データ工学特論（4）

情報基盤センター
天野　浩文

この講義に関するwebサイト:
http://isabelle.cc.kyushu-u.ac.jp/~amano/data-engineering/

前回のおさらい（1）

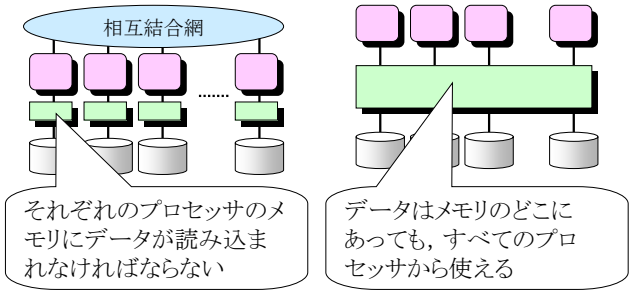
- 最適なストライピングユニットサイズの選択
  - 大きくするか，小さくするか，トレードオフの問題
  - 解析的に求めるのが難しく，今後の課題となっている
- RAIDの例
  - Thinking Machines 社 Scale Array
  - NCR社 6298
  - Hewlett-Packard研究所 TickerTAIP/DataMesh
  - UCB　RAID-II
  - Auspex NetServer
  - NetApp Multiprotocol Filer
  - SiliconGraphics Challenge RAID
  - Sun Microsystems StorEDGE

前回のおさらい（2）

- 並列計算機のアーキテクチャ　…その多様な分類法
  - Flynnの分類（1966）
    - SISD （これは並列ではない）
    - SIMD
    - MISD
    - MIMD
  - 共有メモリ型並列計算機と分散メモリ型並列計算機
  - 超並列計算機
  - SMPクラスタ

前回のおさらい（3）

- 分散メモリ型並列計算機では，それぞれのメモリにデータを読み込む必要がある
  - 共有メモリ型よりも難しくなる

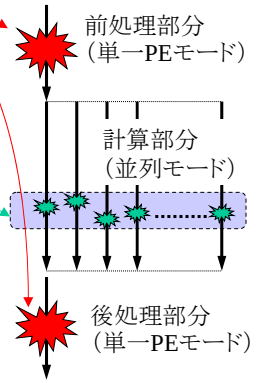


前回のおさらい（4）

- 分散メモリ型並列計算機
  - 多数の**PE（processing element）**と大容量の主記憶によって大量のデータを用いた大規模な計算を行うことができる
  - 複数のディスクを搭載し，それを並列に駆動できる
- 高度な科学技術計算では，計算の精度をあげようとすると必要なデータの量が急激に増大することが多い
  - そのようなプログラムは，計算時間を短縮するため，並列化されることが多い
  - 並列プログラムにディスク入出力を行わせる必要が出てくる

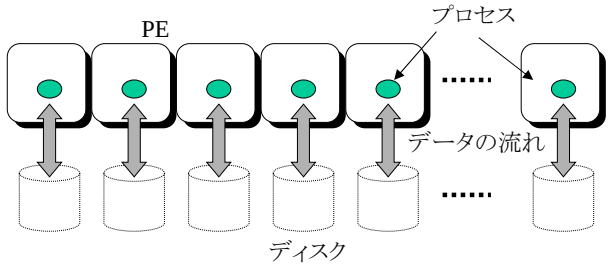
前回のおさらい（5）

- 並列プログラムのデータ入出力
  - 単一のPEが代表で入出力を行う ⇒大容量のデータの入出力には時間がかかる
  - 複数のPEが同時に入出力を行う ⇒プログラマが制御するのは面倒



並列入出力のイメージ

- 複数のPE上で動作している各プロセスが，自分の必要とするデータを並列に入出力
- あたかも，それぞれがローカルディスクにアクセスしているかのように（物理的にはそうでなくてもよい）



