遠隔メモリを用いた大容量仮想メモリ DLM における メモリ管理機構の導入

斉藤 和広† 緑川 博子† 甲斐 宗徳†

†成蹊大学工学研究科情報処理専攻 〒180-8633 東京都武蔵野市吉祥寺北町 3-3-1

E-mail: †dm083406@cc.seikei.ac.jp, {midori, kai}@st.seikei.ac.jp

あらまし 筆者らはローカル物理メモリサイズに制限されず，クラスタの各ノードの遠隔メモリを集めて仮想的に大容量メモリとする分散大容量メモリシステム DLM を構築, 評価してきた．従来の DLM では遠隔メモリ利用のための動的メモリ割り当て機能のみを提供し，メモリ解放と再利用機能は備えていなかった．大容量メモリを必要とするアプリケーションには，始めに必要なデータ領域を確保して最後までそれを用いるものもあるが，実行中に動的割り当てと解放を繰り返し行うものもある．このため, DLM においてもメモリ解放と再利用を行う機構の導入が求められていた．今回，メモリ解放機能と効率的なメモリ利用を実現するための最小限のメモリ管理機構を設計し，DLM へ導入した．本稿ではメモリ管理機構の概要について述べ，Red-Black Tree へのランダムなノード挿入削除を行うマイクロベンチマークと、実応用である遺伝子情報処理におけるクラスタリング処理におけるメモリ割付・解放関数の初期性能評価を行ったので報告する．

キーワード クラスタコンピューティング， 大容量メモリ，遠隔メモリ，分散メモリ，メモリ管理

The Dynamic Memory Management for Distributed Large Memory System : DLM

Kazuhiro SAITO† Hiroko MIDORIKAWA† and Munenori KAI†

† Graduate School of Engineering, Seikei University,

3-3-1, Kichijoujikita-machi, Musashino-shi, Tokyo 180-8633, Japan

E-mail: † dm083406@cc.seikei.ac.jp, {midori, kai}@st.seikei.ac.jp

Abstract Authors designed and evaluated Distributed Large Memory System: DLM, which gives us very large virtual memory beyond the size of local memory by using remote memory distributed over cluster nodes. Originally, the DLM only supports a function for dynamic memory allocation, but it is required to be equipped with a memory release function for the applications that reuse memory resources in their runtime. We incorporate a simple memory management mechanism into the DLM to support memory release function for efficient memory utilization. In this report, we describe the design and mechanism of the memory management and show an initial performance evaluation for a red-black tree as a micro benchmark and a cluster3, a real bioinformatics application.

Keyword Cluster Computing, Large Memory, Remote Memory, Distributed Memory, Memory Management

1. はじめに

筆者らはローカルの物理メモリサイズに制限されることなく，クラスタの各ノードの遠隔メモリを利用し仮想的に大容量のメモリ空間を提供するシステム，DLM(Distributed Large Memory)を構築, 評価してきた[1]-[4].

従来の DLM には，大容量仮想メモリ利用の方法として遠隔メモリ動的割り当てと遠隔スワップの二つの機能を提供していた．物理計算や遺伝子分析等の大容量データを用いるアプリケーションでは，利用する領域を計算の始めに確保しこれを最後まで利用する形式を取っているものがあるが，実行中に動的割り当てと解放を繰り返し行うものもあるため，DLM にもメモリ資源再利用のための遠隔メモリ解放機能とそのためのメモリ管理機構を試作導入した．

これまで DLM は，カーネルの変更を最小限にしたユーザレベルソフトウェアとして設計･評価してきた．遠隔メモリアクセス用デバイスを用いた OS カーネルスワップによる方式と比べ，ユーザレベルソフトウェアを用いた DLM がより高速かつ高安定性が得られることを示した[1]-[3]．今回新しいメモリ管理機構もユーザレベルソフトウェアとして構築した．

本稿では，第２節で DLM のシステム構成とユーザインタフェースを，第３節で新しく導入した DLM メモリ管理機構を述べ，第４節でその動作確認と初期性能評価を報告する．

2. DLM

現在 DLM には２つのバージョンがあり，DLM のコンパイラを用いることで遠隔メモリサーバ起動をユーザから隠蔽した DLM-S(DLM-Single Client)[1]-[3]と，明示的に常駐のサーバを立ち上げることでマルチクライアント型にした DLM-M(DLM-Multi Client)[3]を構築・評価してきた．本稿での実装・評価は DLM-M で行ったので，このシステム構成とインタフェースを述べる．

2.1. DLM-M のシステム構成

DLM はクラスタ環境における複数台のコンピュータを，あたかも一台のマシン上のメモリ資源であるかのように利用する．そのため，**図１**のようにクラスタノードを計算ノードとメモリサーバの二つに分類する．なお，図１はマルチクライアントバージョンである DLM-M の図である[3].

