分散ファイルシステムの性能評価に関する研究

櫻井, 雅志 / SAKURAI, Masashi

(発行年 / Year)
2011-03-24
(学位授与年月日 / Date of Granted)
2011-03-24
(学位名 / Degree Name)
修士(工学)
(学位授与機関 / Degree Grantor)
法政大学 (Hosei University)

2011年度 修 士 論 文

分散ファイルシステムの
性能評価に関する研究

STUDIES ON EVALUATING PERFORMANCE EFFICIENCY
OF DISTRIBUTED FILE SYSTEMS

指導教官　三浦 孝夫 教授

法政大学大学院工学研究科
電気工学専攻修士課程

09R3110　櫻井 雅志
Masashi SAKURAI

目 次

第1章　序論

1.1　問題の背景

近年、増え続ける情報の分析と保存の重要性がますます高まっている。そこで、複数のコンピュータをネットワークで接続し情報処理する分散コンピューティングが提案され利用されている。Facebook では 2000 台のコンピュータで 21PB を保存し、日々 12TB ものデータが追加されている。分散コンピューティングを用いることにより、1 台のコンピュータを越える処理性能が得ることができる。

また、信頼性ある情報処理が求められており、トランザクション機構が使われている。入れ子トランザクションは分散環境で、従来より高度な排他制御が求められており、競合時の性能予測が困難である。

一方、分散コンピューティングではファイルシステムの信頼性や処理性能も求められている。そこでトランザクション機構とは別の方法を用いる必要がある。トランザクション機能のような高度のデータ整合性を要求するケースは稀であり、むしろ頑丈性や性能を重視する傾向にあり、必要とされる条件もかなり異なる。実際、想定されている信頼性はレコード単位であり、冗長化することで解決できる。分散環境では手近な複製があれば検索効率は向上するが、場所の特定や複製データの同一性を保証する機構など、考慮せねばならない問題も多く、古くから研究が続けられている。

分散システムの評価として、信頼性・経済性・セキュリティ・処理性能などがある。また、分散ファイルの最適配置問題はＮＰ完全であり、理論解はあっても現実的ではないことが多い。むしろ、実環境に即した状況の測定や環境のシミュレーションから判断する方が現実的であることが多い。

1.2　扱う問題

本研究では、分散環境における性能評価を行い競合や影響を調べ、高いパフォーマンスを持つシステムを実現するための問題点や考慮すべき点を明らかにする。

1.2.1　入れ子トランザクションの性能評価

トランザクション機構は、現在さまざまなシステムで用いられている。しかし、長時間トランザクションでは数分から数時間かかるトランザクションの最後にアボートされた場合、今までの処理は無かったことになる。また、長時間施錠し続ける。そこで Moss らにより、入れ子トランザクションが提案されている。

入れ子トランザクションは従来のトランザクションとは異なり、階層構造を持つ。入れ子トランザクションでは、ACID 特性は拡張され、同時に拡張された施錠ルールが用いられる。排他ロックは、直系が維持していれば取得することができ、親トランザクションに継承・併合される。

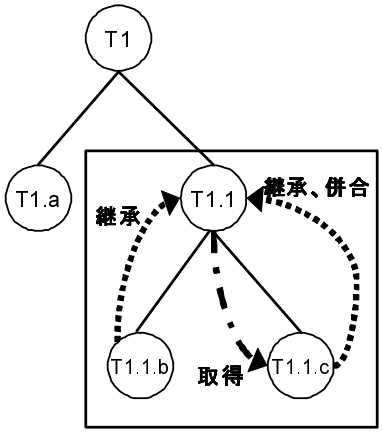

図 1.1: 図1 ロックの流れ

本研究では、仮想的な時間を導入し分散シミュレーションする。仮想的な時間であるサイクルを用いることにより、システムのすべての処理時間や遅延時間が全てとして表され、コンピュータの性能や回線の帯域に依存せずに評価することができる。サブトランザクションの生成では、自ノードであれば、子トランザクションは直ちに生成される。しかし、他ノードでは回線による遅延がおき生成される。また、同時に複数のサブトランザクションを生成する場合でも、同様に遅延の影響を受ける。葉トランザクションでは、同一ノードでは並列演算は起きない。

