Apache Spark と Delta Lake のオリジナルクリエータによるガイド

開発者が Delta Lake と Apache Spark™ を学ぶための 8 つステップ

databricks

Apache Spark と Delta Lake のオリジナルクリエイターによる

開発者が Delta Lake で Apache Spark™ を学ぶための 8 つのステップ

Databricks Inc.
160 Spear Street, 13th
Floor San Francisco,
CA 94105
お問い合わせ

Databricks について

Databricks のミッションは、データサイエンス、エンジニアリング、ビジネスを統合することで、顧客のイノベーションを加速させることです。Apache Spark™ を開発したチームによって設立された Databricks は、データサイエンスチームがデータエンジニアリングやビジネスラインと連携してデータ製品を構築するための統合アナリティクスプラットフォームを提供しています。ユーザーは、ETL やインタラクティブな探索から本番までの分析ワークフローを作成することで、Databricks を利用して、より迅速な Time-to-Value を実現することができます。また、完全に管理されたスケーラブルで安全なクラウドインフラストラクチャを提供することで、運用の複雑さと総所有コストを削減し、ユーザーがデータに集中できるようにしています。Andreessen Horowitz とNEAがベンチャー支援する Databricksは、CapitalOne、Salesforce、Viacom、Amgen、Shell、HP などのグローバルな顧客基盤を有しています。詳細は http://www.databricks.com/ をご覧ください。

目次

序章

設立以来、Databricks のミッションは、データとアナリティクスを統合することで、ビッグデータをシンプルにし、誰もがアクセスできるようにすることです。そして、その使命から逸脱したことはありません。ここ数年、開発者のコミュニティが Spark をどのように使用しているのか、また、組織がどのように Spark を使用して洗練されたアプリケーションを構築しているのかを学びました。

この電子書籍では、KDnuggetsで公開されたコンセプトを拡張、拡張、キュレーションしています。さらに、Delta Lake と Apache Spark 2.x に特化した技術的なブログや関連資産を、Spark の生みの親である Matei Zaharia 氏、Chief Architect の Reynold Xin 氏、Spark SQL と Structured Streaming のリードアーキテクトである Michael Armbrust 氏、Spark MLlib と SparkR の推進者の一人である Joseph Bradley 氏、Structured Streaming のリード開発者である Tathagata Das 氏など、Spark の主要な貢献者や Spark PMC のメンバーが執筆し、紹介しています。

Delta Lakeは、AWS S3、Azure Data Lake Storage、HDFS などの既存のデータレイクファイルストレージの上に置かれるオープンソースのストレージレイヤーです。Delta Lake は、データレイクに信頼性、パフォーマンス、ライフサイクル管理をもたらします。不完全なデータのインジェスト、コンプライアンスのためのデータ削除の困難さ、変更データの取り込みのためのデータ修正の問題はもうありません。安全で拡張性の高いクラウドサービスを利用することで、高品質のデータをデータレイクに取り込む速度と、チームがそのデータを活用する速度を加速させることができます。Linux Foundation がサポートするオープンソースプロジェクトである Delta Lake は、互換性のある任意のリーダーでデータを読み取ることができ、Apache Spark と互換性があります。

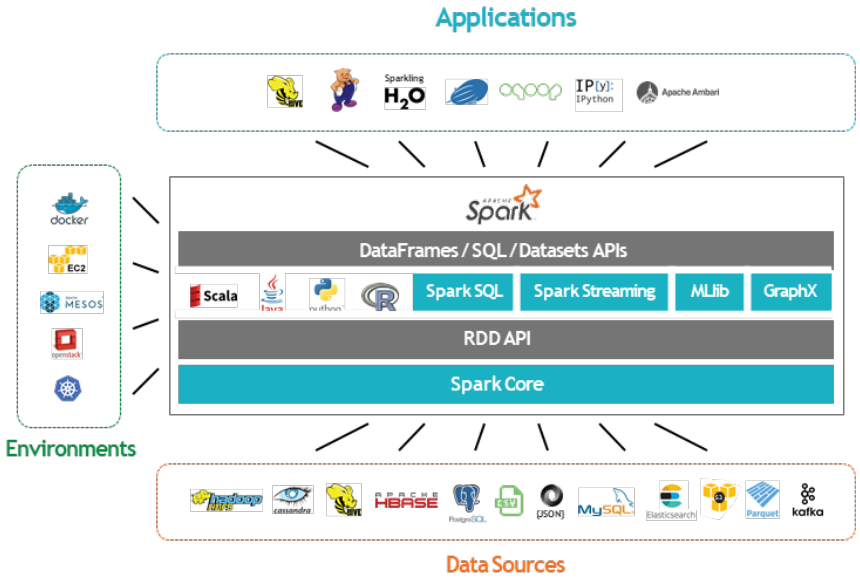
この本は、開発者が Delta Lake と Apache Spark をより深いレベルで理解するためのステップを紹介しています。Delta Lake と Apache Spark を使い始めたばかりの方でも、すでに開発経験のある方でも、Delta Lake と Apache Spark の利点を使いこなすための知識を身につけることができます。

ジュールズ・S・ダムジ（Jules S. Damji）
Apache Spark コミュニティエバンジェリスト

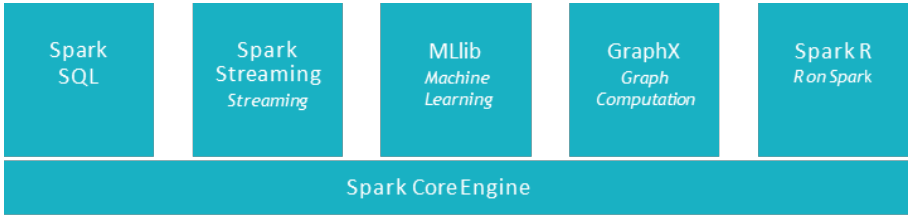
ステップ 1：
Apache Spark を選ぶ理由

Apache Spark を選ぶ理由

まず、Apache Spark は、スピード、使いやすさ、高度な分析のために開発された最もアクティブなオープンソースのデータ処理エンジンで、250 以上の組織から 1000 人以上のコントリビューターが参加しており、開発者や採用者、ユーザーのコミュニティも拡大しています。第二に、分散データ処理のために設計された汎用高速計算エンジンとして、Scala、Java、Python、R などの一般的なプログラミング言語で統一された API を介してアクセス可能なライブラリとして Spark コンポーネントで構成された統一エンジンを介して複数のワークロードをサポートしています。



