システムソフトウェアと
オペレーティング・システム 64－7
マルチメディア通信と分散処理 65－7
（１９９４．５．１９）

フォールトトレラントタプルスペース Hillary のデザイン

齋藤 正史[1] 曽我 正和[1] David Mankins[2] Noemi Paciorek[2]

[1] 三菱電機株式会社 情報システム研究所

[2] Horizon Research, Inc.

本稿では、プロセスの信頼性を実行時に向上させるためのツール、Hillary のデザインについて述べる。Hillary は、複数プロセスの協調動作のためのプロセス間通信を支援するものであり、並列計算のためのプログラミングパラダイムである Linda のモデルを拡張したものである。Hillary におけるタプルスペースは、冗長化されて保持されており障害に対して耐性を持っており、その意味でフォールトトレランシを提供している。

さらに、時間制約をもつ応用プログラムのための優先度の付与、複数のタプルスペースの提供、原子性保証のための機構の提供している。応用プログラムは、Hillary をプロセス間通信の基盤として用いることにより、複数のプロセスにおける協調処理、とりあけ冗長プロセスのモデルを単純化することがでる。

The Design of Hillary - A Fault-Tolerant Tuple Space System-

Masashi Saito[1] Masakazu Soga[1] David Mankins[2] Noemi Paciorek[2]

[1]Computer and Information Systems Lab., Mitsubishi Electric Corp.

[2] Horizon Research, Inc.

This paper describes the design of Hillary, a toolkit to increase realiability of processes. Hillary supports the inter-process communication among cooperative processes. Hillary's design is based on a coordination language, Linda, and extended to implement fault-tolrerancy using spacial redundancy.

In addition to this, Hillary also has rich functionality to support various application programs:

1. prioritized operation for time constraint programs,
2. multiple tuple spaces,
3. atomic operation.

Application programs can be simplified process cooperation models, especially redundant process models to increase reliability using Hillary.

1 はじめに

複数のコンピュータがネットワークにより接続された分散システムを用いて、社会の基盤を構成する応用システムが使用されるようになってきている。これらの応用システムの信頼性や安全性に対する要求は非常に厳しく、その要求を満足するシステムの構築には、多くの労力が費やされている。一方、半導体技術の発展により、これまで問題となっていたハードウェアの信頼性が向上するに従い、ソフトウェアに因る障害によりシステムが停止するという状況が増加してきている。

問題となるソフトウェア障害は、開発時点のテストで発見できなかったタイミングによるものや、開発するプログラムの複雑さが増加したことに起因したものも多い。さらに、オペレーティングシステムそのものの複雑さの増加により、オペレーティングシステムの障害が原因となる場合も増加してきている。

本稿では、ソフトウェアの信頼性を実行時に向上させるためのツール、Hillary のデザインについて述べる。Hillary は、複数プロセスの協調動作のためのプロセス間通信を支援するものであり、並列計算のためのプログラミングパラダイムである Linda[1][1] のモデルを拡張したものである。

Hillary を用いることにより、複雑なプロセス間通信モデルを単純化することができ、応用プログラムそのものの信頼性の向上も期待できる。さらに、Hillary のタプルスペースは、冗長化されており、プロセス間通信の信頼性の向上と冗長プロセスの実現のための基盤が提供できる。

2 Hillary 開発の目的

2.1. 障害の影響

ある問題をコンピュータシステムを用いて解決する場合には、その問題をひとつあるいは複数のプロセスにより抽象化する。ところが、抽象化されたプロセスの実行は、そのプロセス単独で行なわれるのではなく、下位層にある構成要素の存在を仮定している。

各構成要素は、ある意味で独立に動作しているが、各構成要素の障害は独立したものではなく、下位層の障害の影響は上位層の障害を導き出す。つまり、上位層の構成要素は、下位層の構成要素に依存している。図 1. に典型的な依存関係の例を示す。

