なる。

表4-8 写真コメントテーブル

写真ID	コメントID	コメント	コメント投稿者
Photo123	18	カッコいい！	マリー
Photo123	22	かわい〜	アリス
Photo123	29	面白いね	チャールズ

では、アリスが投稿したコメント群を取得するにはどうすればよいだろうか。それにはアリスのプロフィールレコード（彼女のユーザ名をキーとして持つ）とコメントレコードを結合しなくてはならない。我々のスケールアウトアーキテクチャのせいでデータが数多くのサーバに分散して格納されているから、結合の実行には多くのサーバにアクセスする必要があるかもしれない。複数のサーバに接続しなくてはならないことと、一回のクエリで膨大な量のサーバ負荷を生成して他のリクエストまでも遅延させることから、この高コスト操作はリクエストの遅延を増大させる。

スケールアウトシステムで処理すると高コストになりそうな他の種類のクエリとして、group-byによる集約クエリがある。ユーザが趣味を入力しているとする。どの趣味に人気があるのかをアリスに見せるため、それぞれの趣味ごとにユーザ数を数えたいとしよう。この種のクエリでは、全データをスキャンして数え上げる必要がある。このテーブルスキャンは、システムに堪え難い負荷をかける。さらにアリスのページの表示がいつまでたっても終わらないという事態を避けるため、同期処理することができない。

これらの例から考えると、単一レコードの参照やレンジスキャンは高速に実行できるものの、より重い結合や集約は同期実行できないことがわかる。

我々の手法

原則として我々は高コストな操作を非同期で扱うことにしたが、高コストなクエリは実は非同期には扱えない。非同期なクエリが自分のコメントをすべて集め終わったかどうかを見るために、アリスに繰り返し確認してもらうわけにはいかない。

しかしマテリアライズド・ビュー（Agarawal他、2009年）ならば非同期に維持できるし、アリスのログインと同時に（同期で）すぐにビューを見られる†。元のデータに比べれば、非同期に維持されるビューは古くなりがちだが、そもそもアプリケーションは最新ではないレプリカにうまく対処できるように構築されていなければならないのだから、通常は古めのビューを扱うのは問題ない。実際、我々はマテリアライズド・ビューを、データのレプリケーションと変換をする特殊なレプリカとして扱っている。別の仕組みを設計、実装することなく、レプリカを更新するのと同じ仕組みでビューを更新することで、レプリケーションされた元データと同等の信頼性と一貫性をビューが持つことを保証する。

ビューがバックグラウンドで維持されるにしても、その処理は軽くしておきたい。ビューの維持に多量のシステムリソースを使うようだと、同期読み出しと書き込みにも（全リクエストが遅延するため）悪影響を与える。あるいはビューの維持をゆっくり抑えて走らせると、もはや有用ではないくらいビューが古くなる。そこでビューの維持を効率的にする方法を見つけなければならない。アリスが他の人の写真につ

† 現在はまだ、マテリアライズド・ビューは本番環境で稼働していない。

けたコメントをアリス自身が閲覧するという前述の例で考える。主キーではなく外部キー（コメントをつけた人のユーザ名）でクラスタ化したコメントデータのマテリアライズド・ビューを作るとしよう。すると、アリスがつけたコメントすべてがクラスタ化される。さらにこのビューに、ユーザ名をキーとしてアリスのプロフィールレコードを入れて、彼女のプロフィールとコメントをクラスタ化することもできる。キーと外部キーによる結合の計算は、「アリス」で始まるビューレコードの一群をスキャンして結合するだけの簡単なものになる。結果を表4-9に示す。

表4-9 プロフィールレコードとコメントレコードの相互クラスタリング結合

アリス	西海岸	32	アリス・スミス	…	←プロフィールレコード
アリス	Photo123	22	かわいい〜	…	←コメントレコード
アリス	Photo203	43	いいね	…	↓
アリス	Photo418	33	まあまあかな	…	
…					

プロフィールレコードとコメントレコードをビューの中であらかじめ結合していないことに注意してもらいたい。結合するレコードを単に一緒に置くだけで、低コストに結合状態を維持できる。元レコードを1つ更新するとき、たとえそのレコードが他の複数のレコードと結合されていたとしても、ビューのレコード1つを更新するだけで済む。

どうやって同じテーブルにプロフィールレコードとコメントレコードを格納しているのだろうか。2つのレコードはスキーマが異なるため、これは伝統的なデータベースでは難しい。しかしPNUTSはその中核機能として柔軟なスキーマ表現を持つ。1つのテーブル中のレコードはそれぞれ異なる属性群を持つことができる。商品データベースは商品の種類に応じて色、重さ、メモリ容量、味など異なる属性を持つ。この例に代表されるようにウェブのデータは疎であるため、柔軟なスキーマ構造はウェブアプリケーションで非常に有用だ。別々のテーブルから得た結合対象のレコードを一緒に配置してマテリアライズド結合ビューを作ろうとしたとき、結果として柔軟なスキーマが鍵となることがわかった。

非同期ビューの手法は他の種類のクエリに応える際にも同様に役に立つ。group-by集約クエリは、あらかじめデータをグループ化した、そしておそらくはあらかじめ集約したマテリアライズド・ビューで効率よく応えられる。非キー属性の選択といったような、より「簡単な」クエリに最も効率よく応えられるのもマテリアライズド・ビューである。カリフォルニア州サニーベールに住むユーザを取得するクエリを考えよう。我々のユーザテーブルはユーザ名をキーとしているため、このクエリは通常、高コストのテーブルスキャンを必要とする。しかし我々は、このテーブルの位置フィールドに対してマテリアライズド・ビューの仕組みにより2つめのインデックスを張り、そのインデックスを順序付きYDOTテーブルに格納して「カリフォルニア州サニーベール」インデックスレコードをレンジスキャンすることで、このクエリに応えることができる（表4-10）。

表4-10 位置インデックス

位置	ユーザ名
カリフォルニア州サニーベール	アリス
カリフォルニア州サニーベール	マリー
カリフォルニア州サニーベール	スティーブ
カリフォルニア州サニーベール	ザック
...	

他システムにおけるマテリアライズド・ビューと同様に、どのようなクエリが発行されるのか事前にわかっていれば効率的にマテリアライズド・ビューを作れる。幸いにもウェブサーバのクエリでは、事前にわかっているテンプレートに対して、実行時に位置やユーザ名などの特定のパラメタを指定する。アプリケーション開発者はこのようにして、どのクエリがマテリアライズド・ビューを要するほど複雑であるかを事前に知っている。PNUTSに格納したデータにアドホックなクエリを発行する場合は、MapReduceのオープンソース実装であるHadoopを走らせている計算グリッドに、我々のシステムからプラグインを用いてデータを取り出さなくてはならない。

複雑なクエリを扱う別々の仕組みがいくつか揃ったら、クエリを効率的に実行するのを助けるためにクエリプランナを実装すると便利だ。プランナはアプリケーション開発者から負担をいくぶんか取り除き、どのように実行されるのかを過度に気にすることなく、宣言的なクエリ記述を可能にする。とはいえ我々の規模で効率的なクエリプランナを実現するには、高度な統計の収集、負荷監視、ネットワーク監視、その他さまざまな仕組みを用いてプランナが最も効率的なクエリプランを作るように、システム中でボトルネックとなり得るものすべての情報を確実にクエリプランナに与える必要がある。

