DATA( ) {
EXPOSED;

ADF Data Flows Best Practices and Performance Optimizations

Mark Kromer @kromerbigdata
ADF Program Manager

Agenda

- Azure SQL DB ETL Performance
- Transformation optimizations
- Monitoring
- Global Settings
- Best Practices
- Azure Integration Runtimes

Database ETL Performance

Sample timings for Azure SQL DB

Scenario w/Azure SQL DB

- Source: Azure SQL DB Table
- Sink: Azure SQL DB Table
- Table size: 74 columns, 887k rows
- Transforms: Single derived column to mask 3 fields
- Time: 3 mins end-to-end using memory optimized 80-core debug Azure IR
- Recommended settings: Source partitioning on SQL DB Source, current partitioning on Derived Column and Sink

SQL Database Timing

Performance run for csv to Azure sql

Dataset size	15GB	Dataset size	31.44 GB
Storage type	csv to Azsql	Storage type	Azsql to csv
Total Rows	71 Million	Total Rows	71 Million
Total Partition	114	Total Partition	6,8
Compute type	General Purpose	Compute type	General Purpose
Sink type	Azsql	Sink type	CSV
AZSQL Type	Genral purpose: Gen5, 16 vCores	AZSQL Type	Genral purpose: Gen5, 16 vCores

Core	Stage time in Minutes	Sink time in Minutes	azsql to csv with optimize source partition(6) sink time in minutes	azsql to csv with optimize source partition(8) Sink time in minutes
8	70.47	94.15	9.41	10.34
16	42.12	69	9.25	8.18
32	29.29	55.25	8.3	6.49
64	22.7	44	8.49	7.15

Note:Cluster uptime not included

Sink time is total job execution time

Total job exection time is only sink time

After selecting the optimize with source 6 partition performace has improved

Synapse DW Timing

Dataset size	15GB		45GB		Dataset size	15GB
Storage type	CSV TO DW		Blob to Blob		Storage type	DW TO CSV
Total Rows	71 Million		337 million		Total Rows	71 Million
Performance level	Gen2: DW1000c		Gen2: DW1000c		Performance level	Gen2: DW1000c
Total partition	114		358		Total partition	1
Core	**sink stage time in minutes**	**sink time in minutes**	**Sink stage time in minutes**	**sink time in minutes**	**stage time in minutes**	**sink time in minutes**
8	5.11	30.54	24.23	90.41	45.57	46
16	3.18	30.1	13.59	79.17	42.27	42.28
32	3.23	27.2	7.36	81.16	42.11	42.12
64	1.13	26.12	5.14	77.55	45.1	45.13

Compute type: General Purpose

Note:-

Cluster uptime not included

Sink time is total job execution time

Adding cores proportionally decreases time it takes to process data into staging files for Polybase. However, there is a fairly static amount time that it takes to write that data from Parquet into SQL tables using Polybase.

CosmosDB Timing

Performance run for csv to Cosmos db and json to Cosmos DB						
Dataset size	15GB		45GB(csv data converted to json)		Dataset size	15GB
Storage type	CSV TO COSMOS		JSON TO COSMOS (multi region write enabled)		Storage type	COSMOS TO CSV
Total Rows	71 Million		71 Million		Total Rows	71 Million
Cosmos DB Throughput	100000		100000		Cosmos DB Throughput	100000
Core	**sink time in minutes**	**Number of Partition**	**Number of partition**	**Sink time in minutes**	**Number of partition**	**Sink time in minutes**
8	300	114	352	448	10	34.56
16	280	114	352	316	10	21.37
32	270	114	352	257	10	18.28
64	260	114	352	238	10	18.37
128	301	128	352	226		
256	400	255	352	225		

Note:- Out put data genrated in cosmos db:170 GB Data size:123:52 GB Index size:45:31 GB

Cluster uptime not included

Sink time is total job execution time

Note:- when input dataset in csv format if number of expected partition is less then cpu core then performance goes down due to more number of partition data got created

Note:- From csv to cosmos DB with multi region write enabled and disable results are same