入れ子トランザクションは木構造と捉えることができ様々な形状が考えられる。木構造に偏りがまったく無いのは完全N 分木であるが、様々な条件を満たしながら深さを変えることはできない。実験では、全ての内部トランザクションが持つ子はN 又はN-1 の時に木構造に偏りが少ないとし、これを用いる。子トランザクションを同一に扱うときの再実行戦略は、 即時再実行・最低限再実行・再実行なしに分類できる。本研究では、通信状況や競合の相互依存性の影響を調べるため分散環境下での性能を示す。[1]

1.2.2 分散ファイルシステム Hadoop Distributed File System の性能評価

データ量の増加に伴い分散化による効率改善が強く求められている。一方、信頼性に関して、トランザクション機能では性能的に手に負えない状況となるため、分散ファイルの信頼性とトレードオフとなる程度の容量や機能・性能が求められる。

本研究では、HDFS を用いその特性を示す。また、我々は Filesystem in Userspace(FUSE) を用い分散ファイルシステムの性能評価を行う。Hadoop プロジェクトの基盤である HDFS は、Google File System の影響を受け巨大データ向け、高いスケーラビリティ、シングルマスターといった特徴を持つ分散ファイルシステムである。

クライアントは、ネームノードに問い合わせ、データノードから実際のデータであるブロックを読み書きする。また、ネームノードとデータノードは協調し、ブロックを複製する。

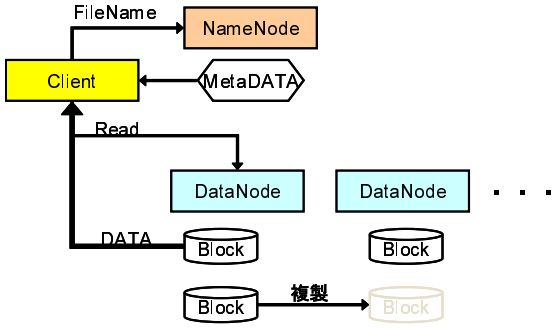

図 1.2: 図 2 HDFS の構造

分散ファイルシステムの処理性能は、構成するデータノードの数やその転送速度に大きく影響を受ける。データノード数が増加するにつれクライアントは、複数のノードから読み込みすることができる。しかし、ネームノードはクライアントの状況を把握する負荷が増加する。

本実験では、FUSE を用いシステムの性能評価及びネームノードの影響を調べるため実験を行う。FUSE はユーザー空間でローカル同様のファイルシステムアクセスを提供する。しかし、VFS や libfuse によるオーバーヘッドによる遅延があり、メタデータの集中管理するネームノードへの影響はシステム全体に影響する。

また、分散ファイルシステムは、通信プロトコルにおいては上位レイヤに相当するため、OS やプロトコル自体の汎用的な負荷を伴う。OS ファイル機能を分散させる代表的なものに NFS (Network File System) がある。容量や施錠など OS と深く結びついた機構は、本稿で論じる分散ファイル環境レベルの単純処理には必要とされず、比較するにはふさわしくない。　しかし、OS レベルを機構を直接介するため、本実験でベースラインを得るために、手動で複製を作成し、疑似分散機構として比較対象とする。

合計 10GB の分散読み込みを行い最後のクライアントが終了した時間を実行時間として評価する。また、単一ボトルネックであるネームノードの CPU 時間、ネットワーク負荷を測定する。[2]

1.3　論文の構成

本研究では、以上の問題について以下の構成で論じる。第 2 章では、入れ子トランザクションの性能評価について論じる。第 3 章では、分散ファイルシステム Hadoop Distributed File System の性能評価ついて論じる。第 5 章で結論とする。

1.4　発表論文

1. 櫻井雅志, 三浦孝夫: 入れ子トランザクションの性能評価, データ工学と情報マネジメントに関するフォーラム (DEIM), 2009.

2. 櫻井雅志, 三浦孝夫: 分散ストレージ Hadoop Distributed File System の性能評価, 電子情報通信学会 2011 年総合大会学生ポスターセッション, 2011.

