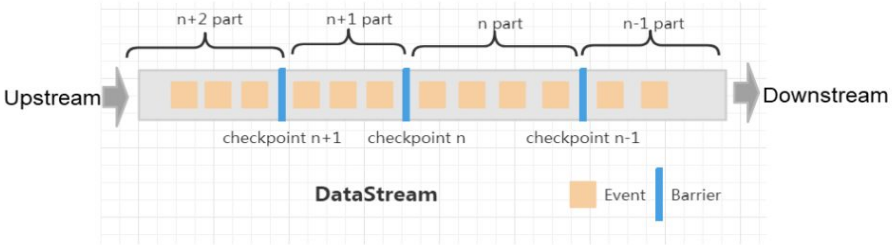
Adaptive Checkpointing in Apache Flink

By: Dominic Maglione, Arkash Jain, Qinfeng(Frank) Li, Dayu (Jasper) Li, Yifei Zhou

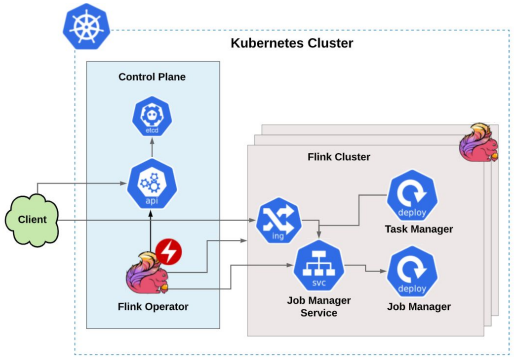
Goals & Achievements

We finished writing all the queries, have docs for setting up a successful cluster and ran multiple experiments.

- Challenges faced especially in variable throughput environment.
- Need for an adaptive checkpointing strategy, particularly where throughput is variable.



NEXM



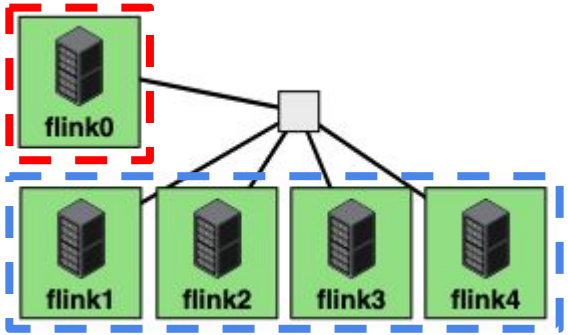
Goals

- Write Queries to test out checkpointing in Flink under varied scenarios
- Set-up a fully automated environment to run & test experiments on a cluster
- Run experiments with varied parallelism and checkpointing intervals

Cluster Setup

For this project, we utilized CloudLab, employing a Flink cluster profile.

- The cluster consisted of **4 Worker Nodes** and **1 Master Node**.
- The **Master Node** contained Flink, Kafka, Hadoop, etc.
- The **Worker Nodes** contained respective configuration files.



Each node was a `c220g2` machine with 10Gbps of bandwidth, running Ubuntu 20.04.

c220g2	163 nodes (Haswell, 20 core, 3 disks)
CPU	Two Intel E5-2660 v3 10-core CPUs at 2.60 GHz (Haswell EP)
RAM	160GB ECC Memory (10x 16 GB DDR4 2133 MHz dual rank RDIMMs)
Disk	One Intel DC S3500 480 GB 6G SATA SSDs
Disk	Two 1.2 TB 10K RPM 6G SAS SFF HDDs
NIC	Dual-port Intel X520 10Gb NIC (PCIe v3.0, 8 lanes) (both ports available for experiment use)
NIC	Onboard Intel i350 1Gb

On the Master Node an SSH key was generated to allow connection between itself and the Worker Nodes...

```
$ ssh-keygen -t rsa -b 4096 && eval "$(ssh-agent -s)" && ssh-add ~/.ssh/id_rsa
```

