Scale Smarter - Rate Limiting AI Inference at Scale

Hello!

I am Tarun Annapareddy

Software Engineer @Google

The AI Quota Bottleneck

- **The Scale Paradox**: Runners scale up with backlog, but the external API's enforce strict global quotas.
- **Reactive Retry loops**: Reactive backoffs (429 error handling) are redundant attempts and waste worker compute cycles.
- **Inaccurate Hacks**: Approximating local limits and GroupByKey are inaccurate and costly.

Goals of API RateLimiter

Proactive Global Control

Users should be able to configure quota of external systems and Runner ensures request volume adheres to global API limits

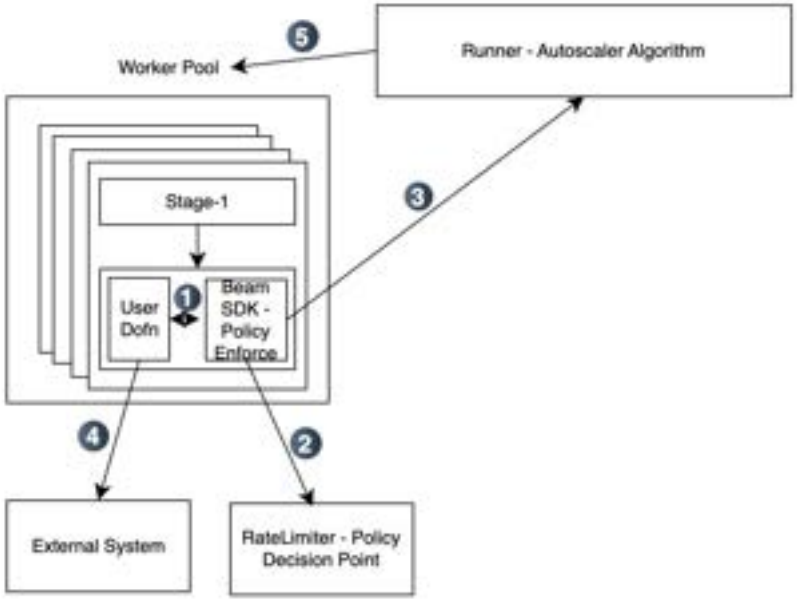
Runner Agnostic reporting & Autoscaler Integration

Any runner should be able to pick up throttling signals and implement their own Autoscaler Algorithm

Provide Abstraction and Integration across IO's

Provide standard Rate Limiter Interface that allows multiple implementations and simple enough to integrate across BeamIO's

The Big Picture



- An External Policy Decision Point is available for Beam
- Users Inform Beam SDK on the quota of the external system.
- Users DoFn/BeamIO can use the RateLimiter as a Policy Enforcement Point before making external API calls
- Beam reports the metrics to Autoscaler
- Runner Autoscaler algorithm takes care of scaling workers to save CPU cycles

RateLimiter Abstractions

RateLimiter Factory

```java
/**
 * A factory that manages connections to rate limit service and creates
lightweight handles.
 *
 */
public interface RateLimiterFactory extends Serializable, AutoCloseable {

  /**
   * Creates a lightweight ratelimiter handle bound to a specific context.
   *
   * <p>Use this when passing ratelimiter to IO components, which doesn't need to
know about the
   * configuration or the underlying ratelimiter service details. This is also
useful in DoFns when
   * you want to use the ratelimiter in a static way based on the compile time
context.
   */
  RateLimiter getLimiter(RateLimiterContext context);

  /**
   * Blocks until the specified number of permits are acquired and returns true if
the request was
   * allowed or false if the request was rejected.
   */
  boolean allow(RateLimiterContext context, int permits) throws IOException,
InterruptedException;
}
```

RateLimiter

```java
/**
 * A RateLimiter allows to fetch permits from a rate limiter service and blocks
execution when the rate limit is exceeded.
 */
public interface RateLimiter extends Serializable, AutoCloseable {

  /**
   * Blocks until the specified number of permits are acquired and returns true if
the request was
   * allowed or false if the request was rejected.
   */
  boolean allow(int permits) throws IOException, InterruptedException;
}
```

RateLimiter Abstractions

RateLimiter Options

```java
public abstract class RateLimiterOptions implements Serializable {
  @SchemaFieldDescription("Address of the rate limiter")
  public abstract String getAddress();

  @Nullable
  @SchemaFieldDescription("Maximum number of retries, defaults to infinite")
  public abstract Integer getMaxRetries();

  @SchemaFieldDescription("Timeout for rate limiter operations, defaults to 5 seconds")
  public abstract Duration getTimeout();
}
```

