Stateful Load Balancing for Parallel Stream Processing

Qingsong Guo, Yongluan Zhou

North University of China, University of Copenhagen

Auto-DaSP 2017 - August 29, 2017

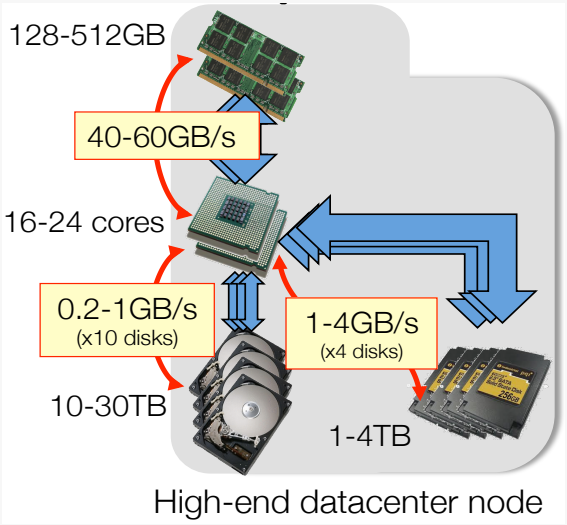
Stream Processing in 20 Years

- Data Stream Management Systems (DSMS)
  - TelegraphCQ, STREAM, Gigscope, Aurora/Borealis, System S, etc.
  - Continuous query (CQ)
  - Low-latency processing: batching → streaming
  - Query results: deterministic → non-deterministic
  - etc.
- When stream meets big data
  - S4, Storm, Spark Streaming, StreamScope, Flink, Kafka/Samza, Millwheel/Dataflow, etc.
  - 3Vs of big data: volume, velocity, variety
  - Scalability, elasticity, task scheduler, fault tolerance etc.
  - Uniformed: batching + streaming

Achieve Real-time Processing

Leverage memory

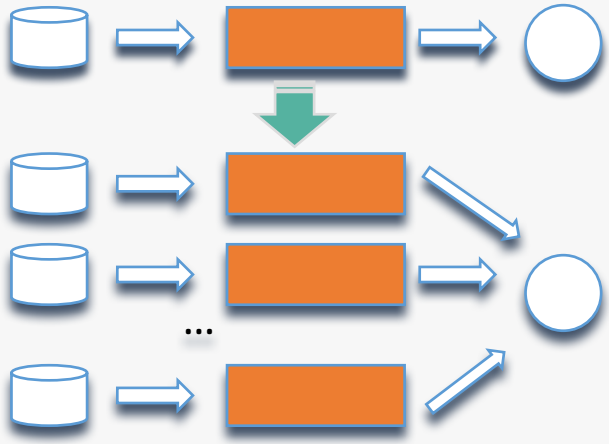
The inputs of over 90% of jobs in Facebook, Yahoo!, and Bing clusters are fitted into memory



Increase parallelism

Reduce work per node improves latency

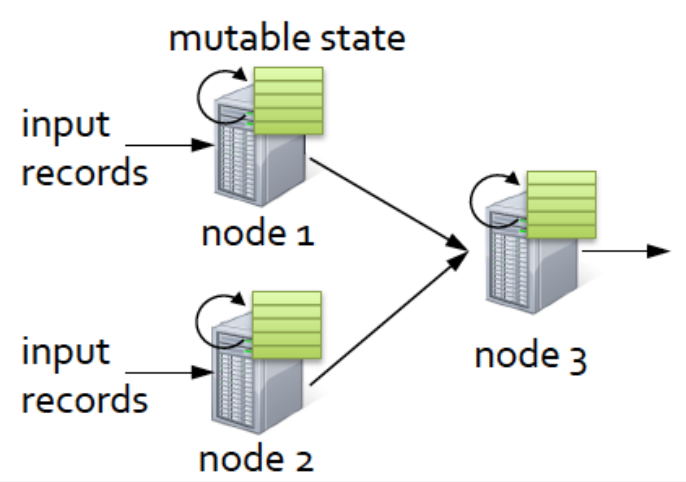
- Low latency scheduler
- Globe state management
- Efficient failure recovery
- Optimization of communication patterns: e.g., shuffle, broadcast



Stateful Stream Processing

State is introduced for reasons such as window-based computation, buffering, fault tolerance, etc.