第2章　入れ子トランザクションの性能評価

2.1　前書き

一般に長時間トランザクションは、異常終了時の回復コストが非常に高く、この対応として入れ子トランザクション概念が提案された。しかし、入れ子トランザクションの処理性能は状況や環境に大きく依存するため、処理性能の予測が困難である。本稿では、入れ子構造の複雑さ、通信状況や競合の相互依存性の影響を調べるため、分散環境下での性能評価を行う。

トランザクション機構は、現在さまざまなシステムで用いられている。トランザクションは分散環境で障害に対処でき、高い信頼性を得ることができる。[2] トランザクションは、信頼性を得るために ACID と呼ばれる特性が提案されている。また、この特性を満たすために、施錠、時刻印、2 相コミット、シャドウページングなどが用いられている。しかしまだなお、さまざまな問題が指摘されている。例えば、長時間トランザクションでは、数分から数時間かかるトランザクションの最後にアボートされた場合、今までの処理は無かったことになる。また、長時間施錠し続ける。

そこで Moss[1] らにより、入れ子トランザクションが提案されている。入れ子トランザクションは従来のトランザクションとは異なり、階層構造を持つ。また、この階層構造は木構造としてとらえることができる。この特徴により、子トランザクションに応じた処理が可能になり、部分的再実行、代替処理、きめ細かい同時制御などを行うことができる。

入れ子トランザクションでは、従来のトランザクションと共通する問題もあるが、入れ子トランザクション特有の現象や問題がある。例えば、子がトランザクションがコミットしたあとに、親トランザクションがアボートしたとき、従来の ACID 特性では、他のトランザクションから見た親トランザクションの原子性と子トランザクションの永続性は相反する。[3] このため、単純なトランザクション機構の処理性能を予測することが容易ではなく、パラメタ設定などで大きな効率の差を生む。

入れ子トランザクションは内部状態に応じて処理を行うため、状況や環境に大きく依存する。本稿では、入れ子トランザクション構造、通信や競合の相互依存性の影響を調べるため、分散環境下での性能評価を行う。

2 章では入れ子トランザクションの概略を述べ、3 章で入れ子トランザクション実現機構について述べる。4 章で性能評価方法を示し、5 章で実験・考察を行い、6 章で結

びとする。

2.2 入れ子トランザクションと従来のトランザクション機構

入れ子トランザクションの概要について述べる。

2.2.1 トランザクション

従来のトランザクション機構は、以下の ACID 特性を満たす。原子性 (A) とは、トランザクションが、成功（コミット）するか失敗（アボート）するかのどちらかであり、アボートは無かったことにしなければならない性質のことであり、一貫性 (C) とは、トランザクションは整合条件を満たす性質である。また、隔離性 (I) とはは、並列動作するトランザクションが、他のトランザクションから見えない性質である。耐久性 (D) は、コミットした場合、結果は永遠に消えない性質である。

この 4 つの ACID 特性により、複数のトランザクションが同時に実行する環境では、それが直列した実行と同じ効果を生むという直列可能性が、トランザクション機構の結果の正しさを保証する。

ACID 特性を満たすために、排他制御に施錠を用いることが多い。施錠は以下のルールが用いられる。読み込み、書き込みには、それに応じたロックを持っている時に行なう。誰もロックを持っていなければ、書き込みロックを持つことができる。誰も書き込みロックを持っていなければ、読み込みロックを持つことができる。ロックはいつでも開放できる。これらのルールを用いた施錠により、他のトランザクションは待機させられ、競合状態が生じないことが保障される。

2.2.2 入れ子トランザクション

入れ子トランザクションは、内部に別のトランザクションを含むことができる。また、この構造は木構造としてとらえることができる。つまり、トランザクションは、複数の子トランザクションを持つことができ、ただ一つだけ親トランザクションを持つ。子を持っていないトランザクションを葉トランザクションと呼び、子を持つトランザクションを内部トランザクションと呼ぶ。また、深さが 0 レベルのトランザクションをトップレベルトランザクションと呼ぶ。

