください。

ページングおよびスワッピングの開始時期と停止時期は、いくつかの `vm` サブシステム属性によって決まります。図 12-6 に、ページングおよびスワッピングを制御する属性の一部を示します。

図 12-6: ページングおよびスワッピングの属性



ZK-0933U-AIJ

この属性の詳細は、次のとおりです。

- vm_page_free_target — ページングを停止するしきい値を指定します。空きページ・リスト上のページ数がこの値に達すると、ページングが停止します。vm_page_free_target 属性の省略時の値は、システムのメモリ量に依存します。表 12-1 を使用して、ご使用のシステムでの省略時の値を確認してください。

表 12-1: vm_page_free_target 属性の省略時の値

メモリ・サイズ	vm_page_free_target の値
512 MB 以下	128
513 ~ 1024 MB	256
1025 ~ 2048 MB	512
2049 ~ 4096 MB	768
4096 MB より大きい	1024

- `vm_page_free_min` — ページの割り当てに対応して、ページの再生を行わなければならないしきい値を指定します。省略時の値は、`vm_page_free_reserved` 属性の 2 倍です。
- `vm_page_free_reserved` — メモリの割り当てを特権タスクに制限する時期を決めるしきい値を指定します。空きページ・リストのページ数がこの値を下回ると、メモリを取得できるのは特権タスクだけになります。省略時の値は 10 ページです。
- `vm_page_free_swap` — アイドル・タスクのスワッピングを開始するしきい値を指定します。空きページ・リストのページ数がこの値を下回ると、アイドル・タスクのスワッピングが開始されます。省略時の値は、次の式で計算されます。

$$\text{vm_page_free_min} + ((\text{vm_page_free_target} - \text{vm_page_free_min}) / 2)$$

- `vm_page_free_optimal` — ハード・スワッピングを開始するしきい値を指定します。空きリストにあるページの数がこの値を 5 秒間下回ると、ハード・スワッピングが開始されます。最初にスワップ・アウトされるのは、スケジューリングの優先順位が最も低く、常駐セットのサイズが最も大きいプロセスです。省略時の値は、次の式で計算されます。
- $$\text{vm_page_free_min} + ((\text{vm_page_free_target} - \text{vm_page_free_min}) / 2)$$
- `vm_page_free_hardswap` — ページ・スワッピングを停止するしきい値を指定します。空きリストにあるページの数がこの値まで増えると、ページングが停止します。省略時の値は、`vm_page_free_target` 属性の値を 16 倍した値です。

ページングおよびスワッピングの属性の変更方法については、12.5 節を参照してください。

以降の項では、ページ再生手順の詳細について説明します。

12.1.4.1 変更ページの事前書き出し

仮想メモリ・サブシステムは、変更された非アクティブ・ページおよび UBC LRU ページをディスクに事前書き出し、メモリ不足を防止しようとします。事前書き出しされているページを再生する場合、仮想メモリ・サブシステムはページが有効であることを確認するだけですみ、性能が向上します。ページ・リストについては、12.1.1 項を参照してください。

仮想メモリ・サブシステムは、空きリスト上のページがもうすぐなくなると予測した場合、現在プロセスまたは UBC が使用している最も古い、変更 (ダーティ) ページを、ディスクに事前書き出しします。

vm サブシステムの `vm_page_prewrite_target` 属性は、サブシステムが事前書き出しを行いクリーンな状態にできる非アクティブ・ページの数を決めます。省略時の値は、`vm_page_free_target * 2` です。

`vm_ubcdirtypercent` 属性は、変更 UBC LRU ページのしきい値を指定します。変更 UBC LRU ページの数がこの値よりも多い場合、仮想メモリ・サブシステムは最も古い、変更 UBC LRU ページをディスクに事前書き出しします。`vm_ubcdirtypercent` 属性の省略時の値は、UBC LRU ページの総計の 10 パーセントです。

また `sync` 関数は、システム・メタデータおよびすべての未書き出しのメモリ・バッファのデータを、定期的にフラッシュ (ディスクへの書き込み) します。たとえば、UFS の場合、変更された i ノードや、遅延ブロック入出力などのデータがフラッシュされます。`shutdown` コマンドなどのコマンドも、独自の `sync` 関数を呼び出します。`sync` 関数による入出力の影響を最小限にするために、カーネルが 1 秒間に実行できるディスク書き込みの最大数を、vm サブシステムの `ubc_maxdirtywrites` 属性の値で指定します。省略時の値は、1 秒間に 5 回の入出力動作です。

12.1.4.2 ページングによるメモリの再生

メモリ要求が多く、空きページ・リスト上のページの数 vm サブシステムの `vm_page_free_target` 属性の値未満になった場合、仮想メモリ・サブシステムはページングを使用して、空きページ・リストを補充します。ページ・アウト・デーモンおよびタスク・スワップ・デーモンはページ再生処理の延長であり、ページングおよびスワッピングを制御します。

ページング処理は、次の手順で実行されます。

1. ページ再生処理が、ページ・スティーラ・デーモンを起動します。このデーモンは、UBC のサイズが借用しきい値以下になるまで、UBC が仮想メモリ・サブシステムから借用しているクリーン・ページをまず再生します。借用しきい値は `ubc_borrowpercent` 属性の値で指定され、省略時の値は 20 パーセントです。UBC ページは変更されないことが多いため、UBC の借用ページを解放すると、素早くページを再生できます。再生したページがダーティな (変更されている) 場合は、そのペー

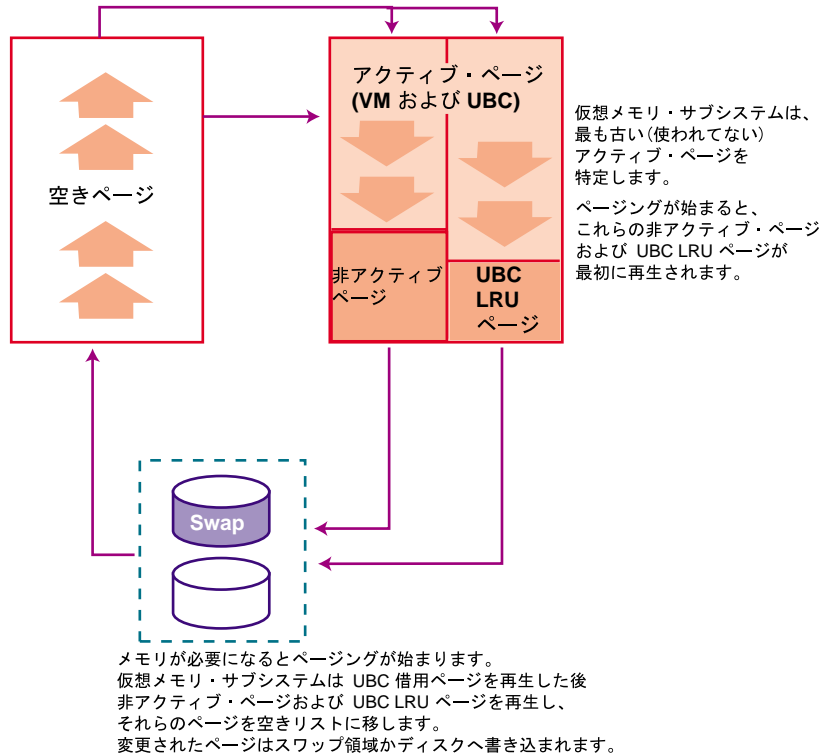
ジの内容をディスクに書き出してから、ページを空きページ・リストに移動しなければなりません。

2. クリーンな UBC 借用メモリを解放しても、空きリストの補充が十分でない場合は、ページ・アウトが発生します。ページ・スティーラ・デーモンが、アクティブ・ページ・リスト内の最も古い非アクティブ・ページおよび UBC LRU ページを再生 (再利用) し、その変更ページの内容をディスクに移動して、クリーン・ページを空きリストに置きます。
3. 空きページが減り続けると、ページングは一層活発になります。空きページ・リスト上のページの数 `vm` サブシステムの `vm_page_free_min` 属性の値 (省略時の値は 20 ページ) 未満になった場合、リストからページを取り出すたびに、ページを再生しなければなりません。

図 12-7 に、ページング動作でのページの動きを示します。

図 12-7: ページング動作

空きリスト内のクリーン・ページはアクティブ・リストへ移され、プロセスおよびUBCで使用される。



ZK-1361U-AIJ

空きリスト上のページの数 `vm` サブシステムの `vm_page_free_target` 属性で指定される限界値まで増えると、ページングが停止します。ただし、個々のページをページングしても、空きリストの補充が十分にできない場合は、スワッピングによって大量のメモリが解放されます (12.1.4.3 項を参照)。

12.1.4.3 スワッピングによるメモリの再生

メモリ要求が継続的に多い場合、仮想メモリ・サブシステムは、個々のページを再生するだけでは空きページ・リストを補充できないことがあります。クリーン・ページを大量に増やすために、仮想メモリ・サブシステムはスワッピングを使用して、プロセスを中断させます。これにより、物理メモリに対する要求が減ります。

タスク・スワップは、プロセスを中断させて、その常駐セットをスワップ領域に書き出し、クリーン・ページを空きページ・リストに移動することに

よって、プロセスをスワップ・アウトします。スワップ・アウトされたプロセスは実行できなくなるため、スワッピングはシステムの性能に大きな影響を与えます。このため、VLM システムや、大規模なプログラムを稼働させるシステムでは、スワッピングを避けてください。

スワッピングの開始時期と終了時期は、次の `vm` サブシステム属性によって制御されます。

- アイドル・タスクのスワッピングは、空きリスト上のページの数在一定時間 `vm_page_free_swap` 属性の値より少なくなったときに開始されます。タスク・スワップは、30 秒以上アイドル状態だったタスクをすべて中断させます。
- ハード・タスク・スワッピングは、空きページ・リスト上のページの数、5 秒以上の間 `vm_page_free_optimal` 属性の値より少なくなったときに開始されます。タスク・スワップは、一度に 1 つずつ、優先順位が最も低く、常駐セットのサイズが最も大きいタスクを中断させます。
- スワッピングは、空きリスト上のページの数 `vm_page_free_hardswap` 属性の値まで増えたときに停止します。
- スワップ・インは、空きリスト上のページの数在一定時間 `vm_page_free_optimal` 属性の値以上になったときに発生します。`vm_inswappedmin` 属性の値は、タスクがスワップ・アウト状態のままではいけない、最小時間 (秒) を指定します。この時間が経過すると、タスクをスワップ領域から取り出せるようになります。省略時の値は 1 秒です。タスクのワーキング・セットがスワップ領域からページ・インされ、そのタスクは実行できる状態になります。`vm_inswappedmin` 属性の値は、システムをリブートすることなく変更できます。

スワッピングの開始時期と停止時期を制御する属性を変更することによって、システムの性能が向上することがあります (12.5 節を参照)。大規模メモリ・システムや、大規模なプログラムを実行するシステムでは、可能であればページングやスワッピングを避けてください。

スワッピング・レートを高くする (ページ再生でのスワッピング時期が早まる) と、スループットが向上する可能性があります。スワップ・アウトされるプロセスが多いほど実際に実行しているプロセスが少なくなり、実行できる仕事量が多くなります。スワッピング・レートを高くすると、長時間スリープしているスレッドがメモリから追い出され、メモリが解放されます。

ただし、スワップ・アウトされたプロセスが必要になった場合の待ち時間が長いため、対話処理の応答速度が低下することがあります。

スワッピング・レートを低くする（ページ再生でのスワッピング時期が遅くなる）と対話処理の応答速度が速くなる可能性があります、スループットは低下します。スワッピング・レートの変更についての詳細は、12.5.2 項を参照してください。

メモリとディスク間のデータ移動を効率化するために、仮想メモリ・サブシステムは同期スワップ・バッファと非同期スワップ・バッファを使用します。仮想メモリ・サブシステムは、これらの 2 種類のバッファを使用して、比較的処理が遅いページ・アウト要求の完了を待たずに、ページ・イン要求をすぐに処理します。

同期スワップ・バッファは、ページ・イン・ページ・フォールトと、スワップ・アウトに使用されます。非同期スワップ・バッファは、非同期のページ・アウトと、変更ページの事前書き出しに使用されます。スワップ・バッファのチューニングについては、12.5.7 項を参照してください。

12.2 性能の高いスワップ領域の構成

スワップ領域の表示や、システムのインストール後に追加のスワップ領域の構成を行うには、`swapon` コマンドを使用します。このスワップ領域の追加を恒久化するには、`vm` サブシステムの `swapdevice` 属性を使用して、`/etc/sysconfigtab` ファイルにスワップ・デバイス指定します。例を次に示します。

```
vm:
    swapdevice=/dev/disk/dsk0b,/dev/disk/dsk0d
```

カーネル・サブシステムの属性の変更については、第 3 章を参照してください。

スワップ領域の割り当てモードおよびスワップ領域の要件については、4.4.1.8 項または 12.1.3 項を参照してください。

次のリストで、性能の高いスワップ領域の構成方法について説明します。

- すべてのスワップ・デバイスが、システムの稼働中に追加されたのではなく、システムのブート時に構成されていることを確認します。
- ページ・フォールトによる遅延を少なくするには、速いディスクをスワップ領域に使用します。

- ビジー状態でないディスクをスワップ領域に使用します。
- スワップ領域を複数のディスクに分散させます。1つのディスクに複数のスワップ・パーティションを置かないでください。分散させるとページングおよびスワッピングがより効率的になり、1つのアダプタ、ディスク、またはバスがボトルネックになるのを防止できます。ページ再生処理では、データが複数のディスクに書き込まれた場合に性能が向上するディスク・ストライピングの一種 (スワップ領域インタリーピング) を使用します。
- 1つのバスがボトルネックになることがないように、スワップ・ディスクを複数の入出力バスに分散させます。
- 複数のスワップ・デバイスを用いるときには、デバイスをストライピングして1つの論理スワップ・デバイスとして構成するのではなく、複数の独立したデバイス (または LSM ボリューム) として構成します。
- ページングが頻繁に発生していて、システムのメモリ量を増やすことができない場合は、スワップ・デバイスに RAID5 を使用することを検討してください。RAID5 についての詳細は、第 9 章を参照してください。

スワップ・デバイスの追加についての詳細は、『システム管理ガイド』を参照してください。性能および可用性を高めるための、ディスクの構成およびチューニングについての詳細は、第 9 章を参照してください。

12.3 メモリ統計情報のモニタリング

表 12-2 では、メモリ使用状況の表示に使用するツールについて説明します。

表 12-2: 仮想メモリおよび UBC を表示するためのツール

ツール	説明	参照先
vmstat	プロセスのスレッド、仮想メモリの使用状況 (ページ・リスト、ページ・フォールト、ページ・イン、およびページ・アウト)、割り込み、および CPU の使用状況 (ユーザ時間、システム時間、およびアイドル時間の割合) の情報を表示する。	12.3.1 項
ps	実行中のプロセスについて、現在の統計情報を表示する。情報には CPU の使用状況、プロセスおよびプロセス・セット、スケジュールの優先順位などがある。 ps コマンドは、プロセスの仮想メモリの統計情報も表示する。この情報にはページ・フォールト、ページ再生、ページ・インの回数、実メモリ (常駐セット) の使用率、常駐セットのサイズ、仮想アドレスのサイズなどがある。	12.3.2 項)
swapon	スワップ領域の利用状況に関する情報と、スワップ・デバイスごとに、割り当てられているスワップ領域の総量、使用中のスワップ領域、空きスワップ領域を表示する。このコマンドを使用して追加のスワップ領域を割り当てることもできる。	12.3.3 項
(dbx) print ufs_geta-page_stats	UBC 統計情報を報告し、ufs_getapage_stats データ構造体を調べて、UBC ページの使用状況を検査する。	12.3.4 項
sys_check	システム構成を分析し、統計情報を表示する。必要に応じて警告やチューニングのガイドラインを示す。	2.3.3 項

表 12-2: 仮想メモリおよび UBC を表示するためのツール (続き)

ツール	説明	参照先
<code>uerf -r 300</code>	システム・メモリの総計を表示する。	詳細は、 <code>uerf(8)</code> を参照。
<code>ipcs</code>	現在アクティブなメッセージ・キュー、共用メモリ・セグメント、セマフォ、リモート・キュー、ローカル・キュー・ヘッダについて、プロセス間通信 (IPC) の統計情報を表示する。 <code>ipcs -a</code> コマンドで表示される情報のうち、 <code>QNUM</code> 、 <code>CBYTES</code> 、 <code>QBYTES</code> 、 <code>SEGSZ</code> 、 <code>NSEMS</code> の各フィールドに表示される情報は特に役立つ。	詳細は、 <code>ipcs(1)</code> を参照。

以降の項では、`vmstat`、`ps`、`swapon`、および `dbx` ツールについて、詳細を説明します。

12.3.1 `vmstat` コマンドを使ってメモリを表示する

仮想メモリ、プロセス、CPU の統計情報を表示するには、次のように入力します。

```
# /usr/ucb/vmstat
```

次のような情報が表示されます。

```

Virtual Memory Statistics: (pagesize = 8192)
procs      memory      pages      intr      cpu
r  w  u  act  free wire  fault cow zero react pin pout  in  sy  cs  us sy  id
2 66 25 6417 3497 1570 155K 38K 50K   0 46K   0   4 290 165   0  2 98
4 65 24 6421 3493 1570  120   9  81   0   8   0 585 865 335 37 16 48
2 66 25 6421 3493 1570   69   0  69   0   0   0 570 968 368   8 22 69
4 65 24 6421 3493 1570   69   0  69   0   0   0 554 768 370   2 14 84
4 65 24 6421 3493 1570   69   0  69   0   0   0 865 1K 404   4 20 76

```

1
2
3
4
5

vmstat の出力の最初の行には、リブート以降の全時間についてまとめたデータが表示され、それ以降の行には最新の時間間隔に対するデータが表示されます。

vmstat コマンドの情報には、CPU や仮想メモリの問題の診断に使用できる情報が含まれています。次のフィールドを調べてください。

① プロセス情報 (procs):

- r — 実行中または実行可能状態のスレッドの数。
- w — 割り込み可能な待ち状態 (イベントまたはリソースを待っていて、割り込みや中断が可能な状態) のスレッドの数。たとえば、ユーザからのシグナルを受けたり、メモリからスワップ・アウトさせることができます。
- u — 割り込み不可の待ち状態 (イベントまたはリソースを待っていて、割り込みや中断ができない状態) のスレッドの数。たとえば、この状態のスレッドはユーザからのシグナルを受けることができません。シグナルを受けるには、待ち状態から抜けなければなりません。割り込み不可の待ち状態のプロセスは、kill コマンドで停止できません。

② 仮想メモリ情報 (memory):

- act — アクティブ・リスト上のページの数 (非アクティブ・ページと UBC LRU ページを含む)。
- free — 空きリスト上のページの数。
- wire — 固定リスト上のページの数。固定リスト上のページは、再生 (再利用) できません。

ページ・リストについての詳細は、12.1.1 項を参照してください。

③ ページング情報 (pages):

- `fault` — アドレス変換フォールトの数。
- `cow` — コピー・オン・ライト・ページ・フォールトの数。親プロセスと子プロセスでメモリ・ページを共用していて、いずれかのプロセスがこのページを変更しようとした場合に、このページ・フォールトが発生します。コピー・オン・ライト・ページ・フォールトが発生すると、仮想メモリ・サブシステムは、新しいアドレスを変換バッファにロードしてから、要求されたページの内容をこのアドレスにコピーし、その内容をプロセスが変更できるようにします。
- `zero` — ゼロフィル・オンデマンド・ページ・フォールトの数。要求されたページが内部データ構造体ではなく、一度も参照されたことがない場合に、このページ・フォールトが発生します。ゼロフィル・オンデマンド・ページ・フォールトが発生した場合、仮想メモリ・サブシステムは利用可能な物理メモリ・ページを割り当て、そのページをゼロで埋めてから、そのアドレスをページ・テーブルに設定します。
- `react` — 非アクティブ・ページ・リストにある間にページ・フォールトが発生した (参照された) ページの数。
- `pin` — ページ・スティーラ・デーモンが要求したページの数。
- `pout` — ディスクにページ・アウトされたページの数。

④ 割り込みの情報 (intr):

- `in` — クロック・デバイス以外で 1 秒間に発生した割り込みの数。
- `sy` — 1 秒間に呼び出されたシステム・コールの数。
- `cs` — 1 秒間に発生したタスクおよびスレッドのコンテキスト切り替えの数。

⑤ CPU 使用状況 (cpu):

- `us` — 通常プロセスおよび優先プロセスのユーザ時間の割合。ユーザ時間には、CPU がライブラリ・ルーチンを実行した時間も含まれます。
- `sy` — システム時間の割合。システム時間には、CPU がシステム・コールを処理した時間も含まれます。
- `id` — アイドル時間の割合。

vmstat コマンドによる CPU 使用状況モニタリングについては、
12.3.1 項を参照してください。

vmstat コマンドを用いてメモリ性能の問題を診断するには、次の手順
を実行します。

- 空きページ・リストのサイズ (free) をチェックします。空きページの
数と、アクティブ・ページ (act) および固定ページ (wire) の値を比較し
ます。空きページ、アクティブ・ページ、および固定ページの合計が、
システムの物理メモリの量に近くなければなりません。free の値は小
さくなくてはなりません、値が定常的に小さく (128 ページ未満)、過
度なページングやスワッピングが発生している場合は、作業負荷に対し
て物理メモリの量が十分でない可能性があります。
- pout フィールドを調べます。ページ・アウトの回数が定常的に多い場
合は、メモリが不足している可能性があります。
- 次のようなコマンド出力は、構成に対して UBC のサイズが小さすぎ
ることを示しています。
 - vmstat コマンドまたは monitor コマンドの出力で、ファイル・シ
ステムのページ・インが頻発しているがページ・アウト動作が少
ないまたは全くない状態が示されるか、空きページ数が非常に少
ないことが示される。
 - iostat コマンドの出力で、スワップ・ディスクの入出力動作が
少ないまたは全くない状態が示されるか、ファイル・システムの
入出力動作が過度であることが示される。詳細は、9.2 節を参
照してください。

ページングが過度になると、2 次キャッシュのミス率が高くなります。この
状況は、次の出力で示されることがあります。

- ps コマンドの出力で、タスク・スワッピング動作が頻繁であることが示
される。詳細は、12.3.2 項を参照してください。
- swapon コマンドの出力で、スワップ領域の使用が過度であることが示
される。詳細は、12.3.3 項を参照してください。

物理メモリの使用状況に関する統計情報を表示するには、次のように入
力します。

```
# vmstat -P
```

次のような情報が表示されます。

```
Total Physical Memory = 512.00 M
                        = 65536 pages
Physical Memory Clusters:

start_pfn    end_pfn      type  size_pages / size_bytes
      0         256      pal      256 /    2.00M
    256     65527      os    65271 /   509.93M
   65527     65536      pal      9 /    72.00k
```

Physical Memory Use:

```
start_pfn    end_pfn      type  size_pages / size_bytes
    256         280  unixtable      24 /    192.00k
    280         287  scavenge       7 /     56.00k
    287         918   text      631 /    4.93M
    918        1046   data      128 /    1.00M
   1046        1209    bss      163 /    1.27M
   1210        1384  kdebug      174 /    1.36M
   1384        1390  cfgmgt      6 /    48.00k
   1390        1392   locks       2 /    16.00k
   1392        1949  unixtable     557 /    4.35M
   1949        1962   pmap       13 /   104.00k
   1962        2972  vmtables    1010 /    7.89M
   2972        65527 managed    62555 /   488.71M
=====
Total Physical Memory Use:    65270 /   509.92M
```

