Real-time Stream Processing for Sensing Environments

Shrideep Pallickara
Department of Computer Science
Colorado State University

October 27, 2015

Outline

- Challenges in Stream processing
- **Neptune**
  - Key Features
  - Profiling refinements
- Contrasting Neptune with Storm

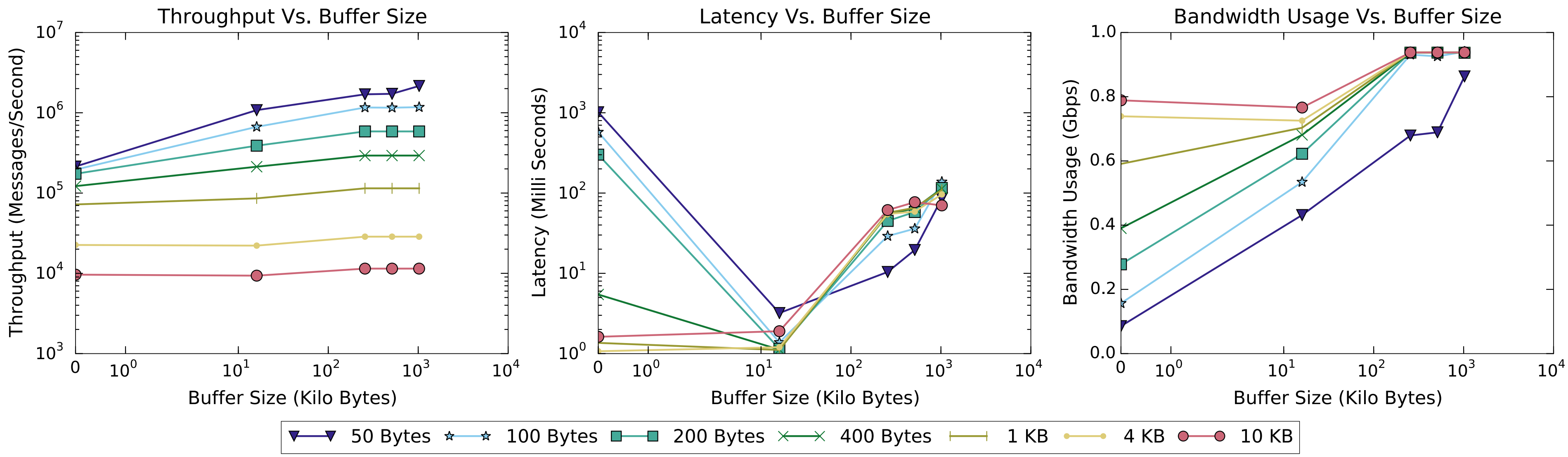
Stream Processing: Challenges in Sensing Environments

- Small packets
- Arrival rates
- Context switches
- Object creations
- Buffer Overflows

Neptune: Key Features

- Builds on Granules (http://granules.cs.colostate.edu)
- Real-time, multi-stage stream processing
  - Stateful computations
  - Communications: direct, publish/subscribe, P2P
- Refinements
  - Application buffering
  - Batched scheduling
  - Object reuse
  - Backpressure for flow control
  - Entropy-based dynamic message compactions

Impact of application layer buffer size on Performance



Batched scheduling: Impact on context switches

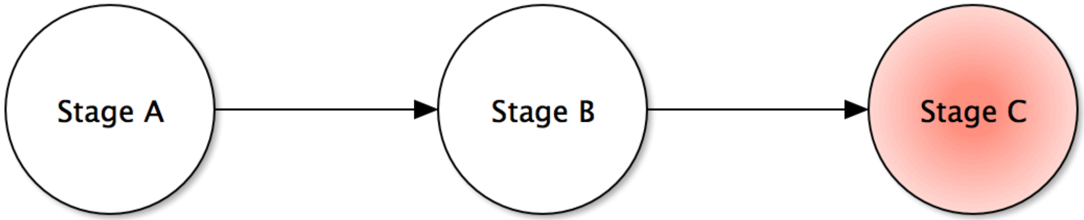
Mode	Context Switches (Tracked every 5 seconds)	
	Mean	Standard Deviation
Batched Scheduling	4085.2	91.8
Individual message processing	89952.5	1086.5