計算ノードはユーザが実際にプログラムを実行するノードで，ユーザプログラムを処理する計算スレッドとメモリサーバとの通信を行う通信スレッドからなる．メモリサーバは計算ノードからのリクエストを受け，メモリを提供するノードである．DLM でのメモリ管理は DLM ページ単位で行っている．DLM ページサイズは OS のページサイズの倍数であればシステム構築時に自由に設定することができる． DLM では DLM ページ管理表によりメモリを管理しており，このエントリを操作することで動的割り当てを可能としている．なお，計算ノードは全ノードの利用 DLM ページを管理しているが，メモリサーバは自身が保持する DLM ページのみを管理している．また，ノード間の通信には TCP/IP を用い，基本的に DLM ページ単位でデータの送受信を行う．

計算ノードで遠隔メモリが必要になったとき，DLM ページ単位での遠隔スワップによってメモリサーバが保持しているデータを利用する．

2.2. ユーザインタフェース

DLM-M では DLM-S と異なり，ライブラリ関数のみを提供し，C 言語で利用することができる．**図２は** DLM 仮想メモリの利用例である．プログラムの流れは，dlm_startup()による DLM の初期設定から始まる．その後，dlm_alloc()により DLM の仮想メモリを利用したい変数に動的に割り当て，dlm_free()を用いることで指定変数を解放することができる．dlm_shutdown()は DLM のサービスを終了するときに用いる．実行時には，**図３**のような DLM で利用する計算ノードとメモリサーバのホスト名と，それぞれで利用可能なメモリ量を記述するホスト指定ファイルをオプション-f で指定する．また-n で利用台数を指定することもできる． DLM-M では，実行前にメモリサーバとするノード上で dlmm_server を起動しておく必要がある[4].

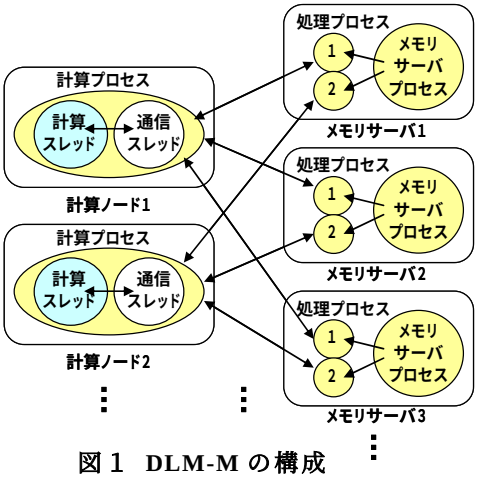

図１ DLM-M の構成

```
#include <dlmm.h>
#define N 1<<30 /* 1G (*4B=4GB) */
int main(int argc,char* argv[]){
  int *a;
  dlm_startup(&argc, &argv); /* DLM-M の初期化 */
  a=(int*)dlm_alloc(sizeof(int)*N); /* 動的割り当て */
  ・・・ /* 計算 */
  dlm_free(a); /* 解放 */
  ・・・
  dlm_shutdown(); /* DLM-M の終了 */
  return 0;
}
```

図２ DLM-M の利用例

```
calhost 2048 // 2GB 計算ノード
memhost1 2048 // 2GB メモリサーバ１
memhost2 4096 // 4GB メモリサーバ２
  :
```

図３ ホスト指定ファイル

3. 遠隔メモリ管理機構

DLM の仮想メモリ解放機能のための管理機構と，これを利用した dlm_alloc 関数と，更に dlm_free 関数について述べる． DLM はユーザレベルソフトウェアでの実装を行っているため，カーネルのページフレーム管理機構とは別の最小限のデータ構造をユーザレベルソフトウェアで設計する．

