分散 database Jungle に関する研究

A Study of distributed database Jungle

平成25年度 学位論文(修士)

琉球大学大学院　理工学研究科
情報工学専攻

大城 信康

要旨

スマートフォンやタブレット端末の普及により, 大量の通信を扱うウェブサービスが現れてきている. しかしそれに伴い, サーバサイド側への負荷も増大しウェブサービスがダウンする事態が出てきている. そのため, スケーラビリティはウェブサービスにおいて重要な性質の 1 つとなっている. スケーラビリティとは, ある複数のノードから構成される分散ソフトウェアがあるとき, その分散ソフトウェアに対して単純にノード を追加するだけで性能を線形に上昇させることができる性質である. そこで, スケーラビリティを持たせるためにアーキテクチャの設計から考えることにした. 当研究室では非破壊的木構造を用いたデータベースである Jungle を開発している. 非破壊的木構造とは, データの編集の際に一度木構造として保存したデータを変更せず, 新しく木構造を作成してデータの編集を行うことを言う.

本研究では, Jungle に分散データベースと永続性の実装を行った. データ分散部分には当研究室で開発中である並列分散フレームワークである Alice を使用した. 結果, 学科の並列環境を用いて複数のサーバノード間でデータの分散を行うことを確認した. また, 例題アプリケーションとして簡易掲示板プログラムの作成を行った. Jungle と Cassandra により作成した掲示板プログラムに対して読み込みと書き込みの負荷をかけ評価を行った.

Abstract

Smartphone and tablet pc are widely used, thereby Web services that handle large amounts of data are emerging. It has caused the webserver is down. Therefore, scalability is important software factor today. Scalability in distributed system is able to increase performance linearly when just added new node to system. In order to make provide scalability, we considered design of architecture.

We are developing a database Jungle. It is use non-destructive tree structure. Non-destructive tree structure is not the destruction of data. Editing of data is done creating by new tree. Jungle was designed as a distributed database. But data distribution and persistent has not yet been implemented in the Jungle.

In this paper, we develop distributed database on jungle for pursuit architecture with scalability. Distributed data on Jungle is developing using parallel distributed framework Alice. As a result, we confirmed that data is distributed between the server node.

目 次

図 目 次

表 目 次

第1章　序論

1.1　序論

1.1.1　研究背景と目的

スマートフォンやタブレット端末の普及により, 大量の通信を扱うウェブサービスが現れてきている. しかしそれに伴い, サーバサイド側への負荷も増大しウェブサービスがダウンする事態が出てきている. そのため, スケーラビリティはウェブサービスにおいて重要な性質の 1 つとなっている. スケーラビリティとは, ある複数のノードから構成される分散ソフトウェアがあるとき, その分散ソフトウェアに対して単純にノード を追加するだけで性能を線形に上昇させることができる性質である. そこで, スケーラビリティを持たせるためにアーキテクチャの設計から考えることにした. 当研究室では非破壊的木構造を用いたデータベースである Jungle を開発している. 非破壊的木構造とは, データの編集の際に一度木構造として保存したデータには変更せず, 新しく木構造を作成してデータの編集を行うことを言う.

本研究では, Jungle に分散データベースと永続性の実装を行った. データ分散部分には当研究室で開発中である並列分散フレームワークである Alice を使用した. 結果, 学科の並列環境を用いて複数のサーバノード間でデータの分散を行うことを確認した.

1.1.2　本論文の構成

本論文では, 始めに分散データベースについて既存の製品を例に上げながら述べる. 第 3 章では, 非破壊的木構造による Jungle の基本設計と, 分散バージョン管理システムを参考にした分散設計について述べる. 第 4 章では, 第 3 章で行った設計を第 5 章では, 第 4 章で実装した分散データベース Jungle の評価を行うため, 簡易掲示板プログラムを実装する. この掲示板プログラムは Jungle と Cassanra それぞれのデータベースを使うものを用意した. 学科の並列環境上で開発した掲示板プログラムを複数のノードで実行させ, 負荷をかけることで Jungle と Cassandra の性能比較を行う. 第 6 章は, 本研究におけるまとめと今後の課題について述べる.

第2章　既存の分散データベース

本章ではまずデータベースの種類である RDB と NoSQL について述べる. 次に分散データシステムにおいて重要な CAP 定理について触れる. 最後に既存の NoSQL データベースとして Cassandra, MongoDB, Neo4j の特徴について述べる.

2.1　RDB と NoSQL

データベースは大別すると RDB と NoSQL に分けられる. RDB とは行と列からなる 2 次元のテーブルによりデータを保持するデータベースである. RDB はデータベースアクセス言語として SQL 言語を持ち, 一台のマシンでデータを扱う分には最適である. しかし, RDB はマシン単体以上の処理性能をだすことができない. そこで, 汎用的な PC をいくつも用意しデータや処理を分散して管理できるデータベースが求められた. それらのデータベースは NoSQL(Not Only SQL) と呼ばれる. 2 次元のテーブルでは無く, Key-Value, ドキュメント, グラフといった表現形式でデータの保持を行う. NoSQL は, SQL を使用するデータベースには向いていない処理を行うことを目的にしている.

2.2　CAP 定理

分散データシステムにおいて次の 3 つを同時に保証することはできない

- 一貫性 (Consistency) 全てのノードはクエリが同じならば同じデータを返す.
- 可用性 (Availability) あるノードに障害が発生しても機能しているノードにより常にデータの読み書きが行える.
- 分断耐性 (Partition-tolerance) ネットワーク障害によりノードの接続が切れてもデータベースは機能し続けることができる.

これは CAP 定理 [1] と呼ばれる. 利用するデータベース選ぶ場合, この CAP 定理を意識しなければならない. 一貫性と可用性を重視したい場合は RDB になる. 分断耐性を必要とする場合は NoSQL データベースとなる. だが NoSQL においても, 一貫性か可用性のどちらを保証しているかで用途が変わってくる.

分散データシステムを考える場合は, この CAP 定理を意識していなければならない.

2.3 Cassandra

Cassandra は 2008 年 7 月に Facebook によってオープンソースとして公開された Key-Value なデータベースである．Amazon の Dynamo という分散キーバリューデータベースの影響を受けて作られている．スキーマレスな NoSQL データベースになる．

Cassandra はサーバノードの配置にコンシステント・ハッシングアルゴリズムを用いる．コンシステント・ハッシングによりノードは論理的にリング上に配置される．リングには数値で表される位置がある．データを書き込む際には, キーとなるハッシュ値に従いそのリングの位置から時計回りに近いサーバノードへと書き込まれる．コンシステント・ハッシングを用いることで, ノードの数が増減した場合に, 再配置をしなくてもよいという利点がある．データの偏りにより少数のサーバへの負荷が大きい場合に, 負荷が高いハッシュ値が指すリング上に新たなノードを追加することで負荷を下げるといった手段もとれる．

データを最大どれだけ配置するかを示すレプリケーションファクタと, データの読み書きをいくつのノードから行うのかを決めるコンシステンシーレベルを設定できる．コンシステンシーレベルには主に ONE, QUORAM, ALL がある．レプリケーションファクタの数値を N とした場合, ONE は 1 つのノード, QUORUM は $N/2 + 1$ のノード, ALL は N のノードへと読み書きを行う．コンシステンシーハッシング, レプリケーションファクタとコンシステンシーレベルの設定により Cassandra は高い可用性と分断耐性を持つ．

