D-Storm: Dynamic Resource-Efficient Scheduling of Stream Processing Applications

Xunyun Liu and Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory
School of Computing and Information Systems
The University of Melbourne, Australia

December, 2017

Outline

Outline

Stream Processing

Stream Data

- Arrive continuously in real time, possible infinite
- Various data sources & various data structures
- Transient value & short data lifespan
- Asynchronous & unpredictable

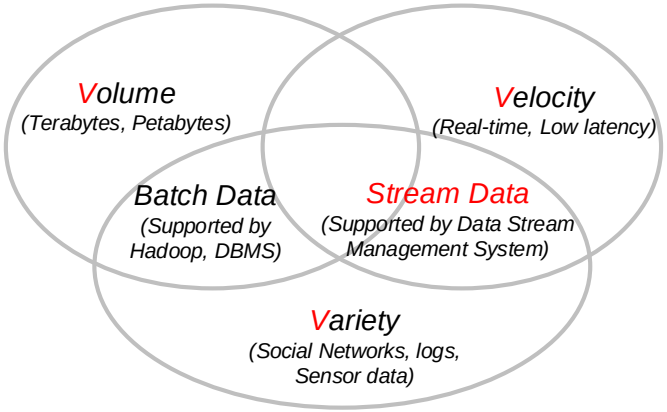


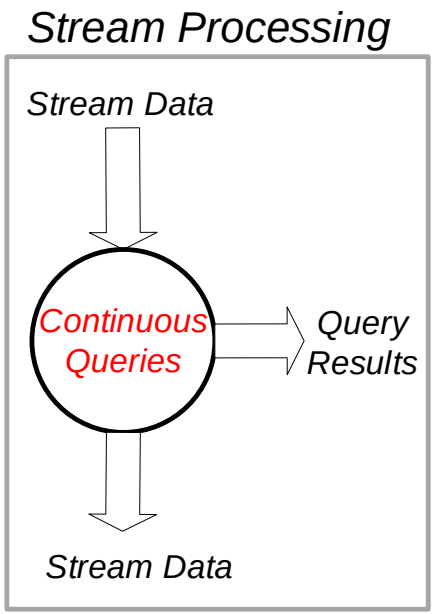
Table: Example usages of stream data versus batch data

	Batch Data	**Stream Data**
Log Analysis	What happened an hour ago?	What is happening in the system?
Billing Analysis	How much did a user spend in the last billing period?	Notify a user that the billing limit is approaching
Fraud Prevention	Audit / forensic evidence	Intervene to stop the ongoing fraud

Stream Processing

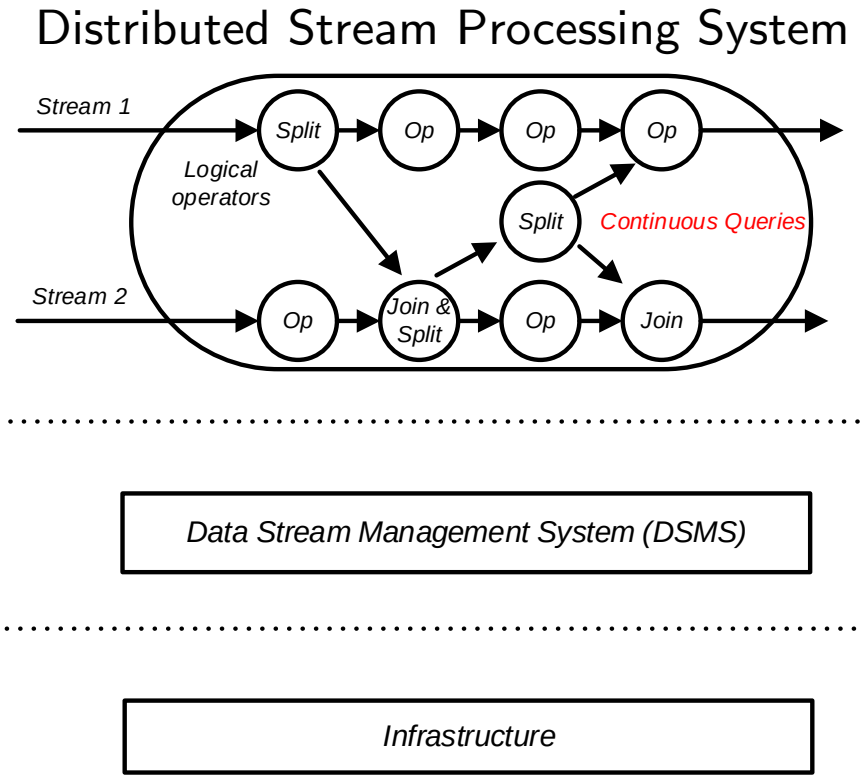
Stream processing is a technique that allows for the collection, integration, analysis, visualization, and system integration of stream data in real time, powering on-the-fly analytic that leads to immediately actionable insights.

- Process-once-arrival
  - Queries over recent data on rolling window
  - Generally independent computations
  - Incremental update of results
- Queries run continuously unless being explicitly terminated
- Suitable for latency-sensitive scenarios