3.1. 計算ノードのデータ構造

従来の DLM では大容量仮想メモリの動的割り当てのみを提供してきたので, DLM ページ単位でのメモリ管理だけで対応することができた．しかしメモリ解放機能を実装するには dlm_alloc による割り当て単位での管理が必要となる．また解放された資源のフラグメンテーションを回避し効率的な再利用するためには，空き領域の管理が必要となる．

dlm_alloc による割り当て単位での管理のためのデータ構造は**図 4** のような単方向リスト構造になっていて，これを変数リストと呼ぶ．これは DLM 仮想メモリ空間上で先頭からアドレス順に連結される．変数リストにおける各ノード（変数ノード）には**図 5** のように変数のメモリ情報等を持つ．また探索の高速化のため，DLM ページ管理表エントリからも各 DLM ページ内の先頭となる変数ノードへリンクしている．図 4 の DLM 仮想メモリ空間からの矢印, 図 5 の DLM ページ管理表エントリからの矢印がその連結を示している．

空き領域管理には，**図 6** のようにサイズ別に双方向連結リストを用いていて，これを隙間リストと呼ぶ．サイズ別隙間管理表の 2^k のエントリには，$2^{k-1}+1$〜2^k B の範囲のサイズの隙間がサイズ順に連結される．管理される空き領域は以下の二種類がある．

- DLM 仮想メモリ空間の先頭と変数の間の隙間
- 変数と変数の隙間

隙間リストにおける各ノード（隙間ノード）にはその隙間情報等があり，更に変数削除時に隙間を生成するときの前後の隙間との結合を容易にするために，**図 7** のように変数ノードに DLM 仮想メモリ上ですぐ前にある隙間ノードを示すポインタ preg が存在する．

3.2. メモリサーバのデータ構造

計算ノードとメモリサーバは DLM ページ単位でメモリのやり取りをするため，メモリサーバと計算ノード間でアドレスを管理する必要がない．メモリサーバは，メモリ割り当て要求を受け取ると必要な数のページを格納するためのスロットと呼ばれる領域を用意してページを保持する．メモリ解放要求を受けると，そのページで利用していたスロットは munmap せずに，再利用できるように図 8 のような空スロットリストに登録する．次に割り当て要求が来たときには，その空きスロットリストに利用可能なスロットがあれば，それを利用して DLM ページを格納する．この場合はメモリサーバは DLM ページ管理表を更新するだけで処理を完了する．

3.3. DLM メモリ動的割り当て

DLM ライブラリ関数の dlm_alloc(size_t size)を用いることで, DLM 仮想メモリへの動的割り当てをすることができる．従来はこの関数が呼ばれた時点での最終

