Vol.2011-EVA-34 No.2
2011/3/7

基幹系グリッドバッチ向け I/O 性能見積もり手法の提案と評価

樫山俊彦† 花井知広† 鈴木芳生†

基幹系グリッドバッチでは，多ノードに起因する性能劣化により従来のストレージ/NW 帯域ベースの性能設計で高精度に見積もれない問題がある．本研究では I/O 時のファイル整合性保持用のメタ情報管理, 高多重 I/O 時の性能劣化に着目した新たな見積もり手法を提案する．本手法ではメタ情報管理時間をファイル数の比例関数，高多重 I/O 性能劣化をシーケンシャル，ランダム性能の確率関数でモデル化し，見積もり精度向上を図った．I/O 発行プログラムによる評価で，従来/提案手法の見積もり値と実測値の乖離率が 36.5%から 14.0%に向上した．提案手法がグリッドバッチ向け性能見積もり手法として有効なことが明らかになった．

Proposal and Evaluation of I/O-Performance Prediction Method for Mission-critical Grid-batch Processing

Toshihiko Kashiyama† Tomohiro Hanai†
and Yoshio Suzuki†

Aiming to solve the performance-degradation problem when multiple computing nodes are in use in mission-critical batch systems (so-called "grid-batch" systems), a new performance-prediction method that focuses on metadata management for file input/output (I/O) control and performance degradation in case of concurrent I/O streams is proposed. To enhance the accuracy of the prediction, this I/O-performance prediction method models metadata management time as a function of number of files and models performance degradation as a probabilistic function of sequential I/O throughput and random I/O throughput. According to an evaluation of the proposed method, the difference between actual and estimated execution time is 14.0%. In contrast, as for the storage/network-based conventional method, the difference is 36.5%. These results demonstrate that the target prediction error, namely, within 20%, was accomplished with the proposed method, which can therefore be considered effective in predicting the performance of grid-batch systems.

1. はじめに

銀行の勘定系システムや企業の生産管理システムに代表される基幹系システム[1]では，多数のバッチ処理が実行されている．近年，MapReduce[2]，GFS (Google File System)[3]が文書解析や BI (Business Intelligence)等の分散バッチ処理システムに用いられている．しかしながら，基幹系システムでは，アプリケーションのソースコード修正に伴う多くのテストが必要となるため, これらの技術を適用するのは難しい. 我々の研究グループでは，既存のバッチシステムからの移行性が高い基幹系システム向けバッチ処理分散並列実行制御技術(グリッドバッチ)を検討している[4].

従来，基幹系バッチ処理では，システム設計時に処理要求時間を満たすハードウェアスペックを計算するため, 性能見積もりを実施していた. 従来の性能見積もりでは，CPU ネックとなる場合， 1 レコードあたりの CPU 処理時間からバッチ全体の処理時間を算出し，I/O ネックとなる場合，1 レコードずつ順に処理するシーケンシャルファイル I/O となるため，ディスクアレイのシーケンシャルリード性能，およびシーケンシャルライト性能や，NW (Network)，FC (Fibre Channel)等のパス帯域からバッチ全体の処理時間を見積っていた．従来は，CPU ネックとなることが多かったが，近年の CPU のマルチコア化・メニイコア化によりノード内のプロセス数が増加し，グリッドバッチ適用に伴いノード数が増加するため，I/O を発行するプロセス数が増加し，I/O ネックとなる割合が高くなってきた. よって, バッチ処理全体の性能を見積もる場合，I/O 処理性能のより精緻な見積もりが必要となる．

しかしながら，グリッドバッチにより多数のノードで処理を実行する場合，複数ノードからの同時アクセスに対しても矛盾なく一貫性を保持する必要があるため，ファイルの同時アクセスをノード間で制御(以下では，メタ情報管理と呼ぶ)する負荷が顕在化することが想定される．特に，グリッドバッチ適用でファイルを分割して複数ノードの各プロセスに割り当てると，1 ファイルあたりのファイルサイズが小さくなり，メタ情報管理の割合が高くなることが考えられる．さらに，ディスクアレイに対して高多重のシーケンシャル I/O が発生すると，スループットが低下することが知られている[5,6]．よって，従来のディスクアレイのシーケンシャルリード性能，シーケンシャルライト性能や，NW，FC 等のパス帯域からの性能見積もりでは高精度に見積もりができず，バッチ処理全体の時間を予測できなくなる．

そこで，本研究では見積もり精度向上に向け，ノード間のファイル整合性保持用のメタ情報管理，および高多重 I/O の性能劣化に着目した新たな I/O 性能見積もり手法を提案する．本研究では，グリッドバッチ向け I/O 性能見積もりにおいても，従来の基幹系システム設計で使用していた安全率 1.2 を実現するため，見積もり精度 20%以

† (株)日立製作所 中央研究所
Central Research Laboratory, Hitachi, Ltd.