他のシステムとの比較

我々がPNUTSについて考え始めた時、他の2つの大規模データベースシステムがGoogleとAmazonから発表されたばかりで、さらにMicrosoftから3つめが登場しようとしていた。設計にあたって、彼らのアイデアのうち我々にも有用なものはあるだろうかと、注意深くこれらのシステムを分析した。これらのシステムのアイデアのうちいくつかは我々に影響を与えたが、我々は多くの点でアーキテクチャの異なる新しいシステムを構築することにした。ここでは、これらのシステムをそれぞれ見て、我々がなぜ彼らと設計原則を異にしたのかを議論する。

GoogleのBigTable

BigTable（Chang他、2006年）はGoogleの数々のウェブアプリケーションを支えるために設計されたシステムだ。このシステムは、基本的に「big table」を数多くの小さなタブレットに水平パーティション化し、これらタブレットをサーバ間に分散させている。スケーラビリティのための基本的な手法、柔軟なスキーマ、順序付けしたストレージは我々の手法と似ている。しかしBigTableとたもとを分かつ、設計上の決断がいくつかある。

1つめの大きな違いはレプリケーションの手法である。BigTableはGoogle File System（GFS：Ghemawat他、2003年）の上に構築されている。データをレプリケーションする際、GFSは3つの異な

るサーバにある3つのコピーデータを同期更新する。この手法は、サーバ間の遅延が小さい単一のcoloではうまく働く。しかし広域に分散した3つのcoloに置かれたサーバを同期更新するのはコストが高すぎる。アリスの話で言えば、インターネットバックボーンへの接続が細いデータセンタに彼女の友達がアクセスしている時、アリスは自分の状態が更新されるまで長時間待たねばならない可能性がある。coloをまたいだレプリケーションを支えるため、我々は時間軸整合性と共に、それに関連したマスタシップ、負荷分散、障害対応の仕組みを開発した。

さらに我々は、GFSとBigTableの間にあるような、データベースサーバとファイルシステムとの間の明確な分離はしないことに決めた。GFSはもともと、MapReduceを始めとする巨大ファイルのスキャンを中心とする負荷のために設計され、最適化されている。BigTableは各々のレコードのバージョン履歴を保存するためにGFSを用い、容量節約のためにSSTableと呼ばれるファイルフォーマットに圧縮保存している。これはつまり、レコードを読み出して更新する時に、データをデコードして、さらにこの圧縮フォーマットへエンコードしなくてはならないことを意味する。さらにBigTableのスキャン中心性のため、BigTableは全ユーザの全位置を取得するといった列中心のスキャンには強い。しかし我々の負荷の多くは、それとは対照的に単一のバージョンの単一のレコードや、狭い範囲のレコードを読み書きするものである。従って我々は、B-tree編成した完全なレコードとしてデータをディスクに保存している。これは主キーによって識別された個別のレコードを高速に見つけてその場で更新するのに最適化した手法である。

他にもPNUTSとBigTableの相違点はある。例えば、アプリケーションがひとつの巨大なテーブルではなく、複数のテーブルを持つことができる。さらに順序付きテーブルとともにハッシュテーブルもサポートした。BigTableに追随しているMegaStore（Furman他、2008年）というDBはトランザクション、インデックス、豊富なAPIを追加しているが、それでもなおBigTableの教義に従っている。

AmazonのDynamo

膨大な量のデータ取り扱いのためにAmazonが近年構築したシステムのひとつがDynamo（DeCandia他、2007年）である。高可用性を持った、巨大なスケールの構造的データストアを作るという我々の目標に最も近いもののひとつである。Dynamoではレコードのことをオブジェクトと呼ぶ。アプリケーションが任意のレプリカに書き込むことを許し、次に説明するゴシッププロトコルによってこれらの更新を他のレプリカに遅延伝搬させることで、書き込みの可用性を実現している。

遅くて障害の発生しやすいネットワークに対処するために更新を遅延伝搬させるという決断は、我々のものとも合致する。しかしレプリケーションの仕組みはまったく違う。ゴシッププロトコルでは、ランダムに選んだレプリカに更新を伝搬させ、さらにまたランダムにレプリカを選んで伝搬させる。このプロトコルは、多くのレプリカが比較的高速に更新されることを確率的に保証するが、それにはこのランダム性が欠かせない。しかし我々の場合では、ランダム性は明らかに最善ではない。アメリカ西海岸のcoloに対してアリスが自分の状態を更新することを考えてみる。ゴシップではこの更新はランダムにシンガポールのレプリカに伝搬し、その更新がさらにランダムにテキサスのレプリカに伝搬し、その更新がさらにランダムに東京のレプリカに伝搬するといったことが起こり得る。この更新は太平洋を3回横切っているが、より決定的な手法であれば貴重な太平洋横断バックボーンの帯域幅を節約して、この更新の転送を（そして他の更新も）1度だけにできる。さらにゴシップでは更新を伝搬しようとするレプリカは他のcoloのどのサーバがレプリカを持っているのかを知る必要があるため、負荷分散や障害回復のためにデータをサー

バ間で移動させるのが難しい。

Dynamoとのもうひとつの大きな違いは整合性維持プロトコルにある。ゴシップは更新が伝搬している当座の間レプリカに不整合があるものの、全データのレプリカが最終的には一致するという結果整合性に向いている。特に、レプリカは後に「不正」と見なされる状態をも取れる。例えばアリスが状態を「おやすみなさい」から「取り込み中」に、位置を「自宅」から「会社」に更新することを考えてみる。更新の順番から見て、アリスにとっての（それが大事なのだが）レコードの適切な状態は（おやすみなさい，自宅）か（取り込み中，自宅）か（取り込み中，会社）しかあり得ない。結果整合性では、この2つの更新が異なるレプリカに対してなされた場合、一部のレプリカは「会社」の更新を先に受けるため、これらのレプリカでは一時的に（おやすみなさい，会社）の状態になる。上司にはこの状態を見られたくない。レコードに対する複数の更新が正しい順序で適用されることを前提としているアプリケーションは、結果整合性よりも強い保証を必要とする。我々の時間軸整合性モデルはレプリカが古くなってしまうことを許す代わりに、古いレプリカでさえも正しい更新順を反映した整合性のとれたバージョンになる。

他にもDynamoとの違いはたくさんある。まず、Dynamoは順序付きテーブルを持たず、ハッシュテーブルのみを提供する。そして我々は負荷分散と障害復旧を目的として、データとサーバの結びつきをより柔軟にしている。特に順序付きテーブルの場合は、事前に予見できないホットスポットが生じ得る。AmazonはDynamoの他にもバイナリデータを格納するS3、インデックス付きの構造化データに対してクエリを実行するためのSimpleDBを提供する。SimpleDBは豊富なAPIを持つものの、アプリケーションはデータのパーティション化に向き合う必要がある。各パーティションには決められた上限容量があるからだ。つまり、1つのパーティション中でのデータ格納量が制限されている。