この統合されたコンピュートエンジンにより、従来の ETL、ストリーミング ETL、インタラクティブクエリ、アドホッククエリ（Spark SQL）、高度なアナリティクス（機械学習）、グラフ処理（GraphX/GraphFrames）、ストリーミング（Structured Streaming）など、多様なワークロードに理想的な環境を Spark は提供しています。



その後のステップでは、開発者の視点から、これらのコンポーネントのいくつかを紹介しますが、まず、キーコンセプトとキー用語を捉えてみましょう。

ステップ 2：
Apache Spark の概念、主要な用語、キーワード

Apache Spark のアーキテクチャの概念、主要な用語とキーワード

2016 年 6 月に KDnuggetsが、「Apache Spark Key Terms Explained」という記事を公開していますが、これはここで紹介するにふさわしいものです。この記事で参照されている以下の Spark のアーキテクチャ用語を、この概念的な語彙に追加してください。

Spark クラスタ

Spark がインストールされているパブリッククラウドやプライベートデータセンターのオンプレミスにあるマシンやノードの集合体。これらのマシンの中には、Spark ワーカー、Spark マスター（スタンドアロンモードではクラスタマネージャ）、少なくとも 1 つの Spark Driver が含まれます。

Spark マスター

その名が示すように、Spark Master JVM は Standalone デプロイモードではクラスタマネージャとして動作し、Spark ワーカーはクォーラムの一部として登録します。デプロイモードに応じて、リソースマネージャとして動作し、どこで何人の Executor を起動するか、クラスタ内のどの Spark Worker を起動するかを決定します。

Spark ワーカー

Spark マスターからの指示を受けると、Spark ワーカー JVM は Spark ドライバに代わってワーカー上の Executor を起動します。タスク単位に分解された Spark アプリケーションは、各ワーカーの Executor 上で実行されます。つまり、ワーカーの仕事は、マスターに代わって Executor を起動することだけです。

Spark Executor

Spark Executor は、Spark がタスクを実行するために割り当てられたコア数とメモリを持つ JVM コンテナです。各ワーカーノードは、設定可能なコア数（またはスレッド数）の Spark Executor を起動します。Executor は Spark タスクを実行するだけでなく、全てのデータパーティションをメモリに保存したり、キャッシュしたりします。

Spark ドライバ

クラスタ内の全てのワーカーがどこにいるかの情報を Spark Master から取得すると、ドライバプログラムは各ワーカーの Executor に Spark タスクを配布します。ドライバはまた、各 Executor のタスクから計算された結果を受け取ります。

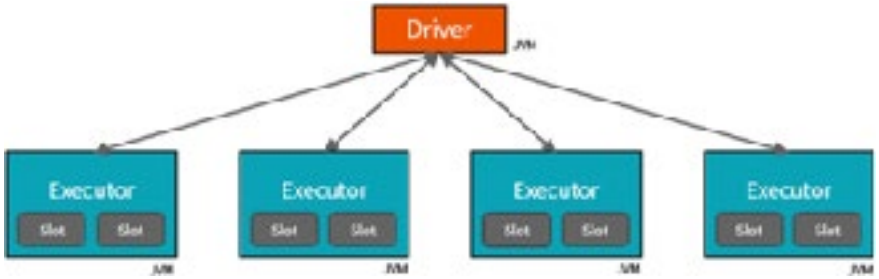


図1 Spark クラスタ

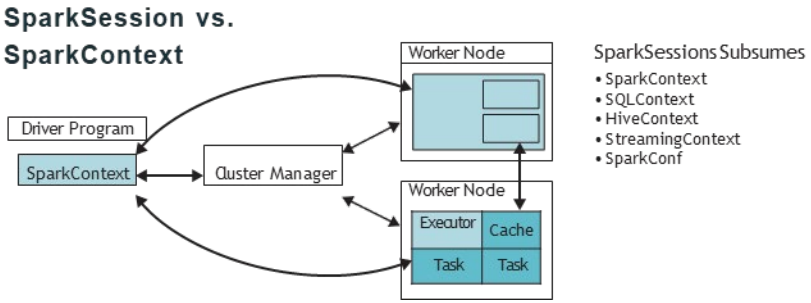


図2 SparkContextと Spark コンポーネントとの相互作用

Spark SessionとSpark Context

図2 に示すように、SparkContext は Spark の全機能にアクセスするための導管であり、JVM には1つの SparkContext しか存在しません。Sparkドライバプログラムは、これを使用してクラスタマネージャに接続し、通信を行い、Spark ジョブを投入します。これにより、Spark の設定パラメータをプログラムで調整することができます。また、SparkContext を介して、ドライバは SQLContext、HiveContext、StreamingContext などの他のコンテキストをインスタンス化して Spark をプログラムすることができます。

しかし、Apache Spark 2.0 では、SparkSession は単一のエントリーポイントから Spark の全ての機能にアクセスできるようになりました。DataFrame や Dataset、Catalogue、Spark Configuration などの Spark の機能へのアクセスがより簡単になるだけでなく、データを操作するためのコンテキストをサブセット化しています。

「Apache Spark 2.0で SparkSessions を使う方法」というブログ記事で詳しく説明されており、付属のノートには SparkSession プログラミングインターフェイスの使い方の例が記載されています。

Spark Deployment Modes Cheat Sheet

Spark は4つのクラスタデプロイモードをサポートしており、それぞれのモードで Spark のコンポーネントを Spark クラスタ内で動作させる場所に特徴があります。全てのモードの中で、1つのホスト上で動作するローカルモードは、学習や実験が最も簡単です。

初心者や中級者の開発者であれば、この精巧なマトリックスをすぐに知る必要はありません。ここにあるのは参考のためであり、リンクは追加情報を提供しています。さらに、ステップ 3 では、デバップスの観点から Spark アーキテクチャのあらゆる側面を深く掘り下げています。

MODE	DRIVER	WORKER	EXECUTOR	MASTER
Local	単一の JVM 上で実行	ドライバと同じ JVM 上で動作	ドライバと同じJVM上で動作	単一ホストでの実行
スタンドアロン	クラスタ内の任意のノードで実行可能	各ノード上の独自の JVM 上で実行	クラスタ内の各ワーカーは、独自の JVM を起動	マスターの開始場所を任意に割り当て
YARN（クライアント）	クラスタの一部ではないクライアント上で実行	YARN ノードマネージャ	YARN ノードマネージャのコンテナ	YARN のリソースマネージャは、YARN のアプリケーションマスターと連携して、Node Managers for Executors 上のコンテナを割り当て
YARN（クラスタ）	YARN のアプリケーションマスター内での実行	YARN クライアントモードと同じ	YARN クライアントモードと同じ	YARNクライアントモードと同じ
Mesos（クライアント）	Mesos クラスタの一部ではなく、クライアントマシン上での実行	Mesos スレーブ上での実行	Mesos スレーブ内のコンテナ	Mesos のマスター
Mesos（クラスタ）	Mesos のマスター内で実行	クライアントモードと同じ	クライアントモードと同じ	Mesos のマスター

