ACM DEBS 2017

Grand Challenge: Runtime Anomaly Detection Method in Smart Factories using Machine Learning on RDF Event Streams

Joong-Hyun Choi[*], Kang-Woo Lee[+], Hyungkun Jung[*] and Eun-Sun Cho[*]

[*]Dept. of Computer Sci. & Eng., Chungnam Nat'l Univ.
[+]Electronics and Telecommunications Research Institute

2017.06.22

Outline

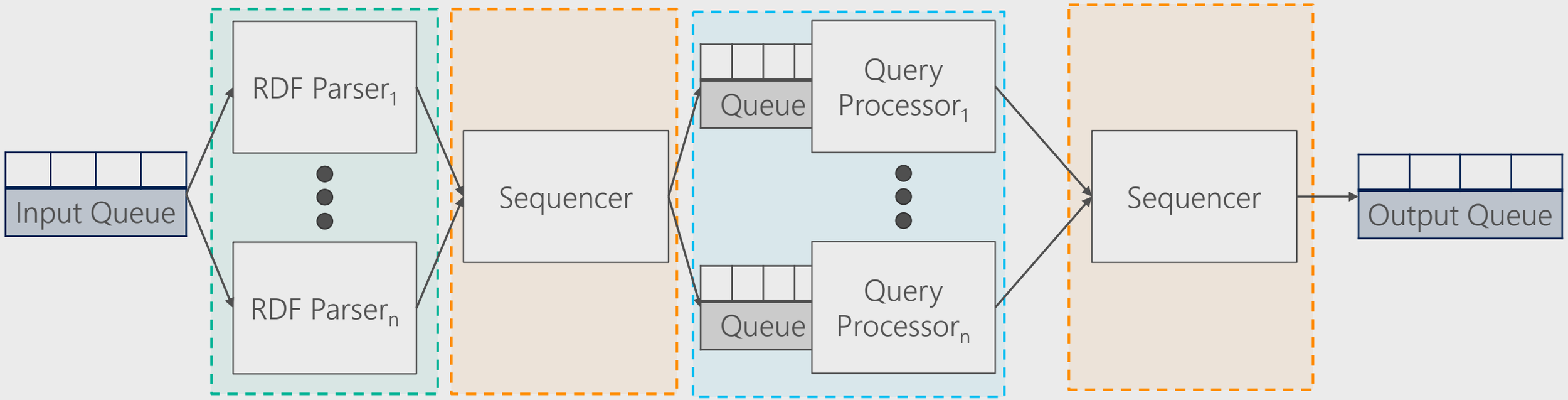
- The 2017 Grand Challenge
- Our Approaches
  - RDF Parser
  - Query Processor
  - Sequencer
- Experimental Results
- Summary

The 2017 ACM DEBS Grand Challenge

- Goal: anomaly detection of manufacturing equipment
  - Both the data set and the automated evaluation platform are provided by the HOBBIT project
  - All data are provided as RDF triples
  - The query has three stages: (1) Finding Clusters, (2) Training a Markov Model and (3) Finding Anomalies
  - Queries require continuously evaluated of cluster centers

Our Approach

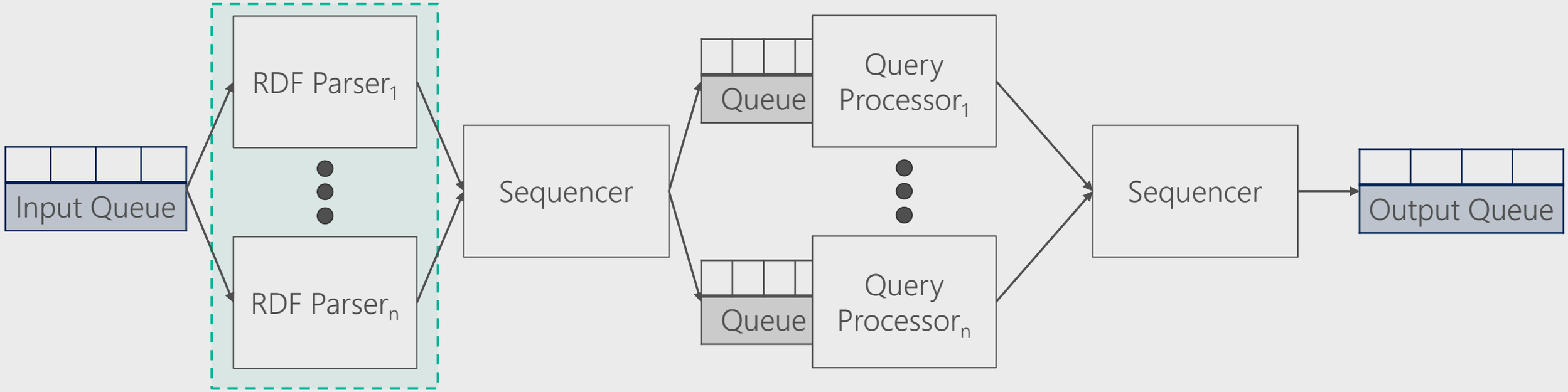
Our Approach



- Our solution has **three operators**
  - RDF Parser, Query Processor and Sequencer

