IP Meeting '97 Tutorial

WWWサーバ管理

篠田陽一
北陸先端科学技術大学院大学
WIDEプロジェクト

WIDE

JAIST

チュートリアルの目標:

- 安定したWebサイトの運用方法
  - 十分な要求処理能力
  - 障害に対する用意が重要であること
- Webを取り巻く末期的な状況の理解と対策
  - スケールしないWebをスケールさせるための方法
    - 現在のWebはスケールしていない
  - サービスの平等性を提供するための方法
    - 遠いWebは遅い
    - 不平等なサービスは、インターネットの精神に合っているか?
  - Webのトラフィックのインパクトを減らす方法

WIDE JAIST

Things to cover

- Webサイトプランニング
- Webサイトの運用
- Webサイト実例
- 未来系のWebサイト
- Caching Proxyに関する話題

WIDE

JAIST

Things NOT covered

- 特定のシステム/サーバの設定
  - How to execute Perl Scripts as CGI binaries on Windows NT
  - How to avoid access to hidden pages
  - ... And such
- クールなページの作り方

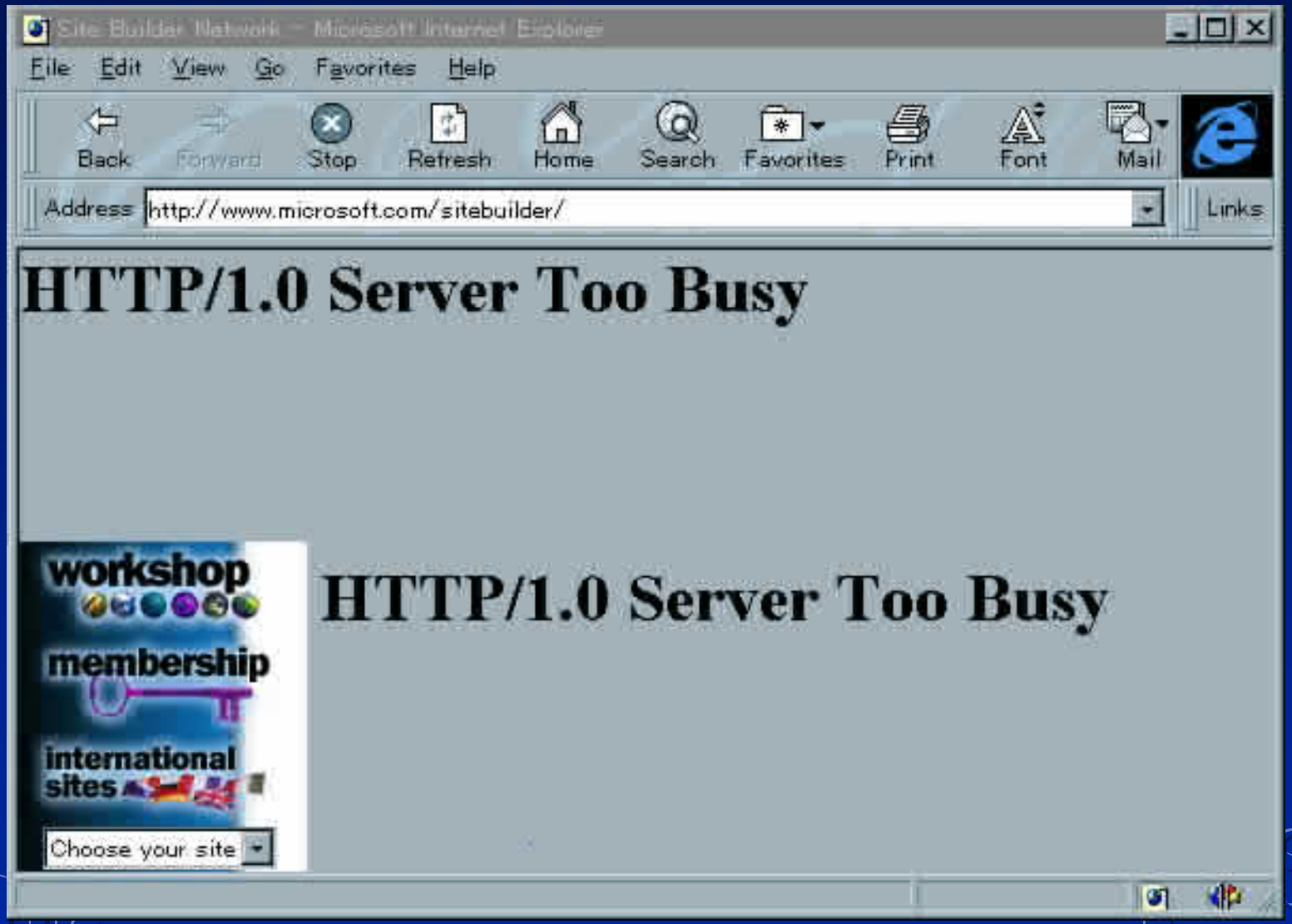
名ばかりの分散システム

- 今のWebシステムは、実は単なるTSSシステム?
  - ある特定のサービスや機能(ドキュメント)は、単一のシステムでしか提供されない。
  - URLにホストが埋め込まれているのが証拠
- 分散システムを安直な方法で構成しすぎた
  - 伝統的なインターネットのアプリケーションで、Webほど特定のサービスが単一点で提供されるものはない。
    - DNS, NetNews, Mail, ...
- とはいっても…

もう止まらない...Webをとりまく状況

- 増加し続ける:
  - Web上のアプリケーション
  - インターネットのユーザ
- ファンシーなインターフェースがサーバの負荷に:
  - CGI, SSI, SHTML, ...
  - クライアント・サイド処理は遅々として普及しない
    - Java, JavaScript, ActiveX, Dynamic HTML, ...
    - 不安定なブラウザ
    - 標準の欠如
- 高速で安定した安価なサイトへの要求
- サイトアーキテクチャに関する系統的な考察が必要

避けるべき状況...


Site Builder Network - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Print Font Mail
Address http://www.microsoft.com/sitebuilder/
Links
HTTP/1.0 Server Too Busy
workshop
membership
international sites
Choose your site
HTTP/1.0 Server Too Busy

WWWサーバ：基本中の基本

- GET methodのみを処理する簡単なWebサーバ
  - コネクションを受け付ける
  - リクエストラインを入力し、URI（パス名）を取り出す
  - 適当なステータスラインとヘッダを出力する
  - URIで指定されたオブジェクトを出力する
  - 掃除をしてから最初に戻る
- もっともコンパクトなサーバの大きさはどれくらいか
  - TinyHTTPD: HTTP/1.0, GETとPOSTのみを扱える
  - Perlで197行
  - `http://www.inka.de/~bigred/sw/`

実用サーバの機能

- URLリダイレクション (Remote Redirection)
  - 他のサーバへ振る
- URI 書き換え (Aliasing)
  - 自身の中で振る
- アクセス制御
  - ACL (ホスト/ネットワーク×URI)
  - User Authentication
- Virtual Hosting (Virtual Server)

実用サーバのプロセス構造

- 基本的な構造
  - シングルプロセス
  - コネクションごとにプロセスをforkする
  - あらかじめ複数のプロセスをforkしておく
  - 上記+足りなくなったら新しいプロセスをforkする
- お得な拡張
  - マルチスレッドプロセス
  - 入出力マルチプレキシング・非同期入出力
  - 出力ヘルパー (e.g.: Squid in http-accelerator mode)

サイトプランニング: アクセスプロファイル (1)

- アクセスプロファイル
  - リクエストの到着率分布
  - ユーザのバンド幅の分布
  - リクエスト種別の分布
- Webサーバ以外のシステムアクティビティの量
  - 補助 (あるいはメイン) プロセスの量
    - CGI
    - データベースアクセス
    - ...
  - DNS
  - 経路制御
  - etc., etc., ...