Managed Pages Break Down:

```
free pages = 1207
active pages = 25817
inactive pages = 20103
wired pages = 15434
ubc pages = 15992
=====
Total = 78553
```

WIRED Pages Break Down:

```
vm wired pages = 1448
ubc wired pages = 4550
meta data pages = 1958
malloc pages = 5469
contig pages = 159
user ptepages = 1774
kernel ptepages = 67
free ptepages = 9
=====
Total = 15434
```

このコマンドとオプションについての詳細は、`vmstat(1)` を参照してください。メモリ・リソースの追加については、12.4 節を参照してください。

12.3.2 ps コマンドを使ってメモリを表示する

システム・プロセスの現在の情報とメモリの使用状況を表示するには、次のように入力します。

```
# /usr/ucb/ps aux
```

次のような情報が表示されます。

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	S	STARTED	TIME	COMMAND
chen	2225	5.0	0.3	1.35M	256K	p9	U	13:24:58	0:00.36	cp /vmunix /tmp
root	2236	3.0	0.5	1.59M	456K	p9	R	+ 13:33:21	0:00.08	ps aux
sorn	2226	1.0	0.6	2.75M	552K	p9	S	+ 13:25:01	0:00.05	vi met.ps
root	347	1.0	4.0	9.58M	3.72	??	S	Nov 07	01:26:44	/usr/bin/X11/X -a
root	1905	1.0	1.1	6.10M	1.01	??	R	16:55:16	0:24.79	/usr/bin/X11/dxpa
mat	2228	0.0	0.5	1.82M	504K	p5	S	+ 13:25:03	0:00.02	more
mat	2202	0.0	0.5	2.03M	456K	p5	S	13:14:14	0:00.23	-csh (csh)
root	0	0.0	12.7	356M	11.9	??	R	< Nov 07	3-17:26:13	[kernel idle]

1

2

3

4

5

6

ps コマンドは、ps コマンド自身を含め、CPU 使用量の降順に、システム・プロセスのスナップショットを表示します。ps コマンドの実行時には、システム・プロセスの状態が変わっていることがあります。

ps コマンドの出力には、CPU や仮想メモリの問題の診断に使用できる、次の情報が含まれます。

- 1 CPU 時間の使用量の割合 (%CPU)。
- 2 実メモリの使用量の割合 (%MEM)。
- 3 プロセスの仮想アドレスのサイズ (VSZ) — このプロセスに割り当てられている可変メモリの総量 (バイト)。
- 4 プロセスの実メモリ (常駐セット) サイズ (RSS) — 仮想ページにマッピングされている物理メモリの総量 (アプリケーションが物理的に使用したメモリの総量で、単位はバイト)。共用メモリは、常駐セットのサイズに含まれます。このため、このサイズの総計は、システムで利用可能な物理メモリの総量を超えることがあります。
- 5 プロセスの状態 (S) — プロセスが次のどの状態にあるかを示します。
 - 実行可能 (R)。
 - スリープ中 (S) — プロセスはイベントまたはリソースを待っています (20 秒未満)。ただし、割り込みや中断は可能です。たとえば、プロセスはユーザ・シグナルを受けることや、スワップ・アウトが可能です。

- 割り込み不可のスリープ中 (U) — プロセスはイベントまたはリソースを待っています。このとき、割り込みや中断はできません。
kill コマンドを使用してこの状態のプロセスを停止させることはできません。この状態のプロセスがシグナルを受け取るには、待ち状態から抜けなくてはなりません。
- アイドル (I) — プロセスは、20 秒以上スリープ状態になっています。
- 終了 (T) — プロセスは終了しています。
- 停止 (H) — プロセスは停止しています。
- スワップ・アウト (W) — プロセスはメモリからスワップ・アウトされています。
- メモリにロック中 (L) — プロセスは、メモリにロックされ、スワップ・アウトできない状態です。
- メモリ要求がソフト限界値を超えている (>)。
- 制御端末を持ったプロセス・グループ・リーダー (+)。
- 優先順位が下げられている (N)。
- 優先順位が上げられている (<)。

⑥ 現在までに使用した CPU 時間 (TIME)。フォーマットは、*hh:mm:ss.ms* です。

⑦ 実行しているコマンド (COMMAND)。

ps コマンドの出力から、どのプロセスがシステムの CPU 時間およびメモリ・リソースを最も消費し、プロセスがスワップ・アウトされているかどうかを調べることができます。実行中またはページング中のプロセスに的を絞ってください。考慮すべき事項を、次に説明します。

- プロセスが大量のメモリを使用している場合 (RSS フィールドおよび VSZ フィールドを参照)、過度なメモリ要求が発生することがあります。アプリケーションのメモリ使用量を少なくする方法については、7.1 節を参照してください。
- プロセスが重複して実行されている場合は、kill コマンドを使用して重複プロセスを終了させます。詳細は kill(1) を参照してください。

- プロセスが大量の CPU 時間を使用している場合、無限ループに陥っている可能性があります。kill コマンドを使用してプロセスを終了させなければなりません。ソース・コードを変更して問題を修正しなければならないことがあります。

また、クラス・スケジューラを使用して特定のタスクやアプリケーションに CPU 時間の割合を指定して割り当てることや (13.2.2 項を参照)、nice コマンドまたは renice コマンドを使用してプロセスの優先順位を下げるすることができます。これらのコマンドは、プロセスのメモリ使用状況には効果がありません。詳細は、nice(1) または renice(8) を参照してください。

- スワップ・アウトされたプロセスをチェックします。s (状態) フィールドを調べてください。このフィールドが w のプロセスはスワップ・アウトされています。プロセスのスワップ・アウトが繰り返されている場合は、メモリ・リソースが不足している可能性があります。メモリ・リソースの追加については、12.4 節を参照してください。

このコマンドとオプションの詳細は、ps(1) を参照してください。

12.3.3 swapon コマンドを使ってスワップ領域の使用状況を表示する

スワップ・デバイスの構成 (割り当てられているスワップ領域の総量、使用中のスワップ領域の量、空きスワップ領域の量) を表示するには、次のように入力します。

```
# /usr/sbin/swapon -s
```

各スワップ・パーティションの情報が次のように表示されます。

```
Swap partition /dev/disk/dsk1b (default swap):
  Allocated space:      16384 pages (128MB)
  In-use space:         10452 pages ( 63%)
  Free space:           5932 pages ( 36%)

Swap partition /dev/disk/dsk4c:
  Allocated space:      128178 pages (1001MB)
  In-use space:         10242 pages (  7%)
  Free space:          117936 pages ( 92%)

Total swap allocation:

  Allocated space:      144562 pages (1.10GB)
  Reserved space:       34253 pages ( 23%)
  In-use space:         20694 pages ( 14%)
  Available space:     110309 pages ( 76%)
```

スワップ領域は、オペレーティング・システムを最初にインストールするときに構成できますが、後日スワップ領域を追加することもできます。次のよ

うなアプリケーション・メッセージは、システムに構成されているスワップ領域が十分でないか、プロセスの限界値に達したことを示します。

```
“unable to obtain requested swap space”  
“swap space below 10 percent free”
```

スワップ領域の要件については、4.4.1.8 項または 12.1.3 項を参照してください。スワップ領域の追加と、性能を高めるためにスワップ領域を分散させる方法については、12.2 節を参照してください。

このコマンドとオプションの詳細は、`swapon(2)` を参照してください。

12.3.4 dbx デバッガを使って UBC を表示する

先読みを無効にしていない場合は、`dbx print` コマンドを使用して `ufs_getapage_stats` データ構造体を調べることで、UBC を表示できます。例を次に示します。

```
# /usr/ucb/dbx -k /vmunix /dev/mem (dbx) print ufs_getapage_stats
```

次のような情報が表示されます。

```
struct {  
    read_looks = 2059022  
    read_hits = 2022488  
    read_miss = 36506  
    alloc_error = 0  
    alloc_in_cache = 0  
}  
(dbx)
```

ヒット率を計算するには、`read_hits` フィールドの値を `read_looks` フィールドの値で割ります。95 パーセントを超えていれば、ヒット率が良いと言えます。上記の例の場合、ヒット率は約 98 パーセントです。

このコマンドとオプションの詳細は、`dbx(1)` を参照してください。

12.4 プロセス用のメモリを増やすためのチューニング

システムでページングやスワッピングが発生している場合、各種のカーネル・サブシステム属性をチューニングすることで、プロセスから利用できるメモリを増やすことができます。

表 12-3 に、プロセス用のメモリ・リソースを増やすためのガイドラインを示します。また、性能上の利点と欠点をリストします。プロセス用のメモリを増やすためのガイドラインによっては、UBC の動作や、ファイル・システ

ムのキャッシュに影響がでることがあります。ページングやスワッピングを止める最もよい方法は、システムの物理メモリを追加することです。

表 12-3: メモリ・リソースのチューニング・ガイドライン

性能上の利点	ガイドライン	欠点
CPU の負荷が低くなり、メモリの要求が少なくなる。	同時に実行するプロセスの数を少なくする (12.4.1 項)。	システムが処理する仕事量が少なくなる。
メモリが解放される。	カーネルの静的サイズを小さくする (12.4.2 項)。	一部の機能が利用できなくなることがある。
負荷が重い場合に、ネットワークのスループットが向上する。	カーネルの <code>malloc</code> 割り当て用に予約するメモリの割合を大きくする (12.4.3 項)。	メモリを消費する。
メモリが少ない場合にシステムの応答時間が改善される。	キャッシュの大きさを小さくする (11.1 節)。	ファイル・システムの性能が低下するおそれがある。
メモリが解放される。	プロセスからのメモリ要求を少なくする (7.1.6 項)。	プログラムの実行効率が低下することがある。

以降の項では、これらのチューニング・ガイドラインを詳しく説明します。

12.4.1 同時に実行するプロセスの数を少なくする

同時に実行するアプリケーションを少なくすることで、性能を改善し、メモリの要求を減らすことができます。アプリケーションをオフピークの時間帯に実行するには、`at` コマンドまたは `batch` コマンドを使用します。

詳細は、`at(1)` を参照してください。

12.4.2 カーネルの静的サイズを小さくする

不要なサブシステムを構成解除することで、カーネルの静的サイズを小さくできます。構成されているサブシステムを表示し、サブシステムを削除するには、`sysconfig` コマンドを使用します。使用中の環境に必要なサブシステムや機能は削除しないでください。

カーネル・サブシステム属性の変更については、第 3 章を参照してください。

12.4.3 カーネルの `malloc` 割り当て用に予約するメモリを増やす

大規模なインターネット・アプリケーションを動かしている場合は、カーネルの `malloc` サブシステム用に予約するメモリの量を増やさなければならないことがあります。これにより、システムのネットワーク負荷が高いときにドロップするパケットの数が減るため、ネットワークのスループットが向上します。ただし、この値を大きくすると消費されるメモリの量も増えます。

関連する属性

次に、カーネルの割り当て用に予約されるメモリに関連する `generic` サブシステム属性を説明します。

- `kmemreserve_percent` — ページ・サイズ (8 KB) 以下のカーネル・メモリ割り当て用に予約される物理メモリの割合を指定します。

値: 1 ~ 75

省略時の値: 0, ただし実際には、使用可能なメモリの 0.4 パーセントと 256 KB のうち小さい方になります。

`kmemreserve_percent` 属性は、リブートすることなく変更できます。

チューニングするかどうかの判断

ドロップしたパケットがあることが `netstat -d -i` コマンドの出力で示された場合や、ドロップしたパケットが `vmstat -M` コマンドの `fail_nowait` 見出しの下に出力された場合は、`kmemreserve_percent` 属性の値を大きくしてください。このような状況は、ネットワークの負荷が高い場合に発生します。

カーネル・サブシステム属性の変更については、第 3 章を参照してください。

12.5 ページング動作およびスワッピング動作の変更

以降の項で示すように、ページング動作とスワッピング動作を変更することで、性能を改善できることがあります。

- ページングしきい値を大きくする (12.5.1 項)。
- スワッピング・レートを管理する (12.5.2 項)。
- スワッピングを積極的に行うようにする (12.5.3 項)。
- プロセスの常駐セットのサイズを制限する (12.5.4 項)。

- ダーティ・ページの事前書き出しの頻度を管理する (12.5.5 項)。
- ページ・イン・クラスタおよびページ・アウト・クラスタのサイズを管理する (12.5.6 項)。
- 入出力要求を管理する (12.5.7 項)。
- メモリ・ロックを使用する (7.1.7 項)。

12.5.1 ページングしきい値を大きくする

ページングとは、プログラムの一部 (ページ) をメモリ内へ、またはメモリから外へ転送することです。 ページングは頻繁に発生しないようにしてください。 ページングが開始されない、空きリストの最小ページ数を指定できます。 ページングについての詳細は、 12.1.4 項 を参照してください。

関連する属性

vm サブシステムの `vm_page_free_target` 属性は、ページングを開始する前の空きリストの最小ページ数を指定します。 `vm_page_free_target` 属性の省略時の値は、システムのメモリ量によって決まります。

次の表を使用して、システムの省略時の値を調べてください。

メモリ・サイズ	<code>vm_page_free_target</code> の値
512 MB 以下	128
513 ~ 1024 MB	256
1025 ~ 2048 MB	512
2049 ~ 4096 MB	768
4096 MB より大きい	1024

`vm_page_free_target` 属性は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

`vm_page_free_target` の値は小さくしないでください。

システムでページングが発生していない場合は、`vm_page_free_target` 属性の値を大きくしないでください。 十分なメモリ・リソースがあり、深刻なメモリ不足のために性能上の問題が発生している場合は、

vm_page_free_target 属性の値を大きくしてください。ただし、この値を大きくすると、メモリの少ないシステムではページング動作が多くなり、値が大きすぎるとメモリが無駄になります。ページングおよびスワッピングの属性についての詳細は、12.1.4 項を参照してください。

vm_page_free_target 属性の省略時の値を大きくする場合は、vm_page_free_min 属性の値も大きくすると良いでしょう。

カーネル・サブシステム属性の変更については、第3章を参照してください。

12.5.2 スワッピング・レートを管理する

空きページ・リストがスワッピングしきい値を下回ると、スワッピングが開始されます。スワッピングが頻繁に発生しないようにしてください。スワッピングを開始および終了するタイミングは、指定することができます。スワッピングについての詳細は、12.1.4 項を参照してください。

関連する属性

変更ページの事前書き出しに関連する vm サブシステム属性を以下のリストに示します。

- vm_page_free_optimal — ハード・スワッピングが開始されるしきい値を指定します。空きリスト上のページの数がこの値を 5 秒間下回ると、ハード・スワッピングが開始されます。

値: 0 ~ 2,147,483,647

省略時の値: 次の数式で自動的に算出されます。

$$\text{vm_page_free_min} + ((\text{vm_page_free_target} - \text{vm_page_free_min}) / 2)$$

- vm_page_free_min — ページ・スワッピングが開始されるしきい値を指定します。空きリスト上のページの数がこの値を下回ると、ページングが開始されます。

値: 0 ~ 2,147,483,647

省略時の値: 20 (ページ, または vm_page_free_reserved の値の 2 倍)。

- `vm_page_free_reserved` — メモリの割り当てを特権タスクに制限する時期を決めるしきい値を指定します。空きリスト上のページの数がこの値を下回ると、特権タスクのみがメモリを取得できるようになります。

値: 1 ~ 2,147,483,647
省略時の値: 10 (ページ)

- `vm_page_free_target` — 空きページ・リスト上のページの数がこの値まで増えると、ページングが停止します。

省略時の値は、システム上で利用できる管理対象メモリの容量に従って決まり、次の表のとおりです。

使用可能なメモリ (MB)	<code>vm_page_free_target</code> (ページ)
512 未満	128
512 ~ 1023	256
1024 ~ 2047	512
2048 ~ 4095	768
4096 以上	1024

`vm_page_free_optimal`、`vm_page_free_min`、`vm_page_free_target` 属性の値は、システムをリブートすることなく変更できます。カーネル・サブシステム属性の変更についての詳細は第 3 章を参照してください。

チューニングするかどうかの判断

システムでページングが発生していない場合は、`vm_page_free_optimal` の値を変更しないでください。

`vm_page_free_optimal` 属性の値を小さくすると、対話処理の応答速度が向上しますが、スループットは低下します。

`vm_page_free_optimal` 属性の値を大きくすると、長期間スリープしているスレッドがメモリ外に移動され、メモリが解放されて、スループットが向上します。スワップ・アウトされるプロセスが多いほど、実際に実行しているプロセスが少なくなり、より多くの仕事を処理できるようになります。ただし、スワップ・アウトされたプロセスが必要になった場合の待ち時間が長くなり、対話処理の応答速度が遅くなる場合があります。

`vm_page_free_optimal` の値は、一度に 2 ページだけ大きくしてください。vm サブシステムの `vm_page_free_target` 属性の値より大きくしないでください。

12.5.3 タスク・スワッピングを積極的に行うようにする

空きページ・リストがスワッピングしきい値 (vm サブシステムの `vm_page_free_swap` 属性で指定) より下回った場合、スワッピングが開始されます。スワッピングが頻繁に発生しないようにしてください。アイドル・タスクを積極的にスワップ・アウトするかどうかを、指定することができます。スワッピングについての詳細は、12.1.4 項を参照してください。

関連する属性

vm サブシステムの `vm_aggressive_swap` 属性は、タスク・スワップがアイドル・タスクを積極的にスワップ・アウトするかどうかを指定します。

値: 1 または 0
省略時の値: 0 (無効)

チューニングするかどうかの判断

タスク・スワッピングを積極的に行うと、システムのスループットが向上します。ただし、対話処理の応答性能が低下します。通常は、タスク・スワッピングを積極的に行う必要はありません。

`vm_aggressive_swap` 属性の値は、リブートすることなく変更できます。カーネル属性の変更についての詳細は、第 3 章を参照してください。

12.5.4 スワッピングを避けるために常駐セットのサイズを制限する

Tru64 UNIX は、特に指定しないかぎりプロセスの常駐セットのサイズを制限しません。アプリケーションでは、`setrlimit()` コールで `RLIMIT_RSS` リソース値を指定することにより、プロセスごとのメモリ常駐ページ数の限界値を設定できます。ただし、アプリケーションでのプロセスの常駐セットのサイズの制限は必須ではなく、システム・ワイドの省略時の限界値はありません。したがって、プロセスの常駐セットのサイズは、システムのメモリ制限によってのみ制限されます。メモリの要求が空きページの数を上回ると、常駐セットのサイズが大きいプロセスがスワッピングの候補となります。スワッピングについての詳細は、12.1.4 項を参照してください。

常駐セットのサイズが大きいためプロセスがスワッピングされるのを防ぐには、常駐セットのサイズに対して、プロセス固有の限界値とシステム・ワイドの限界値を指定します。

関連する属性

常駐セットのサイズ制限に関連する `vm` サブシステム属性を、以下のリストに示します。

- `anon_rss_enforce` — プロセスの常駐セットの大きさの制御や、空きページの窮迫時に、プロセスが使用している可変メモリをスワップ・アウトする (プロセスをブロックする) 時期の制御を行うためのレベルを指定します。

値: 制限なし (0), ソフト限界値 (1), ハード限界値 (2)。
省略時の値: 0 (制限なし)

`anon_rss_enforce` を 1 または 2 に設定すると、`vm_rss_max_percent` 属性によって、プロセスの常駐セットのサイズにシステム・ワイドの限界値を強制できるようになります。

`anon_rss_enforce` を 1 (ソフト限界値) に設定すると、`vm_rss_block_target` および `vm_rss_wakeup_target` 属性を設定することによって、プロセスのブロックと、可変メモリのページングを細かく制御できます。

- `vm_rss_max_percent` — すべてのプロセスに対して常駐セットのサイズのシステム・ワイドの限界値となる、可変メモリの総ページ数に対する割合を指定します。この属性の値は、`anon_rss_enforce` 属性が 1 または 2 に設定されている場合にのみ有効です。

値: 0 ~ 100
省略時の値: 100 パーセント

この割合を小さくすると、すべてのプロセスに対して、常駐セットのサイズのシステム・ワイドの限界値を強制することができます。ただし、この限界値は、特権プロセスと非特権プロセスのどちらにも適用され、`setrlimit()` コールによってより大きな常駐セット・サイズをプロセスに設定していても、それが無効になるので注意してください。

- `vm_rss_block_target` — プロセスの常駐セットからの可変メモリのスワッピングを開始する、空きページ数のしきい値を指定します。可変

メモリのページングは、空きページ数がこの値と同じか下回った場合に開始されます。プロセスは、空きページ数が `vm_rss_wakeup_target` 属性で指定された値まで増加するまで、ブロックされます。

値: 0 ~ 2,147,483,647

省略時の値: `vm_page_free_optimal` と同じ。

この値を大きくすると、ハード・スワッピングが開始される前に可変メモリがページングされます。この値を小さくすると、可変メモリのページングは、ハード・スワッピングが開始される時点よりも遅くなります。

- `vm_rss_wakeup_target` — 可変メモリがスワップ・アウトされているプロセスのブロックを解除する、空きページ数のしきい値を指定します。空きページの数がこの値まで増えると、プロセスのブロックが解除されます。

値: 0 ~ 2,147,483,647

省略時の値: `vm_page_free_optimal` と同じ。

この値を大きくすると、タスクのブロックが解除される前に、より多くのメモリが解放されます。この値を小さくするとタスクのブロック解除が早くなりますが、解放されるメモリは少なくなります。

- `vm_page_free_optimal` — ハード・スワッピングが開始されるしきい値を指定します。空きリスト上のページの数がこの値を 5 秒間下回ると、ハード・スワッピングが開始されます。

値: 0 ~ 2,147,483,647

省略時の値: 次の式で自動的に計算されます。

$$vm_page_free_min + ((vm_page_free_target - vm_page_free_min) / 2)$$

チューニングするかどうかの判断

システムでページングが発生していない場合は、常駐セットのサイズを制限する必要はありません。

常駐セットのサイズを制限する場合、特定プロセスの場合とシステム・ワイドの場合のどちらでも、`vm` サブシステム属性 `anon_rss_enforce` を併用して、常駐セットのサイズのソフト限界値とハード限界値のどちらかを設定しなければなりません。

ハード限界値を有効にすると、タスクの常駐セットはこの限界値を超えることはできません。タスクがハード限界値に達すると、タスクの可変メモリのページがスワップ領域に移動され、常駐セットのサイズが限界値以下に抑えられます。

ソフト限界値を有効にすると、可変メモリのページングが、次の条件で開始されます。

- タスクの常駐セットがシステム・ワイドまたはプロセスごとの限界値を超えた場合
- 空きページ・リストのページの数 `vm_rss_block_target` 属性の値より少ない場合

`anon_rss_enforce` 属性を変更したときには、システムのリブートが必要です。 `vm_page_free_optimal` , `vm_rss_maxpercent` , `vm_rss_block_target` , `vm_rss_wakeup_target` 属性は、システムをリブートすることなく変更できます。

