Plumb: Efficient Processing of Multi-User Pipelines (Poster)

Poster October 12, 2018; USC/ISI Technical Report ISI-TR-731, January 2019

Abdul Qadeer and John Heidemann *USC/ISI* *{aqadeer, johnh}@isi.edu*

As the field of big data analytics matures, workflows are increasingly complex and often include components that are shared by different users. Individual workflows often include multiple stages, and when groups build on each other's work it is easy to lose track of computation that may be shared across different groups.

We propose *Plumb*, a workflow system for multi-stage workflow where parts of computation and output are shared across different groups. Plumb focuses on blocked, streaming workflows, a middle ground between large-file batch processing and small-record streaming. A particular challenge with this problem domain is structural and computational skew, where the computation of different stages and of different blocks in a stage can vary by a factor of ten due to differences in the work or data.

Plumb's first goal is to identify duplication of computation and storage that can occur when different groups share components of a pipeline. When different users are responsible for different portions of the workflow, work done in common stages will be duplicated if each user assumes they begin with raw input, particularly as the workflow evolves over the course of development. This problem of computational sharing was recently recognized at Microsoft [2]; we identify duplication both in computation and in storage of intermediate results. While databases sometimes save and share intermediate results, automated discovery is more challenging in today's loosely structured big-data workflows, where processing modules are largely opaque to the system.

The second problem we take on is *skew*. Prior work has identified data skew, where many data items fall into one processing bin, slowing the overall workflow [1]. We identify and address two new types of skew: computational skew and structural skew. Computational skew occurs when a bin of data takes extra long to process, not necessarily because there is more data, but because the data interacts with the processing algorithms to take extra time. Structural skew occurs when one stage of the processing pipeline is noticeably slower than other stages.

We address both of these types of skew in Plumb by scheduling additional processing elements when one data block or one stage falls behind. Plumb decouples processing for each stage of the workflow, buffering output when required and allowing each stage to be scheduled independently. However, to avoid the cost of data buffering, Plumb also allows stages to run concurrently when they are well matched. This decoupling also addresses computational skew, since additional computation can be brought to bear when specific data inputs take extra time.

Plumb is designed for *large-block, streaming* workloads. Traditional map-reduce has focused on batch processing, and systems such as Spark [3] consider streams of small records. We have identified a class of applications that involve long-term streams of data, but where the processing requires examination of large blocks of data (say, 10 to 1000 megabytes of data at a time) to capture temporal or spatial locality, to integrate with existing tools, and to support fault tolerance and recovery in long-running data processing. Applications that require large-block data preclude use of adaptive sharding schemes to present skew. We have designed Plumb to support large-block, streaming workloads and exploit this "middle ground" where per-data scheduling is possible.

In this poster we propose *Plumb*, a framework for processing of a multi-stage pipeline. Plumb integrates pipelines contributed by multiple users, detecting and eliminating duplication of computation and intermediate storage. It tracks and adjusts computation of each stage, creating more processing instances as required to accommodate both structural and computational skew. Plumb also tracks I/O-boundness of each stage and generate alerts for users for possible merging of I/O bound stage to a CPU-bound stage. Plumb currently uses named large size files as a proxy for large-blocks.

We exercise Plumb with the processing pipeline for B-Root DNS traffic. Compared to the currently operational, hand tuned system, we expect Plumb to provide one-third the latency while utilizing 22% less CPU. Moreover, the Plumb abstractions enable multiple users to contribute to processing with minimal coordination, and it keep latency low during normal conditions, while adapting to cope with dramatic changes to traffic and processing requirements when handling denial-of-service attacks.

The contribution of this poster is to provide an organization-wide processing substrate Plumb that can be used to solve commonly occurring problems and to achieve a common goal. Plumb makes multi-user sharing a first-class concern by providing pipeline-graph abstraction. This abstraction is simple and based on fundamental model of input-processing-output but is powerful to capture processing and data duplication. Plumb then employs best available solutions to tackle problems of large-block processing under structural and computational skew without user intervention.

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-18-2-0280. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

REFERENCES

[1] Emilio Coppa and Irene Finocchi. 2015. On Data Skewness, Stragglers, and MapReduce Progress Indicators. In *Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC '15)*. ACM, New York, NY, USA, 139–152. https://doi.org/10.1145/2806777.2806843

[2] Alekh Jindal, Shi Qiao, Hiren Patel, Zhicheng Yin, Jieming Di, Malay Bag, Yifung Lin Marc Friedman, and Sriram Rao Konstantinos Karanasos. 2018. Computation Reuse in Analytics Job Service at Microsoft. In *Proc. of ACM SIGMOD International Conference on Management of Data*. ACM, Houston, TX, USA. https://doi.org/10.1145/3183713.3190656

[3] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 423–438.

