3K−7

メッセージの追い越しがある分散環境における プロセスの実行停止 / 再開方式

六沢 一昭　山本 礼己　川合 英夫　今井 明　仲瀬 明彦
沖電気工業(株) 総合システム研究所　(財) 新世代コンピュータ技術開発機構　(株) 東芝総合研究所

1 はじめに

プロセス群の、終了検出と強制終了、実行の停止 / 再開は分散環境におけるプロセス制御の基本機能である。終了検出と強制終了については既に発表した[1]。本稿では実行の停止 / 再開処理について述べる。

強制終了と停止 / 再開は本質的に異なる。強制終了は1回行なうだけであるが、停止 / 再開は何回でも繰返しうる。このため、強制終了では1種類のメッセージを送るだけで処理できるため制御メッセージ間の追い越しは問題にならなかったが、停止 / 再開では停止を行なうメッセージ(%stop)と再開を行なうメッセージ(%start)を何回でも送信しうるので、互いの追い越しを考慮することが必要である。

本稿では、%stop と %start の送信を1ビット情報を用いて制御することにより実行の停止 / 再開を行なう方式について述べる。

2 計算モデル

以下に示す計算モデルを仮定する。

- 有限個のプロセッサ(PE)があり、互いに結合されている。通信は非同期なメッセージによって行なう。メッセージの追い越しの起こる可能性があり、送信した順序で到着するとは限らない。
- システムには有限個のプロセスプールがあり異なったIDを持つ。プロセスプールは1つの制御プロセスと有限個の子プロセス(以下、"プロセス"と略す)からなる。プロセスはいつでも終了しうる。また同じIDを持つ新しいプロセスをいつでも生成できる。
- 負荷分散などのためにプロセスを他のPEへ投げ出すことがある。投げ出されてから到着するまでの時間は不定である。

同一PEに存在する同じIDを持つプロセス群はサブプールを構成する。PEが受信したプロセスは同じIDのサブプールに加えられる。同じIDのサブプールが存在しなかった場合は新しく生成する。サブプールは属するプロセスの実行を停止 / 再開することができる。また、属するプロセスがすべて終了すると終了し、制御プロセスへ終了を示すメッセージ(%terminated)を送る。

実行の停止 / 再開とは、制御プロセスがメッセージを送ることによって同じプロセスプールに属するすべてのプロセスの実行を停止 / 再開させることとする。ただしすべてのプロセスが停止 / 再開したことの確認は行なわない。

3 方針

処理を非同期、局所的(↔ 同期、大域的)に行なうため、以下の方針を設定した。

1. 制御メッセージ間の追い越しを(例えば到着確認メッセージによって)防止することは行なわない。
2. 制御メッセージ(%stop など)の放送は行なわない。各PEはサブプールの生成をメッセージ(%ready)によって制御プロセスへ伝え、制御プロセスはサブプールの存在する(ことが期待される)PEへのみ制御メッセージを送る。

制御プロセスは%readyを受信したら送信元PEを記憶し%terminatedを受信したら削除する。記憶しているPEにはサブプールが存在する(ことが期待される)。[1]

4 必要なこと

%stop と %start が互いに追い越す可能性があるので、サブプールは%stop と %start の受信数の差で実行の停止 / 再開を決定するしかない。このため「%readyを受信したら制御プロセスの状態(停止 or 実行中)に従って%stop あるいは%start を送信する」という方式はうまく行かない。サブプールは何度でも生成 / 終了を繰り返し%readyを送信するので、%stop、%start を無秩序に何回でも送信してしまうからである(図1)。

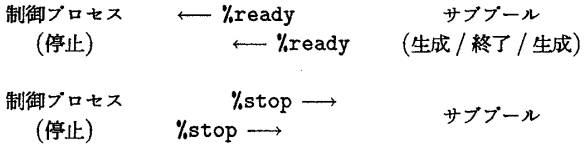

図1: %stop を何回でも送信してしまう.

