# Adaptive Checkpoint in Apache Flink

Team 2
Minghui Yang, Yinan An, Yuhe Peng, Junchen Liu, Bhaskar

# Goals & Achievements

# Checkpointing In Flink

Enable Checkpoint in Original Flink Application:

```
// start a checkpoint every 1000 ms
env.enableCheckpointing(1000);
```

Problems:

- The checkpoint interval is **statically** configured and **cannot change** unless the job is halted and restarted.
- **When the rate increases**, the job will accumulate state more quickly, thus, requiring longer to recreate this state during recovery.
- **When the rate drops,** checkpointing frequently will only waste unnecessary resources

# Goal

Design, implement, and evaluate an **automatic checkpoint interval adjustment policy** for Apache Flink.

# Achievement Overview

1. Completed 4 automatic checkpoint interval adjustment policies for Apache Flink in “Simplified-Flink”: including new APIs for users, metrics submission, calculate new interval and reschedule checkpoint.

| ID | Status | Acknowledged | Trigger Time |
|---|---|---|---|
| 3 | COMPLETED | 3/3 | 2022-04-26 19:58:01 |
| 2 | COMPLETED | 3/3 | 2022-04-26 19:54:27 |
| 1 | COMPLETED | 3/3 | 2022-04-26 19:54:17 |

Data from server 20.127.185.109

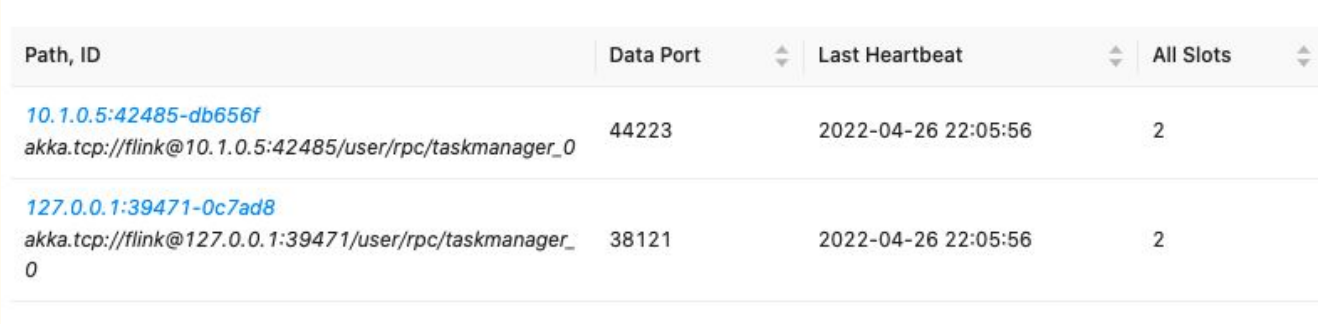
# Achievement Overview

2. Packaged and deployed Simplified-Flink with adaptive checkpoint on servers.

3. Deployed Kafka on a server and injected data with a Flink application.

4. Developed a main Flink Application for experiments.

5. Designed and completed 2 kinds of experiments to evaluate our policies.



| Path, ID | Data Port | Last Heartbeat | All Slots |
|---|---|---|---|
| 10.1.0.5:42485-db656f<br>akka.tcp://flink@10.1.0.5:42485/user/rpc/taskmanager_0 | 44223 | 2022-04-26 22:05:56 | 2 |
| 127.0.0.1:39471-0c7ad8<br>akka.tcp://flink@127.0.0.1:39471/user/rpc/taskmanager_0 | 38121 | 2022-04-26 22:05:56 | 2 |

Deployed on servers

# Artifact

- How users set up a checkpoint adapter in a Flink Application
- Workflow in Simplified-Flink

# Set up a checkpoint adapter

**Prerequisite:** set up periodic checkpoint

**3 steps to set up a Checkpoint Adapter:**

1. Set up maximum tolerable downtime

```
env.enableCheckpointing(3000L);
env.enableCheckpointAdapter(10000L);
```

Downtime: For jobs currently in a failing/recovering situation, the time elapsed during this outage.

