Benchmarking Distributed Stream Processing Platforms for IoT Applications

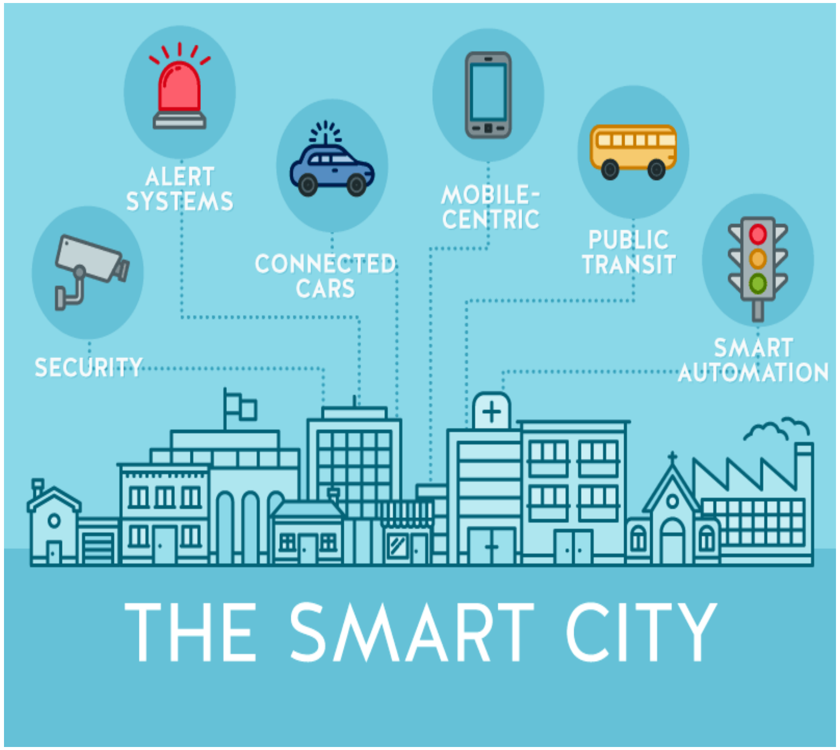
Anshu Shukla and Yogesh Simmhan
Department of Computational & Data Sciences
Indian Institute of Science

Eighth TPC Technology Conference on Performance Evaluation & Benchmarking (TPCTC 2016)

5-Sep-16

Motivation: Internet of Things (IoT)

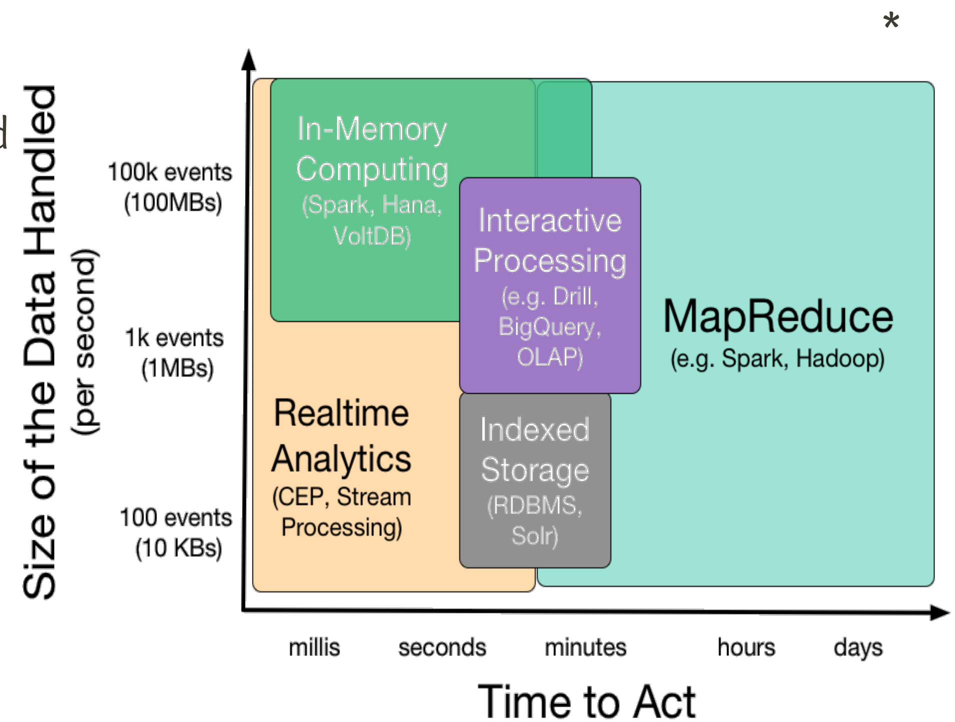
- **Billions of sensors** monitoring physical infrastructure, human beings and virtual entities in realtime, distributed across network
- *Data streams* drive realtime processing, analytics & decisions making
- *Streaming applications* **control feedback** on physical structure, notifications to users



http://1.bp.blogspot.com/-N5JfgIr1DaE/Vbpn0ilwYXI/AAAAAAAAjws/D1_DeUmxf8c/s1600/Smart-Cities-Features.png