%stop と %start について「サブプールに受信されなかった数」と「終了したサブプールが受信していた数」

An Execution Control for Distributed Processing Systems with Non-FIFO Communication
Kazuaki ROKUSAWA (OKI), Reki YAMAMOTO, Hideo KAWAI, Akira IMAI (ICOT), Akihiko NAKASE (TOSHIBA)

[1] %ready と %terminated の追い越しにはカウンタを設けて対処する[1]。

を制御プロセスが知ることができれば実行の停止 / 再開を制御することができる。これらとメッセージの送信数からサブプールによって受信される (ことが期待される) %stop 及び %start の数がわかり、それによりサブプールの状態 (停止 or 実行中) を把握できるからである。サブプールの状態がわかれば制御プロセスの状態に応じて %stop あるいは %start を必要な数だけ送信すればよい。

5 stop ビットを用いる方式

前述した処理は、制御プロセスが PE ごとにカウンタを持つことによって実現できる。しかし %stop 及び %start の受信数を

『同数』あるいは『%stop がひとつ多い』

の 2 状態に限定すると、各 PE に対して 1 ビット情報を持つだけで実現可能になる。ビット (stop ビット) の意味は以下のとうり。

ON (対応する PE へ)%stop をひとつ多く送信している。サブプールは (もし存在するなら) 停止している (する)。

OFF 同数の %stop と %start を送信している。サブプールは (もし存在するなら) 実行中である (になる)。

実行の停止 / 再開処理を以下に示す。

サブプールの存在しない PE が %stop あるいは %start を受信したら、%nakStop、%nakStart を返信する。

サブプール は、停止中に %start を受信したら実行を再開し、%stop を受信したら %nakStop を返信する。実行中に %start を受信したら %nakStart を返信し、%stop を受信したら実行を停止する。

制御プロセス生成時 はすべての stop ビットを OFF に設定する (生成時は "実行中" であるとする)。

表 1: %ready 及び %nak を受信した時の処理

制御プロセス		停止	実行中
%ready 受信	ON	何もしない	%start 送信 OFFにセット
	OFF	%stop 送信 ONにセット	何もしない
%nakStop 受信	ON	処理 A	OFFにセット
	OFF	%stop 送信	%stop 送信
%nakStart 受信	ON	%start 送信	%start 送信
	OFF	ONにセット	処理 B

処理 A サブプールが存在するならば %stop 送信. 存在しなければ OFFにセットする.

処理 B サブプールが存在するならば %start 送信. 存在しなければ ONにセットする.

停止処理: 制御プロセスの状態を "停止" に設定し、サブプールの存在する (ことが期待される)PE のうち stop ビットが OFF であるものへ %stop を送信する。ビットは ON にセットする。

再開処理: 制御プロセスの状態を "実行中" に設定し、サブプールの存在する (ことが期待される)PE のうち stop ビットが ON であるものへ %start を送信する。ビットは OFF にセットする。

制御プロセス が %ready 及び %nakStop、%nakStart を受信した時の処理は表 1に示す。

制御メッセージが消滅する保証

以下に示すように通信中の制御メッセージは減少して行く。

制御プロセス - サブプール間 サブプールは、%stop を吸収する (%nak を返信しない) 状態と %start を吸収する状態を交互に繰り返す (表 2)。

制御プロセス - サブプールの存在しない PE 間 制御プロセスは %nakStop を吸収する状態と %nakStart を吸収する状態を交互に繰り返す (表 3)。

%stop と %start の送信数はつり合っているので、制御メッセージは有限時間内にすべてなくなることが保証される。

表 2: %stop あるいは %start の受信

サブプール	停止	実行中
%stop 受信	%nakStop 送信	停止する
%start 受信	実行を再開する	%nakStart 送信

表 3: サブプールの存在しない PE からの %nak 受信

制御プロセス	ON	OFF
%nakStop 受信	OFFにセット	%stop 送信
%nakStart 受信	%start 送信	ONにセット

6 おわりに

メッセージの追い越しがある分散環境におけるプロセス群の実行の停止 / 再開処理について述べた。本方式では、%stop 及び %start の受信数を 2 状態に限ることにより、PE あたり 1 ビットを制御プロセスに持たせるだけで停止 / 再開を制御することができる。

参考文献

[1] K.Rokusawa, N.Ichiyoshi et. al. "An Efficient Termination Detection and Abortion Algorithm for Distributed Processing Systems," ICPP, 1988.