



Streaming Report

Functional Comparison and Performance Evaluation

毛玮

王华峰

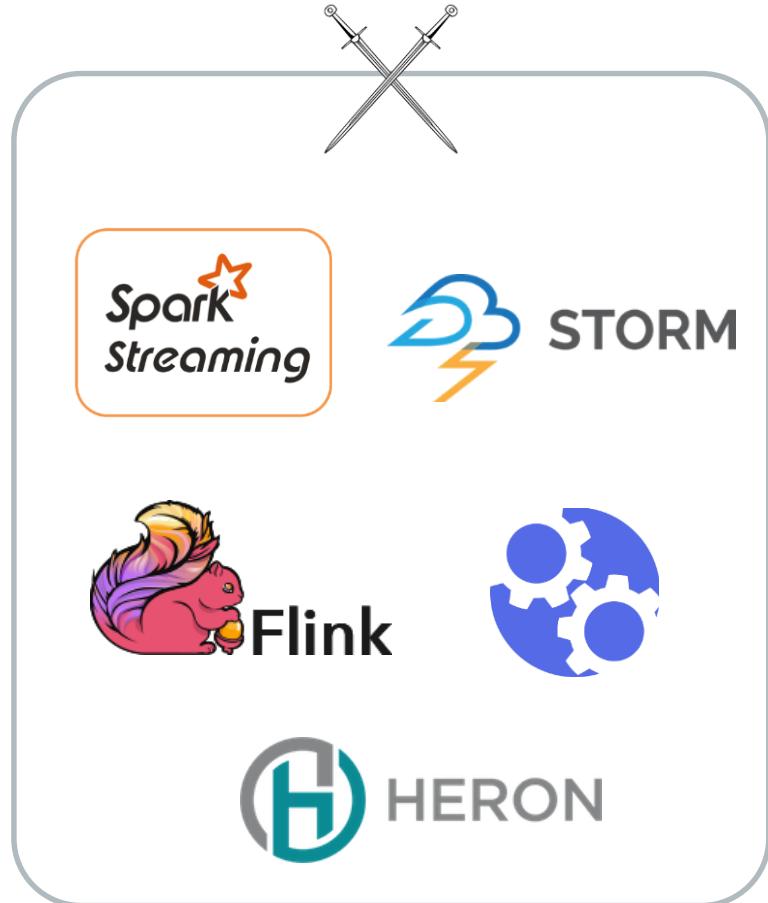
张天伦

2016/9/10

Overview

- Streaming Core
- MISC
- Performance Benchmark

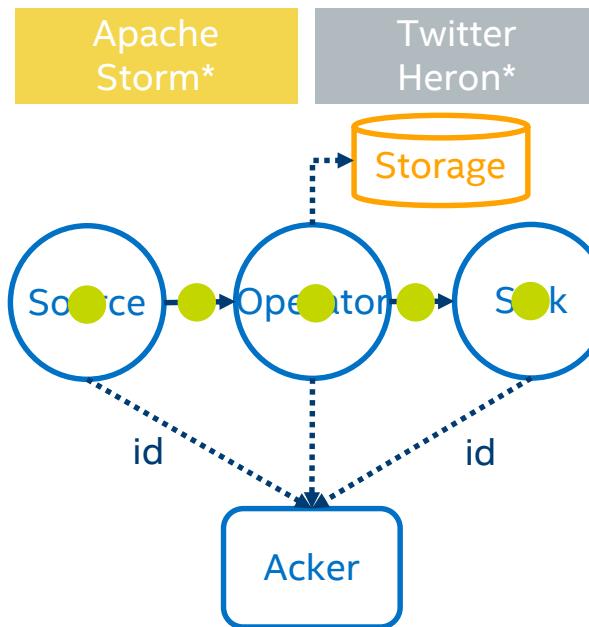
Choose your weapon !



Execution Model + Fault Tolerance Mechanism

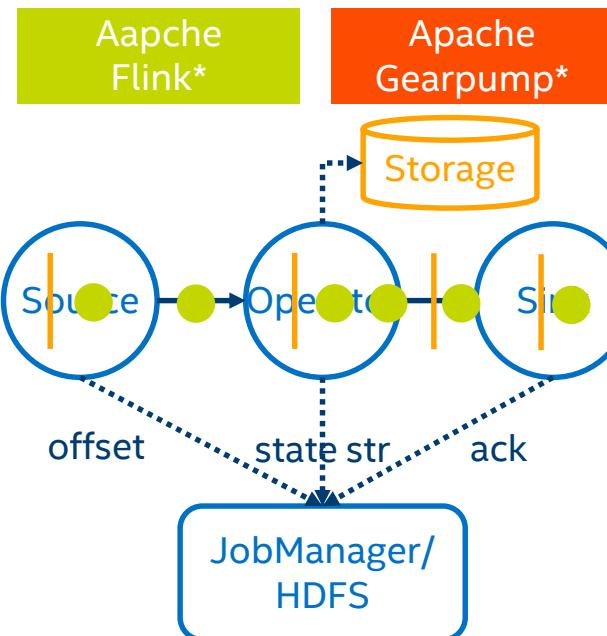
Continuous Streaming

Ack per Record



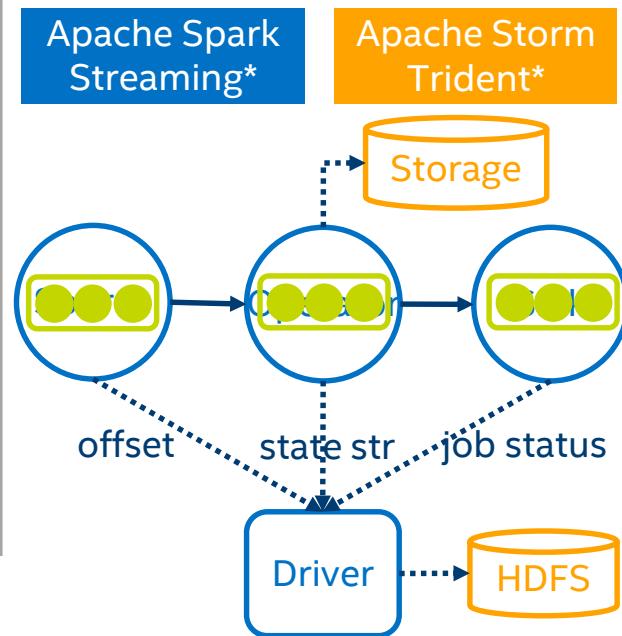
Continuous Streaming

Checkpoint “per Batch”



Micro-Batch

Checkpoint per Batch



This is the **critical** part, as it affects many features

Continuous Streaming

Ack per Record

Apache
Storm*

Twitter
Heron*

Continuous Streaming

Checkpoint “per Batch”

Apache
Flink*

Apache
Gearpump*

Micro-Batch

Checkpoint per Batch

Apache Spark
Streaming*

Apache Storm
Trident*

Low Latency

High Latency

High Overhead

Low Overhead

Low Throughput

High Throughput

Delivery Guarantee

Apache
Storm*

Twitter
Heron*

Aapche
Flink*

Apache
Gearpump*

Apache Spark
Streaming*

Apache Storm
Trident*

At least once

- Ackers know about if a record is processed successfully or not. If it failed, replay it.
- There is no state consistency guarantee.

