分散OS XEROにおけるコンテキストの分散かつ永続的な管理法について

2G−3

成田 篤信　加藤 和彦　猪原 茂和　益田 隆司
東京大学 理学部 情報科学科

1 はじめに

分散 OS XERO における全てのプログラムやデータは、コンテキストと呼ばれるモジュールとして統一的に管理される。永続空間に保管されているコンテキストをタスク上に動的にロードし、実行を行ない、再び永続空間にアンロードするという一連の操作を繰り返すことにより計算が進行する。タスク上には必要に応じ複数のコンテキストをロードし、それらを結合/分離させながら計算を行なう [1]。

一方、タスクにロードされていないコンテキストは、DSRと呼ばれる分散透明な空間に永続的に保存される。この空間は分散環境内の各ホストから統一されたインターフェースでアクセスされ、コンテキストの入出力、また、プログラム間の通信や同期などにも利用することができるように設計されている。

このプログラミングモデルを用いると、プログラムの動的な結合と分離、プログラムの実行状態の永続的保存、プロセッサ間でのプログラム移送などが自然に表現できるようになる。本文では、この XERO のプログラミングモデルについて簡単に説明した後、コンテキストを永続的かつ分散的に管理する方法について述べる。また、永続空間に対するコンテキストの入出力方法の設計と分散管理の実現および実験結果について説明する。

2 XEROのプログラミングモデル

分散OS XEROにおけるプログラミングモデルは、図1のように表される。このプログラミングモデルは次の4つの概念で整理される。線形的にアドレッシングが可能な仮想アドレス空間であるタスク (task)、システムが扱うプログラムやデータの基本単位であるコンテキスト (context)、CPU による計算の実行経路であるスレッド (thread)、そして、コンテキストを永続的に格納するための格納庫 DSR(Distributed Shared Repository) である。

XERO におけるデータや実行プログラムは全てコンテキストという基本単位として扱われ、通常は DSR の中に永続的に格納されている。コンテキストを DSR から取り出し、タスク上にロードし、その上をスレッドが走行することによって計算が進行する。一つのタスクには必要に応じて複数のコンテキストをロード、実行することができ、コンテキスト間でシンボルを動的にリンクさせることにより、プログラムのより緊密な協調実行を行うことができる。また、タスク上にロードされているコンテキストはいつでも DSR に書き戻すことができる。この動作をコンテキストのアンロードと呼ぶ。たとえ計算の進行中のコンテキストであっても、その計算状態を保存したまま永続的に格納できるようになっている [3]。

タスクの生成時にはまず最初に特権を持ったタスク内管理プログラムが最初にロードされ、制御がそこに移される。このプログラムはタスクスーパーバイザ (以下 TSV) と呼ばれ、タスク上にロードされるコンテキストの様々な管理を行なう。TSV は、DSR からのコンテキストのロード / アンロード、アドレス空間の割当、スレッドの割当てと制御、コンテキスト間のシンボルのリンクや通信等の機能を提供し、タスク消滅時まで存在している。TSV はカーネルの機能の一部をユーザ空間に持ってきたものと考えることができる。このため、使用環境に従って TSV を比較的容易に変更できるようになっている。

一方、DSR はコンテキストを分散的に永続保存する仮想的な格納庫である。この格納庫は、一つのネットワーク内に1つだけ存在し、ネットワークに結合されたどのサイトからも同じ様に見え、決められた入出力操作で読み書きが行われる。DSR を利用すると、コンテキスト間/ネットワーク間の通信、同期、プログラムのマイグレーション等を実現することができる [2]。

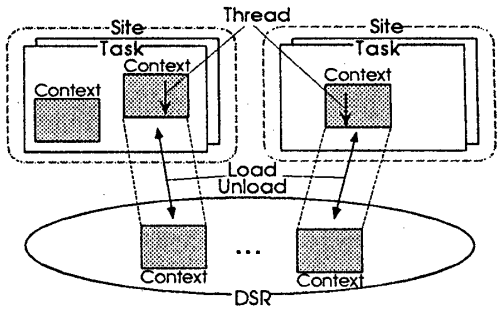

図 1: XERO のプログラミングモデル

3 DSR の実現

3.1 DSR のインターフェース

DSR はコンテキストを分散的に永続保管する格納庫であり、基本的にネットワーク内のどこからも同じ空間であるように見える。DSR に保管されている各コンテキストには DSR 内で一意に決まるキーが与えられており、それをもとにコンテキストの取り出し、格納、読み書きという操作が行なわれる。DSR に対する基本操作とそのデータ型を以下に示す。

```
context_in:   key → addr
context_rd:   key → addr
context_out:  (addr,key) → unit
context_wt:   (addr,key) → unit
```

Persistent Management of Contexts in XERO Distributed Operating System, by Atsunobu NARITA, Kazuhiko KATO, Shigekazu INOHARA, and Takashi MASUDA (Dept. of Information Science, Univ. of Tokyo)

context_in はDSRにあるコンテキストをタスクにロードしそのコンテキストをDSRから消去する。context_out は反対にタスク上のコンテキストにキーを与え、DSRに格納する。この時、タスク上のコンテキストは消去される。また、context_rd はDSRからコンテキストを消去せずにコンテキストの読み出しを行い、context_wt は、タスク上のコンテキストを消去せずにコンテキストをDSRに書き込む操作である。コンテキストに対する各操作は直列化され、矛盾なくコンテキストの移動・読み書きが行われる。