Data Platforms for IoT

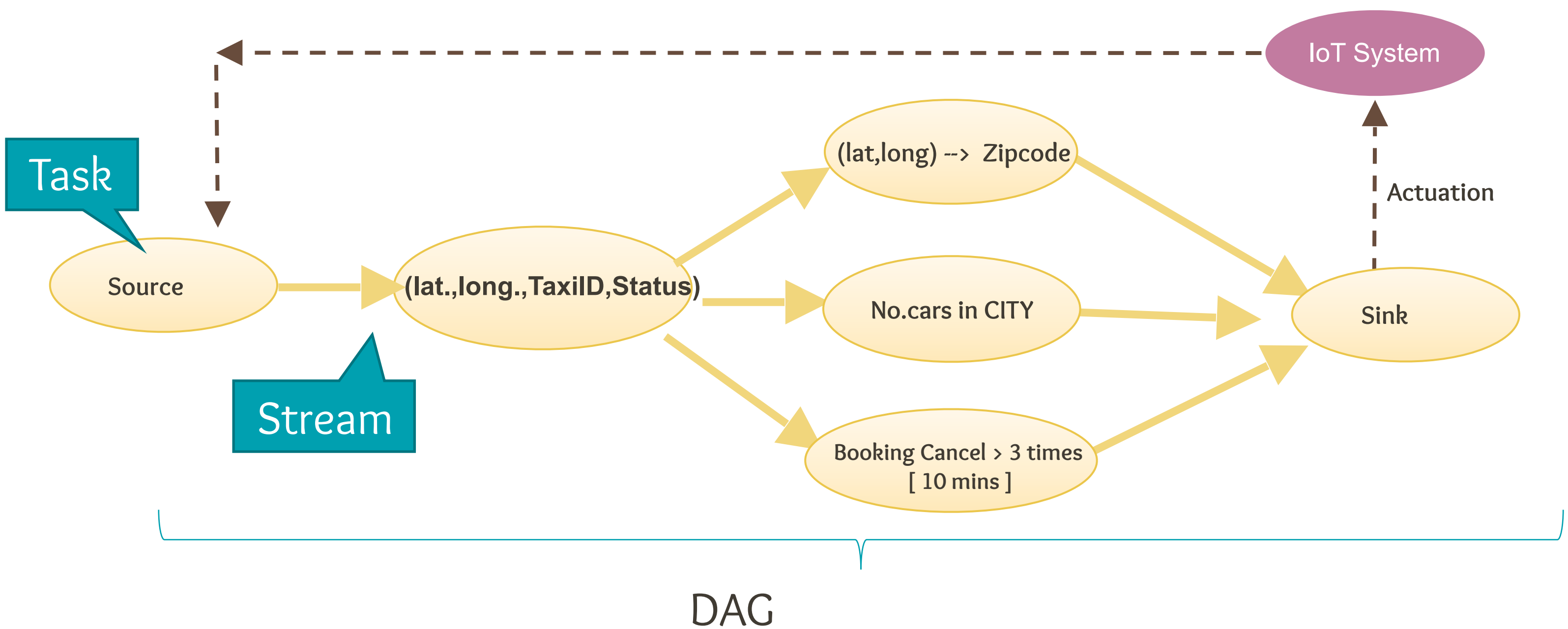
- MapReduce/Hadoop
  - Large volume, high latency on distributed hosts
  - *Training models over sensor data archives*
- Complex Event Processing
  - Declarative pattern matching over event tuples
  - *Detect critical events, changing trends*
- **Distributed Stream Processing Systems (DSPS)**
  - **Fast data rates, Weaker message order**
  - **Distributed execution, fault-tolerant**
  - ***Flexible composition & responsive coordination of Observe, Orient, Decide, Act (OODA) loop***
- Realtime Processing
  - Mission critical apps, Strong latency guarantees
  - *Embedded medical devices, power networks*



*Big Data Analytics Platforms for Real-time Applications in IoT, Yogesh Simmhan & Srinath Perera, Big Data Analytics: Methods and Applications, Eds. Saumyadipta Pyne, B.L.S. Prakasa Rao, S.B. Rao, 2016 (to appear)

Distributed Stream Processing System [DSPS]

- Scalable, Distributed and fault-tolerant computations over data streams
- Composition of streaming applications
  - Directed Acyclic Graph (DAG) of tasks and message streams
  - Unbounded sequence of opaque messages as streams
  - User-define logic blocks as tasks



Need for a DSPS Benchmark for IoT

- Uniformly compare performance of DSPS for IoT applications
  - Helps select DSPS platform for IoT domain
- Understanding gaps in capabilities of DSPS
  - Composition of meaningful IoT applications

- Benchmark design goals
  - Categories & platform-independent implementations of
    - Common IoT tasks
    - Classes of non-trivial IoT applications
  - Different real-world IoT data streams

Gaps in related work

- *IoTAbench:* Emphasis on scalable synthetic data generation and not on application logic
- *CityBench:* RDF stream processing systems

- *Stream Bench:* 7 micro-benchmarks on 4 different synthetic workload suites, does not consider aspects unique to IoT applications and streams
- *Spark Bench*: Specific to Apache Spark, goal is to identify spark tuning parameters, not platform-agnostic
- *Yahoo Cloud Serving Benchmark (YCSB):* Compares different key-value stores on the Cloud
- *Bigbench:* Simulates online retail enterprise data with queries covering velocity, volume and variety of data