Exactly once

- State is persisted in durable storage
- Checkpoint is linked with state storage per Batch

Native State Operator

Apache
Storm*

Twitter
Heron*

Aapche
Flink*

Apache
Gearpump*

Apache Spark
Streaming*

Apache Storm
Trident*

Yes*

- Storm:
 - ✓ KeyValueState
- Heron:
 - X User Maintain

Yes

- Flink Java API:
 - ✓ ValueState
 - ✓ ListState
 - ✓ ReduceState
- Flink Scala API:
 - ✓ mapWithState
- Gearpump
 - ✓ persistState

Yes

- Spark 1.5:
 - ✓ updateStateByKey
- Spark 1.6:
 - ✓ mapWithState
- Trident:
 - ✓ persistentAggregate
 - ✓ State

Dynamic Load Balance & Recovery Speed

Apache
Storm*

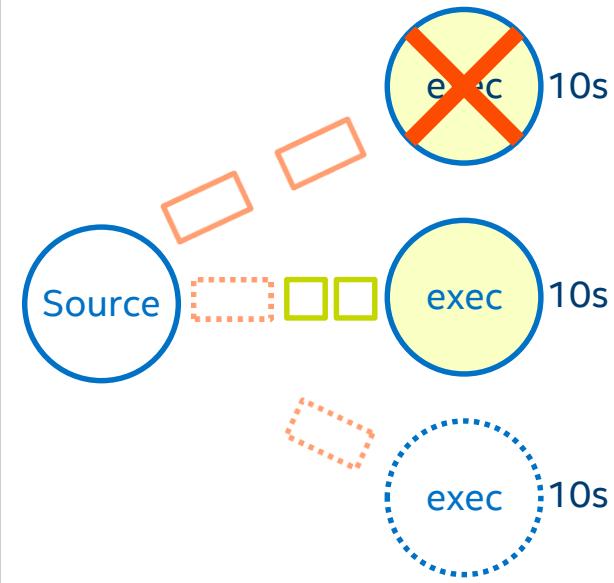
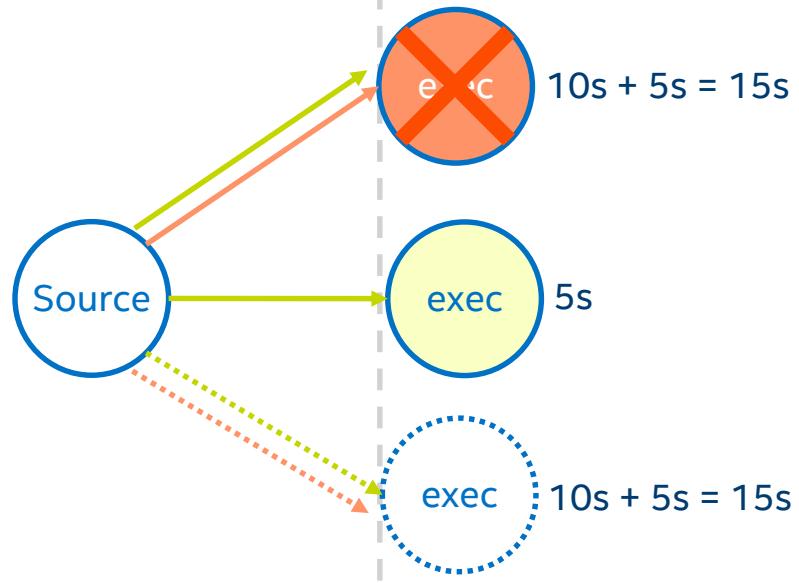
Twitter
Heron*

Aapche
Flink*

Apache
Gearpump*

Apache Spark
Streaming*

Apache Storm
Trident*

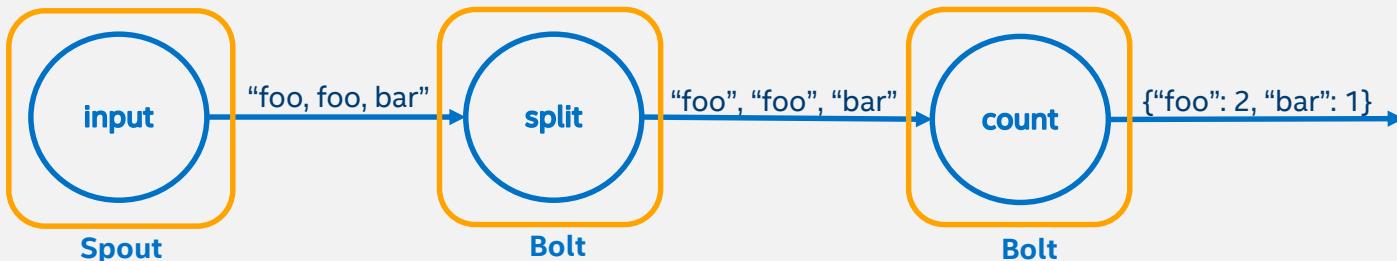


API

Compositional

- Highly customizable operator based on basic building blocks
- Manual topology definition and optimization

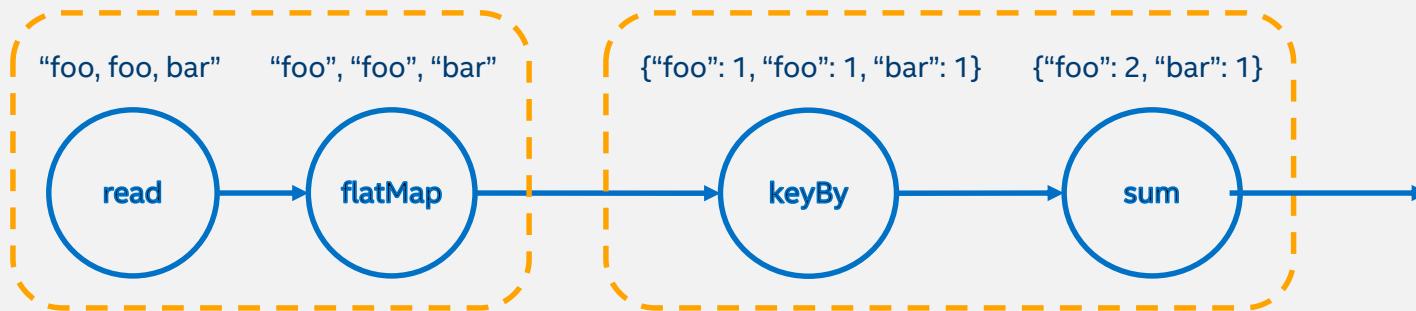
```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("input", new RandomSentenceSpout(), 1);
builder.setBolt("split", new SplitSentence(), 3).shuffleGrouping("spout");
builder.setBolt("count", new WordCount(), 2).fieldsGrouping("split", new Fields("word"));
```



Declarative

- Higher order function as operators (filter, mapWithState...)
- Logical plan optimization

```
DataStream<String> text = env.readTextFile(params.get("input"));
DataStream<Tuple2<String, Integer>> counts = text.flatMap(new Tokenizer()).keyBy(0).sum(1);
```



Statistical

- Data scientist friendly
- Dynamic type

Apache Spark
Streaming*

Apache
Storm*

Twitter
Heron*

Python

```
lines = ssc.textFileStream(params.get("input"))
words = lines.flatMap(lambda line: line.split(","))
pairs = words.map(lambda word: (word, 1))
counts = pairs.reduceByKey(lambda x, y: x + y)
counts.saveAsTextFiles(params.get("output"))
```

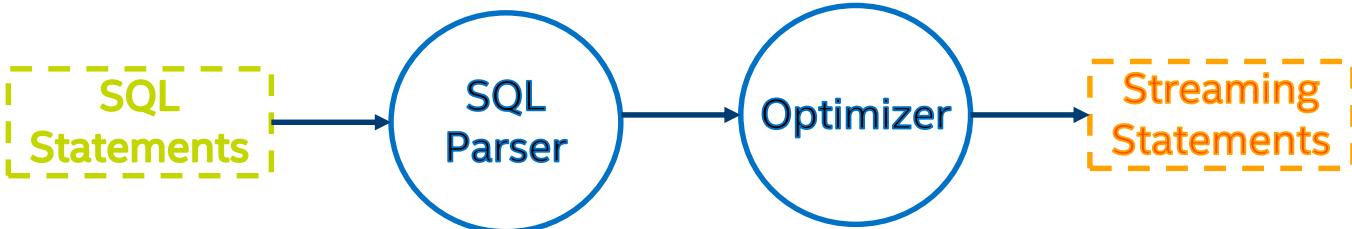
° Structured
Streaming*

° Apache
Storm*

R

```
lines <- textFile(sc, "input")
words <- flatMap(lines, function(line) {
  strsplit(line, " ")[[1]]
})
wordCount <- lapply(words, function(word) {
  list(word, 1L)
})
counts <- reduceByKey(wordCount, "+", 2L)
```