並列入出力のためのしくみ

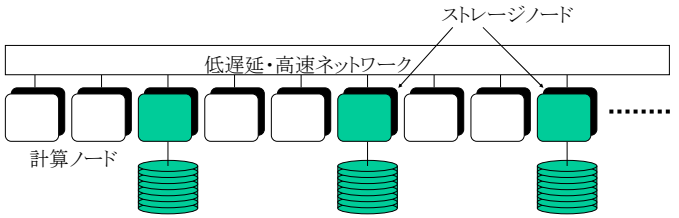
- **並列ファイルシステム**
  （どちらかというと）複数のディスクをまとめて1つのファイルシステムのイメージを提供
  - Vestaファイルシステム
  - Acaciaファイルシステム
- **並列入出力ライブラリ**
  （どちらかというと，ディスクはバラバラで，それを並列に使うイメージを提供）
  - Bordawekar らの並列入出力プリミティブ
  - Galbreath らの応用指向並列入出力パッケージ
- **並列入出力インタフェース**
  - MPIの集団型入出力インタフェース

Vestaファイルシステム

- 超並列計算機SPシリーズのための並列ファイルシステム（IBM ワトソン研究所）
  - 科学技術計算用
    科学技術計算におけるファイル入出力
    …大規模ファイルの読み書きが中心
- Vesta 並列ファイルシステムの設計方針
  - **並列性**
    入出力を行う応用プログラムからディスクの駆動プログラムまで，すべて並列に動作させる
  - **スケーラビリティ**
    超並列計算機のPE数が増大しても拡張可能に
  - **MIMDスタイル**
    同一ファイルに対して異なる関数を実行可能

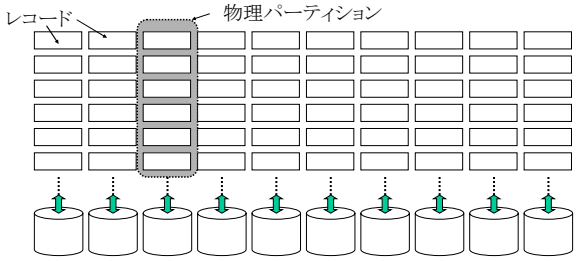
Vesta の想定する環境

- **計算ノード**と**ストレージノード**からなる超並列計算機
  - 計算ノード
    計算を実行．データの入出力はストレージノードに要求
  - ストレージノード
    計算ノードからの入出力要求を処理．それぞれがディスクアレイを保有．



物理分割（Physical Partitioning）ー（1）

- Vestaファイルの基本構造…2次元
  ファイル⇒**物理パーティション**の集まり
  物理パーティション⇒レコードの線形並び
- 物理パーティション数とレコードサイズはファイル作成時に決定
  …その後は変化しない

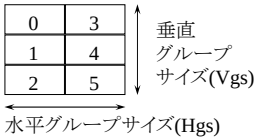


物理分割（Physical Partitioning）ー（2）

- レコードの**基本順序** (canonical ordering)
  1レコード単位で各物理パーティション上をストライピング
- 物理パーティションの意味
  1つのストレージノードにまとめて蓄えられるデータのまとまり
- ファイルの格納に使われるストレージノードとの対応
  - 実際のストレージノード数より小さいとき
    ⇒各物理パーティションを1ストレージノードに格納
  - 実際のストレージノード数より大きいとき
    ⇒全パーティションを全ストレージノードに均等分配

論理分割(Logical Partitioning)ー(1)

