次世代ファイルシステムZFSへのお誘い

2007年09月26日（火）jus勉強会

株式会社エンターモーション
システム開発部

重村 法克

nork@FreeBSD.org
n_shigemura@entermotion.jp
nork@ninth-nine.com

アジェンダ1

- 次世代ファイルシステムZFSとは
  - ZFSの構成
  - ZFSとボリュームマネージャーの違い
  - ZFSとディスクアレイの違い
  - ZFSにフラグメンテーションはあるか
  - ZFSでデフラグは必要か
- UFS2とZFSの違い
- FreeBSDにおけるZFSの事情と今後

アジェンダ2

- ZFSでできること
  - ストレージプールの操作
    - ストレージプールの作成/一覧/破棄
    - ストレージプールへの追加/削除
    - ストレージプールデバイスへの接続/切り離し
    - ストレージプールデバイスの置き換え
    - ストレージプールのステータス表示/変更
    - ストレージプールのメンテナンス
- ZFSでできること
  - ファイルシステムの操作
    - ファイルシステムの作成/一覧/破棄

アジェンダ3

- ZFSでできること
  - ファイルシステムの操作
    - ボリュームの作成
    - ファイルシステムのパラメータチューンニング
    - ファイルシステムのスナップショット
    - ファイルシステムのクローン/プロモート
    - ファイルシステムのバックアップ
- 運用事例
  - PostgreSQLのPoint In Time Recoveryとの組み合わせ
  - FreeBSDのCVSリポジトリの収納