SQL



Fusion Style

```
InputDStream.transform((rdd: RDD[Order], time: Time) =>
{
    import sqlContext.implicits._
    rdd.toDF.registerTempTable
    val SQL = "SELECT ID, UNIT_PRICE * QUANTITY
              AS TOTAL FROM ORDERS WHERE UNIT_PRICE *
              QUANTITY > 50"
    val largeOrderDF = sqlContext.sql(SQL)
    largeOrderDF.toRDD
})
```

Apache Spark
Streaming*
Apache
Flink*

Pure Style

```
CREATE EXTERNAL TABLE
    ORDERS (ID INT PRIMARY KEY, UNIT_PRICE INT, QUANTITY
    INT)
    LOCATION 'kafka://localhost:2181/brokers?topic=orders'
    TBLPROPERTIES '{...}'
```



```
INSERT INTO LARGE_ORDERS SELECT ID, UNIT_PRICE *
    QUANTITY
    AS TOTAL FROM ORDERS WHERE UNIT_PRICE * QUANTITY >
    50
```

bin/storm sql XXXX.sql

Structured
Streaming
Apache Storm
Trident*



Summary

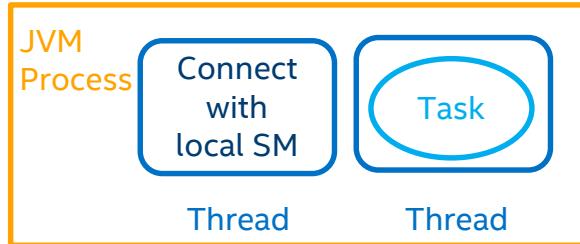
	Compositional	Declarative	Python/R	SQL
Apache Spark Streaming*	✗	✓	✓	✓
Apache Storm*	✓	✗	✓	NOT support aggregation, windowing and joining
Apache Storm Trident*	✗	✓	✗	
Apache Gearpump*	✓	✓	✗	✗
Apache Flink*	✗	✓	✗	Support select, from, where, union
Twitter Heron*	✓	✗	✓°	✗

*Other names and brands may be claimed as the property of others.



Runtime Model

- Single Task on Single Process

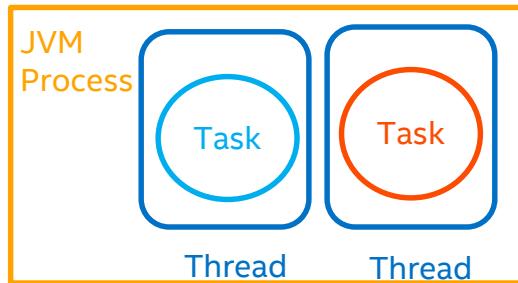


Twitter
Heron*



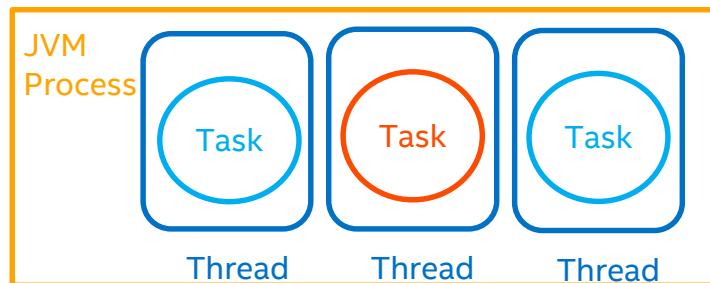
- Multi Tasks of Multi Applications on Single Process

Apache
Flink*



Task

task from application A



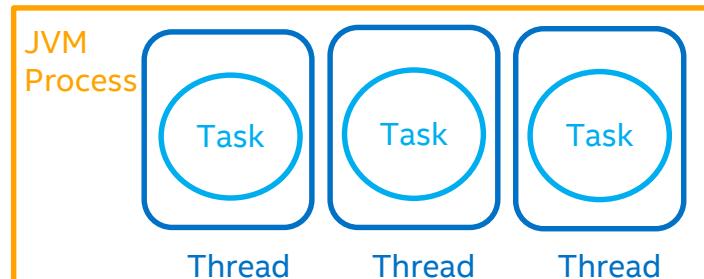
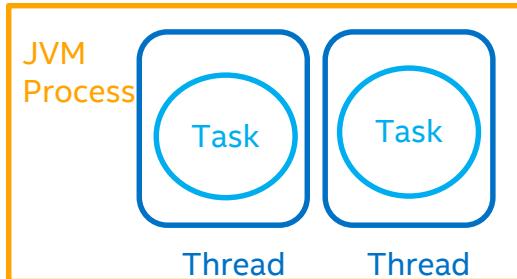
Task

task from application B

- Multi Tasks of Single application on Single Process

- Single task on single thread

Apache Spark
Streaming*

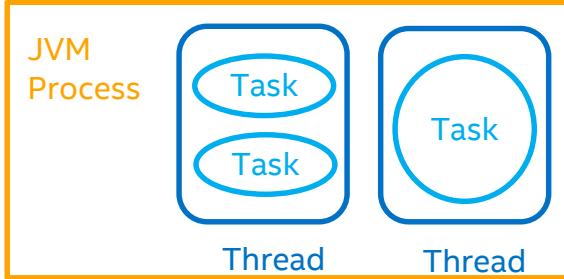
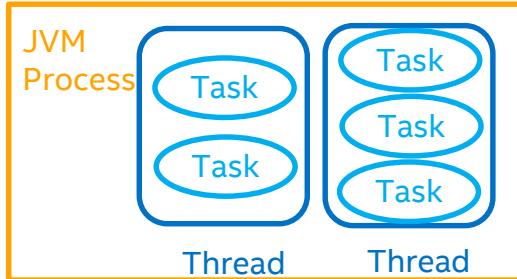


- Multi tasks on single thread

Apache
Storm*

Apache
Storm
Trident*

Apache
Gearpump*



MISC

- Window Support
- Out-of-order Processing
- Memory Management
- Resource Management
- Web UI
- Community Maturity

Window Support

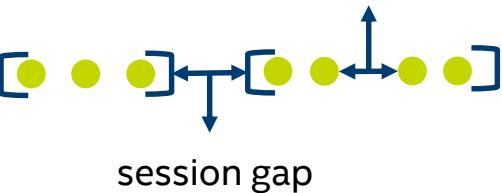
- Sliding Window



- Count Window



- Session Window



smaller than gap

	Sliding Window	Count Window	Session Window
Apache Spark Streaming*	√	✗	✗°
Apache Storm*	√	√	✗
Apache Storm Trident*	√	√	✗
Apache Gearpump*	√°	✗	✗
Apache Flink*	√	√	√
Apache Heron*	✗	✗	✗

*Other names and brands may be claimed as the property of others.

Out-of-order Processing

	Processing Time	Event Time	Watermark
Apache Spark Streaming*	✓	✓°	X°
Apache Storm*	✓	✓	✓
Apache Storm Trident*	✓	X	X
Apache Gearpump*	✓	✓	✓
Apache Flink*	✓	✓	✓
Twitter Heron*	✓	X	X

*Other names and brands may be claimed as the property of others.