MicrosoftのAzure SDS

MicrosoftはAzureサービス（http://www.microsoft.com/azure/）の一部として大規模版のSQL Server（SQLデータサービスと呼ばれる）を構築した。ここでもまた、要点は水平パーティショニングによるスケーラビリティにある。SDSの素晴らしさは縦横にインデックスを付与したデータによる拡張クエリ能力と、SQL Serverをクエリ処理エンジンとして提供しているところにある。しかしSDSは厳格に強制されたパーティショニングによってこのクエリ表現力を達成している。アプリケーションが自身のパーティションを作成すると、簡単にはパーティション分けを変更できない。つまり、あるパーティションに対して表現力豊かなクエリを発行できるものの、パーティションが巨大になったり高負荷になっても、このシステムは簡単に自動でそのパーティションを分割してホットスポットに対処することができない。我々のシステムはテーブルという抽象化の背後にあるパーティション化を隠蔽しているので、負荷対策、障害対策のためにパーティションを作成したり変更したりできる。これは我々のクエリモデルが表現力ではかなわないことを意味する（我々のシステムはパーティションをまたいだ複雑なクエリをサポートしない）が、前述のビューなどのクエリ機能を拡張する方法を模索し続けている。

SDSとのもうひとつの違いは、PNUTSがシステムの基本機能として広域レプリケーションを備えていることである。SDSの少なくとも初期のリリースでは、負荷を単一のデータセンタで受け持つことを前提としており、プライマリレプリカが完全に利用不能になった時にのみリモートのコピーが使われる。我々は、シンガポール、ベルリン、リオ・デ・ジャネイロにいるアリスの友達には、それぞれのローカルにある「プライマリ」のコピーから彼女の更新を受け取れるようにしたかった。

他の関連システム

我々と同様にスケーラビリティと柔軟性を求める企業がさまざまな他のシステムを構築している。FacebookはCassandra（Lakshman他、2008年）というDynamo似のインフラの上にBigTable似のデータモデルを築き、ピア・ツー・ピアのデータストアを構築した。結果的にCassandraは結果整合性のみを提供している。

Flickr（Pattishall）やFacebook（Sobel、2008年）で使われているMySQLのシャーディング（sharding）手法に代表される分割データベースは、多数のサーバにまたがらせてデータをパーティション化することによりスケーラビリティを提供する。しかしシャーディングシステムは通常、我々が望むほどのスケールの自由さやデータの広域レプリケーションを提供しない。SimpleDBとまったく同様に、データをあらかじめパーティション化しなくてはならない。さらに、マスタとなって書き込みを受け付けるレプリカは1つのみである。PNUTSでは、異なるデータセンタにあるすべてのレプリカに（レコードごとに違うものの）書き込み可能である。

Yahoo!の他のシステム

PNUTSは、Yahoo!で構築しているいくつものクラウドシステムのうちのひとつである。他に2つ、データ管理を目的とするクラウドコンポーネントが存在するが、PNUTSとは異なる問題のためのものである。MapReduceフレームワーク（Dean、Ghemawat、2007年）のオープンソース実装であるHadoop（http://hadoop.apache.org）は、巨大なデータファイルを超並列で分析処理する。Hadoopはスキャンに最適化したHDFSというファイルシステムを持つ。これはMapReduceジョブが主にスキャンを中心とする負荷を発生させるためである。それとは対照的に、PNUTSは個々のレコードの読み書きに注力している。もうひとつのシステムはMObStorといい、画像や動画などの巨大なオブジェクトを格納するために設計されている。MObStorの目的は、不変なオブジェクトのための低遅延で低コストなストレージを提供することにある。多くのアプリケーションはレコードのストレージ、データ分析、オブジェクト検索を必要とすることから、我々はこの3システムをシームレスに結合しようとしている。2009年のCooperらによる論文では、これらのシステムを包括的クラウドへと結合する我々の取り組みを概観している。

まとめ

我々がPNUTSプロジェクトに乗り出したとき、頭の中には数千のサーバと複数の大陸にシームレスにスケールできるシステムがあった。そのようなシステムを構築するには、単なる技術力以上のものが必要だった。我々は、データベースの分野においてすでに解決済みの多くの議論を蒸し返さなくてはならなかった。ACIDを投げ捨てるという決断は比較的簡単であったが、それに代わる何かを開発しなくてはならないということを間もなく理解し、その結果として時間軸整合性モデルを開発した。このモデルは設計上は比較的シンプルだが、重箱の隅をつつくような複雑なケースに対処し、効率的な機構を実装し、アプリケーションのユースケースをこのモデルに対応させるという仕事は、深い思考と多くの繰り返しを要した。もうひとつ書き記しておきたいことは、ユーザも我々も、クエリ言語をシンプルなものに制限することに対して、現実を甘く見過ぎていたということだ。開発者が本物のアプリケーションをPNUTS上に構築しようとし始めるに従い、我々が扱うことのできない少数の複雑なクエリが、このシステムの採用にとって大きな障害であることを理解するようになった。もしこれらのクエリを扱う仕組みを開発しなかったら、開発者達はアプリケーションロジック中でネステッドループ結合などを用いて高コストな処理を実

装したり、彼らの負荷を支えるために外部のインデックスに頻繁にデータをエクスポートするなどの複雑な回避策を講じねばならなかっただろう。

クラウドデータ管理はまだ生まれたばかりの分野のため、多くの異なるシステムが設計、実装、運用されている。PNUTSシステムに体現されたアイデアによって、簡単に管理できて、幅広く使えて、複数のアプリケーションを同時に収容できるクラウドデータベースシステムに近づきたい。柔軟で、効率的で、世界中から使う事ができて、しかも非常に強靭なデータバックエンドをアプリケーションに提供したい。

謝辞

PNUTSはYahoo!の多くの人々の共同作業により生まれた。開発を主導したのはP.P.S. NarayanとChuck Neerdaelsである。このプロジェクトには他に、Adam SilberstienとRodrigo Fonsecaの2名の研究者が参加した。Brad McMillenとPat Quaid helpはPNUTSのアーキテクチャとYahoo!のクラウドシステムの中での位置づけについて支援してくれた。他に、以下の設計者、開発者がこのプロジェクトに参加した。Phil Bohannon、Ramana Yerneni、Daniel Weaver、Michael Bigby、Nicholas Puz、Hans-Arno Jacobsen、Bryan Call、Andrew Feng。

参考文献

Azure Services Platform. http://www.microsoft.com/azure/.

Agrawal, P., A. Silberstein, B. F. Cooper, U. Srivastava, and R. Ramakrishnan. "Asynchronous View Maintenance for VLSD Databases." In SIGMOD, 2009.

Chang, F. et al. "Bigtable: A distributed storage system for structured data." In OSDI, 2006.

Cooper, B. F., E. Baldeschwieler, R. Fonseca, J. J. Kistler, P.P.S. Narayan, Chuck Neerdaels, Toby Negrin, Raghu Ramakrishnan, Adam Silberstein, Utkarsh Srivastava, and Raymie Stata. "Building a cloud for Yahoo!" IEEE Data Engineering Bulletin, 32(1): 36-43, 2009.

Cooper, B. F., R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. "PNUTS: Yahoo!'s hosted data serving platform." In VLDB, 2008.

Dean, J. and S. Ghemawat. "MapReduce: Simplified data processing on large clusters." In OSDI, 2004.

DeCandia, G. et al. "Dynamo: Amazon's highly available key-value store." In SOSP, 2007.

Furman, J. J., J. S. Karlsson, J.-M. Leon, A. Lloyd, S. Newman and P. Zeyliger. "Megastore: A Scalable Data System for User Facing Applications." In SIGMOD, 2008.

Ghemawat, S., H. Gobioff, and S.-T. Leung. "The Google File System." In SOSP, 2003.

Lakshman, A., P. Malik, and K. Ranganathan. "Cassandra: A Structured Storage System on a P2P Network." In SIGMOD, 2008.