アジェンダ4

- まとめ
- 参考文献

次世代ファイルシステムZFSとは

ZFSの特徴について述べる。

ZFS理解のためのベースとして説明するが、実際のところはトライアンドエラーということで:-)。

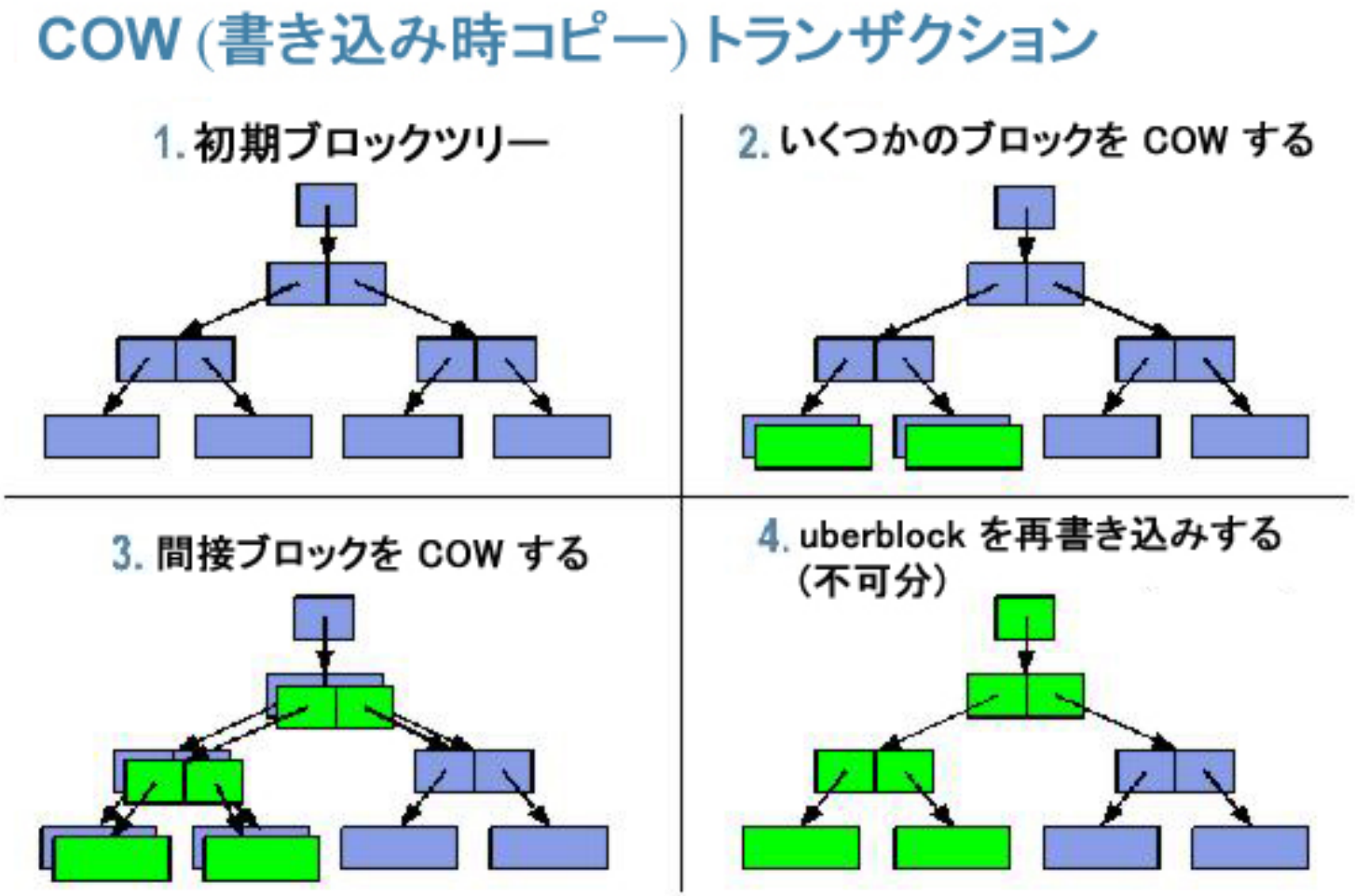
次世代ファイルシステム ZFSとは

- ZFSは、それまでのファイルシステムの概念をふっとばす（!）新しいファイルシステム。
  ZFSには以下の特徴を備えている。
  - 単純な管理機能（コマンド自体は2つ）。
  - Copy on Write メカニズムによるトランザクションの導入とfsck不要の一貫性保持。
  - エンドtoエンドのデータ整合性。
  - 桁外れのスケーラビリティ。

次世代ファイルシステム ZFSとは

- ストレージプールの導入。
- 既存ファイルシステム技術の延長にあるものではなく、メモリ管理機構の考え方を二次記憶に適用する方針で設計されていること。
- 何よりも、ZFSにもたらされた機能を利用することで、長年に渡って来て運用された常識が変わる。

次世代ファイルシステム ZFSとは

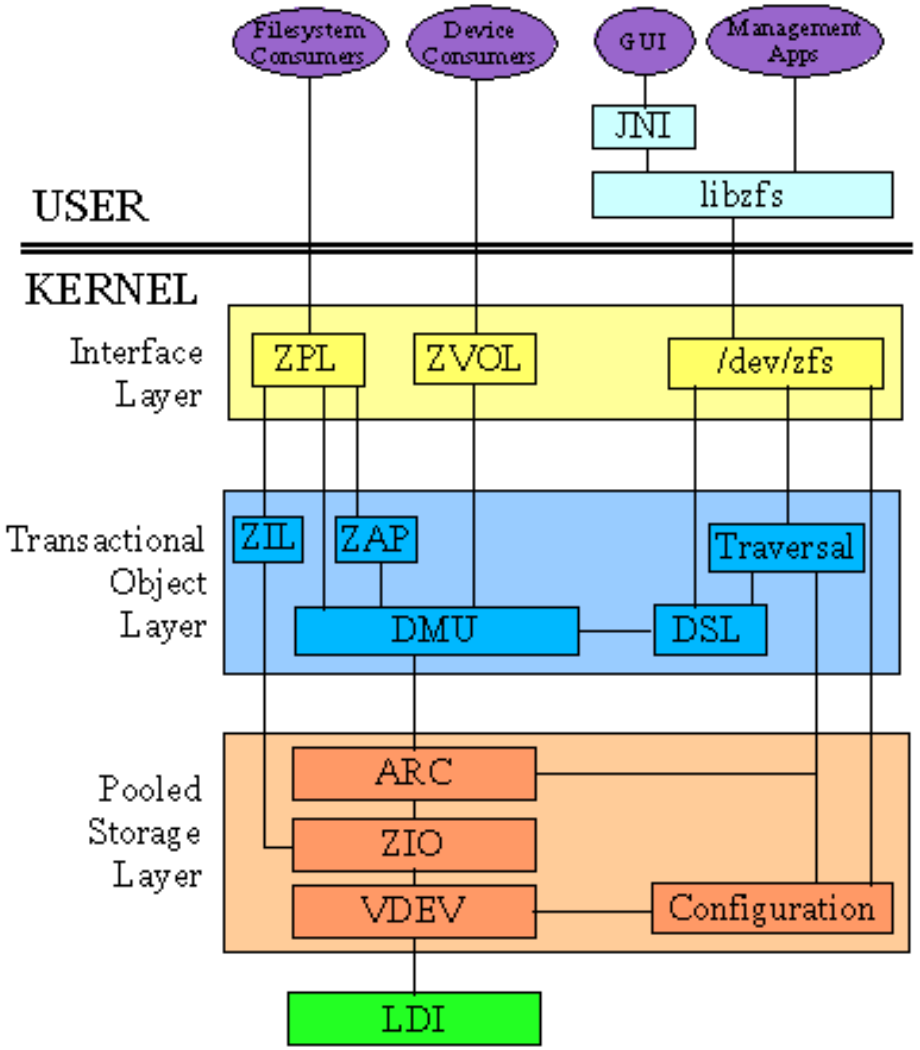


出典:
http://www.sun.com/bigadmin/hubs/multilingual/japanese/content/zfs_part1.scalable.jsp より