- Traditional model
  - Processing pipeline of nodes
  - Each node maintains mutable state
  - Each input record updates the state and new records are sent out



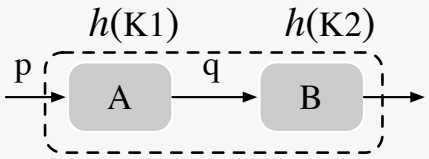
- Reconfiguration of computation
  - Mutable state get lost if node fails
  - State should be redistributed across nodes if the placement plan changes
  - Runtime adaptation for load variations, scale-out/scale-in
  - All these cases involve state migration

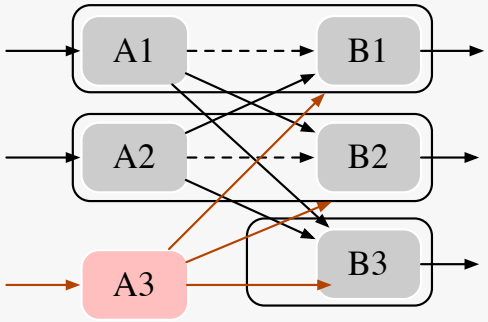
State Migration

Pause-migration-resume procedure

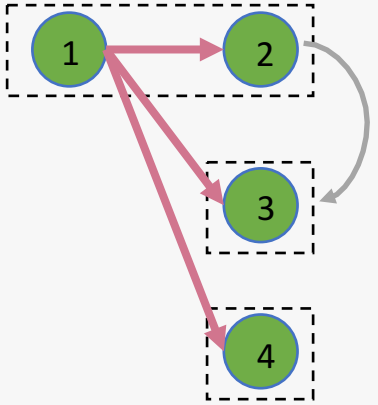
- Pause the execution (o2)
- Install new operator on target node (O2 on node 2)
- Serialize state of O2 & send to new node (node 2)
- Redirect tuples to new node (node 1 → node 2)

