C-019

# ソフトウェアによる制御システム向け Fault Tolerant Computer の提案
# Proposal of Software-based Fault Tolerant Computer System for Control System

大島 訓† Satoshi Oshima
関山 友輝† Tomoki Sekiyama
吉田 雅徳† Masanori Yoshida
青木 英郎† Hideo Aoki
長井 昭裕† Akihiro Nagai
大平 崇博† Takahiro Ohira

## 1. はじめに

社会インフラを支える制御システムでは，障害発生による社会的影響が極めて大きいため，信頼性・可用性とリアルタイム性の両立が求められる．こうした要求にこたえるための手法の一つに，フォールト・トレラント・コンピュータ(FTC)がある．FTC は，制御システムを構成する計算機の CPU やメモリ，I/O 等の構成部品を多重化し，常時その両系を動作させることにより、片方で障害が発生しても無停止で動作を継続できる．アクティブ・スタンバイ構成と比較して，障害発生時のフェイルオーバ時間が短時間で済むため，リアルタイム性の確保がしやすい，アプリケーションやミドルウェアをフェイルオーバ前提で構成する必要がないため，構成を比較的シンプルに保ちやすい，さらに，外部からは単一のシステムに見えるため、制御対象側で冗長化を意識することが難しい個所にも適用できるといった様々な利点がある．

その半面，CPU 等を多重化した特殊なハードウェアが必要になるという課題がある．特に、近年の CPU は，半導体プロセスの微細化により，様々な周辺機能を CPU ダイ内に内蔵しているため，ハードウェアによる FTC の実現が年々困難になっている．

本提案では上述の制御システムへの適用を目的として，複数の一般的な PC サーバをネットワークで疎結合し，ソフトウェアによって FTC を実現する計算機構成手法を提案する．本手法では，OS およびミドルウェア部分にソフトウェアによってサーバを同期動作させるための仕組みを組み込むことで，従来のように専用ハードウェアを用いることなく，FTC を実現している．

## 2. FTC の構成

### 2.1 システムの構成

本提案では，ネットワークで接続された対象を制御するシステムを想定し，図 1に示すように，2 台の RAS 用 PC サーバ(RAS サーバ)，4 台の制御計算用 PC サーバ(制御サーバ)，4 台の Voter と呼ばれる多数決用コントローラ，RAS サーバと制御サーバを接続する管理ネットワーク，制御サーバと Voter を接続する内部ネットワーク，Voter と制御対象を接続する外部ネットワークで構成する．内部，外部のそれぞれのネットワークは 2 重化されている．

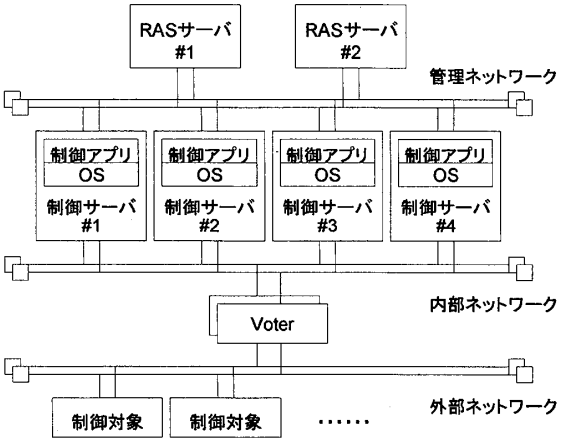


図1　ソフトウェア FTC のシステム構成

### 2.2 RAS サーバの構成

RAS サーバは，本 FTC のコンソール，制御サーバや Voter に障害発生した際のログの格納先という，役割を持つ．RAS サーバが故障すると，FTC はコンソールやログの格納先を失うため，RAS サーバは 2 重化されており，個別に FTC からの切り離し，および，再接続が可能である．また，2 重化された管理ネットワークに接続するため，2 系統の NIC を持つ．

