計算機アーキテクチャ 80－5
（1990．1．25）

Linked Dataのその構造に基づく記憶空間の構成

前川博俊, 安田弘幸, 實藤隆則, 澤田佳明, 福田譲治

ソニー(株) 総合研究所

記号処理の分野において、ポインタによってリンクされたデータ (Linked Data) は重要で不可欠である。我々は Linked Data をその構造上の性質を活かして記憶空間上に表現し、効率の良い記憶管理を実現する記憶構成方式、SOSO (Structure Oriented Storage Organization) を考案した。SOSO は、従来の方式に比べて、主記憶と二次記憶からなる階層的記憶構成における記憶管理やガーベージ・コレクションを効率良く処理できる。また、ポインタのための記憶容量が少なくて済み、主記憶、二次記憶のそれぞれの空間を独立に拡張できるという柔軟性を持つ。我々は、シミュレーションによって SOSO の実行効率の良いことを確かめた。

A Structure Oriented Storage Organization for Linked Data

Hirotoshi MAEGAWA, Hiroyuki YASUDA, Takanori SANETO,
Yoshiaki SAWADA, and George FUKUDA

Corporate Research Laboratories
Sony Corporation
4-14-1 Asahi-cho
Atsugi, Kanagawa 243
Japan

We introduce a storage organization for linked data structures which are important and indispensable in symbolic processing. The organization called SOSO (Structure Oriented Storage Organization) is based on the structural characteristics of linked data and has an efficient capability of managing the structures on it. SOSO is capable of processing storage management and garbage collection on hierarchical storage systems consisting of main and secondary storages more efficiently than conventional storage systems. SOSO needs less storage capacity for pointers of structures. Spaces of main and secondary storages are able to be expanded independently. We confirmed the execution efficiency by a simulation.

1. はじめに

ポインタでリンクされたデータ(以下、Linked Data)は、知識処理や自然言語処理などの分野で広く用いられている。Linked Dataは、その構造の柔軟性故、記号処理の分野に向いているが、一方で、記憶空間上で物理的局所性が乏しいという性質を持つ。

現在の計算機システムでは、データはページングなどを用いた仮想記憶によって管理することが多い。Linked Dataのような局所性の少ないデータをそのような方式で管理したとき、ページング回数が増加し、システム全体の処理効率の低下する傾向がある。

我々はLinked Dataを、ポインタでリンクされた構造を持つという性質を活かして記憶空間上に表現する記憶構成方式、SOSO(Structure Oriented Storage Organization)を考案した[1-2]。SOSO上では、記憶管理やガーベージ・コレクション(GC)を効率良く実現できる。

2. Linked Dataとその管理

2.1. Linked Dataとその性質

Linked Dataは、構造的にdirected graphとなる(図2.1)。各ノードは、そのノードのデータと、複数の他のノードへのポインタを持つことができる。Linked Dataの大きな特徴のひとつは、ポインタの付け換えによる構造の柔軟性である。この柔軟性は一方で、データの物理的局所性を失わせる。この性質は、アドレスを基に記憶空間を管理しているアーキテクチャに向いているとは言えない。

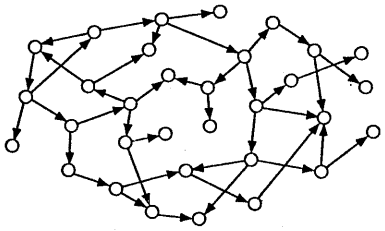


図2.1 Linked Data

2.2. 仮想記憶管理とその問題点

主記憶と二次記憶からなる階層的な記憶構成では通常、ページングなどを用いた仮想記憶[3]によってデータを管理する(図2.2)。Linked Dataをこのような方式で記憶管理したとき、そのデータの非局所性とガーベージのため、ページングによって主記憶・二次記憶間で転送されるデータには不要なものがあり、ページング回数の増加する傾向がある。ページングの頻度が多くなると、二次記憶アクセスが増加し、システム全体の処理効率が低下する。