- Vesta のファイルの「見え方 (view)」を制御
  - テンプレート…レコードのレイアウトの最小単位



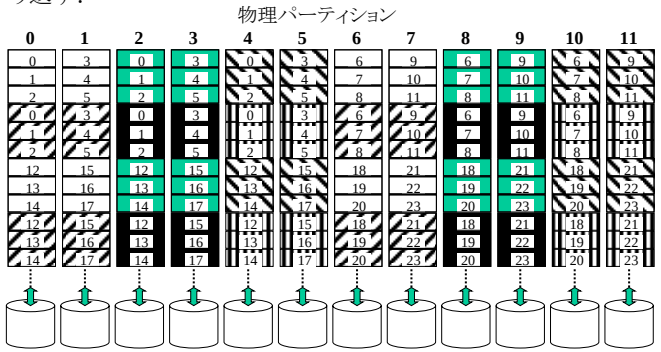
1カラムは同一物理パーティション内に割り付ける
隣接するカラムは「隣接」する物理パーティションに割り付ける

  - タイルパターン…これの繰り返しで物理パーティションをカバー

左のように設定すると，6系統の論理パーティションが作成される⇒6つのプロセスで利用

論理分割(Logical Partitioning)ー(2)

- 物理パーティション(下の例では12本)がカバーされるまでタイルパターンを繰り返す．レコード数が増大すると，これを下方向に繰り返す．



Vesta ファイルへのアクセス(1)

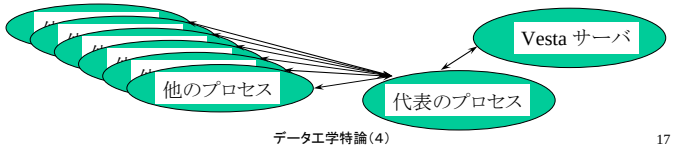
- 通常の UNIX では**open** システムコールで
  - ファイルの有無のチェック
  - パーミッションのチェック
  - ファイルシステムのテーブルにエントリを追加
  - プロセスのファイルディスクリプタテーブルにエントリを追加
- Vestaでは…UNIXの **open** に相当する操作を細分化
  - **`Vesta_create`** … Vesta ファイルの生成
  - **`Vesta_attach`** … 既存のVestaファイル情報の取得
  - **`Vesta_open`** … 取得した情報をもとにローカルにファイルをオープン

Vesta ファイルへのアクセス(2)

- **`Vesta_create`** … Vestaファイルの生成
  単一のプロセスが代表して呼び出す
  ⇒この呼び出し部分だけはどうしても並列にならない
  (Vesta サーバ側は並列に動作可能)
- パラメータ
  - その Vesta ファイルが使用する物理パーティション数
  - レコードサイズ(ストライピングユニットサイズ)

Vesta ファイルへのアクセス（3）

- **Vesta_attach** …Vesta ファイル情報の取得
  - 物理パーティション数と位置
  - レコードサイズ
- **Vesta_attach_share, Vesta_attach_accept**
  巨大ジョブの場合には間接通信も可能
  - 代表のプロセスが **Vesta_attach**を実行
  - 続いて **Vesta_attach_share** を実行
  - 他のプロセスは **Vesta_attach_accept** を実行



Vesta ファイルへのアクセス（4）

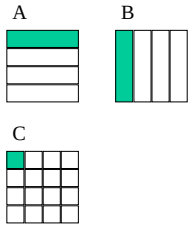
- **Vesta_open** …Vestaファイルを各プロセスがオープン
  - パラメータ
    - ファイル名
    - 水平・垂直グループサイズ
    - 水平・垂直インターリーブ
    - 論理パーティション番号
  - コールのたびに特定の論理パーティションに対するファイルディスクリプタが返る（各プロセスがそれぞれ独立のオフセットを持つ）
- **Vesta_open_share, Vesta_open_accept**
  論理パーティション内でのオフセットを共有できる

Vesta ファイルへのアクセス（5）

- **Vesta_read, Vesta_write**
  オープンした論理パーティションへのアクセス
  - フラグ
    複数の物理パーティションにまたがるアクセスの制御の「慎重さ」を制御
    - **CAUTIOUS**：複数のプロセスが同一パーティションを操作する際の原子性 atomicity と線形順序を保証
      （原子性…各プロセスのデータがまとめて入出力されるか，あるいはまったく入出力されないかのどちらかであるように保証されること）
    - **RECKLESS**：保証しない
- **Vesta_read_q, Vesta_write_q**
  非同期読み出し・書き込みのスタート