### 2.3 制御サーバの構成

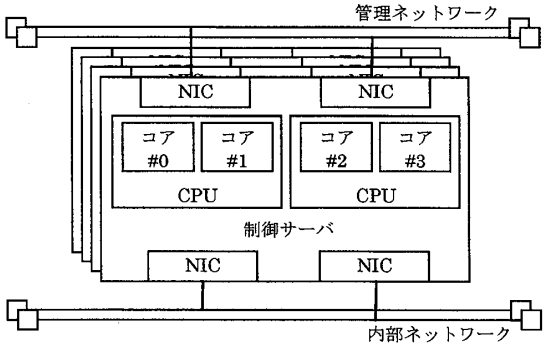


図 2　制御サーバの構成

図 2 にしめすように，制御サーバは，外部から Voter 経由で制御に必要となるデータを受信し，制御に必要となる演算を行い，制御信号を Voter 経由で制御対象に出力する役割をもつ．

† (株)日立製作所　Hitachi, Ltd.

制御サーバは，Voter と内部ネットワークで接続するための 2 重化された NIC を有する．また，制御サーバ間の同期動作や FTC から切り離された制御サーバを再び同期参入させるための再組み込み等，様々な機能をリアルタイムに実現するため，合計 4 つの CPU コアをもつ CPU で構成されている．

### 2.4 Voter の構成

Voter は，外部装置と制御サーバとの通信を仲介する役割を果たす．制御サーバから内部ネットワークを経由して送信されたパケットは，一旦 Voter に受信され，同一内容であることを確認したうえで，パケットを 1 つに集約して外部に送出される．このときパケット内容に相違があれば，多数決により多数派となったものを選択し，少数派のサーバは FTC から切り離すことにより，信頼性・可用性の向上を図っている．Voter 自身も 4 重化されており，従系は主系一台からパケットを出力されるのを監視し，多数決確定後一定時間制御パケット出力がなければ，フェイルオーバすることで，Voter が単一故障点とならないようにしている．

## 3. ソフトウェア FTC 実現のための基本方針

本章では，ソフトウェア FTC 実現のための基本方針を説明する．

### 3.1 コア毎の役割分担

本 FTC では，以下のようにコア毎の役割分担をしている．

コア 0：通信デーモン，ほとんどすべてのカーネルスレッド/デーモンの実行
コア 1：アプリケーションのプロセス実行
コア 2：メモリ回収，再組み込みのためのメモリコピー
コア 3：コア間マイグレーションによる同期合わせ(3.2節)

ハードウェア割り込みについては，コア 1 以外で処理することで，アプリケーションの実行を阻害しないようにし，リアルタイム性の確保を図っている．

以下の節では，コア毎の役割分担の考え方について詳細に述べる．

### 3.2 リアルタイムプロセス同期の考え方

OS 内には，カーネルスレッドや様々なサービスを提供するデーモンなど，アプリケーションのプロセス以外の多くのスレッド/プロセスが存在する．しかし，アプリケーションの同期実行を保証するためにはすべてのスレッド/プロセスを同期実行させる必要は無い．

本 FTC では，同期対象とするプロセスを SCHED_FIFO クラスのアプリケーション(以下同期タスク)に限定する．また，特定のシステムコールが発行された場合のみ，プロセススケジューリングを実行するカーネル内の関数である schedule()が実行されるようにし，schedule()の実行回数を一致させ，さらにプロセススケジューラ内で，各制御サーバが同じタスクを選択するよう通信を行なうことで，制御サーバ間のタスク同期実現を目指す．さらに，3.1 節で述べたように，同期タスクが走行するコアはコア 1 に固定化し，同期タスク以外のスレッド/プロセスはコア 1 以外に割り当てることで同期実行への外乱要因を除去している．

ただし，OS の処理の中には，特定のイベントが成立するまで，schedule()を繰り返し発行し待つという処理が多数存在する．イベントが成立するタイミングは，デバイスの応答のような制御サーバ間で異なるタイミングのものであるため，この状態のままタスクをコア 1 上で走らせ続けると，schedule()の実行回数を一致させることができない．

そこで，schedule()を繰り返し発行してイベント待ちする個所については，一旦実行コアをコア 1 からコア 3 に移し，すべての制御サーバで状態が一致してからタスクをコア 1 に戻す機能[1]を導入する．これをコア間マイグレーションと呼ぶ．これによって，コア 1 上で schedule()実行回数がばらつくことを防いでいる．

