aws

Apache Spark 任務效能調校 AWS Glue 的最佳實務

: Apache Spark 任務效能調校 AWS Glue 的最佳實務

Table of Contents

簡介 1
關鍵主題 2
Architecture 2
彈性分散式資料集 3
延遲評估 4
Spark 應用程式術語 5
平行處理 6
Catalyst 最佳化工具 7
調查效能問題 9
使用 Spark UI 識別瓶頸 9
調校效能的策略 11
效能調校的基線策略 11
Spark 任務效能的調校實務 11
擴展叢集容量 12
CloudWatch 指標 12
Spark 使用者介面 13
使用最新版本 14
減少資料掃描量 15
CloudWatch 指標 15
Spark 使用者介面 16
平行化任務 24
CloudWatch 指標 24
Spark 使用者介面 25
最佳化隨機播放 29
CloudWatch 指標 30
Spark 使用者介面 31
將規劃開銷降至最低 38
CloudWatch 指標 38
Spark 使用者介面 39
最佳化使用者定義的函數 40
標準 Python UDF 41
向量化 UDF 41
Spark SQL 42
將 panda 用於大數據 43

資源 44
文件歷史紀錄 45
..... xlvi

Apache Spark 任務效能調校 AWS Glue 的最佳實務

Roman Myers、Takashi Onikura 和 Noritaka Sekiyama，Amazon Web Services (AWS)

2023 年 12 月 (文件歷史記錄)

AWS Glue 提供不同的調校效能選項。本指南定義了 Apache Spark AWS Glue 調校的關鍵主題。然後，它提供基準策略，供您 AWS Glue 在調校 Apache Spark 任務時遵循這些策略。使用本指南了解如何透過解譯 中可用的指標來識別效能問題 AWS Glue。然後整合策略來解決這些問題、將效能最大化並將成本降至最低。

本指南涵蓋下列調校實務：

- 擴展叢集容量
- 使用 AWS Glue 最新版本
- 減少資料掃描量
- 平行處理任務
- 將規劃開銷降至最低
- 最佳化隨機播放
- 最佳化使用者定義的函數

Apache Spark 中的關鍵主題

本節說明用於調校 Apache Spark 效能 AWS Glue 的 Apache Spark 基本概念和關鍵主題。在討論真實世界調校策略之前，請務必了解這些概念和主題。

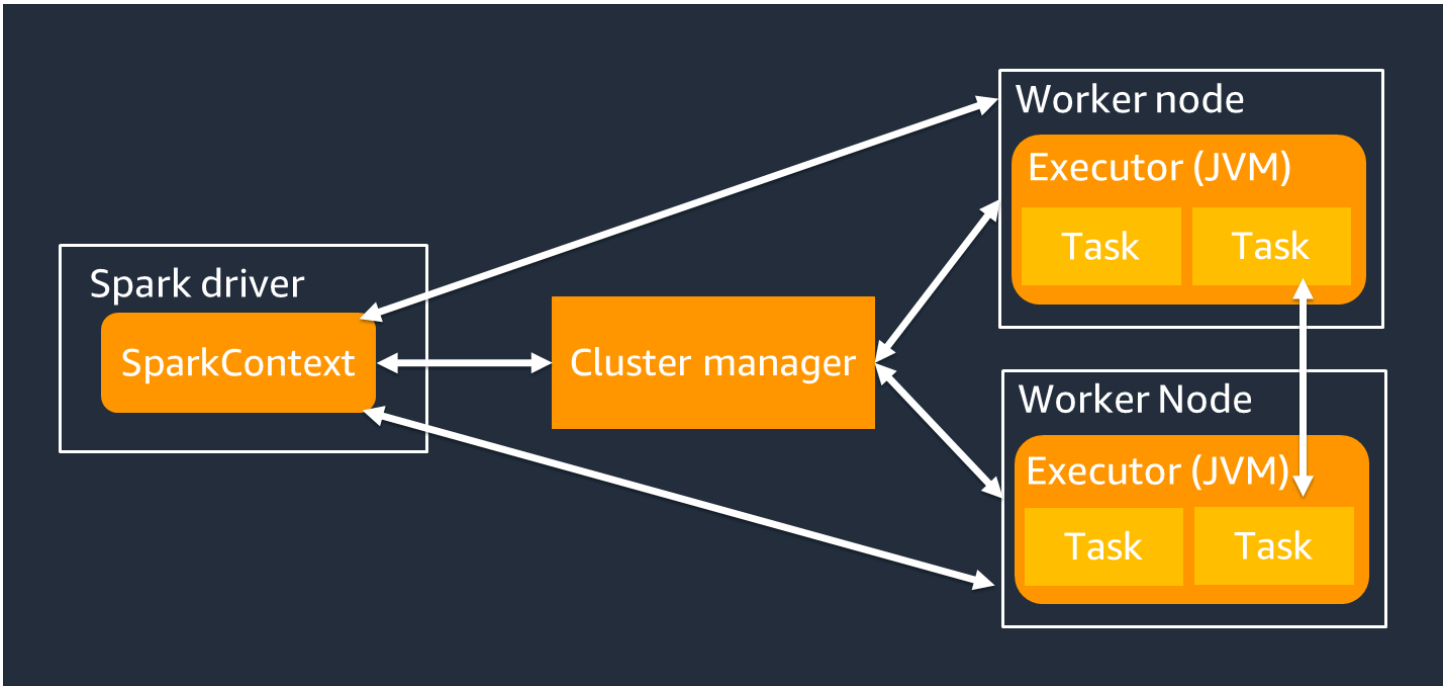
Architecture

Spark 驅動程式主要負責將您的 Spark 應用程式分割為可在個別工作者上完成的任務。Spark 驅動程式有下列責任：

- 在程式碼`main()`中執行
- 產生執行計畫
- 搭配叢集管理員佈建 Spark 執行器，以管理叢集上的資源
- 排程任務並請求 Spark 執行器的任務
- 管理任務進度和復原

您可以使用 `SparkContext` 物件來與 Spark 驅動程式互動，以進行任務執行。

Spark 執行器是一種工作者，用於保存從 Spark 驅動程式傳遞的資料和執行中的任務。Spark 執行器的數量會隨著叢集的大小而增加和減少。

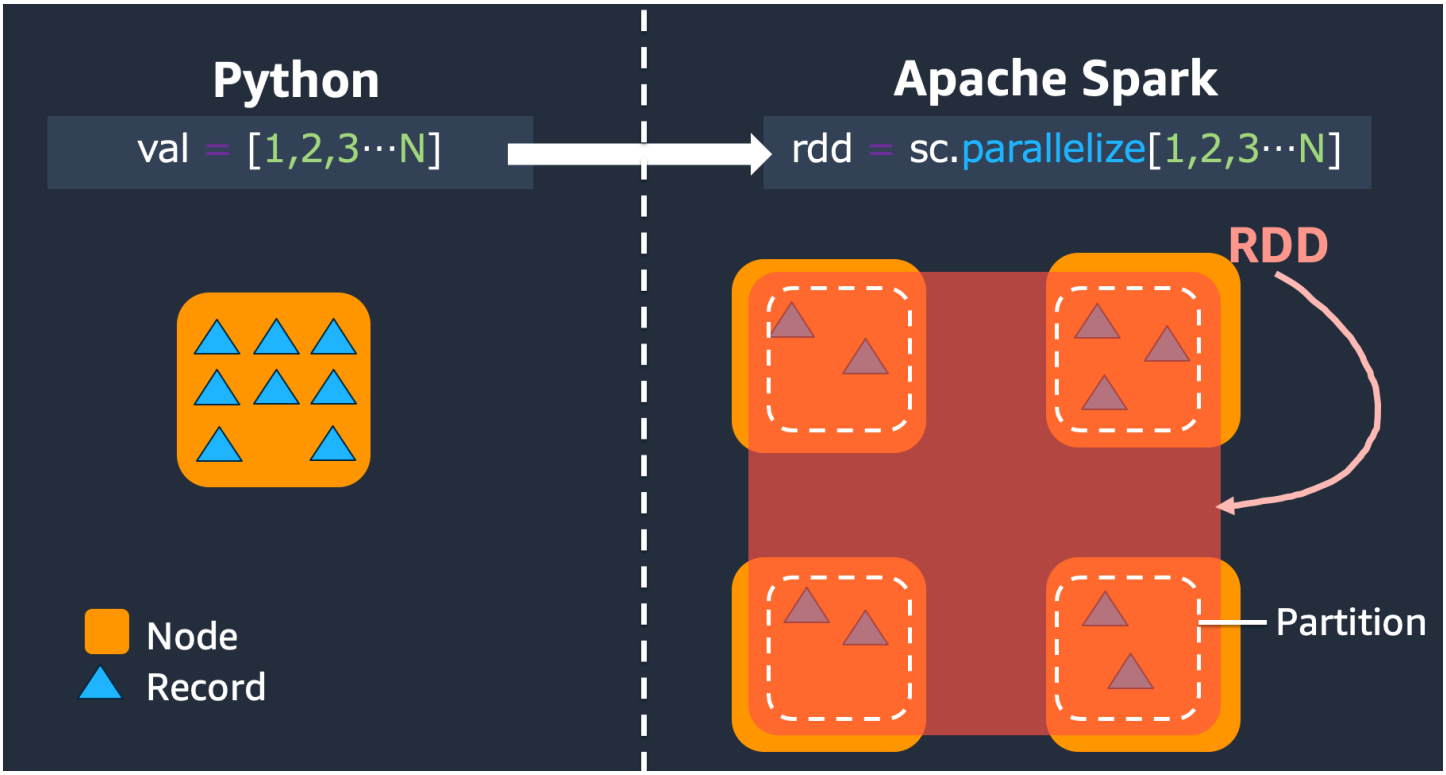


> **Note**
>
> Spark 執行器具有多個槽，以便平行處理多個任務。根據預設，Spark 支援每個虛擬 CPU (vCPU) 核心一個任務。例如，如果執行器有四個 CPU 核心，則可以執行四個並行任務。

彈性分散式資料集

Spark 會執行在 Spark 執行器之間存放和追蹤大型資料集的複雜任務。當您編寫 Spark 任務的程式碼時，不需要考慮儲存體的詳細資訊。Spark 提供彈性分散式資料集 (RDD) 抽象，這是可以平行操作的元素集合，並可分割叢集的 Spark 執行器。

下圖顯示 Python 指令碼在一般環境中執行時，以及在 Spark 架構 (PySpark) 中執行時，如何在記憶體中存放資料的差異。



- Python – 在 Python 指令碼`val = [1,2,3...N]`中寫入 會將資料保留在執行程式碼的單一電腦上的記憶體中。
- PySpark – Spark 提供 RDD 資料結構，以載入和處理分散在多個 Spark 執行器上的記憶體的資料。您可以使用 等程式碼產生 RDD`rdd = sc.parallelize[1,2,3...N]`，Spark 可以在多個 Spark 執行器之間自動分配和保留記憶體中的資料。

在許多 AWS Glue 任務中，您可以透過 DynamicFrames 和 Spark DataFrames 使用 RDDs AWS Glue 。這些摘要可讓您定義 RDD 中資料的結構描述，並使用該額外資訊執行更高階的任務。由於它們在內部使用 RDDs，因此資料會以透明的方式分佈並載入到下列程式碼中的多個節點：

- DynamicFrame

```
dyf= glueContext.create_dynamic_frame.from_options(
    's3', {"paths": [ "s3://<YourBucket>/<Prefix>/"]},
    format="parquet",
    transformation_ctx="dyf"
)
```

- DataFrame

```
df = spark.read.format("parquet")
    .load("s3://<YourBucket>/<Prefix>")
```

RDD 具有下列功能：

- RDDs由分成多個部分的資料組成，稱為分割區。每個 Spark 執行器都會在記憶體中存放一或多個分割區，而資料會分散到多個執行器。
- RDDs 是不可變的，這表示它們在建立之後無法變更。若要變更 DataFrame，您可以使用轉換，如下節所定義。
- RDDs會跨可用節點複寫資料，以便從節點故障中自動復原。

延遲評估

RDDs支援兩種類型的操作：從現有資料集建立新資料集的轉換，以及在資料集上執行運算後將值傳回驅動程式的 動作。

- 轉換 – 由於 RDDs 不可變，因此您只能使用轉換來變更它們。

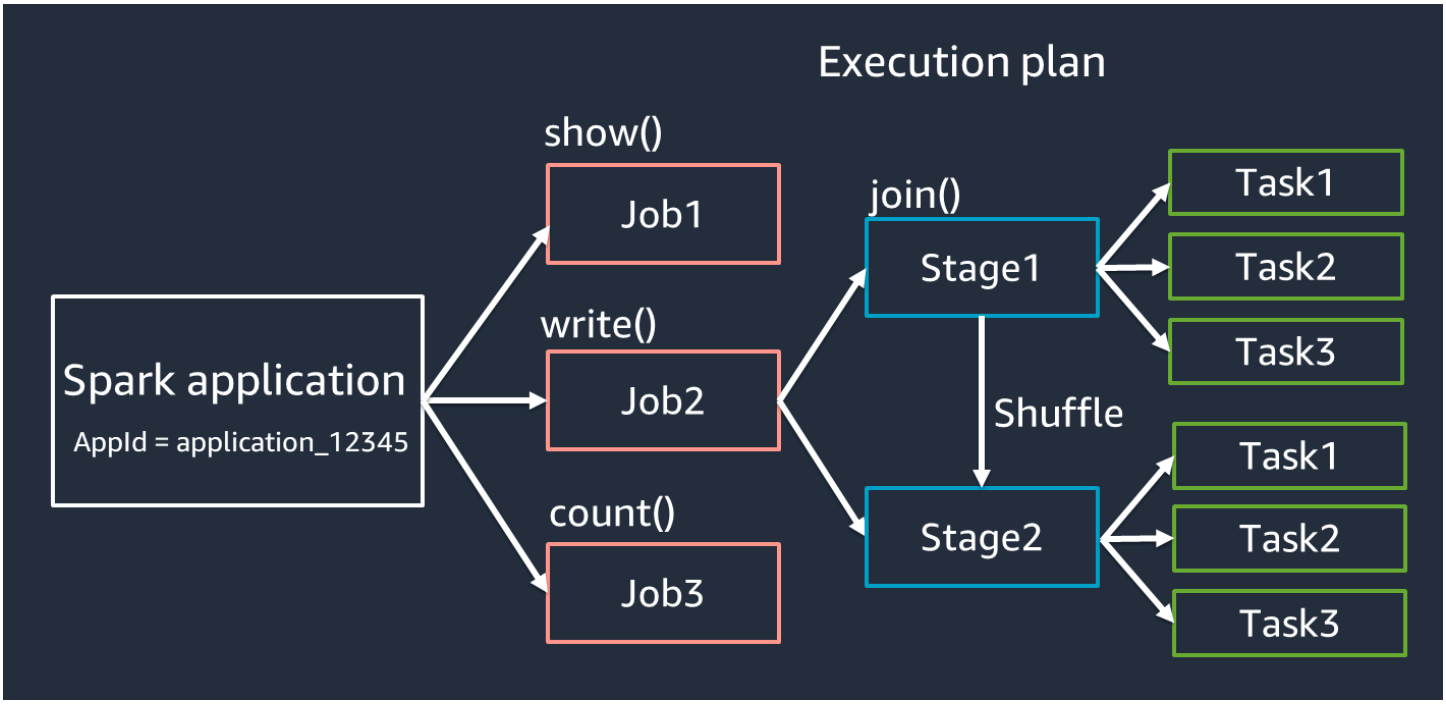
  例如， `map` 是一種轉換，透過 函數傳遞每個資料集元素，並傳回代表結果的新 RDD。請注意，`map`方法不會傳回輸出。Spark 會存放未來的抽象轉換，而不是讓您與結果互動。在您呼叫 動作之前，Spark 不會對轉換採取行動。
- 動作 – 使用轉換，您可以建立邏輯轉換計畫。若要啟動運算，請執行 動作，例如 `write`、`show`、`count`或 `collect`。

Spark 中的所有轉換都是延遲的，因為它們不會立即運算其結果。反之，Spark 會記住套用至某些基本資料集的一系列轉換，例如 Amazon Simple Storage Service (Amazon S3) 物件。只有在動作需要將結果傳回給驅動程式時，才會計算轉換。此設計可讓 Spark 更有效率地執行。例如，假設透過`map`轉換建立的資料集僅由大幅減少資料列數量的轉換耗用，例如 `reduce`。然後，您可以將經過兩個轉換的較小資料集傳遞給驅動程式，而不是傳遞較大的映射資料集。