利用例

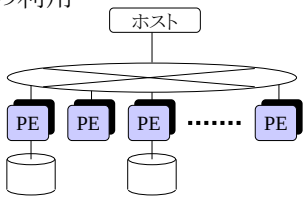
- 行列積 C=A×B




```
pdim=sqrt(PROC_NUM);
blk_size=MAT_SIZE/pdim;
float A[MAT_SIZE][blk_size],
      B[blk_size][MAT_SIZE],
      C[blk_size][blk_size];
Vgs=blk_size; Vi=pdim;
Hgs=MAT_SIZE; Hi=1;
fda=Vesta_open("A",Vgs,Vi,Hgs,Hi,me/pdim);
Vgs=MAT_SIZE; Vi=1;
Hgs=blk_size; Hi=pdim;
fdb=Vesta_open("B",Vgs,Vi,Hgs,Hi,me%pdim);
Vgs=blk_size; Vi=pdim;
Hgs=blk_size; Hi=pdim;
fdc=Vesta_open("C",Vgs,Vi,Hgs,Hi,me);
Vesta_read(fda,A,MAT_SIZE*blk_size);
Vesta_read(fdb,B,MAT_SIZE*blk_size);
for (i=0; i<blk_size; i++)
  for (j=0; j<blk_size; j++) {
    C[i][j]=0;
    for (k=0; k<MAT_SIZE; k++)
      C[i][j] += A[k][j] * B[j][k]
  }
Vesta_write(fdc,C,blk_size*blk_size);
```

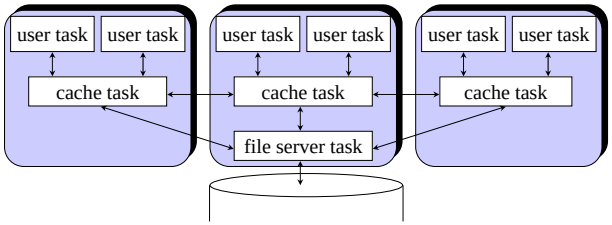
並列ファイルシステム Acacia

- 並列計算機 Fujitsu AP1000 用ファイルシステム
  - **ファイルアクセスモード**の指定
  - **ディレクトリアクセスモード**の指定
  - **ファイルレイアウト**の制御
  - **分散ファイルキャッシング**の利用
- AP1000
  …並列処理研究用計算機
  - 最大1024プロセッサ（ディスク装着時は512）
  - PEのサブセットにディスクを搭載することができる



実装の概要

- **キャッシュタスク**
  - 各PEでデータをキャッシュ
  - ユーザタスク，ファイルサーバタスクとの通信
- **ファイルサーバタスク**
  - ディスクのあるPEにのみ配置
  - ディスクへのアクセスを担当

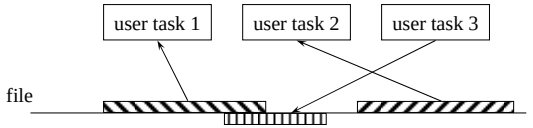


ファイルアクセスモード（1）

- 複数のPEが同一のファイルへのアクセスするときのアクセスのしかたを制御
- 複数のモードをサポート
  - **独立モード**
  - **複製モード**
  - **共有モード**
    - 無指定
    - 固定順，固定長
    - 任意順，固定長
    - 任意順，可変長

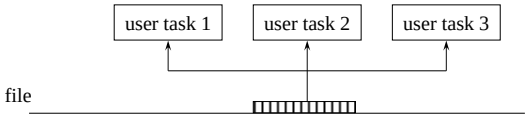
ファイルアクセスモード（2）

- **独立モード**
  - 各PEが独立のファイルポインタを持つ
    ⇔同一ファイル中の任意の位置に独立にアクセス
  - 各キャッシュタスクが独立にファイルサーバタスクと通信
  - ファイルの一貫性はファイルサーバタスクが管理
  - 分散ファイルキャッシング（後述）を用いても効率はあまり望めない