また，同期タスクと非同期のプロセスが共有リソースを持った場合，あるいは両者間で通信を行った場合，各制御サーバ間の非同期プロセスの状態不一致が同期タスクに影響を及ぼし，処理の不一致を引き起こす．

そこで，本 FTC では，同期タスクと非同期タスクの間の通信は，以下の三点に限定し，それぞれについて同期継続出来るよう対策を講じている．

(1)同期タスクの起動
(2)シグナルの送信
(3)UNIX ドメインソケットによるメッセージ送受信

### 3.3 再組み込み

制御サーバは，ハードウェアの故障等により停止や処理の不一致を起こし同期対象から切り離されることがある．同期対象外となった計算機が，メンテナンス作業後，同期状態に再参入する機能を再組み込みと呼ぶ．再組み込み実現の基本的な考え方を示す．

再組み込み実現のためには，同期対象タスクがアクセスするリソースに対し，動作中のまま状態を一致化させる必要がある．状態を一致化させるリソースには，ファイルとメモリの内容がある．

#### 3.3.1 メモリ再同期

再組み込み実現のため，同期実行中の制御サーバから同期参入する制御サーバに対し，メモリのコピーを実施する必要がある．詳細は[2]に示されているとおり，一度目は全体コピーを行い，二度目以降は一度目のコピー以後，書き換えの発生した場所のみを差分コピーするという方式により，メモリコピーの差分を徐々に収束させていく方式をとる．また，メモリコピー負荷が同期タスクの実行を妨害しないよう，メモリコピーはコア 2 で実施する

#### 3.3.2 ファイル再同期

ファイルについては，同期タスクが利用するファイルを事前にリストアップしておき，更新があったファイルのみを繰り返しリモートコピーすることにより，メモリと同様に差分を収束させていく手法を採る．

#### 3.3.3 同期ポイント

メモリおよびファイルの再同期が完了した段階で，再組み込みの最終段階である同期参入を実現するため，本 FTC では，タスクに対し，①「同期ポイント」を示すシステムコールを発行する，②同システムコール呼出し後

には一定時間アプリケーションが停止しても良い状態にする，という 2 点の制限を課している．

メモリ再同期とファイル再同期が十分収束した段階で，本 FTC は，各制御サーバ上の同期タスクがすべて同期ポイントに到達すると，同期ポイントで一旦同期タスクの実行を停止させ，メモリ内容およびファイルを完全に一致させた後，同期参入を実行し，再同期対象制御サーバをオンラインに戻す．

### 3.4 メモリ管理

コア 1 上のタスク/OS へのメモリ割り当ての際，一部の制御サーバが空きメモリの枯渇によってメモリ回収が発生した場合，その制御サーバにおいて同期タスクの実行が遅れ，同期を維持出来なくなる要因となる．

そこで，本 FTC では，ユーザ空間でのメモリ確保とカーネル空間でのメモリ確保，それぞれに対し，同期実行中にメモリ回収が発生しないよう対策している．

#### 3.4.1 カーネル空間におけるメモリ割り当て

本 FTC がベースとしている，Linux®では，カーネル内部で slab allocator によるメモリ確保を実行した際や，直接 alloc_pages()を実行した際にメモリ回収が走る可能性がある．メモリ回収が走るタイミングは，システム全体の空きメモリ量の割合が，全てのコアで共通の回収閾値を下回った場合に発生する．特に，Linux®では，空きメモリを可能な限りファイルシステムのキャッシュである page cache として利用するポリシであることから，回収閾値は，低い数値に抑えられている．一般的なシステムでは，page cache として利用中のメモリのうち，書き換えが発生していない page は即座に回収可能であることから，空きメモリとみなすことができるため，回収閾値が低くとも問題は発生しない．しかし，本 FTC では，回収が走る制御サーバと走らない制御サーバが現れることにより，同期タスクの動作にバラつきが発生する要因となりえることから，対策が必要である．

本 FTC では，回収閾値を，コア毎に設定できるよう拡張し，コア 0,1,3 の回収閾値を低く設定し，コア 2 の回収閾値を高く設定することで，メモリ回収を基本的にはコア 2 でしか実行しない仕様としている．