サイトプランニング: アクセスプロファイル (2)

- リクエストの到着率分布
  - 短時間(数分から数十分) 平均 v.s. 瞬間最大値
  - Webサイトの運用ポリシー
    - すべてのリクエストを尊重する: 瞬間最大値を意識
      - 完全を追求することは難しい
    - コストパーフォーマンスを尊重する: 短時間平均を意識
      - CPを追求しすぎるとカタストロフィックな状況をまねく可能性あり
  - 提供する情報のタイムリネスとユーザの挙動
    - コンテントの作り方にも考慮の余地あり
    - IWE96での失敗例:
      - 状況: コンサート実況中継イベント
      - 発端: 直前になっても実況中継システムの用意が完了せず、事前に案内したページには「ただいま準備中です」とのみ表示
      - 結果: 「準備中」が解除されることを待つユーザの連続アクセス

サイトプランニング: アクセスプロファイル (3)

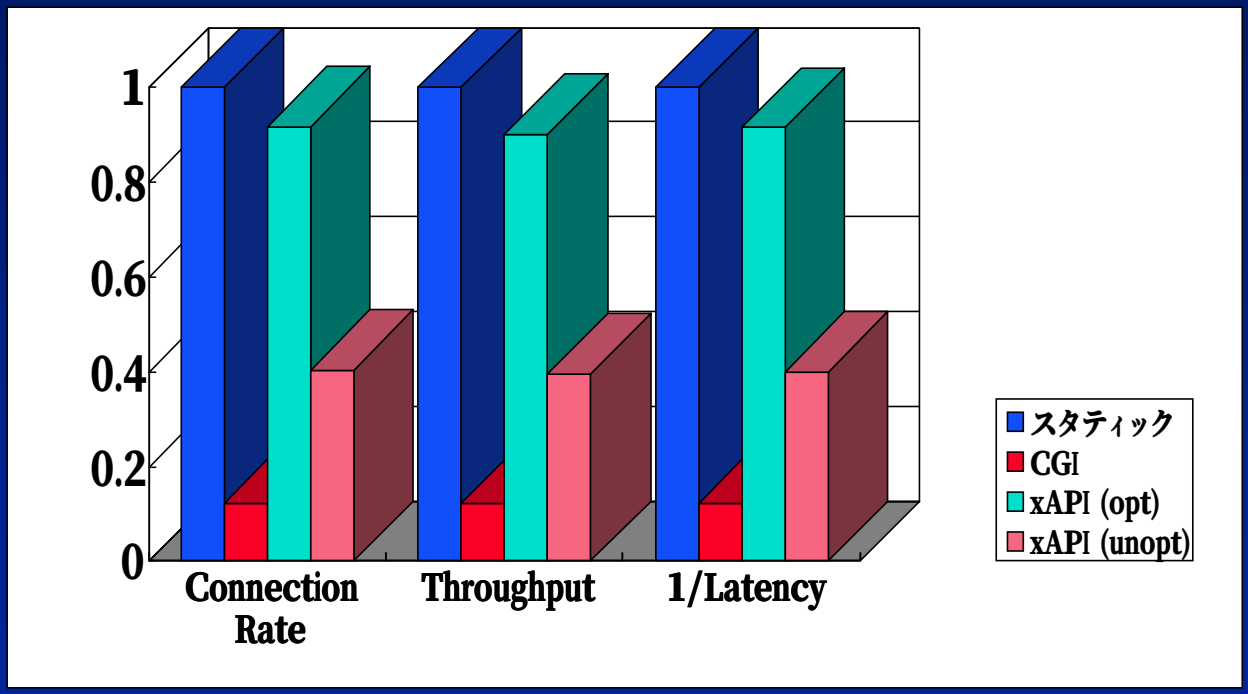
- ユーザのバンド幅の分布
  - LAN / WAN: 28.8Kbps ～ 10Mbps ～ ...
- 遅いリンクを通したアクセスはサイト能力の致命的なボトルネックになりうる
  - 平均的なドキュメントサイズを5KBと仮定
  - 平均的な実効転送速度を1KB/secと仮定
  - 1リクエストあたりの処理時間 ＝ 5KB / (1KB/sec) = 5 sec
  - 毎秒平均100リクエストを処理するためには500コネクションを同時に処理できる能力が必要
    - 逆にいうと、100コネクションの能力なら20リクエスト

サイトプランニング: アクセスプロファイル (4)

- リクエスト種別の分布
  - スタティックドキュメント
    - ファイルセット
    - ファイルサイズの分布
    - 平均値
  - ダイナミックドキュメント
    - プランニングでもっとも難しい問題

サイトプランニング: ダイナミックドキュメントのインパクト

- ダイナミックドキュメントはサーバのパフォーマンスにダイレクトに影響する
  - 普通のCGIはスタティックドキュメントの最低10倍の負荷



http://www.bsdi.com/products/internet-server/benchmarks/webperfより、
BSDI/16クライアントの場合の値をスタティックドキュメントを１として正規化

サイトプランニング: ダイナミックドキュメントのインパクト (2)

- ダイナミックドキュメントのインパクト評価
  - RPC/サーバ・クライアントシステムに関する知識が必要
  - サービスそのものの負荷
    - データベースアクセス等
  - Webサーバプロセスに対する負荷: サービスのプロファイル
    - 所要時間 (duration)
    - 遅延 (latency)

サイトプランニング: ダイナミックドキュメントのインパクト(3)

- ダイナミックドキュメントのインパクト例:
  - スタティックファイルの取得は高速に行えるのに、ダイナミックドキュメントの取得は果てしない時間を要し、極端な場合はサーバエラーとなる(多くの場合、サーバのコンフィギュレーションエラーとして報告される)
  - スタティックドキュメントはWebの常駐プロセスが処理
  - ダイナミックドキュメントの生成は、プロセスの生成やプログラムの起動といった、CPU以外のシステムリソースを多量に消費するアクティビティを伴う...プロセス生成・プログラム起動の失敗
    - メモリ不足
    - ファイルディスクリプタ不足
    - プロセススロット不足

サイトプランニング: ダイナミックドキュメントのインパクト(4)

- ダイナミックドキュメントのインパクトを軽減する
  - クライアントサイド処理の積極的な導入
  - サーバAPIの利用：(極端な方言を避けて...)
  - ひんぱんに起動されるスクリプト系CGIプログラムの削減
    - 処理はスクリプト言語で記述し、常駐プロセス化する
    - 常駐プロセスへのインターフェースのみをマシンバイナリのCGIプログラムとする
    - メモリに余裕があれば、sticky bit を利用する
- (教訓)
  - Webは万能解ではない
  - システムプログラミングの鍛練を怠ってはいけない

