Implentation of an simplistic Interface for Big Data Workload Scheduler in Kubernetes

Implentation of an simplistic Interface for Big Data Workload Scheduler in Kubernetes

Lukas Schwerdtfeger

A thesis submitted to the
Faculty of Electrical Engineering and Computer Science
of the
Technical University of Berlin
in partial fulfillment of the requirements for the degree
Bachelor Technische Informatik

Berlin, Germany
December 22, 2015

Main supervisor:

Prof. Dr. habil. Odej Kao, Technical University of Berlin

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den

Zusammenfassung

Kurze Zusammenfassung der Arbeit in 250 Wörtern.

Abstract

Short version of the thesis in 250 words.

Acknowledgements

This chapter is optional. First of all, I would like to...

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The current population is producing more and more data. This creates an excellent opportunity for many businesses. Businesses willing to profit from collected data by using it to improve their sales strategies have to collect and store GigaBytes and upwards to ExaBytes of data. With storage costs becoming more affordable, companies are even less likely to toss away potential valuable data, creating so-called Data Lakes.

Collecting data is only the first step. It takes many stages of processing, through aggregation and filtering, to extract any meaningful information. Usually, the sheer mass of collected data makes it not very useful, to begin with.

Unfortunately, when working with ExaBytes of data, it is no longer feasible to work on a single machine. Especially when dealing with a stream of data produced by a production system and the information collected from yesterday's data is required the next day, or ideally immediately.

Scaling a single machine's resources to meet the demand is also not feasible. It is either very expansive or might just straight up not be possible. On the other hand, cheap commodity hardware allows the scaling of resources across multiple machines is much cheaper than investing in high-end hardware or even supercomputers.

The complexity of dealing with a distributed system can be reduced using the abstraction of a cluster. A **Cluster Resource Manager** is used, where a system of multiple machines forms a single coherent cluster that can be given tasks to.

Stream Processing of Data across such a cluster can carry out using Stream or Batch Processing Frameworks, such as Apache Spark or Apache Flink. These Frameworks already implement the quirks of dealing with distributed systems and thus hide the complexity.

The problem is that multiple Batch Jobs running on a single cluster need resources that need to be allocated across the cluster. While Resource Allocation is the Task of the Cluster Resource Manager, the manager usually does not know how to allocate its resources optimal and often requires user *(TODO: 1)* to specify the resources that should be allocated per job. This usually leads to either too little resources being allocated per job, starving jobs and increasing the runtime, or more often over-committing resources and thus leaving resources in the cluster

unused.

Another problem that arises is the fact that even though the reoccurring nature of Batch Jobs, not all BatchJobs use the same amount of Resources. Some are more computationally intensive and require more time on the CPU, while others are more memory intensive and require more or faster access to the machine's memory. Others are heavy on the I/O usage and use most of the system's disk or network devices. This shows up in vastly different Job runtime (also total runtime) depending on the Scheduling of Batch Jobs across the Cluster.

Finding an intelligent Scheduling Algorithm that can identify reoccurring Jobs and estimate their resource usage based on collected Metrics and thus create optimal scheduling is not an easy task. It also requires a lot of setup when dealing with a Cluster Resource Manager.

1.1.0.1 TODO:

1. not just a user but the cluster user which is submitting the job

1.1.0.2 Open:

◇ How much detail is required here?

1.2 Problem Description

(TODO: Cluster Resource Manager, like YARN, were focused around Batch-Application because they existed because of Apache Hadoop/Map-Reduce ecosystem)

Cluster Resource Managers, like YARN, emerging from Apache Hadoop, were centered around Batch Frameworks.

With the rise of Cloud Computing and all the benefits that come with it, companies were quick to adopt new cloud computing concepts. The Concept of a Cluster Resource Manager introduced a notion of simplicity to those developing applications for the cloud. A Cluster Resource Manager now managed many aspects that used to be handled by dedicated Operations-Teams.

Kubernetes, a Cluster Resource Manager that was initially developed by Google, after years of internal use, provided an all-around approach to Cluster Resource Management for not just Batch-Application. The global adoption of Kubernetes by many leading companies, led to the growth of the ecosystem around it. Kubernetes has grown a lot since and has become the new industry standard, benefiting from a vast community.

Old Batch-Application-focused Cluster Resource Managers that used to be the industry standard are being pushed away by Kubernetes. Unfortunately, vastly different Interfaces or Scheduling Mechanism between other Cluster Resource Managers usually block the continuation of existing research done in the field of Batch Scheduling Algorithms.