# Set up a checkpoint adapter

**3 steps to set up a Checkpoint Adapter:**

2. Set a metrics submission interval:  2 ways for each task to submit metrics

- Submit metrics after completes a checkpoint(default)
- Submit metrics periodically

```
// to set up a checkpoint adapter with metrics submission
public static void main(String[] args){
        env.enableCheckpointAdapter(long recoveryTime);
        /** 1. submit after completing a checkpoint */
        env.setCheckpointAdapterMetricInterval();
        /**
         * 2. submit periodically
         * represent metric submission period will be 10000s
         * */
        env.setCheckpointAdapterMetricInterval(10000L);
}
```

# Set up a checkpoint adapter

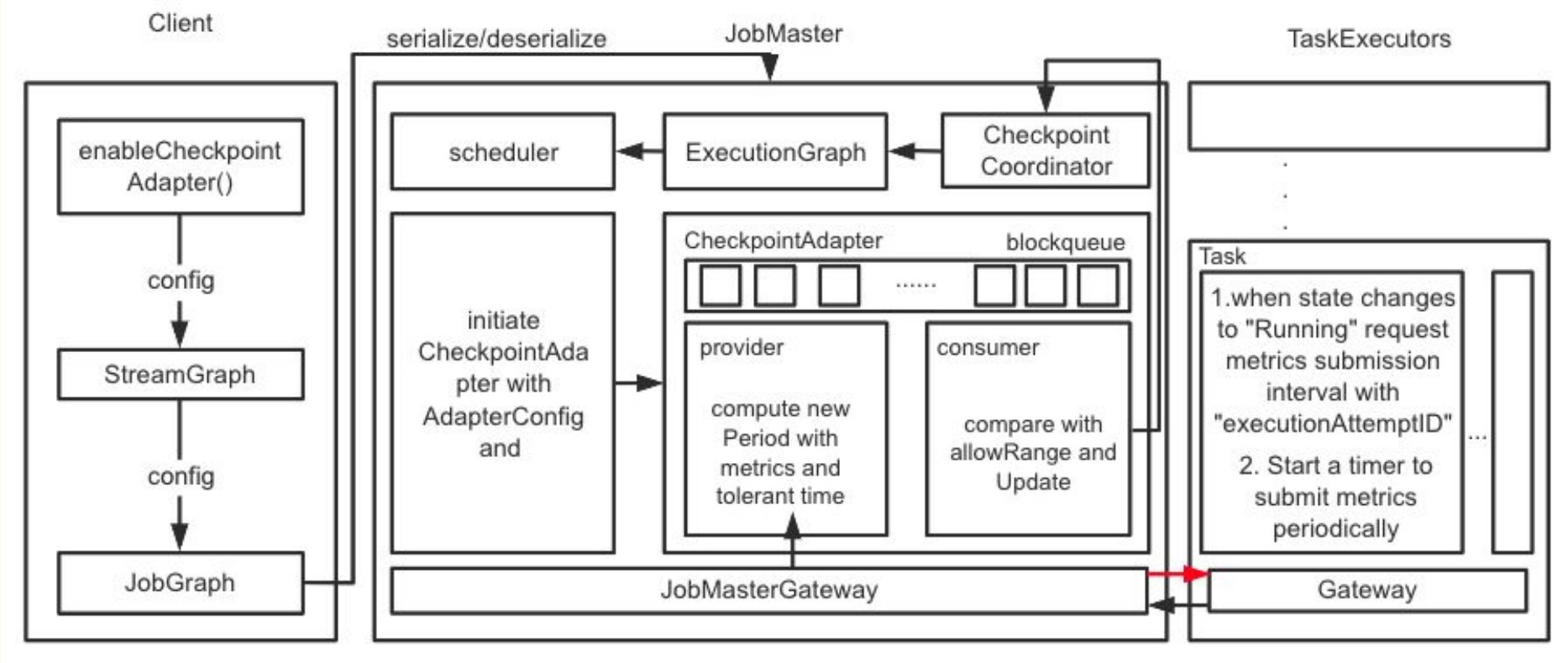
**3 steps to set up a Checkpoint Adapter:**

