5N-7

システムXEROにおけるプロセス間大量データ通信機能

猪原茂和　溝上敏文　加藤和彦　益田隆司

東京大学　理学部

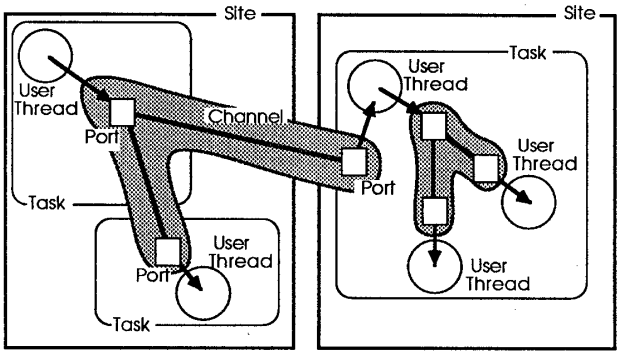
1. はじめに

近年のコンピュータ技術の進歩はめざましく、LANとワークステーションを用いた分散環境は急速に普及し始めている。分散環境上では処理すべきデータも各サイトに分散しているため、ローカルサイト内だけで処理を行なう場合に比べてアプリケーション開発にかかる労力は非常に大きい。分散環境上で効率良く動作するData Intensive Applicationの開発のためには、新たなプログラミング支援・分散実行支援・永続的オブジェクト管理支援の各環境が必要となる("Data Intensive Application"とは、データ　－それも大型で様々な内部構造を持っている－　に対する処理が中心となるアプリケーションを総称した言葉である［加藤89］)。

我々は、分散処理環境においてData Intensive Applicationの構築をサポートするためのシステムXEROの設計と開発を行っている［加藤89］［脇田89］。本稿では、XEROにおけるData Intensive Applicationの分散実行支援環境について述べる。

2. XEROにおける分散実行環境の概観

この章ではXEROにおける分散実行環境の概観について説明する。XEROでは分散環境として下図のような状況を想定している。



2.1 タスク、スレッド、ユーザスレッド

システム資源（仮想アドレス空間、ファイルディスクリプタ等）はタスクを単位として割り当てられる。タスクはその中に仮想並列に実行されるスレッドを複数持つ。同一タスク内の各スレッドは、アドレス空間を共有しているため、データを授受する場合に必ずしもコピーを行なう必要がない。また、同一タスク内でのスレッド切り替えは、ページレジスタやその他のシステム情報を書き換えることなく行なえるので、UNIXなどにおけるプロセス切り替えに比べて速く行なうことができる。

アプリケーションを構成する各手続きは、スレッドとして実行される。これらのスレッドは特にユーザスレッドと呼んで、タスク内に同居するシステムスレッド（後述する「チャネル」のように、システム側が提供するスレッド）と区別することにする。

Interprocess Huge Data Communication for Distributed Data Intensive Applications :the XERO approach
Shigekazu INOHARA, Toshifumi MIZOKAMI, Kazuhiko KATO , and Takashi MASUDA
University of Tokyo

2.2 分散環境での大量データ通信

分散環境でのプログラムでは、システムからどのような通信機能が提供されているかが、効率に大きく影響する。とくにData Intensive Applicationの場合、大量のデータを効率良く送ることが性能上最も大切となる。

ユーザスレッド間で大量のデータを授受する場合、全体を1度に送るのではなく、部分的な構造を単位としたストリームとして送れば、処理にパイプライン的な並列性が出る。XEROでは、2つのユーザスレッドはストリームデータを流す「チャネル」によって通信を行なう。チャネルとは、ネットワーク上での通信を統一的に扱うために導入した仮想的な通信路である。チャネルを使うことによって、ユーザスレッドは、ネットワーク上で分散透明にストリームデータを受け渡しすることができるようになる。

3. Data Intensive Application用の通信機能に対する要求

協調的な動作をするユーザスレッド群が、相互にデータを流しあって処理を進めるためには、ユーザスレッド間のデータの通信路であるチャネルにいくつかの高度な機能が必要となる。この章では、Data Intensive Application の動作環境としてのユーザスレッド間の通信機能に対して要求される機能を考える。

