クラスタ型高可用性(HA)システムのサーバ間同期制御とプログラム型制御方式

Server Synchronous Control Method and Program Type Control Method for Cluster HA System

遠藤　浩太郎　K. Endo
山田　晃智　A. Yamada

高可用性(HA：High Availability)システムを目的としたクラスタ制御ソフトウェア(以下，HAカーネルと呼称)を開発し，さまざまな多重化システムに応用している。HAカーネルは，複数サーバの同時障害にも対応できる独自のクラスタ同期制御方式，およびクラスタ全体を一貫した視点で記述できるプログラム型の制御方式が特長である。また，プラットホームに依存せず抽象化した機能だけでコンパクトなモジュールにすることができた。

We have developed a cluster control software (hereinafter referred to as an HA kernel) for a cluster high-availability system, and have applied it to various distributed server complex systems.

The HA kernel is characterized by a new cluster synchronous control method which can respond to simultaneous faults in two or more servers, and by a control program which can describe the whole cluster from a consistent viewpoint. We have succeeded in making the HA kernel compact by giving it only abstracted functions independent of the platform.

1 まえがき

複数のサーバをネットワークで結合したクラスタ技術が注目されている。当社は，HAシステムを目的としたクラスタ制御ソフトウェア(HAカーネル)をいち早く開発し，さまざまな多重化システムに応用している。HAカーネルは次のような特長をもっている。

(1) 複数のプラットホーム上で動作可能で，コンパクトなモジュール構成である。
(2) 複数サーバの同時障害にも耐えられるクラスタ同期制御である。
(3) クラスタ全体を一貫した視点で記述できるプログラム型制御方式(以下，シナリオスクリプトと呼称)である。

特に，クラスタ同期制御とプログラム型制御方式は，他のクラスタ制御ソフトウェアにはない独自の技術である。

ここでは，HAカーネルの特長について述べる。

2 HAカーネル

2.1 HAカーネルの動作環境

HAカーネルは，シナリオスクリプト，リソース，運用管理ツールと関係して動作する。

(1) シナリオスクリプトは，HAカーネルの動作を規定するある種のプログラムである。
(2) リソースは，アプリケーション，共有ディスク装置，共有ディスク切換えコマンドなどをまとめた概念であり，HAカーネルの制御対象となる。
(3) 運用管理ツールは，HAカーネルが管理するクラスタ状態をネットワークを経由して監視，操作するツールである。

この三つの関係のほかに，各サーバのHAカーネルは相互に関連して動作する。

2.2 HAカーネルの動作

HAカーネルの動作の例を示す。

(1) 停止　　サーバAが障害により停止する。
(2) 検出　　サーバAのHAカーネルからのハートビートが来なくなったことを，生き残っているサーバのHAカーネルが判断する。これを行うために必要な構成情報(ハートビートの間隔，タイムアウト値，クラスタを構成しているサーバの数，各サーバのアドレス，など)は，すべてシナリオスクリプトにあらかじめ記述されている。
(3) 決定　　生き残っているサーバにいるそれぞれのHAカーネルは，全サーバがサーバAの停止を検出したことを確認する。さらに，どのサーバがアプリケーションを引き継ぐべきかについて意見を統一する。
(4) 再開　　引き継ぐべきサーバにいるHAカーネルはリソースに対して引継ぎ処理を指示する。実際に引継ぎ処理を行うのはリソースである。指示される処理の内容はシナリオスクリプトにあらかじめ記述されている。

3 複数のプラットホームで動作可能なモジュール構成

HAカーネルはクラスタシステムのかなめであり，もっ

とも高い信頼度が要求されるモジュールである。この部分をプラットホームから独立させ，コンパクトにまとめると，クラスタシステム全体の開発効率と信頼性が高まる。

HAカーネルは，開発の初期の段階からWindowsNT®(注1)/Pentium™(注2)，Solaris(注3)/SPARC(注4)，AIX/PowerPCの各プラットホームで動作するように設計された。プラットホーム固有のコーディングを必要とするモジュールは，そのほとんどをHAカーネルの制御対象であるリソース側になるようにした。プラットホーム固有の設定値や動作は，シナリオスクリプトに記述するようにした。結局，HAカーネルは，プラットホームに依存しない抽象化された機能だけをもつクラスタ制御ソフトウェアになった。

HAカーネルの規模は，以前のクラスタ制御ソフトウェアに比べて非常に小さくなった。同時に制御できるサーバの台数を2台から4台へ拡張し，マルチプラットホームに対応したにもかかわらず，ソースコードの行数は約1/2になった。これは，抽象化された機能をオブジェクト指向プログラミングによってうまく記述できたからである。

