モバイル通信プロトコル

大阪大学　大学院情報科学研究科
情報ネットワーク学専攻
中田　明夫

授業の目的（シラバスより）

- モバイル通信環境と高速情報通信ネットワークを組み合わせた大規模ネットワークで利用される通信プロトコルについて講義すると共に、高信頼度情報ネットワーク実現のためのネットワーク設計法について説明する。
  - ネットワーク環境下での分散システム構築手法について、教科書に沿って講義

授業の進め方

- 教科書にそって講義
  - 教科書：　Andrew S. Tanenbaum and Martin van Steen: “Distributed Systems: Principles and Paradigms”,Prentice Hall. ISBN:0131217860
    - 訳本：水野忠則,宮西洋太郎,鈴木健二,西山　智,佐藤文明,東野輝夫 訳「分散システム 原理とパラダイム」ピアソン・エデュケーション,ISBN:4894715562
    - モバイル関係も内容に含まれるが、それのみに特化しない
    - 受講者の要望に応じて内容を若干変更することもある
  - 授業の資料（スライド）
    - 次回の授業からは、以下のURLから各自ダウンロード
      http://www-higashi.ist.osaka-u.ac.jp/~nakata/mobile-cp/
- 成績評価：出席状況30%、レポート70%

始めに

第1章

分散システムとは？

- 独立した複数の計算機の集まり
- ユーザからは一つの一貫したシステムに見える
  - システムを構成している計算機の違いや、それらが内部で行う通信は、ユーザには見えない
- スケーラビリティ
  - 拡張や大規模化が比較的容易
- 耐故障性
  - どこかが故障していても、継続的に稼動
    - ユーザには気づかせない

分散システムの例

- 銀行のオンラインシステム(ATM)
- ネットワークで結ばれた複数のワークステーション
  - 分散ファイルシステム（NFSなど）
- World　Wide　Web（WWW）
  - 分散文書システム

分散システムの目標

- ユーザとリソースとの接続手段の提供
- 透過性(transparency)の実現
- 開放性(openness)の実現
- スケーラビリティ(scalability)の達成

ユーザとリソースとの接続

- 複数ユーザで遠隔リソースを共有
  - リソースの例
    - プリンタ、計算機、ストレージ（ハードディスクなど）
    - データ、ファイル、ネットワークなど
  - なぜ共有するか？
    - 共有した方が安くつく（経済的な理由）
    - 複数人で共同作業や情報交換を行うため

透過性

- 透過性（Transparency）
  - プロセスやリソースが分散していることをユーザから隠蔽
  - ユーザからは一つの計算機システムに見えるシステム
    - transparent（透過的）

分散システムの透過性(1/8)

図 1-2　分散システムの透過性［ISO,1995］

透過性	説明
アクセス	データ表現における相違並びにリソースがいかにアクセスされているかを隠蔽する。
位置	リソースがどこに位置するのかを隠蔽する。
移動	リソースが他の位置に移動するかもしれないことを隠蔽する。
再配置	リソースが使用中に、他の位置に移動するかもしれないことを隠蔽する。
レプリケーション	リソースが複製されていることを隠蔽する。
並行	リソースがいくつかの競合ユーザに共有されていることを隠蔽する。
障害	リソースの障害と回復を隠蔽する。
永続	（ソフトウェア）リソースがメモリまたはディスク上のどちらに存在するかを隠蔽する。

分散システムの透過性(2/8)

- アクセス透過性(access transparency)
  - データ表現やリソースアクセス方法を隠蔽
    - 整数の表現(little endian,big endian)、OSによるファイル名表現の差異などを隠蔽
    - → CPUやOSの差異を隠蔽

分散システムの透過性(3/8)

- 位置透過性(location transparency)
  - リソースの位置を隠蔽
    - リソースに論理的な名前のみを割り当てることで達成
      - 例）URL、DNSホストネーム、パスネームなど
        - http://www.google.com/
        - /home/nakata/
        - /dev/mouse
        - など