Contributions

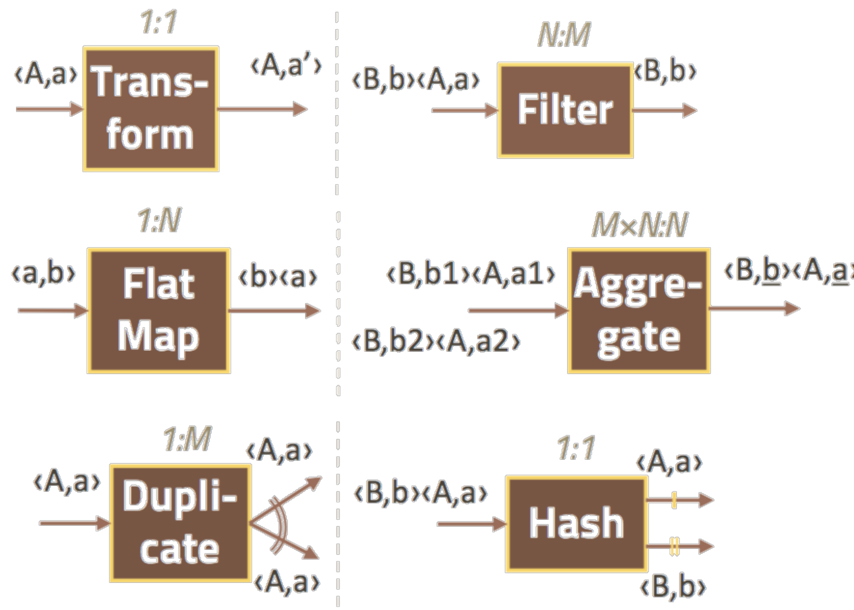
1. **Classification** of essential tasks & IoT applications, and characteristics of their data sources
2. Propose **IoT Benchmark for DSPS**
   a. Micro-benchmark tasks implemented from the above classification
   b. Reference IoT applications implemented for
      - *Pre-processing & statistical summarisation* (STATS)
      - *Predictive Analytics* (PRED)
3. Practical **evaluation** on Apache Storm DSPS

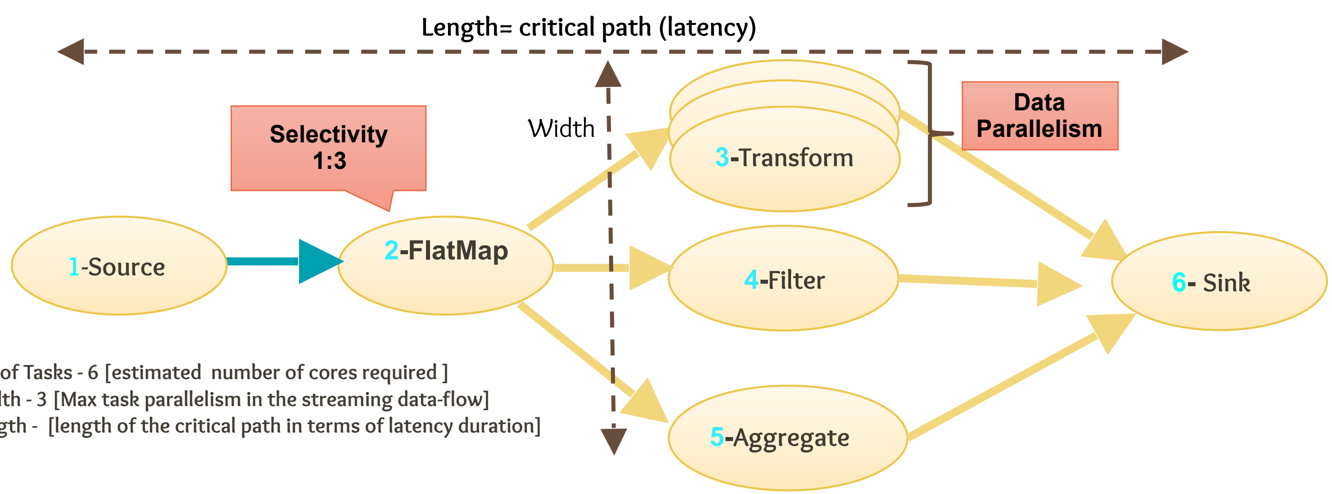
Classification & Metrics

IoT characteristics, DSPS capabilities, Performance Metrics

Dataflow Semantics

- Dataflow patterns determine consumption & routing of messages
- Selectivity ratio: Number of output messages emitted by a task on consuming a logical unit input message(s)
  - *Decide input rate for downstream tasks*
- Number of tasks, Edge Degree, Length/ Width of Dataflow affect performance, scalability

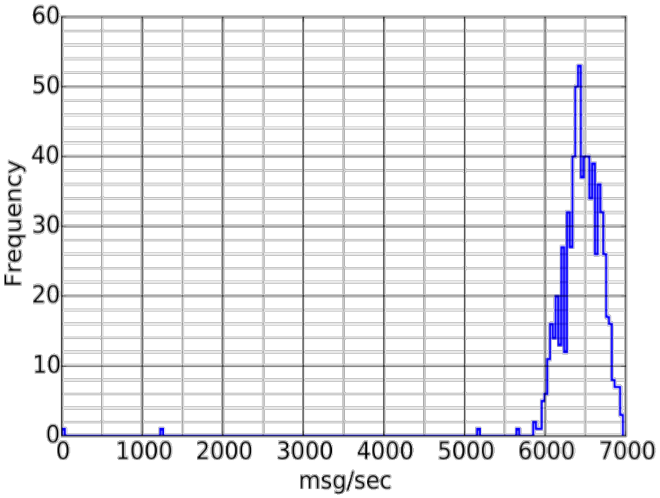




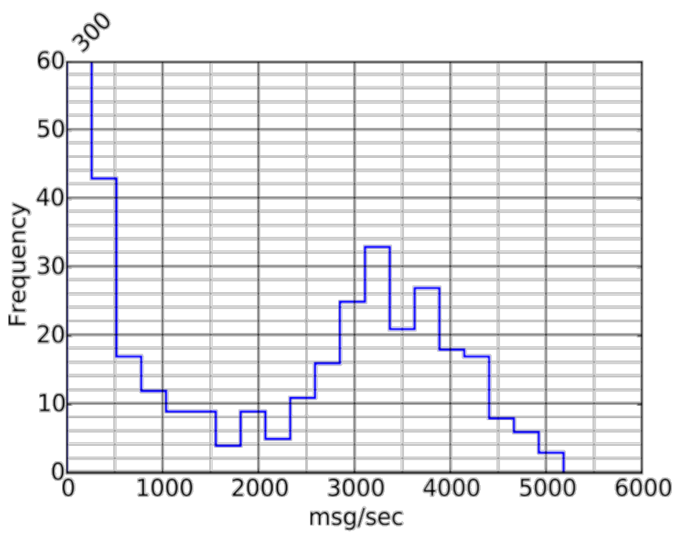
No. of Tasks - 6 [estimated number of cores required]
Width - 3 [Max task parallelism in the streaming data-flow]
Length - [length of the critical path in terms of latency duration]

