10. クラスタシステム構築に関する知識 I

1. 科目の概要

HA (High Availability)クラスタと HPC (High Performance Computing)クラスタについて説明し、クラスタシステム構築に必要な技術を解説する。またクラスタシステムにおけるプログラミング技術として必須である並列プログラミングやマルチスレッドプログラミングの基礎も紹介する。

2. 習得ポイント

本科目の学習により習得することが期待されるポイントは以下の通り。

習得ポイント	説　明	シラバスの対応コマ
I-10-1. クラスタシステムの種類(HAクラスタとHPCクラスタ)	高可用性(HA)クラスタと高性能(HPC)クラスタについて、それぞれの概要と特徴、背景と歴史を説明する。また、各クラスタシステムの目的と位置づけを比較し、各システムに固有の技術を概説する。	1
I-10-2. 負荷分散(ロードバランサー)の種類と構成	HAクラスタの基本技術である負荷分散技術について、その基本的な概念、種類と構成および構成要素を解説する。負荷分散を実現するロードバランサーの様々な実装(レイヤ4/7)を紹介し、それぞれの構成、特徴について説明する。	1
I-10-3. 負荷分散アルゴリズムと死活監視	負荷分散を実現する代表的なアルゴリズム(ラウンドロビン等)を紹介し、その具体的な手順を説明する。またサーバの死活監視技術について触れ、HAクラスタ運用時に発生する作業について説明する。	1
I-10-4. LVS (Linux Virtual Server)によるロードバランサーの実現方法	LVS (Linux Virtual Server)を利用してHAクラスタを構築する方法について解説する。その構成要素を説明し、IPVSカーネルの構築、ipvsadmおよびkeepalivedのビルド、テスト環境の構築、DSR (Direct Server Return)の構築といった具体的な手順を示す。	2
I-10-5. VRRP(Virtual Router Redundancy Protocol)による冗長化	ルータを冗長化するVRRP (Virtual Router Redundancy Protocol)とは何かについて解説し、ネットワークの構成方法からkeepalivedの設定といった冗長化の実現方法を示す。さらにkeepalived運用時の留意点など実践的なポイントを紹介する。	3
I-10-6. スーパーコンピュータの歴史とクラスタによる実現	科学技術計算基盤としてのスーパーコンピュータについて、その概要と歴史、最近の動向を紹介する。近年急増しているクラスタによるスーパーコンピュータの実現方法について触れ、PCクラスタに関するいくつかのプロジェクトを紹介する。	4
I-10-7. 並列計算機の種類と構成	並列プログラミングの概念を理解するために、まず計算の高速化と並列処理の必要性を示す。また並列計算機の種類について触れ、プロセッサやメモリの結合方式、バスのトポロジーなど、構成方式の違いについて解説する。	5
I-10-8. 並列処理プログラミングの基本、並列化処理	並列プログラミングの基本的な技法と、並列処理の効率性について説明する。代表的なアルゴリズムについて並列化の技法を紹介し、さらに並列プログラミング環境と並列プログラミングで利用できるツールを紹介する。	5
I-10-9. マルチスレッドプログラミングの基礎	並列プログラミングの一種であるマルチスレッドプログラミングの基礎について解説する。スレッドとは何か、スレッドとプロセスの違い、スレッドの生成と終了など、スレッドプログラミングに関する基本的な概念を説明する。	6,7
I-10-10. スレッド間の通信とスレッドの同期	pthreadを題材として、具体的なマルチスレッドプログラミング方法を示す。またマルチスレッドプログラミングで必須の技術である同期とスレッド間通信の実現方法を紹介する。	6,7

【学習ガイダンスの使い方】

1. 「習得ポイント」により、当該科目で習得することが期待される概念･知識の全体像を把握する。
2. 「シラバス」、「IT 知識体系との対応関係」、「OSS モデルカリキュラム固有知識」をもとに、必要に応じて、従来の IT 教育プログラム等との相違を把握した上で、具体的な講義計画を考案する。
3. 習得ポイント毎の「学習の要点」と「解説」を参考にして、講義で使用する教材等を準備する。

3. IT 知識体系との対応関係

「10. クラスタシステム構築に関する知識Ⅰ」と IT 知識体系との対応関係は以下の通り。

科目名	基本レベル(Ⅰ)							応用レベル(Ⅱ)							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10. クラスタシステム構築に関する知識	<クラスタシステム概論、HA クラスタ(1)>	<HAクラスタ(2)>	<HAクラスタ(3)>	<コンピュータシミュレーション>	<並列プログラミング概論>	<並列プログラミング 実践(1) マルチスレッドプログラミング>		<並列プログラミング実践(2) HPF (High Performance Fortran)とOpenMP>	<並列プログラミング 実践(3) MPI (Message Passing Interface)>			<Beowulf PC クラスタの構築>	<SCoreクラスタ>	<PCクラスタの周辺技術>	<グリッド・コンピューティング>

[シラバス：http://www.ipa.go.jp/software/open/ossc/download/Model_Curriculum_05_10.pdf]

<IT 知識体系上の関連部分>

分野		科目名	1	2	3	4	5	6	7	8	9	10	11	12	13
組織関連事項と情報システム	1	IT-IAS 情報保証と情報セキュリティ	IT-IAS1. 基礎的な問題	IT-IAS2. 情報セキュリティの仕組み(対策)	IT-IAS3. 運用上の問題	IT-IAS4. ポリシー	IT-IAS5. 攻撃	IT-IAS6. 情報セキュリティ分野	IT-IAS7. フォレンジック(情報証拠)	IT-IAS8. 情報の状態	IT-IAS9. 情報セキュリティサービス	IT-IAS10. 脅威分析モデル	IT-IAS11. 脆弱性		
	2	IT-SP 社会的な観点とプロフェッショナルとしての課題	IT-SP1. プロフェッショナルとしてのコミュニケーション	IT-SP2. コンピュータの歴史	IT-SP3. コンピュータを取り巻く社会環境	IT-SP4. チームワーク	IT-SP5. 知的財産権	IT-SP6. コンピュータの法的問題	IT-SP7. 組織の中のIT	IT-SP8. プロフェッショナルとしての倫理的な問題と責任	IT-SP9. プライバシーと個人の自由				
応用技術	3	IT-IM 情報管理	IT-IM1. 情報管理の概念と基礎	IT-IM2. データベース問合わせ言語	IT-IM3. データアーキテクチャ	IT-IM4. データモデリングとデータベース設計	IT-IM5. データと情報の管理	IT-IM6. データベースの応用分野							
	4	IT-WS Webシステムとその技術	IT-WS1. Web技術	IT-WS2. 情報アーキテクチャ	IT-WS3. デジタルメディア	IT-WS4. Web開発	IT-WS5. 脆弱性	IT-WS6. ソーシャルソフトウェア							
ソフトウェアの方法と技術	5	IT-PF プログラミング基礎	IT-PF1. 基本データ構造	IT-PF2. プログラミングの基本的構成要素	IT-PF3. オブジェクト指向プログラミング	IT-PF4. アルゴリズムと問題解決	IT-PF5. イベント駆動プログラミング	IT-PF6. 再帰							
	6	IT-IPT 技術を統合するためのプログラミング	IT-IPT1. システム間通信	IT-IPT2. データ割り当てと交換	IT-IPT3. 統合的コーディング	IT-IPT4. スクリプティング手法	IT-IPT5. ソフトウェアセキュリティの実現	IT-IPT6. 種々の問題	IT-IPT7. プログラミング言語の概要						
	7	CE-SWE ソフトウェア工学	CE-SWE0. 歴史と概要	CE-SWE1. ソフトウェアプロセス	CE-SWE2. ソフトウェアの要求と仕様	CE-SWE3. ソフトウェアの設計	CE-SWE4. ソフトウェアのテストと検証	CE-SWE5. ソフトウェアの保守	CE-SWE6. ソフトウェア開発・保守ツールと環境	CE-SWE7. ソフトウェアプロジェクト管理	CE-SWE8. 言語翻訳	CE-SWE9. ソフトウェアのフォールトトレランス	CE-SWE10. ソフトウェアの構成管理	CE-SWE11. ソフトェアの標準化	
	8	IT-SIA システムインテグレーションとアーキテクチャ	IT-SIA1. 要求仕様	IT-SIA2. 調達/手配	IT-SIA3. インテグレーション	IT-SIA4. プロジェクト管理	IT-SIA5. テストと品質保証	IT-SIA6. 組織の特性	IT-SIA7. アーキテクチャ						
システム基盤	9	IT-NET ネットワーク	IT-NET1. ネットワークの基礎	IT-NET2. ルーティングとスイッチング	IT-NET3. 物理層	IT-NET4. セキュリティ	IT-NET5. アプリケーション分野	IT-NET6. ネットワーク管理							
	10	CE-NWK テレコミュニケーション	CE-NWK0. 歴史と概要	CE-NWK1. 通信ネットワークのアーキテクチャ	CE-NWK2. 通信ネットワークのプロトコル	CE-NWK3. LANとWAN	CE-NWK4. クライアントサーバコンピューティング	CE-NWK5. データのセキュリティと整合性	CE-NWK6. ワイヤレスコンピューティングとモバイルコンピューティング	CE-NWK7. データ通信	CE-NWK8. 組込み機器向けネットワーク	CE-NWK9. 通信技術とネットワーク概要	CE-NWK10. 性能評価	CE-NWK11. ネットワーク管理	CE-NWK12. 圧縮と伸張
			CE-NWK13. クラスタシステム [10-Ⅰ-1]	CE-NWK14. インターネットアプリケーション	CE-NWK15. 次世代インターネット	CE-NWK16. 放送									
	11	IT-PT プラットフォーム技術	IT-PT1. オペレーティングシステム	IT-PT2. アーキテクチャと機構	IT-PT3. コンピュータインフラストラクチャ	IT-PT4. デプロイメントソフトウェア	IT-PT5. ファームウェア	IT-PT6. ハードウェア							
	12	CE-OPS オペレーティングシステム	CE-OPS0. 歴史と概要	CE-OPS1. 並行性	CE-OPS2. スケジューリングとディスパッチ	CE-OPS3. メモリ管理	CE-OPS4. セキュリティと保護	CE-OPS5. ファイル管理	CE-OPS6. リアルタイムOS	CE-OPS7. OSの概要	CE-OPS8. 設計の原則	CE-OPS9. デバイス管理	CE-OPS10. システム性能評価		
コンピュータアーキテクチャとプロセッサ	13	CE-CA0 コンピュータのアーキテクチャと構成	CE-CA00. 歴史と概要	CE-CA01. コンピュータアーキテクチャの基礎	CE-CA02. メモリシステムの構成とアーキテクチャ	CE-CA03. インタフェースと通信	CE-CA04. デバイスサブシステム	CE-CA05. CPUアーキテクチャ	CE-CA06. 性能・コスト評価	CE-CA07. 分散・並列処理 [10-Ⅰ-5]	CE-CA08. コンピュータによる計算	CE-CA09. 性能向上			
複数領域にまたがるもの	14	IT-ITF IT基礎	IT-ITF1. ITの一般的なテーマ	IT-ITF2. 組織の問題	IT-ITF3. ITの歴史	IT-ITF4. IT分野(学科)とそれに関連のある分野(学科)	IT-ITF5. 応用領域	IT-ITF6. IT分野における数学と統計学の活用							
	15	CE-ESY 組込みシステム	CE-ESY0. 歴史と概要	CE-ESY1. 低電力コンピューティング	CE-ESY2. 高信頼性システムの設計	CE-ESY3. 組込み用アーキテクチャ	CE-ESY4. 開発環境	CE-ESY5. ライフサイクル	CE-ESY6. 要件分析	CE-ESY7. 仕様定義	CE-ESY8. 構造設計	CE-ESY9. テスト	CE-ESY10. プロジェクト管理	CE-ESY11. 並行設計(ハードウェア、ソフトウェア	CE-ESY12. 実装
			CE-ESY13. リアルタイムシステム設計	CE-ESY14. 組込みマイクロコントローラ	CE-ESY15. 組込みプログラム	CE-ESY16. 設計手法	CE-ESY17. ツールによるサポート	CE-ESY18. ネットワーク型組込みシステム	CE-ESY19. インタフェースシステムと混合信号システム	CE-ESY20. センサ技術	CE-ESY21. デバイスドライバ	CE-ESY22. メンテナンス	CE-ESY23. 専門システム	CE-ESY24. 信頼性とフォールトトレランス	