次世代ファイルシステム ZFSとは

- ファイルシステムとストレージの統括
  - ストレージプールを導入することでファイルシステムとストレージをダイナミックに融合。従来のストレージはストレージの、ファイルシステムはファイルシステムのといった、さかいなく、相互の情報を用いて運用される。
  - ファイルはストレージプールからオンデマンドで確保。
  - ストレージは必要な分だけストレージプールに提供。
- 1つ1つの機能は既存のファイルシステム、ボリュームマネージャ、RDBMSの概念にあるものであるが、高度に昇華した使い勝手を提供している。

ZFSの構成



- ZPL
  ZFS POSIX Layer
- ZVOL
  ZFS Volume
- ZIL
  ZFS Intent Log
- ZAP
  ZFS Attribute Processor
- DMU
  Data Management Unit
- DSL
  Dataset and Snapshot Layer
- ARC
  Adaptive Replacement Cache

出典：
http://www.sun.com/bigadmin/hubs/multilingual/japanese/content/zfs-source.jsp より

ZFSとボリュームマネージャーの違い

- ZFSはボリュームマネージメント機能を有する
  では既存のボリュームマネージャーとの違いは何か。
  - ボリュームマネージャーはあくまでもストレージのマネージメントを行うだけ。
  - ボリュームマネージメントを行うことで、容量の増減があっても、ファイルシステムに反映できない。通常growfsといったコマンドを使用する必要がある。
  - よってボリュームマネージャーの仕事は、ファイルシステムへ影響をあたえないことを至上命題とする。
  - ZFSのボリュームマネージャーは容量を変更できる。もちろん一般のボリュームマネージャーでも容量は変更できうるものがあるが…。

ZFSとボリュームマネージャーの違い

- 最大の違いはファイルシステムのメタデータと連動していること。
  - 従来は、ストレージとストレージの結合はコンカチネート。または、同容量同士、同じタイミングで構成した場合でのみストライピング。
  - しかし、ZFSではストレージプール上ではコンカチネートであるが（実際にはそういう風に振る舞ってるように見えるだけ）、ファイルシステムレベルでストレージへの割り当てが行われるため、ファイル毎にストライピングしている（動的ストライプ）。

ZFSとディスクアレイの違い

- ZFSはディスクアレイを機能（RAID1, RAID5, RAID6）をサポートしている。
  ではなぜZFSはディスクアレイをサポートしているのか。
  - 各RAID（RAID0を除く）で期待されていることはディスクの保全性とコストとのバランスである。一般のRAIDシステムはハード障害に対しては十分機能する。
  - ZFSではブロックサイズ（4096バイトから128キロバイト可変）毎にチェックサムを取っているため、ソフト障害に強い。つまり、ハードウェアは正常にもかかわらず、ファイルが損傷している場合に強い。またRAID5（RAID6）が持つ問題（write hole problem）についても強い。

ZFSとディスクアレイの違い

- RAID5は、例えばA, Bブロックのデータからパリティ P(A, B)を計算し、書き込み位置を計算した上で書き込みが行われる。
- RAIDZでは、例えば、それぞれのディスクa, b, cにメタデータを有し、A, B, P(A, B)それぞれのチェックサムを計算して書き込みを行う。またそれぞれの書き込み位置はファイルシステムのメタデータ上に記録される。その上でストレージ上の最適な位置に置かれる。

ZFSにフラグメンテーションはあるか

- 原理的にフラグメントフリーな設計はないので、なんらかの対応が必要である。
- ZFSではブロック長を4KB, 8KB, 16KB, 32KB, 64KB, 128KBと可変長にすることで、本質的に減らす仕組みがある。これは例えば128KBまでのファイルはフラグメンテーションは原理的に起きない。
- 128KBより大きなファイルではフラグメンテーションが発生するが、I/Oリクエストとしては十分な長さと言える。
- 個人的には、フラグメンテーションよりも、大きなブロックをCoWする方が気になるかも。

ZFSでデフラグは必要か

- たぶん気にしなくていいというのが自分結論。
- フラグメントが発生しにくい作りになっている点がひとつ。フラグメントが発生しても十分お釣りがくるI/Oリクエストの長さであることがふたつ。
- そもそもストライピングしている時に、断片化してるという判断ができるかどうか。
- ZFS版の空飛ぶ絨毯を欲して、内周・外周への配置やら何やらを気にするくらいなら、新しいハードディスク追加した方がいいと思う。A, Bブロックを並列にリクエストしてるところでBがB'になったところで、リクエストの並列度は変わらない。

ZFSとUFS2の違い

ZFSとUFS2の違いについてまとめる。

ZFSに導入された機能により管理手順が変わるのはもちろんであるが、運用概念が変わったものも多数ある。このあたりを把握した上で、ZFSを利用したいところ。

ZFSとUFS2の違い

- 事実上パーティショニング不要
  - ZFSは必要な量、必要な分を、ストレージプールから確保するので、パーティショニングといった、事前にディスク使用量を規定しておく必要が無い。
  - とは言え、全体で使用できるディスク容量は限られているので、特定のファイルシステムが他のファイルシステムへの影響は否定できない。
  - しかし、リザベーション機能により、最低ディスク使用量を規定できる。よって / （ルートパーティション）１個という状態よりは、例えば /var ディレクトリを保護しやすい。

