Developing and Operating Real-Time Applications at Tencent

Xiaogang SHI
robbieshi@tencent.com

Outline

PART I
Background

PART II
Introduction to Oceanus

PART III
Improvement to Flink

About Tencent

Tencent 腾讯

SOCIAL NETWORK

- The largest social communities in China in terms of MAU
- Weixin & Wechat: 1 billion MAU
- QQ: 807 million MAU

FINTECH

- The leading mobile payment platform by active users and transactions in China.
- 900 million MAU
- 1 billion transactions daily

MEDIA

VIDEO

- The leading online video streaming platform in China in terms of mobile DAU and subscriptions
- 89 million subscriptions
- Up 58% year-over-year

MUSIC

- The largest online music platform in China
- 800 million MAU

NEWS

- The top mobile news app by active users
- 150 million MAU

GAMES

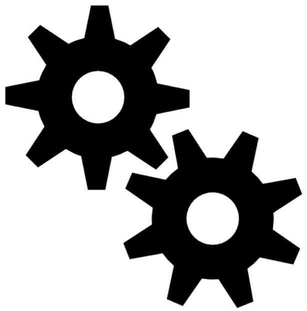
- The largest gaming company in the world by active users and revenue
- 200 million MAU

CLOUD

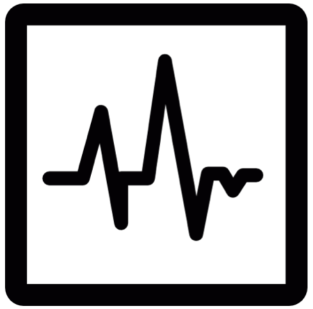
- Covered 25 regions and operated 53 availability zones

https://www.tencent.com/en-us/articles/8003551553167294.pdf

Real-Time Applications at Tencent



ETL



Monitoring

Real-Time BI

Online Learning

210 Million

Maximum number of messages received per second

17 Trillion

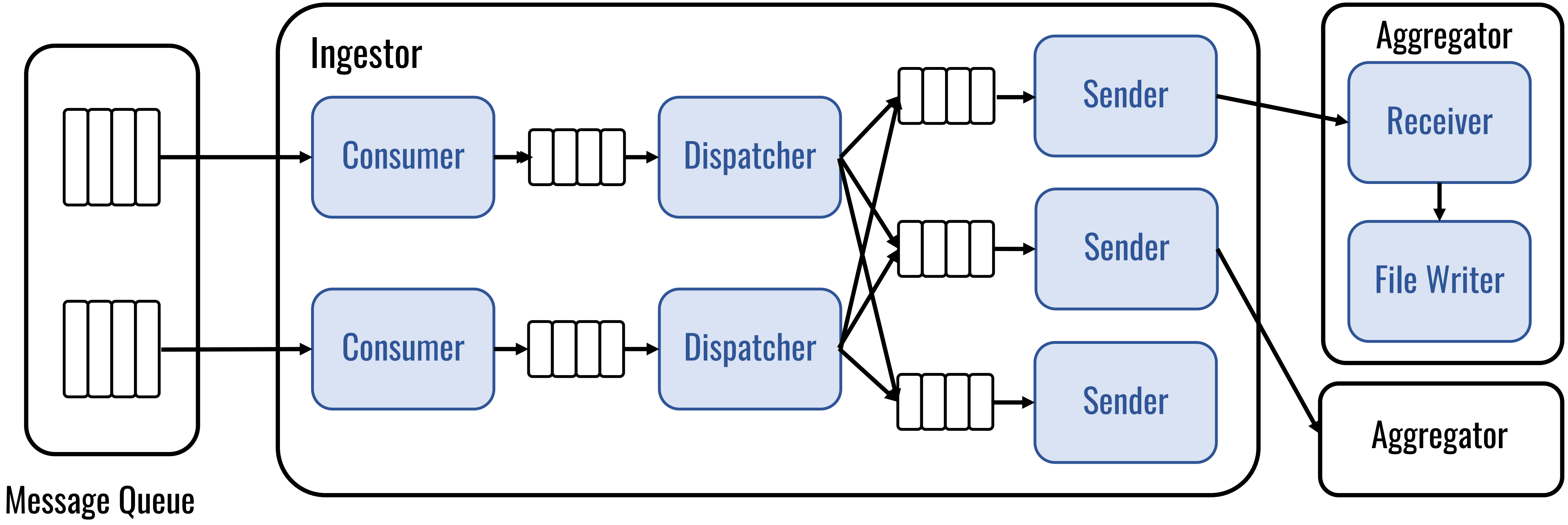
Number of messages received per day

3 PB

Amount of data received per day

Why Flink?

An ETL pipeline as an example



It's very difficult to improve performance while ensuring correctness.