12.5.5 変更ページの事前書き出しを管理する

vm サブシステムは、変更された (ダーティ) ページをディスクに事前書き出しすることによって、メモリ不足を防止しようとします。事前書き出しされたページを再生する場合、仮想メモリ・サブシステムはページが有効であることを確認するだけですみ、性能が向上します。仮想メモリ・サブシステムは、空きリスト上のページがもうすぐなくなると予測した場合、最も古い非アクティブ・ページおよび UBC LRU ページをディスクに事前書き出しします。事前書き出しに関連する属性は、チューニングできます。事前書き出しについての詳細は、12.1.4.1 項を参照してください。

関連する属性

変更ページの事前書き出しに関連する vm サブシステムの属性を、以下のリストに示します。

- `vm_ubcdirtypercent` — UBC がディスクへの書き込みを開始する時点のダーティ (変更) ページの割合を指定します。

値: 0 ~ 100

省略時の値: 10 パーセント

- `vm_page_prewrite_target` — メモリ不足の心配があるときに、`vm` サブシステムがディスクに事前書き出しする変更 UBC (LRU) ページの最大数を指定します。

値: 0 ~ 2,147,483,647

省略時の値: `vm_page_free_target * 2`

- `vm_page_free_target` — 空きページ・リストのページの数がこの値まで増えると、ページングが停止します。

システム上で使用できる管理対象メモリの容量に従って決まる省略時の値を、次の表に示します。

使用可能なメモリ (MB)	<code>vm_page_free_target</code> (ページ)
512 未満	128
512 ~ 1023	256
1024 ~ 2047	512
2048 ~ 4095	768
4096 以上	1024

`vm_page_prewrite_target` 属性と `vm_ubcdirtypercent` 属性は、システムをリブートすることなく変更できます。

チューニングするかどうかの判断

システムでページングが発生していない場合、`vm_page_prewrite_target` 属性の値を変更する必要はありません。

`vm_page_prewrite_target` 属性の値を小さくすると、ピーク時の処理性能が向上しますが、メモリ不足が発生したときには性能が大幅に低下します。

`vm_page_prewrite_target` 属性の値を大きくすると、次のようになります。

- メモリ不足が発生しても性能が大幅に低下することはありませんが、ピーク時の処理性能が低下します。
- 連続したディスク入出力が増加しますが、システムがクラッシュした場合の、ファイル・システムの完全性は向上します。

`vm_page_prewrite_target` 属性の値は、64 ページずつ大きくします。

UBC LRU ダーティ・ページの事前書き出しの頻度を高くするには、`vm_ubcdirtypercent` 属性の値を 1 (パーセント) ずつ小さくします。

カーネル属性の変更については、第 3 章を参照してください。

12.5.6 ページ・イン・クラスタおよびページ・アウト・クラスタのサイズを管理する

仮想メモリ・サブシステムは、必要となるページの数予測して、スワップ・デバイスに対して追加ページの読み込みおよび書き出しを行います。スワップ・デバイスへのアクセス時に追加するページ数を指定することができます。

関連する属性

ページの読み取りおよび書き込みに関連する `vm` サブシステム属性を、以下のリストに示します。

- `vm_max_rdpgio_kluster` — スワップ・デバイスに渡されるページ・イン (読み取り) クラスタの最大サイズをバイト数で指定します。

値: 8192 ~ 131,072

省略時の値: 16,384 (バイト) (16 KB)

- `vm_max_wrpigio_kluster` — スワップ・デバイスに渡されるページ・アウト (書き込み) クラスタの最大サイズをバイト数で指定します。

値: 8192 ~ 131,072

省略時の値: 32,768 (バイト) (32 KB)

`vm_max_rdpgio_kluster` 属性および `vm_max_wrpigio_kluster` 属性を変更したときには、システムのリブートが必要です。カーネル・サブシステム属性の変更についての詳細は、第 3 章を参照してください。

チューニングするかどうかの判断

大規模メモリ・システムでプロセスのスワッピングを行う場合は、`vm_max_rdpgio_kluster` 属性の値を大きくしてください。この値を大きくすると、メモリ内に読み込まれるページ数が多くなり、システムがページ・フォールトの処理に費やす時間が少なくなるため、ピーク時の処理性能が向上しますが、消費するメモリの量が多くなり、システムの性能が低下します。

プロセスのページングとスワッピングを行う場合は、`vm_max_wrp-gio_kluster` 属性の値を大きくしてください。この値を大きくすると、ピーク時の処理性能が向上し、メモリは確保されますが、ページ・インの回数が多くなり、システム全体での処理性能が低下します。

12.5.7 スワップ・パーティションでの入出力要求を管理する

空きページ・リストがスワッピングのしきい値を下回ると、スワッピングが開始されます。スワッピングの頻度が高くなりすぎないようにしてください。同時にスワップ・パーティションに対して発生させることのできる、処理待ちの同期および非同期入出力要求の数を指定することができます。スワッピングについての詳細は、12.1.4 項を参照してください。

同期スワップ要求は、ページ・イン動作とタスクのスワッピングに使用されます。非同期スワップ要求は、ページ・アウト動作と、変更ページの事前書き出しに使用されます。

関連する属性

スワップ・パーティションでの要求に関連する `vm` サブシステム属性のリストを、以下に示します。

- `vm_syncswapbuffers` — スワップ・パーティションで同時に処理待ちにできる同期入出力要求の数を指定します。同期スワップ要求は、ページ・イン動作とタスクのスワッピングに使用されます。

値: 1 ~ 2,147,483,647
省略時の値: 128 (要求)

- `vm_asyncswapbuffers` — スワップ・パーティションで同時に処理待ちにできる非同期入出力要求の数を指定します。非同期スワップ要求は、ページ・アウト動作と、変更ページの事前書き出しに使用されます。

値: 0 ~ 2,147,483,647
省略時の値: 4 (要求)

チューニングするかどうかの判断

`vm_syncswapbuffers` 属性の値は、同時実行をシステムが容易にサポートできるプロセス数の概数と等しくなるようにします。この値を大きくすると、システム全体のスループットが向上しますが、メモリを消費します。

`vm_asyncswapbuffers` 属性の値は、スワップ・デバイスが同時にサポートできる入出力転送数の概数と等しくなるようにします。LSM を使用している場合は、`vm_asyncswapbuffers` 属性の値を大きくし、ページ・イン要求が非同期のページ・アウト要求より遅れるようにします。この値を小さくするとメモリの消費量が多くなりますが、対話処理の応答速度は速くなります。

`vm_syncswapbuffers` 属性と `vm_asyncswapbuffers` 属性は、システムをリブートすることなく変更できます。カーネル・サブシステム属性の変更についての詳細は、第 3 章 を参照してください。

12.6 共用メモリ用の物理メモリの予約

粒度ヒントにより、物理メモリの一部を、ブート時に共用メモリ用として予約できます。この機能により、変換索引バッファ (TLB) で 1 ページ以上のメモリがマッピングでき、共用ページ・テーブル・エントリ機能が有効になります。共用ページ・テーブル・エントリ機能により、キャッシュのヒット率が高くなる可能性があります。

データベース・サーバによっては、粒度ヒントを使用すると、実行時の性能が 2 ~ 4 パーセント向上し、共用メモリのデタッチ時間が短くなります。粒度ヒントを使用すべきかどうかは、データベース・アプリケーションのドキュメントを参照してください。

通常のアプリケーションでは、粒度ヒントの代わりに、セグメント化共用メモリ (SSM) 機能 (省略時の機能) を使用してください。

粒度ヒントを有効にするには、`vm` サブシステムの `gh_chunks` 属性に値を指定しなければなりません。さらに、粒度ヒントをより効率的にするには、アプリケーションを変更して、共用メモリ・セグメントの開始アドレスとサイズを、8 MB 境界に合わせます。

粒度ヒントを有効にする方法については、12.6.1 項および 12.6.2 項を参照してください。

12.6.1 粒度ヒントを使用するようにカーネルをチューニングする

粒度ヒントを使用するには、ブート時に共用メモリ用に予約する、4 MB の物理メモリ・チャンクの数指定しなければなりません。このメモリは、他の用途には使用できず、システムに返却したり再生 (再利用) することもできません。

共用メモリ用にメモリを予約するには、`gh_chunks` 属性にゼロ以外の値を指定します。たとえば、4 GB のメモリを予約する場合は、`gh_chunks` の値として 1024 を指定します (1024 * 4 MB = 4 GB)。512 を指定すると、2 GB のメモリが予約されます。

`gh_chunks` 属性に指定する値は、データベース・アプリケーションにより異なります。プロセスと UBC から利用できるメモリが少なくなるため、メモリを過度に予約しないでください。

注意

粒度ヒントを有効にした場合は、`ipc` サブシステムの `ssm_threshold` 属性の値に 0 を設定して、セグメント化共用メモリを無効にしてください。

予約したメモリの量が適切かどうかを確認することができます。たとえば、`gh_chunks` 属性の値にまず 512 を指定します。その後、共用メモリを割り当てるアプリケーションが動作している間に、次の `dbx` コマンドを入力します。

```
# /usr/ucb/dbx -k /vmunix /dev/mem

(dbx) px &gh_free_counts
0xffffffff0000681748
(dbx) 0xffffffff0000681748/4x
ffffffff0000681748: 00000000000000402 0000000000000004
ffffffff0000681758: 0000000000000000 0000000000000002
(dbx)
```

上記の例は、次の情報を示しています。

- 1 番目の数 (402) は、512 ページ・チャンク (4 MB) の数を示します。
- 2 番目の数 (4) は、64 ページ・チャンクの数を示します。
- 3 番目の数 (0) は、8 ページ・チャンクの数を示します。
- 4 番目の数 (2) は、1 ページ・チャンクの数を示します。

メモリを節約するには、共用メモリを使用するアプリケーションが動作している間に空いている 512 ページ・チャンクが 1 ~ 2 個だけになるまで、`gh_chunks` 属性の値を小さくします。

次の `vm` サブシステム属性も、粒度ヒントに影響します。

- `gh_min_seg_size` — 共用メモリ・セグメントのサイズを指定します。このサイズを超えるメモリは `gh_chunks` 属性で予約されたメモリから割り当てられます。省略時の値は 8 MB です。
- `gh_fail_if_no_mem` — 1 (省略時の値) が設定されている場合、`shmget` 関数は、要求したセグメント・サイズが `gh_min_seg_size` 属性で指定された値よりも大きく、`gh_chunks` 領域内に要求を満たすだけのメモリがないときには、失敗を示す結果を返却します。

`gh_fail_if_no_mem` 属性の値が 0 の場合は、`gh_chunks` 属性で予約されているメモリの量よりも要求が大きいときには、ページング可能メモリ領域のメモリを切り出して要求全体が満たされます。

- `gh_keep_sorted` — 粒度ヒント用に予約するメモリをソートするかどうかを指定します。省略時は、予約されたメモリはソートされません。
- `gh_front_alloc` — 粒度ヒント用に予約したメモリを、下位の物理メモリ・アドレスから割り当てる (省略時の設定) かどうかを指定します。この機能は、メモリ・ボードの数が奇数の場合に便利です。

さらに、アタッチ要求のサイズおよびアドレスが境界に合っていないことを示すメッセージが、システム・コンソールに表示されます。境界が合っていないアタッチのメッセージは、1 つの共用メモリ・セグメントに対して 1 回だけ表示されます。

カーネル・サブシステム属性の変更については、第 3 章を参照してください。

12.6.2 粒度ヒントを使用するようにアプリケーションを変更する

共用メモリ・セグメントの開始アドレスとサイズを 8 MB 境界に合わせると、粒度ヒントの効率を良くできます。

第 3 レベルのページ・テーブル・エントリを共用するには、共用メモリ・セグメントのアタッチ・アドレス (`shmat` 関数で指定) と共用メモリ・セグメントのサイズ (`shmget` 関数で指定) が、8 MB 境界に合っていなければなりません。これは、アドレスとサイズの下位 23 ビットが 0 でなければならないということです。

アタッチ・アドレスと共用メモリ・セグメント・サイズは、アプリケーションが指定します。また、System V の共用メモリでは、最大 2 GB - 1 までの共用メモリ・セグメント・サイズが使用できます。2 GB より大きい共用メモリ・セグメントを必要とするアプリケーションでは、複数のセグメントを使用

することでこのような領域を作成できます。この場合、ユーザがアプリケーションに指定する共用メモリの合計サイズは、8 MB 境界に合っていない限りなりません。さらに、System V の共用メモリ・セグメントの最大サイズを指定する `shm_max` 属性の値は、8 MB 境界に合っていない限りなりません。

アプリケーションに指定する共用メモリ・サイズの合計が 2 GB より大きい場合は、`shm_max` 属性の値に 2139095040 (0x7f800000) を指定できます。この値は、`shm_max` 属性に指定できる最大値 (2 GB - 8MB) で、この値でもページ・テーブル・エントリは共有されます。

ページ・テーブル・エントリが共用されているか確認するには、次の一連の `dbx` コマンドを使用します。

```
# /usr/ucb/dbx -k /vmunix /dev/mem

(dbx) p *(vm_granhint_stats *)&gh_stats_store
struct {
    total_mappers = 21
    shared_mappers = 21
    unshared_mappers = 0
    total_unmappers = 21
    shared_unmappers = 21
    unshared_unmappers = 0
    unaligned_mappers = 0
    access_violations = 0
    unaligned_size_requests = 0
    unaligned_attachers = 0
    wired_bypass = 0
    wired_returns = 0
}
(dbx)
```

最高の性能を得るには、`shared_mappers` カーネル変数が共用メモリ・セグメントの個数と同じで、`unshared_mappers`、`unaligned_attachers`、および `unaligned_size_requests` 変数がゼロでなければなりません。

共用メモリがどのように共用メモリ・セグメントに分割されるかによって、共用されないセグメントが存在することもあります。これは、8 MB 境界に合っているのが、開始アドレスとサイズのどちらかだけの場合に発生します。この状況は、避けられない場合もあります。多くの場合、`total_unmappers` の値が `total_mappers` の値よりも大きくなります。

共用メモリのロックでは、単一のロックがロックのハッシュ配列に変更されます。vm サブシステムの `vm_page_lock_count` 属性の値を変更することで、ロックのハッシュ配列のサイズを変更できます。省略時の値は 0 です。

12.7 ビッグ・ページによる性能改善

ビッグ・ページ・メモリ割り当てでは、仮想メモリの 1 つのページを 8, 64, または 512 ページの物理メモリに対応づけることができます。物理メモリの現在のページ・サイズは 8 KB なので、仮想メモリの 1 つのページに、64, 512, または 4906 KB のメモリに対応づけることができます。ビッグ・ページを使用すると、変換索引バッファ (TLB) のヒット・ミスによる性能への影響を最小限に抑えることができます。その結果、大量のデータをマップする必要があるアプリケーションの性能が改善されることがあります。

ブート時にメモリを予約し System V の共用メモリでのみ使用できる粒度ヒントと異なり、ビッグ・ページは実行時にメモリを割り当て、System V の共用メモリ、スタック・メモリ、およびテキスト・セグメントの他に、可変メモリ (たとえば、`mmap` および `malloc`) をサポートしています。

ビッグ・ページ・メモリの割り当ては、大量の物理メモリ・リソースを備えたシステムで稼働している大規模データベースのように、メモリを多用するアプリケーションで使ったときに、最も効果があります。メモリ・リソースが限られているシステムや、負荷の重さに比較してメモリ・リソースが不足しているシステムでは、ビッグ・ページの使用はお勧めできません。同様に、大量のメモリ・ブロックを必要とする、メモリ多用型のアプリケーションがシステムで稼働していない場合は、ビッグ・ページを使用しても効果が得られない可能性があります。

12.7.1 ビッグ・ページを使用する

ビッグ・ページの有効化と使用は、以下の `vm` カーネル・サブシステムの属性によって制御します。

`vm_bigpg_enabled` — ビッグ・ページの有効化

ビッグ・ページを有効 (1) にするか、無効 (0) にします。

ビッグ・ページを有効にすると、自動的に粒度ヒントが無効になります。`vm_bigpg_enabled` を 1 にしたときには、`gh_chunks`, `rad_gh_regions`, および関連する属性は無視されます。

ビッグ・ページを無効にしたときには、関連する `vm_bigpg*` 属性は、無視されます。

省略時の値: 0 (無効)

ブート時にのみ設定できます。

vm_bigpg_thresh — ページ・サイズごとの空きメモリの配分

4 種類のページ・サイズ (8, 64, 512, および 4096 KB) ごとに、空きページ・リストに確保する物理メモリの割合 (パーセント) です。

メモリのページが解放されたときには、より大きなページを形成するために、解放されたページとそれに隣接するページの結合が試みられます。8 KB のページが解放されたときには、64 KB のページを形成するために、解放されたページと他の 7 つの隣接するページの結合が試みられます。結合に成功したときには、64 KB のページが解放されたことになるため、512 KB のページを形成するために、そのページと他の 7 つの 64 KB ページの結合が試みられます。このページは、可能ならば、4 MB のページを形成するために、他の 7 つの 512 KB のページと結合されます。この結合処理は、これで終わります。

vm_bigpg_thresh 属性は、ページ・サイズごとの空きメモリの結合を開始するメモリ量のしきい値を指定します。vm_bigpg_thresh が 0 パーセントの場合は、サイズが 8, 64, 512 KB のページが解放されたときには、必ずそのサイズのページの結合が試みられます。その結果、小さいページがすべて結合され、空きページのサイズがすべて 4096 KB になる可能性があります。

vm_bigpg_thresh が 6 パーセント (省略時の設定) の場合は、システム・メモリの 6 パーセント以上が 8 KB ページになっているときのみ、8 KB ページの結合が試みられます。他の大きいページ・サイズについても同じことがいえます。結果として、空きページのサイズは、6 パーセントが 8 KB になり、6 パーセントが 64 KB になり、6 パーセントが 512 KB になります。残りの空きページのサイズは 4096 KB になります。これは、空きメモリが十分あり、システム・メモリの 6 パーセントを 512 KB ページに割り当てることができることを前提としています。空きメモリが減少したときには、最も大きいページ・サイズ (4096 KB) への空きページの割り当てが最初に影響を受けます。続いて、512 KB ページへの割り当て、64 KB ページへの割り当ての順に影響を受けます。

vm_bigpg_thresh の値を小さくするほど、より多くのページが結合されるため、小さいサイズの使用可能メモリの量が少なくなります。このようにすると、小さいサイズのページの割り当て要求を満たすため

に、大きいページを小さいサイズに分割しなければならないため、性能が低下する可能性があります。 `vm_bigpg_thresh` が大きすぎる場合は、使用可能な大きいサイズのページが少なくなるため、アプリケーションでは、ビッグ・ページの利点を生かすことができなくなります。通常は、省略時の値で十分ですが、システムで稼働する業務で、小さいページを多く必要とする場合は、この値を大きくすることができます。

省略時の値: 6 パーセント。最小値: 0 パーセント。最大値: 25 パーセント。

ブート時と実行時に設定することができます。

12.7.2 メモリ・オブジェクトにビッグ・ページを使用するための条件を指定する

メモリ・オブジェクトの種類ごとに、ビッグ・ページを使用するための条件を指定する属性は、`vm_bigpg_anon`、`vm_bigpg_seg`、`vm_bigpg_shm`、`vm_bigpg_ssm`、および `vm_bigpg_stack` であり、これらの省略時の値は 64 です。これは、プロセスからのメモリ要求のサイズのうち、拡張仮想ページ・サイズの恩恵を受けることができるサイズの最小値を KB 単位で表しています。

この省略時の値 (64) の場合、カーネルは、64 KB 以上のメモリ割り当て要求に対して、要求されたサイズに従って、1 つまたはそれ以上の仮想ページ (サイズは、8 KB、64 KB、512 KB、および 4096 KB が混在) を作成して要求に応えます。この属性値は、ページ・サイズを指定しているわけではありません。つまり、省略時の値の 64 KB は、すべての仮想ページのサイズが 64 KB になることを意味しているわけではありません。代わりに、カーネルは、要求された総メモリ量に最も適したページ・サイズ (またはサイズの組み合わせ) を、要求に強制される境界合わせ上の制限の範囲内で選択します。カーネルは、64 KB より小さいメモリ割り当て要求に対しては、1 つの仮想ページを 8 KB の物理メモリにマップする省略時のアルゴリズムを使用して処理します。

この属性の値を 64 より大きくすると、ビッグ・ページ・メモリ割り当ての恩恵を受けることができるアプリケーションが、64 の場合のサブセットに制限されます。たとえば、属性の値を 8192 にすると、8192 KB 以上のサイズで割り当てを要求するプログラムのみに 8 KB より大きい仮想ページが割り当てられます。

`vm_bigpg_anon` , `vm_bigpg_seg` , `vm_bigpg_shm` , `vm_bigpg_ssm` , または `vm_bigpg_stack` の値を 0 にすると、それぞれの属性に対応するメモリ・オブジェクトの種類に対するビッグ・ページ・メモリ割り当てが無効になります。たとえば、`vm_bigpg_anon` を 0 にすると、可変メモリの割り当てを要求するプロセスに対するビッグ・ページ・メモリの割り当てが無効になります。特定の種類のメモリに対するビッグ・ページ・メモリ割り当てを無効にしても、はっきりした効果は期待できません。

システムがブートされた後の `vm_bigpg_anon` , `vm_bigpg_seg` , `vm_bigpg_shm` , `vm_bigpg_ssm` , または `vm_bigpg_stack` への変更は、新たに実行されるメモリ割り当てにのみ影響を与えます。実行時の変更は、すでに実施済みのメモリ・マッピングには影響を与えません。

以下の属性に 1 ~ 64 の値を設定すると、64 を設定したものとみなされます。

注意

以下のオブジェクト毎の属性を省略時の値 (64 KB) 以外に変更するときには、最寄のサポート担当者に相談してください。

`vm_bigpg_anon` — 可変メモリ用のビッグ・ページ

カーネルがプロセスのアドレス空間内の仮想ページを複数の物理ページに対応づけるようになるために、ユーザ・プロセスが要求しなければならない可変メモリの最小量 (KB 単位) を設定します。可変メモリは、`mmap()` , `nmmap()` , `malloc()` , および `amalloc()` の呼び出しで要求されます。メモリ・マップ・ファイルに対しては、可変メモリはサポートされていません。

注意

プロセスの常駐セット・サイズを制限するための `anon_rss_enforce` 属性の値が 1 または 2 のときには、この設定が優先され、可変メモリとスタック・メモリのビッグ・ページ・メモリ割り当てが無効になります。可変メモリとスタック・メモリのビッグ・ページ・メモリ割り当てを有効にしたいときには、`anon_rss_enforce` に 0 を設定してください。

省略時の値: 64 KB

ブート時と実行時に設定可能。

`vm_bigpg_seg` — プログラム・テキスト・オブジェクト用のビッグ・ページ

カーネルがプロセスのアドレス空間内の仮想ページを複数の物理ページに対応づけるようになるために、ユーザ・プロセスが要求しなければならないプログラム・テキスト・オブジェクトの最小量 (KB 単位) を設定します。プログラム・テキスト・オブジェクトの割り当ては、プロセスがプログラムを実行するときやシェアード・ライブラリをロードするときに発行されます。`vm_segment_cache_max` と `vm_segmentation` の説明も参照してください。

省略時の値: 64 KB

ブート時と実行時に設定可能。

`vm_bigpg_shm` — 共用メモリ用のビッグ・ページ

カーネルがプロセスのアドレス空間内の仮想ページを複数の物理ページに対応づけるようになるために、ユーザ・プロセスが要求しなければならない System V 共用メモリの最小量 (KB 単位) を設定します。System V 共用メモリの割り当ては、`shmget()`、`shmctl()`、および `nshmget()` の呼び出しで発行されます。

