１． 分散システム（Distributed system）とは

定義：

統合化コンピュータ用ソフトウェアを整備して，ネットワークによって統合された自立コンピュータの集合体.

分散システムのアプリケーション

・ ユーザ群が利用する汎用コンピューティングシステム
・ 自動バンキングシステム
・ マルチメディア・コミュニケーションシステム

などといった，ほとんどすべてのコンピュータ商用/技術アプリケーションが含まれる.

分散システムの主要な特徴

（１） 資源共有（resource sharing）

☒ プリンタ，大規模ディスクなどといったハードウェア資源などの共有
☒ データやコンパイラなどのソフトウェア資源の共有

効率的に資源を共有するためには，各々資源は他の資源へアクセスし，手際よく処理され，そして確実に首尾一貫して更新されることを可能にする通信インタフェースを提供するプログラムによって管理されなくてはならない.

資源共有モデル

・ クライアントサーバモデル（client-server model）

定められたタイプの資源群に対して，資源管理者（resource manager）として振舞う複数のサーバプロセスと，共有するハードウェア，ソフトウェア資源へのアクセスを要求するタスクから構成されるクライアントプロセス群からなる.

クライアントサーバモデルでは，すべての資源はサーバプロセスによって管理される. このことから，クライアントサーバモデルにおいてサーバプロセスは，自身が管理する資源の集中型提供者とすることができる.

・ オブジェクト指向モデル（object-based model）

このモデルにおいて，実行するプログラムの実体は，操作へのアクセスを提供する，メッセージ通信インタフェースを持ったオブジェクトとされる. 分散システムにおけるオブジェクト指向モデルでは，各々の共有資源はオブジェクトとみなされる. オブジェクトは唯一のものと識別され，ネットワーク内のどこへでも移動することが可能である.

オブジェクト指向モデルを適用するための問題点として，オブジェクトはそれらの状態の表示を含み，オブジェクト管理者（object manager）はその状態にアクセスするためにオブジェクトと共に存在しなくてはならない. つまりオブジェクトがネットワーク内を移動すると，必ずオブジェクト管理者は移動先にコピーされなくてはならない.

（２） 開放性（openness）

開放性とは，システムをさまざまな方法で拡張できるかどうかを決定する特性である．分散システムの開放性とは，新しい資源共有サービスを，既存のサービスの破壊や重複なしに，どの程度追加できるかということを決定する．

各コンピュータのカーネル（kernel）は，システムコールによって他のプログラムからアクセスされる機能と資源を供給するために，コンピュータのハードウェアを管理し，制限する．BSD-UNIX やその他の OS のプロセス間通信と通信プロトコルの標準化は，コンピュータシステム設計で開放性を達成するための範囲を劇的に広げた．どのコンピュータのどの OS で実行されるサーバプロセスも，同じネットワーク上で実行しているクライアントプロセスへ資源を提供できるようになった．これらによって，アプリケーションプログラムへ提供される機能や資源は，ローカル OS がサポートするシステムコールによって制限されなくなった．

この方法によって資源共有機能を提供するように設計されたシステムは，拡張性があることから，開放型分散システムと呼ばれる．開放型分散システムでは，ハードウェアレベルでは，ネットワークにコンピュータを追加することによって，ソフトウェアレベルでは，新しいサービスを導入しアプリケーションプログラムに資源を共有させることによって拡張される．開放型分散システムの利点としては，個々の製造業者からの独立性である．

（３） 並行性（concurrency）

コンピュータに 1 つの処理装置があるならば，各プロセスは交互に実行される．コンピュータにｎ個の処理装置があるならば，ｎ個までのプロセスは同時に実行できる．すなわち並行（parallel）に処理でき，作業効率は単純計算でｎ倍である．

資源共有に基づく並列処理は 2 つの理由から発生する．

- 多数のユーザが同時にコマンドを投入したり，アプリケーションプログラム間で相互に作用しあう．
- 多数のサーバプロセスが並行に実行し，各クライアントプロセスから異なる要求に対して応答する．

前者は，1 つまたはそれ以上のアプリケーションプロセスが，各々の動作中のユーザのために実行することから発生する．後者は，各々のタイプの資源に対して，1 つまたはそれ以上のプロセスが存在する場合に発生する．

