Comparative Analysis of Stream Processing Systems

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Sulong Zhou
Shawn Zhong
Suyan Qu

Problem Statement

- In the past few decades, the explosion of modern information and technology has led to high demands for processing streaming data with low latency and high accuracy. In response, many different stream processing systems were developed.
- With that many different systems, it is important to be able to evaluate the systems and know which system is more suitable for the workload we want.
- Here we present our reviews for 5 different streaming processing systems: Apache Storm, Spark Streaming, Apache Flink, and Kafka Streams.
- Also, we compare the performance of 2 of them: Spark streaming and Kafka Streams, based on Yahoo Streaming Benchmarks.

Stream Process Systems

- **Apache Storm**: Twitter's 1st-generation of real-time, fault-tolerant stream processing system. Here master communicates with worker nodes through Zookeepers, which is also responsible for monitoring workers.
- **Apache Heron**: Twitter's 2nd-generation stream processing system improved from Apache Storm. It separates monitoring from execution for scalability.
- **Spark Streaming**: System based on Apache Spark. Spark Streaming uses micro-batches and accumulates result to achieve streaming data processing.
- **Apache Flink**: Flink processes data as real streams. Data are processed upon arrival to provide low latency.
- **Kafka Streaming**: Kafka streaming processes data in real-time. It is based on existing project Kafka and is designed to be light-weighted.

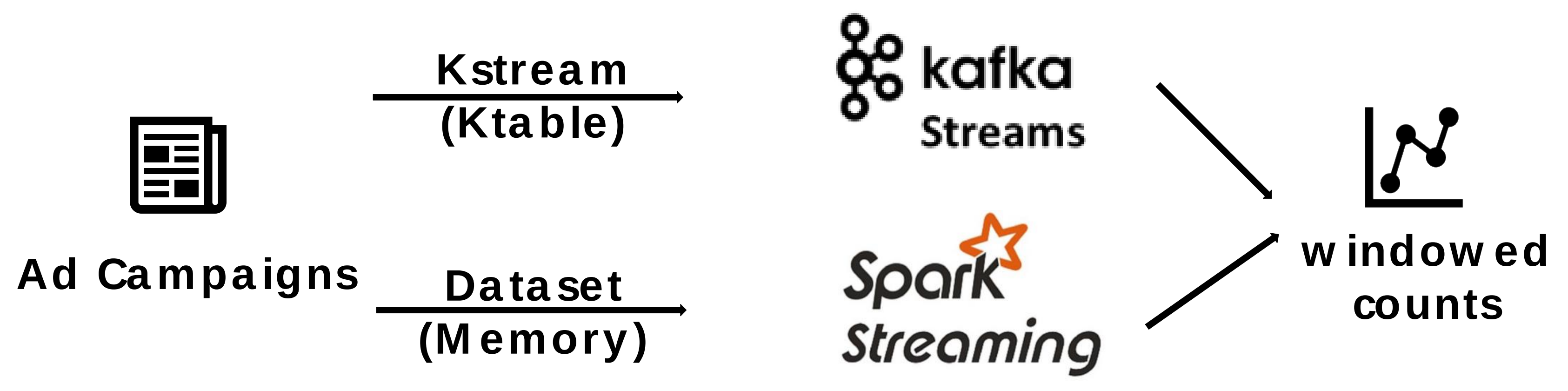
Approach

We compare these processing systems from two main aspects: performance and social factors

- Performance metrics include latency, throughput, fault, etc.
- Social factors consist of development activity and history,

Evaluation Design

- For performance analysis, we use Yahoo Benchmarks, where the input data consist of randomly generated user actions with ad blocks. Each action includes a user ID, page ID, ad ID, ad type, event type, event time, and IP address. The processing system will count number of actions, such as views, clicks, on each ads.
- For social factors, we focus on two areas: google search history, representing the popularity and trend of these stream processing systems, and GitHub commit histories, indicating if the system is actively being developed.

