2G−5

UNIX プロセスの分散スナップショット

遠藤英幸　青柳龍也　有山正孝
電気通信大学

1 はじめに

ネットワーク環境が整うに従い、UNIX オペレーティングシステムにおいて、分散型のアプリケーションが増えつつある。しかしそれらにおいては、一般にクラッシュのリカバリーについてはあまり考慮がなされていないのが現状である。

分散型のアプリケーションは、シミュレーションなど非常に長時間の計算を必要とするものも多く、計算途中において何らかの原因で — マシンの一つがシステムダウンした。ネットワークが輻輳を起こした。などなど — 計算がストップした場合、その損失は時として非常に大きなものとなる。

これらのことを考えると、リカバリーの必要性は高いものと思われる。

分散システムにおいてクラッシュのリカバリーを考えるには、その分散システム中の各プロセスと通信チャネルの一貫性のとれた大域的な状態 — スナップショット — を得る必要がある。

このスナップショットを得るアルゴリズムとして、K. M. Chandy と L. Lamport によって発表された分散スナップショット・アルゴリズム [1] がある。

本研究では、この分散スナップショットのアルゴリズムを用いて、UNIX のプロセスによって構成された分散システムの、一貫性のとれた大域的な状態を得ることを試み、クラッシュからのリカバリーについて考察する。

2 分散スナップショット・アルゴリズムの概要

K. M. Chandy と L. Lamport によって発表された分散スナップショット・アルゴリズムは最初から一貫性のとれた大域状態をとるアルゴリズムである。本研究ではこの分散スナップショットアルゴリズムを用いる。

以下、この分散スナップショット・アルゴリズムについて説明する。

説明のため、図1のような、二つのプロセスと二つの通信チャネルから成る簡単な分散システムを考える。

p と q はプロセスであり、c および c' は、これを通ってメッセージが送られる通信チャネルである。

この分散システム上で、プロセス p、q およびチャネル c、c'の状態を記録することを考える。

スナップショットを取るアルゴリズムは、ベースとなる分散システムの計算をサポートをする作業であり、したがって、ベースとなる計算に影響を与えないようにしなければ意味がない。

メーセージは一般にベースとなる計算によって送受信されるが、分散スナップショット・アルゴリズムによって使用される、マーカーと呼ばれる特別なメッセージを考える。

そして、p および q について、以下の二つのルールを導入する。

■ プロセス p のマーカー送信ルール
p と接続していて、p から出ていくすべてのチャネル c に関して、p が自分の状態を記録した場合、c に次のメッセージを送信するよりも前に、c を通してマーカーを送信する。

■ プロセス q のマーカー受信ルール
チャネル c から、マーカーを受信したとき：

```
if q は自分の状態を記録していない then
  begin q は自分の状態を記録する;
        q は c の状態を「空列」として記録する
  end
else
  q は c の状態を、「q の状態を記録してから
  マーカーを受信するまでの間に c から届いた
  メッセージ列」として記録する。
```

このアルゴリズムを用いて、UNIX プロセスによって構成される分散システムのスナップショットを取ることを試みる。

3 UNIX プロセスの状態の取得について

実際に UNIX のプロセスの状態を記録しようとする場合、いったい何を保存すれば必要十分なのかと云うことが問題となる。

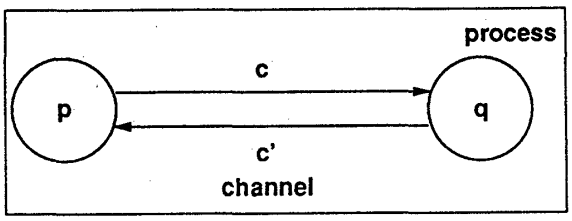

図 1: 簡単な分散システムのモデル

*Distributed snapshot of UNIX processes.
ENDOU Hydeyuki, AOYAGI Tatsuya, ARIYAMA Masataka
The University of Electro-Communications

一つの方法としては、プロセスのメモリー・イメージをダンプする方法が考えられる [2]。この方法によると、非常に汎用性の高いシステムを構築できる。あるいは、ベースとなる分散システムのプログラム時に、記録を行なう変数を指定して、スナップショットを取る時にはこの変数群だけを記録し、リカバリーも分散システムごとに用意する [3] と云う方法も考えられる。