System Configuration

A separate `flink` user account was created for collaboration and provided `sudo` privileges.

```bash
# Create the user (follow prompts)
$ sudo adduser <user>

# Add user to Sudoers group
$ sudo adduser <user> sudo

# Login to user
$ su -l <user>
```

Additional storage was set up by mounting a 1.1TB partition to the `/home/flink/HDD` directory.

```bash
$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0   1.1T  0 disk
├─sda1   8:1    0    16G  0 part /
├─sda2   8:2    0     3G  0 part
├─sda3   8:3    0     3G  0 part [SWAP]
└─sda4   8:4    0   1.1T  0 part /home/flink/HDD
sdb      8:16   0   1.1T  0 disk
sdc      8:32   0 447.1G  0 disk
```

Each machine was secured with the Uncomplicated Firewall (`ufw`), allowing specific IP addresses.

"One cannot be prepared for something while secretly believing it will not happen." – Nelson Mandela

Software Setup

The project aimed to improve Flink's checkpointing system, by first upgrading from v1.14 to v1.18.

- While experiments were ran using Flink 1.18 binaries, our modified source code can be found here.

Hadoop was installed, and the `HADOOP_HOME` environment variable was set up.

- `flink-shaded-hadoop-2-uber-2.8.3-10.0.jar` file MUST be added to `/lib`.
- Hadoop configuration files, such as `core-site.xml` and `hdfs-site.xml`.

Kafka was installed and configured, with optional `systemd` service files for automatic startup.

- Despite best practice Kafka was kept in the `$HOME` directory.
- Note that Zookeeper must be started before Kafka in order to work.

NEXMark Benchmark

The benchmark simulates streams of data including joins and partitions.

```xml
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>3.0.0-M5</version>
    <configuration>
        <reportsDirectory>${project.build.directory}/surefire-
        <includes>
            <include>**/*Test.java</include>
            <include>**/*Tests.java</include>
            <include>**/testQuery3.java</include>
            <include>**/testQuery4.java</include>
            <include>**/testQuery6Stateful.java</include>
            <include>**/testQuery7.java</include>
        </includes>
    </configuration>
</plugin>
```

- Install the Surefire Library &
- Add the Junit Dependency

Have the environment set-up and write sample tests

```java
@Test
public void testHighestBidCalculation() throws Exception {
    StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironme
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
    env.enableCheckpointing( interval: 1000, CheckpointingMode.EXACTLY_ONCE);

    ParameterTool params = ParameterTool.fromArgs(new String[]{});
    env.setParallelism(1);
    env.getConfig().setAutoWatermarkInterval(1000);

    List<Bid> bids = new ArrayList<>();
    bids.add(new Bid( auction: 1,  bidder: 1,  price: 10, System.currentTimeMillis(),  extra:
    bids.add(new Bid( auction: 1,  bidder: 2,  price: 20, System.currentTimeMillis(),  extra:
    bids.add(new Bid( auction: 1,  bidder: 3,  price: 30, System.currentTimeMillis(),  extra:
    bids.add(new Bid( auction: 1,  bidder: 4,  price: 40,  dateTime: System.currentTimeMillis
```