4. OSS モデルカリキュラム固有の知識

OSS モデルカリキュラム固有の知識として、Linux 上で動作するクラスタシステムがある。High Availability Cluster(HAC)と High Performance Computing(HPC)に関する知識を Linux 上の OSS 実装を通して習得する。

科目名	第1回	第2回	第3回	第4回	第5回	第6回	第7回
10.クラスタシステム構築に関する知識1	(1) High Availability(HA) Cluster とHigh Performance Computing(HPC) Cluster (2) HA クラスタ概論	(1) LVS で実現するロードバランサー	(1) ロードバランサーの冗長化 (2) keepalived の運用のテクニック	(1) 科学技術研究基盤としてのコンピュータシミュレーション (2) スーパーコンピュータ動向	(1) 計算の高速化と並列処理の必要性 (2) 並列計算機の分類 (3) 並列計算機の構成方式 (4) プロセッサ間のネットワー (5) 並列処理のプログラミング (6) 並列処理の効率 (7) 数値計算の並列化 (8) 並列プログラミング環境とツール	(1) マルチスレッドプログラミング入門 (2) マルチスレッドプログラミングの基礎	

（網掛け部分は IT 知識体系で学習できる知識を示し、それ以外は OSS モデルカリキュラム固有の知識を示している）

スキル区分	OSS モデルカリキュラムの科目	レベル
ネットワーク分野	10　クラスタシステム構築に関する知識 I	基本
習得ポイント	I-10-1. クラスタシステムの種類(HA クラスタと HPC クラスタ)	
対応する コースウェア	第 1 回 (クラスタシステム概論、HA クラスタ (1))	

I-10-1. クラスタシステムの種類(HA クラスタと HPC クラスタ)

HA クラスタと HPC クラスタについて、それぞれの概要と特徴、背景と歴史を説明する。また、各クラスタシステムの目的と位置づけを比較し、各システムに固有の技術を概説する。

【学習の要点】

* 複数台のコンピュータから構成されるクラスタシステムにより、一台のコンピュータでは得られないサービスを提供できる。
* HA クラスタにより、一台のコンピュータでは得られない高可用性を実現できる。
* HPC クラスタにより、一台のコンピュータでは得られない高性能を実現できる。

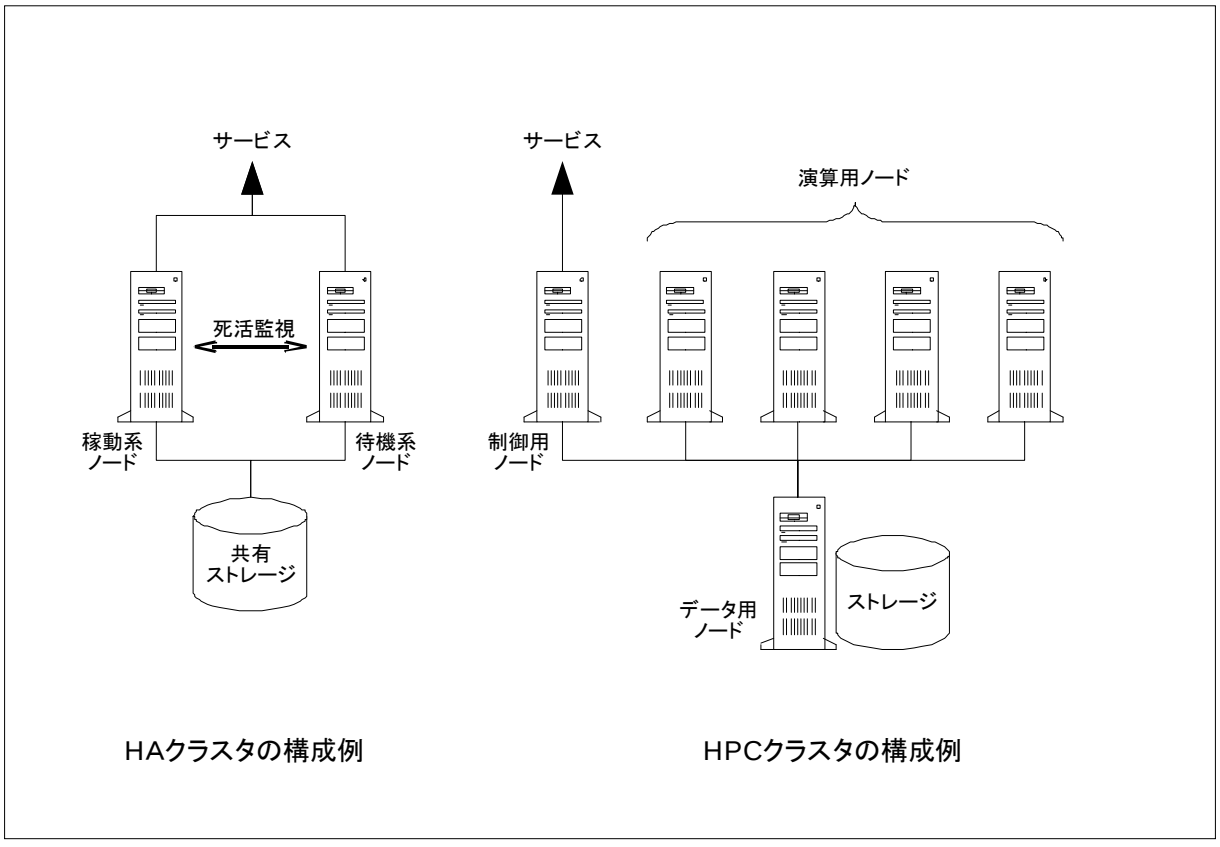

図 I-10-1.　HA クラスタと HPC クラスタ

【解説】

1) クラスタとは

複数台のコンピュータを協調動作させ、全体としてサービスを提供するシステムのことを、クラスタ(コンピュータ・クラスタ、クラスタシステム)という。そのなかで、クラスタを構成するコンピュータの一台一台のことをノードと呼ぶ。

クラスタには、高い可用性を目的とした HA(High Availability)クラスタと、高性能な計算環境を実現することを目的とした HPC(High Performance Computing)クラスタの2種類がある。

2) HA クラスタ(High Availability Cluster)

一台のコンピュータでは得られない高可用性の確保を目的としたクラスタシステムを HA クラスタという。HA クラスタはさらに、その実現方式によりフェイルオーバークラスタと負荷分散クラスタの2種類に分けられる。

＊ フェイルオーバークラスタ

同一の役割を持たせた複数台のコンピュータを、実際にサービスを提供する稼働系(現用系)ノードと、それをバックアップするための待機系ノードの2グループに分ける。稼働系に障害が発生した場合、待機系のノードを稼働系ノードに切り替える。このような仕組みでサービスを中断させることなく可用性を高めることを目的としたクラスタが、フェイルオーバークラスタである。これらはまた Active-Standby 型クラスタや高可用密結合クラスタとも呼ばれる。通常、単に HA クラスタと言った場合、このフェイルオーバークラスタを指す。

＊ 負荷分散クラスタ

同一の役割を持たせた複数台のコンピュータを、実際にサービスを提供する稼働系ノードとして並列に動作させる。このシステムでは、一台が停止しても残りのノードでサービスを継続することが可能となる。さらに、この仕組みによって高可用性を確保すると同時に、複数ノードで処理を分担することにより、一台のサーバでは得られなかった処理性能を確保することを目的としたクラスタシステムが、負荷分散クラスタである。Active-Active 型、あるいは密結合並列クラスタとも呼ばれる。

3) HPC クラスタ(High Performance Computing Cluster)

複数台のコンピュータを結合させて演算処理を分担し、一台のコンピュータでは得られない高性能を確保することを目的としたクラスタを HPC クラスタという。HPC クラスタはスーパーコンピュータの一形態ととらえることができる。通常は、主に大規模な演算を必要とする数値解析、構造解析やシミュレーションといった科学技術計算に HPC クラスタが利用される。HA クラスタとは異なり、各ノードの役割は一律ではない。HPC クラスタを構成するノードは、分担して演算を行う複数台の演算用ノード、演算用ノードを取りまとめる制御用ノード、データベースを扱うデータ用ノードなど、様々な役割をそれぞれが担っている。