Spark 應用程式術語

本節涵蓋 Spark 應用程式術語。Spark 驅動程式會建立執行計畫，並以數個抽象概念控制應用程式的行為。下列術語對於使用 Spark UI 進行開發、偵錯和效能調校非常重要。

- 應用程式 – 根據 Spark 工作階段 (Spark 內容) 。由唯一 ID 識別，例如 `<application_XXX>`。
- 任務 – 根據 為 RDD 建立的動作。任務包含一或多個階段。
- 階段 – 根據 為 RDD 建立的隨機播放 。階段包含一或多個任務。隨機播放是 Spark 重新分配資料的機制，因此會在 RDD 分割區之間以不同的方式分組。某些轉換，例如 `join()`，需要隨機播放。最佳化隨機播放調校實務中會更詳細地討論隨機播放。
- 任務 – 任務是 Spark 排定的最小處理單位。系統會為每個 RDD 分割區建立任務，而任務數量是階段中同時執行的最大數量。

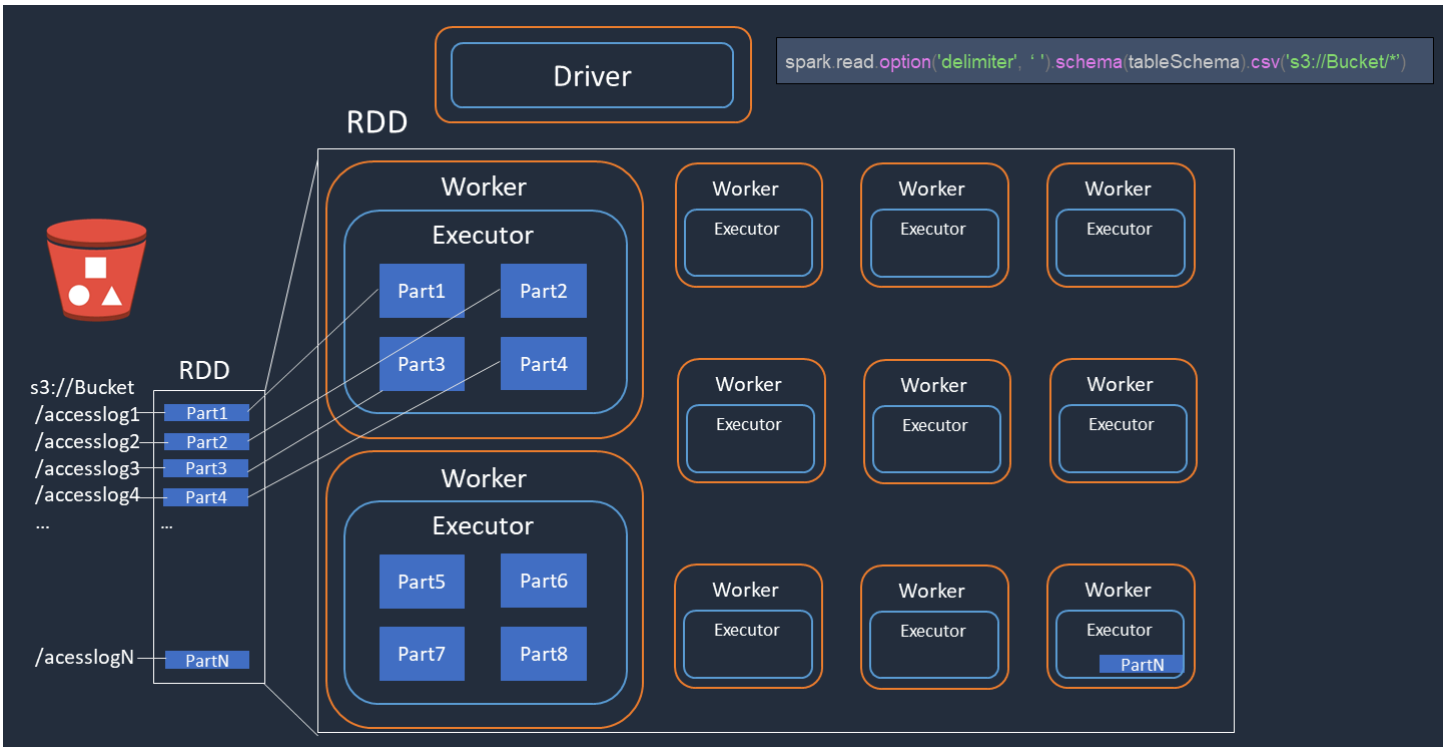


> **Note**
>
> 最佳化平行處理時，任務是最重要的考量事項。任務數量隨 RDD 數量而擴展

平行處理

Spark 會平行處理載入和轉換資料的任務。

請考慮您在 Amazon S3 上執行存取日誌檔案 （名為 `accesslog1 ... accesslogN`) 的分散式處理的範例。下圖顯示分散式處理流程。

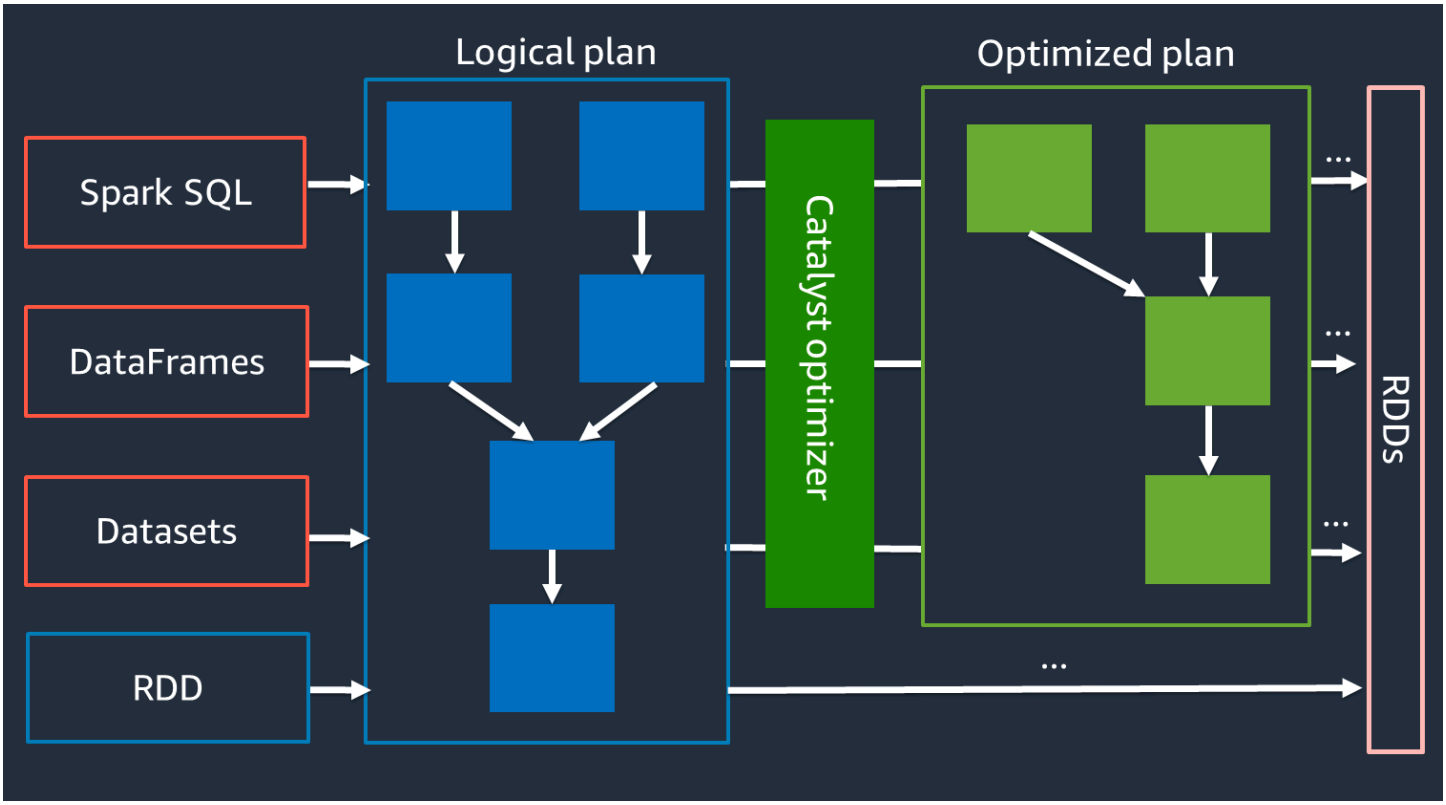


1. Spark 驅動程式會建立執行計畫，以便在多個 Spark 執行器之間進行分散式處理。
2. Spark 驅動程式會根據執行計畫指派每個執行器的任務。根據預設，Spark 驅動程式會為每個 S3 物件 () 建立 RDD 分割區 （每個分割區對應於 Spark 任務）`Part1 ... N`。然後，Spark 驅動程式會將任務指派給每個執行器。
3. 每個 Spark 任務都會下載其指派的 S3 物件，並將其存放在 RDD 分割區的記憶體中。如此一來，多個 Spark 執行器會平行下載和處理其指派的任務。

如需分割區初始數量和最佳化的詳細資訊，請參閱平行處理任務一節。

Catalyst 最佳化工具

在內部，Spark 使用稱為 Catalyst 最佳化工具的引擎來最佳化執行計畫。Catalyst 具有查詢最佳化工具，您可以在執行高階 Spark APIs 時使用，例如 Spark SQL、DataFrame 和資料集，如下圖所述。



由於 Catalyst 最佳化工具無法直接與 RDD API 搭配使用，因此高階 APIs 通常比低階 RDD API 更快。對於複雜的聯結，Catalyst 最佳化工具可以透過最佳化任務執行計畫來大幅改善效能。您可以在 Spark UI 的 SQL 索引標籤上查看 Spark 任務的最佳化計劃。

自適應查詢執行

Catalyst 最佳化工具會透過稱為自適應查詢執行的程序來執行執行期最佳化。自適應查詢執行會使用執行時間統計資料，在任務執行時重新最佳化查詢的執行計畫。自適應查詢執行針對效能挑戰提供數種解決方案，包括合併隨機播放後分割區、將排序合併聯結轉換為廣播聯結，以及扭曲聯結最佳化，如下列各節所述。

自適應查詢執行可在 AWS Glue 3.0 和更新版本中使用，且預設會在 AWS Glue 4.0 (Spark 3.3.0) 和更新版本中啟用。您可以在程式碼`spark.conf.set("spark.sql.adaptive.enabled", "true")`中使用 來開啟和關閉自適應查詢執行。

Coalescing 隨機播放後分割區

此功能會根據map輸出統計資料，減少每次隨機播放後的 RDD 分割區（餘音）。其可簡化執行查詢時隨機播放分割區編號的調校。您不需要設定隨機分割號碼以符合您的資料集。在您擁有足夠大的隨機播放分割區初始數量之後，Spark 可以在執行時間選擇適當的隨機播放分割區編號。

當 `spark.sql.adaptive.enabled`和 都設為 true 時`spark.sql.adaptive.coalescePartitions.enabled`，會啟用並行隨機播放後分割區。如需詳細資訊，請參閱 Apache Spark 文件。

將排序合併聯結轉換為廣播聯結

此功能會在您聯結兩個大小非常不同的資料集時識別，並根據該資訊採用更有效率的聯結演算法。如需詳細資訊，請參閱 Apache Spark 文件。最佳化隨機播放區段會討論聯結策略。

扭曲聯結最佳化

資料扭曲是 Spark 任務最常見的瓶頸之一。它描述了資料偏向特定 RDD 分割區（因此是特定任務）的情況，這會延遲應用程式的整體處理時間。這通常會降低聯結操作的效能。扭曲聯結最佳化功能會動態處理排序合併聯結中的扭曲，方法是將扭曲的任務分割（並視需要複寫）為大致均勻大小的任務。

當 `spark.sql.adaptive.skewJoin.enabled` 設為 true 時，會啟用此功能。如需詳細資訊，請參閱 Apache Spark 文件。最佳化隨機播放區段會進一步討論資料扭曲。

使用 Spark UI 調查效能問題

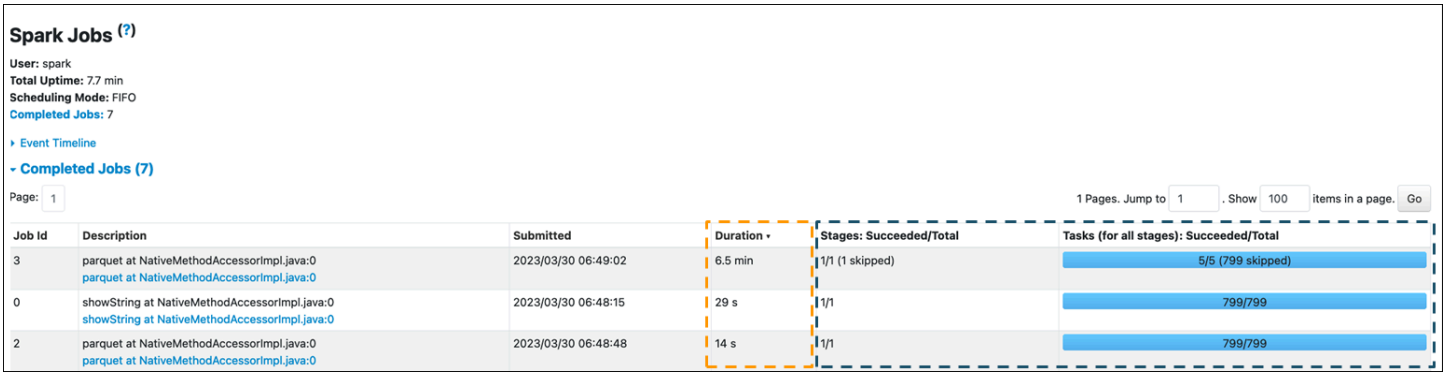
在您套用任何最佳實務來調整 AWS Glue 任務的效能之前，強烈建議您分析效能並找出瓶頸。這將協助您專注於正確的事物。

為了快速分析，Amazon CloudWatch 指標提供任務指標的基本檢視。Spark UI 提供更深入的效能調校檢視。若要搭配 使用 Spark UI AWS Glue，您必須為 AWS Glue 任務啟用 Spark UI。熟悉 Spark UI 之後，請遵循調校 Spark 任務效能的策略，根據您的問題清單識別並減少瓶頸的影響。

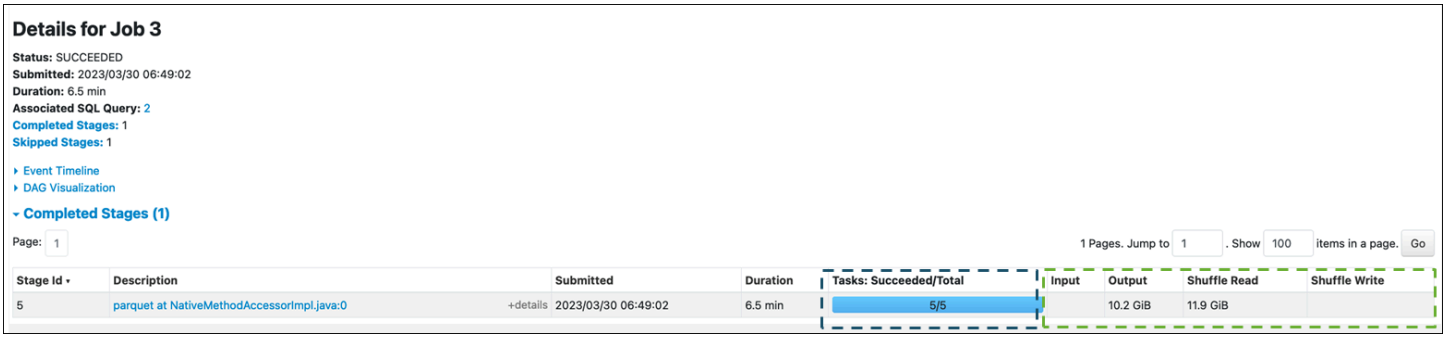
使用 Spark UI 識別瓶頸

當您開啟 Spark UI 時，Spark 應用程式會列在資料表中。根據預設， AWS Glue 任務的應用程式名稱為 `nativespark-<Job Name>-<Job Run ID>`。根據任務執行 ID 選擇目標 Spark 應用程式，以開啟任務索引標籤。不完整的任務執行，例如串流任務執行，會列在顯示不完整的應用程式中。