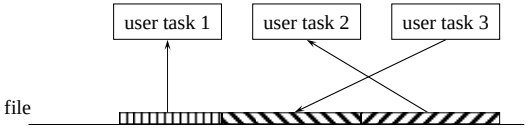
ファイルアクセスモード（3）

- **複製モード**
  - 各PEがファイルポインタを持つが，**値はすべて同じ**
  - 全PEが同じ位置で**同じデータをアクセス**
  - ディスクを持つファイルサーバが読み出し・書き込みを代行
  - 読み出しの場合，ファイルサーバが読んだデータを全PEのキャッシュタスクにブロードキャスト



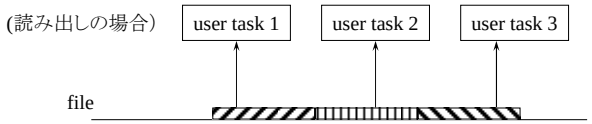
ファイルアクセスモード（4）

- **共有モード**
  - ファイルポインタが各PEで共有される（同時に使われるというよりも，使い回しされるというイメージ）
  - 各PEがアクセスするデータに重なりがなく，連続
- 共有モード(1)－無指定
  - ファイルアクセスが直列化されるため非効率



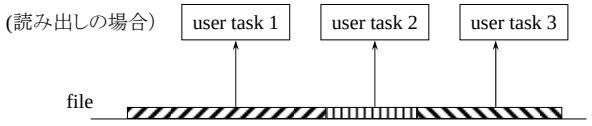
ファイルアクセスモード（5）

- 共有モード(2)－固定順・固定長
  - 各タスクからのアクセスが読み出し・書き込みのいずれかに統一される
  - 順序はPEの番号順，サイズが固定
  - 各タスクのアクセス位置を独立に計算できる



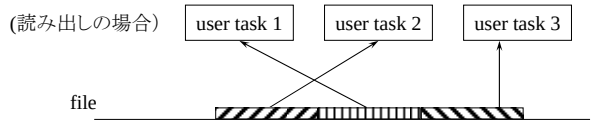
ファイルアクセスモード（6）

- 共有モード(3)－固定順・可変長
  - アクセスは読み・書きのいずれかに統一される
  - 順序はPEの番号順，サイズが可変
  - ファイル内の各レコードのサイズを**ファイルレイアウト**（後述）として別途登録しておけば，前のタスクのアクセス完了前に他のタスクのアクセス位置を計算できる（ただし，書き込み時にレコードサイズの変更はできない）



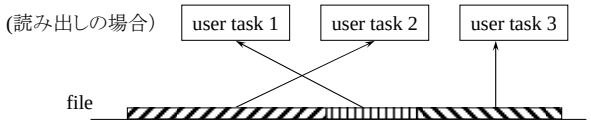
ファイルアクセスモード（7）

- 共有モード(4)－任意順・固定長
  - アクセスは読み出し・書き込みのいずれかに統一
  - 順序は任意（ファイルサーバが決定），サイズは固定
  - 前のタスクのアクセス完了前に他のタスクのアクセス位置を計算できる
  - ネットワークが有効利用できる順序をサーバが勝手に決めてよい



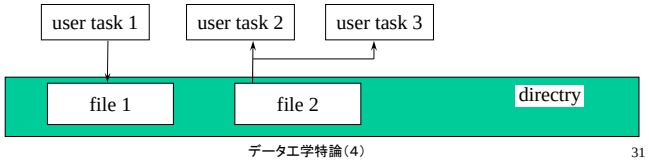
ファイルアクセスモード（8）

- 共有モード(5)－任意順・可変長
  - アクセスは読み出し・書き込みのいずれかに統一される
  - 順序は任意（ファイルサーバが決定），サイズも可変
  - ファイルレイアウトの利用による効率化
  - 順序付けによるネットワークの有効利用