2.2.3 入れ子トランザクションの ACID 特性

入れ子トランザクションでは、以下の ACID 特性に拡張される。原子性は従来と同様であり、一貫性はトランザクションは整合条件を満たす性質である。隔離性は、拡

張され親トランザクションは自分の子トランザクションに限り見ることができる。耐久性では、トップレベルトランザクションがコミットした場合に限り永続的である。

また、施錠は以下のように拡張されたのルールが用いられる。読み込み、書き込みは、それに応じたロックを「持っている」時に行なう。そして、誰もロックを持っていなく、かつ、全てのロックを直系の先祖が「維持」していれば、書き込みロックを持つことができる。誰も書き込みロックを持っていなく、かつ、全ての書き込みロックを直系の先祖が維持していれば、読み込みロックを持つできる。子がコミットするとき、子が「持っている」ロックと「維持」しているロックを親に継承する。親は継承したロックを維持する。トランザクションがアボートするときは、単純にロックを開放する。

例えば、以下の木構造を考える。T1 はトップレベルトランザクションであり、T1 は T1.1 と T1.a の子トランザクションを持ち、T1.1 は T1.1.b と T1.1.c の子トランザクションを持つ。T1.1.b がコミットしたとき、その結果は T1.1 と T1.1.c からは見えるが T1、T1.a や他のトランザクションからは見えない。

また、ロックの流れは以下の通りである。また図 1 に示す。T1.1.b が書き込みロックを取得し、「持つ」。T1.1.b がコミットし、T1.1 はロックを継承し、ロックを「維持」する。このとき T1.1 がアボートすると T1.1 は維持しているロックを開放する。T1.1 がアボートしないときを考える。T1.1.c は T.1 からロックを取得し持ち、T1 はロックをそのまま維持する。T1.1.c がコミットすると、T1.1 はロックを継承し、ロックを「併合」する。

2.3 入れ子トランザクション実現機構

本章では、入れ子トランザクションの実現機構について述べる。

2.3.1 実現機構

システムには、複数のノードが存在する。ノードは、演算装置、通信回線、揮発性メモリ、永続性メモリからなる。ノードは、複数のトランザクションと一つのトランザクションマネージャが存在する。

葉トランザクションは、演算処理を行うことができる。この演算処理は、各ノードで 1 サイクルあたり 1 回行うことができる。演算処理を行う必要が無ければ、行わない。また葉トランザクションは、データの読み込み及び書き込み、コミット、アボートを行う。読み込み、書き込み、コミットはブロックされる可能性がある。

内部トランザクションは、さまざまな情報を元に判断を行うことができる。例えば、子トランザクションの状態や終了状態、経過時間やデッドライン、ノードの状態や負荷状態がある。しかし、その情報を元にできる判断は多くはない。内部トランザクショ