Pattishall, D. V. "Federation at Flickr: Doing Billions of Queries Per Day."http://www.scribd.com/doc/2592098/DVPmysqlucFederation-at-Flickr-Doing-Billions-of-Queries-Per-Day.

Ramakrishnan, R. and J. Gehrke. Database Management Systems. McGraw-Hill, New York, NY, 2002.

Silberstein, A., B. F. Cooper, U. Srivastava, E. Vee, R. Yerneni, and R. Ramakrishnan. "Efficient bulk insertion into a distributed ordered table." In SIGMOD, 2008.

Sobel, J. "Scaling out." Facebook Engineering Blog, August 2008.

5章

情報プラットフォームとデータサイエンティストの登場

Jeff Hammerbacher

図書館と脳

17歳のとき、私は米国インディアナ州フォートウェインの食料雑貨店のレジ係をクビになった。失業状態は大学入学までのわずか2か月間だったが、私にとっては格好の機会だった。クビになったことは両親には話さず、毎日午後になると黒ズボンに黒靴、白シャツ、上着というレジ係の服装で家を出た。両親にしてみれば、クーポンをスキャンしに行く準備が整ったように見えただろう。しかし、実際は10時間という勤務時間を公立図書館での読書に充てていたのである。

好奇心旺盛な人は、自分の脳がどのように働くのか知りたがるものだ。17歳の私はかなり好奇心旺盛で、図書館での時間を使って脳がどのように働き、壊れ、回復するのかを学んだ。人間の脳は、平衡感覚を保つ、体温を調節する、まばたきがときどき行われているかを確認することのほかに、大量の情報を収集、処理、作成する機能を持つ。私たちは、自分たちの置かれた状況にどう対処するか、言葉遣いや手足の位置をどうするかという身近なことから、配偶者の選択や教育といった長期的な計画にまで無意識に反応している。脳の働きの中でも興味深い機能は、単に知覚データに反応を起こす能力ではなく、計画を立てて新しい情報を作成するという、情報の宝庫としての脳の役割である。私はその仕組みについて学びたいと考えていた。

しかし、脳についてひとつやっかいなことがある。それはそれぞれ1つの体に格納されているということだ。そこで、たくさんの脳から情報を集めるために、図書館を建てる必要がある。図書館学の分野は、図書館に格納された情報を1か所に集めて、将来利用できるようにするための技術を大量に生み出した。これについては、Alex Wrightが書いた『Glut』(Joseph Henry Press刊)がおもしろい。これからの検索では、図書館は情報を格納するだけでなく、新しい情報を作成するという重要な役割を担っていく。これを哲学者Daniel Dennettは次のように表現している。「学者は図書館をもうひとつ作るための手段である」。

図書館や脳は、情報プラットフォームの一例である。情報プラットフォームは、企業が情報の収集、処理、作成を行うために努力した足跡であり、これによって経験的データからの学習過程が加速化される。2006年にFacebookプロジェクトに参加した私は、必然的に情報プラットフォームの構築から始めた。Facebookのユーザ数が驚異的に増加したため、開発したシステムは最終的に数PB(ペタバイト)ものデータを管理することになった。本章では、Facebookの情報プラットフォームの構築で直面した数々の問題や、オープンソースのソフトウェアから解決方法を見つけたときに学んだ教訓について述べる。また、情

報を使って大量のデータを扱う製品やサービスを開発したり、企業が立てた目標を達成するのを手助けする、というデータサイエンティストの重要な役割についても説明したい。過去数十年の間に、企業が情報プラットフォームの構築で問題に直面したときどう対処してきたか。これも見ていくことにしよう。

話を始める前に、さきほどの後日談を。私が食料雑貨店をクビになった代わりに図書館に通うという賢い計画は、思うように行かなかった。読書三昧の至福の日々を数日過ごしたある晩、図書館の外に出ると自分の車が見つからない。当時、車が見つからないことはよくあったが、そのとき駐車場には車が1台もなかった。何か問題が起こったとピンときた。母が私の計画に勘づいて、車をレッカー移動させたのだ。家までの長い道のりを歩いて帰る間、大事な教訓を学んだ。「自分で見つけた解決方法は懐疑的な目で見ること。そして、母親をだまそうとしないこと」。

自己認識能力を持つようになったFacebook

Facebookは2005年9月に大学生以外にも公開されるようになり、高校生もアカウント登録が許可された。忠誠心の高いユーザは激怒したが、Facebookのチームはサイトの方向性は正しいと考えていた。それが正しかったという確証は持てないが。

その後、Facebookはほぼすべての大学の学生たちに広まったが、うまく軌道に乗らない大学もいくつか残っていた。成功した例とそうでない例とでは、何が違ったのだろうか？　その成功を促した要因は一体何なのか？

2006年2月、私がFacebookの面接を受けたとき、Facebook側はこれらの課題を積極的に解決しようとしていた。私は大学で数学を学んだあと、1年近くウォール街で働き、金利や、価格の複雑な金融派生商品、そして住宅ローンのヘッジ資金を予測するモデルを開発した。プログラミング経験は少々、学業成績は散々、というのが私の経歴である。そんな私をFacebookはリサーチサイエンティストとして採用すると言ってきた。

同じころ、Facebookはレポートおよび分析の担当責任者を採用した。この責任者は、私よりも問題領域に関する経験をかなり積んでいた。私たちは、もう1人のエンジニアとともにデータの収集、格納を行うためのインフラストラクチャを構築し始めた。これによって、製品に関する前述の課題を解決できるはずだ。

最初、オフラインで情報を格納するために、FacebookのMySQLサーバの層にクエリを送信するPythonスクリプトを書き、リアルタイムでイベントログを処理するデーモンプロセスをC++で書いた。このスクリプトは予定どおりに動き、1日に約10GB（ギガバイト）のデータを集めた。あとでわかったことだが、このシステムは一般的に「ETL」と呼ばれる「データの抽出、加工、書き込み」処理を行っていた。

このPythonスクリプトとC++で書いたデーモンプロセスが、Facebookのソースシステムから一度データを吸い上げる。そして、オフラインでクエリを実行するために、私たちはそのデータをMySQLデータベースに入力した。さらに、MySQLデータベースに入力されたデータを一通り調べるスクリプトとクエリを作成して、有用な表現に情報集約した。このオフラインデータベースは意思決定支援に適しており、「データウェアハウス」として知られている。

最終的には、オフラインのMySQLデータベースからデータを抽出して、集めた情報の概要を内部ユーザに表示する簡単なPHPスクリプトを作成した。このとき初めて、私たちはサイトの特定の機能がユーザ活動にどのような影響を与えるのかという重要な課題を解決できるようになった。初めのころは、ログインしていないユーザ用のデフォルトページのレイアウト、ユーザの招待ルート、電子メールのコンタク

トリストのインポート機能のデザインといったいくつかのルートを分析して、どうしたらユーザ活動を最大限まで増やせるかを調べた。さらに、履歴データを使って簡単な製品の開発を始めた。この中には、ブランド広告主に人気があるスポンサ付きグループメンバの機能を集約する社内プロジェクトも含まれていた。

当時私は理解していなかったが、私たちはETLフレームワーク、データウェアハウス、企業ダッシュボード†といった、基本的な「ビジネスインテリジェンスシステム」を構築していたのである。