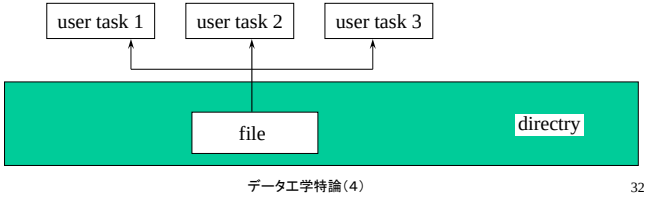
ディレクトリアクセスモード（1）

- 各PEが多数のファイルにアクセスする場合，ファイルアクセスよりもディレクトリアクセスが性能低下の原因となる可能性
- 複数プログラムモード（デフォルト）
  - 各ファイルのオープン処理は独立
  - 同一ファイルのオープンは直列化される
  - ディレクトリを持つファイルサーバがボトルネックになる可能性が大きい



ディレクトリアクセスモード（2）

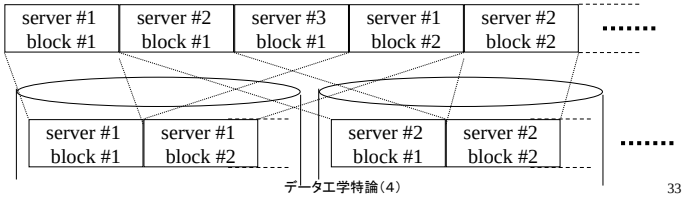
- 単一プログラムモード
  - 各PEが同じファイルのオープンを並行に行う場合に利用可能
  - 同じファイルポインタを共有する必要はない．
  - クローズは独立に実行可能



ファイルレイアウト制御（1）

●ファイルサーバタスク
–逐次型
–これらを論理的にまとめたものが**並列ファイルサーバ**の役割をする

●並列ファイル
–複数の順次ファイル(sequential file) に格納された固定長のブロックを Round-Robin 順序で並べたもの



ファイルレイアウト制御（2）

● ファイルレイアウトパラメータ…ファイルごとに指定可能
– **逐次型ファイルサーバの数と順序**
• 並列ファイルをインターリーブして格納するサーバの数
• サーバの間の順序
– **ブロック長**
• 逐次サーバ内で処理されるブロックのサイズ
– **冗長度**
• 多数のディスクが使用されることによるMTTFの低下を防ぐための冗長情報の与え方

ファイルレイアウト制御（3）

●ファイルレイアウトパラメータ（つづき）
– 逐次型ファイルサーバの数と順序
– ブロック長
– 冗長度
– **論理レコード長**
• 可変長レコードが使用される場合は，各レコードのサイズを事前に指定可能
• 各逐次サーバはこれをできるだけ同一ブロック内に配置

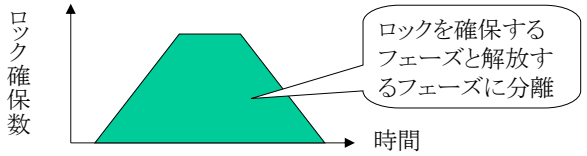
分散ファイルキャッシング（1）

●各タスクがアクセスするブロックに重なりがある場合
–一度読み込んだデータをキャッシュしておくとディスクアクセスを削減できる
–しかも，一度読み込んだデータが次に役に立つのは，別のプロセッサにおいてである
–各タスクのアクセスの順序をうまく制御しなくては不正な結果やデッドロックが生じることがある

