

ORACLE®

最強プレイヤーが語る：
今まさに、Oracle Solaris 11!

ジャストプレイヤー株式会社
代表取締役 CEO兼CTO
瀧 康史



 #odddtky

日本オラクル、今年最大の技術トレーニング・イベント

**Oracle DBA &
Developer Day 2013**

Agenda

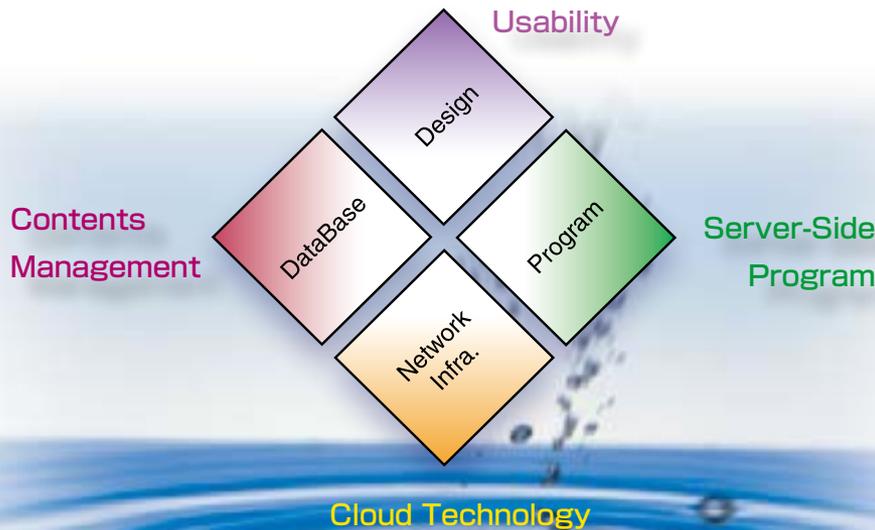
「サービス最適化のPDCAで、Solaris 11が貢献すること」

ジャストプレイヤー株式会社について



JUSTPLAYER → すぐに遊べるソフトウェアの開発

When You Want Is When You Play → 欲しい時が、遊ぶ時



事業内容

- クラウドインフラ賃貸事業、ホスティング賃貸事業
- クラウドソフトウェア制作事業
- WEBソフトウェア制作事業

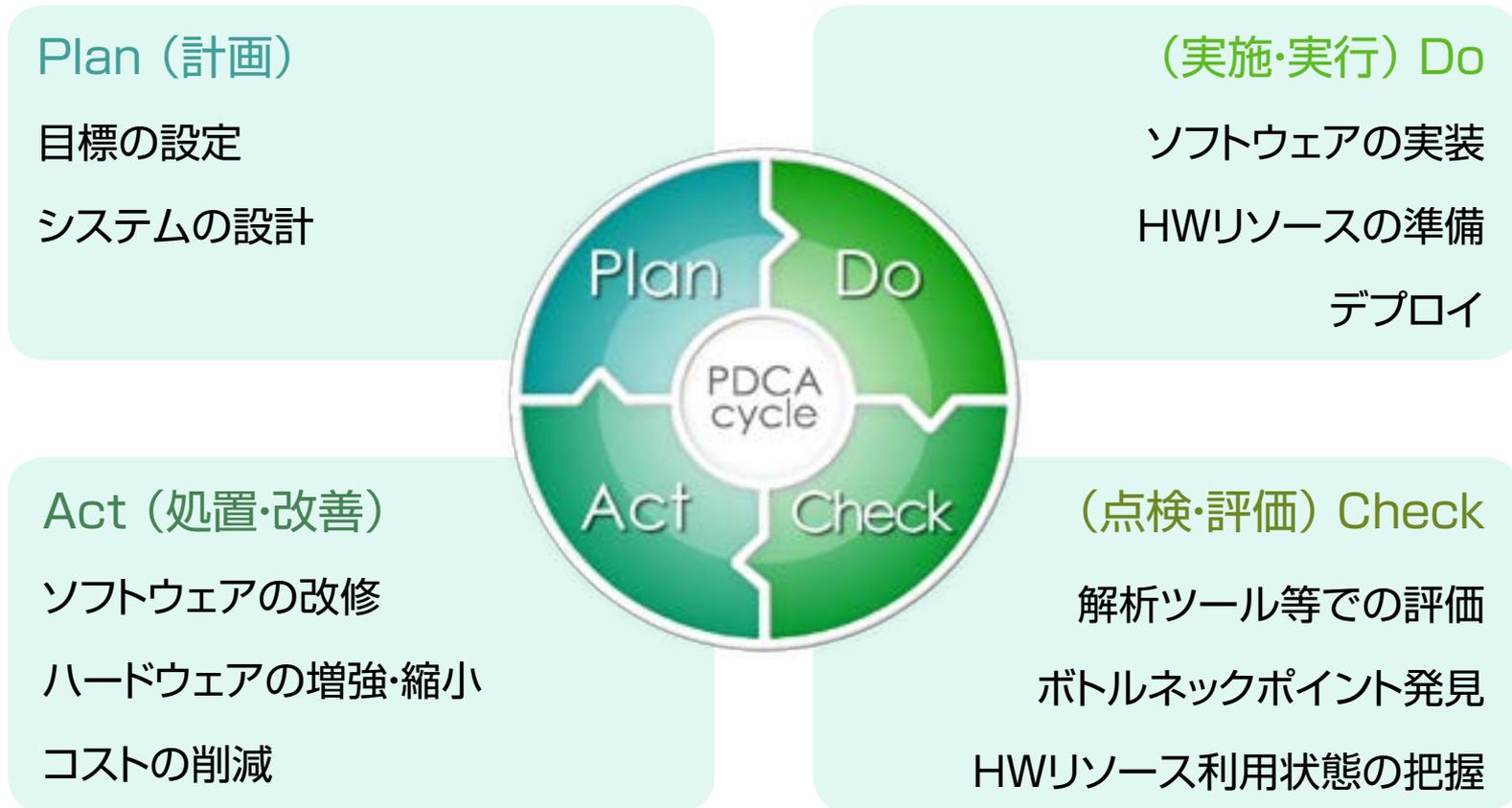
その他

- 第二種電気通信事業者届出 届出番号 C-18-1421
- 静岡県中小企業新事業活動促進法に基づく、経営革新計画の承認

まずは座学から

IT サービスの最適化

下記のサイクルを回して、サービスをどんどん最適化します。



サービス≡ソフトウェア+ITインフラストラクチャ+リソース+その他

良いサービスのゴール

ITサービスの設計者にとって良いサービスのゴールとは……

- 高速に動く
- 高い信頼性
- 安価である

時代が変わっても余り変化しないもの

- 高速とは、ユーザの待ち時間が少ないこと。
- 高い信頼性、サービスが安定的に稼働していること。

時代背景で異なるもの

- 安価とは…コストの構築要素が異なるため。

安価とは？

安価とはコストが安いこと。コストの構成要素

- 初期費用
- 維持費用(月額費用+年額費用など)

ただし、「安価」の判断が難しいのは、

- サービスの継続期間、トータルのコストが異なる。
- 企業にとって重要なのは「払いやすいか?」であって「TCO」ではない。
 - 例) 黒字の時の1千万と赤字の時期の1千万の違いには価値の違いがある。

安価にサービスを「運用」するには、何をすれば？

EoD ではない

開発コストを短縮する代わりに良いサーバを使うという考えは時代遅れになってきた。ソフトウェアチューニングに労力が再び必要に。

例) 専用サーバ時代とクラウドサーバ時代の変化

	専用サーバ時代	クラウドサーバ時代
初期コスト	開発費用(人月代)が高コスト。 EoDで開発期間短縮、遅くなる代わりに良いハードを買う。	開発費用(人月代)が高コスト。 月額HWリソースの極小化を!
維持コスト	コロケーション費用。ハードウェアがリースならリース代。	ランニングコストはHWリソースに比例して高まる。