スキル区分	OSS モデルカリキュラムの科目	レベル
ネットワーク分野	10　クラスタシステム構築に関する知識 I	基本
習得ポイント	I-10-2. 負荷分散(ロードバランサー)の種類と構成	
対応する コースウェア	第 1 回 (クラスタシステム概論、HA クラスタ (1))	

I-10-2. 負荷分散(ロードバランサー)の種類と構成

HA クラスタの基本技術である負荷分散技術について、その基本的な概念、種類と構成および構成要素を解説する。負荷分散を実現するロードバランサーの様々な実装を紹介し、それぞれの構成、特徴について説明する。

【学習の要点】

＊ 負荷分散を実現する方式には、代表的なもので、DNS ラウンドロビン、L4 ロードバランサー、L7 ロードバランサーの3通りがある。

＊ DNS ラウンドロビンは、DNS の問合せ結果を順繰りに変えることで負荷分散を実現する方式である。具体的には、DNS 正引き問い合わせの結果として返される IP アドレスとしてクラスタ内各ノードの IP を順番に返し、クライアントからのアクセスをクラスタ内の複数ノードへ分散させる。

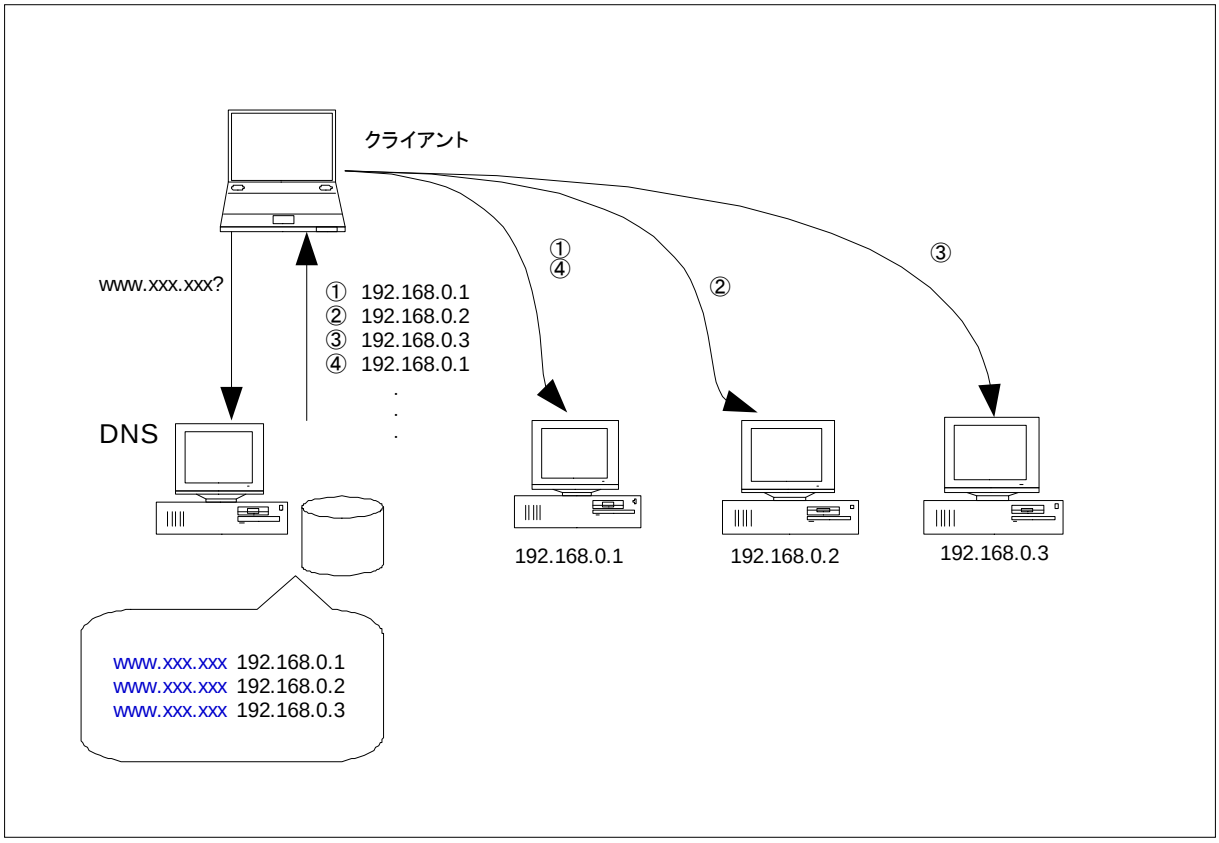

図 I-10-2.　DNS ラウンドロビン

【解説】

1）負荷分散方式の種類

負荷分散を実現する代表的な方式には、DNS ラウンドロビン、L4 ロードバランサー、L7 ロードバランサーがある。

＊ DNS ラウンドロビン

DNS ラウンドロビンは、DNS の問合せに対する回答を順繰りに変えることによって負荷を分散させる方式である。具体的には、DNS 正引き問い合わせの結果として返される IP アドレスとして、クラスタ内各ノードの IP を順番に返す。その結果としてクライアントからのアクセスをクラスタ内の複数ノードへ分散させることが可能となる。DNS ラウンドロビンによる負荷分散システムは、DNS サーバと、サービスを提供する複数のリアルサーバ(ノード)から構成される。DNS サーバの設定のみで実現でき最も容易に実現することができる。しかし以下のような問題点もある。

- サービスを遂行する際に複数回のアクセスが必要となる場合、セッションを保持できない可能性がある。同一のクライアントからアクセスがあったとしても、アクセスの度に接続するサーバが異なる可能性があるため、サーバ側にセッション情報を保持できないからである。
- DNS によって返される情報(レコード)の内容によっては、クライアントにキャッシュが長く残ることがある。その場合、キャッシュが有効な間は特定のノードにのみアクセスされてしまうことになる。
- 特定のクライアントが 1 つのノードに対して負荷の高いアクセスを続ける場合に効率よく負荷分散を実現することができない。DNS ラウンドロビンではノードの負荷に関係なく各ノードの IP を順番に返すので、負荷分散効率が悪い。

＊ L4 ロードバランサー

OSI 参照モデルにおける第4層、トランスポート層の情報を利用して負荷分散を行う方法が L4 ロードバランサーである。本方式では、ロードバランサー専用機と、サービスを提供する複数のリアルサーバによって負荷分散が実現される。本方式の場合、ロードバランサーが停止するとシステム全体の停止してしまう。そのような状況を避けるために、ロードバランサーを複数台用意し、冗長化構成を取るという措置が一般的である。ロードバランサー専用機を独立して用意せず、負荷分散を実現するためのソフトウェアを、リアルサーバ上に共存させることでロードバランサーを実現する場合もある。

＊ L7 ロードバランサー

L7 ロードバランサーとは、OSI 参照モデルにおける第7層、アプリケーション層の情報を利用して負荷分散を行う装置である。L4 ロードバランサーと同様に、ロードバランサーを実現する機器と複数台のリアルサーバから構成される。

オープンソースによる L7 ロードバランサーの代表的な実装例を以下に挙げる。

- HA Proxy http://haproxy.1wt.eu/
- mod_proxy_balancer http://httpd.apache.org/docs/2.2/en/mod/mod_proxy_balancer.html
- POUND http://www.apsis.ch/pound/
- Perlbal http://www.danga.com/perlbal/
- LVS http://www.linuxvirtualserver.org/

スキル区分	OSS モデルカリキュラムの科目	レベル
ネットワーク分野	10　クラスタシステム構築に関する知識 I	基本
習得ポイント	I-10-3. 負荷分散アルゴリズムと死活監視	
対応する コースウェア	第 1 回（クラスタシステム概論、HA クラスタ（1））	

I-10-3. 負荷分散アルゴリズムと死活監視

負荷分散を実現する代表的なアルゴリズムを紹介し、その具体的な手順を説明する。またリアルサーバの死活監視技術について触れ、HA クラスタ運用時に発生する作業について説明する。

【学習の要点】

* 負荷分散システムでは、死活監視により障害が検出された場合でも残りの部分で稼動を続けるため、高可用性を実現することができる。
* 負荷分散を実現する代表的なアルゴリズムに、ラウンドロビン、リーストコネクション、ハッシュ、ファーストアンサーがある。
* リアルサーバの死活監視方法として、ICMP、TCP、独自プロトコルによるものなどがある。
* 負荷分散と死活監視は別個に動作する。したがってシステム全体として適切に動作させるためには、障害検出時に負荷分散対象の情報を更新するよう連携する必要がある。