RateLimiter Context

```java
public abstract class EnvoyRateLimiterContext implements RateLimiterContext {

  @SchemaFieldDescription("Domain of the rate limiter.")
  public abstract String getDomain();

  @SchemaFieldDescription("Descriptors for the rate limiter.")
  public abstract ImmutableMap<String, String> getDescriptors();
}
```

Sample RateLimiter UseCase

RateLimiter DoFn Setup

```java
@Setup
public void setup() {
  // Create the RateLimiterOptions.
  RateLimiterOptions options =
RateLimiterOptions.builder().setAddress(rlsAddress).build();

  // Static RateLimtier with pre-configured domain and descriptors
  RateLimiterFactory factory = new EnvoyRateLimiterFactory(options);
  RateLimiterContext context =
      EnvoyRateLimiterContext.builder()
          .setDomain(rlsDomain)
          .addDescriptor("database", "users")
          .build();
  this.rateLimiter = factory.getLimiter(context);
}
```

RateLimiter Usage

```java
@ProcessElement
public void processElement(@Element String element, OutputReceiver<String>
receiver)
      throws Exception {
    try {
      Preconditions.checkNotNull(rateLimiter).allow(1);
    } catch (Exception e) {
      throw new RuntimeException("Failed to acquire rate limit token", e);
    }

    // Simulate external API call
    Thread.sleep(100);
    receiver.output("Processed: " + element);
  }
}
```

RateLimiter Run Inference Integration

Python

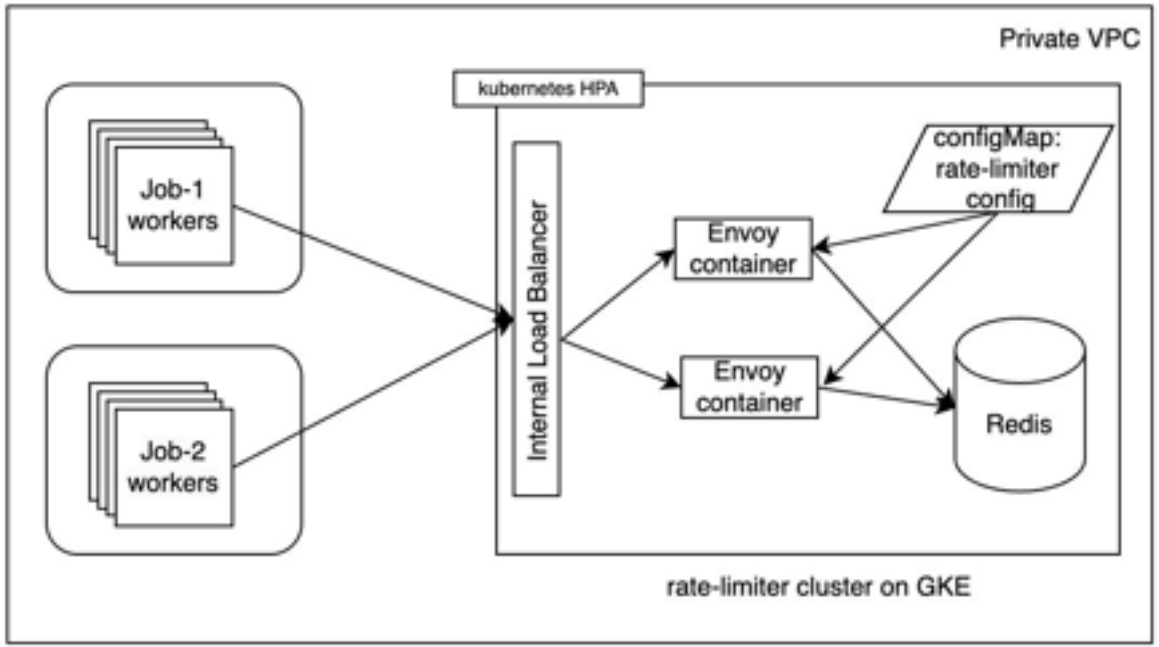
```python
# Initialize the EnvoyRateLimiter
rate_limiter = EnvoyRateLimiter(
    service_address=known_args.rls_address,
    domain="mongo_cps",
    descriptors=[{
        "database": "users"
    }],
    namespace='example_pipeline')

# Initialize the VertexAIModelHandler with the rate limiter
model_handler = VertexAIModelHandlerJSON(
    endpoint_id=known_args.endpoint_id,
    project=known_args.project,
    location=known_args.location,
    rate_limiter=rate_limiter)

# Input features for the model
features = [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0],
            [10.0, 11.0, 12.0], [13.0, 14.0, 15.0]]

with beam.Pipeline(options=pipeline_options) as p:
  _ = (
      p
      | 'CreateInputs' >> beam.Create(features)
      | 'RunInference' >> RunInference(model_handler)
      | 'PrintPredictions' >> beam.Map(logging.info))
```