サービスは「**運用**」される。どのぐらいの期間、使うのか?遅くてHWリソース消費の激しいソフトウェアは「事業のお荷物」になってしまう。

要するに……

速さは正義

テーマ1 (練習問題).
複数のウェブサービスを、
1台のマシンに集約したい。

*ここではまだSolarisは出てきません。

問題提起

背景

2002年、起業して1年の当社は、満を持してソフトの配信サービスを立ち上げたが全く売れなかった。

問題

- リリースしたばかりのサービスを終了できない
- サーバが「1台」も!!!占有されてしまう
 - 月額1万円の高コストを、どう維持するのか？

プラン

アイデア

- 同じサーバ上で別サービスを始める。
 - その結果、月額出費を軽減できる。

課題

- サービス利用顧客の隔離性をどう担保するか?
- 同居させたいもの
 - 自社配信システム
 - 顧客のウェブサイト

サービスを実現する単位

- 自社の配信システム
 - httpd、perlのCGI、顧客情報が入ったDB
- 顧客のウェブサイト
 - httpd、perlのCGI、phpプログラム、顧客情報が入ったDB

これを実現するユニット、セットはなんだろう？

1. プロセス、プロセスのグループ(SolarisではProjectやTask)
2. OSのユーザ・ランド
3. OSのカーネルのサービスモジュール
4. ハードウェアとOSのドライバ群

隔離性はどこまで必要か？

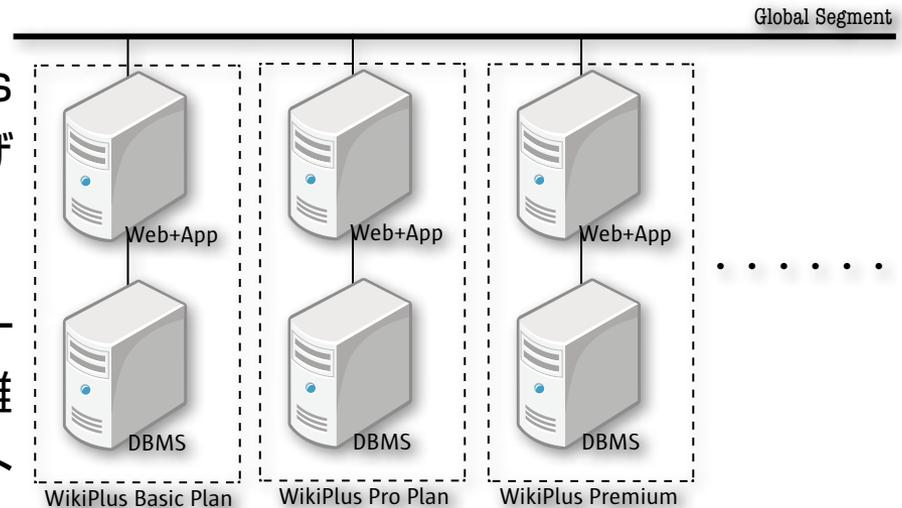
サーバの負荷	どちらも大した負荷はないため気にしないことにする。
CGIプログラム	相互汚染は避けたい。将来、顧客のウェブサイトに顧客自身がCGIを置いて、互いにプログラムやデータの参照をされたくない。 SuExecでプロセスをUIDにして分ければよいのでは？
PHPプログラム	mod_phpにしてApacheの権限で動かすと、全ての1つの権限で動いてしまう。 phpをcgiで動かし、ActionHandlerで呼び出せばよい
配信コンテンツ	公開ディレクトリ(DocumentRoot)は問題ないが、秘匿エリアは見られないようにしたい。 VirtualHost毎に別のUIDでDocumentRootにデプロイし、読み出しはApacheの動作権限(o+r)で構わないのでは？
データベース	互いに見られないようにしたい。DBにパスワード認証をつける。 ※ただしウェブの場合、プログラム中にパスワードを埋める傾向があるので、そこを参照できないようにする。permissionで可。

テーマ2.
新サービス毎に、
サーバが増える問題を
解決する

問題提起

背景

- 2005年に弊社ではWikiPlusというCMSサービスを立ち上げた。
- これにはApacheのVirtualHostで顧客単位を隔離する仕組みがあり、1セット (WEB+DB)のサーバで、複数の顧客に対応する機能を持っていた。



問題

- プラン毎にサーバセットが立ち上がるため、サービスプランやOEMプランが増える度にサーバが増えていった。
- リソースも増え、管理(監視やバックアップを含む)がどんどん大変に……

プラン

アイデア

- 同じサーバセットで別のシステムを収容する(前と同じ)
 - 仮想化して、別の顧客を収容しても、問題ない隔離性を確保する

課題

- 仮想化コストがどのようにかかるのか?
- 隔離性はどこまで担保されるのか?
- 社内で過去チャレンジした仮想化と比べてどうか?
 - VMware Workstation V3+NASでの仮想化
 - FreeBSD Jail
 - User Mode Linux

そもそもサーバ仮想化とは？

「ハードウェア資源と、
サーバ(サービスを実現する単位)を分離する」



「1台の物理サーバの上に複数の仮想サーバ」を動かせるベネフィットより、
「複数台の物理サーバの上に複数の仮想サーバ」を動かせる利便性。

仮想化のファイナンシャル・メリット

- リソースへの投資サイクルと、サービスのライフサイクルを分離できる。
 - 始めるまでどの規模のリソースが必要かわからないサービスは多い。
 - 時期によって繁盛記が異なるサービスもある。

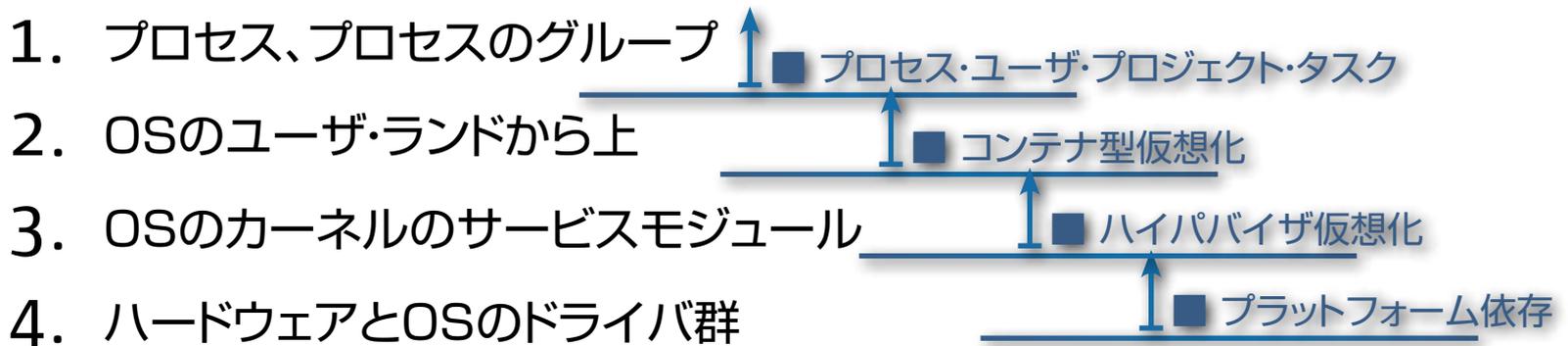
具体的には、

- ハードウェア資源は足りなくなったときに買えば良い。
- リースバックの時にサーバリプレースがスムーズ。
- サービスの開始が、ハードウェアの納期に縛られずに済む。
- サービスの縮退が、ファイナンシャルのサイクルに縛られずに済む。

どの層で隔離するか？

どの層で隔離するか？は、自分のサービスを収容する、必要な「サーバ」がどう
いう単位(隔離レベル)のものなのか？を考えることで決まる。

たとえば…



※上に行くほど、仮想化オーバーヘッド(仮想化によるリソース消費量)が下がる
(=高速)代わりに隔離レベルが落ちる。

仮想化製品の種類

ハイパバイザ型(Type I/II)

- Oracle VM Server
- VMware ESX Server
- Citrix Xen Server
- Redhat KVM
- Microsoft Hyper V
- Oracle VirtualBox

