Securing Stream Processing Systems against Untrusted Code

Jordan Boulanger, Mentored by Dr. David Lillethun
Department of Computer Science, Seattle University

SEATTLE U
COLLEGE OF SCIENCE AND ENGINEERING

Abstract

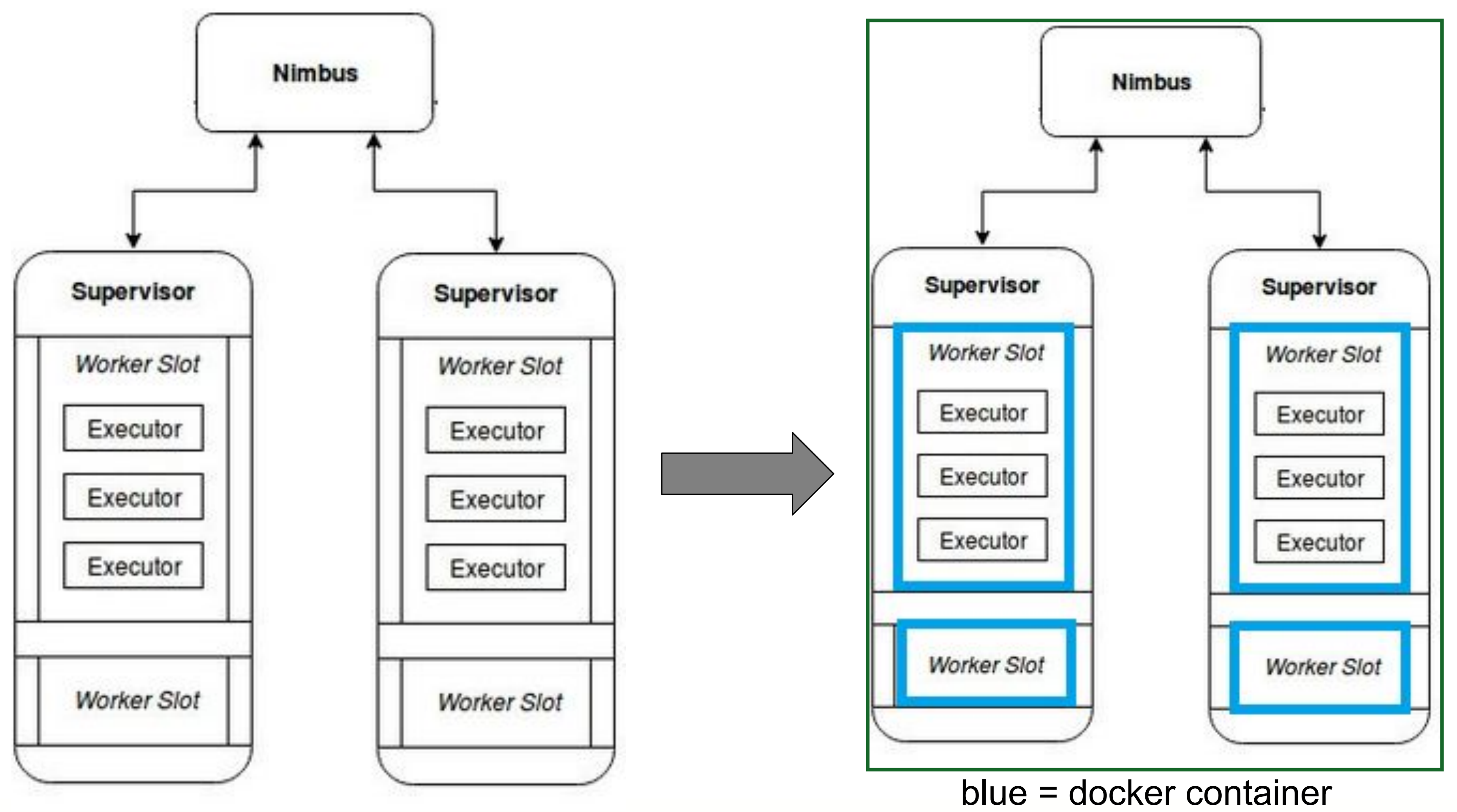
Modern companies today must be able to process increasingly vast amounts of data in real-time. Frameworks such as Apache Storm were created to tackle this challenge. In stream processing, data processing functions, sometimes called components, are chained together to form a topology that accomplishes some larger task. Our research goal was to provide isolation and a controlled environment for each component, rather than having them execute in a shared environment. This allows for the potential use of unaudited code without worrying about side effects or malicious intent.