- The evaluation platform provides two queues
  - Input Queue, Output Queue

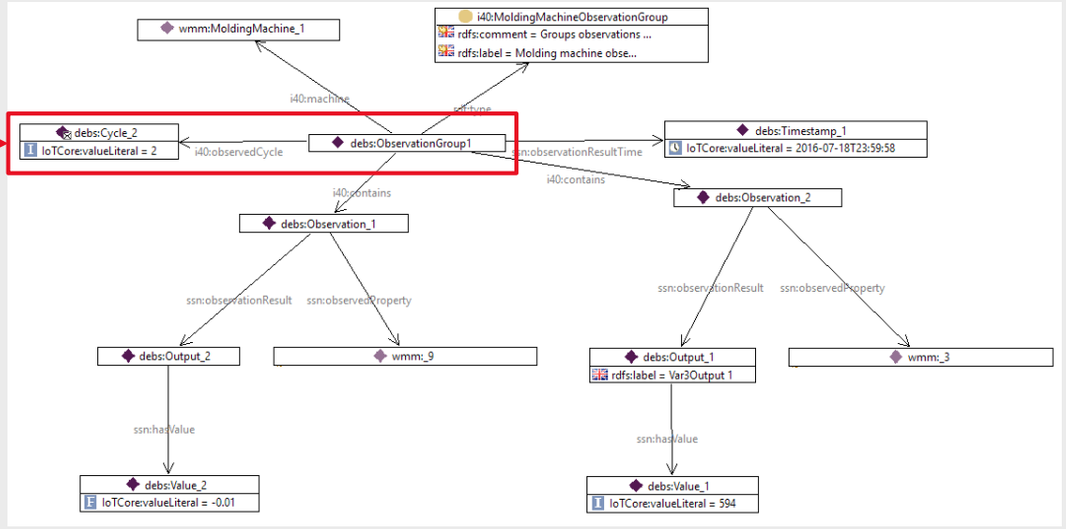
Our Approach



RDF Parser

- Events data of this year's challenge are provided in RDF tuples with sensor values
- Usually, it is not easy to handle RDF tuples

```
<debs#ObservationGroup_0> <I40#observedCycle> <debs#Cycle_0> .
<debs#Cycle_0> <rdf#type> <I40#Cycle> .
<debs#Cycle_0> <IotCore#valueLiteral> "13"^^<XML#int> .
```



- However, the RDF stream in challenge consists of two noticeable properties
  - It is represented in N-Triples
  - The order of the RDF triples is strict

Example of RDF Stream in GC

Very simple, easy to parse line-by-line

Machine that the event is raised from

```
<debs#ObservationGroup_0> <rdf#type> <I40#MoldingMachineObservationGroup> .
<debs#ObservationGroup_0> <ssn#observationResultTime> <debs#Timestamp_0> .
<debs#ObservationGroup_0> <I40#machine> <Metadata#Machine_59> .
<debs#ObservationGroup_0> <I40#observedCycle> <debs#Cycle_0> .
<debs#Cycle_0> <rdf#type> <I40#Cycle> .
<debs#Cycle_0> <IotCore#valueLiteral> "13"^^<XML#int> .
<debs#Timestamp_0> <rdf#type> <IotCore#Timestamp> .
```

➔ These features allow us to understand RDF streams without manipulating conventional complex graph structures

Sensor values

```
<debs#Observation_0> <ssn#observedProperty> <Metadata#_59_4> .
<debs#Output_0> <rdf#type> <ssn#SensorOutput> .
<debs#Output_0> <ssn#hasValue> <debs#Value_0> .
<debs#Value_0> <rdf#type> <I40#NumberValue> .
<debs#Value_0> <IotCore#valueLiteral> "9433.11"^^<XML#double> .
...
```