Flink's Strength

Efficient support for states
Automatic distribution while rescaling

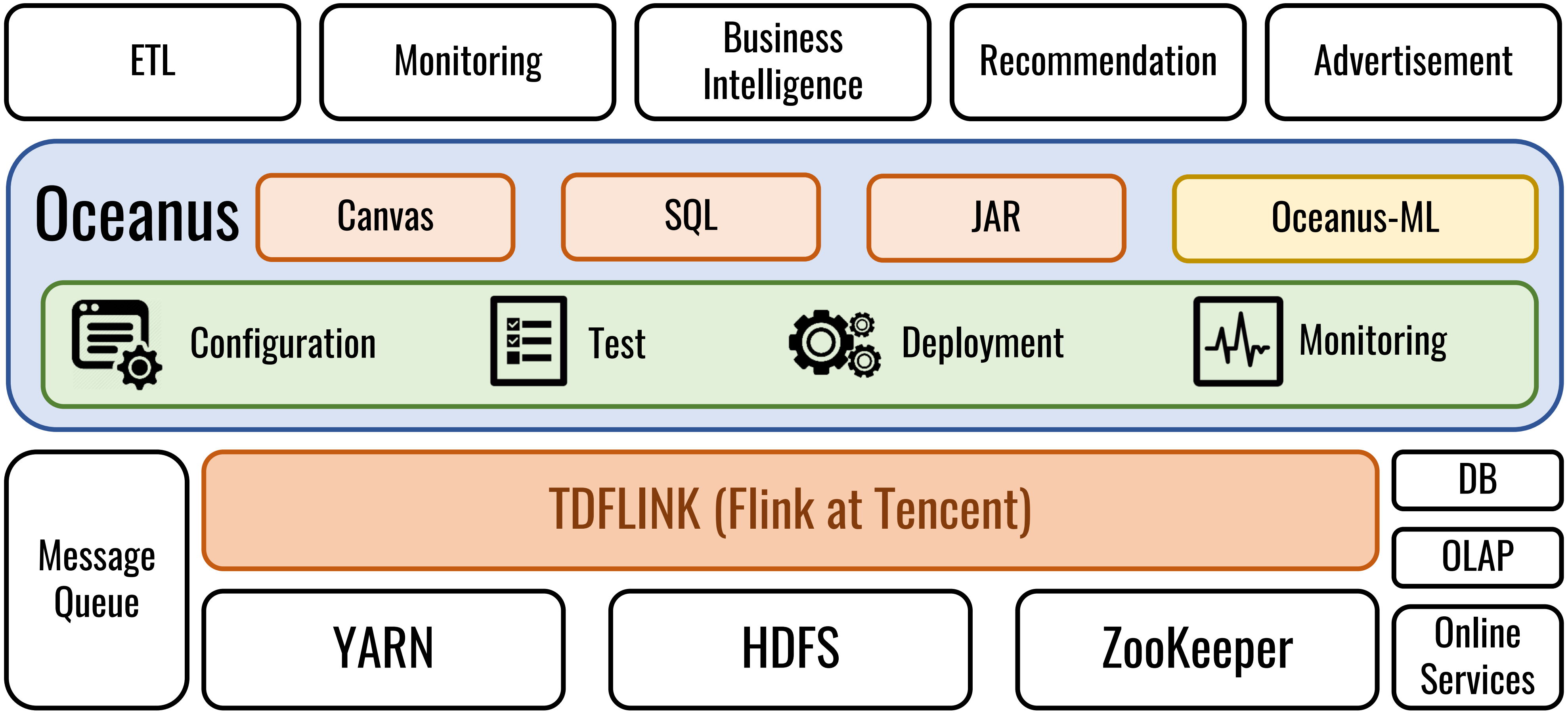
AT-LEAST and EXACTLY-ONCE guarantees while tolerating failures