サイトプランニング　ハードウェアの選定

- 考慮すべき要素は判っている
  - CPUサブシステム
  - メモリサブシステム
  - ディスクサブシステム
  - ネットワークサブシステム
- で、何を考慮すればいいんだ？
  - サブシステム単体性能の組み合わせによるシステム全体の性能の推測
  - サブシステム間のバランス
    - ボトルネックになっているサブシステムはないか
    - オーバースペックになっているサブシステムはないか
  - その他の要素：拡張性, 保守性, 信頼性, ...
- 用途や負荷が決まっているシステムでも選定や構成の決定が難しい

Webサイトプランニング　ベンチマークの利用

- ベンチマークを参照･評価･利用することによって、あるシステムのおおまかな性能を知ることが可能
- 注意すること
  - ベンチマーク対象システムのコンフィギュレーション
  - ベンチマークで測定される･されない性能項目の把握
  - ベンチマークソフトウェアのバージョン
  - ベンチマーク合戦

Webサイトプランニング　Webのベンチマーク

- WebStone
  - http://www.sgi.com/Products/WebFORCE/WebStone/
  - 現行バージョン：2.01
  - ベンチマークプロファイル（含ワークロード）が変更できる
  - ダイナミックドキュメントも扱える
  - 同一ワークロードなら詳細な性能比較が可能
- SPECweb
  - http://www.specbench.org/osg/web96/
  - 現行バージョンSPECweb96
  - ベンチマークプロファイルが固定
  - 96年度版はスタティックドキュメントが対象
  - 「つかみ」には便利

Webサイトプランニング　SPECweb96（1）

SPECweb96の例：SPECweb96 = 310

Webサイトプランニング　SPECweb96 (2)

SPECweb96の例: SPECweb96 = 626

Webサイトプランニング　SPECweb96, so?

- SPECweb96による「つかみ」

Vender	System	SPECweb96
Digital	Alphaserver 2000 5/300	570
Digital	Alphaserver 2000 5/300	681
Hewlett	9000 Series 700 Model B160L	380
Hewlett	HP 9000 Model K420	630
Hewlett	HP9000/D360	704
Hewlett	NetServer 6/200 LH Pro	536
IBM	RS/6000 43P-140	459
Sun Mic	Netrai 3.1 / u2-200	626
Sun Mic	Ultra 1 Model 170	310

Webサイトプランニング　ソフトウェア(1)

- 安定かつ容易なオペレーションのためのbaseline
  - サイト特有の要求を満たすことは重要だが、それらに拘束されてはいけない
  - 「早い」が「新しい」ものよりは、「そこそこ早くて」しかも「枯れている」ものを選択する
  - 特別な機能が必要でない限り、「よく使われている標準的な」ものを採用する
  - フリー v.s.商用の問題は、構築・運用担当者のスキルv.s.バジェットの問題に帰着する
- 問題の起きそうな部分は論理的あるいは物理的に別システムに分離してしまう

Webサイトプランニング　ソフトウェア(2)

- オペレーティングシステム
  - 経験のあるものを極力採用する
    - 運用
    - プログラム開発
    - 性能に関する直感
- HTTPデーモン（サーバソフトウェア）
  - Per-connection forkingするものは絶対に避ける
  - 方言のせいで、コンテンツが蓄積されると、変更することが難しいことがある → コンテンツルールで縛る
  - 同じH/Wプラットフォームでの異なるS/Wのベンチマークがある場合には積極的に利用する

Webサイトプランニング　ソフトウェア(3)

- ファイルシステム
  - RAIDは便利だが経験も必要：落とし穴がたくさんある
    - RAID5のジャーナリングは遅い
    - RAIDの構成と性能に関する経験を十分に蓄積する
    - 障害復旧練習を十分行ってから実用化する
  - 大きなファイルシステムと小さなファイルシステム
    - 大きなファイルシステム
      - 管理が簡単
      - 障害があった場合は大変
    - 小さなファイルシステム
      - 管理労力は増える
      - 障害時におけるgraceful degradationが可能

Webサイトプランニング　ソフトウェア(4)

- TCP/IPネットワークコード
  - 最新のものを使用する：e.g.: Solaris2.5は2.4の3割増しの性能を提供できる
  - パッチ・アップデート情報に気をくばる（TCP/IPネットワークコードはいまだに枯れきっていない！）
- CGIプログラム
  - 何に使うのかわからないものは落とす
- データベース

コンテンツ管理

- コンテンツを効率よく管理するためのルールを作ることが必要
  - コンテンツのモジュラリティ
    - ファイル(URL)のパス (plain and CGI)
      - 相対パス
      - ホスト名のエンコーディング
    - コンテンツのクラスタを比較的自由にサイト内で移動させられる
    - 他のプラットフォーム (h/wおよびs/w)への移行が容易
  - コンテンツの内容に関するセキュリティ規則

コンテンツ管理

- アップローディング手続き
  - プロダクションサーバで直接コンテンツを作成する
    - コンテンツが長時間にわたって不安定になる
    - マルチユーザでプロダクションサーバを運用することのセキュリティ上の問題
  - 別環境で作成したものを、一気に転送する
    - コンテンツは過渡的に不安定になる
    - 環境が一致しない
  - ステージング用のサーバを用いる
    - プロダクションサーバと同一環境の別マシン
    - ハードウェア資源が余計に必要
    - レンタルサーバでの標準になるか？

Webサーバのセキュリティ (1)

- 基本的なシステムセキュリティ
- 基本的なネットワークセキュリティ
- コンテンツに関連したセキュリティ
  - ディレクトリの自動インデキシング
    - ユーザ名やファイルパーミッションなどが漏れる
  - サーバ機能のオーバーライド
    - CGI実行、HTTPメソッド、...
    - 階層的なオーバーライドに注意
  - HTTPから他のアクセスメソッド(e.g. ftp)への非連携問題
- アクセス制御
  - 他の要素 (e.g.: ファイアウォール) と協調して実現するような
  - アクセス制御はできるだけ避ける：管理が難しい

Webサーバのセキュリティ (2)

- CGIセキュリティ
  - 禁止してしまう
  - 制限 (ポリシー) を設けて許可する
    - ポリシーの例
      - 「して良いこと」と「してはならないこと」に関する明確なルールを設ける
      - すべてのCGIプログラムをソースレベルで理解し管理する：機能と能力を把握していないものは組み入れない
    - ポリシー強制メカニズム
      - chrootやCGI実行パスの制限を活用する
      - スクリプト言語の能力に制限を加える
  - 管理しない

Webサーバのセキュリティ (3)