分散システムの透過性(4/8)

- 移動透過性(migration transparency)
  - リソースが他の場所に移動しても同じ手段でアクセス可能
    - 例）Webサーバが移動しても同じURLでアクセス可能など
- 再配置透過性(relocation transparency)
  - リソースをアクセス中に他の場所に移動することを許す
    - 例）無線通信のローミング（通信を切断することなく端末を移動）

分散システムの透過性(5/8)

- レプリケーション(複製)透過性（replication transparency）
  - リソースのコピーが幾つか存在することを隠蔽
    - リソースをコピーし、それがアクセスされている場所近くに配置
      - 可用性（availability）や性能を向上する目的
    - コピーされたリソース(レプリカ(replica)）はすべて同じ名前を持つ必要--- 透過性を保つため
      - 位置透過性も同時にサポートする必要

分散システムの透過性(6/8)

- 並行透過性（concurrency transparency）
  - リソースを複数ユーザで共有
    - 同時に同じリソースにアクセス
    - → 一般に競合が発生
  - 各ユーザに対して、リソースを共有している事実を隠蔽
    - 共有リソースを(矛盾の無い)一貫した状態に
      - 排他制御（ロック）機構、トランザクション機構などで実現

分散システムの透過性(7/8)

- 障害透過性(failure transparency)
  - リソースに障害が発生したこと、および、後に障害から復旧したことを隠蔽
  - 分散システムでは最も実現が困難
    - 故障したリソースと、（反応が）非常に遅いリソースを区別するのが困難
      - 特にネットワークを隔てて離れたリソースに関して

分散システムの透過性(8/8)

- 永続透過性(persistence transparency)
  - リソースが(揮発性の(volatile))メモリ上にあるか、ディスク上にあるかを隠蔽
  - 例）
    - オブジェクト指向データベース
      - ディスクに蓄積されたオブジェクトのメソッド呼び出し
      - データベースサーバは、オブジェクトの状態をディスクからメインメモリ（1次記憶）にコピーし、メソッドを実行し、（必要なら）変更された状態をディスク（2次記憶）に書き戻す
      - サーバが状態を1次記憶から2次記憶に書き戻したか否かはユーザにはわからない
    - ライトバックキャッシュ

透過性実現の度合い(1/2)

- 透過性の度合いを高めると一般にシステムの性能は悪化（トレードオフ関係）
  - 障害透過性の例
    - インターネットアプリケーション
      - サーバに何回か接続を試みた後、接続不可能ならあきらめる
        - サーバの一時的な障害を隠蔽しようとすると、何回も再接続を試みる必要あり →動作が遅くなる
        - 早めに再接続をあきらめるか、ユーザがキャンセルできるようにした方が性能がよい

透過性実現の度合い(2/2)

- 透過性の度合いを高めると一般にシステムの性能は悪化(トレードオフ関係)
  - 複製透過性の例
    - リソースの複製
      - 非常に離れた場所(異なる大陸)に複数存在
      - それらの内容が全て同じであることを保証したい
      - 複製のどれか一つを変更すると他の複製に変更を伝播させるのに非常に時間が掛かる

開放性(Openness)(1/2)

- 開放型(オープン)システム(Open System)
  - 標準化されたサービスを提供するシステム
    - 例)ネットワークの送受信のルール
      - プロトコルとして標準化
- → 作成者の異なるシステムを相互接続

開放性(Openness)(2/2)

- 開放型分散システム(Open Distributed System)
  - サービスは**インタフェース**として定義
    - 利用可能な関数名、パラメタの型、返り値、可能な例外などを記述
    - サービスのシンタックスのみ記述
      - サービスの処理内容(セマンティクス)の記述は困難
      - 現状では自然言語で記述
  - インタフェースをきちんと記述できると…
    - 違う会社が作った異なる実装同士を組み合わせて分散システムを構成可能