State migration is time-consuming and dominate

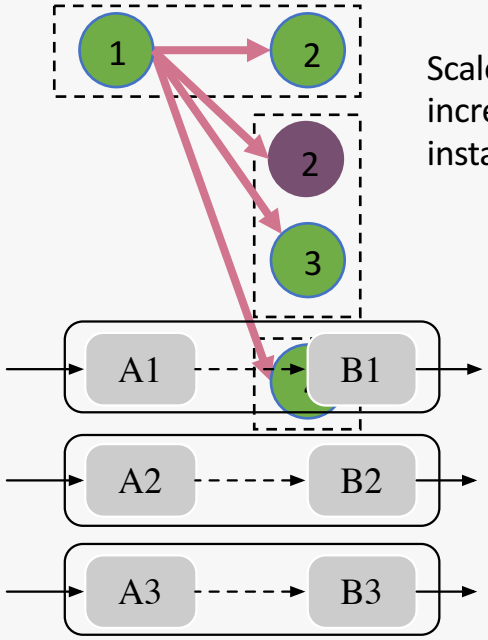




Example job: 4 operators, 3 nodes



Migrate O2, from node 1 to node 2



Scale-out by increase an instance for O2

Challenge from Load Variations

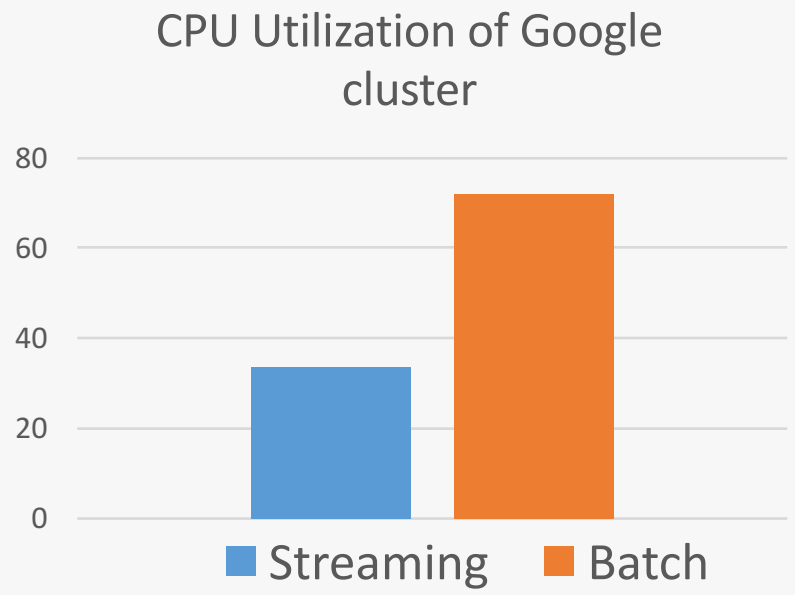
Problems of load variations

1. Unmatched provision: low resource efficiency
2. Load imbalance: low processing latency and bad system throughput

Handling load variations

Operator placement & Adaptations

1. Dynamic scaling
2. Load balancing



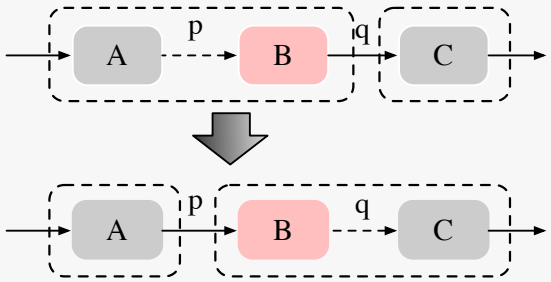
Two Optimization Problems

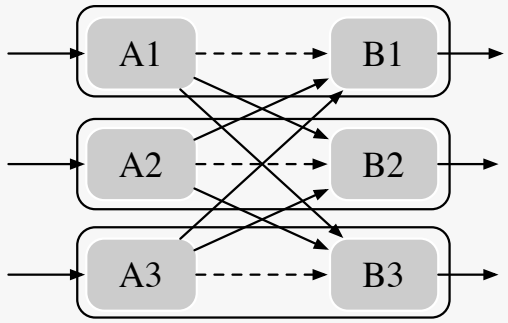
Dynamic reconfiguration

Load variations, e.g., changes of data rate and data distribution
Commensurate provision: adaptive data partitioning to achieve load balancing and to scale the number of parallel instances of each operator to avoid over-provisioning or under-provisioning.

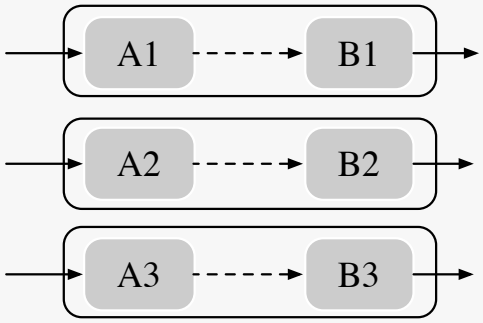
Communication minimization

Reduce data shuffle
Optimizing the operator placement
to minimize cross-node communication can significantly reduce the resource consumption in a DSPE.





(b)



(c)

Project Enorm

Enorm was launched in 2013 at SDU, 1 faculty and 3 PhD students
A *distributed stream processing engine* (DSPE) extends Apache Storm with some essential properties such as

- flexible window computation,
- elastic resource management, operator placement strategies,
- automatic scaling,
- load balancing,
- globe state management,
- optimized fault-tolerance,
- etc.

Enorm borrows the basic concept from Storm

- Spout, Bolt, Topology (operator graph)
- Task & its execution model
- Stream grouping schemes

Basic Concepts of Storm

Spout

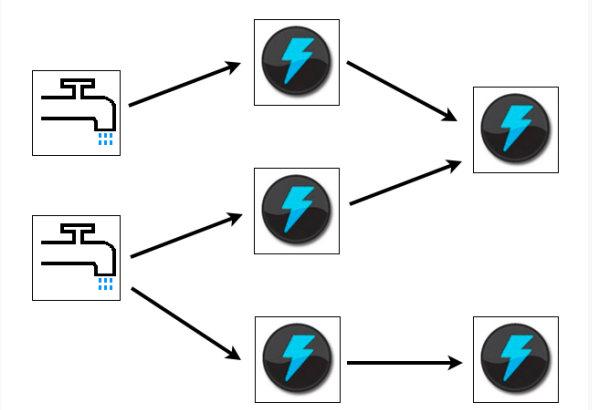
Ingest source streams from Kestrel and Kafka queues or read data from Twitter streaming API HDFS, Hive, etc.

Bolts

Processes input streams and produces new streams. It could be user-defined functions or standard SQL operators, such as Filters, Aggregation, Joins, etc.

