Predicting Scaling Efficiency of Distributed Stream Processing Systems via Task Level Performance Simulation

Johannes Rank, Maximilian Barnert, Andreas Hein, Helmut Krcmar
Technical University of Munich
85748 Garching, Germany
{johannes.rank, maximilian.barnert, andreas.hein, helmut.krcmar}@tum.de

Abstract

Stream processing systems (SPS) are a special class of Big Data systems that firms employ in (near) real-time business scenarios. They ensure low-latency processing through a high degree of parallelization and elasticity. However, firms often do not know which scaling direction: horizontally, vertically, or mixed, is the best strategy in terms of CPU performance to scale those systems. Especially in cloud deployments with a pay-per-use model and cluster sizes that can span dozens of cores and machines, firms would profit from more accurate measurement-based approaches. In this paper, we show how to predict the CPU consumption of Apache Flink for different scaling scenarios using the Palladio Component Model. Our approach models the individual streaming tasks that make up the application and parametrizes it with fine-grained CPU metrics obtained by combining BPF profiling and querying the CPU's performance measurement unit. Through this "task-level model approach", we can achieve highly accurate predictions, despite using a simple model and only requiring a few measurements for parametrization. Our experiment also shows that we achieve more accurate results than an alternative approach based on regression analysis.

1 Introduction

Stream processing systems (SPS) are designed to continuously process large volumes of data at high speed and with minimum latency. To achieve this, SPS offer two flexible scaling options to cope with increased load scenarios: vertical and horizontal scaling. Vertical scaling refers to adding more resources to an existing machine (also known as scale-up). Horizontal scaling refers to adding workers/nodes (we will use both terms interchangeable) and distributing the workload among the cluster (also known as scale-out). Both directions also come with a performance drain. Scale-up often increases the overhead on a single machine due to increased state management, while scale-out often results in protocol overhead [5]. To date, the most common way to determine the most CPU-efficient configuration of SPS is to measure each setting individually. However, especially in cloud deployments for which costs are accounted on a per-use basis, such an approach is time-consuming, costly, and becomes increasingly infeasible the larger the cluster gets. For this reason, we propose an approach that predicts the CPU resource consumption for different cluster configurations via a simulation approach, based on the Palladio Component Model (PCM) [2]. We focus on developing a simple and fast-to-implement approach that focuses on a fine-grained depiction of the streaming application's inner workings by parameterizing the individual streaming tasks that make up the application. In this paper, we present our modeling approach and demonstrate in an experiment how we predict the resource consumption of Apache Flink, as one of the most prominent streaming engines, for different cluster configurations.

2 Modeling and Parametrization

How well an SPS can scale vertically or horizontally depends on various factors. However, the streaming application itself is the most significant factor for scaling. The scaling direction affects each type of streaming task to a different extent. For example, stateful processing, as performed by windowing operations, often requires that events are grouped by a key beforehand. A horizontal scaling approach hence results in a re-partitioning of the events among the workers to ensure that the same node processes all data items with the same key. This induces overhead concerning serialization and network transmission. Hence, understanding the internal performance behavior of the streaming application and how it is influenced by different scaling directions, is a key insight when simulating an SPS. We demonstrate our approach based on the Yahoo Streaming Benchmark (YSB) [4]. As depicted in Figure 1, the YSB is a full application benchmark that features a chain of common streaming tasks (e.g. Deserialize, Filter, Projection, Join, Window) including a *KeyBy* to re-partition the data stream among workers. The performance will be considerably influenced by the number of worker nodes, making the YSB an ideal candidate for our simulation. As a streaming engine, we use Apache Flink, a sophisticated engine widely used in the industry. We use a

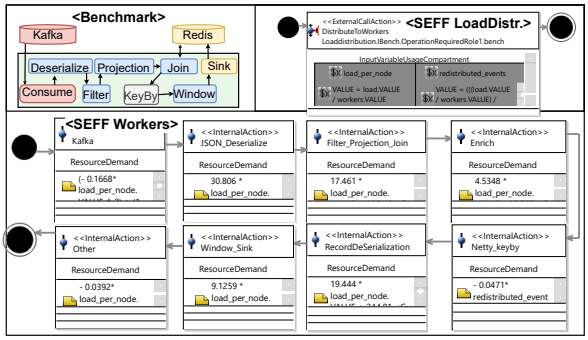

Figure 1: The YSB's Service Effect Specifications

constant workload of 600k events per second (e/s), as this results in solid base utilization of 60%.