スケーラビリティ

- スケーラビリティ(scalability)
  - システムを容易に大規模化できること
    - システムのサイズに関して
      - ユーザ数やリソース数を容易に増やせる
    - システムの地理的な規模に関して
      - ユーザやリソースの場所がどんなに遠く離れていてもよい
    - システム管理に関して
      - システムがどんなに多くの組織にまたがっていても、管理が容易であること

スケーラビリティ問題(1/5)

- システムのサイズを大規模化
  - ユーザ数やリソース数を増やそうとすると…
    - 中央集中型(centralized)サービス、データ、アルゴリズムでは限界がある
  - 中央集中型サービス(centralized service)
    - サーバが一つだけ →ユーザが増えるとボトルネックに
  - 中央集中型データ(centralized data)
    - 例:オンライン電話帳
      - データが増えるといつかはディスクに収まりきらなくなる

スケーラビリティ問題(2/5)

- 中央集中型アルゴリズム(centralized algorithm)
  - 例:メッセージの経路制御
    - ネットワーク上の全ての情報を集めて最適経路を計算し、結果を各マシンに分配
      → ネットワークに多大な負荷をかける
  - 経路制御には分散型(decentralized)アルゴリズムのみ使用されるべき

スケーラビリティ問題(3/5)

- 分散型アルゴリズム（decentralized algorithm）
  1. いかなるマシンもシステム全体の完全な情報を持たない
  2. 各マシンは局所的な情報のみを元に意思決定を行う
  3. 一つのマシンの障害が全体のアルゴリズムの動作に影響しない
  4. 大域的な時計の存在を仮定しない
     - 分散システムの全ての時計を正確に同期させるのは不可能

スケーラビリティ問題(4/5)

- 地理的なスケーラビリティ(geographical scalability)
  - LAN向けに設計された分散システムをWAN用に大規模化するのは困難
    - LAN向けのシステムは同期通信(synchronous communication)に基づく
      - 同期通信（synchronous communication）
        - サービスのリクエストを送信後、停止(block)して応答待ち
      - LANでは通信遅延が小さいのでOK (高々2~300マイクロ秒)
      - WANでは通信遅延が数百ミリ秒 → 同期通信を用いた対話的(interactive)アプリケーションの作成は困難

スケーラビリティ問題(5/5)

- 地理的なスケーラビリティ(geographical scalability)
  - LAN向けに設計された分散システムをWAN用に大規模化するのは困難
    - WAN上の通信はLANに比べて信頼性がより低く、1対1通信ベース
      - 信頼性が高いブロードキャスト通信が利用できない

スケーリング技術

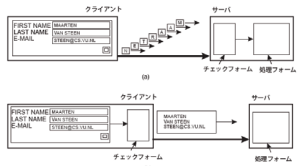
- スケーラビリティ問題の解決法
  - 通信遅延の隠蔽
  - 分散化(distribution)
  - 複製化(replication)

通信遅延の隠蔽

- 地理的なスケーラビリティ問題の解決法
- 基本アイディア:
  - リモートサーバからのレスポンスを待たない
    - 例）リクエストを出した後、ブロックして返事を待つ代わりに、何か他の仕事をする
    - 返事が到着したら割り込みがかかり、以前出したリクエストに関する処理を完了する
      - 非同期通信(asynchronous communication)
    - 対話的なアプリケーションには使えない

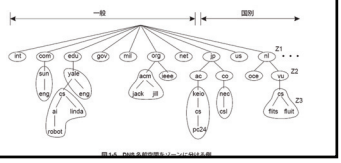
通信遅延の隠蔽

- 地理的なスケーラビリティ問題の解決例
  - Webのフォームを使ってデータベースにアクセス
  - 入力フォームの構文チェック
    - サーバ側でチェックを行い、結果を待つ → 通信遅延により遅くなる
    - クライアント側で構文チェックを行い、完全な入力データをサーバに送信



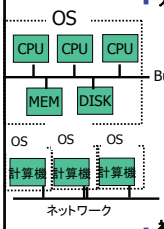
分散化(distribution)