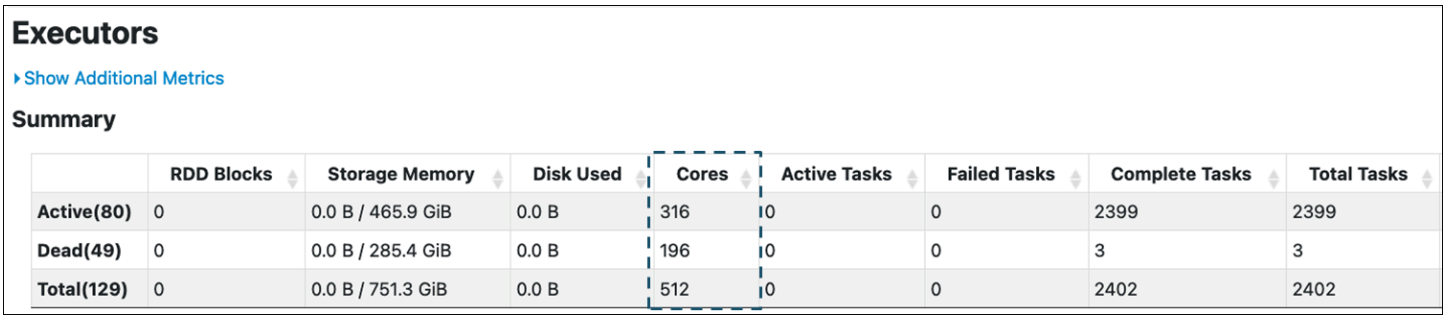
任務索引標籤會顯示 Spark 應用程式中所有任務的摘要。若要判斷任何階段或任務失敗，請檢查任務總數。若要尋找瓶頸，請選擇持續時間來排序。選擇描述欄中顯示的連結，深入了解長時間執行任務的詳細資訊。



任務的詳細資訊頁面會列出階段。在此頁面上，您可以看到整體洞見，例如持續時間、成功任務數和總任務數、輸入和輸出數，以及隨機讀取和隨機寫入量。



執行器索引標籤會詳細說明 Spark 叢集容量。您可以檢查核心的總數。下列螢幕擷取畫面中顯示的叢集總共包含 316 個作用中核心和 512 個核心。根據預設，每個核心可以同時處理一個 Spark 任務。



Executors

▸ Show Additional Metrics

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks
Active(80)	0	0.0 B / 465.9 GiB	0.0 B	316	0	0	2399	2399
Dead(49)	0	0.0 B / 285.4 GiB	0.0 B	196	0	0	3	3
Total(129)	0	0.0 B / 751.3 GiB	0.0 B	512	0	0	2402	2402

根據工作詳細資訊頁面上5/5顯示的值，階段 5 是最長的階段，但只使用 512 個核心中的 5 個核心。由於此階段的平行處理非常低，但需要大量時間，因此您可以將其識別為瓶頸。為了改善效能，您想要了解原因。若要進一步了解如何辨識並減少常見效能瓶頸的影響，請參閱調校 Spark 任務效能的策略。

調校 Spark 任務效能的策略

準備調校參數時，請使用以下最佳實務：

- 在開始識別問題之前，先決定您的效能目標。
- 嘗試變更調校參數之前，先使用指標來識別問題。

為在調校任務時得到最一致的結果，應制定調校工作的基線策略。

效能調校的基線策略

一般而言，效能調校依照以下工作流程進行：

1. 決定效能目標。
2. 量測指標。
3. 識別瓶頸。
4. 降低瓶頸的影響。
5. 重複步驟 2-4，直到達到預期的目標。

首先，確定您的效能目標。例如，您的其中一個目標是在 AWS Glue 3 小時內完成任務的執行。定義目標之後，請測量任務效能指標。識別指標和瓶頸的趨勢，以符合目標。特別是，識別瓶頸對於故障診斷、偵錯和效能調校最為重要。在執行 Spark 應用程式期間，Spark 會在 Spark 事件日誌中記錄每個任務的狀態和統計資料。

在 中 AWS Glue，您可以透過 Spark 歷史記錄伺服器提供的 Spark Web UI 檢視 Spark 指標。 AWS Glue 針對 Spark 任務，可以將 Spark 事件日誌傳送至您在 Amazon S3 中指定的位置。 AWS Glue 也提供範例AWS CloudFormation 範本和 Dockerfile，以在 Amazon EC2 執行個體或本機電腦上啟動 Spark 歷史記錄伺服器，因此您可以使用 Spark UI 搭配事件日誌。

在您確定績效目標並識別指標來評估這些目標之後，您可以開始使用以下章節中的策略來識別和修復瓶頸。

Spark 任務效能的調校實務

您可以使用下列策略來調校 Spark AWS Glue 任務的效能：

- AWS Glue 資源：
  - 擴展叢集容量
  - 使用 AWS Glue 最新版本
- Spark 應用程式：
  - 減少資料掃描量
  - 平行化任務
  - 最佳化隨機播放
  - 將規劃開銷降至最低
  - 最佳化使用者定義的函數

使用這些策略之前，您必須能夠存取 Spark 任務的指標和組態。您可以在 AWS Glue 文件中找到此資訊。

從 AWS Glue 資源的角度來看，您可以透過新增 AWS Glue 工作者並使用 AWS Glue 最新版本來實現效能改善。

從 Apache Spark 應用程式的觀點來看，您可以存取數個可改善效能的策略。如果將不必要的資料載入 Spark 叢集，您可以將其移除，以減少載入的資料量。如果您已使用不足的 Spark 叢集資源且資料 I/O 很低，您可以識別要平行化的任務。如果需要大量時間，您可能也想要最佳化繁重的資料傳輸操作，例如聯結。您也可以最佳化任務查詢計劃，或降低個別 Spark 任務的運算複雜性。

若要有效率地套用這些策略，您必須諮詢指標來識別何時適用這些策略。如需詳細資訊，請參閱下列各節。這些技術不僅適用於效能調校，也適用於解決out-of-memory(OOM) 錯誤等典型問題。

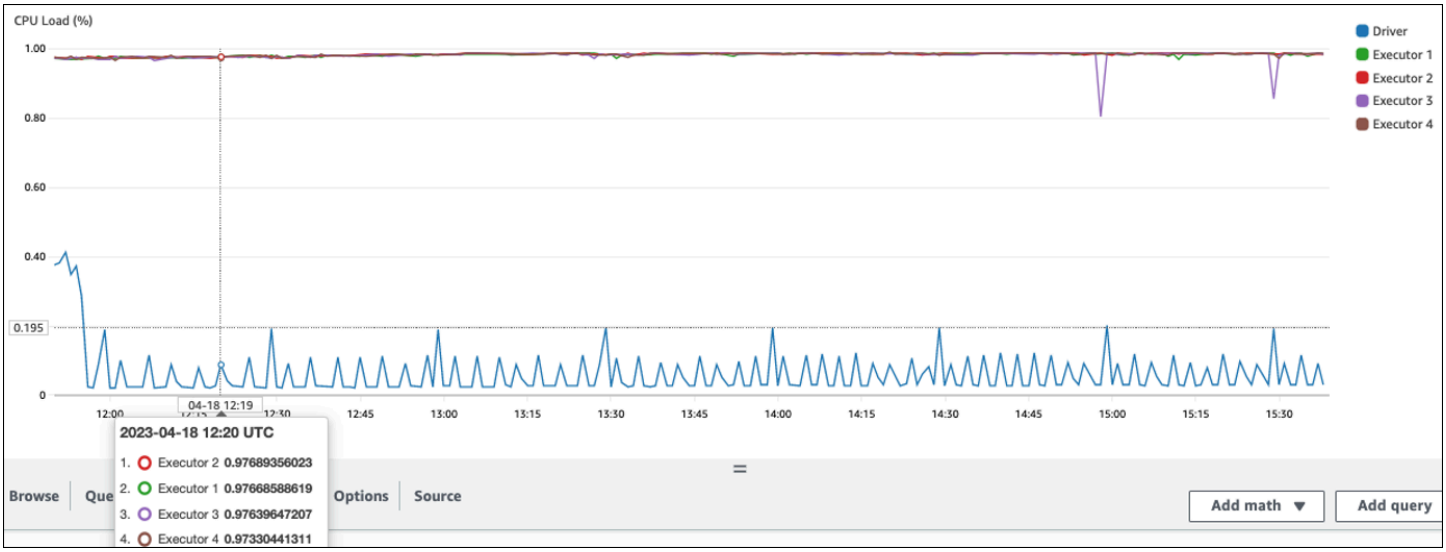
擴展叢集容量

如果您的任務花費太多時間，但執行器耗用足夠的資源，且 Spark 正在建立相對於可用核心的大量任務，請考慮擴展叢集容量。若要評估這是否適當，請使用下列指標。

CloudWatch 指標

- 檢查 CPU 負載和記憶體使用率，以判斷執行器是否耗用足夠的資源。
- 檢查任務執行的時間長度，以評估處理時間是否太長而無法滿足您的效能目標。

在下列範例中，四個執行器以超過 97% 的 CPU 負載執行，但在大約三個小時後仍未完成處理。

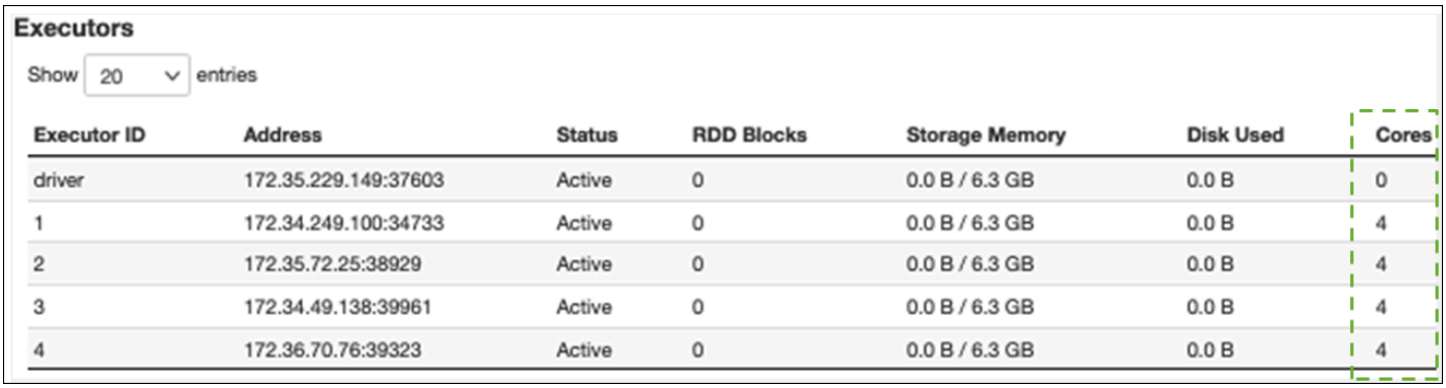


> **Note**
>
> 如果 CPU 負載低，您可能不會受益於擴展叢集容量。

Spark 使用者介面

在任務索引標籤或階段索引標籤上，您可以查看每個任務或階段的任務數量。在下列範例中，Spark 已建立58100任務。

在執行器索引標籤上，您可以看到執行器和任務的總數。在下列螢幕擷取畫面中，每個 Spark 執行器都有四個核心，可以同時執行四個任務。



在此範例中，Spark 任務的數量 (58100) 遠大於執行器可同時處理的 16 個任務 (4 個執行器 × 4 個核心）。

如果您觀察到這些症狀，請考慮擴展叢集。您可以使用下列選項來擴展叢集容量：

- Enable AWS Glue Auto Scaling – Auto Scaling 適用於 3.0 AWS Glue 版或更新版本中的 AWS Glue 擷取、轉換和載入 (ETL) 和串流任務。 會根據每個階段的分割區數量或任務執行時產生微批次的速率 AWS Glue ，自動從叢集新增和移除工作者。

  如果您發現即使 Auto Scaling 已啟用，工作者數量也不會增加的情況，請考慮手動新增工作者。不過，請注意，手動擴展一個階段可能會導致許多工作者在稍後階段處於閒置狀態，因此成本更高，因此效能不會增加。

  啟用 Auto Scaling 之後，您可以在 CloudWatch 執行器指標中查看執行器的數量。使用下列指標來監控 Spark 應用程式中執行器的需求：

  - `glue.driver.ExecutorAllocationManager.executors.numberAllExecutors`
  - `glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors`

  如需指標的詳細資訊，請參閱AWS Glue 使用 Amazon CloudWatch 指標進行監控。
- 橫向擴展：增加工作者數量 AWS Glue – 您可以手動增加工作者數量 AWS Glue 。只有在您觀察到閒置的工作者時，才新增工作者。此時，新增更多工作者將增加成本，而不會改善結果。如需詳細資訊，請參閱平行處理任務。
- 擴展：使用較大的工作者類型 – 您可以手動變更 AWS Glue 工作者的執行個體類型，以使用具有更多核心、記憶體和儲存體的工作者。較大的工作者類型可讓您垂直擴展和執行密集資料整合任務，例如記憶體密集型資料轉換、偏斜彙總，以及涉及 PB 資料的實體偵測檢查。

  向上擴展也有助於 Spark 驅動程式需要更大容量的情況，例如，因為任務查詢計畫相當大。如需工作者類型和效能的詳細資訊，請參閱 AWS 大數據部落格文章使用新的大型工作者類型 G.4X 和 G.8X 擴展 AWS Glue Apache Spark 任務。

  使用較大的工作者也可以減少所需的工作者總數，透過減少加入等密集操作中的隨機播放來提高效能。

使用 AWS Glue 最新版本

建議使用最新版本 AWS Glue 。每個版本內建數種最佳化和升級，可能會自動改善任務效能。例如，AWS Glue 4.0 提供下列新功能：

- 全新最佳化的 Apache Spark 3.3.0 執行期 – AWS Glue 4.0 以 Apache Spark 3.3.0 執行期為基礎，為開放原始碼 Spark 帶來相當的效能改善。Spark 3.3.0 執行期以 Spark 2.x 的許多創新為基礎。
- 增強型 Amazon Redshift 連接器 – AWS Glue 4.0 和更新版本提供 Apache Spark 的 Amazon Redshift 整合。整合以現有的開放原始碼連接器為基礎，並增強其效能和安全性。整合可協助應用程式以高達 10 倍的速度執行。如需詳細資訊，請參閱有關 Amazon Redshift 與 Apache Spark 整合的部落格文章。
- 使用 CSV 和 JSON 資料 – 3.0 版及更新版本進行向量化讀取的 SIMD 型執行新增了最佳化讀取器，相較於資料列型讀取器，可大幅加快整體任務效能。 AWS Glue 如需 CSV 資料的詳細資訊，請參閱使用向量化 SIMD CSV 讀取器最佳化讀取效能。如需 JSON 資料的詳細資訊，請參閱搭配 Apache Arrow 單欄式格式使用向量化 SIMD JSON 讀取器。

每個 AWS Glue 版本都會包含此類型的升級，包括連接器、驅動程式和程式庫更新。如需詳細資訊，請參閱 AWS Glue 版本和將 AWS Glue 任務遷移至 AWS Glue 4.0 版。

