

# [CosmosDB] - Write data into cosmos collection

Last updated by | Jackie Huang | Jan 4, 2022 at 12:24 AM PST

---

## Contents

- [Issue](#)
- [Root Cause](#)
- [Resolution](#)
- [Reference](#)
- [Additional Information](#)

## Issue

When customer uses dataflow to move/transfer data from different stores into CosmosDB, the activity keep running for long time with no data written or has low performance.

## Root Cause

Because of wrong settings on partitioning, throughput, writeThroughputBudget, batchSize... on cosmos sink, much RU (request unit) is used to find related document across partitions, then to have insert/upsert/delete/update operation on it. At the same time, high RU usage leads to cosmos server side throttling which make request takes more time to get response than normal.

## Resolution

When ADF dataflow customers try to move/transfer data from different stores (for example, azure blob, SQL DW...) into cosmosDB, the backend will use spark cluster of Azure Databricks to finish the execution. The relationship between input data (dataframe) and spark cluster cores is: each partition of dataframe is 1 task which will running on 1 core, the spark cluster core number determines how many dataframe partitions can be written into cosmosDB at the same time in parallel.

- If dataframe's partition number == spark cluster core number, both compute resource (CPU, Memory, IO...) and cosmos throughput are efficiently used, so the data transfer can be finished in short time.
- If dataframe's partition number < spark cluster core number, the compute resource (CPU, Memory, IO...) are not efficiently used because some cores are idle with no task execution.
- If dataframe's partition number > spark cluster core number, the compute resource (CPU, Memory, IO...) are efficiently used, but task (per partition) needs to be executed sequentially, whole data transfer needs more time to finish execution.

In order to get better performance in writing data into cosmosDB, the customer can use following rules to tune dataflow activity settings:

Sink Settings Mapping Optimize Inspect Data preview ●

---

Update method

☒ Allow insert  
☐ Allow delete  
☐ Allow upsert  
☐ Allow update

Collection action

☒ None ☐ Recreate collection

Batch size ⓘ

Partition key ⓘ

Throughput ⓘ

Write throughput budget ⓘ

**Batch size:** An integer that represents how many objects are being written to Cosmos DB collection in each batch. Usually, starting with the default batch size is sufficient. To further tune this value, note:

- Cosmos DB limits single request's size to 2MB. The formula is "Request Size = Single Document Size \* Batch Size". If you hit error saying "Request size is too large", reduce the batch size value.
- The larger the batch size, the better throughput ADF can achieve, while make sure you allocate enough RUs to empower your workload.

**Partition key:** Enter a string that represents the partition key for your collection

**Throughput:** Set a higher throughput setting here to allow documents to write faster to CosmosDB. Keep in mind the higher RU costs based upon a high throughput setting. Minimum is 400.

**Write throughput budget:** An integer string that represents the total RUs that you want to allocate in this cosmos write operation, out of the total throughput allocated to the collection, the large value can get better performance.

Set partitioning with "Hash" under cosmos sink "Optimize".

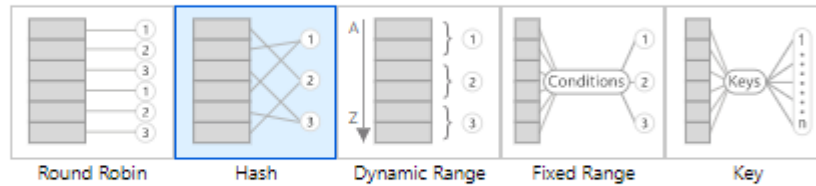
For the value of "Column values to hash on", if input data (dataframe) has the same column like cosmosDB partition key (for example, "city"), you can directly use it; otherwise, you can use column mapping (for example, mapping dataframe column "X" with cosmosDB partition key column "city") or use "select" transformation to rename the "X" as "city" before cosmos sink, then set its value as "city" here.