3. Calculate and update checkpoint interval referring to metrics: 4 strategies

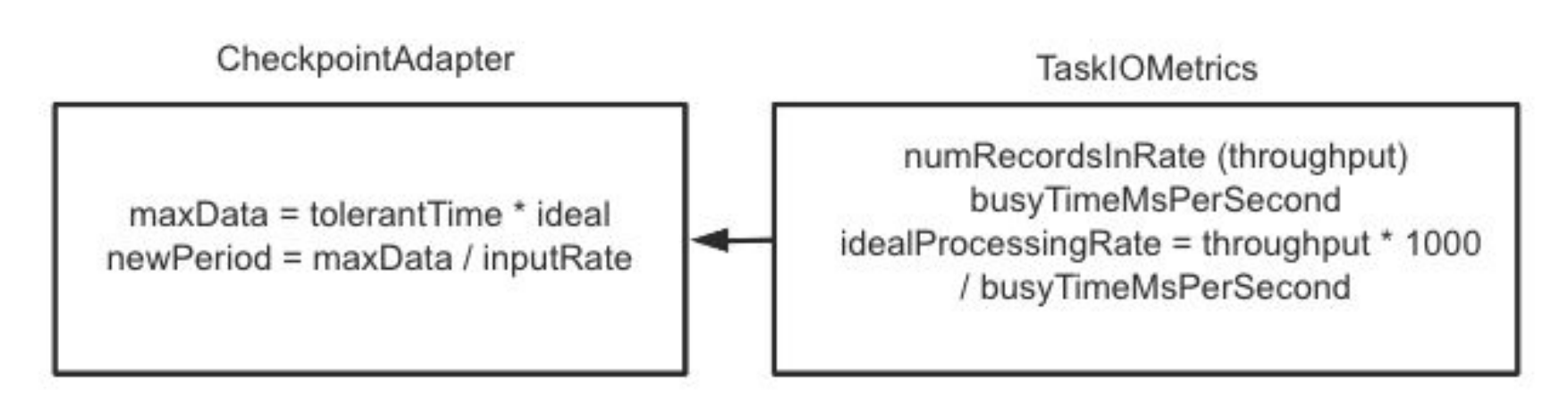
```
public static void main(String[] args){
        env.enableCheckpointAdapter(long recoveryTime);
        env.setCheckpointAdapterAllowRange(0.3) // represent 30%
        env.setCheckpointAdapterChangeInterval(10000L)
        env.setCheckpointAdapterDebounceMode(true)
```

Example: If we only set allowRange, it will update checkpoint interval when the calculated value is below/over allowRange (20%, 40%...) of current interval