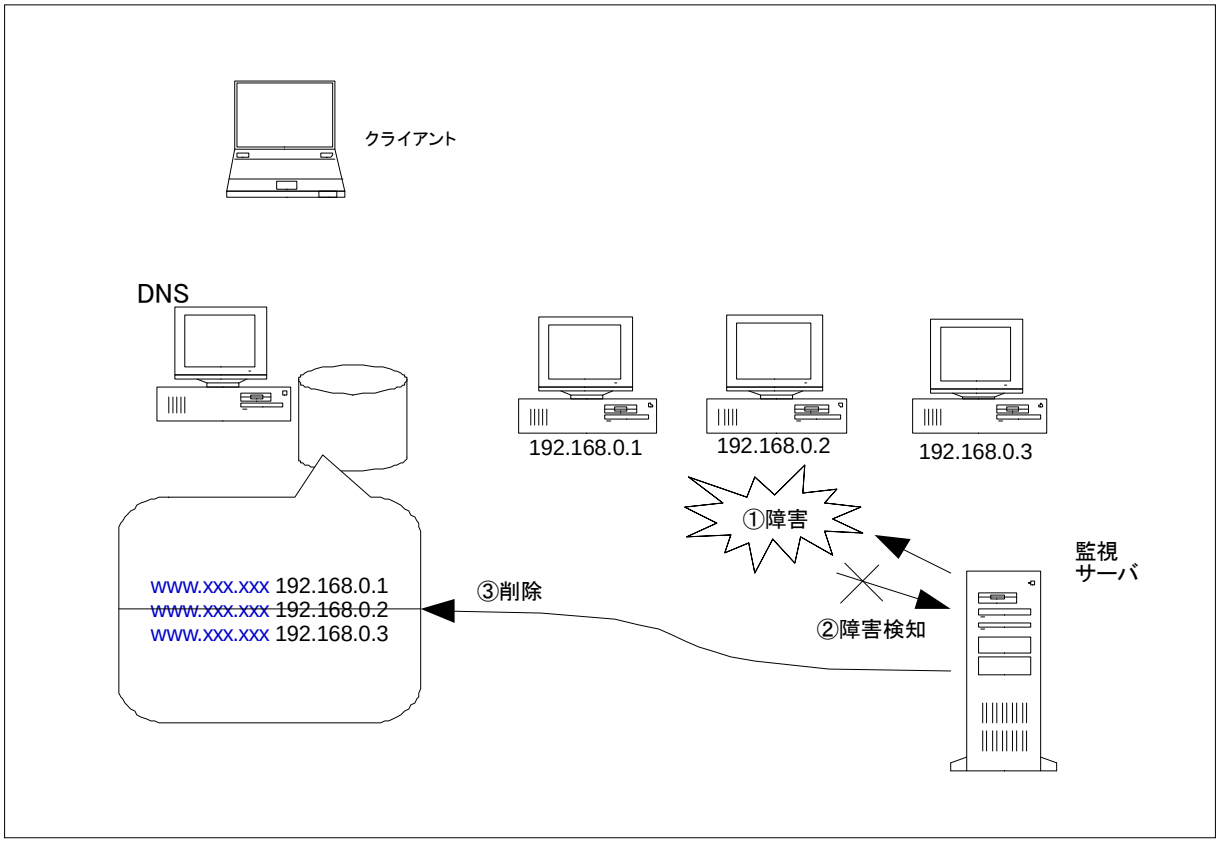

図 I-10-3.　ラウンドロビン時の死活監視

【解説】

1) リアルサーバを選出するアルゴリズム

負荷分散において、ロードバランサーがリアルサーバを選出するアルゴリズムには、以下のようなものがある。

＊ ラウンドロビン(round robin)

クライアントからの要求のたび、各リアルサーバを順々に選出する。各リアルサーバの負荷を考慮せずに接続すべきサーバを選出するため、分散効率は悪い。

＊ リーストコネクション(least connections)

クライアントからの接続数が最も少ないリアルサーバを選出する方式である。各リアルサーバの負荷としてサーバの稼働状況は考慮されず、接続数のみが考慮される。

＊ ハッシュ(hash)

クライアントの IP アドレスにハッシュ関数を適用し、そのハッシュ値をもとにリアルサーバを選出する。各リアルサーバの負荷はまったく考慮されないが、同じクライアントからのアクセスに対し同じリアルサーバを選出できるので、複数回のアクセスに跨るセッションを適切に処理することが可能となる。

＊ ファーストアンサー(first answer)

応答速度が最も早いリアルサーバを選出する。各リアルサーバの負荷として、応答速度のみが考慮される。

2) リアルサーバの死活監視方式

負荷分散を実現するシステムでは、各ノードの死活状況を管理する必要がある。障害が発生してサービスを提供できなくなったサーバは、すみやかに振り分け対象から除外されなければならない。リアルサーバの死活監視を行う方式のうち代表的な方法を以下に示す。

＊ ICMP

リアルサーバが正常に稼動しているかどうかを判断するために、ICMP Echo Request(ping)を利用する。ping を送信し、応答が返ってきたかどうかで死活状況を判定する。ただし、ICMP による方法では、対象サーバが IP レベルで機能していることは確認可能であるが、アプリケーションレベルで機能しているかどうかを確かめることはできない。

＊ TCP

監視対象のリアルサーバに対し、TCP 接続の確立を試みる。TCP 接続が確立できた場合、そのリアルサーバは正常稼動していると判断する。この方法では、対象のリアルサーバが提供するサービス(アプリケーション)に対する TCP ポートへの接続を試みることで、サービスが要求を受信可能な状態であるかどうかまでを確かめることができる。

＊ その他のプロトコル、独自プロトコル

監視方法を独自に取り決め、その方法に従って、監視対象のリアルサーバへアクセスする。その結果として決められた応答がかえってきた場合に、そのリアルサーバは正常稼動していると判断する。コネクションレスである UDP 上のサービスに対し監視を行う場合や、監視対象のサービスが利用できるかどうかまで確認するよう必要がある場合に、この方法が利用される。例えば、Web サーバに対し、ホームページへのアクセスが可能かどうかを確認するために監視用のページを用意しておき、当該 URI へのリクエストに対して正常にレスポンスが返ってくるかどうかで判断する、という方法がある。

スキル区分	OSS モデルカリキュラムの科目	レベル
ネットワーク分野	10　クラスタシステム構築に関する知識 I	基本
習得ポイント	I-10-4. LVS (Linux Virtual Server)によるロードバランサーの実現方法	
対応する コースウェア	第 2 回 (HA クラスタ (2))	

I-10-4. LVS (Linux Virtual Server)によるロードバランサーの実現方法

LVS (Linux Virtual Server)を利用して HA クラスタを構築する方法について解説する。その構成要素を説明し、IPVS カーネルの構築、ipvsadm および keepalived のビルド、テスト環境の構築、DSR (Direct Server Return)の構築といった具体的な手順を示す。

【学習の要点】

* LVS(Linux Virtual Server)は、Linux サーバにおいて、HPC、HA、ロードバランサの機能を動作させるソフトウェア群である。
* LVS を用いたロードバランサーは、IPVS 対応カーネル、ipvsadm、keepalived、iptables により構成される。
* LVS による負荷分散には、振り分け時のアドレス変換方式によって主に NAT 方式と DSR 方式の2通りがある。
* LVS でのロードバランサは、Linux ディストリビューションによっては、モジュールをロードするだけで構築することができる。

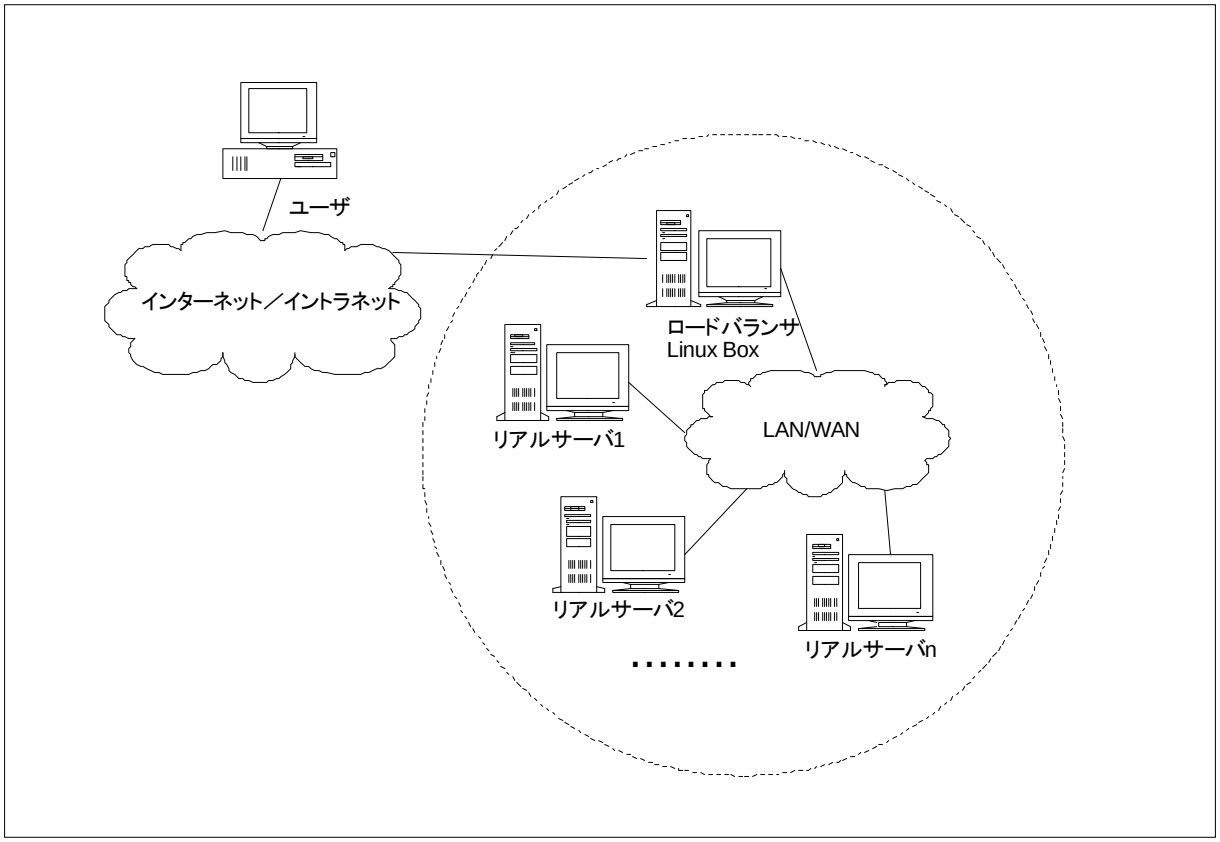

図 I-10-4.　LVS の概念

【解説】

1) LVS を用いたロードバランサーの構成要素

＊ IPVS

Linux カーネル内に実装された IP ロードバランサー

＊ ipvsadm

IPVS の負荷分散ルールを設定するための管理ツール

＊ keepalived

LVS の分散先リアルサーバの死活監視を行うデーモン

＊ iptables

Linux カーネルに実装されたパケットフィルタリング機能およびそれを操作するコマンド

＊ iproute

IP アドレスやルーティングテーブルを操作するための管理ツール

2) NAT 方式と DSR 方式について

LVS による負荷分散には、振り分け時のアドレス変換方式によって主に NAT 方式と DSR 方式の2通りがある。

＊ NAT

ロードバランサーは、クライアントからのリクエストを受け取り、宛先 IP アドレスをリアルサーバのアドレスに書き換えて、リアルサーバへ転送する。また、リアルサーバからクライアントへのレスポンスを受け取り、送信元 IP アドレスをロードバランサーのアドレスに書き換えてからクライアントへ返す。このように、ロードバランサーにて、往路では宛先 IP、復路では送信元 IP の書き換えを行う。

＊ DSR (Direct Server Return)