3.2 DSR の利用例

コンテキストマイグレーション XEROのコンテキストは、たとえ実行中であってもDSRという空間を通ってどのホストへも移送が可能である。特に、従来のプロセスマイグレーションに比べ、アドレス空間の生成という大きなオーバーヘッドを伴う操作を必要としないため、マイグレーションにかかるコストが軽減されている。また、プロセスよりも細かい単位で移送を行うことができる点も特長の一つである。

プログラミング言語への永続性の付与 これまでのプログラミング言語は、主記憶上の揮発的データを操作の主な対象としてきた。DSRプログラミングモデルは特定のプログラミング言語に依存しない技術であり、ポインタを含むデータやプログラムの実行イメージまでも、ユーザに意識させずに、永続的に保存できるものである。このため、このDSRのモデルは任意のプログラミングモデルに永続性を与える基本的な方法となると考えられる。

3.3 DSR Manager

ネットワーク内の各ホストには、DSRに対する操作を提供するDSR Managerというプログラムが動いている。各DSR Managerは協調して動作し、コンテキストの分散格納管理やキーの名前管理を行い、各ユーザに対して分散透明なインターフェースを提供するようになっている。

また、DSRにコンテキストの格納場所は物理的に分散している。コンテキストの物理位置は基本的にはユーザには透明になっており、どこのサイトからも、同じ操作によってアクセスが可能となっている。

ユーザコンテキストがコンテキスト入出力要求を発すると、まずTSVがこの要求を受け取り、ローカルなDSR Managerに要求を出す。必要なコンテキストがそのホスト内に管理されていない場合は、DSR Managerはさらに他のホストのDSR Managerにその要求を転送する。コンテキストが見つかり要求が受け入れられると、コンテキストの存在するサイトのDSR Managerはコンテキストの転送の準備を整えた後、転送許可のメッセージと転送を行なうためのポート番号を、要求を発行したTSVに返す。確認を受けたTSVは、コンテキストを直接管理しているDSR Managerとの間でコンテキストの転送を開始する。

DSRに格納されるコンテキストに与えるキーは、物理位置独立でネットワーク内で一意に決まるものとしている。キーからコンテキストの物理位置を得る方法は、WelchとOusterhoutらによって提案されたプレフィックステーブルの技術[4]を用いることにした。キーの名前空間を階層構造とし、キーのプレフィックスを見ることによりその物理位置を特定するという方法である。

DSRに対するコンテキストの入出力機構を利用すると、ネットワーク間のメッセージ通信を実現することができる。送り側はコンテキストに特定のキーをつけてDSRに投げ込み、メッセージ受取側はそのキーを持つコンテキストを取出す。この通信方法では通常、コンテキストをいったん永続記憶上に書き込むことになるが、送り側がcontext_out、受取側がcontext_inで通信を行なう時、受取側のcontext_inが先に実行された場合にはコンテキストを永続記憶上に書き込む必要はなく、送り側から受取側へコンテキストを直接移送すれば良いことになる。DSR Managerは、コンテキストの送り側受取側のプログラムには直接移送であるか否かは見えないような方法で、この直接移送をサポートするように設計されている。

4 実験

現在、XEROのプログラム実行環境の実現を進めており、これまでに、Sony NEWS 1750(CPU m68030, memory 8MB)上で、コンテキストを生成するためのコンパイラ、TSV、DSR Managerの実現を行なった。この環境を用い、簡単なプログラムを作成実行してみた。素数を小さいものから順に生成するプログラムを一つのコンテキスト(約16KB)として作成し、このコンテキストをEthernetによって結ばれた2つの計算機で交互にロードさせてみた。コンテキストの再配置処理および、DSR Managerの協調動作が有効に機能していることを確認した。また、このコンテキストのロード+アンロード処理に約0.6秒を要した。

5 おわりに

XEROのプログラミングモデルについて述べ、コンテキストの扱いについて説明した。さらにコンテキストを分散的、永続的に格納するDSRの概念について説明し、その実現法について述べた。さらに、このプログラミング環境の実現を行ない、実験結果について述べた。今後の課題として、DSRの様々な利用法に応じた最適化技法の開発と性能評価を行なう予定である。

参考文献

[1] 加藤, 猪原, 成田, 千葉, 益田. 無指向オペレーティングシステムXERO の設計. 第3回コンピュータシステムシンポジウム予稿集,p.1-9. 情報処理学会,March 1991.

[2] K. Kato, A. Narita, S. Inohara and T. Masuda. Distributed shared repository : a unified approach to distributed and persistent information management. Technical Report of Dept. of Information Science, Univ. of Tokyo.

[3] 成田, 加藤, 猪原, 益田. 分散 OS XERO におけるマルチコンテキストの実現方式について. 情報処理学会 SWoPP '91 予稿集 OS-51-4 p.25-32 1991.

[4] B. Welch and J. Ousterhout. Prefix tables: a simple mechanism for locating files in a distributed system. In Proc. IEEE Int. Conf. on Distributed Computing Systems, pages 184-189, 1986.