ビジネスインテリジェンスシステム

1958年、Hans Peter Luhnは「IBMシステムジャーナル」で、あるシステムに関する論文を発表した。このシステムは、ユーザ活動の特徴を表す「関心事のプロフィール」に基づいた「アクションポイント」に、文書を「選択的に情報提供」する。彼は驚くべき予言をしてみせた。この論文のタイトルは「ビジネスインテリジェンスシステム」。おそらく、このとき初めて「ビジネスインテリジェンス」という用語が登場したと思われる。

このシステムは、リアルタイムで情報を流す以外にも集めた文書全体にわたって「情報検索」すなわちサーチすることができた。Luhnは、アクションポイントの重要性を強調することで、目的達成における情報処理の役割に焦点を当てた。すなわち、単にデータを集めて集約するだけでは十分ではなく、企業はデータから集められた知識をもとに、基幹業務の遂行能力を向上させる必要がある。また、Luhnは定期的にデータをふるいにかけ、必要に応じて抜粋した情報をアクションポイントに転載する「レポータ」機能を提案している。

ビジネスインテリジェンスの分野は、Luhnの論文が発表されてから半世紀以上にわたって発展してきた。ビジネスインテリジェンスという用語は、構造化データの管理とより密接なつながりを持つようになった。現在の典型的なビジネスインテリジェンスシステムは、データソース群からデータウェアハウスに定期的にデータを引っ張るETLフレームワークで構成されている。その他にも、ビジネスアナリストが社内向けのレポートを作成するために使用するビジネスインテリジェンスツールがある。Luhnが描いたビジョンから現在の状況にいたるまで、私たちはどのような道のりをたどってきたのだろうか？

まず、1970年にE. F. Coddがデータのリレーショナルモデルを提案した。その後、1970年代中頃にはIBMが実際に動くリレーショナルデータベース管理システム（RDBMS）のプロトタイプを開発した。RDBMSの登場でユーザ向けのアプリケーション構築が非常に容易になり、1980年代初期までの間にRDBMSはつぎつぎに使われるようになった。

1983年、Teradataは意思決定支援向けに設計された初のリレーショナルデータベースをWells Fargoに販売した。それから数年後の1986年、Ralph Kimballが同じ市場向けのデータベースを開発するためにRed Brick Systemsを設立した。さまざまなシステムがTeradataとRed Brick Systemsの製品を使って開発されたが、データウェアハウスに関する標準的なテキストが初めて発行されたのは1991年だった。

Bill Inmonの「Building the Data Warehouse」（Wiley刊）は、データウェアハウスの設計に関するわかりやすい本で、データウェアハウスの構築方法や事例が詳しく載っている。Inmonは、既存のデータソースやビジネス目標を十分に研究した上で企業データモデルを構築するよう提唱している。

1995年、Inmonの著書が人気を博し、企業のデータセンタ内にデータウェアハウスが急増したころ、

† 訳注：経営状態を視覚化するビジネス管理ツール。

The Data Warehouse Institute (TDWI) が設立された。TDWIはカンファレンスやセミナーを開いて、データウェアハウスのビジョンを明確にしたり知識を広める上で、依然として重要な役割を担っている。同年、スタンフォード大学がWHIPS研究構想を発表したとき、データウェアハウスは学会で認められるようになった。

1996年、Ralph Kimballは「The Data Warehouse Toolkit」(Wiley刊)を出版して、Inmonの方法に異論を唱えた。Kimballは、データウェアハウスの世界への別のルートを推奨している。企業データモデルを捨てるという方法だ。Kimballは異なるビジネスユニットによってデータ「マート」を作り上げるべきで、それらは「バス」でつなげることができると主張する。さらに、正規化データモデルを使う代わりに、ディメンショナルモデリングの使用を提唱している。ディメンショナルモデリングは、データウェアハウスの実装時に多く見られる特定の作業負荷に対応するために、リレーショナルデータモデルに多少手を加えたものだ。

月日とともにデータウェアハウスの規模が大きくなるにつれて、ビジネスアナリストは小規模のデータサブセットを迅速に操作したいと思うようになってくる。そのデータサブセットは、いくつかの「ディメンション(次元)」でパラメータ化される。このような考えから、1997年にMicrosoftのJim Grayらの研究グループがCUBE演算子を導入した。この新しい演算子によって、小規模で多次元のデータセットに迅速なクエリを実行できるようになった。

リレーショナルモデルは、ユーザ向けのアプリケーション構築には向いているが、情報プラットフォームの構築には最適ではないかもしれない。そういった兆候をディメンショナルモデリングやCUBE演算子が示していた。さらに、Luhnが提案するビジネスインテリジェンスシステムの中心要素はテーブルではなく、文書やアクションポイントだ。その一方で、あらゆる世代のエンジニアはリレーショナルデータ処理を行うシステム構築に関する相当な専門知識を持っていた。

さて、過去の話はこれぐらいにして、Facebookの話題に戻ろう。

データウェアハウスの停止と復活

Facebookでは、MySQLデータウェアハウスに常時多くのデータを書き込み、クエリを実行していた。実際のサイトで利用しているデータベースでクエリを実行するだけなのだが、自分たちのデータウェアハウスでのクエリの実行時間が非常に長いことに驚いた。データウェアハウスの専門家と話し合ったところ、クエリが複雑でデータ量が多いため、実行に数時間から数日かかるのは当たり前であることがわかった。

ある日、データベースのサイズが1TB近くに達したとき、mysqldデーモンプロセスが突然停止した。しばらく調査した後、データベースを再起動することにした。再起動が始まったので、その日は帰宅した。

翌朝仕事に戻ると、データベースはまだ復旧中だった。多数のクライアントが変更するデータに一貫性を持たせるために、データベースサーバは「REDOログ」や「WAL (write-ahead log:先書きログ)」と呼ばれるすべての編集を記録するリストを保持している。もしデータベースサーバが突然停止して再起動したら、REDOログから最新の編集内容を読み込んで、できるだけ早く元の状態に戻す。私たちが開発したデータウェアハウスの規模では、MySQLデータベースの復旧にかなりの処理時間を要した。データウェアハウスが動き出すまで3日かかったのである。

この時点で、データウェアハウスをOracleに移行することにした。Oracleのデータベースソフトウェアは、大容量データセットの管理に適している。さらに、高価な高密度ストレージとSunの強力なサー

バを購入して、新しいデータウェアハウスを動かし始めた。

MySQLからOracleへの移行で、この2つの標準的なリレーショナルデータベースの実装に違いがあることがわかった。大量のデータをインポート、エクスポートする機能では、メカニズムが完全に異なっていた。さらにSQLの方言を使用していたため、多くのクエリを書き直す必要があった。もっと悪いことに、Oracleで使用しているPythonのクライアントライブラリは非公式で多少バグがあったため、開発者に直接連絡しなければならなかった。

この大変な作業が数週間続いた末、スクリプトをOracleの新しいプラットフォーム上で動くように書き換えることができた。夜ごとの処理は問題なく進み、Oracleの周辺ツールをいくつか使ってみようと盛り上がっていた。特に、Oracleには私たちがPythonで書いたスクリプトの代わりになると期待されるOracle Warehouse Builder（OWB）と呼ばれるETLツールがあった。しかし、残念ながらこのソフトウェアは私たちがサポートしなければならない膨大なデータソースを前提としていなかった。その当時、Facebookは毎晩集めたデータから作られた数万のMySQLデータベースを持っており、Oracleでも私たちが直面していたETLの問題には対処できなかったのである。しかし、数TBのデータを持つデータウェアハウスを動作させることができてうれしかった。