リアルサーバをまとめるスイッチと独立にロードバランサーが存在し、負荷分散を行った結果のレスポンスがサーバからロードバランサーを介さず直接に返送される方式である。NAT 方式と異なり、DSR 方式ではクライアントとリアルサーバの間でデータを中継する際に IP アドレスの書き換えを行わない。そのため、IP ヘッダに書かれる送信元、あるいは宛先の IP アドレスが、実際の IP と異なる局面が生じ問題が生じる可能性がある。これを正しく処理するためには、iptables と iproute (ip コマンド)を用いて専用の設定を行わなければならない。

3) LVS を用いたロードバランサーの構築

＊ インストール

IPVS は、最近の RedHat 系 Linux ディストリビューションでは、標準でローダブルモジュールとして用意されていることが多く、モジュールをロードするだけで利用可能である。

また ipvsadm および keepalived についても、最近の RedHat 系 Linux ディストリビューションでは標準で RPM パッケージとして用意されていることが多く、このパッケージをインストールするだけで使用することができる。

＊ テスト環境の構築

DSR 方式により、LVS を用いたロードバランサーのテスト環境を構築する。1台のロードバランサー用サーバから、2 台の Web サーバへアクセスを分散させる。ロードバランサー用サーバに IPVS を設定し、後方にある 2 台の Web サーバへアクセスを分散させる。

スキル区分	OSS モデルカリキュラムの科目	レベル
ネットワーク分野	10　クラスタシステム構築に関する知識 I	基本
習得ポイント	I-10-5. VRRP(Virtual Router Redundancy Protocol)による冗長化	
対応する コースウェア	第 3 回 (HA クラスタ (3))	

I-10-5. VRRP(Virtual Router Redundancy Protocol)による冗長化

VRRP (Virtual Router Redundancy Protocol)とは何かについて解説し、ネットワークの構成方法から keepalived の設定といった冗長化の実現方法を示す。さらに keepalived 運用時の留意点など実践的なポイントを紹介する。

【学習の要点】

* VRRP により、ルーターの高可用性を実現できる。
* keepalived は、VRRP を用いたロードバランサーの冗長化が可能である。
* プリエンプティブモードは、最もプライオリティが高いルーターが故障から復旧した場合の動作を指定するパラメーターである。

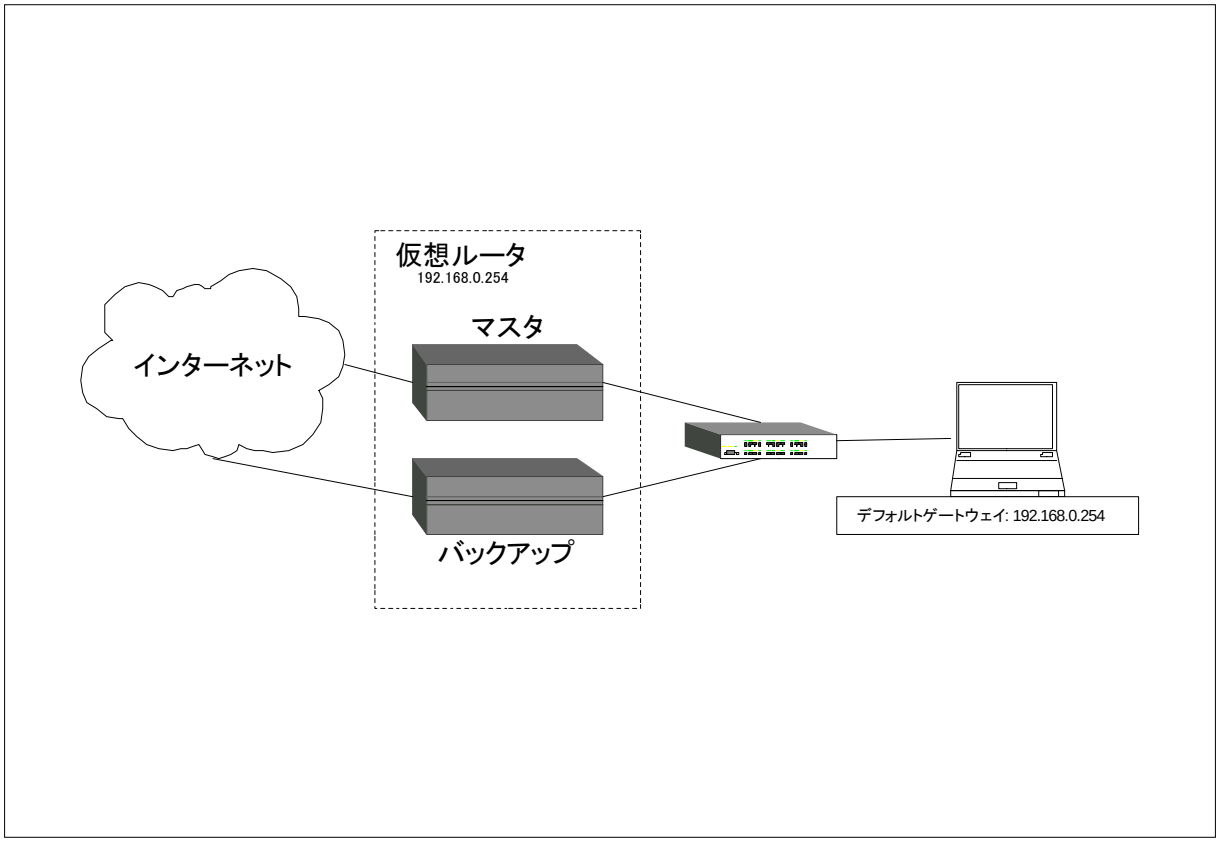

図 I-10-5.　VRRP 概念図

【解説】

1) VRR (Virtual Router Redundancy Protocol)とは

VRRP とは、稼動系のルーター(マスター・ルーター)と待機系のルーター(バックアップ・ルーター)とを組み合わせて仮想的に 1 台のルーターとみなす、ルーターの冗長構成を実現するためのプロトコルである。稼動系に障害が発生した場合、その障害を検知した待機系が、稼動系に自動的に切り替わる(これをフェイルオーバーと言う)。

VRRP ではまず、マスター・ルーターが、自分自身が正常に稼働中であることを周囲に示す。具体的には Advertisement Message をマルチキャスト送信することで稼働状況を通知する。バックアップ・ルーターは、この Advertisement Message を受信している限り、マスター・ルーターが正常稼働状態にあると判断する。一定時間この Advertisement Message が受信されなかった時には、バックアップ・ルーターはマスター・ルーターになるべく Advertisement Message を送信する。その際に、自分自身よりもプライオリティの高いバックアップ・ルーターからの Advertisement Message を受信しなければ、そのバックアップ・ルーターがマスター・ルーターとして稼働するようになる。

2) VRRP によるロードバランサーの冗長化

I-10-4 の「テスト環境の構築」で構築した Web サーバの負荷分散環境では、ロードバランサーが1台のみであった。この構成では、ロードバランサーが障害等により停止した場合に、Web サーバへアクセスできなくなる。高い信頼性が求められる環境では、サーバ1台の障害でサイト全体が停止することは許されず、冗長化を行う必要がある。なお、ロードバランサーがひとつしかなくそれが動作しているか否かでシステム全体が有効か否かを左右する状態を、ロードバランサーが単一障害点(Single Point of Failure)となっている、という。

keepalived は、VRRP を用いて、ロードバランサーの冗長化構成を保つためのソフトウェアである。バックアップ用のロードバランサー機を増設し、keepalived に VRRP による冗長化設定を行うことで、マスター用のロードバランサーが停止したとしても、バックアップ用のロードバランサー機がマスターとなり負荷分散機能を継続させることが可能となる。このようにバックアップ機器がロードバランサーの役割を引き継ぐことで、システム全体の停止を防ぐことができる。

3) プリエンプティブモードについて

プリエンプティブモードは、最もプライオリティが高いルーターが故障から復旧した場合の動作を指定するパラメーターである。プリエンプティブモードを有効にすると、復旧したルーターがマスターとなる(そのルーターが、復旧時点で動作しているルーターよりもプライオリティが高い場合)。このとき、プライオリティが高いルーターが頻繁に停止と復旧を繰り返すような動作をしたとすると、マスター・ルーターの交代が頻繁に発生し、その結果、ネットワークは不安定となる。このような状況を避けるためには、プリエンプティブモードを無効とすればよい。プリエンプティブモードが無効であれば、プライオリティが高いルーターが復旧した時にマスターを交代しないようにすることができ、システム全体の安定性を高めることが可能である。

スキル区分	OSS モデルカリキュラムの科目	レベル
ネットワーク分野	10　クラスタシステム構築に関する知識 I	基本
習得ポイント	I-10-6. スーパーコンピュータの歴史とクラスタによる実現	
対応する コースウェア	第 4 回（コンピュータシュミレーション）	

I-10-6. スーパーコンピュータの歴史とクラスタによる実現

科学技術計算基盤としてのスーパーコンピュータについて、その概要と歴史、最近の動向を紹介する。さらにクラスタによるスーパーコンピュータの実現方法について触れ、PC クラスタに関するいくつかのプロジェクトを紹介する。

【学習の要点】

* 現在のスーパーコンピュータは、クラスタによるものが主流を成している。
* クラスタによってスーパーコンピュータを実現するには、並列処理に対応したプログラミングが必須である。