Flexible and powerful programming interfaces for stream processing

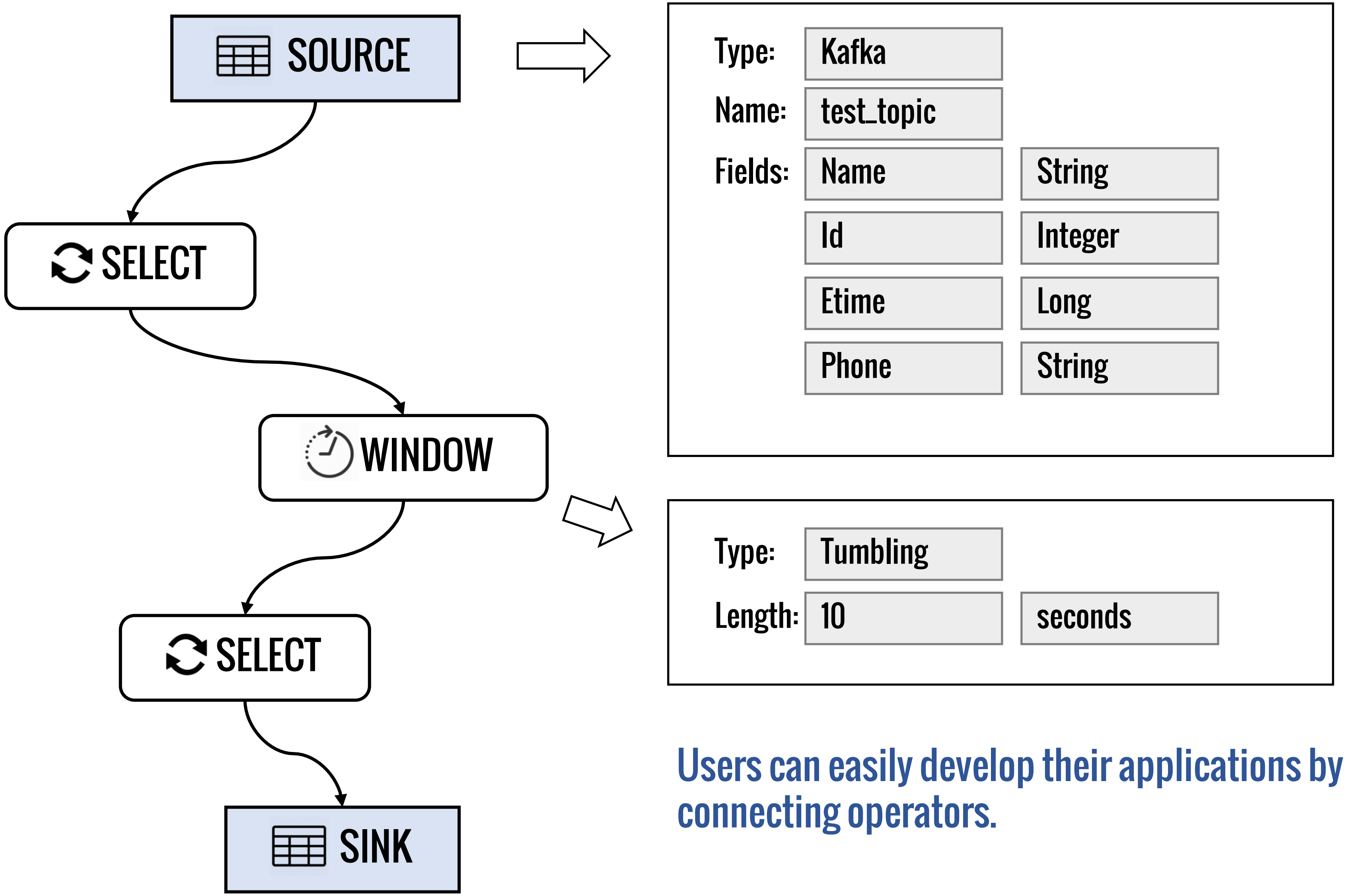
Low latency and high throughput

Oceanus Overview

A unified platform to develop and operate real-time applications

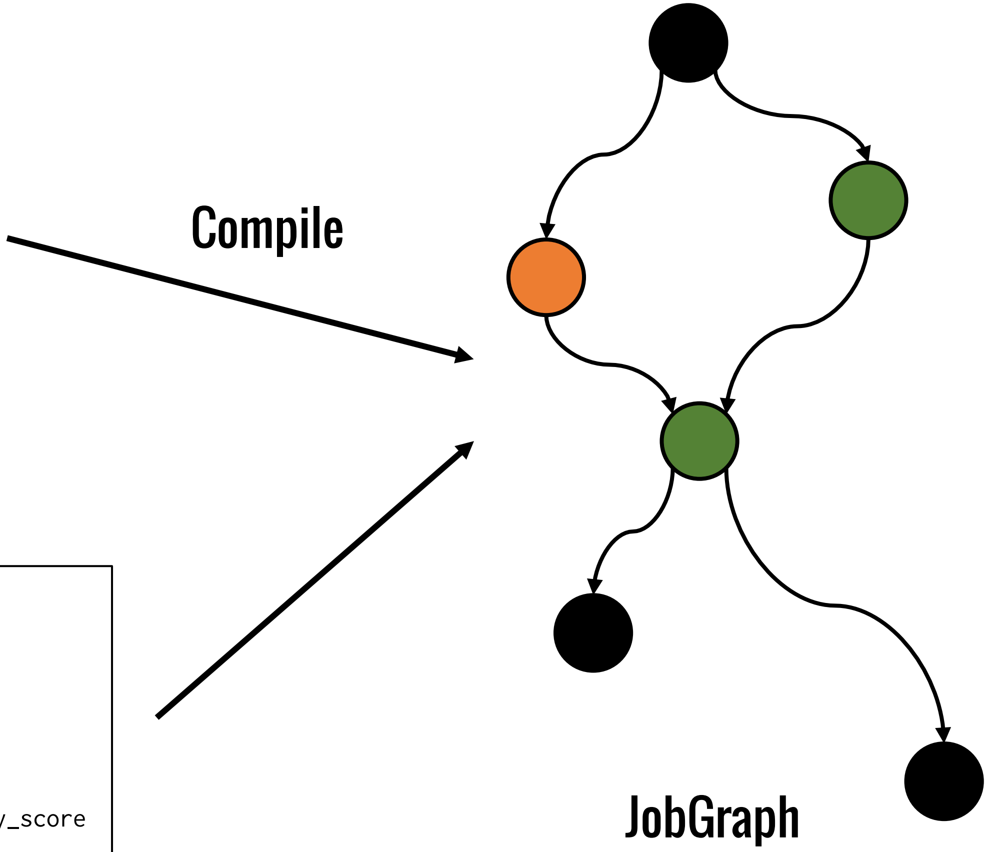


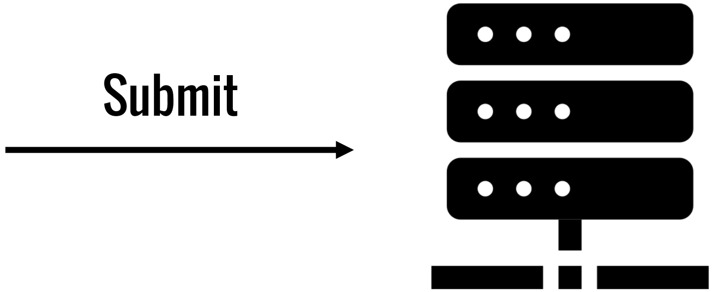
Developing with Oceanus



Users can easily develop their applications by dragging and connecting operators.

