ADAPTIVE ONLINE SCHEDULING IN STORM

Leonardo Aniello, Roberto Baldoni and Leonardo Querzoni

aniello|baldoni|querzoni@dis.uniroma1.it

DEBS 2013
Arlington - 3/7/2013
AGENDA

- Intro
- What STORM is and how it works
- Our schedulers
  - topology based
  - traffic based
- Evaluation results
- Conclusions
The Big Data surge

“every day, we create 2.5 quintillion bytes of data - so much that 90% of the data in the world today has been created in the last two years alone”

THE BIG DATA SURGE

[Domo, Data never sleeps (http://www.domo.com/learn/7/236#videos-and-infographics), 2012.]
THE BIG DATA SURGE

“the business behind Big Data will globally create 4.4 million IT jobs by 2015”

[Gartner; http://www.gartner.com/newsroom/id/2207915]
Motivations

- Big Data applications are typically characterized by
  - large volumes
  - high velocity
  - extreme variety

- Classic data mining/analysis solutions hardly meet these requirements

- New solutions that scale
  - NoSQL storage systems (Cassandra, BigTable, Dynamo, ...)
  - Parallel batch processing systems (Hadoop, ...)
  - Stream processing solutions (Storm, ...)
STORM

- Storm is an open source distributed realtime computation system
  - Provides abstractions for implementing event-based computations over a cluster of physical nodes
  - Manages high throughput data streams
  - Performs parallel computations on them
- It can be effectively used to design complex event-driven applications on intense streams of data
- Currently used by Twitter (owns it!), Groupon, The Weather Channel, Taobao, etc.
An application is represented by a topology:
STORM

- A storm cluster is constituted by a Nimbus node and $n$ Worker nodes.
STORM

- A topology is run by submitting it to Nimbus
  - Nimbus allocates the execution of components (spouts and bolts) to the worker nodes using a scheduler
    - Each component has multiple instances (parallelism)
    - Each instance is mapped to an executor
  - A worker is instantiated whenever the hosting node must run executors for the submitted topology
  - Each worker node locally manages incoming/outgoing streams and local computation
    - The local supervisor takes care that everything runs as prescribed
  - Nimbus monitors worker nodes during the execution to manage potential failures and the current resource usage
Storm’s default scheduler (*EvenScheduler*) applies a simple *round robin* strategy.
STORM

- Storm’s defaults scheduler has both pros and cons:
  - + simple
  - + good load distribution
  - + does not impose any overhead at runtime
  - + general purpose
  - - topology-agnostic
  - - does not consider network load $\Rightarrow$ may negatively impact event processing latency

- Pluggable schedulers: custom schedulers can be designed and plugged in Nimbus
GOALS

- Reduce average *event processing latency* by making the scheduler adaptive to specific application characteristics.

- Desiderata:
  - general-purpose
  - small or no overhead at run time
  - may give up fair load distribution (but no overloads!)

- Outcome: two schedulers!
  - Topology-aware: offline, adapts to the general topology characteristics
  - Traffic-aware: online, adapts to actual load distribution
ADAPTIVE SCHEDULERS

▪ Rationale

- Consider two executors implementing components A and B that are connected in the topology
- The tuples that travel from A to B will be transferred
  - through slow network links if the executors are scheduled on distinct worker nodes
  - through faster inter-process communication if their executors are scheduled on different workers assigned to the same worker node
  - through very fast intra-process communication if their executors are scheduled in the same worker

In topologies where the computation latency is dominated by tuples transfer time, limiting the number of tuples that have to be sent and received through the network can contribute to improve performance.

▪ Caveat: only acyclic topologies can be scheduled
**Topology-based Scheduler**

- **Characteristics**
  - works offline (like the default scheduler)
  - takes into account connections among components in the topology
  - does not consider run-time load characteristics or groupings

- **Two phases**
  - Phase 1: Executors are ordered and assigned to slots in a specific order
  - Phase 2: Slots are assigned to worker nodes as in the default scheduler (round-robin)
**Topology-based Scheduler**

- **Executor-to-slot assignment strategy**
  - Components are partially ordered
  - It is always possible to derive a linearization $s$ from this partial order
  - The scheduler iterates over $s$: given a component $c_j$ it assigns its executors to a slot where executors of a component $c_i$, such that $c_i$ emits tuples toward $c_j$, have been already assigned
  - A tuning parameter $\beta$ is considered to avoid empty slots:
    - Executors of the $i$-th component are assigned to empty slots if $i \geq \beta \ast \text{number-of-components}$
**Topology-based Scheduler**

A < B < C
A < B < E
D < E

s={A,B,C,D,E}

Worker node 1:

- Slot 1: A, B, C
- Slot 2: B, C
- Slot 3: B, C, E
- Slot 4: D

Worker node 2:

- Slot 1: A, B, C, E
- Slot 2: B, C, E
- Slot 3: B, C
- Slot 4: D

---

ADAPTIVE ONLINE SCHEDULING IN STORM - LEONARDO QUERZONI - DEBS 2013 - 1/3/2013
TRAFFIC-BASED SCHEDULER

■ Characteristics
  ◆ works online
  ◆ adapts the scheduling to reduce network traffic among worker nodes
  ◆ scheduling is fired every time inter-node traffic can be reduced by a given %