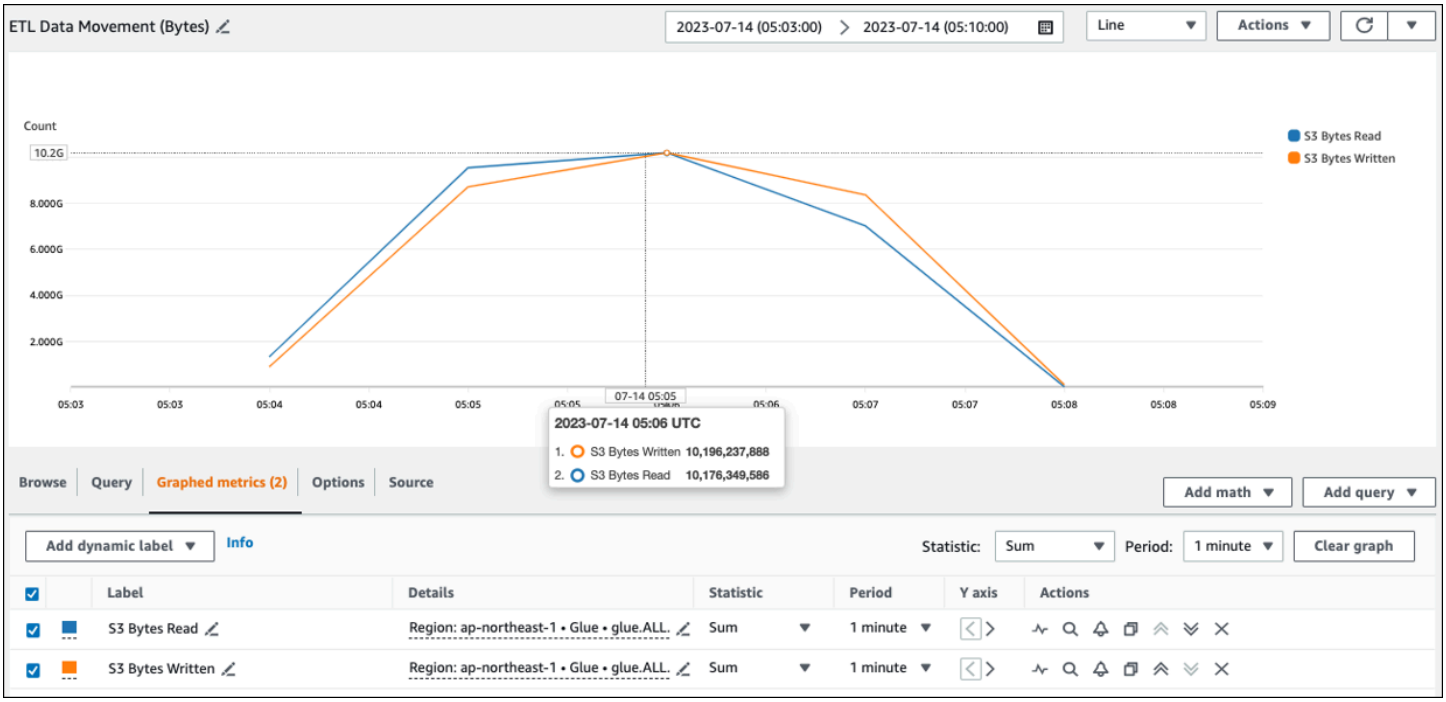
減少資料掃描量

若要開始，請考慮僅載入您需要的資料。您只需減少每個資料來源載入 Spark 叢集的資料量，即可改善效能。若要評估此方法是否適當，請使用下列指標。

您可以在 CloudWatch 指標中檢查來自 Amazon S3 的讀取位元組，以及在 Spark UI 中檢查更多詳細資訊，如 Spark UI 一節所述。

CloudWatch 指標

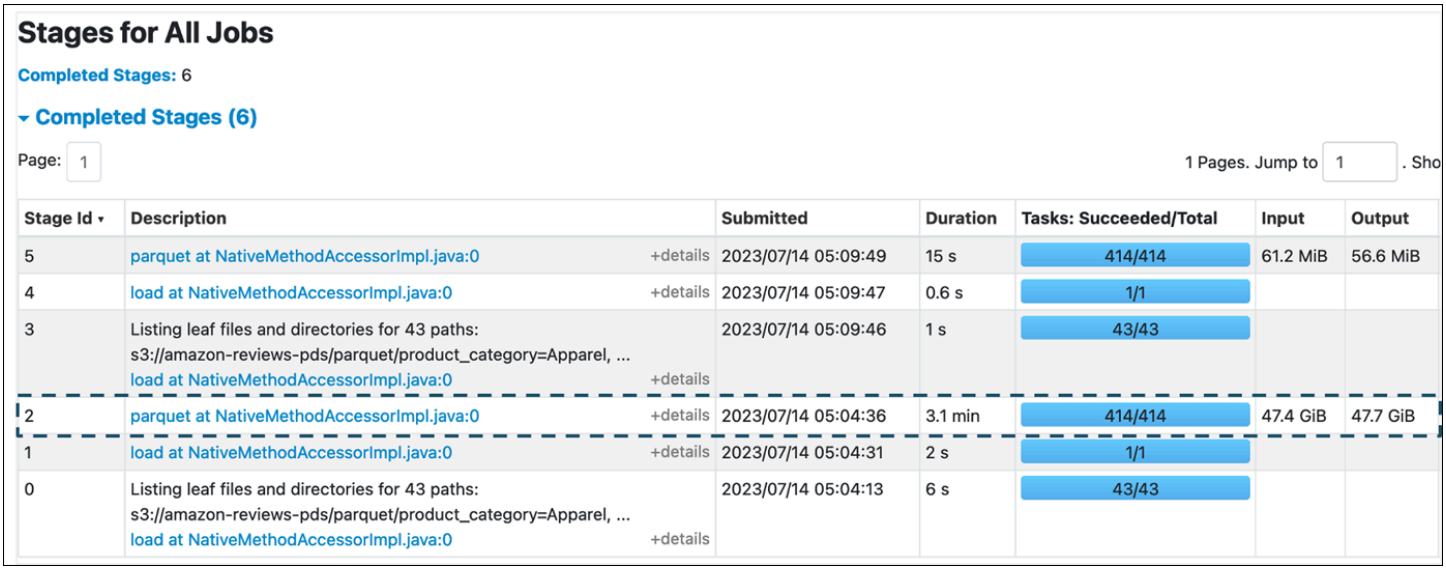
您可以在 ETL 資料移動（位元組）中查看 Amazon S3 的大約讀取大小。此指標顯示自上次報告以來，所有執行器從 Amazon S3 讀取的位元組數。您可以使用它來監控來自 Amazon S3 的 ETL 資料移動，而且您可以將讀取與來自外部資料來源的擷取速率進行比較。



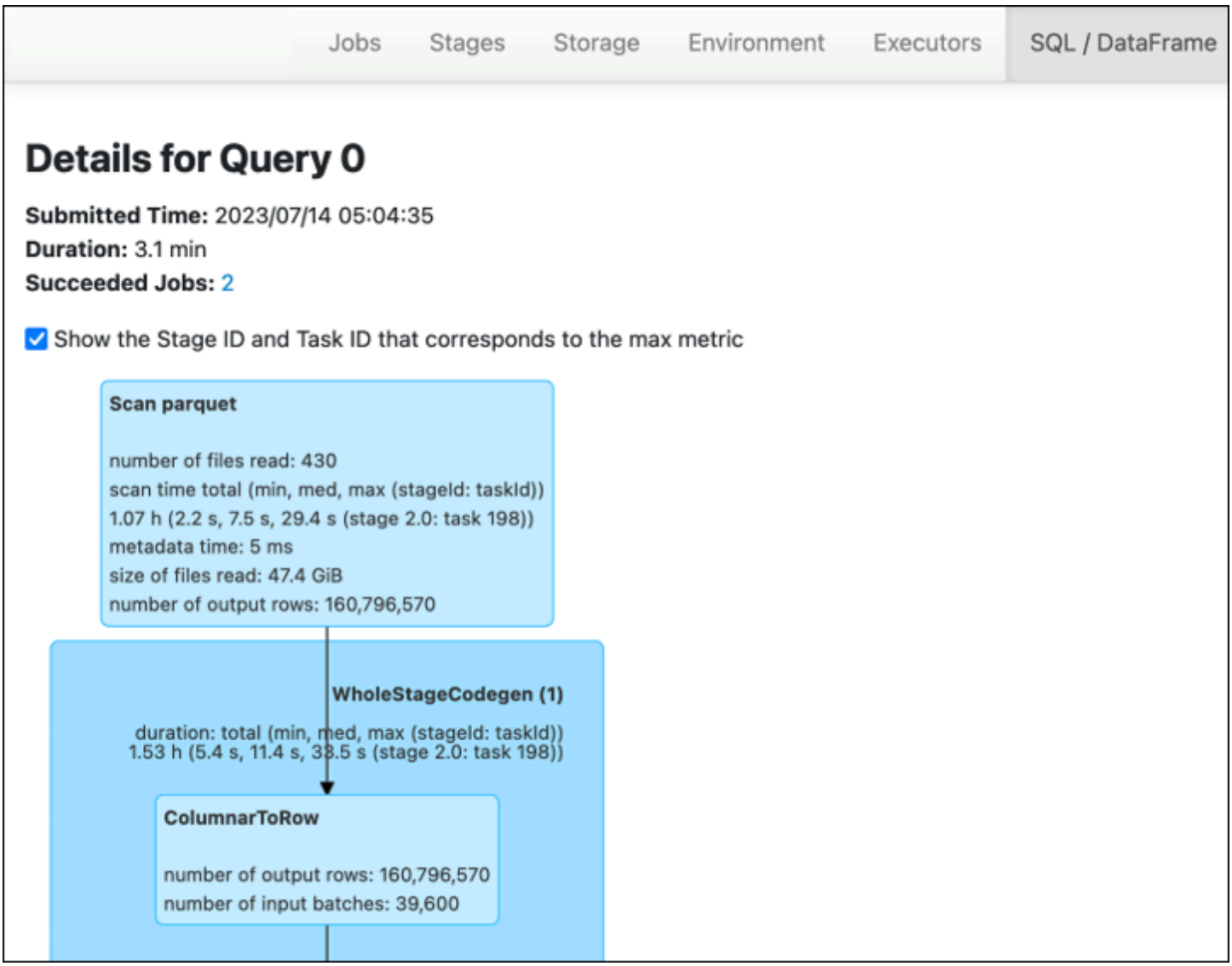
如果您觀察到比預期更大的 S3 位元組讀取資料點，請考慮下列解決方案。

Spark 使用者介面

在 AWS Glue 適用於 Spark UI 的階段索引標籤上，您可以看到輸入和輸出大小。在下列範例中，階段 2 讀取 47.4 GiB 輸入和 47.7 GiB 輸出，而階段 5 讀取 61.2 MiB 輸入和 56.6 MiB 輸出。



當您在 AWS Glue 任務中使用 Spark SQL 或 DataFrame 方法時，SQL / DataFrame 索引標籤會顯示這些階段的更多統計資料。在此情況下，階段 2 會顯示檔案讀取數量：430，檔案讀取大小：47.4 GiB，輸出資料列數量：160，796，570。



如果您發現您正在讀取的資料與您正在使用的資料之間有很大的差異，請嘗試下列解決方案。

Amazon S3

若要減少從 Amazon S3 讀取時載入任務的資料量，請考慮資料集 的檔案大小、壓縮、檔案格式和檔案配置 （分割區）。 AWS Glue 對於 Spark 任務通常用於原始資料的 ETL，但為了有效率的分散式處理，您需要檢查資料來源格式的功能。

- 檔案大小 – 我們建議將輸入和輸出的檔案大小保持在中等範圍內 （例如 128 MB)。太小的檔案和太大的檔案都可能導致問題。

大量小型檔案會導致下列問題：

- Amazon S3 上的大量網路 I/O 負載，因為對許多物件（相較於存放相同資料數量的幾個物件Head) 提出請求（例如 ListGet、 或）所需的額外負荷。
- Spark 驅動程式上的大量 I/O 和處理負載，這會產生許多分割區和任務，並導致過度平行處理。

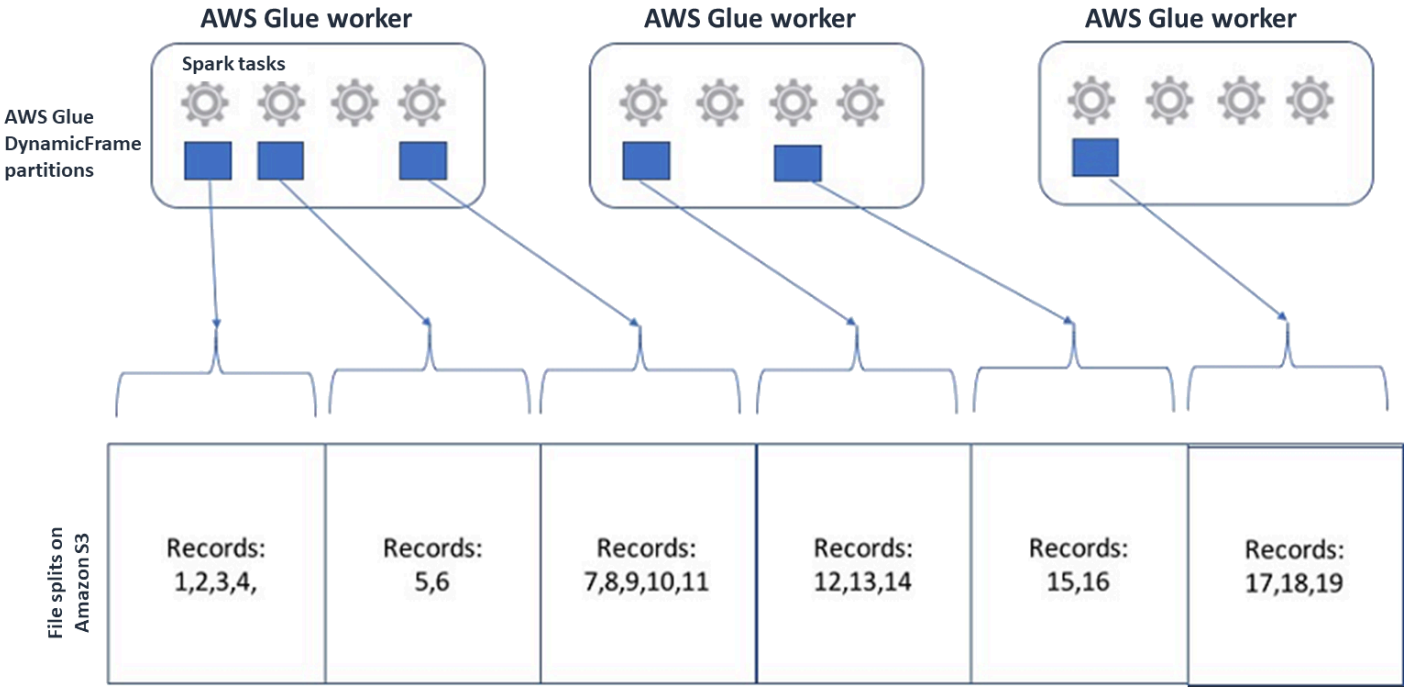
另一方面，如果您的檔案類型無法分割（例如 gzip) 且檔案太大，Spark 應用程式必須等到單一任務完成讀取整個檔案。

若要減少為每個小型檔案建立 Apache Spark 任務時產生的過度平行處理，請使用 DynamicFrames 的檔案分組。此方法可降低 Spark 驅動程式發生 OOM 例外狀況的機率。若要設定檔案分組，請設定 groupFiles和 groupSize 參數。下列程式碼範例使用 ETL 指令碼中的 AWS Glue DynamicFrame API 搭配這些參數。

```
dyf = glueContext.create_dynamic_frame_from_options("s3",
    {'paths': ["s3://input-s3-path/"],
    'recurse':True,
    'groupFiles': 'inPartition',
    'groupSize': '1048576'},
    format="json")
```

- 壓縮 – 如果您的 S3 物件位於數百 MB 中，請考慮壓縮它們。有多種壓縮格式，可廣泛分類為兩種類型：
  - gzip 等無法分割 的壓縮格式需要一個工作者解壓縮整個檔案。
  - 可分割壓縮格式，例如 bzip2 或 LZO（索引），允許檔案的部分解壓縮，可平行化。

  對於 Spark（和其他常見的分散式處理引擎），您可以將來源資料檔案分割為引擎可以平行處理的區塊。這些單位通常稱為分割。資料為可分割格式之後，最佳化的 AWS Glue 讀取器可以透過提供 GetObject API Range選項，僅擷取特定區塊，從 S3 物件擷取分割。請考慮下圖，了解這在實務上如何運作。



只要檔案為最佳大小或檔案可分割，壓縮的資料就能大幅加速您的應用程式。較小的資料大小可減少從 Amazon S3 掃描的資料，以及從 Amazon S3 到 Spark 叢集的網路流量。另一方面，壓縮和解壓縮資料需要更多 CPU。所需的運算量會隨著壓縮演算法的壓縮比率而擴展。選擇可分割壓縮格式時，請考慮此取捨。

> **Note**
>
> 雖然 gzip 檔案通常無法分割，但您可以使用 gzip 壓縮個別 parquet 區塊，而且這些區塊可以平行化。

- 檔案格式 – 使用單欄式格式。Apache Parquet 和 Apache ORC 是熱門的單欄式資料格式。Parquet 和 ORC 會採用資料欄型壓縮、根據其資料類型編碼和壓縮每個資料欄，以有效率地存放資料。如需 Parquet 編碼的詳細資訊，請參閱 Parquet 編碼定義。Parquet 檔案也可以分割。