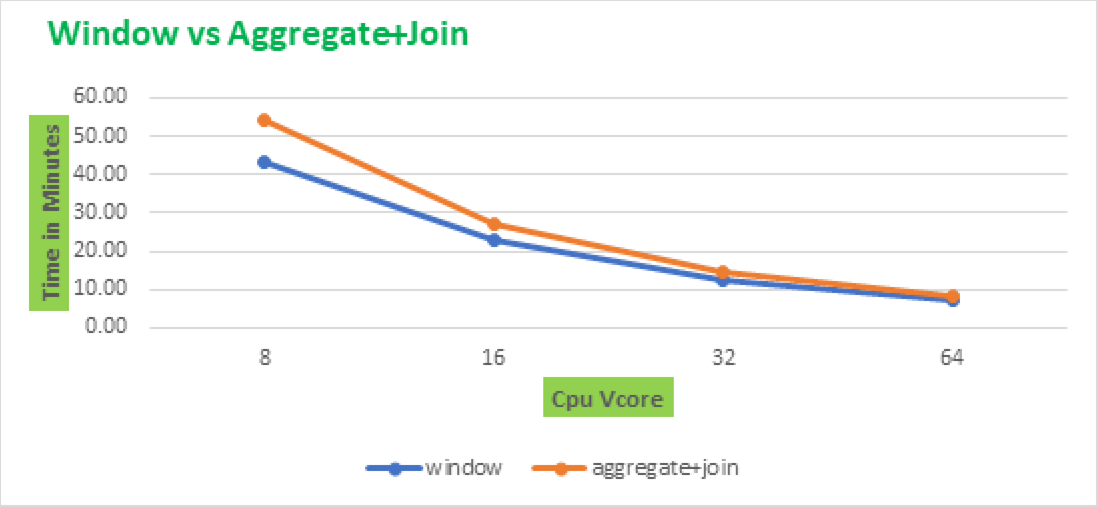
when cpu core is more then number of partition performance get decreased

Compute type: General Purpose

Transformation Performance

Window / Aggregate Timing

Dataset size	76GB		Tpch dataset has been used for tests Lineitem table size 76Gb Aggregate on sum L_TAX window on L_SUPPLYKEY
Storage type	ADLSGEN2		
Total Rows	600Million	600013353	
Result time	Sink time		
Compute type	General Purpose		
Sink type	CSV		
Total Partition	199	197	
Core	**window**	**aggregate+join**	
8	43.44	54	
16	23.11	27.17	
32	12.49	14.36	
64	7.40	8.51	



Compute type: General Purpose

- Performance improvement scales proportionately with increase in Vcores
- 8 Vcore to 64 Vcore performance increase is around 5 times more

Transformation Timings

Performance run for CSV Transformation

Dataset size	15GB,20GB
Storage type	ADLSGEN2
Total Rows	71 Million
Result time	Sink time
Total Partition	200

Core	csv left join	csv lookup	Exists
8	14.26	28.49	12.5
16	7	22.45	12.48
32	5.3	9.5	5.15
64	2.44	5.38	3.14

Note:- Each run has ran sequentially
Cluster uptime not included
Note: Time in Minutes, All number is avrage of two runs

Goal: Find which transformation is faster among these left join, lookup, Exists. These all transformation has same output

Compute type: General Purpose

Dataset size	15GB,20GB
Storage type	ADLSGEN2
Total Rows	71 Million
Result time	Sink time
Total Partition	200

Core	window	aggregate+join (two source)	aggregate + join (one source)
8	8.25	10.49	10.19
16	7.56	7.1	6.25
32	3.4	3.54	4.12
64	2.44	2.46	2.21

Transformation recommendations

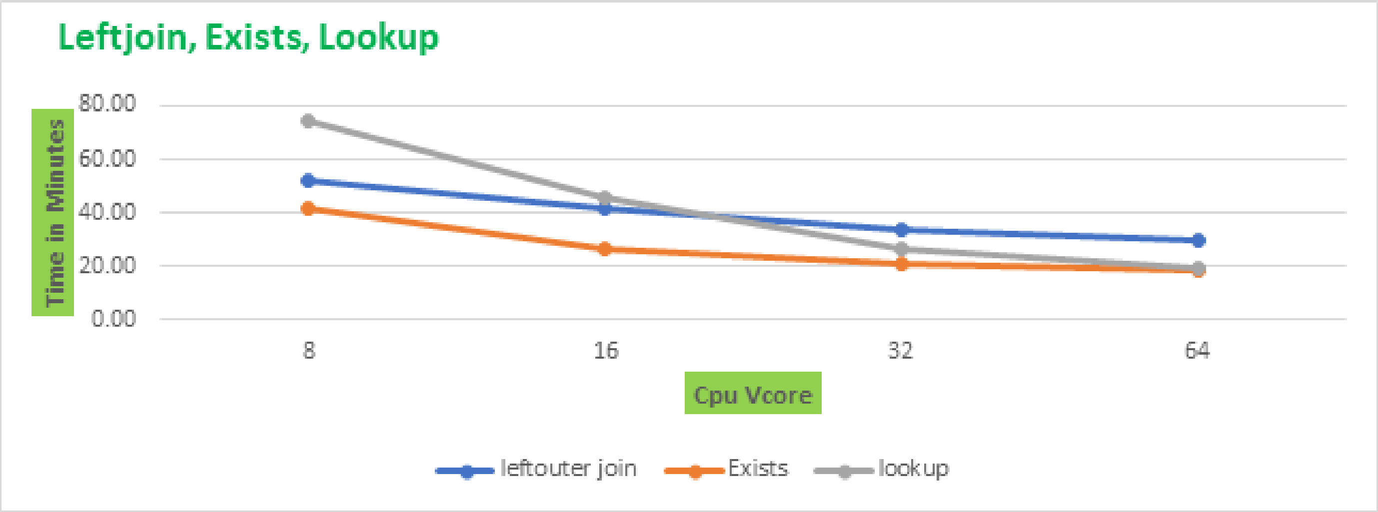
- When ranking data across entire dataset, use Rank transformation instead of Window with rank()
- When using rowNumber() in Window to uniquely add a row counter to each row across entire dataset, instead use the Surrogate Key transformation