IoT Data stream Characteristics

- **Input Throughput**
  - Rate at which messages enter the source tasks of the application dataflow
  - *Determines the parallelism of tasks in DSPS, ability to scale and meet latency goals*
- **Throughput Distribution**
  - Variation of input throughput over time.
  - E.g. High demand for cabs in morning/evenings
  - *Dynamic load balancing, elasticity*
    - **CITY dataset [1]:** Environmental sensors at 7 cities across 3 continents;
    - **TAXI dataset [2]:** Smart transportation messages from 2M trips of New York city taxis
- Impact of Message size, temporal ordering can also be considered



(a) CITY @1000× msg/sec



(b) TAXI @1000× msg/sec

[1] http://map.datacanvas.org/#!/data
[2] http://www.debs2015.org/call-grand-challenge.html

IoT Streaming Application Classes

- **Extract-Transform-Load (ETL) and Archival**
  - Pre-processing like data format transformations, normalizing observation units, interpolate missing data items
  - Archive a copy of the data offline
- **Summarization and Visualization**
  - Statistical aggregation and analytics to "*orient*" the decision
  - Generate visualizations to present it to end-users and decision makers
- **Prediction and Pattern Detection**
  - Prediction to determine the future state and "*decide*" if any reaction is required
  - Identify the patterns of interest
- **Classification and notification**
  - Mapping decisions to specific "*actions*"
  - Notifying the entities in IoT system

Categories of IoT Tasks

- Parse - parsing encoded messages from the stream sources *[XML, CBOR, JSON, SenML]*
- Filter - filtering on specific attribute values *[bloom, range filter]*
- Statistical Analytics - finding outliers, second order moments, distinct count
- Predictive Analytics - predictions using past and current messages, online model training *[WEKA library, R Packages]*
- Pattern Detection - identify patterns across events using CEP engines
- Visual Analytics - periodic charts for streaming application as part of the dataflow *[D3.js, JFreeChart]*
- IO Operations - accessing external storage or messaging services to access data *[file/Cloud storage, database ops, pub-sub]*

Evaluation Metrics

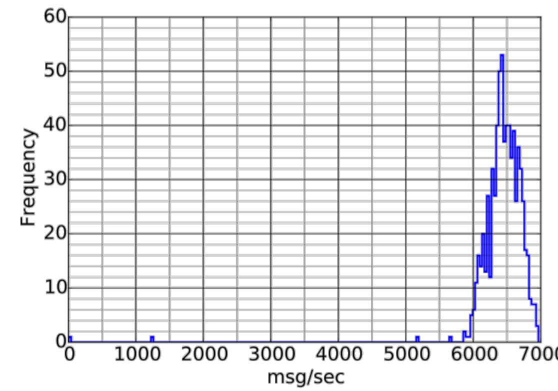
- Metrics to meet performance & scalability needs for IoT applications from DSPS
- Latency
  - Time between message consumed at source task(s) and their *causally dependent* messages generated at sink task(s),***lesser is better***
  - *Decides task parallelism for required input rate*
- Peak Rate
  - Aggregated peak output message rate generated from sink task(s),***higher is better***
  - *Estimated resource requirement for required input rate*
- Jitter
  - Variation in *observed vs. expected output throughput* based on input rate and task selectivity,***close to zero is better***
  - *Stability of task for given rate with sustainable queuing of messages*
- CPU and Memory Utilisation
  - Uniform distribution of load on VMs, hotspots causing bottlenecks
  - *Identify which VMs are the potential bottlenecks/under-used, scale-out/in*

Benchmarks

Datasets used

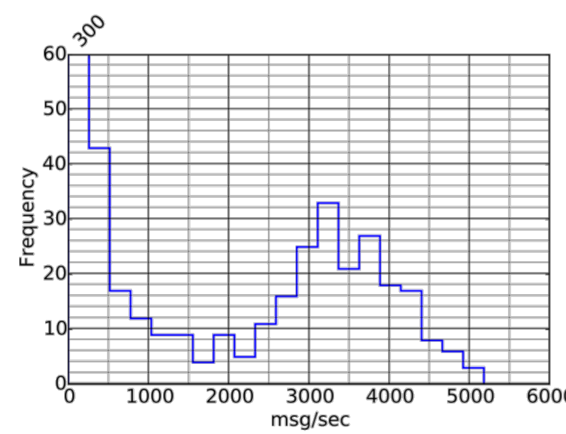
Dataset	Attributes	Format	Size(*bytes*)	Peak Rate(*msg/sec*)	Distribution
CITY [1]	9	CSV	100	7,000	Normal
TAXI [11]	17	CSV	191	4,000	Bimodal

- CITY: *Urban environmental sensing* data from 90 sensors at 7 cities, sampled per minute
  - Scaled to 1000x to simulate 90,000 sensors
  - timestamp, source, long, lat, temperature, humidity, light, dust, airquality