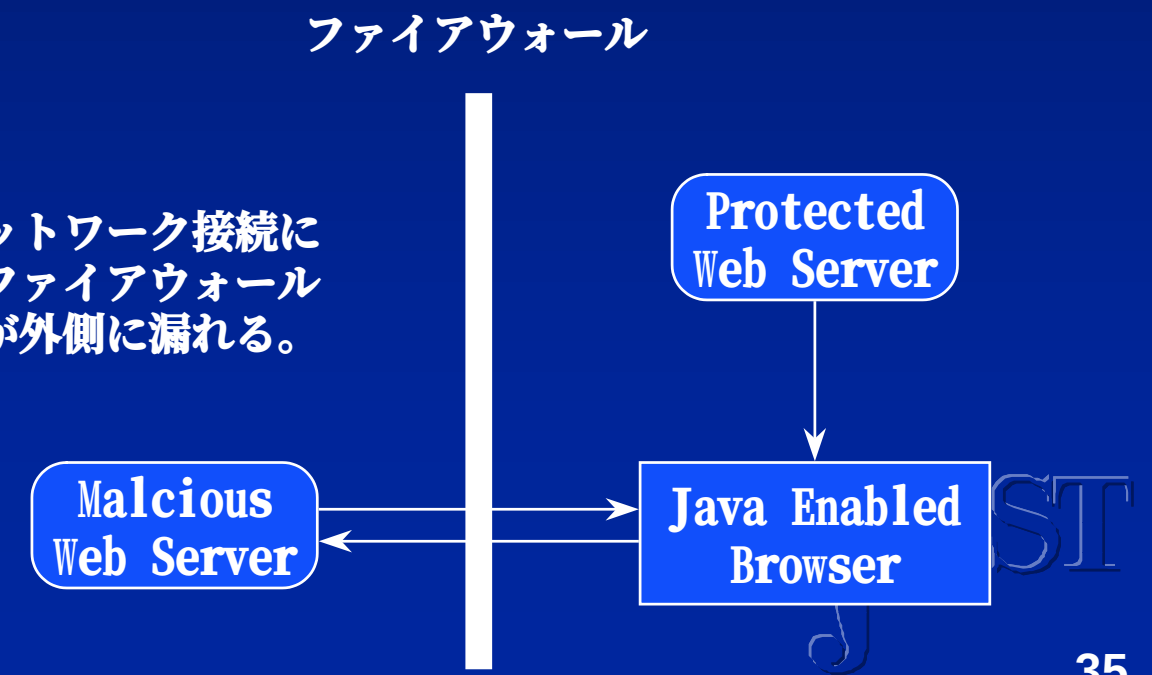
- 例：PUTメソッドの落とし穴 ... PUT & SSI
  - 「PUTメソッドを使ってファイルをアップロードしたい」という要求をときどき見かけるが...
  - PUTの使用許可には十分注意する必要がある（特にSSIを使用している場合）
    - PUTされたSSI入りドキュメントをGETされると、任意のプログラムの実行を許してしまう。
    - PUTできるディレクトリからはGETさせない / GETできるディレクトリへはPUTさせない
    - PUTできるディレクトリではSSIを禁止する
  - コンフィギュレーションミスが無いようにしよう
    - PUTとSSIをひとつのサーバで同居させないのが一番

Webサーバのセキュリティ (4)

- 複数のサブシステムから構成されるサイト
  - サブシステムへの侵入
- Intranetでは状況がさらに複雑に...

Webサーバのセキュリティ (5)

- 例：Intranetのセキュリティ・ホール
  - クライアント（ブラウザソフトウェア）も含めて考える必要がある
  - 実行可能なデータのダウンロードに気をつける（ポリシー強制メカニズムが問題：FWによるmime-typeの監視が必要?）



Webサーバのチューニング

- ファイルシステム
  - アクセスが複数ドライブに分散されるようにする
  - ループバックファイルシステムが利用できる
- ログ
  - ばかにならない量のログが生成される
  - さしあたって使用しないログはとらない (エージェントログ等)
  - エラーログは必ずとる。そして必ず監視する
- ネットワークコードのパラメータ
  - listenキューの長さ
    - サーバ性能を超えて到着する要求をごく短時間バッファリングできる (限界はある)
- クッキーの発行に関する考慮
  - キャッシング・プロキシを有効に利用するには...

Webサーバの監視

- 基本システムアクティビティ
  - 障害徴候：ディスク、メモリ、ネットワーク
  - 認証誤り
- サーバログ
  - サーバのコンフィギュレーション誤り
  - ポリシー違反のコンテンツ
  - 過負荷状態：CGIの起動エラー（重要）
- ログに現れないサーバイベント
  - コネクションの拒否：能力を超えた要求
  - ネットワークモジュールのステータス検査コマンドでわかる場合もある

大規模サイト構築: WWWはスケールしない!!!

- URLはインターネット内の単一のオブジェクトを指す
  - レイテンシ
    - 光速しばり
  - サーバ負荷
    - Simple document retrievalはOKだとしても
    - CGI, SSI, サーバープッシュ, クライアントプル
    - ファイルディスクリプタ数の上限
    - TCP コネクション (数とレートの上限)

大規模サイト構築へのアプローチ

- Brute force
  - より大規模なシステムを必要に応じて導入していく
- Divide and conquer（分割統治）
  - By request
    - 同じ機能を持ったシステムを複数用意する方法
    - ホモジニアスなアプローチ
  - By function
    - 機能別にシステムを分割していく方法
    - ヘテロジニアスなアプローチ
    - 機能のわけ方には限界がある。
      - ある機能を実現したシステムが飽和したらどうするか...
  - By content（by-function の変形）
    - 特定のドキュメントを特定のサーバに割り当てる