表1. デプロイモードと各 Spark コンポーネントの実行場所を示したチートシート

Spark のアプリ、ジョブ、ステージ、タスク

Spark アプリケーションの解剖学は通常 Spark の操作で構成されます。Spark の RDD、DataFrame、Dataset の API を使ってデータセットを変換したり、アクションを実行したりします。例えば、Spark アプリで DataFrame や Dataset に対して collect() や take() などのアクションを実行すると、ジョブが作成されます。ジョブは単一または複数のステージに分解され、ステージはさらに個々のタスクに分割されます。多くの場合、複数のタスクが同じエグゼキュータ上で並列に実行され、それぞれがメモリ上で分割されたデータセットの単位を処理します。

このビデオでは、Sameer Farooqui 氏がそれぞれのステージについて詳しく説明しています。 彼は、Spark のジョブがどのようにして複数のステージに分解され、その後にタスクが追加され、Spark ワーカー上の実行者に分配されるかを説明しています。

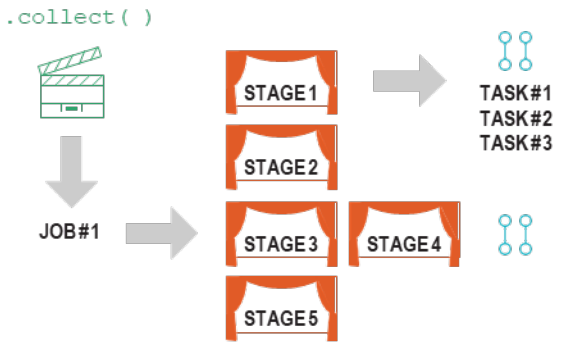
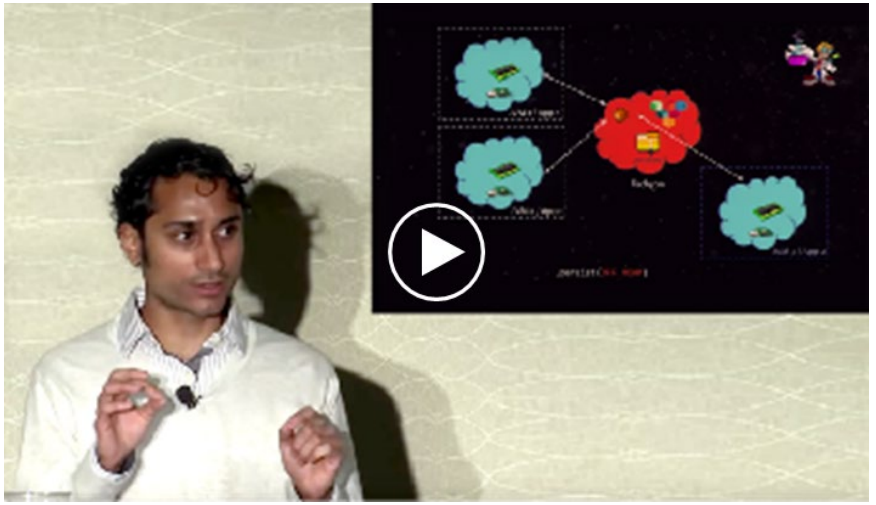


Fig 3. Anatomy of a Spark Application

ステップ 3：
高度な Apache Spark の
内部構造とコア

高度な Apache Spark 内部構造と Spark コア

Spark の全てのコンポーネントがどのように相互作用するかを理解し、Spark のプログラミングに習熟するためには、Spark のコアアーキテクチャを詳しく理解することが必要です。ステップ 2 で定義された重要な用語や概念を説明します。Spark Summit のトレーニングビデオでは、Spark の核心に触れることができます。



コアアーキテクチャ以外にも、以下のようなことを学びます。

- データをどのように分割し、変換し、 Spark クラスタ内の Spark ワーカーノード間で「Shuffle」と呼ばれるネットワーク転送中に転送するか
- 仕事がどのようにステージやタスクに分解されているか
- ステージが有向非周期グラフ（DAG）としてどのように構築されているか
- タスクがどのようにして分散実行のためにスケジュールされるか

ステップ 4.
DataFrame、データセット、Spark SQL の要点

DataFrame、データセット、Spark SQL の要点

ステップ2と3では、リンク先のビデオをご覧になった方は、RDD（Resilient Distributed Dataset）について知っているかもしれませんが、RDD は Spark のデータ抽象化の中心的な概念であり、DataFrame や Dataset を含む他の高レベルのデータ抽象化や API の基盤となるものです。

Apache Spark 2.0 では、RDD と Spark SQL エンジンに基づいて構築された DataFrame と Dataset が、高レベルで構造化された分散データの抽象化の中核を形成しています。これらがマージされ、Spark のライブラリやコンポーネント間で統一された API を提供しています。

データセットとDataFrame

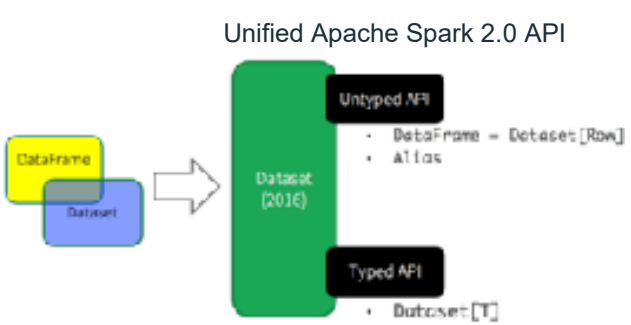


図3 Apache Spark にまたがる統一された API

DataFrame は Spark ではデータカラムと呼ばれ、データをどのように整理するかの構造やスキーマを定義します。この構成によって、データの処理、計算の表現、クエリの発行方法が決まります。例えば、データは 4 つの RDD パーティションに分散されていて、各パーティションには、”Time”、”Site”、”Req” という 3 つの名前付きカラムがあります。このように、名前付きカラムでデータにアクセスするための自然で直感的な方法を提供します。