コンテナ型

- Oracle Solaris Zones
- Linux lxc/cgroups, OpenVZ
- FreeBSD jail

ハイパバイザ型のメリット

2つの大きなメリット

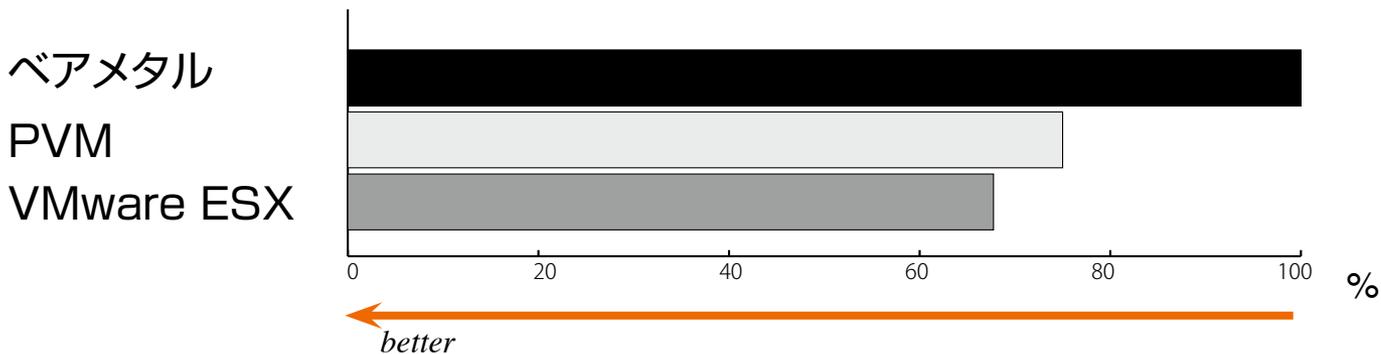
- 1つの物理機で、異なる種類のOSを動かすことができる
- ライブマイグレーションが可能
 - 仮想サーバ単位で、収容ホスト（物理機）間を稼働させたままを移動。
 - 現実問題として、「ハードが壊れ始めたら、その上のVMはもう信じられない」ので、壊れそうなら移動するなんてことを検出するのは実際は難しい。
 - ハイパバイザのメンテナンス時に逃がすというメリットの方が大きい。

ハイパバイザ型のデメリット

デメリット

- 1ゲストOSあたりのメモリ利用量が多い
- 仮想化オーバーヘッドが大きくなる時がある
 - あるWEBプログラムはVMware ESXで仮想化するだけで、同時リクエスト数がか7割程度まで落ち込むものがあった。

■ あるWEBプログラム(Java Servlet + 某RDB)の例



ハイパバイザ型の注意点

意外と製品毎に機能差が……

- パラ・バーチャリゼーション・ドライバの出来次第で速度が大きく変わる。
 - サーバだと特に、ネットワーク、ストレージのPVDは欲しい。
- 「ハイパバイザが動作保証するOS」と、「OSが動作保証するハイパバイザ」の**双方がきちんと整合**していないと、肝心なところでサポートが受けられない可能性もある。
- マイグレーション機能の差異はそれなりにある。
- スナップショットが使いにくい物も多い。
 - これはストレージ側で解決することもできる。

コンテナ型仮想化のメリット

- リソースを使わない
 - カーネルは共通。プロセスが本当に利用するメモリ、CPUだけでよい。
- パフォーマンスボトルネックがほとんど無い!
 - グローバルゾーンから見たら1つのプロセスに過ぎない。
 - 全体でのボトルネック解析がしやすい!
- ブートが速い
 - カーネルが起動しない分、ブート、リブートがとても速い



資源共有率が高いので、
1つのサーバにたくさん収容できる

コンテナ型のデメリット

ハイパバイザ型に比べて、次のようなデメリットがあります。

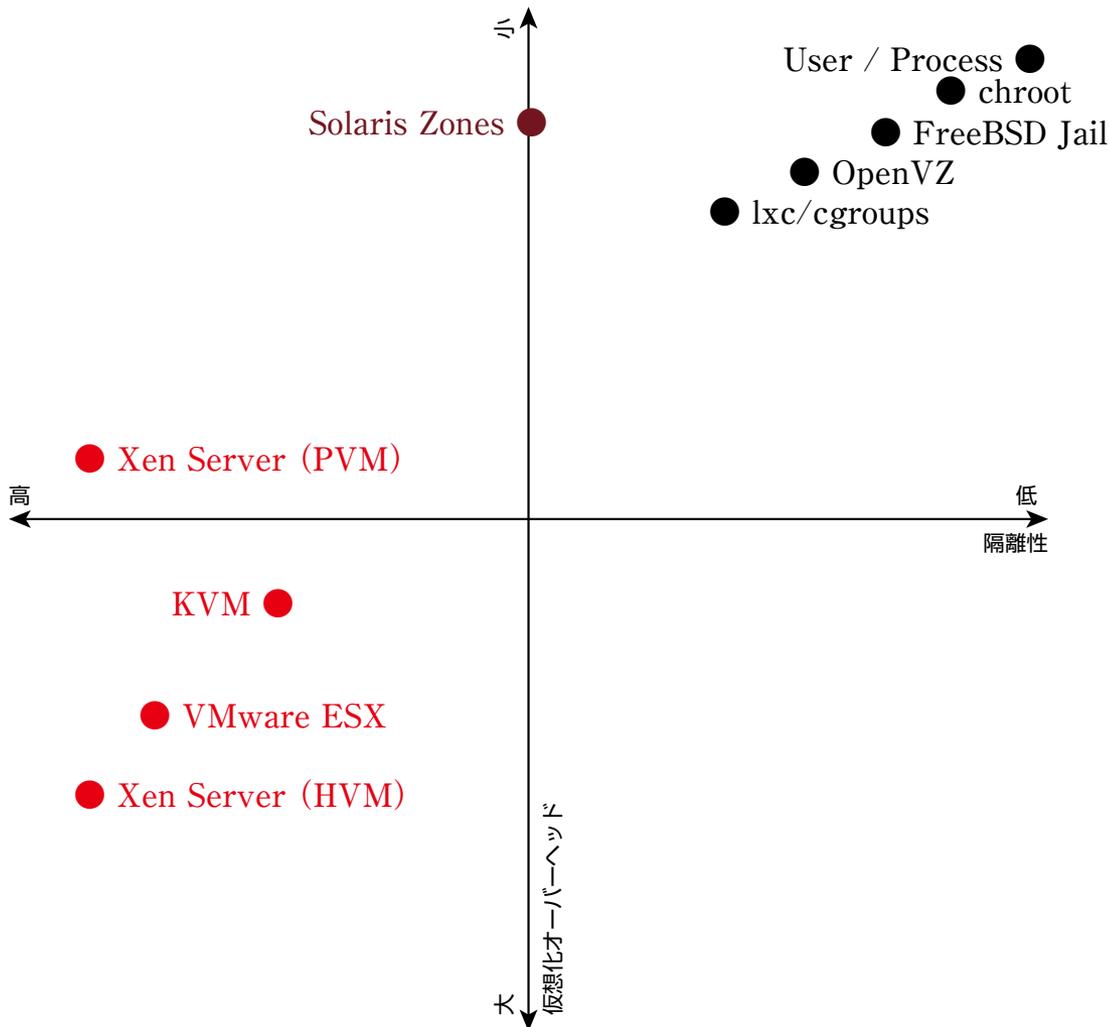
- 異なるOS、異なるバージョンのOSの動作は基本的には不可能
 - そのためにOSバージョンアップの依存関係が激しくできる。
- 互いの負荷影響を受けやすい(コンテナごとの隔離性が低いため)
- 一部に、動かないアプリケーションがある
- プロセス以外の仮想化が概ねない。ディスク、ネットワークなどなど…。
- ライブマイグレーションなどの機能がない

カーネルが共通で資源共有率が高いことによる裏返し

Oracle Solaris Zones の特徴

- 他の仮想サーバの負荷影響を受けにくい。
 - リソース管理 (CPU制限、メモリ制限、カーネルリソース制限) が強力。
- 動かないアプリがほとんどない!
 - OS標準の機能で、パッチ等の提供ではないため、ノングローバルゾーン (仮想サーバ) とグローバルゾーン (収容機) にほとんど差が無い。
 - その他仮想化機能 (ネットワーク、ストレージ、パッケージシステム) との親和性
- ネットワークスタックの委譲と仮想化
 - NAT-BOX、ファイルサーバ、L/B等々を作ることができる
- ファイルシステムの委譲と仮想化
 - ZFSによるファイルシステム単位の委譲。
- 旧バージョンのOSを(ある程度)混在可能
 - Solaris 10 Zones等。ただしSolaris11の異バージョン収容は不可。
- マイグレーション(別のマシンへの移設)が可能