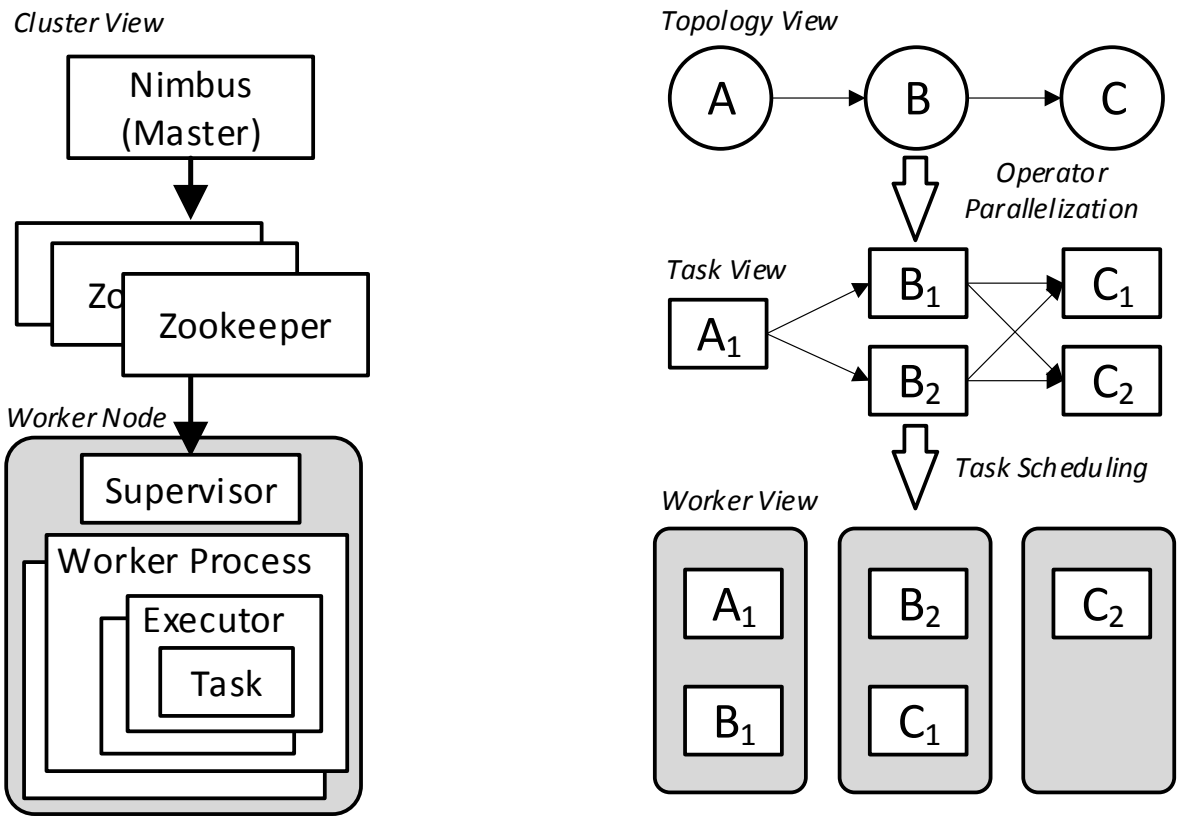


Stream Processing

- Logic level
  - Inter-connected operators
  - Data streams flow through these operators to undergo different types of computation
- Middleware level
  - Data Stream Management System (DSMS)
  - Apache Storm, Samza ...
- Infrastructure level
  - A set of distributed hosts in cloud or cluster environment
  - Organised in Master/Slave model



The structural view and logical view of a Storm Cluster

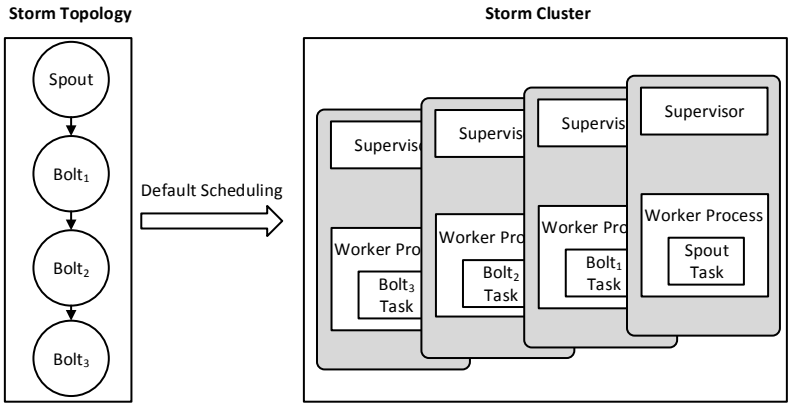


(a) Structural view

(b) Logical view

Task Scheduling in Apache Storm

- Static schedulers
  - Default scheduler
    - Assign executors as evenly as possible between all the workers
    - Round-robin procedure



  - Peng's Resource Aware Scheduler (RAS) [1]
- Dynamic scheduler
  - Throughput-oriented schedulers [2, 3]
  - Latency-oriented schedulers [4, 5, 6]
  - Communication-reduction schedulers [7, 8, 9]

Issues with Existing Schedulers

- Static assignment
  - Not able to tackle runtime changes
  - Require users' intervention to specify resource requirements
- Resource agnostic
  - Mismatch of task resource demands and node resource availability
  - Leading to over/under utilisation
- Load balancing design
  - Endeavour to distribute the workload as evenly as possible
  - Unable to perform resource consolidation when the input is small
  - Collocating multiple applications results in resource contention