TPCH Timings

TPCH CSV in ADLS Gen 2

Dataset size	76GB,16GB		Tpch dataset has been used for tests Lineitem table size 76Gb Order Table size 16Gb Column O_ORDERKEY
Storage type	ADLSGEN2		
Total Rows	600Million	600013353	
Result time	Sink time		
Total Partition	175	190	200
Core	**leftouter join**	**Exists**	**lookup**
8	52.23	41.9	74.57
16	41.59	26.55	45.29
32	33.53	20.49	26.53
64	29.59	18.23	19.17

Compute type: General Purpose



Optimizing transformations

Each transformation has its own optimize tab
- Generally better to not alter -> reshuffling is a relatively slow process

Reshuffling can occur if data is very skewed
- One node has a disproportionate amount of data

For Joins, Exists and Lookups:
- If you have a many of these transforms, memory optimized greatly increases performance
- Use cached lookup w/cached sink
- Can ‘Broadcast’ if the data on one side is small
- **Rule of thumb: Less than 50k rows**

Use Window transformation partitioned over segments of data
- For Rank() across entire dataset, use the Rank transformation instead
- For RowNumber() across entire dataset, use the Surrogate Key transformation instead

Transformations that require reshuffling like Sort negatively impact performance

Azure Data Factory Data Flow Performance

** Includes cold cluster start-up time*

IR Size	ADLS Source File				Load						Sink					
	File Type	Single File Size	File Count	Total MB	Type	Partition		Load Time		MB/s	Type	Partition		sink process time		MB/s
						num	size	min	sec			num	size	min	sec	
8-8-general	small	1MB	51200	50807	CDM	1970	26MB	12	4	70	CDM	1970	26MB	14	8	60
				50807	CDM	1970	26MB	11	28	74	Dataset	1970	26MB	13	22	63
				50807	Dataset	1970	26MB	11	36	73	CDM	1970	26MB	12	51	66
				50807	Dataset	1970	26MB	9	28	89	Dataset	1970	26MB	10	44	79
8-8-general	medium	242MB	210	50791	CDM	210	242MB	6	37	128	CDM	210	242MB	6	47	125
				50791	CDM	210	242MB	6	44	126	Dataset	210	242MB	6	55	122
				50791	Dataset	210	242MB	7	22	115	CDM	210	242MB	7	40	110
				50791	Dataset	210	242MB	5	50	145	Dataset	210	242MB	6	10	137
8-8-general	large	1067MB	50	50794	CDM	50	1067MB	6	23	133	CDM	50	1067MB	6	27	131
				50794	CDM	50	1067MB	7	6	119	Dataset	50	1067MB	7	10	118
				50794	Dataset	50	1067MB	8	6	105	CDM	50	1067MB	8	23	101
				50794	Dataset	50	1067MB	5	31	153	Dataset	50	1067MB	5	49	146
16-16-general	small	1MB	51200	50807	CDM	1970	26MB	5	47	146	CDM	1970	26MB	7	59	106
				50807	CDM	1970	26MB	5	49	146	Dataset	1970	26MB	7	49	108
				50807	Dataset	1970	26MB	5	40	149	CDM	1970	26MB	7	3	120
				50807	Dataset	1970	26MB	4	21	195	Dataset	1970	26MB	6	13	136
16-16-general	medium	242MB	210	50791	CDM	210	242MB	3	45	226	CDM	210	242MB	3	54	217
				50791	CDM	210	242MB	3	50	221	Dataset	210	242MB	4	9	204
				50791	Dataset	210	242MB	3	58	213	CDM	210	242MB	4	16	198
				50791	Dataset	210	242MB	2	42	314	Dataset	210	242MB	3	9	269
16-16-general	large	1067MB	50	50794	CDM	50	1067MB	4	19	196	CDM	50	1067MB	4	22	194
				50794	CDM	50	1067MB	3	54	217	Dataset	50	1067MB	3	59	213
				50794	Dataset	50	1067MB	4	24	192	CDM	50	1067MB	4	41	181
				50794	Dataset	50	1067MB	2	43	312	Dataset	50	1067MB	3	2	279
4-4-general	small	1MB	51200	50807	CDM	1970	26MB	21	37	39	CDM	1970	26MB	23	31	36
				50807	CDM	1970	26MB	21	8	40	Dataset	1970	26MB	22	54	37
				50807	Dataset	1970	26MB	19	24	44	CDM	1970	26MB	20	48	41
				50807	Dataset	1970	26MB	15	5	56	Dataset	1970	26MB	16	19	52
4-4-general	medium	242MB	210	50791	CDM	210	242MB	12	17	69	CDM	210	242MB	12	27	68
				50791	CDM	210	242MB	12	31	68	Dataset	210	242MB	12	50	66
				50791	Dataset	210	242MB	12	48	66	CDM	210	242MB	13	5	65
				50791	Dataset	210	242MB	9	24	90	Dataset	210	242MB	9	50	86
4-4-general	large	1067MB	50	50794	CDM	50	1067MB	12	45	66	CDM	50	1067MB	12	48	66
				50794	CDM	50	1067MB	12	34	67	Dataset	50	1067MB	12	50	66
				50794	Dataset	50	1067MB	12	58	65	CDM	50	1067MB	13	15	64
				50794	Dataset	50	1067MB	9	6	93	Dataset	50	1067MB	9	24	90