仮想オーバーヘッドのイメージ図

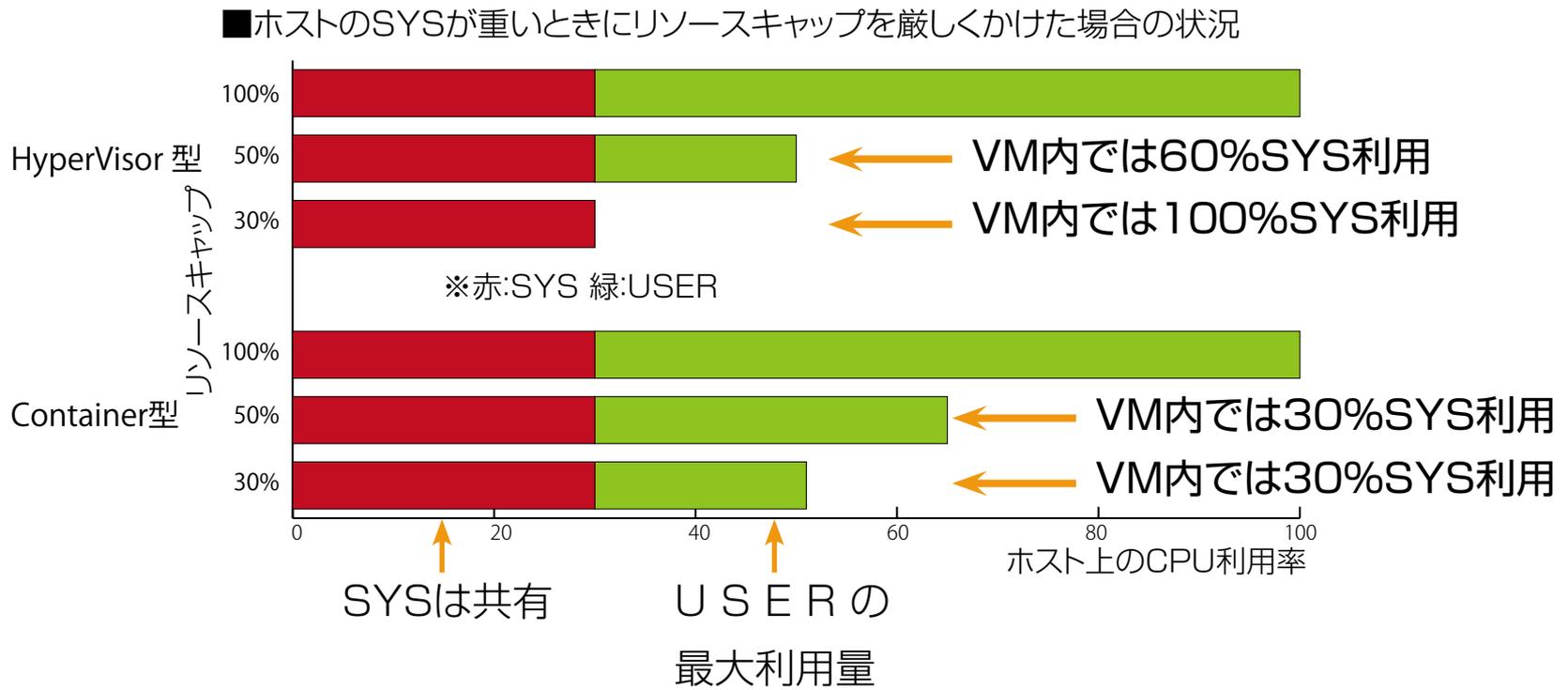


左図は、私の利用感による「イメージ」

- 左右や隔離性を意味し、左に行くほど隔離性が高い。
- 上下は仮想化によるオーバーヘッド量。
- 赤いものは、別OSを動作させることができる。

CPU 隔離性の性格の違い

ハイパーバイザ型とコンテナ型で、リソース制限時の臨界点挙動が大きく異なることに注意!



実際の導入 (ハイブリッド導入)

ハイブリッド構成でクラウド環境を作成

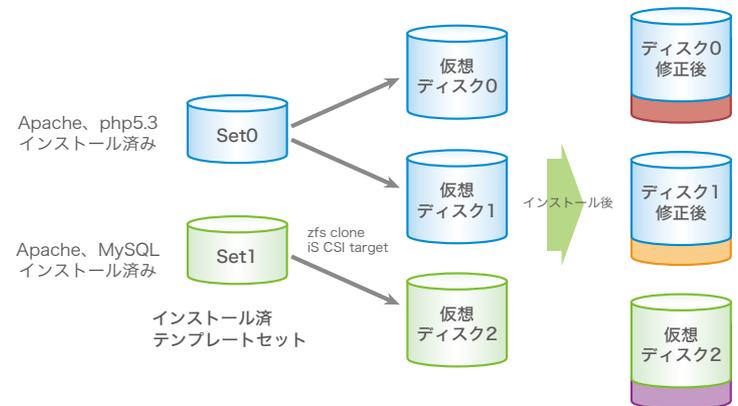
(下記は現在の状況であって、WikiPlusの最適化時のものではありません)

- Solaris 11 Zones
 - リソース利用率が高い=1サービスあたりのコストが安価なため。
- VMware ESX
 - 動作保証のOS、**被**動作保証ハイパバイザのカバレッジが広い。
- Oracle VM for SPARC
 - 隔離性の高さ。SPARC資産の継承ができるため。

Solaris 11のZoneをなるべく利用し、OS依存のアプリはVMwareに、CPU依存はOracle VM for SPARCで運用するスタイル。

ストレージの仮想化 (ZFS)

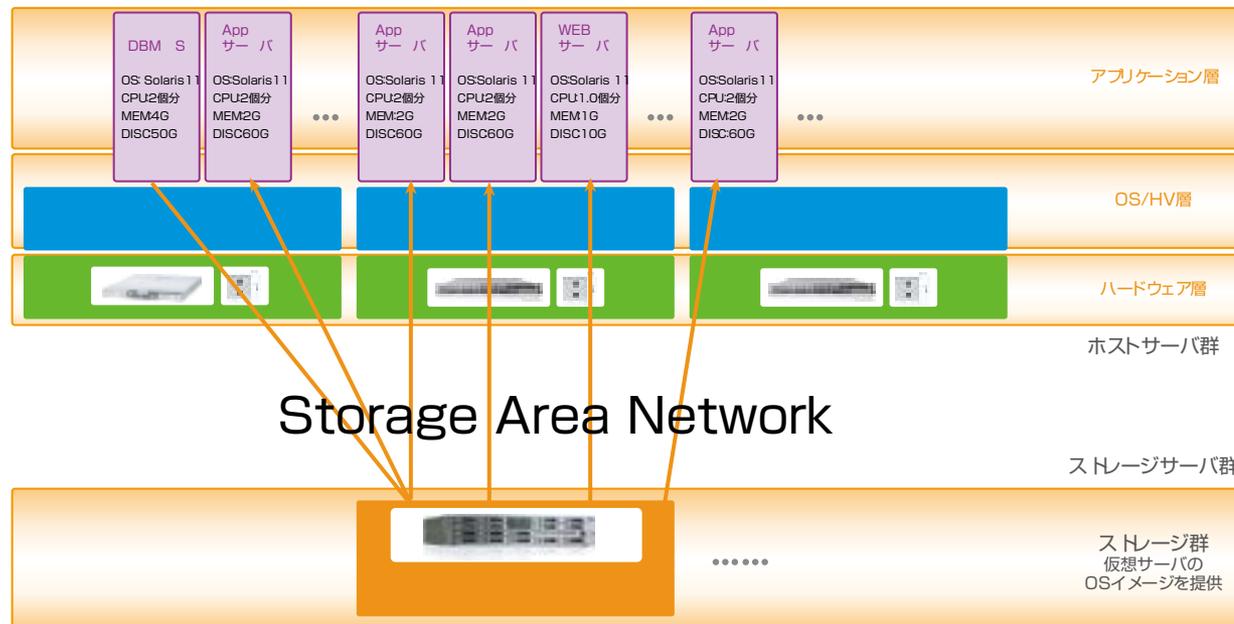
- ZFSによる管理
 - ZFSはツリー構造でデータセット (ファイルシステム) を管理している。
 - 仮想サーバ毎に、ZFSを委譲して、利用することが可能。
 - 仮想サーバ毎の利用量のCAPの制御。
 - スナップショット、ロールバック、クローンなどができ、仮想化と相性が良い。
 - スナップショット毎に、バックアップを簡単に取ることができる。
 - ただし、仮想サーバ毎のストレージのアクセス頻度管理は難しい (知る方法はあるが、制御は難しい)



非常に高価なアプライアンス機器と同じ機能を、PCサーバでも実現できる

SAN (COMSTAR) の利用

- COMSTARを利用した管理
 - iSCSI TARGET機能 (COMSTAR) 。ZFSと連携して、高価なDAS並の機能を標準で利用することができる。
 - フェイルセーフのための多重化も簡単にできる