Memory Management

	JVM Manage	Self Manage on-heap	Self Manage off-heap
Apache Spark Streaming*	✓	✓°	✓°
Aapche Flink*	✓	✓	✓
Apache Storm*	✓	✗	✗
Apache Gearpump*	✓	✗	✗
Twitter Heron*	✓	✗	✗

*Other names and brands may be claimed as the property of others.



Resource Management

	Standalone	YARN	Mesos
Apache Spark Streaming*	✓	✓	✓
Apache Storm*	✓	✓°	✓°
Apache Storm Trident*	✓	✓°	✓°
Apache Gearpump*	✓	✓	✗
Aapche Flink*	✓	✓	✗
Twitter Heron*	✓	✓	✓

*Other names and brands may be claimed as the property of others.



Web UI

	Submit Jobs	Cancel Jobs	Inspect Jobs	Show Statistics	Show Input Rate	Check Exceptions	Inspect Config	Alert
Apache Spark Streaming*	✗	✓	✓	✓	✓	✓	✓	✗
Apache Storm*	✗	✓	✓	✓	✓°	✓	✓	✗
Apache Gearpump*	✓	✓	✓	✓	✓°	✓	✓	✗
Apache Flink*	✓	✓	✓	✓	✗	✓	✓	✗
Twitter Heron*	✗	✗	✓	✓	✓°	✓	✓	✗

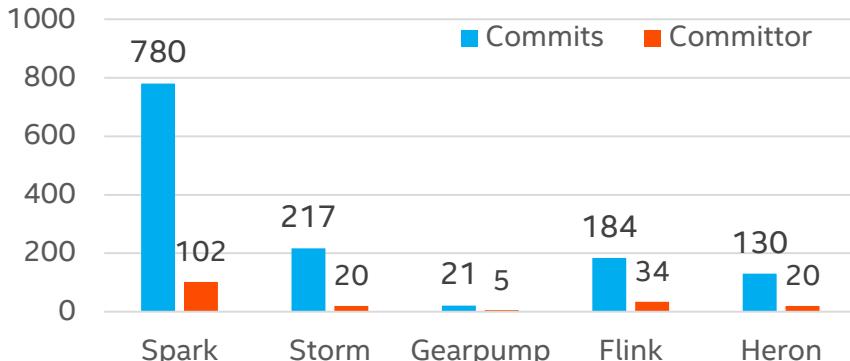
*Other names and brands may be claimed as the property of others.



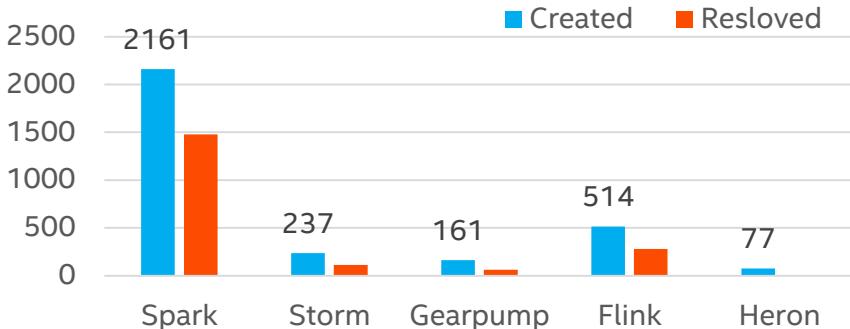
Community Maturity

	Initiation Time	Apache Top Project	Contributors
Apache Spark Streaming*	2013	2014	926
Apache Storm*	2011	2014	219
Apache Gearpump*	2014	Incubator	21
Apache Flink*	2010	2015	208
Twitter Heron*	2014	N/A	44

Past 1 Months Summary on GitHub



Past 3 Months Summary on JIRA



*Other names and brands may be claimed as the property of others.



Performance Benchmark

HiBench 6.0

Test Philosophical

- “Lazy Benchmarking”
- Simple test case infer practical use case

The Setup

Name	Version
Java	1.8
Scala	2.11.7
Apache Hadoop*	2.6.2
Apache Zookeeper*	3.4.8
Apache Kafka*	0.8.2.2
Apache Spark*	1.6.1
Apache Storm*	1.0.1
Apache Flink*	1.0.3
Apache Gearpump*	0.8.1

- Apache Heron* require specific Operation System (Ubuntu / CentOS / Mac OS)
- Structured Streaming doesn't support Kafka source yet (Spark 2.0)

Apache Kafka* Cluster

- **CPU:** 2 x Intel(R) Xeon(R) CPU E5-2699 v3@ 2.30GHz
- **Mem:** 128 GB
- **Disk:** 8 x HDD (1TB)
- **Network:** 10 Gbps

x3

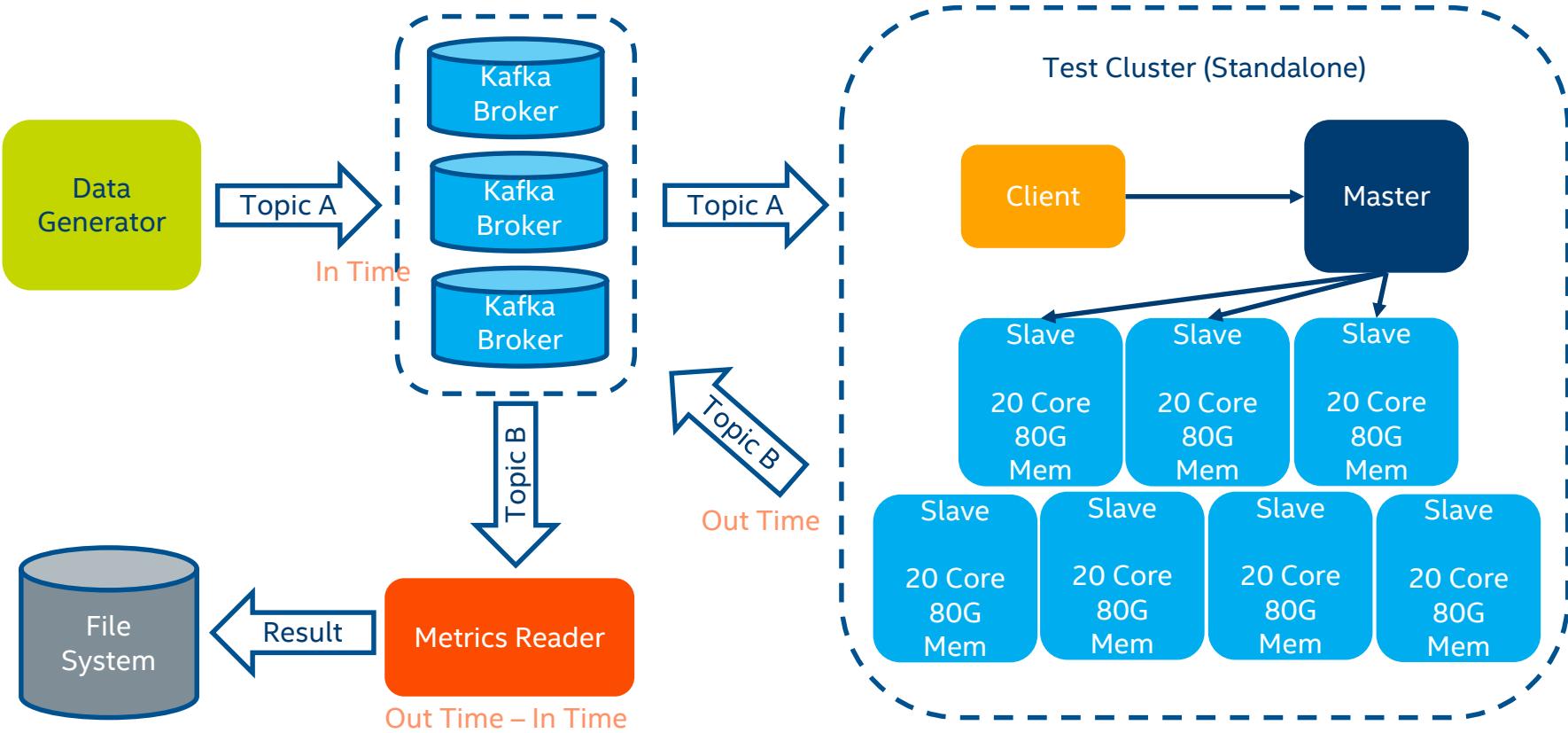
Test Cluster

- **CPU:** 2 x Intel(R) Xeon(R) CPU E5-2697 v2@ 2.70GHz
- **Core:** 20 / 24
- **Mem:** 80 / 128 GB
- **Disk:** 8 x HDD (1TB)
- **Network:** 10 Gbps