DataFrame の構造

図4 カラムに名前を付けたDataFrameのサンプル

一方、データセットは、厳密なコンパイル時の型の安全性を提供するためにさらに一歩進んで、ランタイムではなくコンパイル時に特定のタイプのエラーが検出されます。

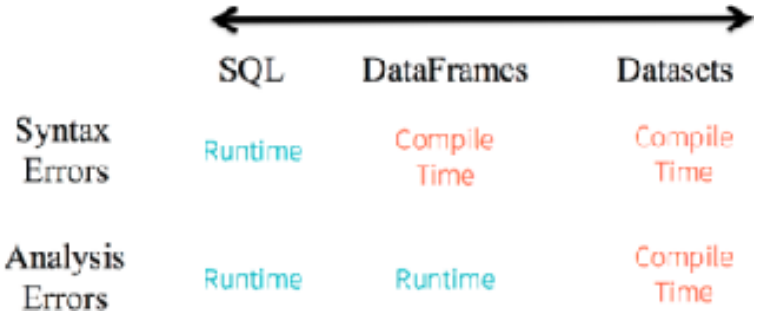


図5 DataFrameとデータセットで検出されたエラータイプのスペクトル

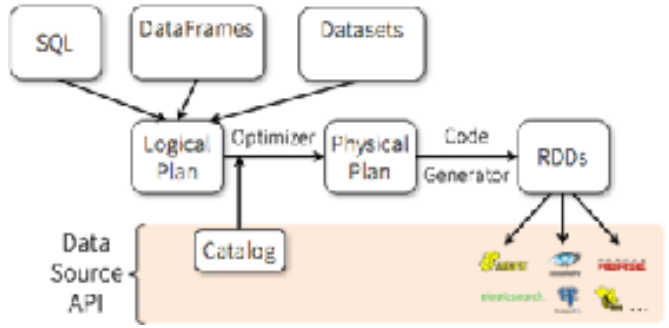


図6 DataFrame, Dataset, SQLでの高レベルの計算で実行されるジャーニー

データの構造やデータの種類によって、Spark は計算をどのように表現するか、どのような型付きカラムや型付き名前のフィールドにアクセスするか、どのようなドメイン固有の操作を使用するかを理解することができます。構造化されたデータや型付きデータをデータセットとして解析することで、Spark は Spark 2.0 の Catalyst オプティマイザーを使ってコードを最適化し、Project Tungsten を使って効率の良いバイトコードを生成します。

DataFrame とDataset は、高レベルのドメイン固有の言語 API を備えており、filter、sum、count、avg、min、max などの高レベルの演算子を使用してコードを表現することができます。Spark SQL、Python、Java、Scala、R Dataset/Dataframe API で計算を表現する場合でも、図6 に示すように、全ての実行計画が同じ Catalyst 最適化を受けているため、生成されるコードは同じです。

例えば、Scala でのこの高レベルのドメイン固有のコードや、SQL での同等のリレーショナルクエリは、同じコードを生成します。Person という名前の Dataset Scala オブジェクトと SQL テーブル "person“ を考えてみましょう。

```
// a dataset object Person with field names fname, lname, age, weight
// access using object notation
val seniorDS = peopleDS.filter(p=>p.age > 55)
// a dataframe with structure with named columns fname, lname, age, weight
// access using col name notation
val seniorDF = peopleDF.where(peopleDF(“age”) > 55)
// equivalent Spark SQL code
val seniorDF = spark.sql(“SELECT age from person where age > 35”)
```

なぜ Spark でデータを構造化することが重要なのか、DataFrame、Dataset、Spark SQL が効率的な Spark のプログラミング方法を提供しているのかを紹介しています。この Spark Summit では、Spark PMC でありコミッターでもある Michael Armbrust 氏が Spark の構造化の動機とメリットを語っています。

また、これらの技術資産では、DataFrame や Dataset、JSONファイルのような構造化データの処理や Spark SQL クエリの発行にどのように使用するかについても説明しています。

1. Apache Spark のデータセット入門
2. 3 つの API： RDD、DataFrame、Dataset
3. データセットと DataFrame ノートブック

ステップ 5.
GraphFrames によるグラフ処理

GraphFrames による グラフ処理

Spark には GraphX という名前の汎用 RDD ベースのグラフ処理ライブラリがあり、分散コンピューティングに最適化され、グラフアルゴリズムをサポートしていますが、いくつかの課題があります。Java や Python の API がなく、低レベルの RDD API をベースにしている。このような制約があるため、Project Tungsten や Catalyst Optimizer を通じて Dataframe で導入された最近のパフォーマンスや最適化を利用することができません。

対照的に、DataFrame ベースの GraphFrames は、これらの制約に対応しています。GraphX と似たようなライブラリを提供しますが、Java、Scala、Python の高レベルで表現力のある宣言的な API、Dataframe API を使った強力な SQL のようなクエリの発行機能、グラフの保存と読み込み、Apache Spark 2.0 のパフォーマンスとクエリの最適化を利用しています。さらに、GraphXとの統合も可能です。つまり、GraphFrame をシームレスに同等の GraphX 表現に変換することができます。

都市と空港の簡単な例を考えてみよう。図7 のダイヤグラムでは、都市の空港コードを表すために、全ての頂点は DataFrame の行として、全ての辺は DataFrame の行として、それぞれの名前付きの列と型付きの列として表すことができます。

これらの頂点と辺の DataFrame をまとめて GraphFrame とします。

グラフ

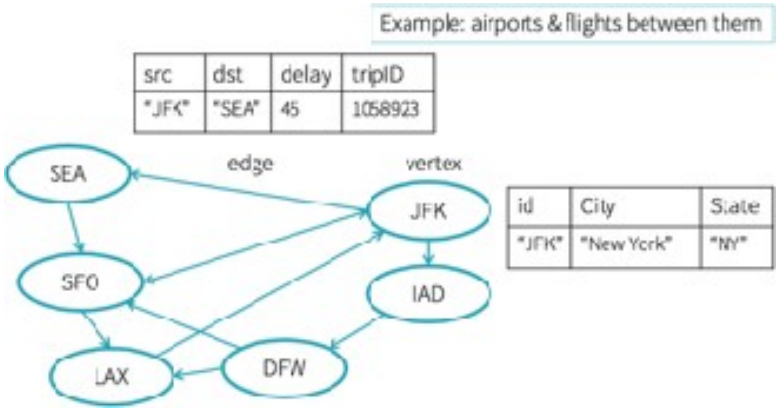


図7 GraphFrame で表現された都市のグラフ

この上の絵をプログラムで表現するとしたら、次のように書きます。

```
// create a Vertices DataFrame
val vertices = spark.createDataFrame(List(("JFK", "New York",
"NY"))).toDF("id", "city", "state")
// create a Edges DataFrame
val edges = spark.createDataFrame(List(("JFK", "SEA", 45,
1058923))).toDF("src", "dst", "delay", "tripID")
// create a GraphFrame and use its APIs
val airportGF = GraphFrame(vertices, edges)
// filter all vertices from the GraphFrame with delays greater an 30 mins
val delayDF = airportGF.edges.filter("delay > 30")
// Using PageRank algorithm, determine the Airport ranking of importance
val pageRanksGF =
airportGF.pageRank.resetProbability(0.15).maxIter(5).run()
display(pageRanksGF.vertices.orderBy(desc("pagerank")))
```