  直欄格式依資料欄分組值，並將其存放在區塊中。使用單欄式格式時，您可以略過對應於您不打算使用之資料欄的資料區塊。Spark 應用程式只能擷取您需要的資料欄。一般而言，更好的壓縮率或略過資料區塊意味著從 Amazon S3 讀取較少的位元組，進而獲得更好的效能。這兩種格式也支援下列下推方法來減少 I/O：

  - 投影下推 – 投影下推是一種僅擷取應用程式中指定資料欄的技術。您可以在 Spark 應用程式中指定資料欄，如下列範例所示：
    - DataFrame 範例： `df.select("star_rating")`

  - Spark SQL 範例： `spark.sql("select start_rating from <table>")`
- 述詞下推 – 述詞下推是一種有效處理 `WHERE` 和 `GROUP BY`子句的技術。這兩種格式都有代表資料欄值的資料區塊。每個區塊都會保留區塊的統計資料，例如最大值和最小值。Spark 可以使用這些統計資料，根據應用程式中使用的篩選條件值來判斷是否應讀取或略過區塊。若要使用此功能，請在條件中新增更多篩選條件，如下列範例所示：
  - DataFrame 範例： `df.select("star_rating").filter("star_rating < 2")`
  - Spark SQL 範例： `spark.sql("select * from <table> where star_rating < 2")`
- 檔案配置 – 根據資料的使用方式，將 S3 資料儲存到不同路徑中的物件，您可以有效率地擷取相關資料。如需詳細資訊，請參閱《Amazon S3 文件》中的使用字首整理物件。 AWS Glue 支援以 格式將金鑰和值儲存到 Amazon S3 字首`key=value`，並依 Amazon S3 路徑分割您的資料。透過分割資料，您可以限制每個下游分析應用程式掃描的資料量，從而改善效能並降低成本。如需詳細資訊，請參閱管理 中 ETL 輸出的分割區 AWS Glue。

  分割會將資料表分成不同的部分，並根據年、月、日等資料欄值，將相關資料保留在分組檔案中，如下列範例所示。

```
# Partitioning by /YYYY/MM/DD
s3://<YourBucket>/year=2023/month=03/day=31/0000.gz
s3://<YourBucket>/year=2023/month=03/day=01/0000.gz
s3://<YourBucket>/year=2023/month=03/day=02/0000.gz
s3://<YourBucket>/year=2023/month=03/day=03/0000.gz
...
```

  您可以使用 中的資料表建立模型，以定義資料集的分割區 AWS Glue Data Catalog。然後，您可以使用分割區剔除來限制資料掃描的數量，如下所示：

  - For AWS Glue DynamicFrame，設定 `push_down_predicate` (或 `catalogPartitionPredicate`)。

```
dyf = Glue_context.create_dynamic_frame.from_catalog(
    database=src_database_name,
    table_name=src_table_name,
    push_down_predicate = "year='2023' and month ='03'",
)
```

  - 針對 Spark DataFrame，設定固定路徑以刪除分割區。

```
df = spark.read.format("json").load("s3://<YourBucket>/year=2023/month=03/*/*.gz")
```

- 對於 Spark SQL，您可以設定 where 子句從 Data Catalog 刪除分割區。

```
df = spark.sql("SELECT * FROM <Table> WHERE year= '2023' and month = '03'")
```

- 若要在使用 寫入資料時依日期分割 AWS Glue，請在 DataFrame 的 DynamicFrame 或 partitionBy() 中設定 partitionKeys，資料欄中的日期資訊如下所示。
  - DynamicFrame

```
glue_context.write_dynamic_frame_from_options(
    frame= dyf, connection_type='s3',format='parquet'
    connection_options= {
        'partitionKeys': ["year", "month", "day"],
        'path': 's3://<YourBucket>/<Prefix>/'
    }
)
```

  - DataFrame

```
df.write.mode('append')\
    .partitionBy('year','month','day')\
    .parquet('s3://<YourBucket>/<Prefix>/')
```

這可以改善輸出資料消費者的效能。

如果您無法存取 來變更建立輸入資料集的管道，則分割不是一個選項。反之，您可以使用 glob 模式來排除不需要的 S3 路徑。在 DynamicFrame 中讀取時設定排除。例如，下列程式碼排除 2023 年的第 01 到 09 個月天數。

```
dyf = glueContext.create_dynamic_frame.from_catalog(
    database=db,
    table_name=table,
    additional_options = { "exclusions":"[\"**year=2023/month=0[1-9]/**\"]" },
    transformation_ctx='dyf'
)
```

您也可以在 Data Catalog 的資料表屬性中設定排除：

- 索引鍵：`exclusions`

  - 值：["**year=2023/month=0[1-9]/**"]
- 太多 Amazon S3 分割區 – 避免在包含各種值的資料欄上分割 Amazon S3 資料，例如具有數千個值的 ID 資料欄。這可以大幅增加儲存貯體中的分割區數量，因為可能的分割區數量是您已分割的所有欄位的乘積。太多分割區可能會導致下列情況：
  - 從 Data Catalog 擷取分割區中繼資料的延遲增加
  - 增加小型檔案的數量，這需要更多 Amazon S3 API 請求 (List、 Get和 Head)

  例如，當您在 partitionBy或 中設定日期類型時partitionKeys， 等日期層級分割yyyy/mm/dd適用於許多使用案例。不過， yyyy/mm/dd/<ID>可能會產生太多分割區，進而對整體效能產生負面影響。

  另一方面，一些使用案例，例如即時處理應用程式，需要許多分割區，例如 yyyy/mm/dd/hh。如果您的使用案例需要大量分割區，請考慮使用AWS Glue 分割區索引來減少從 Data Catalog 擷取分割區中繼資料的延遲。

資料庫和 JDBC

若要在從資料庫擷取資訊時減少資料掃描，您可以在 SQL 查詢中指定where述詞 （ 或 子句 ）。不提供 SQL 界面的資料庫將提供自己的查詢或篩選機制。

使用 Java Database Connectivity (JDBC) 連線時，請使用 where子句為下列參數提供選取查詢：

- 對於 DynamicFrame，請使用 sampleQuery 選項。使用時create_dynamic_frame.from_catalog，請設定引additional_options數，如下所示。

```
query = "SELECT * FROM <TableName> where id = 'XX' AND"
datasource0 = glueContext.create_dynamic_frame.from_catalog(
  database = db,
  table_name = table,
  additional_options={
      "sampleQuery": query,
      "hashexpression": key,
      "hashpartitions": 10,
      "enablePartitioningForSampleQuery": True
    },
  transformation_ctx = "datasource0"
)
```

當 時using create_dynamic_frame.from_options，設定引connection_options數，如下所示。

```
query = "SELECT * FROM <TableName> where id = 'XX' AND"
datasource0 = glueContext.create_dynamic_frame.from_options(
    connection_type = connection,
    connection_options={
        "url": url,
        "user": user,
        "password": password,
        "dbtable": table,
        "sampleQuery": query,
        "hashexpression": key,
        "hashpartitions": 10,
        "enablePartitioningForSampleQuery": True
    }
)
```

- 對於 DataFrame，請使用 查詢選項。

```
query = "SELECT * FROM <TableName> where id = 'XX'"
jdbcDF = spark.read \
.format('jdbc') \
.option('url', url) \
.option('user', user) \
.option('password', pwd) \
.option('query', query) \
.load()
```

- 對於 Amazon Redshift，請使用 AWS Glue 4.0 或更新版本來利用 Amazon Redshift Spark 連接器中的下推支援。

```
dyf = glueContext.create_dynamic_frame.from_catalog(
    database = "redshift-dc-database-name",
    table_name = "redshift-table-name",
    redshift_tmp_dir = args["temp-s3-dir"],
    additional_options = {"aws_iam_role": "arn:aws:iam::role-account-id:role/rs-role-name"}
)
```

- 如需其他資料庫，請參閱該資料庫的文件。

AWS Glue 選項

- 若要避免對所有連續任務執行進行完整掃描，並僅處理上次任務執行期間不存在的資料，請啟用任務書籤。
- 若要限制要處理的輸入資料數量，請使用任務書籤啟用週框執行。這有助於減少每個任務執行的掃描資料量。

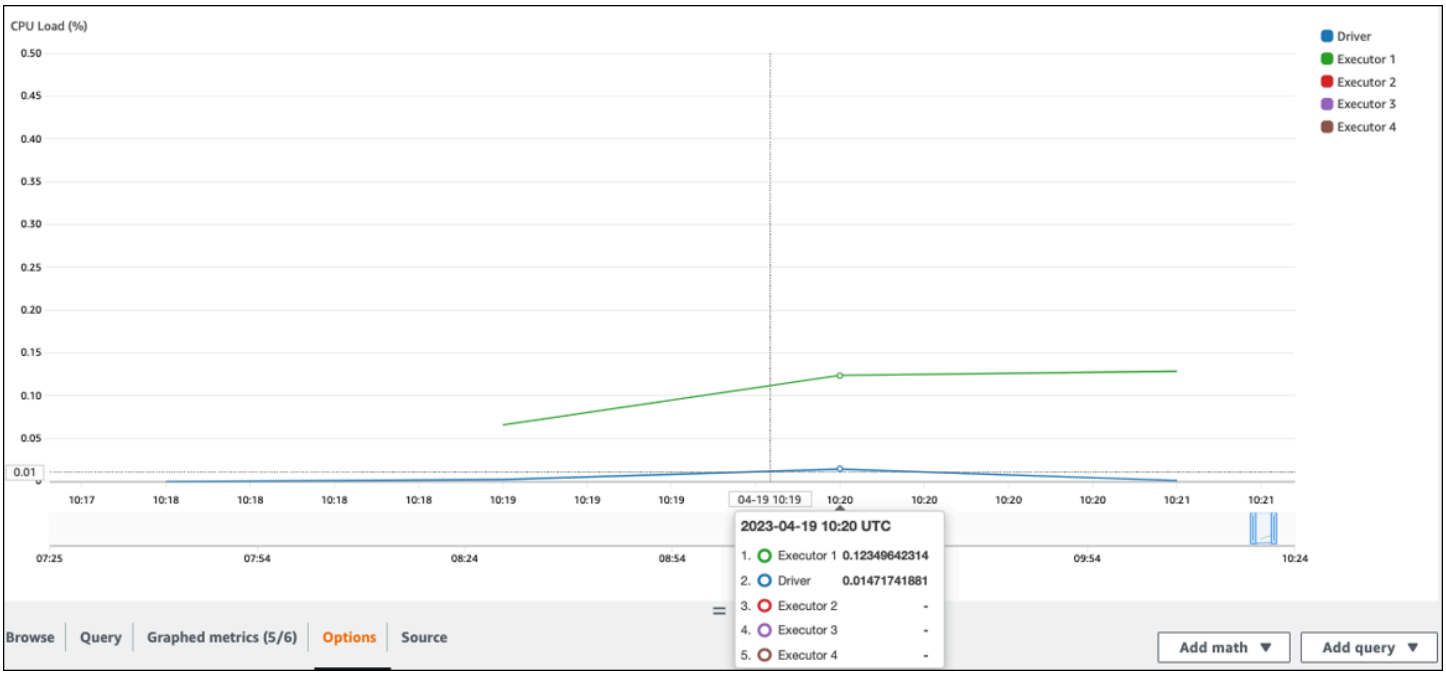
平行化任務

若要最佳化效能，請務必平行處理資料載入和轉換的任務。正如我們在 Apache Spark 中討論的關鍵主題所述，彈性分散式資料集 (RDD) 分割區的數量很重要，因為它決定平行處理的程度。Spark 建立的每個任務都會以 1：1 為基礎對應至 RDD 分割區。若要獲得最佳效能，您需要了解如何決定 RDD 分割區的數量，以及如何最佳化該數量。

如果您沒有足夠的平行處理，以下症狀將記錄在 CloudWatch 指標和 Spark UI 中。

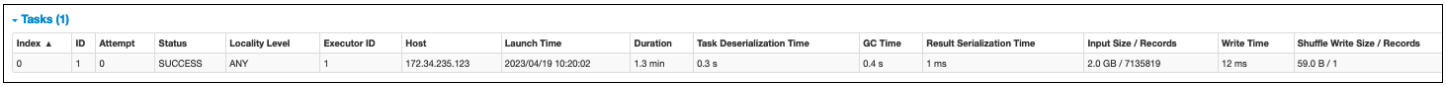
CloudWatch 指標

檢查 CPU 負載和記憶體使用率。如果某些執行器未在任務的某個階段處理，則適合改善平行處理。在此情況下，在視覺化時間範圍內，執行器 1 正在執行任務，但剩餘的執行器 (2、3 和 4) 並未執行。您可以推斷 Spark 驅動程式未指派這些執行器的任務。



Spark 使用者介面

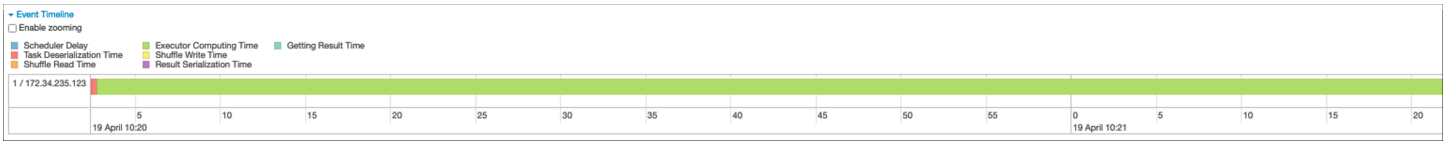
在 Spark UI 的階段索引標籤上，您可以查看階段中的任務 數量。 在此情況下，Spark 只執行一個任務。



- Tasks (1)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	Task Deserialization Time	GC Time	Result Serialization Time	Input Size / Records	Write Time	Shuffle Write Size / Records
0	1	0	SUCCESS	ANY	1	172.34.235.123	2023/04/19 10:20:02	1.3 min	0.3 s	0.4 s	1 ms	2.0 GB / 7135819	12 ms	59.0 B / 1

此外，事件時間軸會顯示執行器 1 處理一個任務。這表示此階段中的工作完全在某個執行器上執行，而其他執行器則處於閒置狀態。



如果您觀察到這些症狀，請嘗試每個資料來源的下列解決方案。

從 Amazon S3 平行載入資料

若要平行處理來自 Amazon S3 的資料載入，請先檢查預設的分割區數量。然後，您可以手動判斷分割區的目標數量，但請務必避免擁有太多分割區。

決定預設的分割區數量

對於 Amazon S3，初始數量的 Spark RDD 分割區 （每個分割區對應至 Spark 任務） 取決於 Amazon S3 資料集的功能 （例如，格式、壓縮和大小）。當您從存放在 Amazon S3 中的 CSV 物件建立 an AWS Glue DynamicFrame 或 Spark DataFrame 時，初始 RDD 分割區數量 (NumPartitions) 可大致計算如下：

- 物件大小 <= 64 MB： `NumPartitions = Number of Objects`
- 物件大小 > 64 MB： `NumPartitions = Total Object Size / 64 MB`
- 不可分割 (gzip)： `NumPartitions = Number of Objects`

如減少資料掃描量一節所述，Spark 會將大型 S3 物件分成可平行處理的分割。當物件大於分割大小時，Spark 會分割物件，並為每個分割建立 RDD 分割區 （和任務）。Spark 的分割大小取決於您的資料格式和執行時間環境，但這是合理的開始近似值。有些物件是使用 gzip 等不可分割的壓縮格式進行壓縮，因此 Spark 無法分割它們。

`NumPartitions` 值可能會因資料格式、壓縮、 AWS Glue 版本、 AWS Glue 工作者數量和 Spark 組態而有所不同。

例如，當您使用 Spark DataFrame 載入單一 10 GB `csv.gz` 物件時，Spark 驅動程式只會建立一個 RDD 分割區 (`NumPartitions=1`)，因為 gzip 是不可分割的。這會導致一個特定 Spark 執行器的繁重負載，並且不會將任何任務指派給剩餘的執行器，如下圖所述。

在 Spark Web UI 階段索引標籤上檢查階段的實際任務數量 (`NumPartitions`)，或在程式碼`df.rdd.getNumPartitions()`中執行 以檢查平行處理。

遇到 10 GB gzip 檔案時，請檢查產生該檔案的系統是否可以以可分割格式產生該檔案。如果這不是選項，您可能需要擴展叢集容量來處理檔案。若要有效對您載入的資料執行轉換，您需要使用重新分割來重新平衡叢集中整個工作者的 RDD。