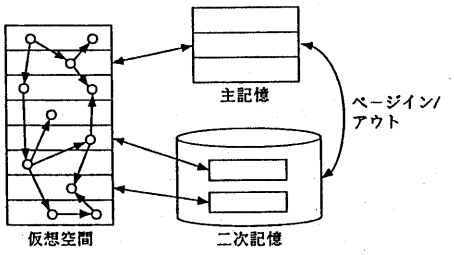


図2.2 ページングによる記憶管理

記憶空間上でデータの局所性を保つこととガーベージを減らすことができれば、不要なデータ転送を減少させることが可能となる。そのため、ページングやGCについて種々の試みがなされているが[4-9]、処理効率を大きく向上させることは難しい。コピー方式やコンパクション方式のGCによって、定期的に局所性を作る方法では、記憶の空間効率が良くなく、GCが起動される前はデータの局所性を大きくは期待できない。アプリケーションプログラムにおいて、積極的に配列を使ったりする方法は、処理効率は良くなるがLinked Dataが持つべき柔軟性を失わせる。

2.3. Linked Dataの表現

この局所性の問題は、構造的に柔軟なデータをアドレス付けされた空間に物理的に配置することによって生じる。しかし、Linked Dataは、そのリンクを手繰ってアクセスできる限り各ポインタは記憶空間上に一意に表現される必要はない。各セルが独立のアドレスを持っていても、直接のポインタを持っていなければ、リンクを手繰ることなしに直接アクセスすることはできない。別のポインタであれば、それぞれが正しく手繰れる限り、その記憶空間上の表現は同じであっても構わない。

我々は、この構造上の性質を活かしてLinked Dataを記憶空間上に表現し、そのリンクの情報によって記憶管理できる記憶構成方式、SOSOを考案した。

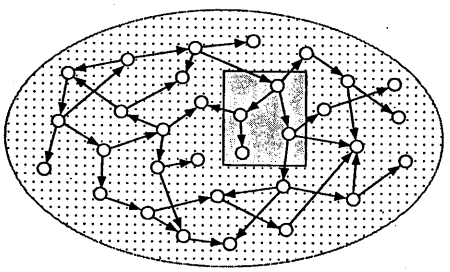


図3.1 Linked Dataの空間

3. 構造に基づく記憶空間の構成(SOSO)

3.1. Linked Dataの空間(図3.1)

Linked Dataの空間は、各ノードの位置が物理的に定められている必要はない。各ノードからリンクしている他のノードの位置が得られれば十分である。記憶空間は従って、アドレス付けされた固定領域である必要はなく、開領域を構成できる。

データを計算機システム上で扱うとき、データの演算機構上には論理的に近いデータを持ち、残りのデータは周辺記憶に表現するのが効率が良い。演算機構上のデータは、全体の記憶空間上でデータのリンクに従って移動する閉領域によって得ることができる。

3.2. Linked Dataのための記憶空間の構成

Linked Dataを主記憶と二次記憶からなる階層的記憶構成上で効率良く表現するため、SOSOでは、演算機構のための主記憶領域の外部においては、Linked Dataをサブグラフに分けて管理する(図3.2)。このサブグラフをストラクチャと呼ぶ。ストラクチャは、二次記憶アクセスに適した大きさを持つ論理的な局所性を持ったデータの集合である。

二次記憶ストラクチャのデータが演算機構において必要となったとき、そのストラクチャを主記憶に移動する。これをストラクチャ・インと呼ぶ。ストラクチャ・インするための空領域が主記憶上に得られないときは、主記憶上で別のストラクチャを抽出して二次記憶に移す。この動作をストラクチャ・アウトと呼ぶ。ストラクチャ・イン/アウトの際、データの表現される空間が変わるので、ストラクチャの中のポインタ表現は変換する必要がある。

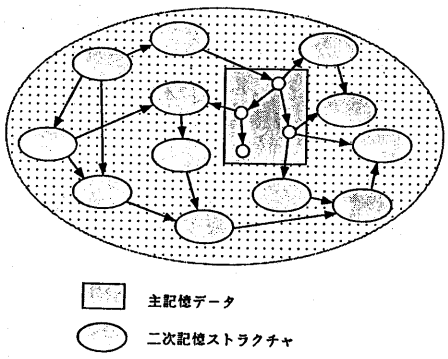


図3.2 Linked Dataの記憶空間の構成

ストラクチャ・アウトするデータは、今後アクセスされる可能性の少ないものが望まれる。Linked Dataの計算の過程において、処理のプロセスの単位を考えたとき、特に独立なプロセス間でデータを共有するような処理を記述しない限り、プロセス単位でのデータの局所性を期待できる。従って、そのプロセスの実行環境のうち、制御及びアクセス上遠いデータを選べば、効率良くストラクチャを扱うことができる。

