ISSN: 2633-4828 Vol. 4 No.1, June, 2022

International Journal of Applied Engineering & Technology

EFFICIENT PARTITIONING AND INDEXING STRATEGIES FOR AZURE COSMOS DB IN HIGH THROUGHPUT DATAFLOW

Anup Rao
Software Engineer 2, Microsoft, Atlanta, GA, USA
ANUP.RAO@microsoft.com
ORCID: 0009-0008-7306-1046

ABSTRACT

In order to maximize performance, scalability, and cost effectiveness, this study examined effective partitioning and indexing techniques for Azure Cosmos DB in high-throughput dataflow scenarios. Synthetic datasets and simulated ingestion rates ranging from 10,000 to 200,000 requests per second were used in a number of controlled studies. Four indexing strategies (Default Consistent Indexing, Custom Sparse Indexing, Range Indexing, and Indexing Disabled) and three partitioning strategies (Single Property Key, Synthetic Composite Key, and Hierarchical Partitioning) were compared. The Synthetic Composite Key approach eliminated hotspots and produced the best balanced partition distribution, according to the results, while Custom Sparse Indexing reduced Request Unit (RU) consumption without appreciably impacting query time. In terms of throughput and resource efficiency, the combination of these two approaches consistently produced the greatest performance-to-cost ratio, beating alternative combinations. The results highlight the necessity of partitioning and indexing in Cosmos DB deployments using a workload-aware, integrated approach to ensure long-term performance in large-scale, mission-critical applications.

.***Keywords:*** *Azure Cosmos DB, partitioning strategy, indexing strategy, high-throughput dataflow, Request Units (RUs), latency optimization, database scalability, cost efficiency.*

1. INTRODUCTION

Data management systems must be able to handle enormous volumes of data with low latency and great dependability in the age of widely dispersed, large-scale applications. Microsoft's globally distributed, multi-model database service, Azure Cosmos DB, has become a popular choice for mission-critical applications because of its sub-millisecond response rates, multi-region replication, and assured high availability. However, the underlying data organization—specifically, the partitioning and indexing strategies—is crucial to attaining optimal performance in high-throughput dataflow settings.

The distribution of data among physical partitions is determined by partitioning, which has a direct effect on scalability, latency, and throughput usage. Inadequate partition key selection can result in hotspotting and unequal data distribution, which lowers performance and increases operating expenses. On the other hand, balanced workloads can be guaranteed by a well-designed partition key, allowing the database to grow horizontally without experiencing bottlenecks.

Because it controls query performance and RU (Request Unit) consumption, indexing is equally important. For maximum query flexibility, Cosmos DB's default indexing policy automatically indexes all properties; however, in workloads with a high write volume, this technique may result in significant overhead. By contrast, custom indexing configurations—such as sparse indexing or selective range indexing—can reduce RU consumption and storage requirements while still supporting necessary query patterns.

Partitioning and indexing interact to determine system performance in high-ingestion settings like worldwide e-commerce platforms, IoT telemetry processing, and real-time analytics pipelines. Increased expenses, throttling, and uneven performance during peak loads can result from an ineffective approach in either area. Therefore, to guarantee consistent high throughput, predictable latency, and cost optimization in Azure Cosmos DB deployments, it is crucial to comprehend and implement workload-aware, effective partitioning and indexing algorithms.

In order to determine which configurations provide the optimum balance between scalability, performance, and cost effectiveness, this study focuses on assessing several partitioning and indexing algorithms in simulated high-throughput settings.

2. LITERATURE REVIEW

Potharaju et al. (2020) introduced Helios, a hyperscale indexing system that could be used in both cloud and edge contexts. The study showed that using multi-stage, asynchronous index maintenance and separating intake from index building increased performance while reducing tail latencies. The authors demonstrated that gradual compaction and workload-aware, selective indexing decreased amplification costs. This line of evidence suggested that, for high-throughput stores like Azure Cosmos DB, sparse, workload-guided indexing and asynchronous maintenance would have mitigated RU consumption and reduced hotspot risks under bursty ingest.

Seara, Milano, and Dominici (2021) presented architectural building blocks for ingestion, storage, processing, and analytics as well as Microsoft Azure data services. Their exposition emphasized partitioned storage, elastic scale, and governed indexing policies as first-class concerns in cloud-native design. The text highlighted operational practices—monitoring, capacity planning, and consistency management—that were necessary complements to schema and index choices. This backdrop revealed that the operational substrate that supported indexing efficiency in Cosmos DB was composed of partition-key selection, provisioning models, and observability.