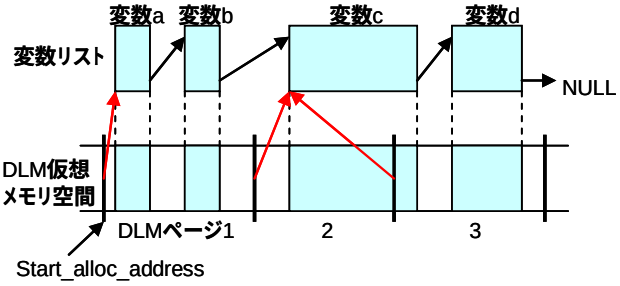

図 4　変数リストと DLM 仮想メモリ空間の関係

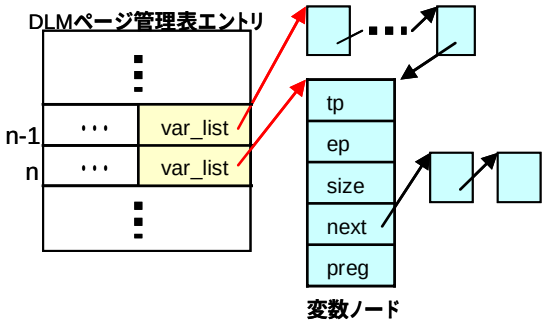

図 5　DLM ページ管理表から変数リストへのリンク

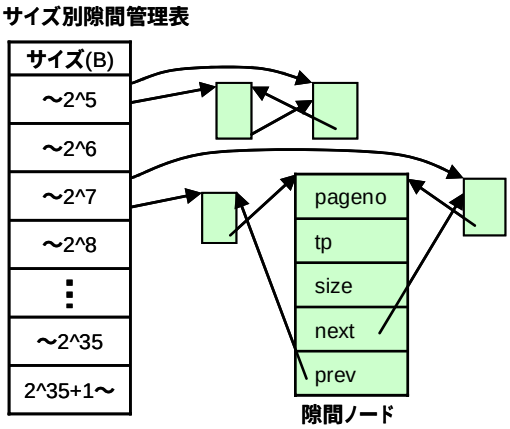

図 6　計算ノードの隙間リスト

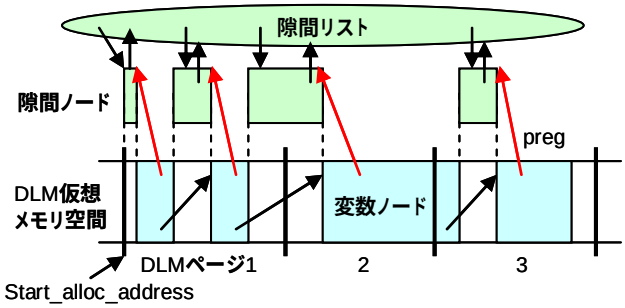

図 7　変数ノードから隙間ノードへのリンク

割り当てアドレスから順に割り当てるだけであったが，変数管理と隙間管理が追加されたことにより，大幅に構造が変わった．

まず，隙間が一つも存在しないか，引数 size 以上の隙間が存在しない場合を考える．この場合は従来の

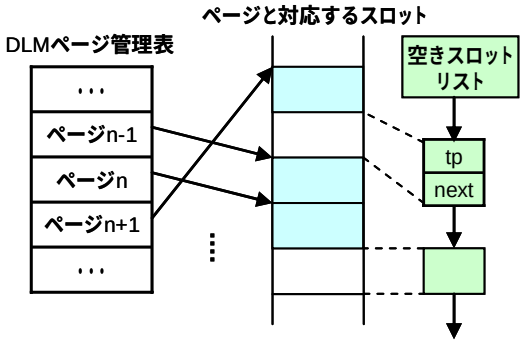

図８　メモリサーバの DLM ページ管理とスロット管理

dlm_alloc()と同様に必要な DLM ページ数分を適切なノードに割り当てる．size 分全てを割り当てたら DLM ページ表を更新し，変数リストに登録する．

次に利用可能な隙間が存在する場合を考える．始めに size より大きい最小サイズの隙間ノードを探索し，発見した隙間ノードの先頭アドレスから順に DLM ページ単位で割り当てていく．このとき以下のパターンが考えられる．

① その DLM ページにおいて既にどこかのノードに割り当てられている場合，その DLM ページ管理表を更新するのみ．
② その DLM ページがどこにも割り当てられていなくて計算ノードに空きがある場合，その分を mmap し DLM ページ管理表を更新する．
③ その DLM ページが割り当てられてなく計算ノードに空きがない場合，空きのあるメモリサーバに割り当て要求を出し，すぐに DLM ページ管理表を更新する．メモリサーバは要求を受け取ると，空きスロットが存在すれば空きスロットリストから，なければ mmap でスロットを作り指定 DLM ページを保存し，DLM ページ管理表を更新する．

これらを size 分完了したら，この変数情報を変数リストに挿入する．最後に隙間ノードを削除し，この隙間ノードを全て使い切らなかった場合は残った隙間を新しく隙間ノードとして登録する．そしてこのとき作成した変数ノードの次の変数ノードの preg をこの新しい隙間ノードへのポインタとして登録する．また使い切った場合にはこの preg を NULL にする．

3.4. DLM メモリ解放

DLM ライブラリ関数の dlm_free(void* p)を用いることで，DLM 仮想メモリに割り当てたデータを解放し，再利用可能にすることができる．

この関数が呼ばれると，まず引数 p の変数情報を変数リストから取得する．次に，最初の DLM ページから順に DLM ページ単位で解放していく．このとき以下のパターンが考えられる

① その DLM ページにおいて他に変数が残っている場合，必要に応じてその DLM ページ管理表を更新するのみ．
② その DLM ページに他に変数がなく，計算ノードが保持していた場合，その分を munmap し DLM ページ管理表を更新する．
③ その DLM ページに他に変数がなく，メモリサーバが保持していた場合，メモリサーバに解放要求を出し DLM 管理表を更新する．メモリサーバはその要求を受け取ると指定 DLM ページの指定しているスロットを一つずつ空スロットリストに繋ぎ，DLM ページ管理表を更新する．

次に解放する変数の前後の変数ノードの情報から解放する変数の前後の隙間を取得し，その変数が開放されることで出来る隙間とそれらの隙間を結合する．結合後，後ろの変数の preg を更新する．またこのとき，DLM 仮想メモリ空間上で最後尾の変数だった場合はその前にある隙間ノードを削除するだけである．そして最後に解放した変数ノードを削除する．