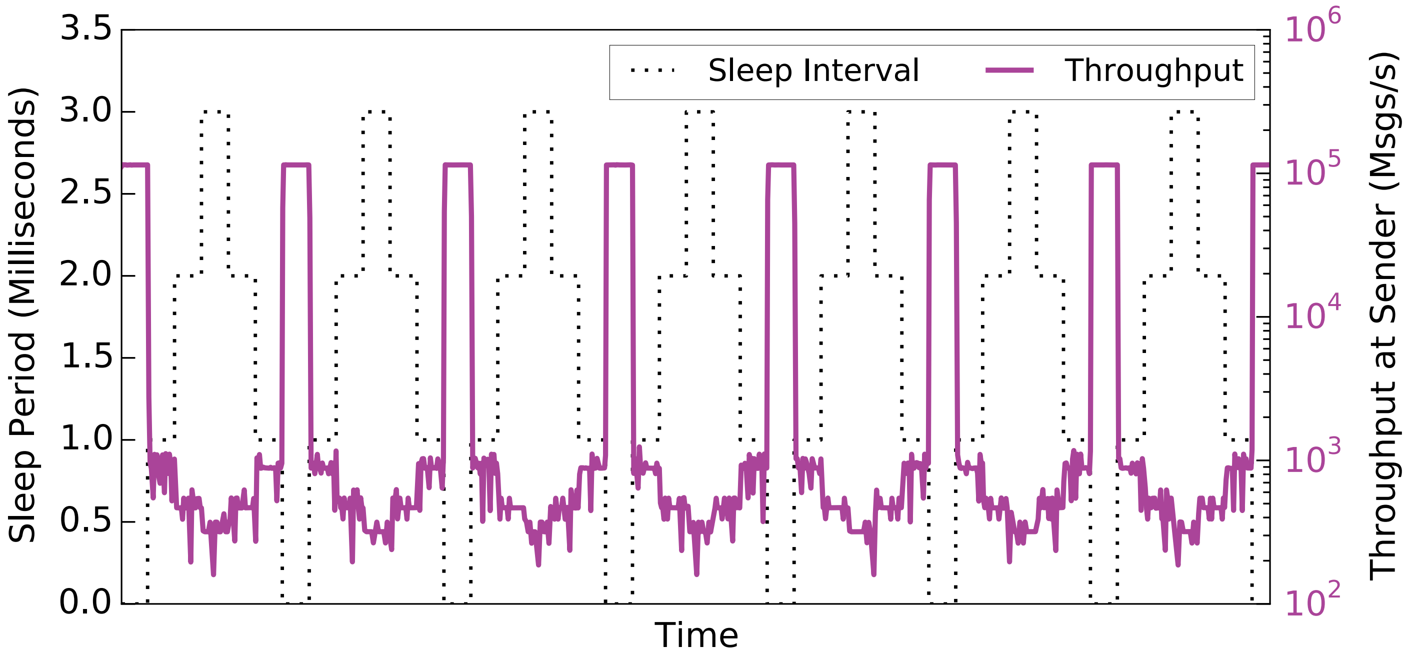
N.B: The number of context switches is 22 times lower with batched scheduling

Object Reuse: Without it, the JVM spends too long coping with memory pressure

	Time spent on garbage collection
Without Object Reuse	8.63%
With Object Reuse	0.79%

Backpressure: It's better to throttle upstream than to be overrun downstream





N.B: Data emission rate at stage 1 is adjusted according to the processing rate at stage 3.