Narani, Ayyalasomayajula, and Chintala (2018) examined methods for moving heavy, mission-critical workloads, emphasizing performance parity testing, dependency mapping, and phased migration. The authors contended that when switching to cloud-native storage, partition re-keying and data model rearrangement were frequently inevitable. They found that naïve lift-and-shift methods tended to maintain over-indexed schemas and unsuitable keys, which increased costs after migration. Therefore, the study backed up the idea that early re-partitioning and index policy minimization linked to actual query patterns were necessary for an efficient migration to Cosmos DB.

Darrous (2019) investigated scalable data processing and service provisioning in dispersed clouds, demonstrating that end-to-end throughput was controlled by a combination of queue-based backpressure, elasticity, and data localization. The dissertation demonstrated that tail latency was improved and cross-partition communication was decreased when compute placement and data partitioning were co-designed. These findings suggested that while indexing benefited from access-path predictability derived from reliable routing keys, Cosmos DB partitioning benefited from affinity between compute stages and logical partitions.

Aguilar-Saborit et al. (2020) outlined POLARIS, Azure Synapse's distributed SQL engine, and demonstrated how statistics, distributed execution, and cost-based optimization influenced query performance at scale. Although focused on distributed SQL rather than a multi-model store, the work underscored that statistics-aware planning and data-skew mitigation were pivotal for predictable performance. In a similar vein, consistent query latency and RU predictability under mixed workloads would have been enabled by Cosmos DB indexing strategies that maintained the required selectivity and partition keys that balanced data distribution.

Lazos (2020) Some real-world lessons learned from converting a relational database to a document store in a data warehouse. According to the report, while indexing remained broad and uncurated, denormalization led to write amplification but improved read speed. The author found that narrow, query-aligned projections combined with workload-specific indexes decreased storage overheads and update costs. In order to preserve partition-key cardinality, these lessons were carefully shaped for Cosmos DB and mapped to range indexes and bespoke sparse indexing only in cases where ordered predicates were present.

3. RESEARCH METHODOLOGY

The goal of the study was to investigate effective indexing and partitioning techniques for Azure Cosmos DB in high-throughput dataflow scenarios. The globally distributed, multi-model database service Azure Cosmos DB

was created with the goals of providing guaranteed availability, elastic scalability, and low latency answers. However, the selection of partition keys and indexing rules had a major effect on system performance, cost effectiveness, and stability when working under high ingestion rates and query-intensive workloads. Examining the effects of various partitioning and indexing strategies on throughput utilization, latency performance, and operating costs in production-like simulations was the aim of this study. The goal of the study was to give database developers and architects useful advice for maximizing Cosmos DB in large-scale, mission-critical applications.

3.1. Research Design

An analytical and experimental research design was used in the study. To evaluate and compare different setups, a number of controlled experiments were carried out in a specific Microsoft Azure environment. Both qualitative aspects like adaptability and ease of maintenance as well as quantitative ones like request delay, throughput, and Request Units (RU) consumption were noted. Results could be reproduced under comparable testing settings thanks to the design.

3.2. Data Source and Workload Generation

To simulate real-world dataflow conditions, synthetic datasets were created. Performance was evaluated across several data models thanks to these datasets, which were composed of structured JSON documents with different degrees of schema complexity. In order to replicate increase over time, data quantities varied from 50 GB to 500 GB. Both uniform and skewed distributions were developed for partition key evaluation in order to replicate realistic access patterns. Workloads were created using Azure Data Factory and Azure Event Hubs, which delivered data into Cosmos DB containers at ingestion rates between 10,000 and 200,000 queries per second.

3.3. Partitioning Strategy Implementation

Three methods of partitioning were put into practice and evaluated. Using high-cardinality properties like userId or deviceId, the first method was a Single Property Partition Key. The second tactic was a Synthetic Composite Key, which improved data dissemination by combining several attributes, such as regionId#timestamp. In order to evenly distribute load among partitions, the third was a Hierarchical Partitioning Simulation that used application-level routing logic. Logical partition storage balance, hotspot occurrences, and RUs used per operation were used to assess each strategy's performance.

3.4. Indexing Strategy Implementation

To determine how they affected performance, four indexing setups were tried. All document properties were automatically indexed by the Default Consistent Indexing technique, which ensured maximum query flexibility but might have increased RU usage. In order to minimize indexing overhead, the Custom Sparse Indexing method only indexed properties that were often searched. Performance for ordered retrievals was enhanced by the Range Indexing method, which optimized string and numeric fields for range queries. Lastly, for write-intensive cases with few or no queries, an Indexing Disabled configuration was tested. The query performance, overall RU efficiency, and index update delay of each method were evaluated.