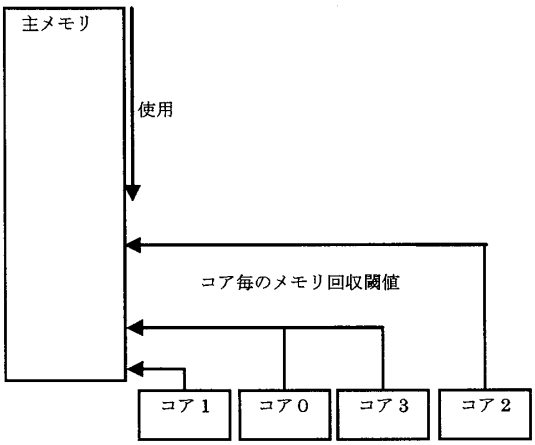


**図 3　コア毎のメモリ回収閾値**

カーネル内で使用するメモリを事前割り当てするよう改修することでメモリ回収を抑える方式も考えられるが，どの程度事前割り当てをすれば安全といえるかどうかは，アプリケーションの特性によって大きく異なるため，本 FTC では採用しなかった．

#### 3.4.2 ユーザ空間におけるメモリ割り当て

同期タスクが使用するメモリ空間については，起動時に mlock システムコールにより全ての空間に予め物理メモリを割当てることにより．ページフォルトによって実行が阻害されないようにする．また，事前にアプリケーション全体のメモリ使用量を十分に設計して，メモリ不足が発生しないよう留意する必要がある．

#### 3.4.3 インターフェースライブラリのための malloc()変更

本 FTC が利用している Linux®は，アプリ向けのインターフェースライブラリとして glibc を利用している．glibc の実装では，printf()などの関数が一度目に実行された際に内部で使用するメモリを malloc()を用いて確保し，二度目以降はメモリ確保を行わないものがある．

こうした関数を再同期後にタスクが初めて実行した際，再同期した制御サーバだけがメモリ確保を実行することとなるため，メモリ確保のためのシステムコール内で schedule()が発生すると，処理の不一致を起こす原因となる．

そこで，コア 1 ではインターフェースライブラリ内の malloc()の延長で schedule()が発生しないように改修を加え，malloc()が途中でブロックされず走りきるようにすることで，schedule()呼出の有無を一致させている．

### 3.5 同期と非同期の境界面

プロセススケジューラによるタスク同期によって，コア 1 上のタスクは同期実行されることが保証されるが，本 FTC では外部と Voter を介して外部と通信しているため，同期タスクが直接通信を受信すると，通信タイミングによっては制御サーバ間で結果が異なる要因となることがある．

同様に非同期プロセスが，同期タスクを起動する場合，および，シグナルを送信した場合について，同期が継続できるよう対策を行う必要がある．対策内容については[1]で詳細に説明している．

## 4. おわりに

制御システム向けに FTC を複数の一般的な PC サーバとソフトウェアで実現するための計算機システム構成方法を提案した．

ソフトウェアによる FTC は，特殊なハードウェアを必要とせず，プロセッサ等のハードウェアの実装に依存しないため，ハードウェアの世代を超えた長期間運用出来る利点があり，今後，ますます発展する技術であると考えられる．特にソフトウェアによる FTC で，リアルタイム性と再組み込みを両立させたことは，大きな進歩であると言える．

### 参考文献

[1]関山 友輝, 他, “ソフトウェアによる制御用 FTC の実現に向けたオペレーティングシステム同期機能の開発”, FIT2010 第 9 回情報科学技術フォーラム (2010).

[2]吉田 雅徳, 他, “フォールトトレラントコンピュータ向け計算機再組み込みのためのメモリ再同期手法”, FIT2010 第 9 回情報科学技術フォーラム (2010).
[3]Sorin, Daniel J., "Fault Tolerant Computer Architecture," Morgan and Claypool Publishers, (2009)
[4]Schlözer, J., "Hazard Analysis of a Fail-safe Computer System based on COTS Components," SIGNAL+DRAHT 98, HEFT 10, pp.41-45, (2006)
[5]Bartlett, J., et al., "Fault Tolerance in Tandem Computer Systems," Tandem Technical Report 90.5, (1990)

Linux®は，Linus Torvalds の商標です.
その他記載の会社名，製品名はそれぞれの会社の商標もしくは登録商標です.