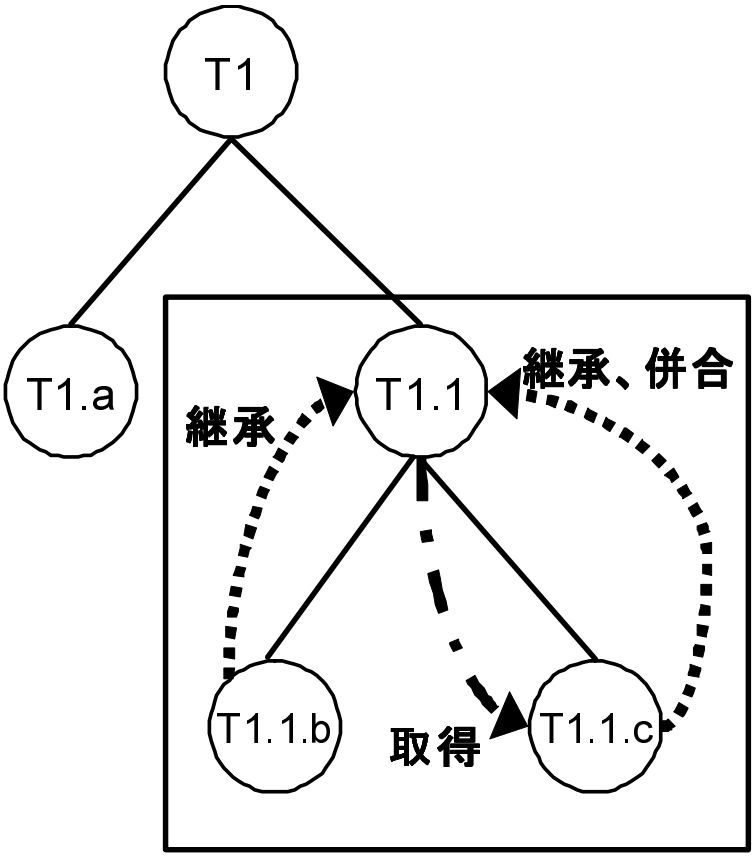

図 2.1: ロックの流れ

ンが行う判断は、わずか 4 つだけである。子を生成するかアボートさせる。自分をコミットするかアボートする

各ノード、トランザクションはメッセージを送受し、トランザクションを実行していく。まず、ノードが「create transaction」メッセージを受け取り、トップレベルトランザクションが生成される。トップレベルトランザクションは、「create subtransaction」により、内部か葉トランザクションを生成する。葉トランザクションは「lock」、「unlock」、「inheritance」といったメッセージをやり取りする。最後に、コミットした場合は親に「commited」を通知し、親トランザクションは判断を行う。

2.3.2 実現機構の問題

従来のトランザクション処理システムには無い問題が生じる。一つ目は、トランザクションが構造を持つことにより、その構造内での通信の必要性が生じる。このとき、自分の子トランザクションが同一ノードにいる保障はなく、他ノードであれば遅延が発生する。また、施錠におけるロックの取得、継承も同様の問題が起こる。二つ目に、木の構造、形状による差異が存在する。内部や葉トランザクションの数、偏りや深さ

によって差が生じる。

2.4 性能評価

我々は、シミュレーションにより、性能評価を行う。シミュレーションは、MPI の実装である MPICH2 を用いて行った。MPI により各ノードの時刻やメッセージの同期を行い、性能等に依存しない評価を行うことができる。

2.4.1 評価方法

我々は、葉トランザクションの深さを変えて、トップレベルトランザクションのコミット時のサイクルを比較する。同じシミュレーションを 8 回行い、コミット時間の平均を各深さで比較する。

以下に我々が用いるシミュレーションについて述べる。

2.4.2 仮想時間

我々は、本シミュレーションに仮想的な時間「サイクル」を導入する。サイクルは整数である。システムの全ての処理時間や遅延時間といった全ての時間が、全てサイクルで表される。サイクルを導入することにより、コンピュータの性能や回線の帯域に依存せずに評価することができる。本実験ではサイクルを消費するのは葉トランザクションの演算処理とした。例えば、回線遅延が 3 サイクルとする。サブトランザクションの生成では、自ノードであれば、子トランザクションは直ちに生成される。しかし他ノードでは、回線による遅延が起き、生成されるのは 3 サイクル後である。また、同時に複数のサブトランザクションを生成する場合でも、同様に 3 サイクル後である。しかし、葉トランザクションの演算処理は異なる。同一ノードでは、並列演算は起きない。7 サイクルと 5 サイクルの長さを持つ葉トランザクションは 12 サイクル後には両方終了している。ここで、スケジューラの問題が起こるが、本シミュレーションでは単純に先に生成された実行可能なトランザクションを実行する。異なるノードで実行された場合、7 サイクル後及び 5 サイクルに終了するがその終了を他ノードが知るのは 10 サイクル及び 8 サイクル後である。

2.4.3 木構造

入れ子トランザクションの木構造はさまざまな形状が考えられる。本シミュレーションでは、特に深さによる影響を調べる。そこで次の簡潔な構造を考える。

- サブトランザクションは、その全ての子トランザクションを同等に扱う。

- 葉トランザクションは全て同一レベルである。
- 木構造に偏りが少ない。

木構造に偏りがまったく無いのは完全 N 分木であるが、様々な条件を満たしながら深さを変えることはできない。実験では、全ての内部トランザクションが持つ子は N 又は N-1 の時に木構造に偏りが少ないとし、これを用いる。