4. 初期性能評価

今回導入した遠隔メモリ管理機構の初期性能を評価した．評価にはマイクロベンチマークとして STAMP[4]のライブラリを用い，また実際のアプリケーションに近い例として，遺伝子分析で用いられる Cluster3.0[5]を用いた．なお，評価は**表１**のマシンを用いて計算ノード１台とメモリサーバ１台で行い，DLM ページサイズは 128KB に設定した．

4.1. マイクロベンチマーク

STAMP[4]はマルチスレッドによるトランザクション処理のメモリの性能を調べるためのベンチマーク集である．今回は動作確認と初期評価のためにこの中のライブラリである Red-Black Tree を実装した rbtree.c の逐次版を用いて，以下のような簡易のトランザクション処理を行った（main.c）．

① 50 万個のランダム整数を挿入(insert)
② 1000 個のランダム整数を削除
③ 1000 個のランダム整数を挿入
④ ②と③を 100 回繰り返す(transaction)
⑤ 残っているノードを全削除(release)

このプログラムの特徴は，変数と隙間の数が非常に多くできることと，隙間のサイズが一定サイズに多く集中することである．このとき，乱数は種を固定したときの long 型の 0～1,000,000 で，全てのノードとランダム整数は malloc で動的に確保される．ノードのサイズは 48B なので，動的メモリの最大使用量は約 28MB

表１ 実行環境

ホスト	サーバマシン q1	クラスタマシン hp0
マシン	HP ML150 G3	HP ML150 G2 x 7 Nodes
CPU	Xeon E5310 1.6GHz QuadCore x 2CPU	Xeon 2.8GHz x 2CPU HyperThread
メモリ	8GB	1GB
L2キャッシュ	4MB/CPU	1MB/CPU
OS	Linux kernel2.6.23.17-88.fc7 x86_64	Linux kernel2.6.20-1.2320.fc5 x86_64
コンパイラ	gcc version 4.1.1 20070105	gcc version 4.1.1 20070105
NIC	NC7781 OnBoard Gigabit NIC	Broadcom 5721 PCI-Express Gigabit NIC
ネットワーク	1GbitEthernet	

となる．

次に，malloc と free をそれぞれ dlm_alloc と dlm_free に代えたもの(main_dlmm.c)を用意し，同様に上記の処理を行った．このとき，本応用でのメモリ使用量に対する計算ノードが保持するローカルメモリ量の割合(ローカル率)を 100%, 74%, 56%に変えて実行した．そしてこれらの実行時の①(insert)の時間，②～④(transaction)の時間，⑤(release)の時間をそれぞれ計測した．

表２がその結果である．alloc 列が dlm_alloc(malloc)関数の実行時間，free 列が dlm_free(free)関数の実行時間，count それぞれの実行回数，Total が Red-Black Tree 操作の時間も含めた各処理の合計時間となっている．なお insert 処理においての free は，挿入のために一度ノードを動的に割り当てるが，すでに値が存在する場合はそれを挿入せずにすぐ free するよう rbtree.c で実装されているからである．

このプログラムは，変数の数は全体で最大 401,033 個，DLM ページ１つの中では最大 2,730 個で，各 DLM ページ内の変数リスト長が非常に長くなる．変数探索ではその変数がある DLM ページの先頭の変数からリストを辿るため，一回の変数探索の平均メモリアクセス回数が 1365 回と非常に多くなる．また，隙間数は⑤の release 時に最大で約十万個にまで達した．更に多くの隙間のサイズが同じため，隙間管理表の１エントリに集中してしまう．すなわち，Red-Black Tree は alloc と free の繰り返し頻度が高く，変数・隙間リストの探索回数も多いため，メモリ管理上非常に過酷な例である．このような状況においては malloc と free に比べ dlm_alloc と dlm_free 共に非常に低い性能結果となった．しかし alloc time や free time においては例えローカル率が低くなろうとも，main_dlmm.c の 74%や 56%の Total 実行時間のような急激な速度低下は見られなかった．