さらに、クリックストリームを記録し始めた。初日は丸1日かけて400ギガバイトの非構造化データをOracleデータベースに送り込んだ。ここで、私たちはまた自分たちのデータウェアハウスに懐疑的な目を向けることになる。

データウェアハウスを超えて

IDCは、デジタル空間は2011年までに1800エクサバイト（1エクサバイト＝1024の6乗バイト）に拡大すると予測している。このデータのほとんどはリレーショナルデータベースで管理されることはない。構造化データに合わせて、非構造化データから情報を抽出できるデータ管理システムの開発が急務であるが、解決方法の合意がほとんど得られていないのが現状である。

とりわけ、自然言語のデータは豊富で情報があふれているが、データウェアハウスによる管理はあまり行われていない。文書保管庫や音声録音によって保存されることがほとんどだ。こうした自然言語やその他の非構造化データを管理するために、各企業はさまざまな新しい分野に向けたデータウェアハウスベンダの製品を視野に入れている。その一例が企業検索と呼ばれる分野である。

ほとんどの検索会社が、ワールドワイドウェブと呼ばれるハイパーリンクが埋め込まれた文書の集まりをナビゲートするツールを構築する中で、社内文書の管理に特化した企業もあった。1996年にケンブリッジ大学の研究者たちが設立したAutonomy Corporationは、ベイズ推論アルゴリズムを使って重要な文書の管理を容易にした。1997年、ノルウェーではFast Search and Transfer（FAST）が設立された。同社の核となる技術は、より直接的なキーワード検索とランキングである。2年後、構造化メタデータを使って文書を収集する「ファセット検索」と呼ばれる技術に注目してEndecaが設立された。Googleは、検索分野での経験を活かして、2000年に企業検索アプライアンスを発表した。

企業検索の分野は、わずか2、3年の間に数十億ドル（数千億円）規模の市場に発展した。現在ではデータウェアハウス市場からほぼ完全に独立している。Endecaは、典型的なビジネスインテリジェンスのツールをいくつか持っており、データベースベンダの中にはテキストマイニング機能をシステムの中に組み込んでいるところもある。しかし、構造化および非構造化の企業データを管理する完璧な統合ソリューションはいまだ実現していない。

企業検索とデータウェアハウスは、両者とも生産性向上のために企業の情報資源を活用するという大規模な問題への技術的なソリューションである。話は1944年にさかのぼる。MITのKurt Lewin教授は、「計画、行動、行動した結果を検証するというステップの循環」を使用するフレームワークとして「アクションリサーチ」を提唱した。同じ問題へのより現代的なアプローチ方法をPeter Sengeが見いだした。この方法は「学習する組織」という概念に基づいており、彼の著作「The Fifth Discipline」(Broadway Business刊、邦訳「最強組織の法則」)の中で詳しく述べている。どちらの管理理論も、以前の行動から集められた情報を吟味してから行動を適応させる企業の能力をかなり重視している。このことから、情報プラットフォームはアクションリサーチの循環を実装するのに必要な情報を収集、処理、作成する「学習する組織」に必須のインフラストラクチャであると言える。

さて、構造化データと非構造化データの管理について述べたところで、Facebookの話に戻ろう。

チーターとゾウ

Facebookのクリックストリームのログを記録し始めた初日、400GB(ギガバイト)を超えるデータが集まった。このデータの書き込み、インデックス作成、集約処理で、Oracleデータウェアハウスにかなりの負荷をかけた。大幅にチューニングしても、1日分のクリックストリームデータを24時間以内に集約できなかった。はっきりしているのは、データベースの外でログファイルを集約して、あとで行うクエリで使うため、要約した情報のみ保存する必要があるということだった。

ちょうどそのころ、幸いにも大規模なWebサイトのトップエンジニアが私たちのチームに加わった。彼はWebスケールのクリックストリームデータを処理したことがあった。わずか2、3週間で彼はCheetah(チーター)と呼ばれる並列化ログ処理システムを開発したのだ。このシステムは、1日分のクリックストリームデータを2時間で処理できる。この成功に大喜びした。

成功はしたものの、Cheetahにはいくつか問題があった。1つは、クリックストリームデータの処理後、生データは保存用ストレージに格納されるため、再度クエリを実行できない点である。さらにCheetahは、読み取り操作の帯域幅に制約のある共有のNetAppファイラから、クリックストリームデータを取得していた。各ログファイルの「スキーマ」は、クエリの実行対象の中に保存されるのではなく、処理スクリプトの中に組み込まれている。私たちは処理状況を収集せずに、cronと呼ばれるUnixの基本ユーティリティを使ってCheetahのジョブをスケジューリングしたため、高機能の負荷分散ロジックを適用できなかった。しかし一番重要なことは、Cheetahがオープンソースではなかったという点だ。私たちのチームは少人数だったため、独自システムの開発や保守、そして新規ユーザにそのシステムを使ってもらうためのトレーニングを行う余裕がなかった。

Cheetahに替わる有力候補として、2005年後半にDoug CuttingとMike Cafarellaが始めたのが、Apache Hadoopプロジェクトだ。これはDougの息子がゾウのぬいぐるみにつけた名前に由来している。Hadoopプロジェクトは、Apache 2.0ライセンスのもとでGoogleの分散ファイルシステムとMapReduceテクノロジを実装することを目的としていた。Yahoo!は2006年1月にDoug Cuttingを採用し、Hadoopの開発に膨大な技術リソースをつぎ込んだ。2006年4月、Hadoopは188台のサーバを使って47時間で1.9TBのデータのソートが可能になった。Hadoopの設計はCheetahに比べていくつかの点で優れていたが、当時私たちが必要としていた速度には達していなかった。しかし、2008年4月までにHadoopは910台のサーバを使って209秒で1TBのデータをソートできるようになった。性能向上に伴い、60台の未使用のWebサーバに500ギガバイトのSATAドライブ3台を設置するように運用チームを説得できたため、

Facebookで初めてのHadoopクラスタの計画を進めた。

まず、HadoopとCheetahにログの一部を送信することから始めた。Hadoopはプログラムによる拡張性を備え、過去のデータに対してもクエリを実行できることから、おもしろいプロジェクトがいくつか登場した。あるアプリケーションは、Facebookで交流するユーザ同士のペアすべてにスコアをつけて、2人の相性を診断した。このスコアは、検索やニュースフィードのランキングに利用できた。しばらくして、Cheetahの作業をすべてHadoopに移行し、旧システムを引退させた。その後、トランザクションデータベースの収集処理もHadoopに移行した。

Hadoopの導入で、システムは非構造化データおよび構造化データの分析結果を大量に格納できるようになった。プラットフォームの処理能力が1日に数百TB、数千ジョブに向上したことで、新しいアプリケーションを構築したり、新しい質問に答えられるようになることがわかった。それは単に、データを保存、検索できるようになった規模のおかげである。

Facebookがすべてのユーザに対して門戸を開くと、ユーザ数は数か国で急速に増えた。しかし、当時は国別に分類されたクリックストリームデータを粒度の細かい分析にかけることはできなかった。Hadoopクラスタ導入後は、すべての履歴アクセスログをHadoopに読み込んで、いくつかの簡単なMapReduceジョブを書くことで、カナダやノルウェーなどの国でFacebookがどれだけ急速に普及したかを再現できた。