また、本シミュレーションでは、子トランザクションの終了状態に着目して実験を行う。終了状態とは、コミット・アボート・未終了である。子の終了状態は判断される情報の中でもっとも基本的かつ重要なものである。子の数と終了状態を元にアクション表が書けるが、子トランザクションを同等に扱う場合は以下 3 つのルールいずれかと 2 つのパラメタで記述できる。パラメタ α と β は次の不等式を満たす。

$$0 \leq \alpha \leq \beta \leq \text{子トランザクション数}$$

1. 即時再実行

   コミットした子トランザクションが α を超えるまで、アボートした子トランザクションを即再実行する。

2. 最低限再実行

   アボートした子トランザクションが $\beta - \alpha$ を超えたら、その分だけ子トランザクションを再実行する

3. 再実行なしアボートした子トランザクションが $\beta - \alpha$ を超えたら、アボートする。

上記に加え全ルールで、コミットした子トランザクションが β 以上なら他の子トランザクションをアボートさせて、直ちにコミットする。

また、コミットしたとき、次の不等式を満たす。

$$\alpha \leq \text{子トランザクションのコミット数} \leq \beta$$

2.5 実験

2.5.1 準備

本実験では、表 1 のパラメタを用いてシミュレーションを行った。また、施錠の競合による待ちを想定せず、データの読み込み及び書き込みによる待ちはメッセージの遅延時間に依存する。これは、トランザクションは常に悲観的ロックを用いるとは限らないためである。

表 2.1: パラメタ

並列度	3[ノード]
メッセージ遅延	1-5[サイクル]
葉トランザクションの処理時間	15-25[サイクル]
深さ	1-5[レベル]
葉トランザクションの合計	100
トランザクションの合計	101〜
α	50[%]
β	50[%]

2.5.2 実験結果

各実験を各深さで 8 回行った。その結果を表 2, 図 2 に示す。今実験では、図 2 より明らかに子トランザクションが深くなるほど、コミットに至るまでの平均時間が急激に早くなっていることが確認できる。また、最小値や最大値を見ても平均時間が早くなっていることが明らかである。

また、平均との差を図 3 に示す。図 3 より、実験毎の差が大きく開かないことがわかる。

表 2.2: 実験結果 時間と深さ（平均、最小、最大）

深さ [level]	平均 [サイクル]	最小 [サイクル]	最大 [サイクル]
1	790.6	779	811
2	608.6	545	675
3	489.4	408	567
4	345.3	302	392
5	162.4	149	173

2.5.3 考察

葉トランザクションの数及び深くすると次のようなことが考えられる。トランザクションの生成、コミット・アボートの通知の遅延やサブトランザクション数の増大である。一般にトランザクション数の増大は性能を落とす遠因になるが、入れ子トランザクションでは逆に内部トランザクションがコミットしやすくなったと考えられる。例

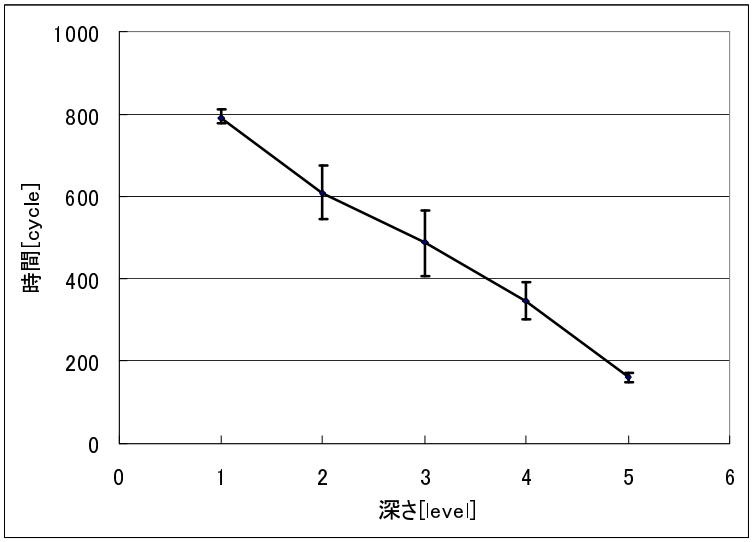

図 2.2: 実験結果 時間と深さ（平均、最小、最大）

えば、深さが 2 レベルで葉トランザクションが 16 個であれば、最低でも 4 個のトランザクションのコミットが必要だが、深さが 4 でであれば、一つのトランザクションがコミットしてれば親トランザクションはコミットできる。