3.3. SOSOの特徴

SOSOは以下の特徴を持つ。

- 論理的な局所性を持ったデータの単位で二次記憶を管理するため、ページング方式などの従来方式に比べて二次記憶のアクセス頻度の減少が期待できる。
- 全体の記憶空間は開領域であり、その空間上の主記憶のための閉領域もその大きさを元の空間とは独立に設定できる。従って、主記憶空間も二次記憶空間もそれぞれ独立に拡張可能である。
- 主記憶上のポインタは主記憶空間において、二次記憶ストラクチャ上のポインタはストラクチャ空間において表現する。従って、ポインタを仮想空間内で一意的に表現する従来の方式に比べて空間効率が良い。主記憶・二次記憶間、二次記憶ストラクチャ間の参照は特別のポインタで表現するが、そのポインタ数は全体のポインタ数に比べて少ないと考えられる。

・二次記憶ストラクチャ上のデータは、論理的な局所性を持つことが期待できる。ストラクチャ単位にGCを実行すれば、GCは二次記憶をアクセスせずに実行できる。

・ノード、すなわちデータセルを格納する位置の自由度が大きいので、可変容量セルが扱いやすい。

SOSOは反面、次のような性質を持つ。

・我々は逆ポインタのないLinked Dataの表現を考えているので、ストラクチャ・アウトの際、そのデータに主記憶上の他のデータセルからの参照のあったとき、その被参照データセルを間接的な二次記憶へのポインタとして主記憶上に残さなければならない。その被参照を認識するための多重参照マークまたは参照カウンタが必要である。

・主記憶と二次記憶ストラクチャとで空間の要素の表現が異なるので、ストラクチャ・イン/アウトの際、ポインタ表現の変換の手続きが要る。

・論理的局所性の大きいストラクチャを抽出するためのアルゴリズムを工夫しなければならない。

SOSOによって二次記憶アクセスの頻度が少なくなれば、後者の性質があっても、システム全体の処理効率は向上すると考えられる。

4. SOSOにおける記憶管理

4.1. データ表現(図4.1)

主記憶上のポインタは、主記憶空間のアドレスで表現する。主記憶上で二次記憶のデータは「間接ポインタ」によって参照する。二次記憶ストラクチャ上のポインタは、そのストラクチャ内のアドレスで表現する。

二次記憶ストラクチャ間、及び、二次記憶から主記憶に対する参照、被参照は、ストラクチャ・イン/アウトの際の二次記憶アクセスを少なくするため、以下の参照表で管理する。

1) 外部参照表

ストラクチャ毎に持ち、二次記憶ストラクチャデータの外部に対する参照、被参照を管理する。

2) 逆参照表

主記憶データの二次記憶に対する被参照を管理する。全体でひとつの逆参照表を持つ。

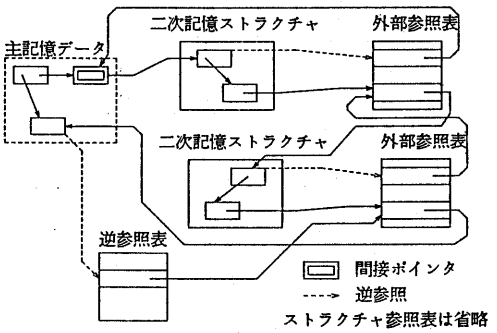


図4.1 SOSOにおけるデータ表現

3) ストラクチャ参照表

全体でひとつ持ち、外部参照表を管理する。外部参照表のコンパクション、および、二次記憶ストラクチャの参照カウントに使用する。

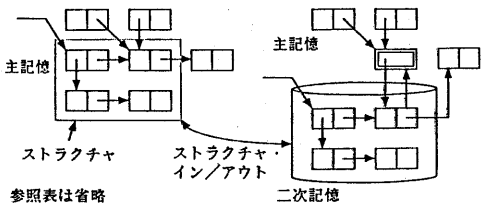


図4.2 SOSOにおける記憶管理

4.2. ストラクチャ・インの処理(図4.2)