10 Gbps

x7

Architecture



Framework Configuration

Framework	Related Configuration
Apache Spark Streaming*	7 Executor 140 Parallelism
Aapche Flink*	7 TaskManager 140 Parallelism
Apache Storm*	28 Worker 140 KafkaSpout
Apache Gearpump*	28 Executors 140 KafkaSource

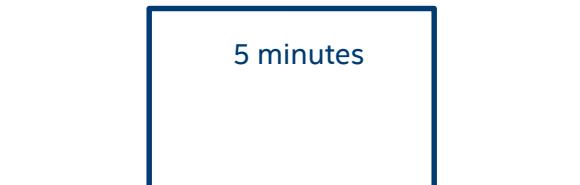
Raw Input Data

- Kafka Topic Partition: 140
- Size Per Message (configurable): 200 bytes
- Raw Input Message Example:

"0,227.209.164.46,nbizrgdziebsaecsecujfjcqtvnpcnxwiopmddorcxnlidzgoi,1991-06-10,0.115967035,Mozilla/5.0 (iPhone; U; CPU like Mac OS X) AppleWebKit/420.1 (KHTML like Gecko) Version/3.0 Mobile/4A93Safari/419.3,YEM,YEM-AR,snowdrops,1"

- Strong Type: class UserVisit (**ip**, **sessionId**, **browser**)

- Keep feeding data at specific rate for 5 minutes



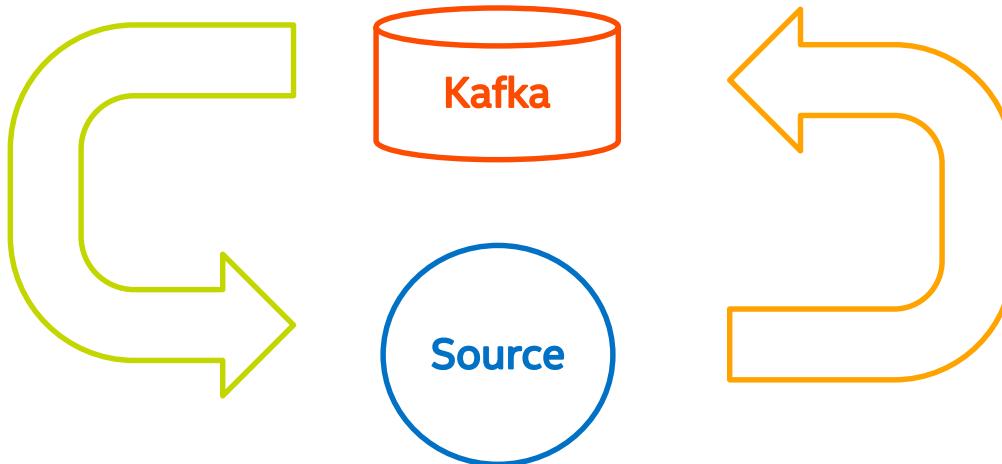
Data Input Rate

Throughput	Message/Second	Kafka Producer Num
40KB/s	0.2K	1
400KB/s	2K	1
4MB/s	20K	1
40MB/s	200K	1
80MB/s	400K	1
400MB/s	2M	10
600MB/s	3M	15
800MB/s	4M	20

Let's start with the simplest case

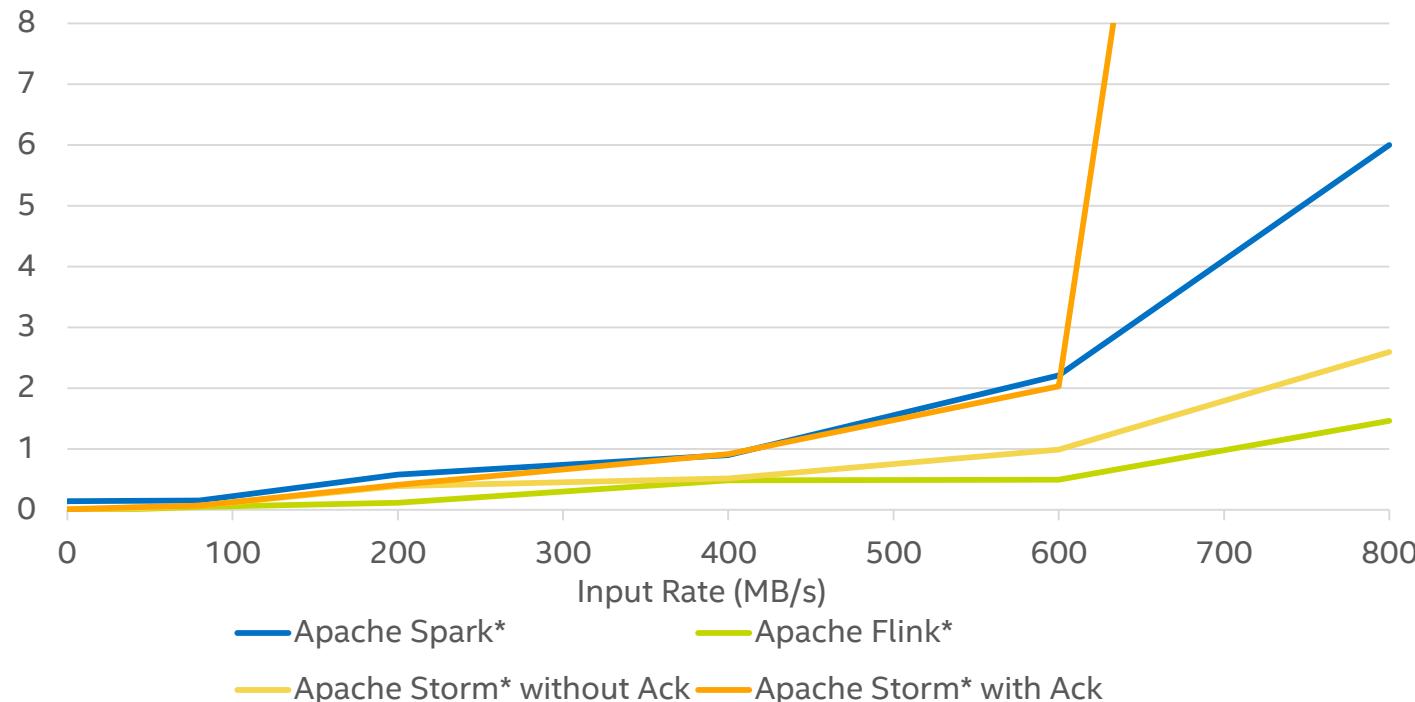
Test Case: Identity

The application reads input data from Kafka and then writes result to Kafka immediately, there is no complex business logic involved.



