A Dynamic Resource Demand Analysis Approach for Stream Processing Systems

Johannes Rank, M.Sc.

Chair for Information Systems (Prof. Dr. Helmut Krcmar)
Technische Universität München
johannes.rank@tum.de
Motivation

- **What is Event Stream Processing?**

  - Examples: Market feed processing, infrastructure monitoring, fraud detection (Stonebraker, M., et al. 2005)

- **Importance of Performance** for Stream Processing
  - For SPS performance is not only a quality of service aspect, but **vital** for the whole business scenario to succeed (Stonebraker, M., et al. 2005)
  - **Crucial need for building scalable systems** to enable the processing of vast amounts of streamed data (Bedini et al. 2013)
Stream Processing Systems Diversity

- **Stream Processor Engines (SPE)**
  - Flink
  - Apex
  - IBM Infosphere Streams
  - SAP Hana Streaming Analytics
  - Apache Spark Streaming
  - Apache Storm

- **Stream Processing Application (SPA)**

<table>
<thead>
<tr>
<th>SPE</th>
<th>Language Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flink</td>
<td>Java, Python</td>
</tr>
<tr>
<td>Apex</td>
<td>Java, JavaScript, Python, R, Ruby</td>
</tr>
<tr>
<td>IBM Infosphere Streams</td>
<td>SPL (Streams Processing Language), Java, C++</td>
</tr>
<tr>
<td>SAP Hana Streaming Analytics</td>
<td>CCL (Continuous Computation Language)</td>
</tr>
<tr>
<td>Apache Spark Streaming</td>
<td>Java, Python</td>
</tr>
<tr>
<td>Apache Storm</td>
<td>Java, Python, Ruby, Javascript, Perl</td>
</tr>
</tbody>
</table>

**How to compare performance between systems?**
Related Work

- Related work focuses on **throughput** and **latency**
  - Throughput and latency (Chintapalli, S., et al. 2016)
  - Maximum sustainable throughput (Karimov et al. 2018)
  - Latency measurement for individual processing stages (Dongen et al. 2018+2020)

  ➢ Easy to measure
  ➢ But no insights into the resource demands

- Resource efficiency becomes increasingly important for stream processing
  - IoT edged computing with limited resources (e.g. Raspberry Pi 3) (Xhafa, F., et al. 2020)
  - Cost advantage in large-scale deployments
Measuring resource demand of individual operations of the streaming application and the engine itself …
- without language centric tools (e.g. Java Profiler),
- dynamically (applicable for running applications),
- without source code knowledge
- and production safe (non-disruptive performance overhead)
Toolchain

1. Collect all PIDs and TIDs of the SPE and Application
2.1 Trace consumed events/data in bytes
2.2 For all PIDs identified in step 1, count the number of cycles and instructions via PMC
2.3 For the PID of the streaming application sample stack traces at 999 Hz
3+4 Combine the results from 2.1 – 2.3 to calculate the absolute CPU demand for the SPE and application, as well as the individual cpu/byte demand for every processing task
Technology

- **eBPF** (Extended Berkley Package Filter) – Step 2.2
  - Added to the Linux Kernel in release 3.18 (KernelNewbies 2014)
  - Allows to process events in Kernel space
  - Bpftrace is a high-level language for eBPF
  - Enables efficient stack sampling (Phase 2.3) and Workload tracing (Phase 2.1)

Before eBPF

- **Kernel Event**
- **Tracepoint Record**
- **Perf Buffer**
- **Periodically copy to user space**
- **Parse Record**
- **Result / Histogram**

With eBPF

- **Kernel Event**
- **BPF Program**
- **Copy to user space once**
- **Format Output**

Gregg, B. (2019)
Technology

- **PMC** (Performance Monitoring Counters) – Step 2.2 (Gregg, B. 2019)

  - Programmable counters on the CPU
  - Dedicated registers on the CPU to collect performance metrics
    - Counting the number of cycles or instructions costs practically no performance overhead

  - PMCs need to be supported by a hypervisor in virtualized environments
    - Supported by Xen
    - Available in AWS since 2017

  - Access to PMC via the perf_events utility
  - BPF tracers may call the perf_events utility to access PMC information

Gregg, B. (2019)
Experiment

- Execute the Yahoo streaming benchmark (Chintapalli, S., et al. 2016) and measure the performance demand of Apache Flink in a single-node configuration
- Measured with two load variants 2k events/s and 4k events/s
Experiment

- Methodnames are obtained via the java symbol names (requires jdk-debug package)
- For each processing task the actual consumed CPU instructions can be collected
- Results are consistent for measurements >30min (distribution of CPU consumption among tasks)
- Minor processing tasks such as the „filter“ are not visible due to their neglectable performance impact
- No considerable performance overhead during measurement

![Graph showing CPU instructions for processing 1 Byte from Kafka]

<table>
<thead>
<tr>
<th></th>
<th>2k Instr</th>
<th>2k IPC</th>
<th>4k Instr</th>
<th>4k IPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>810 bil</td>
<td>0.73</td>
<td>1661 bil</td>
<td>0.75</td>
</tr>
<tr>
<td>SPE/Cluster</td>
<td>13.5 bil</td>
<td>0.29</td>
<td>12.5 bil</td>
<td>0.29</td>
</tr>
<tr>
<td>SPE/Client</td>
<td>1.0 bil</td>
<td>0.24</td>
<td>1.9 bil</td>
<td>0.23</td>
</tr>
</tbody>
</table>

Average CPU Instructions
Conclusion

✓ Dynamically applicable (but JVM symbol translation requires startup parameter)
✓ No source-code knowledge required (task dependency cannot be reverse-engineered)
✓ Small performance overhead during monitoring (when sampling rate <= 999 Hz)
✓ Broad support of different SPEs (eBPF part of Linux Kernel)
✓ Extensive insights into the actual resource consumption of SPE and SPA

- Major operations are visible but low performance operations might be neglected (e.g. Filter operation)
- Sampling induces high disk utilization after monitoring for dumping the stacktrace (spare resources in production scenarios necessary)
Future Work

Yahoo Streaming Benchmark
- Fully integrate toolchain into the Yahoo Streaming Benchmark
- Benchmark the resource efficiency of contemporary SPS

Performance Prediction
- Yielded metrics can be integrated into a model-based performance prediction approach
- Example: Scalability predictions based on the Palladio Component Model (Becker, S., et al. 2009)
References


Thank you for your attention!

Questions?

mail: johannes.rank@tum.de