Envoy RateLimiter Setup



High Level AutoScaling Algorithm

Throttle time signal provides back pressure against the desire to upsize and forward pressure to downsize.

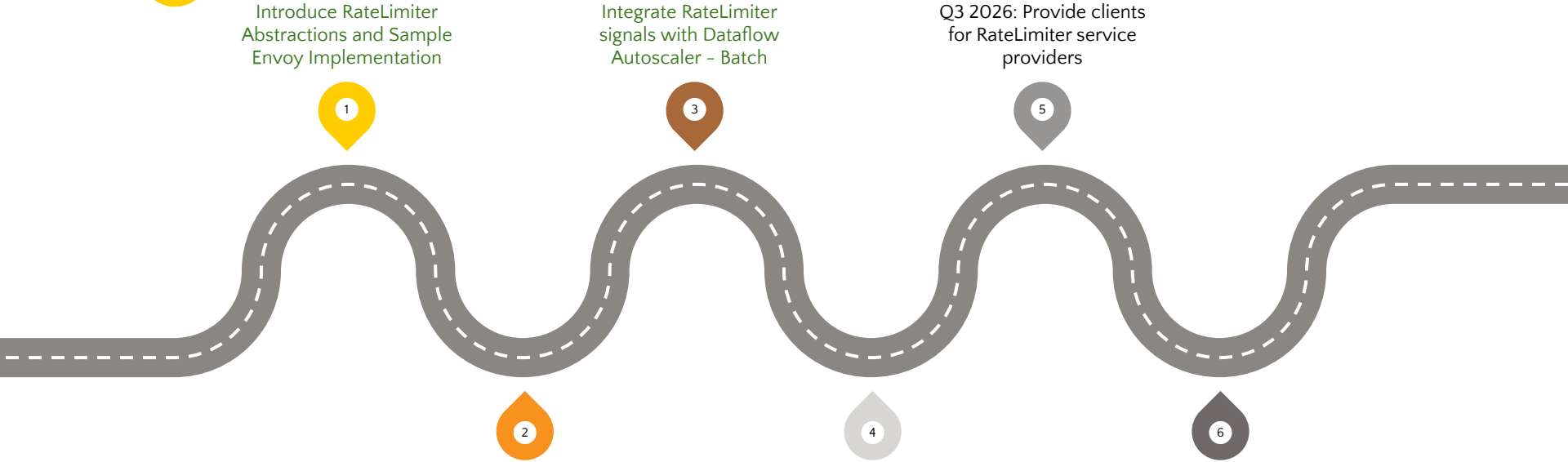
At a high level:

> *If observed throttle time for a stage is non-zero, scale the parallelism of a stage by f.*
> *f=(recent done time -recent throttle time) / recent done time*
> *adjusted stage parallelism=min(f*active threads, desired parallelism)*

recent done time and recent throttle time refer to windowed portions of total done time and total wait time from the past 60 seconds. This window provides visibility into whether prior decisions caused the stage to incur throttling while incorporating only immediately relevant measure of done time and throttle time.

Results: Dataflow RunnerV2

Roadmap


1
Introduce RateLimiter Abstractions and Sample Envoy Implementation
2
Integrate RateLimiter with Well known BeamIO-RunInference
3
Integrate RateLimiter signals with Dataflow Autoscaler - Batch
4
Q3 2026: Integrate RateLimiter signals with Dataflow Autoscaler - Streaming
5
Q3 2026: Provide clients for RateLimiter service providers
6
Q4 2026: Integrate RateLimiter to all available BeamIO

Thanks!

Any questions ?

You can find me at

- LinkedIn: tarun-annapareddy
- tarunannapareddy1997@gmail.com