GraphFrames では、3 種類の強力なクエリを表現することができます。1つ目は、どのようなトリップに大きな遅延が発生する可能性が高いかといった、頂点やエッジに対する単純な SQL 型のクエリです。第二に、グラフ型のクエリ、例えば、何個の頂点に何個のエッジが入ってきて、何個のエッジが出てくるのかを表現することができます。最後に、頂点と辺の構造的なパターンやパスを提供し、グラフのデータセットからそれらのパターンを見つけることで、モチーフクエリを表現します。

さらに、GraphFrames は GraphX でサポートされている全てのグラフアルゴリズムを簡単にサポートしています。例えば、PageRank を使って重要な頂点を見つけたり、ソースからデスティネーションへの最短パスを決定したり、BFS（Breadth First Search）を実行したりすることができます。また、社会的なつながりを探るために、強くつながっている頂点を決定することもできます。

GraphFrames の Web セミナーでは、Spark Committer の Joseph Bradley 氏が、GraphFrames を使ったグラフ処理について、その動機や使いやすさ、DataFrame ベースの API の利点などを紹介しています。また、Web セミナーの一部であるノートブックのデモを通して、GraphFrames を使用して、前述のクエリやアルゴリズムのタイプを簡単に発行できることを学ぶことができます。

上記の Web セミナーを補完するために、以下の 2 つのブログとノートを用意し、DataFrame ベースの GraphFrames の入門編と実践編を紹介しています。

1. GraphFrames の紹介
2. Apache Spark のための GraphFrames を用いたオンタイムフライトパフォーマンス

Apache Spark 2.0 以降、Machine Learning MLlib や Streaming を含む多くの Spark コンポーネントでは、パフォーマンスの向上、使いやすさ、抽象度の高さ、構造などの理由から、同等の Dataframe API を採用する傾向が強まっています。必要に応じて、GraphX の代わりに GraphFrames を使用することもできます。以下に GraphX と GraphFrames の比較を簡単にまとめてみました。

GraphFrames と GraphX の比較

	GraphFrames	GraphX
Built on	DataFrames	RDDs
Languages	Scala, Java, Python	Scala
Use cases	Queries & algorithms	Algorithms
Vertex IDs	Any type (in Catalyst)	Long
Vertex/edge attributes	Any number of DataFrame columns	Any type (VD, ED)
Return types	GraphFrame or DataFrame	Graph[VD, ED], or RDD[Long, VD]

図8 比較チートシート

最後に、GraphFrames の高速化が続いていますが、Ankur Dave 氏による Spark Summit の講演では具体的な最適化が示されています。Spark 2.0 に対応した GraphFrames パッケージの新バージョンが Spark パッケージとして提供されています。

ステップ 6.
構造化ストリーミングによる継続的なアプリケーション

構造化ストリーミングによる継続的なアプリケーション

Spark の短い歴史のほとんどの間、Spark のストリーミングは、ストリーミングアプリケーションの記述を簡単にするために進化し続けてきました。今日、開発者が必要としているのは、ストリーム内の要素を変換するための単なるストリーミングプログラミングモデルだけではありません。その代わりに、リアルタイムでデータに連続的に反応するエンドツーエンドアプリケーションをサポートするストリーミングモデルが必要です。私たちはこれを、リアルタイムでデータに反応する継続的なアプリケーションと呼んでいます。

継続的なアプリケーションには多くの側面があります。例えば、バッチデータとリアルタイムデータの両方との対話、ストリーミング ETL の実行、バッチとストリームからのダッシュボードへのデータ提供、静的データセットとリアルタイムデータを組み合わせたオンライン機械学習の実行などがあります。現在、このようなファセットは、単一のアプリケーションではなく、別々のアプリケーションで処理されています。

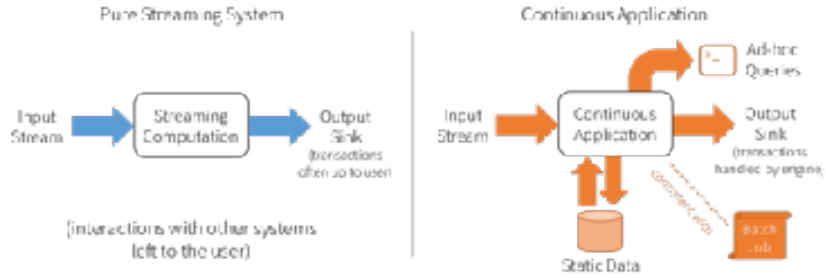


図9 従来のストリーミング対構造化ストリーミング

Apache Spark 2.0 は、継続的なアプリケーションを構築するための新しい高レベル API である Structured Streaming の基礎を築いた。

Apache Spark 2.1 はデータソースとデータシンクのサポートを拡張し、イベントタイム処理の透かしやチェックポイントなどのストリーミング操作を強化しました。

ストリームがストリームではない場合

構造化ストリーミングの中心となるのは、データのストリームをストリームとしてではなく、束縛されていないテーブルとして扱うという考え方です。ストリームから新しいデータが到着すると、Dataframe の新しい行が結合されていないテーブルに追加されます。

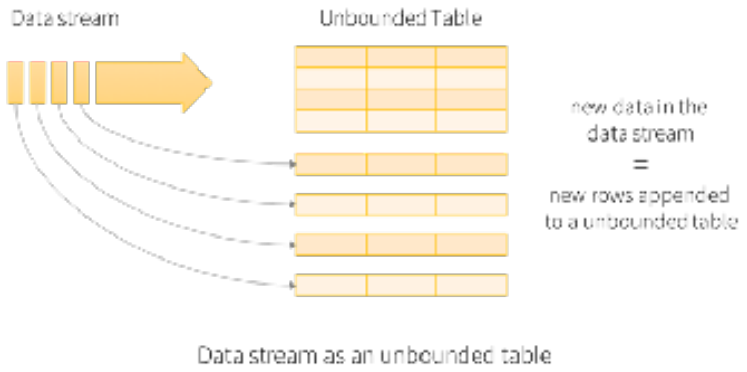


図10 DataFrame の束縛されないテーブルとしてのストリーム

そして、静的なテーブルの場合と同じように、束縛されていないテーブルに対して計算を実行したり、SQL タイプのクエリ操作を発行したりすることができます。このシナリオでは、開発者はストリーミング計算をバッチ計算のように表現することができ、データがストリームに到着すると Spark が自動的にインクリメンタルに実行します。これは強力ですね!