内を目標とする．

本報告の構成は以下の通りである．まず，2 章では，グリッドバッチの概要，およびグリッドバッチ向け I/O 性能見積もり手法の問題について説明し，本研究の課題を示す．次に，3 章では，課題を解決するノード間のファイル整合性保持用のメタ情報管理，および高多重 I/O の性能劣化に着目した新たな I/O 性能見積もり手法について説明する．4 章では，ファイル I/O 発行プログラムによって従来手法，および提案手法の見積もり値と実測値を比較することで，提案手法の有効性を確認する．最後に 5 章では，結論と今後の課題を述べる．

2. グリッドバッチ向け I/O 性能見積もり手法における問題および本研究の課題

2.1 グリッドバッチの概要

従来の基幹系バッチシステム，分散バッチシステム，およびグリッドバッチシステムの概要を図 1 に示す．従来の基幹系バッチシステムでは，ジョブスケジューラがバッチジョブを管理し，バッチジョブ実行制御がアプリケーション(AP)を実行する．AP はファイルシステムを介してディスクアレイにファイル I/O を発行する．

MapReduce[2]，GFS (Google File System)[3]等の分散バッチシステムでは，複数の計算ノード上で AP が実行される．また，コンピュータクラスタにおけるスケジューリング方法も多数提案されており[7,8]，商用製品でも分散バッチ処理向けのジョブスケジューラやファイルシステムが提供されている[9,10,11]．しかしながら，基幹系システムでは，AP のソースコードを新しいインターフェース(I/F)に修正するため，多くのテストが必要となり，これらの技術を適用するのは難しい．

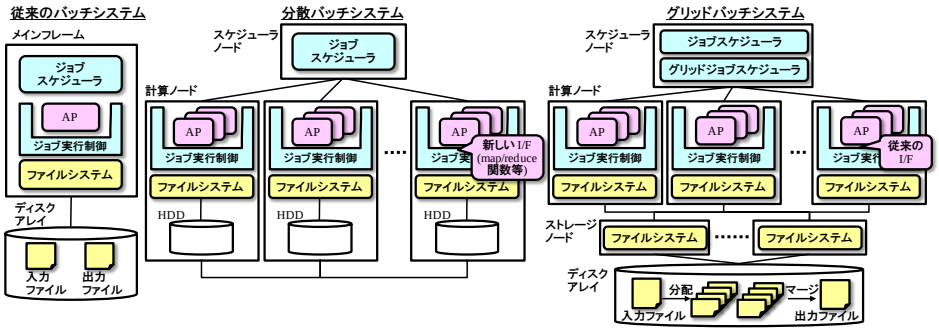

図 1　グリッドバッチ概要

一方で，グリッドバッチシステムでは，既存 I/F を用いるため，アプリケーションの修正量が少ない[4]．グリッドジョブスケジューラは細粒度のジョブに分割し，各計算ノードに割り振る．計算ノードでは，バッチジョブ実行制御により AP が複数プロセスで実行され，AP はファイルシステムを介してストレージノードに，ストレージノードはディスクアレイにそれぞれファイル I/O を発行する．AP の入力ファイルはプロセス単位に分配され，計算ノードの各プロセスに割り当てられる．また，各プロセスで実行された出力結果はマージされる．

2.2 従来の I/O 性能見積もり手法の問題

従来の性能見積もりでは，CPU ネックとなる場合は 1 レコードあたりの CPU 処理時間からバッチ全体の処理時間を算出し，I/O ネックとなる場合は入出力ファイルサイズ，およびシーケンシャルリード性能[MB/秒]，シーケンシャルライト性能[MB/秒]から処理時間を見積もっていた．シーケンシャルリード/ライト性能は，ディスク数やディスクアレイのキャッシュサイズにより変動し，文献[12]のようなストレージガイドラインにより提供される．ただし，ディスクアレイのシーケンシャルリード性能，シーケンシャルライト性能が NW 帯域(1Gbps→125MB/秒)や FC 帯域(4Gbps→500MB/秒)を超える性能を持つ場合には，NW や FC のパス帯域までの性能となる．そこで，入力ファイルサイズを F_I，出力ファイルサイズを F_O，シーケンシャルリードスループットを T_{SR}，シーケンシャルライトスループットを T_{SW}，I/O パス帯域を B_P とすると I/O ネックとなる場合の従来の見積もり I/O 実行時間 t_C に関して，以下の式が成り立つ．

$$t_C = F_I / min(T_{SR}, B_P) + F_O / min(T_{SW}, B_P) \quad (1)$$

グリッドバッチでは，図 1 に示すように入出力ファイルが分割され，計算ノードの各プロセスに割り当てられる．1 プロセスあたりの入力ファイルサイズを F_{IP}，1 プロセスあたりの出力ファイルサイズを F_{OP}，計算ノード数を N_N，ノードあたりのプロセス数を N_P，1 プロセスあたりの入出力ファイル数を N_{FP} とすると，全体ファイルサイズ $F_I + F_O$，全体ファイル数 N_F に関して以下の式が成り立つ．

$$F_I + F_O = (F_{IP} + F_{OP}) N_N N_P \quad (2)$$
$$N_F = N_{FP} N_N N_P \quad (3)$$

グリッドバッチでは，複数ノードからの同時アクセスに対しても矛盾なく一貫性を保持する必要があるため，ファイルの同時アクセスをノード間で制御(以下では，メタ情報管理と呼ぶ)する負荷が顕在化することが想定される．例えば，複数ノードから同一ファイルへの同時書込みがある場合でもファイルが破損してはならず，どのタイミ