Developing with Oceanus






```
INSERT INTO uni_hot_stat (
    song_id,
    batch_time,
    qy_play_score
)
SELECT
    song_id,
    sum(play_scorE) AS qy_play_score
FROM kfk_u002
GROUP BY song_id
```

Jobs are managed with refined Cluster Clients.

Users can easily configure their applications with visualized job graphs.

Operating with Oceanus

Improve operating efficiency with rich metrics.

Job Details

Vertices | **Metrics (Records)** | Metrics (Bytes)

ID	Name	Watermark (max)	InQueue Usage (max)	OutQueue Usage (max)	Input Records (sum)	Output Records (sum)	Input Rate (sum)	Output Rate (sum)
0	Source: Collection Source bc764cd8ddf7a0cff126f51c16239658	N/A	0.00%	100.00%	0	3,425,953,507,895	0	3,527,340
1	Map -> Sink: Unnamed 20ba6b65f97481d5570070de90e4e791	N/A	0.00%	0.00%	3,425,953,469,803	0	3,527,350	0

1. An operator is bottleneck if its in-queue is full while its out-queue is not full.
2. The ratio between the throughput of different operators remain roughly the same when the parallelism changes. We can utilize this property to configure the parallelism (Don’t work well with window operators).
3. There may exist data skew when the difference between the maximum and the minimum throughput is very large.

Operating with Oceanus

Much information can be obtained from thread stacks.

Checkpoint timeout: lock unreleased by blocked user functions, slow hdfs writes, blocked user checkpoint functions
Performance issues

Metrics | **Threads** | Log | Stdout

ID	Name	CPU ↓	State	Stack
70	Map -> Sink: Unnamed (1/1)	99.77%	RUNNABLE	org.apache.flink.streaming.api.operators.AbstractStreamOperator.setKeyContextElement(AbstractStreamOperator.java:625 org.apache.flink.streaming.api.operators.AbstractStreamOperator.setKeyContextElement1(AbstractStreamOperator.java:61 org.apache.flink.streaming.runtime.io.StreamInputProcessor.processInput(StreamInputProcessor.java:201) org.apache.flink.streaming.runtime.tasks.OneInputStreamTask.run(OneInputStreamTask.java:107) org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:301) org.apache.flink.runtime.taskmanager.Task.run(Task.java:721) java.lang.Thread.run(Thread.java:748)
68	Source: Collection Source (1/1)	80.32%	RUNNABLE	org.apache.flink.runtime.state.KeyGroupRangeAssignment.assignToKeyGroup(KeyGroupRangeAssignment.java:59) org.apache.flink.runtime.state.KeyGroupRangeAssignment.assignKeyToParallelOperator(KeyGroupRangeAssignment.java:48) org.apache.flink.streaming.runtime.partitioner.KeyGroupStreamPartitioner.selectChannels(KeyGroupStreamPartitioner.ja org.apache.flink.streaming.runtime.partitioner.KeyGroupStreamPartitioner.selectChannels(KeyGroupStreamPartitioner.ja org.apache.flink.runtime.io.network.api.writer.RecordWriter.emit(RecordWriter.java:101) org.apache.flink.streaming.runtime.io.StreamRecordWriter.emit(StreamRecordWriter.java:81) org.apache.flink.streaming.runtime.io.RecordWriterOutput.pushToRecordWriter(RecordWriterOutput.java:107) org.apache.flink.streaming.runtime.io.RecordWriterOutput.collect(RecordWriterOutput.java:89) org.apache.flink.streaming.runtime.io.RecordWriterOutput.collect(RecordWriterOutput.java:45) org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:7 org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:7 org.apache.flink.streaming.api.operators.StreamSourceContexts$NonTimestampContext.collect(StreamSourceContexts.java: org.apache.flink.streaming.api.functions.source.FromIteratorFunction.run(FromIteratorFunction.java:43) org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:94) org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:58) org.apache.flink.streaming.runtime.tasks.SourceStreamTask.run(SourceStreamTask.java:99) org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:301) org.apache.flink.runtime.taskmanager.Task.run(Task.java:721) java.lang.Thread.run(Thread.java:748)
62	SystemResourcesCounter probing thread	0.19%	TIMED_WAITING	java.lang.Thread.sleep(Native Method) org.apache.flink.runtime.metrics.util.SystemResourcesCounter.run(SystemResourcesCounter.java:128)

Improvement to Flink

Job Management

- Avoid split-brain with Zookeeper transactions
- Non-disruptive recovery of job masters
- Fine-grained recovery of tasks with cached result partitions