図11 ストリーミングとバッチの類似コード

DataFrame/Dataset API に基づいて、Structured Streaming API を使用するメリットは、図11 のコードにあるように、バッチ DataFrame に対する DataFrame/SQL ベースのクエリが、ストリーミングクエリに似ていることですが、ちょっとした変更点があります。バッチ版では、静的にバインドされたログファイルを読み込みますが、ストリーミング版では、バインドされていないストリームを読み込みます。このコードは一見単純に見えますが、複雑さは開発者には隠されており、フォールトトレランス、インクリメンタルクエリ実行、アイドエンプテンシー、正確な一度きりのセマンティクスのエンドツーエンド保証、順序外のデータ、透かしイベントなどの負荷を、基礎となるモデルと実行エンジンが担っています。これら全てのオーケストレーションをカバーしているのが

Spark Summit での Tathagata Das 氏のテクニカルトークで説明されている。さらに重要なのは、Structured Streaming を用いたストリーミング ETL がいかに従来の ETL を省略するかを実演することでケースを説明していることです。

データソース

Structure Streaming のデータソースは、データを生成したり、読み込んだりできるエンティティを指します。Spark 2.x は 3 つのビルトインデータソースをサポートしています。

ファイルソース

ディレクトリやファイルは、ローカル ドライブ、HDFS、または S3 バケット上でデータ ストリームとして機能します。DataStreamReader インターフェイスを実装したこのソースは、avro、JSON、テキスト、CSV などの一般的なデータ形式をサポートしています。それ以降

ソースはリリースごとに進化を続けており、追加のデータ形式やオプションについては最新のドキュメントをチェックしてください。

Apache Kafka ソース

Apache Kafka 0.10.0 以降に準拠したこのソースでは、構造化ストリーミング API が Kafka の全てのセマンティクスに準拠して、サブスクライブされたトピックからデータをポーリングしたり、読み込んだりすることができます。この Kafka 統合ガイドでは、構造化ストリーミング API とこのソースの使用方法について詳しく説明しています。

ネットワークソケットソース

デバッグやテストに非常に便利なソースで、ソケット接続から UTF-8 テキストデータを読み取ることができます。このソースはテスト用にのみ使用されるため、他の 2 つのソースのようにエンドツーエンドの耐障害性を保証するものではありません。

簡単なコード例を見るには、「データ ソースとシンク」の「構造化されたストリーミング」ガイドのセクションを参照してください。

データシンク

データシンクは、処理されたデータや変換されたデータを書き込める場所です。Spark 2.1以降、3つの組み込みシンクがサポートされており、ユーザー定義のシンクはforeachインターフェースを使って実装できます。

ファイルシンク

ファイルシンクは、名前が示すように、ローカルファイルシステム上の指定されたディレクトリ内のディレクトリまたはファイルであり、HDFSまたはS3バケットは、処理または変換されたデータが着陸することができるリポジトリとして機能することができます。

Foreach シンク

アプリケーション開発者によって実装された ForeachWriter インタフェースを使用すると、処理されたデータや変換されたデータを任意の宛先に書き込むことができます。例えば、NoSQL や JDBC シンクへの書き込み、リスニングソケットへの書き込み、外部サービスへの REST コールの呼び出しなどが考えられます。開発者としては、open()、process()、close()の3つのメソッドを実装します。

コンソールシンク

主にデバッグ目的で使用され、ストリーミングアプリケーションのトリガーが起動されるたびに、出力をコンソール/標準出力にダンプします。ドライバ側でメモリ使用量が多いので、データ量が少ない場合のデバッグ目的にのみ使用してください。

メモリーシンク

コンソールシンクと同様に、データがインメモリテーブルとして保存されるデバッグ目的にのみ機能します。可能であれば、このシンクはデータ量が少ない場合にのみ使用してください。

DataFrame と Dataset のストリーミング操作

DataFrame と Dataset 上での選択、投影、集約に関する操作のほとんどが Structured Streaming API でサポートされていますが、サポートされていないものはいくつかあります。

例えば、デバイスデータのストリームを DataFrame に読み込んだ後、これらの操作を実行するシンプルな Python コードは以下のようになります。

```
devicesDF = ... # streaming DataFrame with IOT device data with schema
{ device: string, type: string, signal: double, time: DateType }
# Select the devices which have signal more than 10
devicesDF.select("device").where("signal > 10")
# Running count of the number of updates for each device
type  devicesDF.groupBy("type").count()
```

イベント時間の集計とウォーターマーク

DStreams にはなかった重要な操作が Structured Streaming で利用できるようになりました。時間軸上でのウィンドウ処理では、データレコードがシステムに受信された時刻ではなく、そのデータレコード内でイベントが発生した時刻でデータを処理することができます。ウィンドウ処理は、groupBy 操作と同様に分類されるため、groupBy 操作と同様にウィンドウ処理を実行できます。このことを説明するガイドの短い抜粋を以下に示します。

```
import spark.implicits._
val words = ... // streaming DataFrame of schema { timestamp:
Timestamp,  word: String }
// Group the data by window and word and compute the count of each group
val deviceCounts = devices.groupBy( window($"timestamp", "10 minutes",
"5  minutes"), $"type"
).count()
```

特にストリーミングデータとそれに伴う遅延は、データが常に連続して到着するとは限らないため、順序外れや遅延のデータを処理する機能は重要な機能です。データレコードの到着が遅すぎたり、順番が狂っていたり、ある時間のしきい値を超えて蓄積され続けていたりするとどうなるのでしょうか？

透かしとは、データのイベント時間が無用とみなされる処理時間間隔にマークを付けたり、しきい値を指定したりすることができるスキームのことです。さらに、破棄することも可能である。ストリーミングエンジンは、そのしきい値や間隔内の遅延データだけを効率的かつ効果的に保持することができます。構造化ストリーミング API の仕組みや表現方法については、プログラミングガイドの透かしのセクションを参照してください。この短いスニペット API の呼び出しと同じくらい簡単です。

```
val deviceCounts = devices
    .withWatermark("timestamp", "15 minutes")
    .groupBy(window($"timestamp", "10 minutes", "5 minutes"),
        $"type")
    .count()
```

次のステップ

構造化ストリーミングについて深く学んだ後は、Structure Streaming Programming Model を読んでみてください。開発者やユーザーは、これらの複雑さを心配する必要はありません。基礎となるストリーミングエンジンは、フォールトトレランス、エンドツーエンドの信頼性、および正しさの保証の責任を負います。