ングでの読み取り要求でも最新のファイルが読み取られる必要がある．グリッドバッチで N_N, N_P が増加すると，式(2)より F_{IP}，もしくは F_{OP} が小さくなり，式(3)より N_F が増加する．そのため，従来のバッチ処理と比較し，メタ情報管理時間の割合が高くなり，式(1)で示した従来の性能見積もり手法では，高精度に見積もりできない問題がある．

さらに，ディスクアレイに対して高多重のシーケンシャル I/O が発生すると，スループットが低下することが知られている[5,6]. そのため, グリッドバッチでノード数，プロセス数が増加すると，高多重のシーケンシャル I/O が発行されることになり，式(1)で示した従来の性能見積もり手法では，高精度に見積もりできない問題がある．

近年，ストレージやファイル I/O は過去に性能見積もりのための多数のベンチマークが提案されている[13]．共有ファイルストレージである NAS (Network Attached Storage)では，SPEC SFS[14]と呼ばれるベンチマークが定義され，ファイルのリード処理，ライト処理に加えて，ファイル新規作成処理等のメタ情報管理を含めた性能をベンチマーク指標として出力する．また，DB 向けでは TPC-C や TPC-H 等のモデルアプリケーションに基づいたベンチマーク[15]が提案され，一部業務で性能見積もりに活用されている．文献[12]に示されるようなストレージガイドラインでは，シーケンシャルリード/ライト性能(MB/秒), ランダムリード/ライト性能(IOPS: I/O per second)を提供しており，NAS 性能ガイドラインでは，上記性能に加え，SPEC SFS の値を提供している．しかしながら，これらのベンチマークはグリッドバッチにおける上記問題を解決することはできず，基幹系グリッドバッチシステム向けベンチマークは提供されていない．

2.3 本研究の課題

グリッドバッチに適用する場合の従来 I/O 見積もり手法の問題を受け，見積もり精度向上に向けて表 1 に示すメタ情報管理に着目した I/O 性能見積もり手法の提案，および高多重 I/O の性能劣化に着目したファイル I/O 性能見積もり手法の提案を本研究の課題とした．

表 1　グリッドバッチ向け I/O 性能見積もり手法の問題と本研究の課題

グリッドバッチ向け I/O 性能見積もり手法の問題	本研究の課題
グリッドバッチシステムにおけるメタ情報管理時間の増加	メタ情報管理に着目した I/O 性能見積もり手法の提案
ディスクアレイに対する高多重シーケンシャル I/O 発行時の，ディスクアレイの全体スループット低下	高多重 I/O の性能劣化に着目したファイル I/O 性能見積もり手法の提案

3. I/O 性能見積もり手法の検討

3.1 課題に対するアプローチ

表 1 に示した課題の解決策として，表 2 に示すメタ情報管理時間をファイル数の線形関数で近似し，性能劣化をシーケンシャル I/O スループットとランダム I/O スループットの確率関数で近似するアプローチでグリッドバッチ向け I/O 性能見積もり手法を提案する．

表 2　課題に対するアプローチ

本研究の課題	アプローチ
メタ情報管理に着目した I/O 性能見積もり手法の提案	メタ情報管理時間をファイル数の線形関数で近似
高多重 I/O の性能劣化に着目したファイル I/O 性能見積もり手法の提案	性能劣化をシーケンシャル I/O スループットとランダム I/O スループットの確率関数で近似

3.2 メタ情報管理に着目した I/O 性能見積もり手法の提案

表 2 に示す最初のアプローチに対して，まず，メタ情報管理時間が増加する処理パターンを全体のバッチ処理パターンから抽出する．バッチ処理では，ファイルから読み込んだデータを別ファイルに書き出すことが基本となるため，入出力ファイル数や処理内容により，処理パターンを分類できる．基幹系システムでは COBOL で実装されることが多く，文献[16]では COBOL AP の処理パターンを入出力ファイル数，および処理内容から分類している．これらにはレコード抽出，レコード集計，ファイル分配，ファイルマージ，単独チェック，関連チェック，重複チェック，シーケンシャルファイル更新，ランダムファイル更新，レポート作成等の処理パターンが含まれる．

上記に示した処理パターンにおいて，メタ情報管理時間が顕在化する I/O 関数の要素(ファイル create や open 等)を含む場合，全体実行時間に対するメタ情報管理時間の割合が上昇する．ここで，ファイル分配処理パターンでは，ファイル create が複数回実行され，ファイルマージ処理パターンでは，ファイル open が複数回実行される．特に，グリッドバッチでは入力ファイルを各プロセスに振り分ける分配処理や，各プロセスで実行した処理結果を 1 つのファイルにマージする処理が増加されることが想定され，分配処理における出力ファイル数(分配ファイル数)，およびマージ処理における入力ファイル数(マージファイル数)も増加すると考えられる．よって，基準となる処理パターンであるレコード抽出パターン(以下では，単純処理と呼ぶ)に加え，分配処理，マージ処理を本研究のターゲット処理パターンとする．