Introduction

Goal: Modify Apache Storm to provide isolation between topology components.

Method: Containerization technology is used to provide a separate environment for each instance of a component. We choose to leverage Docker as a means of accomplishing this goal. A custom scheduler was also added to storm to ensure that each worker only runs executors (threads) for one component. This means there must be more workers per cluster/supervisor but that each worker can be allocated less resources.

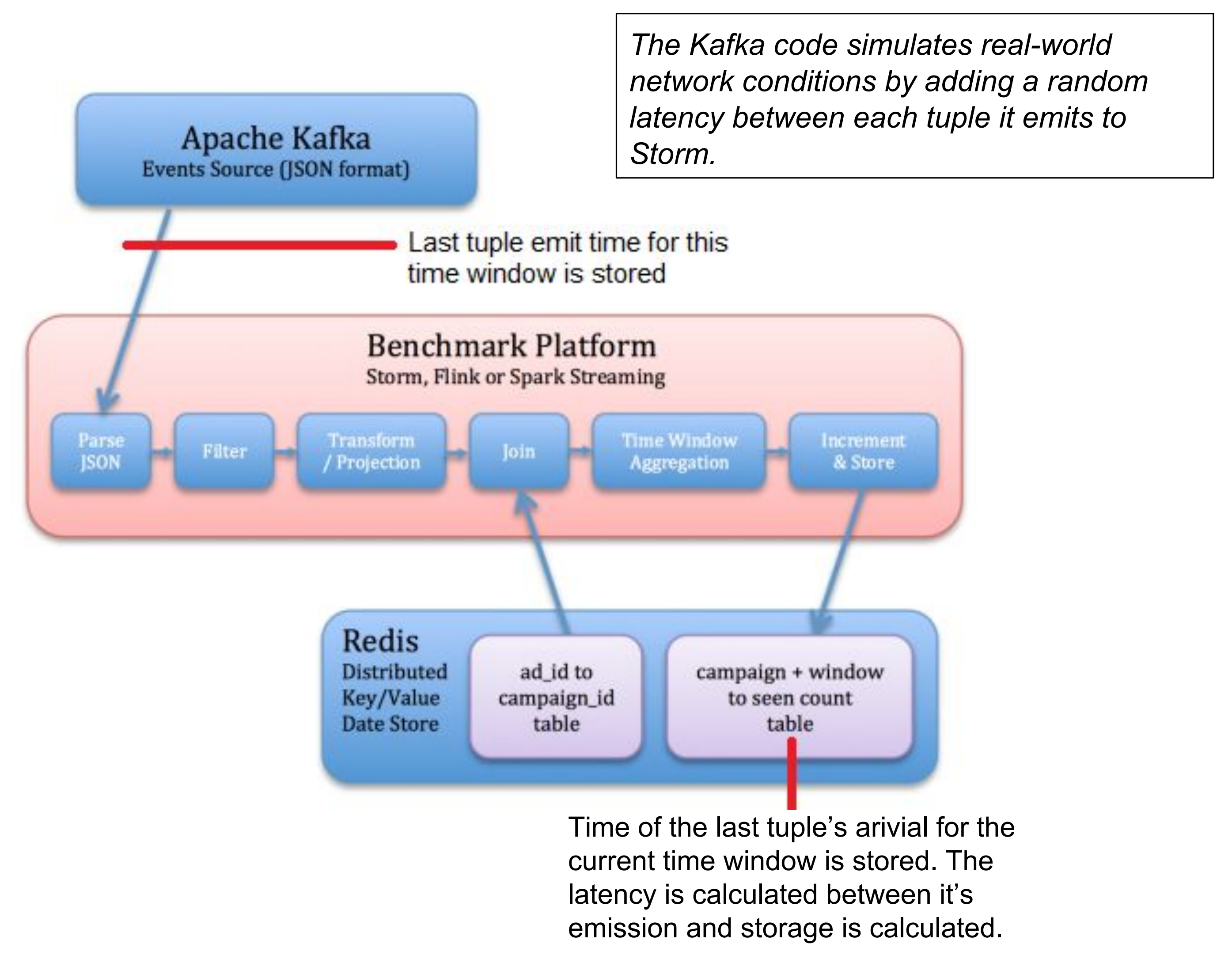
Theory



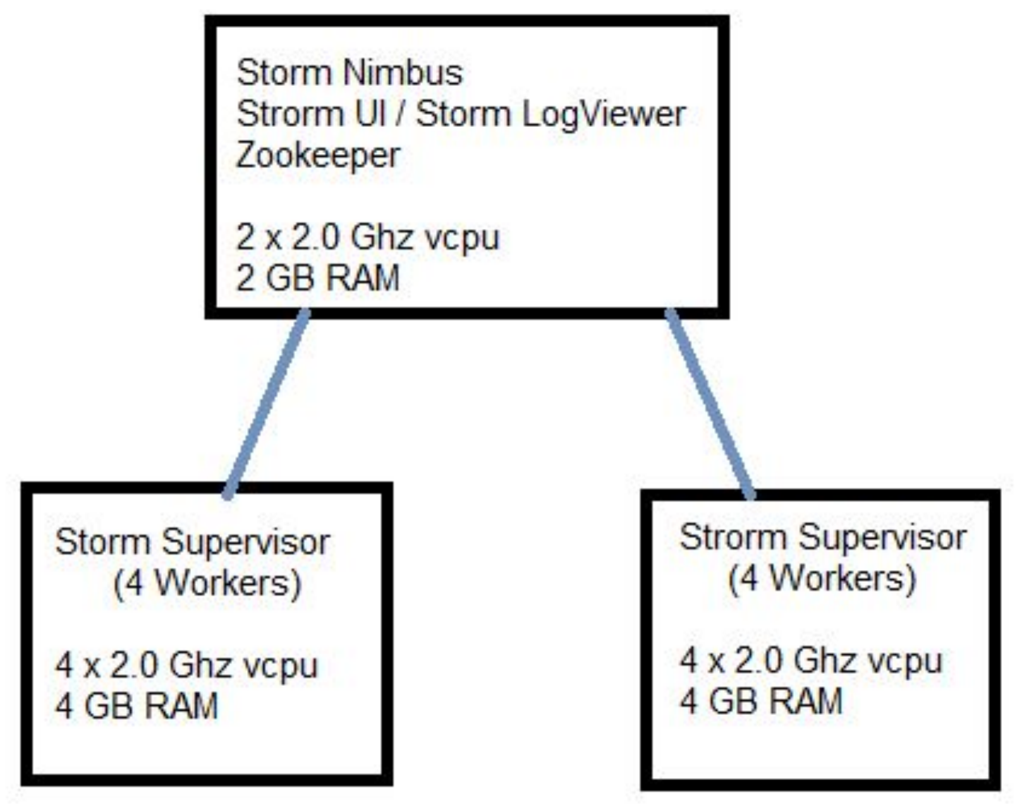
blue = docker container

Experimental Design

Benchmark Topology Designed by Yahoo


The Kafka code simulates real-world network conditions by adding a random latency between each tuple it emits to Storm.
Apache Kafka
Events Source (JSON format)
Last tuple emit time for this time window is stored
Benchmark Platform
Storm, Flink or Spark Streaming
Parse JSON
Filter
Transform / Projection
Join
Time Window Aggregation
Increment & Store
Redis
Distributed Key/Value Date Store
ad_id to campaign_id table
campaign + window to seen count table
Time of the last tuple's arivial for the current time window is stored. The latency is calculated between it's emission and storage is calculated.

Storm Test Cluster Specs (connected by vlan)



