**DynamoDB Best Practices for Querying and Scanning Data**

This section covers some best practices for using Query and Scan operations in Amazon DynamoDB.

**Performance Considerations for Scans**

In general, Scan operations are less efficient than other operations in DynamoDB. A Scan operation always scans the entire table or secondary index. It then filters out values to provide the result you want, essentially adding the extra step of removing data from the result set.

If possible, you should avoid using a Scan operation on a large table or index with a filter that removes many results. Also, as a table or index grows, the Scan operation slows. The Scan operation examines every item for the requested values and can use up the provisioned throughput for a large table or index in a single operation. For faster response times, design your tables and indexes so that your applications can use Query instead of Scan. (For tables, you can also consider using the GetItem and BatchGetItem APIs.)

Alternatively, design your application to use Scan operations in a way that minimizes the impact on your request rate.

**Avoiding Sudden Spikes in Read Activity**

When you create a table, you set its read and write capacity unit requirements. For reads, the capacity units are expressed as the number of strongly consistent 4 KB data read requests per second. For eventually consistent reads, a read capacity unit is two 4 KB read requests per second. A Scan operation performs eventually consistent reads by default, and it can return up to 1 MB (one page) of data. Therefore, a single Scan request can consume (1 MB page size / 4 KB item size) / 2 (eventually consistent reads) = 128 read operations. If you request strongly consistent reads instead, the Scan operation would consume twice as much provisioned throughput—256 read operations.

This represents a sudden spike in usage, compared to the configured read capacity for the table. This usage of capacity units by a scan prevents other potentially more important requests for the same table from using the available capacity units. As a result, you likely get a ProvisionedThroughputExceeded exception for those requests.

The problem is not just the sudden increase in capacity units that the Scan uses. The scan is also likely to consume all of its capacity units from the same partition because the scan requests read items that are next to each other on the partition. This means that the request is hitting the same partition, causing all of its capacity units to be consumed, and throttling other requests to that partition. If the request to read data is spread across multiple partitions, the operation would not throttle a specific partition.

The following diagram illustrates the impact of a sudden spike of capacity unit usage by Query and Scan operations, and its impact on your other requests against the same table.


        4 different scenarios showing provisioned throughput intervals, requests, and good
          and bad results on a table.
      

As illustrated here, the usage spike can impact the table's provisioned throughput in several ways:

1. Good: Even distribution of requests and size
2. Not as good: Frequent requests in bursts
3. Bad: A few random large requests
4. Bad: Large scan operations

Instead of using a large Scan operation, you can use the following techniques to minimize the impact of a scan on a table's provisioned throughput.

* **Reduce page size**

Because a Scan operation reads an entire page (by default, 1 MB), you can reduce the impact of the scan operation by setting a smaller page size. The Scan operation provides a *Limit* parameter that you can use to set the page size for your request. Each Query or Scan request that has a smaller page size uses fewer read operations and creates a "pause" between each request. For example, suppose that each item is 4 KB and you set the page size to 40 items. A Query request would then consume only 20 eventually consistent read operations or 40 strongly consistent read operations. A larger number of smaller Query or Scan operations would allow your other critical requests to succeed without throttling.

* **Isolate scan operations**

DynamoDB is designed for easy scalability. As a result, an application can create tables for distinct purposes, possibly even duplicating content across several tables. You want to perform scans on a table that is not taking "mission-critical" traffic. Some applications handle this load by rotating traffic hourly between two tables—one for critical traffic, and one for bookkeeping. Other applications can do this by performing every write on two tables: a "mission-critical" table, and a "shadow" table.

Configure your application to retry any request that receives a response code that indicates you have exceeded your provisioned throughput. Or, increase the provisioned throughput for your table using the UpdateTable operation. If you have temporary spikes in your workload that cause your throughput to exceed, occasionally, beyond the provisioned level, retry the request with exponential backoff. For more information about implementing exponential backoff, see [Error Retries and Exponential Backoff](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Programming.Errors.html#Programming.Errors.RetryAndBackoff).

**Taking Advantage of Parallel Scans**

Many applications can benefit from using parallel Scan operations rather than sequential scans. For example, an application that processes a large table of historical data can perform a parallel scan much faster than a sequential one. Multiple worker threads in a background "sweeper" process could scan a table at a low priority without affecting production traffic. In each of these examples, a parallel Scan is used in such a way that it does not starve other applications of provisioned throughput resources.

Although parallel scans can be beneficial, they can place a heavy demand on provisioned throughput. With a parallel scan, your application has multiple workers that are all running Scan operations concurrently. This can quickly consume all of your table's provisioned read capacity. In that case, other applications that need to access the table might be throttled.

A parallel scan can be the right choice if the following conditions are met:

* The table size is 20 GB or larger.
* The table's provisioned read throughput is not being fully used.
* Sequential Scan operations are too slow.

**Choosing TotalSegments**

The best setting for TotalSegments depends on your specific data, the table's provisioned throughput settings, and your performance requirements. You might need to experiment to get it right. We recommend that you begin with a simple ratio, such as one segment per 2 GB of data. For example, for a 30 GB table, you could setTotalSegments to 15 (30 GB / 2 GB). Your application would then use 15 workers, with each worker scanning a different segment.

You can also choose a value for TotalSegments that is based on client resources. You can set TotalSegments to any number from 1 to 1000000, and DynamoDB lets you scan that number of segments. For example, if your client limits the number of threads that can run concurrently, you can gradually increase TotalSegments until you get the best Scan performance with your application.

Monitor your parallel scans to optimize your provisioned throughput use, while also making sure that your other applications aren't starved of resources. Increase the value for TotalSegments if you don't consume all of your provisioned throughput but still experience throttling in your Scan requests. Reduce the value for TotalSegments if the Scan requests consume more provisioned throughput than you want to use.