4 クラスタ同期制御

4.1 抽象化されたHAカーネルの動作

HAカーネルの処理は，局所プロセスと大域プロセスの二つに分けられる。

局所プロセスは，各サーバで別々に行われる処理であり，サーバ障害の検出，LAN障害の検出，運用管理ツールからの操作の受け付け，リソースの制御などがある。2.2節の例でいえば，“検出”と“再開”のところである。

大域プロセスは，サーバ間で同期をとって行われる処理であり，クラスタ状態の更新，サーバの組込み，サーバの切離しがある。2.2節の例でいえば“決定”のところである。

局所プロセスの実現方法には技術的な課題はほとんどない。それは，サーバ上の普通のプロセスとほとんど変わりがないからである。当社の実装での特長は，汎(はん)用性を考えて，シナリオスクリプトに局所プロセスの動作がプログラムされる点である。つまり，HAカーネルは，シナリオスクリプトのインタプリタであり，運用管理ツールからの操作の種類や，リソースの種類について何も知らない。それらの情報とプログラムは，シナリオスクリプトに納められている。一方，大域プロセスの実現方法は，クラスタ制御ソフトウェアのもっとも重要なポイントである。これについては次節で述べる。

4.2 複数サーバ同時障害に耐えるクラスタ同期制御

(注1) WindowsNTは，Microsoft社の商標。
(注2) Pentiumは，インテル社の商標。
(注3) Solarisは，Sun Microsystems社の商標。
(注4) SPARCは，SPARC International社の商標。

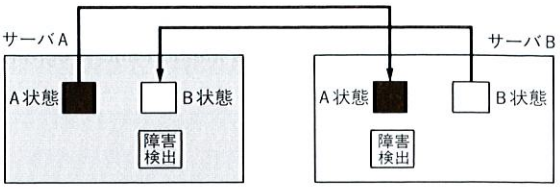

図1．クラスタ状態を分散管理する方法　各サーバはすべてのサーバの状態を記憶するが自サーバの状態だけを更新できる。

Distributed management of cluster states

	サーバA	サーバB
初期状態	[△ △]	[△ △]
1	[○ △]	[△ △]
2	[○ △] ⟶	[○ △]
3		[○ △]
4		[? ○]

(△：組込み状態，○：アプリケーション実行中，?：状態不明)
[△ △]は，左側がサーバA，右側がサーバBの状態を表す。
1．サーバAでアプリケーション開始が要求され，サーバAでアプリケーションが実行される。
2．サーバAにおいてアプリケーション実行中であることをサーバBに通知する。
3．サーバAが障害により停止する。
4．サーバBは，サーバAの障害を検出し，アプリケーションを再開する。

図2．クラスタ状態を分散管理した場合の動作と状態遷移例　各サーバが“サーバAの状態”，“サーバBの状態”という形でクラスタの状態を認識している。

Example of operation in case of distributed management

まず，問題点をはっきりさせるため，対照的な二つの実現方法を考える。

(1) 方法1：クラスタ状態を分散して管理(図1)　この方法では，それぞれのサーバはすべてのサーバの状態を記憶するが，自分の状態だけを更新することができる。図2に動作の例を示す。

とくに問題ないように見えるが，各サーバでほぼ同時にアプリケーション開始要求があった場合に，両方のサーバでアプリケーションを開始してしまうかもしれない(逐次性の問題)。

このような状況は，4台のクラスタシステムではさらに頻繁に発生する。例えば，サーバAの状態がアプリケーション実行中であることを，他のサーバB～Dに通知しようとしている途中で，サーバAが障害で停止してしまうと，サーバB～Dの間でサーバAの停止前の状態の認識が矛盾するかも知れない(原子性の問題)。

これは，クラスタ全体の状態遷移に一貫性がないということである(一貫性の問題：上記二つの問題を合わせて)。この問題は，各サーバが勝手にそれぞれの状態を更新するために発生する。そこでもう一つの実現方法を考えてみる。

(2) 方法2：クラスタの状態を制御サーバで集中管理(図3)　この方法では，クラスタを構成するサーバは，クラスタ状態を制御サーバに問い合わせ，クラスタ状態の更新を制御サーバに要求する。図4に動作の例を

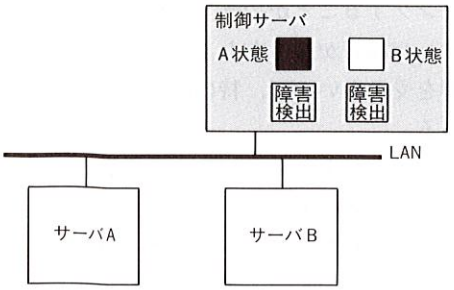

図3．クラスタ状態を集中管理する方法　制御サーバはクラスタ内のすべてのサーバの状態を管理する。

Centralized management of cluster states

	サーバA	制御サーバ	サーバB
初期状態		[△ △]	
1	→	[□ △]	
2	←	[□ △]	
3	アプリケーション実行	[□ △]	
4	→	[○ △]	
5		[○ △]	
6		[○ △]	サーバAの障害検出
7		[？ □]	←
8		[？ □]	→
9		[？ □]	アプリケーション実行
0		[？ ○]	←