ZFSとUFS2の違い

- 事実上パーティショニング不要
  - むしろ /var/log, /var/tmp を隔離することで他の /var ディレクトリを保護する運用も可能。[どこまですればよい?]
  - むしろ細かいファイルシステムの作成・運用が可能なため、セオリーがない。これから試行錯誤していくことになる。

ZFSとUFS2の違い

- 事実上newfsしない。
  - ZFSはmkdirするノリでファイルシステムを作成する。とは言えマウントポイントなのでmvはできない。mkdirほど簡単ではないが、mkdirほど簡単にできてしまうので、意識する必要がある。［要注意］
- 事実上fsckしない。
  - ZFSは常にトランザクションにより一貫性を確保している。この一貫性はジャーナリングではなくCopy On Writeのため、ロールフォワーディングがない。
  - キャッシュ層を通さないアクセス（DIRECT I/O）を処理するためにロギング - ZIL（ZFS Intent Log）を行っている。このログを反映するためにロールフォワードがありうる。

ZFSとUFS2の違い

- 事実上quotaが無い。
  - ユーザー/グループquotaが無い。ZFSで言うquotaはファイルシステムの上限（ソフトリミット）の定義である。またファイルシステム単位で制約を設定する。
  - よってユーザーのホームディレクトリ単位で「ファイルシステムを作る」必要がある。［要注意］
- chflagsが使えない。
  - ファイルフラグのマッピングについては移植を諦めてるようである。

ZFSとUFS2の違い

- 事実上ACLが使えない。
  - ZFSにはACLを持っているが…。
  - FreeBSDにはNFSv4/NTFS ACLをサポートするAPI（syscall）が存在しない。
  - TrustedBSDの成果が入ればNFSv4/NTFS ACLが使えるとは聞いている。進捗は知らない:-)。
  - またZFSはPOSIX ACLをサポートしていない。

ZFSとUFS2の違い

- スナップショットが強力で高速
  UFS2のスナップショットと比べいくらか機能が強化されている。スナップショットを作る、という指示は同じでも…
  - スナップショットを取るのに必要なディスク容量はUFS2のそれよりずっと少ない。
  - スナップショットを取るのに必要な時間が短い。
    「Use the ZFS, Luke」(c)McKusickと言ったとか言わないとか。
  - スナップショットをクローン化して（zfs clone）スナップショットをwritableに運用可能。更に、プロモートする（zfs promte）と、オリジンから切り離してファイルシステムとして運用できる。正直そんな運用できねーと思ってるので検証してない。たぶん切り戻し作業で重宝する。

ZFSとUFS2の違い

- スナップショットが強力で高速
  - 従来のファイルシステム（これはUFS2を含め）にはスナップショットとそれをマウントする概念がない。
  - よってZFSとUFS2ではマウント方法とマウントポイントが違う。
  - 「.zfs/snapshot/スナップショット名」をアクセスすると指定されたファイルシステムの「スナップショット」をread only mount されて見えるようになる。
  - このマウント処理は automount で、自動でunmountされたことを見たことがない:-）。

FreeBSDにおけるZFSの事情と今後

FreeBSDにおけるZFSの開発状況と、Solarisとの違いについて述べる。

Solarisの話はほとんどの場合通じるけど、どうしてもFreeBSD固有の話が出るということで:-)。

FreeBSDにおけるZFSの事情と今後

- 移植は完了しており、2007年4月時点での本家の全機能は網羅されている。事実上、OpenSolarisレベルのコードを即座に反映可能な状態にあるが、リリース時期が近いこともあり、そう頻繁に更新を行っていない。
- OS的事情により、いくつか使えない機能がある。先に説明したNFSv4/NTFS ACLが代表的だが、iSCSIは最近まで検証できなかった。もっともiSCSIをブートできるNICが少ないので、USB HDD程度の使い道しかないように思われるが…。またPXEブートしても、NFSマウントが関の山なので（以下略。

FreeBSDにおけるZFSの事情と今後

- SolarisのZoneに相当する機能として、jail化が行われている（zfs jail）。
- ZFS関連のデータ（キャッシュ）として/etc/zfs以下が使用されている。
- FreeBSDでも基本的には/etc/zfsを使用しているが、/（ルート）ディレクトリをZFSにすることができる関係で、/boot/zfs/zpool.cacheにzpool情報を収納している。
- ブートローダがZFSに対応していないため、/bootディレクトリはUFS2である必要がある。現在ZFSBootは開発中である（Solaris版とは別に）。