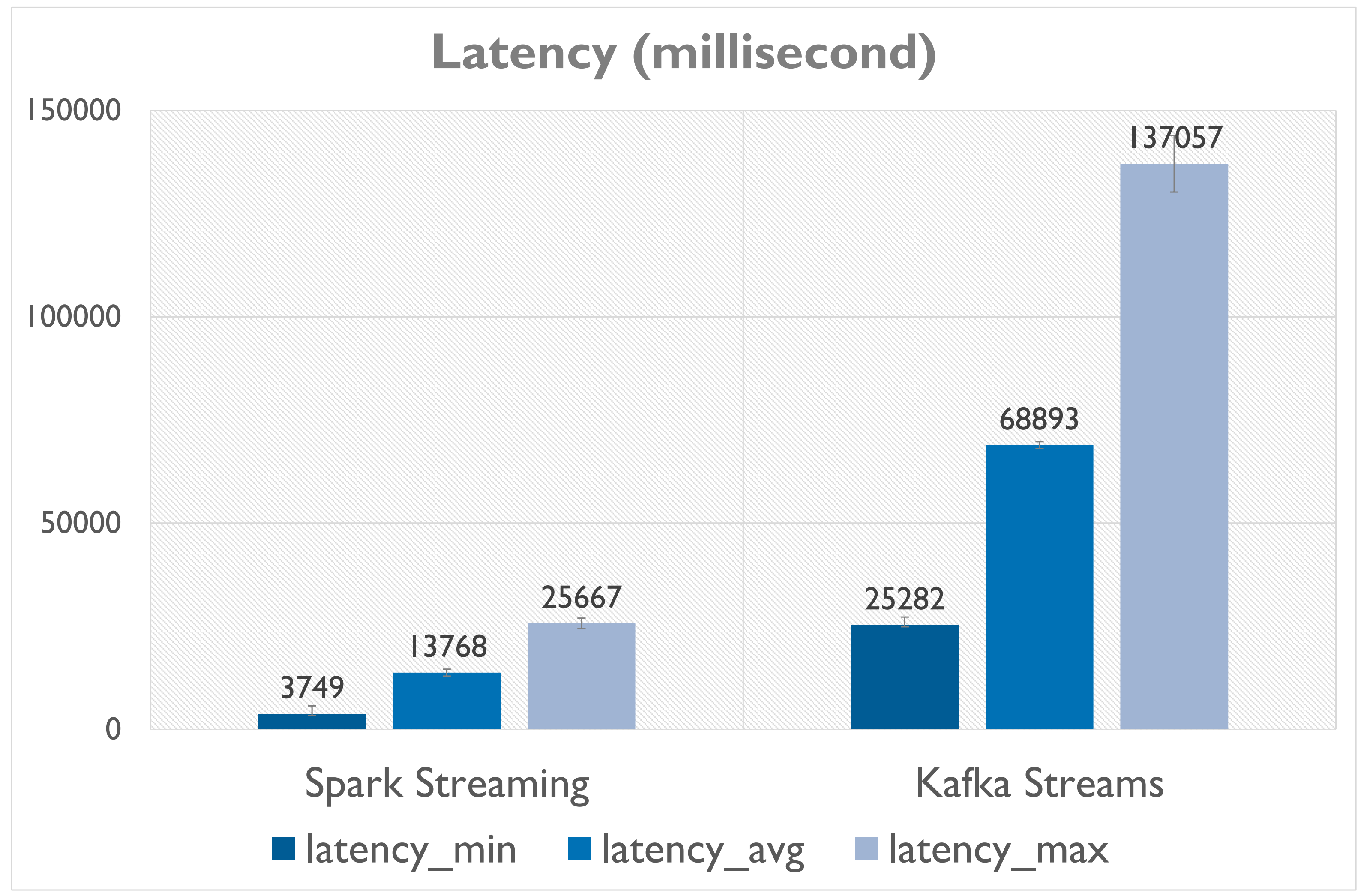


Social Factors

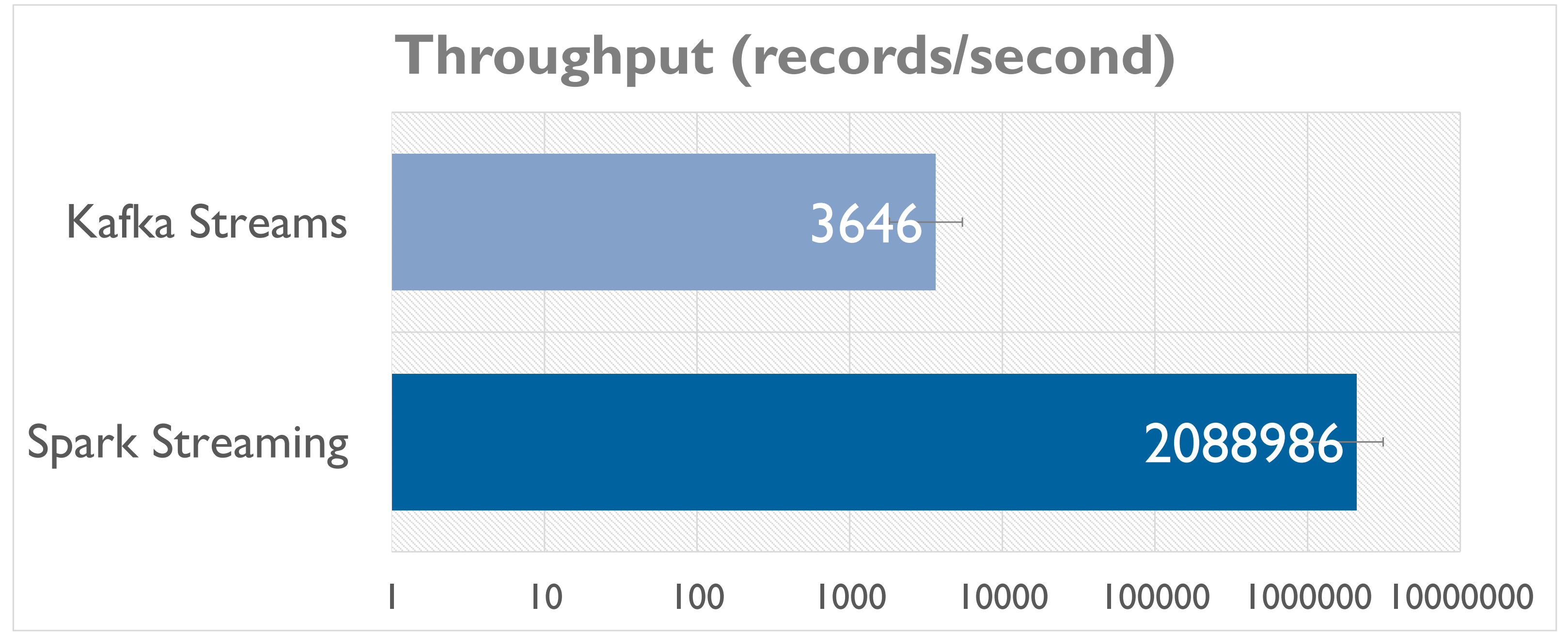
System	Commits Past Year	Committer	Lines of Code	Search Index*	Open Issues
Storm	228	305	377,166	10	1033
Heron	201	106	252,322	2	363
Spark**	2565	1451	962,336	36	1859
Flink	4312	595	1,429,183	79	2913
Kafka	1201	606	459,984	23	2616

* Search Index: the prevailing trend of the framework.
** Spark: statistics is for the entire Spark program.

Benchmark & Discussion



The latency was calculated in two steps. First, the timestamp of the latest record that was received for a pair of campaign and event-time-window during the windowed counts phase. Second, the latency was indicated by the difference between this timestamp and the Kafka ingestion timestamp of the output. As shown in above figure, we compared the latency between spark streaming and Kafka streams, the Kafka streams has much higher latency.



The throughput was denoted by the average number of records in the processing duration. As shown in above figure, the throughput in Spark Streaming achieved 2M/s, which completely outperformed Kafka streams.