手動判斷分割區的目標數量

視資料屬性和 Spark 實作特定功能而定，即使基礎工作仍可平行處理，最終仍可能具有較低的`NumPartitions`值。如果 `NumPartitions` 太小，請執行 `df.repartition(N)` 以增加分割區數量，以便將處理分散到多個 Spark 執行器。

在這種情況下，執行 `NumPartitions` `df.repartition(100)`會從 1 增加到 100，建立 100 個資料分割區，每個分割區都有可指派給其他執行器的任務。

操作會將整個資料平均`repartition(N)`分割 (10 GB/100 個分割區 = 100 MB/分割區），避免資料扭曲至特定分割區。

> **Note**
>
> `join` 執行 等隨機播放操作時，分割區的數量會根據 `spark.sql.shuffle.partitions`或的值動態增加或減少`spark.default.parallelism`。這有助於 Spark 執行器之間更有效率地交換資料。如需詳細資訊，請參閱 Spark 文件。

您在決定分割區的目標數量時，目標是最大化佈建 AWS Glue 工作者的使用量。 AWS Glue 工作者數量和 Spark 任務數量與 vCPUs 數量相關。Spark 支援每個 vCPU 核心一個任務。在 3.0 AWS Glue 版或更新版本中，您可以使用下列公式計算分割區的目標數量。

```
# Calculate NumPartitions by WorkerType
numExecutors = (NumberOfWorkers - 1)
numSlotsPerExecutor =
  4 if WorkerType is G.1X
  8 if WorkerType is G.2X
```

```
  16 if WorkerType is G.4X
  32 if WorkerType is G.8X
NumPartitions = numSlotsPerExecutor * numExecutors

# Example: Glue 4.0 / G.1X / 10 Workers
numExecutors = ( 10 - 1 ) = 9  # 1 Worker reserved on Spark Driver
numSlotsPerExecutor       = 4  # G.1X has 4 vCpu core ( Glue 3.0 or later )
NumPartitions = 9  * 4    = 36
```

在此範例中，每個 G.1X 工作者都會提供四個 vCPU 核心給 Spark 執行器 (`spark.executor.cores = 4`)。Spark 為每個 vCPU Core 支援一個任務，因此 G.1X Spark 執行器可以同時執行四個任務 (`numSlotPerExecutor`)。如果任務需要相同的時間量，則此分割區數量會充分利用叢集。不過，某些任務需要比其他任務更長的時間，進而建立閒置核心。如果發生這種情況，請考慮`numPartitions`將 2 或 3 相乘，以分解並有效率地排程瓶頸任務。

太多分割區

過多的分割區會建立過多的任務。這會導致 Spark 驅動程式負載過重，因為與分散式處理相關的額外負荷，例如管理任務和 Spark 執行器之間的資料交換。

如果您任務中的分割區數量遠大於您的分割區目標數量，請考慮減少分割區數量。您可以使用下列選項來減少分割區：

- 如果您的檔案大小很小，請使用 AWS Glue groupFiles。您可以減少啟動 Apache Spark 任務以處理每個檔案所產生的過度平行處理。
- 使用 `coalesce(N)`將分割區合併在一起。這是一個低成本的程序。減少分割區數量時，`coalesce(N)` 優先於 `repartition(N)`，因為 會`repartition(N)`執行隨機播放來平均分配每個分割區中的記錄數量。這會增加成本和管理開銷。
- 使用 Spark 3.x 自適應查詢執行。如 Apache Spark 中關鍵主題一節所述，自適應查詢執行提供 函數，以自動合併分割區的數量。當您在執行執行之前不知道分割區數量時，可以使用此方法。

從 JDBC 平行載入資料

Spark RDD 分割區的數量取決於組態。請注意，根據預設，只會執行單一任務來透過`SELECT`查詢掃描整個來源資料集。

AWS Glue DynamicFrames 和 Spark DataFrames 都支援跨多個任務的平行 JDBC 資料載入。這可透過使用`where`述詞將一個`SELECT`查詢分割為多個查詢來完成。若要平行化 JDBC 的讀取，請設定下列選項：

- For AWS Glue DynamicFrame，設定 `hashfield` (或 `hashexpression`)和 `hashpartition`。若要進一步了解，請參閱平行讀取 JDBC 資料表。

```
connection_mysql8_options = {
    "url": "jdbc:mysql://XXXXXXXXXX.XXXXXXX.us-east-1.rds.amazonaws.com:3306/test",
    "dbtable": "medicare_tb",
    "user": "test",
    "password": "XXXXXXXXX",
    "hashexpression":"id",
    "hashpartitions":"10"
}
datasource0 = glueContext.create_dynamic_frame.from_options(
    'mysql',
    connection_options=connection_mysql8_options,
    transformation_ctx= "datasource0"
)
```

- 針對 Spark DataFrame，設定 `numPartitions`、`lowerBound`、 `partitionColumn`和 `upperBound`。若要進一步了解，請參閱 JDBC 至其他資料庫。

```
df = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:mysql://XXXXXXXXXX.XXXXXXX.us-east-1.rds.amazonaws.com:3306/
test") \
    .option("dbtable", "medicare_tb") \
    .option("user", "test") \
    .option("password", "XXXXXXXXXX") \
    .option("partitionColumn", "id") \
    .option("numPartitions", "10") \
    .option("lowerBound", "0") \
    .option("upperBound", "1141455") \
    .load()

df.write.format("json").save("s3://bucket_name/Tests/sparkjdbc/with_parallel/")
```

使用 ETL 連接器時，從 DynamoDB 平行載入資料

Spark RDD 分割區的數量由 `dynamodb.splits` 參數決定。若要平行處理來自 Amazon DynamoDB 的讀取，請設定下列選項：

- 增加 的值`dynamodb.splits`。

- 遵循 for Spark 中 ETL 的連線類型和選項中說明 AWS Glue的公式來最佳化 參數。

從 Kinesis Data Streams 平行載入資料

Spark RDD 分割區的數量取決於來源 Amazon Kinesis Data Streams 資料串流中的碎片數量。如果您的資料串流中只有幾個碎片，則只會有幾個 Spark 任務。這可能會導致下游程序的平行處理率低。若要平行化 Kinesis Data Streams 的讀取，請設定下列選項：

- 從 Kinesis Data Streams 載入資料時，增加碎片數量以取得更多平行處理。
- 如果您在微型批次中的邏輯夠複雜，請考慮在捨棄不需要的資料欄之後，在批次開始時重新分割資料。

如需詳細資訊，請參閱最佳化 AWS Glue 串流 ETL 任務成本和效能的最佳實務。

資料載入後平行處理任務

若要在資料載入後平行處理任務，請使用下列選項來增加 RDD 分割區的數量：

- 重新分割資料以產生更多分割區，特別是如果負載本身無法平行化，則在初始載入之後。

  在 DynamicFrame 或 DataFrame `repartition()`上呼叫 ，指定分割區的數量。良好的經驗法則是可用核心數目的兩倍或三倍。

  不過，寫入分割的資料表時，這可能會導致檔案爆炸 （每個分割區可能會在每個資料表分割區中產生檔案）。若要避免這種情況，您可以依資料欄重新分割 DataFrame。這會使用資料表分割區資料欄，以便在寫入之前整理資料。您可以指定更多分割區，而不會在資料表分割區上取得小型檔案。不過，請小心避免資料扭曲，其中某些分割區值最終會伴隨大部分的資料，並延遲任務的完成。
- 有隨機播放時，請增加 `spark.sql.shuffle.partitions`值。這也有助於解決隨機播放時的任何記憶體問題。

  當您有超過 2，001 個隨機分割區時，Spark 會使用壓縮的記憶體格式。如果您的數字接近此值，您可能想要設定超過該限制`spark.sql.shuffle.partitions`的值，以取得更有效率的表示法。

最佳化隨機播放

`join()` 和 等特定操作`groupByKey()`需要 Spark 執行隨機播放。隨機播放是 Spark 重新分配資料的機制，因此會在 RDD 分割區之間以不同的方式分組。隨機播放有助於修復效能瓶頸。不過，由於隨機

播放通常涉及在 Spark 執行器之間複製資料，因此隨機播放是複雜且昂貴的操作。例如，隨機播放會產生下列成本：

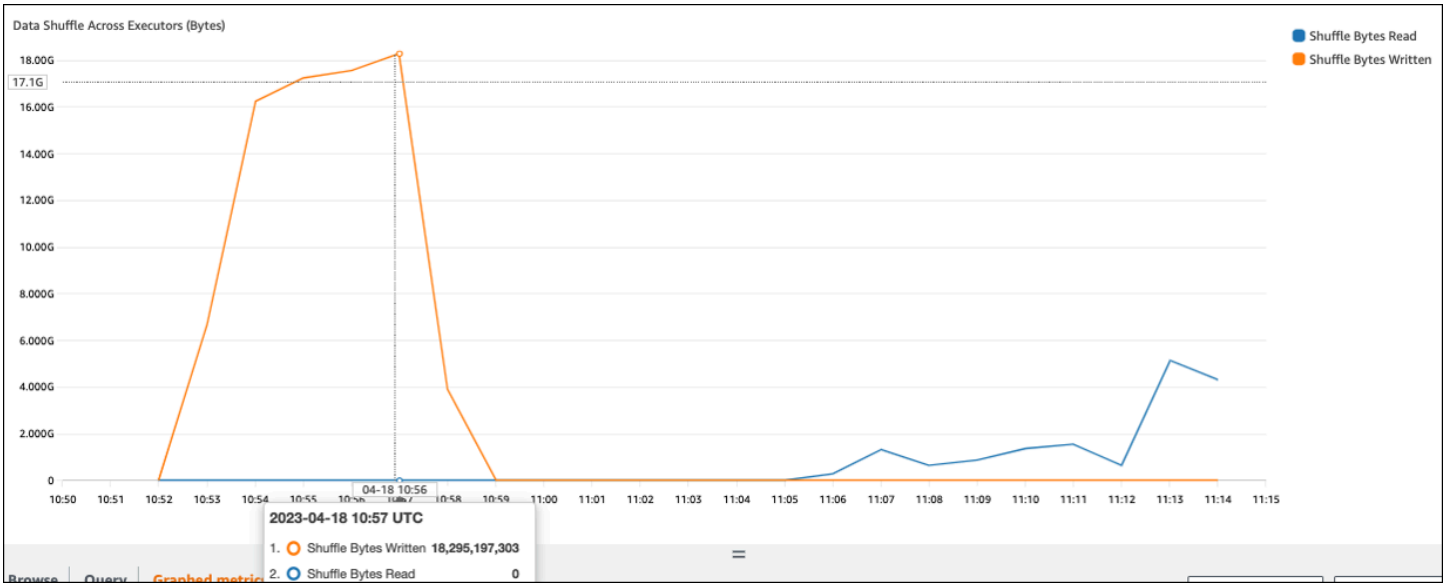
- 磁碟輸入/輸出：
  - 在磁碟上產生大量中繼檔案。
- 網路 I/O：
  - 需要許多網路連線（連線數目 = `Mapper × Reducer`)。
  - 由於記錄會彙總到可能託管在不同 Spark 執行器上的新 RDD 分割區，因此您資料集的很大一部分可能會在 Spark 執行器之間透過網路移動。
- CPU 和記憶體負載：
  - 排序值並合併資料集。這些操作是在執行器上規劃的，對執行器施加繁重負載。

隨機播放是 Spark 應用程式效能降低的最重要因素之一。儲存中繼資料時，可能會耗盡執行器本機磁碟上的空間，這會導致 Spark 任務失敗。

您可以在 CloudWatch 指標和 Spark UI 中評估隨機播放效能。

CloudWatch 指標

如果隨機播放位元組寫入值高於隨機播放位元組讀取，則 Spark 任務可能會使用隨機播放操作，例如 `join()`或 `groupByKey()`。



Spark 使用者介面

在 Spark UI 的階段索引標籤上，您可以檢查隨機讀取大小/記錄值。您也可以在執行器索引標籤上查看。

在下列螢幕擷取畫面中，每個執行器會與隨機播放程序交換約 18.6GB/4020000 的記錄，隨機播放讀取大小總計約為 75 GB)。

隨機溢出 （磁碟） 欄顯示大量資料溢出記憶體到磁碟，這可能會導致磁碟完整或效能問題。

Aggregated Metrics by Executor

Executor ID ▲	Address	Shuffle Read Size / Records	Shuffle Spill (Memory)	Shuffle Spill (Disk)
1	172.35.205.23:46731	18.6 GB / 40210300	98.1 GB	16.8 GB
2	172.35.195.173:46185	18.7 GB / 40246767	117.2 GB	17.3 GB
3	172.36.135.106:35913	18.6 GB / 40253921	101.6 GB	16.6 GB
4	172.34.131.223:46879	18.6 GB / 40190741	99.5 GB	16.4 GB

如果您觀察到這些症狀，而且與效能目標相比，階段需要太長的時間，或者失敗並出現 `Out Of Memory`或 `No space left on device`錯誤，請考慮下列解決方案。

最佳化聯結

加入資料表`join()`的操作是最常用的隨機播放操作，但通常是效能瓶頸。由於加入是一項昂貴的操作，我們建議您不要使用它，除非它對您的業務需求至關重要。詢問下列問題，再次確認您有效使用資料管道：

- 您是否正在重新計算也可以在您可以重複使用的其他任務中執行的聯結？
- 您是否加入 ，將外部索引鍵解析為輸出消費者未使用的值？

在您確認加入操作對於您的業務需求至關重要之後，請參閱下列選項，以符合您需求的方式最佳化您的加入。

在加入之前使用下推

在執行聯結之前，篩選掉 DataFrame 中不必要的資料列和資料欄。這具有下列優點：

- 減少隨機播放期間的資料傳輸量
- 減少 Spark 執行器中的處理量
- 減少資料掃描量

```
# Default
df_joined  = df1.join(df2, ["product_id"])

# Use Pushdown
df1_select =
 df1.select("product_id","product_title","star_rating").filter(col("star_rating")>=4.0)
df2_select = df2.select("product_id","category_id")
df_joined  = df1_select.join(df2_select, ["product_id"])
```

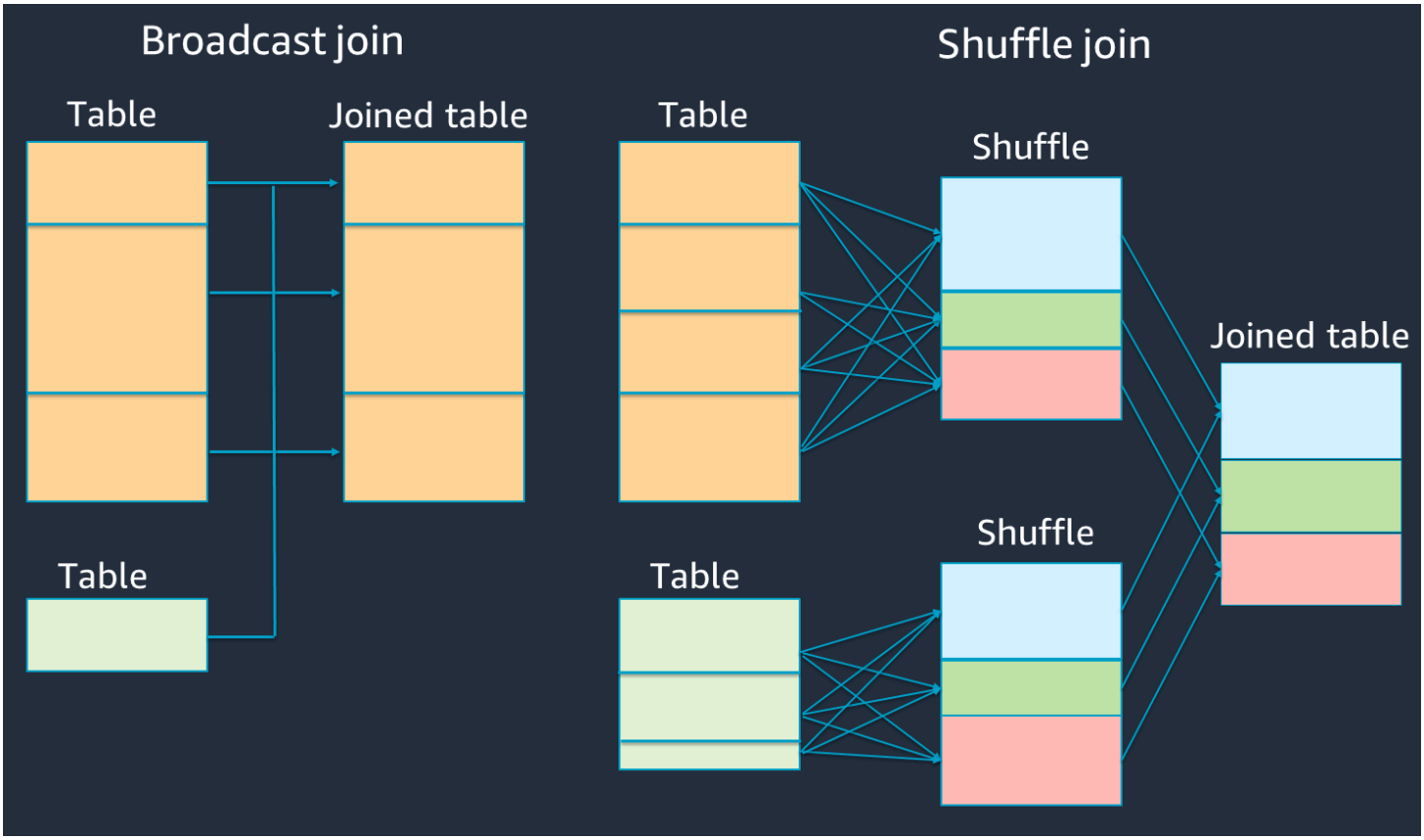
使用 DataFrame 聯結

嘗試使用 SparkSQL、DataFrame 和資料集等 Spark 高階 API，而不是 RDD API 或 DynamicFrame 聯結。您可以使用 等方法呼叫，將 DynamicFrame 轉換為 DataFrame`dyf.toDF()`。如 Apache Spark 中的關鍵主題章節所述，這些聯結操作會在內部利用 Catalyst 最佳化工具的查詢最佳化。