省略時の値: 64 KB

ブート時と実行時に設定可能。

`vm_bigpg_ssm` — セグメント化共用メモリ用のビッグ・ページ

カーネルがプロセスのアドレス空間内の仮想ページを複数の物理ページに対応づけるようになるために、ユーザ・プロセスが要求しなければならないセグメント化共用メモリ (共用ページ・テーブルを用いた System V 共用メモリ) の最小量 (KB 単位) を設定します。セグメント化共用メモリの要求は、`shmget()`、`shmctl()`、および `nshmget()` の呼び出しで発行されます。

`ssm_threshold` IPC 属性の値が 0 のときには、`vm_bigpg_ssm` 属性は無効になります。`ssm_threshold` の値は、`SSM_SIZE` の値以上でなければなりません。省略時の設定では、`ssm_threshold` の値は `SSM_SIZE` です。詳細は、`sys_attrs_ipc(5)` を参照してください。

省略時の値: 64 KB

ブート時と実行時に設定可能。

`vm_bigpg_stack` — スタック・メモリ用のビッグ・ページ

カーネルがプロセスのアドレス空間内の仮想ページを複数の物理ページに対応づけるようになるために、ユーザ・プロセス・スタックに必要なメモリの最小量 (KB 単位) を設定します。スタック・メモリは、ユーザのためにカーネルが自動的に割り当てます。

プロセスの常駐セット・サイズを制限するための `anon_rss_enforce` 属性の値が 1 または 2 のときには、この設定が優先され、可変メモリとスタック・メモリのビッグ・ページ・メモリ割り当てが無効になります。可変メモリとスタック・メモリのビッグ・ページ・メモリ割り当てを有効にしたいときには、`anon_rss_enforce` に 0 を設定してください。

省略時の値: 64 KB

ブート時と実行時に設定可能。

詳細は、`sys_attrs_vm(5)` を参照してください。



CPU 性能の管理

CPU リソースを最適化することで、性能を改善できることがあります。この章では、次のような作業を実行する方法について説明します。

- CPU モニタリング・ツールを使用して CPU 性能についての情報を得る (13.1 節)
- プロセッサを追加するか、あるいはクラス・スケジューラによりタスクとアプリケーションの実行を制御することで CPU 性能を改善する (13.2 節)

13.1 CPU 性能に関する情報のモニタリング

表 13-1 では、CPU の使用状況について情報収集するためのツールについて説明します。

表 13-1: CPU のモニタリング・ツール

名前	説明	参照先
ps	実行中のプロセスについて、現在の統計情報を表示する。この情報には CPU の使用状況、プロセッサおよびプロセッサ・セット、スケジュールの優先順位などがある。	12.3.2 項
vmstat	プロセスのスレッド、仮想メモリの使用状況 (ページ・リスト、ページ・フォールト、ページ・イン、ページ・アウト)、割り込み、CPU の使用状況 (ユーザ時間、システム時間およびアイドル時間の割合) の情報を表示する。最初にブート時点以降の統計情報が表示され、次に指定した時間間隔で統計情報が表示される。	12.3.1 項

表 13-1: CPU のモニタリング・ツール (続き)

名前	説明	参照先
uptime	システムの平均負荷と、直前 5 秒間、直前 30 秒間、直前 60 秒間の実行キュー内のジョブ数を表示する。また、uptime コマンドは、システムにログインしているユーザ数と現在までにシステムが稼働した時間も表示する。	13.1.1 項
(kdbx) cpustat	CPU の統計情報 (さまざまな CPU 状態での経過時間の割合など) を表示する。	13.1.2 項
(kdbx) lockstats	システム上の各 CPU について、ロック・クラスごとのロック統計情報を表示する。	13.1.3 項
sys_check	システムの構成を分析し、統計情報を表示する。また、カーネル変数の設定およびメモリと CPU リソースをチェックし、SMP システムおよびカーネル・プロファイルの性能データと統計情報を表示する。	詳細は、2.3.3 項または sys_check(8) を参照。
プロセス・チューナ	実行中のプロセスについて、現在の統計情報を表示する。CDE の「アプリケーション・マネージャ」から「プロセス・チューナ」のグラフィカル・ユーザ・インタフェース (GUI) を起動して、プロセスの一覧とその特性の表示、自身または全ユーザについて実行中のプロセスの表示、プロセスの優先順位の表示および変更、プロセスへのシグナル送信を行う。 プロセスのモニタリング中に、表示するパラメータ (CPU の使用量の割合、仮想メモリのサイズ、状態、nice 優先順位) を選択したり、表示をソートしたりすることができる。	13.2.2.1.1 項

表 13-1: CPU のモニタリング・ツール (続き)

名前	説明	参照先
monitor	稼働中のシステムでさまざまな性能データを収集し、情報をグラフィカルな形式で表示するか、またはバイナリ・ファイルに保存する。	詳細は、monitor(3)を参照。
top	システムの状態 (CPU リソースを大量に使用するプロセスのリストなど) についてのレポートを継続的に表示する。	top コマンドは Tru64 UNIX Freeware CD-ROM で提供。詳細は、 ftp://eecs.nwu.edu/pub/top を参照。
ipcs -a	現在アクティブなメッセージ・キュー、共用メモリ・セグメント、セマフォ、リモート・キュー、ローカル・キュー・ヘッダについて、プロセス間通信 (IPC) の統計情報を表示する。ipcs -a コマンドで表示される情報のうち、QNUM、CBYTES、QBYTES、SEGSZ、NSEMS の各フィールドに表示される情報は特に役に立つ。	詳細は、ipcs(1) を参照。
w	現在の時刻、システムが開始されてから経過した時間、システムにログインしているユーザ、直前 5 秒間、直前 30 秒間、直前 60 秒間の実行キュー内のジョブ数を表示する。	詳細は、w(1) を参照。
xload	システムの平均負荷を、定期的に更新されるヒストグラムで表示する。	詳細は、xload(1X) を参照。

以降の節では、ps、vmstat、uptime、cpustat、および lockstats コマンドについて詳しく説明します。

13.1.1 uptime コマンドを使って平均負荷をモニタリングする

uptime コマンドは、システムが稼働した時間と平均負荷を表示します。平均負荷では、ディスク入出力を待機しているジョブの数、nice または renice コマンドで優先順位が変更されたアプリケーションの数をカウント

します。平均負荷の数値は、直前 5 秒間、直前 30 秒間、直前 60 秒間に実行キューにあった平均ジョブ数です。

uptime コマンドの例を次に示します。

```
# /usr/ucb/uptime
1:48pm up 7 days,  1:07,  35 users,  load average: 7.12, 10.33, 10.31
```

このコマンドでは、現在の時刻、システムが起動されてからの経過時間、システムにログインしているユーザの数、直前 5 秒間、直前 30 秒間、直前 60 秒間の平均負荷が表示されます。

このコマンド出力からは、負荷が増えているか減っているかが判断できません。許容できる平均負荷は、システムのタイプと使い方によって異なります。一般に、大規模なシステムでは負荷が 10 であれば高く、負荷が 3 であれば低いといえます。ワークステーションの負荷は 1 か 2 になるようにしてください。

負荷が高い場合は、どのようなプロセスが実行されているか `ps` コマンドで確認します。アプリケーションによっては、オフピーク時に実行してもよいものがあります。`ps` コマンドについての詳細は、12.3.2 項を参照してください。

`nice` または `renice` コマンドを用いてアプリケーションの優先度を下げ、CPU 時間の使用量を減らすこともできます。詳細は、`nice(1)` および `renice(8)` を参照してください。

13.1.2 kdbx デバッガ lockstat 拡張を使って CPU 使用状況をチェックする

kdbx デバッガの `cpustat` 拡張は、CPU の統計情報を表示します。たとえば、次の各状態で CPU 時間が経過した割合などです。

- ユーザ・レベル・コードの実行
- システム・レベル・コードの実行
- `nice` 機能で設定した優先度での実行
- アイドル状態
- 待ち状態 (入出力のペンディングによるアイドル)

kdbx デバッガの `cpustat` 拡張は、アプリケーションの開発者が、システム全体の並行処理がどの程度効率的に行われているかを判断する際に役に立ちます。

特に指定しなければ、kdbx `cpustat` 拡張は、システム内の全 CPU について統計情報を表示します。たとえば次のようになります。

```
# /usr/bin/kdbx -k /vmunix /dev/mem
(kdbx)cpustat
Cpu      User (%)   Nice (%)  System (%)  Idle (%)   Wait (%)
=====
0         0.23       0.00      0.08        99.64      0.05
1         0.21       0.00      0.06        99.68      0.05
```

詳細は、『*Kernel Debugging*』と `kdbx(8)` を参照してください。

13.1.3 kdbx デバッガ `lockstat` 拡張を使ってロックの使用状況をチェックする

kdbx デバッガの `lockstats` 拡張は、システム内の各 CPU について、ロック・クラスごとに次のようなロック統計情報を表示します。

- 構造体のアドレス
- ロック統計情報が記録されるロックのクラス
- ロック統計情報が記録される CPU
- ロックのインスタンス数
- プロセスがロックを取得しようとした回数
- プロセスがロックを取得しようとして失敗した回数
- プロセスがロックに失敗した回数の割合
- プロセスがロック待ちに費やしたトータル時間
- 1つのプロセスがロックを待った時間の最大値
- 1つのプロセスがロックを待った時間の最小値

たとえば次のように入力します。

```
# /usr/bin/kdbx -k /vmunix /dev/mem
(kdbx)lockstats
```

詳細は、『*Kernel Debugging*』と `kdbx(8)` を参照してください。

13.2 CPU 性能の改善

システムは、使用できる CPU 時間を競合するプロセス間で効率的に配分して、ユーザおよびアプリケーションのニーズに合った性能を実現できなければなりません。CPU の使用状況を最適化することで、性能を改善できます。

CPU の性能を改善するためのガイドラインを表 13-2 に示します。

表 13-2: 主な CPU 性能改善のガイドライン

ガイドライン	性能上の利点	欠点
プロセッサを追加する (13.2.1 項)。	CPU リソースが増加する。	適用はマルチプロセッシング・システムに限られ、仮想メモリの性能に影響することがある。
クラス・スケジューラを使用する (13.2.2 項)。	CPU リソースが重要なアプリケーションに割り当てられる。	なし
ジョブの優先順位を設定する (13.2.3 項)。	重要なアプリケーションの優先順位が確実に最高になる。	なし
ジョブをオフピークの時間帯にスケジューリングする (13.2.4 項)。	システムの負荷が分散される。	なし
advfsd デーモンを停止する (13.2.5 項)。	必要とする CPU パワーが減る。	AdvFS グラフィカル・ユーザ・インターフェースを使用していない場合にのみ適用できる。
ハードウェア RAID を使用する (13.2.6 項)。	CPU のディスク入出力オーバーヘッドが軽減され、ディスク入出力の性能が向上する。	コストが増加する。

以降の節では、CPU リソースを最適化する方法について説明します。CPU リソースを最適化しても性能の問題が解決しない場合は、CPU を高速なプロセッサにアップグレードする必要があります。

13.2.1 プロセッサを追加する

マルチプロセッシング・システムでは、プロセッサを追加して計算能力を強化できます。マルチプロセッシングの効果が最も大きい作業負荷は、同時に実行できる複数のプロセスまたは複数のスレッドからなる処理です。このような

処理には、データベース管理システム (DBMS) サーバ、インターネット・サーバ、メール・サーバ、計算サーバがあります。

アイドル時間の割合が小さいマルチプロセッシング・システムでは、プロセッサを追加することで改善できます。アイドル時間をチェックする方法については、12.3.1 項を参照してください。

プロセッサを追加する前に、仮想メモリまたは入出力サブシステムで性能の問題が発生していないことを確認しなければなりません。たとえば、メモリ・リソースが十分でないシステムでは、プロセッサの数を増やしても性能は改善できません。

さらに、プロセッサの数を増やすと、入出力とメモリ・サブシステムの要求が増加し、ボトルネックが生じることもあります。

プロセッサを追加し、システムがメタデータを多用する (すなわち、小規模のファイルをいくつもオープンしてくり返しアクセスするシステム) 場合は、ライトバック・キャッシュを備えた RAID コントローラを使用して同期書き込み動作の性能を改善できます (9.4 節を参照)。

13.2.2 クラス・スケジューラを使用する

クラス・スケジューラでは、タスクやアプリケーションがプロセッサ (CPU) にアクセスする時間を制限することで、その実行を制御する手段が提供されます。たとえば、プリント・スプーラのようなデーモンではアクセス時間を少なくします。CPU は他のタスクの実行に、より時間を割くことができます。このために、プリント・デーモン `/usr/sbin/lpd` には、使用可能な CPU 時間の特定の割合以上は割り当てないようにします。ユーザの UID (ユーザ識別子) のようなリソース UID でクラス分けして、各クラスに必要な CPU アクセス時間を割り当てることができます。

この機能によって、最も重要な業務に必要な処理時間が与えられるように、システム・リソースを割り当てることができます。たとえば、2 つのバージョンの業務用データベースを使う場合を考えます。1 つは実際の業務で使用し、もう 1 つは異なるチューニング・パラメータを設定したテスト・コピーです。この場合、テスト・コピーは異なるクラスに割り当て、日常業務に影響を与えないようにします。

クラス・スケジューラを設定して使用するには、以下の手順を実行します。

1. CPU リソースの割り当てを計画する。

2. `class_admin` を使用して、クラス・データベースを設定し、保守する。
3. クラスを作成し、クラスにメンバを追加する。
4. `show` コマンドを使ってクラスのエントリを検証する。
5. エントリをデータベースに保存する。
6. クラス・スケジューリングを有効にして、デーモンを起動する。

クラス・スケジューラのコマンドを使って、以下のように、スケジューリングのモニタリングと制御を行います。

- `stat` のような `class_admin` コマンドを、対話型セッションを実行せずに、コマンド行あるいはシェル・スクリプトで実行します。
- `runclass` コマンドを使用し、特定のクラス用に設定された優先順位に従って、タスクを実行します。

以降の項で、クラス・スケジューリングを使用するための系統立てたアプローチについて説明しますが、必ずしも説明している順番で実行する必要はありません。クラス・スケジューラにアクセスする方法には、次の 2 つがあります。

- 手動

コマンド行から `class_admin` コマンドを実行し、省略時のデータベースを構成し、クラスとクラス・メンバを追加し、そしてクラス・スケジューリングを有効にして CPU リソース共用問題の簡単な解決をはかります。

- グラフィカル・インタフェース

SysMan Menu サブオプションのクラス・スケジューリングで利用できるグラフィカル・ユーザ・インタフェースを使用します。これは、[Monitoring and Tuning] メニュー・オプションの下にあります。

SysMan Menu を実行する方法は、『システム管理ガイド』を参照してください。13.2.2.6 項では、グラフィカル・インタフェースの使用方法を説明しています。正しいデータ・エントリについての補足情報は、オンライン・ヘルプを参照してください。

以下のリファレンス・ページに、クラス・スケジューラ・コマンドとオプションについての詳細が説明されています。

- `class_scheduling(4)`

- `class_admin(8)`
- `runclass(1)`
- `sysman(8)`

`class_admin` コマンドのオンライン・ヘルプを表示するには、次のコマンドを実行します。

```
# /usr/sbin/class_admin help
```

13.2.2.1 クラス・スケジューラの概要

クラス・スケジューラを使用するには、まずデータベース・ファイルを作成し、そのファイルに1つ以上のクラスを設定する必要があります。クラスにはそれぞれ、CPU 値 (利用可能な合計 CPU 時間に対するパーセンテージ) が割り当てられ、これによって処理時間へのアクセスが制御されます。1つのクラスには、1つ以上のアプリケーションまたはアプリケーション群を割り当てることができます。クラスは、次のような一意のシステム・プロセス識別子に従って識別されます。

- **UID** — ユーザ識別子。ユーザ・アカウント (ログイン) ごとに割り当てられる一意の番号。
- **GID** — グループ識別子。複数のユーザ・アカウントが同じグループに属することを示すために割り当ての番号または名前。
- **PID** — プロセス識別子。システムがプロセスごとに割り当ての一意の番号。
- **PGID** — プロセス・グループ識別子。システムがプロセス・グループごとに割り当ての一意の番号。
- **SESS** — セッション識別子。システムがセッションごとに割り当ての一意の番号。

PID, PGID および SESS は、通常、一時的な識別子であるので、リブート後には失われ、タスクが完了すると存在なくなります。これらはデータベースに保存されないの、システムやタスクを再起動すると使用できなくなります。

データベースが確立されたら、クラス・スケジューリングを有効にできます。この操作によって、クラス・スケジューリング・デーモンが起動され CPU アクセス制限が有効になります。この他にも、クラスの見直し、内容

やスケジューリング・パラメータの変更，コンポーネントやクラス全体の削除を行うコマンドがあります。クラス・スケジューリング・データベースを構成して有効にすると，次のことができるようになります。

- `runclass` を使用し，特定のクラスに対して指定した CPU アクセス値に従ってタスク (プロセス) を実行する。たとえば，対話式の操作には，プリント・デーモンのようなバックグラウンド・プロセスよりもはるかに高い値を設定するといったことが可能です。プリント・ジョブに一時的に高い値を使用するには，対話式操作と同じクラスで `lpr` コマンドを実行します。
- `class_admin` コマンドを使用して，スクリプト内からクラス・スケジューリング・コマンドを実行する。

13.2.2.1.1 関連ユーティリティ

プロセスのモニタとチューニングに際して，次のユーティリティも使用できます。

- `nice` コマンド
- プロセス・チューナ (`dxproctuner`) グラフィカル・インタフェース。CDE の「アプリケーション・マネージャ - システム管理」の「モニタリング/チューニング」フォルダで使います。
- SysMan Menu から，`iostat` および `vmstat` コマンドが起動できます。

13.2.2.1.2 クラス・スケジューラの起動

クラス・スケジューラは，コマンド行インタフェースとグラフィカル・ユーザ・インタフェースの両方で利用することができます。クラス・スケジューラは，使用するインタフェースに合わせて，次のような方法で呼び出すことができます。

- SysMan Menu からは，[モニタリング/チューニング] を選択して，次に [クラス・スケジューリング] タスクを選択します。
- コマンド行からは，次のいずれかのコマンドを入力します。

```
# sysman class_sched  
  
# sysman -menu "Class Scheduling"
```

- CDE では、以下の操作を行います (グラフィカル・ユーザ・インタフェースとして CDE を使用していると仮定)。
 1. CDE のフロントパネルから、「アプリケーション・マネージャ」を選択します。
 2. 「システム管理」を選択します。
 3. 「モニタリング/チューニング」を選択します。
 4. クラス・スケジューラのアイコンを選択します。

以降の項では、コマンド行を使った方法を中心に説明し、グラフィカル・インタフェースについては簡単な概略を示します。グラフィカル・インタフェースを使用する方法についての詳細は、オンライン・ヘルプを参照してください。

13.2.2.2 クラス・スケジューリングのプランニング

CPU リソースをどう割り当てるかは、システム環境によって異なり、どのリソースまたは優先順位を考慮するかにも依存します。通常は、ユーザに対する応答時間が長ならないよう、対話式タスクに大きな CPU 率を割り当てます。ほとんどのバッチ・プロセスやバックグラウンド・プロセスには、少ない CPU 率を割り当てますが、特定のバックグラウンド・プロセスでは、大きな CPU 率を必要とする場合もあります。たとえば、夜間バックアップを行おうとする場合、それが適切な時間内に完了しないほど CPU 率を低くしようとは考えないでしょう。

対話式プロセスより優先すべきリアルタイム・タスクが存在する場合は、とるべきアクションは違ってきます。この場合、まずプロセスをクラスに割り当てるベース・ラインを設計します。そして、プロセスをモニタするとともにユーザのフィードバックを集め、それを考慮してクラス間でタスクを移動したり、クラスの CPU アクセス時間を変更したりすることにより、データベースをチューニングすることができます。

クラス・スケジューリングを構成するときは、テスト・プロセスの作成にルート・アカウントを使わないでください。ルート・アカウント・プロセスは、既存の制限付きクラスに割り当てた場合でも、必ず他のプロセスより優先されます。

13.2.2.3 クラス・スケジューラの構成

`class_admin` コマンドを使用して、初期データベースを構成します。このコマンドには次の機能があります。

- クラスのデータベースを作成、管理するための (サブ・コマンドを伴う) 対話式コマンド。このデータベースは、`/etc/class` というバイナリ・ファイルに格納され、手動で変更することはできません。オプションのリストについては、`class>` コマンド・プロンプトで `help` と入力します。
- コマンド・プロンプトで `class_admin` コマンドを実行できる、またはシェル・スクリプトにコマンドを取り込めるコマンド・モード。

`enable` コマンドを使用してクラス・スケジューリングを有効にするには、その前にデータベースを構成する必要があります。`class_admin` コマンドを入力したときにデータベースが存在しなければ、コマンドは対話式セッションを起動し、データベースを構成するよう求めます。スクリプトによって `class_admin` コマンドが起動すると、システムの省略時の設定を使用してデータベースが自動的に構成されます。

次の例は、`class_admin` を使用した対話式構成セッションを示します。実際の出力では、行が 80 カラムに合わせてフォーマットされます。

```
# /usr/sbin/class_admin
                                Class Scheduler Administration

configure:

Shall processes that have not been explicitly
assigned to a defined class be assigned to a
'default' class? Enter (yes/no) [no]: yes

Enforce class scheduling when the CPU is otherwise
idle? (yes/no) [no]: yes

How often do you want the system to reset class usage?
Enter number of seconds (1): 2
class>
```

上の構成値により、次の事項が設定されます。

- スケジューリングするには、プロセスをクラスに割り当てる必要があります。最初のプロンプトに `yes` と答えると、`default` という特別なクラスが作成されます。定義されたクラスに明示的に割り当てられていないプロセスは、この省略時のクラスに割り当てられます。

このプロンプトに `no` と答えた場合、定義済みのクラスに明示的に割り当てられたプロセスだけが、クラス・スケジューリングされます。

- 2 番目のプロンプトに `yes` と答えると、システムがアイドル状態のときには、クラスがそれらに割り当てられた CPU 時間率を超えることができます。 `no` と答えた場合、CPU に他の作業が存在しなくても、クラスは割り当てられた時間率に限定されます。
- 3 番目のプロンプトでは、全部のクラスについての標準リセット時間が設定できます。たとえば、省略時の時間を 1 秒という短い時間に設定した場合、各クラスはより頻繁に CPU にアクセスする機会が与えられますが、1 回のアクセス時間は短くなります。

クラス・スケジューリングの対象となっている対話式ジョブがある場合、応答時間を短くするために小さい数字 (数秒) を使用します。バッチ・ジョブだけをクラス・スケジューリングする場合、応答時間は問題でないので、大きな値を指定しても構いません。

上の例では、省略時のクラスが作成され、現在のプロセスが全部、この省略時のクラスに割り当てられています。CPU がアイドル状態でもクラス・スケジューリングが強制され、クラスの使用は 5 秒ごとにリセットされます。

現在の構成を表示するには、`show` コマンドを使用します。

```
class> show
Configuration:
-Processes not explicitly defined in the database are
  class scheduled.
-If the processor has some idle time, class scheduled
  processes are not allowed to exceed their cpu percentage.
-The class scheduler will check class CPU usage every 2
  seconds.

Class scheduler status: disabled  current database: /etc/class

Classes:

default targeted at 100%:
  class members:
    Every one not listed below
```