# Workflow in Simplified Flink



# Calculation in Simplified Flink

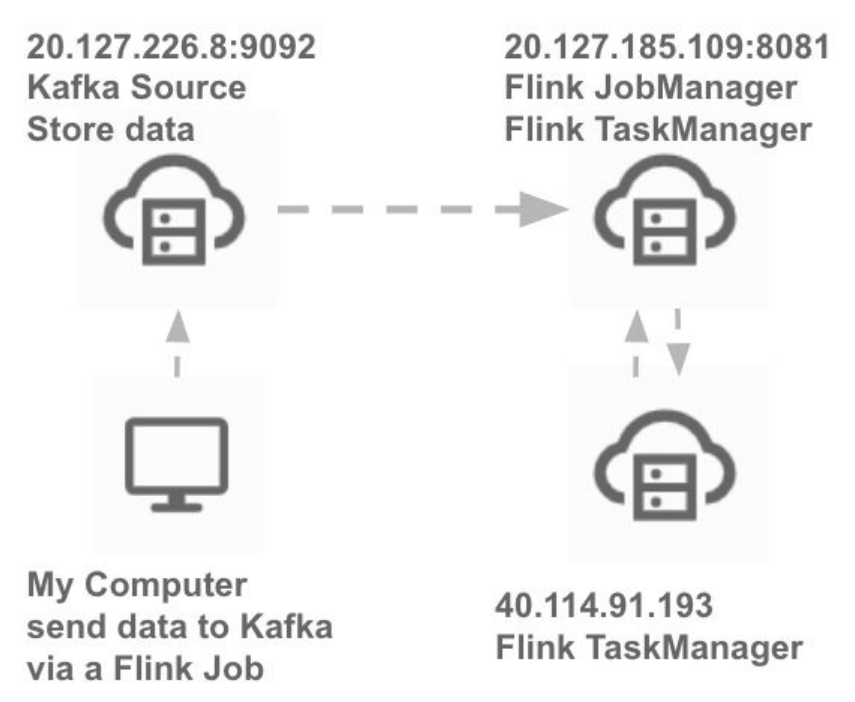


# Experiment

- Experiment plan
- Deploy Simplified-Flink
- Deploy Kafka
- Develop A Job for Experiment

# Experiment - Setup

- **Dataset:** Google cluster data - TaskEvent
- **Platform:** Azure (to emulate real production environment)
- **Main Application Design:**
  - File system as statebackend
  - Kafka Source (Replayable)
  - Large state in operators to ensure we have downtime
  - Latency Calculation



# Exp-1: Evaluate Overhead

## Goal: Find out the cost by adaptive checkpoint.

## Overview:

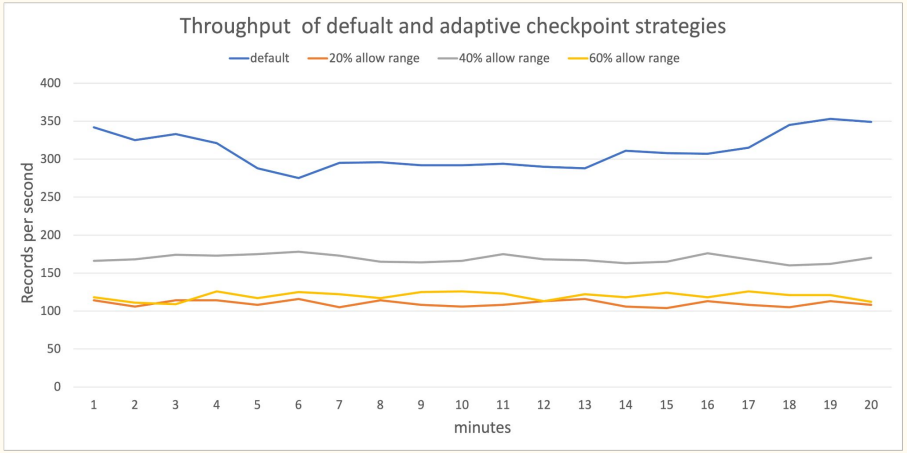
In this experiment, we didn’t vary speed. We used and modified the main application to evaluate the latency and throughput under non-adaptive strategy and under adaptive strategy (only “allowRange” strategy). And then compared the cost of the adaptive strategy.

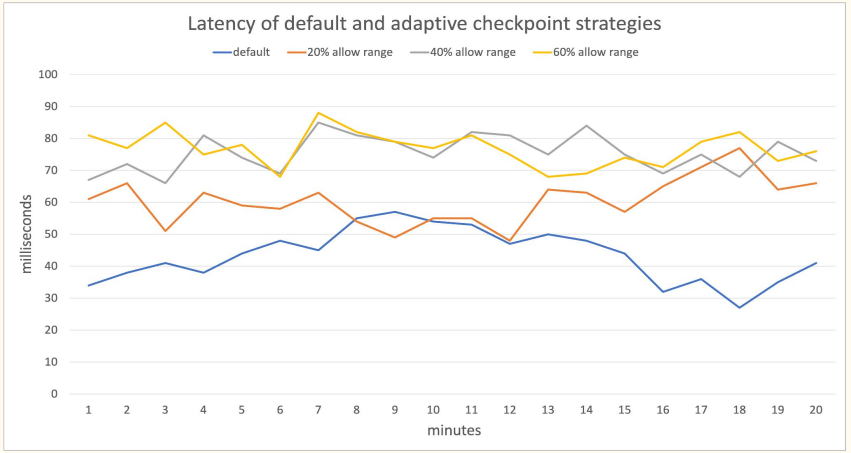
## Process:

- Default checkpoint interval: 10000 ms
- Default recovery time: 30000 ms

# Exp-1: Evaluate Overhead

Change “allowRange” and keep “Metrics Submission Interval” Constant:

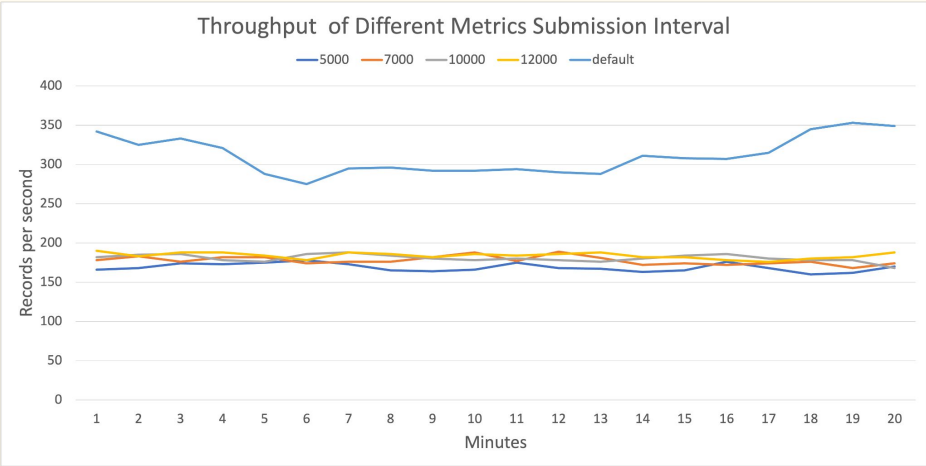


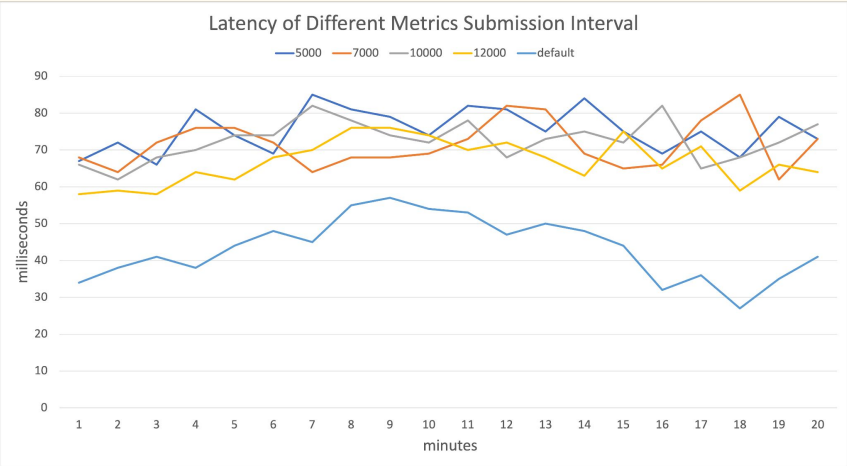




1. Adaptive Checkpoint adds additional overhead during metrics submission and reschedule Checkpoint.
2. Although “allowRange” changed, it did not change much when keep “Metrics Submission Interval” constant, indicating that the metrics submission may have greater impact on the throughput/latency.