SoCC '18, October 11–13, 2018, Carlsbad, CA, USA

ACM ISBN 978-1-4503-6011-1/18/10.
https://doi.org/10.1145/3267809.3275461

Plumb: Efficient Processing of Multi-User Pipelines

Abdul Qadeer
University of Southern California / ISI
aqadeer@isi.edu

John Heidemann
University of Southern California / ISI
johnh@isi.edu

Introduction

Multi-user processing pipelines of high-value data are often inefficient due to computational redundancy and choice of execution methodology. Low throughput, low cost-efficiency and higher latency are the consequences of such inefficiency.

Computational redundancy is when different users run same (or similar) processing to generate same data. Such processing or data duplication is not apparent to individual users, whose jobs run in isolation for security reasons.

Execution methodology dictates the way processing pipeline is run on a shared cluster. Users' processing usually consists of cascading steps of multiple programs to constitute a pipeline --- Each program potentially having widely different I/O and CPU needs (a problem we call *skew*). There are myriad ways to implement and run such pipelines. End-users are often not aligned with global optimization due to time or complexity and pick straightforward, but suboptimal solutions.

Our contributions and solution:
We provide an abstraction layer that captures processing and data duplication across users still keeping data security and user illusion of isolation. Additionally this view across users makes it possible to enforce currently best possible execution methodology for all jobs. We provide **Plumb system** that implements above mentioned abstraction layer for a sub-set of bigdata problems that we call *large-block streaming*. Plumb provides following services:

- Automatedly finds and removes duplicate processing across users
- Efficiently stores data to remove data-duplication via hard-link emulation
- Mitigates skew via pipeline disaggregation and dynamic compute resource assignment based on stage slowness.

Problem

Large-block streaming applications are a subset of bigdata analytics that require large amount of data consumption per program instance. B-root DNS data is an example of high-value data that is amenable for large-block streaming. Multiple users tap into it and process it to generate derivative data. There are two reasons for large-block streaming:

Reason	Example
Performance	A program doing TCP-reassembly of a DNS flow mostly has its data in single large-block.
	A compression algorithm generally has better chance to compress well on larger data.
User convenience	Desire to utilize legacy programs that work for large files instead of writing new parallel code.

Skew is a phenomenon where stages of a pipeline take arbitrarily different CPU time-to-completion. There are two sources of skew: *Structural skew* and *computational skew*. **Structural skew** is the property of the way pipeline is constructed. Any stage can exhibit unexpected increase in execution time due to unexpected data as well. We call such data based slowness **computational skew**. Such data is common for Internet services where a malicious user can send arbitrary data. Such skew is a **new source of straggler** and re-running the stage with same data elsewhere will add to the problem. Additionally adaptive data sharding is not possible due to large-blocks requirement.

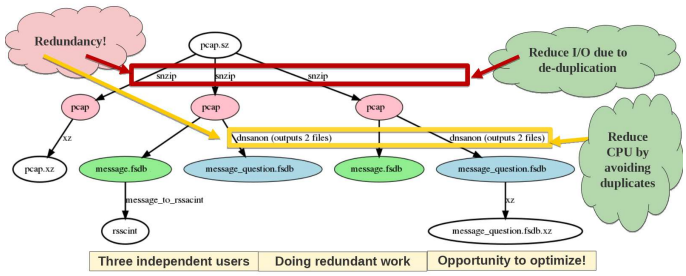
Measure of efficiency:
Our main measure of efficiency is *high throughput* and *high cost-efficiency* while providing *best-effort low latency*.