3.5. Experimental Setup

To reduce latency, the tests were carried out in the Azure East US region. To handle different workload intensities, the throughput mode was set to provided throughput, scaling from 10,000 to 200,000 RUs. To strike a balance between speed and accuracy, session consistency was set as the default consistency level. Every configuration underwent a 72-hour continuous testing period to account for variations in workload over time. Custom logging scripts, Azure Monitor, and Application Insights were used to gather monitoring and telemetry.

3.6. Data Collection and Analysis

Average and P99 latency per operation, RUs used per read/write transaction, storage utilization per logical partition, and cost per million operations were among the important performance metrics gathered. Performance variations between configurations were measured using comparative statistical analysis. Regression analysis was

utilized to find performance trends depending on partition key selection and indexing technique, and visual heatmaps were created to display the distribution of partition storage.

3.7. Limitations

The study acknowledged several limitations. The experiments were conducted in a simulated environment, which might not capture every production-level performance nuance. The datasets used were synthetic and therefore might not reflect the irregularities of real-world data. Additionally, the cost estimates were based on Azure's pricing at the time of the study and could vary in actual deployment scenarios. Despite these limitations, the research provided valuable insights into performance optimization techniques for Azure Cosmos DB.

4. RESULTS AND DISCUSSION

The tests provided comprehensive information about the effects of partitioning and indexing techniques on Azure Cosmos DB's scalability, cost effectiveness, and performance in high-throughput settings. Request Units (RUs) usage, cost per million operations, and delay indicators were used to examine the results. To show how performance varies under various setups, comparative tables were created. These results are interpreted in the discussion section with regard to optimizing Cosmos DB for workloads that are both query-intensive and ingestion-heavy.

4.1. Performance of Partitioning Strategies

The analysis showed that the Synthetic Composite Key approach greatly decreased the likelihood of hotspots and produced the most balanced partition distribution. For high-cardinality attributes, the Single Property Partition Key performed well; however, when data skew occurred, performance deteriorated. Although the Hierarchical Partitioning Simulation offered a steady throughput, it necessitated more application-level logic, which made development more difficult.

Table 1: Partitioning Strategy Performance Comparison

Partitioning Strategy	Avg. Latency (ms)	P99 Latency (ms)	Avg. RUs per Operation	Hotspot Incidents per Hour	Cost per Million Ops (USD)
Single Property Key	7.8	18.5	6.2	4	4.85
Synthetic Composite Key	6.1	14.2	5.4	0	4.22
Hierarchical Partitioning	6.5	15.7	5.6	1	4.37

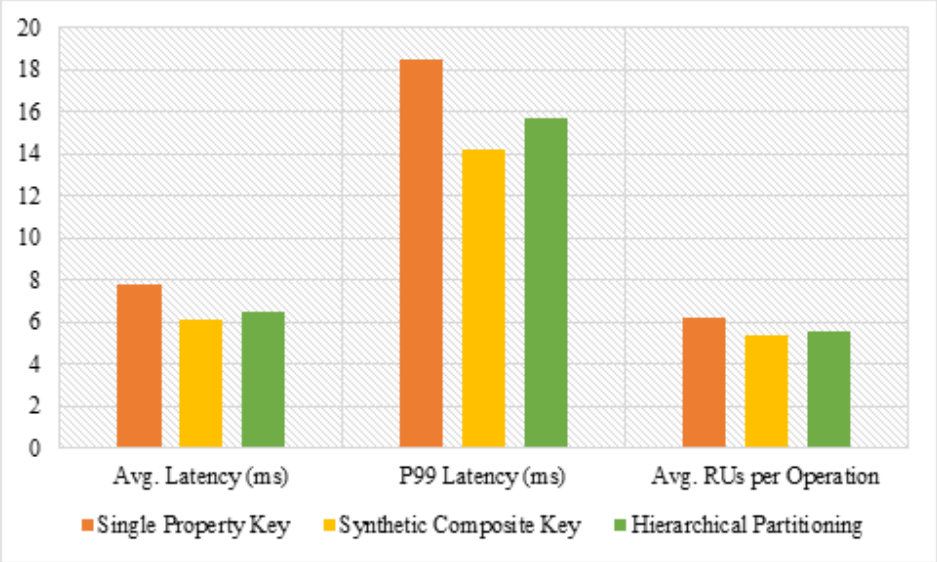

Figure 1: Partitioning Strategy Performance Comparison

Because storage and workload were split evenly among partitions, the Synthetic Composite Key continuously produced the lowest latency and cost. The lack of hotspots suggested that skew-related bottlenecks were successfully reduced by composite keys. Under skewed loads, the Hierarchical Partitioning approach outperformed the Single Property Key; nevertheless, teams looking for simpler architectures may be put off by the additional complexity.