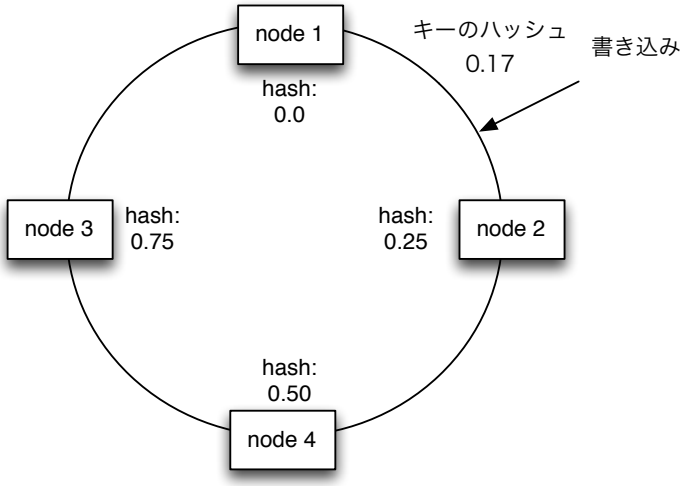

図 2.1: コンシステンシー・ハッシング

2.4 MongoDB

MongoDB は 2009 年に公開された NoSQL のデータベースである. JSON フォーマットのドキュメントデータベースであり, これはスキーマが無いリレーショナルテーブルに例えられる. スキーマが無いため, 事前にデータの定義を行う必要がない. そのためリレーショナルデータベースに比べてデータの追加・削除が行いやすい.

MongoDB は保存したデータを複数のサーバに複製をとる. これはレプリケーション (replication) と呼ばれる. また, 1 つのサーバが全てのデータを持つのでなく, ある範囲の値を別々のサーバに分割させて保持する. これをシャーディング (sharding) という. MongoDB はレプリケーションとシャーディングにより分断耐性と一貫性を持つ.

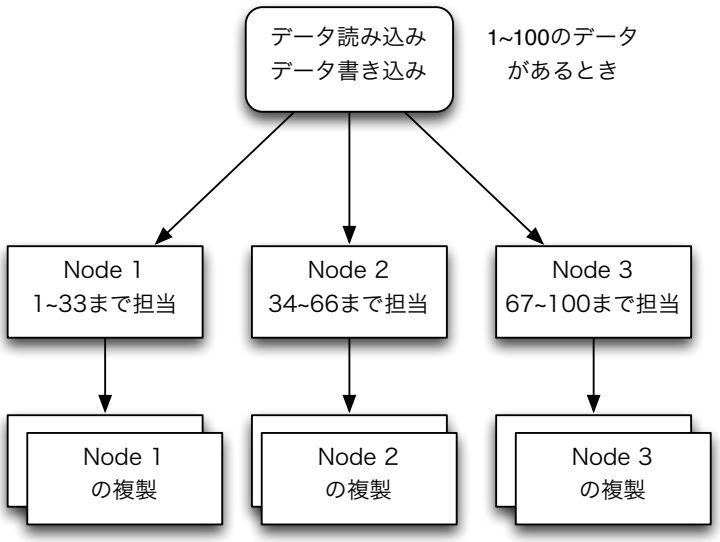

図 2.2: シャーディング

2.5 Neo4j

Neo4j は, グラフデータベースと呼ばれる NoSQL のデータベースである. データをグラフとして保存する. グラフはノードとリレーションシップにより表され, それぞれがプロパティを持つことができる. リレーションシップはグラフでいうところのエッジにあたる. ノードからリレーションシップを辿り, 各プロパティをみることでデータの取得を行うことができる. 通常データベースでは, データの取り出しに価の結合や条件の判定を行う. だが, グラフデータベースグラフはどれだけデータが大きくなろうがノードからノードへの移動は 1 ステップですむ. そのため, どれだけデータが大きくなろうと, データが小さい時と同じ計算量でデータの取得が行える.

Neo4j はマスターとスレーブの関係になるクラスタを構成することで分散データベースとして機能する. マスターに書かれたデータはスレーブに書き込まれるが, すぐに全てのスレーブに書き込まれるわけではない. したがってデータの整合性が失われる危険がある. スレーブサーバは現在保持しているデータを返すことができる. そのため Neo4j は高い読み取り性能の要求に答えることができる可用性と分断耐性を持つ.

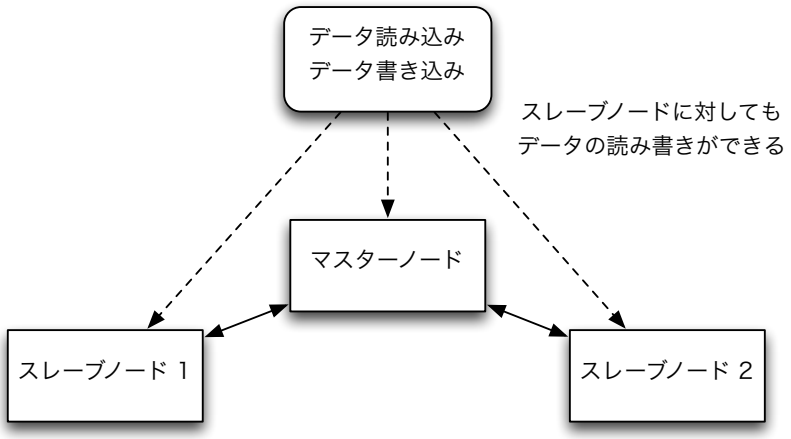

図 2.3: マスターとスレーブによるクラスタ

第3章　木構造データベースJungleの分散設計

Jungle はスケーラビリティのある CMS の開発を目指して当研究室で開発されている非破壊的木構造データベースである．一般的なコンテンツマネジメントシステムではブログツールや Wiki・SNS が多く, これらのウェブサイトの構造は大体が木構造であるため, データ構造として木構造を採用している．現在 Java と Haskell によりそれぞれ言語で開発されており本研究で扱うのは Java 版である．

本章ではまず破壊的木構造と, 非破壊的木構造の説明をし, Jungle におけるデータ分散の設計について述べる．

3.0.1　破壊的木構造

破壊的木構造の編集は, 木構造で保持しているデータを直接書き換えることで行う．図 3.1 は破壊的木構造の編集を表している．

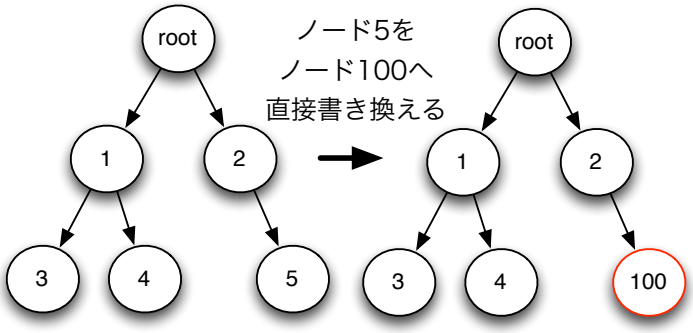

図 3.1: 破壊的木構造の編集

破壊的木構造は, 編集を行う際に木のロックを掛ける必要がある．この時, データを受け取ろうと木を走査するスレッドは書き換えの終了を待つ必要があり, 閲覧者がいる場合は木の走査が終わるまで書き換えをまたなければならない．これではロックによりスケーラビリティが損なわれてしまう．

3.0.2　非破壊的木構造

非破壊的木構造は破壊的木構造とは違い, 一度作成した木を破壊することはない. 非破壊的木構造においてデータの編集は, ルートから編集を行うノードまでコピーを行い新しく木構造を作成することで行われる. 図 3.2 は非破壊的木構造のデータ編集を示している.