- 分散化：システムを小さい部分に分解して、分散配置
  - 例）DNS(Domain Name System)
    - 名前空間
      - domainによって階層的に構成
      - 複数のzoneに分割
      - 各zoneに属する名前はそれぞれ一つのネームサーバが扱う



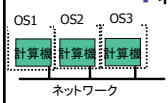
分散システムのソフトウェア(1/2)

- 分散システムのOS
  - 密結合システム（tightly-coupled systems）
    - OSはシステム全体のリソースを大域的に管理
    - **分散OS(Distributed Operating System,DOS)**
      - 例）マルチプロセッサOS（UNIXなどの普通のOSのマルチプロセッサ対応版）
        - 複数のCPU,メモリ,ディスクなどを一つのOSで**一元管理**
        - ハードウェアバスで結合
        - 共有メモリで通信
      - マルチコンピュータOS
        - 複数の計算機上に**同じ**OSを走らせる
        - ネットワークで結合
        - メッセージパッシングで通信
    - **複数の独立した計算機群の上で動作するものではない**



分散システムのソフトウェア(2/2)

- 疎結合システム（loosely-coupled systems）
  - それぞれが**異なる**OSを走らせている複数の計算機群の集合
  - 各OSはリモート計算機に対してネットワークを通じてサービスを提供
  - **ネットワークOS(Network Operating System,NOS)**
    - 例）UNIXワークステーション
      - リモート計算機に対してrlogin,rsh,rcpなどのサービスを提供
      - **複数の計算機を一つの一貫したシステムに見せているわけではない**
  - DOSとNOSの欠点を解決→ ミドルウェア



ミドルウェア

- **ミドルウェア**＝独立した複数の計算機を一つの一貫したシステムに見せるソフトウェア
  - 個々の計算機上のNOSとアプリケーションの間に設けたソフトウェアのレイヤ

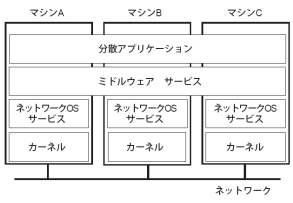


図 1-22　ミドルウェアとしての分散システムの一般構造

ミドルウェアのモデル（パラダイム）

- ファイルモデル
  - 全てのリソースをファイルとみなす
  - UNIX, Plan 9など
- RPC（Remote Procedure Call）モデル
  - リモートマシンの手続きをローカルと同じように呼び出せる → 通信の発生を意識させない
- 分散オブジェクト(distributed object)モデル
  - 各オブジェクトはインタフェース（メソッドの集合）を実装（内部構造は隠蔽）
  - 一つの計算機に実装され、ローカルからも他のリモート計算機からメソッドが呼び出される → RPCと同様
- 分散文書(distributed document)モデル
  - WWW

ミドルウェアのサービス

- 高位の通信手段を提供
  - 低位のネットワーク通信を隠蔽
  - 例）RPC、リモートメソッド呼び出しなど
- ネーミングサービスの提供
  - リソースを探したり共有することが可能（電話帳のように）
- 永続的な記憶領域の提供
  - 分散ファイルシステム、データベースなど
- 分散トランザクションの提供
  - 共有データの読み書き操作がatomicに行われることを保証

ミドルウェアと開放性(Openness)

- ミドルウェア上に分散システムを構築
  - アプリケーションはOSに非依存(そのかわり、特定のミドルウェアに依存)
  - ミドルウェアのインタフェースの定義が不完全 → 同じインタフェースに従うアプリケーションが異なるミドルウェア上で動作不可
  - 通信の実装に異なるプロトコルの利用 → 相互接続不可
- → ミドルウェアが用いるプロトコル、および、アプリケーションとのインタフェースはすべて同じであるべき

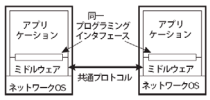


図 1-23 開放型ミドルウェアベースド分散システムでは、ミドルウェアがアプリケーションに提供するインタフェースと同様に、各ミドルウェアで使われるプロトコルは同一である