プロセスの次のステップは、クラスを作成し、UID や SESS のような適切な識別子を使用して、クラスにタスク、デーモン、またはユーザ・アカウントのようなシステム・プロセスを設定することです

13.2.2.4 クラスの作成と管理

データベースが構成されていれば、次のようにクラスを管理することができます。

- クラスの作成
 - クラスへのプロセスの追加
 - クラスからのプロセスの削除
- 任意のクラスの CPU アクセス値 (時間率) の変更
- 空のまたは空でないクラス全体の破壊
- クラス・メンバと構成設定の詳細表示
- 現在の優先順位の設定に対する実際の CPU 使用統計情報の表示

これらのオプションの一部については、以降の項で簡単に説明します。コマンド・オプションの詳細については、オンライン・ヘルプおよびリファレンス・ページを参照してください。

13.2.2.4.1 クラスを作成する

クラスを作成するには、コマンド・モードを使用するか、または次のように対話式セッションを開始します。

```
# class_admin
class> create high_users 50
```

コマンド・モードの場合は、次のように入力します。

```
# class_admin create batch_jobs 10
batch_jobs created at 10% cpu usage
```

```
changes saved
```

最初のコマンドは、`high_users` という名前のクラスを作成し、CPU の使用上限として 50 パーセントを割り当てます。2 番目のコマンドは、`batch_jobs` という名前のクラスを作成し、10 パーセントの CPU 使用上限を割り当てます。コマンド・モードでは、変更内容が `/etc/class` 内のデータベースに自動的に保存されます。対話式でクラスに変更を加える場合は、`save` コマンドを使用して、データベースに加えた変更を確定します。`quit` コマンドでセッションを終了する際に保存されていない変更があれば、対話式セッションを終了する前に、変更内容を保存するかまたは廃棄するかを尋ねる次のプロンプトが表示されます。


```
class> quit
Class scheduler database modified.
Save changes? (yes/no) [yes]:yes

changes saved
```

13.2.2.4.2 クラス内での識別子タイプを管理する

クラス・スケジューラが認識する一意に割り当てた識別子 (PID, GID, または UID など) によって, プロセスがどのクラスのメンバであるかが識別されます。クラスを作成した後, add コマンドを使って, UID と GID またはプロセスを 1 つ以上のクラスに追加することができます。使用する識別子 (ID) のタイプを指定し, 1 つ以上の一意の識別子を入力する必要があります。UID および GID は /etc/passwd および /etc/group ファイルから判断できます。また, 代わりに, アカウント・マネージャ (dxaccounts) というグラフィカル・インタフェースを使用して, UID およびグループ情報を表示できます。

プロセス識別子は, システム・ファイルから, または ps などのコマンドによって取得することができます。ps コマンドにより, PID, PGID, および SESS の値が判断できます。次のコマンドを使用して, システムで実行中のプロセス全部の PID を表示できます。

```
# /sbin/ps aj

USER  PID PPID  PGID  SESS JOBC S   TTY          TIME COMMAND
walt  5176 5162  5176  2908  1  S   ttypl      0:01.30 -sh (csh)
root 12603 5176 12603  2908  1  R   + ttypl      0:00.05 ps aj
```

このコマンドの詳細と, コマンド使用時に表示されるプロセス・データ項目の最終的な一覧については, ps(1) を参照してください。

次の識別子がサポートされています。

gid

/etc/group ファイルにあるグループ識別子番号。たとえば, あるクラスにメンバを追加する場合, この番号を使用すると, そのグループに割り当てられているすべてのユーザが追加されます。

uid

/etc/passwd ファイルにあるユーザ識別番号。たとえば, あるクラスにメンバを追加する際に, この番号により, その UID が割り当てられている特定のユーザだけが追加されます。

pgrp

プロセス・グループ識別子。ps aj コマンドで表示される上記テーブル例の PGID という見出しにあるエントリを参照してください。

session

セッション識別子。ps aj コマンドで表示される上記テーブル例の SESS という見出しにあるエントリを参照してください。

pid

プロセス識別子。ps aj コマンドで表示される上記テーブル例の PID という見出しにあるエントリを参照してください。

uid および gid の 2 タイプの識別子は、リブートまたはクラス・スケジューリングを停止、再起動した後も続けて使用できるので、最も頻繁に使用することになるでしょう。dxaccounts のようなアカウント管理ツール、または SysMan Menu の [アカウント] オプションを使用して、ユーザおよびグループの UID および GID をリストすることができます。タイプが pgrp, session, および pid の識別子は一時的な識別子で、リブートやプロセス終了後には失われます。

13.2.2.4.3 クラス・スケジューラを有効にする

クラス・スケジューラ・デーモンを有効にするには、次のコマンドを実行します。

```
# class_admin enable
Class scheduling enabled and daemon \
/usr/sbin/class_daemon started.
```

デーモンを無効にするには、次のコマンドを実行します。

```
# class_admin disable
Class scheduling disabled.
```

13.2.2.4.4 クラスメンバーを追加する

プロセスをクラスに追加するには、次の対話式モードの例に示すように、add コマンドを入力します。

```
class> add batch_jobs uid 234 457 235
```

前に指定した一意の識別子のいずれかを使用しなければならず、また、同じ識別子は1つのクラスで2回以上追加できません。次に示すように、同じ手順をコマンド・モードまたはスクリプトから実行することもできます。

```
# class_admin add batch_jobs uid 234 457 235
uid 234 457 235 added to high_users
```

コマンド・モードでは、クラスに追加した内容は、自動的に `/etc/class` データベースに保存されます。

13.2.2.4.5 クラスからメンバを削除する

クラスから1つ以上のプロセスを削除するには、対話式モードまたはコマンド・モードで `delete` コマンドを入力します。次に例を示します。

```
class> delete high_users uid 11
uid 11 deleted from high_users
```

この例では、`high_users` クラスから `11` という1つのUIDを削除します。

13.2.2.4.6 その他のクラス管理オプション

次のオプションについての詳細は、`class_admin(8)` を参照してください。

- クラスの優先順位の変更。次に例を示します。

```
class> change batch_jobs 20
batch_jobs retargeted at 20%
```

- 空のまたは空でないクラス全体の破壊。次に例を示します。

```
class> destroy high_users
high_users is not empty.
to destroy anyway? [yes/no]:yes
high_users destroyed
```

- スケジューリング・データベースのロードと保存。次に例を示します。

```
class> load database_performance
current database modified and not saved
load new database anyway (destroys changes)? (yes/no) [yes]: \
yes
database database_performance loaded
```

この例では、現在のデータベースに対する保存されていない変更があることが検出され、ユーザに変更内容を保存するかどうかを尋ねるプロンプトが表示されます。

- 現在の優先順位の設定に対する実際のCPU使用統計情報の表示。次に例を示します。

```
class> stats
Class scheduler status: enabled
```

class name	target percentage	actual percentage
high_users	50%	40.0%
batch_jobs	10%	2.0%

13.2.2.5 runclass コマンドの使用

スケジューラ・クラスを作成し、クラス・スケジューリングを有効にした後、`runclass` コマンドを使用して特定のクラスでコマンドを実行することができます。root ユーザ (スーパーユーザ) として `runclass` コマンドを使うと、プロセスが既存クラスに割り当てられていても、CPU リソースへ無制限にアクセスすることができます。省略時の設定では、ルート・プロセスは制限されていません。このため、ユーザ・プロセスはルート・アカウントで必要とされるリソースをロックすることはできません。クラス・スケジューラの構成をテストする必要がある場合は、非特権ユーザ・アカウントを使ってログインし、プロセスを作成する必要があります。ダミーのユーザ・アカウントを設定してこのようなテストを実行する方法も考えられます。

次のコマンドは、`runclass` コマンドを使用して、端末ウィンドウを開き、それをあらかじめ作成した `high_users` に割り当てます。

```
# runclass high_users xterm
```

次のコマンドは、端末プロセスの `pgrp` 番号が現在そのクラスのメンバとして識別されていることを示します。

```
# class_admin show
.
.
.
class members:
pgrp 24330      pgrp 24351      pgrp 24373
```

この例では、`xterm` プロセスの識別子がクラスに追加されます。次のコマンドを使用して、プロセスを表示することができます。

```
#ps agx | grep xterm
```

詳細については、`runclass(1)` を参照してください。

13.2.2.6 クラス・スケジューリング・グラフィカル・インタフェースの使用

クラス・スケジューラは、SysMan Menu で [モニタリング/チューニング] タスクから [クラス・スケジューリング] オプションを選択することによって起動することができます。また、共通デスクトップ環境 (CDE) の「アプリケーション・マネージャ」を使って起動することもできます。

前の項で説明した、クラス・スケジューラをコマンド行で使用方法では、初期構成で次の手順を実施します。

1. クラスとクラスに割り当てるプロセス、ユーザ、またはグループを計画します。
2. クラスを作成しデータベースに追加することにより、データベースを構成し名前をつけます。
3. 新しいデータベースを現在のデータベースとして定義します。
4. クラス・スケジューリング・デーモンを開始します。

この手順を SysMan Menu の [クラス・スケジューリング] メイン・メニュー・オプションを使って行うことができます。このオプションには、次の3つのサブオプションがあります。

クラス・スケジューラの設定

クラス・スケジューリングを構成し、初期化する中心的なオプションです。このオプションを選択すると、「クラス・スケジューラの設定: *hostname*」というウィンドウが表示されます。ここから、以下のオプションのいずれかを選択します。

- カレントにする... — このオプションを使って、既存のデータベースを選択し、現在のデータベースにします。システムを最初に使う場合は、オプション・リストから省略時のデータベースしか選択できません。このデータベースは、ブレース・ホルダであり、クラスは含まれません。この省略時設定を修正するか、または新しいデータベースを作成してリストに追加します。
- 新規... — このオプションを使って、新しいデータベースを作成し、オプションのデータベースのリストに追加します。データ・エントリのウィンドウが表示されるので、データベースに名前をつけて、クラスを選択するか新しく作成します。
- コピー... — このオプションを使って、既存のデータベースをファイルにコピーし、新しいデータベースの下敷きにすることができます。このコピーに付けるファイル名と保存場所をたずねるプロンプトが表示されるので入力します。

- 修正... — このオプションを使って、既存のデータベースの構成を変更します。変更する前に元のデータをとっておきたい場合は、最初に [コピー...] オプションを使用する必要があります。
- 削除 — このオプションを使って、オプション・リストからデータベースを削除します。一度削除したデータベースは、復元することができません。

[新規...] オプションは、主要なオプションであり、最も頻繁に使います。詳細は、13.2.2.7 項で説明しています。[修正...] オプションも、これと同じインタフェースで、既存のクラスとデータベースを変更することができます。

残りのメニュー・オプションは、確認が必要ですが、多くのデータを入力することはありません。たとえば、データベースを削除しようとすると、データベースを削除してよいかどうかを確認してくるだけです。

クラス・スケジューラの起動/再起動

このオプションを使って、クラス・スケジューリング・デーモンを開始します。デーモンが停止した場合は、再起動します。選択が適切かどうかを確認してきます。

クラス・スケジューラの停止

このオプションを使って、クラス・スケジューリング・デーモンを停止します。選択が適切かどうかを確認してきます。

SysMan Menu を実行する方法は、『システム管理ガイド』を参照してください。13.2.2.6 項では、グラフィカル・インタフェースの使用方法を説明しています。正しいデータ・エントリについての補足情報は、オンライン・ヘルプを参照してください。

13.2.2.7 データベースの作成と変更

[新規...] または [修正...] オプションを選択すると、「クラス・スケジューラの設定: クラス・スケジューラ・データベースの作成/修正」というタイトルの画面が表示されます。この画面で、次の手順に従って新しいデータベースを作成します。

1. 「名前:」フィールドに、作成したいデータベースの名前を入力します。名前は、たとえば、`served_applications` のようにデータベースの

役割を反映するものにし、オプションのリストに表示されたときにすぐ分かるようにします。

2. [有効なスケジューリング・クラス] というオプション・リストから既存のクラスを選択します。初めてデータベースを設定する場合は、クラスが表示されず、[新規...] オプションを選択できるだけです。
3. 新しいクラスを作成するために、[新規...] ボタンを押し「新規ベース・クラスの作成」というタイトルのウィンドウを表示します。このウィンドウで、以下のステップを実行します。
 - 「クラス名」フィールドにクラスの名前を入力します。クラス名は、たとえば `principal_users` のようにクラスのメンバがすぐ分かるものにします。
 - 対象クラスに割り当てる CPU 時間のパーセント値に対応する CPU 割り当てラベルの隣までスライダ・バーを動かして、値を調節します。
 - 「メンバ・タイプ」フィールドのプルダウン・メニューから、プロセスをこのクラスに割り当てる際に用いる識別子のタイプを選択します。リブートの後は、グループ ID とユーザ ID だけが保持され、セッション ID、プロセス・グループ ID、およびプロセス ID は保持されません。
 - 「メンバ」フィールドに、`/etc/passwd` ファイルに登録されているユーザ名、`/etc/group` ファイルに登録されているグループ名、または次のコマンドを実行して得られるプロセス ID を入力します。

```
# /sbin/ps aj
```
 - [了解] ボタンを選択して、クラスの入力を終了し、前のウィンドウに戻ります。または、[適用] ボタンを選択してそのエントリを終了し、同じウィンドウで別のクラスを作成します。クラスの作成を途中でやめたい場合は、[取消] を選択してください。
4. 作成したクラスは、「有効なスケジューリング・クラス」のオプション・リストに表示されます。クラス名の他に、CPU 時間の割り当てパーセントとメンバのタイプも表示されます。ここまでくると、次のようにして、データベースに追加するクラスを選択することができます。
 - クラスを選択し、ハイライトさせます。
 - [選択] ボタンを押し、そのクラスをデータベースに追加します。

5. 必要なクラスをすべて選択したら、[了解] ボタンを押して新しいデータベースを作成します。新しいデータベースは、「有効なスケジューリング・データベース」のリストに追加されます。

また、「クラス・スケジューラの設定: スケジューリング・データベースの作成/修正」ウィンドウを使って、次のようにクラスの保守と管理を行うことができます。

- [コピー...] オプションを使ってクラスをコピーし、新しいクラスのベースとして使用する。
- [修正...] オプションを使って、クラスの特性を変更する。
- [削除...] オプションを使って、クラスを破壊し、「有効なスケジューリング・クラス」から永久に削除する。

新しく作成したデータベースを使用し始めるには、次の手順を実行します。

1. 「クラス・スケジューラの設定 *hostname*」というタイトルのウィンドウが表示されていない場合は、SysMan Menu を開始し、[クラス・スケジューラの設定] オプションを選択します。
2. 必要なデータベースをクリックしてハイライトさせます。次に、[カレントにする...] ボタンを選択します。選択を確認するか取り消すかをたずねてきます。
3. [了解] ボタンを押して SysMan Menu の [クラス・スケジューリング] オプションに戻ります。[クラス・スケジューラの起動/再起動] というタイトルのオプションを選択します。選択を確認するかどうかをたずねてきます。

これらの手順を終了すると、クラス・スケジューリング・デーモンが開始され、指定したスケジューリング・データベースが使われます。データベースが予想どおりに動作するかどうかを検証し監視するには、端末のコマンド行で `show` コマンドを実行します。たとえば、スケジューリングの統計を表示するには、次のコマンドを入力します。

```
# class_admin stats

Class scheduler status: enabled \
current database: /etc/.cl_lab1

class name      target percentage  actual percentage
prio-tasks-lab      10              10
```


タスクとシステム性能をモニタするためには、ある程度の時間が必要ですし、各クラスを調整し、望む結果が得られるようにする必要もあるでしょう。

13.2.3 ジョブの優先順位を設定する

ジョブに優先順位を設定して、重要なアプリケーションが先に実行されるようにすることができます。 `nice` コマンドを使用すると、コマンドに優先順位を指定できます。 `renice` コマンドを使用すると、実行中のプロセスの優先順位を変更できます。

詳細は、`nice(1)` および `renice(8)` を参照してください。

13.2.4 ジョブをオフピークの時間帯にスケジューリングする

ジョブをスケジューリングして、オフピークの時間帯に実行されるようにしたり (`at` および `cron` コマンドを使用)、負荷レベルが低下した時点で実行されるようにする (`batch` コマンドを使用) ことができます。これにより、CPU とメモリおよびディスク入出力サブシステムの負荷を軽減できます。

詳細は、`at(1)` および `cron(8)` を参照してください。

13.2.5 advfsd デーモンを停止する

`advfsd` デーモンを使用すると、SNMP (Simple Network Management Protocol) のクライアント (Netview など) が AdvFS ファイル・システム情報を要求できます。AdvFS のグラフィカル・ユーザ・インタフェース (GUI) を使用していない場合は、`advfsd` デーモンを停止することで、CPU リソースを解放し、`advfsd` デーモンが定期的にディスクをスキャンしないようにすることができます。

`advfsd` デーモンがブート時に起動されないようにするには、`/sbin/rc3.d/S53advfsd` の名前を `/sbin/rc3.d/T53advfsd` に変更します。

デーモンを直ちに停止するには、次のコマンドを実行します。

```
# /sbin/init.d/advfsd stop
```

13.2.6 ハードウェア RAID を使って CPU の入出力オーバーヘッドを軽減する

RAID コントローラを使用すると、ディスク入出力の性能を改善する多くの機能を利用だけでなく、CPU のディスク入出力オーバーヘッドも軽減できます。ハードウェア RAID についての詳細は、9.4 節を参照してください。

用語集

この用語集では、Tru64 UNIX の性能、可用性、およびチューニングの説明に使用する用語をリストしています。

AL_PA

AL_PA (Arbitrated Loop Physical Address) は、ファイバ・チャネル・ループ上のノードをアドレス指定するために使用される。ノードは、データを送信する準備が整うと、自身を識別する AL_PA を含むファイバ・チャネル基本信号を送信する。

BMT

ビットファイル・メタデータ・テーブル (BMT) (*bitfile metadata table*) を参照

Compaq Analyze

エラー・イベントの分析と変換を行う診断ツール。

eager モード (eager mode)

即時モード (*immediate mode*) を参照

E_Port

2 つのスイッチ間の通信に使用される拡張ポート。

Fast SCSI

入出力デバイスが同期モードで高いピーク転送速度を達成できるようにしたもの。

Fast10

Fast SCSI を参照

Fast20

UltraSCSI を参照

FC-AL

調停ループ (*arbitrated loop*) を参照

F_Port

ファブリック内のポート (ファブリック・ポート) のことを F_port と呼ぶ。それぞれの F_port には、製造時に 64 ビットの一意のノード名と 64 ビッ

トの一意のポート名が割り当てられる。 ノード名とポート名を合わせると、ワールド・ワイド名になる。

FL_Port

ループ機能を備えた F_Port のことを FL_Port と呼ぶ。

IPC

プロセス間通信 (*interprocess communication*) を参照

lazy キュー (lazy queue)

非同期書き込み要求がキャッシュされる論理的な一連のキュー。

lazy モード (lazy mode)

延期モード (*deferred mode*) を参照

namei キャッシュ (namei cache)

仮想ファイル・システム (VFS) が、最近アクセスしたファイル名とそれに対応する vnode をキャッシュする場所。

NetRAIN

この名前の由来は Redundant Array of Independent Network Adaptors インタフェースであり、ある種のネットワーク接続障害から保護するメカニズムを提供する。

N_Port

それぞれのノードは、データを送受信するためのファイバ・チャネル・ポートを少なくとも 1 つ備えていなければならない。このノード・ポートのことを N_Port と呼ぶ。それぞれのポートには、製造時に 64 ビットの一意のポート名 (ワールド・ワイド名) が割り当てられる。ポイント・ツー・ポイント・トポロジでは、N_Port は他の N_Port に直接接続される。ファブリック・トポロジでは、N_Port は F_Port に接続される。

NL_Port

調停ループ・トポロジでは、情報はループを回るように伝送される。ループ上で動作させることができるノード・ポートのことを NL_Port (ノード・ループ・ポート) と呼ぶ。情報は、デスティネーションに届くまで、それぞれの NL_Port によって複写される。それぞれのポートは、製造時にノードに組み込まれた 64 ビットの一意のポート名 (ワールド・ワイド名) を持っている。

PAL (Privileged Architecture Library)

CPU キャッシュ，2 次キャッシュ，3 次キャッシュ，および物理メモリの間のアドレスとデータの移動を制御する。この移動は，オペレーティング・システムからは見えない。

RAID

RAID (redundant array of independent disks) 技術は，高いディスク入出力性能とデータの可用性を提供する。Tru64 UNIX オペレーティング・システムは，複数のディスクとソフトウェア LSM (Logical Storage Manager) を使用して RAID 機能を提供する。ハードウェア・ベースの RAID 機能は，インテリジェント・コントローラ，キャッシュ，ディスク，およびソフトウェアによって実現される。

RAID0

RAID0 機能は，ディスク・ストライピングとも呼ばれ，データをブロックに分割し，そのブロックをアレイ内の複数のディスクに分散する。ディスク入出力の負荷を複数のディスクやコントローラに分散することによって，ディスク入出力の性能を改善する。ただし，1 つのディスクに障害が発生するとディスク・アレイ全体が利用不能になるため，ストライピングを使用すると可用性が低下する。

RAID1

RAID1 機能は，ミラーリングとも呼ばれ，アレイ内の別のディスク上にデータのコピーを保持する。データを二重化することによって，データの高可用性が実現される。RAID1 では，データを 2 箇所から読み取ることができるため，ディスク読み取りの性能が向上する。ただし，RAID1 ではデータを 2 回書き込まなければならないため，ディスク書き込みの性能は低下する。 n 個のディスクのミラーリングには， $2n$ 個のディスクが必要。

RAID3

RAID3 機能は，データをブロックに分割して，そのデータをディスク・アレイ内に分散 (ストライプ) し，データへの並列アクセスを可能にする。RAID3 では，データの可用性も向上する。ディスクに障害が発生した場合には，別のディスクに格納された冗長パリティ情報を使用して，データが再生される。RAID3 では，パリティ情報用に余分なディスクが必要である。RAID3 では帯域幅が改善されるが，スループットは改善されない。RAID3 では，大量のシーケンシャル・データを転送するアプリケーションの入出力性能が改善される。

RAID5

RAID5 機能は、アレイ内のディスクにデータ・ブロックを分散する。冗長パリティ情報が複数のディスクに分散されるため、各アレイ・メンバは、ディスク障害が発生したときにデータを再生するために使用する情報を保持している。RAID5 では独立したデータ・アクセスが可能で、入出力操作を同時に処理できる。RAID5 では、大規模なファイル入出力操作、複数の小規模データの転送、および読み取り操作のデータ可用性が改善され、性能が向上する。RAID5 は、書き込みを多用するアプリケーションには適していない。

raw 入出力 (raw I/O)

ファイル・システムを使用しないディスクやディスク・パーティションへの入出力。raw 入出力ではバッファとキャッシュがバイパスされるため、ファイル・システムを使用する入出力よりも性能が改善される。

SCSI

SCSI (Small Computer System Interface) は、デバイスおよび相互接続の技術。

SCSI バス速度 (SCSI bus speed)

帯域幅 (*bandwidth*) を参照

SMP

シンメトリック・マルチプロセッシング (SMP) は、同じオペレーティング・システムを実行して、共有メモリにアクセスし、命令を同時に実行できる、マルチプロセッサ・システムの機能。