Topology (operator graph)

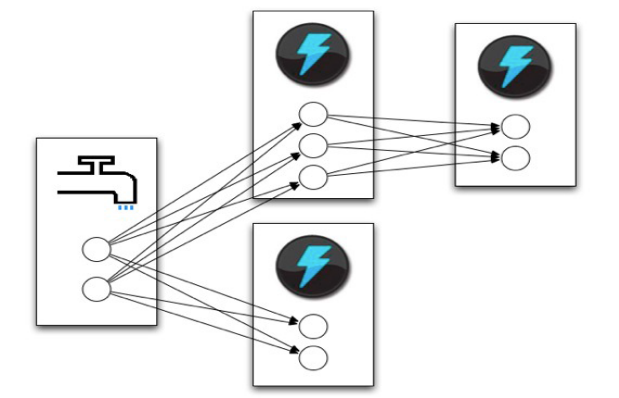
A directed acyclic graph(DAG) of spouts and bolts

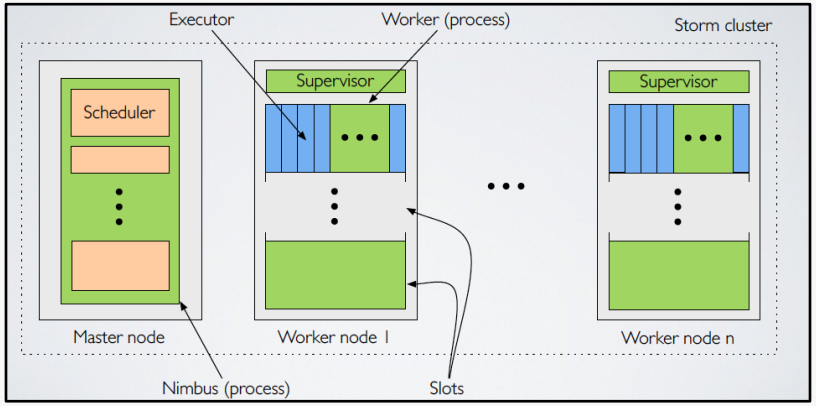


Parallel Stream Processing with Storm

Task & execution

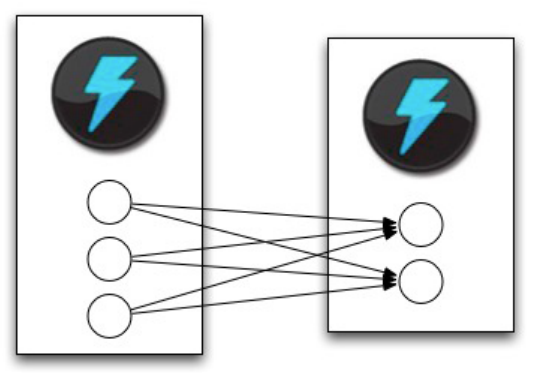
Spouts and bolts execute as many tasks
Tasks are scheduled and spread across the storm cluster





Stream grouping

It defines how to dispatch output tuples.
Shuffle grouping: pick a random task
Fields grouping: mod hashing on a subset of tuple fields
All grouping: send to all tasks

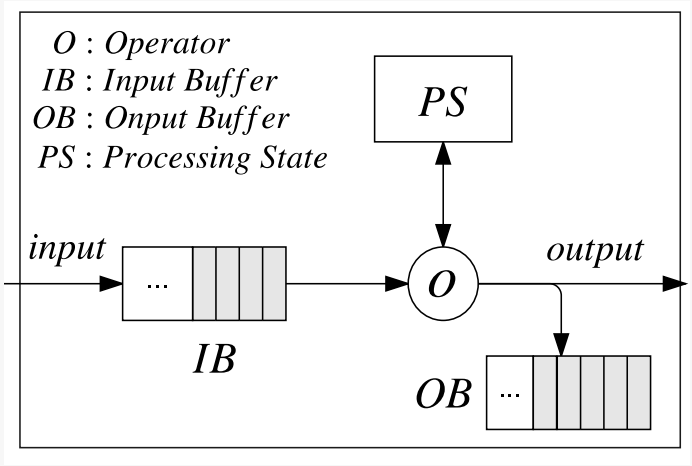


Operator Model

Operator model

Input, **output**: relational stream
Function, **sliding window**
Processing state, e.g., stateful or stateless
Partition key for stream grouping

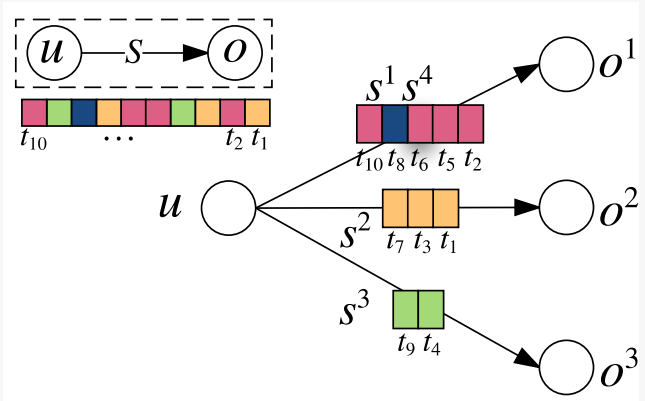
1. **Shuffle grouping** for stateless operator
2. **Key grouping** for stateful operator



