

3 クラウドコンピューティングテクノロジー

3.1 はじめに

クラウドコンピューティングにより、コンピューティングリソースを所有することから利用することへと、パラダイムが移り変わると言われています。IJでは、莫大なデータの処理や、効率的なインフラの実現のため、数年前から、クラウドコンピューティングを支える技術を開発、運用しています。

ここでは、IJが独自に開発・実装し、サービス基盤で利用している、クラウド技術基盤の一つである、「ddd」と呼ばれる分散システムについて説明します。

可用性を増大させることです。単純にコンピュータを並べるだけでなく、それらを協調動作させる技術が必要になります。

莫大なデータの保持や処理を、効率的に行うことが期待されているクラウドには、必然的に大量のデータが集まります。また、求められる可用性もますますシビアなものになってきています。クラウドコンピューティングにとって、大規模な分散システムは極めて重要な要素となりつつあります。

3.2 分散システムとは?

dddについて説明する前に、ここで説明する分散システムを定義しておきます。分散システムの定義は、(1)複数のコンピュータノードで構成されていること、(2)それを利用するユーザからは単一のシステムとして見えること、の2つです。

分散システムの例としては、巨大なデータを保持する分散ストレージや、分散データ処理などが挙げられます。(図-1、図-2)

分散システムの目的は、複数台のコンピュータノード(以下ノード)を用いることにより、一台のコンピュータよりもシステムとしての処理能力を向上させたり、

3.3 分散システムの実例

分散システムは現在活発に技術開発が続けられている分野です。いくつか有名な例を紹介します。

- Google File System (GFS)
巨大で大量のデータを扱うためにGoogleで作られた分散ファイルシステム。実装は、Google外部には公開されていない。(http://labs.google.com/papers/gfs.html)
- MapReduce
Googleで考案された大規模分散処理フレームワーク(詳細は後述)。(http://labs.google.com/papers/mapreduce.html)
- Amazon Dynamo
Amazonが開発した、分散キーバリューストア(key-

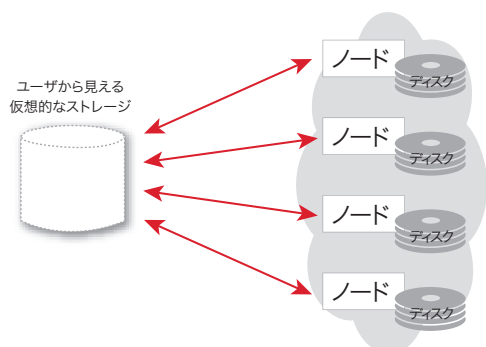


図-1 分散ストレージ

データを複数ノードで分散保持する。ユーザからはひとつのストレージに見える。

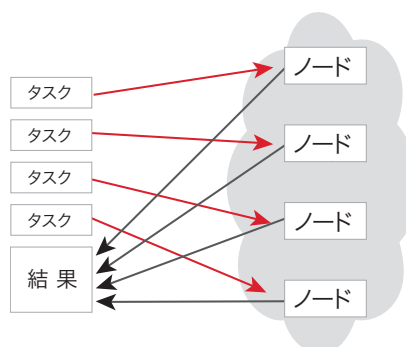


図-2 分散データ処理

データ処理内容をたくさんのタスクに分割し、複数ノードで並列に処理する。

value store)。これも実装は公開されていない。
(http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html)

- Hadoop

上記GFSとMapReduceを参考に作られた分散システム。Javaで記述され、オープンソースとして公開されている。(http://hadoop.apache.org/)

これらの中で、Google File SystemやAmazon Dynamoは分散ストレージに、MapReduceは分散データ処理技術に分類されます。IIJが開発したdddは、その両方の機能を併せ持ったものになっています。

3.4 ddd開発の経緯

dddはdistributed database daemonの略称で、IIJバックボーンを流れる膨大なトラフィックを解析するために作られました。

インターネットサービスプロバイダ (ISP) は、自社のバックボーンを安定運用するために、バックボーンルータのトラフィック情報を取得し、解析しています。多くの場合は、ルータの各インタフェースのカウント値を、SNMPで取得してグラフ化するということが行われていますが、インタフェースの単純なIN/OUT情報だけではトラフィックの実態が分からないため、監視や分析をする上で不十分なことがあります。