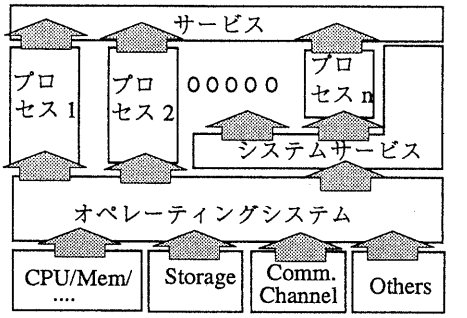

図 1. 障害の依存関係

図 1. 中の各構成要素は、それぞれ異なる形式の障害を起こす可能性がある。これは、障害のモデル [2], [3] により抽象化可能であるが、現実の障害は、障害が起きた瞬間に発見できることは少なく、しばらく後に別の現象として障害の影響が見えることが多い。

例えば、CPU や Memory は、単純にクラッシュする場合もあれば、ある特定の命令に対して任意障害を起こす可能性もある。障害が起こった場合には、処理終了後の検算により始めて発見されたり、オペレーティングシステムの不具合としてクラッシュ障害として扱われる場合もある。その意味で、障害の直接の原因を究明するのは非常に骨のおれる仕事である。

ある下位層の障害は結果的に上位層に影響を及ぼすことに注目し、応用プロセスとオペレーティングシステム層の間で、障害対処を行なう方式を採用する。これにより、オペレーティングシステムのクラッシュ障害、例えば Unix の Panic と CPU の障害などを区別する必要がなくなり、障害をそのものを単純に取り扱うことが可能となる。

これらの障害要因の区別は、リカバリのためには必要となるが、本稿ではその情報は別の手段により提供される仮定している。

1. Linda はアメリカの Scientific Computing Associates の登録商標です。

2.2. 対象システムと要求事項

障害に対する耐性を重視するシステムは、発電所や交通システムなどの制御システムがある。これらのシステムは、センサ、アクチュエイタとこれらを制御する複数のコンピュータにより構成される。

このようなシステムは、リアクティブシステムとしてモデル化される場合が多く、環境の変化に応じて、環境に対して何らかの動作を行なう。ところが、その動作に伴う値が更に別のセンサの値を変えることになり、結果的には図2.のようなプロセス間通信が必要となる。

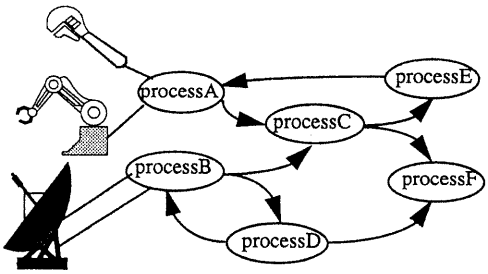

図2. リアクティブシステムのプロセス構成例

このようなシステムにおいて障害をマスクする、もしくは影響を最小限にとどめることにより、システムの信頼性を向上させる必要がある。さらに、このようなプロセス群が協調して動作する場合には以下のような要求がある。

1. プロセスは時間制約を持つ
2. 構成要素の分散は本質的
3. データの流れはクライアント/サーバとは限らない
4. 繰り返し可能な要求/返答としてモデル化が困難
5. 各プロセスが入力データに依存して状態を変化

2.3. 冗長システム

2.2.節で示した要求を満たしながら、信頼性を向上させるための手法として、以下の 2 つがある。

1. 時間的冗長度の増加によるもの
2. 空間的冗長度の増加によるもの

時間的冗長度の増加とは、障害が起こった場合に、その障害を発見した後、同一の処理を再実行し、処理を完了する方法である。処理の再実行は、最初から実行し直す方法でも良いが、チェックポイントをとることにより途中状態を保存し、途中状態からの再実行により、回復にかかる時間を減少させるという手法をとる場合が多い。

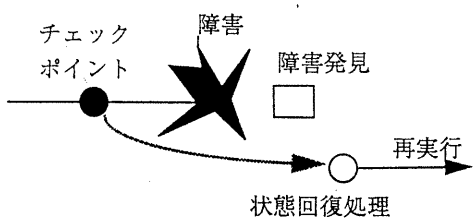

図3. 時間的冗長度の増加

ところが、チェックポイントをとるという処理は、障害が起こる可能性に依存せずに実行しなければならず、正常実行での性能低下が起こるだけでなく、回復にかかる時間の上限値も非常に大きなものとなる。

