Introducing faster polling scale-up for AWS Lambda functions configured with Amazon SQS

by James Beswick | on 06 NOV 2023 | in Amazon Simple Queue Service (SQS), AWS Lambda, Serverless | Permalink | Share

This post was written by Anton Aleksandrov, Principal Solutions Architect, and Tarun Rai Madan, Senior Product Manager.

Today, AWS is announcing that AWS Lambda supports up to five times faster polling scale-up rate for spiky Lambda workloads configured with Amazon Simple Queue Service (Amazon SQS) as an event source.

This feature enables customers building event-driven applications using Lambda and SQS to achieve more responsive scaling during a sudden burst of messages in their SQS queues, and reduces the need to duplicate Lambda functions or SQS queues to achieve faster message processing.

Overview

Customers building modern event-driven and messaging applications with AWS Lambda use the Amazon SQS as a fundamental building block for creating decoupled architectures. Amazon SQS is a fully managed message queueing service for microservices, distributed systems, and serverless applications. When a Lambda function subscribes to an SQS queue as an event source, Lambda polls the queue, retrieves the messages, and sends retrieved messages in batches to the function handler for processing. To consume messages efficiently, Lambda detects the increase in queue depth, and increases the number of poller processes to process the queued messages.

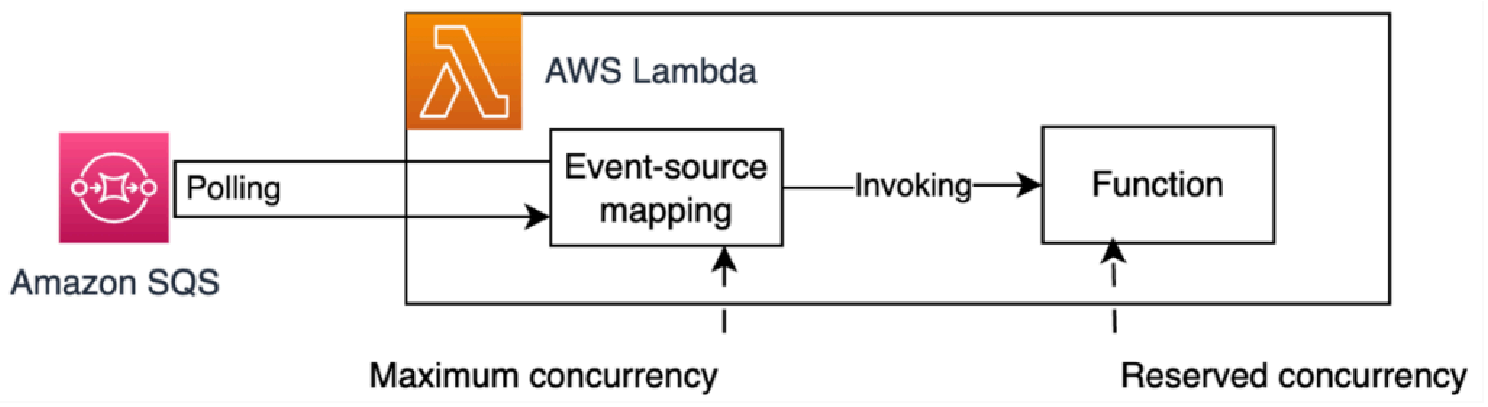
Up until today, the Lambda was adding up to 60 concurrent executions per minute for Lambda functions subscribed to SQS queues, scaling up to a maximum of 1,250 concurrent executions in approximately 20 minutes. However, customers tell us that some of the modern event-driven applications they build using Lambda and SQS are sensitive to sudden spikes in messages, which may cause noticeable delay in processing of messages for end users. In order to harness the power of Lambda for applications that experience a burst of messages in SQS queues, these customers needed Lambda message polling to scale up faster.

With today’s announcement, Lambda functions that subscribe to an SQS queue can scale up to five times faster for queues that see a spike in message backlog, adding up to 300 concurrent executions per minute, and scaling up to a maximum of 1,250 concurrent executions. This scaling improvement helps to use the simplicity of Lambda and SQS integration to build event-driven applications that scale faster during a surge of incoming messages, particularly for real-time systems. It also offers customers the benefit of faster processing during spikes of messages in SQS queues, while continuing to offer the flexibility to limit the maximum concurrent Lambda invocations per SQS event source.

Controlling the maximum concurrent Lambda invocations by SQS

The new improved scaling rates are automatically applied to all AWS accounts using Lambda and SQS as an event source. There is no explicit action that you must take, and there’s no additional cost. This scaling improvement helps customers to build more performant Lambda applications where they need faster SQS polling scale-up. To prevent potentially overloading the downstream dependencies, Lambda provides customers the control to set the maximum number of concurrent executions at a function level with *reserved concurrency*, and event source level with *maximum concurrency*.

The following diagram illustrates settings that you can use to control the flow rate of an SQS event-source. You use reserved concurrency to control function-level scaling, and maximum concurrency to control event source scaling.