毎日、数百万の半公共的な会話がFacebookユーザの「wall」と呼ばれる掲示板上で行われている。内部の推定では、wallのメッセージデータベースのサイズは、ブログ界の10倍に及ぶとしている。しかし、Hadoop導入前はこれらの会話の内容はデータ分析用としてアクセスできなかった。2007年、言語学と統計学に強い関心を持つ夏期研修生のRoddy Lindsayがデータチームに加わった。Roddyは単独で、Hadoopを使ってLexiconという毎晩数TBのメッセージデータを処理し続ける強力な動向分析システムを構築した。ユーザは、http://facebook.com/lexiconで各自の結果を見ることができる。

Facebookアプリケーションを評価するスコアリングシステムの構築には、複数の異なるシステムから1つのレポジトリにデータを格納することが重要だとわかった。2007年5月にFacebookプラットフォームが登場してすぐに、ユーザの元にアプリケーションの追加依頼が殺到した。このため、ユーザにとって有用なアプリケーションと、スパムと認識されるアプリケーションとを区別するツールの必要性を感じた。そこで、APIサーバから集めたデータ、ユーザプロファイル、サイトの活動データを使って、ユーザに最も有用と思われるアプリケーションに招待状を割り当てるスコアリングアプリケーションのモデルを構築した。

データの不合理な有効性

Googleの研究者たちは、最近の論文で機械学習の難題をいくつか解決する上で学んだことを報告している。音声認識や自動翻訳の問題について、次のことを言っている。「多くのデータに基づいた単純なモデルは、少ないデータに基づいた複雑なモデルに常に勝る」。私はこれについてとやかく言うつもりはないが、複雑なモデルで成功しているドメインがあることも確かだ。しかし彼らの経験上、多くのデータに基づいた単純なモデルの方が、幅広い種類の問題でうまくいっているのが事実である。

Facebookでは、Hadoopはデータの不合理な有効性（the unreasonable effectiveness）を活用するためのツールだった。たとえば、サイトを別の言語に翻訳するときに、その言語を話すユーザに翻訳作業の協力を求めた。同僚のデータサイエンティストであるCameron Marlowは、Wikipediaをくまなく探索し

て各言語のトライグラムの頻度を数えた。この頻度を使って、彼は簡単な分類器を構築した。この分類器でユーザが書いた掲示板のメッセージを調べると、その人の言語を決定できる。翻訳プログラムにユーザを採用するときに、この分類器を使うことでターゲットを絞りこむことができた。FacebookもGoogleも、多くのアプリケーションで自然言語データを利用している。この話題に関しては、本書の第14章でPeter Norvigが述べている。

Googleは、現代のビジネスインテリジェンスシステムの3段階目の進化を示唆している。すなわち、これからのビジネスインテリジェンスシステムは、1つのシステムで構造化データと非構造化データを管理するだけでなく、「単純なモデルで大量のデータを処理」する機械学習法のために十分なデータを格納できる規模にまで拡大できなければならない、と指摘する。

新しいツールと応用研究

FacebookでHadoopクラスタを導入した初期のころ、そのユーザの大半は新しい技術を好んで使いたがるエンジニアだった。私たちは、情報を企業のより多くの部署で利用しやすくするために、Hadoopの上で動作するHiveと呼ばれるデータウェアハウス用のフレームワークを作成した。

Hiveには、MapReduceロジックを埋め込む機能を持つSQLライクなクエリ言語が含まれるとともに、テーブル分割、サンプリング、任意のシリアライズされたデータを処理する機能が備わっている。この最後の機能は重要である。なぜなら、Hadoopに集められたデータは常に構造が変化しているからである。ユーザがシリアライゼーションフォーマットを指定できるようにすることで、どのようなデータ構造を解析するのかという問題を、Hiveにデータを読み込む責任者にまかせることができる。さらに、HiPalと呼ばれるHiveクエリを構築する簡単なユーザインタフェースを構築した。この新しいツールを使うと、マーケティングや製品管理、セールス、カスタマーサービスといったエンジニア以外の人でも数TBを超えるデータを扱うクエリを書くことができるようになった。社内で数か月使った後、HiveはApache 2.0ライセンスのもとで正式なサブプロジェクトとしてHadoopに寄贈され、現在も活発に開発が行われている。

私たちは、Hiveの他にもArgusというチャートとグラフを共有するポータルサイト（IBMのMany Eyesプロジェクトからヒントを得た）や、Databeeというワークフロー管理システム、PyHiveというPython言語でMapReduceスクリプトを書くフレームワーク、Cassandraというエンドユーザ向けに構造化データを提供するストレージシステム（現在はApache Incubatorとしてオープンソースで利用できる）を開発した。

新しいシステムが定着した結果、単一のHadoopクラスタが管理する複数のデータ層に行き着いた。アプリケーションログや、トランザクションデータベース、Webクロールなどの企業のデータすべてを、定期的にHadoop分散ファイルシステム（HDFS）に生データの形で集めた。毎晩数千のDatabee処理によって、このデータの一部を構造化形式に変換し、Hiveが管理するHDFSのディレクトリ内にそのデータを置く。さらに、Argusが提供するレポートを作成するためにHiveの中で情報集約が行われる。HDFS内では、個々のエンジニアのホームディレクトリ下に「サンドボックス」を持つことにより、プロトタイプのジョブが実行可能になった。

現在、このクラスタは約2.5PBのデータを格納し、1日15TBの割合で新しいデータが追加されている。毎日3000を超えるMapReduceのジョブが実行され、55TBのデータを処理する。クラスタ上で実行されるジョブの優先順位を調整するために、複数のキュー間でリソースを平等に分割するジョブスケジューラ

も開発した。

FacebookのHadoopクラスタ導入で、内外のレポートの強化、A/Bテストのパイプライン、さまざまな大規模データを扱う製品やサービスの他に、興味深い応用研究プロジェクトが可能になった。

ユーザがあるサイトに長期間参加するかを予測する上で最も重要な要因は何か？　これについて、データサイエンティストのItamar RosennとCameron Marlowは長いこと研究を重ねた。私たちは自分たちのプラットフォームを使って、ユーザサンプルの選択や、異常値の除去、さまざまな特徴量の算出を行った。参加を予測する方法はいろいろあるが、そのうちの最小角回帰法を用いた。Hadoopを使って算出した特徴量の中には、プロファイル属性に基づく友達ネットワークの密度やユーザカテゴリも含まれている。

新規ユーザにコンテンツを投稿してもらうにはどうしたらいいか？　これについて社内で別の研究が行われた。その内容は、2009年のCHIカンファレンスで発表された「Feed Me：Motivating Newcomer Contribution in Social Network Sites（ソーシャルネットワークサイトの新規ユーザにいかに投稿意欲を湧かせるか）」という論文に書かれている。Facebookデータチームの最近の研究では、Facebookソーシャルグラフを通して情報がどのように流れるかに注目している。この研究のタイトルは「Gesundheit! Modeling Contagion through Facebook News Feed（Facebookニュースフィードによる感染にご注意を！）」で、2009年のICWSMカンファレンスで発表された。

Facebookで共有されている情報プラットフォームを使うことで、日々証拠が集められ、仮説が検証され、アプリケーションが開発され、新しい見識が生まれる。他社でも並行して同様のシステムを開発中だった。

MADスキルとCosmos

2009年のVLDBカンファレンスで発表された論文「MAD Skills: New Analysis Practices for Big Data（MADスキル：大量データ向けの新しい分析技術）」では、Fox Interactive Media（FIM）での分析環境が詳しく説明されている。FIMのチームは、HadoopとGreenplumのデータベースシステムを組み合わせて、Facebookとは異なる、データ処理用の使いやすいプラットフォームを独自に開発した。