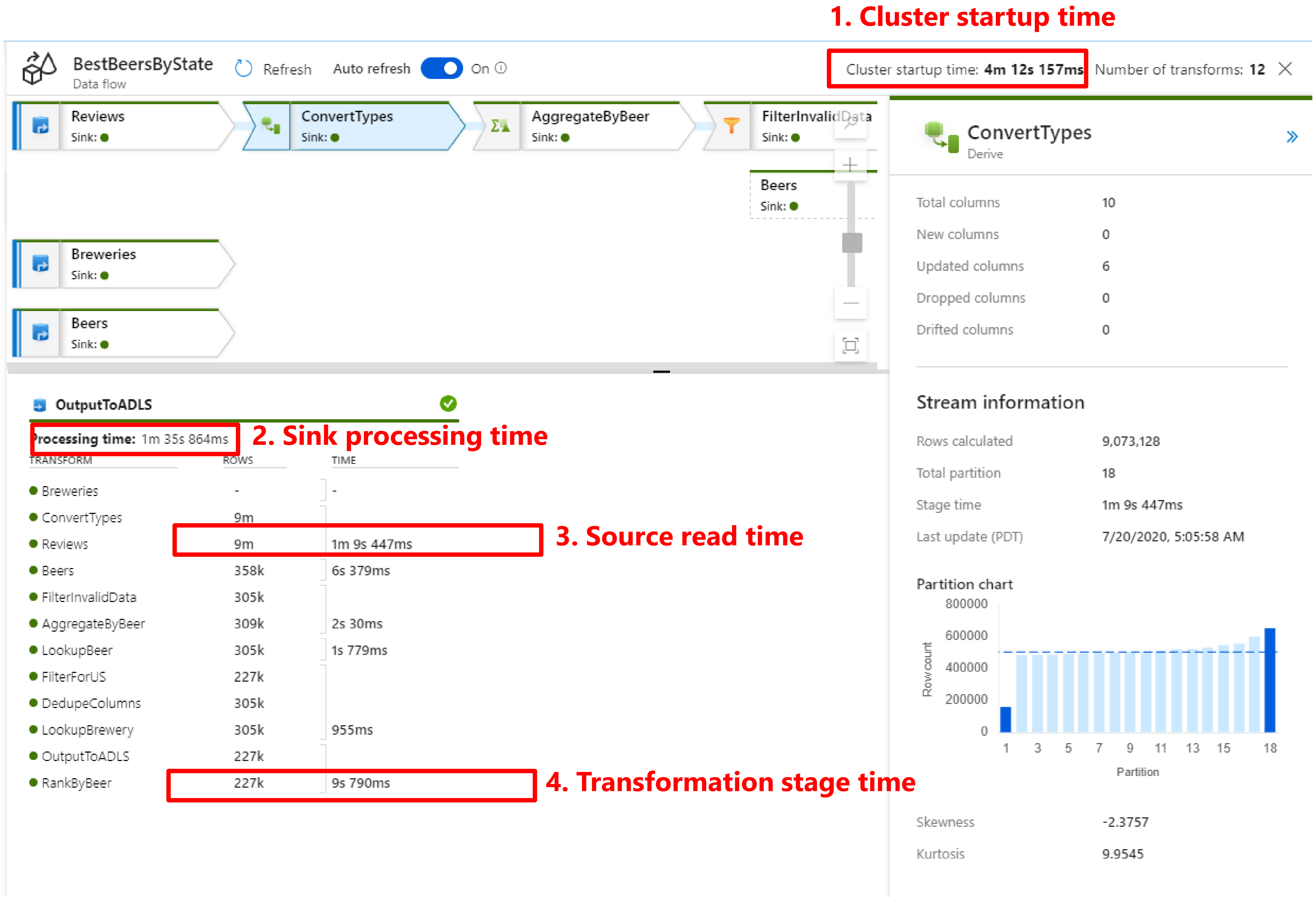
Azure Synapse Data Flow Performance

** Includes cold cluster start-up time*

IR Size	ADLS Source File				Load						Sink					
	File Type	Single File Size	File Count	Total MB	Type	Partition		Load Time		MB/s	Type	Partition		sink process time		MB/s
						num	size MB	min	sec			num	size MB	min	sec	
8-8-general	small	1MB	51200	50807	CDM	250	204	10	36	80	CDM	250	204	11	20	75
				50807	CDM	250	204	11	21	75	Dataset	250	204	12	1	70
				50807	Dataset	250	204	12	9	70	CDM	250	204	12	19	69
				50807	Dataset	250	204	7	56	107	Dataset	250	204	8	8	104
8-8-general	medium	242MB	210	50791	CDM	53	958	8	47	96	CDM	53	958	8	51	96
				50791	CDM	53	958	8	15	103	Dataset	53	958	8	20	102
				50791	Dataset	53	958	8	55	95	CDM	53	958	8	58	94
				50791	Dataset	53	958	5	31	153	Dataset	53	958	5	35	152
8-8-general	large	1067MB	50	50794	CDM	50	1067	9	25	90	CDM	50	1067	9	28	89
				50794	CDM	50	1067	8	44	97	Dataset	50	1067	8	48	96
				50794	Dataset	50	1067	8	39	98	CDM	50	1067	8	42	97
				50794	Dataset	50	1067	6	8	138	Dataset	50	1067	6	11	137
16-16-general	small	1MB	51200	50807	CDM	250	204	5	54	144	CDM	250	204	6	30	130
				50807	CDM	250	204	5	52	144	Dataset	250	204	6	30	130
				50807	Dataset	250	204	6	27	131	CDM	250	204	6	38	128
