R-Storm:
A Resource-Aware Scheduler for STORM

• Mohammad Hosseini
• Boyang Peng
• Zhihao Hong
• Reza Farivar
• Roy Campbell
Introduction

• STORM is an open source distributed real-time data stream processing system
  • Real-time analytics
  • Online machine learning
  • Continuous computation
Resource Aware Storm versus Default

• Micro-benchmark
  30-47% higher throughput
  69-350% better CPU utilization than default Storm

• For Yahoo! Storm applications:
  R-Storm outperforms default Storm by around 50% based on overall throughput.
Definitions of Storm Terms

- **Tuples** - The basic unit of data that is processed.
- **Stream** - an unbounded sequence of tuples.
- **Component** - A processing operator in a Storm topology that is either a Bolt or Spout (defined later in the paper).
- **Tasks** - A Storm job that is an instantiation of a Spout or Bolt (defined later in the paper).
- **Executors** - A thread that is spawned in a worker process (defined later) that may execute one or more tasks.
- **Worker Process** - A process spawned by Storm that may run one or more executors.
An Example of Storm topology
Intercommunication of tasks within a Storm Topology
An Example Storm Machine
STORM Topology

Physical Computer Clusters

Rack 1

Rack 2

Rack 3

STORM Topology

Spout_1
T1
T2
T3

Bolt_1
T4
T5

Bolt_2
T6
T7
T8

Bolt_3
T9
T10
Related Work

• **Little** prior work on resource-aware scheduler in STORM!

• The default scheduler: Round-Robin
  - Does not look into the resource requirement of tasks
  - Assigns tasks evenly & disregard resource demands

• Adaptive Online Scheduling in Storm (Aniello et al.)
  - Only takes into account the CPU usage!
  - Shows 20-30% improvement in performance

• System S Scheduler (Joel et al.)
  - Only accounts for processing power and is complex
Problem Formulation

• Targeting 3 types of resources
  • CPU, Memory, and Network bandwidth
• Limited resource budget for each cluster and the corresponding worker nodes
• Specific resource needs for each task

Goal:
Maximizing the overall utilization while decreasing the resources used!
Problem Formulation

• Set of all tasks $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \ldots\}$, each task $\tau_i$ has resource demands
  • CPU requirement of $c_{\tau_i}$
  • Network bandwidth requirement of $b_{\tau_i}$
  • Memory requirement of $m_{\tau_i}$

• Set of all nodes $\mathcal{N} = \{\theta_1, \theta_2, \theta_3, \ldots\}$
  • Total available CPU budget of $W_1$
  • Total available Bandwidth budget of $W_2$
  • Total available Memory budget of $W_3$
Problem Formulation

• $Q_i$: Throughput contribution of each node

• Assign tasks to a subset of nodes $N' \in N$ that minimizes the total resource waste:

Maximize $\{N' \subseteq N\} \sum_{\theta \in N'} Q_{\theta_i}$ subject to

$\sum_{\tau_i \in N'} c_{\tau_i} \leq W_1$, $\sum_{\tau_i \in N'} b_{\tau_i} \leq W_2$, $\sum_{\tau_i \in N'} m_{\tau_i} \leq W_3$

(CPU, Bandwidth, Memory)
Heuristic Algorithm

- Designing a 3D resource space
  - Each resource maps to an axis
  - Can be generalized to nD resource space
  - Trivial overhead!

- Based on:
  - $\text{min} \ (\text{Euclidean distance})$
  - Satisfy hard constraints
Problem Formulation

✗ Using binary Knapsack Problem
  • Select a subset of tasks

✓ Using complex variations of KP
  • Multiple KP (multiple nodes)
  • m-dimensional KP (multiple constraints)
  • Quadratic KP (successive tasks dependency)

→ Quadratic Multiple 3D Knapsack Problem
  • We call it QM3DKP!
  • NP-Hard!
Scheduling and intercommunication demands

1. Inter-rack communication is the slowest
2. Inter-node communication is slow
3. Inter-process communication is faster
4. Intra-process communication is the fastest
Heuristic Algorithm