単純処理，分配処理，およびマージ処理パターンにおける I/O 発行詳細を表 3 に示す．分配処理では 1 個の入力ファイルでシーケンシャルリード，N 個の出力ファイル

でシーケンシャルライトとなる．分配処理特有のパラメータとしては，分配ファイル数，および分配アルゴリズム(ラウンドロビン分配方式，順次分配方式)がある．マージ処理では，入力ファイルが N 個でシーケンシャルリード，出力ファイルが 1 個でシーケンシャルライトとなり，パラメータにマージファイル数，およびマージアルゴリズム(ラウンドロビンマージ方式，順次マージ方式)がある．

表 3　ターゲット処理パターンにおける I/O 発行詳細

処理パターン	I/O 発行詳細	入力ストリーム (I/O 関数)	出力ストリーム (I/O 関数)	その他のパラメータ
単純処理	1 入力ファイルより 1 レコードを read し，1 出力ファイルにレコード単位で write (出力ファイルのサイズは，入出力ファイルサイズ比によって相対的に指定).	シーケンシャル×1 (ファイル create×1)	シーケンシャル×1 (ファイル open×1)	入出力ファイルサイズ比
分配処理	1 入力ファイルより 1 レコードを read し，N 個のいずれかの出力ファイルに 1 レコードを write．1 レコード毎に出力ファイルを切り替えるラウンドロビン分配方式，(入力レコード数/N)レコードを write 後に次の出力ファイルに切り替える順次分配方式がある．	シーケンシャル×1 (ファイル create×1)	シーケンシャル×N (ファイル open×N)	分配ファイル数
マージ処理	N 個のいずれかの入力ファイルより 1 レコードを read し，1 出力ファイルに 1 レコードを write．1 レコード毎に入力ファイルを切り替えるラウンドロビンマージ方式，1 個の入力ファイルをすべて write 後に次の入力ファイルに切り替える順次マージ方式がある．	シーケンシャル×N (ファイル create×N)	シーケンシャル×1 (ファイル open×1)	マージファイル数

以下では，分配，およびマージ処理において，I/O ネックとなる場合の I/O 性能見積もり式を示す．分配処理では，表 3 に示すように N 個の出力ファイルに write するため，N 個のファイル create がメタ情報管理として最も大きな割合を占める．よって，メタ情報管理時間をファイル create メタ情報管理定数×分配数で近似する．そこで，ファイル create メタ情報管理定数を C_C，分配ファイル数を N_{SF} とすると，分配処理パターンにおける I/O 処理時間 t_S は以下の式で計算される．

$$t_S = F_I / min(T_{SR} , B_P) + F_O / min(T_{SW} , B_P) + C_C N_{SF} \quad (4)$$

マージ処理では，N 個の入力ファイルから read するため，N 個のファイル open がメタ情報管理として最も大きな割合を占める．よって，メタ情報管理時間をファイル open メタ情報管理定数×マージ数で近似する．分配処理パターン同様，ファイル open メタ情報管理定数を C_O，マージファイル数を N_{MF} とすると，マージ処理パターンにおける I/O 処理時間 t_M は以下の式で計算される．

$$t_M = F_I / min(T_{SR} , B_P) + F_O / min(T_{SW} , B_P) + C_O N_{MF} \quad (5)$$

分配処理パターン，およびマージ処理パターンにおける処理スループットは $(F_I + F_O) / t_S$，もしくは $(F_I + F_O) / t_M$ で計算される．

3.3 高多重 I/O の性能劣化に着目したファイル I/O 性能見積もり手法の提案

単一ノードでプロセス数が増加するとき，CPU ネックの場合は，CPU コア数まで比例して性能向上することが見込まれる．一方で，NW や FC 等のパス帯域ネック，およびストレージネックとなっている場合には，I/O の帯域を各プロセスで分け合うのみとなり，かつ I/O が競合することにより全体スループットが低下する．ストレージネックの場合，多重 I/O により，HDD のブロックが隣り合う確率が低くなり，I/O サイズ=AP バッファサイズのランダムリード，もしくはランダムライト性能に近づく．隣り合うブロックとなる確率を $1/N_P$ として，隣り合うブロックの場合に式(1), (4), (5)で示したシーケンシャル性能となり，隣り合うブロックでなかった場合にランダム性能(式(1), (4), (5)をランダムリード，ランダムライトスループットで算出した値)となるように確率関数でモデル化する．そこで，ランダムリードスループットを T_{RR}，ランダムライトスループットを T_{RW} とすると，1 プロセスにおける単純処理ランダム I/O スループット T_{RP}，および複数プロセス時の単純処理全体スループット T_{MP} は以下の式で計算される．

$$T_{RP} = (F_I + F_O) / \{F_I / min(T_{RR} , B_P) + F_O / min(T_{RW} , B_P)\} \quad (6)$$

$$\begin{aligned} T_{MP} &= (sequential\ I/O\ throughput)\ (1 / N_P) + (random\ I/O\ throughput)\ (1 - 1 / N_P) \\ &= [\ (F_I + F_O) / \{F_I / min(T_{SR} , B_P) + F_O / min(T_{SW} , B_P)\ \}\ ]\ (1 / N_P) \\ &\quad + [(F_I + F_O) / \{F_I / min(T_{RR} , B_P) + F_O / min(T_{RW} , B_P)\ \}\ ]\ (1 - 1 / N_P) \end{aligned} \quad (7)$$

