Optimizing Near-Data Processing for Spark

Sri Pramodh Rachuri, Arun Gantasala, Prajeeth Emanuel, Anshul Gandhi
Stony Brook University

Robert Foley, Peter Puhov
FutureWei

Theodoros Gkountouvas, Hui Lei
OpenInfra Labs

srachuri@cs.stonybrook.edu
Overview

- General Purpose Servers
  - CPU, Memory, Storage
  - Inefficient utilization
  - Fragmentation of resources
Overview

- General Purpose Servers
  - CPU, Memory, Storage
  - Inefficient utilization
  - Fragmentation of resources

- Disaggregated infrastructure (DI)
  - Optimized for specific resource
  - Reduces amount of unused resources
  - Easy rolling upgrades
  - High dependence on networks
    - Potential performance bottleneck
Overview

- Compute Optimized Cluster
  - High computation resources
  - Low storage space
Overview

- Compute Optimized Cluster
  - High computation resources
  - Low storage space

- Storage Optimized Cluster
  - High storage space
  - Low computation resources
Overview

- Compute Optimized Cluster
  - High computation resources
  - Low storage space

- Storage Optimized Cluster
  - High storage space
  - Low computation resources

- Connected over network
  - Large datasize => high transfer time
Overview

- **Compute Optimized Cluster**
  - High computation resources
  - Low storage space

- **Storage Optimized Cluster**
  - High storage space
  - Low computation resources

- **Connected over network**
  - Large datasize => high transfer time
Motivation - NDP
Motivation - NDP
Motivation - NDP

Example

Calculating total sales of a store in 1994 using records of size 1 TB from 1990 to 2020.

- Filter by year: ~30x Reduction : 34 GB
- Drop columns: 5-10x Reduction : 4-7 GB
- Sum rows : Returns int : 8 B
Motivation - NDP

Example
Calculating total sales of a store in 1994 using records of size 1 TB from 1990 to 2020.

- Filter by year: ~30x Reduction : 34 GB
- Drop columns: 5-10x Reduction : 4-7 GB
- Sum rows: Returns int : 8 B

Near Data Processing (NDP)

- Processing in storage cluster - “Pushdown”
- Reduction in transfer size
Motivation - NDP

Example

Calculating total sales of a store in 1994 using records of size 1 TB from 1990 to 2020.

- Filter by year: ~30x Reduction: 34 GB
- Drop columns: 5-10x Reduction: 4-7 GB
- Sum rows: Returns int: 8 B

Near Data Processing (NDP)

- Processing in storage cluster - “Pushdown”
- Reduction in transfer size
Motivation - NDP

Example

Calculating total sales of a store in 1994 using records of size 1 TB from 1990 to 2020.

- Filter by year : ~30x Reduction : 34 GB
- Drop columns : 5-10x Reduction : 4-7 GB
- Sum rows : Returns int : 8 B

Near Data Processing (NDP)

- Processing in storage cluster - “Pushdown”
- Reduction in transfer size
Motivation - NDP

Example
Calculating total sales of a store in 1994 using records of size 1 TB from 1990 to 2020.

- Filter by year: ~30x Reduction : 34 GB
- Drop columns: 5-10x Reduction : 4-7 GB
- Sum rows : Returns int : 8 B

Near Data Processing (NDP)

- Processing in storage cluster - “Pushdown”
- Reduction in transfer size

How to implement NDP?
Processing at resource constrained devices: Can they handle the pushdown?
How to implement and optimize NDP pushdown?
Background

Spark and HDFS without NDP
Spark and HDFS with NDP

- Operations pushed to datanodes
Spark and HDFS with NDP

- Operations pushed to datanodes
Selective Pushdown

- Some operations pushed to datanodes
Selective Pushdown

- Some operations pushed to datanodes
Selective Pushdown

- Some operations pushed to datanodes

Which operations to Pushdown?
Prior Work

NDP implementations

- Octopus [CloudCom’15]
- PushdownDB [ICDE’2020]
- λFlow [CCGRID’2019]

More related works and detailed comparisons can be found in the paper
Prior Work

NDP implementations

- Octopus [CloudCom’15]
- PushdownDB [ICDE’2020]
- λFlow [CCGRID’2019]

We aim to study performance of NDP in λFlow-like systems and then optimize it

More related works and detailed comparisons can be found in the paper
System Design