そこで利用されるのが、フロー情報と呼ばれる、より詳細なトラフィック情報です。ISPのバックボーンで使われているハイエンドなルータやスイッチは、パケットの送信元及び送信先のIPアドレスや、ポート番号等が含まれるフロー統計情報を出力できるものがあり、それを

受信して解析することにより、SNMPでは把握できない、詳細な通信状況を監視することができるようになります。

例えば、図-3は、フロー情報を元にトラフィックの宛先となるAS番号別に色分けして表示したグラフです。右端近辺でトラフィックが急に落ち込んでいる部分がありますが、フロー情報を用いて解析すると、一部のAS宛での通信のみが減少していることがわかります。SNMPでは総量しか把握が出来ないので、異常や状況の変化の原因特定が難しいような場合でも、フロー情報を用いると詳細な解析が可能になります。

フロー情報を解析するに当たっての問題点は、そのデータ量が膨大であるということです。細かい通信内容を分析できる反面、必然的にデータ量が多くなってしまいます。一般的には、フロー情報を受信した直後に、ある程度の集計処理を行って、データ量を減らしてから格納することが多いようです。例えば、送信元アドレス単位で集計して、合計値のみを格納するというやり方です。しかし、集計した値では後で細かく解析することができなくなるため、IIJではフロー情報をほぼ全て保持し、必要になった段階で抽出や集計処理を行うようにしています。このようなやり方では、IIJのように多くのネットワーク機器を運用しているISPでは、極めて大きなデータを取り扱わなければなりません。

以前、IIJでは、フロー情報を一般的なリレーショナルデータベースに格納していました。しかし、フロー情報は、一つひとつのレコードは小さいものの、5分あたり数十万から数百万レコードに及ぶこともあり、従来のデータベースではごく短期間の情報しか保持できません。

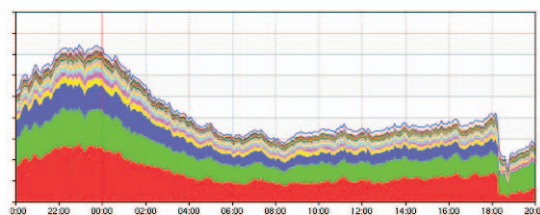


図-3 AS番号別に色分けしたトラフィックグラフ

そこで、長期間にわたる膨大なフロー情報を扱うため、IJでは独自の分散システムを開発することとしました。それがdddです。数百億超の大量のフロー情報レコードを保持し、様々な条件で高速にデータを抽出したり、集計したりすることができます。

なお、dddは、フロー情報を扱うことを第一の目的として開発が始まりましたが、現在ではセキュリティサービスやアプリケーションサービスのログ解析、内外のセキュリティ情報の分析等、より広範なデータの処理に活用しています。

3.5 dddの概要

dddは、IJ社内で使われている分散システムで、低コストなPCを利用しながら、高いスケーラビリティを実現した分散システムです。

dddには次のような特徴があります。

- pure P2Pによる、単一障害点を持たない構成
- データの自動冗長化機能を備え、動的に拡張可能な分散ストレージ
- MapReduceによる高速なデータ処理

dddの各ノードは以下のような3層構造になっています。全てのノードは同一の構造をしています。(図-4)

各項目について以下、順に説明していきます。

3.5.1 pure P2P

dddの各ノードは、中央管理ホストを持たないpure P2Pのネットワークを構成します。中央管理ホストが無いいため単一障害点もありません。すべてのノードは対等で、それぞれが協調して動作します。後述のデータ冗長化機能と併せて、いつ、どのノードが故障してもシステム全体としては問題なく稼働します。

新規にノードを追加する場合は、既存のどれか1つのノードに接続して起動します。新規ノードは、接続した既存ノードから他のノードの情報(IPアドレスなど)を取得します。同時に、新規ノードの情報は他の既存のノードに広報されていきます。

dddは、起動している間ずっと他のノードと情報を数秒おきに交換しあっています。よって、各ノードは他のすべてのノードを情報を知っていることになります。このように相互に情報を共有しているノードの一群をクラスタと呼びます。