主記憶上で間接ポインタをアクセスしたとき、それが参照する二次記憶ストラクチャを主記憶に転送する。その際、二次記憶上のストラクチャ表現を主記憶上のデータ表現へ変換する。この変換は、二次記憶制御部などで効率良く実行できる。主記憶上のデータセルは、フリーセルの集合から供給する。

転送したストラクチャ内に他の二次記憶ストラクチャへのポインタがあった場合、それは間接ポインタによって参照する。

ストラクチャ・インするデータが間接ポインタから参照されていたとき、その間接ポインタをなくし、直接参照に変換する。他の二次記憶ストラクチャから参照されていたとき、外部参照表上の被参照によって、その参照元を主記憶データへの参照に書き換える。

4.3. ストラクチャ・アウトの処理(図4.2)

ストラクチャ・アウトする主記憶上のデータは、ストラクチャ・インと同様、二次記憶上のストラクチャ表現に変換し、転送する。ストラクチャ・アウトで不要となった主記憶上のデータセルは、フリーセルとして扱う。

ストラクチャに主記憶上の他のデータセルから参照のあったとき、その被参照データセルを二次記憶データへの間接ポインタとして残す。二次記憶データから参照のあったときは、逆参照表によって、その外部参照表を書き換える。

ストラクチャ・アウトは、GCによっても充分な量のフリーセルが得られないときに起動する。

4.4. ストラクチャの抽出

Linked Dataは処理される過程において、通常図4.3のような処理環境を構成する。図の各ノードは、プロセスやタスクなどの処理単位におけるアクセスや制御のためのデータ環境である。システムの処理の形態に応じて、制御の遠い処理単位を選択することは可能であり、そこでのデータ環境を他の処理単位からの被参照が少なくなるように抽出することができる。

ストラクチャ・アウトするデータをどの様に抽出するかの詳細は、処理系のデータの扱い方による。例えば、シャローバインディングによる処理系であれば、制御スタックの深いところでの、アクセスや制御の退避環境を対象ストラクチャとすることができる。

5. GC

SOSOにおけるGCは、主記憶上ではデータセル単位に、二次記憶上ではストラクチャ単位に処理する。

主記憶上では、そのデータセルに多重参照マークまたは参照カウンタが必要であるが、どちらを使用するかは処理系の作り方と扱うデータによる。いずれの場合も主記憶上でのみ管理すれば良いので、記憶制御部において効率の良い処理をすることが可能である。

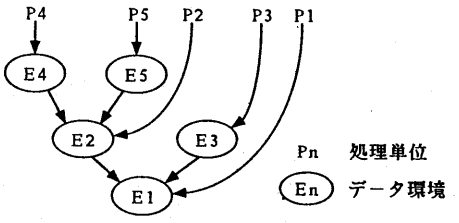


図4.3 Linked Dataの処理環境

二次記憶ストラクチャのGCは、ストラクチャ参照表上の参照カウントを用いて処理する。参照表は記憶制御部で管理するので、実時間GCにおいても、一括GCにおいても、二次記憶を実際にアクセスする必要がなく、GCの高速実行が実現できる。外部参照表のコンパクションは、ストラクチャ参照表を用いて処理する。

GCは、簡単なアルゴリズムで効率良く処理できる。

5.1. 実時間GC

インタラクティブなシステム使用環境を実現するには、実行単位の小さいGCが必要である。SOSOでは、主記憶上の小さな物理領域をsweepして、参照マークあるいは参照カウントによって主記憶上のデータセルのガーベージを回収することができる。データセルが比較的静的で大きなものであれば、他に例えば、ZCT(Zero Count Table)を使った遅延型GC[10]をsweep方式の代替手段として使用できる。