Spark のコミッターである Tathagata Das 氏から直接 Structured Streaming について学び、付属のノートブックを使用した Structured Streaming 継続的アプリケーションのハンズオン体験ができます。

追加のワークショップノートでは、Structured Streaming API を使用して IoT デバイスのストリーミングデータを処理する方法を説明します。

Apache Spark 2.0 の Structured Streaming API：ストリーミングのための新しい高レベル API

同様に、Structured Streaming Programming Guide では、サポートされているシンクとソースの使用方法を簡単な例で紹介しています。

構造化ストリーミングプログラミングガイド

ステップ 7.
人間のための機械学習

人間のための機械学習

人間のレベルでは、機械学習とは、統計的学習技術とアルゴリズムを大規模なデータセットに適用してパターンを特定し、これらのパターンから確率的な予測を行うことです。モデルの簡略化された見方は、数学的な関数 f(x) です。大規模なデータセットを入力として、関数 f(x) を繰り返しデータセットに適用して、予測を伴う出力を生成します。モデル関数は、例えば、線形回帰や決定木などの様々な機械学習アルゴリズムのいずれかです。

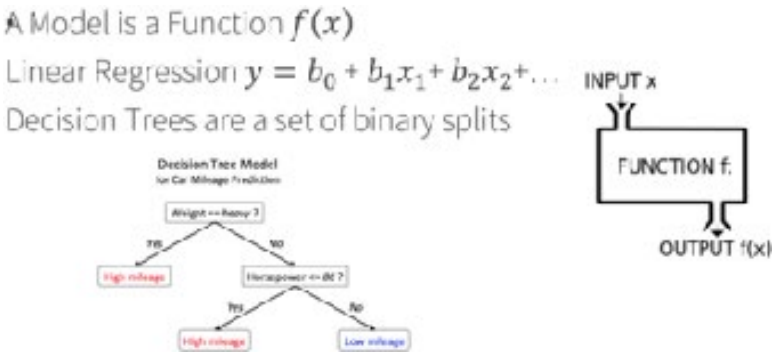


図12 数理関数としてのモデル

Apache Spark のコアコンポーネントライブラリである MLlib は、ロジスティック回帰から k-means、クラスタリングまで、多数の教師あり・教師なし学習アルゴリズムを提供しており、これらの数理モデルを構築することができます。

キーワードと機械学習アルゴリズム

機械学習の入門的な重要用語については、Matthew Mayo の Machine Learning Key Terms, Explained が、次のページにある Databricks の Web セミナーで説明されたいくつかの概念を理解するための貴重な参考文献となっています。また、こちらのリンク先には、機械学習アルゴリズムに関するドキュメントと一緒に、機械学習を支える概念を、Databricks ノートブックのコード例とともに紹介しているハンズオン入門ガイドがあります。

機械学習のパイプライン

Apache Spark の DataFrame ベースの MLlib は、モデルとユーティリティとしてアルゴリズムのセットを提供し、データサイエンティストが機械学習パイプラインを簡単に構築できるようにします。scikit-learn プロジェクトから借用した MLlib パイプラインは、開発者が複数のアルゴリズムを 1 つのパイプラインやワークフローにまとめることを可能にします。

通常、機械学習アルゴリズムの実行には、前処理、特徴抽出、モデルフィッティング、検証の段階を含む一連のタスクを実行します。Spark 2.0 では、このパイプラインは Spark がサポートしている言語に依存せずに保持され、再ロードすることができます（下記のブログリンクを参照）。

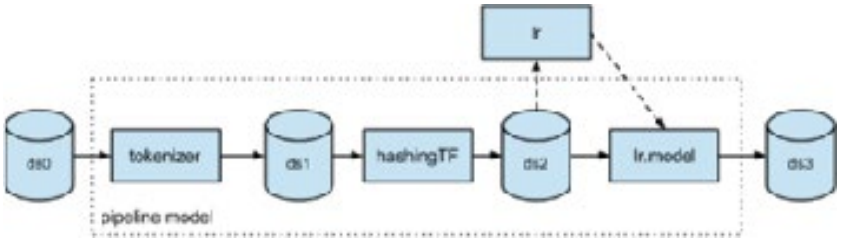


図13 機械学習のパイプライン

Apache Spark MLlib の Web セミナーでは、機械学習、Spark MLlib、Spark の機械学習のユースケースの概要、Python、pandas、scikit-learn、R などの一般的なデータサイエンスツールが MLlib とどのように統合されているかを説明します。

さらに、実践的な体験ができる 2 冊のノートと、機械学習モデルの持続性に関するブログでは、なぜ、何を、どのように機械学習が高度なアナリティクスにおいて重要な役割を果たしているのかについての洞察を得ることができます。

1. Apache Spark を使った scikit-learn の自動スケーリング
2. 2015 年州別住宅価格中央値
3. 人口と住宅価格の中央値。単一変数を用いた線形回帰
4. Apache Spark 2.0 での機械学習モデルの保存と読み込み

これらのガイドされた手順に沿って、全てのビデオを見て、ブログを読んで、付属のノートを試してみれば、開発者として Apache Spark 2.x を学ぶ道を歩むことができると信じています。

ステップ 8.
高信頼性データレイクと
データパイプライン

データレイクによる
データ信頼性の課題

失敗した書き込み

データを書き込んでいる本番ジョブが大規模な分散環境では避けられない障害が発生すると、部分的な書き込みや複数回の書き込みによってデータが破損する可能性があります。必要なのは、書き込みが完全に行われるか、全く行われないか（複数回の書き込みを行わず、スプリアスデータを追加しない）のいずれかを確実に行うことができるメカニズムです。失敗したジョブは、クリーンな状態に回復するためにかなりの負担を強いられる可能性があります。

一貫性の欠如

複雑なビッグデータ環境では、バッチデータとストリーミングデータの両方を考慮することに興味があるかもしれません。データが追加されている間にデータを読み込もうとすると、一方では新しいデータのインジェストを継続したいという欲求があります。

一方で、データを読んでいる人は誰でも一貫した見方を好みます。これは特に、複数のリーダやライタがいる場合に問題となります。もちろん、書き込みが完了している間に読み取りアクセスを停止したり、読み取り中に書き込みアクセスを停止したりすることは望ましくないし、現実的ではありません。

スキーマの不一致

大規模な最新のビッグデータ環境によく見られるように、複数のソースからコンテンツをインジェストする場合、同じデータが同じ方法でエンコードされていること、つまりスキーマが一致していることを確認することは困難な場合があります。同様の問題は、データ要素のフォーマットがデータエンジニアリングチームに通知されずに変更された場合にも発生します。どちらも低品質で一貫性のないデータになる可能性があり、使い勝手を向上させるためにクリーンアップが必要になります。スキーマを観察して強制する機能は、これを軽減するのに役立つでしょう。

