see Integrating Amazon DynamoDB and Amazon ElastiCache by using read-through caching in AWS Prescriptive Guidance.

Troubleshooting throttling in Amazon DynamoDB

DynamoDB implements throttling for two primary purposes: maintaining overall service performance and cost control. Throttling either serves as an intentional safeguard that prevents performance degradation when consumption rates exceed capacity or as a cost control mechanism when you reach maximum throughput or service quota limits. When throttling occurs, DynamoDB returns specific exceptions with detailed information about why the request was throttled and which resource was impacted. Each reason for throttling corresponds to specific CloudWatch metrics that provide additional insights into the frequency and patterns of throttling events.

The following diagram illustrates the four primary scenarios where DynamoDB implements protective throttling:

1. **Key range throughput exceeded (in both modes)**:

   Consumption directed at specific partitions exceeds the internal partition-level throughput limits.

2. **Provisioned throughput exceeded (in provisioned mode)**:

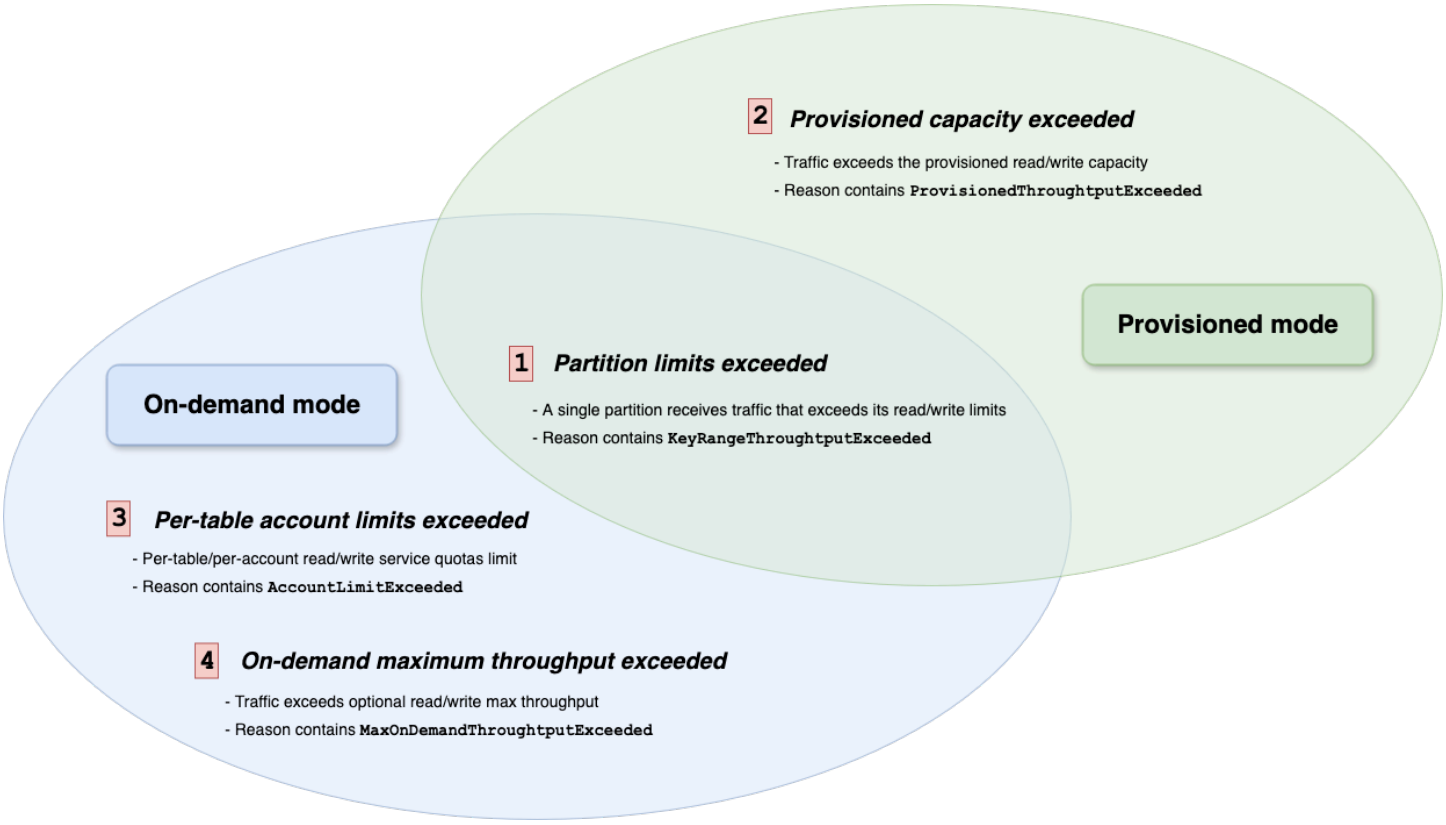
   Consumption exceeds the provisioned capacity units (read or write) configured for a table or a global secondary index (GSI).

3. **Account-level service quotas exceeded (in on-demand mode)**:

   Consumption causes a table or GSI to exceed the per-table account-level service quotas for read/write throughput in the current AWS Region. These quotas serve as backstop safeguards and can be increased.

4. **On-demand maximum throughput exceeded (in on-demand mode)**:

   Consumption exceeds the configured maximum throughput limits set for a table or GSI. These are limits you configure specifically for cost control purposes.



This guide is organized to help you understand and work with throttling in DynamoDB. First, we help you identify the specific type of throttling affecting your workload through a diagnostic framework.

Then, the resolution guide section offers specific guidance for each throttling scenario, including CloudWatch metrics to monitor for detection and analysis, and recommended steps for optimization. By following this structured approach, you can better diagnose the root cause of throttling and implement the appropriate solution to ensure your DynamoDB tables operate efficiently.

To get started, follow the section called “Diagnosing throttling” to learn how to identify which type of throttling is affecting your workload and implement the recommended resolution strategy.

Topics

- Diagnosing throttling
- DynamoDB throttling resolution guide
- Understanding Global Secondary Index (GSI) write throttling and back pressure in DynamoDB
- CloudWatch throttling metrics

Diagnosing throttling

When your application experiences throttling, DynamoDB provides detailed exception information and targeted CloudWatch metrics to help you diagnose these events.

This section presents a systematic approach to understanding throttling events in your DynamoDB applications. It shows how to interpret throttling exceptions, correlate them with CloudWatch metrics for deeper insights, and understand what changes would reduce throttling in your DynamoDB applications.

Understanding throttling exceptions

When DynamoDB throttles a request, it returns specific exceptions with detailed diagnostic information. For example, in Java, these include `ProvisionedThroughputExceededException`, `RequestLimitExceeded`, or `ThrottlingException`.

Each exception includes `ThrottlingReasons`, a collection of individual `ThrottlingReason` containing two key fields to help you identify and understand the throttling:

- **A reason** - A concatenated field following the format `<ResourceType><OperationType><LimitType>`
- **A resource ARN** - The Amazon Resource Name (ARN) of the affected table or index

The reason field follows a consistent patern that helps you understand exactly what's happening:

- **ResourceType** (What is being throttled): `Table` or `Index`
- **OperationType** (What kind of operation): `Read` or `Write`
- **LimitType** (Why the throttling occurred):
  - `KeyRangeThroughputExceeded`: This occurs when a specific partition backing your table or index has consumed read or write capacity exceeding the internal per-partition throughput limits.
  - `ProvisionedThroughputExceeded`: This happens on a provisioned table or global secondary index when the read or write consumption rate has exceeded the provisioned amount.
  - `AccountLimitExceeded`: This happens on an on-demand table or index when the read or write consumption rate has exceeded the maximum consumption rate for a table and its indexes as set at the account level. You can request a raise in this quota.