(a) CITY @1000× msg/sec

- TAXI: *Smart transportation* messages from 2M trips from 20K taxis in NYC, one per trip
  - Bi-modal rate distribution *[morning & evening commutes];* 1000x scaling for temporal freq.
  - Medallion, Hack_license, Pickup_datetime, Dropoff_datetime, Trip_time, Trip_distance, Pickup_long, Pickup_lat, Dropoff_long, Dropoff_lat, Payment_type, Fare_amount, Surcharge, Mta_tax, Tip_amount, Tolls_amount, Total_amount



(b) TAXI @1000× msg/sec

[1] http://map.datacanvas.org/#!/data
[2] http://www.debs2015.org/call-grand-challenge.html

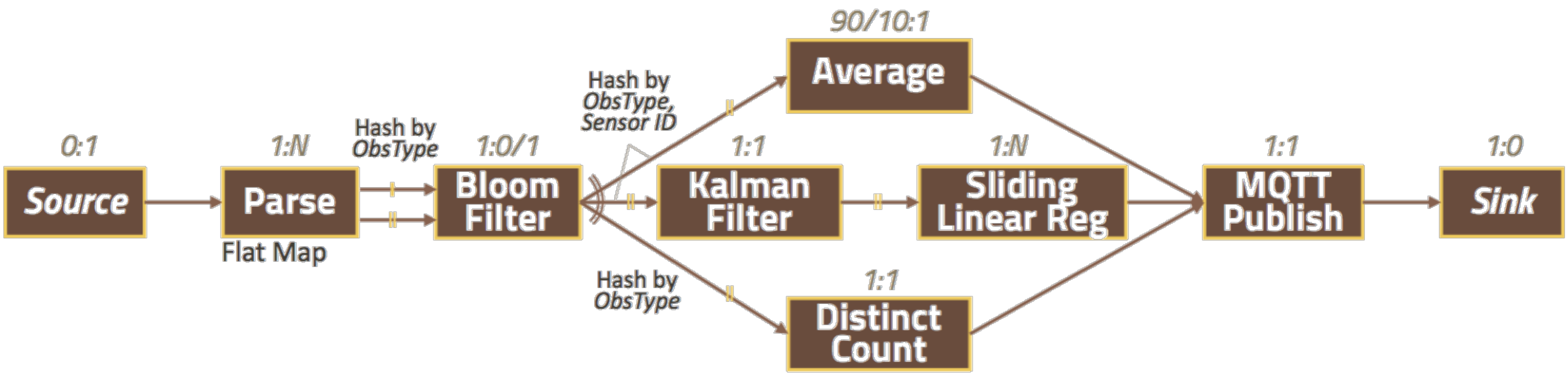
Micro-benchmark Tasks

Task Name	Code	Category	Pattern	σ Ratio	State
XML Parsing	XML	Parse	Transform	1:1	No
Bloom Filter	BLF	Filter	Filter	1:0/1	No
Average	AVG	Statistical	Aggregate	N:1	Yes
Distinct Appox. Count	DAC	Statistical	Transform	1:1	Yes
Kalman Filter	KAL	Statistical	Transform	1:1	Yes
Second Order Moment	SOM	Statistical	Transform	1:1	Yes
Decision Tree Classify	DTC	Predictive	Transform	1:1	No
Multi-variate Linear Reg.	MLR	Predictive	Transform	1:1	No
Sliding Linear Regression	SLR	Predictive	Flat Map	N:M	Yes
Azure Blob D/L	ABD	IO	Source/Transform	1:1	No
Azure Blob U/L	ABU	IO	Sink	1:1	No
Azure Table Query	ATQ	IO	Source/Transform	1:1	No
MQTT Publish	MQP	IO	Sink	1:1	No

Benchmarks available at http://github.com/dream-lab/bm-iot

Application Benchmarks [STATS]

- Streaming statistical analysis and filtering of data
  - Data filtering of outliers on individual observation types using a Bloom filter
  - Statistical analytics on observations from individual sensor/taxi IDs
  - Approximate count of distinct readings for every observation type
  - Publishing the results using MQTT task
  - Different types of patterns, selectivities, task parallelisms
    - » Application uses Hash,filter,Aggregate as dataflow patterns