隨機播放和廣播雜湊聯結和提示

Spark 支援兩種類型的聯結：隨機聯結和廣播雜湊聯結。廣播雜湊聯結不需要隨機播放，而且可能需要比隨機播放聯結更少的處理。不過，它僅適用於將小型資料表加入大型資料表時。加入可以容納單一 Spark 執行器記憶體的資料表時，請考慮使用廣播雜湊聯結。

下圖顯示廣播雜湊聯結和隨機聯結的高階結構和步驟。



每個聯結的詳細資訊如下：

- 隨機聯結：
  - 隨機雜湊聯結會聯結兩個資料表，而不會排序和分配兩個資料表之間的聯結。它適用於可存放在 Spark 執行器記憶體中的小型資料表聯結。
  - 排序合併聯結會分配要依索引鍵聯結的兩個資料表，並在聯結之前對其進行排序。它適用於大型資料表的聯結。
- 廣播雜湊聯結：
  - 廣播雜湊聯結會將較小的 RDD 或資料表推送至每個工作者節點。然後，它會與較大 RDD 或資料表的每個分割區進行對應端組合。

    當您的其中一個 RDDs 或資料表可以容納在記憶體中或可以容納在記憶體中時，它適用於聯結。盡可能進行廣播雜湊聯結是有益的，因為它不需要隨機播放。您可以使用聯結提示從 Spark 請求廣播聯結，如下所示。

```
# DataFrame
from pySpark.sql.functions import broadcast
```

```
df_joined= df_big.join(broadcast(df_small), right_df[key] == left_df[key],
 how='inner')

-- SparkSQL
SELECT /*+ BROADCAST(t1) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;
```

如需聯結提示的詳細資訊，請參閱聯結提示。

在 AWS Glue 3.0 和更新版本中，您可以透過啟用自適應查詢執行和其他參數，自動利用廣播雜湊聯結。當任一聯結端的執行時間統計資料小於自適應廣播雜湊聯結閾值時，自適應查詢執行會將排序合併聯結轉換為廣播雜湊聯結。

在 AWS Glue 3.0 中，您可以透過設定 來啟用自適應查詢執行。`spark.sql.adaptive.enabled=true`根據預設，AWS Glue 4.0 中會啟用自適應查詢執行。

您可以設定與隨機播放和廣播雜湊聯結相關的其他參數：

- `spark.sql.adaptive.localShuffleReader.enabled`
- `spark.sql.adaptive.autoBroadcastJoinThreshold`

如需相關參數的詳細資訊，請參閱將排序合併聯結轉換為廣播聯結。

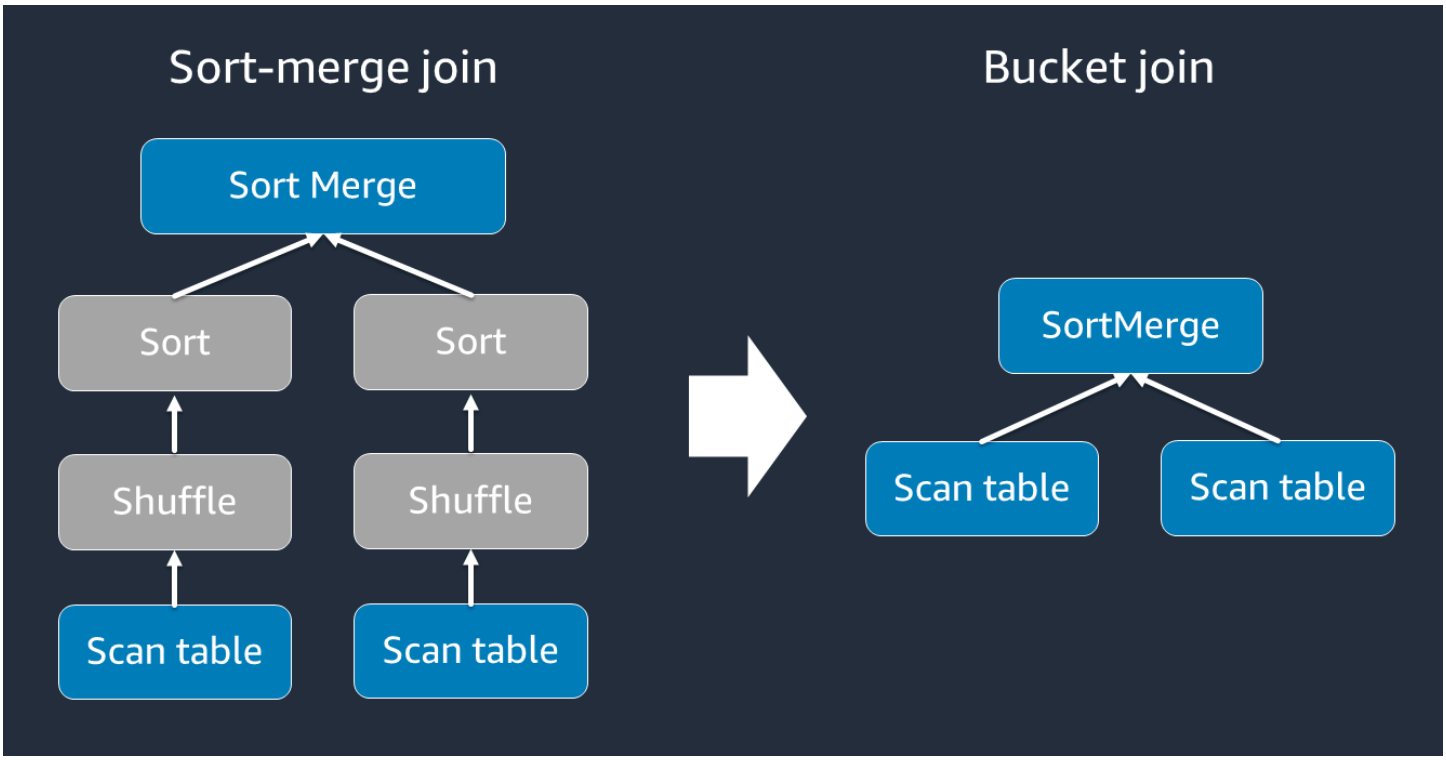
在 AWS Glue 3.0 和 或更新版本中，您可以使用其他加入提示進行隨機播放來調整您的行為。

```
-- Join Hints for shuffle sort merge join
SELECT /*+ SHUFFLE_MERGE(t1) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;
SELECT /*+ MERGEJOIN(t2) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;
SELECT /*+ MERGE(t1) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;

-- Join Hints for shuffle hash join
SELECT /*+ SHUFFLE_HASH(t1) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;

-- Join Hints for shuffle-and-replicate nested loop join
SELECT /*+ SHUFFLE_REPLICATE_NL(t1) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;
```

使用儲存貯體

排序合併聯結需要兩個階段：隨機排序和排序，然後合併。這兩個階段可能會使 Spark 執行器超載，並在某些執行器合併時導致 OOM 和效能問題，而其他執行器同時排序。在這種情況下，可以使用儲存

貯體有效率地加入。儲存貯體會在聯結金鑰上預先隨機排序和預先排序您的輸入，然後將該排序資料寫入中介資料表。預先定義排序的中介資料表，可以降低加入大型資料表時的隨機排序和排序步驟成本。



儲存貯體資料表適用於下列項目：

- 經常透過相同金鑰加入的資料，例如 `account_id`
- 載入每日累積資料表，例如可在一般資料欄上儲存的基底和差異資料表

您可以使用下列程式碼建立儲存貯體資料表。

```
df.write.bucketBy(50, "account_id").sortBy("age").saveAsTable("bucketed_table")
```

在聯結前重新分割聯結金鑰上的 DataFrames

若要在聯結前重新分割聯結金鑰上的兩個 DataFrames，請使用下列陳述式。

```
df1_repartitioned = df1.repartition(N,"join_key")
df2_repartitioned = df2.repartition(N,"join_key")
df_joined  = df1_repartitioned.join(df2_repartitioned,"product_id")
```

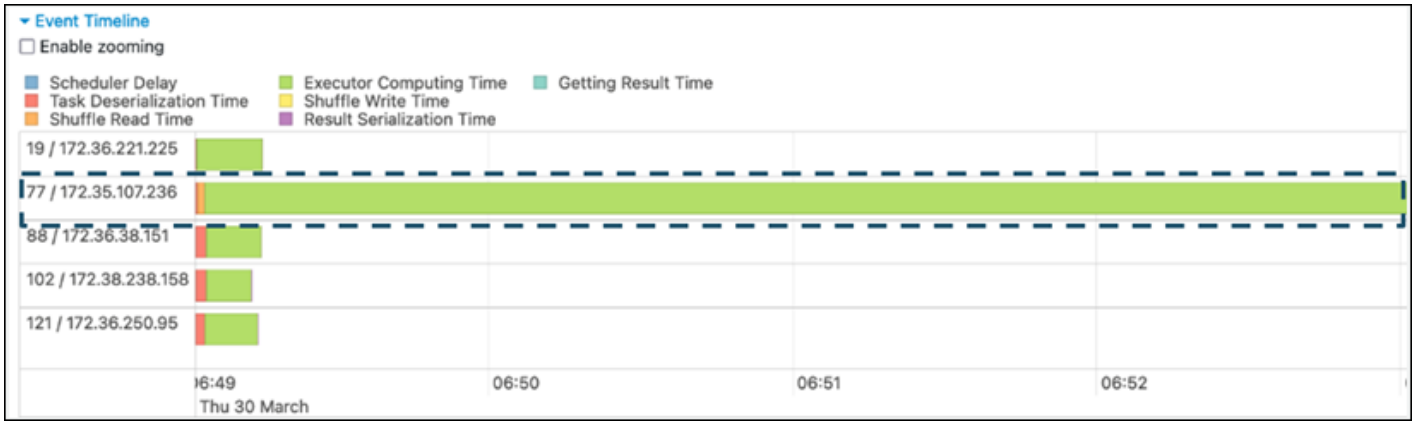
這會先在聯結金鑰上分割兩個（仍然分開）RDDs，再啟動聯結。如果兩個 RDDs具有相同分割程式碼的相同金鑰上分割，則您計劃聯結的 RDD 記錄，在隨機切換聯結之前，有很高的可能性會共置在相同的工作者上。這可以透過減少聯結期間的網路活動和資料扭曲來改善效能。

克服資料扭曲

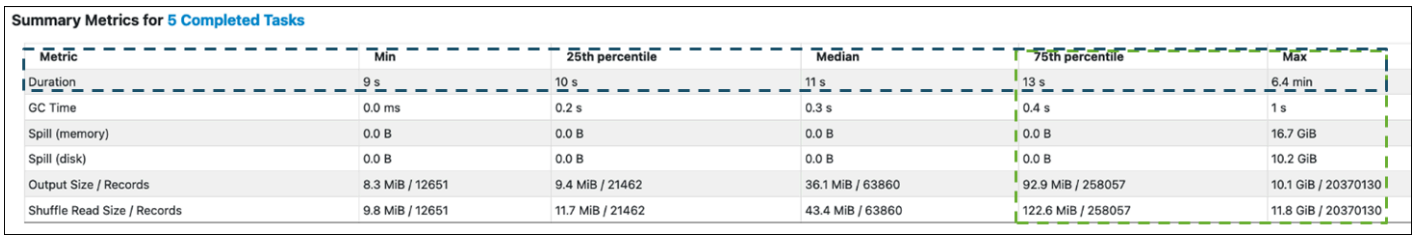
資料扭曲是 Spark 任務瓶頸的最常見原因之一。當資料未平均分散到 RDD 分割區時，就會發生這種情況。這會導致該分割區的任務花費比其他分割區長得多，因而延遲應用程式的整體處理時間。

若要識別資料扭曲，請在 Spark UI 中評估下列指標：

- 在 Spark UI 的階段索引標籤上，檢查事件時間軸頁面。您可以在下列螢幕擷取畫面中看到任務分佈不均勻。分佈不均勻或執行時間過長的任務可能表示資料扭曲。



- 另一個重要的頁面是摘要指標，顯示 Spark 任務的統計資料。下列螢幕擷取畫面顯示持續時間、GC 時間、溢出（記憶體）、溢出（磁碟）等具有百分位數的指標。



Metric	Min	25th percentile	Median	75th percentile	Max
Duration	9 s	10 s	11 s	13 s	6.4 min
GC Time	0.0 ms	0.2 s	0.3 s	0.4 s	1 s
Spill (memory)	0.0 B	0.0 B	0.0 B	0.0 B	16.7 GiB
Spill (disk)	0.0 B	0.0 B	0.0 B	0.0 B	10.2 GiB
Output Size / Records	8.3 MiB / 12651	9.4 MiB / 21462	36.1 MiB / 63860	92.9 MiB / 258057	10.1 GiB / 20370130
Shuffle Read Size / Records	9.8 MiB / 12651	11.7 MiB / 21462	43.4 MiB / 63860	122.6 MiB / 258057	11.8 GiB / 20370130

當任務平均分散時，您會在所有百分位數中看到類似的數字。當資料扭曲時，您會在每個百分位數中看到非常偏差的值。在此範例中，任務持續時間在最小值、第 25 個百分位數、中位數和第 75 個百分位數中少於 13 秒。雖然 Max 任務處理的資料是第 75 個百分位數的 100 倍，但其 6.4 分鐘的持續時間約為 30 倍。這表示至少有一個任務（或最多 25% 的任務）花費比其餘任務更長的時間。

如果您看到資料扭曲，請嘗試下列動作：

- 如果您使用 AWS Glue 3.0，請設定 來啟用自適應查詢執行spark.sql.adaptive.enabled=true。預設會在 AWS Glue 4.0 中啟用自適應查詢執行。

  您也可以透過設定下列相關參數，對聯結引入的資料扭曲使用自適應查詢執行：

  - spark.sql.adaptive.skewJoin.skewedPartitionFactor
  - spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes
  - spark.sql.adaptive.advisoryPartitionSizeInBytes=128m (128 mebibytes or larger should be good)
  - spark.sql.adaptive.coalescePartitions.enabled=true (when you want to coalesce partitions)

  如需詳細資訊，請參閱 Apache Spark 文件。

- 對聯結索引鍵使用具有大量值的索引鍵。在隨機聯結中，會針對金鑰的每個雜湊值決定分割區。如果聯結索引鍵的基數太低，雜湊函數更可能執行在分割區之間分佈資料的錯誤工作。因此，如果您的應用程式和商業邏輯支援它，請考慮使用較高的基數索引鍵或複合索引鍵。

```
# Use Single Primary Key
df_joined = df1_select.join(df2_select, ["primary_key"])

# Use Composite Key
df_joined = df1_select.join(df2_select, ["primary_key","secondary_key"])
```