Experimental on various RDF Parsing Methods

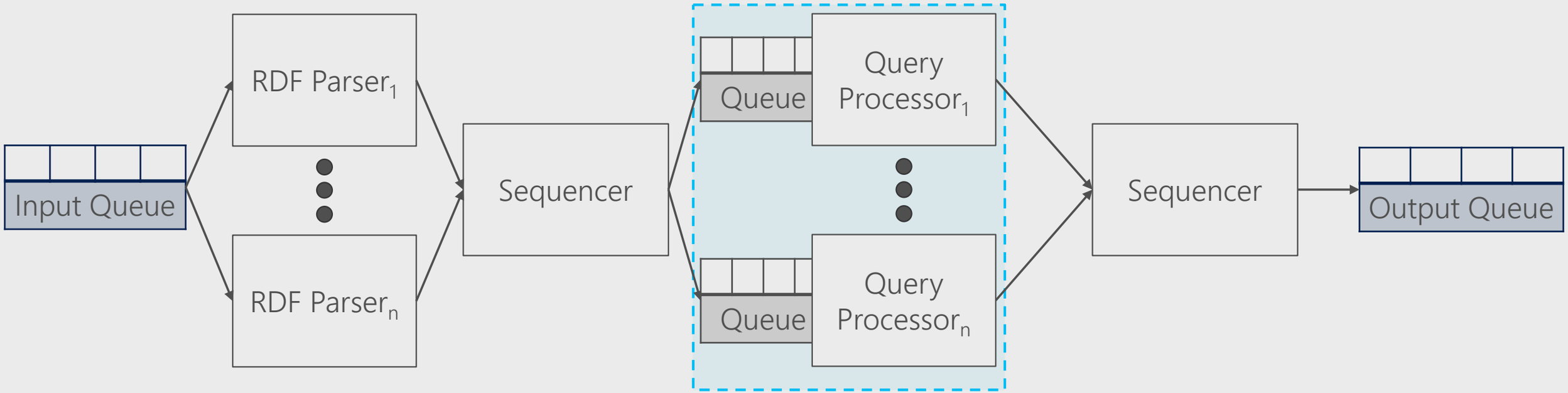
- The RDF data have a size of 7.9GB and translates to 50,000 events

	Total processing time(s)	Ratio relative to line-by-line parser
Jena SPARQL	234	13.2
Jena RDF API	131	7.4
Java Pattern Matcher	56	3.1
Line-by-line Parser (ours)	17	1

The Jena-based methods take longer than others because of its generality

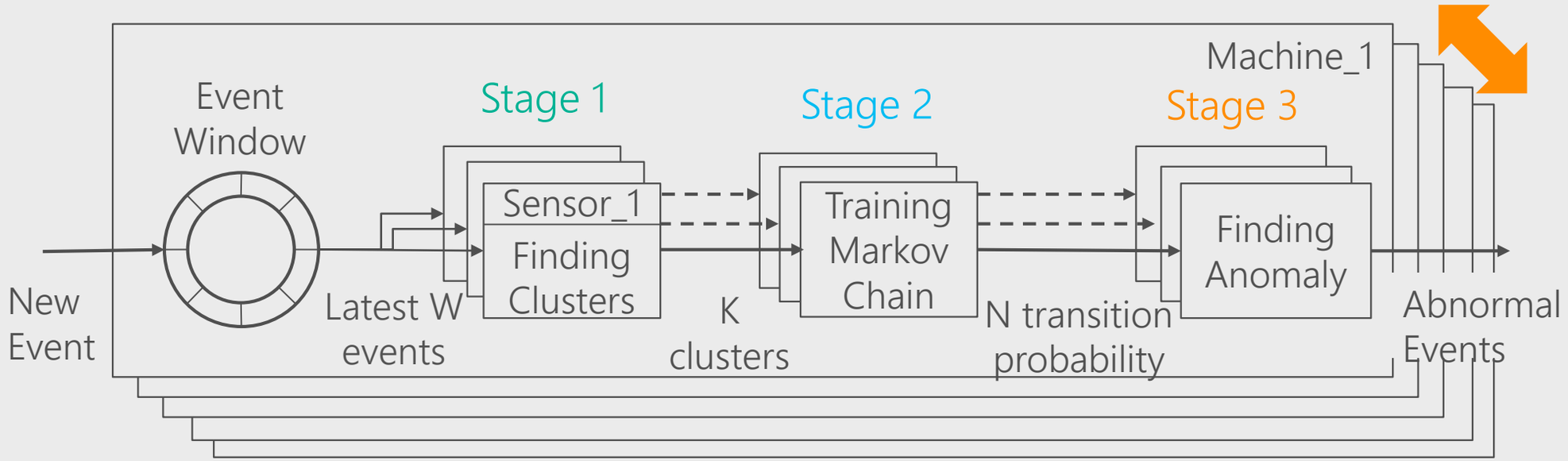
The best performance, dedicated solution
It's thought to be approve for the sensor event stream