Sink Settings Mapping **Optimize** Inspect Data preview

Partition option \*

☐ Use current partitioning
 ☐ Single partition
 ☒ Set Partitioning

Partition type \*



Number of partitions \*

32

Column values to hash on \*

Column

city

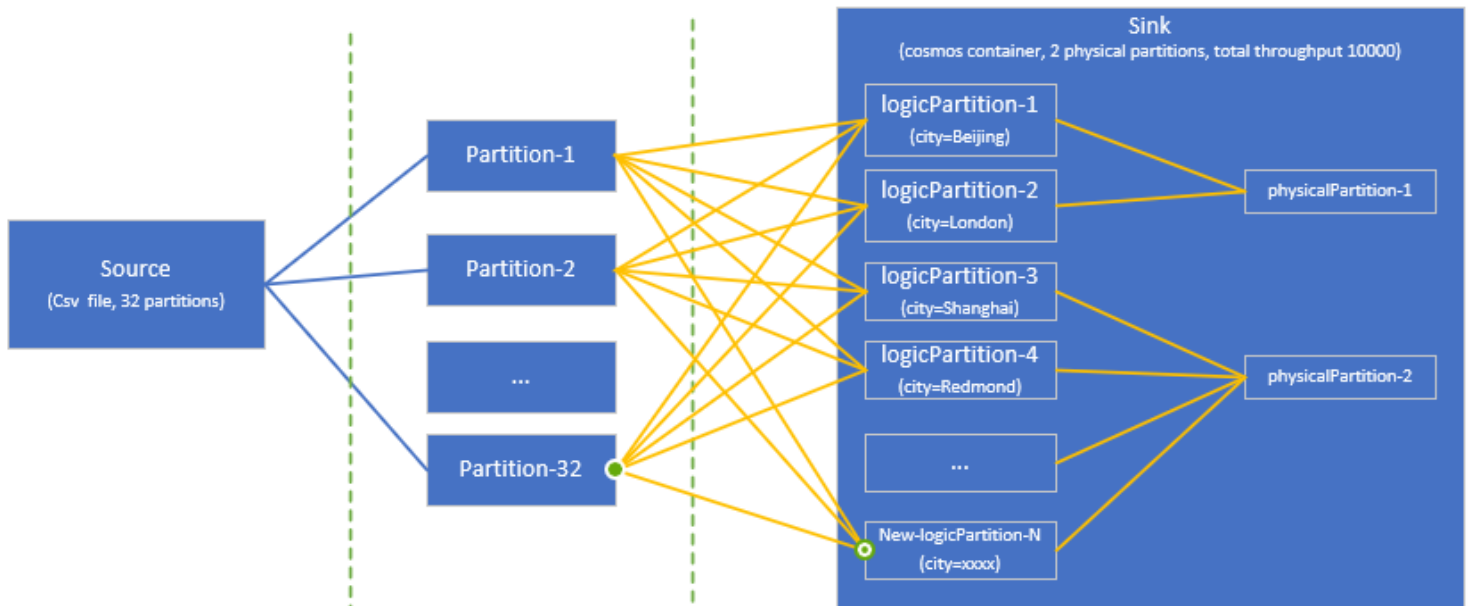
abc

+

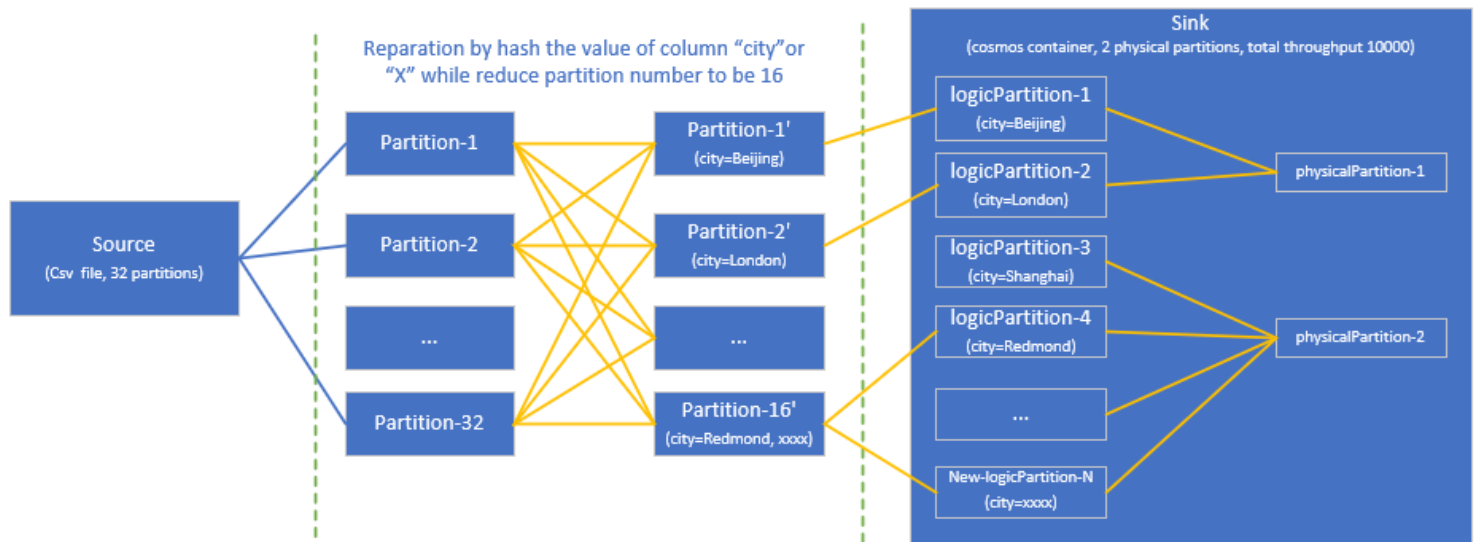
-

For the value of "Number of partitions", you can set it based on:

1. If dataframe partition number (for example: 32) == spark cluster core number (for example, 32), just set the value as 32.