システム間の比較

図 1-24 マルチプロセッサオペレーティングシステム、マルチコンピュータオペレーティングシステム、ネットワークオペレーティングシステム、並びにミドルウェアベースド分散システムの比較

項目	分散OS マルチプロセッサ	分散OS マルチコンピュータ	ネットワークOS	ミドルウェアベースドDS
透過性の程度	非常に高い	高い	低い	高い
すべてのノードで同じOS	○	○	×	×
OSのコピーの数	1	N	N	N
通信の基礎	共有メモリ	メッセージ	ファイル	モデル特有
リソース管理	全体、集中	全体、分散	ノード毎	ノード毎
スケーラビリティ	×	適度に	○	多様
開放性	閉鎖型	閉鎖型	開放型	開放型

クライアント-サーバモデル

- 分散システムの各プロセスをどのように構成するか？
  - 一つのモデル
    - サービスを要求するプロセス(クライアント)、および、サービスを提供するプロセス(サーバ)で構成
      - クライアント-サーバモデル

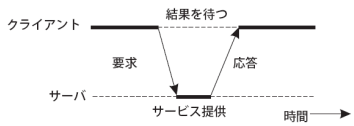


図 1-25 一般的なクライアントとサーバ時間の相互作用

アプリケーションレイヤリング

- クライアント-サーバアプリケーション
  - 多くがデータベースへのアクセスの形
  - 以下の3つの論理レベルに分解
    - ユーザインタフェースレベル(User-Interface Level)
    - プロセシングレベル(Processing Level)
    - データレベル(Data Level)

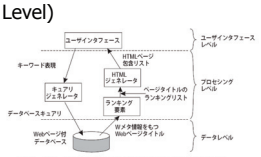


図1-26 インターネットサーチエンジンを3つの異なる層に分けた場合の一般構造

ユーザインタフェースレベル

- 通常はクライアント側に実装
  - エンドユーザがアプリケーションと対話できるようにするプログラムで構成
    - キャラクタベース端末
    - GUI(Graphical User Interface)
      - グラフィカル表示とマウス操作
      - X Window Systemなど

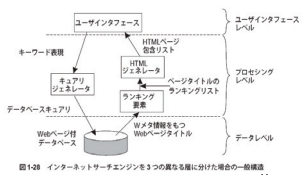


図1-26 インターネットサーチエンジンを3つの異なる層に分けた場合の一般構造

プロセシングレベル

- クライアントとサーバのどちらに実装されるか --- アプリケーション依存
  - インターネットサーチエンジン
    - ユーザインタフェースレベル:キーワード入力を受け付け、webページのリストを返す
    - データレベル:インデックス化されたwebページのデータベース
    - プロセシングレベル:キーワード文字列からデータベースクエリへの変換

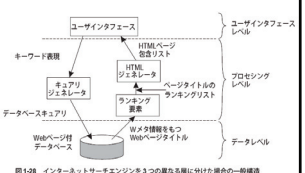


図1-26 インターネットサーチエンジンを3つの異なる層に分けた場合の一般構造

データレベル

- 通常はサーバ側に実装
  - データは永続的(persistent)
    - アプリケーションが実行されていなくてもデータは存在しつづける
    - ファイルシステム、データベースなどとして実装

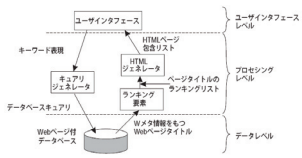


図1-28 インターネットサーチエンジンを3つの異なる層に分けた場合の一般構造

クライアント-サーバアーキテクチャ

- クライアント-サーバモデルにおける3つの論理レベルの構成
  - 最もシンプルな構成
    - クライアント:ユーザインタフェースレベルを実装
    - サーバ:残り2つのレベルを実装
  - 全ての処理はサーバで行われ、クライアントは単なる端末に過ぎない → 分散システムとは呼べない
  - 他の構成の可能性