- `MaxOnDemandThroughputExceeded`: This happens on an on-demand table or index when the read or write consumption rate has exceeded the user-provided maximum consumption rate configured for the table or index. You can raise this value yourself to any value up to the account limit or set to -1 to indicate no user-provided limit.

The resource ARN identifies exactly which table or index is being throttled:

- For tables: `arn:aws:dynamodb:[region]:[account-id]:table/[table-name]`
- For indexes: `arn:aws:dynamodb:[region]:[account-id]:table/[table-name]/index/[index-name]`

Examples of complete throttling reasons:

- `TableReadProvisionedThroughputExceeded`
- `IndexWriteAccountLimitExceeded`

This helps identify exactly what resource is being throttled, what type of operation caused it, and why the throttling occurred.

Example exceptions

Example 1: Provisioned capacity exceeded on a GSI

```
{
    "ThrottlingReasons": [
        {
            "reason": "IndexWriteProvisionedThroughputExceeded",
            "resource": "arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders/
index/OrderDateIndex"
        }
    ],
    "awsErrorDetails": {
        "errorCode": "ProvisionedThroughputExceeded",
        "errorMessage": "The level of configured provisioned throughput for the index
 was exceeded",
        "serviceName": "DynamoDB",
        "sdkHttpResponse": {
            "statusText": "Bad Request",
            "statusCode": 400
```

```
        }
    }
}
```

In this example, the application receives a `ProvisionedThroughputExceededException` with the reason `IndexWriteProvisionedThroughputExceeded`. Writes to the `OrderDateIndex` are being throttled because the write consumption has exceeded the GSI's configured provisioned write capacity.

Example 2: On-demand maximum throughput exceeded

```
{
    "ThrottlingReasons": [
        {
            "reason": "TableReadMaxOnDemandThroughputExceeded",
            "resource": "arn:aws:dynamodb:us-east-1:123456789012:table/UserSessions"
        }
    ],
    "awsErrorDetails": {
        "errorMessage": "Throughput exceeds the maximum OnDemandThroughput configured
 on table or index",
        "errorCode": "ThrottlingException",
        "serviceName": "DynamoDB",
        "sdkHttpResponse": {
            "statusText": "Bad Request",
            "statusCode": 400
        }
    }
}
```

In this example, reads from the `UserSessions` table are being throttled because they exceed the configured maximum on-demand throughput limit on the table.

DynamoDB throttling diagnosis framework

When your application encounters throttling, follow these steps to diagnose and resolve the issue.

Step 1 - Analyze the `ThrottlingReason` details

1. Check the **reason** field to identify the specific reason for throttling. The reason details the type of resource throttled (table or index), the type of operation causing the throttling event (read or write), and the limit type that was exceeded (partition, provisioned throughput, account limit).

2. Check the **resourceArn** field to identify which resource (table or GSI) is being throttled.
3. Use this combined information to understand the full context of the throttling issue.

   For example, consider this scenario where you receive the following exception `ProvisionedThroughputExceededException` with a throttling reason `TableWriteKeyRangeThroughputExceeded`. The impacted resourceARN is `arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders`.

   This combination informs you that write operations to your `CustomerOrders` table are being throttled. The throttling is occurring at the partition level (not the table level, which would show as `TableWriteProvisionedThroughputExceeded`). The root cause is that you've exceeded the maximum throughput capacity for a specific partition key value or range, indicating a hot partition issue.

   Understanding this relationship between the exception elements helps you implement the appropriate mitigation strategy - in this case, addressing the hot partition rather than increasing the table's overall provisioned capacity.

Step 2 - Identify and analyze the related CloudWatch metrics

1. **Identify your metrics:** Each throttling reason in DynamoDB directly corresponds to specific CloudWatch metrics that you can monitor to track and analyze throttling events. You can systematically derive the appropriate CloudWatch metric names from the throttling reason.
2. Match your throttling reason to the corresponding CloudWatch metrics using this reference table:

   Complete throttling reasons and CloudWatch metrics reference

Category	Throttling reason	Primary CloudWatch metrics
Provisioned capacity exceeded	TableReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents
	TableWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents

Category	Throttling reason	Primary CloudWatch metrics
	IndexReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents (GSI)
	IndexWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents (GSI)
Partition limits exceeded	TableReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents
	TableWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents
	IndexReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents (GSI)
	IndexWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents (GSI)
On-demand maximum exceeded	TableReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents
	TableWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents
	IndexReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents (GSI)

Category	Throttling reason	Primary CloudWatch metrics
	IndexWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents (GSI)
Account limits exceeded	TableReadAccountLimitExceeded	ReadAccountLimitThrottleEvents
	TableWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents
	IndexReadAccountLimitExceeded	ReadAccountLimitThrottleEvents (GSIs)
	IndexWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents (GSIs)

For example, if you received `IndexWriteProvisionedThroughputExceeded`, at a minimum, you should monitor the `WriteProvisionedThroughputThrottleEvents` CloudWatch metric for the specific index identified in the `ResourceArn`.

3. Monitor these metrics in CloudWatch to understand the frequency and timing of the throttling events, differentiate between read and write throttling, identify time patterns when throttling increases, and track your capacity utilization trends.

   DynamoDB publishes detailed metrics for each table and global secondary index. The metrics (`ReadThrottleEvents`, `WriteThrottleEvents`, and `ThrottledRequests`) aggregate all throttling events across your table and its indexes.

Step 3 - Identify your throttled keys and high access rates using CloudWatch Contributor Insights (for partition-related throttling)

If you identified partition-related issues in Step 1 (such as `KeyRangeThroughputExceeded` errors), CloudWatch Contributor Insights for DynamoDB can help you diagnose which specific keys are driving your traffic and experiencing throttling events in your table or index.

1. Enable CloudWatch Contributor Insights for your throttled table or index based on your `ResourceARN`.

   You can choose the *Throttled keys* mode to focus exclusively on the most throttled keys. This mode is ideal for continuous monitoring as it only processes events when throttling occurs. Alternatively, the *Accessed and throttled keys* mode helps you look for patterns in your most accessed keys.

2. Analyze the reports to identify problematic patterns. Look for keys with disproportionately high access or throttling rates, correlate throttling and traffic patterns. You can create integrated dashboards combining Contributor Insights graphs and DynamoDB CloudWatch metrics.

For detailed information about enabling and using CloudWatch Contributor Insights, see Using CloudWatch Contributor Insights for DynamoDB.

Step 4 - Determine the appropriate solution

After diagnosing the specific cause of throttling, implement recommended solution based on your specific context. The appropriate solution depends on multiple factors, including your throttling scenario, table's capacity mode, table and key design decisions, access patterns and query efficiency, global and secondary index configuration, and overall system architecture and integration points.

For detailed solutions to address your specific throttling scenarios, see the the section called "Resolution guide" section. This resource provides targeted remediation strategies customized to your particular throttling reason and capacity mode configuration.

Step 5 - Monitor your progress

1. Track your CloudWatch metrics that correspond to your throttling scenario.
2. Validate that your mitigation strategies are effective by observing a decrease in throttling events.

DynamoDB throttling resolution guide

This section provides targeted resolution guidance for each specific throttling reason that DynamoDB may return. Each entry includes suggested resolution approaches based on best practices and corresponding CloudWatch metrics to monitor.

DynamoDB implements 16 distinct throttling reasons across four main categories. Use the throttling reasons from your application's exception to navigate directly to the relevant guidance.