# Exp-1: Evaluate Overhead

Change metrics submission interval, allowRange: 40%




From experiment 1, we got the conclusion that using Adaptive Checkpoint has smaller throughput, so we supposed that the more frequently the metrics is submitted, the smaller the throughput will be. However, no obvious trend can be seen in the data we obtained:

1. Probably because the metrics submission intervals we chose were not very representative.
2. We assume that the network speed is constant, but it actually fluctuates as data is retrieved from Kafka. Overhead does not directly affect throughput/latency as much as network speed

# Exp-2: Evaluate Efficiency

Goal: Find out in which cases adaptive checkpointing may be preferable to Flink's default approach.

## Overview:

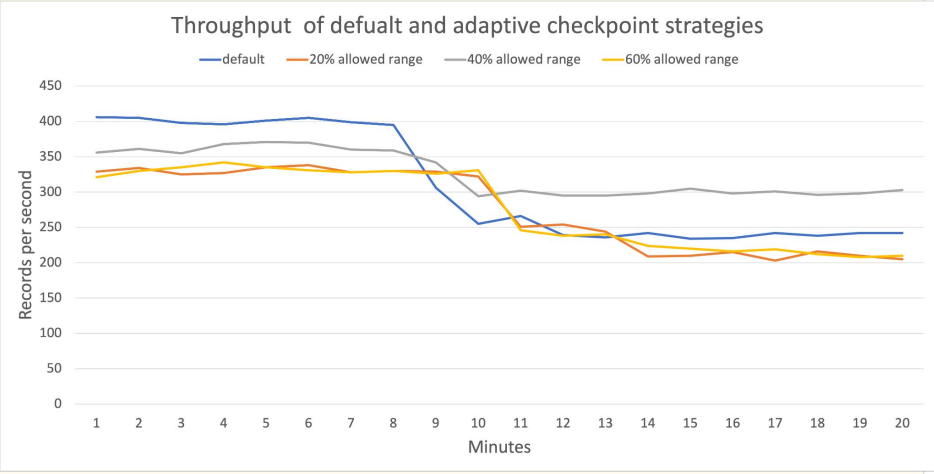
In this experiment, we varied speed, crashed and recovered the application. We used and modified the main application to evaluate the latency, throughput, checkpoint time, and downtime under non-adaptive strategy and under allowRange strategy. And then found out in which cases adaptive checkpointing may be preferable to Flink's default approach.

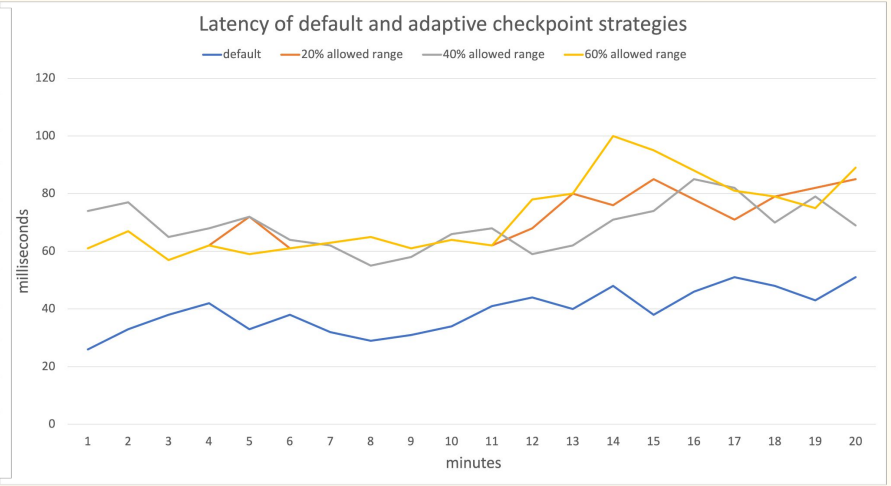
## Process:

- Default checkpoint interval: 10000 ms, default recovery time: 30000 ms
- Slow down to half of the original speed after 30000 reocords
- Crash 1 TaskManager at around 6th minute, and crash 1 TaskManager at the same time at around 6th minute to collect downtime