使用快取

當您使用重複DataFrames時，請使用 或 df.persist()將計算結果快取到每個 Spark 執行器的記憶體和磁碟上，以避免額外的隨機播放df.cache()或運算。Spark 也支援在磁碟上保留 RDDs 或跨多個節點複寫 (儲存層級)。

例如，您可以透過新增 來保留 DataFramesdf.persist()。當不再需要快取時，您可以使用 unpersist 來捨棄快取的資料。

```
df = spark.read.parquet("s3://<Bucket>/parquet/product_category=Books/")
df_high_rate = df.filter(col("star_rating")>=4.0)
df_high_rate.persist()

df_joined1 = df_high_rate.join(<Table1>, ["key"])
df_joined2 = df_high_rate.join(<Table2>, ["key"])
```

```
df_joined3 = df_high_rate.join(<Table3>, ["key"])
...
df_high_rate.unpersist()
```

移除不需要的 Spark 動作

避免執行不必要的動作，例如 `count`、 `show` 或 `collect`。如 Apache Spark 中的關鍵主題一節中所述，Spark 是延遲的。每次執行動作時，可能會重新計算每個轉換的 RDD。當您使用許多 Spark 動作時，會呼叫每個動作的多個來源存取、任務計算和隨機執行。

如果您在商業環境中不需要 `collect()` 或其他動作，請考慮將其移除。

> **Note**
>
> 盡可能避免`collect()`在商業環境中使用 Spark。`collect()` 動作會將 Spark 執行器中計算的所有結果傳回 Spark 驅動程式，這可能會導致 Spark 驅動程式傳回 OOM 錯誤。為了避免 OOM 錯誤，Spark `spark.driver.maxResultSize = 1GB` 預設會設定 ，將傳回 Spark 驅動程式的資料大小上限限制為 1 GB。

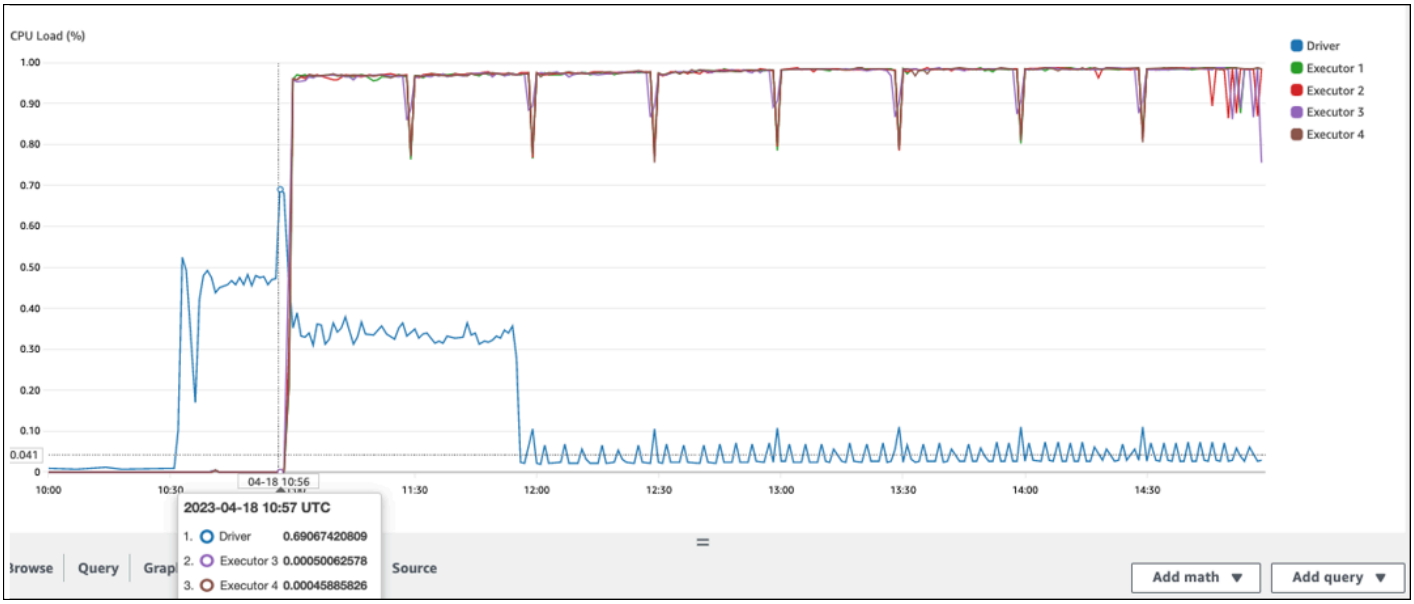
將規劃開銷降至最低

如 Apache Spark 中討論的關鍵主題所述，Spark 驅動程式會產生執行計畫。根據該計畫，任務會指派給 Spark 執行器以進行分散式處理。不過，如果有大量小型檔案或 包含大量分割區， AWS Glue Data Catalog Spark 驅動程式可能會成為瓶頸。若要識別高規劃開銷，請評估下列指標。

CloudWatch 指標

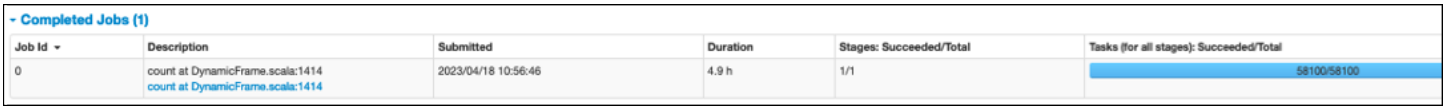
檢查 CPU 負載和記憶體使用率是否有下列情況：

- Spark 驅動程式 CPU 負載和記憶體使用率會記錄為高。一般而言，Spark 驅動程式不會處理您的資料，因此 CPU 負載和記憶體使用率不會遽增。不過，如果 Amazon S3 資料來源有太多小型檔案，列出所有 S3 物件和管理大量任務可能會導致資源使用率過高。
- 在 Spark 執行器中開始處理之前有很長的間隙。在下列範例螢幕擷取畫面中，Spark 執行器的 CPU 負載太低，直到 10：57，即使 AWS Glue 任務是從 10：00 開始。這表示 Spark 驅動程式可能需要很長的時間才能產生執行計畫。在此範例中，擷取 Data Catalog 中的大量分割區，並列出 Spark 驅動程式中的大量小型檔案需要很長時間。



Spark 使用者介面

在 Spark UI 的任務索引標籤上，您可以看到提交的時間。在下列範例中，Spark 驅動程式在 10：56：46 啟動 job0，即使 AWS Glue 任務在 10：00：00 啟動。



您也可以在任務索引標籤上查看任務（所有階段）：成功/總時間。 在此情況下，任務數量會記錄為 58100。如平行處理任務頁面的 Amazon S3 一節所述，任務數量大約對應至 S3 物件數量。這表示 Amazon S3 中大約有 58，100 個物件。

如需此任務和時間表的詳細資訊，請檢閱階段索引標籤。如果您發現 Spark 驅動程式存在瓶頸，請考慮下列解決方案：

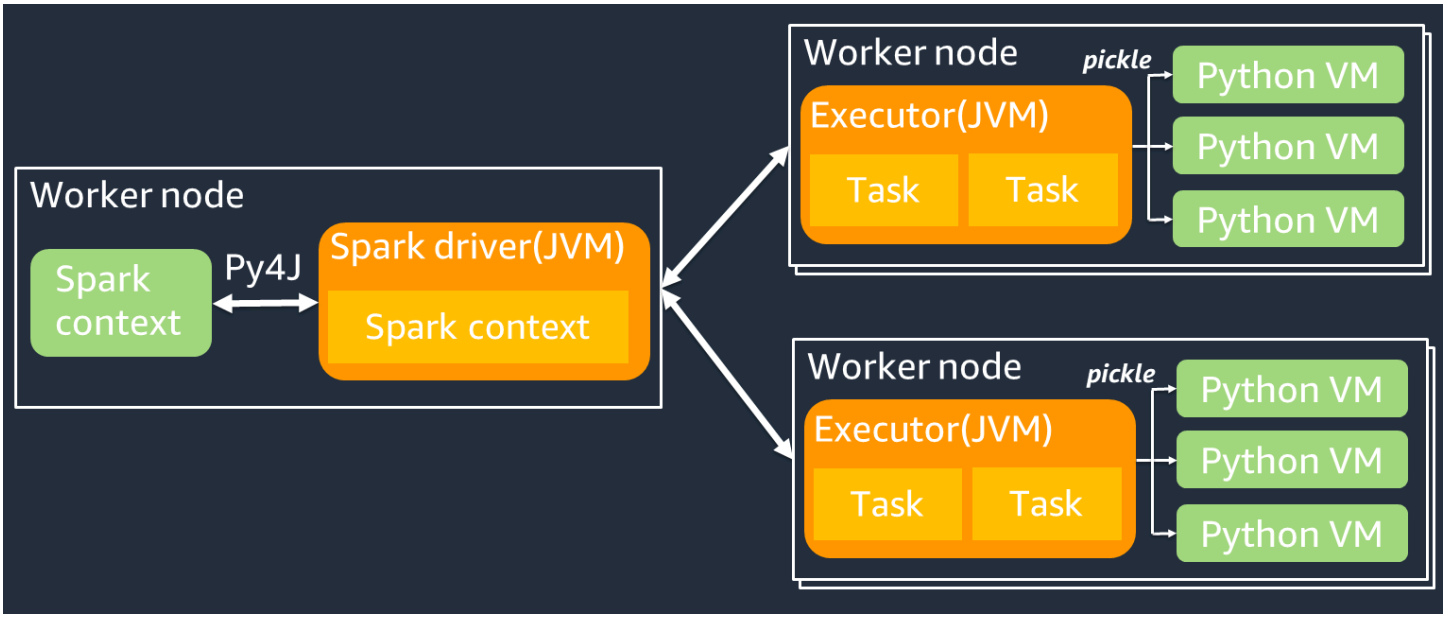
- 當 Amazon S3 有太多檔案時，請考慮平行處理任務頁面的太多分割區區段中有關過度平行處理的指引。
- 當 Amazon S3 具有太多分割區時，請考慮減少資料掃描頁面的太多 Amazon S3 分割區區段中有關過度分割區的指引。如果有許多分割區可降低從 Data Catalog 擷取分割區中繼資料的延遲，請啟用AWS Glue 分割區索引。如需詳細資訊，請參閱使用 AWS Glue 分割區索引改善查詢效能。
- 當 JDBC 有太多分割區時，請降低該`hashpartition`值。
- 當 DynamoDB 有太多分割區時，請降低該`dynamodb.splits`值。

- 當串流任務的分割區過多時，請降低碎片的數量。

最佳化使用者定義的函數

PySpark RDD.map中的使用者定義函數 UDFs) 和 通常會大幅降低效能。 PySpark 這是因為在 Spark 的基礎 Scala 實作中準確代表 Python 程式碼所需的額外負荷。

下圖顯示 PySpark 任務的架構。當您使用 PySpark 時，Spark 驅動程式會使用 Py4j 程式庫從 Python 呼叫 Java 方法。呼叫 Spark SQL 或 DataFrame 內建函數時，Python 和 Scala 之間的效能差異不大，因為這些函數使用最佳化的執行計畫在每個執行器的 JVM 上執行。



如果您使用自己的 Python 邏輯，例如使用 `map/ mapPartitions/ udf`，任務將在 Python 執行時間環境中執行。管理兩個環境會產生額外負荷成本。此外，記憶體中的資料必須進行轉換，以供 JVM 執行時間環境的內建函數使用。Pickle 是預設用於 JVM 和 Python 執行時間之間交換的序列化格式。不過，此序列化和還原序列化成本的成本非常高，因此以 Java 或 Scala 編寫的 UDFs 比 Python UDFs 更快。

若要避免 PySpark 中的序列化和還原序列化額外負荷，請考慮下列事項：

- 使用內建 Spark SQL 函數 – 考慮使用 Spark SQL 或 DataFrame 內建函數取代您自己的 UDF 或映射函數。執行 Spark SQL 或 DataFrame 內建函數時，Python 和 Scala 之間的效能差異不大，因為任務會在每個執行器的 JVM 上處理。
- 在 Scala 或 Java 中實作 UDFs – 考慮使用以 Java 或 Scala 撰寫的 UDF，因為它們在 JVM 上執行。

- 將 Apache Arrow 型 UDFs 用於向量化工作負載 – 考慮使用 Arrow 型 UDFs。此功能也稱為向量化 UDF (Pandas UDF)。Apache Arrow 是一種與語言無關的記憶體內資料格式， AWS Glue 可用來在 JVM 和 Python 程序之間有效率地傳輸資料。這目前對使用 Pandas 或 NumPy 資料的 Python 使用者最有益。

  Arrow 是一種單欄式（向量化）格式。其使用方式並非自動，可能需要稍微變更組態或程式碼，才能充分利用並確保相容性。如需詳細資訊和限制，請參閱 PySpark 中的 Apache Arrow。

  下列範例會比較標準 Python、向量化 UDF 和 Spark SQL 中的基本增量 UDF。

標準 Python UDF

範例時間為 3.20（秒）。

範例程式碼

```
# DataSet
df = spark.range(10000000).selectExpr("id AS a","id AS b")

# UDF Example
def plus(a,b):
    return a+b
spark.udf.register("plus",plus)

df.selectExpr("count(plus(a,b))").collect()
```

執行計畫

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- HashAggregate(keys=[], functions=[count(pythonUDF0#124)])
+- Exchange SinglePartition, ENSURE_REQUIREMENTS, [id=#580]
+- HashAggregate(keys=[], functions=[partial_count(pythonUDF0#124)])
+- Project [pythonUDF0#124]
+- BatchEvalPython [plus(a#116L, b#117L)], [pythonUDF0#124]
+- Project [id#114L AS a#116L, id#114L AS b#117L]
+- Range (0, 10000000, step=1, splits=16)
```

向量化 UDF

範例時間為 0.59（秒）。

向量化 UDF 比上一個 UDF 範例快 5 倍。檢查 Physical Plan時，您可以看到 ArrowEvalPython，其中顯示此應用程式是由 Apache Arrow 向量化。若要啟用向量化 UDF，您必須在程式碼spark.sql.execution.arrow.pyspark.enabled = true中指定 。

範例程式碼

```
# Vectorized UDF
from pyspark.sql.types import LongType
from pyspark.sql.functions import count, pandas_udf

# Enable Apache Arrow Support
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")

# DataSet
df = spark.range(10000000).selectExpr("id AS a","id AS b")

# Annotate pandas_udf to use Vectorized UDF
@pandas_udf(LongType())
def pandas_plus(a,b):
    return a+b
spark.udf.register("pandas_plus",pandas_plus)

df.selectExpr("count(pandas_plus(a,b))").collect()
```

執行計畫

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- HashAggregate(keys=[], functions=[count(pythonUDF0#1082L)],
 output=[count(pandas_plus(a, b))#1080L])
+- Exchange SinglePartition, ENSURE_REQUIREMENTS, [id=#5985]
+- HashAggregate(keys=[], functions=[partial_count(pythonUDF0#1082L)],
 output=[count#1084L])
+- Project [pythonUDF0#1082L]
+- ArrowEvalPython [pandas_plus(a#1074L, b#1075L)], [pythonUDF0#1082L], 200
+- Project [id#1072L AS a#1074L, id#1072L AS b#1075L]
+- Range (0, 10000000, step=1, splits=16)
```

Spark SQL

範例時間為 0.087（秒）。

Spark SQL 比量化 UDF 快得多，因為任務會在每個執行器的 JVM 上執行，而沒有 Python 執行時間 。如果您可以使用內建函數取代 UDF，建議您這麼做。

範例程式碼

```
df.createOrReplaceTempView("test")
spark.sql("select count(a+b) from test").collect()
```

將 panda 用於大數據

如果您已經熟悉 pandas 並想要將 Spark 用於大數據，則可以在 Spark. AWS Glue 4.0 和更新版本上使用 pandas API 來支援它。若要開始使用，您可以使用官方筆記本 Quickstart：Pandas API on Spark。如需詳細資訊，請參閱 PySpark 文件。

資源

- AWS Glue
- 效能調校 (Spark SQL 指南)
- AWS Glue 最佳化研討會

文件歷史紀錄

下表描述了本指南的重大變更。如果您想收到有關未來更新的通知，可以訂閱 RSS 摘要。

變更	描述	日期
初次出版	—	2024 年 1 月 2 日

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。