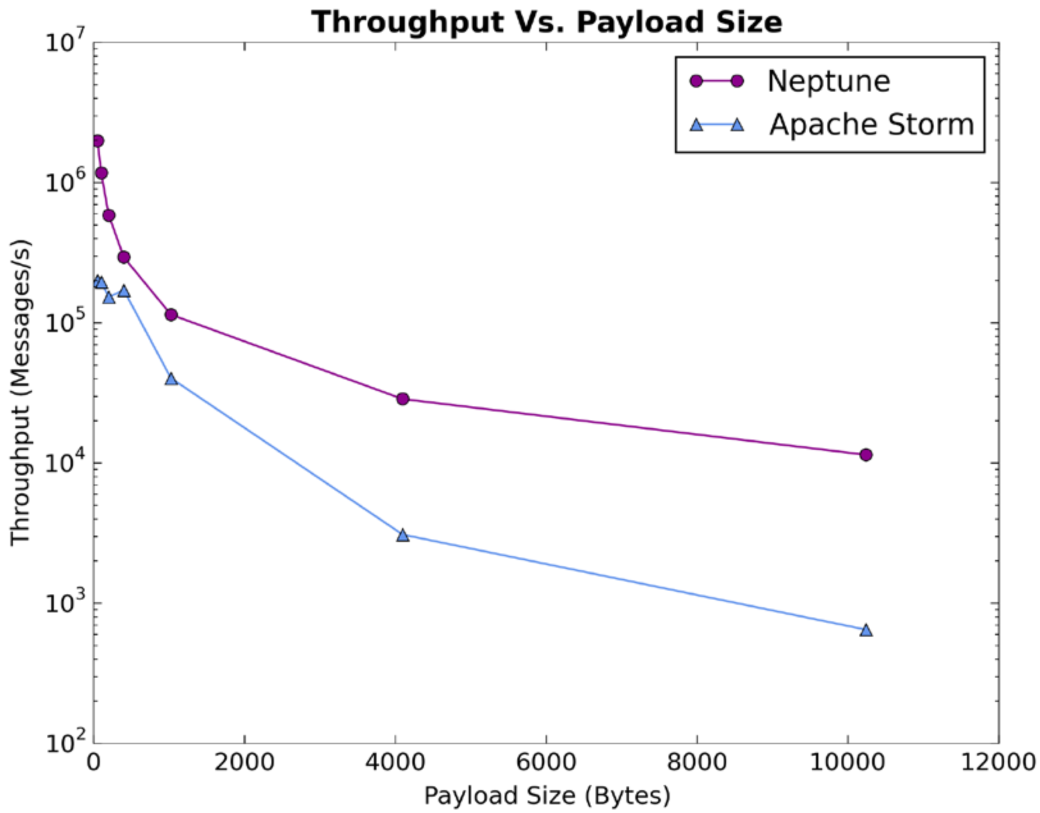
Contrasting Neptune & Storm

October 27, 2015

Evaluation

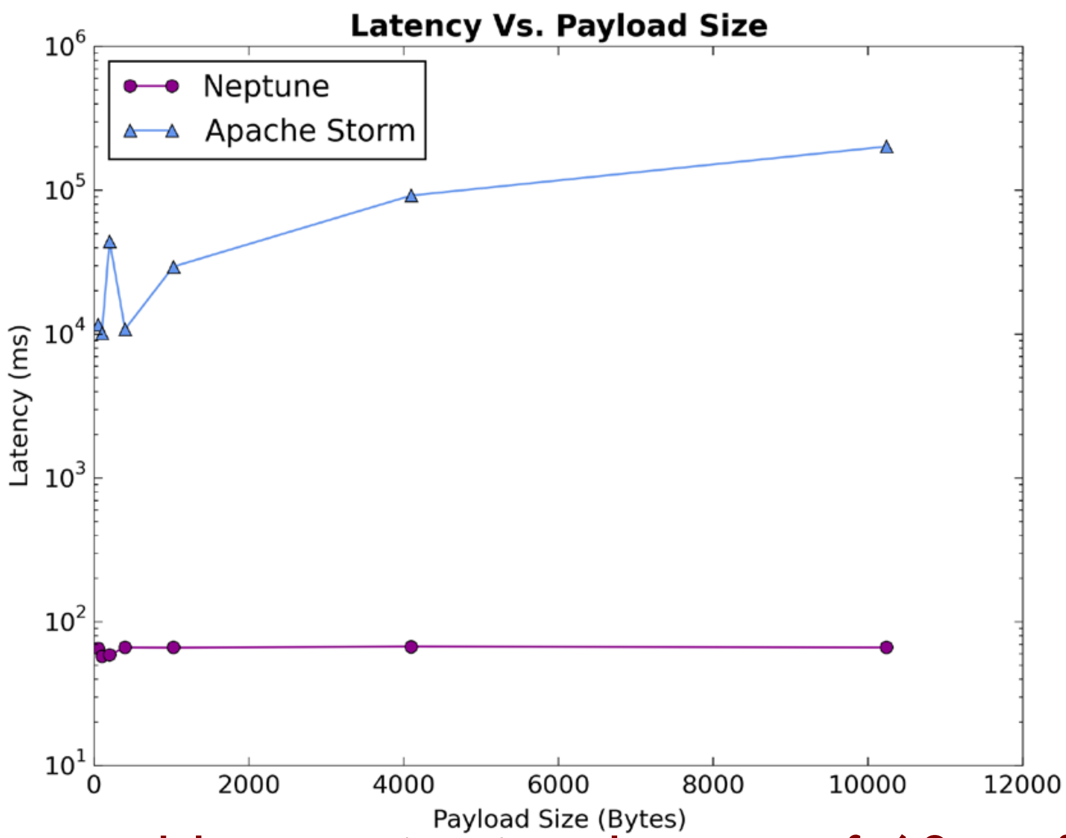
- Metrics
  - Latency, throughput, and bandwidth utilization
  - CPU and memory utilization
- Two sets of benchmarks
  - 3-stage relay based stream processing