mvn test --am -T 4 -up -fn

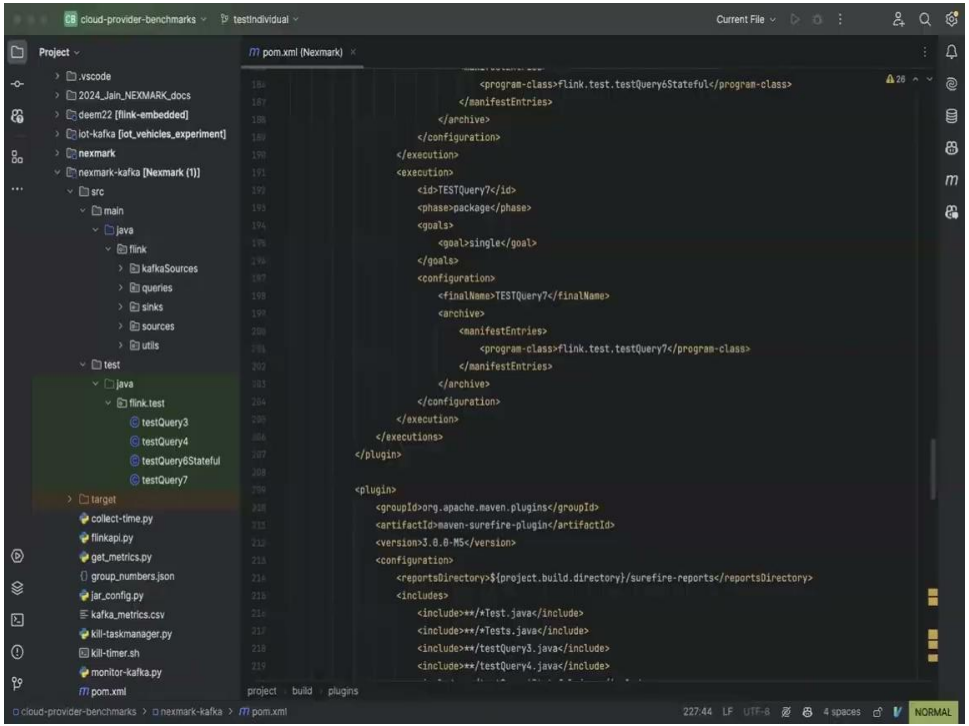
```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  24.526 s (Wall Clock)
[INFO] Finished at: 2024-04-24T22:35:59-04:00
[INFO] ------------------------------------------------------------------------