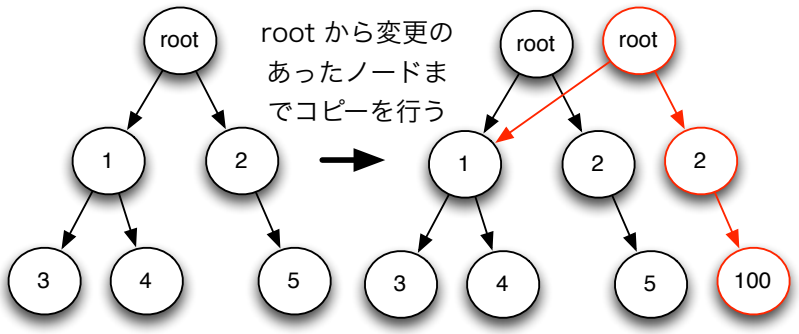

図 3.2: 非破壊的木構造の編集

非破壊的木構造におけるデータ編集の手順を以下に示す.

1. ルートから編集を行うノードまでのパスを調べる (図 3.3).
2. 編集を行うノードのコピーをとる. コピーをとったノードへデータの編集を行う (図 3.4).
3. 調べたパスに従いルートからコピーしたノードまでの間のノードのコピーをとり繋げる (図 3.5).
4. コピーしたルートノードは編集を行っていないノードへの参照を貼り新しい木構造を作る (図 3.6).

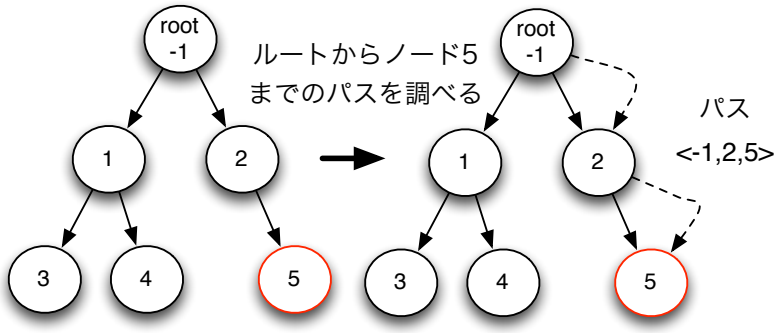

図 3.3: 非破壊的木構造の編集手順 1

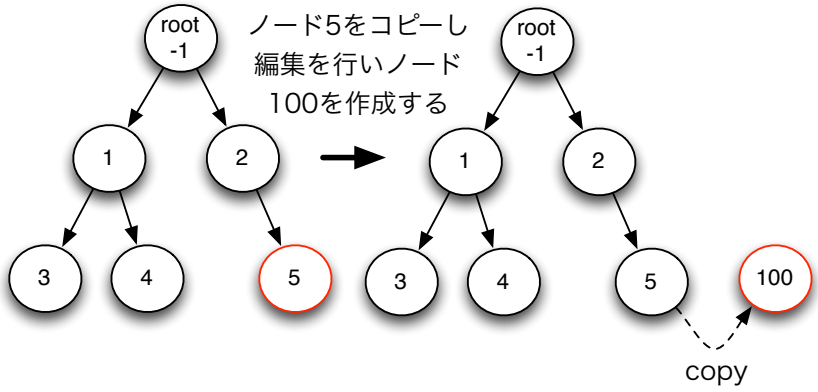

図 3.4: 非破壊的木構造の編集手順 2

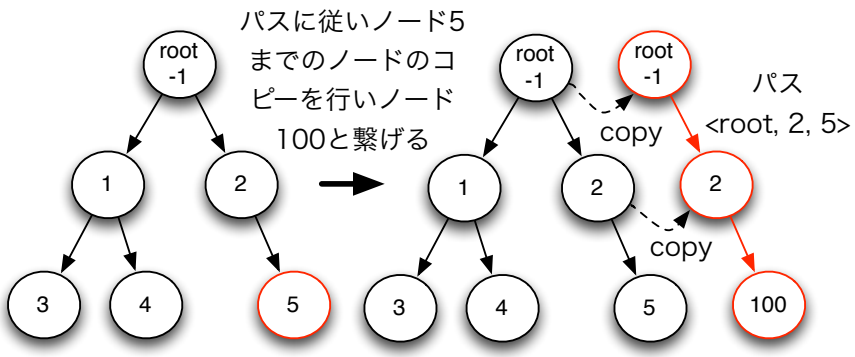

図 3.5: 非破壊的木構造の編集手順 3

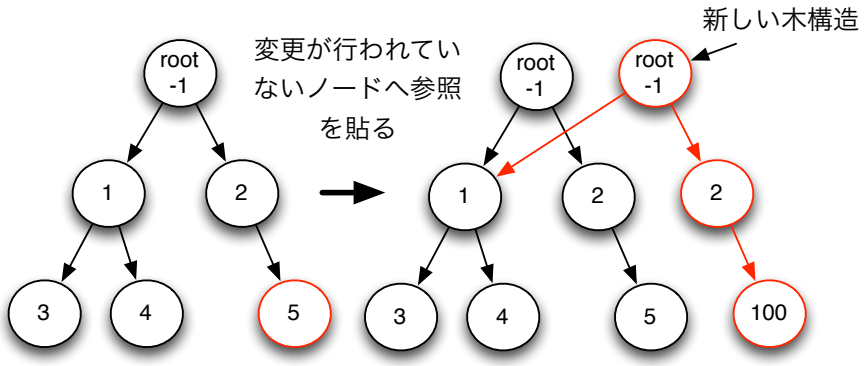

図 3.6: 非破壊的木構造の編集手順 4

非破壊的木構造においてデータのロックが必要となる部分は, 木のコピーを作終えた後にルートノードを更新するときだけである. データ編集を行っている間ロックが必要な破壊的木構造に比べ, 編集中においてもデータの読み込みが可能である (図 3.7). そのため, 破壊的木構造に比べスケールがしやすくなっている.

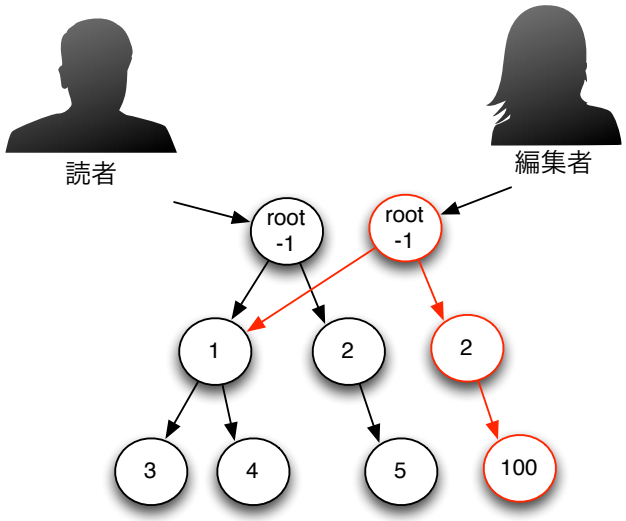

図 3.7: 非破壊的木構造による利点

3.1　Jungle におけるデータへのアクセス

Jungle ではデータをそれぞれの Node が attribute として保持する. attribute は String 型の Key と ByteBuffer の value のペアにより表される. Jungle でデータへのアクセス

は, この Node へのアクセスをさす. Node へのアクセスは, 木の名前と Node を指すパスにより行える. このパスは NodePath と呼ばれる (図 3.8).