Parallel processing

Operator instances $\mathcal{I} = \{o^1, \cdots, o^m\}$
Substreams $\mathcal{S} = \{s^1, \cdots, s^p\}$
Assignment $\mathcal{F}c: \mathcal{S} \rightarrow \mathcal{I}$

Plan 1: $s^1, s^4 \rightarrow o^1, s^2 \rightarrow o^2, s^3 \rightarrow o^3$
Plan 2: $s^1 \rightarrow o^1, s^2 \rightarrow o^2, s^3, s^4 \rightarrow o^3$



Component-based Parallelization(CBP)

Component

An induced subgraph C of the operator graph is said to be a *component* if and only if C is connected and its operators are compatible.

Compatibility: A set of operators are compatible iff the intersection of their partitioning keys is not empty. It is non-transitive.

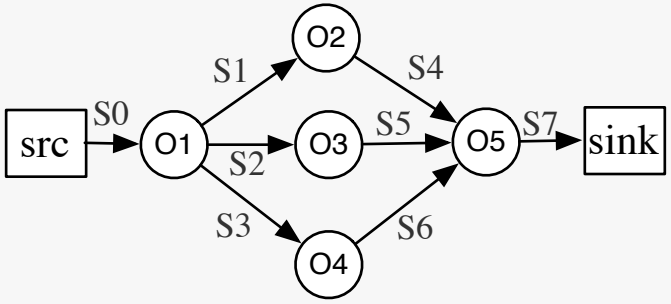
Connectivity: Communication in a node is replaced by local memory access and thus intra-component communications are eliminated.

Example

A simplified version of Linear Road Benchmark that calculates tolls with a *position-speed* stream

5 operators: traffic statistics, accident detection, toll calculation ,…

Partitioning compatibility: Operators has common attribute

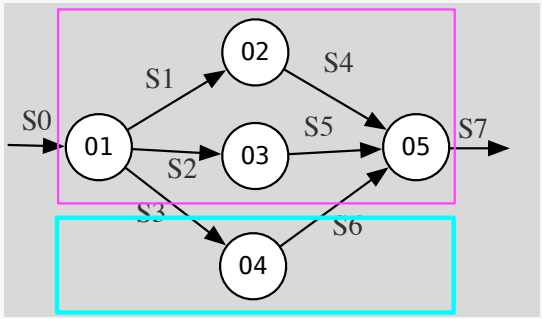


Operator	Partition Key
O1:Forwarder	{*Ts*, *Vid*, *XWay*, *Dir*, *Seg*, *Spd*, *Pos*, Type}
O2:AcdDetector	{*Ts*, *Vid*}
O3:AvgSpeed	{*Vid*, *XWay*, *Dir*, *Seg*}
O4:SegVolume	{*Xway*, *Dir*, *Seg*}
O5:TollCalculator	{*Xway*, *Dir*, *Seg*}

Component-based Parallelization

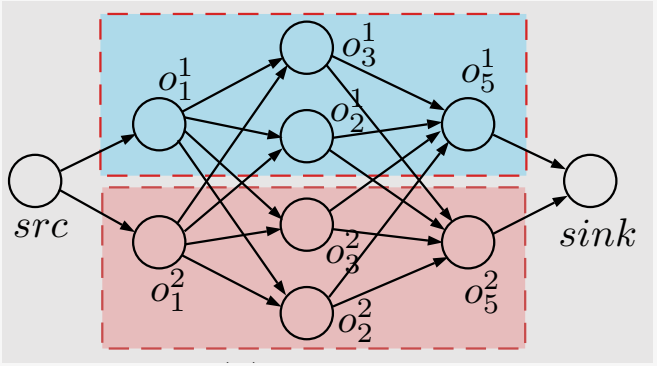
Optimization goals

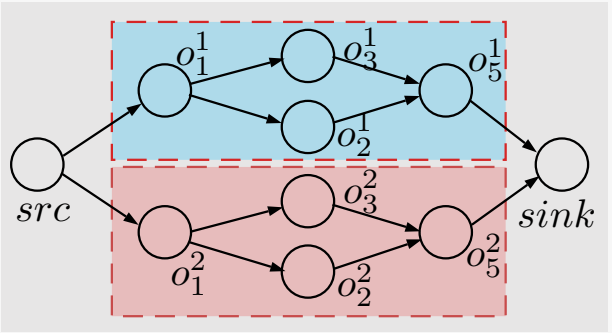
1. Runtime resource reconfiguration:
   Unmatched provision, imbalance
2. Communication cost minimization:
   Operator placement, task allocation