この論文の題名は、「Magnetic、Agile、Deep」というFIMプラットフォームの3つの理念を表している。「Magnetic」は、企業データモデルに適合する構造化データだけではなく、企業からのデータすべてを保存したいという願望を表している。同様に「Agile」プラットフォームはスキーマの変更に適切に対処するためにアナリストが即時にデータ処理を行い、必要に応じてデータモデルを変更できることを意味する。「Deep」はより複雑な統計分析を実践することを表している。

FIMの環境では、単一のGreenplumデータベース内でデータをステージング、プロダクション、レポーティング、サンドボックスのスキーマに分ける。これは、前述したFacebookのHadoop内の複数の層とまったく同じだ。

それとは別に、Microsoftは自社のデータ管理スタックの詳細を公表した。同社は「Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks（シーケンシャル構成要素による分散データ並列プログラム）」や「SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets（大規模データセットの簡単かつ効率的な並列処理）」の論文の中で、私たちがFacebookで構築したものと非常に似ている情報プラットフォームについて記述している。このインフラストラクチャには、Cosmosと呼ばれる分散ファイルシステムと、Dryadと呼ばれる並列データ処理システムが含まれている。しかも、同社

はSCOPEと呼ばれるSQLライクなクエリ言語を生み出した。

3つのチームがそれぞれ独自の技術を生み出し開発を進めた結果、大量のデータ処理向けの似たようなプラットフォームができ上がった。一体何が起こったのだろうか？　構造の指定をデータ保存機能から切り離して、データ検索用のAPIを導入することで、大規模なWebサイトのストレージシステムは「データベース」的なものから「テータスペース」的なものに移行し始めているのである。

データスペースとしての情報プラットフォーム

聞くところによると、Yahoo!やQuantcast、Last.fmなどの企業にも似たようなペタバイト級のプラットフォームがあるらしい。この規模のプラットフォームは、ほとんどがリレーショナルデータベースや従来のデータウェアハウスのモデリング技術を使っていないため、もはやデータウェアハウスではない。これらのプラットフォームは、データの一部しかインデックス化されておらず、かなりリッチなAPIを提供しているため、企業検索システムでもない。さらに、従来のデータ分析を行うだけでなく、製品やサービスの開発にも利用されている。データ処理用の共有プラットフォームは、脳や図書館と同様に、企業が情報を収集、処理、作成する場所としての機能を果たす。しかも、うまくいけば経験的データから企業の学習ペースが早まる効果もある。

データベースの分野では、純粋なリレーショナルデータ管理から、「データスペース」と呼ばれる大規模なデータセットのストレージやクエリに適した広範なシステムに研究テーマを移行する動きが見られる。論文「From Databases to Dataspaces: A New Abstraction for Information Management（データベースからデータスペースへ：情報管理の新しい概念）」(http://www.eecs.berkeley.edu/~franklin/Papers/dataspaceSR.pdf) では、ストレージシステムがすべてのデータフォーマットを受け入れ、各データに応じたデータアクセス用APIを提供する必要性を強調している。

これまで述べてきた情報プラットフォームは、データスペースの実例である。これは、企業がエンジニアリング、分析、レポーティングに適したさまざまなデータアクセス用APIを公開して、あらゆる部門からペタバイト級の構造化データおよび非構造化データを集め、そのデータを管理する単独のストレージシステムである。業界でこのようなシステムが普及すれば、今後もデータベースの分野でデータスペースの理論的な基礎や実用的な意義の研究が行われるだろうと私は期待している。

情報プラットフォームは、「学習する組織」の重要な構成要素である。その学習過程を加速化したり、情報プラットフォームを利用するときに最も重要な役割を果たすのが「データサイエンティスト」という新しい職業だ。

データサイエンティスト

GoogleのチーフエコノミストHal Varianは最近のインタビューで、前述の情報プラットフォームから情報を抽出できる人材の必要性を強調した。Varianは次のように述べている。「どこでも安く入手できるもの、そしてそれを補うサービスを探そう。どこでも安く入手できるものは何か？　データだ。データを補うものは何か？　分析である」。

Facebookでは、ビジネスアナリスト、統計学者、エンジニア、リサーチサイエンティストといった従来の肩書きは、私たちのチームにとってまったく魅力的なものではなかった。各役割の作業負荷は多種多様である。ある日の、あるメンバーの行動は、多段階の処理パイプラインをPythonで書き、仮説検定を設計し、統計ソフトウェアRを用いてデータサンプルの回帰分析を行い、Hadoopで大量のデータを扱う

製品やサービスのアルゴリズムを設計して実装し、分析結果を明瞭かつ簡潔な方法で組織の他のメンバーと話し合う、といった感じだ。このように数多くの仕事をこなすのに必要なスキル一式を表すために、私たちは「データサイエンティスト」という肩書きを作り出した。

金融サービス業界では、クオンツと呼ばれるデータサイエンティストが新しく開発した複雑なモデルを検証するために、過去の市場活動に関する膨大なデータを蓄積している。産業界以外でも、多くの理系大学院生がすでにデータサイエンティストの役割を演じていることに気づいた。Facebookデータチームに採用されたうちの1人は、生命情報工学の研究室出身で、データパイプラインの構築やオフラインでのデータ分析を行っていた。CERNの有名な大型ハドロン衝突型加速器では、実験で膨大なデータが生み出され、研究の飛躍的進歩を期待する大学院生たちがそのデータを収集して詳しく調べている。

Davenport と Harris 著「Competing on Analytics」(Harvard Business School Press、2007年刊)、Baker 著「Numerati」(Houghton Mifflin Harcourt、2008年刊)、Ayres 著「Super Crunchers」(Bantam、2008年刊) などの近著で、さまざまな業種で企業が自ら集めた情報をもとに生産性を上げるためには、データサイエンティストの役割が重要である、と強調している。今後数年以内に、研究団体によるデータスペースの調査とともに、データサイエンティストの役割をさらに明確に定義する必要が出てくる。データサイエンティストの役割を明確化することで、教育カリキュラムの構成、推進体系の構築、カンファレンスの開催、本の執筆といった、専門的職業として認められたことに付随するさまざまな仕事が可能になるだろう。それとともに、急増する情報プラットフォームの専門家としてのニーズに応えるために、データサイエンティストの要員は今後拡大し、あらゆる企業の学習過程がさらに加速化するだろう。

まとめ

Facebookの情報プラットフォームの開発で問題に直面したときは、過去にさかのぼり、さまざまな問題領域にわたって他の人が同様の問題をどのように解決しようとしたかを調べることが役に立った。当初、エンジニアの私が採った方法は、そのとき利用できる技術によって解決方針を決めるというもので、後から考えると目先のことしか見えていなかった。最大の課題は、データウェアハウスや企業検索システムなどの特定の技術システムに注目するのではなく、「学習する組織」のインフラストラクチャおよび人員体制の構築という規模の大きな問題に常に目を向けておくことだった。

情報プラットフォームの構築に使用されるハードウェアやソフトウェアは今後急速に進化し、データサイエンティストに必要なスキルも同様の速度で変化すると私は確信している。学習過程の高速化という目標を掲げ続けることで、企業とサイエンスの両方に利益がもたらされることだろう。未来はデータサイエンティストの手にかかっているのだ。