Resource Management

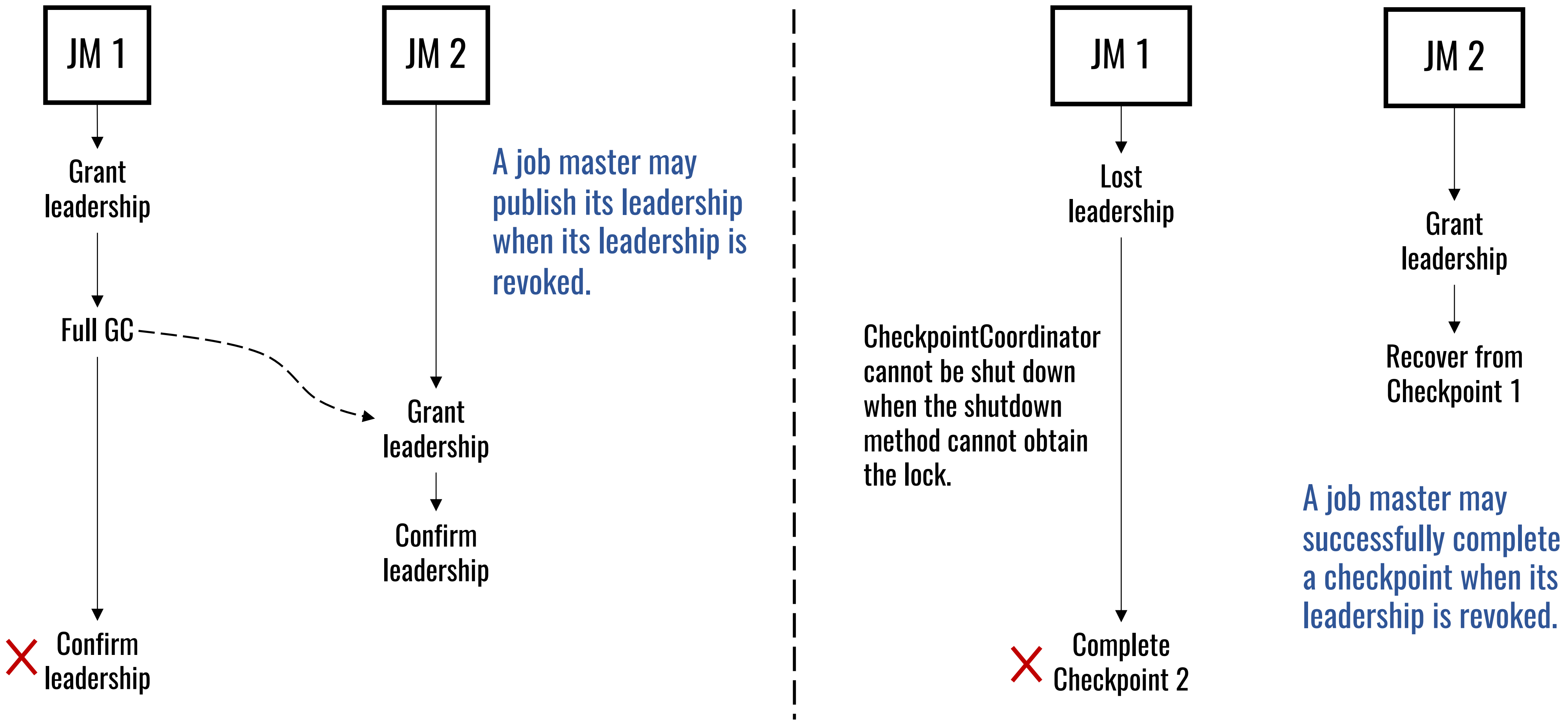
- Fine-grained resource allocation
- Improve scheduling efficiency

Performance & Usability

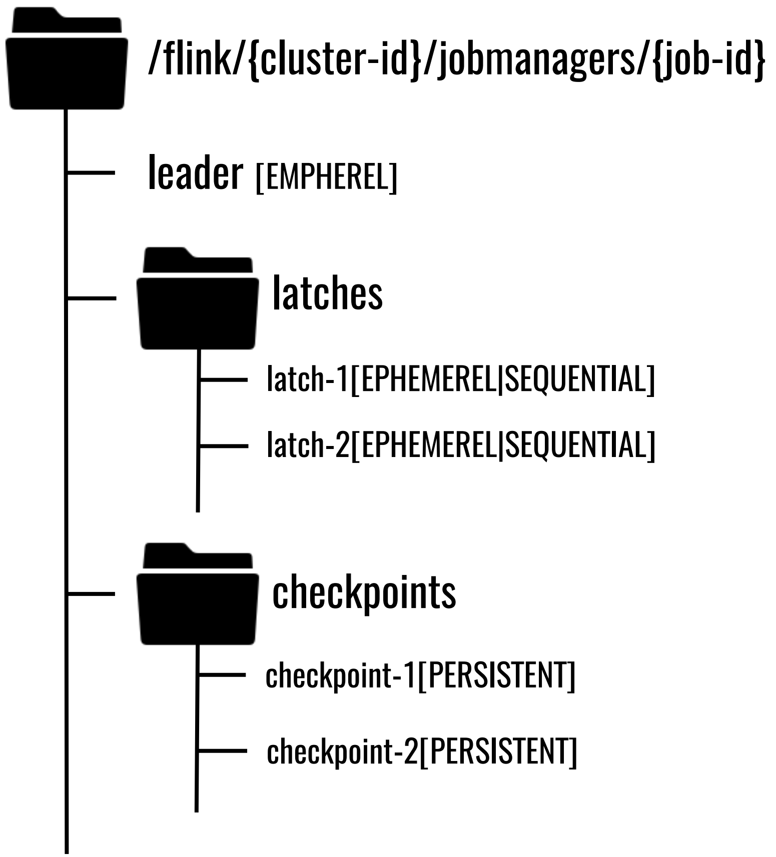
- Local keyed streams
- Incremental windows
- UDX
- DimJoin
- Top N

Refine leader coordination

Current problems: It’s difficult to reason about leadership in Flink.