Key range throughput exceeded (hot partitions)

These throttling reasons occur when individual partitions exceed their throughput limits, affecting both provisioned and on-demand modes:

- TableReadKeyRangeThroughputExceeded
- TableWriteKeyRangeThroughputExceeded
- IndexReadKeyRangeThroughputExceeded
- IndexWriteKeyRangeThroughputExceeded

Provisioned throughput exceeded

These throttling reasons occur when consumption rates exceed provisioned capacity limits in provisioned mode:

- TableReadProvisionedThroughputExceeded
- TableWriteProvisionedThroughputExceeded
- IndexReadProvisionedThroughputExceeded
- IndexWriteProvisionedThroughputExceeded

Account limits exceeded

These throttling reasons occur when consumption rates exceed account-level throughput quotas in your AWS Region:

- TableReadAccountLimitExceeded
- TableWriteAccountLimitExceeded
- IndexReadAccountLimitExceeded
- IndexWriteAccountLimitExceeded

On-demand maximum throughput exceeded

These throttling reasons occur when consumption rates exceed configured maximum throughput limits in on-demand mode:

- TableReadMaxOnDemandThroughputExceeded
- TableWriteMaxOnDemandThroughputExceeded
- IndexReadMaxOnDemandThroughputExceeded
- IndexWriteMaxOnDemandThroughputExceeded

1- Key range throughput exceeded (hot partitions)

Amazon DynamoDB enforces specific throughput limits at the partition level for both table and global secondary index (GSI). Each partition has a maximum number of read capacity units (RCUs) and write capacity units (WCUs) per second. When partitions receive concentrated traffic that exceeds these limits, they experience throttling while other operations may remain underutilized, creating "hot partitions." DynamoDB's partition-level throttling operates independently for reads and writes - a partition may throttle reads while writes continue normally, or vice versa. This throttling can occur even when your table or GSI has sufficient overall capacity. To learn more about:

- DynamoDB partition limits and effective partition key design addressing hot partition prevention, see Best practices for designing and using partition keys effectively in DynamoDB.
- General partition concepts and data distribution, see Partitions in DynamoDB.
- Additional guidance and real-world scenarios for managing partition keys and throughput, see the section called “Additional resources”.

When individual partitions exceed their throughput limits, DynamoDB returns a `KeyRangeThroughputExceeded` throttling reason type in the throttling exception. The information identifies that a partition is experiencing high traffic and which operation type (read or write) is causing the issue.

Key range throughput exceeded mitigation measures

This section provides resolution guidance for partition-level throttling scenarios. Before using this guide, ensure you have identified the specific throttling reasons from your application's exception handling, and determined the Amazon Resource Name (ARN) of the affected resource.

For information on retrieving throttling reasons and identifying throttled resources, see the section called “Diagnosis framework”.

Before diving into specific throttling scenarios, first, check if the problem resolves automatically:

- DynamoDB often adapts to hot partitions through its automatic split-for-heat mechanism. If you see throttling events that stop after a short period, your table may have already adapted by splitting the hot partition. When partitions split, each new partition handles a smaller section of the keyspace, which can help distribute the load more evenly. In many cases, no further action is needed as DynamoDB has automatically resolved the issue.

  For more information about the *split-for-heat mechanism*, see the section called “Additional resources”.

If the throttling persists, refer to the specific throttling scenarios below for targeted remediation options:

- the section called “TableReadKeyRangeThroughputExceeded”
- the section called “TableWriteKeyRangeThroughputExceeded”
- the section called “IndexReadKeyRangeThroughputExceeded”
- the section called “IndexWriteKeyRangeThroughputExceeded”

TableReadKeyRangeThroughputExceeded

When this occurs

The consumption rate of one or more partitions in your DynamoDB table exceeds the partition's read throughput limit. This throttling occurs regardless of your table's total provisioned capacity and affects both provisioned and on-demand tables. You can monitor the CloudWatch metrics in the section called “Common diagnosis and monitoring” to analyze your throttling event.

Remediation options

Consider these steps to address your throttling events:

For both provisioned and on-demand modes:

- **Pre-warm capacity:** If throttling persists, check if your table is limited by its the section called “Warm throughput” capacity. Use warm throughput or increase read provisioned capacity in

advance for expected traffic increases. Increasing warm throughput improves your table's ability to handle sudden traffic spikes before throttling occurs. Over time, if your actual throughput consistently approaches the warm throughput levels, DynamoDB may split busy partitions based on observed usage patterns.

- **Identify your hot keys:** If the table didn't resolve it automatically and your warm throughput is high or raising it didn't help, you'll need to identify specific hot keys. Use the section called "Identifying hot keys using CloudWatch Contributor Insights" to determine if any particular partition key values are hot. This is a first step to target your mitigation efforts effectively. Note that identification may not always be straightforward, particularly with rolling hot partitions (where different partitions become hot over time) or when throttling is triggered by operations like scans. For these complex scenarios, you may need to analyze your application's access patterns and correlate them with the timing of throttling events.
- **Depending on your use case, consider using eventually consistent reads:** Switch from strongly consistent to eventually consistent reads, which consume half the RCUs and can immediately double your effective read capacity. For best practices on implementing eventually consistent reads to reduce read capacity consumption, see the section called "DynamoDB read consistency".
- **Improve partition key design:** As a long-term solution, consider the section called "Improving partition key design" to distribute access more evenly across partitions. This approach often provides the most comprehensive resolution to hot partition issues by addressing the root cause. However, it requires careful planning as it involves significant migration challenges.

TableWriteKeyRangeThroughputExceeded

When this occurs

The consumption rate of one or more partitions in your DynamoDB table exceeds the partition's write throughput limit. This throttling occurs regardless of your table's total provisioned capacity and affects both provisioned and on-demand tables. You can monitor the CloudWatch metrics in the section called "Common diagnosis and monitoring" to analyze your throttling event.

Remediation options

Consider these steps to address your throttling events:

For both provisioned and on-demand modes:

- **Pre-warm capacity:** If throttling persists, check if your table is limited by its Understanding DynamoDB warm throughput capacity. Use warm throughput or increase write provisioned

capacity in advance for expected traffic increases. Increasing warm throughput improves your table's ability to handle sudden traffic spikes before throttling occurs. Over time, if your actual throughput consistently approaches the warm throughput levels, DynamoDB may split busy partitions based on observed usage patterns.

- **Identify your hot keys:** If the table didn't resolve it automatically and your warm throughput is high or raising it didn't help, you'll need to identify specific hot keys. Use the section called "Identifying hot keys using CloudWatch Contributor Insights" to determine if any particular partition key values are hot. This is a first step to target your mitigation efforts effectively. Consider these common patterns:
  - If you see the same partition key appearing frequently in your throttling data, this indicates a concentrated hot key.
  - If you do not see repeated keys but are writing data in an ordered way (such as sequential timestamps or scan-based operations that follow keyspace order), you likely have rolling hot partitions where different keys become hot over time as your writes move through the keyspace.

  Note that write throttling can also occur with operations like `BatchWriteItem` or transactions that affect multiple items simultaneously. When individual items within a `BatchWriteItem` request are throttled, DynamoDB does not propagate these throttling errors to the application code. Instead, DynamoDB returns information about the unprocessed items in the response, which your application must handle by retrying those specific items. For transactions, the entire operation fails with a `TransactionCanceledException` if any item experiences throttling. For these complex scenarios, you may need to analyze your application's write patterns and data ingestion workflows, correlate them with the timing of throttling events, and implement appropriate retry handling strategies.
- **Improve partition key design:** As a long-term solution, consider the section called "Improving partition key design" to distribute access more evenly across partitions. This approach often provides the most comprehensive resolution to hot partition issues by addressing the root cause. However, it requires careful planning as it involves significant migration challenges.

IndexReadKeyRangeThroughputExceeded

When this occurs

The consumption rate of one or more partitions in your DynamoDB GSI exceeds the partition's read throughput limit. This throttling occurs regardless of your GSI's total provisioned capacity and

affects both provisioned and on-demand tables. You can monitor the CloudWatch metrics in the section called "Common diagnosis and monitoring" to analyze your throttling event.

