Monitoring U-SQL Execution

Overview

U-SQL jobs are executed in parallel. You can use the job graph, and Visual Tools for Azure Data Lake to monitor, analyze, and optimize U-SQL jobs.

This demo includes an optional exercise that requires:

- Visual Studio 2015 or greater
- Visual Studio Azure Tools and SDK

Creating an Azure Data Lake Analytics Account

Note: If you completed the previous labs, and still have your Azure Data Lake Analytics account, you can skip this exercise and proceed straight to *Reviewing a Job Graph*.

In this exercise, you will create an Azure Data Lake Analytics Account and associated Azure Data Lake store.

Create an Azure Data Lake Analytics Account

Before you can use Azure Data Lake Analytics to process data, you must create an Azure Data Lake Analytics account, and associate it with at least one Azure Data Lake store.

1. In a web browser, navigate to http://portal.azure.com, and if prompted, sign in using the Microsoft account that is associated with your Azure subscription.
2. In the Microsoft Azure portal, in the Hub Menu, click **New**. Then in the **Intelligence and analytics** menu, click **Data Lake Analytics**.
3. In the **New Data Lake Analytics Account** blade, enter the following settings, and then click **Create**:
    - **Name**: *Enter a unique name (and make a note of it!)*
    - **Subscription**: *Select your Azure subscription*
    - **Resource Group**: *Create a new resource group with a unique name*
    - **Location:** *Select any available region*
    - **Data Lake Store:** *Create a new Data Lake Store with a unique name (and make a note of it!)*
    - **Pin to dashboard**: *Not selected*
4. In the Azure portal, view **Notifications** to verify that deployment has started. Then wait for the resources to be deployed (this can take a few minutes.)

Upload Source Data Files

In this lab, you will use Azure Data Lake Analytics to process web server log data.

1. In the folder where you extracted the lab files, open the **iislogs** folder and then use a text editor to view the **2008-01.txt** file.
2. Review the contents of the file, noting that it contains some header rows (prefixed with a # character) and some space-delimited web server request records for the month of January in 2008. Then close the file without saving any changes. The other files in this folder contain similar data for February to June 2008.
3. In the Azure portal, view the **Data Explorer** page for your Azure Data Lake Analytics account, and create a new folder named **iislogs** in the root of your Azure Data Lake store.
4. Open the newly created **iislogs** folder. Then click **Upload**, select all of the files in the local **iislogs** folder (you can hold the CTRL key to select multiple files) and upload them.
5. Repeat the previous step to upload the **2008-07.txt** file from the local **July** folder to the **iislogs** folder in your Azure Data Lake store.

Create a Database

Creating a database enables you to store data in a structured format, ready to be queried by jobs.

1. In the Azure portal, on the blade for your Azure Data Lake Analytics account, click **New Job**.
2. In the **New U-SQL Job** blade, in the **Job Name** box, type **Create DB**.
3. In the code editor, enter the following code:

```
CREATE DATABASE IF NOT EXISTS webdata;

USE DATABASE webdata;

CREATE SCHEMA IF NOT EXISTS iis;

CREATE TABLE iis.log
(date string,
 time string,
client_ip string,
username string,
server_ip string,
port int,  method
string,  stem
string,  query
string,  status
string,
server_bytes int,
client_bytes int,
time_taken int?,
user_agent string,
referrer string,
 INDEX idx_logdate CLUSTERED (date))
DISTRIBUTED BY HASH(client_ip);


@log =
```

```
EXTRACT date string,
time string,
client_ip string,
username string,
server_ip string,
port int,         method
string,         stem
string,         query
string,         status
string,
server_bytes int,
client_bytes int,
time_taken int?,
user_agent string,
referrer string FROM
"/iislogs/{*}.txt"
USING Extractors.Text(' ', silent:true);

INSERT INTO iis.log
SELECT * FROM @log;

CREATE VIEW iis.summary
AS
SELECT date,
       COUNT(*) AS hits,
       SUM(server_bytes) AS bytes_sent,
       SUM(client_bytes) AS bytes_received
FROM iis.log
GROUP BY date;
```

4. Click **Submit Job**, and observe the job status as it runs.
5. When the job has finished running, return to the blade for your Azure Data Lake Analytics account and click **Data Explorer**.
6. In the **Data Explorer** blade, under **Catalog**, verify that the **webdata** database is now listed (alongside the default **master** database).