■ Two phases
  ◆ Phase 1: apply an initial schedule (one from the default scheduler is OK)
  ◆ Phase 2: continuously monitor the system for inter-executor traffic and re-schedule whenever a performance improvement is expected
  • 2.a: assign executors to slots
  • 2.b: assign slots to workers
Traffic-based Scheduler

- Architectural modifications

Diagram:

- Scheduler plugin
- Monitoring thread
- Executor
- Worker process
- Topology $G(V,T)$, $w$
- Deployment plan
- Nimbus
- Performance log
- Worker nodes
- Slots
Traffic-based Scheduler

- Executor assignment to slots
  - consider couples of flow-connected executors by descending flow intensity
  - if none of the two is scheduled, schedule both on the least loaded slot
  - else build a triple containing
    - the least loaded slot
    - the slots where the two executors have been possibly assigned
  - check the various assignments possibilities (max 9) for the one that produces the least inter-slot traffic
  - proceed until all executors are assigned to slots

- Follow a similar approach to assign slots to worker nodes
TRAFFIC-BASED SCHEDULER

<table>
<thead>
<tr>
<th>Ex 1</th>
<th>Ex 2</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>E</td>
<td>10</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>E</td>
<td>1</td>
</tr>
</tbody>
</table>

Worker node 1

Worker node 2

Slot 1

Slot 2

Slot 3

Slot 4
EVALUATION

Setup
- 1 host for Nimbus + Zookeeper
- 8 hosts for worker nodes (5 slots each)
- 10Gb LAN

Two test topologies
- Reference topology
- DEBS’13 Grand challenge topology (reduced version of the first query)
EVALUATION - RESULTS

- Reference topology - 7 stages, 4|3|2 parallelism

![Graph showing latency over time for different topology-based and traffic-based settings with a decrease of ~15% and ~25%]
**EVALUATION - RESULTS**

- **Inter-node traffic**

![Graph showing inter-node traffic for parallelism 2|2|2 and 4|4|4.](image)

**Parallelism 2|2|2**

**Parallelism 4|4|4**
EVALUATION - RESULTS

- Scheduling flexibility - 5 stages, 5|5|5 parallelism

![Graph showing inter-node traffic (tuple/sec) vs Alpha for different scheduling methods.]

- Default
- Topology-based
- Traffic-based
EVALUATION - RESULTS

- GC-inspired topology

![Graph showing latency and inter-node traffic over time for default, Topology-based, and Traffic-based configurations.]

- Latency (ms) and inter-node traffic (tuples/s) are plotted against time (s).

- The graph illustrates the performance of different configurations over time.
CONCLUSIONS

- We presented two general purpose pluggable schedulers for Storm
  - Topology-based scheduler
  - Traffic based scheduler
- Both are aimed at **reducing event processing average latency**
- The schedulers are publicly available

http://www.dis.uniroma1.it/~midlab/software/storm-adaptive-schedulers.zip
FUTURE DIRECTIONS

- Energy-aware scheduling
  - Re-schedule topologies at run-time depending on energy consumption
  - Goal: meet specific global energy consumption rates
- What do you need
  - Per-server energy consumption measurements
  - Fine grained (in time) measurement of energy consumption
  - A way to analyze this stream of data (!)
  - A way to inject results of this analysis in the scheduler
THANKYOU!

Q&A
EVALUATION - SETUP DETAILS

- Setup
  - 1 host for Nimbus + Zookeeper
  - 8 hosts for worker nodes (5 slots each)

- All hosts
  - Ubuntu 12.04
  - CPU 2 cores at 2.8Ghz
  - RAM 3Gb
  - Disk 15Gb
  - Network 10Gb LAN
Global rate $R$

- $i$-th spout executor sets its tuple rate as

$$R_i = R(1 - V(1 - 2i))$$

- where $C_0$ is the number of executors for the first component, that is the spout itself, and $i = 0, ..., C_0 - 1$ and $V$ is a $[0-1]$ parameter

- Therefore, each spout executor emits tuples at a distinct fixed rate and the average of these rates is exactly $R$.

- The total input rate for the topology can be controlled (its value is $C_0 \cdot R$) and a certain degree of irregularity can be introduced on traffic intensity (tuned by parameter $V$)
EVALUATION - LOAD REFERENCE TOP.

- Bolts in the reference topology have been implemented so as to forward the received value with probability 1/2 and to emit a different constant value (fixed for each executor) the rest of the times.

- The traffic between executors whose communication is setup using fields grouping is affected by this mechanism since it makes the tuple rates much higher for some executor pairs.

- This choice models realistic situations where certain pairs of executors in consecutive stages communicate more intensively than others.
GOALS

Dynamic Optimization

Compile time
Operator separation
Fusion
State sharing
Algorithm selection

Submission time
Redundancy elimination
Fission
Placement

Runtime discrete
Load balancing

Runtime continuous
Operator reordering
Batching
Load shedding

Other challenges:
Settling
Accuracy
Stability
Overshoot