				50807	Dataset	250	204	4	21	195	Dataset	250	204	4	30	188
16-16-general	medium	242MB	210	50791	CDM	53	958	5	26	156	CDM	53	958	5	30	154
				50791	CDM	53	958	5	21	158	Dataset	53	958	5	26	156
				50791	Dataset	53	958	5	22	158	CDM	53	958	5	25	156
				50791	Dataset	53	958	5	40	149	Dataset	53	958	5	44	148
16-16-general	large	1067MB	50	50794	CDM	50	1067	5	45	147	CDM	50	1067	5	49	146
				50794	CDM	50	1067	5	15	161	Dataset	50	1067	5	19	159
				50794	Dataset	50	1067	5	32	153	CDM	50	1067	5	35	152
				50794	Dataset	50	1067	4	24	192	Dataset	50	1067	4	27	190
4-4-general	small	1MB	51200	50807	CDM	250	204	20	36	41	CDM	250	204	21	33	39
				50807	CDM	250	204	24	57	34	Dataset	250	204	25	55	33
				50807	Dataset	250	204	23	29	36	CDM	250	204	23	41	36
				50807	Dataset	250	204	16	20	52	Dataset	250	204	16	38	51
4-4-general	medium	242MB	210	50791	CDM	53	958	17	58	47	CDM	53	958	18	3	47
				50791	CDM	53	958	17	34	48	Dataset	53	958	17	41	48
				50791	Dataset	53	958	19	36	43	CDM	53	958	19	39	43
				50791	Dataset	53	958	10	41	79	Dataset	53	958	10	45	79
4-4-general	large	1067MB	50	50794	CDM	50	1067	17	21	49	CDM	50	1067	17	27	49
				50794	CDM	50	1067	18	46	45	Dataset	50	1067	18	51	45
				50794	Dataset	50	1067	19	4	44	CDM	50	1067	19	8	44
				50794	Dataset	50	1067	11	0	77	Dataset	50	1067	11	3	77