●これは，並行処理における一貫性制御と同じ問題

一貫性制御方式の例（1）

- **二相ロック**（Two Phase Locking）方式…古典的方法
  - ロックをかけるフェーズとはずすフェーズが分離されるようにする．
    （ロックをはずすフェーズが始まるまでは全ロックを保持）
  - 各タスクが任意の順序でアクセスしても，等価な結果を生む直列スケジュールが存在することが保証される．
  - ただし，デッドロックの可能性あり

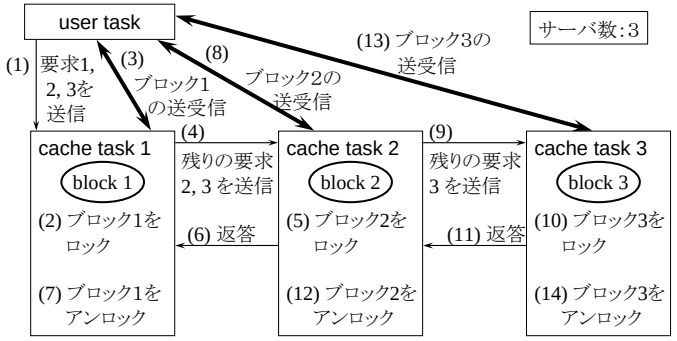


一貫性制御方式の例（2）

- **木プロトコル**（Tree Protocol）
  - アクセスする共有資源の間に半順序をつけ，全タスクがその順にしたがってロックを確保する
  - 2つのフェーズに分かれていなくても，等価な結果を生む直列スケジュールが存在することが保証される．
  - デッドロックを回避できることが保証される．
- Acaciaでは…
  2ブロック逐次ロック（Two Block Sequential Locking）
  木プロトコルの変種
  - ユーザタスクのアクセスするブロックの順序が，全ユーザタスクで一定になっていると仮定
  - そうすると，すべてのロックを保持しつづける必要はなく，最大2ブロックまででよい．

分散ファイルキャッシング（2）

- キャッシュタスク間のロック委託による2ブロック逐次ロック



分散ファイルキャッシング（3）

(1) ユーザタスクは全要求（1, 2, 3）をキャッシュタスク1に送る．
(2) キャッシュタスク1はブロック1をロックする（取れるまで待つ）．
(3) キャッシュタスク1はブロック1を送受信（方向は読み・書きによる）．
(4) キャッシュタスク1は残りの要求（2, 3）を次のキャッシュタスク2に転送する．
(5) キャッシュタスク2はブロック2をロックする（取れるまで待つ）．
(6) キャッシュタスク2はブロック2が取れたことをキャッシュタスク1に報告する．
(7) キャッシュタスク1はブロック1をアンロックする．
(8) キャッシュタスク2はブロック2を送受信（方向は読み・書きによる）．
(9) キャッシュタスク1は残りの要求（2, 3）を次のキャッシュタスク2に転送する．
(10) キャッシュタスク3はブロック3をロックする（取れるまで待つ）．
(11) キャッシュタスク3はブロック3が取れたことをキャッシュタスク1に報告する．
(12) キャッシュタスク2はブロック2をアンロックする．
(13) キャッシュタスク3はブロック3を送受信（方向は読み・書きによる）．
(14) キャッシュタスク3はブロック3をアンロックする．

分散ファイルキャッシング（4）

- 2ブロック逐次ロック
  - ステップ(3)と(4)は交換しても可
- 一般に，ロックによる並行処理では
  - リソースの空きを待つ順序のグラフに閉路がなければ，デッドロックはおきない
- 2ブロック逐次ロックは，ブロックのアクセス順序が線形であるので，複数のタスクがいてもサイクルはできない
- しかし，個々のキャッシュタスクに入ってくる要求の順序は線形でない