Our Approach



Query Processor (1/2)

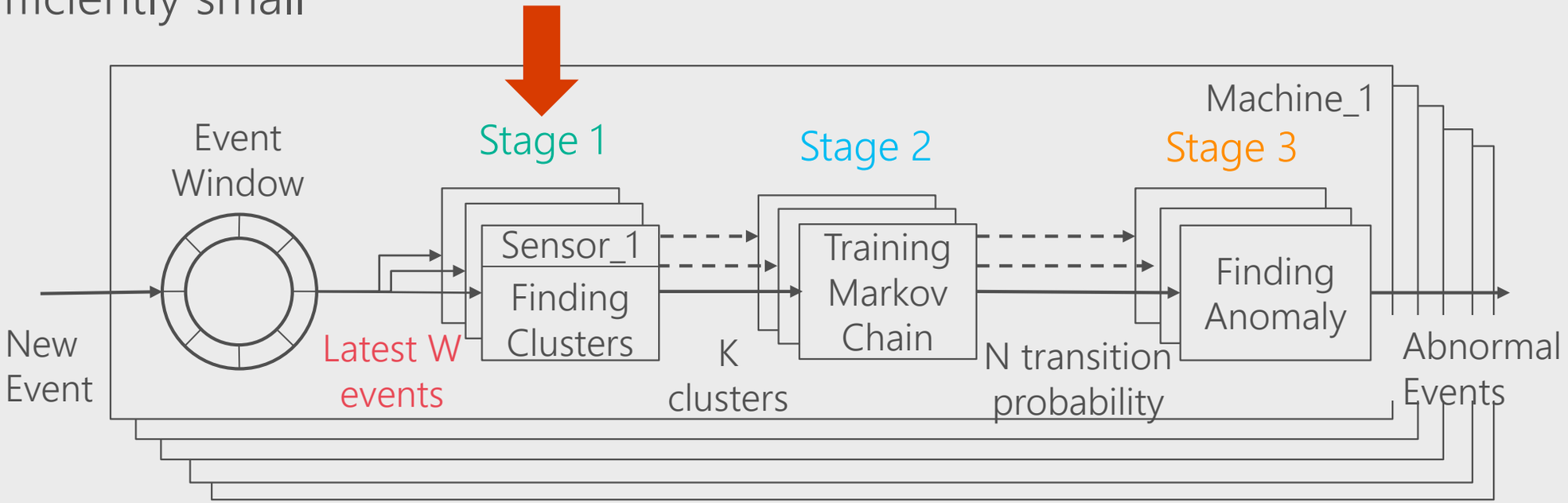
- Goal: to detect abnormal manufacturing equipment
  - The Query Processor has three stages
  - Anomalies of different sensors can be found simultaneously



Query Processor (2/2)

- Goal: to detect abnormal manufacturing equipment
  - The parameter W of a query is the main factor for our system performance
  - It's related to the number of elements used in clustering and training Markov chain

➔ Cluster calculation entails tremendous overhead unless the number of elements are not sufficiently small



Finding Clusters

- This year's challenge uses one-dimensional data for clustering
- There was an optimal K-means clustering method* for one-dimensional data
- However, we could not use this method in this year's grand challenge
  - They did not allow manual assignments of initial centroids
  - They did not allow us to follow our own conflict resolution principle when a single value matches multiple clusters

*Haizhou Wang and Mingzhou Song. 2011. Ckmeans. 1d. dp: optimal k-means clustering in one dimension by dynamic programming. The R journal 3, 2 (2011), 29.

A Revised K-Means Method (1/2)

- Idea: Matching only selected set of clusters in assignment phase
  - Initial centroids are sorted
  - Recalculating the distances between a given value and the centroids of selected clusters

A Revised K-Means Method (2/2)

- Example
  - K: 5, Values: 30, 20, 10, 42, 50, 36, 45, 45, 45

Initial Centroid	
C_1	10
C_2	20
C_3	30
C_4	42
C_5	50

→

Cluster Values
10
20
30
36,42,45,45,45
50

→

Updated Centroid	
C_1	10
C_2	20
C_3	30
C_4	42.6
C_5	50

A Revised K-Means Method (2/2)