ETL Performance Monitoring

Identifying bottlenecks



1. Sequential executions can lower the cluster startup time by setting a TTL in Azure IR
2. Total time to process the stream from source to sink. There is also a post-processing time when you click on the Sink that will show you how much time Spark had to spend with partition and job clean-up. Write to single file and slow database connections will increase this time
3. Shows you how long it took to read data from source. Optimize with different source partition strategies
4. This will show you bottlenecks in your transformation logic. With larger general purpose and mem optimized IRs, most of these operations occur in memory in data frames and are usually the fastest operations in your data flow

Global configurations that effect performance

Logging level (pipeline activity)

- Verbose (default) is most expensive
- You can get a small increase in performance for large data flows without detailed logging
- Trade-off: Less diagnostics

Error row handling (sink transformation)

- Expect 5%-10% perf hit
- Trade-off: Provides detailed logging and continuation of data flow on database driver errors

Run in parallel (pipeline activity)

- Currently only available for “connected” streams, i.e. multiple sinks from a single stream
- Can write to multiple sinks at same time
- Use with new branch, conditional split

Parallel activity executions (pipeline activity)

- If you place data flow activities on your pipeline canvas without connector lines, your data flows can all start at the same time, lowering overall pipeline execution times.

ETL Performance Best Practices

Best practices - Sources

When reading from file-based sources, data flow automatically partitions the data based on size

~128 MB per partition, evenly distributed

Use current partitioning will be fastest for file-based and Synapse using PolyBase

Enable staging for Synapse

For Azure SQL DB, use Source partitioning on column with high cardinality

Improves performance, but can saturate your source database

Reading can be limited by the I/O of your source

Best practices – Debug (Data Preview)

Data Preview

Data preview is inside the data flow designer transformation properties
Uses row limits and sampling techniques to preview data from a small size of data
Allows you to build and validate units of logic with samples of data in real time
You have control over the size of the data limits under Debug Settings
If you wish to test with larger datasets, set a larger compute size in the Azure IR when switching on "Debug Mode"
Data Preview is only a snapshot of data in memory from Spark data frames. This feature does not write any data, so the sink drivers are not utilized and not tested in this mode.

Best practices – Debug (Pipeline Debug)

Pipeline Debug

Click debug button to test your data flow inside of a pipeline

Default debug limits the execution runtime so you will want to limit data sizes

Sampling can be applied here as well by using the “Enable Sampling” option in each Source

Use the debug button option of “use activity IR” when you wish to use a job execution compute environment

This option is good for debugging with larger datasets. It will not have the same execution timeout limit as the default debug setting

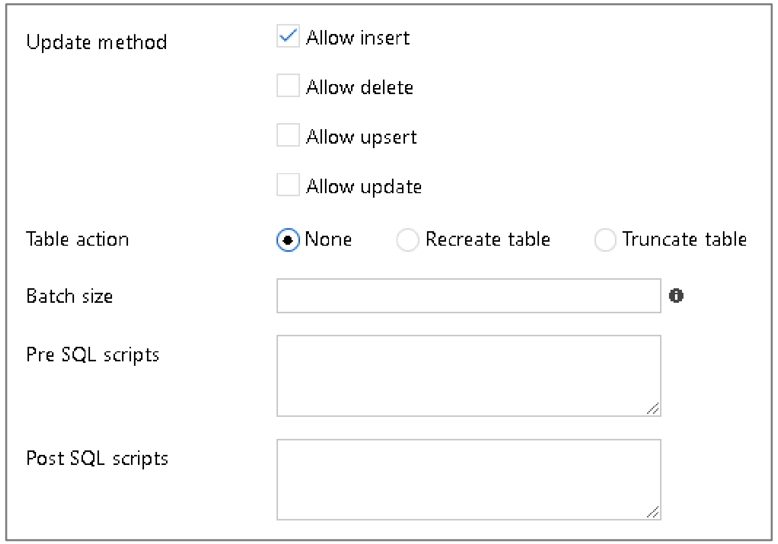
Best practices - Sinks

SQL:

Disable indexes on target with pre/post SQL scripts
Increase SQL capacity during pipeline execution
Enable staging when using Synapse
Use Source Partitioning on Source under Optimize
Set number of partitions based on size of IR

File-based sinks:

Use current partitioning allows Spark to create output
Output to single file is a slow operation
Often unnecessary by whoever is consuming data
Can set naming patterns or use data in column
Any reshuffling of data is slow