Remediation options

Consider these steps to address your throttling events:

- **Pre-warm capacity:** If throttling persists, check if your GSI is limited by its Understanding DynamoDB warm throughput capacity. Use warm throughput or increase read provisioned capacity in advance for expected traffic increases. Increasing warm throughput improves your GSI's ability to handle sudden traffic spikes before throttling occurs. Over time, if your actual throughput consistently approaches the warm throughput levels, DynamoDB may split busy partitions based on observed usage patterns.
- **Identify your hot keys:** If the GSI didn't resolve it automatically and your warm throughput is high or raising it didn't help, you'll need to identify specific hot keys. Use the section called "Identifying hot keys using CloudWatch Contributor Insights" to determine if any particular partition key values are hot. This is a first step to target your mitigation efforts effectively. Note that for GSIs, the partition key distribution may differ significantly from your base table, creating different hot key patterns.
- **Redesign GSI partition keys:** Consider whether your GSI key design might be creating artificial hot spots (such as status flags, date-only keys, or boolean attributes) that concentrate reads on a small number of partitions. Consider using composite keys that combine the low-cardinality attribute with a high-cardinality attribute (e.g., "ACTIVE#customer123" instead of just "ACTIVE") or apply the section called "Write sharding" techniques to the base table items that affect GSI distribution to distribute writes across multiple partitions. While querying sharded data requires additional application logic to aggregate results, this approach prevents throttling by distributing access patterns more evenly.

IndexWriteKeyRangeThroughputExceeded

When this occurs

The consumption rate of one or more partitions in your DynamoDB GSI exceeds the partition's write throughput limit. This throttling occurs regardless of your GSI's total provisioned capacity and affects both provisioned and on-demand tables. You can monitor the CloudWatch metrics in the section called "Common diagnosis and monitoring" to analyze your throttling event.

Remediation options

Consider these steps to address your throttling events:

- **Redesign GSI partition key:** Review your GSI partition key design to verify it has sufficient cardinality (uniqueness) to distribute writes evenly. A common cause of GSI write throttling is using low-cardinality attributes as GSI partition keys (such as status flags with only a few possible values). Even when your base table has well-distributed partition keys, your GSI can still experience hot partitions if its partition key concentrates writes to a small number of values. For example, if 80% of your items have status="ACTIVE", this creates a severe hot partition in a status-based GSI. Consider using composite keys that combine the low-cardinality attribute with a high-cardinality attribute (e.g., "ACTIVE#customer123" instead of just "ACTIVE") or apply the section called “Write sharding” techniques to the base table items that affect GSI distribution to distribute writes across multiple partitions. While querying sharded data requires additional application logic to aggregate results, this approach prevents throttling by distributing access patterns more evenly.
- **Pre-warm capacity:**Check if your GSI is limited by its Understanding DynamoDB warm throughput capacity. Use warm throughput or increase read provisioned capacity in advance for expected traffic increases. Increasing warm throughput improves your GSI's ability to handle sudden traffic spikes before throttling occurs. Over time, if your actual throughput consistently approaches the warm throughput levels, DynamoDB may split busy partitions based on observed usage patterns.
- **Optimize GSI projections:** Apply the section called “Optimizing GSI projections” techniques to reduce write volume to GSIs. Projecting fewer attributes can significantly reduce the write capacity consumed by each GSI update.

Common diagnosis and monitoring

When troubleshooting partition-level throttling, several CloudWatch metrics can help identify hot partitions and confirm the root cause.

Essential CloudWatch metrics

Monitor these key metrics to diagnose partition-level throttling:

- **Partition-level throttling events:** `ReadKeyRangeThroughputThrottleEvents` and `WriteKeyRangeThroughputThrottleEvents` track when individual partitions exceed their throughput limits. `ReadThrottleEvents` and `WriteThrottleEvents` track when any read or write requests exceed the provisioned capacity.

- **Capacity consumption:** `ConsumedReadCapacityUnits` and `ConsumedWriteCapacityUnits` show overall usage patterns.

Resolution procedures

Identifying hot keys using CloudWatch Contributor Insights

Use this procedure to identify which partition keys are causing throttling.

1. Enable CloudWatch Contributor Insights on your table or GSI to track the most throttled keys. Consider keeping CloudWatch Contributor Insights enabled continuously for real-time throttling alerts by using the *Throttled keys* mode. This mode focuses exclusively on throttled requests by only processing events when throttling occurs. This targeted monitoring is a cost effective way to maintain continuous visibility into throttling issues.
2. Identify which keys are causing the hot partition issues.
3. (If the full *Accessed and throttled keys mode* is enabled) Analyze the access patterns over time to determine if hot keys are consistent or occur during specific periods.

Improving partition key design

Use this approach when you can modify your table schema to better distribute traffic across partitions. When possible, this is the most effective long-term solution for hot partition issues. Ideally, partition key design should be carefully considered during the initial table design phase.

Partition key redesign represents a fundamental change to your data model that impacts your entire application ecosystem. Before proceeding with this approach, carefully consider these significant limitations:

- **Data migration complexity:** Redesigning partition keys requires migrating all existing data, which can be resource-intensive and time-consuming for large tables.
- **Application code changes:** All application code that reads or writes to the table must be updated to use the new key structure.
- **Production impact:** Migrating to a new key design often requires downtime or complex dual-write strategies during transition.

For comprehensive guidance and principles on partition key design, see the section called "Partition key design" and the section called "Distributing workloads".

Optimizing GSI projections

Review your application's query patterns to determine exactly which attributes need to be available when querying the GSI, and limit your projections to just those attributes. When you update attributes that aren't projected into a GSI, no write operation occurs on that GSI, reducing write throughput consumption during updates. This targeted projection strategy optimizes both performance and cost while still supporting your application's query requirements. Note that projecting fewer attributes reduces write capacity consumption but may require additional base table reads.

For more information about efficient projection strategies, see Best Practices for Using Secondary Indexes in DynamoDB.

Additional resources

The following blog posts provide hands-on examples and practical details for the concepts covered in this guide:

- For hands-on guidance about scaling DynamoDB and managing hot partitions, see Part 1: Scaling DynamoDB - How partitions, hot keys, and split for heat impact performance.
- For detailed information about how DynamoDB's split-for-heat mechanism works, its benefits, and implementation details, see Part 3: Summary and best practices.
- For detailed write sharding strategies, see the section called “Write sharding”.

2- Provisioned throughput exceeded

Provisioned capacity throttling occurs when your application's consumption rate exceeds the read or write capacity units (RCUs/WCUs) configured for your tables or global secondary indexes. While DynamoDB provides burst capacity to handle occasional traffic spikes, sustained requests beyond your provisioned limits result in throttling. When this happens, DynamoDB returns a `ProvisionedThroughputExceeded` throttling reason type in the throttling exception. The reason identifies whether the issue is with read or write operations and whether it affects the base table or a global secondary index.

Throttling can occur regardless of whether Auto Scaling is enabled. Auto Scaling adapts to increases in consumption, but it does not respond instantly and it is constrained by the maximum capacity limits you configure. This means throttling can still occur during sudden traffic spikes or when consumption exceeds your maximum Auto Scaling limits.

Provisioned throughput exceeded mitigation measures

This section provides resolution guidance for provisioned capacity throttling scenarios. Before using this guide, ensure you have identified the specific throttling reason from your application's exception handling, and determined the Amazon Resource Name (ARN) of the affected resource. For information on retrieving throttling reasons and identifying throttled resources, see the section called “Diagnosis framework”.