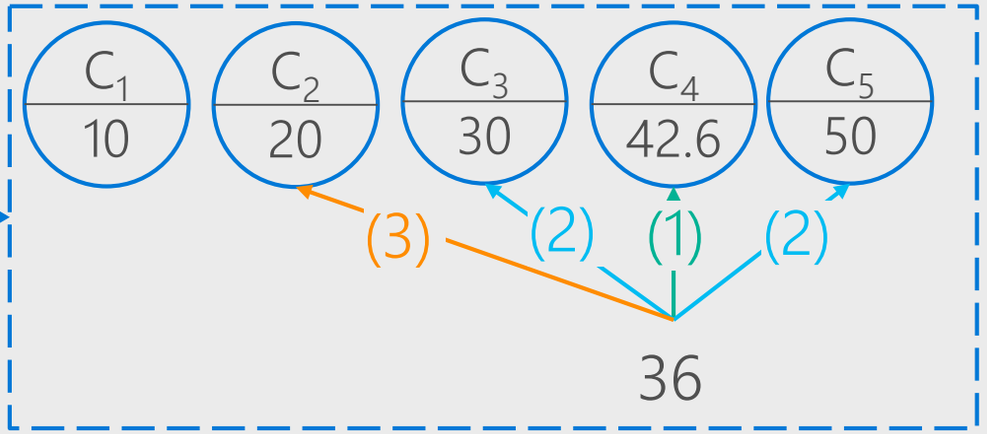
- Example
  - K: 5, Values: 30, 20, 10, 42, 50, 36, 45, 45, 45

- Recalculating distance of 36 value
  - (1) Calculating distance of previously assigned cluster
  - (2) Calculating distance of two nearest clusters from previously clusters
  - (3) Calculating distance of clusters in a direction with a small distance in (2)

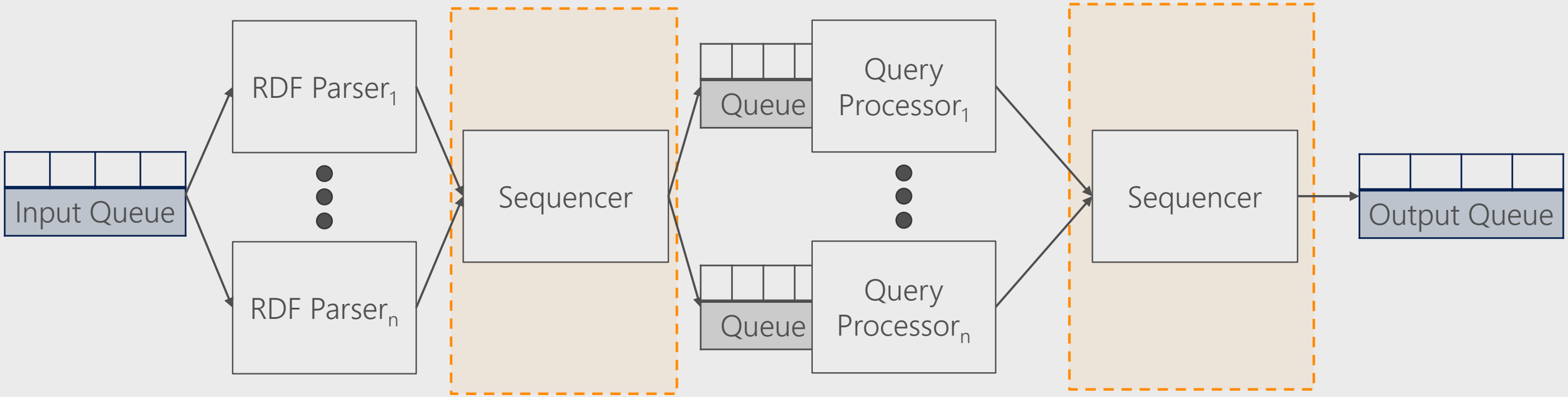
Initial Centroid	
C_1	10
C_2	20
C_3	30
C_4	42
C_5	50