Outline

Our Approach

Dynamic Resource-efficient Scheduling

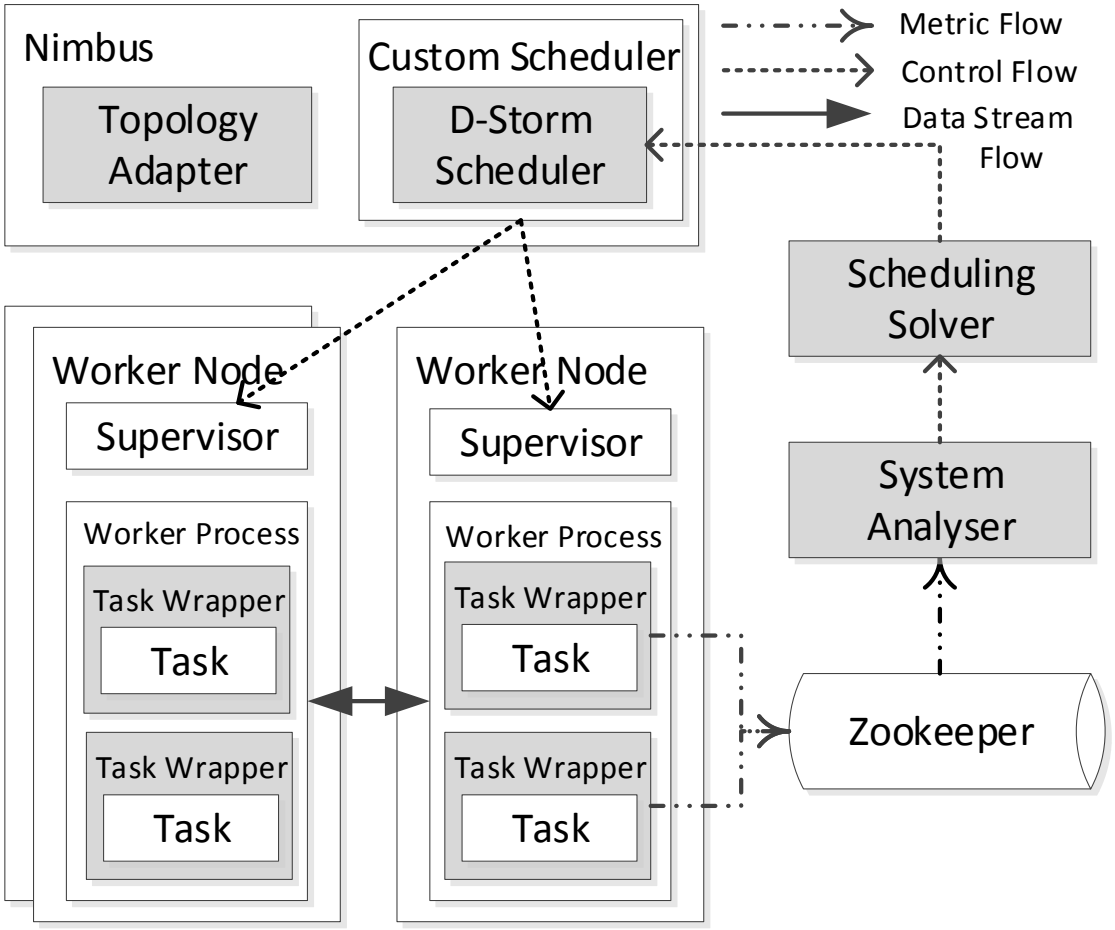
A user-transparent mechanism in DSMS to schedule streaming applications as compact as possible, matching the resource demands of streaming tasks to the capacity of distributed nodes, so that fewer resources are consumed to achieve the same performance target

Core contributions:

- Profile streaming tasks at runtime to get accurate resource demands
- Model and solve the scheduling problem as a bin-packing variant to enable resource consolidation
- Reschedule the application automatically with a MAPE loop