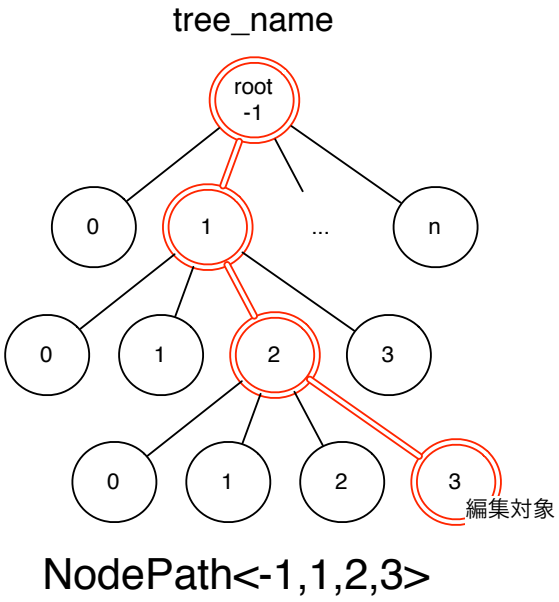

図 3.8: Node の attribute と NodePath

3.2　Jungle におけるデータ編集

3.2.1　NodeOperation

Jungle による最小のデータ編集は Node の編集を指す. Node 編集のために API が用意されており, この API は NodeOperation と呼ばれる. NodeOperation には次の 4 つの API が用意されている.

- `addNewChild(NodePath _path, int _pos)` NodePath で指定された Node に子供となる Node を追加する API である. pos で指定された番号に子供として追加を行う.
- `deleteChildAt(NodePath _path, int _pos)` NodePath と pos により指定される Node を削除する API である.
- `putAttribute(NodePath _path, String _key, ByteBuffer _value)` Node に attribute を追加する API である. NodePath は attribute を追加する Node を指す.
- `deleteAttribute(NodePath _path, String _key)` _key が示す attribute の削除を行う API である. NodePath は Node を示す.

NodeOperation はあくまで最小のデータ編集の単位である．アプリケーションレベルの実装にもよるが, Jungle によるデータの編集は NodeOperation が複数集まった単位によって行われる．この複数の NodeOperation の集まりを TreeOperationLog という．

3.2.2　TreeOperationLog

Jungle 内部では NodeOperation は順次ログに積まれていき, 最終的に commit されることで編集が完了する．この時, ログに積まれた複数の NodeOperation は TreeOperationLog として扱われる．以下に TreeOperationLog の具体的な例を示す (3.1).

Listing 3.1: トポロジーマネージャーの利用

```
1 [APPEND_CHILD:<−1>:pos:0]
2 [PUT_ATTRIBUTE:<−1,0>:key:author,value:oshiro]
3 [PUT_ATTRIBUTE:<−1,0>:key:mes,value:hello]
4 [PUT_ATTRIBUTE:<−1,0>:key:timestamp,value:0]
```

このログは今回の研究で使用したベンチマーク用掲示板プログラムにおける書き込みにより行われるログである (図 3.9).

大文字の英字は実行した NodeOperation の種類を表す．<> により囲まれている数字は NodePath を示す．NodePath の表記以降は Node の position や attribute の情報を表している．

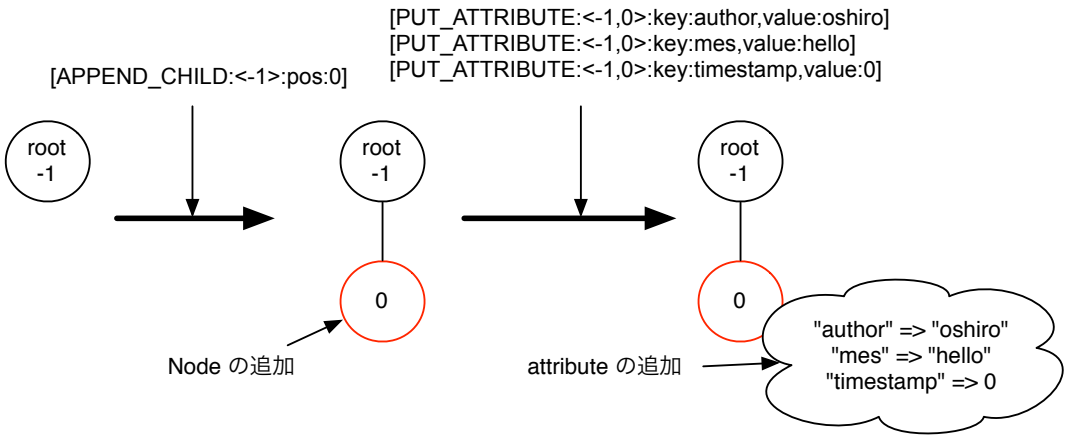

図 3.9: TreeOperationLog の具体例

図 3.9 の説明を行う．まず, `APPEND_CHILD` により Root Node の 0 番目の子供となる Node の追加を行う．次に, 追加を行った Node に対して `PUT_ATTRIBUTE` により attribute の情報を持たせていく．attribute の内容に作者の情報を表す auther, メッセージの内容を表す mes, そしてタイムスタンプを timestamp とそれぞれキーにすることで追加される．

以上が掲示板プログラムにおける 1 つの書き込みで発生する TreeOperationLog である．

3.3　分散バージョン管理システムによるデータの分散

Jungle は Git や Mercurial といった分散バージョン管理システムの機能を参考に作られている. 分散バージョン管理システムとは, 多人数によるソフトウェア開発において変更履歴を管理するシステムである. 分散管理システムでは開発者それぞれがローカルにリポジトリのクローンを持ち, 開発はこのリポジトリを通すことで進められる (図 3.10). ローカルのリポジトリは独立に損刺し, サーバ上にあるリポジトリや他人のリポジトリで行われた変更履歴を取り込みアップデートにかけることができる. また逆に, ローカルのリポジトリに開発者自身がかけたアップデートを他のリポジトリへと反映させることもできる. 分散管理システムでは, どれかリポジトリが壊れたとしても, 別のリポジトリからクローンを行うことができる. ネットワークに障害が発生しても, ローカルにある編集履歴をネットワーク復旧後に伝えることができる. そのため, 可用性と分断耐性が高いと言える.