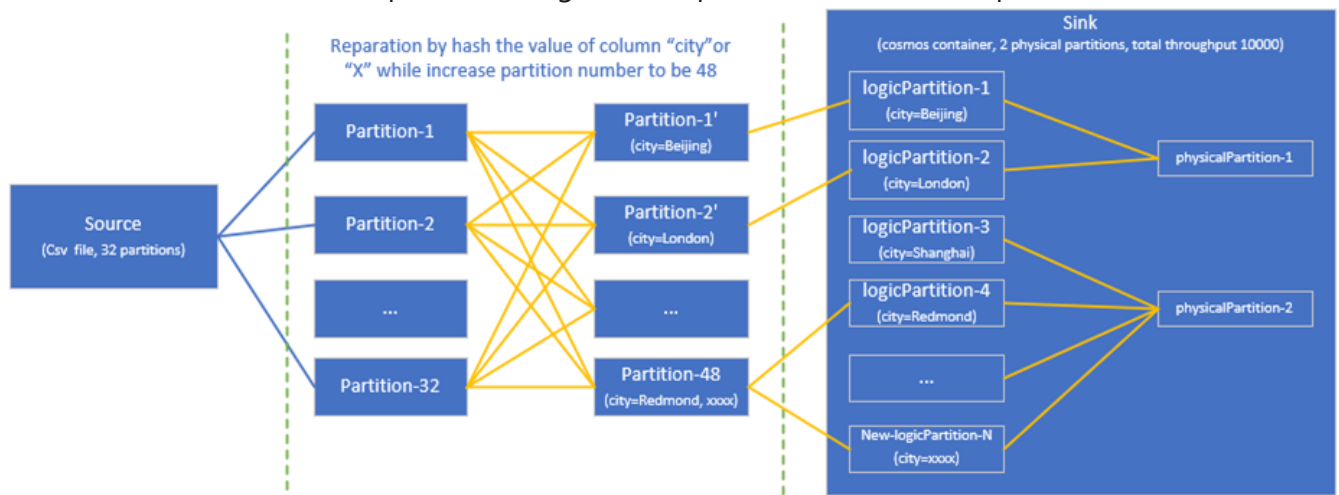


2. If dataframe partition number (for example: 32) > spark cluster core number (for example, 16), you could have a try to repartition dataframe to make its partition number equals core number.



**Note:** Theoretically, if dataframe partition number is significantly higher than spark core number, for example 300 vs 16 or so. Like a threshold of spark cluster core number \*5 or \*10, we suggest to have repartition.

3. If dataframe partition number (for example: 32) < spark cluster core number (for example, 48):
  - If your provisioned cosmosDB collection throughput is not the bottleneck, you can repartition dataframe to have at least 48 partitions to get better performance, for example:



- If your provisioned cosmosDB collection throughput can even be reached from the 32 cores then any repartitioning wouldn't help – because the bottleneck is the provisioned throughput, so just keep as is.




The example of dataflow cosmosDB sink settings:

```

source(output (
    id as string,
    code as string,
    city as string
),
allowSchemaDrift: true,
validateSchema: false,
ignoreNoFilesFound: false) ~> source1
source1 derive(id = concat(id, '-', code)) ~> DerivedColumn1
DerivedColumn1 select(mapColumn(
    id,
    city
),
skipDuplicateMapInputs: true,
skipDuplicateMapOutputs: true) ~> Select1
Select1 sink(allowSchemaDrift: true,
validateSchema: false,
deletable:false,
insertable:true,
updateable:false,
upsertable:false,
format: 'document',
partitionKey: ['/city'],
throughput: 10000,
totalWriteThroughputBudget: 9000,
partitionBy('hash', 32,
    city
),
skipDuplicateMapInputs: true,
skipDuplicateMapOutputs: true) ~> sink1

```

## Reference

1. Partitioning and horizontal scaling in Azure Cosmos DB <https://docs.microsoft.com/en-us/azure/cosmos-db/partitioning-overview> 
2. Request Units as a throughput and performance currency in Azure Cosmos DB <https://docs.microsoft.com/en-us/azure/cosmos-db/request-units> 
3. Provision throughput on Azure Cosmos containers and databases <https://docs.microsoft.com/en-us/azure/cosmos-db/set-throughput> 

## Additional Information

- Icm Reference: N/A
- Author: Zhangyi Yu
- Reviewer: Zhangyi Yu; Shawn Xiao
- Keywords:

How good have you found this content?