Refine leader coordination

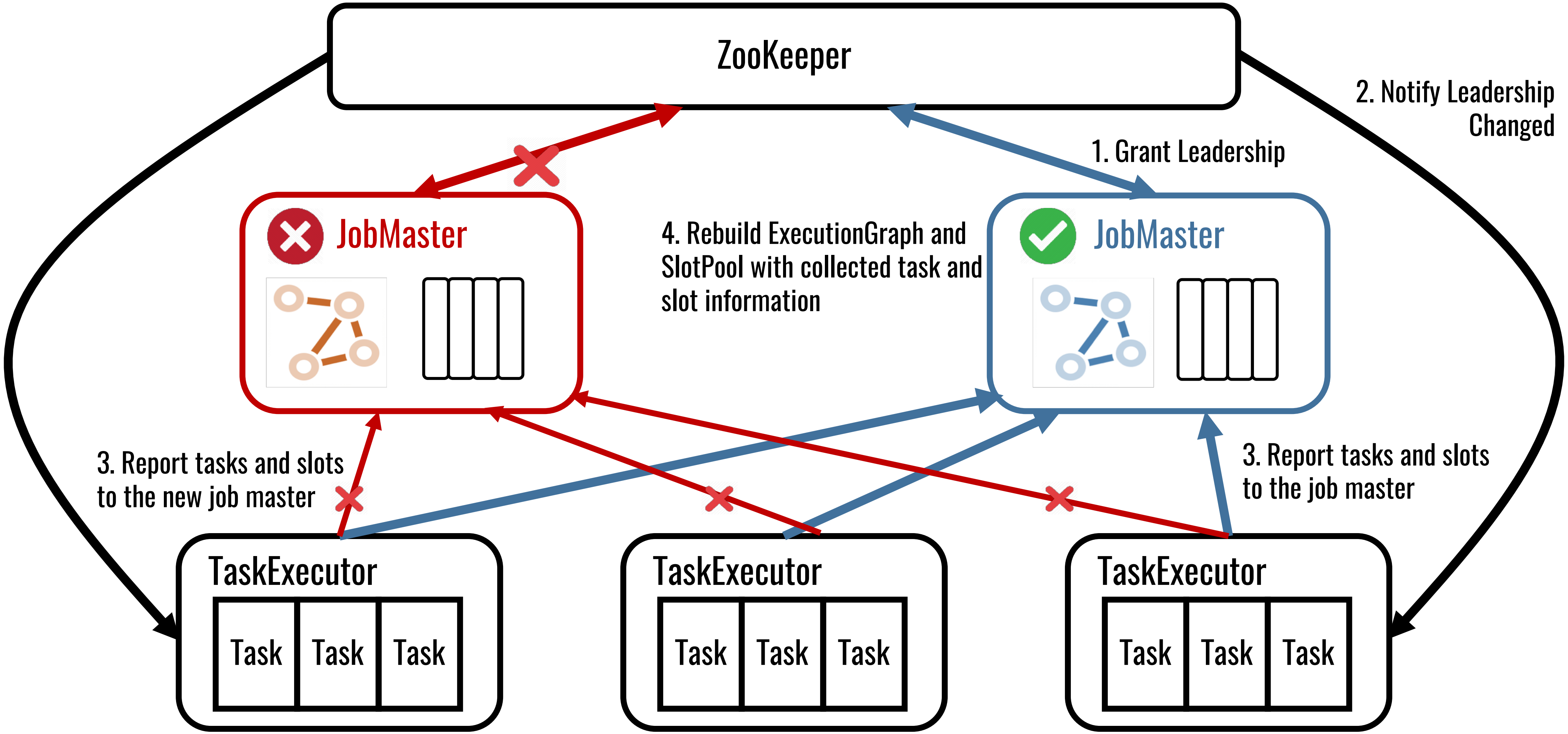


- Each leader contender creates a EPHEMERAL and SEQUENTIAL latch.
- The contender whose latch's sequential number is smallest is elected as the leader.
- A leader's leadership is granted as long as its latch exists.
- Each contender can only access states when it has granted leadership and its latch still exists.

```
zkClient.inTransaction()
    .check().forPath(myLatch).and()
    .setData().forPath(dataPath).and()
    .commit()
```