CBP essentials

Leverage the compatibility of operators
Intra-query parallelism
Intra-operator parallelism
Scalability of OBP





Grouping Schemes

Key grouping and state movement

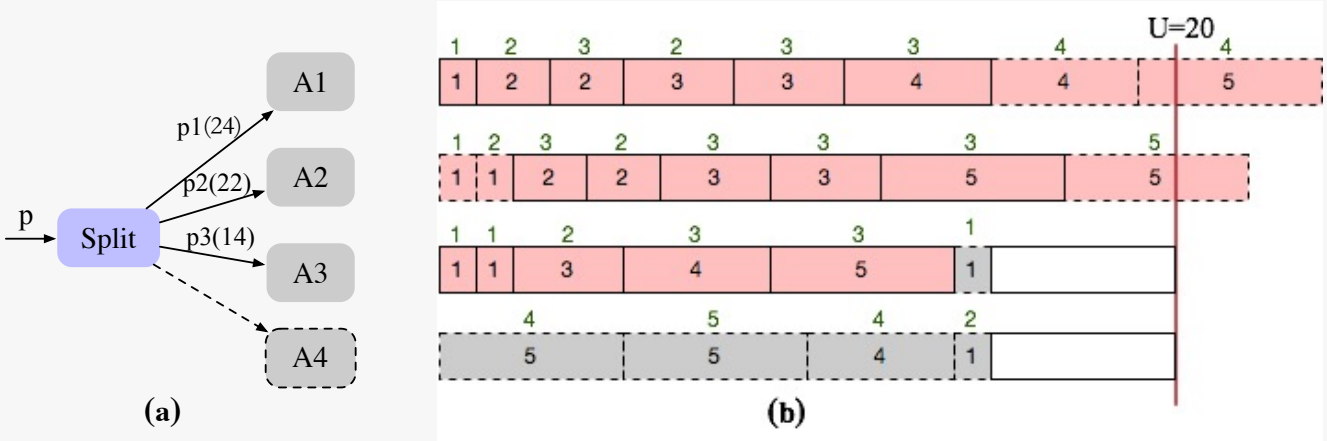
Load balancing in adaptation

Each substream is marked with the number of tuples

The same number of state partition with its load

Change the assignment

16 state movements



Minimum Cost Load Balancing (MCLB)

Problem Statement

- Given a uneven assignment F1, the execution of load balancing is to compute a new assignment F2 that balances load for all instances.
- The MCLB problem asks for such an assignment F2 with the minimum state movements.

- Bi-objective optimization problem
- Complexity
  - NP-hard
  - Approximate solutions

Statistics Measurement

- Substreams $s_1 \ldots s_p$ statistic windows of length Δ
- Histogram $Y_t = (y_{1t}, y_{2t}, \ldots, y_{pt})^T$
  - yit records the load for si at the t-th window

Average load & load variance

$$\begin{cases} \bar{y}_t = \frac{1}{p} \sum_{i=1}^{p} y_{it} \\ var(Y_t) = E[Y_t^2] - \big(E[Y_t]\big)^2 \end{cases}$$

- Substream si can be represented as a load series $X_i = (y_{i1}, \ldots, y_{im})$

Average load & load variance

$$\begin{cases} E[X_i] = \frac{1}{m} \sum_{t=1}^{m} y_{it} \\ var(X_i) = E[X_i^2] - \big(E[X_i]\big)^2 \end{cases}$$

$$\begin{cases} cov(X_1, X_2) = E[X_1 X_2] - E[X_1]E[X_2] \\ \rho_{12} = \frac{cov(X_1, X_2)}{\sqrt{var(X_1)} \cdot \sqrt{var(X_2)}} \end{cases}$$

a_ij=1 if s_i is assigned to instance o^j

Metrics

- Encoding the assignment as a matrix $\boldsymbol{A} = [a_{ij}]_{p \times n}$
- For instances $(o^1, \ldots, o^n)$, the load vector $L_t = (l_{1t}, l_{2t}, \ldots, l_{nt})^T$ at t-th window is given by a linear transformation $\boldsymbol{A}^T Y_t = \bar{\boldsymbol{l}}_{\boldsymbol{t}}$