Result

P99 Latency (s)



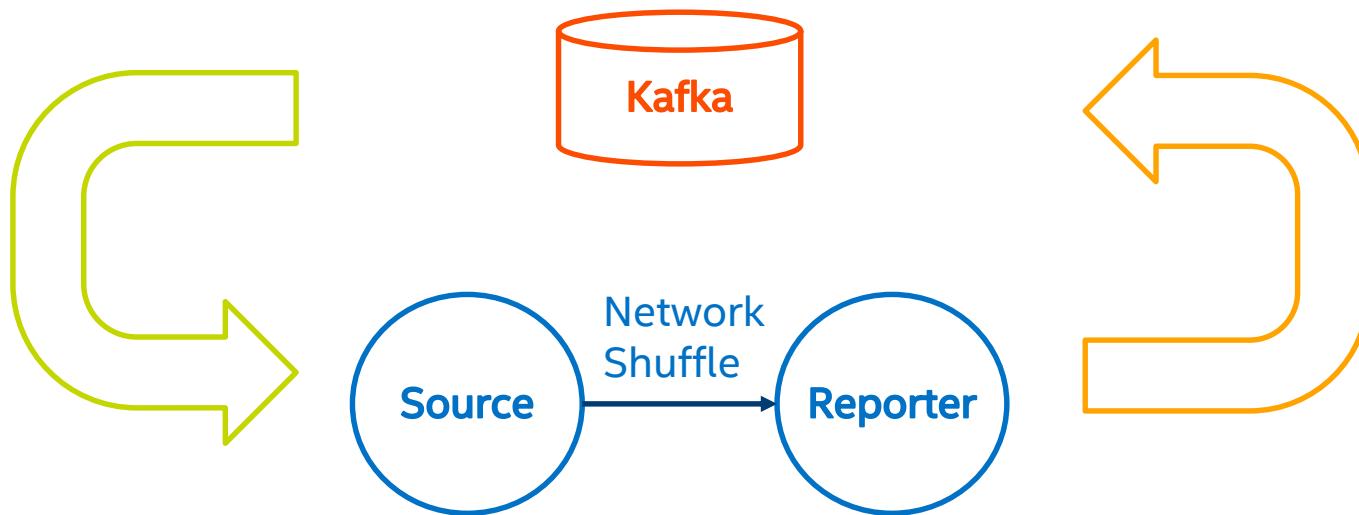
*Other names and brands may be claimed as the property of others.

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

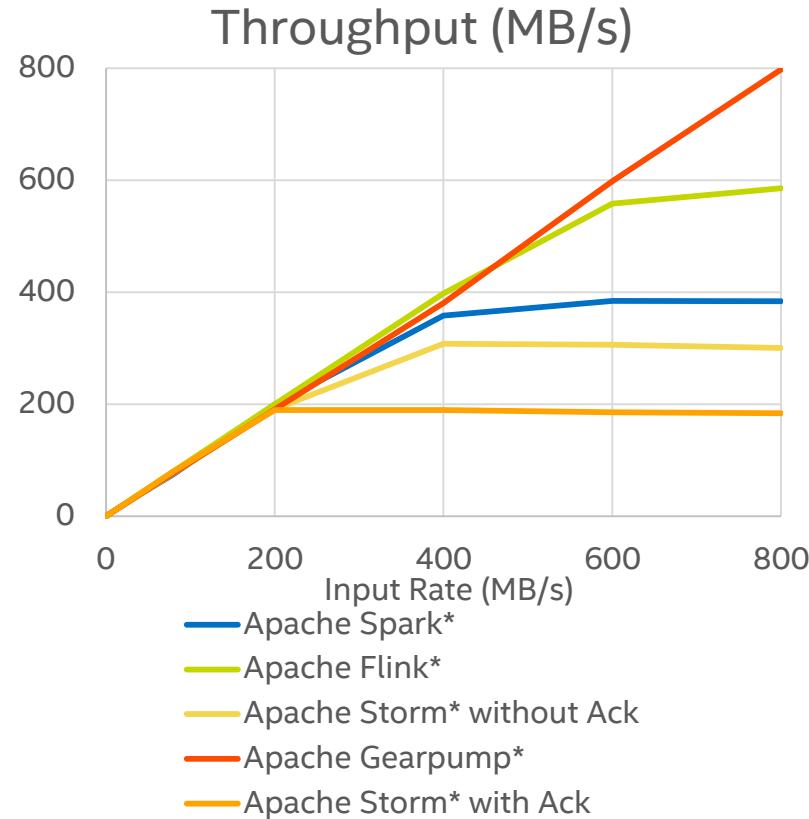
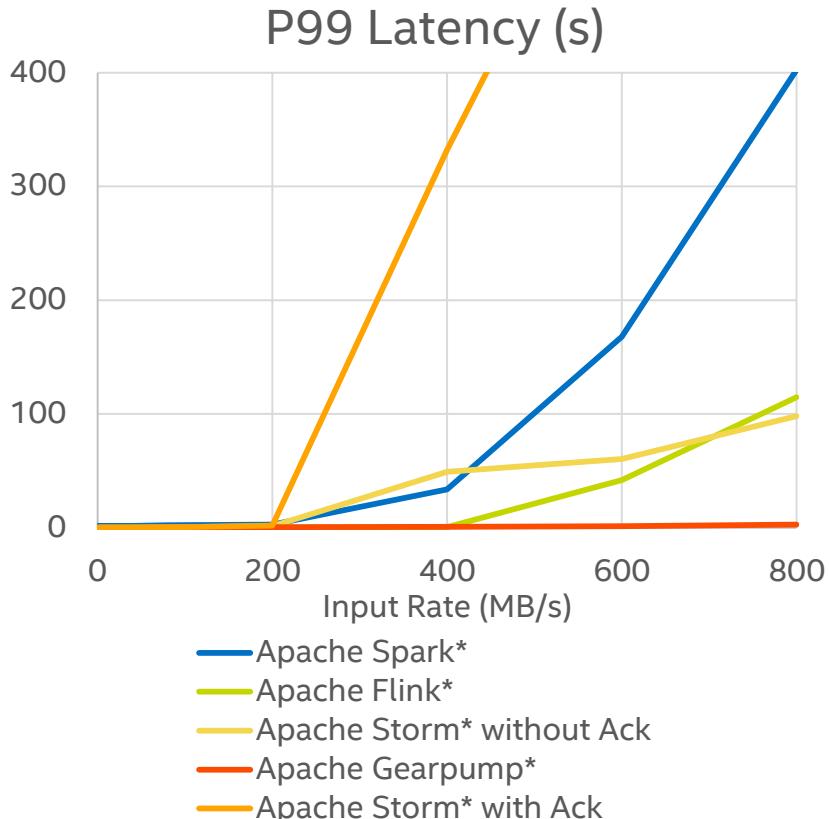
Q: What if source data are skew
or even packed?

Test Case: Repartition

Basically, this test case can stand for the efficiency of data shuffle.



Result



*Other names and brands may be claimed as the property of others.