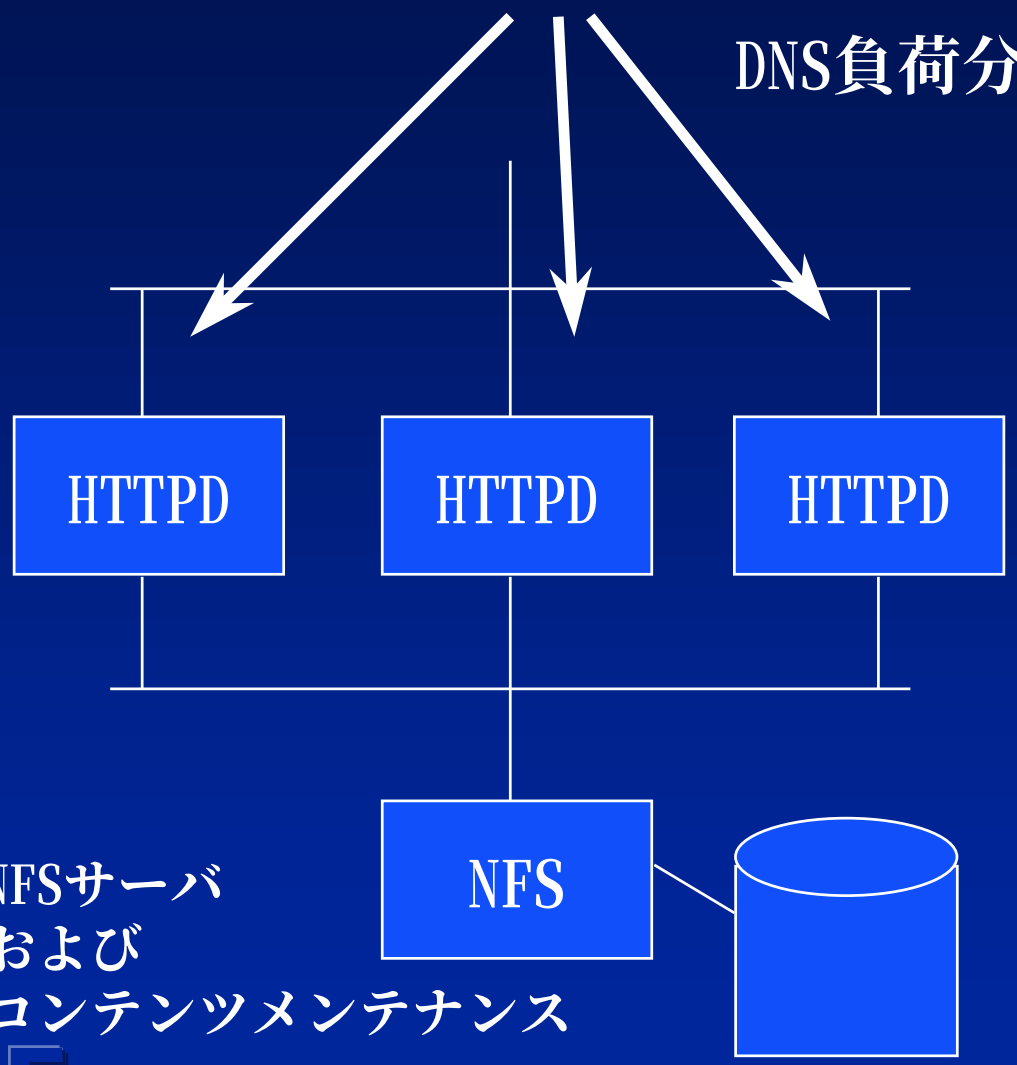
サイトスケーラビリティの要素

- 性能（スループットとレスポンス）
  - H/Wの性能
  - ネットワークの性能
  - クライアントまでの物理的な距離
- 管理
  - システム
  - コンテンツ
- コスト
- 透過性
- 障害対策
  - 内部：復旧のための手段
  - 外部：サービスの継続性

Webサイトの分散・複製

- 問題点
  - システムのアーキテクチャは一種類か
    - バイナリのCGIプログラム?
  - コンフィギュレーション (環境) は同一か?
    - Document root ?
    - Path to executables ?
    - Path to Binary system programs ?
  - すべてのシステムは同じ機能を提供しているか?
  - コンテンツの同期と配布
    - 単一ソースと複数ソース
  - データベースの取り扱い

典型的な集中型Webサイト例



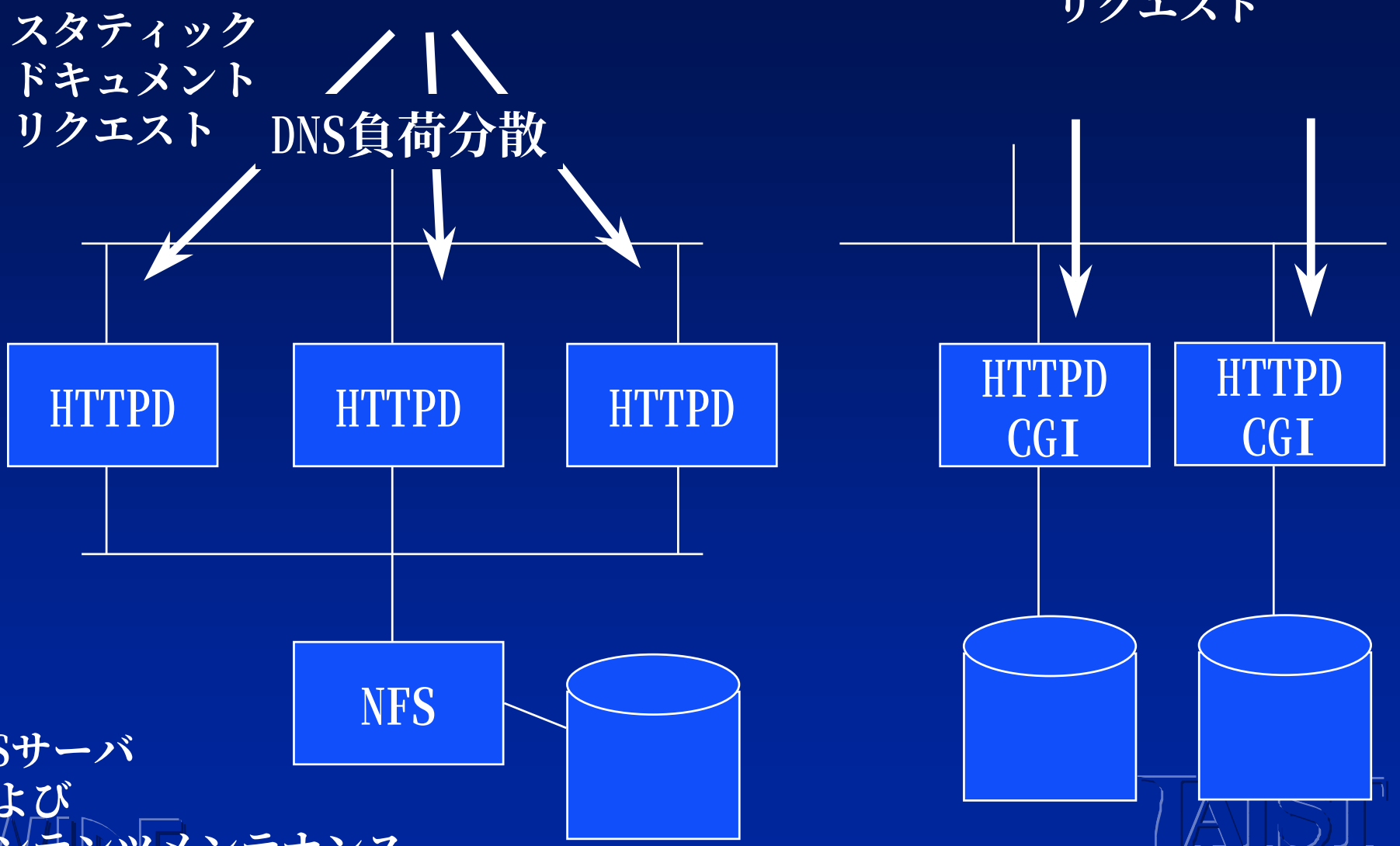
ほぼ等機能の
サーバアレー

NFSサーバ
および
コンテンツメンテナンス

WIDE

JAIST

典型的な集中型Webサイト例(2)