FreeBSDにおけるZFSの事情と今後

- 仮りにZFSBootがあったとしても、インストーラー（sysinstall）が致命的に腐っているため（しょうがないところであるけど）、ZFSでインストールしようにも極めて手順が大変である。また、実際にはライセンス（CDDL）の都合もあり、GENERICカーネルに含まれないため、インストール後も地雷を踏む可能性が高い（/boot/loader.confを設定しよう！）。
- ZIL（ZFS Intent Log）回りでメモリリークが確認されている。本家でも議論になってることからSolarisでも発生しうる問題と思われるが、今のところ修正に関する進捗は不明。

FreeBSDにおけるZFSの事情と今後

- 本家が対応してない機能はFreeBSDでも対応していない。
  - 意外にも、トップレベルストレージに接続されているHDDの切り離しができない（＝容量を減らせない）。切り離しはできないがリプレースメントは可能。
  - 暗号化できない。こちらはgeom_geli(4)と使うのがいいと思われる。
  - クラスタリングできない。

ZFSでできること

ここではZFS操作のための実際のオペレーションについて解説する。実機を見ながら作業するのがベストではあるが、雰囲気だけつかめれば幸いである。

ストレージプールの操 作

※まず箱を用意しましょう。ということで、ストレージをストレージプールに追加する方法から始める。

- ストレージプールの作成
  zpool create プール名 デバイス名
  - プール名は基本的に英数字で。例：tank
  - デバイス名はmirror, raidz, raidz2, spare を予約名として、
  - 実際のデバイス名を指定。例：da0, ad1, /dev/ad0
  - 特にオプションは無くていいかな。ストレージプール自体の属性を指定したいところはあるかもしれないけど、それはzfs set FOO=BAR プール名で対応すると。

ストレージプールの操 作

- ストレージプールの一覧
  zpool list［プール名］
  - 色々オプションあるけどとりあえずこれだけ覚えておけばOK。
- ストレージプールの破棄
  zpool destroy［プール名］
  - 指定されたストレージプールの削除。

ストレージプールの操 作

- ストレージプールへの追加
  zpool add プール名 デバイス名
  - デバイスの追加は基本的にzpool create のものと同じ。
  - MIRRORそのものへの追加は attach 参照のこと。
- ストレージプールからの削除
  zpool remove プール名 デバイス名
  - 現時点ではホットスペアしか削除できない。普通のデバイスは削除できない（本家のTODO）。
  - MIRRORそのものからの削除は detach 参照のこと。

ストレージプールの操 作

- ストレージプールデバイスへの接続
  zpool attach プール名 デバイス名 新デバイス名
  - シングルディスク構成に新しいデバイスを追加するとミラーリング、2台構成のミラーに新しいデバイスを追加すると3台構成のミラーになる。
  - RAIDZに新しいデバイスを追加することはできない。
- ストレージプールデバイスからの切り離し
  zpool detach デバイス名
  - MIRROR構成されてるものからしか外せない。

ストレージプールの操 作

- ストレージプールデバイスの置き換え
  zpool replace プール名 旧デバイス名［新デバイス名］
  - すごく危険なので注意しよう。
  - ホットスワップした場合、デバイス名が一緒になることがある。その時は zpool replace プール名 ad1 ad1 のような指定の仕方になる。
  - ホットスペアを含めて使用した場合のノウハウがないので、組み合わせチェックして検証しておきましょう。ちなみに組み合わせチェックできるほど台数持ってないので検証してない:-)。

ストレージプールの操 作

- ストレージプールのステータス表示
  zpool status［プール名］
  zpool iostat［プール名］
  - とりあえず紹介にとどめておく。
- ストレージプールのステータス変更
  zpool offline プール名 デバイス名
  zpool online プール名 デバイス名
  zpool clear プール名［デバイス名］
  - 指定したデバイスをオンライン/オフライン/エラークリアを行う。
  - オフラインにするとデグレードするので注意。

ストレージプールの操 作

- ストレージプールのメンテナンス
  zpool scrub［-s］プール名
  zpool upgrade［プール名］
  - zpool scrubでディスク全体のチェックサムが正しいか検証される（当然全てのブロックを読み込む）。-s オプションで停止することが可能。
  - zpool upgradeでメタデータのアップグレードを行う。ZFSがFreeBSDのツリーに取り込まれてから1度も行われていない。取り込まれる前にはgzip圧縮サポートでアップグレードが行われた。

ストレージプールの操 作

- ストレージプールのメンテナンス
  zpool histroy［プール名]
  - zpool historyでzpool, zfsコマンドが適応されたときの情報）（SHELLの履歴程度に）を見ることができる。
- ストレージプールのメンテナンス
  zpool import ...
  zpool export［プール名］
  - ストレージプールを外部から持ってくるのと外部で使えるようにするためのコマンド。
  - 色々と作法があるみたいだが、あまり理解してない。