回復までにかかる時間は、障害が起きてから、その発見までのかかる時間と、直前のチェックポイントのチェックポイントでの状態の回復にかかる時間となり、プロセスが時間制約を持つ場合には、その要求を満足することは難しい。

空間的冗長度の場合には、同時に複数のプロセスにより同一の処理を行ない、障害が起きた場合にも、継続的に処理を行なうものである。したがって、プロセスの実行のためには、冗長度に応じた資源、CPU サイクルやメモリ容量が必要となる。

これらのプロセスを同一コンピュータ上で実行した場合には、コンピュータそのものの障害により、冗長化されたプロセス群が停止することになり、障害の影響伝播を考えた場合には、良い解決方法であるとは言えない。

この問題は、自律的に動作する複数のコンピュータ上で冗長プロセスを実行することにより解決できる。この場合には、冗長プロセスへの入力を同時に行ない、かつ、入力データの順序がある意味を持つマルチキャストプロトコルを用いる (図4)[4]。

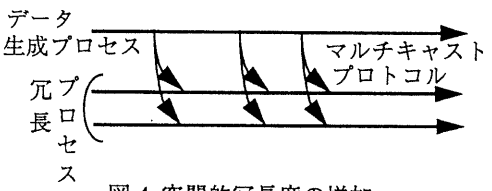

図 4. 空間的冗長度の増加

空間的冗長度の増加を用いた場合には、時間制約についても満足するようなシステムを構成可能である。しかし、冗長化されたプロセスが出力を行なう場合には、あるプロセス郡から他のプロセス郡への Many-To-Many のマルチキャストのプロトコルを使用する必要がある。

プロセスの階層が小さい場合には、最後にメッセージを受けとるプロセスにおいて、すべてのメッセージを受けとることが可能であるが、図 2 で示したシステムにおいては、このようなプロセスが明確ではない。

このようなシステムにおいて、同様の冗長性を保持するためには、すべての冗長プロセスの間で出力に対する合意を行ない、合意した値を他プロセスに出力を行なうとする方法がある。これにより、出力を行なう前に合意プロトコルを実行し、出力するならば、通常の One-to-Many のマルチキャストプロトコルが使用可能となる。

しかし、このような複雑な処理を応用プログラムで行なうことを強要するのは、応用プログラムの複雑さを増加することになり、障害の温床となりやすい。

3 Hillary のデザイン

3.1. Hillary の概要

Hillary はこのような問題点を、複数プロセスの協調動作でのプロセス間通信を支援することにより解決するものである。

プロセス間通信のパラダイムとして、すべてのプロセスが冗長プロセス郡として実装されている場合にもプロセス間通信と簡便なインタフェースを提供することを目的として、並行プロセスの協調のためのモデルである Linda に注目した。Linda を用いれば、応用プロセスは複雑なプロセス間通信を、タプルスペースへのアクセスとして簡潔に記述することが可能となる。

Hillary は Linda のタプルスペースを冗長化にしたものであり、以下の特徴を持つ。

1. フォールトトレランシの付与
2. 時間制約のための拡張の提供
3. タプルスペースへの持続性の付与
4. 複数のタプルスペースの抵抗
5. 原子性保証のための機構の提供

3.2. Hillary のインタフェース

Hillary は Linda のインタフェースを拡張したものを提供している。Hillary では、複数のタプルスペースを提供しているため、ts_creat()/ts_destroy() 操作により名前サービスを経由して得られるハンドルをタプル識別子として、すべての操作で使用する。従って、Hillary のアプリケーションインタフェースは以下のようになる。

```
in(ts_handle_type tsid, va_list args)
out(ts_handle_type tsid, va_list args)
rd(ts_handle_type tsid, va_list args)
```

さらに、これらの非ブロック操作として、

```
inp(ts_handle_type tsid, va_list args)
outp(ts_handle_type tsid, va_list args)
rdp(ts_handle_type tsid, va_list args)
```