Reviewing a Job Graph

When you run a U-SQL job, the processing that must be performed is split into stages, and each stage is split into one or more vertices. This architecture defines the job graph, and you can examine it during and after job processing to help understand the data flows and processing tasks involved in completing the job, and to identify potential bottlenecks.

Run a Job in the Azure Portal

When you submit a job in the Azure portal, the default degree of parallelism is 1.

1. In the Azure portal, on the blade for your Azure Data Lake Analytics account, click **New Job**.
2. In the **New U-SQL Job** blade, in the **Job Name** box, type **Daily Averages**.
3. In the code editor, enter the following code:

```
USE DATABASE webdata;
```

```
@requests =
  SELECT DateTime.Parse(date) AS date,
        COUNT(*) AS requests,
        SUM(server_bytes) AS bytes_sent,
        SUM(client_bytes) AS bytes_received
  FROM iis.log
  GROUP BY DateTime.Parse(date);

@daily_averages =
  SELECT (int)date.DayOfWeek AS weekday,
date.ToString("dddd") AS dayname,          AVG(requests)
AS AvgRequests,
         AVG(bytes_sent) AS AvgBytesSent,
         AVG(bytes_received) AS AvgBytesRecvd
  FROM @requests
  GROUP BY (int)date.DayOfWeek,
date.ToString("dddd");

  OUTPUT @daily_averages
    TO "/Outputs/daily_averages.csv"
    ORDER BY weekday
    USING Outputters.Csv();
```

This code queries the **iis.log** table to return the number of requests and total bytes sent and received for each date. It then summarizes this resultset to return the average number of requests, bytes sent, and bytes received for each day of the week.

4. Ensure that the **Parallelism** property is set to **1**, and then click **Submit Job**.
5. As the job is running, observe the job graph as the job runs, noting the number of vertices, time taken, volume of data read and written, and rows returned in each stage.
6. When the job has finished, note the **Duration** in the **Job Summary** pane, and then expand the **Show More** section and verify that **Parallelism** for this job was set to **1**.
7. Note the number of seconds indicated for the **Running** phase of the job – this indicates the amount of time spent processing the job vertices.
8. Above the job graph diagram, note that the display is currently configured to show **Progress**. Then in the drop-down list, select **Time**, **Read**, and **Written** in turn to see viosualizations for these metrics.
9. Reset the view to **Progress**, and then click the **Start Playback** (▶) button. This replays the job visualization as a speeded-up time-lapse, enabling you to see how long each stage took to complete.
10. Click the **Output** tab and select **daily_averages.csv** to see a preview of the results generated by the job.

Increase the Degree of Parallelism(AUs)

Before you run a job, you can specify the degree of parallelism you want to reserve for the job. This determines how many vertices can be executed concurrently as the job runs. The cost of a job is calculated as *degree of parallelism* x *job duration*, so cost-optimization is a balance between the **Parallelism** value reserved and the number of minutes saved through increased parallelism.

1. In the Azure portal, return to the **New U-SQL Job** blade for the **Daily Averages** job.
2. Increase the **AUs** property to **4**, and then click **Submit Job**.
3. As the job is running, observe the job graph as the job runs, noting the number of vertices, time taken, volume of data read and written, and rows returned in each stage.
4. When the job has finished, note the **Duration** in the **Job Summary** pane, and then expand the **Show More** section and verify that **Parallelism** for this job was set to **4**.
5. Note the number of seconds indicated for the **Running** phase of the job – this indicates the amount of time spent processing the job vertices.
6. Click the **Output** tab and select **daily_averages.csv** to verify that the job returned the same results as before.

Note: With such a small volume of data, and relatively trivial query logic, the time saved by increasing parallelism is not particularly striking. However, with a larger volume of data, you can significantly optimize more complex jobs by increasing the degree of parallelism.

Using the Visual Studio Tools for Azure Data Lake

The Visual Studio tools for Azure Data Lake include U-SQL job analysis and optimization capabilities.

Note: To complete this exercise, you must be using a Windows computer with Visual Studio and the Visual Studio Azure Tools and SDK installed.

Create a U-SQL Visual Studio Project

As you learned in the previous lab, the Visual Studio Azure Tools and SDK includes templates for Azure Data Lake projects.