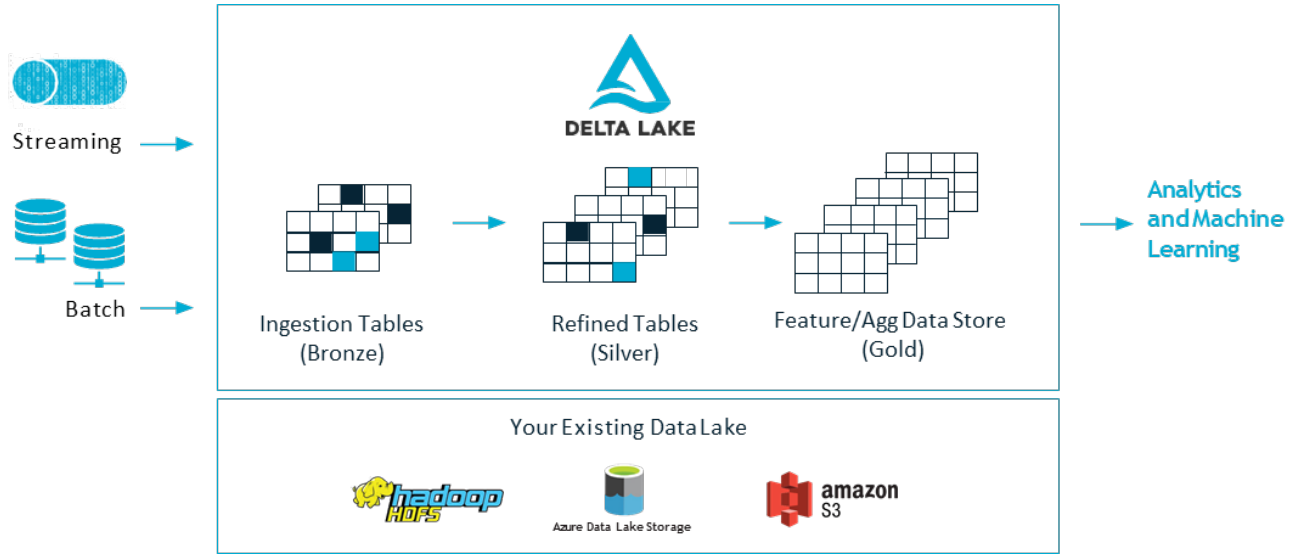
Delta Lake：
新しいストレージ層

ACID取引

データレイクは通常、複数のデータパイプラインが同時にデータの読み書きを行っており、データエンジニアはトランザクションがないため、データの整合性を確保するために面倒なプロセスを経なければなりません。Delta Lake は、データレイクに ACID トランザクションをもたらします。最強の分離レベルであるシリアライズ性を提供します。

スケーラブルなメタデータ処理

ビッグデータでは、メタデータ自体も「ビッグデータ」になり得る。Delta Lake はメタデータをデータと同じように扱い、Spark の分散処理能力を活用して全てのメタデータを処理します。その結果、Delta Lake は何十億ものパーティションやファイルを持つペタバイト規模のテーブルを簡単に扱えるようになりました。



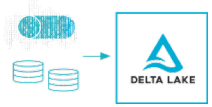
Delta Lake：
新しいストレージ層

タイムトラベル（データのバージョン管理）

デルタレイクはデータのスナップショットを提供し、開発者は監査やロールバック、実験の再現のために、以前のバージョンのデータにアクセスしたり、戻したりすることができます。バージョニングの詳細については、こちらのブログ「大規模データレイクのためのデルタタイムトラベルの導入」をお読みください。

オープンフォーマット

Delta Lake の全てのデータは Apache Parquet 形式で保存されており、Parquet に固有の効率的な圧縮とエンコーディングスキームを利用できます。

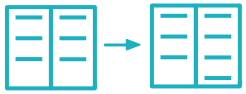


統一されたバッチとストリーミングのソースとシンク

Delta Lake のテーブルは、バッチテーブルであると同時に、ストリーミングのソースとシンクでもあります。ストリーミングデータのインジェスト、バッチヒストリカルのバックフィル、インタラクティブなクエリは全て、箱から出してすぐに使えるようになっています。

スキーマの施行

Delta Lake は、スキーマを指定して強制する機能を提供します。これにより、データ型が正しく、必要なカラムが存在することを確認し、不正なデータがデータ破損の原因となるのを防ぐことができます。



スキーマの進化

ビッグデータは常に変化し続けています。Delta Lake は、面倒な DDL を必要とせず、自動的に適用できるテーブルスキーマへの変更を可能にします。

Apache Spark API と 100% 互換

Delta Lake は、一般的に使用されているビッグデータ処理エンジンである Spark と完全に互換性があるため、開発者は既存のデータパイプラインを最小限の変更で使用することができます。

Delta Lake を使ってみる

Delta を使い始めるのは簡単です。具体的には、Delta テーブルを作成するには Parquet の代わりに Delta を指定するだけです。

Parquet を

```
dataframe
.write
.format("parquet")
.save("/data")
```

…delta に変更するだけ

```
dataframe
.write
.format("delta")
.save("/data")
```

DELTA LAKE

クイックスタートとサンプルのノーブックを使って、今日から Delta Lake をお試しいただけます。

以下のブログでは、デルタに関する事例やニュースを共有しています。

- 大規模データレイクのためのデルタタイムトラベルの導入
- Databricks Delta を使用したリアルタイムアトリビューションパイプラインの構築
- Databricks Delta を利用したストリーミング株式データ分析の簡素化

詳細はドキュメントを参照してください。

結論

Databricks のミッションは、データサイエンティスト、エンジニア、ビジネスユーザーが利用しやすい環境で、組織がデータ問題にすぐに取り組めるように、データアナリティクスを統一することです。この電子ブックに掲載されているブログ記事、ノートブック、ビデオの技術トークのコレクションが、お客様の最大のデータ問題を解決し、チームが高品質のデータを活用する速度を加速させるための洞察力とツールを提供してくれることを願っています。この電子書籍の技術的な内容をお楽しみいただけたのであれば、シリーズの過去の書籍をチェックしたり、Databricks の Delta Lake と Apache Spark のエキスパートによる技術的なヒント、ベストプラクティス、ケーススタディを Databricks ブログでご覧ください。

Databricks を無料でお試しいただけます。詳しくはこちらをご覧ください。