（△：組込み状態，○：アプリケーション実行中，□：アプリケーション開始要求，？：状態不明）
1.サーバAから制御サーバにサーバAでのアプリケーション開始要求が通知され，クラスタ状態が更新される。
2.サーバAは，制御サーバにクラスタ状態を問い合わせる。
3.サーバAは，アプリケーションを実行する。
4.サーバAから制御サーバにサーバAでのアプリケーション実行中が通知され，クラスタ状態が更新される。
5.サーバAが障害により停止する。
6.サーバBは，サーバAの障害を検出する。
7.サーバBから制御サーバにサーバAの停止が通知され，制御サーバ上でアプリケーションの引継ぎ先が決定され，クラスタ状態が更新される。
8.サーバBは，制御サーバにクラスタ状態を問い合わせる。
9.サーバBは，アプリケーションを開始する。
0.サーバBから制御サーバにサーバBでのアプリケーション実行中が通知され，クラスタ状態が更新される。

図4．クラスタ状態を集中管理した場合の動作と状態遷移例　制御サーバが，“サーバAの状態”，“サーバBの状態”という形でクラスタの状態を認識している。

Example of operation in case of centralized management

示す。

この方法は，制御サーバが停止した場合の耐障害性に明らかな欠点がある（耐障害性の問題）。しかし，前の方法と異なりクラスタ状態の一貫性が保証されているという長所がある。また，局所プロセスと大域プロセスの区別がはっきりしている。すなわち，クラスタを構成している各サーバで実行される処理が局所プロセスであり，制御サーバで実行される処理が大域プロセスである。

当社では，HAカーネルの実現において方法2の良いところをモデルにし，方法1に近い実装をするような方法とした。すなわち，クラスタ状態の一貫性の問題を解決するために，局所プロセスはあたかも制御サーバがいるかのように大域プロセスとやりとりをする。さらに，すべてのサーバ上にクラスタ状態を保ち，すべてのサーバ上で大域プロセスを実行するという方法にした（図5）。これにより，大域プロセスの耐障害性の問題が解決する。サーバが停止しても停止したサーバを“切り離す”だけで大域プロセスは継続する。

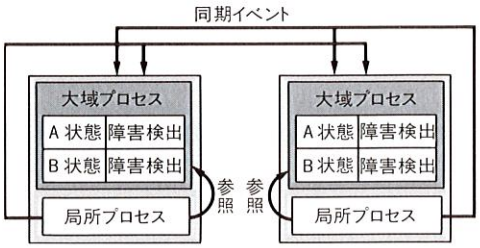

図5．HAカーネルの実装　すべてのサーバ上にクラスタ状態を多重化して保つ。

Implementation of HA kernel

次に，多重化されたクラスタ状態の一貫性はどのようにして保たれるのかを考えてみる。まず，大域プロセスが何をするのかを整理すると次の三つが挙げられる。

(1) 局所プロセスから大域プロセスへの通知（具体的には，障害の検出，管理ツールからの操作，局所プロセスの処理の完了，組込み要求，切離し要求など。これらをまとめて同期イベントと呼ぶ）を受け付ける。

(2) 同期イベントを受け付けると，クラスタ状態を適切な状態へ変更する。

(3) 局所プロセスから大域プロセスへのクラスタ状態の問合せに対して回答する。

そこで大域プロセスを多重化し，一貫性を保つために，次のことを実現した。

(1) サーバがクラスタに組み込まれるとき，クラスタ状態をそのサーバにコピーする（クラスタ状態のコピーをするのはこのときだけである）。

(2) 同期イベントの送信は，逐次的（同期イベントの到着順序が各サーバで同一）かつ原子的（同期イベントを出したサーバが直後に停止したとしても，すべてのサーバに到着するか，すべてのサーバに到着しないかのどちらか）なマルチキャストである。

(3) 同期イベントの種類と前のクラスタ状態から，一意的に次のクラスタ状態を決定する。

これは，初めの状態および入力列が同じで，同じプログラムを動かせば，各サーバの大域プロセスは同一の動作をするということである。

逐次的かつ原子的なマルチキャストの実現方法は知られているが，当社は，組込み時のクラスタ状態の配布や，定期的なクラスタ状態の同一性チェックと共通化できるような独自のプロトコルを開発し実装した。

