Master of Information Engineering

Universidad de Los Andes

Master's Thesis

Network Aware Elastic Key-Splitting in Distributed Stream Processing Systems

Diva Mercedes Martínez Laverde

Universidad de los Andes

Master of Information Engineering

Universidad de Los Andes

Master's Thesis

Network Aware Elastic Key-Splitting in Distributed Stream Processing Systems

Author:	Diva Mercedes Martínez Laverde
1st examiner:	ToFind
2nd examiner:	ToFind
Assistant advisor:	Nicolás Cardozo Álvarez Ph.D.
Submission Date:	June 2019

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

June 2019 Diva Mercedes Martínez Laverde

Acknowledgments

If someone helped you or supported you through your studies, this page is a good place to tell them how thankful you are.

"People sometimes ask me if it is a sin in the Church of Emacs to use vi. Using a free version of vi is not a sin; it is a penance. So happy hacking"

-Richard Stallman

Abstract

This document will serve as an example to you, of how to use LaTeXto write your CSE Master's Thesis. It will have examples and recomendations, and hopefully a few laughs. Because this is the abstract, it will have to convince you that this template is something you want to use. It has been proven, that without using this template, writing your thesis will be much more dificult. The template is based on previous work, and has been improved apon and updated. The result of this template is a modern latex template that everyone can contribute to and use for their studies of CSE @ TUM.

Some more great abstract tips can be found here: Great Abstract tips

Contents

1 Introduction

2 Preliminaries

This section is a primer on stream processing. It covers the formalization of streams and related concepts, as well as the basics of the main characteristics that shape the general architecture of a Data Stream Processing System (DSPS). Subsequently, it provides an overview on the most common Quality of Service metrics, and how these are affected by architectural decisions, particularly regarding elasticity and distribution. It concludes with the definition of load balancing and the techniques used to handle this challenge, with a great emphasis on key splitting.

2.1 Data Stream Modeling

A stream is defined as an unbound sequence of tuples $\langle a_{\{i\}}, \tau \rangle$. The $a_{\{i\}}$ are attributes of the tuple and may be of variable length or of fixed width in a structured data context. In the later, the $a_{\{i\}}$ would follow an schema S which describe the expected data type and content of each attribute. τ is the timestamp of each data point.

2.1.1 Operators

Streams are processed by functional units commonly known as operators or processing elements. Typical examples of operators include one-to-one mappings, filters and joins. Operator are classified as stateless or stateful.

A stateless operator will process a tuple regardless of the previously seen items. Some examples are filters and simple one-to-one transformations, projections and constant functions. Stateless operators may produce a result taking into account values of an attribute. In the classic word-count example each item will produce the tuple $\langle seenword, 1 \rangle$. The fundamental property of such operators is their lack of knowledge of the composition of previous tuples or their results.

In contrast, a stateful operator will store the information of previous tuples, or the results it has produced for each item or the block itself. In the word-count example the second stage stores for each word the accumulated count and updates the result with each new seen occurrence. In this scenario, the stored information is an aggregated result. It is possible for the operator to need the items itself. For example, the calculation of a moving median of the last n data points.

2.1.2 Directed Acyclic Graphs

The way data flows from one operator to another is defined in an abstract manner using a directed acyclic graph (DAG). The edges of the DAG represent the streams, and nodes represent both the operators which transform data and the sources and sinks as well. Sources are data producers such as sensors, IoT devices and social media, whose data is consumed by the system. Sinks are the final consumers of the process. The later are commonly web applications or storage systems.

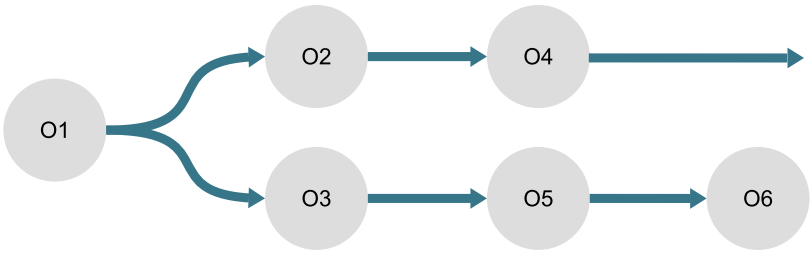

Figure 2.1: Abstract workflow

Given the task of counting the amount of hashtags a user has used, figure 2.1 is a possible DAG representing the work involved. $O1$ is the source that continuously feeds tweets into the system. $O3$ transforms the tweets removing the non-hashtag components of the tweet and generating ordered pairs of user and hashtag appearing together. $O5$ groups the resulting items in order to update the cumulated count. $O2$ and $O4$ might be part of an unrelated task (for instance, counting the tweets of only verified accounts). This is possible since a DAG may have several sinks.

A DAG is an abstract view of the data flow and general workflow. However the actual implementation runs in a parallel, distributed system, where each operator can be replicated across physical machines to obtain better performance metrics.

2.1.3 Physical Plan

The introduction of distribution in stream processing allows operators to run on different physical nodes. Data parallelism gives the opportunity of processing more than one item simultaneously. Together they bring the concept of sub-stream processing. Figure 2.2 show how the DAG described by figure 2.1 might be deployed in a parallel and distributed environment.