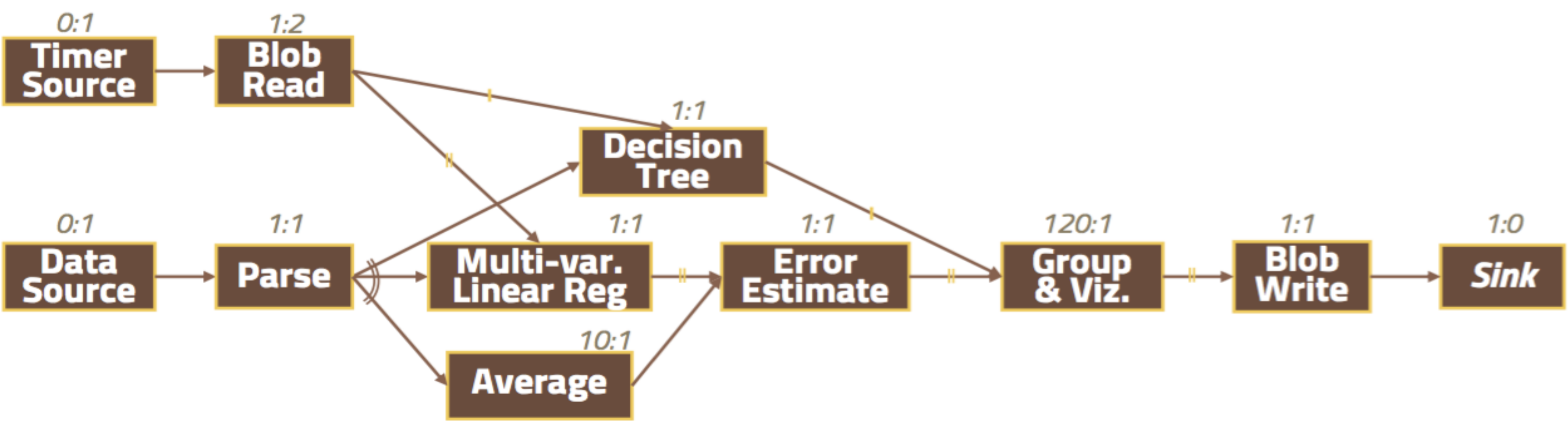


(a) Pre-processing & statistical summarization dataflow (STATS)

Benchmarks available at http://github.com/dream-lab/bm-iot

Application Benchmarks [PRED]

- Streaming predictive analysis of data
  - Timer Source and blob read - for reading updated model files using timestamp
  - Decision tree uses trained model to classify messages into classes [good,average or poor air quality]
  - Multi-Variate Linear regression & Average used for finding the error estimate
  - Group and Viz. - Batch and plot the summarised results for different observations
  - Blob write - Result files are written to Cloud storage for hosting on a portal
    - » Application uses duplicate,join,aggregate as dataflow patterns
    - » Multiple source of Input streams



(b) Predictive Analytics dataflow (PRED)

Benchmarks available at http://github.com/dream-lab/bm-iot

Experiments

Apache Storm 1.0.0

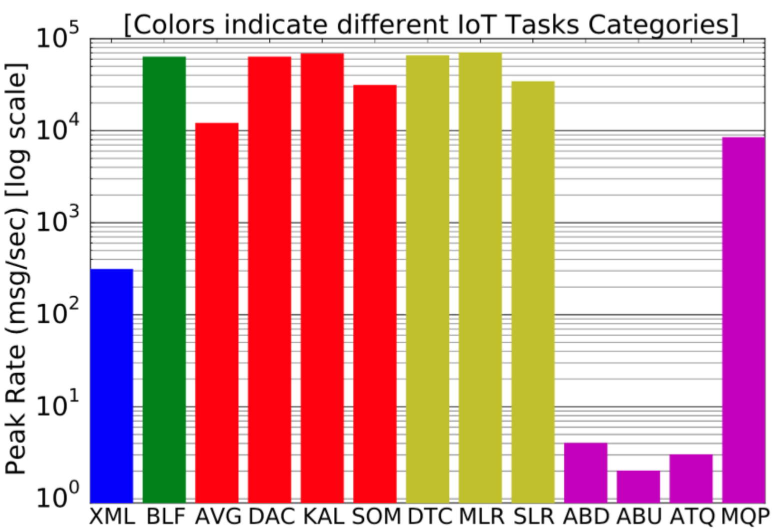
Experimental setup

- Apache Storm 1.0.0 DSPS
  - Executed on Microsoft Azure Cloud VMs
  - OpenJDK 1.7 and CentOS
- Micro-Benchmarks
  - Benchmark task runs on one exclusive D1 VM, 1 thread
    - 1-core Intel Xeon E5@2.2 GHz, 3.5 GiB RAM, 50 GiB SSD
    - *Used to finding peak rate*
  - Master services, source & sink tasks run on a D8 VM
    - 8-core Intel Xeon E5@2.2 GHz, 28 GiB RAM, 400 GiB SSD
    - *Ensures spout is not bottleneck at peak rate*
- Application Benchmarks [STATS & PRED]
  - D8 VMs as per input rate for all tasks of the dataflow
  - Experiments run for ~ 10 mins
  - 1000× scaling ➲ 7 days of data for CITY and TAXI

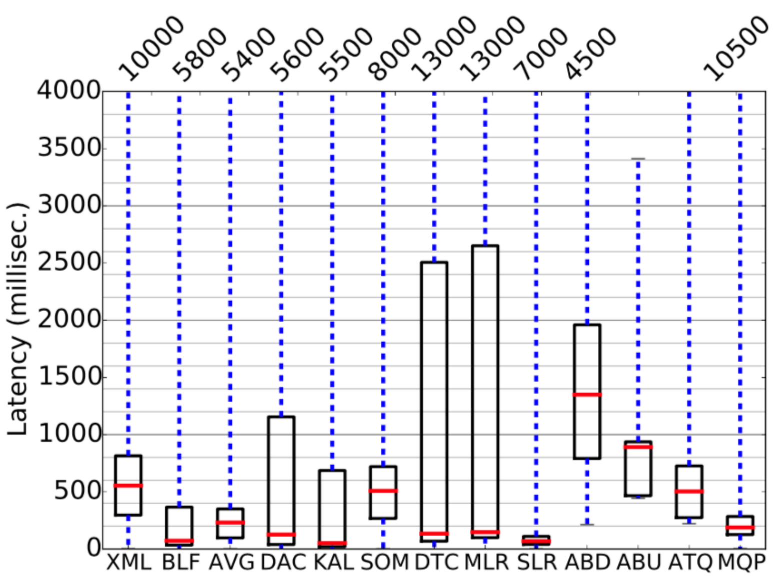
$$\textbf{Application runtime} = \frac{7 \text{ days} * 24 \text{ hours} * 60 \text{ mins}}{1000 * scaling} = \textbf{10.08 mins}$$