キャッシュタスクは，複数のタスクからの要求に並列・並行に応答できなければならない

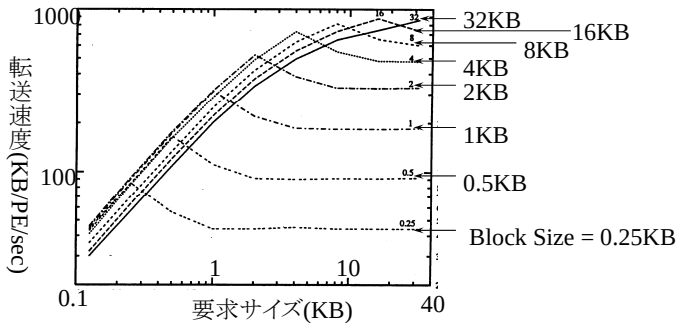
分散ファイルキャッシングの性能（1）

- 実験環境
  - ディスクなしPEのみのAP1000でシミュレーション(ストレージサーバはホスト)
  - キャッシュ入れ替えアルゴリズム：LRU
- パラメータ

キャッシュブロックサイズ(byte)	256	512	1024	2048	4096	8192	16384
要求サイズ(byte)	128	256	512	1024	2048	4096	8192
PE 数	2	4	8	16	32	64	128

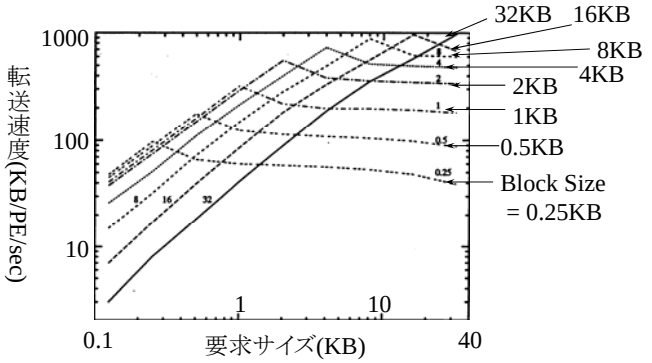
分散ファイルキャッシングの性能（2）

- 全PEがファイル全体に対し連続読み出しをかけるときのPE1台あたりの転送性能



分散ファイルキャッシングの性能（3）

- 各PEが重なりのない連続セグメントに書き込むとき

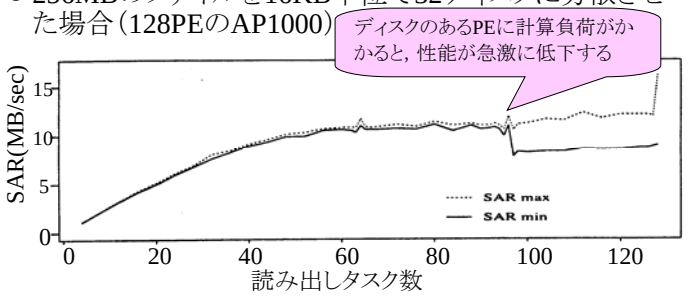


Acaciaファイルシステムの総合的な転送性能（1）

- 総合的な転送性能の指標
  - **持続総合転送速度**(Sustained Aggregate Rate)
    - max：各PEの転送速度の単純和
    - min：最も遅いPEでの処理が終わるまでの時間で転送データ量の総和を割る
- 実験で評価
  - 前提
    - ファイルのオープン・クローズのオーバーヘッドは無視
    - キャッシュによる性能向上を無視（毎回ディスクアクセス）
    - ファイルアクセスモードの選択による性能向上も無視

Acaciaファイルシステムの総合的な転送性能（2）

- 256MBのファイルを16KB単位で32ディスクに分散させた場合（128PEのAP1000）



•タスクが増えると，最初はSARが上昇，その後はだんだん飽和
•ディスクのあるPEでもユーザタスクが走り始めると SAR_{min} が極端に悪化

まとめ

- 並列プログラムのデータ入出力
  - 単一プロセッサに代表してやらせる
  - 各プロセッサで並列にやる
- 並列入出力のためのしくみ
  - **並列ファイルシステム**
    - Vesta
    - Acacia