  - Manufacturing equipment ACM DEBS Grand Challenge
- Storm was optimized for high throughput

Throughput: Neptune outperformed Storm by an order of magnitude



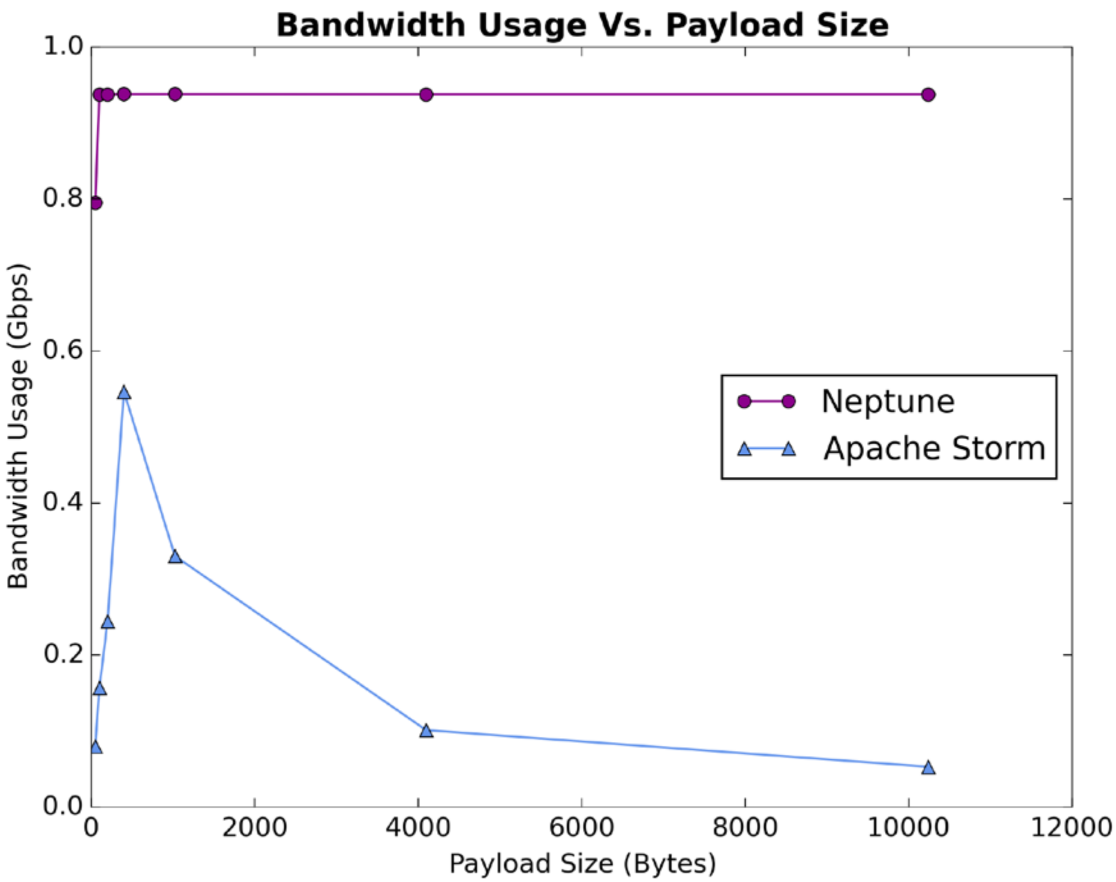
N.B: Neptune was able to achieve ~2 million messages/s (50 bytes) which is 10 times higher than Storm.