Non-disruptive recovery of job masters

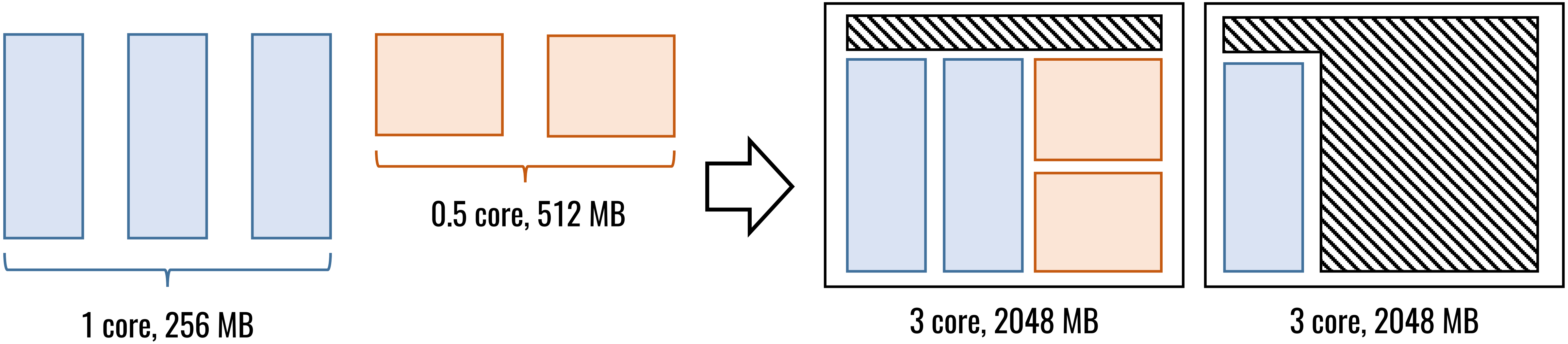
Avoid restarting tasks when the job master fails.



Fine-grained resource allocation

Current Problems:

- The resource specification for operators does not take effect in resource allocation.
- Slots are allocated according to the number of available slots in task managers, instead of the amount of available resources.
- Yarn containers may be killed when the used resources exceed the allocated ones.

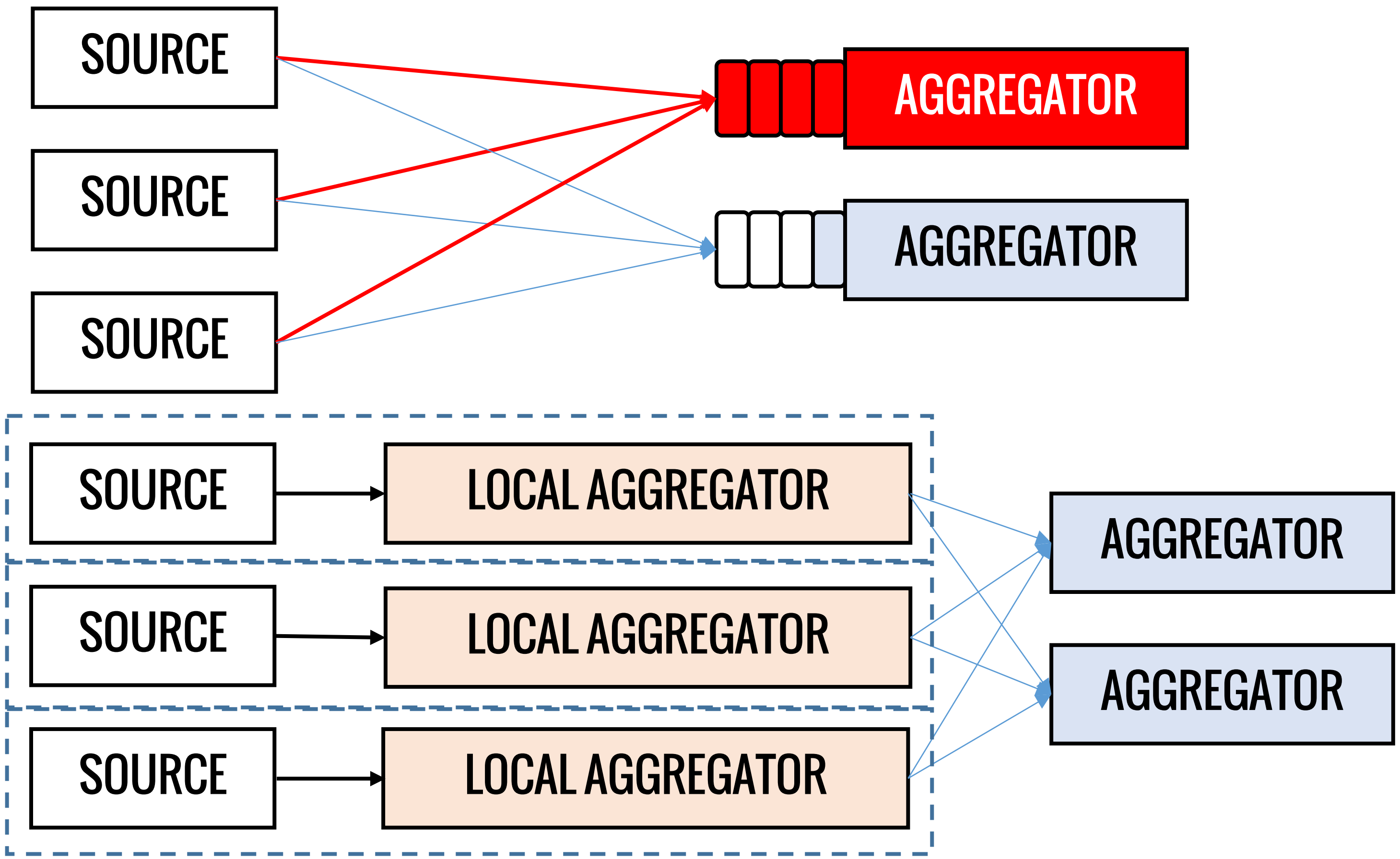


- Users can specify the resources needed by operators.
- A slot's resources are calculated by accumulating the resources of the operators in the slot.