ファイルシステムの操 作

- ファイルシステムの作成
  zfs create［［-o ﾌﾟﾛﾊﾟﾃｨ＝値］］... ファイルシステム
  - ファイルシステムの作成。基本的に「プール名/ディレクトリ名/...」と記述する。
  - 後でいくらでも変えられるけど、特性に応じてプロパティを設定しておこう。
  - 個人的には -o atime=off -o compression=gzip-9 かな:-）。こまめにマウントポイントを変えるのもよいかと。
    mountpoint=/home。
  - プロパティ値は上位のファイルシステムから継承するので、特にマウントポイントあたりは、ほとんど指定する必要がない。

ファイルシステムの操 作

- ZFSボリュームの作成
  zfs create［-s］［-b ﾌﾞﾛｯｸｻｲｽﾞ］-V 容量 ﾎﾞﾘｭｰﾑ名
  - 以下のオペレーションに相当する動作
    - dd if=/dev/zero of=/dev/zvol/ﾎﾞﾘｭｰﾑ名
      bs=ﾌﾞﾛｯｸｻｲｽﾞ count=容量÷ﾌﾞﾛｯｸｻｲｽﾞ
    - qemu-img create /dev/zvol/ﾎﾞﾘｭｰﾑ名 容量
  - CoWはブロックサイズに依存するので、このファイルを使用するアプリケーションの性質に応じて設定すること。
  - ZFS的にはファイルができたように振る舞う。そのため-sオプションがない場合、実際に領域を確保する（リザベーション）。

ファイルシステムの操作

- ファイルシステムの一覧
  zfs list［ﾌｧｲﾙｼｽﾃﾑ|ﾃﾞｨﾚｸﾄﾘ|ZFSﾎﾞﾘｭｰﾑ|ｽﾅｯﾌﾟｼｮｯﾄ］
  - スナップショットを含めた一覧を参照可能。
  - まぁ参考程度かな。
- ファイルシステムの破棄
  zfs destroy［-r］［ﾌｧｲﾙｼｽﾃﾑ|ｽﾅｯﾌﾟｼｮｯﾄ|ZFSﾎﾞﾘｭｰﾑ］
  - 禁断の rm -fr 級呪文。Don't miss it!

ファイルシステムの操 作

- ファイルシステムのパラメーターチューニング
  zfs get all|ﾌﾟﾛﾊﾟﾃｨ［ﾌｧｲﾙｼｽﾃﾑ|ｽﾅｯﾌﾟｼｮｯﾄ|ZFSﾎﾞﾘｭｰﾑ］
  zfs set ﾌﾟﾛﾊﾟﾃｨ=値［ﾌｧｲﾙｼｽﾃﾑ|ｽﾅｯﾌﾟｼｮｯﾄ|ZFSﾎﾞﾘｭｰﾑ］
  - 次表で紹介するパラメータ各種を参照または変更できる。
  - 設定されたパラメータは、即座に有効になるが、既存のデータに反映されることはない。
  - 典型的には圧縮可否で、無圧縮状態のところで圧縮有効にしても、ファイルシステムの使用量は変わらない。
  - メタデータ中に記録が行われるので、例えばチェックサムアルゴリズムを変更しても、既存のものは従来のチェックサムで計算される。

ファイルシステムの操作

プロパティ名	意味	値	ディフォルト	備考
quota	ﾌｧｲﾙｼｽﾃﾑの上限	ｻｲｽﾞ | none	none	
reservation	ﾌｧｲﾙｼｽﾃﾑの下限	ｻｲｽﾞ | none	none	
volsize	ZFSﾎﾞﾘｭｰﾑのｻｲｽﾞ	ｻｲｽﾞ	作成時の値	volblocksizeの自然数倍
volblocksize	ZFSﾎﾞﾘｭｰﾑのﾌﾞﾛｯｸｻｲｽﾞ	ﾌﾞﾛｯｸｻｲｽﾞ	8KB	512Bから128KBの2の累乗
mountpoint	ﾏｳﾝﾄﾎﾟｲﾝﾄ	ﾊﾟｽ | legacy | none	継承による	
sharenfs	NFS共有可否	exportsｵﾌﾟｼｮﾝ | off	off	これを設定後killall -HUP mountd実行のこと
shareiscsi	iSCSI共有可否	on | off	off	ZFSﾎﾞﾘｭｰﾑに適応
checksum	ﾁｪｯｸｻﾑｱﾙｺﾞﾘｽﾞﾑの選択	on | off | fletcher2 | fletcher4 | sha256	on	fletcher2と同義
compression	圧縮ｱﾙｺﾞﾘｽﾞﾑの選択	on | off | lzjb | gzip | gzip-N	off	gzipはgzip-6に相当、onはlzjbと同義
atime	ｱｸｾｽﾀｲﾑの記録	on | off	on	
device	ﾃﾞﾊﾞｲｽﾌｧｲﾙの有効可否	on | off	on	devfsだから今どきoffでもいいかも
exec	ﾌﾟﾛｸﾞﾗﾑ実行の可否	on | off	on	
setuid	setuid属性の有効性可否	on | off	on	
readonly	読み込みのみ属性	on | off	off	
snapdir	.zfsを見せるか見せないか	hidden | visible	hidden	ls -aした時に.zfsを見せるかどうかの設定
aclmode	chmod(2)でのACL制御	discard | groupmask | passthrough	groupmask	
aclinherit	ACL制御の継承	discard | noallow | secure | passthrough	secure	
canmount	zfs mount -aでﾏｳﾝﾄするか	on | off	on	
xattr	拡張属性の使用可否	on | off	on	
copies	CoW時のｺﾋﾟｰ回数	1 | 2 | 3	1	
jailed	jail環境下での使用可否	on | off	off	