Cluster Values
10
20
30
36,42,45,45,45
50

Updated Centroid	
C_1	10
C_2	20
C_3	30
C_4	42.6
C_5	50

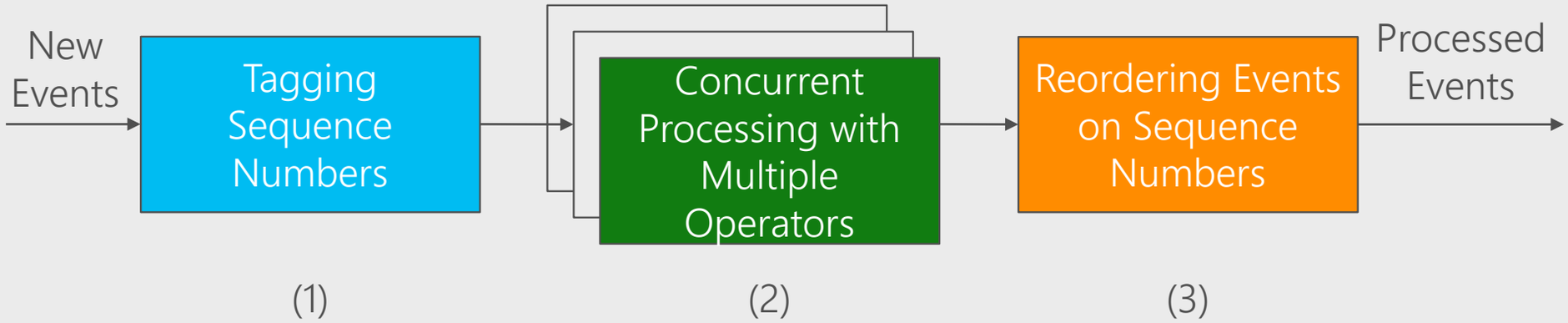


Our Approach



Sequencer

- Goal: To keep the original order of events,
  - We assign a sequence number to each event, and keep track and manage them



- To improve the throughput of our system,
  - We simply let multiple RDF Parsers and multiple Query Processors run concurrently