を提供する。ここで、va_list 型で示した可変長引数が、Linda のタプルに相当する。さらに、優先度つき出力のための操作として、pout も提供する。pout は、out 操作と同等の動作を行なうが、指定された優先度に基づく送信順序の管理を行なう。

```
pout(ts_handle_type tsid,
        unsigned tuple_priority, va_list args)
```

in/out などの操作は、同期操作であり、要求に応じて動的タプルスペース割り当てを行なっている。このような操作に加えて、時間制約の強いシステムで要求される非同期入出力操作、タプルスペースの先割り当て操作の提供を検討している。しかし、現段階では実装上の制約もあり、対象外としている。

この他に、複数の操作を原子操作として取り扱うための操作として、以下の 3 つの操作を提供する。これらは時間的冗長度の提供するため

の基本操作として使用される。

```
atomic_handle_type atomic_id = begin_atomic()
commit_atomic(atomic_handle_type atomic_id)
abort_atomic(atomic_handle_type atomic_id)
```

新しいスレッドの実行を開始する eval も提供するが、現在は Hillary のサーバが内部的に提供するものだけを対象にしている。

3.3. Hillary による障害の対処

Hillary による障害の対処方法を説明するために、簡単なクライアント/ サーバプログラムの例を用いる。サーバプログラムのスケルトンを図 5(a) に示す。この場合には、4 行目と 6 行目の間でサーバが障害を起こした場合には、クライアントからのリクエストが消失してしまう。このような状況を避けるために、サーバプログラムをひとまとまりの原子操作として記述する (図 5(b))。

クライアント側でも同様の手段を用い、回復可能な処理が記述できる (図 5(c))。

```
1:  server() {
2:      while( TRUE) {
3:          in(ts, "request", index, ?req);
4:          ..... /* service */
5:          out(ts, "response", index++, response);
6:      }
7:  }
```

(a) サーバプログラム

```
1:  server(){
2:      while(TRUE) {
3:          atomic_handle_type id;
4:          id = begin_atomic();
5:          in(ts, "request", index, ?req);
6:          ..... /* service */
7:          out(ts, "response", index, response);
8:          commit_atomic(id);
9:          index++;
10:     }
11: }
```

(b) リクエスト消失を防ぐサーバプログラム

```
1:  client(){
2:      int index;
3:      atomic_handle_type id;
4:      id=begin_atomic();
5:      in(ts, "server_ index", ?index);
6:      out(ts, "server_index", index+1);
7:      out(ts, "request", index, request);
8:      commit_atomic(id);
9:      in(ts, "response", index, ?response);
10: }
11:     (c) クライアントプログラム
```

図 5. Hillary のプログラム例

図 5. の例は応用プログラムが時間的冗長度を持ち信頼性を向上させている。

空間的冗長度を増加させるための、最も簡単な方法は、冗長度を示す数だけクライアントがリクエストを出す方法である。

サーバにおける実現方法としてはクライアントからのリクエストを必ず in() するのではなく、タプルスペースに保持される冗長度を示す値と検査し、予定された冗長度に達していない場合には rd() し、達している場合には in() する。このような方式を用いることにより、任意の冗長度を持つ実行が実現できる。

3.4. タプルスペースマネジャ

タプルスペースマネジャは複数の複製されたプロセスにより実行され、複製されたタプルスペースの管理を行なう。タプルスペースは ts_creat() 操作により生成され、その際に属性として以下の 4 種類の値を指定できる。

- 記憶領域属性: 揮発性/ 不揮発性
- スコープ: パブリック/ ローカル
- 記憶領域サイズ: 割り当てバイト数
- 冗長度: 複製を生成する数

記憶領域属性を不揮発とした場合には、タプルスペースでの冗長化を行なわずに障害から回復することが可能となる。しかし、ディスク等の低速の不揮発性記憶領域を使用する必要性があるため、性能とのトレードオフで選択しなければならない。

タプルスペースのアクセススコープとしては、生成したプロセスからのみアクセスできるローカルと、タプルスペースのハンドルを知っていればアクセス可能なパブリックが選択できる。現在のデザインでは、ハンドルの受渡しに関するセキュリティは考慮していない。