3.1 ネットワーク透明性

ユーザスレッド間通信は、

- 異なるサイト上のユーザスレッド間
- 同一サイトの異なるタスク上のユーザスレッド間
- 同一タスク上のユーザスレッド間

の3種類に分けられる。しかし、各ユーザスレッドのプログラムが実行時の状況（ユーザスレッドがネットワークのどこに置かれるか、ネットワーク上のどのユーザスレッドと通信するかなど）と独立に記述できることは、プログラム開発上重要である。このためこれらの通信をユーザスレッドに対して透明に実現する必要がある。

3.2 ストリームの共有

複数のユーザスレッドが同じストリームを必要とする場合に、複数のユーザスレッドが一つのチャネルを参照することができれば、冗長な計算をしなくてすむ。これは、データベースの問い合わせ処理における共通部分式の除去［Maier］にあたる。消費者ユーザスレッド毎にデータの消費速度が異なると共に、同一ストリームを参照する消費者ユーザスレッドが動的に増えることもある。このため、チャネルは生成されたデータをチャネル消滅時まで保存しなければならない。

3.3 ストリームの再参照

処理によっては、1度参照したストリームを何度も参照し直さなければならないことがある。関係データベース演算の一つであるJoin演算をネステッドループアルゴリズムによって実現した場合などはその例である。チャネルが再参照の能力を持っていれば各ユーザスレッドが個々にコピーを作る必要がないので、サイト内に存在するストリームデータを冗長性なく保持できると共に、ディスクなしサイト上のユーザスレッドでも、どこにキャッシングのための領域を設けるかという問題を生じない。さらにユーザ

プログラムは部分的な再参照のためのキャッシングのみを行なえばよいので、記述を簡単にできる。

3.4 フロー制御

生産者スレッドと消費者スレッドの処理速度は一般に異なるので、データ授受の調整のためにフロー制御が必要になる。ユーザスレッドのスケジューリングによってフロー制御を行なうが、大きく分けて2つの方法が考えられる。

1つは消費者側の送ったデマンド（ストリームデータの要求）によって生産者側の計算が進行する「要求駆動」である。まず消費者側が生産者側に対しデマンドを送り、生産者側からの応答を待つ。両者を並列に動かすためには消費者側のデマンドを、実際の消費以前に「先出し」する必要がある。

もう1つは生産者側によって作られたデータによって消費者側の計算が進行する「データ駆動」である。生産者は 消費者のデマンドを待つことなくデータを生成できるが、消費者側と生産者側との間に溢れたデータをチャネルの管理するキャッシングディレクトリに記憶する必要がある。

データ駆動による制御を行なえば要求駆動による制御を行なった場合よりパイプライン性は高い。しかし消費者が必要としない余分なデータまでつくってしまう場合があること、キャッシングディレクトリへの入出力がオーバーヘッドとなりやすいことが欠点である。このためユーザスレッド間の通信をどのフロー制御によって行なうかを、ユーザが指定できるのがよい。

3.5 キャッシングディレクトリ管理

キャッシングディレクトリは、一度生成したストリームデータを保存しておくのに使われる。要求駆動でも消費者が複数の場合、各消費者からのデマンドの到着がまちまちであるから、これらのデマンドのうち最初に到着したものによって生産者側の計算を進行させ、生成したストリームデータはキャッシングディレクトリに記憶していく。2番目以降のデマンドにはキャッシングディレクトリからデータを取り出すことになる。キャッシングディレクトリをストリーム生成側に置くか、ストリーム参照側に置くか、または（ディスクレスサイトの場合に）他のサイトに置くかによってネットワークを流れるデータ量が違い効率を大きく左右するので、キャッシングディレクトリの位置を指定できる必要がある。