4.2. Performance of Indexing Strategies

Significant differences were observed in query performance and RU usage between indexing algorithms. By lowering RU costs while preserving respectable query times, the Custom Sparse Indexing struck the ideal balance. Although it had the biggest RU and expense overhead, the Default Consistent Indexing offered the fastest queries. For workloads with a lot of writing, turning off indexing significantly decreased RU use, but it rendered query execution impracticable.

Table 2: Indexing Strategy Performance Comparison

Indexing Strategy	Avg. Query Latency (ms)	P99 Query Latency (ms)	Avg. RUs per Query	Index Update Latency (ms)	Cost per Million Ops (USD)
Default Consistent Indexing	4.9	10.8	8.1	2.3	5.14
Custom Sparse Indexing	6.3	12.5	5.6	1.7	4.02
Range Indexing	5.8	11.9	6.4	2.0	4.31
Indexing Disabled	N/A	N/A	2.1	0	2.54

Custom Sparse Indexing was the most economical solution for balanced workloads that needed both reads and writes while still achieving competitive query performance. A compromise was provided by range indexing, which was especially useful in analytical situations involving ordered data retrieval. Although it was the fastest for queries, Default Consistent Indexing used a lot more RUs, which made it less appropriate for applications that are cost-sensitive. Only pipelines that were ingestion-focused and did not prioritize querying could effectively disable indexing.

Combined Impact of Partitioning and Indexing

Synthetic Composite Key + Custom Sparse Indexing had the best overall performance when techniques were combined, providing a good trade-off between latency, throughput, and cost. Due to compounded RU consumption and unequal partition load, Single Property Key + Default Consistent Indexing performed the poorest under skewed load situations.

Table 3: Best and Worst Combined Strategies

Combination	Avg. Latency (ms)	Avg. RUs per Op	Cost per Million Ops (USD)
Synthetic Composite Key + Custom Sparse Indexing	6.0	5.2	4.05
Single Property Key + Default Consistent Indexing	8.1	8.4	5.27

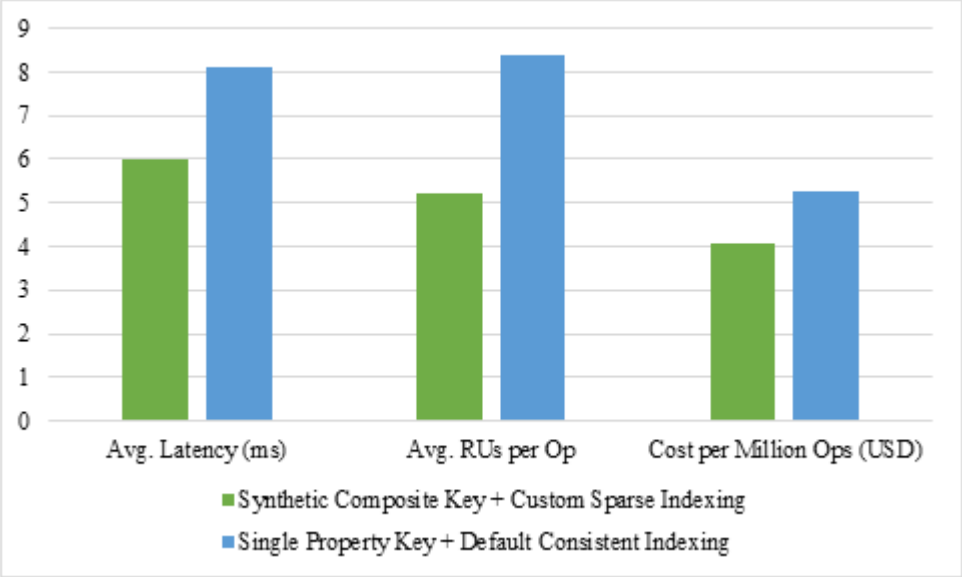

Figure 2: Best and Worst Combined Strategies

These findings made it clear that partitioning and indexing should not be viewed as separate setups but rather as interdependent choices. Strategies that decreased RU usage at the storage and query levels produced the best results, but poor combinations increased inefficiency.