Before diving into specific throttling scenarios, first consider if the throttling is actually a problem that needs resolution:

- Occasional throttling is normal and expected in well-optimized DynamoDB applications. Throttling simply means you're consuming 100% of what you've provisioned. If your application handles throttling gracefully with retries and your overall performance meets requirements, the throttling may not require immediate action.
- However, if throttling is causing unacceptable client-side latency, degrading user experience, or preventing critical operations from completing in a timely manner, then proceed with the mitigation options below.

When you need to address throttling issues, first determine if your throttling is caused by:

- **Temporary traffic spikes:** Short-duration increases in traffic that exceed your provisioned capacity but aren't sustained. These require different strategies than continuous high traffic.
- **Continuous high traffic:** Sustained workloads that consistently exceed your provisioned capacity.

For traffic spikes, consider the strategies from the *Handle traffic spikes with Amazon DynamoDB provisioned capacity* blog in the section called “Additional resources”.

For continuous high traffic, consider the capacity adjustment options below:

- the section called “TableReadProvisionedThroughputExceeded”
- the section called “TableWriteProvisionedThroughputExceeded”
- the section called “IndexReadProvisionedThroughputExceeded”
- the section called “IndexWriteProvisionedThroughputExceeded”

TableReadProvisionedThroughputExceeded

When this occurs

Your application's read consumption rate exceeds the provisioned read capacity units (RCUs) configured for your table. You can monitor the CloudWatch metrics in the section called "Common diagnosis and monitoring" to analyze your throttling event.

Resolution approach

Consider these strategies to resolve read capacity throttling:

- **Switch to on-demand capacity mode:** Consider switching your table to on-demand if you experience frequent throttling from traffic spikes. On-demand eliminates provisioning concerns and automatically scales with your workload.
- **If staying with provisioned mode and Auto Scaling is not enabled:**
  - Consider increasing the table read capacity.
  - Enable Auto Scaling for read capacity on your table.
- **If Auto Scaling is enabled (default for tables created in the console):**
  - Optimize your table's read Auto Scaling parameters.

TableWriteProvisionedThroughputExceeded

When this occurs

Your application's write consumption rate exceeds the provisioned write capacity units (WCUs) configured for your table. You can monitor the CloudWatch metrics in the section called "Common diagnosis and monitoring" to analyze your throttling event.

Resolution approach

Consider these strategies to resolve write capacity throttling:

- **Switch to on-demand capacity mode:** Consider switching your table to on-demand if you experience frequent throttling from traffic spikes. On-demand eliminates provisioning concerns and automatically scales with your workload.
- **If staying with provisioned mode and Auto Scaling is not enabled:**
  - Consider increasing the table write capacity.

- Enable Auto Scaling for write capacity on your table.
- **If Auto Scaling is enabled (default for tables created in the console):**
  - Optimize your table's write Auto Scaling parameters.

IndexReadProvisionedThroughputExceeded

When this occurs

Read consumption on a Global Secondary Index (GSI) exceeds the GSI's provisioned read capacity units (RCUs). You can monitor the CloudWatch metrics in the section called "Common diagnosis and monitoring" to analyze your throttling event.

Resolution approach

Consider these strategies to resolve GSI read capacity throttling:

- **Switch to on-demand capacity mode:** Consider switching the base table to on-demand if you experience frequent throttling from traffic spikes. On-demand eliminates provisioning concerns and automatically scales with your workload.
- **If staying with provisioned mode and Auto Scaling is not enabled:**
  - Consider increasing the GSI read capacity.
  - Enable Auto Scaling for read capacity on your GSI.
- **If Auto Scaling is enabled (default for tables created in the console):**
  - Optimize your GSI's read Auto Scaling parameters.

IndexWriteProvisionedThroughputExceeded

When this occurs

Updates to items in the base table trigger writes to a GSI that exceed the GSI's provisioned write capacity. This causes back-pressure throttling on base table writes. You can monitor the CloudWatch metrics in the section called "Common diagnosis and monitoring" to analyze your throttling event.

Resolution approach

Consider these strategies to resolve GSI write capacity throttling:

- **Switch to on-demand capacity mode:** Consider switching the base table to on-demand if you experience frequent throttling from traffic spikes. On-demand eliminates provisioning concerns and automatically scales with your workload.
- **If staying with provisioned mode and Auto Scaling is not enabled:**
  - Consider increasing the GSI write capacity.
  - Enable Auto Scaling for write capacity on your GSI.
- **If Auto Scaling is enabled (default for tables created in the console):**
  - Optimize your GSI's write Auto Scaling parameters.

Common diagnosis and monitoring

When troubleshooting throughput errors, several CloudWatch metrics can help identify the root cause.

Essential CloudWatch metrics

Monitor these key metrics to diagnose provisioned capacity throttling:

- **Throttling events:** `ReadProvisionedThroughputThrottleEvents` and `WriteProvisionedThroughputThrottleEvents` track when requests are throttled for this reason.`ReadThrottleEvents` and `WriteThrottleEvents` track when any read or write requests exceed the provisioned capacity.
- **Capacity consumption:** `ConsumedReadCapacityUnits` and `ConsumedWriteCapacityUnits` show actual usage.
- **Provisioned capacity:** `ProvisionedReadCapacityUnits` and `ProvisionedWriteCapacityUnits` show configured limits.

Resolution procedures

Increasing table throughput capacity

Use this procedure when Auto Scaling is not enabled and you need immediate capacity increase.

1. Update your table's provisioned capacity using the DynamoDB console, AWS CLI, or SDK:
   - **For read capacity:** Increase the `ReadCapacityUnits` parameter, which specifies the maximum number of strongly consistent reads consumed per second before DynamoDB throttles requests.

- **For write capacity:** Increase the `WriteCapacityUnits` parameter, which specifies the maximum number of writes consumed per second before DynamoDB throttles requests.

2. Verify that your new capacity settings don't exceed the per-table throughput quotas and that your total account consumption remains below the per-account throughput quotas for your Region. If you're approaching these limits, consider switching to on-demand capacity mode instead.

Configuring table Auto Scaling to adjust the read or write capacity of your table or GSI

Configure DynamoDB Auto Scaling to automatically adjust read or write capacity based on traffic patterns. You can configure Auto Scaling independently for both tables and GSIs, with separate controls for read and write capacity units.

1. Enable Auto Scaling for read capacity, write capacity, or both on your table or GSI.
2. Set a target utilization percentage with headroom for traffic spikes.

> **Note**
>
> Lower target utilization increases costs and scaling frequency. Targets below 40% may cause over-provisioning. Monitor usage patterns and costs to balance performance and efficiency.

3. Set capacity boundaries:

- **Minimum RCUs/WCUs:** Maintains sufficient capacity during low-traffic periods.
- **Maximum RCUs/WCUs:** Accommodates peak traffic demands and protects against runaway scaling events.

For guidance on configuring and managing DynamoDB Auto Scaling, see Managing throughput capacity automatically with DynamoDB Auto Scaling.

> **Note**
>
> Auto Scaling typically takes several minutes to respond to traffic changes. For sudden traffic spikes, your table's burst capacity provides immediate protection while Auto Scaling

adjusts. Configure target utilization with adequate headroom to allow time for scaling operations and to preserve burst capacity for unexpected demand.

Optimizing your table's or index's read or write Auto Scaling settings

Use this procedure when Auto Scaling is enabled but throttling still occurs. You can tune Auto Scaling independently for both tables and global secondary indexes (GSIs), with separate controls for read and write capacity units.

- **Adjust target utilization:** Consider lowering the target utilization for your table or GSIs to trigger scaling earlier before throttling occurs. Ensure that you monitor your traffic after making these adjustments. See the section called "Configuring table Auto Scaling to adjust the read or write capacity of your table or GSI" for more information about capacity consumption and cost implications.
- **Review capacity boundaries:** Ensure your minimum and maximum capacity settings align with your actual workload patterns.