（４） 規模透過性（scalability）

分散システムは，多数の異なる規模で効果的かつ効率的に動作しなくてはならない．分散システムにおいて規模が拡大するとき，システムとアプリケーションソフトウェアに変更があってはならない．規模透過性は，現在の分散システムの構成要素においてかなり実現されている．

（５） フォールトトレラント性（fault tolerance）

フォールトトレラント性を有したコンピュータ設計方法

・ ハードウェア冗長性（Hardware Redundancy）：冗長な構成要素の使用
例：ファイルサーバの複製
・ ソフトウェア回復性（Software Recovery）：故障から回復するためのプログラム
例：パーマネントデータの回復，もしくはロールバック可能なソフトウェア開発

分散システムでは冗長性に優れた方法で実現できる．つまり重要なアプリケーション（Critical Application）が継続的に実行できるようにするために，必要なサーバを複数も足せることが可能である．

分散システムは，ハードウェアの故障に対して高度な可要性（availability）を供給する．分散システムでは 1 つのサーバが故障しても，影響はそのサーバの仕事だけであり，故障したサーバのプロセスやユーザは，他の正常なサーバに振り分けられる．

（6） 透過性（transparency）

定義：ユーザとアプリケーションプログラマから，分散システムの構成要素が分離していることを隠すこと

構成要素の分離とは，以下のことが可能であることを意味する．
・ プログラムを完全に並列に実行
・ システム全体の崩壊なしに，故障を抑制・回復
・ セキュリティ，プロテクションを実現するため通信チャネルを分離し制御のために使用
・ 構成要素の追加，削除によるシステムの成長，あるいは衰退

8 つの透過性

① アクセス透過性
ローカルと遠隔の情報オブジェクトを，まったく同じ操作によってアクセス可能にする

② 位置透過性
情報オブジェクトの位置に関する知識なしにそのオブジェクトにアクセス可能にする

③ 並行透過性
プロセス間に衝突なしに共有情報オブジェクトを利用して，並行操作を可能にする

④ 複製透過性
信頼性と性能を向上させるために，複数の情報オブジェクトの遂行を可能にする

⑤ 故障透過性
ユーザとアプリケーションプログラムに，故障を隠蔽し，その仕事の完了を可能にする

⑥ 移動透過性
情報オブジェクトが操作に影響なくシステム内を移動できることを可能にする

⑦ 性能透過性
システムの負荷の変化に従い，性能を改善するために再構成することを可能にする

⑧ 規模透過性
システムやアプリケーションに，システムの構造やアプリケーションアルゴリズムを変更することなく規模を拡張することを可能にする

２．　参考文献に関して

2.1　ORB・分散コンピューティングを用いた Web リンク収集

・ システム概要

- ☒ 分散オブジェクト技術 HORB を用いた分散コンピューティング環境下での，Web のハイパーリンクを収集するシステム
- ☒ Master-Slave 方式の負荷配分の採用
  - ☼ Master は初期条件（解析する URL 数，リンクの深さ）に基づいた Slave のタスク割り当てを行う

・ マシン構成

- ☒ Pentium2-400MHz，128M-SDRAM
- ☒ Master PC：1 台，Slave PC：10 台，PostgreSQL データベースサーバ：1 台

・ Master-Slave の働き

- ☒ Mater

Master は Slave のタスク量を決める URL 数とリンク収集の深さの初期変数を持ち，データベースから取得した URL データを Slave へわたす．

常に各 Slave のタスク処理を監視しており，タスクが終了した Slave に新しい URL タスクをわたす．

Slave からわたされたリンクデータをデータベースへ保存する．また Master はリンクデータから未解析 URL を新たなタスク URL として保存する．

- ☒ Slave

Slave は割り当てられた URL 先の Web ページを解析し，リンクを取得する．Slave はタスクを終了すると，取得したリンクデータを Master にわたす．

・ 実験結果について

実験結果を見てみると，Slave PC の台数が増えれば増えるほど，収集ページ数とリンク数の 1 台あたりの比率が落ちているのがわかる．実験内容が実時間を用いた実験であるならば，台数が多くなることによる，Master のタスク割り当てや通信レイテンシなどによって，1 台あたりの作業効率が落ちるということにも納得がいくが，実際どうだったのか？