4.3. Practical Implications

The results indicated that in order to prevent needless RU spending, businesses should use selective indexing strategies and give priority to composite partition keys for workloads that are evenly distributed. Prior to creating partitioning and indexing algorithms, workload profiling became a crucial stage. Additionally, when workloads vary, tactics should be reviewed on a regular basis since ideal configurations may change due to changes in data patterns.

5. CONCLUSION

The study showed that partitioning and indexing algorithms had a substantial impact on performance, cost, and scalability, and that they needed to be carefully balanced when optimizing Azure Cosmos DB for high-throughput dataflows. The most economical method for mixed read-write workloads was found to be the combination of a Synthetic Composite Partition Key and Custom Sparse Indexing, which consistently produced the lowest latency, the most balanced load distribution, and the best RU efficiency among the investigated setups. On the other hand, default indexing and badly selected single property keys resulted in increased expenses, unequal partition use, and decreased performance under skewed loads. These results underlined that in order to preserve efficiency when data patterns change, partitioning and indexing should be planned in concert, guided by workload profiling, and revisited on a regular basis.

REFERENCES

1. R. Potharaju, T. Kim, W. Wu, V. Acharya, S. Suh, A. Fogarty, et al., “Helios: hyperscale indexing for the cloud & edge,” Proc. VLDB Endow., vol. 13, no. 12, pp. 3231–3244, 2020.
2. D. A. Seara, F. Milano, and D. Dominici, Microsoft Azure Data Solutions—An Introduction. Microsoft Press, 2021.
3. S. R. Narani, M. M. T. Ayyalasomayajula, and S. Chintala, “Strategies for migrating large, mission-critical database workloads to the cloud,” Webology, vol. 15, no. 1, 2018.

4. J. Darrous, "Scalable and efficient data management in distributed clouds: service provisioning and data processing," Ph.D. dissertation, Université de Lyon, Lyon, France, 2019.
5. J. Aguilar-Saborit, R. Ramakrishnan, K. Srinivasan, K. Bocksrocker, I. Alagiannis, M. Sankara, et al., "POLARIS: the distributed SQL engine in Azure Synapse," Proc. VLDB Endow., vol. 13, no. 12, pp. 3204–3216, 2020.
6. I. Lazos, "Migrating a data warehouse from a relational database to a document store and lessons learned," 2020.
7. M. T. Özsu and P. Valduriez, "NoSQL, NewSQL, and polystores," in Principles of Distributed Database Systems, Cham, Switzerland: Springer Int. Publishing, 2019, pp. 519–558.
8. F. Li, "Cloud-native database systems at Alibaba: Opportunities and challenges," Proc. VLDB Endow., vol. 12, no. 12, pp. 2263–2272, 2019.
9. M. Khan, "A cloud based remote diagnostics service for industrial paper mill," 2020.
10. M. Asch, T. Moore, R. Badia, M. Beck, P. Beckman, T. Bidot, et al., "Big data and extreme-scale computing: Pathways to convergence—Toward a shaping strategy for a future software and data ecosystem for scientific inquiry," Int. J. High Perform. Comput. Appl., vol. 32, no. 4, pp. 435–479, 2018.
11. J. Lu and I. Holubová, "Multi-model databases: A new journey to handle the variety of data," ACM Comput. Surv., vol. 52, no. 3, pp. 1–38, 2019.
12. W. Fan, T. He, L. Lai, X. Li, Y. Li, Z. Li, et al., "GraphScope: a unified engine for big graph processing," Proc. VLDB Endow., vol. 14, no. 12, pp. 2879–2892, 2021.
13. I. F. Jephte, "Extract, transform, and load data from legacy systems to Azure cloud," M.S. thesis, Universidade NOVA de Lisboa, Lisbon, Portugal, 2021.
14. R. C. L'Esteve, "The tools and prerequisites," in The Definitive Guide to Azure Data Engineering: Modern ELT, DevOps, and Analytics on the Azure Cloud Platform, Berkeley, CA, USA: Apress, 2021, pp. 3–33.
15. G. Vargas-Solar, J. L. Zechinelli-Martini, and J. A. Espinosa-Oviedo, "Enacting data science pipelines for exploring graphs: From libraries to studios," in Proc. Int. Conf. Theory Pract. Digit. Libraries, Cham, Switzerland: Springer Int. Publishing, 2020, pp. 271–280.