Cosmos DB

Set throughput and batch size to meet performance requirements

Azure Integration Runtime Best Practices

Data Flows use JIT compute to minimize running expensive clusters when they are mostly idle

Generally more economical, but each cluster takes ~4 minutes to spin up

IR specifies what cluster type and core-count to use

Memory optimized is best, compute optimized doesn’t generally work for production workloads

When running Sequential jobs utilize *Time to Live* to reuse cluster between executions

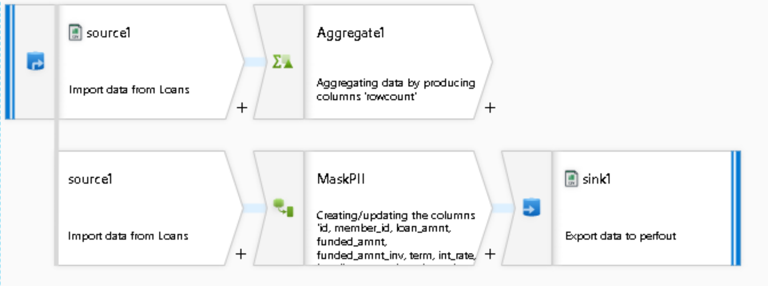
Keeps compute resources alive for TTL minutes after execution for new job to use

Maximum one job per cluster

Reduces job startup latency to ~1.5 minutes

Rule of thumb: start small and scale up

Azure IR – General Purpose

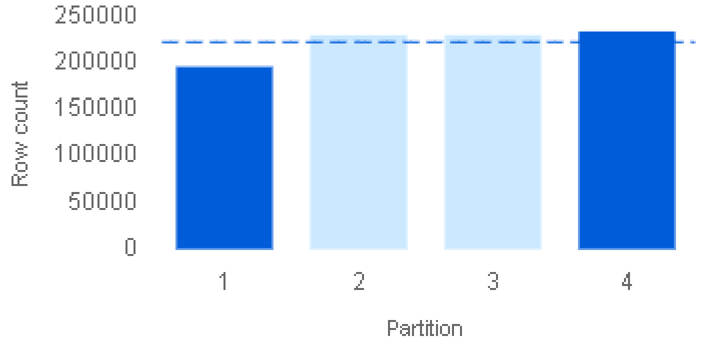


- This was General Purpose 4+4, the default auto resolve Azure IR
- For prod workloads, GP is usually sufficient at >= 16 cores
- You get 1 driver and 1 worker node, both with 4 vcores
- Good for debugging, testing, and many production workloads
- Tested with 887k row CSV file with 74 columns
- Default partitioning
  - Spark chose 4 partitions
- Cluster startup time: 4.5 mins
- Sink IO writing: 46s
- Transformation time: 42s
- Sink post-processing time: 45s

Stream information

Rows calculated	887,379
Total partition	4
Stage time	42s 669ms
Last update (PDT)	8/10/2020, 5:56:24 PM

Partition chart



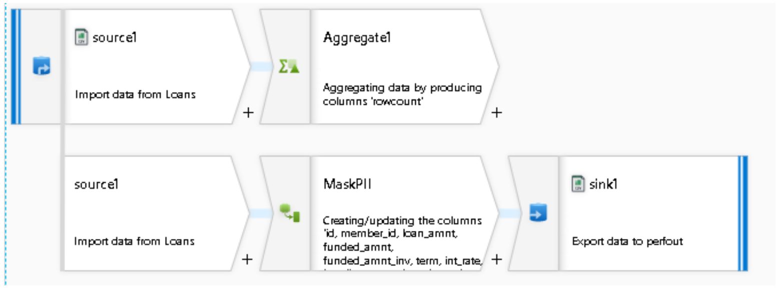
Skewness	-0.9657
Kurtosis	1.7295
Sink processing time	46s 943ms

TYPE	RUN START	DURATION	STATUS	INTEGRATION RUNTIME
ExecuteDataFlov	2020-08-11T00:50:36.48789	00:06:17	Succeeded	DefaultIntegrationRuntime (East US)

Azure IR – Compute Optimized

- Computed Optimized intended for smaller workloads
- 8+8, this is smallest CO option and you get 1 driver and 2 workers
- Not suitable for large production workloads
- Tested with 887k row CSV file with 74 columns
- Default partitioning
  - Spark chose 8 partitions
- Cluster startup time: 4.5 mins
- Sink IO writing: 20s
- Transformation time: 35s
- Sink post-processing time: 40s
- More worker nodes gave us more partitions and better perf than General Purpose