Switching to on-demand capacity mode

For general information about switching capacity modes, see the section called "Switching capacity modes". Refer to the Service Quotas to learn about specific constraints when switching mode.

Increasing GSI throughput capacity

Use this procedure when Auto Scaling is not enabled on your GSI or you need immediate capacity increase.

1. Update the GSI's provisioned capacity using the DynamoDB console, AWS CLI, or SDK:
   - **For read capacity:** Increase the `ReadCapacityUnits` parameter for the specific GSI, which specifies the maximum number of reads the GSI can consume per second before DynamoDB throttles requests. Note that GSIs only support eventually consistent reads.
   - **For write capacity:** Increase the `WriteCapacityUnits` parameter for the specific GSI, which specifies the maximum number of writes the GSI can consume per second before DynamoDB throttles requests.
2. Ensure that the GSI's provisioned throughput capacity remains within the per-account and per-table throughput quotas.

Additional resources

- For detailed information about handling traffic spikes in DynamoDB provisioned capacity tables, including various strategies from utilizing Auto Scaling and burst capacity to strategic throttle management, see Handle traffic spikes with Amazon DynamoDB provisioned capacity.
- For information about how to use a cron expression to schedule a scaling policy, see Optimize costs by scheduling provisioned capacity for DynamoDB.
- For hands-on information about monitoring and analyzing throughput utilization patterns for your DynamoDB tables in provisioned capacity mode, see How to evaluate throughput utilization for Amazon DynamoDB tables in provisioned mode.

3- Account limits exceeded

On-demand tables do not have provisioned capacity levels to manage, but DynamoDB enforces account-level throughput limits to prevent runaway execution and ensure fair resource usage across all customers. These per-table account limits serve as adjustable safeguards, set for each account and Region combination. When your read or write consumption rate exceeds these limits, DynamoDB returns an `AccountLimitExceeded` throttling reason type in the throttling exception. The default per-table account limits automatically apply when tables do not have custom maximum throughput settings configured. You can optionally configure maximum throughput settings for finer cost control and predictability, or request quota increases through the the section called “Quotas” console if your application requirements exceed the default limits.

Account limit exceeded mitigation measures

This section provides resolution guidance for account limit throttling scenarios. Before using this guide, ensure you have identified the specific throttling reasons from your application's exception handling, and determined the Amazon Resource Name (ARN) of the affected resource. For information on retrieving throttling reasons and identifying throttled resources, see the section called “Diagnosis framework”.

Before diving into specific throttling scenarios, first determine if action is actually needed:

- **Evaluate performance impact:** Check if your application is still meeting its performance requirements despite the throttling. Many applications operate successfully at or near account limits, particularly during bulk operations or data migrations.
- **Review throttling patterns:** If throttling is intermittent and your application handles retries effectively, the current limits may be sufficient for your workload.

If your application performs acceptably even when occasionally hitting account limits, you might choose to simply monitor the situation rather than implementing immediate changes.

If you determine that throttling is causing unacceptable performance issues or reliability concerns, select a specific throttling reason below to find recommended mitigation options:

- the section called “TableReadAccountLimitExceeded”
- the section called “TableWriteAccountLimitExceeded”
- the section called “IndexReadAccountLimitExceeded”
- the section called “IndexWriteAccountLimitExceeded”

TableReadAccountLimitExceeded

When this occurs

Your table's read consumption has exceeded the account-level per-table read throughput quota for your Region. You can monitor the CloudWatch metrics in the section called “Common diagnosis and monitoring” to analyze your throttling event.

Resolution approach

Use the following steps to resolve this throttling:

- **Request quota increases:**

  Request an increase to the per-table read throughput limit (Quota code L-CF0CBE56). For detailed steps on how to submit the request, see Requesting per-table quota increases.

TableWriteAccountLimitExceeded

When this occurs

Your table's write consumption has exceeded the account-level per-table write throughput quota for your Region. You can monitor the CloudWatch metrics in the section called “Common diagnosis and monitoring” to analyze your throttling event.

Resolution approach

Use the following steps to resolve this throttling:

- **Request quota increases:** Request an increase to the per-table write throughput limit (Quota code L-AB614373). For detailed steps on how to submit the request, see Requesting per-table quota increases.

IndexReadAccountLimitExceeded

When this occurs

The read operations directed at a Global Secondary Index (GSI) consume more throughput than your account's per-table read quota allows in your current AWS Region. The account-level per-table read throughput quota applies collectively to a table and all its GSIs combined. You can monitor the CloudWatch metrics in the section called "Common diagnosis and monitoring" to analyze your throttling event.

Resolution approach

Choose the appropriate resolution based on your account's capacity distribution:

- **Request quota increases:** Request an increase to the per-table read throughput limit (Quota code L-CF0CBE56). For detailed steps on how to submit the request, see Requesting per-table quota increases.
- **Optimize GSI usage:** Review GSI design and query patterns to reduce unnecessary read capacity consumption.

IndexWriteAccountLimitExceeded

When this occurs

Write operations to your base table generate corresponding updates to a GSI that collectively exceed the account-level per-table write throughput quota for your AWS Region. Every write to a base table item that contains attributes indexed by a GSI trigger a corresponding write operation to that GSI. These combined write operations count toward your per-table write throughput quota. You can monitor the CloudWatch metrics in the section called "Common diagnosis and monitoring" to analyze the patterns and timing of these throttling events and identify which operations are causing the excessive GSI write activity.

Resolution approach

Choose the appropriate resolution based on your account's capacity distribution:

- **Request quota increases:** Request an increase to the per-table write throughput limit (Quota code L-AB614373) to accommodate higher GSI write traffic from base table operations. The per-table write throughput quota applies to the entire table, including all of its GSIs. For detailed steps on how to submit the request, see Requesting per-table quota increases.
- **Optimize GSI projections:** Review GSI projections and design to reduce write volume to GSIs.

Common diagnosis and monitoring

When troubleshooting account limit exceeded throttling events, several CloudWatch metrics can help identify whether you're hitting per-table or account-wide limits and understand your capacity distribution patterns.

Essential CloudWatch metrics

Monitor these key metrics to diagnose account limit throttling:

- **Account limit throttling events:** `ReadAccountLimitThrottleEvents` and `WriteAccountLimitThrottleEvents` track when requests are throttled due to account-level limits. `ReadThrottleEvents` and `WriteThrottleEvents` track when any read or write requests exceed the provisioned capacity.
- **Capacity consumption by resource:** `ConsumedReadCapacityUnits` and `ConsumedWriteCapacityUnits` for each table and GSI show individual resource usage.
- **Account-wide consumption:** Use CloudWatch metric math expressions to sum consumed capacity across all tables and GSIs to monitor total account usage.

Resolution procedures

Requesting per-table quota increases

If your applications need to operate beyond the current per-table throughput limits, you should submit a quota increase request using the procedure below. Each DynamoDB table in your AWS account (together with all its associated GSIs) is subject to these throughput quotas within a specific Region. These quotas represent the maximum read or write capacity that any individual table and its GSIs can collectively consume, and they apply independently to each table rather than as an aggregate across all tables in your account.

Optionally, you can also set lower limits on a per-table or per-GSI basis by configuring their maximum on-demand throughput settings.

1. Identify the specific quota that needs to be increased:
   - **Per-table read throughput limit** (Quota code L-CF0CBE56): Default 40,000 RCUs per table
   - **Per-table write throughput limit** (Quota code L-AB614373): Default 40,000 WCUs per table
2. Use the AWS Service Quotas console to request an increase:
   - Navigate to the DynamoDB service in Service Quotas
   - Find the appropriate quota using the quota code
   - Request an increase based on your projected peak usage
3. Provide justification for the increase, including:
   - Current usage patterns and peak traffic requirements
   - Business justification for the increased capacity
   - Timeline for when the increased capacity is needed

> **Note**
>
> Quota increases typically take 24-48 hours to process. Plan your requests accordingly and consider temporary mitigation strategies while waiting for approval.