また、図 2 に線形性が見て取れる。この線形の x 切片は 6.1[レベル] である。この値は、100 個の葉トランザクションを用いて作られる 2 文木の最大の深さと関係があると考えられる。今実験では、半分の子トランザクションがコミットしたときにコミットを行う。子トランザクションが 4 個であれば 2 個コミットした時点でコミットする。そして、子トランザクションが 3 個のときに、1 個コミットした場合は、50%の確率でコミットする。さらに、子トランザクションが 1 個のときは、子がコミットしていなくても 50%の確率でコミットする。このとき、メッセージ遅延等を無視すれば 0 サイクルでコミットする。任意のトランザクションがコミットしたとき、木全体で子トランザクションを 2 個か 1 個であり、このコミットは連鎖する。子トランザクションが一つになりはじめるのは、5.6 レベルからである。メッセージ遅延によりいくらか深くなり、直線近似により 6.1 レベルという値が出てきたと考えられる。

これらよりは、メッセージの遅延より、深くし内部トランザクションコミットしやすくなったほうが、トップレベルのコミットが早くなると結論付けられる。

2.6 結び

本稿では、サブトランザクションの有用性を示し、さらにシミュレーションにより葉トランザクション数が一定の元で、葉のレベルを深くすることによるコミットが非常に早くなることを示した。

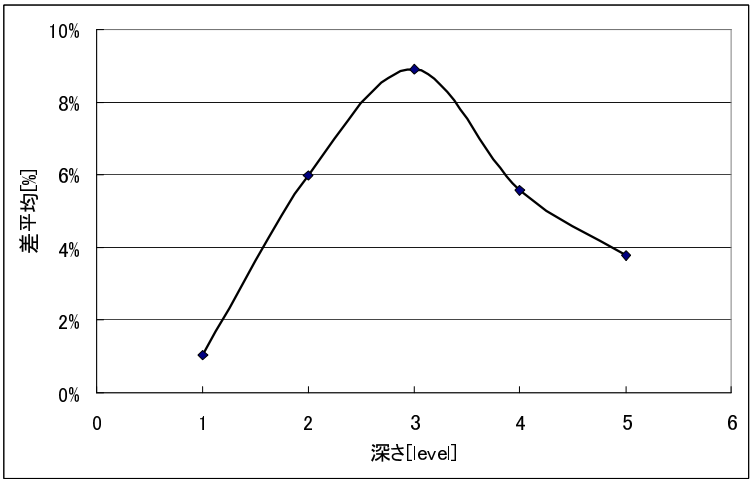

図 2.3: 実験結果 平均との差と深さ

第3章　分散ストレージ Hadoop Distributed File Systemの性能評価

3.1　前書き

今日、増え続ける情報の分析と保存の重要性がますます高まっている。我々は大容量向け高性能分散ファイルシステムである Hadoop Distributed File System(HDFS)[4] を用いその特性を示す。また、我々は Filesystem in Userspace(FUSE) を用い分散ストレージの性能評価を行う。Hadoop プロジェクトの基盤である HDFS は、GFS[5] の影響を受け巨大データ向け、高いスケーラビリティ、シングルマスターといった特徴を持つ分散ファイルシステムである。クライアントは、ネームノードに問い合わせ、データノードから実際のデータであるブロックを読み書きする。また、ネームノードとデータノードは協調し、ブロックを複製する。