Approach

Automatedly find & remove redundant processing for multiple-users:
- Simple (input-program-output) based user facing API
- Two processing steps considered similar if they have same (input,output)
- Store similar data physically once using hard-link-emulation over HDFS
- Make a common optimized pipeline
- User illusion of their private data that they can manipulate
- Strong data security

Mitigate effects of skew:
- Disaggregate pipeline and run each step separately
- Inter stage communication is via HDFS backed queues
- (Re)Assign compute workers based on queue lengths

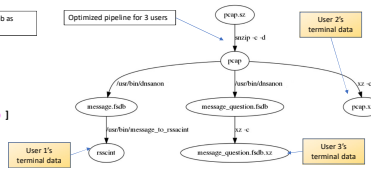
Example: 3 users processing on B-root DNS data:



Simple Abstraction to Capture Inefficiencies

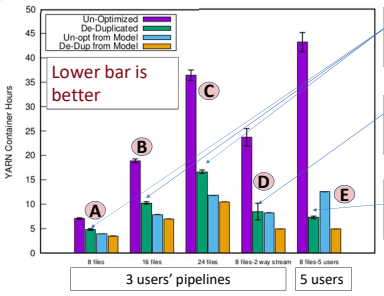
- Input and output names come from global namespace that is visible to all users
- Global namespace helps system de-duplicate processing and keeping single copy of each file
- (Input,program,output) style tuple enables system to run it separately from rest of pipeline to mitigate structural skew
- Each input and output is backed by queue. Resource assignment based on queue length mitigates computational skew

```
# 3rd User's Pipeline
-
  input:    pcap.sz
  program:  "snzip -c --d"
  output:   pcap
-
  input:    pcap
  program:  "/usr/bin/dnsanon"
  output:   [ message_question.fsdb, message.fsdb ]
-
  input:    message_question.fsdb
  program:  "xz -c"
  output:   message_question.fsdb.xz
```



High Throughput and Low Storage Use

Research question: What is effect of processing de-duplication on system throughput?
Answer: Process de-duplication substantially reduces cluster container hours. These benefits increase as load or number of redundant processing stages across users increase.

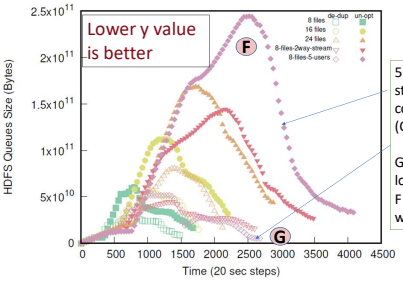


Relative benefit of de-duplication non-linearly increases with increasing load
(Compare A, B and C)

De-duplication benefits increase when a new shared stage added
(Compare A with D)

De-duplication benefits increase when 2 additional users tap data at tail of pipeline
(Compare A with E)

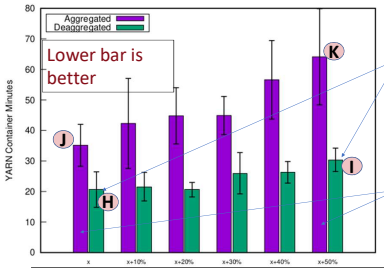
Research question: How data de-duplication via hard-link-emulation reduces storage?
Answer: Due to one physical copy of each file, storage use is low. These storage benefits increase with increasing load or increased sharing.



5 users' de-duplicated pipeline utilize 5X fewer storage as compared to duplicated configuration.
(Compare G and F)

G = 5-users' de-duplicated pipeline with input load of 8 files
F = 5-users' pipelines with duplications and with input load of 8 files

Research question: How disaggregated processing pipeline improves throughput?
Answer: Disaggregated configuration only utilizes sum of CPU need of all pipeline stages, that is optimal. Aggregated configuration uses containers proportional to the slowest stage.



2 Stage pipeline with original structural skew of x and then first stage slowed down additionally in 10% increment

Disaggregated execution configuration only slows down proportional to added skew.
(From 20 minutes to 30 minutes)
(Compare H and I)

Aggregated execution configuration slows down twice from x to x+50%.
(From 35 minutes to 62 minutes)
Aggregated execution holds onto all containers until slowest stage finishes.
(Compare J and K)

Conclusion and Future Work

- Processing de-duplication improves system throughput as load increases or sharing increases.
- Data de-duplication optimally utilize storage by keeping single physical copy and yet providing user illusion of private data. Data security is fully enforced.
- Disaggregated pipeline stage run and reassignment of compute workers based on stage queue lengths help keeping throughput high and latency lower.
- By changing similarity definition, we can utilize system for broader classes of big-data (future work). Plumb system can be used as pre-processing engine in any multi-user environment.

This research is sponsored by Air Force Research Laboratory under agreement number FA8750-18-2-0280

See our technical report ISI-TR-727 for more details: https://www.isi.edu/publications/trpublic/pdfs/isi-tr-727.pdf

Symposium on Cloud Computing 12 Oct, 2018 Carlsbad California