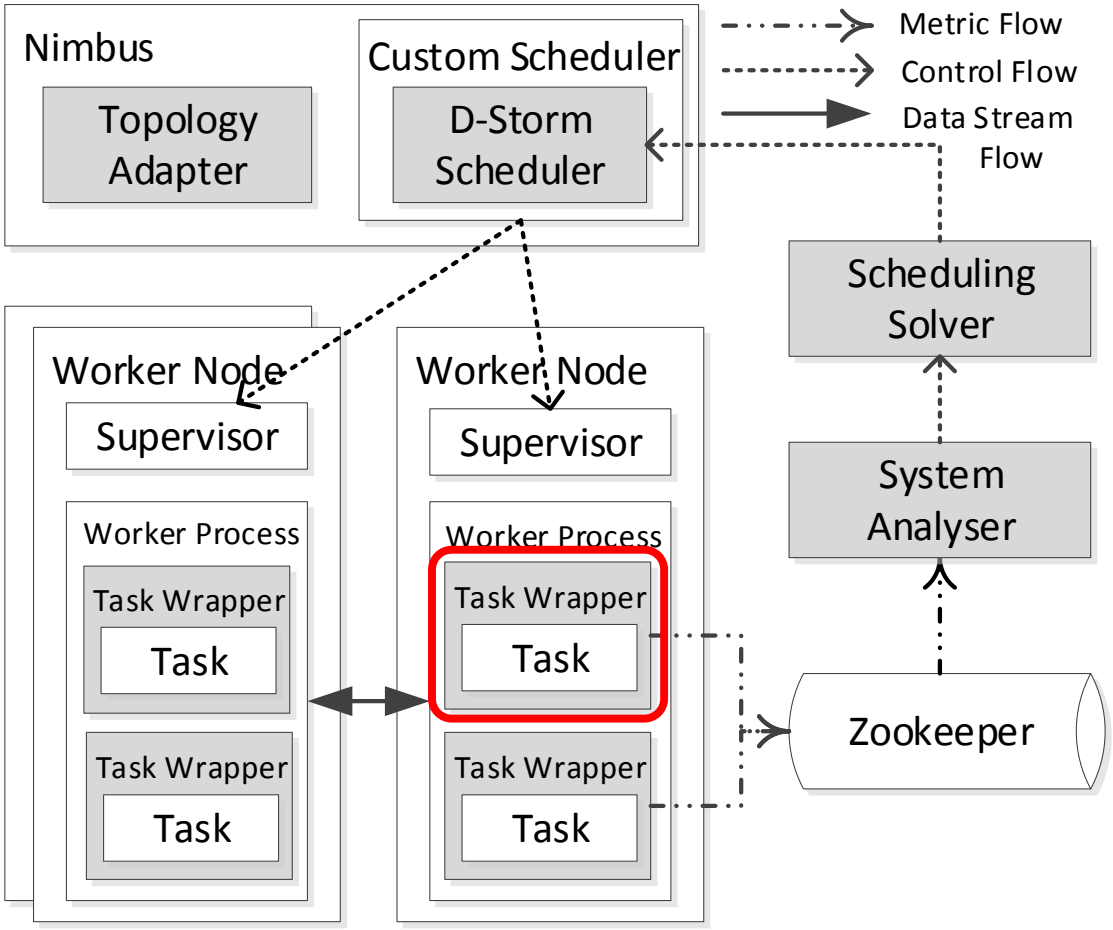
D-Storm Framework

The extended D-Storm architecture



D-Storm Framework

The extended D-Storm architecture

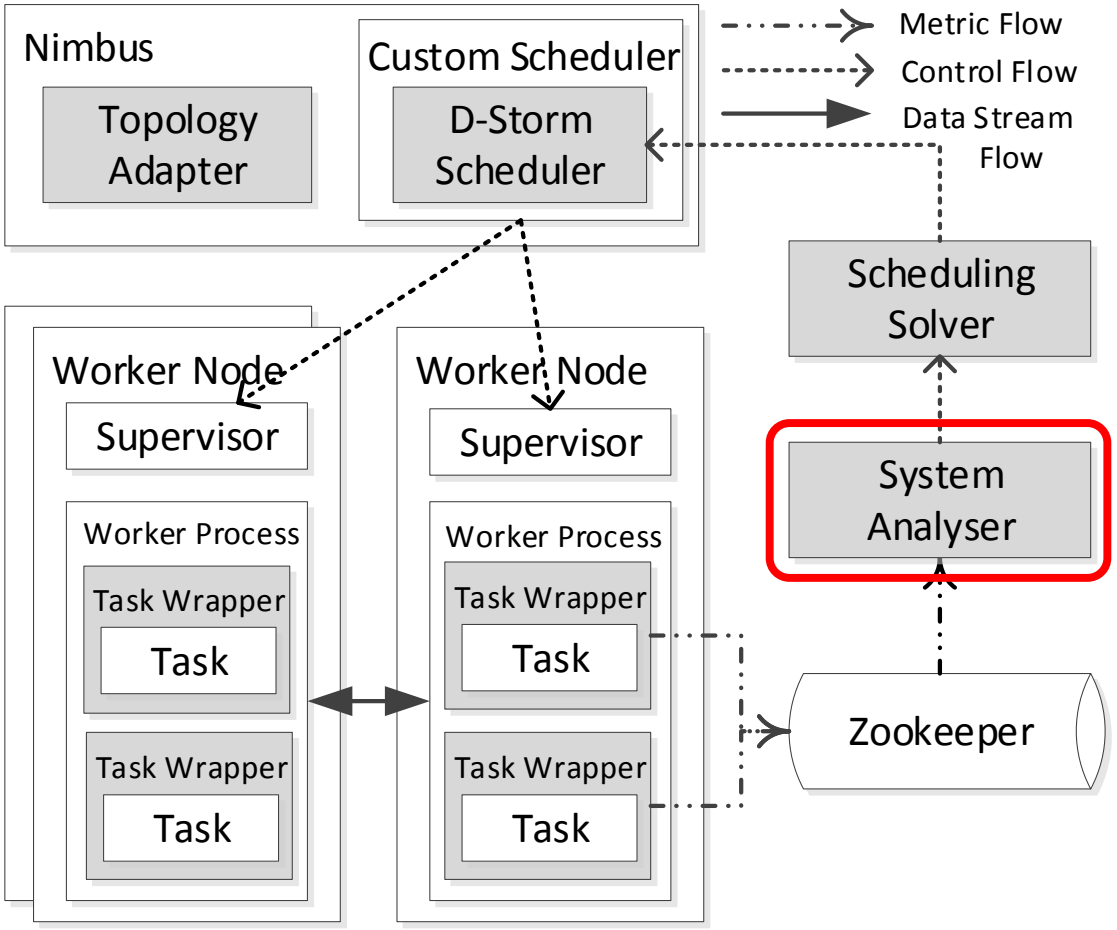


Task Wrapper:

- A middle tier between task and executor abstractions
- Obtain task CPU usages
- Log communication traffics

D-Storm Framework

The extended D-Storm architecture



Task Wrapper:

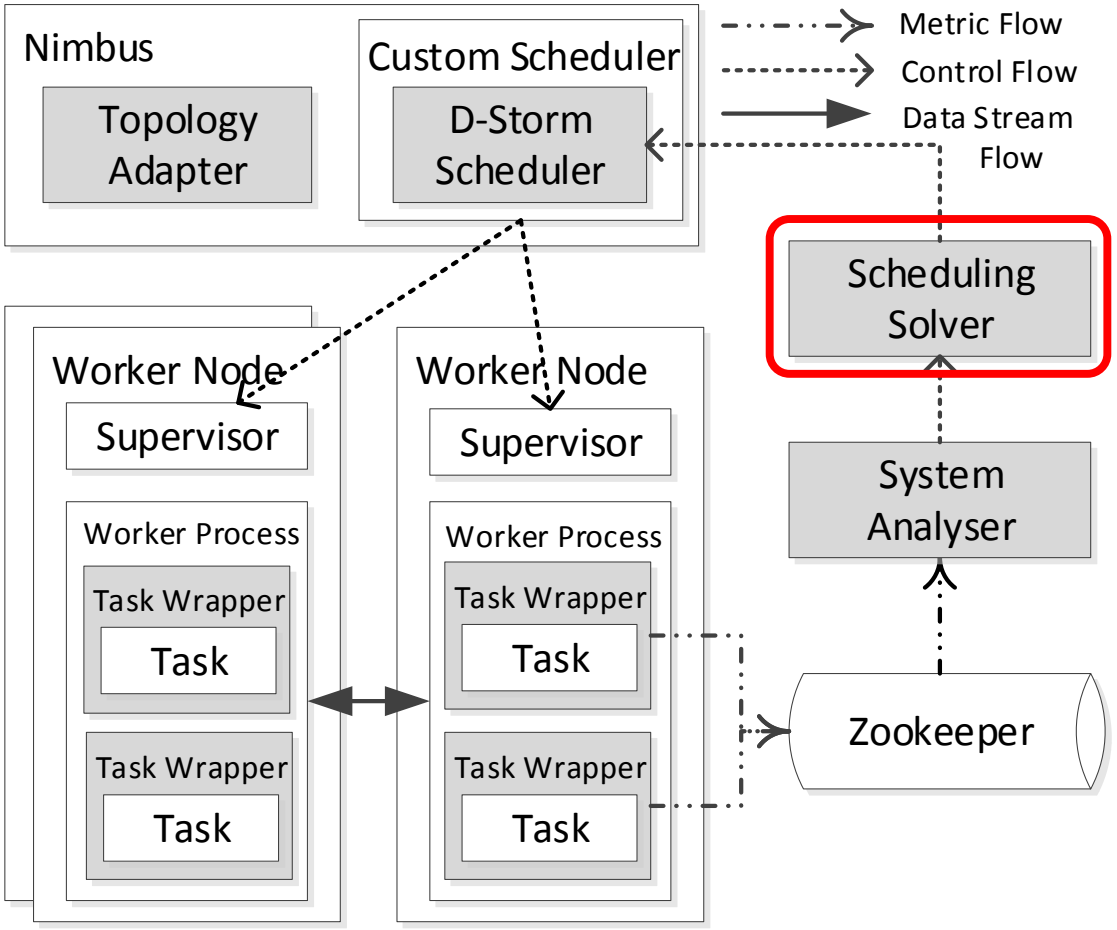
- A middle tier between task and executor abstractions
- Obtain task CPU usages
- Log communication traffics