主記憶からの参照のなくなった二次記憶ストラクチャとその外部参照表は、ガーベージとして回収する。

5.2. 一括GC

実時間GCでは、多重参照マークによる主記憶セル、参照カウントのオーバフローした主記憶セルと二次記憶ストラクチャ、および、環状リストはそのガーベージを回収できない。定期的に、主記憶と二次記憶ストラクチャの参照をマーク付けし、主記憶とストラクチャ参照表をsweepすることによって、これらのガーベージを回収する。

6. 評価

6.1. 空間効率

仮想記憶とSOSOのそれぞれにおける記憶管理の空間効率について評価する。

仮想記憶とSOSOのそれぞれでポインタ以外のデータ量は同じとし、それは考えないことにする。総数2^k個のポインタを扱っているとする。 SOSOにおける主記憶の大きさを2^n、二次記憶ストラクチャの大きさを2^m、二次記憶ストラクチャに対する外部参照表の大きさの比の平均値をsとすると、仮想記憶とSOSOのポインタの空間使用量 V_v, V_s はそれぞれ、

$$V_v=k2^k$$
$$V_s=n2^n+m(2^k-2^n)+ks(2^k-2^n)$$

となる。ただし、ストラクチャ参照表と外部参照表は無視できるとする。また、参照カウントは使用していないとする。

ここで、$k=n+r$ $(r>0)$とすると、

$$V_v-V_s=\left\{(k-m-ks)2^r-(n-m-ks)\right\}2^n$$

$s<\frac{k-m}{k}$であれば、$V_s<V_v$は十分成り立つ。

$\because$ $s\geq\frac{n-m}{k}$ のときは明らか

$s<\frac{n-m}{k}$ のとき $\frac{k-m-ks}{n-m-ks}2^r>1$

実際的な値として、$k=32$, $m=10$とすると、$s<\frac{22}{32}$ であれば良い。二次記憶ストラクチャのデータセルの内、およそ $\frac{2}{3}$ に参照と被参照があっても、空間効率の良いことになる。理想状態として、$s\approx0$とすると、

$$\frac{V_s}{V_v}\approx\frac{m}{k}$$

が得られる。

6.2. 動作効率

仮想記憶とSOSOのそれぞれにおける記憶管理の動作効率について評価する。

同じ問題を扱ったときの仮想空間と SOSO の主記憶・二次記憶間の総転送セル数をそれぞれ、 N_v 、 N_s とする。ページテーブルの書き換えなど仮想化に必要なセルあたりの平均処理時間を t_v 、ポインタの張り換えなど SOSO に必要なセルあたりの平均処理時間を t_s とする。主記憶・二次記憶間のセルあたりの平均転送時間を t_c とすれば、仮想記憶とSOSOの記憶管理に要する時間T_v, T_sはそれぞれ、

$$T_v=N_v(t_c+t_v)$$
$$T_s=N_s(t_c+t_s)$$

となる。ここで、$N_v=\alpha N_s$, $t_s=\beta t_v$ とすると、

$$T_v-T_s=(\alpha-1)N_st_c+\frac{\alpha-\beta}{\beta N_st_s}$$

$T_s<T_v$であるためには、

$$(\alpha-1)N_st_c>\frac{\beta-\alpha}{\beta N_st_s}$$

$\alpha\geq\beta$ならばこの式は成立する。$\alpha<\beta$のときは、

$$\frac{t_s}{t_c}<\frac{\beta(\alpha-1)}{\beta-\alpha}$$

我々は、主記憶・二次記憶間の転送セル数を評価するための簡単なシミュレーションを行なった[1)]。このシミュレーションでは、GCは SOSOの主記憶においてのみ行なったが、そのときのαの値はおおよそ、2ないし4である。βはアルゴリズム上で評価すると、約4である。 $\alpha=2$, $\beta=4$とすると、

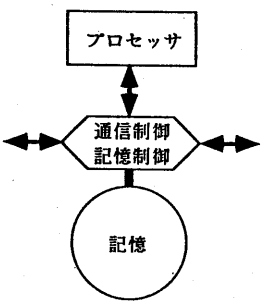
$$\frac{t_s}{t_c}<2$$

この式は、ストラクチャ・イン/アウトの際の手続きが、ストラクチャデータの転送の2倍の時間を要しても良いことを意味するが、十分満足できる条件である。さらに、このシミュレーションでは、仮想記憶における主記憶と二次記憶とにわたるGCを実行していない。仮想記憶管理でコピー方式のGCを行なうとき、システムの負荷の大きい場合GCの処理は実際のデータ処理の200%くらい動作することがある。GCの処理も加味すれば、SOSOは動作効率にお

