Towards Hybrid Programming in Big Data

Peng Wang\textsuperscript{1}, Hong Jiang\textsuperscript{2}, Xu Liu\textsuperscript{3}, Jizhong Han\textsuperscript{1}

Institute of Information Engineering, Chinese Academy of Sciences\textsuperscript{1}
University of Nebraska-Lincoln\textsuperscript{2}
College of William and Mary\textsuperscript{3}

Presented at HotCloud ‘15
July 2015
Diverse Programming models

- Hadoop
- Pregel
- Hive
- Spark
- Giraph
- GraphLab
- Dryad
- Naiad
Real-world Apps Require Multiple Phases

- Typical machine learning workflow

Preprocessing -> Feature Extraction -> Model Training -> Model Evaluation -> Optimization

Cleaning & normalization -> Cleaning & normalization -> Classification, regression, or ranking
Current Solutions

- Monolithic systems
  - e.g., Hadoop

- Same-Cluster-Sharing systems
  - e.g., YARN, Mesos
Outline

• Motivation
• Model
• Implementation
• Evaluation
• Related Work
• Conclusions & Future Work
Overview

**Unified Data Space (UDS)**

- **Dryad** - based computation (e.g., Sort)
- **Pregel** - based computation (e.g., PageRank)

1. **Compile**
   - Dryad
   - Spark
   - Pregel
   - PowerGraph

2. **Execute (SPMD mode)**

Unified Data Space (UDS)
Hybrid Program Example

// PowerGraph-based pagerank computation
func pagerank(input, output, ...) {
    // PowerGraph interfaces
    ...
    prJob = NewGraphCompute("pagerank")
    prJob.Execute(input, output,...)
}

// Dryad-based sort computation
func sort(input, output , ...) {
    // Dryad interfaces
    ...
    sortJob = NewDataflow("sort")
    sortJob.Execute(input, output, ...)
}

// hybridization
func main() {
    input = "hdfs://dataset/input"
    share = "cache://dataset/share"
    output = "hdfs://dataset/output"
    engine = NewEngine()
    engine.Start()
    pagerank(input, share, ...)
    sort(share, output, ...)
    engine.Stop()
}
UDS: Unified Data Space

- Four-spheres: HDFS (input & output), file (temporary data), cache (memory-based sharing), pipe (network communication)
- File-system like abstraction
- Access an object by URL (e.g., file://host:port/path/to/object)
• Operations: Read (URI) & Write(URI)
• Microtask behavior: Read → Process → Write
• Statelessness & Location independence
• Microtask communicate with each other via Sharing Unified Data Space
Case Study: MapReduce

MapReduce dataflow

- Read
- Write
Case Study: MapReduce

MapReduce Tasks interact via UDS
Hybrid Program Execution

```go
def main() {
    input = "hdfs://dataset/input"
    share = "cache://dataset/share"
    output = "hdfs://dataset/output"
    engine = NewEngine()
    engine.Start()
    pagerank(input, share, ...)
    sort(share, output, ...)
    engine.Stop()
}
```
Outline

- Motivation
- Model
- Implementation
- Evaluation
- Related Work
- Conclusions & Future Work
• Developed by Go language
• **UDS implementation**
  – HDFS sphere is written by wrapping the libhdfs C library using the cgo tool in Go
  – Other spheres are implemented by native go code
• **Implement prototypes of existing programming models**
  – Dryad, Spark, Pregel, and PowerGraph
  – Each prototype requires less than 2K LOC
Implementation

NewWriter("cache://worker-1:port/data")
NewWriter("file://worker-1:port/data")
NewReader("cache://worker-1:port/data")
NewReader("file://worker-1:port/data")

①②③④⑤⑥
Outline

• Motivation
• Model
• Implementation
• **Evaluation**
• Related Work
• Conclusions & Future Work
Experimental Setup

• Amazon EC2
  – 50 m3.xlarge nodes

• Comparison with Hadoop 1.2.1 & Spark (GraphX) 1.2

• Case study 1: Machine learning application
  – Principal Component Analysis (PCA) + Logistical Regression (LR)
  – Generated dataset (45GB)

• Case study 2: Graph Analysis
  – PageRank + TopK
  – Real-world graph dataset (twitter & uk-2007)
Evaluation

- Inband outperforms out-of-band by 25% to 47%
Evaluation

- Transformer shows competing performance with GraphX
- Inband provides limited improvement over the out-of-band
  - Intermediate dataset is small
Related work

- Programming models (e.g., Spark+GraphX)
  - Spark/GraphX supports pre-defined graph parallel and data parallel operations on a common data structure (i.e., RDD)
  - Transformer supports user-defined computations on the same physical dataset (i.e., file-like objects)

- Data sharing (e.g., Tachyon)
  - Tachyon is fast memory-based distributed file system
  - Tachyon is efficient for inter-job data sharing
  - In-band mechanism is designed for intra-job data sharing
Conclusions and Future Work

• Conclusions
  – Propose a common runtime unifies diverse programming models
  – Develop a hybrid programming system with promising performance
  – In-band sharing mechanism improves performance without sacrificing productivity

• Future Work
  – Integrate more existing programming models
  – Extend Transformer with transparent fault tolerance for hybrid programming
  – Improve the performance and scalability of hybrid programming
Thanks!

Q&A