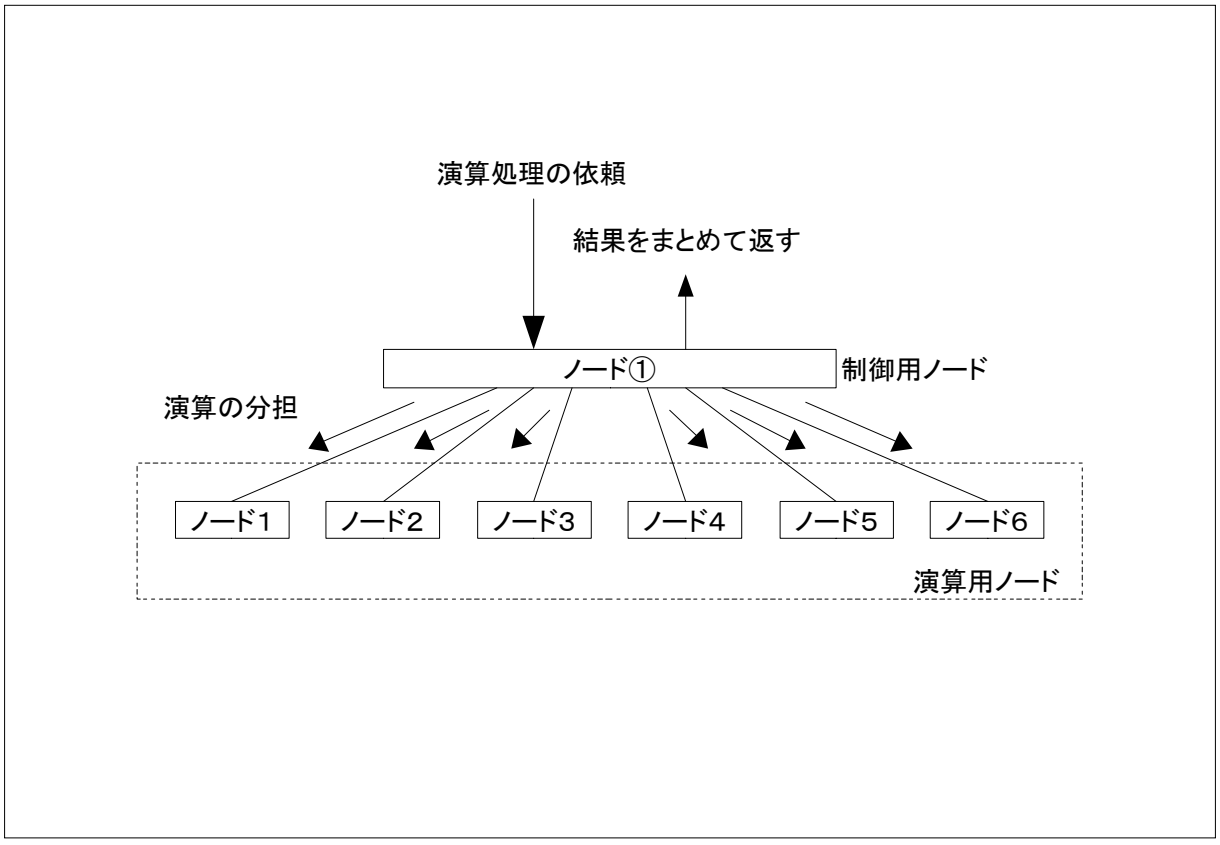

図 I-10-6.　HPC クラスタ概念図

【解説】

1) スーパーコンピュータの概要と歴史

大規模な演算を必要とする数値解析、構造解析やシミュレーションといった科学技術計算での利用を主目的とした、非常に高速な演算処理を行うコンピュータあるいは複数台のコンピュータからなるシステムを、スーパーコンピュータという。

2) スーパーコンピュータの歴史

スーパーコンピュータの誕生は、1970 年代にさかのぼる。当初は、一度に複数の演算を処理できる非常に高速なプロセッサをスーパーコンピュータ専用に独自に開発したものが多かった。その特殊性と開発に多大なリソースがつぎ込まれた結果、スーパーコンピュータは非常に高価なコンピュータとして認識され、導入する組織も限られたものであった。その後、汎用プロセッサを搭載したコンピュータを数多く並べて並列処理させる、クラスタ型のスーパーコンピュータが登場した。現在では、価格性能比の点から、クラスタ型が主流となっている。

3) クラスタによるスーパーコンピュータの実現

クラスタ型のスーパーコンピュータは、HPC (High Performance Computing)クラスタと呼ばれる。クラスタによってスーパーコンピュータを実現するには、分担して演算を行う演算用ノードを多数、結合する。さらに演算用ノードを取りまとめる制御用ノードが用意される。必要に応じてデータベースを扱うデータ用ノードとなどのコンピュータが用意されることもある。

複数の演算用ノードで並列処理を行うには並列処理に対応したプログラムが必要であり、ノード間での同期、処理待ち、データ送受信といった処理を組み込むことになる。これらの処理は特殊なプログラミング技術が必要となるが、並列プログラミングに特化したプログラミング言語や、並列化を実現する部分をライブラリとして用意し簡単に利用できるようにしたプログラミング環境も存在する。これらを利用することで、比較的容易にクラスタ型スーパーコンピュータを利用することもできる。

4) 主要な HPC クラスタに関するプロジェクト

ここでは、HPC クラスタに関する主要なプロジェクトをいくつか挙げる。

* Beowulf　　http://www.beowulf.org/
  Linux と OSS で実現
* Score　　http://www.pccluster.org/
  日本で開発
* TSUBAME (Tokyo-tech Supercomputer and UBiquitously Accessible Mass-storage Environment)　　http://www.gsic.titech.ac.jp/~ccwww/
  東京工業大学に導入されている
* PACS-CS　　http://www.ccs.tsukuba.ac.jp/PACS-CS/
  筑波大学で開発

スキル区分	OSS モデルカリキュラムの科目	レベル
ネットワーク分野	10　クラスタシステム構築に関する知識 I	基本
習得ポイント	I-10-7. 並列計算機の種類と構成	
対応する コースウェア	第 5 回（並列プログラミング概論）	

I-10-7. 並列計算機の種類と構成

並列プログラミングの概念を理解するために、まず計算の高速化と並列処理の必要性を示す。また並列計算機の種類について触れ、プロセッサやメモリの結合方式、バスのトポロジーなど、構成方式の違いについて解説する。

【学習の要点】

＊ 近年のスーパーコンピュータは価格性能比の観点から、汎用プロセッサによるノードを多数結合して並列に動作させることにより、全体として演算処理性能のスループットを得る「HPC クラスタ」によるスーパーコンピュータが主流である。

＊ HPC クラスタの性能を引き出すためには、その上で動作するプログラムが並列処理に対応していなければならない。

＊ 並列計算機のメモリ結合方式には、共有メモリモデルである UMA と NUMA、メッセージ交換モデルである NORA がある。

＊ インターコネクトのトポロジーには、共有メモリ型ではバスベースおよびスイッチベース、分散メモリ型ではメッシュ型や動的ネットワーク型、完全結合ネットワーク型などがある。

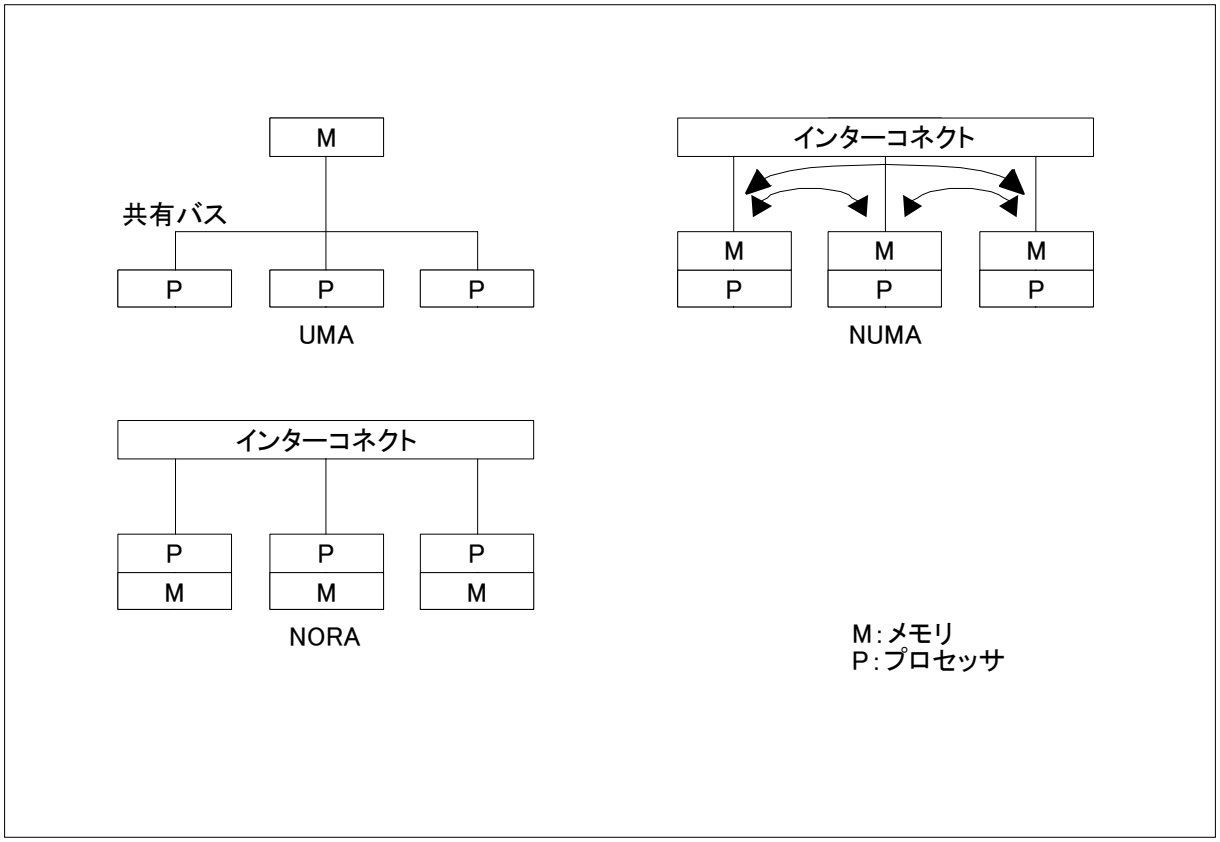

図 I-10-7.　HPC クラスタにおけるメモリの結合方式

【解説】

1) 計算の高速化と並列処理の必要性

軍事用途における核シミュレーションや、金融用途における株式市場シミュレーション、人体シミュレーションや気象予報計算、飛来する宇宙線の解析など、大規模計算を現実的な時間内に完了したい需要は高い。このような需要を背景として、計算機の高速化はこれまで重要視されてきた。しかし単一のコンピュータにより実現される演算性能には限界があることが、近年、認識されるようになった。そこで、昨今のスーパーコンピュータは価格性能比の観点から、汎用プロセッサによるノードを多数結合して並列に動作させることにより全体として演算処理性能のスループットを得る「HPC クラスタ」によるスーパーコンピュータが主流となっている。

HPC クラスタの性能を引き出すためには、その上で動作するプログラムが並列処理に対応していなければならない。例え複数ノードのHPCクラスタシステム上でプログラムを実行したとしても、そのプログラムが並列化を全く考慮していなければ、実際に処理を行うノードはひとつだけであり並列コンピュータの恩恵を受けることはできない。