Results: Micro-benchmark

- Peak rate is 3,000 msg/sec or higher *except XML parsing & Azure ops being CPU (310 msg/sec) and SLA (5 msg/sec) bound*
- Wide variability in latency box plots
  - Non-uniform task execution times, slow executions causes input tuples to queue up
  - Higher input rates are more sensitive to even small per-tuple framework overhead



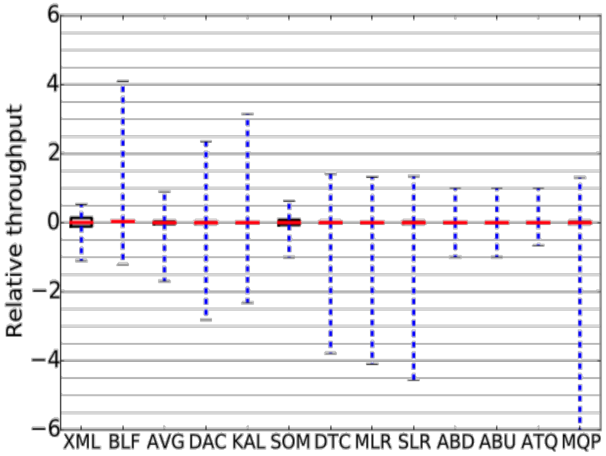
(a) Peak Throughput

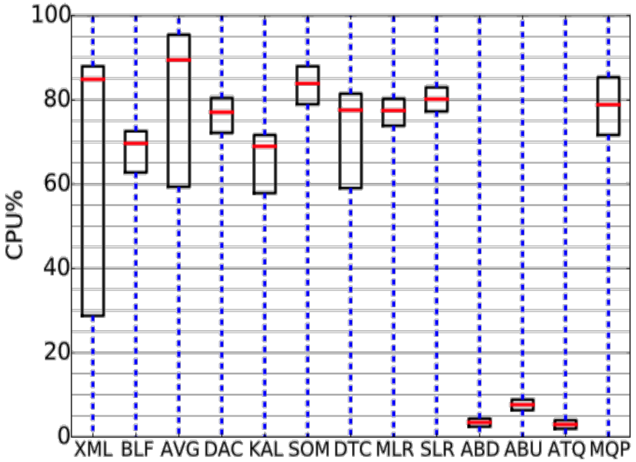


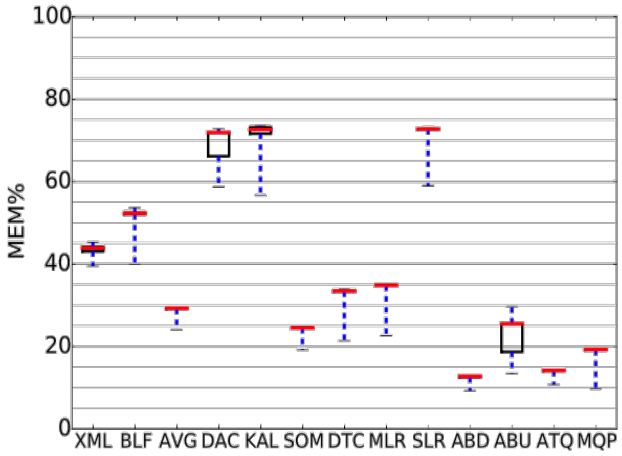
(b) End-to-end latency

Results: Micro-benchmark

- Jitter: *Close to zero* values indicates stability of Storm at peak rate without unsustainable queuing
- CPU and MEM utilization
  - Single-core VM used at 70% or above *except Azure tasks being I/O driven*
  - High memory utilization for tasks supporting high throughput indicates messages waiting in memory queue