UNIX プロセスの状態を記録することに関しては、実際にシステムを実装し、テストを行なって検討する必要がある。

4 システムの実装に関して

基本とする分散プログラムは、プロセス間通信の手段としては socket を用いるものとする。そしてスナップショットを取るため、ソケットライブラリをそっくりマスクする一連の関数を用意する。このライブラリ群を用いて分散システムを構築した際のプログラミング上の変更点を、従来のソケット・ライブラリを用いた場合と最小限にするため、このライブラリ群のふるまいは、オリジナルのシステムコールと同じにする。

これらのルーチンの名前には、プレフィックスとして snapshot を意味する ss が付けられている。

socket 関係の connect、accept、close のシステムコールに対応して、ss_connect、ss_accept、ss_close と云う関数を用意する。これらの関数は、プロセス間通信の接続の状態を記録する以外は、対応するシステムコールと同様に働く。

また、マーカーのやりとりとその処理をするために、read、write システムコールに相当するルーチンとして、ss_read、ss_write と云う関数を導入する。これは、マーカーがやりとりされる時以外は、read、write システムコールと全く同様に動作する。

マーカーが送られて来た場合 ss_read はプロセスの状態を記録する実際の作業を行なう。そして、次に ss_write が呼ばれたときに、メッセージを送るのに先だってマーカーを送り出す動作を行なう。

5 実装上の課題

以下に、このシステムを実装するうえで、考えられる課題を列挙する。

- ファイルの扱いが困難。

  スナップショットを取ったときに、ファイルも記録しなければならないが、ファイルのサイズによっては、記録を取るのに無視できない時間がかかり、ベースとなる計算を妨害する結果となる。多くの場合、これは非現実的である。

  しかし、計算途中の、ファイルへのアクセスは非常に基本的なことであり、これができないと制限が大きくなりすぎる。限定された形であっても、ファイルの状態の記録を可能にする必要がある。

- 接続の状態が動的に変化するシステムへの対応。

- 例えばあるホストがシステムダウンしたとき、そのホストが長時間リカバリされない場合などが多いことが考えられる。そのような状況に対処するために、リカバリーを行なうホストを、別なホストに置き換える機能が必要とされるかもしれない。

- 自発的に自己の状態を記録する、いわばトリガーとなるプロセスはうまく選ばなくてはならない。この選択を自動化できるのかどうか、検討が必要である。

- スナップショットをとるタイミングや頻度はどの程度制御できるようにするのが良いか。

6 おわりに

以上本稿では、UNIX プロセスの分散スナップショットをとるシステムについて、その方法と、実装する上での課題、問題点を述べた。

性能の制限やプログラム上の制約も多いが、分散アプリケーションの種類によっては、有用であると思われる。

現在、実験用のシステムを試作中である。今後は、実験システムを用いて、パフォーマンスの問題などについて検討していく予定である。

なお、一貫性のとれた大域状態をとるアルゴリズムとしては、ここで紹介したアルゴリズムの他に、Tony T-Y. Juang と S. Venkatesan によるアルゴリズムもある [4]。このアルゴリズムは、分散システムを構成する各プロセスが自分自身の状態を個別に記録し、クラッシュした場合に時間的にロールバックして、記録されたプロセスの状態の集合の中から、一貫性のとれた組み合わせを発見すると云うアルゴリズムである。こちらのアルゴリズムの検討も行なう予定である。

参考文献

[1] Chandy, K. M & Lamport, L. :
"Distributed Snapshots: Determining Global States of Distributed Systems."
ACM TOCS, Vol. 3, No.1, 1985

[2] 多田 芳克 & 寺田 実:
"移植性・拡張性に優れた C のコルーチンライブラリー実現法"
電子情報通信学会論文誌 (D-I) J73-D-I No.12 (1990/12)

[3] Liskov, B. & Schheifer, R. :
"Guardians and Actions: Linguistic Support for Robust, Distributed Programs." *ACM TOPLS*, Vol. 5, No.3, 1983

[4] Juang, Tony. T-Y & Venkatesan, S.:
"Efficient Algorithms for Crash Recovery in Distributed Systems."
LNCS 472, December 1990