In this example, $O3$ is a stateless operator. Therefor, the transformation can be performed on every item regardless of the operator instance receiving it. In contrast, $O5$ is a stateful operator and works in a way that data needs to be repartitioned depending on which instance has the information of the item's key. Note that such difference has consequences on the patterns for data transportation between instances. Data from $O1$ to $O3$ travels on a one-to-one way, and a repartitioning one from $O3$ to $O5$

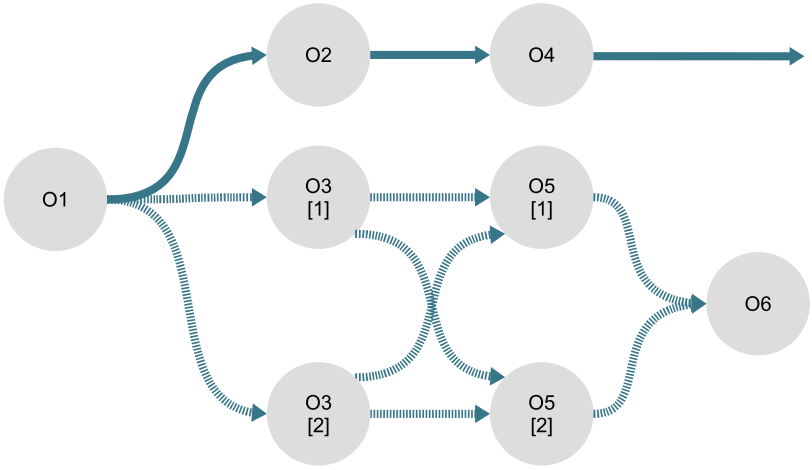

Figure 2.2: Physical plan

The process through which figure 2.1 becomes figure 2.2 includes a series of optimizations. It is possible to reorder, combine, divide and eliminate operators and still obtain the same desired results. (Hirzel, Soulé, Schneider, Gedik, & Grimm, 2014) provides a comprehensive survey on these optimizations.

2.1.4 Parallelization in Stream Processing

Explicar que el paralelismo se puede evidenciar en dos modos a grandes rasgos: paralelismo de tareas o de datos. Explicar brevemente paralelismo de tareas y entrar en detalle en paralelismo de datos Explicar qué es elasticidad

Decir que el paralelismo, en conjunto con la distribución ponen sobre la mesa características interesante de los sistemas de procesamientos de datos, o hacen que determinadas decisiones del modelo afecten sus capacidades de paralelismo. Explicar cuáles son estas caracterísicas: Type of SP System, Programming Model, Sub-Stream Processing, Modeo de insfraestructura, Manejo de estado en operadores. (Röger & Mayer, 2019) Decir por qué en cada clasificación se escogió el modelo que se escogió para trabajar bajo las garantías del sistema.

Parallelization and Elasticity methods (Röger & Mayer, 2019)
- Task parallelization (Röger & Mayer, 2019)
- Shuffle grouping (Röger & Mayer, 2019)
- Key partitioning functions (Röger & Mayer, 2019)
- Key-based splitting: State Management and Algorithms (Röger & Mayer, 2019)
- Operator Elasticity methods (Röger & Mayer, 2019) - Operator parallelization methods (Röger & Mayer, 2019)

2.2 QoS Metrics

QoS-Related Challenges(Chakravarthy, 2009)

2.3 Load Balancing in the context of elasticity and distribution

Problem description: Load Balancing (Hirzel et al., 2014), Load Balancing (Röger & Mayer, 2019)

State management (considerations): State in Operators (Röger & Mayer, 2019). State management (Röger & Mayer, 2019): Safety and Profitability (Schneider, Hirzel, & Gedik, 2013)(Röger & Mayer, 2019)

Other optimizations

Comparison between most common engines taking into account all that was said in this section before Most common Engines (Kamburugamuve, Fox, Leake, & Qiu, 2013) - Storm (Kamburugamuve et al., 2013) (Röger & Mayer, 2019)
- Heron (Röger & Mayer, 2019)
- Spark (Röger & Mayer, 2019)
- Flink (Röger & Mayer, 2019)

Conclusion: Open Research Problems (Schneider et al., 2013)

3 Motivation

Key splitting is importatn for load balancing in the context of data stream processing applications. However, most key spliting works don't take into account the bahaviur of their algorithms in the precense of elasticity

4 Related Work

5 Solution

References

Chakravarthy, S. (2009). *Stream data processing : a quality of service perspective : modeling, scheduling, load shedding, and complex event processing*. New York: Springer.

Hirzel, M., Soulé, R., Schneider, S., Gedik, B., & Grimm, R. (2014, mar). A catalog of stream processing optimizations. *ACM Computing Surveys*, *46*(4), 1–34. Retrieved from https://doi.org/10.1145%2F2528412 doi: 10.1145/2528412

Kamburugamuve, S., Fox, G., Leake, D., & Qiu, J. (2013). Survey of distributed stream processing for large stream sources. *Grids Ucs Indiana Edu*, *2*, 1–16.

Röger, H., & Mayer, R. (2019, apr). A comprehensive survey on parallelization and elasticity in stream processing. *ACM Computing Surveys*, *52*(2), 1–37. Retrieved from https://doi.org/10.1145%2F3303849 doi: 10.1145/3303849

Schneider, S., Hirzel, M., & Gedik, B. (2013). Tutorial. In *Proceedings of the 7th ACM international conference on distributed event-based systems - DEBS '13*. ACM Press. Retrieved from https://doi.org/10.1145%2F2488222.2488268 doi: 10.1145/2488222.2488268