Latency: Neptune provides consistent performance



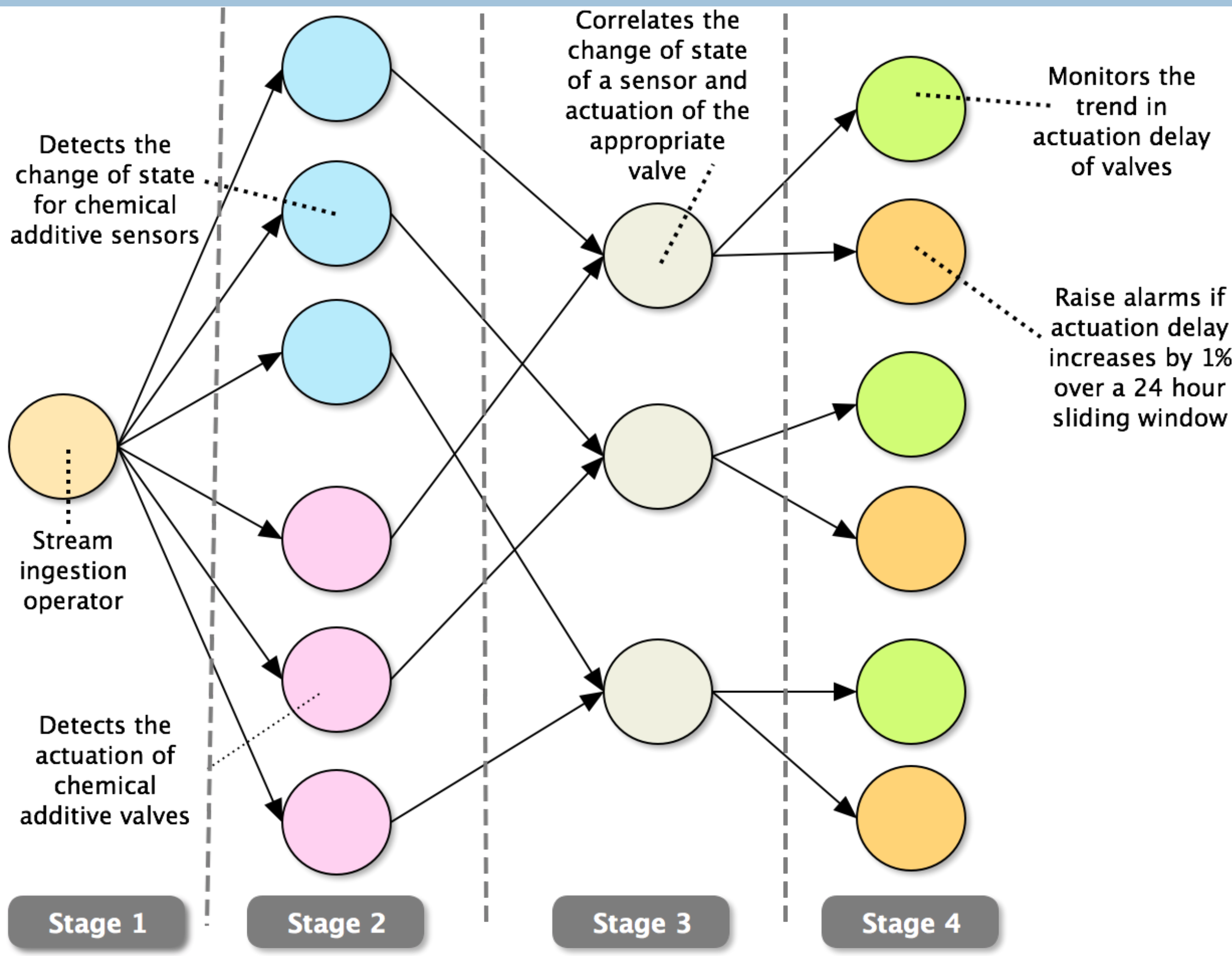
N.B: Neptune was able to maintain a latency of 68 ms for 99% of the messages for 100 bytes messages.