Load imbalance

$$var(L_t) = \frac{1}{n} \sum_{i=1}^{n} (l_{it} - \bar{l}_t)^2$$

Normally we use $|max - min|$ to measure imbalance

State movements

Given an uneven assignment F1 and a new assignment F2, a state partition psi will be moved to another instance if the allocations given by F1 and F2 are different, i.e., $\mathcal{F}_1(s_i) \neq \mathcal{F}_2(s_i)$

$$\psi(\mathcal{F}_1, \mathcal{F}_2) = \boldsymbol{x} \cdot \boldsymbol{d} = \sum_{i=1}^{p} x_i d_i$$

x_i is a binary variable,
$x_i = 1$ if $\mathcal{F}_1(s_i) \neq \mathcal{F}_2(s_i)$

$\boldsymbol{d} = (d_1, \ldots, d_p)^T$

Eager Load Balancing (ELB)

Basic idea

- ELB performs LB eagerly for each statistic window and attempts to reduce state movements as many as possible
- Heuristics:
  - (1) Distribute hot spots as evenly as possible
  - (2) Fit the load of each instance into [v, u] and make it close to $\frac{v+u}{2}$

Phase 1: identify overloaded and underloaded instances

- Calculate load vector $L_t = (l_{1t}, \ldots, l_{nt})$ with Y_t and $\mathcal{F}_1$
- Calculate overall load $w = \sum_{j=1}^{n} l_{jt}$, average load $\bar{l} = \frac{w}{\pi}$, and new parallelism $\pi = \lceil \frac{2w}{u+v} \rceil$
- Add/remove $|\pi - n|$ and identify overloaded instance set OI and under loaded instances UI by compare their load with $\bar{l} = \frac{w}{\pi}$

Eager Load Balancing (example)

Phase 2: identify substreams to be reassigned

1. Each time we choose the largest substream from an overloaded instance < $\theta = \min\{l_{jt} - \bar{l}_t, \frac{u-v}{2}\}$

Phase 3: reassign the identified substreams in PQ

1. Substreams in UI are listed a descending order of loads
2. The reassignment processes in a first-fit procedure
3. The instance will be removed from UI and added into OI if it is overloaded

Correlation-based Load Balancing (CLB)

Basic idea

1. CLB execute a LB every m (m>1) statistic windows with an assignment that fits for the m histograms $\boldsymbol{Y} = (Y_1, \ldots, Y_m)$
2. The cost for state movements can be ignored if m is large enough.
3. Substreams are view as load series and to reduce imbalance by minimize correlation among the substreams assigned to the same instance.

Overall load imbalance

$$\sum_{j=1}^{m} var(L_j) = \sum_{j=1}^{m} (\frac{1}{n} \sum_{i=1}^{n} l_{ij}^2 - \bar{l}_j^2) = \frac{1}{n} \sum_{j=1}^{m} \sum_{i=1}^{n} l_{ij}^2 - \sum_{j=1}^{m} \bar{l}_j^2$$

For the i-th instance assigned with substreams $S_i = \{s_1, \ldots, s_r\}$

$$\sum_{i=1}^{n} var(N_i) = \frac{1}{m} \sum_{i=1}^{n} \sum_{j=1}^{m} l_{ij}^2 - \sum_{i=1}^{n} \eta_i^2$$

$$\eta_i = E(N_i) = \sum_{s_i \in S_i}^{|S_i|} E(X_i)$$

Load series $N_i = X_1 + \cdots + X_r$

The equivalence

$$\min \sum_{j=1}^{m} var(L_j) \Leftrightarrow \min \sum_{i=1}^{n} var(N_i)$$

Correlation-based Load Balancing (cont.)

Load series $X = X_1 + \cdots + X_p$

$var(X) = var(X_1 + \cdots + X_p)$

In addition

$$var(X) - \sum_{k=1}^{n} var(N_k) = 2 \sum_{X_i \in S_k, X_j \in S_z, k \neq z} cov(X_i, X_j)$$

The right component is *cross covariance* which counts the covariance of the substreams that falls into different subsets

Minimize load imbalance $\hbar(\mathcal{F})$, is equivalent to a partition of S into subsets S1…Sn that maximize $var(X) - \sum_{k=1}^{n} var(N_k)$

$$\min \hbar(\mathcal{F}) \Leftrightarrow \max var(X) - \sum_{k=1}^{n} var(N_k)$$