```

Run a test command to execute Queries 3, 4, 5, 6

NEXMark Benchmark

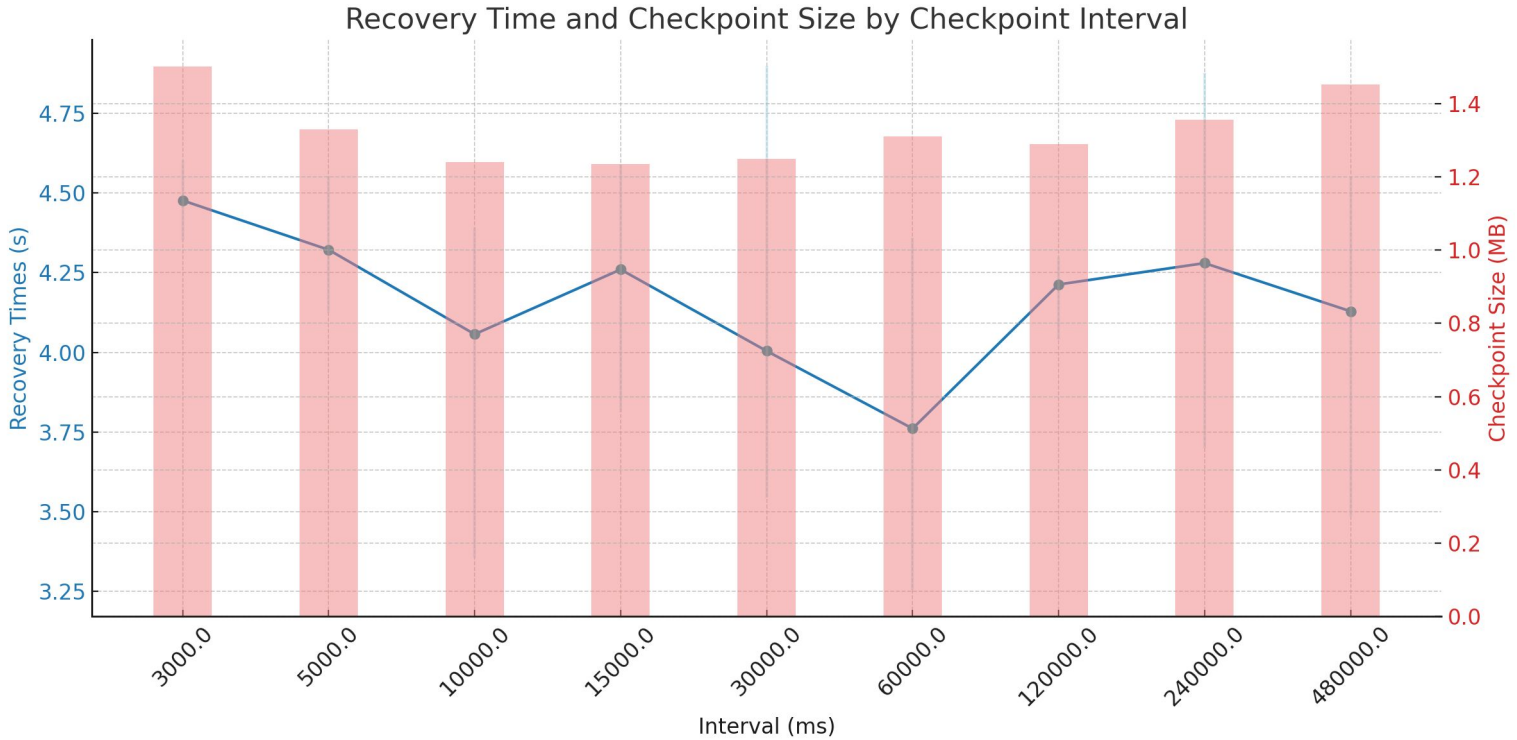
Our contribution includes writing 3 queries, extensive tests and rewriting existing queries