次に、main_dlmm.c の実行時間に対する通信，メモリ・変数・隙間操作の時間の割合を dlm_alloc と

表２ main と main_dlmm のメモリ操作別実行結果

rbtree		local	alloc(s)	count	free(s)	count	Total(s)
insert	main		0.13235	500002	0.01979	106494	1.420
	main_dlmm	100%	5.77808		1.27921		8.490
		74%	6.53484		1.63273		494.349
		56%	7.38121		2.04381		1858.074
transaction	main		0.01463	47419	0.00937	39894	0.186
	main_dlmm	100%	0.70120		0.69541		1.824
		74%	1.34586		1.58248		770.246
		56%	1.44040		1.75564		1483.255
release	main		0	0	0.31215	401033	0.425
	main_dlmm	100%	0		90.82537		91.070
		74%	0		90.30602		91.088
		56%	0		99.96436		376.969

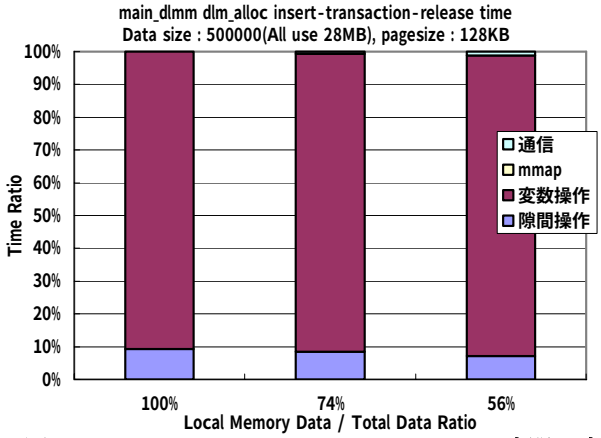

図９ main_dlmm における dlm_alloc の時間の割合

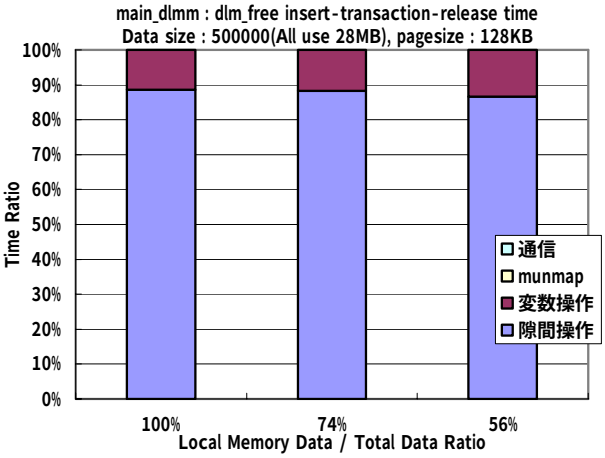

図１０ main_dlmm における dlm_free の時間の割合

dlm_free それぞれでグラフ化したものが**図９**と**図１０**である．図９から，dlm_alloc におけるオーバヘッドとなっているのが変数操作，図１０から dlm_free では隙間操作であることがわかった．これは，先に述べたような，変数・隙間リストの探索が大きな原因となっていることの裏づけとなったといえる．

4.2. Cluster 3.0

遺伝子分析のための様々なクラスタリング手法を提供するアプリケーションである Cluster3.0[5]の C 言語コマンドライン版を用いて性能評価を行った．本評

表３ Cluster3.0 の実行時間

Cluster 3.0		Time(s)				
	local	alloc	realloc	calloc	dlm free	Total
Original		0.017208	0.001193	0.000012	0.009185	9.637763
DLM version	100%	0.199455	0.112264	0.000022	0.148814	10.93558
	96%	0.204349	0.112475	0.000022	0.151385	13.99166
	77%	0.225566	0.112512	0.000022	0.16416	195.533
	58%	0.228844	0.112608	0.000022	0.16443	495.0805
	38%	0.237418	0.1123	0.000022	0.164922	906.9881
	19%	0.244479	0.11241	0.000022	0.1665	1201.194
count		24773	4937	1	24774	

価では，このアプリケーションのデフォルトの階層クラスタリングと，1.2MB の遺伝子情報入力ファイル demo.txt を用いた．なお，プログラム中に出てくるすべての動的割り当て・解放は dlm_alloc と dlm_free に変換した．

表３は実行時間と，各メモリ操作関数で要した時間とそれらの呼び出された回数，さらに全処理に要した時間を表にしたものである．このうち，Original が DLM 用に変換する前の元の Cluster3.0 のプログラムの実行時間で，DLM version は計算ノードのローカル率別に実行した．count はそれぞれのメモリ操作関数が呼ばれた回数である． Red-Black Tree での実行時と同様，ローカル率の割合が低い場合でも大きな性能低下がなかった．また全体の実行時間に対する比も 1%未満と少なかった．これは，利用された変数の数が 12433 個と Red-Black Tree に比べ非常に少ない，これは，一つの変数のサイズが大きい(～39,456B)ので DLM ページ内部の変数ノード数が少ないことと，最大で隙間が４つしかできなかったことに起因する．