System Analyser:

- A boundary checker on metrics
- Determine whether the current system state is normal
- Two abnormal states:
  - Unsatisfactory performance
  - Consolidation required

D-Storm Framework

The extended D-Storm architecture

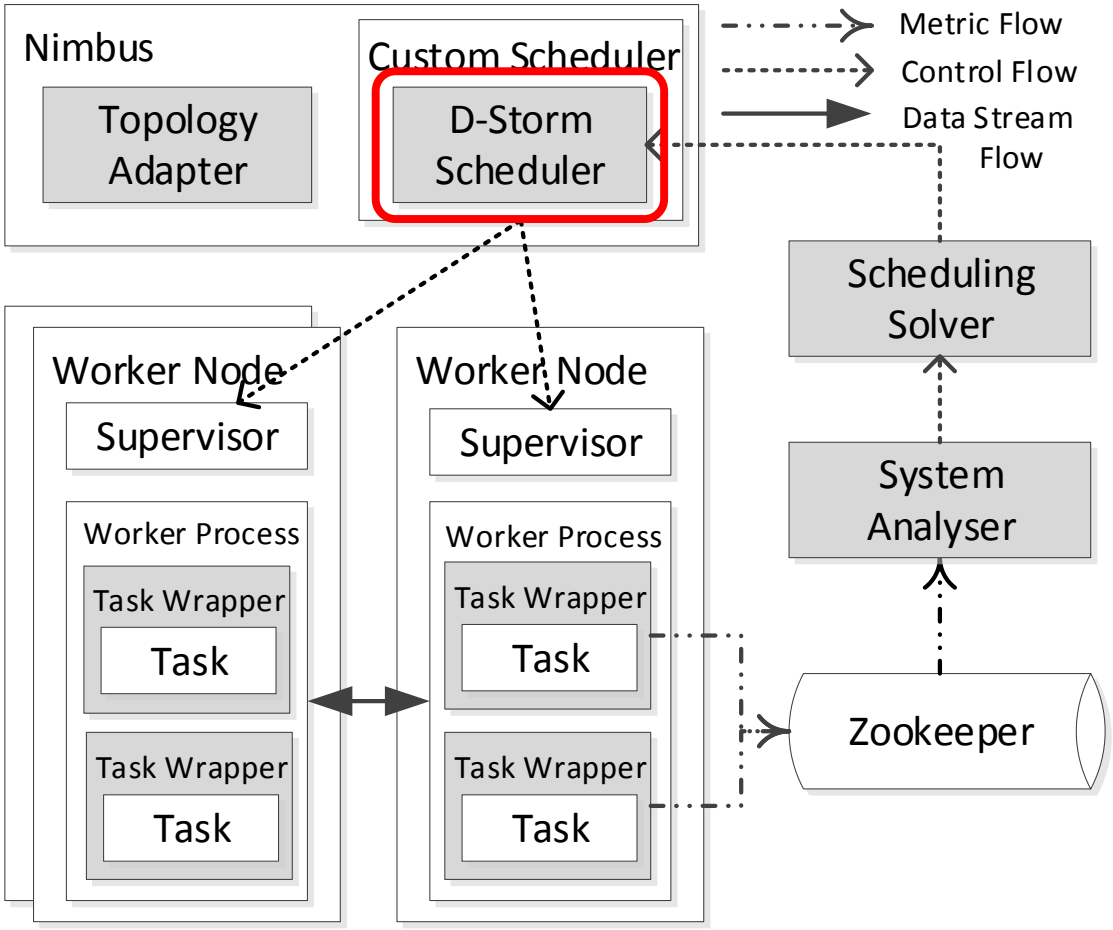


Scheduling Solver:

- Invoked by the System Analyser reporting an abnormal system state
- Perform scheduling calculation, devising a new plan

D-Storm Framework

The extended D-Storm architecture



Scheduling Solver:

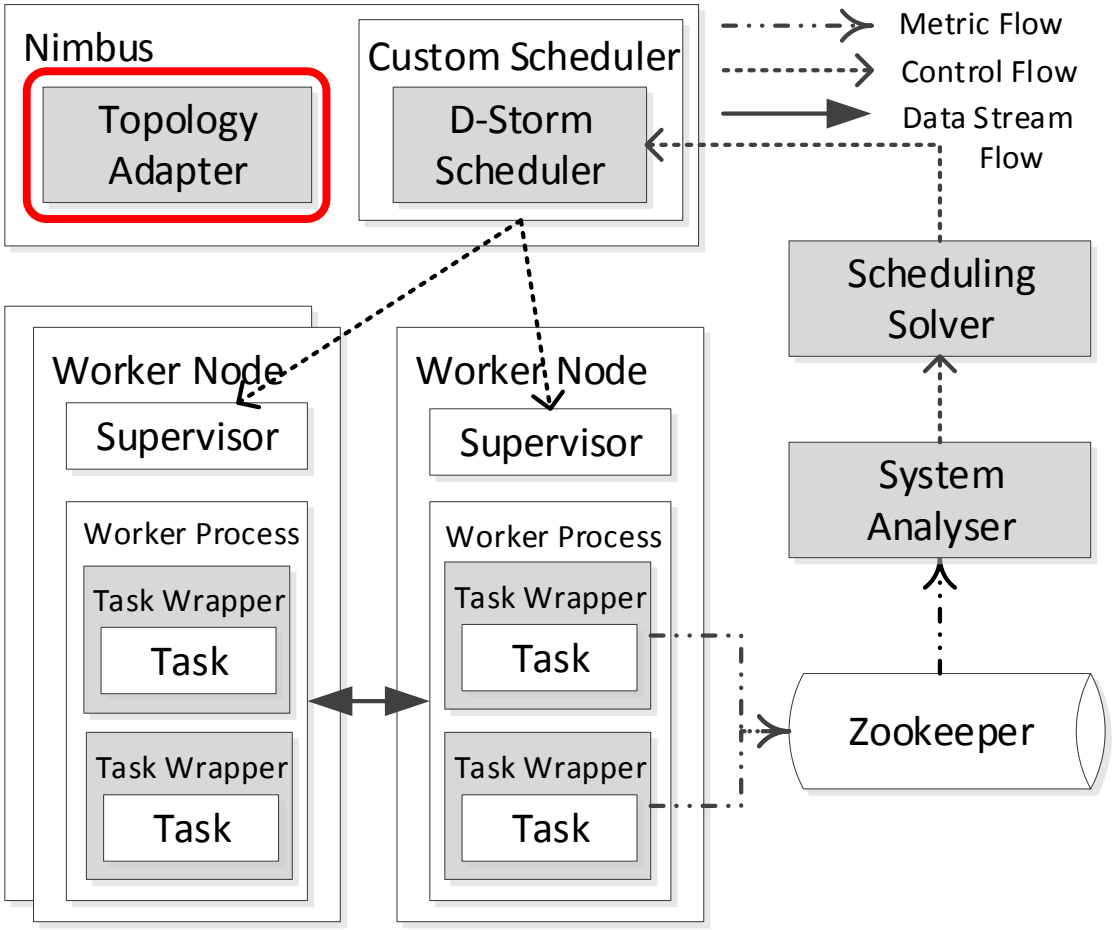
- Invoked by the System Analyser reporting an abnormal system state
- Perform scheduling calculation, devising a new plan

D-Storm Scheduler:

- Implement the *IScheduler* interface
- Put the new plan into effect

D-Storm Framework

The extended D-Storm architecture



Scheduling Solver:

- Invoked by the System Analyser reporting an abnormal system state
- Perform scheduling calculation, devising a new plan

D-Storm Scheduler:

- Implement the *IScheduler* interface
- Put the new plan into effect

Topology Adapter:

- Mask changes made for profiling

Outline

Problem Formulation

Table: Symbols used for dynamic resource-efficient scheduling

Symbol	Description
n	The number of tasks to be assigned
τ_i	Task i, $i = 1, ..., n$
m	The number of available worker nodes in the cluster
ν_i	Worker node i, $i = 1, ..., m$
$W_c^{\nu_i}$	CPU capacity of ν_i, measured in a point-based system
$W_m^{\nu_i}$	Memory capacity of ν_i, measured in Mega Bytes (MB)
$\omega_c^{\tau_i}$	Total CPU requirement of τ_i in points
$\omega_m^{\tau_i}$	Total memory requirement of τ_i in Mega Bytes (MB)
$\rho_c^{\tau_i}$	Unit CPU requirement for τ_i to process a single tuple
$\rho_m^{\tau_i}$	Unit memory requirement for τ_i to process a single tuple
ξ_{τ_i,τ_j}	The size of data stream transmitting from τ_i to τ_j
Θ_{τ_i}	The set of upstream tasks for τ_i
Φ_{τ_i}	The set of downstream tasks for τ_i
$\varkappa$	The volume of inter-node traffic within the cluster
V_{used}	The set of used worker nodes in the cluster
m_{used}	The number of used worker nodes in the cluster

$$
\begin{aligned}
\text{minimise} \quad & \varkappa(\boldsymbol{\xi}, \boldsymbol{x}) = \sum_{i,j \in \{1,..,n\}} \xi_{\tau_i,\tau_j} (1 - \sum_{k \in \{1,..,m\}} x_{i,k} * x_{j,k}) \\
\text{subject to} \quad & \sum_{k=1}^{m} x_{i,k} = 1, \qquad i = 1, ..., n, \\
& \sum_{i=1}^{n} \omega_c^{\tau_i} x_{i,k} \leq W_c^{\nu_k} \quad k = 1, ..., m, \qquad (1) \\
& \sum_{i=1}^{n} \omega_m^{\tau_i} x_{i,k} \leq W_m^{\nu_k} \quad k = 1, ..., m,
\end{aligned}
$$

where $\boldsymbol{x}$ is the control variable that stores the task placement in a binary form: $x_{i,k} = 1$ if and only if task τ_i is assigned to machine ν_k.

Scheduling Algorithm

Algorithm 1: The multidimensional FFD heuristic scheduling algorithm

Input: A task set $\vec{\tau} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ to be assigned

Output: A machine set $\vec{\nu} = \{\nu_1, \nu_2, \ldots, \nu_{m_{\text{used}}}\}$ with each machine hosting a disjoint subset of $\vec{\tau}$, where m_{used} is the number of used machines

```
Sort available nodes in descending order by their resource availability
  as defined in Eq. (4)
m_used ← 0
while there are tasks remaining in τ⃗ to be placed do
    Start a new machine ν_m from the sorted list;
    if there are no avaiable nodes then
        return Failure
    Increase m_used by 1
    while there are tasks that fit into machine ν_m do
        foreach τ ∈ τ⃗ do
            Calculate ϱ(τ_i, ν_m) according to Eq. (5)
        Sort all viable tasks based on their priority
        Place the task with the highest ϱ(τ_i, ν_m) into machine ν_m
        Remove the task from τ⃗
        Update the remaining capacity of machine ν_m
return ν⃗
```

Algorithm Design:

- Based on greedy heuristic
- Large machines are utilised first
- Task priority updated dynamically
- Seek to increase the sum of internal communications at the current node