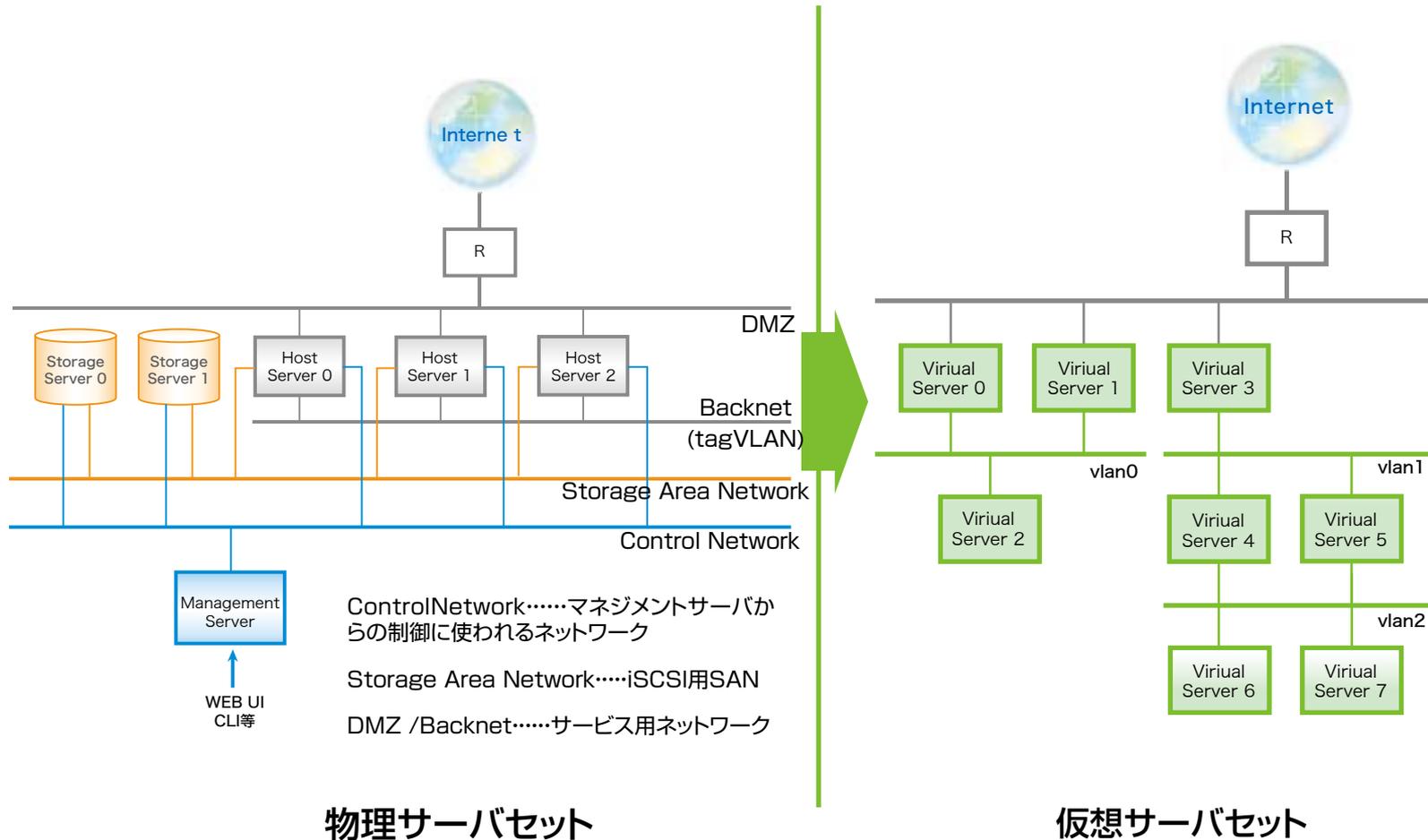
ネットワークの仮想化

- dladm(CROSSBOW等)を使った仮想ネットワークの管理
 - 仮想サーバ毎に、仮想NICを作成して委譲。
 - 仮想NICの帯域制御。
 - spoofingプロテクト。
 - CPU負荷の制御。
 - IPの固定制御。
 - 仮想Switch (Etherstub) の作成
 - 仮想Bridgeの作成。
 - VLAN、フィルタ、ルーティング等
 - 仮想サーバへのMACアドレスの貼り付け。
 - 帯域の測定、監視など。

仮想ネットワークとスイッチの
tagVLANを組み合わせることで、
全体でネットワークを仮想化

物理ネットワークと仮想ネットワーク

これらを、物理結線の直しや、スイッチの再起動なしに行う



得られたもの

- 1サービスあたりの収容の向上
 - 1サービス2サーバから、8サービス1サーバへ
 - →収容度16倍アップ
 - ※サーバスペックの向上をもちろん含みます
- 複数の異なるサービスの収容
 - 新事業展開の円滑化
 - WIKIPLUSベースのウェブサービスの水平展開など
- バックアップの容易化
 - ストレージサーバで一括バックアップ (zfs send + ssh) ができる
 - 以前は、1つづつrsync+sshでバックアップしていた。

後半に続く

後半では下記のテーマで話を進めます。

テーマ3

- VMサイズの極大化問題
- レスポンス速度の低下問題

テーマ4

- SANストレージの信頼性問題
- SANストレージの応答速度悪化問題
- ハイパーバイザへのアクセス不能問題

テーマ3. VMサイズの極大化問題と、 レスポンス速度低下問題。

問題提起

背景

WikiPlusは歴代の様々な担当者の手によって機能拡張され、バージョンアップしていった。

問題

- CMSサーバのVMが大きくなる問題が発生し、収容率の低下が問題に。
- またいつからかレスポンス速度が悪化し、遅いサービスになってしまった。

なぜ VM が巨大化したか？

インスタンス内で利用している主要プロセスがメモリを使うようになった。

具体的には…

- Apacheが特に巨大化する
 - prstatで見ると1インスタンス120MB程度
 - 再起動直後は20MB程度だが、phpのリクエストを受け付けると徐々に大きくなり、次第に120MB程度までふくれあがる。
 - 単純計算で120MB x MaxClientで設定した数のswap容量が必要に……

メモリ利用量を知る

- ForkしたApacheのインスタンスが個別に確保するメモリ
 - ForkなのでメモリはCoW。つまりForkした直後はメモリを使わない。
- mod_phpが動作すると、Apacheがメモリを確保する。
 - 一度確保したらhttpdのforkしたプロセスが終了するまで保持。
 - Forkして増えたhttpdは、MaxRequestPerChildの回数、リクエストを受け付けると、自動的に終了してメモリ解放（ただしデフォルトは無制限）
 - リクエストの波が収まり、プロセス数がシュリンクする時に減る（最近のapacheは割と減らすのは早い）

下記のコマンドで知ることができる

```
prstat -s size
```

```
prstat -s swap
```

Fork プロセスのメモリ使用量

物理メモリ利用量＝

Forkする前のメモリ量(全てのForkプロセスの共通エリア)
＋(Fork後に確保、変更したメモリ)×インスタンス数(MaxClient)

SWAP利用量＝

Forkしたプロセス全てのメモリ×インスタンス数(MaxClient)

わかること

- Fork型のプロセスは、メモリをととても食う。
 - つまり沢山のアクセスには向かない。大量アクセスにはスレッド型
- ほぼ「確保されるだけ」のSWAPエリアが、preforkされるシステムには必須
 - Solaris Zone用のマシンには、容量の大きなブート用SSDが必須に。

PHP の動作方法を知る

- mod_php
 - preforkしたapache上で動く (mod_phpはworkerでは事実上動かない)
 - メリット: Apacheからリクエストが同一プロセス上で飛ぶので、速い。
 - デメリット: メモリ効率がめちゃめちゃ悪い
- CGI/FastCGI/FPM
 - 呼び出しは遅い (FastCGIなら許せるかなあ?)
 - メモリ効率は良い。