UBC

ユニファイド・バッファ・キャッシュ (*Unified Buffer Cache*) を参照

UBC LRU

UBC LRU (Unified Buffer Cache least-recently used) ページは、UBC で使用されているページの中で最も古いページ。

UltraSCSI

SCSI-2 構成の 2 倍の性能を実現する、デバイス (アダプタまたはコントローラ) およびディスクのストレージ構成。UltraSCSI (Fast-20 と呼ばれる) では、帯域幅とスループットが改善され、ケーブル長を延長できる。

VLDB

大規模データベース (VLDB: very-large database) システムのこと。大規模で複雑なストレージ構成を使用する VLM システムである。代表的な VLM/VLDB システム構成を、次に示す。

- 複数の高速 CPU を持つ SMP システム
- 4 GB を超える物理メモリ
- 複数の高性能ホスト・バス・アダプタ
- 高性能、高可用性を実現するための RAID ストレージ構成

VLM

大規模メモリ (VLM: very-large memory) システムのこと。64 ビット・アーキテクチャ、マルチプロセッシング、および 2 GB 以上のメモリを使用する。

vnode

オープン・ファイルに対するカーネル・データ構造。

空きリスト (free list)

クリーンな状態で、使用されていないページ (空きリストのサイズによって、ページ再生の発生時期が決まる)。

アクティブ・リスト (active list)

仮想メモリ・サブシステムや UBC が使用しているページ。

アダプティブ RAID 3/5 (adaptive RAID 3/5)

動的パリティ *RAID (dynamic parity RAID)* を参照

エクステンツ (extent)

AdvFS がファイルに割り当てる、ディスク・スペースの連続領域。

延期モード (deferred mode)

スワップ領域割り当てモードの一種。このモードでは、変更された仮想ページをシステムがスワップ領域に書き込む必要が発生するまで、スワップ領域が予約されない。延期モードは、lazy モードとも呼ばれる。

カーネル変数 (kernel variable)

カーネルおよびサブシステムの動作や性能を決定する変数。カーネル変数のアクセスには、システム属性やパラメータが使用される。

回転待ち時間 (rotational latency)

ディスクが特定のディスク・セクタまで回転するための時間 (ミリ秒)。

書き込み時コピー・ページ・フォールト (copy-on-write page fault)

読み取り専用仮想ページをプロセスが変更しようとしたときに発生するページ・フォールト。

仮想アドレス空間 (virtual address space)

アプリケーションが物理メモリにマッピングできるページの配列。仮想アドレス空間は、可変メモリ (スタック、ヒープ、`malloc` 用のメモリ) とファイル・バック・メモリ (プログラム・テキストまたはシェアード・ライブラリ用のメモリ) に使用される。

仮想メモリ・サブシステム (virtual memory subsystem)

物理メモリの一部、ディスク・スワップ領域、およびデーモンとアルゴリズムを使用して、プロセスと UBC へのメモリ割り当てを制御するサブシステム。

可変メモリ (anonymous memory)

スタック、ヒープ、または `malloc` に使用される変更可能なメモリ。

キャッシュ (cache)

データを一時的に保持する場所。待ち時間を短縮して性能を改善するために使用される。CPU キャッシュと 2 次キャッシュは、物理アドレスを保持する。ディスク・トラック・キャッシュとライトバック・キャッシュは、ディスク・データを保持する。キャッシュには、揮発性のもの (ディスク・データやバッテリーによるバックアップがない) と不揮発性のものがある。

キャッシュ・ヒット (cache hit)

キャッシュ内にデータが見つかること。

キャッシュ・ヒット率 (cache hit rate)

キャッシュの有効度を示す指標。

キャッシュ・ミス (cache miss)

キャッシュ内にデータが見つからなかったこと。

クラスタ (cluster)

高可用性を実現するために、データを共用するサーバのグループ (クラスタ・メンバ・システム) をゆるやかに結合したもの。一部のクラスタ製品は、高速で信頼できる通信のために、高性能インタコネクトを使用する。

経路 (route)

パケットがあるシステムから別のシステムへネットワークを通過する際にたどる道筋。これによって、別のネットワーク上の他のシステムと通信することができる。経路は、各システムのルーティング・テーブルまたはルーティング・データベースに保存される。

高可用性 (high availability)

ハードウェアやソフトウェアのリソースの障害に耐える能力。高可用性は、リソースを多重化することによって単一故障点を取り除くことにより実現される。可用性は、リソースの信頼性の面からも測られる。すべてのリソースには、保護できる障害の数に限界がある。

構成 (configuration)

システムやクラスタを形成するハードウェアやソフトウェアの集まり。たとえば、CPU、メモリ・ボード、オペレーティング・システム、およびミラー・ディスクは、構成の一部である。

構成する (configure)

ハードウェアやソフトウェアの構成を設定または変更すること。たとえば、入出力サブシステムの構成には、SCSI バスの接続や、ミラー・ディスクの設定が含まれる。

固定メモリ (wired memory)

固定され、ページングによる再生 (reclamation) ができないメモリ・ページ。

固定リスト (wired list)

固定され、再生できないページ。

作業負荷 (workload)

通常状態の任意の時点で計測した、システム上で動作中のアプリケーションと、システムを使用しているユーザの総数。

シーク時間 (seek time)

ディスクのヘッドが特定のディスク・トラックへ移動するための時間(ミリ秒)。

シーケンシャル・アクセス・パターン (sequential access pattern)

ディスク上の連続したブロックのデータを読み書きするアクセス・パターン。

常駐セット (resident set)

物理アドレスにマッピングされているすべての仮想アドレスのセット (つまり、プロセスの実行中にアクセスされたすべてのページ)。

ショート・ページ・フォールト (short page fault)

要求されたアドレスが仮想メモリ・サブシステムの内部データ構造体内で見つかったときに発生するページ・フォールト。

冗長性 (redundancy)

高可用性を実現するために、リソースを多重化すること。たとえば、別のディスクにデータをミラーリングしたり、パリティ RAID を使用することによって、データに冗長性を持たせることができる。クラスタを設定することによってシステムの冗長性が実現でき、複数のネットワーク・コネクションを使用することによってネットワークの冗長性が実現できる。リソースの冗長レベルを高くするほど、リソースの可用性が高くなる。たとえば、4 つのメンバ・システムで構成されたクラスタは、2 つのシステムで構成されたクラスタよりも冗長レベルが高く、可用性も高くなる。

シリアル SCSI (serial SCSI)

パラレル SCSI の速度、距離、接続性の制限を緩和し、ホット・スワップやフォールト・トレランスなどの高可用性機能も提供する。シリアル SCSI は、次世代の SCSI である。

シンプル・ネーム・サーバ (SNS) (simple name server)

名前、アドレス、および属性を 15 分以下の時間だけ保持し、その情報をファブリック内の他のデバイスに提供するスイッチ・サービス。SNS は、ファイバ・チャネル標準で定義され、既知のアドレスに存在する。ディレクトリ・サービスとも呼ばれる。

信頼性 (reliability)

データが紛失するような故障が起きるまでに、構成要素が動作を続ける平均時間。一般に、データ紛失までの平均時間 (MTDL: mean time to data loss)、平均初期故障時間 (MTTF: mean time to first failure) または平均故障間隔 (MTBF: mean time between failures) として表わされる。

推奨転送サイズ (preferred transfer size)

ディスク入出力での、デバイス・ドライバにとって最も効率の良い転送サイズ。この値は、デバイス・ドライバによって提示される。

スイッチ・ゾーニング (switch zoning)

どのサーバが相互に通信可能であるか、および各ストレージ・コントローラのホスト・ポートと通信可能であるかを制御する。スイッチ・ゾーニングはまた、ストレージ・システムのレベルでアクセスを制御する。

スイッチのカスケード接続 (cascaded switches)

複数のスイッチを相互に接続して、スイッチのネットワークを形成したもの。メッシュ・ファブリック (*meshed fabric*) も参照

スケーラビリティ (scalability)

リソースを追加して期待どおり性能が改善され、性能が大幅に低下することなく作業負荷の増加に対応できる、システムの能力。

スケーラブル (scalable)

システムのハードウェア・リソースを追加することで、期待どおりの性能向上が得られる、システム能力のこと。

ストライピング (striping)

ディスク・アレイ内の複数のディスクにデータを分散すること。これにより並列アクセスが可能となり、入出力の性能が向上する。ストライピングは、RAID 0 とも呼ばれる。ストライピングでは、シーケンシャルなデータ転送の性能と、高帯域幅を必要とする入出力操作の性能が改善される。

スループット (throughput)

入出力サブシステムや構成要素が入出力操作を実行できる速さ。スループットは、小規模な入出力操作を多数実行するアプリケーションで特に重要である。

スワッピング (swapping)

中断状態のプロセスの変更 (ダーティ) ページをスワップ領域に書き込み、クリーン・ページを空きリストに置くこと。スワッピングが発生するのは、空きリスト上のページの数特定のしきい値よりも少なくなったときである。

スワップ・アウト (swap out)

低優先度のプロセスまたは常駐セット・サイズが大きいプロセスの変更ページを、物理メモリからスワップ領域にすべて移動すること。スワップ・アウトが発生するのは、空きページ・リスト上のページの数、特定の数を一定時間下回ったときである。スワップ・アウトは、空きページ・リスト上のページの数、特定の数に達するまで続く。

スワップ・イン (swap in)

プロセスを実行するために、スワップ・アウトされたプロセスのページをディスク・スワップ領域から物理メモリに移動すること。スワップ・インが発生するのは、空きページ・リスト上のページの数、特定の数を一定時間超えたときだけである。

スワップ・デバイス (swap device)

ディスクの一部に構成されたブロック・デバイス。

スワップ領域インタリーピング (swap-space interleaving)

ストライピング (*striping*) を参照

静的固定メモリ (static wired memory)

ブート時に割り当てられ、オペレーティング・システムのデータやテキスト、システム・テーブル用に使用される固定メモリ。静的固定メモリは、最近アクセスされた UNIX ファイル・システム (UFS) および CD-ROM ファイル・システム (CDFS) のメタデータを保持するメタデータ・バッファ・キャッシュとしても使用される。

セレクトティブ・ストレージ・プレゼンテーション (SSP) (selective storage presentation)

どのサーバが各ストレージ・ユニットにアクセスできるかを制御する。SSP はまた、ストレージ・ユニットのレベルでアクセスを制御する。

ゼロフィル・オンデマンド・ページ・フォールト (zero-filled-on-demand page fault)

要求されたアドレスが初めてアクセスされたアドレスだったときに発生するページ・フォールト。

ゾーニング (zoning)

ゾーニングにより、リソースを分割して、管理やアクセス制御を行うことができる。ゾーニングによって、1 台のスイッチで複数のクラスタ、さらには複数のオペレーティング・システムに対してサービスを行うようにすることで、ハードウェア・リソースを有効に活用することができる。ゾーニングでは、ファブリックが複数のゾーンに分割されるので、各ゾーンは実質的に仮想ファブリックになる。

ゾーン (zone)

ファブリックに接続されたファイバ・チャネル・デバイスの論理的なサブセット。

即時モード (immediate mode)

スワップ領域の割り当てモードの一種。このモードでは、変更可能な仮想アドレス空間の作成時にスワップ領域が予約される。即時モードは省略時のスワップ領域割り当てモードであり、eager モードとも呼ばれる。

属性 (attributes)

動的に構成可能なカーネル変数。この値は、システムの性能を改善するために変更できる。新しい属性値は、カーネルを再構築しなくても有効になる。

ソフトウェア RAID (software RAID)

ソフトウェア (LSM など) によって RAID 機能を実現するストレージ・サブシステム。

ソフト・ゾーニング (soft zoning)

シンプル・ネーム・サーバ (SNS) に基づいてゾーニングが強制される、ソフトウェアによる方式。ソフト・ゾーニングは、すべてのホストがソフト・ゾーニングに対応している場合に使用でき、ソフト・ゾーニングを許可するように設定されていないホストがあるときには使用できない。

帯域幅 (bandwidth)

入出力サブシステムや構成要素がデータ・バイトを転送できる速度。帯域幅は、大量のシーケンシャル転送を行うアプリケーションで特に重要である。転送速度 (*transfer rate*) も参照

遅延 (delay)

待ち時間 (*latency*) を参照

チューニング (tune)

カーネル変数の値を変更してカーネルを修正し、システムの性能を改善すること。

調停ループ (arbitrated loop)

ループ内のノード間のリンクによって形成されたループに沿ってフレームが伝送されるファイバ・チャネル・トポロジ。ループ内のすべてのノードが帯域幅を共有するため、ノードとケーブルを追加するに従って帯域幅がわずかに減少する。

データ・パス (data path)

これによって、バスの実際の帯域幅が決まる。

ディスク・アクセス時間 (disk access time)

ミリ秒単位で計測した、シーク時間と回転待ち時間の合計。アクセス時間の短縮は、小規模な入出力操作を多数行うアプリケーションで特に重要である。回転待ち時間 (*rotational latency*)、シーク時間 (*seek time*) も参照

ディスク・クォータ (disk quotas)

これを使用することによって、システム管理者は、各ユーザが使用できるディスク・スペースを制限でき、ディスク・スペースの使用量を監視できる。

ディスク・パーティション (disk partitions)

ディスク・パーティションは、ディスクを論理的に分割したものであり、ファイルをさまざまなサイズの独立した領域に体系的に配置することを可能にする。パーティションには、ファイル・システムと呼ばれる構造でデータが格納され、またページングやスワッピングなどのシステム動作に使用することもできる。

転送速度 (transfer rate)

帯域幅 (*bandwidth*) を参照

転送方式 (transmission method)

SCSI バス仕様の電氣的な実装方式。

動的固定メモリ (dynamically wired memory)

システム・ハッシュ・テーブルなどの、動的に割り当てられるデータ構造体に使用される固定メモリ。ユーザ・プロセスも、`mlock` 関数などの仮想メモリ・ロック・インタフェースを使用して、アドレス空間に動的固定メモリを割り当てる。

動的パリティ RAID (dynamic parity RAID)

アダプティブ RAID3/5 とも呼ばれる。動的パリティ RAID は RAID3 と RAID5 の特徴を組み合わせ、広範囲のアプリケーションのディスク入出力の性能と可用性を改善する。アダプティブ RAID3/5 は、必要な作業負荷に応じて、データ転送を多用するアルゴリズムと入出力操作を多用するアルゴリズムの間で動的に調整を行う。

ネットワーク・アダプタ (network adapter)

ネットワーク・インタフェース・カード (NIC) (*network interface card*) を参照

ネットワーク・インタフェース (network interface)

ネットワーク・インタフェース・カード (NIC) (*network interface card*) を参照

ネットワーク・インタフェース・カード (NIC) (network interface card)

ネットワークに物理的に接続するために使用する回路基板。NIC は、ネットワーク・アダプタまたはネットワーク・インタフェースとも呼ばれる。

ノード (node)

フレームの送信元と送信先のこと。ノードには、コンピュータ・システム、RAID (redundant array of independent disks) アレイ・コントローラ、ディスク・デバイスなどがある。それぞれのノードは、製造時にノード内に組み込まれる 64 ビットの一意のノード名 (ワールド・ワイド名) を持っている。

能力 (容量) (capacity)

システム・リソースの理論上の最大スループット、またはディスクに保持できる最大データ量 (バイト)。能力の上限に達したリソースは、ボトルネックとなって性能を低下させることがある。

ハードウェア RAID (hardware RAID)

インテリジェント・コントローラ、キャッシュ、およびソフトウェアによって RAID 機能を提供するストレージ・サブシステム。

ハード・ゾーニング (hard zoning)

ファイバ・チャネル・フレームをハード的にブロックすることにより、すべてのファブリック・スイッチにおいて物理レベルでゾーンが強制される。

バス・エクステンダ (bus extenders)

バス・エクステンダは、UltraSCSI 技術で、システムとストレージ間の接続距離を延長するために使用される。

バス・セグメント (*bus segments*) も参照

バス・セグメント (bus segments)

バス・セグメントは、UltraSCSI 技術で、システムとストレージ間の接続距離を延長するために使用される。

バス・エクステンダ (*bus extenders*) も参照

パラメータ (parameters)

静的に構成可能なカーネル変数。この値は、システムの性能を改善するために変更できる。新しいパラメータ値を有効にするには、カーネルを再構築しなければならない。多くのパラメータには、対応する属性がある。

パラレル SCSI (parallel SCSI)

種類が豊富で、さまざまな性能および構成が選択できる、最も一般的なタイプの SCSI。

パリティ RAID (parity RAID)

RAID 機能の一種。データの再生に使用する冗長情報を別のディスクに格納するか複数のディスクに格納して、データの可用性を高くする。パリティ RAID は、RAID3 の一種としても知られている。

非アクティブ・ページ (inactive pages)

プロセスが使用しているページの中で最も古いもの。

ビットファイル・メタデータ・テーブル (BMT) (bitfile metadata table)

ビットファイル・メタデータ・テーブルは、ボリューム上のファイルの拡張を記述する。

ファイル・バック・メモリ (file-backed memory)

プログラム・テキストやシェアード・ライブラリ用に使用されるメモリ。

ファブリック (fabric)

単独のスイッチ、または相互に接続された数台のスイッチで構成され、送信元のノード (送信側) とデスティネーション・ノード (受信側) の間でフレームをルーティングする。

フェイルオーバー (fail over / failover)

ハードウェアやソフトウェアの障害後に冗長リソースを自動的に使用して、リソースを利用可能な状態に保つこと。たとえば、クラスタのメンバ・システムの 1 つに障害が発生すると、そのシステム上で動作していたアプリケーションは、自動的に他のメンバ・システムにフェイルオーバーされる。

物理メモリ (physical memory)

システムに実装されているメモリ・ボードの総容量。物理メモリは、固定メモリとして使用されるか、プロセスと UBC で共用される。

フラッシュ・キュー (flush queue)

読み取り要求と同期書き込み要求がキャッシュされるキュー。フラッシュ・キューは、主に書き込み要求や同期書き込みをバッファリングするために使用される。

ブロック・キュー (blocking queue) も参照

フレーム (frame)

すべてのデータは、フレームと呼ばれる情報パケットの単位で転送される。フレームは、2112 バイトに制限されている。情報が 2112 バイト以上からなる場合、情報は、複数のフレームに分割される。

プロセス間通信 (interprocess communication)

プロセス間通信 (IPC) とは、2 つ以上のプロセスの間で情報を交換すること。

ブロック・キュー (blocking queue)

読み取り要求と同期書き込み要求がキャッシュされるキュー。ブロック・キューは、主に読み取り要求とカーネル同期書き込み要求用に使用される。フラッシュ・キュー (*flush queue*) も参照

ページ (page)

システムが割り当てることができる物理メモリの最小単位 (8 KB のメモリ)。

ページ・アウト (page out)

変更された (ダーティ) ページの内容を、物理メモリからスワップ領域へ書き込むこと。

ページ・イン (page in)

ディスクから物理メモリへページを移動すること。

ページ・イン・ページ・フォールト (page-in page fault)

要求されたアドレスがスワップ領域内にあるときに発生するページ・フォールト。

ページ・カラリング (page coloring)

プロセスの常駐セット全体を、2 次キャッシュにマッピングしようとする。

ページ・テーブル (page table)

現在の仮想アドレスから物理アドレスへの変換エントリを保持している配列。

ページ・フォールト (page fault)

仮想メモリ・サブシステムへの命令。要求されたページを探し、ページ・テーブル内の仮想アドレスから物理アドレスへの変換情報を整えさせる。

ページング (paging)

プロセスと UBC に割り当てられたページを再使用できるように再生する処理。

ユニファイド・バッファ・キャッシュ (Unified Buffer Cache) も参照

ページング可能メモリ (pageable memory)

固定されていない物理メモリ。

ボトルネック (bottleneck)

ほぼ能力一杯まで使用され、性能低下の原因となっているシステム・リソース。

待ち時間 (latency)

特定の動作が完了するまでの時間。待ち時間は、遅延とも呼ばれる。高性能を実現するには、待ち時間が短くなければならない。入出力待ち時間はミリ秒単位、メモリ待ち時間はマイクロ秒単位で計測される。メモリ待ち時間は、メモリ・バンクの構成およびシステムのメモリ要件によって異なる。

マルチプロセッサ (multiprocessor)

共通の物理メモリを共用する複数のプロセッサ (CPU) を備えたシステム。

ミラーリング (mirroring)

データのコピーを、別のディスク上に保持すること。これにより、データの可用性が高くなり、ディスクの読み取り性能が向上する。ミラーリングは、RAID 1 とも呼ばれる。

メッシュ (mesh)

メッシュ・ファブリック (meshed fabric) を参照

メッシュ・ファブリック (meshed fabric)

スイッチのカスケード接続構成であり、スイッチまでのネットワーク (スイッチも含む) に障害が発生しても、SAN 接続ノードへのデータ・パスが失われないようにすることができる。

ユニファイド・バッファ・キャッシュ (Unified Buffer Cache)

最近アクセスされたファイル・システム・データのキャッシュに使用される物理メモリ。

ランダム・アクセス・パターン (random access pattern)

ディスク上のさまざまな位置にあるブロックのデータを読み書きするアクセス・パターン。

リソース (resource)

ユーザやアプリケーションが利用できるハードウェアやソフトウェア構成要素 (CPU, メモリ, ネットワーク, ディスク・データなど)。

リンク (link)

N_Port 間または N_Port と F_Port の間の物理的な接続。リンクは、情報送信用と情報受信用の 2 つの接続で構成される。あるノードの送信接続は、リンク先のノードから見ると受信接続になる。リンクには、光ファイバ、同軸ケーブル、シールド・ツイスト・ペアがある。

ワーキング・セット (working set)

現在、物理アドレスにマッピングされている仮想アドレスのセット。ワーキング・セットは常駐セットのサブセットであり、プロセスの常駐セットのスナップショットである。

ワールド・ワイド名 (WWN)

IEEE (Institute of Electrical and Electronics Engineers) によってサブシステムに割り当てられ、製造会社によって出荷前に設定される一意の番号。サブシステムに割り当てられるワールド・ワイド名 (WWN) は、決して変わらない。ファイバ・チャネル・デバイスは、ノード名とポート名の両方の WWN (それぞれ 64 ビットの数) を持つ。



A

Advanced File System

(AdvFS を参照)

AdvFS

smooth sync キュー 11-29
UBC の使用 1-29
アクセス構造体のチューニング
 グ 11-10
ウェイト・キュー 11-28
エクステント・マップを表示す
 る 11-25
構成のガイドライン 11-11
コンソール・キュー 11-29
直接入出力を有効にする ... 11-15
デバイス・キュー 11-27
転送サイズ 11-20
データの損失を防ぐ 11-13
データを分散させる 11-16, 11-17
トランザクション・ログを移動す
 る 11-21
入出力キュー 11-27
表示 11-22
表示する ... 11-22, 11-24, 11-25
ファイル・ドメインの断片化を解消
 する 11-19
ファイルの管理 11-11
ファイルをストライピングす
 る 11-17