Scheduling Algorithm

Algorithm 2: The multidimensional FFD heuristic scheduling algorithm

Input: A task set $\vec{\tau} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ to be assigned

Output: A machine set $\vec{\nu} = \{\nu_1, \nu_2, \ldots, \nu_{m_{\text{used}}}\}$ with each machine hosting a disjoint subset of $\vec{\tau}$, where m_{used} is the number of used machines

```
1  Sort available nodes in descending order by their resource availability
     as defined in Eq. (4)
2  m_used ← 0
3  while there are tasks remaining in τ⃗ to be placed do
4      Start a new machine ν_m from the sorted list;
5      if there are no avaiable nodes then
6          return Failure
7      Increase m_used by 1
8      while there are tasks that fit into machine ν_m do
9          foreach τ ∈ τ⃗ do
10             Calculate ϱ(τ_i, ν_m) according to Eq. (5)
11         Sort all viable tasks based on their priority
12         Place the task with the highest ϱ(τ_i, ν_m) into machine ν_m
13         Remove the task from τ⃗
14         Update the remaining capacity of machine ν_m
15 return ν⃗
```

Algorithm Design:

- Based on greedy heuristic
- Large machines are utilised first
- Task priority updated dynamically
- Seek to increase the sum of internal communications at the current node

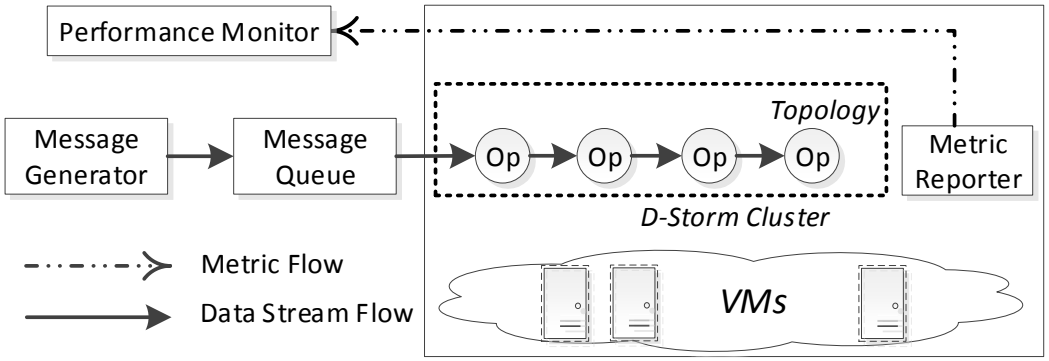
Complexity Analysis:

- Line 1 requires at most quasilinear time $O(mlog(m))$
- Line 8 to Line 14 will be repeated for at most n times
- Line 10 consumes linear time of n
- The worst case complexity: $O(mlog(m) + n^2 log(n))$

Outline

Experiment Setup

- Nectar IaaS Cloud
  - 16 worker nodes: 2 VCPUs, 6GB RAM and 30GB disk
  - 1 Nimbus, 1 Zookeeper, 1 Kestrel node
- Extended on Apache Storm v1.0.2, with Oracle JDK 8, update 121
- Two test applications
  - Synthetic linear application
  - Twitter Sentiment Analysis topology
- Profiling environment



Experiment Setup

- Synthetic linear application
  - 3 synthetic bolts concatenated in serial
  - Produce different patterns of resource consumption, such as CPU bound and I/O bound computations.
- Twitter Sentiment Analysis topology
  - 11 operators constituting a tree-like topology
  - Sentimental score calculated using AFFINN, a list of words associated with pre-defined sentiment values.

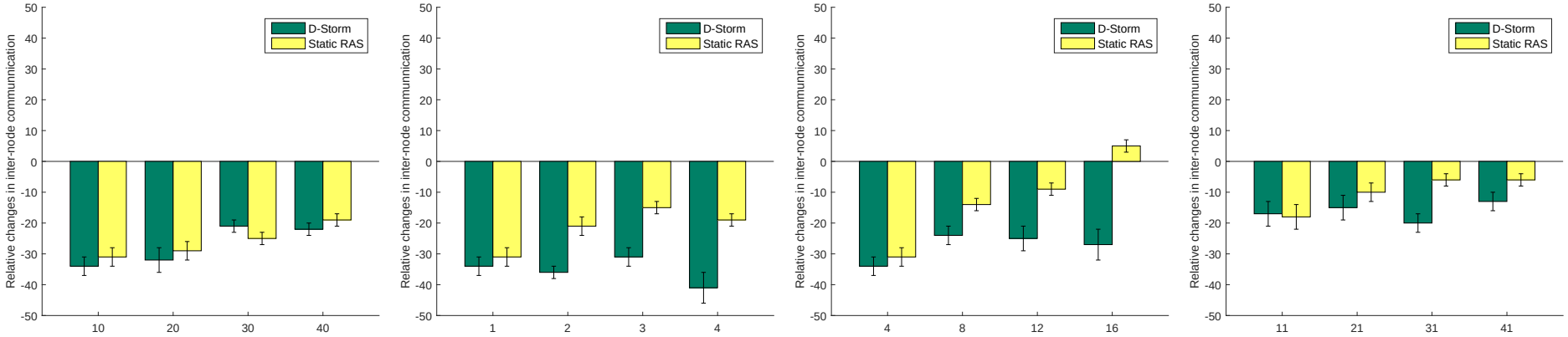
Table: Evaluated configurations and their values

Configuration	**Value**
C_s (synthetic topology)	**10**, 20, 30, 40
S_s (synthetic topology)	**1**, 2, 3, 4
T_s (synthetic topology)	**4**, 8, 12, 16
Number of tasks (realistic application)	**11**, 21, 31, 41

Applicability Evaluation

Research Question

Whether D-Storm is applicable to different types of streaming applications and capable of reducing the total amount of inter-node communication?