# Exp-2: Evaluate Efficiency

Change “allowRange” and keep “Metrics Submission Interval” Constant:

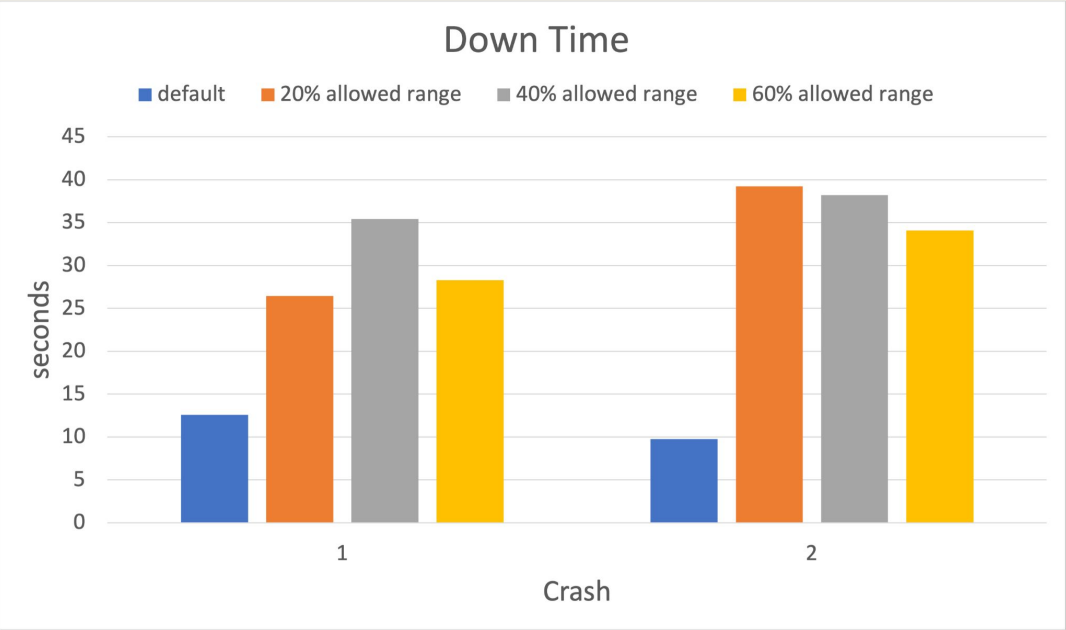



1. Comparing **the grey line(40%) and the blue line**, when the speed is reduced, the throughput of the Adaptive is better, because the checkpoint interval is extended and overhead caused by checkpoint is reduced.
2. In other cases, the adaptive case has no obvious advantage, probably because the overhead brought by the adaptive counteracts the benefit of checkpoint reduction.

# Exp-2: Evaluate Efficiency

Tolerant time: 30 sec




We automatically adjust checkpoint interval to be as close as the user-set value to reduce checkpoint overhead. Our strategy succeeded in keeping the value in this range.

# Conclusion

1. Adaptive Checkpoint will add overhead

2. When input rate of stream data changes greatly, the adaptive strategy is better than Original one, because the advantage of reduced checkpoint overhead is greater than the disadvantage of additional overhead caused by the adaptive strategy.

# Challenges

# Challenge

1. Spent a long time to find an appropriate timing to inform each Task of the metrics submission interval.
2. Spent a long time to build Jobs and custom Flink.  There was a build problem with kafka-connector.
3. Deploy on server. It took us a long time to realize that low performance machines couldn't run Flink. No experience to deploy Flink on servers.
4. Debug on servers. There are some bugs you can never find locally and it’s hard to debug. There was a RPC error. We had to change code in Simplified-Flink, rebuild it in the server and restart the cluster to see if we fix it.
5. Get Latency Metrics: “LatencyMarker” didn’t work with our kafka-connector. Solve this problem by attaching a timestamp with Event