解決策

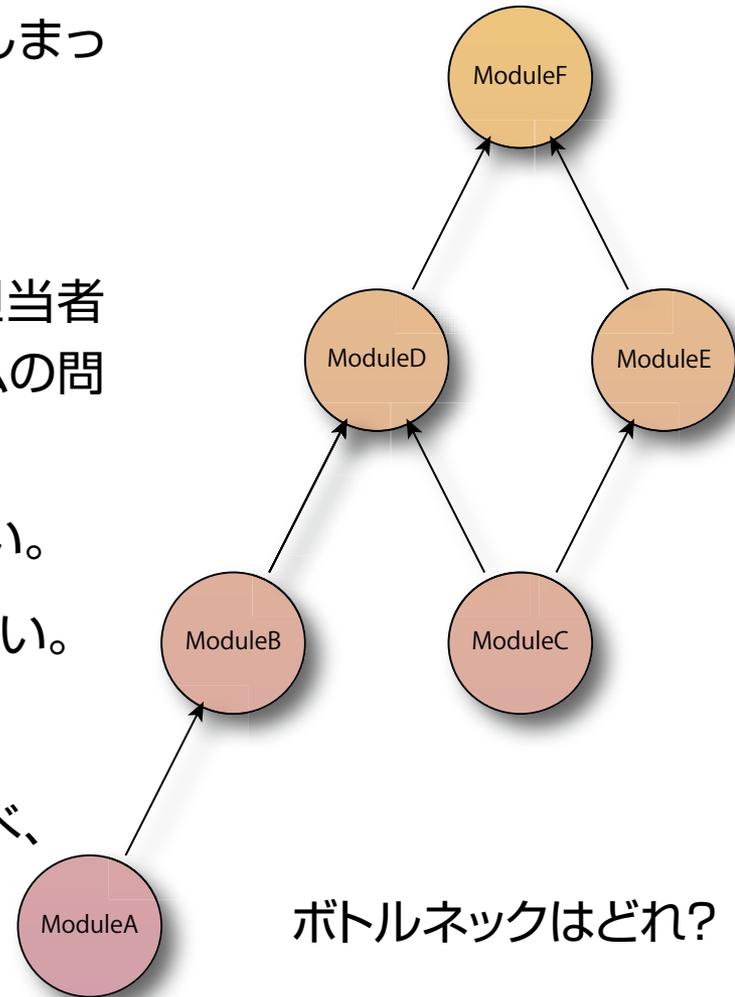
- Apacheはworkerで実行する(多量アクセスが可能)
- phpはFastCGIを利用して実行
 - ウェブ1ページに必要なphpの実行は平均して1。同時アクセスは6。

プログラムのボトルネックを探る

いつの頃か、プログラムが遅くなってしまった。

- 接ぎ木されたプログラム
 - バージョンアップを続け、さらに担当者が変わり、接ぎ木されたプログラムの問題。
 - カプセル化されると更に分からない。
 - 通常、動いてるモジュールは触らない。

プログラムの本当のボトルネックをしらべ、「遅い関数」を見つけ出す必要がある。



ボトルネックはどれ？

DTrace を利用した調査

インストール方法

```
pkg install -v dtrace/toolkit
```

phpの遅い関数を調べる

- php_calltime/php_calldist/php_funccall/php_cputime/php_cpudist
 - php関数毎の処理時間等を測定する。php_calltime.dが、スーパーセット。exclusive/inclusive/calltimeから比較可能。
 - dist系は表示違い。本番機では表示が多すぎて見るのがむしろ大変。
 - php_cpu*は、CPU処理時間だけを計測。
 - php_funccallは呼び出し回数だけ。

DTrace を利用する

関数の呼び出しの流れを調べる

- php_flow/php_flowinfo/php_flowtime
 - 現在実行されているphp関数のsnoop (本番機でやるととても重いので注意)
 - flowでは関数が、flowinfoでは呼ばれていく順が、flowtimeでは処理時間が表示

多量のリソースを利用していないか?

- php_malloc/php_syscall/php_syscolor
 - libc経由のmalloc状況や、syscallの発行状況を見る。

重いVirtualHostはいないか?

- php_who
 - 実行されているPID、UID、ファイルをみる (どのVirtualHost?)

得られたもの

- どのロジック(≡関数、クラス)がメモリを食っていたのかわかる。
 - メモリの消費の激しいロジックを見直し、場合によっては外部プロセス化することも。
 - 結果、平均メモリ使用量が80MB→20MB程度まで縮小。1/4の高集積化
- どのロジックが実行時間を使っていたかがわかる。
 - 遅い関数の最適化。キャッシュなどの仕組みを入れることも。
 - ロジックの見直し。リアルタイム性がいらぬ遅い部分をなるべく再構築するタイミングを減らす。
 - 結果、応答時間が3000msecから100msec未満まで短縮。1/30。

※4倍の集積度で、30倍のレスポンス速度により、1分間の最大処理リクエスト数が、**30x4=120倍**に向上。仮想化前に比べると、さらに16倍。1,920倍の高収容化。単純計算で1リクエストの費用が**1/1920**に!

テーマ4.
SANストレージの信頼性と、
応答速度悪化問題
ハイパバイザの
アクセス不能問題

問題提起

背景

- 仮想化により、サービスの数が増え、様々な顧客が収容されることになった。

問題

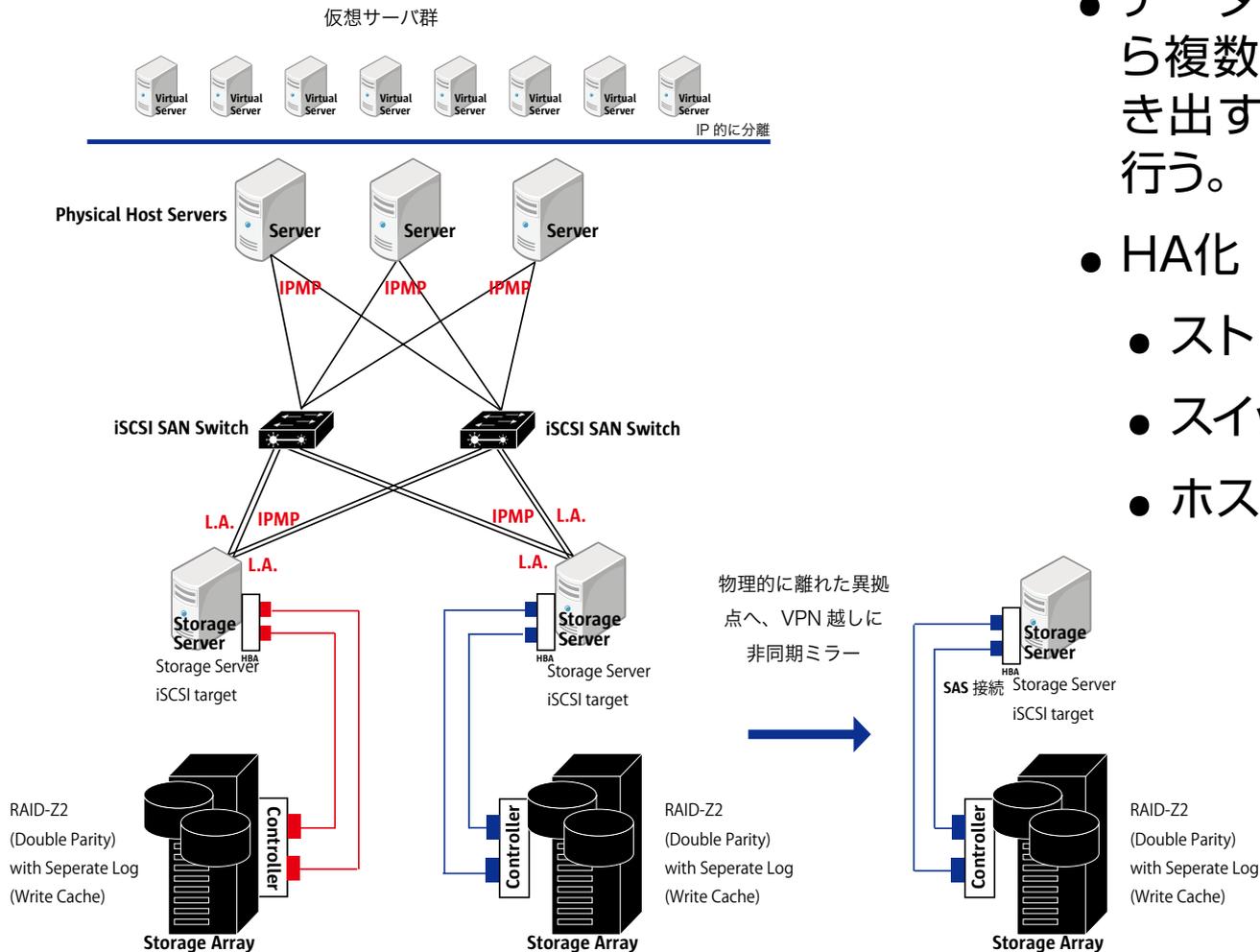
- 1つのSANストレージが収容するサービスの量が増大化することで、このストレージがダウンしたときのインパクトが甚大化した。
- SANストレージのダウンだけでなく、応答不良により、連鎖してダウンするサービスが目立つようになってきた。安定性の向上を図りたい。
- SANストレージに対するオペレーションミスに耐性を持たせたい。
- 稀にハイパーバイザーやGlobalZoneにアクセスできないシチュエーションが発生するようになってきた。この原因を突き詰めた。