ファイルシステムの操 作

- ファイルシステムのスナップショット
  zfs snapshot［-r］ﾌｧｲﾙｼｽﾃﾑ@名前
  - 適当な名前を決めてファイルシステムのスナップショットをとる。名前は日付を入れるなどするとよい。
  - -r オプションで指定されたファイルシステム以下のファイルシステムをまとめてスナップショットが取れる。

ファイルシステムの操作

- ファイルシステムのクローン化
  zfs clone ｽﾅｯﾌﾟｼｮｯﾄ ﾌｧｲﾙｼｽﾃﾑ|ZFSﾎﾞﾘｭｰﾑ
  - 指定されたスナップショットをファイルシステムまたはZFSボリュームとして読み書き可能マウントする。
  - この時点ではオリジナルのファイルシステムとの紐付けが行われている。
- ファイルシステムのプロモート
  zfs promote ﾌｧｲﾙｼｽﾃﾑ
  - クローン化で指定したファイルシステムをオリジナルとの紐付けを解除する。

ファイルシステムの操 作

- ファイルシステムのバックアップ
  zfs send［-i ｽﾅｯﾌﾟｼｮｯﾄ1名］ｽﾅｯﾌﾟｼｮｯﾄ2 > ﾌｧｲﾙ
  - ｽﾅｯﾌﾟｼｮｯﾄ2からフルバックアップをとる。
  - -i オプションをつけると、ｽﾅｯﾌﾟｼｮｯﾄ1からの差分バックアップが得られる。ｽﾅｯﾌﾟｼｮｯﾄ1の名前にはファイルシステム名は付かない（@より後）。
- ファイルシステムのリストア
  zfs receive ﾌｧｲﾙｼｽﾃﾑ|ZFSﾎﾞﾘｭｰﾑ|ｽﾅｯﾌﾟｼｮｯﾄ < ﾌｧｲﾙ
  - フルバックアップの場合 zfs create する。差分バックアップの場合してされたファイルシステムは存在している必要がある。

ファイルシステムの操作

- それ以外にも…

zfs rename	ﾌｧｲﾙｼｽﾃﾑの名前変更
zfs rollback	ｽﾅｯﾌﾟｼｮｯﾄのロールバック
zfs inherit	ﾌﾟﾛﾊﾟﾃｨの継承
zfs mount	ﾌｧｲﾙｼｽﾃﾑのマウント
zfs unmount	ﾌｧｲﾙｼｽﾃﾑのアンマウント
zfs share	ﾌｧｲﾙｼｽﾃﾑの共有
zfs unshare	ﾌｧｲﾙｼｽﾃﾑの非共有
zfs jail	ﾌｧｲﾙｼｽﾃﾑのjail化
zfs unjail	ﾌｧｲﾙｼｽﾃﾑのunjail化

ZFSを使用した運用事例

ZFSの全ての機能を網羅したわけではないが、ZFSがあると便利な点があるので試した事例をいくつか。。

PostgreSQLのPoint In Time Recoveryとの組み合わせ

PostgreSQL 8.0より導入されたPoint In Time Recovery機能について解説する。

- 実際にやろうと思ったら実は大変である。
- おもな原因はPostgreSQL内で完結しないため、PostgreSQL外で色々なんとかしないといけない。
- また運用方法、ポリシー他、色々な要因でこの、なんとかする方法にバッチ化するのが難しい（万人において…ではあるが）。
- もちろんセオリーはあるので、それに基づいてバッチを作るのはありである。その環境でしか使えないが。

PostgreSQLのPoint In Time Recoveryとの組み合わ せ

- 運用としてはもちろん、PostgreSQL側の設定はあるが主にデータベースクラスターディレクトリ（/usr/local/pgsql/data）をどうするかが問題となる。
  - データベースバックアップマーカーの設定
    SELECT PG_START_BACKUP('バックアップ識別子');
  - /usr/local/pgsql/data 以下（pg_xlogを除く）のバックアップ［要注意］
  - データベースバックアップマーカーの削除
    SELECT PG_STOP_BACKUP('バックアップ識別子');