DNS負荷分散の問題点

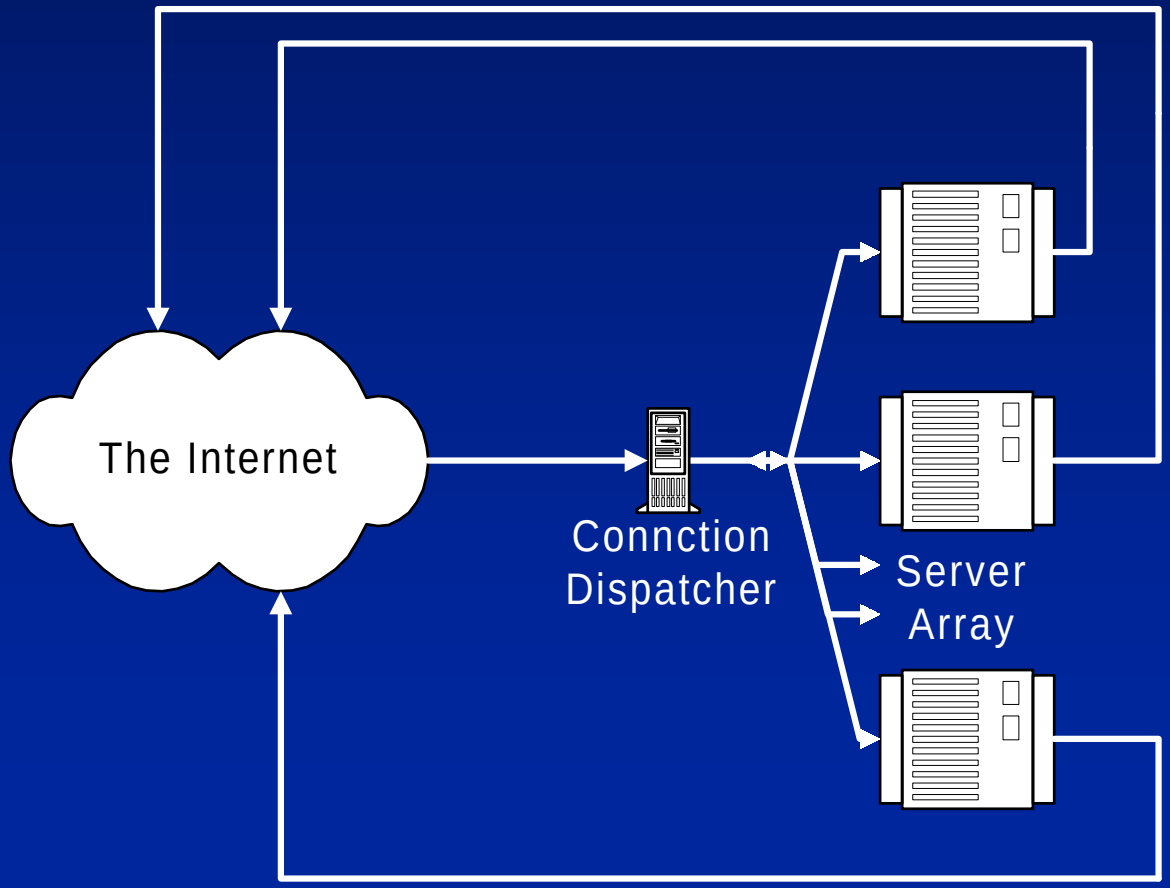
- システムダウンが外部に見えてしまう
- システム構成を短時間で変更することが不可能

WIDE

JAIST

TCPコネクションディスパッチング (1)

- 透過的にサービスのスケーラビリティをあげるためのテクニック
- TCPのコネクションレベルで負荷分散をはかる
- 4層スイッチ (?) の一種



TCPコネクションディスパッチング (2)

- 複数のインプリメンテーションがすでに存在
  - CISCO: LocalDirector
    - NATの応用
  - IBM: TCP Router
    - 単方向ディスパッチング
    - SP2に特化: ディスパッチにMACアドレスを使用
  - JAIST:
    - TCP Routerと似ているが、ディスパッチにIPアドレスを使用
  - Others (Commercial & Free)

TCPコネクションディスパッチング (3)

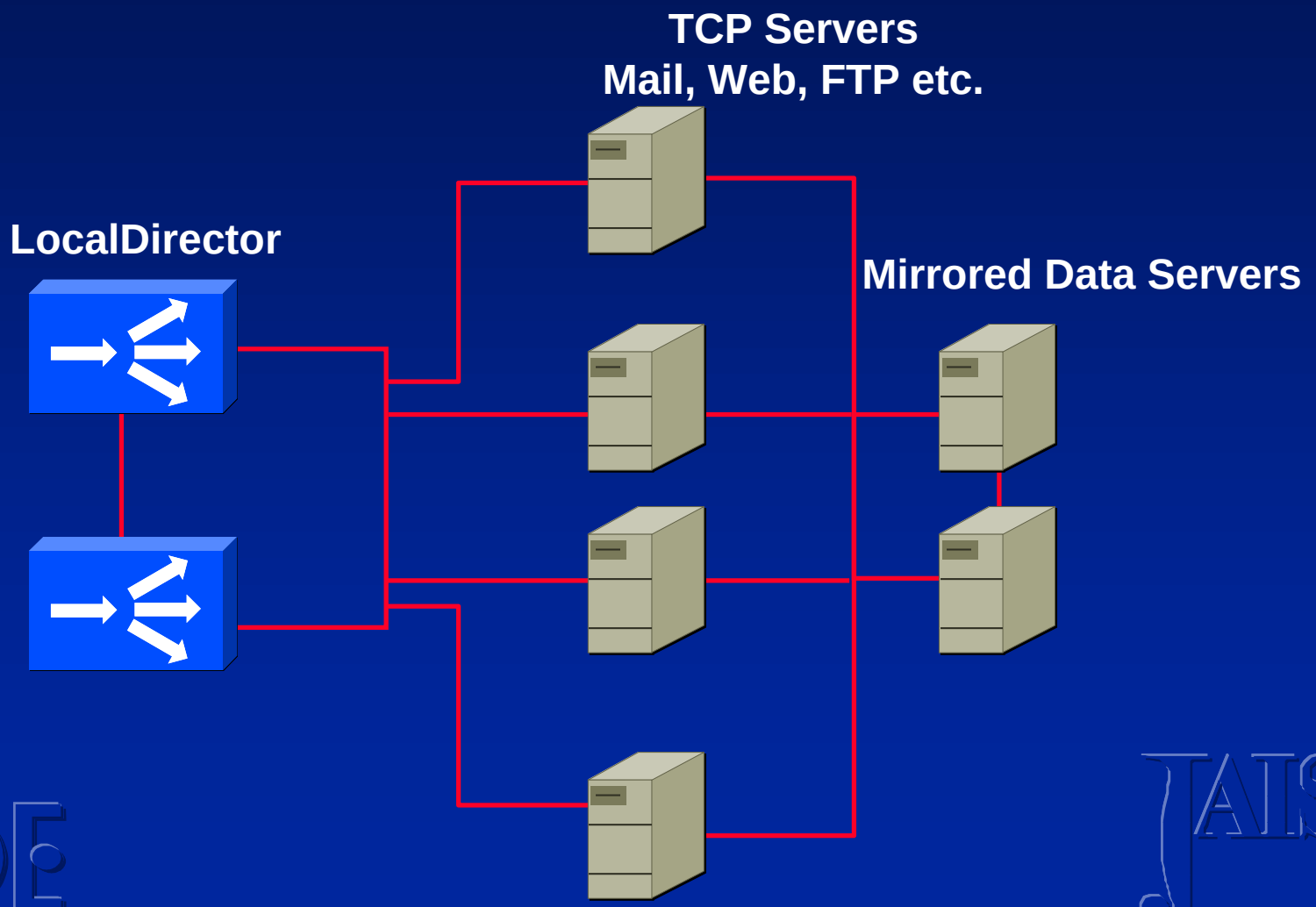
- 可能性
  - ローカルな負荷分散
    - 同じ処理能力を持つシステムは、大規模な単一システムより小規模な複数のシステムで構成したほうが安価
    - スケーラビリティ
  - TCPのコネクション能力の上限を超えられる可能性
  - 他のテクノロジとの組み合わせによる可能性
    - サーバリダイレクション+ コネクションディスパッチング

TCPコネクションディスパッチング (4)

- 問題点
  - NAT方式
    - スケーラビリティ
      - 複数箇所の書き換えが必要
      - 両方向のトラフィックを処理することが必要
  - 単方向ディスパッチ
    - 下りのトラフィックを、次の上りのトラフィックを見て推測しなければならない
  - 共通
    - 複数のコネクション間の隠れた依存性
      - ftp
      - http with hidden state (e.g.: cookie)

Local Director 応用例

Totally Reliable Fully Redundant Fault Tolerant



WIDE

JAIST

地理分散：大規模サイトのもう一つの形態

- サービスの分散配置
  - 地理的に均一なサービスを提供する
- クライアントから見て最も適切なサーバを選択する
  - ネットワーク距離（メトリック）
  - 往復時間
  - アクセスポリシー
- 問題
  - サーバの位置決め（ロケータ）
  - サーバの選択
  - 自動化：インターネット的に...