Reserved concurrency is the maximum concurrency that you want to allocate to a function. When a function has reserved concurrency allocated, no other functions can use that concurrency.

AWS recommends using reserved concurrency when you want to ensure a function has enough concurrency to scale up. When an SQS event source is attempting to scale up concurrent Lambda invocations, but the function has already reached the threshold defined by the reserved concurrency, the Lambda service throttles further function invocations.

This may result in SQS event source attempting to scale down, reducing the number of concurrently processed messages. Depending on the queue configuration, the throttled messages are returned to the queue for retrying, expire based on the retention policy, or sent to a dead-letter queue (DLQ) or on-failure destination.

The maximum concurrency setting allows you to control concurrency at the event source level. It allows you to define the maximum number of concurrent invocations the event source attempts to send to the Lambda function. For scenarios where a single function has multiple SQS event sources configured, you can define maximum concurrency for each event source separately, providing more granular control. When trying to add rate control to SQS event sources, AWS recommends you start evaluating maximum concurrency control first, as it provides greater flexibility.

Reserved concurrency and maximum concurrency are complementary capabilities, and can be used together. Maximum concurrency can help to prevent overwhelming downstream systems and throttled invocations. Reserved concurrency helps to ensure available concurrency for the function.

Example scenario

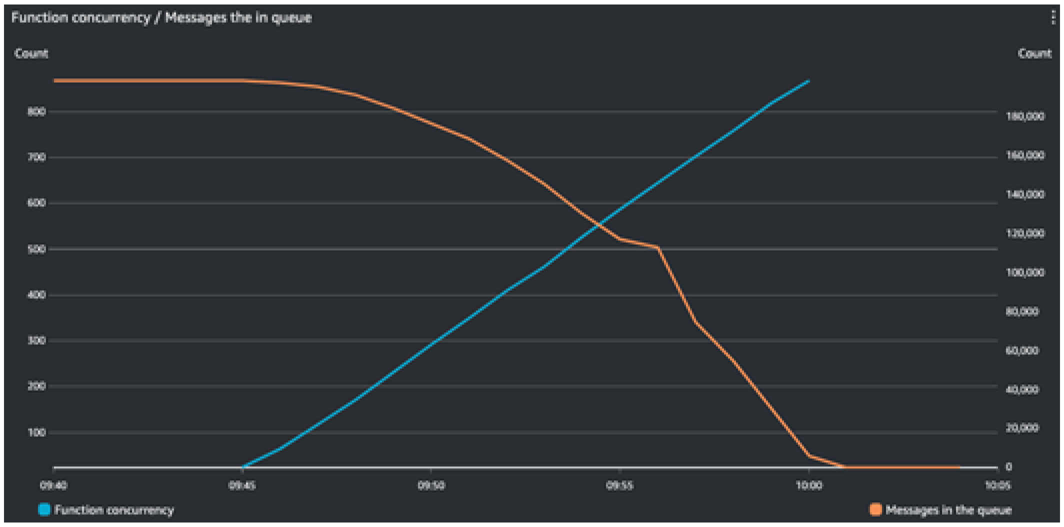
Consider your business must process large volumes of documents from storage. Once every few hours, your business partners upload large volumes of documents to S3 buckets in your account.

For resiliency, you've designed your application to send a message to an SQS queue for each of the uploaded documents, so you can efficiently process them without accidentally skipping any. The documents are processed using a Lambda function, which takes around two seconds to process a single document.