図１１，図１２は図９と図１０同様に操作毎の時間の割合をグラフにしたものである．Red-Black Tree に比べ変数・隙間操作ともに比率は少ないが，やはりこれらがオーバヘッドとなっていることがわかった．

5. おわりに

本稿では，遠隔メモリを用いた大容量の DLM 仮想メモリ空間における動的メモリの管理機構を導入し，その初期評価を行った．現段階では Red-Black Tree のような非常に過酷に alloc と free を繰り返すプログラムにおいては未だ課題が残る結果となったが，一方 Cluster3.0 では全体の実行時間に対するメモリ操作時間の比が 1%以下と非常に少なく，DLM のメモリ管理機構が Red-Black Tree 実行時ほどの大きなオーバヘッドにはならないことが評価からわかった．今後の展望として，今回大きなオーバヘッドとなった変数・隙間管理のリスト構造について，データ構造の再設計も考慮にいれたチューニングを試みたいと考えている．

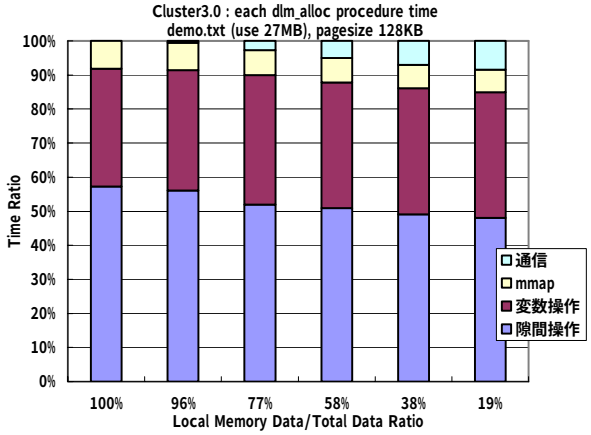

図１１ Cluster3.0 における dlm_alloc の時間の割合

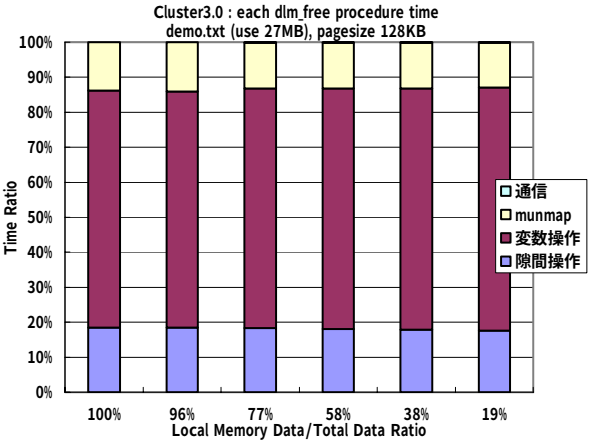

図１２ Cluster3.0 における dlm_free の時間の割合

参 考 文 献

[1] DLM の設計と DLM コンパイラの構築”, 電子情報通信学会研究報告 CPSY 研究会報告, 信学技法 Vol.102, P.29-34, No.398, Dec.2007

[2] 緑川, 黒川, 姫野, “遠隔メモリスワップのユーザレベルソフトウェア DLM と性能評価”,電子情報通信学会研究報告 CPSY 研究会報告, Aug.2008

[3] 緑川, 黒川, 姫野, “遠隔メモリを利用する分散大容量メモリシステム DLM の設計と 10Gb Ethernet における初期評価",情報処理学会論文誌：コンピューテイングシステム,Vol.1 No.3(ACS1), 2008-12

[4] 斉藤, 緑川, 甲斐, ”マルチクライアント向け分散型大容量メモリシステム DLM-M の設計と実装” ,情報科学技術フォーラム FIT2008, FIT 論文集, C-003, Sep.2008

[5] STAMP (Stanford Transactional Applications for Multi -Processing)
http://stamp.stanford.edu/

[6] Cluster 3.0
http://bonsai.ims.u-tokyo.ac.jp/~mdehoon/software/cluster/