3.5.2 分散ストレージ

■キーバリューストア

dddのストレージ部分は、分散キーバリューストア(key-value store)と呼ばれるものに分類されます。キーバリューストアとは、データをキーとバリューの組み合わせで格納する、シンプルなデータベースの一種で、それを複数のノードで分担して管理するのが分散キーバリューストアです。

キーバリューストアは、従来のリレーショナルデータベース(RDB)に比べ機能が限られ、データの結合機能(JOIN)も、トランザクションによる一貫性の保持の機構もなく、基本的にはキーを指定して、対応する値を読み書きすることのみが可能で。

機能が限られるかわりに、キーの値によって担当するノードを振り分けて分散化するのに適しており、スケー

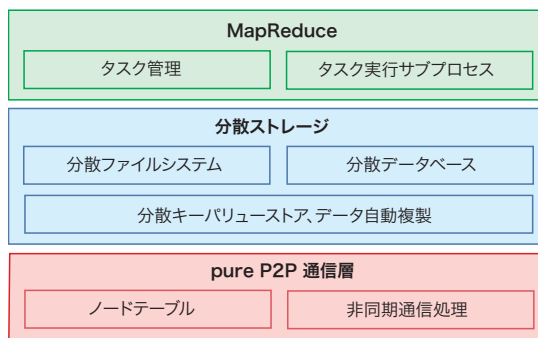


図-4 ddd内部構造概略

ラビリティを高めることができるため、クラウド時代のデータストアと言われることもあります。

キーに応じて担当ノードを振り分けるために、dddではコンシステントハッシュ法と呼ばれるアルゴリズムを採用しています。

コンシステントハッシュ法では、各ノードもキーも論理的にリング上のハッシュ空間に配置されているものとして考えます(この配置はノードの物理的なネットワークポロジとは無関係です)。ノードには、IPアドレスをハッシュした値がノードIDとしてつけられます。また、格納したいキーも同様にハッシュ空間に写像し、そこから反時計回りにまわって最初に遭遇するノードが、そのキーの格納ノードとして処理を担当することとなります。(なお、ハッシュ関数にはSHA1を使っているため、ハッシュ空間は2の160乗の大きさがあります)。この方法では、格納されているキーの数やデータの量とは無関係に、キーの値からハッシュ値を計算するだけで格納ノードを割り出すことが可能です。(図-5)

コンシステントハッシュ法の最大のメリットは、ノードが増減した場合に、影響を受けるキーの範囲が限定されることです。例えば、図の中でノードBが故障でなくなった場合、薄い緑色で示される範囲のキーのみが影響を受け、それ以外のキーは影響を受けません。(図-6)

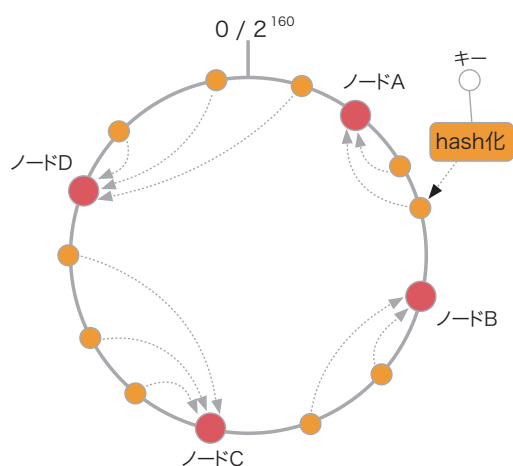


図-5 コンシステントハッシュ法:キーとノードの論理的配置図

分散システム環境では、日常的なノードの増減を前提として考える必要があります。コンシステントハッシュ法は、その増減に強いという利点のため、分散キーバリュースタアではよく使われるアルゴリズムです。

■データの自動的な冗長化

dddでは、冗長化のために、全てのデータは必ず異なる3つのノードに複製されるようになっています。先ほど、コンシステントハッシュ法で、キーのハッシュ値から反時計回りに最初にあたったノードに格納することを解説しましたが、さらに2番目と3番目のノードにも同じデータをコピーします。つまり、同時に3台のノードが壊れない限り、データはいずれかのノードに保持されるため、データ保全の信頼性は高いといえるでしょう。