In this work, we have six cluster configurations available that are defined as **N<x>_C<x>** where **N** defines the number of workers and **C** the available physical CPU cores on a single node. All configurations use 12 physical CPU cores but spread these resources among a variable number of nodes, giving us the cluster configurations N1_C12, N2_C6, N3_C4, N4_C3, N6_C2 and N12_C1. This reflects a common real-world deployment scenario e.g. when deciding between using two MS Azure NC6s instances (6 vCPUs / 4.254$/h) or one NC12s (12 vCPUs / 8.508 $/h). Our usage model uses an open workload with an interarrival time of 1/600, because events arrive at a fixed rate and do not re-enter the system. The system entry call uses load and workers as variables. Load specifies the data rate (e/s) and workers the number of nodes for which the simulation shall be applied. We only have a single resource container that specifies 48 vCPUs with a processing rate of 4.2 GHz. We only require a single System- and Allocationmodel. Hence, the user only needs to change the worker and load parameters as part of the Usage Scenario to simulate different cluster sizes. We have two basic components. The "Load distribution" is a virtual helper component to mimic the load distribution that Kafka provides via its partition concept. Due to our simplified PCM model (that does not model different resource environments), we model the resource demand in dependence of the *load_per_node*. This means the more workers we have (scale-out), the less events a single worker receives. As shown in Figure 1 the SEFF of the load distribution calculates *load_per_node* as well as the number of *redistributed_events*. *Redistributed_events* is the number of events that a single node in the cluster will send and receive as part of the *KeyBy* task (the YSB uses a uniform distribution). The workers' SEFF finally depicts the actual task-level performance behavior. We model one internal action for every streaming task of the YSB. In addition, the *Other* task accounts for general framework overhead such as Garbage Collection. Modelling each task, allows us to specify individual resource demand functions, which will greatly increase the prediction accuracy. For the parametrization we will perform measurments based on the toolchain we presented in [3]. The toolchain profiles the application using the extended Berkley Package Filter (eBPF), which samples stacktraces in kernel space (greater efficiency than other profilers) and also queries the *performance measurement unit* of the CPU. Both results will be combined to obtain the CPU cycles consumed by the individual tasks of the streaming applicaiton. We need to measure the minimum and maximum scale-out configurations (N12_C1 and N2_C6), and the single node configuration (N1_C12). As shown in Figure 2 the first row depicts the total CPU resources consumed by the different clusters (vCPUs). Most tasks profit from a scale-up scenario, which is especially evident when looking at the *KeyBy* task. Since our simplified PCM approach simulates the CPU consumption of a *single node* (the user will need to manually multiply the prediction result with the number of workers to get the total utilization), we also need the resource demand for a single node. The first step is hence to divide for each measurment the total consumption by the respective number of nodes (**N**). As our resource demand will be specified as consumed CPU *cycles* per *event*, we need to transform both axis. On the x-axis we translate the cluster configuration as the load_per_node (600k e/s / nodes), whereas on the y-axis we will multiply the used cores with the processing rate (4.2 GHz) and divide by the load (600k) to get the consumed cycles for a single event [Step2]. Finally, based on the three measurments we can calculate the trendlines. Depending on the curve, we will approximate via a linear or a polynomial function (choose highest R-squared). These will be our resource demand functions, while **(x)** is the number of processed events on a single node of the respective cluster configuration (e.g. 600 for N1, 50 for N12). For the *KeyBy* trendline we will use the *redistributed_events* rather than the number of processed events.

3 Experiment

In this experiment, we predict the CPU consumption for the cluster configurations N3_C4, N4_C3, and N6_C2 and compare the results with actual measurements. Our test environment is an IBM E870 server with 40 Power8 cores (4.2 GHz) and SMT4 setting. We also perform a regression analysis based on the measured total CPU consumption to test if the task-level approach has advantages over a simple black-box approach. As illustrated in Figure 3 we used the same three measurement points (N1_C12, N2_C6 and N12_C1) to build a logarithmic and a polynomial regression function. This way we predicted the CPU utilization via our task-level approach (TCUT), the logarithmic regression (LCUT), and the polynomial regression (PCUT). The RMSEs are calculated by comparing the predicted values with the measured CPU utilization (MCUT). As shown by the RMSE,