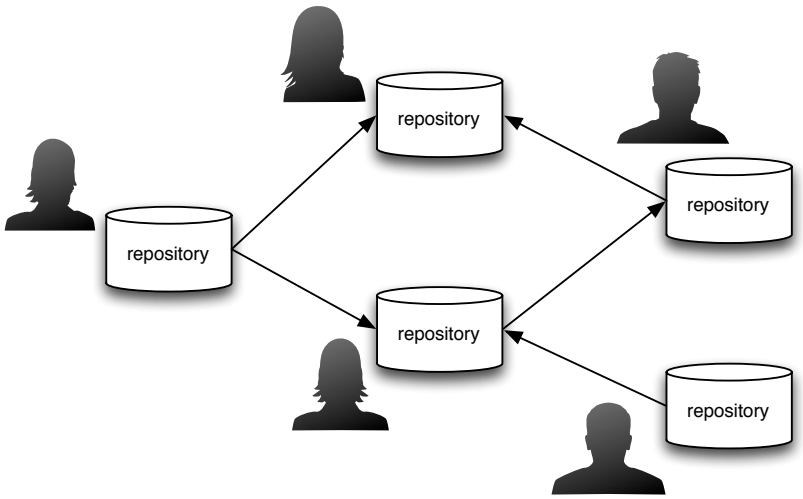

図 3.10: 分散バージョン管理システム

3.3.1　マージによるデータ変更衝突の解決

分散管理システムでは, データの更新時において衝突が発生する時がある. それは, 分散管理システムを参考にしている Jungle においても起こる問題である. データの変更を行うときには, 元のデータに編集が加えられている状態かもしれない. Jungle はリクエストがきた場合, 現在もっているデータを返す. そのためデータは最新のものであるかは保証されない. この場合, 古いデータに編集が加えられ, それを更に最新のデータへ伝搬させなければならない. このように他のリポジトリにより先にデータ編集が行われており, データの伝搬が素直にできない状態を衝突という. この衝突を解決する手段が必要である. 分散管理システムでは衝突に対してマージと呼ばれる作業で解決をはかる. マージは, 相手

のリポジトリのデータ編集履歴を受け取り, ローカルにあるリポジトリの編集と合わせる作業である. データ衝突に対して Jungle はアプリケーションレベルでのマージを実装して貰うことで解決をはかる.

以下にマージが必要な場合とそうでない場合のデータ編集についての図を示す (図??,3.12,3.13).

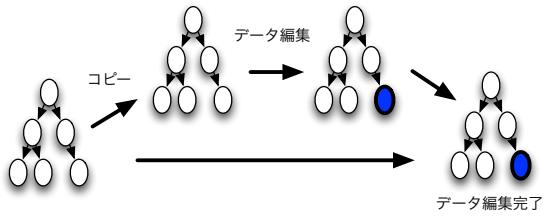

図 3.11: 衝突の発生しないデータ編集

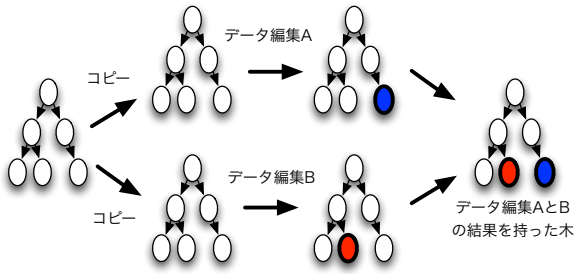

図 3.12: 自然に衝突を解決できるデータ編集

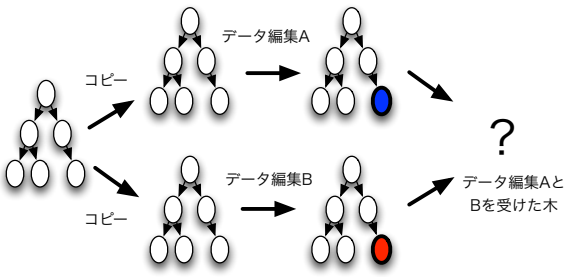

図 3.13: 衝突が発生するデータ編集

3.4　ネットワークトポロジーの形成

分散管理システムを参考に Jungle でもそれぞれのデータベースが独立に動くようにしたい. そのために必要なことはトポロジーの形成と, サーバノード間でのデータアクセス機構である. また, データ分散のために形成したトポロジー上で扱うデータを決めなければならない.

3.4.1　ツリートポロジーの形成

分散データーベス Jungle で形成されるネットワークトポロジーはツリー構造を想定している. ツリー構造ならば, データの整合性をとる場合, 一度トップまでデータを伝搬させることで行える. トップもしくはトップまでの間にあるサーバノードでデータ伝搬中に衝突が発生したらマージを行い, マージの結果を改めて伝搬すればよいからである. また, リング型, スター型, メッシュ側ではデータ編集の結果を他サーバノードに流すとき流したデータが自分自身にくることにより発生するループに気をつける必要がある. ツリー構造の場合は, サーバノード同士の繋がりで閉路が無い. そのため, 自分自身が行ったデータ編集の履歴を繋がっているノードに送信するだけですむ. このルーティングの方式はスプリットホライズンと呼ばれるものである.

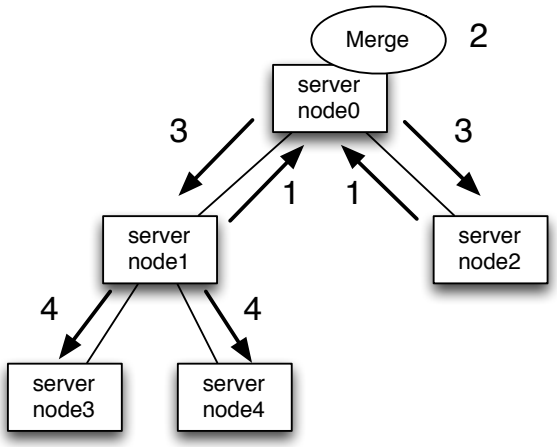

図 3.14: ツリー型の Network Topology

3.4.2　トポロジーの形成手段

Jungle で使用するネットワークトポロジーはツリー型を考えているが, リング型やメッシュ型といった他のネットワークトポロジーによる実装に関しても試す余地はある. そのため, ツリーだけでなく, 自由にネットワークトポロジーの形成を行えるようにしたい.

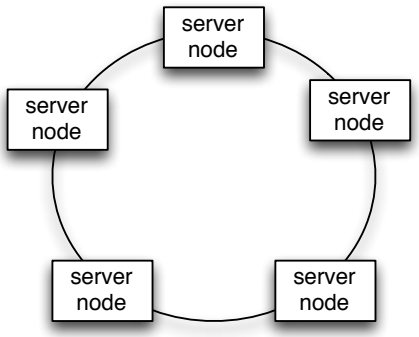

図 3.15: リング型のトポロジー

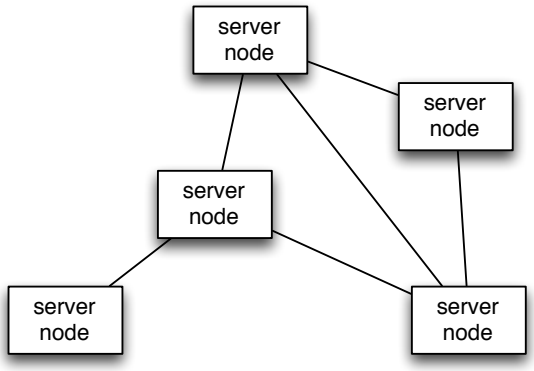

図 3.16: メッシュ型のトポロジー

そこで当研究室で開発を行っている並列分散フレームワークである Alice を使用する. Alice はユーザが望んだマシンへの接続や必要なデータへのアクセスを行う機構と, 接続トポロジー形成機能を提供している.