フラッシュ・キュー 11-27
ブロック・キュー 11-27
ボリュームのバランスをとる 11-16
レイジー・キュー 11-27
レディ・キュー 11-29

AdvfsAccessMaxPercent 属性

アクセス構造体用に予約されている
 メモリの制御 11-10

advfsd デーモン

停止する 13-23

advfsstat コマンド

AdvFS の統計情報を表示す
 る 11-22

AdvfsSyncMmapPages 属性

推奨値 4-11
メモリ・マップされたページを制御
 する 4-11

advscan コマンド

ファイル・ドメインの位置を表示す
 る 11-24

aio_task_max_num 属性

AIO 要求の変更 4-19
推奨値 4-19

anon_rss_enforce 属性

常駐セットのサイズを制限す
 る 12-35

atom ツールキット

アプリケーションのプロファイリン
 グ 2-33

at コマンド
アプリケーションをスケジューリングする 13-23

B

balance コマンド
AdvFS データを分散させる 11-16
batch コマンド
アプリケーションをスケジューリングする 13-23
bio_stats 構造体
表示する 11-37
bufcache 属性
バッファ・キャッシュのサイズを制御する 11-8
buffer_hash_size 属性
テーブル・サイズを制御する 11-8

C

CAM
チューニング 9-15
モニタリング 9-16
cam_ccb_increment 属性
CAM のチューニング 9-16
cam_ccb_low_water 属性
CAM のチューニング 9-15
cam_ccb_pool_size 属性
CAM のチューニング 9-15
chfile コマンド
データ書き込みを防止する. 11-13
データ・ロギングを有効にする 11-13
chvol コマンド
入出力転送サイズを変更する 11-20

class_admin 13-7
管理 13-14
使用 13-12
cluster_maxcontig カーネル変数
ブロックの数を増やす 11-42
collect ユーティリティ
collect データ・ファイルのプロット 2-12
システム情報の収集 2-9
リポート時に自動的に起動... 2-11
Common Access Method 9-15
(CAM も参照)
Compaq Analyze
システム・イベントのモニタリング 2-5
Compaq Analyze (CA)
システム・イベントのモニタリング 2-4
Compaq Continuous Profiling Infrastructure
CPU 時間のモニタリング 6-8
CPI 6-8
(Compaq Continuous Profiling Infrastructure も参照)
CPU
CPI 6-8
(Compaq Continuous Profiling Infrastructure も参照)
ジョブをスケジューリングする 13-23
性能の改善 13-6
内部キャッシュ 1-34
ハードウェア RAID の使用. 13-24
表示する 12-21, 12-25

プロセッサを追加する 13-6
モニタリング 9-4, 13-2, 13-4
リソースを管理する..... 13-7
cpustat 拡張
CPU 統計情報の報告.. 13-2, 13-4
cron コマンド
アプリケーションをスケジューリン
グする 13-23

D

dbx コマンド
UBC 統計情報の表示..... 12-20
UBC の統計情報を表示する 12-29
アプリケーションのデバッグ 2-36
カーネル属性を変更する 5-13
属性のサポート状況の確認.... 3-2
DECEvent ユーティリティ
システム・イベントのモニタリン
グ 2-4, 2-5
defragment コマンド
AdvFS 11-19
delay_wbuffers 属性
クラスタの書き込みを遅らせ
る 11-41
dumpfs コマンド
UFS 情報を表示する 11-35
dxproctuner..... 13-10

E

eager スワップ・モード 4-10

F

fifo_do_adaptive 属性
推奨値 4-12
パイプ処理を変更する 4-12

G

gettimeofday() 関数
性能改善..... 4-2
gh_chunks 属性
共用メモリの予約..... 12-42
推奨値 4-8
変更する..... 4-8
粒度ヒントを使用..... 4-6
gh_fail_if_no_mem 属性
共用メモリの予約..... 12-44
gh_min_seg_size 属性
共用メモリの予約..... 12-44
gprof コマンド
アプリケーションのプロファイリン
グ 2-35

H

hiprof 2-33
(atom ツールキット も参照)
アプリケーションのプロファイリン
グ 2-33
hwmgr ユーティリティ
情報収集..... 2-7
ネットワーク・カード 5-14

I

ifqmaxlen 属性

- 推奨値 6-21
- 送信キューの最大長を大きくする 6-21

inifaddr_hsize 属性

- IP アドレスの検索を速くする 10-15
- 推奨値 6-16
- ハッシュ・パケットの数を増やす 6-16

io_throttle_maxmzthruput 属性

- UFS 入出力のキャッシング 11-38

io_throttle_shift 属性

- UFS 入出力のキャッシング 11-38

iostat コマンド 13-10

- CPU の使用状況の表示 9-4
- ディスクの使用状況の表示 9-4

IPC 8-8

- (System V IPC も参照)

- 表示 12-21
- モニタリング 13-3

ipcs コマンド

- IPC の表示 12-21
- IPC のモニタリング 8-9, 13-3

ipintrq データ構造体

- パケットのドロップをチェックする 10-22

ipport_userreserved_min 属性

- 送信ポートの範囲を変更する 10-11

ipport_userreserved 属性

- 推奨値 4-15, 6-9, 6-19
- 接続ポート数を増やす 6-9, 10-10
- 接続ポートの範囲を変更する 6-19
- 送信接続の回数を変更する ... 4-15

ipqmaxlen 属性

- IP 受信キューの最大長を大きくする 6-20
- 推奨値 6-20
- パケットのドロップを防止する 10-21

ipqs 属性

- IP 受信キューの数を増やす .. 6-20
- IP 入力キューを増やす 10-12
- 推奨値 6-20

i ノード

- 密度を低くする 11-32

K

kdbx コマンド

- カーネルのデバッグ 2-37

kmemreserve_percent 属性

- 推奨値 6-14
- メモリを増やす 6-14, 12-31

kprofile ユーティリティ

- カーネルのプロファイリング 2-35

L

ladebug

- カーネルおよびアプリケーションのデバッグ 2-37

LAG インタフェース

- 可用性の向上 1-28

lazy スワップ・モード 4-10

lockinfo ユーティリティ

- ロック統計情報の収集 2-15

lockstats 拡張

- ロック統計情報を表示する .. 13-2, 13-5

Logical Storage Manager

(LSM を参照)

LSM

RAID サポート..... 9-6, 9-7
機能 9-6
ディスクの管理 9-6
ページ・アウト・レート... 12-42
モニタリング 2-31

lsof コマンド

オープン・ファイルの表示... 2-37

M

malloc 関数

メモリの使用を制御する 7-3

malloc のマッピング

増やす 12-31

max_async_req 属性

RDG テーブル内のセッションの数
を変更する 4-20
推奨値 4-20
パケット用に固定されるページの数
を変更する 4-20

max_objs 属性

RDG 内のオブジェクトの数を変更
する 4-20
推奨値 4-20

max_per_proc_address_space 属性

推奨値 4-18, 6-12
プロセスごとのアドレス・サイズを
変更する 4-18
ユーザ・アドレス空間を大きくす
る 8-7

ユーザ・プロセスのアドレス空間の
限界値を大きくする 6-12

max_per_proc_data_size 属性

最大セグメント・サイズを大きくす
る 8-6
推奨値 4-17, 6-11
データ・サイズを変更する... 4-17
ユーザ・プロセスのデータ・セグメ
ント・サイズの限界値を大きくす
る 6-11

max_per_proc_stack_size 属性

推奨値 4-16
スタック・サイズを変更する 4-16,
8-6

max_proc_per_user 属性

推奨値 4-18, 6-11
プロセス数を変更する 8-3
プロセスの数を変更する 4-18,
6-11

max_threads_per_user 属性

推奨値 4-18, 6-11
スレッド数を変更する . 6-11, 8-4
スレッドの数を変更する 4-18

max_ufs_mounts 属性

UFS マウントの数を増やす 11-34

max_vnodes 属性

オープン・ファイルを増やす 8-14

maxusers 属性

namei キャッシュのサイズを大きく
する 11-2
オープン・ファイルを増やす 8-14
システム・テーブルとデータ構造体
のサイズを大きくする 6-10

システム・テーブルに割り当てる領域を変更する 4-18
システム・リソースを増やす . 8-2
推奨値 4-18, 6-10
MFS 11-33
マウントの限界値 11-34
migrate コマンド
AdvFS データを分散させる 11-16
monitor コマンド
システムのモニタリング 2-31, 13-3
msg_max 属性
メッセージ・サイズを大きくする 8-10
msg_mnb 属性
キュー上のバイト数を制御する 8-10
msg_size 属性
RDG メッセージのサイズを変更する 4-20
推奨値 4-20
msg_tql 属性
メッセージ・キュー・サイズを大きくする 8-11

N

name_cache_hash_size 属性
namei キャッシュの制御 11-2
name_cache_valid_time 属性
namei キャッシュの制御 11-2
NetRAIN
複数のインタフェースの構成 1-26
netstat コマンド 5-7
再送 2-25
再送パケットのチェック ... 10-18

順序の不正なパケット 2-25
ソケット接続 2-23
ソケット・フルのモニタリング 10-20
デバイス・ドライバのエラー 2-22
ドロップしたパケットの表示 12-31
入出力エラーと衝突 2-21
ネットワーク統計情報の調査 . 6-4
ネットワークのモニタリング 2-20
廃棄されたパケットまたは紛失したパケット 2-24
パケットのモニタリング ... 10-22
表示 2-20
不正チェックサム 2-25
プロトコルの統計情報 2-27
メモリの利用率 2-22
ルーティングの統計情報 2-27
new_wire_method 属性
長いシステム時間の解決 4-5
NFS
netstat コマンド 2-20
nfswatch コマンド 2-30
tcpdump ユーティリティ 2-18
UBC の使用 1-30
書き込みの集積 5-5, 5-11
キャッシュ・タイムアウトの限界値 5-6
クライアント・スレッド 5-5
クライアントの問題 5-15
構成 5-4, 5-5
再送 5-6
サーバ側の変更 5-10
サーバ・スレッド 5-4
サーバの応答速度 5-12
サーバの問題 5-7

収集 2-29
 性能上の利点と欠点 5-4
 性能低下の検出 5-3, 5-6
 タイムアウト限界値 5-7
 チューニング・ガイドライン 5-1,
 5-9, 5-16
 ネットワーク・カードの確認 5-14
 表示 2-17, 2-29
 マウント・オプション 5-6
 モニタリング 2-31, 5-2
nfs_*_ticks 属性
 書き込みの集積 5-11
 (**nfs_write_gather** 属性 も参
 照)
nfs_cto 属性
 クライアント間でのファイルの一貫
 性を指定する 5-19
nfs_dnlc 属性
 ディレクトリ名の検索用キャッ
 シュ 5-18
 (**nfs_nnc** 属性 も参照)
nfs_fast_ticks 属性
 サーバが書き込みを遅らせる時
 間 5-13
nfs_nnc 属性
 ネガティブ名キャッシュの検
 索 5-18
nfs_quicker_attr
 ファイル属性のフェッチ 5-20
nfs_slow_ticks 属性
 サーバが書き込みを遅らせる時
 間 5-13
nfs_tcprecspace 属性
 推奨値 5-15
nfs_tcprecvspace 属性
 バッファのサイズを大きくす
 る 5-15
 (**nfs_tcpsendspace** 属性 も参
 照)
nfs_tcpsendspace 属性
 推奨値 5-15
 バッファのサイズを大きくす
 る 5-15
nfs_ufs_lbolt 属性
 書き込みの集積 5-11
 (**nfs_write_gather** 属性 も参
 照)
 カーネルのパラメータを変更す
 る 5-13
nfs_unkn_ticks 属性
 サーバが書き込みを遅らせる時
 間 5-13
nfs_write_gather 属性
 書き込みの集積 5-11
 (**nfs_ufs_lbolt** 属性 も参照)
 サーバが書き込みを遅らせる時
 間 5-11
nfs3_jukebox_delay 属性
 クライアントが再送を開始するまで
 の時間を制御する 5-17
nfs3_maxreadahead 属性
 推奨値 5-17
 読み取り性能を改善する 5-17
 (**nfs3_readahead** 属性 も参照)
nfs3_readahead 属性
 推奨値 5-17
 読み取り性能を改善する 5-17

(**nfs3_maxreadahead** 属性 も参照)

nfsiod デーモン..... 5-4, 5-5

クライアント情報の収集 2-29

クライアントをチューニングする 5-11

nfsstat コマンド

再送の測定..... 5-6

nfsstat ユーティリティ

ネットワークと NFS 統計情報の表示 2-17

nfswatch コマンド

NFS のモニタリング..... 2-31

受信トラフィックのモニタリング 2-30

NFS クライアント

NFS 性能の改善 5-15

NFS サーバ

サーバの性能 5-7

問題 5-7

nice コマンド 13-10

アプリケーションの優先順位を設定する 13-23

システムの負荷の軽減 13-4

O

open_max_hard 属性

オープン・ファイル記述子を制御する 8-15

open_max_soft 属性

オープン・ファイル記述子を制御する 8-15

Oracle

AdvFS 属性の変更 4-11

gettimeofday() 関数 4-2

IPC 通信プロトコルの選択と有効化 4-3

Reliable Datagram 属性の変更 4-19

インターネット属性の変更... 4-14

仮想メモリ属性の変更 . 4-5, 4-12

チューニング 4-1, 4-4

低性能の検出 4-1

プロセス間通信属性の変更... 4-13

プロセス属性の変更..... 4-15

モニタリング 4-1

リアルタイム属性の変更 4-19

P

PAL コード

メモリ管理との関係..... 12-9

path maximum transmission unit

無効にする..... 6-9

per_proc_address_size 属性

推奨値 4-17

プロセスごとのアドレス・サイズを変更する 4-17

per_proc_address_space 属性

ユーザ・アドレス空間を大きくする 8-7

per_proc_data_size 属性

省略時のセグメント・サイズを大きくする..... 8-6

推奨値 4-16

プロセスごとのデータ・サイズを変更する..... 4-16

per_proc_stack_size 属性

推奨値 4-16

スタックの最大サイズを大きくする 8-6

プロセスごとのスタック・サイズを変更する 4-16

Performance Visualizer

クラスタ・イベントのモニタリング 2-31

ping コマンド

リモート・システムの照会... 10-3

pixie プロファイラ 2-33

(atom ツールキット も参照)

アプリケーションのプロファイリング 2-34

PMTU 6-9

(path maximum transmission unit も参照)

pmtu_enabled 属性

PMTU 検出を無効にする 6-9

PMTU の検出を無効にする 10-11

推奨値 6-9

private_cache_percent 属性

キャッシュ・メモリの予約. 12-10

prof コマンド

アプリケーションのプロファイリング 2-34

ps コマンド

CPU 使用状況を表示する .. 12-25

アイドル・スレッドの表示... 2-29

メモリ使用状況を表示する. 12-25

R

rad_gh_regions 属性

推奨値 4-7

変更する 4-7

メモリ・チャンクの変更 4-6

(gh_chunks 属性 も参照)

RAID 1-5

LSM 9-6

製品 1-7

ハードウェア・サブシステム . 9-7

レベル 1-5

RAID レベル..... 1-5

RAID を使用

データの損失を防ぐ..... 11-13

raw 入出力..... 1-4

rm_check_for_ipl

推奨値 4-21

ビットマスクの指定..... 4-21

runclass コマンド 13-8, 13-18

S

sb_max 属性

推奨値 6-23

ソケット・バッファのサイズの指定 6-23

ソケット・バッファのサイズを大きくする..... 10-21

sbcompress_threshold 属性

mbuf クラスタの圧縮を有効にする 6-13, 10-13

推奨値 6-13

sched_stat コーティリティ

CPU 利用率とプロセス統計情報の収集 2-16

screen_cachedepth 属性..... 6-21

- (**screen_cachewidth** 属性 も参照)
- 推奨値 6-22
- スクリーニング・キャッシュのミスを減らす 6-21
- screen_cachewidth** 属性 6-21
- (**screen_cachedepth** 属性 も参照)
- 推奨値 6-22
- スクリーニング・キャッシュのミスを減らす 6-21
- screen_maxpend** 属性
- 推奨値 6-22
- スクリーニング・バッファのドロップを減らす 6-22
- SCSI** 1-8
- (ファイバ・チャネル も参照)
- UltraSCSI バス・セグメントの拡張 1-12
- 転送方式 1-10
- データ・バス 1-8
- バス速度 1-9
- バス・ターミネーション 1-12
- バス長 1-12
- パラレル 1-8
- setrlimit**
- リソース消費の制御 8-1
- setrlimit** システム・コール
- 常駐セットの限界値を設定する 12-35
- shm_max** 属性
- System V 共用メモリ領域の最大サイズを変更する 4-13
- 共用メモリ領域のサイズを大きくする 8-12
- 推奨値 4-13
- shm_min** 属性
- System V 共用メモリ領域の最小サイズを変更する 4-14
- 推奨値 4-14
- shm_mni** 属性
- 一時期に使用できる共用メモリ領域の数を変更する 4-14
- 推奨値 4-14
- shm_seg** 属性
- アタッチできる共用メモリ領域を増やす 8-12
- 一時期に接続できる共用メモリ領域の数を変更する 4-14
- 推奨値 4-14
- showfdmn** ユーティリティ
- ファイル・ドメインとボリ्यूムの統計情報を表示する 11-25
- showfile** ユーティリティ
- AdvFS ファイル情報を表示する 11-25
- showfsets** ユーティリティ
- ファイルセット情報を表示する 11-26
- smooth sync** キュー 11-29
- smoothsync_age** 属性
- UFS 入出力のキャッシング 11-38
- SMP** システム 1-33
- チューニング 6-20
- SO_KEEPAIVE** 属性
- TCP 持続機能を有効にする .. 6-17
- 推奨値 6-18
- sobacklog_drops** 属性
- システムがバケットをドロップした回数をカウントする 6-5

ソケットのモニタリング 10-2,
10-4

sobacklog_hiwat 属性

ソケットのモニタリング 10-2,
10-4

保留中の要求の最大数をカウントする 6-5

somaxconn_drops 属性

システムがパケットをドロップした
回数をカウントする 6-5

ソケットのモニタリング 10-2,
10-4

somaxconn 属性

推奨値 6-12

ソケット・リッスン・キューを大きく
する 10-8

保留中 TCP 接続の最大数を増や
す 6-12

sominconn 属性

推奨値 6-13

ソケット・リッスン・キューを大きく
する 10-8

保留中 TCP 接続の最小数を増や
す 6-13

ssm_threshold 属性

System V の共用領域を変更する 4-13

共用ページ・テーブルを制御する 8-13

共用メモリの割り当て 4-6

共用メモリを無効にする 4-5

推奨値 4-5, 4-13

swapdevice 属性

スワップ領域の指定 12-18

swapon コマンド

スワップ領域の追加 12-18, 12-20

スワップ領域の表示 12-20

スワップ領域を表示する ... 12-28

switchlog コマンド

トランザクション・ログを移動する 11-21

sync

影響を最小限にする 12-14

sys_check ユーティリティ

構成情報の収集 .. 2-13, 6-7, 13-2

sysconfig コマンド

属性のサポート状況の確認 3-1

SysMan Menu

iostat コマンド 13-10

vmstat コマンド 13-10

クラス・スケジューラ 13-10

クラス・スケジューリング.. 13-8,
13-18

System V IPC 8-8

T

tcbhashnum 属性

TCP ハッシュ・テーブルの数を増や
す 6-15

推奨値 6-15

ハッシュ・テーブルの数を増や
す 10-8

tcbhashsize 属性

TCP 検索を高速化する 10-7

TCP ハッシュ・テーブルのサイズを
大きくする 6-8

推奨値 6-8

TCP..... 6-8, 6-12, 6-13
 (Transmission Control Protocol
 も参照)

tcp_keepalive_default 属性
 keepalive を有効にする 10-14

tcp_keepcnt 属性
 keepalive プローブの最大値を指定
 する 10-15
 持続プローブの最大数を指定す
 る 6-18

tcp_keepidle 属性
 アイドル時間の長さを指定す
 る 6-18, 10-15

tcp_keepinit 属性
 TCP タイムアウトの限界値を指定す
 る 10-16
 TCP のタイムアウト限界値を指定す
 る 10-15
 TCP パーシャル接続タイムアウト限
 界値を変更する 6-16
 推奨値 6-16
 接続がタイムアウトになるまでの時
 間を指定する 6-18

tcp_keepintvl 属性
 再送プローブを指定する ... 10-15
 持続プローブの再送間隔を指定す
 る 6-18

tcp_msl 属性
 TCP コンテキストのタイムアウト限
 界値を小さくする 10-17
 TCP 接続コンテキスト・タイムアウ
 トの頻度を高くする 6-18
 推奨値 6-19

tcp_mssdflt 属性

TCP セグメントのサイズを大きくす
 る 10-19

tcp_rexmit_interval_min 属性
 TCP 再送の速度を遅くする .. 6-17
 TCP 再送レートを下げる .. 10-18
 推奨値 6-17

tcpdump ユーティリティ
 情報収集 2-18
 ネットワーク・イベントのモニタリ
 ング 2-31
 ネットワーク・パケットのモニタリ
 ング 10-3
 リモート・システムでのサポートの
 確認 5-15

tcpnodelack 属性
 TCP データの肯定応答を遅延させ
 る 10-19

third..... 2-33
 (atom ツールキット も参照)
 アプリケーションのプロファイリン
 グ 2-33

top コマンド
 システムのモニタリング 2-31,
 13-3

traceroute コマンド
 パケット経路の表示 10-3

Transmission Control Protocol
 (TCP を参照)
 TCP ハッシュ・テーブルのサイズを
 大きくする 6-8

U

UBC 1-32, 12-2
 AdvFS 用 1-29

- NFS 用 1-30
- インターネット・サーバのチューニング 6-23
- 借用しきい値 12-14
- チューニング 11-5
- 表示する 12-29
- 表示ツール 12-20
- ページの表示 12-24
- メモリの割り当て 12-4
- UBC LRU ページ・リスト** 12-3
- ubc_borrowpercent** 属性
 - UBC が借りるメモリ 4-9
 - UBC 借用しきい値の指定 6-23
 - 推奨値 4-9, 6-24
- ubc_maxborrowpercent** 属性
 - UBC のチューニング 11-5
- ubc_maxdirtywrites**
 - UBC のチューニング 11-5
- ubc_maxdirtywrites** 属性
 - 変更ページの事前書き出し 12-13
 - ページの事前書き出し 12-38
- ubc_maxpercent** 属性
 - UBC が利用できるメモリの最大割合の指定 6-23
 - UBC で使用するメモリ 4-8
 - UBC のチューニング 11-5
 - 推奨値 4-8, 6-24
- ubc_minpercent** 属性
 - UBC だけが利用できるメモリの最小割合の指定 6-23
 - UBC のチューニング 11-5
 - 推奨値 6-24
- udp_rcvspace** 属性
 - UDP ソケット・バッファを大きくする 10-20
 - UDP ソケット用受信バッファ・サイズを変更する 4-15
 - 推奨値 4-15
- udp_sendspace** 属性
 - UDP ソケット・バッファを大きくする 10-20
 - UDP ソケット用送信バッファ・サイズを変更する 4-15
 - 推奨値 4-15
- uerf** コマンド
 - メモリの表示 12-21
- UFS**
 - i ノードの密度 11-32
 - smoothsync 11-38
 - クォータ 11-34
 - クラスタとして結合されたブロック 11-33
 - クラスタの書き込みを遅らせる 11-41
 - 構成のガイドライン 11-31
 - 断片化を解消する 11-43
 - チューニングのガイドライン 11-38
 - 表示する 11-35
 - フラグメント・サイズ 11-32, 11-35
 - ブロックを連結する 11-42
 - マウントの限界値 11-34
 - メモリ・ファイル・システム (MFS) 11-33
 - モニタリングする 11-36
 - 連続ブロックの割り当て ... 11-33