WIDE

JAIST

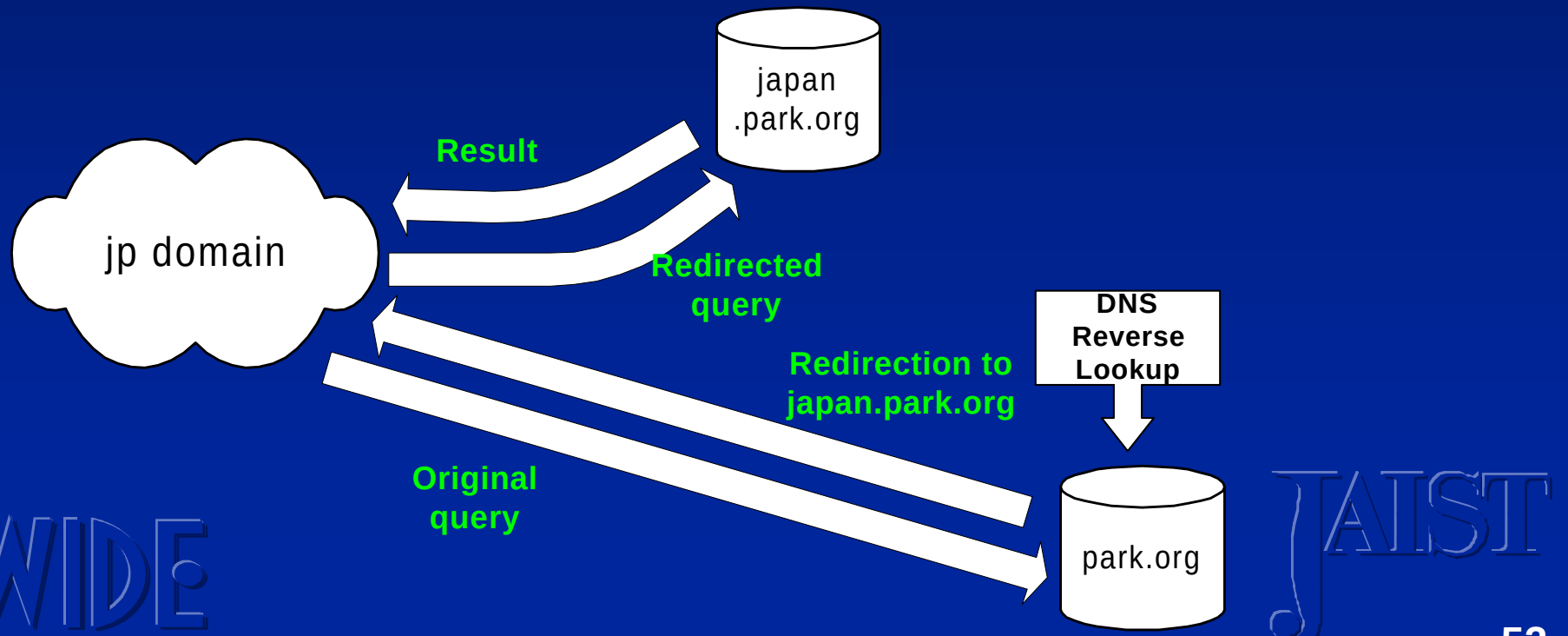
地理分散: 大規模サイトのもう一つの形態

サービスの自動ディスパッチング

- Webサービスにおいて常に小さなネットワークレーテンシを確保するためには、「近く」のサーバへアクセスすればよい
  - ユーザにサーバを選択させる方法は、あまりにも透過性に欠け、あまりにもインターネット的でない
  - HTTPリダイレクションの利用
    - アドレスからドメイン名への逆引きと位置の推測
    - ネットワーク (IP)層が持っている情報の利用
    - 実測値の計測と利用
- 透過的な負荷分散
- サーバのサービス能力の差を隠蔽

FQDNによるリダイレクション

- あるタイミングで
  - クライアントのFQDNを得る
  - トップドメイン (jp, uk, au,...) によるリダイレクションを行う
  - ホスト名を記述しないというルールに従ってコンテンツが作成されている限り、リダイレクト先のホストをアクセスし続ける

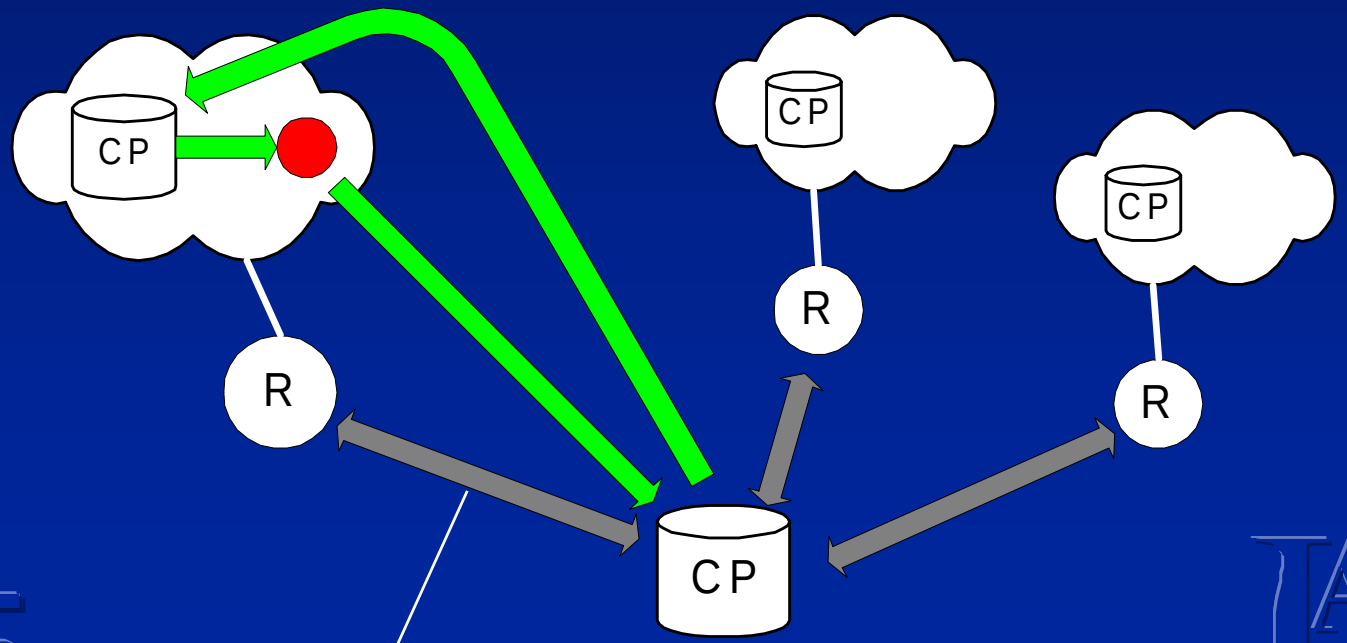


FQDNによるリダイレクション

- 長所: インプリメントが簡単
  - 新規のメカニズムをまったく必要としない
  - CGIによるディレクトリインデックスファイルとして実現
- 短所: 推測に過ぎない
  - ドメイン名の構造は、必ずしもネットワークの構造（地理的あるいはネットワーク的）と一致していない ...
  - com/org/netは完全に破滅
  - FQDNが得られないことも多い