2) 並列計算機の種類

並列計算機の種類には、以下のようなものがある。

* マルチプロセッサ型
  多数のプロセッサを備え、並列処理を行うコンピュータ。
* クラスタ型
  多数のコンピュータをネットワークでつないだシステム。HPC クラスタはこれに相当する。

3) プロセッサやメモリの結合方式

マルチプロセッサ型の並列計算機では、プロセッサやメモリの結合方式に様々なものがある。

* 共有メモリモデル
  - UMA(Uniform Memory Access)
  - NUMA(Non-Uniform Memory Access)
* メッセージ交換モデル NORA(NO Remote-memory Access)

4) インターコネクトのトポロジー

マルチプロセッサ型の並列計算機では、プロセッサやメモリの配置(トポロジー)にも様々なものがある。

* 共有メモリ型
  - バスベースアーキテクチャ
  - スイッチベースアーキテクチャ
* 分散メモリ型
  - 静的ネットワーク(メッシュなど)
  - 動的ネットワーク(クロスバー)
  - 完全結合型ネットワーク
  - オメガネットワーク
  - バスベースネットワーク

スキル区分	OSS モデルカリキュラムの科目	レベル
ネットワーク分野	10　クラスタシステム構築に関する知識 I	基本
習得ポイント	I-10-8. 並列処理プログラミングの基本、並列化処理	
対応する コースウェア	第 5 回（並列プログラミング概論）	

I-10-8. 並列処理プログラミングの基本、並列化処理

並列プログラミングの基本的な技法と、並列処理の効率性について論じる。代表的なアルゴリズムについて並列化の技法を紹介し、さらに並列プログラミング環境と並列プログラミングで利用できるツールを紹介する。

【学習の要点】

＊ 並列処理とは、一つのタスクをより小さなサブタスクに細分化、複数のプロセッサを用いてこれらを並列に処理し、全体の処理効率向上を図る手法である。

＊ プログラムを並列化する方法は、データ並列化、タスク並列化、パイプライン並列化の主に3通りに分けられる。

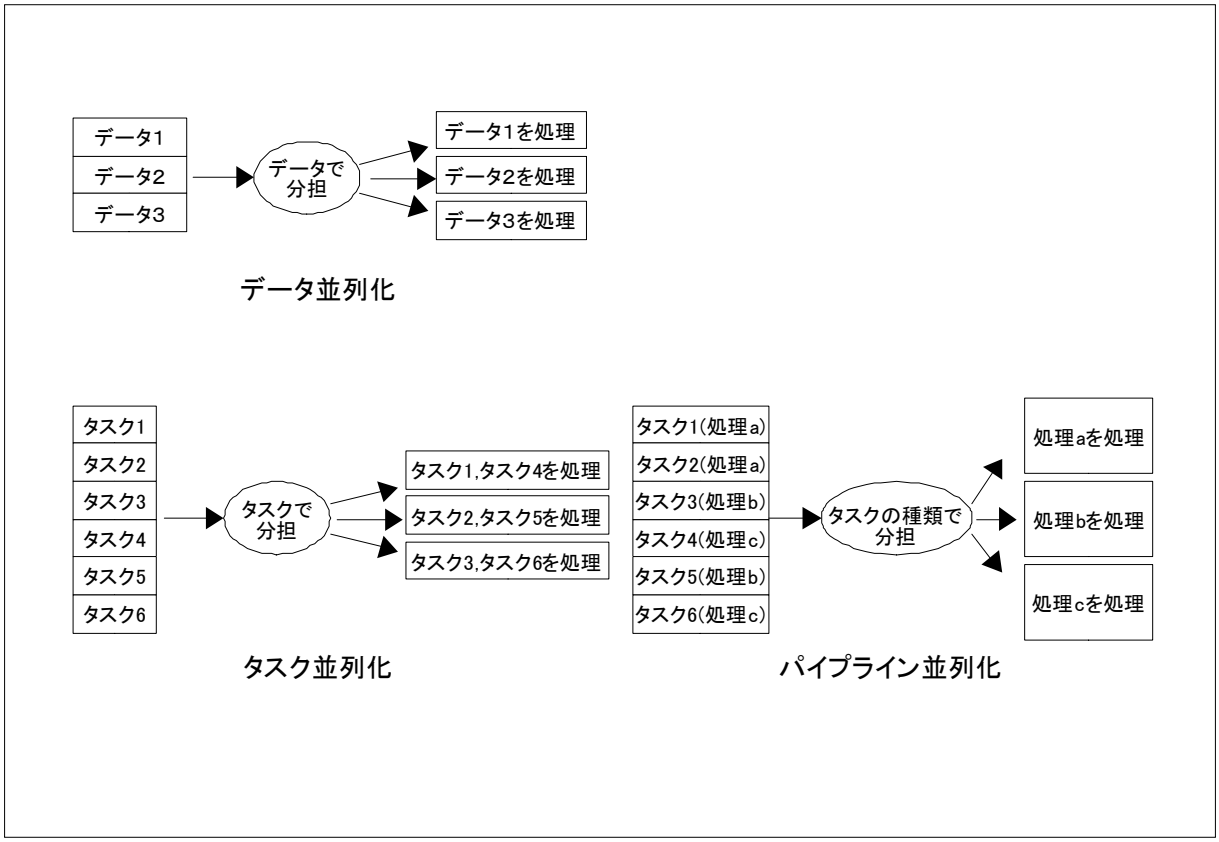

図 I-10-8.　並列化の種類

【解説】

1) 並列処理と逐次処理

並列処理とは、一つのタスクをより小さなサブタスクに細分化し、複数のプロセッサを用いてこれらを並列に処理することによって全体の処理効率向上を図る手法である。一方、単一のプロセッサを用いて一つのタスクを逐次的に処理することを逐次処理と呼ぶ。

2) 並列処理の効率性

ある一つの結果を得るために実行される一つのタスクを、より小さな複数のサブタスクに分割する。その際、他のサブタスクが処理されることにより得られる結果がなければ、処理を開始できないサブタスクが含まれる場合、並列処理は不可能である。

並列コンピュータにおいて、利用可能な全プロセッサが常時100%使い切られている状態が最も効率がよい並列処理の実施状況である。しかし前述のように、プログラムを実行順番に依存しないサブタスクに分割できない場合もある。並列処理の効率性を高めるには、プログラム開発の時点で並列処理を意識したプログラミングを注意深く行い、並列処理に適したプログラムを構成する必要がある。

3) 並列化の種類

プログラムを並列化する方法は、以下の3通りに分けられる。

* データ並列化
* タスク並列化
* パイプライン並列化

4) 並列化された代表的なアルゴリズム

ここでは、既に並列化手法が確立されているアルゴリズムの例を挙げる。

* 行列の部分分割によるLU分解の並列化
* 有限要素法や差分法による、ラプラス方程式の解を求める演算の並列化

5) 並列プログラミング環境とツール

並列プログラミングを行う際に使用する環境およびツールを紹介する。

* OpenMP
  http://www.openmp.org/drupal/
  C/C++やFortran言語における共有メモリ並列プログラミングのためのAPI。
* PVM(Parallel Virtual Machine)
  http://www.csm.ornl.gov/pvm/pvm_home.html
  複数コンピュータにより並列処理を行うためのソフトウェア。
* MPI(Message Passing Interface)
  http://www.mpi-forum.org/
  並列処理でメッセージを送受信するための仕様。
* pthread(POSIX Thread)
  http://www.opengroup.org/onlinepubs/007908799/xsh/pthread.h.html
  UNIX系OSで並列処理を行うためのライブラリ。

スキル区分	OSS モデルカリキュラムの科目	レベル
ネットワーク分野	10　クラスタシステム構築に関する知識 I	基本
習得ポイント	I-10-9. マルチスレッドプログラミングの基礎	
対応する コースウェア	第 6、7 回（並列プログラミング　実践（1）マルチスレッドプログラミング）	

I-10-9. マルチスレッドプログラミングの基礎

並列プログラミングの一種であるマルチスレッドプログラミングの基礎について解説する。スレッドとは何か、スレッドとプロセスの違い、スレッドの生成と終了など、スレッドプログラミングに関する基本的な概念を説明する。

【学習の要点】

* プロセスとスレッドは並列処理の単位であり、どちらも、プロセッサが複数ある環境では並列に実行される。
* プロセスはスタック、メモリ空間共に、各プロセス固有のものを持っており、占有している。これに対しスレッドでは、スタックは各スレッド固有のものを持つが、メモリ空間は同一プロセス内で共有する。
* マルチスレッドによる並列化は、マルチプロセスによる並列化に比べ一般に効率が高い。ただしマルチスレッドプログラミングでは共有資源の排他制御を行わなければならない。また排他制御を行う際には、デッドロックに細心の注意を払う必要がある。