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Observation

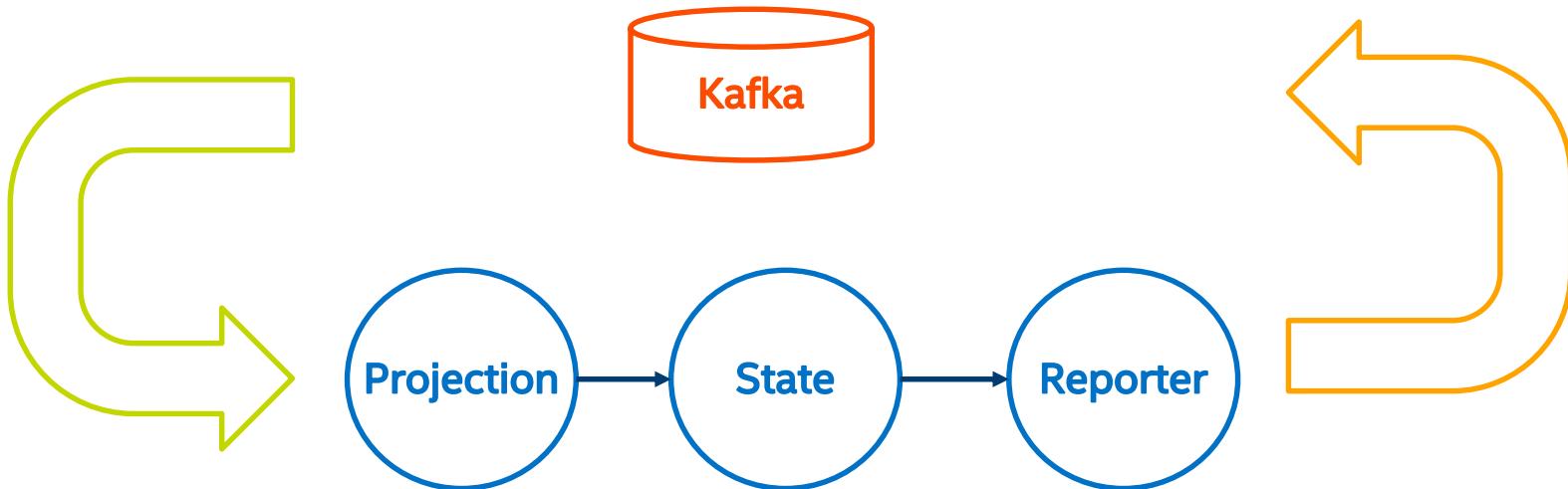
- Flink and Storm has close performance and are better choices to meet **sub-second SLA** requirement if no repartition happened.
- Spark Streaming need to schedule task with additional context. Under tiny batch interval case, the overhead could be dramatic worse compared to other frameworks.
- According to our test, minimum Batch Interval of Spark is about 80ms (140 tasks per batch), otherwise task schedule delay will keep increasing
- Repartition is heavy for every framework, but usually it's unavoidable.
- Latency of Gearpump is still quite low even under 800MB/s input throughput.

Q: What if I want to apply slightly complex logic which need to maintain entire state?

Test Case: Stateful WordCount

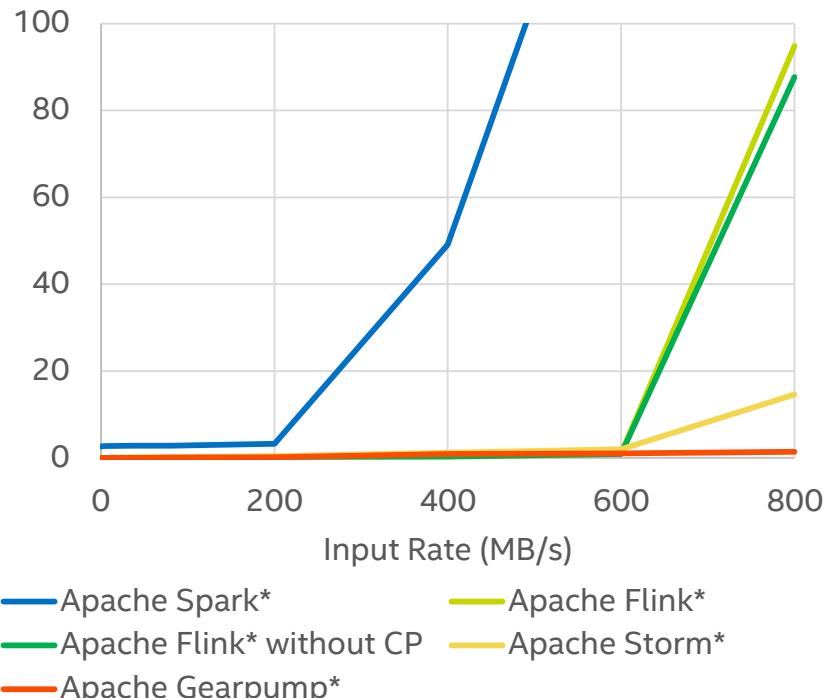
Native state operator is supported by all frameworks we evaluated

Stateful operator performance + Checkpoint/Acker cost

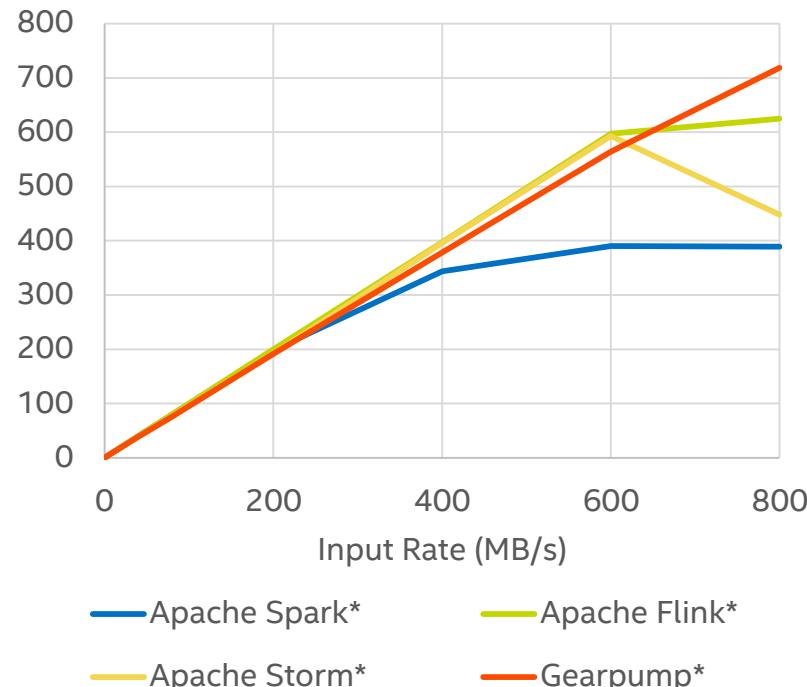


Result

P99 Latency (s)



Throughput (MB/s)



*Other names and brands may be claimed as the property of others.

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

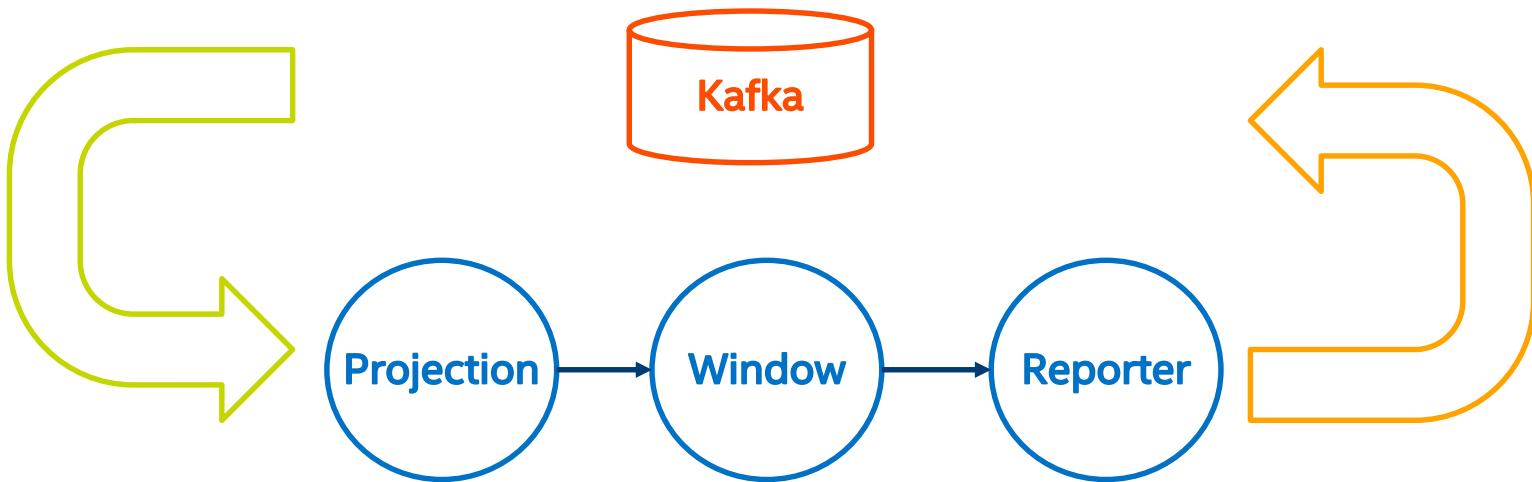
Observation

- Exactly-once semantics usually require state management and checkpoint. But better guarantees come at high cost.
- There is no obvious performance difference in Flink when switching fault tolerance on or off.
- Checkpoint mechanisms and storages play a critical role here.