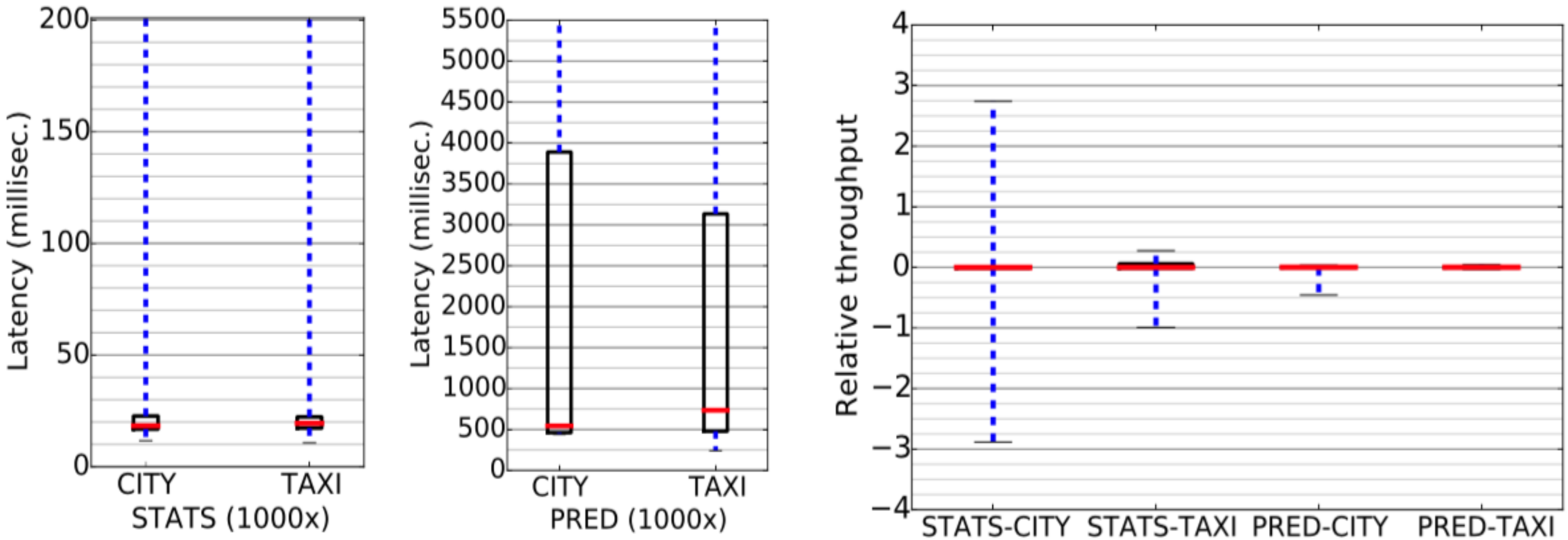






Results: Application benchmark

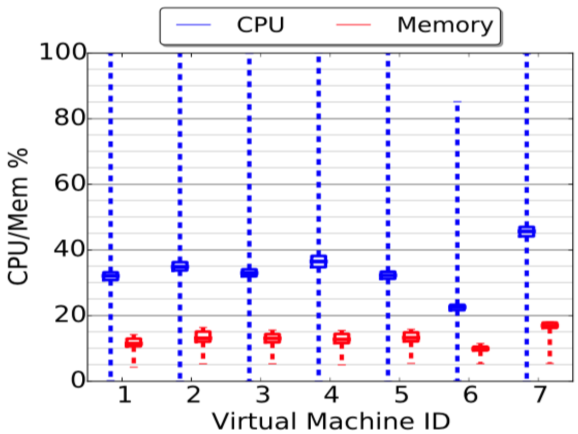
- PRED: Tall latency box plot due to variability in WEKA task execution times *[observed in micro-benchmarks for DTC and MLR]*
- Jitter remains close to zero, indicating sustainable performance



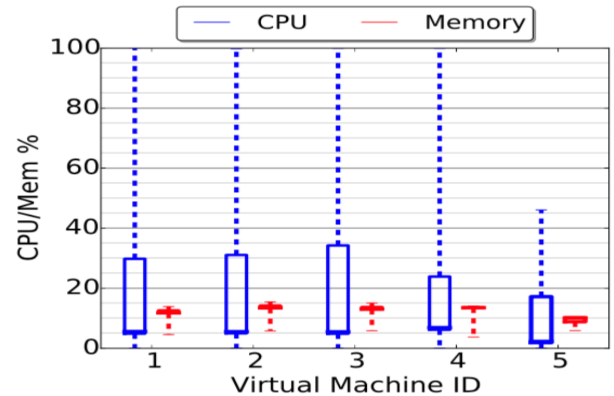
(a) Latency for STATS (b) Latency for PRED (c) Jitter

Results: Application benchmark

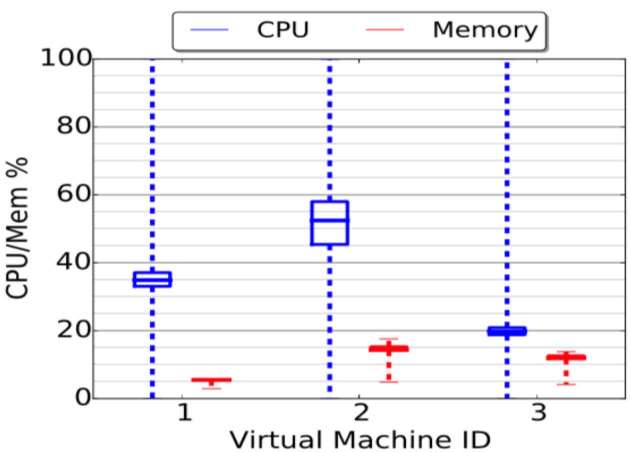
- TAXI : CPU is wider due to bimodal distribution of input rate
- X-axis represents the number of VMs required to support the given input distribution



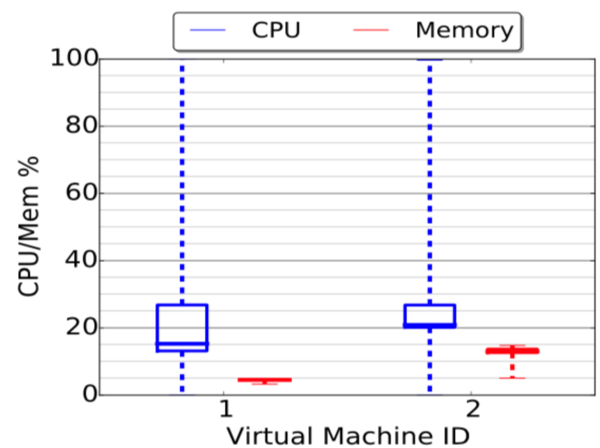
(d) STATS:CITY



(e) STATS:TAXI



(f) PRED:CITY



(g) PRED:TAXI

Summary & Future Work

- Classified essential tasks, IoT applications & data stream characteristics
- Metrics for performance & scalability needs for IoT applications from DSPS
- Evaluation of Micro and application benchmarks on Apache storm 1.0.0 for CITY and TAXI datasets in distributed environment
  - Indicates the viability of Apache Storm for IoT Application Domains

- Introducing new tasks to the task categories [SenML, CBOR parsing,Annotation]
- Benchmark and comparison of Apache Spark Streaming with Storm
- Using CEP engine like Siddhi as Tasks in Pattern recognition category

DREAM:Lab

Distributed Research on Emerging Applications & Machines

dream-lab.in

Indian Institute of Science

Thank you!

Contact : shukla@grads.cds.iisc.ac.in