三つ目の点は，大域プロセスが全サーバで同じプログラムであればよいが，汎用性を考えて，大域プロセスをクラ

スタ状態だけを参照できる同期イベントの処理ハンドラとして，シナリオ スクリプトに記述できるようにし，すべてのサーバにあらかじめ同一のシナリオ スクリプトを配布しておくことで実現した。

図 6 に HA カーネルの動作の例を示す。

(△：組込み状態，○：アプリケーション実行中，□：アプリケーション開始要求，?：状態不明)
1. サーバAから，サーバAでのアプリケーション開始要求の同期イベントが各サーバに通知され，クラスタ状態が更新される。
2. サーバAは，サーバAにいる大域プロセスにクラスタの状態を問い合わせる。
3. サーバAは，アプリケーションを開始する。
4. サーバAから，サーバAでのアプリケーション実行中の同期イベントが各サーバに通知され，クラスタ状態が更新される。
5. サーバAが障害により停止する。
6. サーバBは，サーバAの障害を検出する。
7. サーバBから，サーバAの停止の同期イベントが各サーバに通知され，シナリオ スクリプトにしたがって引継ぎ先が決定され，クラスタ状態が更新される。
8. サーバBは，サーバBにいる大域プロセスにクラスタの状態を問い合わせる。
9. サーバBは，アプリケーションを開始する。
0. サーバBから，サーバBでのアプリケーション実行中の同期イベントが各サーバに通知され，クラスタ状態が更新される。

図 6．HA カーネルの動作例　各サーバにある大域プロセスが“サーバ A の状態”，“サーバ B の状態”という形で，クラスタ状態を管理している。

Example of operation of HA kernel

実際には，局所プロセスからの参照に関して同時性を保つ（サーバ A の大域プロセスから参照したのち，サーバ B の大域プロセスから参照したとき，サーバ B のクラスタ状態はサーバ A のクラスタ状態と同じか新しい）ために，次の 2 点を実現する必要もある。

(1) 他のすべての動作中のサーバが同期イベントを受信するまで，次のクラスタ状態の決定を待つ。

(2) 同期イベントを受信した後は，局所プロセスからの参照への応答を次のクラスタ状態が決定するまで待つ。

5 シナリオ スクリプトによる制御方式

HA カーネルの大域プロセスの動作はシナリオ スクリプトとしてプログラミングする。大域プロセスは，方法 2（図 3）の制御サーバに相当しているので，クラスタ全体を一貫した視点でプログラミングすることができる。これは方法 1（図 1）のような分散制御の場合に比べ次の利点がある。

(1) クラスタ全体の制御を一つのコンテキストでプログラミングすることができ，処理の記述が容易である。

(2) サーバ障害が起きたときに，制御のコンテキストは影響を受けないので，特に障害時の処理の記述が容易である。

(3) 制御のコンテキストが独立しているので，プログラムの動作を解析（デバッグ）することが容易である。

このような特長なしでは，クラスタ制御ソフトウェアの制御の部分そのものを記述することが，クラスタ制御ソフトウェアを一から作り直すことと同様になってしまう。したがって，大域プロセスをシナリオ スクリプトとして記述できることが当社の HA カーネルの最大の長所である。

HA カーネルは，当初その名前のとおり HA システム用に開発された。この論文で用いた動作例も HA システムの動作である。ところがそののち，HA カーネルを 4 台のパラレル データベース システムのクラスタ制御ソフトウェアとして適用する機会を得た。パラレル データベース システムは HA システムとは対称的に，すべてのサーバで同時にサービス ソフトウェアが動作する。したがって，クラスタ制御の内容も HA システムの場合とは異なっているが，HA カーネルをこれに適用させることは容易であった。データベースをリソースとみなすための簡単なインタフェース プログラムを作成し，シナリオ スクリプトに制御の内容を記述するだけで実現できた。これが容易に行えたことは，クラスタ全体を一貫した視点で設計できるこのクラスタ制御ソフトウェアの長所の一例である。

6 あとがき

クラスタ制御ソフトウェアの特長について紹介した。この技術は，最後に述べたパラレルデータベースへの適用のほかに WindowsNT® および UNIX (注5) 上の HA システムとして製品化されている。

文　献

(1) J. グレイ，他（渡辺栄一編訳）：フォールト・トレラント・システム，マグロウヒル (1986)

(注 5) UNIX は，X/Open カンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標。

遠藤　浩太郎　Kotaro Endo
情報・通信システム技術研究所 開発第三担当。
高可用性システム技術の研究開発に従事。
Information & Communications Systems Lab.

山田　晃智　Akitomo Yamada
情報・通信システム技術研究所 開発第三担当主務。
高可用性システム技術の研究開発に従事。
Information & Communications Systems Lab.