PostgreSQLのPoint In Time Recoveryとの組み合わ せ

- データベースクラスターディレクトリ（/usr/local/pgsql/data）のバックアップが大変。
  - tarでバックアップすると更新中のファイルがあるとアーカイビングに失敗する。
  - rsyncでバックアップすると更新中のファイルがあるとrsyncでエラーになる（エラーコード 24)。

PostgreSQLのPoint In Time Recoveryとの組み合わ せ

- そこでスナップショットを取ってから悠々自適にバックアップを取りましょう。ということで。
  - zfs snapshot tank/pgsql@`date +%Y%m%d`
  - zfs send -i `date -v-1d +%Y%m%d` tank/pgsql@`date +%Y%m%d` > /var/backups/....
- またpg_xlogディレクトリを別ファイルシステムにしてしまえばバックアップの手間が軽減。
  - tar -cvf /var/backups/.... --one-file-system -C /usr/local pgsql

FreeBSDのCVSリポジトリの収納

- ただ普通に/home/ncvs以下をミラーするという話。
- ディレクトリの性質に応じて細かくファイルシステムを分けてみた。実際には下記の通りとなる。
  - zfs create -o mountpoint=/home/ncvs -o compression=gzip-9 -o atime=off tank/ncvs
  - zfs create tank/ncvs/doc
  - zfs create tank/ncvs/CVSROOT-doc
  - zfs create -o compression=off tank/ncvs/CVSROOT-doc/commitlogs
- 属性は下位のファイルシステムに引き継がれるため、ほとんどなにもしなくてよい。

FreeBSDのCVSリポジトリの収納

ファイルシステム	ファイルシステム属性	圧縮率
tank/ncvs	comp=gzip-9,atime=off	4.47x
tank/ncvs/CVSROOT-doc	comp=gzip-9,atime=off	3.96x
tank/ncvs/CVSROOT-doc/commitlogs	comp=none,atime=off	1.00x
tank/ncvs/CVSROOT-ports	comp=gzip-9,atime=off	4.35x
tank/ncvs/CVSROOT-ports/commitlogs	comp=none,atime=off	1.00x
tank/ncvs/CVSROOT-projects	comp=gzip-9,atime=off	4.13x
tank/ncvs/CVSROOT-projects/commitlogs	comp=none,atime=off	1.00x
tank/ncvs/CVSROOT-src	comp=gzip-9,atime=off	4.16x
tank/ncvs/CVSROOT-src/commitlogs	comp=none,atime=off	1.00x
tank/ncvs/doc	comp=gzip-9,atime=off	3.20x
tank/ncvs/ports	comp=gzip-9,atime=off	3.16x
tank/ncvs/projects	comp=gzip-9,atime=off	1.79x
tank/ncvs/root	comp=gzip-9,atime=off	1.00x
tank/ncvs/src	comp=gzip-9,atime=off	3.52x
tank/ncvs/www	comp=gzip-9,atime=off	2.11x

FreeBSDのCVSリポジトリの収納

- 1日1回スナップショットを取って（cron）、半年間運用している。
- /home/ncvs以下のディスク使用量が1.15GB。
- スナップショット分を含めても2.49GB。
- 一般にFreeBSDのCVSのリポジトリは2GB必要とされることからすると、実に優秀な結果であるかわかる。

FreeBSDのCVSリポジトリの収納

- FreeBSDのCVSリポジトリのような記録がメインの用途では積極的に圧縮していくのがよい。
- 最近流行のフォレンジックログの類では、ファイルシステムで透過的に圧縮が効くのでなおよい。
- この手の圧縮はでも追記型のログでは使いにくいと思われ。
- zip, lha等により圧縮されたファイルや画像イメージがたくさんあるファイルシステムでは圧縮しない方がいい。
- 小さなファイル（4KB未満）がたくさんあるファイルシステムの使用でも無意味（tank/ncvs/root の例）。

まとめ

- ZFSは使えるか？
  - ***Strong Buy!***
  - しかし、Solarisでは5年運用されてるかもしれないが、FreeBSDでは導入されてから所詮半年のファイルシステムである。実際現時点でメモリリークする問題もある。ファイルシステムは10年かけてやっと...という世界。
  - 忘れてはならないのは、書き込み可能なフルスペックのファイルシステムがFreeBSDに導入されたのは実に久しぶりである。
  - ちなみにpjd氏がcommitした2007年4月6日以降2ヶ月間に渡ってZFSの話がMLに出ない日はなかった:-）。

参考文献

- http://blogs.sun.com/yappri/entry/zfs_intro
- http://www.opensolaris.org/os/community/zfs/docs/zfs_last.pdf
- http://jp.sun.com/products/software/solaris/10/ds/zfs.html
- http://wiki.freebsd.org/ZFS
- http://wiki.freebsd.org/ZFSTuningGuide
- http://www.sun.com/bigadmin/hubs/multilingual/japanese/content.jsp
- http://people.freebsd.org/~pjd/misc/zfs/
- http://blog.ninth-nine.com/
- http://www.ninth-nine.com/presentation/Jus070926.pdf