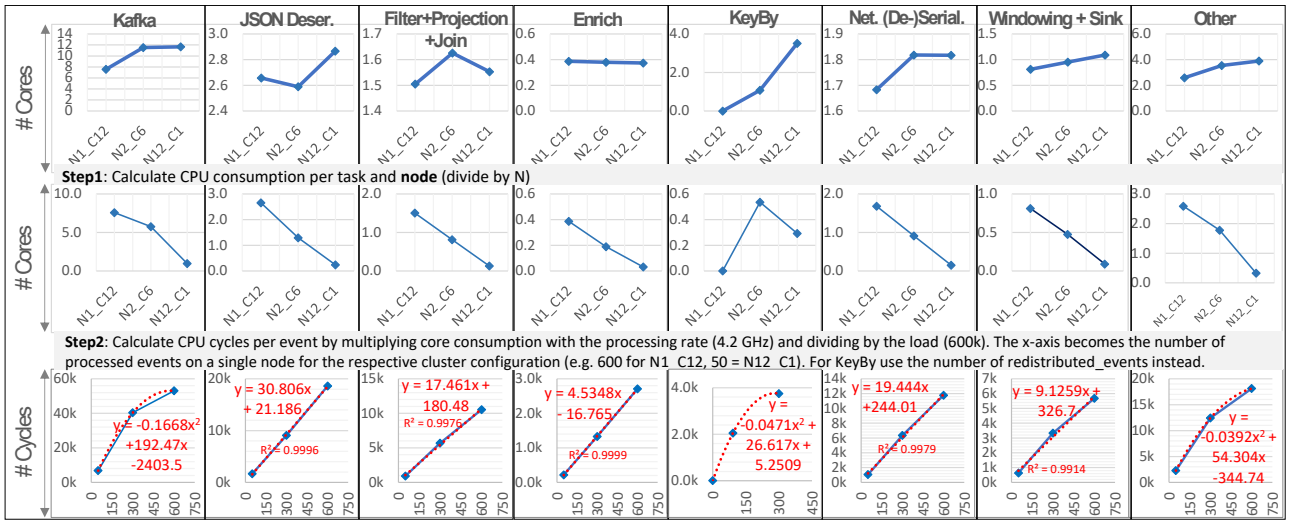

Figure 2: Parametrization Approach

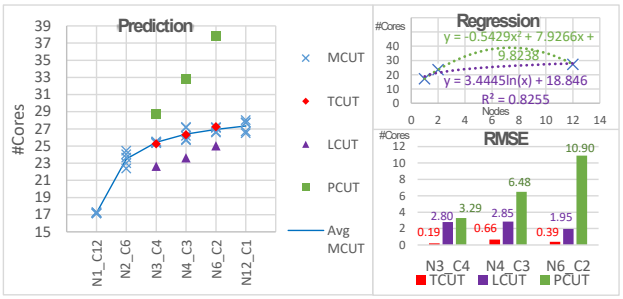

Figure 3: Prediction Results

TCUT achieves accurate results for all three cluster configurations, while both regression methods were not sufficient.

4 Related Work

One of the first approaches to predicting SPS performance was proposed in [1]. Based on PCM, they simulated the response-time of Spark Streaming for an upscaling scenario and achieved quite accurate results with prediction errors between 0.67% and 21.14%. A similar approach was presented in [6], predicting the response time for different framework configuration settings. However, both approaches differ from ours in that they do not predict CPU utilization and but use a simple wordcount example instead of a streaming application that is richt in its task variations. While there is research that simulated the utilization of Apache Storm's bolts using stochastic petri nets [7], they did not translate performance as total CPU utilization and did not show how they estimated the parametrization. Also, they did not predict any scaling scenarios.

5 Conclusion

In this paper, we proposed a new approach for predicting the CPU consumption of SPS for different scaling scenarios. By modeling the streaming application on task-level, we intended to build a simple PCM model that is fast and easy to implement while still achieving accurate prediction results. In an experiment, we demonstrated our approach by simulating the CPU consumption of Apache Flink when running the YSB in different scaling scenarios. We only required the measurement of three configurations to obtain the model parametrization. We then simulated three different cluster configurations and achieved highly accurate results. Our approach also outperformed an alternative prediction via a regression analysis.

References

[1] J. Kroß and H. Krcmar. "Modeling and simulating Apache Spark streaming applications". In: *Softwaretechnik-Trends* 36.4 (2016), pp. 1–3.

[2] R. H. Reussner et al. *Modeling and simulating software architectures: The Palladio approach*. MIT Press, 2016.

[3] J. Rank, A. Hein, and H. Krcmar. "A Dynamic Resource Demand Analysis Approach for Stream Processing Systems". In: *Softwaretechnik-Trends* 40.3 (2020), pp. 40–42.

[4] S. Chintapalli et al. "Benchmarking Streaming Computation Engines: Storm, Flink and Spark". In: *2016 IEEE Intern. Parallel and Distributed Processing Workshops*, pp. 1789–1792.

[5] K. e. a. Hwang. "Scale-Out vs. Scale-Up Techniques for Cloud Performance and Productivity". In: *2014 IEEE 6th Intern. Conference on Cloud Computing Technology and Science*, pp. 763–768.

[6] J. Lin et al. "Modeling and simulation of Spark Streaming". In: *AINA 2018*, pp. 407–413.

[7] J.-I. Requeno, J. Merseguer, and S. Bernardi. "Performance analysis of Apache Storm applications using stochastic Petri nets". In: *IRI 2017*. IEEE, pp. 411–418.