A fundamental principle of building systems in a serverless world is that your functions should do their job as quickly as possible. The longer functions run, the more they cost.

Regardless of what your Lambda functions are doing, in most use cases, a big slice of the processing time will be spent sending data somewhere else.

If the goal is super-fast Lambdas; and sending data costs so much time, we should be trying to make this action as efficient as possible. The solution is batching data before sending.

# Scope

I’ll cover the following topics:

* What is batching?
* Performance: Just how much faster is batching?
* Complexity and error handling: Is batching just too complicated to be worth it?
* Don’t Repeat Yourself (DRY): Can using libraries make batching the most straightforward option?

# What Is Batching?

Many AWS Services have **batch-write APIs**. Terminology can vary between services: “batch write”, “batch send messages,” and “put records,” for example.

However, they all mean the same thing — instead of sending a piece of data as an individual item, you can group separate items and send them in a single transaction, reducing processing time.

# Performance

To prove just how much time is saved, let’s run some tests.

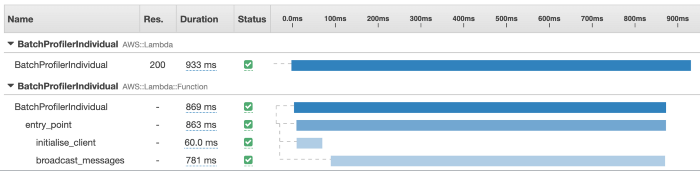
These tests will use two Lambda functions which will send data to an **SQS queue**. They are identical except one use **send\_message**and the other uses **send\_message\_batch**. We’ll use [AWS X-Ray](https://aws.amazon.com/xray/) to profile the running time.

SQS **send\_message\_batch** has a maximum batch size of ten messages.

If you would like to run the following examples yourself, you can! The code and deployment instructions are [available on GitHub](https://github.com/g-farrow/message-batch-profiling).

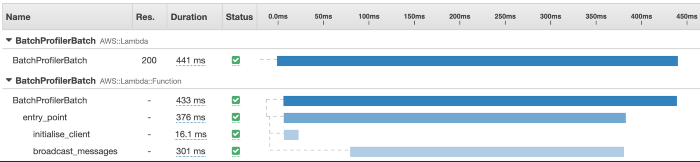
The method that deals with sending messages is **broadcast\_messages**.

Using the individual method **send\_message**, sending ten messages consistently takes 700-800ms:



Sending one message at a time takes ~75ms per message

Doing the same thing with the batch method, send\_message\_batch. We see much quicker processing time. All ten messages are consistently broadcast in ~300ms — about 60% less!



Sending ten messages in a single batch takes ~30ms per message

[Kinesis](https://aws.amazon.com/kinesis/) allows batches of up to 500 items. Running the test again, sending 500 messages to Kinesis, we see the following broadcast message timings:

* put\_record: 20-25 seconds (yes, seconds!).
* put\_records: 600-1,000ms.

That’s a potential reduction in broadcast time of 99.9%!

## Larger batches

Up to this point, the code for the two methods is very similar and reasonably simple. The functions can send all their messages in a single transaction.

But that’s not always going to be the case. For instance, the Lambda may be triggered by a Kinesis stream with inbound batches of maybe 1,000 messages.

To split up 1,000 messages into smaller batches, the Lambda would need to contain code to enable chunking. Although not hugely complicated, this code would need to be implemented across all Lambdas that utilized batch sending.

Furthermore, different AWS services have different maximum limits for sending data in batches. For instance, SQS is ten, but Kinesis allows up to 500.

# Error Handling

Another area we need to pay attention to is **error handling** and how this can be more complex when using batch send methods.

If a failure occurs, the individual sending method will throw an exception or return a response that indicates an error. This error relates to the message that is being sent at the time. Handling this scenario is relatively simple.

If a failure occurs during the use of a batch send method, it is not immediately clear which message is at fault.

Most AWS services will respond to a batch request to inform the client of any failures. To handle failures, the code must be able to **parse** the response. For example, this is the contract for an SQS **send\_message\_batch** request:

The Lambda will need to iterate over any items in Failed and reprocess them or otherwise handle them.

At this point, the code required to handle batches properly is becoming reasonably complex and writing code to handle these scenarios for each service you wish to use, and in each Lambda, you build, is going to be annoying and will waste time.

# Don’t Repeat Yourself

DRY is a fundamental principle of all software development. It is especially true in this situation. The code for chunking up batches, knowing how large a batch may be and then handling any failures or errors is reasonably static.

Regardless of how many Lambdas functions you write, or what their purpose is — handling of batch sending data will be standard across all of them.

Therefore, it is crucial to develop a library that you can implement once but use in multiple places.

## Boto3 Batch Utils

I did precisely that. I have written and maintain a Python library called [boto3-batch-utils](https://github.com/g-farrow/boto3_batch_utils) which abstracts the batch send capabilities of several AWS services.

Using a library such as this allows my Lambda functions all the benefits of efficient batch send — but without the complexity of error handling and batch size management.

sqs\_client = SQSBatchDispatcher('QUEUE\_NAME')broadcast\_messages\_sqs(messages\_list)sqs\_client.flush\_payloads()def broadcast\_messages\_sqs(messages\_list):  
 for message in messages\_list:  
 sqs\_client.submit\_payload(message)

Using boto3-batch-utils requires fewer lines of code than sending individual messages. To learn a bit more about how I approached the library, [you can read the docs](https://g-farrow.github.io/boto3_batch_utils/).

# Richer Functionality

The benefits of using a library go beyond error handling and batch size management. Here are a few of the features I have implemented in Boto3 Batch Utils:

## Client initialization efficiencies

A Boto3 client is only initialized if it is required. It is subsequently re-used across invocations to achieve even more time savings.

## Automatic retry

In some scenarios, failing to send part of a batch may not be a severe issue. For instance, if the client was throttled. If this happens, a retry would likely be successful.

Automatically retrying only, the messages that failed helps to maximize efficiency and reduce failures.

## Limits

The library can store all the AWS service limits for batch sizes. This means you don’t need to remember them or look them up.

# It’s Got to Be Batch!

Whether you use boto3-batch-utils or write your own, if you want to turbocharge your Lambda functions — I strongly recommend using a library to allow you to take advantage of batch sending data.