1. Start Visual Studio.
2. Create a new project named **DailyRequestsByIP** based on the **U-SQL Project** template in the **Azure Data Lake\U-SQL (ADLA)** category. Save the project in the folder where you extracted the lab files.
3. View the **Cloud Explorer** pane and if you are not already connected, sign into your Azure account in this pane. After you have signed in, you should be able to expand the **Data Lake Analytics** node under your subscription to see your Azure Data Lake Analytics service.

Submit a Job

To experiment with the job analysis capabilities in Visual Studio, you will submit a job to your Azure Data Lake Analytics account.

1. If it is not already open, open the **Script.usql** code file from the **Solution Explorer** pane.
1. In the code editor pane, in the **Data Lake Analytics Accounts** drop-down list (in which **(local)** is currently selected), select your Azure Data Lake Analytics account. Then in the **Databases** list (in which **master** is currently selected), select **webdata**.
2. Add the following U-SQL code to the **Script.usql** code file:

```
@months =
  SELECT DISTINCT DateTime.Parse(date).Month AS month,
         DateTime.Parse(date).ToString("MMMM") AS month_name
  FROM iis.log;
```

```
@ip_requests =
  SELECT DateTime.Parse(date) AS date,
client_ip
  FROM iis.log;

@monthly_counts =
  SELECT date.Month AS month,
client_ip,
         COUNT(*) AS requests
  FROM @ip_requests
GROUP BY date.Month,
client_ip;

@named_monthly_counts =
    SELECT m.month, m.month_name, c.client_ip, c.requests
    FROM @monthly_counts AS c
         INNER JOIN
             @months AS m
         ON c.month == m.month;

OUTPUT @named_monthly_counts
  TO "/Outputs/monthly_counts_by_ip.csv"
  ORDER BY month, client_ip
  USING Outputters.Csv();
```

This code generates a table of month numbers and names, and then retrieves the date and client IP address of each requests before aggregating the client IP addresses by month and joining the table of months to return the month name for each month.

Note: This query is more complicated than it needs to be to return the desired output. This is by design so that a more complex job graph is generated.

3. Save the **Script.usql** file. Then, in the code editor pane, in the **Submit** drop-down list. click **Advanced**.
4. In the **Submit Job** dialog box, set the job name to **Get Monthly IP**, ensure that your Azure Data Lake Analytics account is selected, and set **Parallelism** value to **4/30**. Then click **Submit**.
5. Observe the job details as it runs in the **Job View** pane.
6. When the job has finished, in the job graph, right-click the **monthly_counts_by_ip.csv** output and click **Preview**.
7. When the preview opens, verify that the output includes the total requests for each month and IP address.

Review Job Execution

In Visual Studio, you can replay at an increased speed to observe how each stage was executed. This can be extremely useful when you need to identify potential issues in jobs that run for several hours or more.

1. Return to the **Job View** pane, and at the top of the pane, click **Load Profile**.
2. Beneath the job graph diagram, under **Job Playback**, click the play (▶) button.

3. Review the job graph diagram as the time-lapse job execution is displayed, noting any stages that appear to take comparatively longer than the others; or any stages that include retried vertices (indicated as an orange color).
4. Double-click any of the stages to see the details data flow processes it encapsulated.
5. Return to the **job View** pane and change the graph view from **Progress** to **Data read**, and review the information shown in the job graph diagram. Note that you can play back the job in this view to see it as a time-lapse (or just use the slider to view a specific point in time).
6. Review the remaining job graph views, which include **Data written**, **Compute time**, **Average execution time per node**, **Input throughput**, and **Output throughput**. Each of these views offers a different insight into the job execution.

Diagnose Job Issues and Evaluate Efficiency

In addition to reviewing details of job execution tasks, you can use Visual Studio to detect any potential issues in your job and to evaluate how efficiently it used parallelism.

1. In the **Job View** pane, view the **Diagnostics** tab, and note any issues that are displayed here.
2. Click **Resource Usage** and review the graph that is displayed.

   The blue line represents the degree of parallelism you allocated for the job, and the green and red lines indicate how much parallelism was used to queue and run the job.

   If there are significant gaps between the peaks of the queuing and running lines, and the parallelism allocation line, the job may not require the degree of parallelism that was specified – and you could reduce costs by decreasing it.

   Conversely, if the queuing and running lines are frequently constrained by the allocation line, you could potentially reduce the time taken for the job to run by increasing the degree of parallelism.

3. Close Visual Studio, saving your work if prompted.