Bandwidth utilization

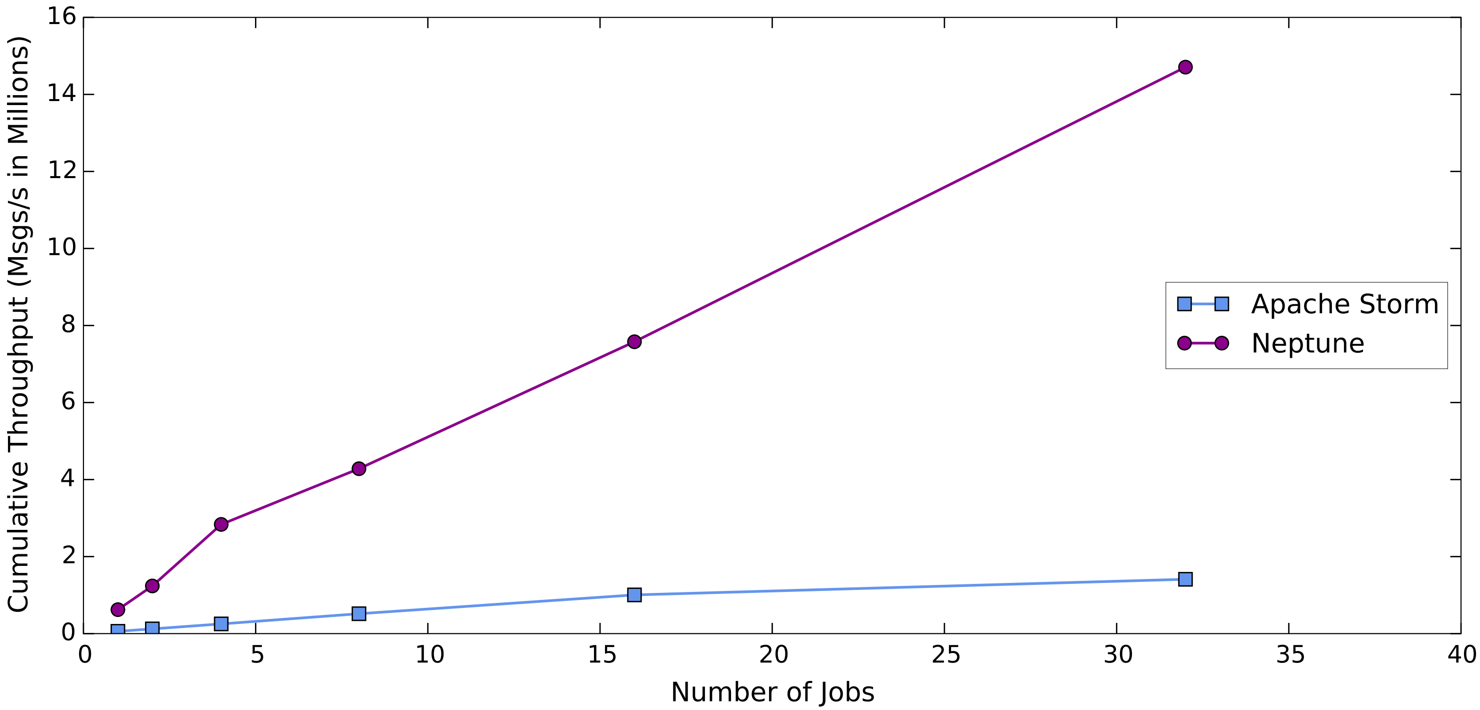


N.B: Neptune was able to maintain a 94% bandwidth consumption for message sizes > 50 bytes.

Equipment monitoring use case

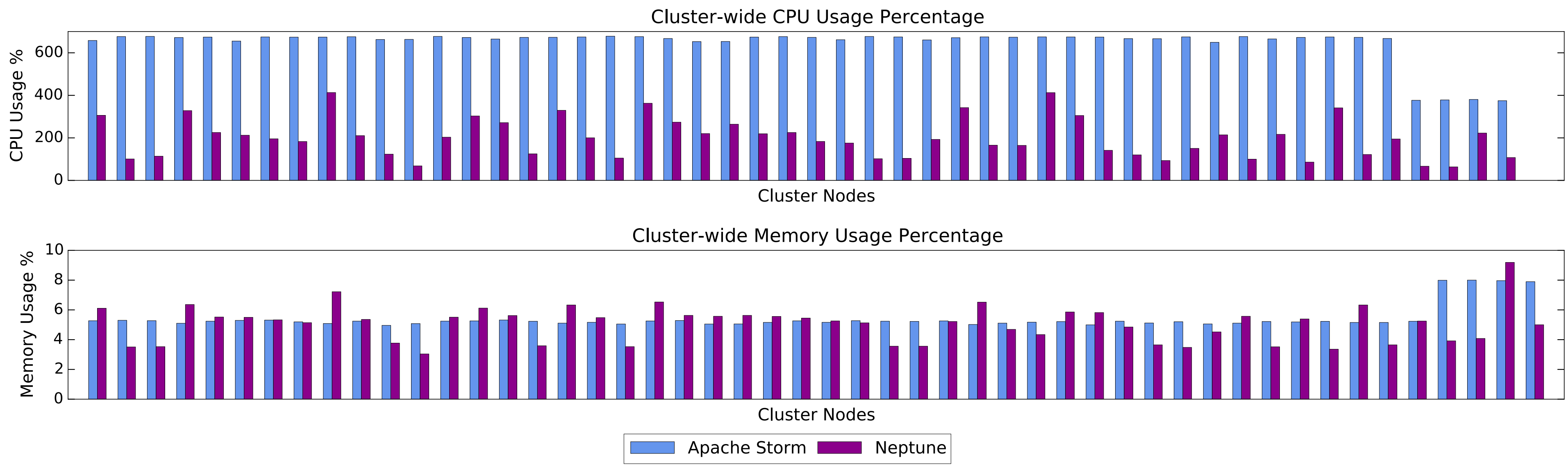


Throughput: Manufacturing equipment use case



N.B: With 32 concurrent jobs, Neptune's cumulative throughput is 8 times higher than Storm's .

Contrasting resource consumption: Manufacturing equipment use case



- Storm's cluster-wide mean CPU utilization is 3.2x higher than Neptune's (t-test: p-value < 0.0001)
- There is no significant difference in memory consumption
  - (t-test: p-value = 0.0863)
- **Neptune does more with less**

Conclusions

- Stream processing requires a holistic framework that accounts for CPU, memory, network, and kernel issues
- Reusing objects reduces memory utilization and forestalls kernel issues
- Buffering utilizes bandwidth effectively
- Backpressure management alleviates memory pressure as well

Acknowledgements

- Graduate students contributing to Granules and Neptune
  - Thilina Buddhika
  - Matthew Malensek
  - Ryan Stern