ストレージサーバの多重化



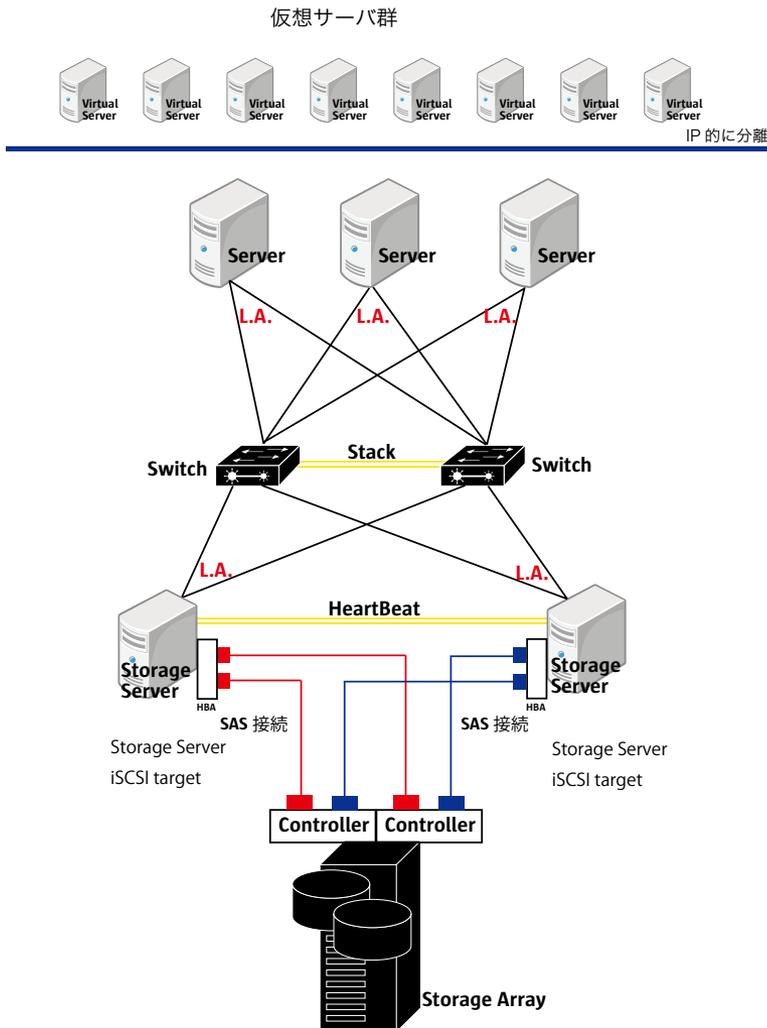
- ストレージサーバには**ダブルパリティのストレージアレイ**を利用。
- 冗長化されたストレージを更に**ホストがストレージサーバをミラーで束ね**、どちらか1系統でストレージ障害(ハード、オペレーションミスなど)が発生しても、データ欠損から守れるように作る。
- ホストサーバとストレージサーバの**両方でスナップショット**を作成し、ホストサーバ上でもストレージサーバ上でもオペレーションミスによる破壊を回避。
- ストレージサーバの内容は、**地理的に離れた別拠点DCに定期的に転送**され、万一、致命的な災害などが発生してプライマリデータセンターが破壊されたとしてもデータは守るようにする。

実際の設計



- データを一つのホストから複数のストレージに書き出すことで、冗長化を行う。
- HA化
 - ストレージ
 - スイッチ
 - ホスト

不採用だった案



メリット

- ストレージが1系統ですむ。
- iSCSI Initiatorが特に何かしなくても、TARGETが冗長化される。

デメリット

- 1つのストレージ破壊でバックアップからの救出が必要になる=数時間のデータロス。

問題提起

問題

- ディスク・フェイルが頻繁に発生し、RAIDがDEGRADEする
 - DEGRADEの瞬間、高い確率でストレージサーバが分単位で停止する
 - 応答速度が悪化し、連鎖して、仮想サーバ系が全てダウンする
 - いわゆる大規模障害の発生

OS毎のフォールトの癖

- Linuxは、Read Only Filesystemになることが多い。
- Solarisは、仮想サーバに見せているzfsがDEGRADEする。
 - かつてはこのDEGRADE時に停止して、サービス全てが全滅したりした。
- 意外とWindowsはがんばるようだ。

実際のストレージサーバの構成例

- OS : Solaris11.1SRU
- RAID Card: LSI SAS 9211(LSI SAS2008)
- Driver: mpt_sas
- SAS Expander: LSI SAS Expander 2x28
- HDD: Seagate Nearline SATAII
- Cache SSD: Intel 520 x 2

ディスク構成

- ディスクは1本ずつOSに見せるJBOD構成
- 2TB 7本でRAIDZ2、これをストライプで束ね、スペアを追加。
- SSDを2系統Separate Logと、L2ARCとして容易

発生する問題

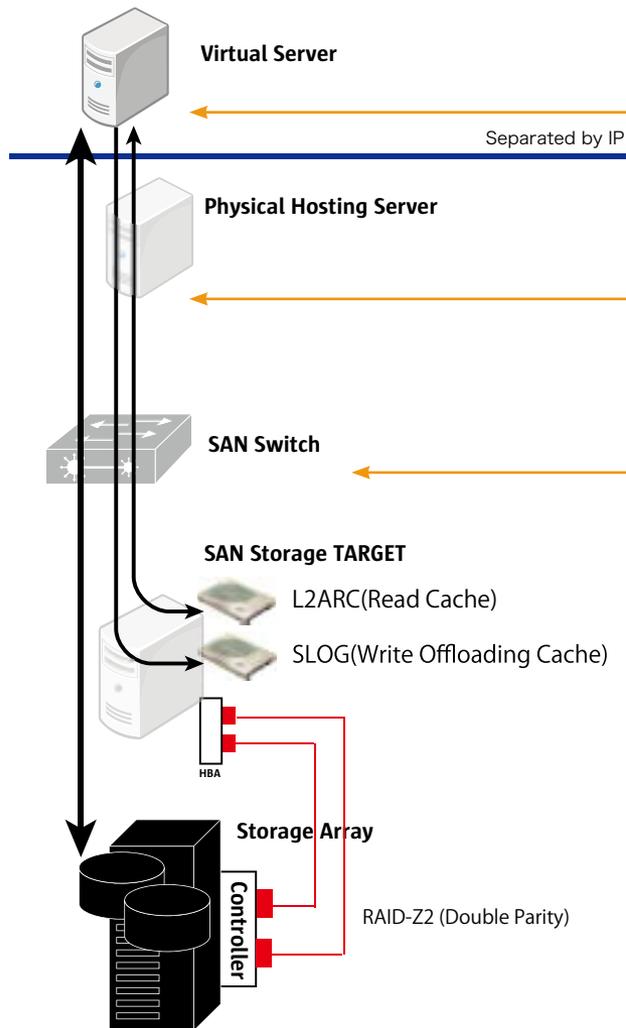
問題点

- IO速度が足りなくなる問題
 - iostat -Ynで%bが100で張り付く。
 - 1秒間隔で張り付く程度なら構わないが、5秒間隔平均で90%ぐらいになるとディスク操作がまったく不能になる。
- ディスクのフェイル問題
 - iostat -Enで、Software Errorがでて、RAIDが頻繁にデグレードする

原因の想定

- ディスクとストレージ全体の処理速度(IOPS)不足では無いか?

