Good morning!

CS 744: DATAFLOW

Shivaram Venkataraman
Fall 2021
- Assignment 2 grades are up! → Yhem Xu, TA
- Midterm grading in progress → next week
- [Course project proposal comments]
- Mid-Semester feedback (next slide)
“Reading papers before lecture on that paper. I struggle to understand and in the class I get some knowledge”

“Learning from papers/understanding them is hard for someone like me who hasn't read a lot of papers”

“The in-class discussion times are very short”

“Around 80-85 mins for the lecture and 20 mins for the discussion.”

“More discussion on applying ideas from paper to different settings”

“not enough time to even think for each question, it felt very congested”

“Smaller (perhaps bi-weekly) quizzes, take home exams with a partner, … solo, etc.”
DATAFLOW MODEL (?)

What is a streaming workload and desirable properties?
MOTIVATION

Streaming Video Provider

- How much to bill each advertiser?
- Need per-user, per-video viewing sessions
- Handle out of order data

Goals

- Easy to program → API design
- Balance correctness, latency and cost

L0 inputs that are used for a given output
**APPROACH**

Separate user API from execution

Decompose queries into

- What is being computed
- Where in time is it computed
- When is it materialized
- How does it relate to earlier results

Data keeps arriving

$O_1 \xrightarrow{\text{SQL query or MapReduce/Spark job}} \text{Output}$

$O_2$
STREAMING VS. BATCH

Streaming
- Event-based system
-as data arrives

Batch
- Wait for time t, accumulate data that arrives
- Trigger batch computation → Spark

Spark Streaming

Spark

Flink

Wait

periodically emit output

Spark

$t_1$ $t_2$
TIMESTAMPS

Event time:
Time at which an event took place
e.g., time at which video was watched

Processing time:
Time at which an event was processed by the system
e.g., time at which video watch event is processed by system
Windowing

Key 2 video-id or user-id

aligned windows

All events for a user within 5 min range = session

Window/bucket

every 1s

t = 0

t = 1

t = 2

t = 3

Fixed

Sliding

Sessions

key 1 key 2 key 3

key 1 key 2 key 3

key 1 key 2 key 3

Tumbling

t = 0 to 3

t = 1 to 4

t = 2 to 5...

aligned

unaligned
System has processed all events up to 12:02:30

- Watermark varies over time
- Hard to know!

Actual watermark:
Ideal watermark:
Event Time Skew:
Processing time =
Event time =
API

ParDo: \[\rightarrow \text{Map in MapReduce / Spark}\]

GroupByKey: \[\rightarrow \text{Very similar in MR / Spark / SCOPE}\]

Windowing
  AssignWindow \[\rightarrow \text{Take a key and put it in a window}\]
  MergeWindow \[\rightarrow \text{merge adjacent windows (session semantics)}\]
\[(k_1, v_1, 13:02, [0, \infty))\),
\[(k_2, v_2, 13:14, [0, \infty))\),
\[(k_1, v_3, 13:57, [0, \infty))\),
\[(k_1, v_4, 13:20, [0, \infty))\)

\[\text{AssignWindows}(\text{Sessions(30m)})\]
\[(k_1, [v_1, 13:02, 13:32)),
(k_2, [v_2, 13:14, 13:44)),
(k_1, [v_3, 13:57, 14:27)),
(k_1, [v_4, 13:20, 13:50))\)

\[\text{DropTimestamps}\]
\[(k_1, [v_1, 13:02, 13:32)),
(k_2, [v_2, 13:14, 13:44)),
(k_1, [v_3, 13:57, 14:27]),
(k_1, [v_4, 13:20, 13:50))\)

\[\text{GroupByKey}\]
\[(k_1, [(v_1, 13:02, 13:32)),
(v_3, [13:57, 14:27)),
(v_4, [13:20, 13:50)),
(k_2, [(v_2, 13:14, 13:44))\)

\[\text{GroupAlsoByWindow}\]
\[(k_1, [((v_1, v_4), 13:02, 13:50)),
([v_3], [13:57, 14:27))\),
(k_2, [[v_2], [13:14, 13:44)])\)

\[\text{ExpandToElements}\]
\[(k_1, [v_1, v_4, 13:50, [13:02, 13:50)),
(k_1, [v_3, 14:27, [13:57, 14:27]),
(k_2, [v_2, 13:44, [13:14, 13:44])\)
TRIGGERS AND INCREMENTAL PROCESSING

Windowing: where in event time are data grouped
Triggering: when in processing time are groups emitted

Strategies
- Discarding = 6 (discard 5)
- Accumulating = 11 (5+6)
- Accumulating & Retracting

\[\begin{align*}
-5 & \quad \downarrow \\
\text{retireck} & \quad \downarrow \\
\text{accumulate} & \\
\text{accumulate} & \quad 11
\end{align*}\]
RUNNING EXAMPLE

PCollection<KV<String, Integer>> input = IO.read(...);
PCollection<KV<String, Integer>> output =
   input.apply(Sum.integersPerKey());

→ sum of values for one key
GLOBAL WINDOWS, ACCUMULATE

PCollection<KV<String, Integer>> output = input
    .apply(Window.trigger(Repeat(AtPeriod(1, MINUTE)))
        .accumulating())
    .apply(Sum.integersPerKey());
PCollection<KV<String, Integer>> output = input
  .apply(Window.trigger(Repeat(AtCount(2))))
  .discarding()
  .apply(Sum.integersPerKey());
PCollection<KV<String, Integer>> output = input
   .apply(Window.into(FixedWindows.of(2, MINUTES)))
   .trigger(Repeat(AtWatermark()))
   .accumulating()
SUMMARY/LESSONS

Design for unbounded data: Don’t rely on completeness
Be flexible, diverse use cases
  - Billing
  - Recommendation
  - Anomaly detection

Windowing, Trigger API to simplify programming on unbounded data
DISCUSSION

https://forms.gle/Yuvk4SfFoHyy4Et36
Outputs for windows are mostly produced once except for 9.

Latency could be higher for results.

 Streaming: 9 is handled as soon as it arrives.
Consider you are implementing a micro-batch streaming API on top of Apache Spark. What are some of the bottlenecks/challenges you might have in building such a system?
NEXT STEPS

Next class: Naiad
Course project proposal feedback