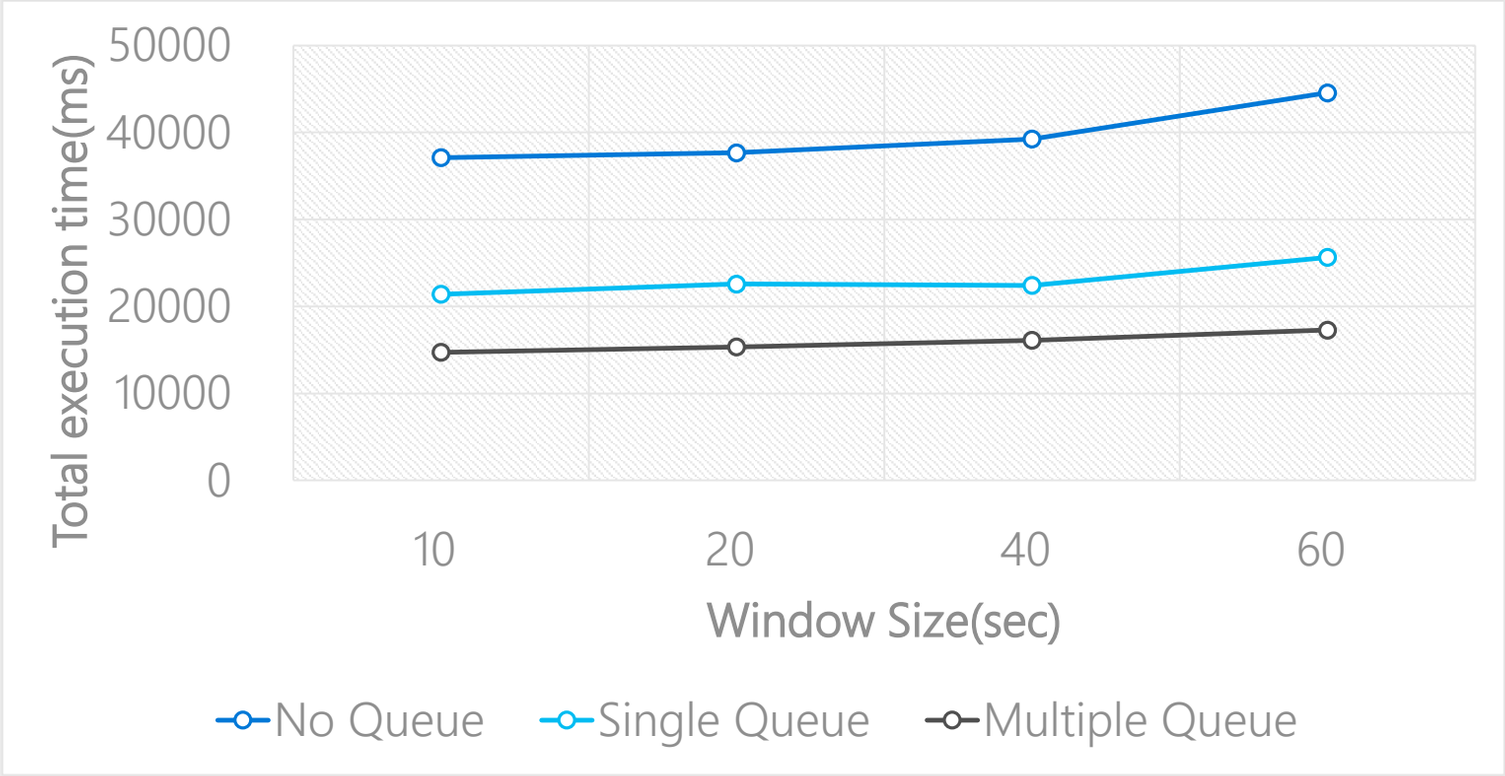
Experiments

Experiments

- The RDF data we used consist of 47 million triples and translated to 50,000 events
  - Ubuntu 16.04.2 LTS, Intel Core i7-6900K, 96GB Memory, OpenJDK Runtime Environment (build 1.8.0_131).
- Test with three level of concurrency
  - **No Queue Mode**: events cannot be processed at the same time
  - **Single Queue Mode**: task parallelism (pipelining). RDF parser and the Query Processor can run concurrently
  - **Multiple Queue Mode**: data parallel processing. Event from different manufacturing equipment executed concurrently.

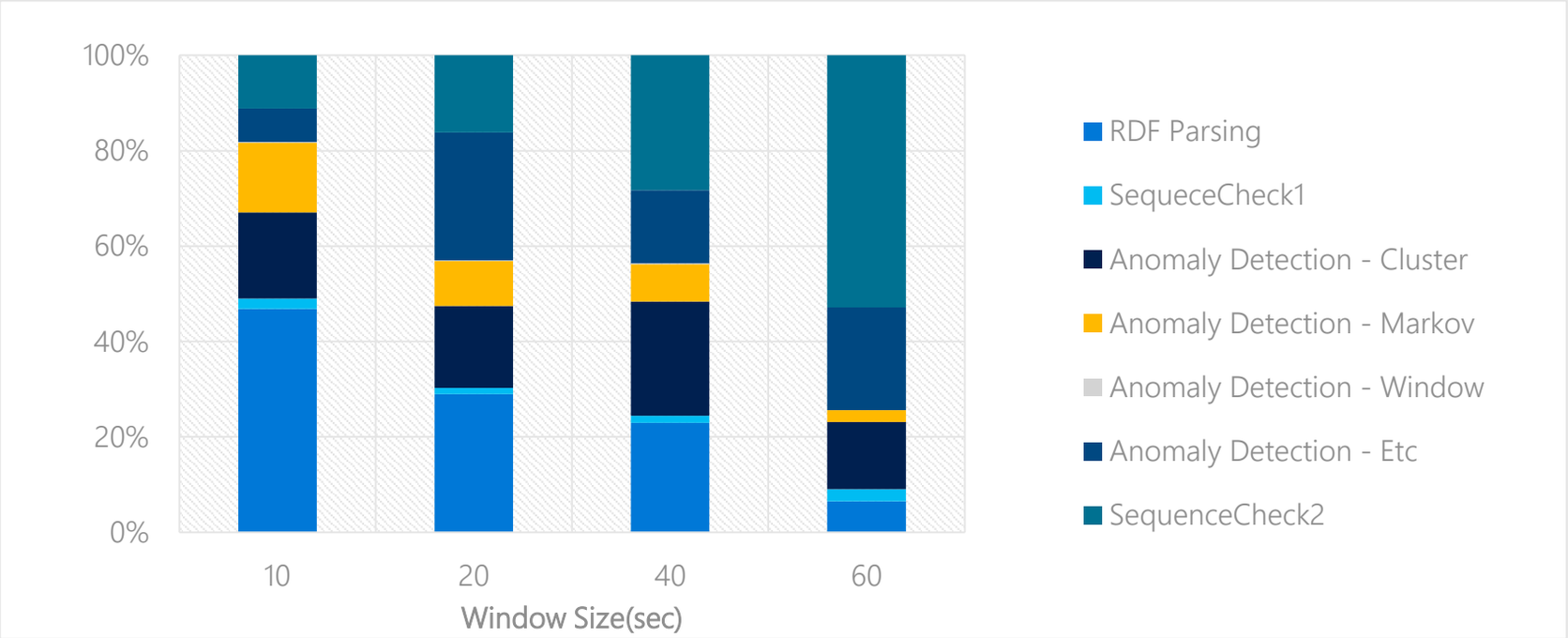
Experiments: Total Execution Times

- The processing time decreases as the queue usage increases
  - No Queue Mode and Single Queue Mode tests took 2.5 times and 1.4 times longer, respectively, compared to the Multiple Queue Mode