複数ノードで実行する場合では，NW のパスの本数が増加するため，計算ノードのパスネックが解消され，ストレージネックやストレージノードの I/O パスネックとなりやすい．よって，計算ノードあたりの I/O パス帯域を B_{CP}，ストレージノードあたりの I/O パス帯域を B_{SP}，計算ノード数を N_{CN}，ストレージノード数を N_{SN} とすると，式(1), (4), (5)で示した見積もり式において $min\ (T_{SR}, B_P)$ を $min\ (T_{SR}, B_{CP}\ N_{CN}, B_{SP}\ N_{SN})$ で置き換えることができる．また，T_{SW}，T_{RR}，および T_{RW} も同様に置きかえることができる．プロセス並列同様，性能劣化を $N_N\ N_P$ の確率関数で近似すると，複数ノード

の全体スループット T_{MN} は以下の式で算出される．

$$T_{MN} = (\text{sequential I/O throughput}) / N_N N_P + (\text{random I/O throughput}) / (1 - 1 / N_N N_P)$$
$$= [\ (F_I + F_O) / \{F_I / \min(T_{SR}, B_{CP} N_{CN}, B_{SP} N_{SN}) + F_O / \min(T_{SW}, B_{CP} N_{CN}, B_{SP} N_{SN})\ \}\ ] / N_N N_P + [\ (F_I + F_O) / \{F_I / \min(T_{RR}, B_{CP} N_{CN}, B_{SP} N_{SN}) + F_O / \min(T_{RW}, B_{CP} N_{CN}, B_{SP} N_{SN})\ \}\ ] / (1 - 1 / N_N N_P) \quad (8)$$

分配処理パターンにおける全体スループット T_{MNS} は式(4), (8)から以下の式で算出される．

$$T_{MNS} = [\ (F_I + F_O) / \{F_I / \min(T_{SR}, B_{CP} N_{CN}, B_{SP} N_{SN}) + F_O / \min(T_{SW}, B_{CP} N_{CN}, B_{SP} N_{SN}) + C_C N_{SF}\ \}\ ] / N_N N_P + [\ (F_I + F_O) / \{F_I / \min(T_{RR}, B_{CP} N_{CN}, B_{SP} N_{SN}) + F_O / \min(T_{RW}, B_{CP} N_{CN}, B_{SP} N_{SN})\ \} + C_C N_{SF}\ \}\ ] / (1 - 1 / N_N N_P) \quad (9)$$

マージ処理パターンにおける全体スループットも式(5), (8)から同様に算出できる．

4. 評価

4.1 評価方法および評価環境

グリッドバッチ向け I/O 性能見積もり手法の評価として，表 3 に示す I/O 発行詳細に基づいて図 2 に示す I/O 発行プログラムを作成した．分配アルゴリズムは順次分配，マージアルゴリズムは順次マージとし，ファイル入出力は C 標準ライブラリ関数(fread, fwrite, fseek, setvbuf 等)を用いて実装した．

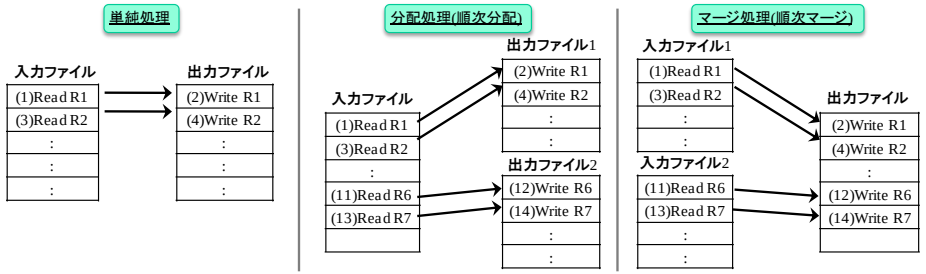

図 2　ターゲット処理パターンにおける read/write 順序

見積もり精度評価手順として，第一に，単純処理の測定実行時間から従来手法のシーケンシャルリード性能，およびシーケンシャルライト性能を算出する．第二に，分配処理の測定実行時間から提案手法におけるメタ情報管理定数 C_C を算出する．同様にマージ処理の測定実行時間から C_O を算出する．最後に，分配処理，およびマージ処理のあるパラメータ(分配数，マージ数)において，従来手法で計算される見積もり値，提案手法で計算される見積もり値，および実測値を比較し，見積もり値と実測値の処理時間乖離率を算出することで見積もり精度を比較する．並列度評価においても同様に見積もり値と実測値の処理時間乖離率から見積もり精度を比較する．

なお，I/O 性能測定の測定条件は以下の通りである．

- 入力ファイルサイズはノードあたり 2GB，レコードサイズは 1KB，AP バッファサイズは 2MB
- 測定間には OS のファイルキャッシュをクリアするため，umount，および mount コマンドを実行．また，ストレージノード上のファイルシステムの終了，および起動を実施
- 測定回数はパラメータあたり 3 回とし，3 回の測定結果の平均を算出