ネットワーク層の情報に基づくリダイレクション

- サーバに近いルータに、クライアントに対するネットワーク・メトリックの問い合わせを行い、
- その結果に基づいたリダイレクションやコネクションのディスパッチを行う

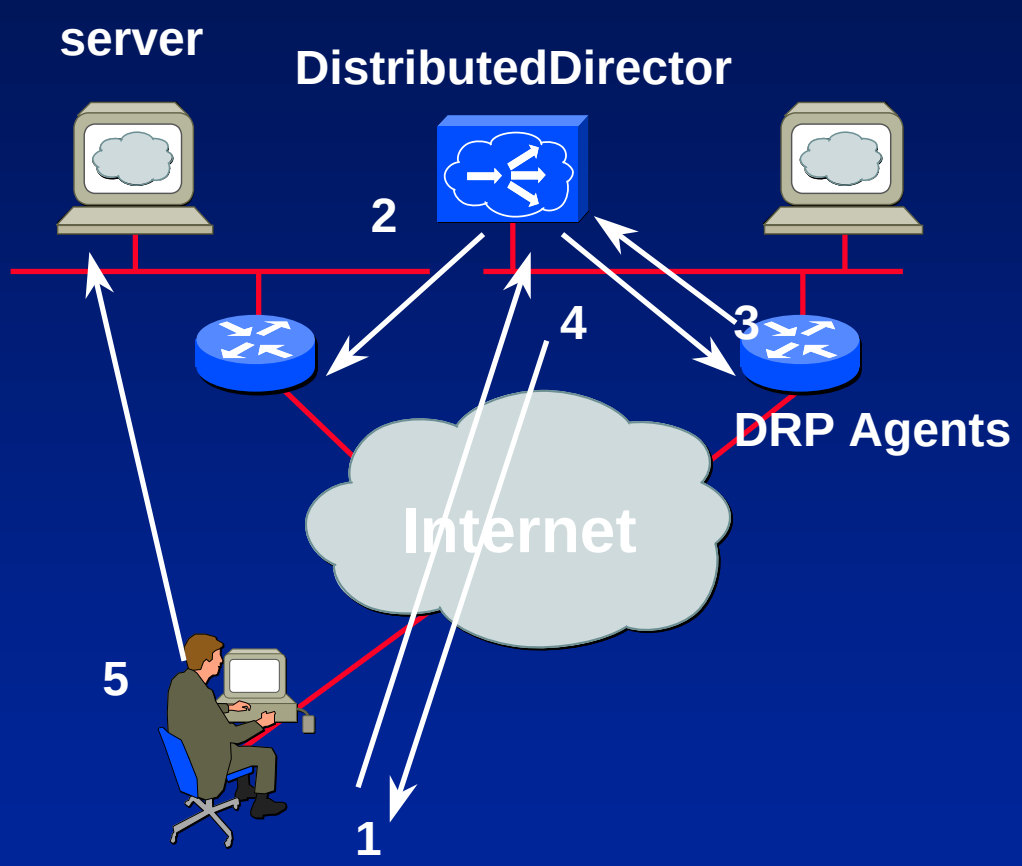


ネットワーク層の情報に基づくリダイレクション

- 長所:
  - 問い合わせるルータ群が適当であれば、経路制御プロトコルの能力・設定の範囲内で最もクライアントに近いサーバが選択できる
- 短所:
  - ルータ群へのRTT分だけのオーバーヘッドが必要
  - メトリック問い合わせのための新しいプロトコルが必要
  - 実際に到達可能であるかが考慮されない
  - ルータに対応したサーバの負荷状態が考慮されない

DistributedDirectorによるリダイレクション

- 1:clientがDDに接続しにいく
- 2:DDがDRP Agentに問い合わせる
- 3:最適サーバーを計算する
- 4:clientに計算結果のサーバーを教える



WIDE

JAIST

実測値によるリダイレクション

IBM's Atlanta Olympic Server

- クライアントからの要求に対し、そのクライアントに対するpingを各分散サーバに依頼し、その結果に基づいてディスパッチ先を決定する
- 長所:
  - 到達可能性を含め、ネットワークの瞬間的な状態を反映したディスパッチィングが可能
- 短所:
  - サーバまでのRTTに加え、pingが安定するまでの時間がさらに必要

WWWの暗黒面

- WWW は最終兵器ではない...
  - WWWはスケールしない
  - HTTPはInternetを殺す
  - HTTPはステートレス
- HTTP-NG???

WIDE

JAIST

HTTPステートレス問題

- 各Requestは、それぞれ個別のTCPセッション
- このステートレスは、2つの問題を発生させる
  - 性能の劣化
  - デジタルコマースなど、認証などを必要とする複数のやり取りが必要な場合
- ソリューション (?)
  - ＝＞ Keep-Alive
    - テクノロジ:
      - GET /~osamu/ HTTP/1.0
      - Connection: Keep-Alive
    - サーバ側の事情: いつ切れるか判らないコネクションをいつまでもキープできない
  - ＝＞Cookieの利用
    - セッションのトラッキング

HTTPは Internetを殺す

- HTTPのトランザクションは非常に短い
  - 平均12KB
  - 大部分が2KB以下
- TCPの最適化は大量の転送を前提に行われてきた
  - Keep-Aliveは救世主になるか？
  - T/TCPは？
- プロトコルに大規模な変更を施すのは難しい
  - HTTP/1.0 -> HTTP/1.1
  - TCP -> T/TCP
  - サーバ･プロキシ間で導入すると効果があがることが報告されている。

プロキシ & キャッシュテクノロジのインパクト

- 同じ情報が同じ伝送路を何度も何度も流れる「無限地獄」の回避
  - ばかにならないWebトラフィックの削減
- レーテンシの削減
  - 遠くの実物より近くのキャッシュ
  - ドキュメントの「新しさ」に関するコストあるいはQoSの概念が必要
- HTTPサーバのアクセラレータ
- コンテンツのオンデマンド自動同期
  - 必要とされる情報のみが同期する
- プリフェッチによるレーテンシの削減
  - トラフィック増加が難点: 準最適な先読み戦略...

近代的/大規模サイトの将来

- CPUやネットワーク性能をスケールさせる手段は利用可能になっている。
  - コスト
  - 耐故障
  - 透過的な広域負荷分散
- 管理面での考慮が重要になってくる
  - システム
  - コンテンツ
    - データベースとの融合
    - コンテンツドキュメントと内容を表わすオブジェクトの連携