- Slot instances are created and destroyed dynamically.
- A task manager creates a slot instance if its available resources are sufficient for the slot.
- A task manager destroys the slot instance if the tasks in the slot finish.

Local keyed streams

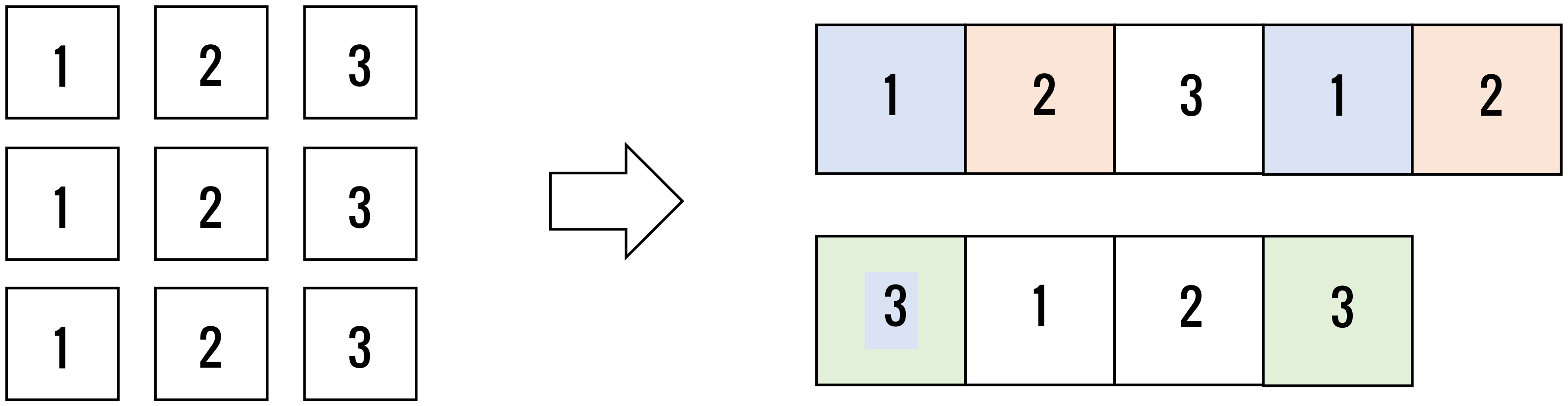
Current Problems: Performance is significantly degraded by data skew.



words.keyBy().count()

words
  .localKeyBy().window().count()
  .keyBy().sum()

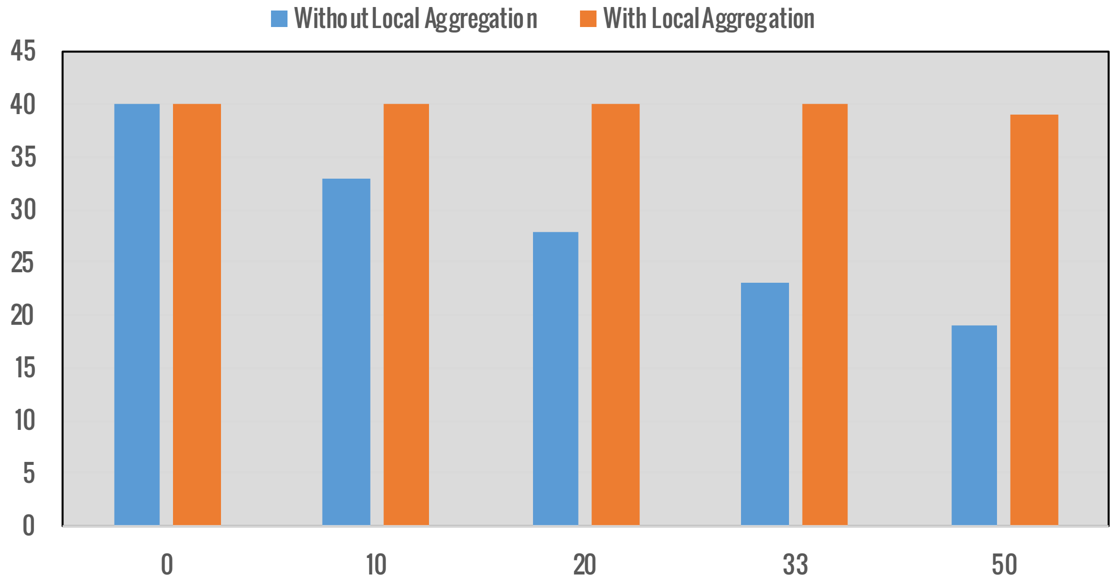
Local keyed streams



Each task has a complete key group range.

Groups are distributed to tasks according the number.

Groups with the same id are merged at restoring.



Usability

UDX

More than 40 UDX are provided

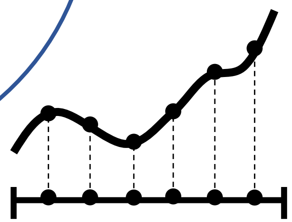
Dim Join

Optimized implement of joins with external storage

Top N

Incremental Window

Allow uses to obtain partial results of windows



Future Work

Improve scheduling efficiency

Unified checkpoint mechanism for both streaming and batch jobs

Incorporating partitioning and timing into optimizer

SuperSQL: efficient data analytics across data sources (Hive, HBase, PostgreSQL, etc) and data centers

Thank You