測定環境は，計算ノード 4 台，ストレージノード 1 台，ディスクアレイ 1 台とした．計算ノード，およびストレージノードは 2 個の CPU (Intel® Xeon® E5405 2.00GHz，4 コア)，8GB ECC DDR2 677 FB-DIMM を備え，OS は Red Hat Enterprise Linux 5.4 とした．ディスクアレイは 1GB のストレージキャッシュ，および 6 台の 300GB 15,000-rpm FC ディスクを備える．各ノードと NW スイッチとの接続は 1Gb Ethernet (MTU=9000)，各ノードと FC スイッチとの接続は 4Gbps FC，ディスクアレイと FC スイッチとの接続は 2Gbps FC×2 とした．ファイルシステム特有の影響を除くため，測定対象ファイルシステムとして 3 種類，すなわち，ext3，NFS (Network File System)，そして基幹系システム向けに設計されている Hitachi Striping File System (HSFS)を用意した．NFS ブロックサイズは 32KB (Linux の最大値)とした．5D+1P の RAID (Redundant Arrays of Inexpensive Disks)グループに 3 つの LU (Logical Unit)を作成し，各ファイルシステムを作成した．

4.2 処理パターン測定結果

ターゲットとする 3 つの処理パターンにおける実行時間を図 3 に示す．本測定は，1 ノード，1 プロセスで測定されている．図 3(a)では，横軸が入出力ファイルサイズ比とし，縦軸を実行時間[秒]とする．各ファイルシステムの実行時間は入出力ファイルサイズ比の 1 次関数で近似できる．各ファイルシステムは同一の RAID グループ上に作成されるため，同等の性能となることが想定されるが，NFS の実行時間が ext3，および HSFS の実行時間よりも大きくなっている．sar コマンドにより取得した CPU, NW, I/O のリソース使用状況を確認したところ, NFS のストレージノードでの I/O wait

が確認され，NFS の先読みが効果的に実行できず，1 プロセスでディスクアレイ性能を引き出せていないことが考えられる(4.3 で再度議論する).

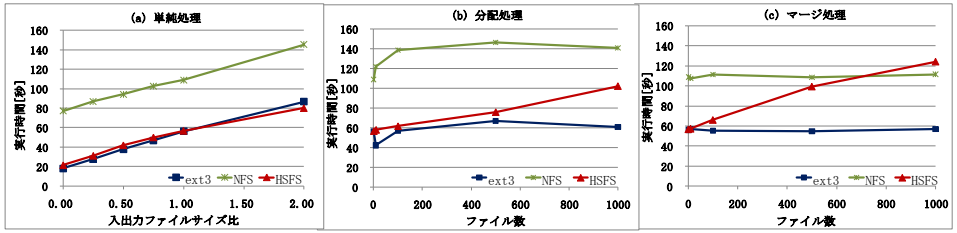

図 3　処理パターン測定結果

以下では，見積もり値と実測値から見積もり精度を評価する．第一に，単純処理入出力比 0.0 の場合，シーケンシャルリードのみとなるため，シーケンシャルリード性能を算出できる．また，入出力比 0.0，および 1.0 の実行時間の差からシーケンシャルライト性能を算出できる．従来見積もり手法で用いるシーケンシャルリード，シーケンシャルライト性能は表 4 に示す通りとなる(1GB=1024MB で算出している).

表 4　従来見積もり手法により算出したシーケンシャルリード/ライトスループット

ファイルシステム	シーケンシャルリードスループット	シーケンシャルライトスループット
ext3	2GB / 17.9 秒 = 114.4 MB/秒	2GB / (56.2 秒-17.9 秒) = 53.5MB/秒
NFS	2GB / 77.0 秒 = 26.6MB/秒	2GB / (108.9 秒-77.0 秒) = 64.2MB/秒
HSFS	2GB / 21.5 秒 = 95.3MB/秒	2GB / (56.8 秒-21.5 秒) = 58.0MB/秒

第二に，分配処理，およびマージ処理の測定実行時間から C_C，および C_O を算出する．図 3(b)，(c)では，横軸にファイル数，縦軸に実行時間[秒]とする．ext3 の実行時間はファイル数に関わらず一定となるのに対し，NFS，HSFS ではファイル数が増加するにつれて，実行時間が増加する．特に，HSFS では書込み途中のデータ読み出し，2 重書込み等が発生しないようにファイルの同時アクセスをノード間で制御しているため，メタ情報管理コストが高くなり，分配数，マージ数増加に伴う実行時間増加が顕著となっている．提案手法において，HSFS での分配処理の実行時間から線形近似を算出すると，C_C=0.044 が得られる．同様に，マージ処理の実行時間から C_O =0.069 が得られる．