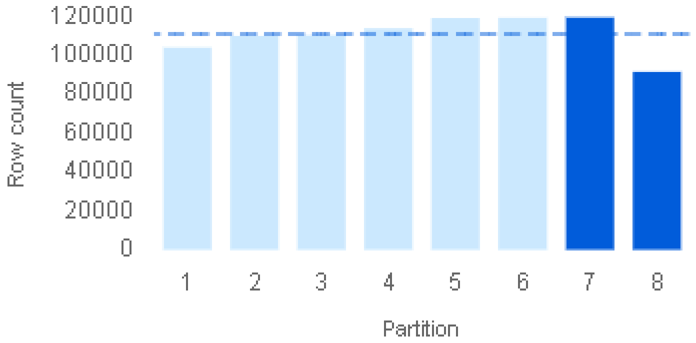
TYPE	RUN START	DURATION	STATUS	INTEGRATION RUNTIME
ExecuteDataFlow	2020-08-11T01:23:35.7624	00:05:37	Succeeded	DefaultIntegrationRuntime (East US)



Stream information

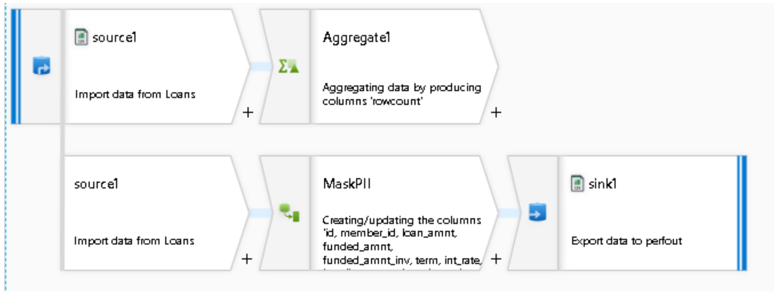
Rows calculated	887,379
Total partition	8
Stage time	23s 901ms
Last update (PDT)	8/10/2020, 6:28:37 PM

Partition chart



Skewness	-0.9488
Kurtosis	2.6823
Sink processing time	25s 638ms

Azure IR – Memory Optimized



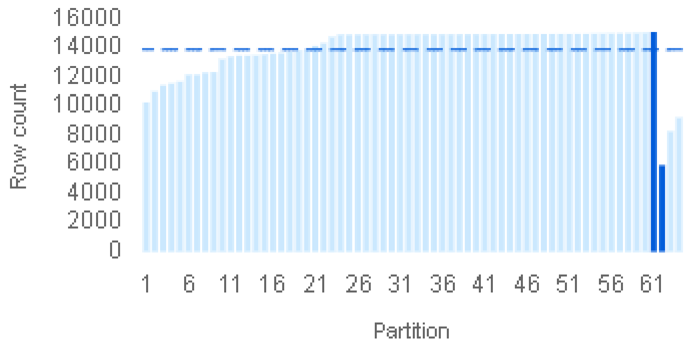
- Memory Optimized well suited for large production workload reliability with many aggregates, lookups, and joins
- 64+16 gives you 16 vcores for driver and 64 across worker nodes
- Tested with 887k row CSV file with 74 columns
- Default partitioning
- Spark chose 64 partitions
- Cluster startup time: 4.8 mins
- Sink IO writing: 19s
- Transformation time: 17s
- Sink post-processing time: 40s

TYPE	RUN START	DURATION	STATUS	INTEGRATION RUNTIME
ExecuteDataFlov	2020-08-11T01:35:20.95875	00:06:09	Succeeded	DefaultIntegrationRuntime (East US)

Stream information

Rows calculated	887,379
Total partition	64
Stage time	17s 42ms
Last update (PDT)	8/10/2020, 6:40:58 PM

Partition chart



Skewness	-2.2327
Kurtosis	8.2938
Sink processing time	19s 254ms

Resources

Complete Data Flows Performance Tuning and Profiles Deck

https://www2.slideshare.net/kromerm/azure-data-factory-data-flow-performance-tuning-101

Data Flows Training

https://www2.slideshare.net/kromerm/azure-data-factory-data-flows-training-sept-2020-update

Data Flows Video Tutorials

https://docs.microsoft.com/en-us/azure/data-factory/data-flow-tutorials

Data Flows Performance Home Page

https://docs.microsoft.com/en-us/azure/data-factory/concepts-data-flow-performance

Copy Data Performance Guidance

https://docs.microsoft.com/en-us/azure/data-factory/copy-activity-performance

DATA() {
EXPOSED;

Learn with us!

View our on-demand playlist:
aka.ms/azuresqlandadf

@AzureSQL
@AzDataFactory

Microsoft Azure