ノードが壊れると、一時的にはデータの複製数が3つではなく2つや1つになります(ここではゼロになってしまうことは考えません)。この状態が長く続かないよう、dddでは残ったデータをコピーして常に複製数を3つに保つようになっています。そのために、dddは図-7のようなihave/sendmeと呼ばれる仕組みがあります。

まず、各ノードは自分の持っているキーをリストアップします。キーの値から、コンシステントハッシュ法により、そのキーを担当しているべきノードを割り出せるの

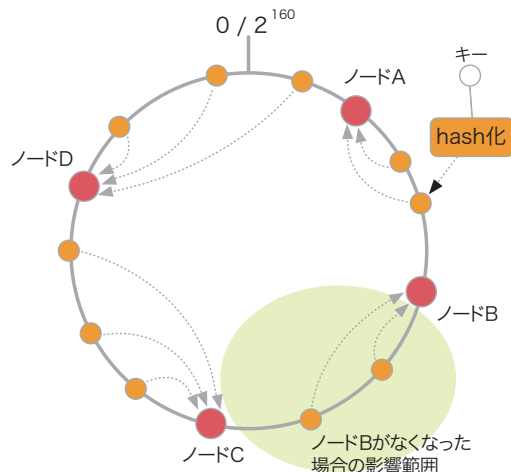


図-6 コンシステントハッシュ法:ノード故障時の影響範囲

で、そのノードに向かって、キーのリストを送ります。これを、ihaveメッセージといいます。ihaveメッセージを受け取った側は、リストに含まれているキーと自分の持っているキーを比べて、持っていないキーがあればそのデータの送信要求を返します。これをsendmeメッセージといいます。sendmeを受け取ったノードは、要求元にデータを送信します。実際には、キーの数は大量にあるため、一度に全てのキー情報を交換するのではなく、少しずつ分割してやりとりします。また、ノードの数が増減した場合は、その影響を受ける範囲のキーについてのみ、コンシステントハッシュ法で割り出されるihave送信先が変化し、新たに担当となったノードにデータが転送されます。図-6で、ノードBが無くなったケースにおいて、薄い緑色で示される範囲のキーの担当は「ノードB、A、D」から、「ノードA、D、C」に変化し、新たに担当となったCにA、Dのいずれかからデータが転送されます。dddは延々とこれを繰り返し、結果として、若干のリードタイムでデータの複製数を3つ保持します。

dddでは、ノードのハードウェアコストを低く抑えるために、RAID等のディスク冗長化技術は採用していません。そのかわり、より上位のレベルでデータを冗長化して、いつノードが故障して脱退しても支障のない仕組みを実現しています。

一方で、分散ストレージでは、同じデータを複数ノードで分散保持するために、データの一貫性を保つことが常にできるとは限りません。複数のノードをネットワー

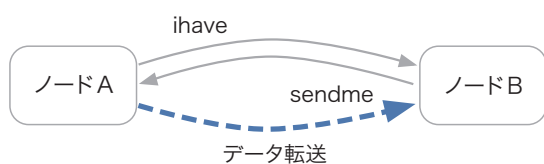


図-7 ihave/sendmeの仕組み

クで接続する分散システムでは、一部のノードが故障したり、ネットワーク的に分断される可能性があり、それでもシステムの可用性を高めるためには、必然的にトレードオフとして一貫性を保証できなくなります。多くの場合、不整合が発生するタイミングはごく短く、時間経過により一貫性が保たれるような機構になっていますが、アプリケーション側で不整合に対処することも必要です。

IIJでは、前述のフロー情報や、各種サーバのログなどを、機器IDと時刻をキーにしてdddの分散ストレージに格納しています。フロー情報もログ情報も、内容は不変であり、一度書き込んだ情報を更新する必要はありません。そのため、上記の一貫性が保証されない特性は、ほとんど問題になりません。ただし、ノードの増減に伴うデータの複製が発生している瞬間は、あるはずのノードにデータがないということが起こりえます。その場合、dddを使うアプリケーション側で、2番目・3番目の候補のノードにリトライしてデータを取得するようにしています。