最後に，HSFS における実測値と見積もり値の比較(ファイル数 500)を図 4 に示す．従来手法の見積もり値と実測値の乖離率が 25.1%となるのに対し，提案手法の見積もり値と実測値の乖離率が 3.8%となり，見積もり精度が向上することが確認できた．また，HSFS のマージ処理(マージ数 500)でも，従来手法の見積もり値と実測値の乖離率が 42.7%となるのに対し，提案手法の見積もり値と実測値の乖離率が 8.0%となり，見積もり精度が向上することが確認できた. よって, 提案手法を用いることで分配処理，マージ処理ともに目標である見積もり精度 20%以内を達成することができた．

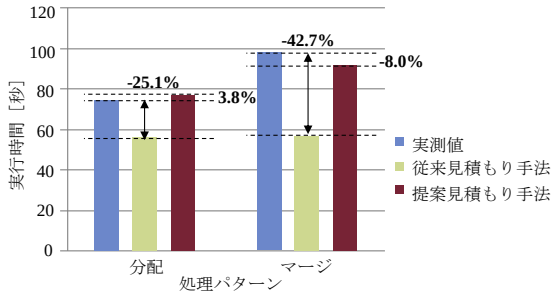

図 4　HSFS における実測値と見積もり値の比較(分配/マージファイル数 500)

4.3　並列度測定結果

次に，プロセス数，およびノード数を変化させた場合の測定結果を図 5 に示す．横軸を並列度[(ノード数，プロセス数)]とし，縦軸をスループット[MB/秒]とする．本測定では，各プロセスのファイルサイズは 2GB / N_P，入出力ファイルサイズ比は 1.0，分配ファイル数，およびマージファイル数は 500 としている．また，ext3 はノード間のファイル共有機能を持たないため，ノード数 1 の場合のみを測定している．HSFS のマージ処理を除き，一般的にはプロセス数，ノード数増加に伴い，スループットが低下する．特に，NFS においてスループット低下が顕著である．

ここで，NFS では，単純処理，分配処理，マージ処理ともに並列度(1, 1)よりも並列度(1, 2)の方が，スループットが向上している．4.2 でも述べたように，NFS の先読みが効果的に実行できず，並列度(1, 1)ではディスクアレイの性能を引き出せてないため，提案手法の見積もり式における近似曲線から外れていると考えられる．並列度(1, 1)を除く単純処理測定結果の-1 次の多項式近似を算出すると，y = 38.6 / x + 36.6 となり，並列度(1, 1)におけるスループット 75.2MB/秒は ext3, および HSFS の実測値に近づく．式(8)におけるシーケンシャル I/O スループットは 75.2MB/秒，ランダム I/O スループットは 36.6MB/秒となる．

一方，HSFS では，図 5(b)，および(c)に示すようにプロセス数，およびノード数が

増加する場合にスループットが向上している．3.2 で示したように，分配処理，およびマージ処理ではメタ情報管理により処理時間が増加する．しかしながら，複数ノードでは，あるノードがメタ情報管理中に他ノードで read 処理，write 処理が実行でき，全体としてスループットが向上していると考えられる．

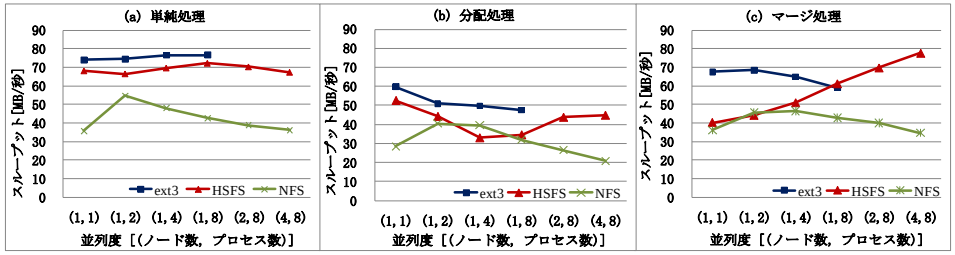

図 5　並列度測定結果

以下では，HSFS，および NFS における並列度(4, 8)の分配処理(分配数 500)を用いて，見積もり精度を評価する．まず，提案手法では，表 4，および式(4)から処理時間が 78.8 秒と算出されるため，分配処理のシーケンシャルスループットは(2GB+2GB)/78.8 秒 =52.0MB/秒となる．次に，測定スループットに対して，–1 次の多項式近似を取ると，–1 次の係数は 12.1 となる．さらに，x=1 のときに y=52.0 となることから見積もり式 y=12.1/x + 39.9 が得られる．よって，並列度(4, 8)におけるスループットは 12.1/(4×8) + 39.9 = 40.2 MB/秒となり，見積もり処理時間は(2048+ 2048)×4/40.2 = 407.6 秒となる．

同様に，NFS では，前述した並列度(1, 1)における単純処理スループット 75.2MB/秒，および C_C=0.024 を用いる．式(4)から分配処理パターンのスループットは 61.6MB/秒となる．並列度(1, 1)を除く NFS の測定スループットから，–1 次の多項式近似を取ると，–1 次の係数は 38.3 となり，y=38.3 / x + 23.3 が得られる．よって，NFS 見積もり処理時間は 668.7 秒となる．