(a) Varying C_s (Sythetic App)

(b) Varying S_s (Sythetic App)

(c) Varying T_s (Sythetic App)

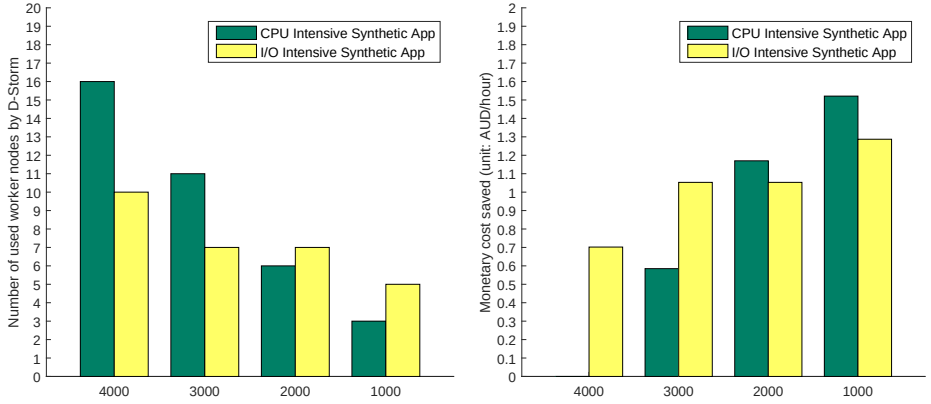
(d) Varying number of tasks (Real App)

Figure: The relative change of the inter-node communication, with the baseline produced by the Default Storm scheduler

Cost-Efficiency Evaluation

Research Question

How is D-Storm performing when the input load decreases and the cluster is under-utilized?



(a) The number of used nodes

(b) The saved monetary cost

Figure: Cost efficiency analysis of D-Storm scheduler as the input load decreases. Fig. 2b is calculated based on the price of m1.medium instances in the AWS Sydney region

Scheduling Overhead Evaluation

Research Question

How long does it take for D-Storm to schedule relatively large streaming applications?

Table: The time consumed in creating schedules by different strategies (unit: milliseconds)

Schedulers \ Test Cases	Parallelism Intensive Synthetic App			
	T_s=4	T_s=8	T_s=12	T_s=16
D-Storm	16.4	16.2	16.8	17.2
Static Scheduler	5.2	5.7	5.5	5.9
Default Scheduler	0.72	0.77	0.75	0.79
	Twitter Sentiment Analysis			
	N_t=11	N_t=21	N_t=31	N_t=41
D-Storm	13.4	13.6	13.8	13.9
Static Scheduler	3.41	3.61	3.53	3.91
Default Scheduler	0.61	0.62	0.65	0.62

Outline

Related Work

In comparison with existing schedulers, we conclude that our work is dynamic resource-aware, communication-aware, self-adaptive, user-transparent and cost-efficient.

Table: Related work comparison

Aspects	**Related Works**							**Our Work**
	[7]	[1]	[8]	[9]	[4]	[10]	[2]	
Dynamic	Y	N	Y	Y	Y	N	Y	Y
Resource-aware	N	Y	N	N	Y	Y	N	Y
Communication -aware	Y	N	Y	Y	N	Y	Y	Y
Self-adaptive	Y	N	Y	Y	N	N	Y	Y
User-transparent	N	N	Y	Y	N	N	N	Y
Cost-efficient	N	Y	N	N	Y	N	N	Y

Outline

Conclusions

- We proposed a resource-efficient algorithm for scheduling streaming application with bin-packing formulation.
- We implemented D-Storm, a prototype scheduler for algorithm validation.
- The compact scheduling strategy leads to the reduction of resource usages as well as the minimisation of inter-node communication
- We adjust the scheduling plan to the runtime changes while remaining sheer transparent to the upper-level application logic.

- Outlook
  - Use meta-heuristics to find a better solution
  - Take the network characteristics into account during the scheduling process, placing intensive task communications on links with higher bandwidth.

Any questions?

References I

- B. Peng, M. Hosseini, Z. Hong, R. Farivar, and R. Campbell, "R-storm: Resource-aware scheduling in storm," in *Proceedings of the 16th Annual Conference on Middleware*, ser. Middleware '15. ACM Press, 2015, pp. 149–161.
- C. Li, J. Zhang, and Y. Luo, "Real-time scheduling based on optimized topology and communication traffic in distributed real-time computation platform of storm," *Journal of Network and Computer Applications*, vol. 87, no. 3, pp. 100–115, jun 2017.
- T. Buddhika, R. Stern, K. Lindburg, K. Ericson, and S. Pallickara, "Online scheduling and interference alleviation for low-latency, high-throughput processing of data streams," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3553–3569, Dec 2017.

References II

- D. Sun, G. Zhang, S. Yang, W. Zheng, S. U. Khan, and K. Li, "Re-stream: Real-time and energy-efficient resource scheduling in big data stream computing environments," *Information Sciences*, vol. 319, pp. 92–112, oct 2015.
- D. Sun and R. Huang, "A Stable Online Scheduling Strategy for Real-Time Stream Computing over Fluctuating Big Data Streams," *IEEE Access*, vol. 4, pp. 8593–8607, 2016.
- A. Chatzistergiou and S. D. Viglas, "Fast heuristics for near-optimal task allocation in data stream processing over clusters," in *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management*, ser. CIKM '14. ACM Press, 2014, pp. 1579–1588.

References III

- L. Aniello, R. Baldoni, and L. Querzoni, "Adaptive online scheduling in storm," in *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '13. ACM Press, 2013, pp. 207–218.
- L. Fischer and A. Bernstein, "Workload scheduling in distributed stream processors using graph partitioning," in *Proceedings of the 2015 IEEE International Conference on Big Data*. IEEE, 2015, pp. 124–133.
- J. Xu, Z. Chen, J. Tang, and S. Su, "T-storm: Traffic-aware online scheduling in storm," in *Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems*, ser. ICDCS '14. IEEE, 2014, pp. 535–544.

References IV

T. Li, J. Tang, and J. Xu, "Performance Modeling and Predictive Scheduling for Distributed Stream Data Processing," *IEEE Transactions on Big Data*, vol. 7790, no. 99, pp. 1–12, 2016.