- **Query 3** - Joining 2 streams of data with where conditions on category, keyed on auctions and sellers and state.
- **Query 4** - Nested query which after getting the maximum price, it joins the category file to get average closing price.
- **Query 5** - Item with the most bids in the past one hour time period; the “hottest” item.
- **Query 6** - For each seller, the average selling price of items sold by that seller.
- **Query 7** - monitors the highest price items currently on auction and return it every ten minutes.



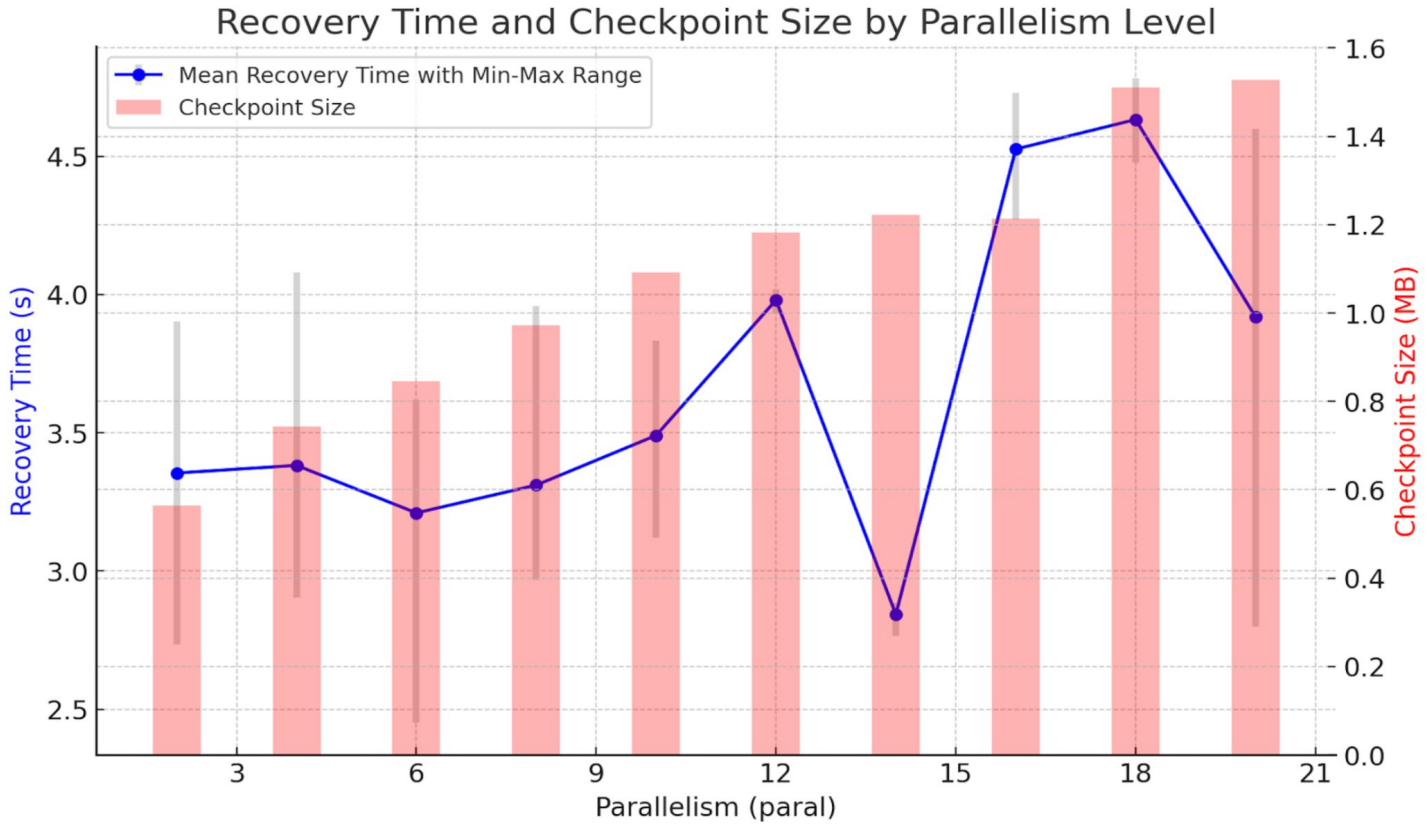
Experiments - intervals

We ran Queries 7 on a distributed cluster with Hadoop enabled and constant ingestion from Kafka streams



Query_7, interval: [3s, 5s, 10s 15s, 30s, 1min, 2min, 4min, 8min], parallelism = 16, 10 mins before failure injection, rate: 10000 per sec

Experiments - parallelism



Query_7, parallelism: [2, 4, 6, 8, 10, 12 ,14, 16, 18, 20], interval = 20s, 10 mins before failure injection

Challenges

Setting up the cluster and parsing the Flink Source code was a huge bottleneck

- Arkash
  - Scarce resources online for NEXMark queries. Hours spent on the flink website reading examples and docs, and office hours.
  - Old queries were flawed which caused a lot of confusion and AI was not helpful.
  - Flink source were extremely dense and understanding recovery required reading multiple internal tickets

- Qinfeng
  - Having a lot of problems setting up multiple nodes on Cloudlab due to version issues (Flink, Kafka, Hadoop).
  - It has taken weeks to sort out all the problems before we were able to run experiments on Cloudlab.
  - Local machine very different than Cloudlab cluster.

- Dominic
  - Lack of existing motivation, documentation, and messy code existed in the **Adaptive-Checkpointing** repository.
  - Multiple different libraries cause problems with each other with **no clear guidance** on how to solve them.
    - Spent hours reading **Hadoop** documentation to determine what was necessary for our use case.
  - Spent too much time hardening the cluster so the system didn't implode on itself and cost us time.

- Dayu
  - Encountered many problems while migrating queries from local to the cluster and hard to find answers.
  - Need rigorous experimental logic and have to ask the professor and TA multiple times.