Technologies Used

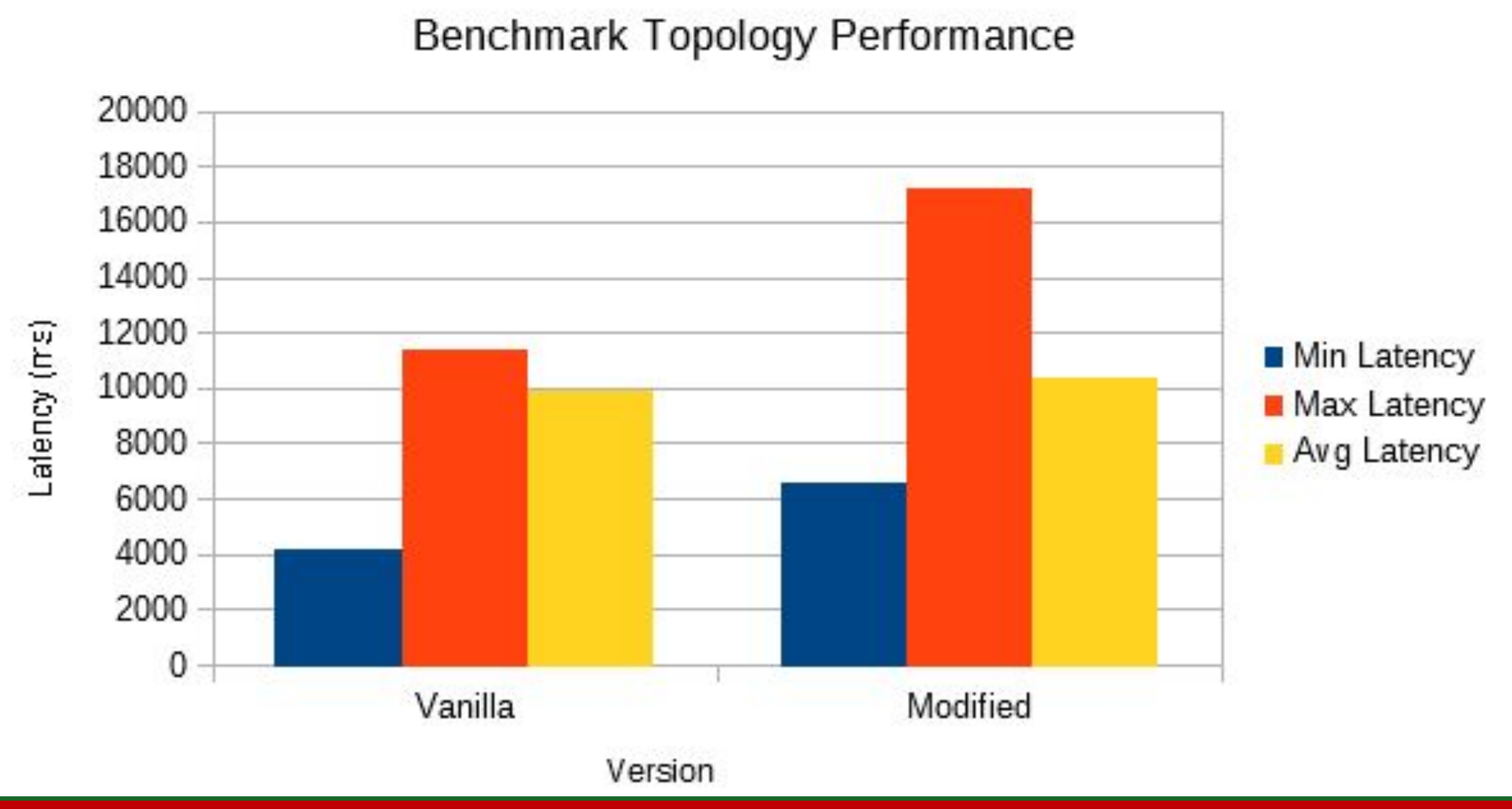
- Apache Storm
- Docker
- KVM
- Maven

Maven™

APACHE STORM™

Results

The benchmark measures the latency between when the last tuple in a time window (test) is emitted by Kafka to when it is stored in Redis. The latency is the time it took the storm cluster to finish any in-transit tuples ahead of the last tuple, process the last tuple, and store it in Redis. For both versions, a "time window" of 10,000 tuples was used, and the latency on the arrival of the 10,000th tuple was measured. The average of 2000 of these latencies produced our final result. Vanilla used the default storm scheduler while the modified version used our custom scheduler. Both used 8 workers total.



Conclusion

The difference in average latencies over 2000 tests was **440** (**4.5% overhead**). The average of 2000 tests demonstrates that adding containers and isolation resulted in minimal overhead. This is consistent with other research on Docker performance. Thus, our modified system provides protection and isolation from potentially untrusted (3rd party) code with minimal impact on performance.

References

Chintapalli, S., Dagit, D., & Evans, B. (2015). Benchmarking Streaming Computation Engines at Yahoo! Retrieved from https://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at

Mouat, A. (2016). *Using docker developing and deploying software with containers.* O'Reilly Media.

Acknowledgements

A special thank you to Dr. David Lillethun for all his guidance, support and understanding.
Funded by the Seattle University College of Science and Engineering.