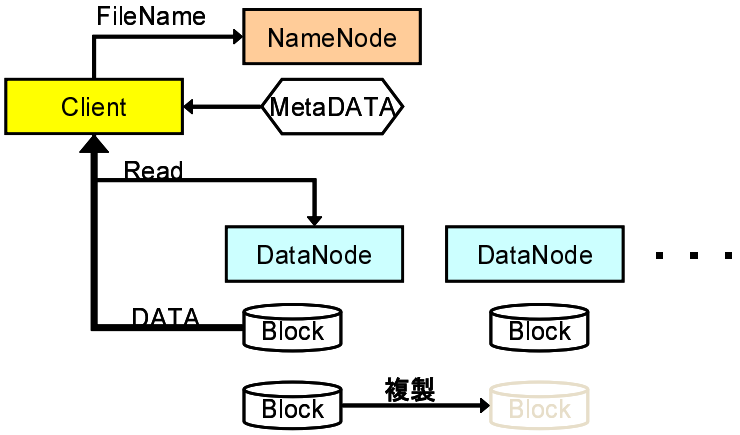

図 3.1: HDFS の構造

3.2 実験

FUSE を用いシステムの性能評価及びネームノードの影響を調べるため実験を行う。FUSE はユーザー空間でローカル同様のファイルシステムアクセスを提供する。しかし、VFS や libfuse によるオーバーヘッドによる遅延があり、メタデータの集中管理するネームノードへの影響はシステム全体に影響する。

また、同じ VFS を用いる NFS と比較検討するが NFS は分散ではないため、本実験では手動で複製を行い疑似分散 NFS とし実験を行う。合計 10GB の分散読み込みを行い最後のクライアントが終了した時間を実行時間として評価する。また、単一ボトルネックであるネームノードの CPU 時間、ネットワーク負荷を測定する。実験環境として、HP nc6400 を用いネームノード 1 台及びデータノード兼クライアント 4 台で構築し、Ubuntu Server 8.04、Hadoop 0.20.2、Sun Java SE1.6.0.20 を用いる。

HDFS はクライアント数が増加するにつれ減少するが、6 クライアントからは増減せず 64 クライアントでは微増している。これは、クライアントのアクセス集中が起きていると考えられる。また、ネームノードの CPU 時間はクライアント数が増加するにつれ、25%上昇するが低く抑えられている。

表 3.1: 実験結果

クライアント数	HDFS	NFS
1	993	983
2	652	992
5	560	1002
8	496	994
32	489	1124
64	500	1172

3.3 結論

本研究では、疑似分散 NFS と比べ HDFS の高い合計性能及びスケーラビリティを示し、HDFS の有用性を示した。

第4章　結論

本研究では、分散環境での処理性能をシミュレーション及び実環境で評価した。

まず、従来のトランザクションを拡張した入れ子トランザクションを用いた。入れ子トランザクションの有用性を示し、さらにシミュレーションにより葉トランザクション数が一定の元で、葉のレベルを深くすることによるコミットが、非常に早くなることを本実験によって示した。本手法を用いることで、システムのターンアラウンドタイムを削減できることを示した。

次に、分散システムのファイルシステムとしてHDFSを用いることにより、クライアントからのアクセス集中により64クライアント以上では処理性能が低下するが、疑似分散NFSと比べHDFSの高い合計性能を示した。また、ボトルネックであるネームノードのCPU時間はクライアント数が増加するにつれ、25%上昇するが低く抑えられており、スケーラビリティが高い。

本研究では、疑似分散NFSと比べHDFSの高い合計性能及びスケーラビリティを示し、HDFSの有用性を示した。

謝辞

本研究を遂行するにあたり、日頃より適切な御指導、御鞭撻をいただいた、法政大学工学部情報電気電子工学科 三浦孝夫教授に深く御礼申し上げます。

並びに、産能大学経営情報学科 塩谷勇教授にも多くの有益なご助言をいただきました。深く感謝いたします。

データ工学研究室の先輩方、同輩、後輩たちにも、本研究の遂行にあたって数多くの助言と快適で能率的な研究室環境を整えていただきました。御礼申し上げます。

修士論文として私の研究をまとめることができたのも、多くの皆様方の御支援、御協力の賜物であります。この場をお借りしまして、厚く御礼申し上げます。

最後に、今までの学生生活を支えてくださった私の両親に深く感謝したいと思います。

参考文献

[1] J, Eliot. and B, Moss.: “Nested Transactions: An Approach to Reliable Distributed Computing”, In *Department of Electrical Engineering and Computer Science*, 1981.

[2] Gray, J. and Reuter, A.: “Transaction Processing, Morgan Kaufmann”, 1993.

[3] Lynch, N., Merritt, M., Weihl, W. and Fekete, A.: “Atomic Transactions, Morgan Kaufmann”, 1994.

[4] Dhruba Borthaku.: “The hadoop distributed file system: Architecture and design”, 2007.

[5] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung.: “The Google File System”, 2003.