• Our proposed heuristic algorithm ensures the following properties:
  1) Two successive tasks are scheduled on closest nodes, addressing the network communication demands.
  2) No hard resource constraint is violated.
  3) Resource waste on nodes are minimized.
R-Storm Architecture Overview

Topology $G = (V,E)$

- Custom Scheduler
  - R-Storm Scheduler
- GlobalState
- Statistics Server

Nimbus

New Scheduling

Supervisor

Executors

Worker Processes

Worker Nodes
Algorithm 1 R-Storm Schedule

1: procedure SCHEDULE
2:   taskOrdering ← TASKSELECTION(())
3: for each Task $\tau$ in taskOrdering do
4:   Node $n$ ← NODESELECTION($\tau$, cluster)
5:   SCHEDULE($\tau$, $n$);
6: end for
7: end procedure
Algorithms Used in Schedule

• Breadth First Topology Traversal

• Task Selection
  • Traverse the topology starting from the spouts since the performance of spout(s) impacts the performance of the whole topology.

• Node Selection
  • If first task in a topology, find the server rack or sub-cluster with the most available resources.
  • Afterwards, find the node in that server rack with the most available resources and schedule the first task on that node.
  • For the rest of the tasks in the Storm topology, we find nodes to schedule based on the Distance using the bandwidth attribute.
Micro Benchmarks

• Linear Topology
• Diamond Topology
• Star Topology

• Network Bound versus Computation Bound
Evaluation Microbenchmarks

• Used Emulab.net as testbed and to emulate inter-rack latency across two sides
• 1 host for Nimbus + Zookeeper
• 12 hosts as worker nodes
• All hosts:
  ▪ Ubuntu 12.04 LTS
  ▪ 1-core Intel CPU
  ▪ 2GB RAM+ 100Mb NIC
Storm Micro-benchmark Topologies

1. Linear Topology

2. Diamond Topology

3. Star Topology
Network-bound Micro-benchmark Topologies

(a) Linear Topology  
(b) Diamond Topology  
(c) Star Topology
Result – Network Bound Micro-benchmarks

Scheduling computed by R-Storm provides on average of around 50%, 30%, and 47% higher throughput than that computed by Storm's default scheduler, for the Linear, Diamond, and Star Topologies, respectively.
Experimental results of Computation-time-bound Micro-benchmark topologies

(a) Linear Topology
(b) Diamond Topology
(c) Star Topology
Figure 10: CPU Utilization Comparison
Computation-time-bound Micro-benchmark

For the Linear topology, the throughput of a scheduling by R-Storm using 6 machines is similar to that of Storm's default scheduler using 12 machines.
Yahoo Topologies: PageLoad and Processing Topology

• Resource Aware Scheduler VS Default Scheduler
  • Comparison of throughput
  • Resource utilization
Typical Industry Topologies Models

(a) Layout of Page Load Topology

(b) Layout of Processing Topology
Experiment Results of Industry Topologies

Experimental results of Page Load Topology

Experimental results of Processing Topology
Results: Page Load and the Processing topologies

On average, the Page Load and Processing Topologies have 50% and 47% better overall throughput, respectively, when scheduled by R-Storm as compared to Storm's default scheduler.
Multiple topologies.

24 machine cluster separated into two 12 machine subclusters.

• We evaluate a mix of both the Yahoo! PageLoad and Processing topologies to be scheduled by R-Storm and Default Storm.
Throughput comparison of running multiple topologies.

Figure 13: Throughput comparison of running multiple topologies.
Average throughput comparison

• PageLoad topology
  • R-Storm (25496 tuples/10sec)
  • Default Storm (16695 tuples/10sec)
  • R-Storm is around 53% higher

• Processing topology
  • R-Storm (67115 tuples/10sec)
  • Default Storm (10 tuples/sec).
  • Orders of magnitude higher
Conclusion

• Resource Aware Scheduler provides a better scheduling that has:
  • Higher utilization of resources
  • Higher overall throughput
Questions?