・ Master-Slave 間のタスクの受け渡し法に関して

Master が Slave のタスクが終了するまで，次のタスクを Slave に割り当てないこの方法だと，負荷の均一化といったことを考える必要がなく，効率的に作業が行えるということになると考えられる．

ただしこの方法は，各 Slave PC が Web リンク収集といった固有の仕事のみを行い，PC の計算リソースにばらつきがない場合，または，タスクの 1 つ 1 つがそれぞれで完結していて，他のタスクに影響を及ぼさない場合に，効率的に作業が行えるといったものであり，Grid のように，実際のネットワーク環境に接続している PC を利用する場合には，やはり負荷の適切な割り当ての手法が必要となってくる．

例えば，Slave-A が計算リソース不足の状態で，Slave-B が計算リソースに余裕があると仮定する．互いに Master からのタスクを受けることができる状態であった場合，Master は両方の Slave に同様のタスクを割り当てるだろう．しかし，Slave-A ではタスクを受けても，そのタスクを処理するリソースは余っておらず，Slave-A に割り当てられた分のタスクは永遠に（Slave-A のリソースに余裕が出るまで）処理されることはない．そのため，作業効率を落としてしまうという結果になりかねない．

2.2 HORB を使用した分散 Web リンク収集システムに関して

・ システムの構造

計算機 10 台を，最も効率的なベーテ木（末端のノード以外がすべて 3 台の計算機と接続した木構造）を成す．

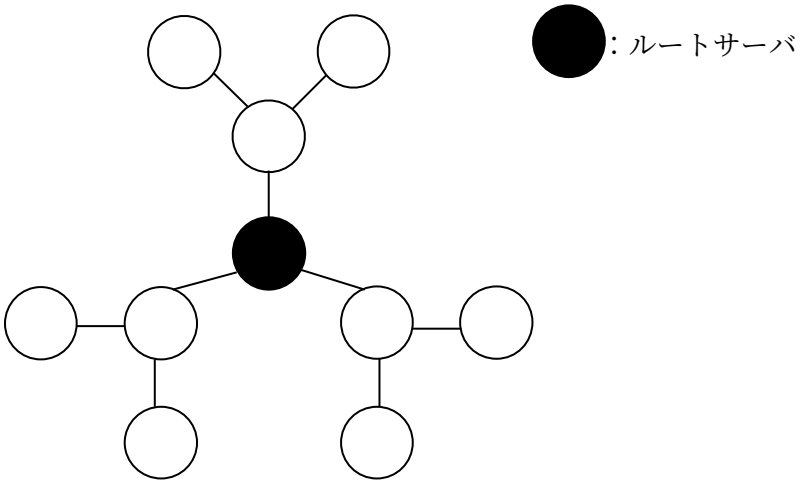

図 1．システムの木構造

・ 通信

このシステムでは, 各ノードは隣接したノードとのみ通信を行う. ここでいう通信とは,

☒ 計算機が収集した URL の共有

☒ 負荷を移動させることによる，各計算機の負荷均一化

・ 各計算機の働き

☒ Web リンクの収集

☒ URL データの管理

☒ 計算機間の通信

以上のタスクを，マルチスレッド処理で行う．

- 負荷均一化のアルゴリズム
  ① 負荷量・稼働時間などをトリガーとし，親サーバに負荷均一化を依頼する．依頼されたサーバがルートサーバでない場合には，さらに親サーバに依頼し，依頼をルートサーバまで送る
  ② ルートサーバは依頼を受けると，子サーバに現在の負荷量を送るように要求する．要求を受けたサーバはさらに自分の子サーバへ負荷量を要求し，ネットワーク全体の総負荷量を収集する．
  ③ ルートサーバから各サーバへ，均一な負荷量をブロードキャストする．
  ④ 各サーバは，自分の負荷が均一負荷量より大きい場合には，負荷量を親サーバへ送る．逆に自分の負荷が小さい場合には，親サーバの負荷を受け取る．
- 負荷量
  その計算機が持つ未探索 URL の数
- 負荷均一処理に移る条件
  前回の記録から負荷量が 10000 以上（高負荷の目安）増えている場合
- 実験結果
  ☒ 負荷均一化には成功したといえる
  ☒ 計算機 1 台あたり処理した URL 数：3000〜16000
  ☒ 全体で処理した URL 数：約 83000
  ☒ 総獲得リンク数：約 550000
  ☒ 負荷均一化開始から終了までに要した時間：数秒から 2 分以上