以上の結果をまとめると，分配処理における実測値と見積もり値の比較(並列度(4,8)，分配ファイル数 500)は図 6 に示す通りとなる．従来手法の見積もり値と実測値の乖離率が 36.5%となるのに対し，提案手法の見積もり値と実測値の乖離率が 14.0%となり，見積もり精度が向上することが確認できた．また，NFS において，従来手法の見積もり値と実測値の乖離率が 40.8%となるのに対し，提案手法の見積もり値と実測値の乖離率が 14.0%となり，見積もり精度が向上することが確認できた．よって，提案手法を用いることで HSFS，NFS ともに目標である見積もり精度 20%以内を達成することができた．

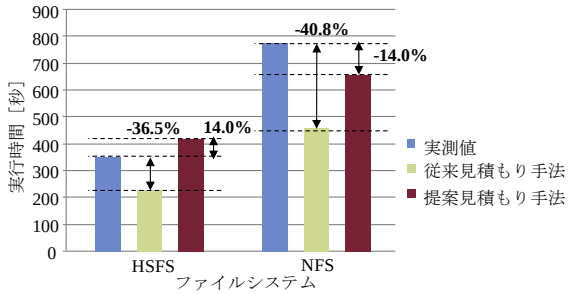

図 6　分配処理における実測値と見積もり値の比較(並列度(4,8)，分配ファイル数 500)

5. おわりに

複数の計算ノードで実行される基幹系バッチシステム(グリッドバッチシステム)では，多ノードから発生する高多重 I/O により従来のディスクアレイのシーケンシャルリード性能，シーケンシャルライト性能や，NW，FC 等のパス帯域からの性能見積もりでは高精度に見積もりできない問題がある．本研究では I/O 時のファイル整合性保持用のメタ情報管理，高多重 I/O 時の性能劣化に着目した新たな見積もり手法を提案した．本手法では，メタ情報管理時間をファイル数の比例関数，高多重 I/O 性能劣化をノード数，プロセス数で決定されるシーケンシャル，ランダム性能の確率関数でモデル化し，見積もり精度向上を図った．HSFS の分配処理において，従来手法の見積もり値と実測値の乖離率 36.5%に対し，提案手法の見積もり値と実測値の乖離率が 14.0%となり，提案手法を用いることで見積もり精度 20%以内達成が確認できた．よって，提案手法がグリッドバッチ向け性能見積もり手法として有効なことが明らかになった．

今後の課題として，まず，提案手法を I/O 制御に活用することが挙げられる．例えば，ノード数，およびディスクアレイの I/O スループットから最適な実行プロセス数を決定することが考えられる．次に，本報告の見積もり式を適用するには，分配処理，マージ処理等の処理パターン，および分配ファイル数，マージファイル数を手動で分析しなければならないため，アプリケーションのソースコード等から上記を自動抽出する技術の開発が必要である．

参考文献

1)　Fowler K.: Mission-Critical and Safety Critical Development, IEEE Instrumentation &

Vol.2011-EVA-34 No.2
2011/3/7

Measurement Magazine, vol.7, no.4, pp.52-59 (2004).
2) Dean J., et al.: MapReduce: Simplified Data Processing on Large Clusters, Commun. ACM Vol.51, No.1, pp.107-113 (2008).
3) Ghemawat S., et al.: The Google file system, Proc. SOSP '03, pp.29-43 (2003).
4) 仲田智将 他: エンタープライズグリッドによる次世代オープンアーキテクチャ, 日立評論 Vol.92, No.5, pp.32-35 (2010).
5) Panagiotakis G., et al.: Reducing Disk I/O Performance Sensitivity for Large Numbers of Sequential Streams, ICDCS2009, pp.22-31 (2009).
6) Li C., et al.: Competitive Prefetching for Concurrent Sequential I/O, EuroSys07, pp.189-202 (2007).
7) Isard M., et al.: Quincy: Fair Scheduling for Distributed Computing Clusters, SOSP 2009 (2009).
8) Zaharia M., et al.: Delay Scheduling: a Simple Technique for Achieving Locality and Fairness in Cluster Scheduling, EuroSys10, pp.265-278 (2010).
9) Antani S.: Batch Processing with WebSphere Compute Grid: Delivering Business Value to the Enterprise, IBM Redbooks (2010), http://www.redbooks.ibm.com/redpapers/pdfs/redp4566.pdf
10) Schmuck F., et al.: GPFS: A Shared-Disk File System for Large Computing Clusters, Proc. FAST 2002 (2002).
11) Lang S., et al.: I/O Performance Challenges at Leadership Scale, Proc. SC09 (2009).
12) Racherla S., et al.: IBM Midrange System Storage Implementation and Best Practices Guide, IBM Redbooks (2010), http://www.redbooks.ibm.com/redbooks/pdfs/sg246363.pdf
13) Trayger A., et al.: A Nine Year Study of File System and Storage Benchmarking, ACM Transactions on Storage, Vol.4, No.2, pp.1-56 (2008).
14) SPEC SFS, http://www.spec.org/osg/sfs/
15) TPC: Transaction Processing Performance Council, http://www.tpc.org/
16) 大野治 他: 多次元部品化方式によるソフトウェア開発の自動化-バッチプログラム用スケルトンの作成とその十分性-, 電子情報通信学会論文誌 Vol.J83-D-I, No.10, pp.1055-1069 (2000).