いても従来方式に比べて良い特性を持つことがわかる。

6.3. 実験システム

我々は実験記号処理システム Lilac (図6) を開発中である。Lilac (Largely Integrated Language Accelerator) は、記憶部を高機能化するための記憶制御部や並列処理のための通信制御部を持ち、記憶制御部において SOSO の処理を行なう。Lilac 上で SOSO による記憶構成と管理を検証する予定である。



プロセッサ
- 基本命令サイクル: 125ns
- データ幅: 32bits
- マイクロプログラム制御
- リスト処理向きアーキテクチャ
- 多方向分岐
- 診断・統計機能
- 仮想記憶化WCS

通信・記憶制御
- プロセッサと同様の基本性能
- 通信制御間パケット通信機能

記憶
- 64bit,8Mwords実装可
- リファレンスカウント
- タグ
- 排他制御

図6.1 実験記号処理システム

7. まとめ

Linked Dataをその構造に従って記憶空間上に表現し、効率の良い記憶管理を実現する記憶構成方式、SOSOを報告した。この方式では、論理的局所性を持つデータのストラクチャに従って記憶管理するため、主記憶と二次記憶からなる階層的記憶構成上でも、データ構造の可変性という柔軟性を保った上で、効率良く記憶管理できる。GCは、ストラクチャ単位に処理するため、二次記憶をアクセスすることなく実行できる。

SOSOでは、リンクのためのポインタは局所的な空間で表現されているのでデータ幅が短く、記憶の空間効率が良い。シミュレーションによって、 SOSO上での記憶管理は動作効率の良い性質を持つことを確かめた。また、開発中の実験記号処理システムにおいて、SOSOを具現化する予定である。

SOSOは、既存の多くのシステムにおいても記憶制御を変更することによって組み込むことが可能である。また、記憶の物理空間を分けることなどにより、従来方式の記憶管理との併用も可能である。

謝辞

研究を遂行するにあたり、日頃からご指導をいただいている大阪大学安井先生、京都大学柴山先生、青山学院大学井田先生に感謝致します。本研究の機会をくださった当社総合研究所宮岡所長、情報通信研究所松田所長に謝意を表します。

【参考文献】

1) 前川 他: Poniter-Linked Dataにおける仮想記憶管理の一手法, 情報処理学会研究会資料, SYM50-1(1989).
2) 前川 他: Linked Data Structuresの記憶管理, 情報処理学会第39回全国大会 1Q-4, pp.1279-1280 (1989).
3) Deitel, H. M.: An Introduction to Operating Systems, Addison-Wesley pp.179-243(1984).
4) Andre, D. L.: Paging in Lisp Programs, The Faculty of Graduate School of The University of Meryland (1986).
5) Cohen, J.: Garbage Collection of Linked Data Structures, Computing Surveys, Vol.13, No.3, pp.341-367(1981).
6) Moon, D. A.: Garbage Collection in a Large Lisp System, Conference Record of the 1984 ACM Symposium on Lisp and Functional Programming, pp.235-246(1984).
7) Ungar, D.: Generation Scavenging: A Non-desruptive High Performance Storage Reclamation Algorithm, Proceedings of Software Engineering Symposium on Practical Software Development Environments, pp.157-167(1984).
8) Lieverman, H. and Hewitt, C.: A Real-Time Garbage Collection Based on the Lifetimes of Objects, Comm. of the ACM, Vol.26, No.6, pp.419-429(1983).
9) 近山隆, 木村康則: ポインタタグ(MRB)による多重参照管理方式, 情報処理学会第35回全国大会 2Q-5, pp.707-708 (1987).
10) Deutch, L. P. and Bobrow, D. G.: An Efficient, Incremental, Automatic Garbage Collector, Comm. of the ACM, Vol.19, No.9, pp.522-526(1976).