- システムネットワークの構造
  今回構成したネットワークが，なぜ木構造であったのか．
- 負荷の均一化について
  まず，負荷の均一化であるが，分散コンピューティングにおいて，ノードごとにかかる負荷にばらつきがでてくるのはわかる．この場合，システムで用いられた計算機は，他の用途がなく，URL の収集処理のみを行っている（と予想される）ため，負荷を均一化することは重要なのかもしれない．
  しかしながら，ただ単純に各計算機にかかる負荷を均一化するだけいいのかというと，少々疑問である．
  例えば，ある計算機 A は，ユーザがほとんど何の処理も行っていないため計算リソースが非常に余った状態であり，負荷も少ない状態とする．しかし計算機 B は，ユーザが何らかの処理に計算機を使用しているため，計算リソースが余っていない状況で，負荷が非常に多い状態だとする．こういった場合，負荷の均一化だけでは，両計算機の負荷が同一になるだけであり，計算機 B が少ない計算リソースで，負荷をこなさなくてはならない状況には変わりないため，負荷の均一化だけでは不十分であると考えられる．

３．　研究の方向性

3.1　研究内容

実際の Grid コンピューティングを考える場合，各ノードである計算機は，こちらの意図する処理以外を行わないわけではなく，各計算機内の計算リソースの状態は動的に変化すると考えられる．2 章でも述べたとおり，単純な負荷の均一化だけでは不十分であるため，動的に変化する計算リソースを考慮した負荷の割り当てというものが必要になると考えられる．

そこで，「動的に変化する計算リソースを考慮した負荷割り当ての実現」目的とした研究を行っていきたいと考えている．

3.2　研究の方針

①　計算対象の決定

まず，分散システムを構築する前に，どういった計算対象をシステムに与えるかということを決定しなくてはならない．現在，負荷均一化法を用いたシステムとの比較実験を考えているため，Web リンク収集を対象にしようかと思っている．

②　各計算機の働きの決定

分散システムを構成する計算機のネットワーク構造については，現在考案中．

各計算機には，負荷の割り当てや他の計算機の行動の監視などを行うサーバと，サーバから割り当てられたタスクをこなすクライアントが挙げられため，Client-Server 型のプログラミングが考えられるが，クライアント側が完全にクライアントとして行動するわけではなく，お互いが要求を出し合う形になるかもしれないため，Server-Server プログラミングも考慮していく必要がある．

③　分散システムの実装と実験

実際に分散システムを実装し，負荷割り当てが目的どおりかつ効率的にできているかを調べる．

④　負荷均一化手法との評価実験

実装したシステムと負荷均一化法を実装したシステムとで評価実験を行う．

４．　研究目的の実現方法

4.1　分散オブジェクト技術 HORB

HORB とは,

- Java 用分散オブジェクト技術である
- Sun の Java と 100%互換性があり，あらゆる処理系で動作する
- 商用でも無償で使える
- HORB 特有のリモートオブジェクトの生成

などといった数々の特徴を持ち，今回の研究にうってつけの技術であると思われる．

4.2　負荷割り当て方法のアルゴリズム概略

現在考えている，負荷割り当て方法のアルゴリズムの概略は，以下のとおりである．

1．各クライアント（ルートサーバ以外のサーバ）が，自分に割り当てられている負荷量と，自分の現在の計算リソースを考慮して，ルートサーバに対して，負荷の割り当ての見直しを要求する．
2．クライアントからの要求を受け取ったルートサーバは，現在の負荷量と計算リソースの状態を送るよう，全クライアントに要求をブロードキャストする．
3．各クライアントは，要求どおり自分の負荷量と計算リソースの状態をルートサーバに送信する．
4．ルートサーバは，各クライアントの状態から，適切な負荷量を算出し，各クライアントに負荷を送信する．

以上のアルゴリズムのうち，手順 4 で「各クライアントの状態から適切な負荷量を算出する」としているが，この適切な負荷量の算出方法については，現在考案中である．

夏季休業課題発表

知識科学研究科　博士前期課程

40 番　丹野聖司