Experiments: Portion of Event Processing Time

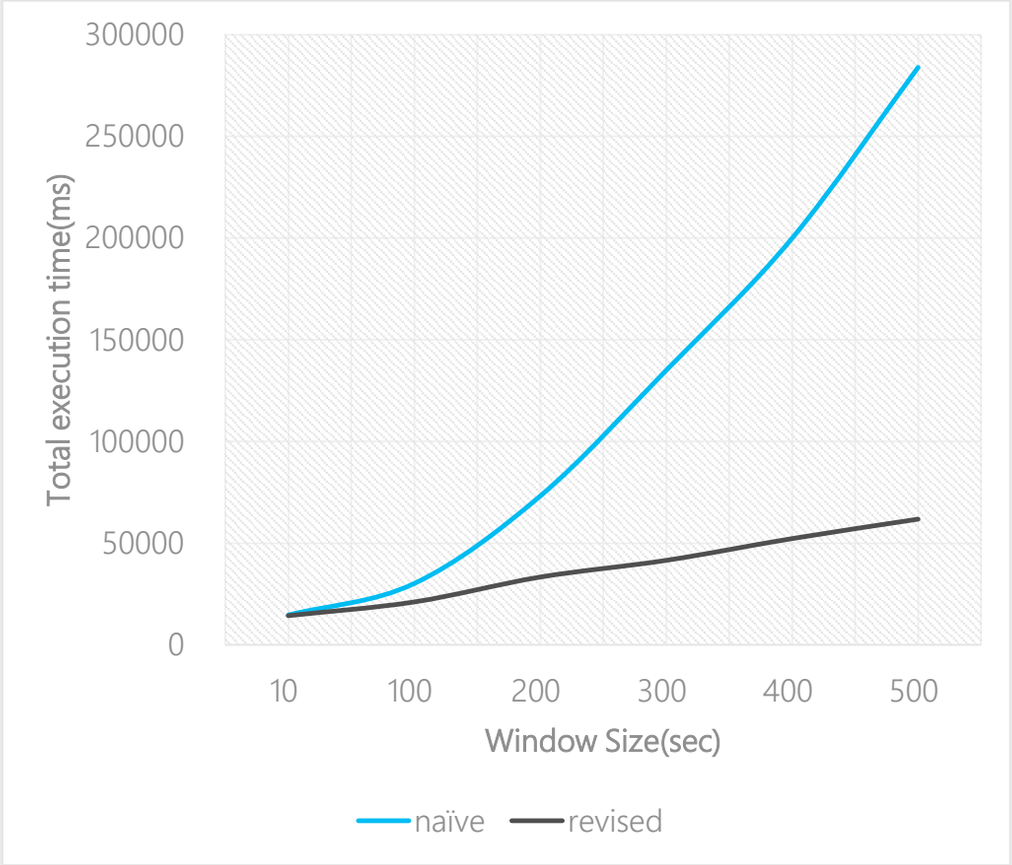
- As the value Window size increases, the portion of other computations increase noticeably compared to the parsing operation
  - The cost of the query processing and the context-switch of the CPU increases
  - CPU idle time for each event also increases



Experiments: A Revised K-Means Method (1/2)

The total execution time of the standard algorithm grows more sharply

Our algorithm is more scalable



Experiments: A Revised K-Means Method (2/2)

Standard algorithm

It takes O($n \times k \times i$)

k:centroids, *n*: values, *i*: repetitions

Our revised algorithm

It takes O($n \times c \times i$)

$c \approx 3$

i

n

k

c

Algorithm 2: Our revised algorithm

```
1  Select K points as the initial centroids
2  Sort the initial centroids
3  repeat
4      foreach point in the All Points do
5          D_prev ← distance between point and previously assigned cluster
6          D_left,right ← distance between point and nearest two clusters from previously assigned cluster
7          if D_left < D_prev then
8              C_new ← search cluster with mininum distance on the left
9          else if D_right < D_prev then
10             C_new ← search cluster with mininum distance on the right
11         else
12             C_new ← previously assigned cluster
13         assign point to the C_new cluster
14     Recompute the centroid of each cluster.
   until The centroids don't change
```

Summary

- Our solution
  - RDF triples are processed efficiently by our line-by-line parser
  - A revised k-clustering algorithm
  - High degree of concurrency in continuous query processing

http://www.debs2017.org/gc/averageLatencyNanos	-467091407	-1	214688000	64232000	49928000	533224704	-398817472	-1
http://www.debs2017.org/gc/throughputBytesPerSecond	2220683.8346539307386	1171138.5298897433095	2219930.6918117105961	1483408.8966442481615	1499940.5361645654775	1411076.9469536351971	2004779.6041545209009	1487190.3705458182376

Thank you!!

eclipse@cnu.ac.kr

PLAS: Programming Languages and System Lab
5th Engineering Building, Chungnam National University, Daejeon, Republic of Korea.