**Starfish: A Self-tuning System for Big Data Analytics**

Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, Shivnath Babu
Department of Computer Science, Duke University

**Workload Optimization**

Optimization Techniques
- Data-flow sharing
- Materialization
- Reorganization

Challenges
- Interactions of above techniques among each other and with scheduling, data layout policies, and configuration parameter settings

Jumbo Operator
- Use a single MapReduce job to process multiple Select-Project-Aggregate operations over a table
- Enables sharing of scans, computation, sorting, shuffling, and output generation

**Provisioning for Hadoop Workloads**

Goal
- Make provisioning decisions based on workload requirements

Provisioning Choices
- Number of nodes
- Cluster configuration
- Network configuration

Long-term vision
- Hadoop Analytics as a Service

**Starfish Overview**

Starfish in the Hadoop Ecosystem

- Java Client
- Pig
- Hive
- Oozie
- Elastic MR

Starfish makes Hadoop MADDER and Self-Tuning
- Magnetism: attracts all sources of data
- Agility: adapts in sync with rapid data evolution
- Depth: supports complex analytics needs

Starfish Overview Components

- Goal: Find good settings for configuration parameters
- Challenges: Data opacity until processing, File-based processing, Heavy use of programming languages
- Approach: Profiler: Uses dynamic instrumentation to learn performance models (job profiles) for unmodified MapReduce programs
- Sampler: Collects statistics about input, intermediate, and output key-value spaces of a MapReduce job
- What-if Engine: Uses a mix of simulation and model-based estimation to predict job performance
- Just-in-Time Optimizer: Searches through the high-dimensional space of parameter settings

**Example analytics workload for Amazon Elastic MapReduce**

- Users (username, age, password)
- GeoInfo (ipaddr, region)
- Clicks (username, url, value)

**Workflow-aware Scheduling**

Scheduling Objectives
- Ensure balanced data layout
- Avoid cascading reexecution under node failure or data corruption
- Ensure power proportional computing
- Adapt to imbalance in load or cost of energy across data centers

Causes of unbalanced data layouts
- Skewed data
- Data-layout-unaware scheduling of tasks
- Addition or dropping of nodes without rebalancing operations

Approach
- Consider interactions between scheduling policies and block placement policies of the storage system
- Use smart scheduling to perform rebalancing automatically
- Exploit opportunities for collocating data sets

**Unbalanced data layout after executing one MapReduce job**

- Disk Usage (%)