Q: How about Window Operation?

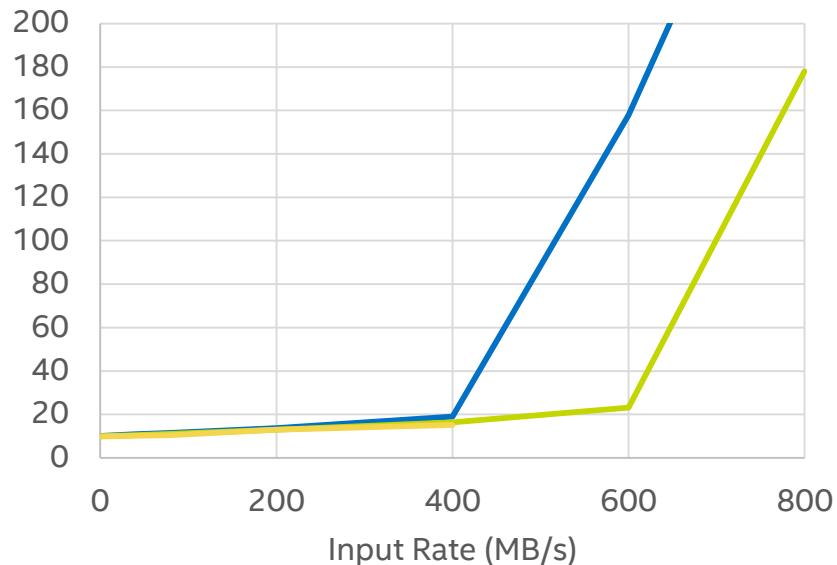
Test Case: Window Based Aggregation

This test case manages a 10-seconds sliding window

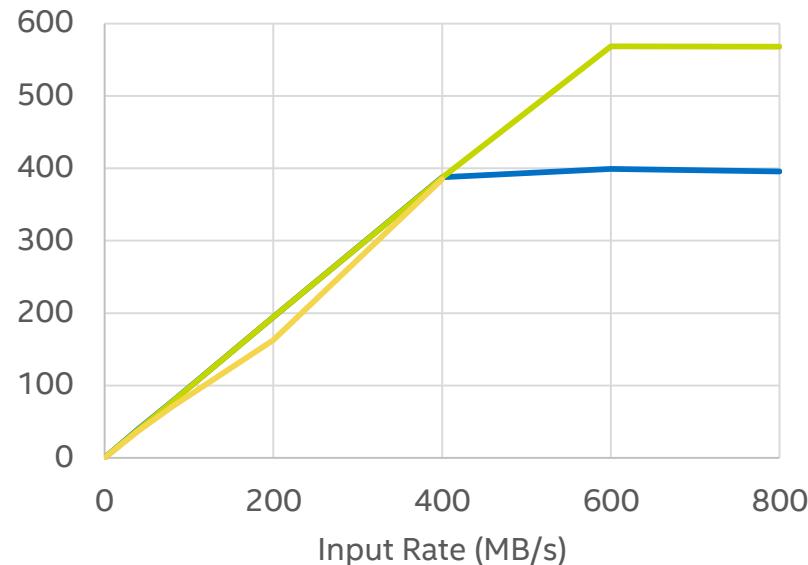


Result

P99 Latency (s)



Throughput (MB/s)



— Apache Spark* — Apache Flink* — Storm*

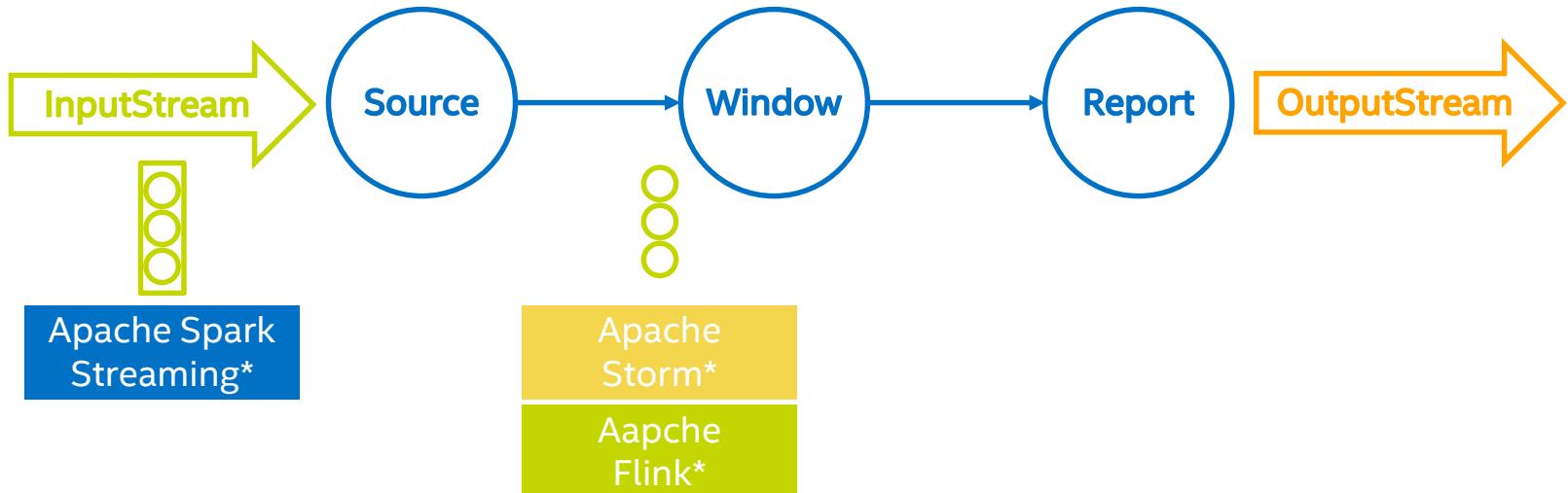
— Apache Spark* — Apache Flink* — Storm*

*Other names and brands may be claimed as the property of others.

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Observation

The native streaming execution model helps here



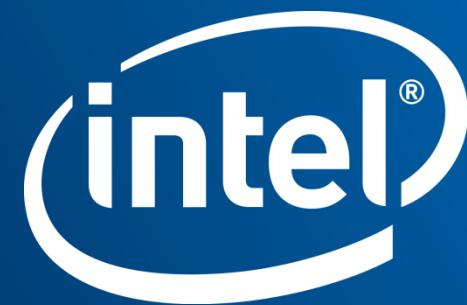
So which streaming framework should I use?

Do your own benchmark

HiBench : a cross platforms micro-benchmark suite for big data
<https://github.com/intel-hadoop/HiBench>

Open Source since 2012

Better streaming benchmark supporting will be included in next release
[HiBench 6.0]



Legal Disclaimer

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

*Other names and brands may be claimed as the property of others.

Copyright ©2016 Intel Corporation.