Optimizing GSI projections and design

Optimize your Global Secondary Index (GSI) projections and design to reduce capacity consumption and improve performance.

Selective projection strategies

If your queries only need to access a few attributes, projecting only those attributes reduces the amount of data written to the GSI when base table items change. For details on projection types, see Projections for Global Secondary Indexes.

1. Analyze query patterns: Review your application's query patterns to identify which attributes are actually accessed through the GSI.
2. Use selective projections: Only project attributes that are actually needed in queries to reduce write volume.

3. Consider KEYS_ONLY: If your queries only need the key attributes, use KEYS_ONLY projection to minimize write volume.
4. Balance read vs. write trade-offs: Projecting fewer attributes reduces write capacity consumption but may require additional base table reads.

Sparse GSI implementation

Sparse GSIs contain only items that have the indexed attribute, rather than all items from your base table. This reduces partition density and improves performance when you frequently query specific subsets of data.

1. Design GSIs that only include items with specific attribute values.
2. Implement conditional indexing by only setting the GSI partition key attribute on items that should be indexed.
3. Use composite keys in sparse GSIs (e.g., status#timestamp) to further distribute traffic within the subset of indexed items.

For more information about implementing these strategies, see Best Practices for Using Secondary Indexes in DynamoDB.

4- On-demand maximum throughput exceeded

When you configure an on-demand table or GSI, you can optionally set a maximum throughput limit (`MaxReadRequestUnits` and `MaxWriteRequestUnits`) at the table or index level to prevent runaway costs or protect downstream systems from being overwhelmed. For more information about maximum throughput, see the section called "DynamoDB maximum throughput for on-demand tables".

When your read or write consumption exceeds these self-imposed limits, additional requests that would exceed the limit receive quick throttle responses. DynamoDB returns exceptions with a `MaxOnDemandThroughputExceeded` throttling reason type, indicating which resource has reached its throughput boundary.

On-demand maximum throughput exceeded throttling

This section provides resolution guidance for on-demand maximum throughput exceeded throttling scenarios. Before using this guide, ensure you have identified the specific throttling

reasons from your application's exception handling, and determined the Amazon Resource Name (ARN) of the affected resource. For information on retrieving throttling reasons and identifying throttled resources, see the section called “Diagnosis framework”.

Before diving into specific throttling scenarios, first consider whether action is actually needed:

- **Evaluate your maximum throughput settings:** These limits were intentionally configured to control costs or protect downstream systems. If you're receiving `MaxOnDemandThroughputExceeded` throttling events, your limits are working as designed. Consider whether increasing these limits aligns with your original cost control or system protection goals.
- **Assess application impact:** Determine if the throttling is actually causing problems for your applications or users. If your applications handle retries effectively and meets their performance requirements despite occasional throttling, maintaining your current limits might be the appropriate choice.
- **Review traffic patterns:** Analyze whether the throttling represents an expected traffic pattern or an unusual spike. For predictable, recurring traffic patterns that consistently exceed your limits, adjusting the maximum throughput settings might be warranted. For temporary spikes, implementing better request distribution strategies might be more appropriate than raising limits.

If after consideration you determine that your maximum throughput settings need adjustment, refer to the specific throttling scenarios below for targeted remediation options:

- the section called “TableReadMaxOnDemandThroughputExceeded”
- the section called “TableWriteMaxOnDemandThroughputExceeded”
- the section called “IndexReadMaxOnDemandThroughputExceeded”
- the section called “IndexWriteMaxOnDemandThroughputExceeded”

TableReadMaxOnDemandThroughputExceeded

When this occurs

Your on-demand table has exceeded its configured maximum read throughput capacity. You can monitor the CloudWatch metrics in the section called “Common diagnosis and monitoring” to analyze your throttling event.

Remediation options

Consider these steps to address your throttling events:

- **Increase maximum throughput limit:** Use the DynamoDB console, AWS CLI, or the DynamoDB `UpdateTable` API to increase the `MaxReadRequestUnits` value for the affected table, then monitor and adjust. This allows your table to handle higher read throughput before throttling occurs.
- **Remove the maximum limit:** Set `MaxReadRequestUnits` to `-1` to remove the ceiling, allowing scaling based on demand up to your account-level throughput quotas. This removes your custom limit but still maintains AWS's account-level safeguards. However, it's important to monitor spending closely after removing this limit, as your table can now consume significantly more capacity before hitting the account-level quotas.

TableWriteMaxOnDemandThroughputExceeded

When this occurs

Your on-demand table has exceeded its configured maximum write throughput capacity. You can monitor the CloudWatch metrics in the section called “Common diagnosis and monitoring” to analyze your throttling event.

Remediation options

Consider these steps to address your throttling events:

- **Increase maximum throughput limit:** Use the DynamoDB console, AWS CLI, or the DynamoDB `UpdateTable` API to increase the `MaxWriteRequestUnits` value for the affected table, then monitor and adjust.
- **Remove the maximum limit:** Set `MaxWriteRequestUnits` to `-1` to remove the ceiling, allowing scaling based on demand up to your account-level throughput quotas. This removes your custom limit but still maintains AWS's account-level safeguards. However, it's important to monitor spending closely after removing this limit, as your table can now consume significantly more capacity before hitting the account-level quotas.

IndexReadMaxOnDemandThroughputExceeded

When this occurs

Read requests to a GSI in on-demand mode have exceeded the GSI's configured maximum read throughput capacity. You can monitor the CloudWatch metrics in the section called "Common diagnosis and monitoring" to analyze your throttling event.

Remediation options

Consider these steps to address your throttling events:

- **Increase GSI maximum throughput limit:** Use the DynamoDB console, AWS CLI, or the DynamoDB `UpdateTable` API to increase the `MaxReadRequestUnits` value for the affected GSI, then monitor and adjust.
- **Remove the GSI maximum limit:** Set `MaxReadRequestUnits` to `-1` for the GSI to remove the ceiling, allowing scaling based on demand up to your account-level throughput quotas. This removes your custom limit but still maintains AWS's account-level safeguards. However, it's important to monitor spending closely after removing this limit.

IndexWriteMaxOnDemandThroughputExceeded

When this occurs

Updates to items in the base table trigger writes to a GSI in on-demand mode that exceed the GSI's configured maximum write throughput capacity, causing back-pressure throttling. You can monitor the CloudWatch metrics in the section called "Common diagnosis and monitoring" to analyze your throttling event.

Remediation options

Consider these steps to address your throttling events:

- **Increase GSI maximum throughput limit:** Use the DynamoDB console, AWS CLI, or the DynamoDB `UpdateTable` API to increase the `MaxWriteRequestUnits` value for the affected GSI, then monitor and adjust.
- **Remove the GSI maximum limit:** Set `MaxWriteRequestUnits` to `-1` for the GSI to remove the ceiling, allowing scaling based on demand up to your account-level throughput quotas. This removes your custom limit but still maintains AWS's account-level safeguards. However, it's important to monitor spending closely after removing this limit.

Common diagnosis and monitoring

When troubleshooting on-demand maximum throughput exceeded throttling events, several CloudWatch metrics can help identify the root cause and scaling patterns.