3.5　並列分散フレームワーク Alice

Alice は当研究室で開発している並列分散フレームワークである. Alice はデータを DataSegment, タスクを CodeSegment という単位で扱うプログラミングを提供している. コードの部分となる CodeSegment は, 計算に必要なデータである DataSegment が揃い次第実行が行われる (図3.17). CodeSegment の結果により出力される新たなデータでは, 別の CodeSegment が実行されるための DataSegment となる. DataSegment と CodeSegment の組み合わせにより並列・分散プログラミングの依存関係が表される.

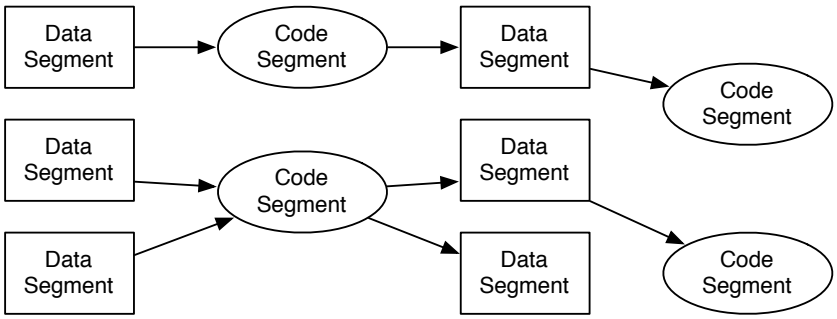

図 3.17: DataSegment と CodeSegment によるプログラムの流れ

3.5.1　MessagePack によるシリアライズ

Alice では DataSegment のデータ表現に MessagePack(http://msgpack.org) を利用している. MessagePack はオブジェクトをバイナリへと変換させるシリアライズライブラリである. Alice によりネットワークを介してデータにアクセスするときは, そのデータが MessagePack でシリアライズが行えることが条件である.

3.6　Jungle のデータ分散

Alice によりトポロジーの形成とデータアクセスの機構が提供された. 後はデータ分散の為にどのデータをネットワークに流すのか決めなければならない. そこで選ばれたのが TreeOperationLog である. TreeOperationLog はデータ編集の履歴になる. どの Node にどのような操作をしたのかという情報が入っている. この TreeOperationLog を Alice を使って他サーバノードに送り, データの編集をしてもらうことで同じデータを持つことが可能となる. Alice を用いるため, この TreeOperationLog は MessagePack によりシリアライズ可能な形にすることが必要である.

3.6.1　CAP 定理と Jungle

ここまでの Jungle の設計を踏まえて, CAP 定理における Jungle の立ち位置を考える. 分散管理バージョンのように独立したリポジトリもち, それぞれが独自の変更を加えることが行えることで一貫性はゆるい. だが, ネットワークから切断されてもローカルで行ったデータの変更をネットワーク復旧後で伝搬できることと, リクエストに対し持っているデータをすぐに返すことができる. つまり Jungle は可用性と分断耐性に優れたデータベースを目指している. 第 2 章で紹介した既存のデータベースと Jungle との CAP 定理の関係を図 3.18 に示す.

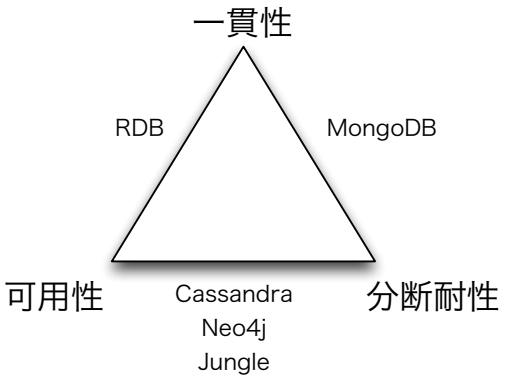

図 3.18: CAP 定理における各データベースの立ち位置

3.7　Jungle データの永続性

Jungle は非破壊でさらにオンメモリにデータを保持するため, 使用するメモリの容量が大きくなる.

第4章　Jungleの分散実装

4.1　TreeOperationLogを用いての分散実装

Jungle でデータ扱うと TreeOperationLog として残ることは述べた．この TreeOperationLog を他のサーバへと送り, Jungle の編集を行って貰うことでデータの分散を行うことができる．ここで問題になることはネットワークトポロジーの形成方法であった．

4.2　Alice のトポロジーマネージャーの利用

Alice はサーバノード同士によるネットワークトポロジー形成の機能を持つ．トポロジーマネージャーの起動は 4.2 の様にポート番号の指定と dot ファイルを引数として渡すことで行う．(4.1).

Listing 4.1: Alice によるネットワークトポロジーマネージャーの起動

```
% java −cp Alice.jar alice.topology.manager.TopologyManager −p 10000 −conf ./topology/tree5.dot
```

ポート番号は Alice により記述された並列分散プログラムの起動時に渡す必要がある．dot ファイルには, トポロジーをどのように形成するかが書かれている．以下に, サーバノード数 5 で, 2 分木ツリー構造を形成する dot ファイルの例を示す (4.2).

Listing 4.2: ネットワークトポロジー設定用 dot ファイル

```
% cat tree5.dot
digraph test {
   node0 −> node1 [label="child1"]
   node0 −> node2 [label="child2"]
   node1 −> node0 [label="parent"]
   node1 −> node3 [label="child1"]
   node1 −> node4 [label="child2"]
   node2 −> node0 [label="parent"]
   node3 −> node1 [label="parent"]
   node4 −> node1 [label="parent"]
}
```

node0 や node1 はサーバノードの名前を示す．サーバノードの間にはラベルがあり, Alice 上ではこのラベルに指定される文字列 (キー) を使うことで他のサーバノードのデータへアクセスすることができる．node0 -¿ node1 はサーバノード同士の繋がりを示している．次に続く label=”child1” は, node0 が node1 のデータに”child1”という文字列を使うことでアクセスできることを示す．

dot ファイルを読み込んだ Alice のトポロジーマネージャーに対して, サーバノードは誰に接続を行えばよいかを訪ねる. トポロジーマネージャーは訪ねてきたサーバノードに対してノード番号を割り振り, dot ファイルに記述している通りにサーバノード同士が接続を行うよう指示をだす.

トポロジーマネージャーは接続要求先を聞いてくるサーバノードに対して名前を割り振り, 接続相手を伝える. dot ファイル 4.2 により形成されるトポロジーを図 4.1 に示す.