3.6 ストリームデータの遅延アクセス機構

サイト間・タスク間の通信量はシステム全体の性能に大きな影響を与える。各サイトの永続的オブジェクトサーバが管理するデータのIDを送ることで、データ本体の転送をその値が必要となるまで遅延する。この機構により、必要最小限の通信量のみでアクセスすることが可能となる。

3.7 自由なユーザスレッド間の結合

UNIXのpipeでは生産者と消費者に血縁関係が必要だった。このような制限はプログラムの記述に大きな障害となる。任意のユーザスレッド同士が自由に結合すること、動的に結合する（結合すべきチャネルのIDをストリームデータにして授受する）ことが必要である。

4. XEROにおける分散環境上の通信機能の実現

前章ではチャネルに要求される諸機能について述べた。この章では、上記のようなXEROの通信機能を提供するチャネルの、実現法について述べる。

4.1 Port as an Active user object

チャネルの実体はポート（一個の入力ポートとN個の出力ポート）である。ポートはチャネルを介して受け渡されるデータの受け渡し口のことで、ユーザスレッドが結合している各チャネルに対して、1つづつ割り当てられる。MachのポートやUNIXのpipe、socketなどは、kernel内部に組み込まれている。XEROでは、ポートをユーザタスク内の1スレッド（ポートスレッド）として実現してある。こうすることで、

- kernelの中に組み込むよりも、高度な処理を実現しやすく、機能を後から付加することも容易にできる。
- ポートの機能の大部分の実現するためにはユーザスレッドと独立な制御をあたえるか、コルーチン的な動作が必要となる。ユーザタスク内のオブジェクトとして実現することによって、ポートの記述が自然なものになる。また仮想空間をユーザスレッドと共有するため、同一タスク内の通信の場合にはkernelを介さずに通信ができる。一方、コンテキストスイッチングによるオーバーヘッドであるが、スレッド切り替えが関数呼び出し程度の重さであることと、大量データを通信する場合にはコンキストスイッチングよりもコピー量が効率に影響を与えることから、性能に与える影響は少ないと考えられる。

4.2 チャネルとポートの生成・消滅・共有

ユーザスレッド間にチャネルを張るには、まずプリミティブcreate_channel( )でチャネルを作った後、open_port( )を各ユーザスレッドが行なって各自のポートを用意する、という手順を踏む。create_channel( )の戻り値はチャネルIDであり、これをストリームデータとして流すこともできる。これを用いれば、任意のユーザスレッド同士を結合することができる。チャネルは以下の要領で結合される。サイトAでcreate_channel( )を発行すると、サイトAに生成したチャネルのIDが登録される。サイトBでそのチャネルIDに対するopen_port( )を発行すると、サイトBではポートスレッドを作ると共に、サイトAに対してポートの結合要求を送る。さらにサイトCでそのチャネルIDに対するopen_port( )が発行されサイトAへ結合要求が送られると、サイトAはサイトBにあるポートとサイトCにあるポートに通信路を設定する。

4.3 実行制御

データの授受法の指定は、チャネル生成時に行なう。create_channel( )の引数に、データ駆動的な制御にするか要求駆動的な制御にするか、更に要求駆動の場合には、要求を先出しするかを指定する。

ストリームからデータを取り出すにはget( )を、作ったデータを流すにはput( )をもちいる。ストリームの再参照を始めるにはrewind( )を用いる。複数のチャネルに対するデマンドやデータの多重待ちが必要となる場合には、select( )を用いてアクセス可能になったチャネルを見つけることができる。

5. おわりに

分散環境上で効率良く動作するData Intensive Applicationの開発のための、大量データ通信機能について述べた。現在、上記のような機能を持った分散実行システムをSun WorkstationとIOMbpsのEthernetによる分散環境上にインプリメント中である。

参考文献

[加藤] 加藤 他: Data Intensive Application 構築支援システムXEROの構想, 情報処理学会第38回全国大会講演論文集, March 1989.

[脇田] 脇田 他: システムXEROにおける高水準データ定義／操作言語, 情報処理学会第38回全国大会講演論文集, March 1989.

[Maier] Maier, D.: TheTheory of Relational Databases, Computer Science Press, 1983.