Essential CloudWatch metrics

Monitor these key metrics to diagnose on-demand maximum throughput exceeded throttling:

- **Maximum throughput throttling events:** `ReadMaxOnDemandThroughputThrottleEvents` and `WriteMaxOnDemandThroughputThrottleEvents` track when requests are throttled due to exceeding maximum limits. `ReadThrottleEvents` and `WriteThrottleEvents` track when any read or write requests exceed the provisioned capacity.
- **Current maximum throughput configured for a table or global secondary index:** `OnDemandMaxReadRequestUnits` and `OnDemandMaxWriteRequestUnits` show the current maximum capacity limits.
- **Actual capacity consumption:** `ConsumedReadCapacityUnits` and `ConsumedWriteCapacityUnits` show actual usage patterns.

Analysis approach

Follow these steps to confirm on-demand maximum throughput exceeded diagnosis:

1. Compare consumed capacity to maximum capacity limits - check if consumption consistently approaches or exceeds maximum limits.
2. Review throttling event frequency and timing to identify patterns. Look for sudden increases in consumed capacity that coincides with your throttling event.
3. Use CloudWatch Contributor Insights to identify which items or partition keys consume the most capacity.

Understanding Global Secondary Index (GSI) write throttling and back pressure in DynamoDB

GSI back-pressure throttling represents one of the most complex throttling scenarios in DynamoDB because it creates an indirect relationship between write operations and throttling—your application writes to a base table but experiences throttling due to capacity constraints on one or several indexes.

Understanding GSI back-pressure throttling

When you write to a DynamoDB table, any global secondary indexes (GSIs) on that table are updated asynchronously using an eventually consistent model. If a GSI doesn't have sufficient capacity to handle these updates, DynamoDB throttles writes to the base table to maintain data consistency. This is called *GSI back-pressure*. For more information about how GSIs work, see Global Secondary Indexes in DynamoDB.

Unlike direct table throttling where the resource being accessed is also the resource causing throttling, GSI back-pressure creates a dependency between the base table and its indexes. Even if your base table has ample capacity, writes will be throttled if any associated GSI cannot handle the update volume. This relationship is particularly important to understand because partition-level constraints apply independently to both the base table and each GSI—each has its own partition structure and corresponding throughput limits.

GSI partitioning is based on the GSI's partition key, which is often different from the base table's partition key. Even if your base table access is perfectly distributed across partitions, your GSI updates might still concentrate on specific partitions, creating hot spots in the GSI. For general best practices on partition key design for both tables and GSIs, see DynamoDB partition key design.

For example, if your base table uses `customerId` as a partition key (evenly distributed) but your GSI uses `status` as a partition key (with limited possible values like "active", "pending", "closed"), updates to items with popular status values can create GSI hot partitions even when base table access is balanced. This creates a particularly challenging scenario where your application might experience throttling due to GSI hot partitions even though both the base table and GSI have sufficient overall capacity and the base table's access pattern appears well-distributed.

Even though the throttling exception points to the GSI (via `ResourceArn`), the actual operation being throttled is the write to the base table. This can be confusing because your application is writing to the base table but receiving an exception about the GSI.

Types of GSI throttling

GSI back-pressure throttling manifests through different exception types depending on the specific capacity constraint:

- **GSI provisioned capacity exceeded:** Occurs when the GSI lacks sufficient write capacity units to handle updates from base table operations. This produces a `ProvisionedThroughputExceededException` with the reason

IndexWriteProvisionedThroughputExceeded, and the ResourceArn points directly to the specific GSI experiencing capacity constraints.

- **GSI on-demand maximum throughput exceeded:** Occurs when GSI write operations surpass configured maximum limits on on-demand tables. This produces a ThrottlingException with the reason IndexWriteMaxOnDemandThroughputExceeded, identifying the specific GSI with configured throughput restrictions.
- **GSI partition limits exceeded:** Happens when individual GSI partitions exceed their throughput limits (hot partitions), even if overall GSI capacity appears sufficient. This generates a ThrottlingException with the reason IndexWriteKeyRangeThroughputExceeded, indicating hot partition issues on the specific GSI identified in the ResourceArn. This is particularly important because GSI partition distribution may differ significantly from the base table's partition distribution, creating hot spots in the GSI even when base table access is evenly distributed.
- **GSI account limits exceeded:** Triggers when write operations to a specific GSI exceed the per-table (or any individual GSI within that table) regional throughput boundaries set at the account level. DynamoDB returns a ThrottlingException with the reason IndexWriteAccountLimitExceeded, pointing to the GSI that pushed its usage beyond account limits. This throttling occurs independently for each GSI that surpasses the limit. For information about per-table, per-account,regional, service quotas, see the section called "Quotas".

CloudWatch throttling metrics

This page provides a comprehensive guide to CloudWatch metrics specifically designed to help you identify, diagnose, and resolve throttling issues in your DynamoDB tables and indexes.

General throttling metrics

- ThrottledRequests
  - Incremented by one when any event within a request is throttled, regardless of how many individual events within that request are throttled. For example, when updating an item in a table with Global Secondary Indexes (GSIs), multiple events occur: a write operation to the base table and a write operation to each index. If any of these individual events are throttled, the ThrottledRequests metric is only incremented once.

    This behavior is important to understand when monitoring and troubleshooting DynamoDB performance, as it may mask the true extent of throttling. For more comprehensive

insights, compare the `ThrottledRequests` metric with the specific event-level metrics like `ReadThrottleEvents`, `WriteThrottleEvents`, and targeted metrics such as `ReadKeyRangeThroughputThrottleEvents` for example. The complete list of these cause-specific metrics is available in this page. Each metric corresponds to particular throttling reasons that are captured within the throttling exception. For guidance on retrieving and interpreting these reasons during throttling events, see the the section called “Diagnosing throttling” section which provides instructions for identifying and resolving the root causes of throttling issues.

- `ReadThrottleEvents`
  - Watch for requests that exceed the provisioned RCU for a table or GSI.
- `WriteThrottleEvents`
  - Watch for requests that exceed the provisioned WCU for a table or GSI.

Detailed throttling metrics by cause

On-Demand throughput throttling

- `ReadMaxOnDemandThroughputThrottleEvents`
  - Number of read requests throttled due to on-demand maximum throughput.
- `WriteMaxOnDemandThroughputThrottleEvents`
  - Number of write requests throttled due to on-demand maximum throughput.

Account-Level throttling

- `ReadAccountLimitThrottleEvents`
  - Number of read requests throttled due to account limits.
- `WriteAccountLimitThrottleEvents`
  - Number of write requests throttled due to account limits.

Partition-Level throttling

- `ReadKeyRangeThroughputThrottleEvents`
  - Number of read requests throttled due to partition limits.
- `WriteKeyRangeThroughputThrottleEvents`
  - Number of write requests throttled due to partition limits.

Capacity analysis metrics

- `OnlineIndexConsumedWriteCapacity`
  - When you add a new GSI to an existing table, DynamoDB performs a backfill operation that copies data from the base table to the new index. This process consumes write capacity units. The `OnlineIndexConsumedWriteCapacity` metric tracks this specific consumption.

    This consumption is separate from and additional to the regular write operations tracked by `ConsumedWriteCapacityUnits`. The regular `ConsumedWriteCapacityUnits` metric for a GSI does not include the write capacity consumed during the initial index creation process.
- `ProvisionedReadCapacityUnits` and `ProvisionedWriteCapacityUnits`
  - View how many provisioned read or write capacity units were consumed over the specified time period, for a table or a specified global secondary index.
  - Note that the `TableName` dimension returns `ProvisionedReadCapacityUnits` for the table only by default. To view the number of provisioned read or write capacity units for a global secondary index, you must specify both `TableName` and `GlobalSecondaryIndexName`.
- `ConsumedReadCapacityUnits` and `ConsumedWriteCapacityUnits`
  - View how many read or write capacity units were consumed over the specified time period. `ConsumedWriteCapacityUnits` does not include the write capacity consumed during the initial index creation process.

For more information on DynamoDB CloudWatch metrics, see DynamoDB Metrics and dimensions.