3.5.3 MapReduce

MapReduceは、Googleで考案された大規模データ処理のためのプログラミングモデルです。名前の通り、データをmapとreduceの2段階に分けて処理するモデルで、うまく使うと多数のノードで効率よく分散並列処理ができます。(図-8)

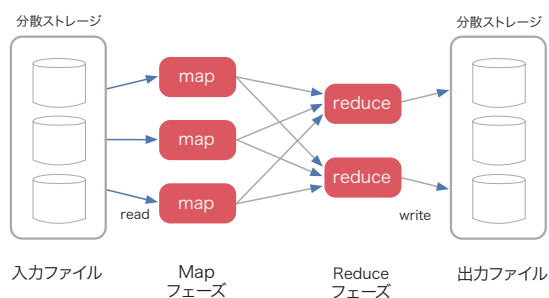


図-8 MapReduce概念図

GoogleのMapReduceのソースコードは公開されていませんが、Googleにより仕組みに関する論文が発表されています。今では、それを元に類似の機能を持った実装がいくつか出てきており、検索エンジンのインデックスの作成や、ログファイルの解析等に使われたりしているようです。dddも、分散ストレージのデータを処理するためにMapReduceの機能を搭載しています。

mapとreduceは、わかりやすく言い換えると「抽出」と「集約」と言えます。map(抽出)のフェーズではデータの中から必要な情報を抽出し、必要に応じて後の処理がしやすい形に変換します。reduce(集約)のフェーズでは、mapされた情報を集約します。処理内容それぞれに依存関係がないならば、その処理は複数のノードに分担して並列実行できます。

MapReduceの、もっとも単純で分かりやすい応用分野は、分散grepでしょう。grepはUnix系のOSに付属しているコマンドで、ファイルの中で指定したパターンにマッチする行を検索して出力するものです。grepは事前に検索のためのインデックスを作らず、その都度総当たりで調べるため、ファイルが巨大であったり、大量にある場合は時間がかかることがあります。MapReduceを使えば、対象ファイルのgrep処理を複数ノードで分担できるため、全体の処理時間を短縮することが期待できます。

IJでは、前述のとおりフロー情報やサーバが出力するログを分散ストレージに格納しており、MapReduceを使ってデータの抽出や加工をしています。これらのデータの解析可能なパラメータの組み合わせは多岐にわたっており、解析するポイントに応じて、様々なパラメータで解析できるようなシステムを構築しています。

フロー情報を解析する場合を例に、MapReduceの実際の動きを説明すると以下のようになります。

- 解析対象ルータ、対象期間と、抽出条件やグルーピングしたい軸等のパラメータを含む、MapReduceジョブリクエストを準備
- クライアントは、dddのノードのどれかに対しMapReduceジョブリクエストを送信する
- リクエストを受けたノードは、対象期間を一定時間間隔で分割する形で、MapReduceジョブを複数のmapタスクとreduceタスクに分解する
- mapタスクを各ノードに割り振り、各ノードはパラメータに従い、抽出やグルーピングの処理をする
- 各ノードでのmapタスクの実行が終わったら、reduceタスクを起動して結果を集約する
- 集約された結果は分散ストレージに書き出される。またはクライアントに返すことも可能

このようなシステムでは、ごく短期間のデータしか保持できなかったり、解析に長い時間がかかったりするものが多いですが、IJでは多数のノードからなる分散システムを用いることで、長期間のデータから実用的な応答速度での解析を実現しています。

3.6 おわりに

IJで開発した、dddという分散システムについて説明しました。dddを用いることで、膨大なデータを保持し処理することができます。今後、プロバイダのインフラで扱うデータはますます増加していくと考えられます。IJでは継続してdddを開発し、社会基盤として信頼性の高いサービスを提供していきます。

執筆者:

前橋 孝広 (まえはし たかひろ)

IJ サービス事業統括本部 システム基盤統括部 システム開発課

dddの実装を始めとした、IJのバックボーンネットワーク運用に関わるプログラムの開発を手がける。