ボトルネックはどこにあるのか？



`iostat -Yn デバイス名 1`
※当該デバイスの**%b**がどの程度あるのか？

`dlstat / dladm show-link / fsstat` など
※帯域は足りているのか？

switchの帯域の飽和状態などを調べる。

`zpool iostat -v 1`
※キャッシュ用SSDの読み書きはどの程度発生しているか？

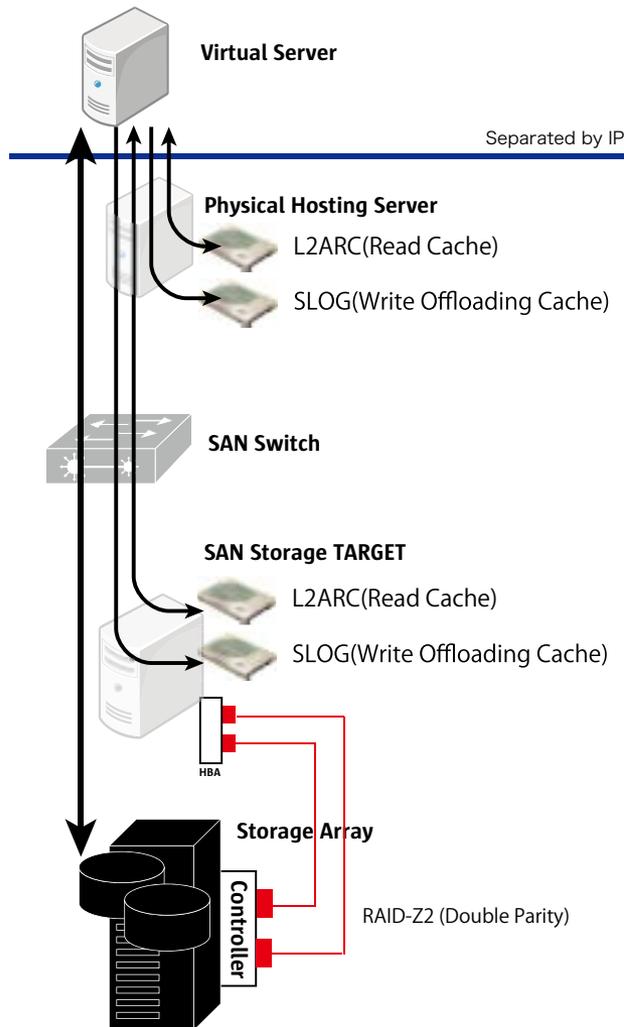
`iostat -Yn デバイス名 1`
※キャッシュ用SSDの**%b**、ディスクの**%b**のバランスは？

`kstat -n arcstats`
※ARCバッファのヒット状態を見る。`arcstat.pl`、`arc_summary.pl`などを入れると見やすい。
※`c(target size)/mru/mfu`等

改善策の案

- Cache用のSSDをつける。L2ARC、ZILの効果は？
 - さらに、試しに2レイヤーでSSDをキャッシュとして投入する。
- RAIDカードを変更してはどうか？
 - ディスクがフェイルするのはRAIDカードの問題か？
- RAIDZ2やめて、ハードRAIDのRAID10にするのはどうか？
 - RAIDZ2とハードRAIDの考察をする。
- ディスクを変えてみるのはどうか？
 - ディスクの種類を変えてみるとどうなるか？

2レイヤーで SSD Cache をつけてみる



収容ホスト(物理サーバ)上で作られたZFSに、L2ARCとSLOGを設定する。

想定

- リード、ライトオフロードがかかるので、ストレージへの負荷が減るのでは無いか?
- またSSDがヒットすると高速にならないか?

結果

- 効果が無いことはないが期待通りではない
- ネットワークは飽和していない
- 物理サーバのARCは飽和していない。
- ライトはストレージのSLOGを軽減したにすぎない。そしてこれは飽和していない。
- 作るのが難しい。zvolでL2ARCとZILを作っては絶対にいけない。

そもそもどの性能がたりないのか？

1. iostatでみると、%bが飽和。ディスクのIOPSが足りない？
2. どのIOPSが足りない？
 - リード
 - メモリ上のARCバッファは十分。mfuのヒット率高め。実効容量の1/200程度
 - L2ARCも十分。実効容量の1/10程度。
 - キャッシュミスヒットでもReadはストライプしている
 - ライト
 - ZILトレイン以外の書き込みはSLOGに書込。SLOGのSSDは未飽和。
3. ライトランザクションで飽和しているのではないか？
 - iostatをもう一度、よく見るとwrite量が多い。
 - RAIDZ2のライト速度は、束ねた1本分のIOPSしかでない。
 - $16本 = 2spares + Stripe(RAIDZ2(7本) \times 2)$
 - ライトIOPSは1本のディスクの2倍。容量は全体の62.5%。

RAID10でいいのではないか？

仮想化用ストレージへの要求

ライトIOPS問題

- リードはキャッシュするが、ライトは大したキャッシュしない。
- 仮想サーバのOSが一定の時間、IOの応答がないと、関連する様々な物がフエイルする。
 - ファイルサーバは比較的「遅くても動けば良い」ものだったが、仮想化用ストレージは、一定の水準を下回ると事実上、障害となる。
 - デグレード中のリシルバークやリビルドが、障害を誘発する。

ディスク容量で枯渇するよりも先にライトIOPSが枯渇

ディスクがフェイルする理由

ZFS
SD
(SCSI Disk Driver)
mpt_sas
(LSI Logic Driver)
RAID CARD LSI 2008
SAS Expander LSI 2x28
Disks :
Seagate Nearline SATAII Disk

SD層からみたエラーは、`iostat -En`で出力することができる。

Soft Error: SD層でリトライした結果、成功した場合カウントされる。

Hard Error: ハードウェア（多分HBA層）でリトライをした結果がSD層に伝えられたもの。

Transport Error: SD層からみて反応がなく、HBAが落としたかディスクが落としたか、確証がないもの。

フェイルするシステムは、Transport Errorが出てUNAVAILする。

つまり、HBAかディスクかどちらかが問題。

HBA かディスクか？

- ディスク
 - UNAVAILになったディスクは、その後テストをすると無事に完了する
 - HBAの問題
 - HBAとSAS Expanderは他のシステムにおいて実績があるもの
- 対策
- RAIDカード交換
 - MegaRAID SAS 2108に交換。VirtualDriveをJBODで1本ずつ。
 - 結果：ZFSからみて、UNAVAILがREMOVEDになる。つまりディスクがいなくなる。
 - ディスク交換
 - Nearline SATAIIを、Nearline SASに交換する。
 - 結果：問題がほぼ起きなくなった。

SATAディスクの応答が無くなることで、切り離しに失敗するのでは無いか？

Solaris 11 の ZFS SAN Storage の作り方

鉄則

- 仮想化用のストレージにする場合、ライトのIOPSを稼ぐ。
- Nearline SATA II を使う場合は、RAIDZ2をやめて、RAIDカードのRAID 10を利用するとよい。
- NearlineのDriveは容量を使い切れない。先にIOPSが飽和するため。
- 2.5inchのOnline SASを利用し、スピンドルの数を増やし、ライトのIOPSを稼ぐ。
- L2ARC用のSSDは実効容量の1/10程度がオススメ。複数台あると良い。
- ARCバッファは1/500～1/100程度がオススメ。
- SLOG用のSSDはARCバッファの半分程度しか使わない。

高負荷時のハイパバイザのアクセス不能問題

問題の現象例

- Global zoneにログイン出来ない(SSH、コンソール)
- コントロール系のツールが応答しない

※pingは通る

問題が発生する直前の状態

- vmstatでSYSが増大しているが、prstat -mLでも、SYSを使っているプロセスは存在しない。
- vmstatでメモリは沢山利用していることはわかるが、prstat -s swapでも、対して使っている物が無い。

解決方法

問題

- 仮想Network Switch、仮想Storageが、CPUやIOを飽和させている。
- これらはカーネルで動作するため、わかりにくい。

解決方法

- カーネルに十分なリソースを割り振る。

Oracle VM for SPARCの場合

- IO用を司るドメインに十分なリソースを割り振る。
- IOドメインとコントロールドメインを別にする。

※x64のハイパバイザの場合は余り良い方法はない

Zoneの場合

- Zoneを動作するリソースプールを隔離する。実際のリソースよりも減らし、Zoneはその中で動作させることで、GlobalZoneの居場所をつくる。

最適化の PDCA で Solaris11 が貢献すること

- 徹底的な「サービス状態の見える化」によって、今、「本番サーバで起きていること」がリアルタイムにわかる。
- ボトルネックの解決のための施策がある。
- 様々なレイヤーと様々な仮想化が技術が秀逸である。

ソフトウェアをいかに「サービス」にし、お客様に「快適なクラウドサービス」として利用していただけるかは、エンジニアの手腕次第です。

Thanks

ジャストプレイヤー株式会社

代表取締役CEO兼CTO

瀧 康史