リクエストを受けたタプルスペースマネジャは、冗長度を指定に応じてそのスペースのためのプロセスグループを構成する。すべての out() 操作はこのプロセスグループに向けた高信頼マルチキャストにより実現される。in() 操作は、グループに対する高信頼マルチキャストによりすべての複製に対して通知され、最も先に返答を返した値を使用する。in() 操作の場合には、す

べての複製からそのタプルを削除する。

このマルチキャストのプロトコルセマンティクスとして、現在は全順序 (Total Order) のみを提供しているが、因果順序 (Causal Order) や、これらの組合わせついての検討も行なっている。

4 Hillary の実装

Hillary は多くの応用で使用されることも目的としている。この場合に、コンパイラの拡張ににより実装した場合には、そのコンパイラを多くの種類のコンピュータに移植する必要があり、現実的でない。

また、Hillary のようなシステムはヘテロジニアスな環境での使用が本質的であると考えている。つまり、オペレーティングシステムによる実装も適切でない。現在、Hillary は図 6. に示すに要素に分割し実装を行なっている。

Hillary の操作において、Sun-RPC のような変数の型に依存したプログラミングをすることは煩わしい。Hillary では、Preprocesosor によりこの問題を解決している。プリプロセッサは、タプルスペースで共有されるデータタイプを自動的に走査し、必要となるスタブコードを自動生成する。

Redundancy Management 層は、アプリケーションからの要求に応じて冗長度の確保を行なう。

Failure Detector は、Hillary のサービスに参加しているプロセスやコンピュータの状態を管理し、それらが動作状態を調べる。他の構成要素の多くは、この Failure Detector の動作に依存する。現在の実装では、Crash 障害のみを対象としている。

現在、これら構成要素を、Unix の上、かつ UDP/IP プロトコル上に socket インタフェースで実装し、最初のプロトタイプの評価を開始したところである。

5 おわりに

本稿ではフォールトトレラントタプルスペース、Hillary のデザインについて述べた。Hillary を用いることにより、複雑なプロセス間通信を簡単なモデルに基づき簡潔に記述できるだけでなく、信頼性の確保も行なえる。

しかし、現在のデザインそのものについても Linda のもつ簡潔性が失われている面もある。また、プロセスの冗長実行を直接的にサポートする機能も必要となろう。その意味で、現在は Hillary というツールキットの実装が完了したところから始まるといっても過言ではない。

今後も、プロトタイプの評価を通して、Hillary をフォールトトレランシを提供する効果的なツールキットとしていきたい。

[参考文献]

[1] Carriero, N. and Gelernter, D.: Linda in Context, Communications of the ACM, 32(4):444-458(1989).

[2] Hadzilacos, V. and Toueg, S.: Faulto-Tolerant Broadcasts and Related Problems, Distributed Systems, Mullender, S. (ed.), second edition, Addison-wesley (1993).

[3] 齋藤正史: 分散システムにおける強い障害モデルの考察, IPSJ-DPS Workshop(1993).

[4] Birman,K.P.: The process group approach to reliable distributed computing, Department of Computer Science, Cornell Universicy, TR93-1216 (1993).

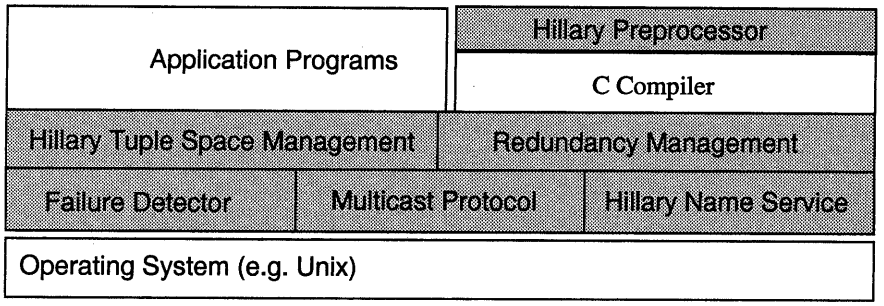

図 6. Hillary のソフトウェア構造