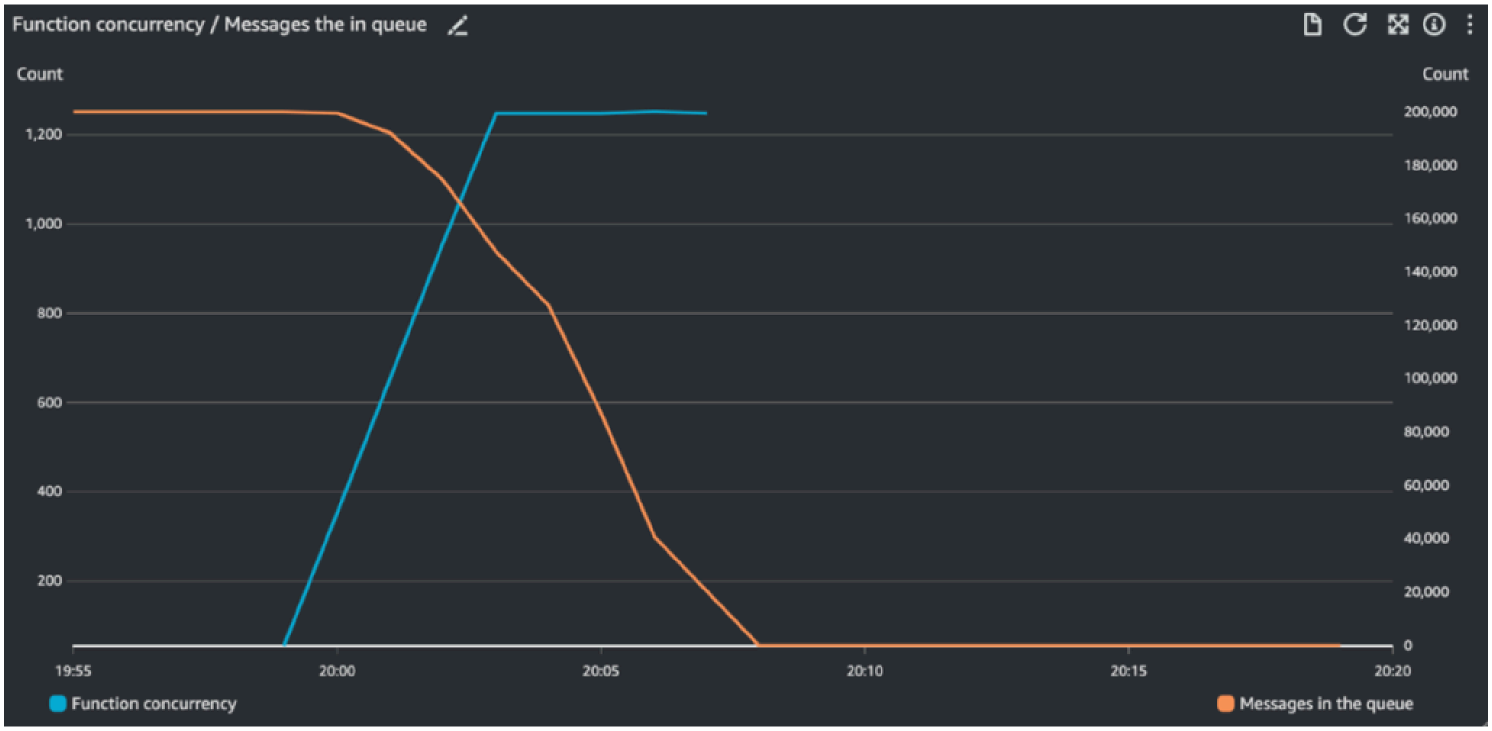
Processing these documents is a CPU-intensive operation, so you decide to process a single document per invocation. You want to use the power of Lambda to fan out the parallel processing to as many concurrent execution environments as possible. You want the Lambda function to scale up rapidly to process those documents in parallel as fast as possible, and scale-down to zero once all documents are processed to save costs.

When a business partner uploads 200,000 documents, 200,000 messages are sent to the SQS queue. The Lambda function is configured with an SQS event source, and it starts consuming the messages from the queue.

This diagram shows the results of running the test scenario before the SQS event source scaling improvements. As expected, you can see that concurrent executions grow by 60 per minute. It takes approximately 16 minutes to scale up to 900 concurrent executions gradually and process all the messages in the queue.



The following diagram shows the results of running the same test scenario after the SQS event source scaling improvements. The timeframe used for both charts is the same, but the performance on the second chart is better. Concurrent executions grow by 300 per minute. It only takes 4 minutes to scale up to 1,250 concurrent executions, and all the messages in the queue are processed in approximately 8 minutes.



Deploying this example

Use the example project to replicate this performance test in your own AWS account. Follow the instructions in README.md for provisioning the sample project in your AWS accounts using the AWS Cloud Development Kit (CDK).

This example project is configured to demonstrate a large-scale workload processing 200,000 messages. Running this sample project in your account may incur charges. See AWS Lambda pricing and Amazon SQS pricing.

Once deployed, use the application under the “sqs-cannon” directory to send 200,000 messages to the SQS queue (or reconfigure to any other number). It takes several minutes to populate the SQS queue with messages. After all messages are sent, enable the SQS event source, as described in the README.md, and monitor the charts in the provisioned CloudWatch dashboard.

The default concurrency quota for new AWS accounts is 1000. If you haven’t requested an increase in this quota, the number of concurrent executions is capped at this number. Use Service Quotas or contact your account team to request a concurrency increase.

Security best practices

Always use the least privileged permissions when granting your Lambda functions access to SQS queues. This reduces potential attack surface by ensuring that only specific functions have permissions to perform specific actions on specific queues. For example, in case your function only polls from the queue, grant it permission to read messages, but not to send new messages. A function execution role defines which actions your function is allowed to perform on other resources. A queue access policy defines the principals that can access this queue, and the actions that are allowed.

Use server-side encryption (SSE) to store sensitive data in encrypted SQS queues. With SSE, your messages are always stored in encrypted form, and SQS only decrypts them for sending to an authorized consumer. SSE protects the contents of messages in queues using SQS-managed encryption keys (SSE-SQS) or keys managed in the AWS Key Management Service (SSE-KMS).

Conclusion

The improved Lambda SQS event source polling scale-up capability enables up to five times faster scale-up performance for spiky event-driven workloads using SQS queues, at no additional cost. This improvement offers customers the benefit of faster processing during spikes of messages in SQS queues, while continuing to offer the flexibility to limit the maximum concurrent invokes by SQS as an event source.

For more serverless learning resources, visit Serverless Land.
TAGS: contributed, serverless