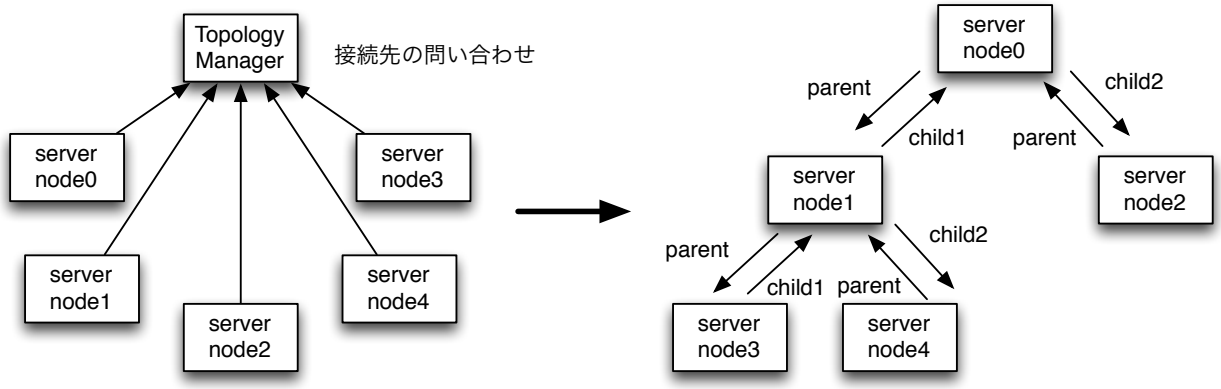

図 4.1: Alice によるネットワークトポロジー形成

矢印に書かれている文字列は, 相手のデータにアクセスするキーを示す. ”child1”, ”child2”, ”parent” というキーを使うことで別のサーバノードにあるデータを取得することができる.

トポロジーマネージャーに最初に接続要求を行う並列分散プログラム側は, 次のように記述する (4.3)

Listing 4.3: Alice を使用してのトポロジー形成

```
1 public static void main( String[] args ) throws Exception
2 {
3   RemoteConfig conf = new RemoteConfig(args);
4   new TopologyNode(conf, new StartBBSCodeSegment(args, conf.bbsPort));
5 }
```

そして, プログラムの起動時にはトポロジーマネージャーが動いているサーバのドメインとポート番号を渡すことでトポロジーの形成が行われれプログラムの処理がはしる. 例えば, mass00.cs.ie.u-ryukyu.ac.jp というサーバ上でポート番号 10000 を指定してトポロジーマネージャーを起動した場合は次のようになる (4.4).

Listing 4.4: トポロジーマネージャーの利用

```
1 % java Program −host mass00.cs.ie.u−ryukyu.ac.jp −port 10000
```

4.3 Alice を用いての分散実装

形成されたトポロジー上でのデータの送受信を行う部分について述べる.

4.4 ログのシリアライズ

ここでログのシリアライズについて述べる.

シリアライズとは, データをネットワーク上に流しても良い形式に変換することである.

4.5　掲示板プログラムにおけるマージの実装

Jungle に分散実装を行った後の問題としてデータ衝突がある．他のサーバノードから送られてくるデータが既に手元で変更を加えた木構造を対象とした場合に発生する問題である．Jungle ではこれをアプリケーション毎にマージを実装することで解決させる．

今回分散実装を行い, 例題として掲示板プログラムを用意した．掲示板プログラムに実装を行ったマージについて述べる．まず Jungle を用いた掲示板プログラムのデータ保持方法を図 4.2 に示す．

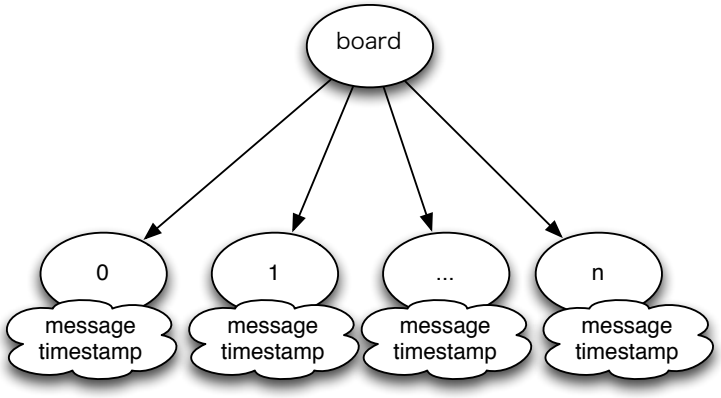

図 4.2: Jungle による掲示板プログラムのデータ保持方法

掲示板プログラムでは各掲示板毎に 1 つの木構造が作成される．掲示板への 1 つの書き込みは子ノードを 1 つ追加することに相当する．また, 各子ノードは attributes として書き込みの内容である message と書き込まれた時間を表す timestamp を保持している．先に追加された順で子ノードには若い番号が割り振られる．

他サーバノードからの書き込みをそのまま子ノードの後ろに追加してしまうと, データの整合性が崩れてしまう．この時の状態を表しているのが図 4.3 と 4.4 になる．

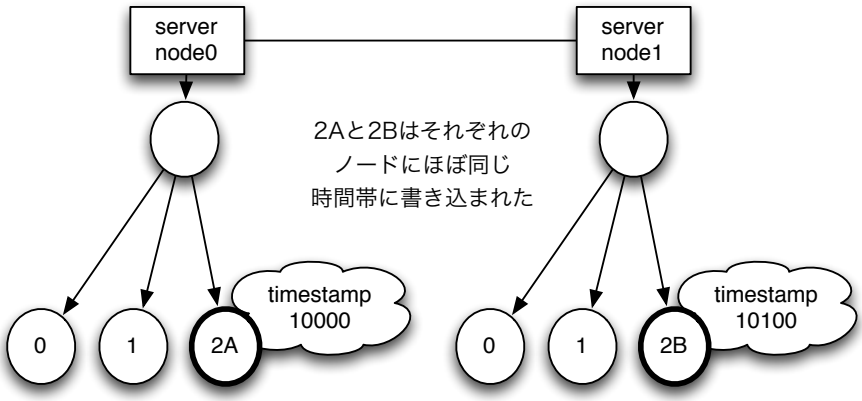

図 4.3: 他サーバノードの編集データ反映による整合性の崩れ 1

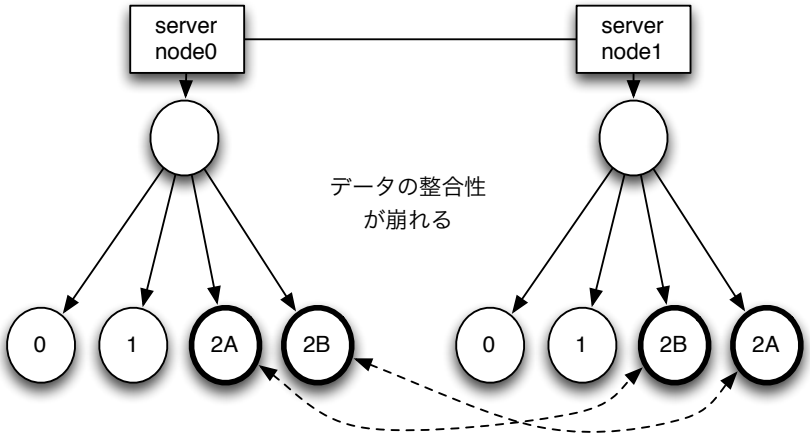

図 4.4: 他サーバノードの編集データ反映による整合性の崩れ 2

図 4.4 の server node0 の木の状態にするのが理想である. 掲示板のへの書き込みの表示は, 書き込みされた時間が早い順に表示されるようにしたい. これを timestamp を利用することで行う. 他サーバノードから来たデータに関しては, timestamp を参照し, 次に自分の保持している木の子ノードの timestamp と比べていくことでデータの追加する場所を決める. これが今回実装を行った掲示板システムにおけるマージになる.