Compute Node

Datasource V2

NDP Client

Spark

Storage Node

HDFS

REST API Handler

NDP Proxy

SQLite Engine

SQLite Streamer

Spark

Hadoop
System Design

NDP Datasource API

- Spark driver for NDP Client
- Post processing of results
NDP Client

- Extracts attributes required for NDP
- Translates query into SQL command
System Design
System Design

Compute Node

Storage Node

Datasource V2 → NDP Client → Spark → NDP Proxy → HDFS → REST API Handler → SQLite Engine → SQLite Streamer
System Design

REST API Handler

- Intercepts HTTP connections from executors to datanodes
- Starts HDFS and SQLite subprocesses
System Design

Compute Node

Storage Node

- NDP Client
- DP Proxy
- HDFS
- REST API Handler
- SQLite Engine
- SQLite Streamer

SQLite Engine

- Parses CSV files to create tables
- Run operations that are pushdowned
SQLite Streamer

- Enables processing while loading data
System Design

Compute Node

Storage Node

Spark

Datasource V2

NDP Client

HDFS

NDP Proxy

REST API Handler

SQLite Engine

SQLite Streamer

- Enables processing while loading data
System Design

Compute Node

- Spark
- Datasource V2
- NDP Client

Storage Node

- HDFS
- NDP Proxy
- REST API Handler
- SQLite Engine
- SQLite Streamer

- Enables processing while loading data

More details in the paper

Which operations to Pushdown?
System Design
System Design

Analytical model - “Net-Aware”

- Predict the best pushdown strategy for an operation
- Using the parameters
  1. Estimated execution time of operations
     - At Spark
     - At HDFS
  2. Estimated time to transfer
     - Input data
     - Output data
System Design

Analytical model - “Net-Aware”

- Predict the best pushdown strategy for an operation
- Using the parameters
  1. Estimated execution time of operations
     - At Spark
     - At HDFS
  2. Estimated time to transfer
     - Input data
     - Output data

NDP of an operation is useful if time taken for
Transfer input (HDFS → Spark) + Compute at Spark
  > Compute at HDFS + Transfer output (HDFS → Spark)
**System Design**

- NDP decision for a particular operation

\[ T_c(Q_{Spark}, X_{Spark}) + T_n(D_{input}) \]

\[ > T_c(Q_{HDFS}, X_{HDFS}) + T_n(D_{output}) \]

- Decide # of operations to pushdown while initializing (design constraints)

- Once in Spark need to continue in Spark (design constraints)
Evaluation - Experimental Results

- 6 Spark nodes
  - Total **70 cores** for executors
  - Total 17.5 GB memory for executors
  - TPC-H Queries

- **10 Gbps** between the clusters
- 1 Gbps per host

- 4 Datanodes (HDFS)
  - **1-4 cores** each
    - Using Docker
  - CPU Freq - 2.67 GHz (original)
    - 1.6 GHz (underclock)
    - Using cpufrequtils
  - Replication factor - 4
  - **100 GB dataset** by DBGEN

- 1 Gbps per host
  - Changed using Tc and NetEm

More details in the paper
Evaluation - Experimental Results

- 1 Job at a time
- Varying cores in datanodes
- Oracle is the best of all selective pushdowns
- Net-aware is our policy
- No pushdown is native spark without NDP
- λFlow is full pushdown

Configuration: Number of storage nodes = 4, storage nodes clock speed = 1.60 GHz, network bandwidth between clusters = 4 Gbps.
Evaluation - Experimental Results

- 1 Job at a time
- Varying cores in datanodes
- Oracle is the best of all selective pushdowns
- Net-aware is our policy
- No pushdown is native spark without NDP
- λFlow is full pushdown

(i) 1 core
(ii) 2 cores
(iii) 4 cores

Configuration: Number of storage nodes = 4, storage nodes clock speed = 1.60 GHz, network bandwidth between clusters = 4 Gbps.
Evaluation -
Experimental Results

- 1 Job at a time
- Varying cores in datanodes

- Oracle is the best of all selective pushdowns
- Net-aware is our policy
- No pushdown is native spark without NDP
- \(\lambda\)Flow is full pushdown

Configuration: Number of storage nodes = 4, storage nodes clock speed = 1.60 GHz, network bandwidth between clusters = 4 Gbps.
Evaluation - Experimental Results