Finding an efficient scheduling algorithm is a complex topic in itself. Usually, the setup required to further research existing scheduling algorithms is substantial. Dealing with different Cluster Resource Manager further complicates continuing on already existing work.

1.2.0.1 TODO:

1.2.0.2 Open:

- ◇ This sections contains a lot of text that may be better suited to the introduction section, but i don't really now what else to put in here
- ◇ Not happy with the ending of this chapter, like introduction it's really only one paragraph at the end that explains the intended contribution of this work

1.3 Goal of this Thesis

1.3.0.1 TODO:

- ◇ Implementation of easy to use Interface that would allow already Batch Job Scheduling Algorithms likes Hugo and Mary to be run with small changes, on the popular Cluster Management Software Kubernetes

1.4 Structure of this Thesis

1.4.0.1 TODO:

- ◇ Thesis starts by giving a brief background to Big Data Streaming Processing, Cluster Management Systems (Kubernetes), and Scheduling
- ◇ Discuss the Approach this thesis takes on tackling the Problem Description, by explaining how scheduling in Kubernetes works and what it takes to Extend Kubernetes (using the Operator Pattern)
- ◇ Implementation Details that a worth mentioning:
  - ○ An architectural Overview.
  - ○ The Process of designing an Interface
  - ○ The Operator that is used to extend Kubernetes
  - ○ Changes that had to be made to existing Algorithms (and their tests)
- ◇ How the work of thesis is evaluated, by testing it's functionality, comparing results from previous work and finally outlining its limitations
- ◇ Comparing the Work that was done to current State of the Art Technology like the Batch Scheduling Framework Volcano and comparing to Scheduling approaches that are not available on Kubernetes
- ◇ A final Conclusion, with a note on future work, that is missing from the current implementation or requires rethinking.

2
Background

2.1 Big Data Stream Processing

2.1.0.1 TODO:

- ◇ Explain why cluster computing is required to deal with the Big Data Problem
- ◇ Explain what makes distributing computation across a cluster hard
- ◇ Explain the Value of already existing Big Data Stream Frameworks like Spark and Flink
- ◇ Explain on a high level how these work
    - ○ Explaining the DAG is required in order to later differentiate between DAG-level scheduling and "Pod"-Level scheduling
    - ○ Driver and Executor Pods
- ◇ Mention the use on Kubernetes using the Spark and Flink Operator

2.1.0.2 Input

- ◇ mehr generisch
- ◇ unterschied stream/batch

2.2 Scheduling

2.2.0.1 TODO:

- ◇ Explain what Scheduling is
- ◇ Different kind of scheduling
    - ○ DAG Scheduling done by Spark (Not what this thesis is about)
    - ○ POD Scheduling done by the Cluster Resource Manager
        - ∗ Co-Location
        - ∗ Packing

- ◇ Explain why Scheduling is Important
  - ○ Co-Location Problem
  - ○ Low Resource Usage (Graph from Google)
  - ○ Results from Hugo/Mary Paper

2.2.0.2 Open:

- ◇ Should maybe start a bit less specific about this Thesis and find more Information about Scheduling in general or is it fine if the Background section starts of general and tailors towards the topic of my thesis?

2.3 Cluster Management Systems

2.3.0.1 TODO:

- ◇ Explain what a Cluster Resource Manager is doing
  - ○ Abstraction of using a single cluster as a single Machine
  - ○ Managing given resources making it scalable by adding more machines
- ◇ Show what are the differences between YARN and Kubernetes
  - ○ YARN: Emerging from Hadoop was design to Work with Batch Applications
  - ○ Kubernetes: All Round Cluster Manager, with a Big Community

2.3.0.2 OPEN:

- ◇ Where do I explain why I am using Kubernetes, this is already required to be part of the Problem Definition?

3

Approach

3.1 Scheduling in Kubernetes

3.2 Extending Kubernetes using the Operator Pattern

4

Implementation

4.1 Architecture

4.2 Designing the Interface

4.3 Operator

4.4 Changes to existing Algorithm

5
Evaluation

5.1 Testing

5.2 Comparing to baseline Runtime

5.3 Limitations

5.4 Discussion

6

State of the Art

6.1 Volcano

6.2 Non Kubernetes

7

Conclusion and Future Work

7.1 Conclusion

7.2 Future Work

Bibliography