第5章　分散木構造データーベース Jungleの評価

5.1　実験方法

5.2　実験環境

5.3　実験結果

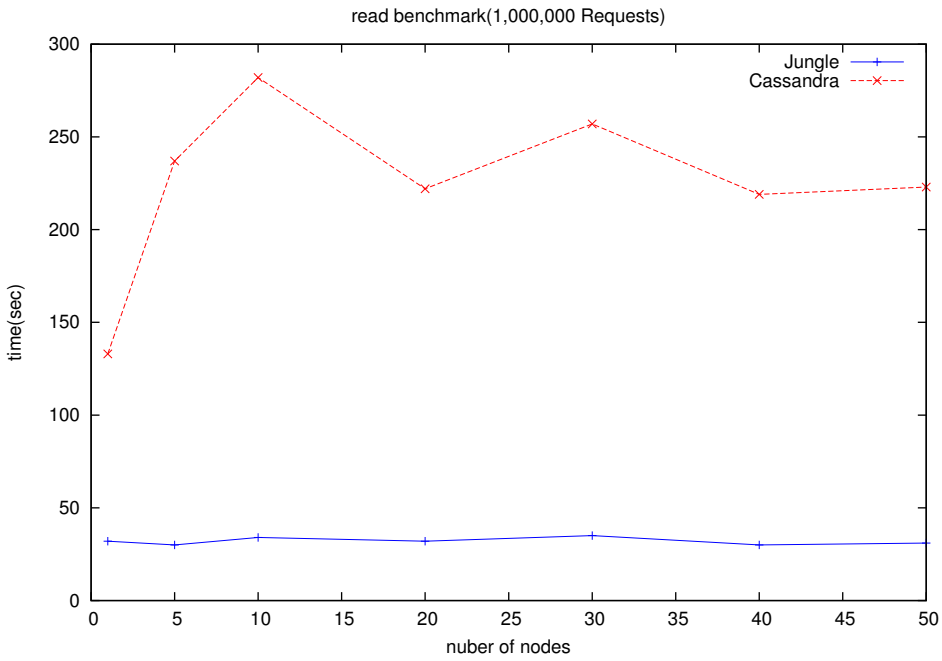

図 5.1: 読み込みベンチマーク結果

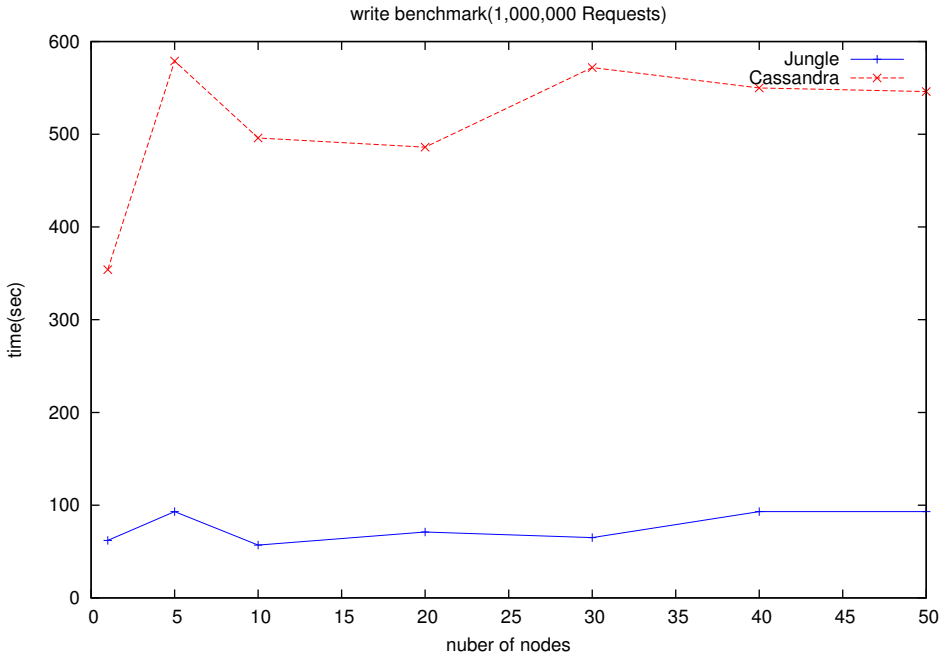

図 5.2: 書き込みベンチマーク結果

第6章　結論

6.1　まとめ

6.2　今後の課題

6.2.1　データ分割の実装

6.2.2　Merger アルゴリズムの設計

6.2.3　Compaction の実装・分断耐性の実装

謝辞

本研究を行うにあたり, ご多忙にも関わらず日頃より多くの助言, ご指導をいただきました河野真治助教授に心より感謝いたします.
また, 様々な研究や勉強の機会を与えてくださった, 株式会社 Symphony の永山辰巳さん, 同じく様々な助言を頂いた森田育宏さんに感謝いたします. 様々な研究に関わることで自身の研究にも役立てることが出来ました.
研究を行うにあたり, 並列計算環境の調整, 意見, 実装に協力いただいた谷成 雄さん, 杉本優さん, 並びに並列信頼研究室の全てのメンバーに感謝いたします.
最後に, 大学の修士まで支えてくれた家族に深く感謝します.

参考文献

[1] Nancy Lynch and Seth Gilbert. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 2002.

[2] 玉城将士, 河野真治. Cassandra を使った cms の pc クラスタを使ったスケーラビリティの検証. 日本ソフトウェア科学会, August 2010.

[3] 玉城将士, 河野真治. Cassandra を使ったスケーラビリティのある cms の設計. 情報処理学会, March 2011.

[4] 玉城将士, 河野真治. Cassandra と非破壊的構造を用いた cms のスケーラビリティ検証環境の構築. 日本ソフトウェア科学会, August 2011.

[5] Avinash Lakshman and Prashant Malik. Cassandra - a decentralized structured storage system. *LADIS*, Mar 2003.

[6] Fay Changand Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable : A distributed storege system for structured data.

[7] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly avaliable key-value store.

[8] Matt Welsh. The staged event-driven architecture for highly-concurrent server applications.

[9] Eric Brewer Matt Welsh, David Culler. Seda : An architecture for well-conditioned , scalable internet services. *SOSP*.

発表履歴

- Java による授業向け画面共有システムの設計と実装, 大城信康, 谷成雄 (琉球大学), 河野真治 (琉球大学), オープンソースカンファレンス 2011 Okinawa, Sep, 2011
- Continuation based C の GCC 4.6 上の実装について, 大城信康, 河野真治 (琉球大学), 第 53 回プログラミング・シンポジウム, Jan, 2012
- GraphDB 入門 TinkerPop の使い方, 大城信康, 玉城将士 (琉球大学), 第 15 回 Java Kuche, Sep, 2012
- ディペンダブルシステムのための木構造を用いた合意形成データベースの提案と実装, 大城信康, 河野真治 (琉球大学), 玉城将士 (琉球大学), 永山 辰巳 (株式会社 Symphony), 情報処理学会システムソフトウェアとオペレーティング・システム研究会 (OS), May, 2013
- Data Segment の分散データベースへの応用, 大城信康, 杉本優 (琉球大学), 河野真治 (琉球大学), 日本ソフトウェア科学会 30 回大会 (2013 年度) 講演論文集, Sep, 2013