- 1 Job at a time
- Varying cores in datanodes

Full pushdown is not useful with weaker storage

- Selective pushdowns
- Net-aware is our policy
- No pushdown is native spark without NDP
- \(\lambda\)Flow is full pushdown

(i) 1 core
(ii) 2 cores
(iii) 4 cores

Configuration: Number of storage nodes = 4, storage nodes clock speed = 1.60 GHz, network bandwidth between clusters = 4 Gbps.
Evaluation - Experimental Results

- 1 Job at a time
- Varying cores in datanodes

Full pushdown is not useful with weaker storage

Gets better with more cores

- No pushdown is native spark without NDP
- λFlow is full pushdown

Configuration: Number of storage nodes = 4, storage nodes clock speed = 1.60 GHz, network bandwidth between clusters = 4 Gbps.
Evaluation - Experimental Results

- 1 Job at a time
- Varying cores in datanodes
- Oracle is the best of all selective pushdowns
- Net-aware is our policy
- No pushdown is native spark without NDP
- Full pushdown is not useful with weaker storage
- Gets better with more cores
- Net-Aware is always close to oracle

Configuration: Number of storage nodes = 4, storage nodes clock speed = 1.60 GHz, network bandwidth between clusters = 4 Gbps.
Evaluation - Experimental Results

- Changed bandwidth between clusters

(i) 1 core
(ii) 2 cores
(iii) 4 cores

Configuration: Number of storage nodes = 4, storage nodes clock speed = 1.60 GHz, network bandwidth between clusters = 1 Gbps.
Evaluation - Experimental Results

- Changed bandwidth between clusters

(i) 1 core
(ii) 2 cores
(iii) 4 cores

Configuration: Number of storage nodes = 4, storage nodes clock speed = 1.60 GHz, network bandwidth between clusters = 1 Gbps.
Evaluation - Experimental Results

- Changed bandwidth between clusters

Full pushdown is now useful with weaker storage because of the weak network link

(i) 1 core

(ii) 2 cores

(iii) 4 cores

Configuration: Number of storage nodes = 4, storage nodes clock speed = 1.60 GHz, network bandwidth between clusters = 1 Gbps.
Full pushdown is now useful with weaker storage because of the weak network link.

Net-Aware is always close to oracle.

Configuration: Number of storage nodes = 4, storage nodes clock speed = 1.60 GHz, network bandwidth between clusters = 1 Gbps.
Evaluation - Experimental Results

- Fewer nodes in HDFS and moderate bandwidth

Configuration: Number of storage nodes = 2, storage nodes clock speed = 2.67 GHz, network bandwidth between clusters = 2 Gbps
Evaluation - Experimental Results

- Fewer nodes in HDFS and moderate bandwidth

A better selective pushdown exists than Full pushdown and No pushdown

Configuration: Number of storage nodes = 2, storage nodes clock speed = 2.67 GHz, network bandwidth between clusters = 2 Gbps
Evaluation - Experimental Results

- Fewer nodes in HDFS and moderate bandwidth

A better selective pushdown exists than Full pushdown and No pushdown

Configuration: Number of storage nodes = 2, storage nodes clock speed = 2.67 GHz, network bandwidth between clusters = 2 Gbps
Evaluation - Experimental Results

- Fewer nodes in HDFS and moderate bandwidth

A better selective pushdown exists than Full pushdown and No pushdown

Net-Aware is always close to oracle

Configuration: Number of storage nodes = 2, storage nodes clock speed = 2.67 GHz, network bandwidth between clusters = 2 Gbps
Evaluation - Experimental Results

- 1 job arrives every 50 seconds
- Averaged over 10 jobs

More experimental results and simulations in the paper
Evaluation - Experimental Results

- 1 job arrives every 50 seconds
- Averaged over 10 jobs

Selective pushdown can make significant difference

More experimental results and simulations in the paper
Evaluation - Experimental Results

- 1 job arrives every 50 seconds
- Averaged over 10 jobs

Selective pushdown can make significant difference

Net-Aware is always close to optimal

More experimental results and simulations in the paper
Conclusion

Summary of our paper

- NDP implementation
- Constructed an analytical model for optimizing NDP
- Experimental evaluation – Net-Aware is close to optimal
- Implemented a discrete event simulator for large clusters (skipped in the interest of time)
Thanks for your attention

Any Questions?

Summary-

- NDP for Spark+HDFS
- Analytical Model
- Experimental evaluation
- Discrete event simulator