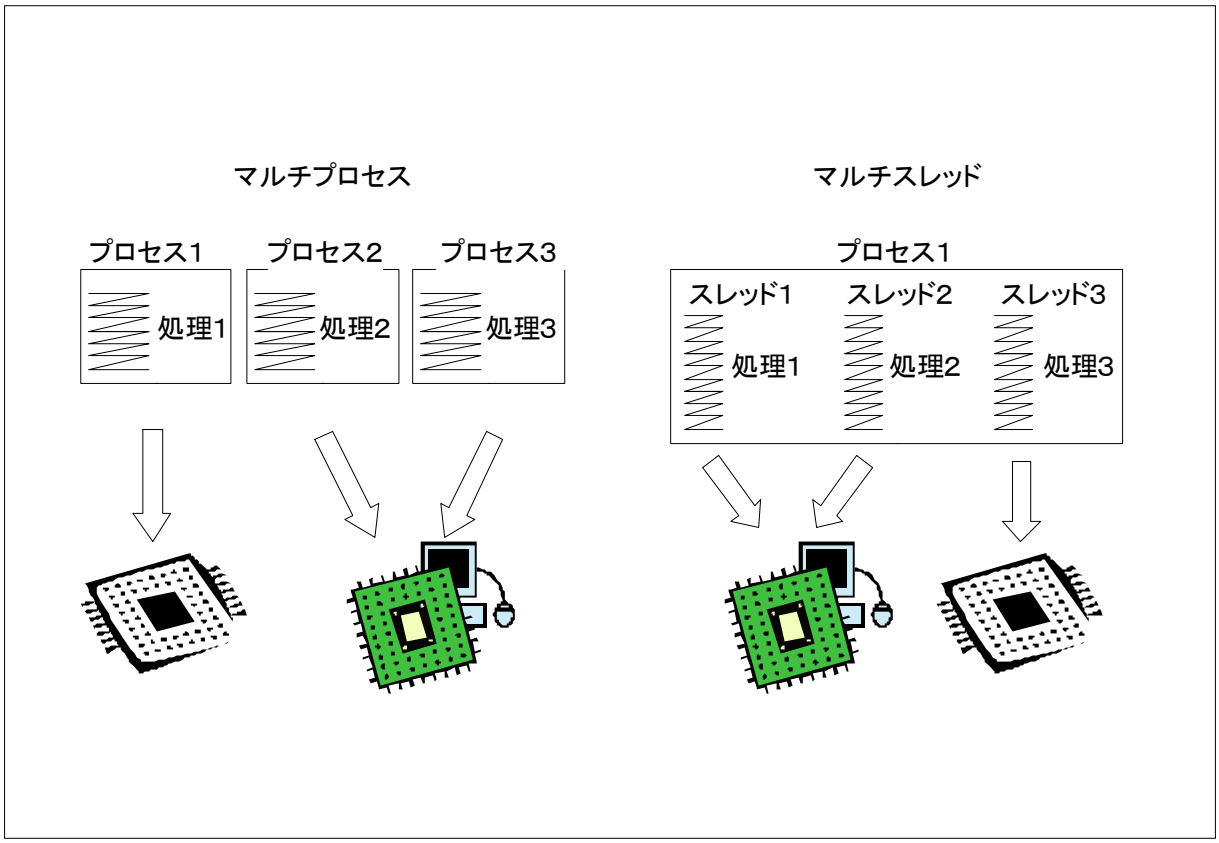

図 I-10-9.　マルチプロセスとマルチスレッド

【解説】

1) プロセスとスレッド

プロセスとスレッドは並列処理の単位である。プロセスもスレッドも、プロセッサが複数ある環境では並列実行される。スタックはそれぞれ固有のものを持っており、占有している。一方で、メモリ空間についてはその扱いが異なる。プロセスは個々に独立したメモリ空間を占有しているが、同一プロセス内のスレッドはメモリ空間を共有しており、各スレッドから共通のメモリ領域へアクセスすることができる。このような違いから、アクティブな動作主体を切り替える場合には一般にプロセススイッチよりスレッドスイッチの方が低コストである。アクティブなプロセスを切り替える場合にはスタックとメモリ空間をそのプロセスのものに置き換えなければいけないのに対し、マルチスレッドをサポートする環境において同一プロセス上でスレッドを切り替える場合には、スタックの置き換えだけでよいからである。なお、プロセスやスレッドを切り替えるためにスタックおよびメモリ空間を置き換える操作を、総じてコンテキストスイッチと呼ぶ。

2) スレッドプログラミングにおける排他制御

マルチプロセスによる並列化に比べ、マルチスレッドによる並列化の方が、メモリ空間を共有している分、一般に高効率であるという利点を持つ。しかし、メモリ空間内の共有データに複数のスレッドがアクセスする場合に、データアクセスのタイミングによっては共有するデータの状態がスレッドの意図しないものに変わってしまうという可能性がある。この問題を回避するために、マルチスレッドプログラミングにおいては排他制御を行う必要がある。排他制御とは、あるスレッドが共有データを処理中に他のスレッドがそのデータへアクセスしないようにロックする処理のことである。また、プログラム内の共有データを処理する部分を、クリティカルセクションと呼ぶ。

3) デッドロック

排他制御を行う際に最も留意すべき問題として、デッドロックがある。例えば、以下の順でプログラムが実行されたときにデッドロックが生じる。ここでは、スレッド X とスレッド Y が、ふたつの共有データ 1 および 2 にアクセスする場合を考える。

* スレッド X がデータ 1 をロックし、スレッド Y にスイッチする。
* スレッド Y がデータ 2 をロックし、スレッド X に再びスイッチする。
* スレッド X がデータ 2 へのアクセスを試みるが、他のスレッド(Y)によりロックされているため、ロック解放待ちとなる。
* スレッド Y がデータ 1 へのアクセスを試みるが、他のスレッド(X)によりロックされているため、ロック解放待ちとなる。

この後、スレッド X、Y はお互いのロック解放を永遠に待ち続ける状態になる。このような状態をデッドロックという。

共有データの種類やスレッド内の処理内容が限られているならば、それらに対するロックの順番をあらかじめ決めておき、全スレッドがその順番を守ることによりデッドロックを回避することができる。先の例では、共有データ 1 および 2 をロックする場合にはデータ 1 のロックを取得してからデータ 2 のロックを取得するというルールを定めておけばよい。しかし、OS の内部における排他制御処理などではそう簡単にデッドロックを回避することができず、難しい問題が存在する。OS の場合様々なプロセスやスレッドが動作するため、全てのプロセスおよびスレッドから様々な資源を一定のルールに従って保護することは困難なためである。

スキル区分	OSS モデルカリキュラムの科目	レベル
ネットワーク分野	10　クラスタシステム構築に関する知識 I	基本
習得ポイント	I-10-10. スレッド間の通信とスレッドの同期	
対応する コースウェア	第 6、7 回（並列プログラミング　実践 （1） マルチスレッドプログラミング）	

I-10-10. スレッド間の通信とスレッドの同期

pthread を題材として、具体的なマルチスレッドプログラミング方法を示す。またマルチスレッドプログラミングで必須の技術である同期とスレッド間通信の実現方法を紹介する。

【学習の要点】

* スレッドの生成には pthread_create()関数を用い、引数に処理対象関数の関数ポインタと、処理対象関数へ渡す引数リストの配列のポイントを渡す。生成したスレッドの終了を待つ場合には、pthread_join()関数を用いる。
* pthread_cond_init()、pthread_cond_wait()、pthread_cond_signal()、pthread_cond_destroy()を用いて条件変数によるスレッドの同期を行う。
* pthread_mutex_init()、pthread_mutex_lock()、pthread_mutex_unlock()、pthread_mutex_destroy()を用いて共有資源の排他制御を行う。

```
#include<stdio.h>
#include<pthread.h>

#define MAX_THREAD_NUM  (2)
        /* 一度に2スレッドまで並行動作を許可*/
#define THREAD_NUM (5)

/* グローバル変数 (全スレッド共有) */
pthread_mutex_t mutex;
pthread_cond_t  cond;
int thread_num = 0;

void thread_func(void *arg) {
    int id = (int)arg;

    /* 条件変数による条件待ち */
    pthread_mutex_lock(&mutex);
    while (thread_num >= MAX_THREAD_NUM)
        pthread_cond_wait(&cond, &mutex);
    thread_num++;
    pthread_mutex_unlock(&mutex);

    printf("Thread %d started.¥n", id);
    sleep(1);
    printf("Thread %d finished.¥n", id);
    /* 条件変数による条件変更の通知 */
    pthread_mutex_lock(&mutex);
    pthread_num--;
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mutex);
}

int main() {
    int i;
    pthread_t handle[THREAD_NUM];

    /* 条件変数の初期化 */
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&cond, NULL);

    for (i = 0; i < THREAD_NUM; ++i)
        pthread_create(&handle[i], NULL,
            (void *)thread_func, (void *)i);

    for (i = 0; i < THREAD_NUM; ++i)
        pthread_join(handle[i], NULL);

    /* 条件変数の破棄*/
    pthread_cond_destroy(&cond);

    return 0;
}
```

図 I-10-10.　pthread プログラミングの例
マルチコア CPU のための並列プログラミング(秀和システム 2006 年発行)より引用

【解説】

1）pthread によるマルチスレッドプログラミングの基礎

＊ スレッドの生成および、スレッドの実行する関数とその引数

pthread_create()関数によりスレッドを生成する。pthread_create()の引数には、スレッドが実行する処理を表す関数への関数ポインタ、その関数へ渡す引数リストの配列などを渡す。また第一引数には、スレッド ID を受け取るために pthread_t 型変数へのポインタを渡す。

＊ スレッドの終了待機

自身から生成された子スレッドの実行により得られた値を最終的にメインスレッドが収集し、表示や格納などを行う場合は、pthread_join()関数を用いて子スレッドの終了を待つことができる。pthread_join()の引数に pthread_create()関数の実行により得られたスレッド ID を渡すことで、待ち合わせ対象のスレッドを指定する。

＊ 条件変数によるスレッド同期

pthread では、条件変数が他のスレッドにより変更されるまで待つ操作を実現する API を持つ。pthread で条件変数によるスレッドの待ち合わせを行う際のポイントは、以下の通りである。

- 条件変数の初期化

  条件変数の初期化を行うには、pthread_cond_init()関数を用い、条件変数 pthread_cond_t 型の条件変数を第一引数に渡す。

- 条件変数による条件変更の通知

  条件変数により条件変更を通知するには、pthread_cond_signal()関数を用い、引数に変更対象の条件変数を渡す。この関数をコールすると、既に pthread_cond_wait()関数をコールしてブロックされているスレッドが一つだけアクティブとなり、処理が再開される。ブロックされている全てのスレッドをアクティブにしたい場合には、pthread_cond_broadcast()関数を用いる。

- 条件変数の破棄

  不要になった条件変数を破棄するには、pthread_cond_destroy()関数を用い、引数に破棄対象の条件変数を渡す。

＊ 共有資源の排他制御

pthread で排他制御処理を行う際のポイントは、以下の通りである。

- 排他制御変数の初期化

  排他制御の準備のために、排他制御変数を初期化するには、pthread_mutex_init()関数を用いる。mutex_t 型の変数を引数として渡すと初期化が完了し、排他制御変数として用いられるようになる。

- 排他制御の開始と終了

  クリティカルセクションに当たるコードが複数スレッドにより並列処理されないように、処理の開始前に排他制御の開始を示す pthread_mutex_lock()関数を、処理の終了後に排他制御の終了を示す pthread_mutex_unlock()関数をコールする。

- 排他制御変数の破棄

  スレッドの処理が終了し排他制御が不要になったら、使用していた排他制御変数を破棄する。排他制御変数を破棄する前に、どのスレッドも既にその排他制御変数を使用していないかを注意する。解放には pthread_mutex_destroy()関数を用い、引数に破棄対象の排他制御変数を渡す。