多段アーキテクチャ

- 多段アーキテクチャ(Multi-tiered Architecture)
  - 3つの論理レベルのプログラムを複数マシンに分散
  - マシンが2つ(クライアントとサーバ)の場合
    - 2段アーキテクチャ(Two-tiered architecture)

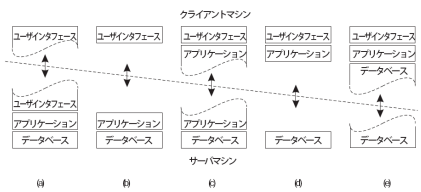


図1-29 その他のクライアントサーバ構成(a)～(e)

3段アーキテクチャ

- 3段アーキテクチャ(three-tiered architecture)
  - クライアントからリクエストを受けたサーバが、さらに別のサーバに(クライアントとして)リクエストを出す
  - 例)
    - ユーザインタフェースレベル:クライアント
    - 処理レベル:アプリケーションサーバ
    - データレベル:データベースサーバ

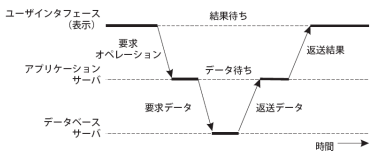
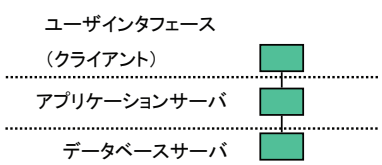


図1-30 サーバがクライアントとして動作する例

最新のアーキテクチャ

- 多段アーキテクチャのような分散化の手法
  - **垂直分散**(vertical distribution)
    - 異なる論理レベルのパートを複数の計算機に分散



最新のアーキテクチャ

- もう一つの分散化の手法
  - **水平分散**(horizontal distribution)
    - 同じ論理レベルのパートを複数の計算機に分散
      → 負荷分散を実現

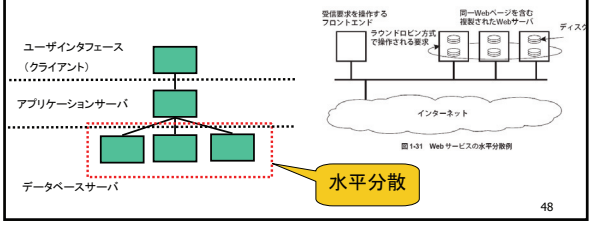
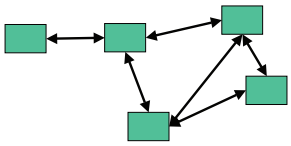


図1-31 Webサービスの水平分散例

最新のアーキテクチャ

- サーバの無いアーキテクチャ
  - ピアツーピア分散(peer-to-peer distribution)
    - クライアントのみから構成
    - 他のクライアントとの接点を探し、そこからサービスを受ける
    - 自分もまたサービスを提供する



1章のまとめ

- 分散システム:
  - あたかも一つの一貫したシステムのように振舞う、複数の独立した計算機群から構成
- 分散システムの目標
  - 透過性、開放性(Openness)、スケーラビリティ
- 分散システムの種類
  - 分散OS(Distributed Operating System,DOS)
  - ネットワークOS(Network Operating System, NOS)
  - ミドルウェアベース分散システム

1章のまとめ(続き)

- 分散システムの内部構成
  - クライアント-サーバモデル
  - 3つの論理レベル:
    - ユーザインタフェースレベル
    - プロセシングレベル
    - データレベル
- クライアント-サーバモデルのアーキテクチャ
  - 垂直分散(vertical distribution)
  - 水平分散(horizontal distribution)

演習問題

1. 各自が普段利用している計算機環境において、どのような透過性が実現されているか？ 透過性を実現している具体的な技術と共に説明せよ。
2. 分散システムにおけるミドルウェアの役割について説明せよ。
3. 垂直分散と水平分散の違いを説明せよ。