ufs_clusterstats 構造体
 UFS クラスタの統計情報を表示する 11-36

ufs_getapage_stats 構造体.. 12-20
 dbx を使って UBC を表示する 12-29

uprofile コーティリティ
 アプリケーションのプロファイリング 2-35

uptime コマンド
 システムの負荷を表示する.. 13-2, 13-3

V

Visual Threads
 アプリケーションのプロファイリング 2-36

vm_aggressive_swap 属性
 スワッピングを積極的に行うようにする 12-35

vm_asyncswapbuffers 属性
 スワップ入出力キューの長さの制御 12-18
 スワップ領域を管理する ... 12-41

vm_bigpg_anon 属性
 可変メモリ用のビッグ・ページの構成 12-49

vm_bigpg_enabled 属性
 ビッグ・ページの有効化 ... 12-46

vm_bigpg_seg 属性
 テキスト・オブジェクト用のビッグ・ページの構成 12-50

vm_bigpg_shm 属性
 共用メモリ用のビッグ・ページの構成 12-50

vm_bigpg_ssm 属性
 セグメント化共用メモリ用のビッグ・ページの構成 12-50

vm_bigpg_stack 属性
 スタック・メモリ用のビッグ・ページの構成 12-51

vm_bigpg_thresh 属性
 ビッグ・ページごとの空きメモリの配分 12-47

vm_max_rdpgio_kluster 属性
 ページ・イン・クラスタのサイズを大きくする 12-40

vm_max_wrpgio_kluster 属性
 ページ・アウト・クラスタのサイズを制御する 12-40

vm_page_free_hardswap 属性
 スワッピングしきい値の設定 12-13

vm_page_free_min 属性
 スワッピング・レートを管理する 12-33
 フリー・リストの最小値の設定 12-13

vm_page_free_optimal 属性
 スワッピングしきい値の設定 12-13
 スワッピング・レートを管理する 12-33

vm_page_free_reserved 属性
 特権タスクのしきい値の設定 12-13

vm_page_free_swap 属性
 スワッピングしきい値の設定 12-13

vm_page_free_target 属性
 スワッピング・レートを管理する 12-33
 ページングしきい値の設定. 12-12
 ページングを制御する 12-32

vm_page_prewrite_target 属性

変更ページの事前書き出し . 12-13
 ページの事前書き出し 12-38

vm_rss_block_target 属性
 常駐セットを制限するための
 空きページしきい値を設定す
 る 12-35

vm_rss_maxpercent 属性
 常駐セットの限界値を設定す
 る 12-35

vm_rss_wakeup_target 属性
 常駐セットを制限するための
 空きページしきい値を設定す
 る 12-35

vm_swap_eager 属性
 推奨値 4-10
 スワップ割り当てモードを変更す
 る 4-10

vm_syncswapbuffers 属性
 スワップ入出力キューの長さの制
 御 12-18
 スワップ領域を管理する ... 12-41

vm_ubcdirtypercent 属性
 UBC のチューニング 11-5
 推奨値 4-10
 ダーティ・ページの割合を変更す
 る 4-10
 ページの事前書き出し 12-13,
 12-38

vm_ubcpagesteal 属性
 UBC のチューニング 11-5

vm_ubcseqpercent 属性
 UBC が 1 つのファイルに対して使
 用できるメモリ 4-9
 UBC のチューニング 11-5

推奨値 4-9

vm_ubcseqstartpercent 属性
 UBC しきい値を変更する 4-9
 UBC のチューニング 11-5
 推奨値 4-9

vmstat コマンド 13-10
 CPU の表示 12-20
 仮想メモリの使用状況のデータを表
 示 6-6
 仮想メモリを表示する 12-20,
 12-21
 ドロップしたパケットの表示 12-31
 ページ・リストの維持管理... 12-3

vnode_age 属性
 namei キャッシュの制御 11-2

vnode_deallocation_enable 属性
 namei キャッシュの制御 11-2

vnodes
 定義 8-14

volstat ユーティリティ
 LSM イベントのモニタリング 2-31

volwatch ユーティリティ
 LSM イベントのモニタリング 2-31

W

w ユーティリティ
 システム情報の表示 13-3

X

X/Open トランスポート・インタ
 フェース 8-8

xload コマンド

システムの負荷のモニタリング 13-3
 システム負荷のモニタリング 2-31
XTL..... 8-8

あ

アイドル時間
 表示 12-22
 モニタリング 9-4
 空きページ・リスト 12-3
 表示 12-22
 アクセス・パターン
 シーケンシャル 1-4
 ランダム 1-4
 アクティブ・ページ・リスト ... 12-3
 表示 12-22
 アダプティブ **RAID3/5**..... 1-6
 アプリケーション
 CPU およびメモリの統計情報 13-2
 Oracle のチューニング 4-1
 アドレス空間 8-7
 インターネット・サーバのチューニング 6-1
 仮想アドレス空間 12-6, 12-26
 コンパイラ 7-2
 シェアード・ライブラリ 7-3
 常駐セットのサイズ 12-26
 性能改善 7-1
 デバッグ 2-32, 2-36, 2-37
 特性 1-36
 ネットワーク・ファイル・システム
 のチューニング 5-1
 パッチ 7-2
 必要とするメモリ量 7-3

表示する 12-25
 プロセスのリソース 8-1
 プロファイリング 2-32, 2-33
 並列処理 7-2
 メモリの使用 12-29
 メモリ・ロック 7-4
 優先順位 13-23
 粒度ヒント 12-44
 アプリケーション・マネージャ
 クラス・スケジューラ 13-10

い

イベントのモニタリング 2-31
 Compaq Analyze 2-5
 DECevent 2-5
 nfswatch コマンド 2-31
 Performance Visualizer 2-31
 volstat ユーティリティ 2-31
 volwatch コマンド 2-31
 イベント・マネージャ 2-4
 イベントのロギング
 オプション 2-3
 イベント・マネージャ
 システム・イベントのモニタリング
 グ 2-4
 インターネット
 サブシステム 1-37
 インターネット・サーバ
 IP アドレスのロギング 6-3
 netstat コマンド 2-20
 SMP システム 6-20
 インターネット属性の変更 ... 6-8,
 6-15
 仮想メモリ属性の変更 6-23

仮想メモリ統計情報のモニタリング	6-6
基本的な推奨チューニング	6-7
構成情報の収集	6-7
高度な推奨チューニング	6-14
スクリーニング・ファイアウォールのチューニング	6-21
スクリーニング・ルータのチューニング	6-21
ソケット属性の変更	6-12, 6-23
ソケット統計情報のモニタリング	6-5
チューニング	6-1
定義	8-1
ネットワーク属性の変更	6-21
ネットワーク統計情報のモニタリング	6-4
のタイプ	6-7
汎用属性の変更	6-14
ハードウェアの構成のガイドライン	6-2
プロセス属性の変更	6-9
マルチプロセス	6-11
メモリとスワップ領域の構成	6-2

え

エクステント・マップ	
表示する	11-25
延期スワップ・モード	4-10

お

オープン・ファイル	
-----------	--

lsuf による表示	2-37
------------	------

か

回転待ち時間	1-4
課金	
リソースのモニタリング	2-6
仮想アドレス空間	12-6, 12-7
サイズの表示	12-26
仮想メモリ	
アクセスしているアドレス	12-8
アドレス空間	12-6
アプリケーションが必要とするメモリ量	7-3
仮想アドレスの変換	12-7
機能	12-2
サブシステム	1-37
常駐セット	12-8
スワッピング動作	12-16
積極的なスワッピング	12-35
表示する	12-21, 12-25
分配	1-32, 12-2
ページ・テーブル	12-7
ページの表示	12-24
ページ・フォールト	12-8
ページングしきい値	12-32
ページング動作	12-14
ワーキング・セット	12-8
管理する	
CPU リソース	13-7
カーネル	
サイズを小さくする	12-30
デバッグ	2-37
プロファイリング	2-35

き

- キャッシュ・アクセス時間 1-34
- 共用メモリ
 - メモリの予約 12-42

く

- クライアント
 - 再送する 5-17
- クラス・スケジューラ... 13-7, 13-9
 - CDE 13-10
 - class_admin 13-8, 13-12
 - class_scheduling 13-8
 - GID 13-9, 13-15
 - nice コマンド 13-10
 - PGID 13-9
 - PID 13-9
 - runclass コマンド... 13-8, 13-18
 - SESS 13-9
 - SysMan Menu 13-10
 - UID 13-9, 13-15
 - 起動 13-10
 - クラスの管理 13-14
 - クラスの作成 13-14
 - クラスの破壊 13-17
 - クラス・メンバを削除する. 13-17
 - クラス・メンバを追加する. 13-16
 - グラフィカル・ユーザ・インタフェース 13-18
 - 構成 13-12
 - 識別子タイプ 13-15
 - データベースのロード 13-17
 - デーモン 13-16
 - プランニング 13-11

- プロセス識別子 13-15
- 無効にする 13-16
- 有効にする 13-16
- 優先順位の変更 13-17

こ

- 構成
 - NFS 5-4
 - sys_check ユーティリティ... 2-13
 - ハードウェア 1-2
- 固定ページ・リスト 12-3
 - 表示 12-22
- 固定メモリ 1-31, 12-1
- コピー・オン・ライト・ページ・フォールト 12-10

さ

- 作業負荷 1-3
 - 把握 1-36
 - リソース・モデルの明確化... 1-36
- サブシステム
 - 一般的なチューニング対象... 1-37
 - インターネット 1-38
 - 仮想メモリ 1-37
 - ソケット 1-38
 - プロセス 1-37
 - プロセス間通信 1-37
- サーバ
 - NFS 性能の改善 5-7

し

- シグナル 8-8
- システム

CPU リソースの最適化 13-6
CPU を追加する 13-6
システム・イベント
tcpdump ユーティリティによるモニタリング 2-31
システム時間
表示 12-22
モニタリング 9-4
システムのジョブ
統計情報の表示 13-3
システムの負荷
nice による軽減 13-4
モニタリング 13-2, 13-3
モニタリングする 13-3
システム・バッファ・キャッシュ 5-7
常駐セット 12-8
サイズの表示 12-26
サイズを制御する 12-35
ショート・ページ・フォールト 12-9
シンメトリック・マルチプロセッシング 1-34
チューニング 6-20
シーク時間 1-4
シーケンシャル・アクセス・パターン 1-4

す

推奨
インターネット・サーバの構成 6-1
スケーラビリティ 1-3
スタック・サイズ
大きくする 8-6
ストライピング

ハードウェア RAID 9-10
ストリーム 8-8
スループット 1-4
スレッド
の不足 6-11
スワッピング
しきい値 12-13, 12-17
性能への影響 12-17
積極的 12-35
ディスク・スペース 12-20
動作 12-16
頻度の制御 12-14
メモリ管理 1-32
レートを変更する 12-33
スワップ・アウト 12-16
スワップ・イン 12-17
スワップ領域 12-41
管理する 12-41
指定 12-18
性能ガイドライン 12-18
入出力キューの長さ 12-18
非同期要求 12-41
表示する 12-28
分散 12-18
モニタリング 9-5

せ

性能
Oracle での改善 4-4
インターネット・サーバの改善 6-1
ネットワーク・ファイル・システム
の改善 5-1
モニタリング 2-1

問題の解決のための方法論的アプローチ 2-2
ゼロフィル・オンデマンド・ページ・フォールト 12-9

そ

即時スワップ・モード 4-10
属性
AdvFS での変更 4-11
NFS クライアント側の変更 .. 5-16
NFS サーバ側の変更 5-10
Reliable Datagram の変更 ... 4-19
値の表示 3-2
インターネット・サブシステムの変更 4-14, 6-8
インターネットの変更 6-15
永久値の変更 3-5
オペレーティング・システムでサポート 3-1
仮想ファイル・システムの変更 4-12
仮想メモリの変更 4-5, 6-23
現在値の変更 3-4
システム・リソースの限界値の設定 6-10
ソケット・サブシステムの変更 6-23
ソケットの変更 6-12
名前の変更 6-1
ネットワークの変更 6-21
汎用の変更 6-14
プロセス間通信の変更 4-13
プロセス・サブシステムの変更 4-15

プロセスの変更 6-9
メモリ・チャンネルの変更 4-21
リアルタイムの変更 4-19
ソケット
IPC 8-8
サブシステム 1-37
チューニング 10-8, 10-21
モニタリング 10-2, 10-4
ソフトウェア **RAID**
(LSM を参照)
ゾーニング 1-21
(ファイバ・チャンネル・スイッチ も参照)

た

帯域幅 1-4
大規模プログラム 8-5
(プログラム・サイズの限界値も参照)

ち

チューニング
AdvFS 1-29
CPU 13-6
IPC の限界値 8-8
NFS 1-30, 5-9, 5-16
Oracle 4-1
UBC 11-5
UFS 11-38
アドレス空間 8-7
アプリケーション・メモリ 7-3
インターネット・サーバ 6-1
オープン・ファイルの限界値 8-14
システム・リソース 8-1

ネットワーク・サブシステム 10-4
ファイル・システム..... 1-29
プログラム・サイズの限界値 . 8-5
プロセスの限界値..... 8-1
ページングとスワッピング. 12-31
メモリ 12-29, 12-31
調停ループ
特徴 1-17
ファブリック・トポロジとの比
較 1-18
直接入出力 11-15

て

ディスク
LSM 9-6
RAID5 1-6
クォータ 2-6
性能の改善..... 9-1
断片化を解消する.. 11-19, 11-43
データの分散 9-1
入出力の分散状況のモニタリン
グ 9-3
入出力分散のガイドライン.... 9-2
ハードウェア RAID..... 9-7
ハードウェア RAID サブシステムで
の使用..... 9-8
表示する 11-35
ファイル・システムの分散.... 9-3
ディスク・クォータ
UFS..... 11-34
ディスク使用量の制限 2-6
デバッグ
dbx コマンド..... 2-36

kdbx コマンド 2-37
ladebug..... 2-37
アプリケーション..... 2-32
転送方式 1-10
データのキャッシング
物理メモリ..... 1-33
データの損失を防ぐ
AdvFS 11-13
RAID を使用 11-13
データ・パス..... 1-8
データを分散させる
AdvFS 11-17
デーモン
クラス・スケジューラ 13-16

と

同期スワップ・バッファ 12-18
同期入出力 11-27
同期要求 12-41
動的パリティ **RAID**..... 1-6

に

入出力クラスタ
クラスタの読み書きを表示す
る 11-36

ね

ネットワーク
IP アドレスの検索 10-15
IP 入力キュー..... 10-12
keepalive 10-14

LAG インタフェース 1-28
 mbuf クラスタの圧縮 10-13
 NetRAIN 1-26
 NFS の限界値 5-7
 PMTU の検出 10-11
 TCP コンテキストのタイムアウト限
 界値 10-17
 TCP 再送レート 10-18
 TCP セグメントのサイズ .. 10-19
 TCP データの肯定応答 10-19
 TCP の検索速度 10-7
 TCP ハッシュ・テーブル 10-8
 UDP ソケット・バッファ .. 10-20
 サブシステム 1-25, 1-37
 冗長ネットワークの使用 1-26
 送信接続ポート 10-10, 10-11
 ソケット・バッファのサイズ 10-21
 ソケット・リッスン・キュー 10-8
 チューニングのガイドライン 10-4
 ネットワーク・アダプタ 1-24
 ネットワーク・インタフェー
 ス 1-24
 ネットワーク・インタフェース・
 カード 1-24
 パケットのドロップをチェックす
 る 10-21
 パケットのドロップを防止す
 る 10-21
 パーシャル TCP タイムアウトの限
 界値 10-16
 表示 2-17
 モニタリング 2-20,
 2-31, 10-2, 10-3
 ルーティング 1-27

は

パイプ 8-8
 バス
 速度 1-9
 ターミネーション 1-12
 データの分散 9-1
 長さ 1-12
 ハードウェア
 構成 1-2
 情報収集 2-7
 (hwmgr ユーティリティ も参
 照)
 ハードウェア **RAID** 9-7
 (RAID も参照)
 RAID サポート 9-10
 機能 9-8
 構成のガイドライン 9-7, 9-11
 ストライプ・サイズ 9-12
 スペア・ディスク 9-15
 製品 9-10
 ディスク・データを分散させ
 る 9-12
 ディスク容量 9-12
 デュアル冗長コントローラ... 9-15
 ミラー・ディスクのストライピン
 グ 9-14
 ライトバック・キャッシュ... 9-9,
 9-14
 ハードウェア構成
 概要 1-2
 図の作成 1-2
 ハードウェアの構成
 インターネット・サーバの性能改善
 のガイドライン 6-2

ひ

非アクティブ・ページ・リスト 12-3
ビッグ・ページ・メモリ 12-46
非同期スワップ・バッファ ... 12-18
非同期入出力..... 11-27
表示
 NFS..... 5-2
 スワップ領域 12-20
 ファイル・システム..... 11-22
 メモリ 12-19
表示する
 UFS..... 11-35
 ファイル・システム..... 11-35

ふ

ファイバ・チャンネル 1-13
 (SCSI も参照)
 SCSI の制限を克服..... 1-13
 距離 1-14
 ゾーニング..... 1-21
 (ファイバ・チャンネル・スイッチ も参照)
 調停ループ..... 1-17
 データ転送速度 1-13
 トポロジ 1-14
 ファブリック 1-15
 ポイント・ツー・ポイント... 1-14
ファイバ・チャンネル・スイッチ
 ゾーニングが必要な場合 1-21
ファイル・システム
 AdvFS 11-11
 NFS..... 11-44

UFS..... 11-31
チューニング 11-38
表示 11-22
分散 9-3
リソース..... 1-29
ファイルセット
 表示する..... 11-26
ファイル・ドメイン
 表示 11-25
ファイルをストライピングする
 AdvFS 11-17
ファブリック 1-15
物理メモリ 1-31
 UBC 1-32, 12-2
 仮想メモリ..... 1-32, 12-2
 共用メモリ用の予約..... 12-42
 固定 1-31, 12-1
 データのキャッシング 1-33
 プロセス..... 1-32
 分配 1-31, 12-1
フラッシュ・キュー 11-27
プログラム・サイズの限界値
 チューニング 8-5
プロセス
 サブシステム 1-37
 常駐セットのサイズの限界値 12-35
プロセス間通信 8-8
 (IPC も参照)
 サブシステム 1-37
プロセス・チューナ 13-10
 プロセス情報の表示..... 13-2
ブロック・キュー 11-27
プロファイリング
 アプリケーション..... 2-32

へ

- 並列処理
 - アプリケーションで使用する . 7-2
- 変更ページの事前書き出し ... 12-13
 - 管理する 12-38
- ページ
 - 維持管理 12-3
 - サイズ 1-31, 12-1
 - 再生 12-2, 12-10
 - 表示 12-22
- ページ・アウト 12-15
 - 表示 12-22
- ページ・イン
 - 表示 12-22
- ページ・イン・ページ・フォールト 12-9
- ページ・カラリング 12-10
- ページ・テーブル 12-7
- ページ・フォールト 12-9
- ページ・マッピング
 - ビッグ・ページ・メモリの割り当て 12-46
- ページ・リスト
 - 維持管理 12-3
- ページング 12-14
 - しきい値 12-12, 12-14
 - しきい値を大きくする 12-32
 - 属性 12-11
 - 表示 12-24
 - 頻度の制御 12-14
 - ページの再生 12-2
 - メモリ管理 1-32

ほ

- ポイント・ツー・ポイント 1-14
- ボトルネック 1-4

ま

- 待ち時間 1-4
- マルチプロセッシング 13-6
- マルチプロセッサ 1-33

み

- ミラーリング
 - RAID1 1-6
 - ハードウェア RAID 9-10

め

- メタデータ・バッファ・キャッ
シュ 12-4
- チューニング 11-8
- ハッシュ・チェーン・テーブルのサ
イズ 11-8
- 表示する 11-37
- メモリ
 - アプリケーションの性能の改
善 12-46
- メモリ管理 1-31, 12-1
(ページング も参照)
 - CPU キャッシュ・アクセス .. 1-34
 - PAL コード 12-9
 - UBC 12-4
 - 概要 1-31
 - スワッピング 1-32, 12-16
 - スワップ・バッファ 12-18

動作 12-1
バッファ・キャッシュ 1-34
変更ページの事前書き出し. 12-13,
12-38
ページの維持管理 12-3
ページング 1-32, 12-14
メタデータ・バッファ・キャッ
シュ 12-4
メモリ・リソースを増やす. 12-29
ロックする 7-4
メモリ・ファイル・システム
(MFS を参照)

も

モニタリング
CAM 9-16
CPU 13-1
lsof コマンド 2-37
monitor コマンド 2-31
top コマンド 2-31
アプリケーション 2-32
オープン・ファイル 2-37
ソケット 10-4
ディスク入出力の分散 9-3
ディスク入出力の分散状況 9-4
ネットワーク 10-2

ゆ

優先順位
アプリケーションの変更 ... 13-23
ユニファイド・バッファ・キャッ
シュ 12-4

(UBC も参照)
ユーザ・アドレス空間
大きくする 8-7
ユーザ時間
表示 12-22
モニタリング 9-4
ユーザ・データ・セグメント
大きくする 8-6

ら

ライトバック・キャッシュ
ハードウェア RAID 9-9, 9-14
マルチプロセッシング・システ
ム 13-6
ランダム・アクセス・パターン.. 1-4

り

リアルタイム・プロセス間通信
パイプとシグナル 8-8
リソース
CPU 1-33
ディスク・ストレージ 1-5
ネットワーク 1-24
ネットワーク・サブシステム 1-25
ファイル・システム 1-29
メモリ 1-31
粒度ヒント
共用メモリの予約 12-42

る

ルーティング
ネットワーク・パス 1-27

ループ・トポロジ
特徴 1-17

ろ

ロック
モニタリング 13-2, 13-5

わ

割り込み
表示 12-23
ワーキング・セット 12-8

Tru64 UNIX ドキュメントの購入方法

Tru64 UNIX ドキュメントのご購入については、弊社担当営業または日本ヒューレット・パッカートの各営業所/代理店にお問い合わせください。

各ドキュメント・キットの注文番号は以下のとおりです。ドキュメント・キットに含まれるマニュアルの内容については『ドキュメント概要』を参照してください。

キット名	注文番号
Tru64 UNIX Documentation CD-ROM	QA-6ADAA-G8
Tru64 UNIX Documentation Kit	QA-6ADAA-GZ
End User Documentation Kit	QA-6ADAB-GZ
- Startup Documentation Kit	QA-6ADAC-GZ
- General User Documentation Kit	QA-6ADAD-GZ
- System and Network Management Documentation Kit	QA-6ADAE-GZ
Developer's Documentation Kit	QA-6ADAF-GZ
Reference Pages Documentation Kit	QA-6ADAG-GZ
TruCluster Server Documentation Kit	QA-6BRAA-GZ
Tru64 UNIX 日本語ドキュメント・キット	QA-6ADJB-GZ
スタートアップ・ドキュメント・キット	QA-6ADJC-GZ
一般ユーザ・ドキュメント・キット	QA-6ADJD-GZ
システム/ネットワーク管理ドキュメント・キット	QA-6ADJE-GZ
プログラミング・ドキュメント・キット	QA-6ADJF-GZ
CDE 翻訳ドキュメント・キット	QA-6ADJG-GZ
TruCluster Server 日本語ドキュメント・キット	QA-05SJA-GZ
Advanced Server for UNIX 日本語ドキュメント・キット	QA-5U2JA-GZ



マニュアルに対するご意見

Tru64 UNIX

システムの構成とチューニング

AA-RM7AC-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