Experimental Evaluation

Tested solutions

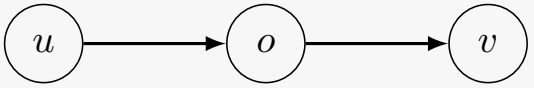
ELB, **CLB**

PKG: Implements key grouping that tuples are randomly distributed to two downstream instances, but it is designed for stateless LB

UHLB: Universal hash function rather than key grouping

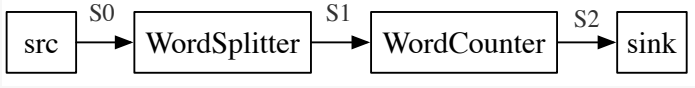
Simulation

- A simple topology with 3 operators
- Load imbalance
- Percentage of state movements

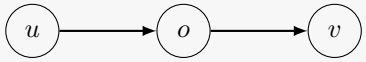


Processing latency

- A simple topology for counting words every 1 minute
- Processing latency
- Speedup of throughput



Simulation Results



Synthetic stream s1, s2

Data rate: Poisson process X(t) $\lambda : Prob\{Z_m \leq \tau\} = 1 - e^{-\lambda\tau}, \lambda = 10000$

Distribution: Gaussian and Zipf

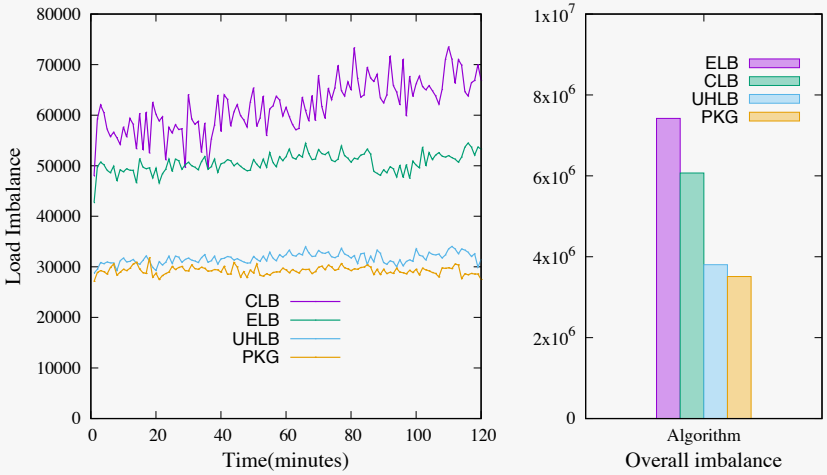


Fig.1 Load imbalance over time

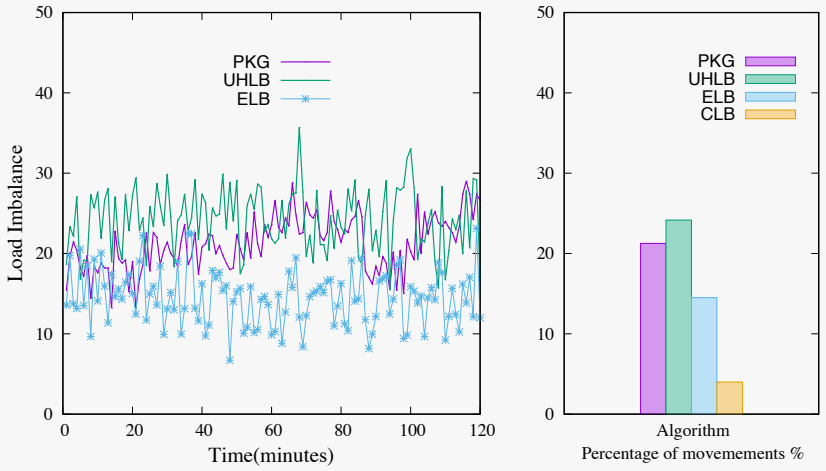
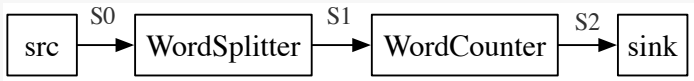


Fig.2 Percentage of state movements

Processing Latency



The experiments are conducted on EC2 with medium VM instances.

We evaluated the processing latency by explicitly scaling out the operator WordCounter that counts the occurrence for each word every 1 minute over the Twitter stream.

latency	CLB	ELB	PKG	UHLB
max	1103.13	1109.51	1551.30	1505.13
mean	0.76	0.73	0.92	1.01
median	0.30	0.33	0.38	0.38
95%	1.12	0.68	1.70	1.89

Table 1. Processing latencies (*ms*)

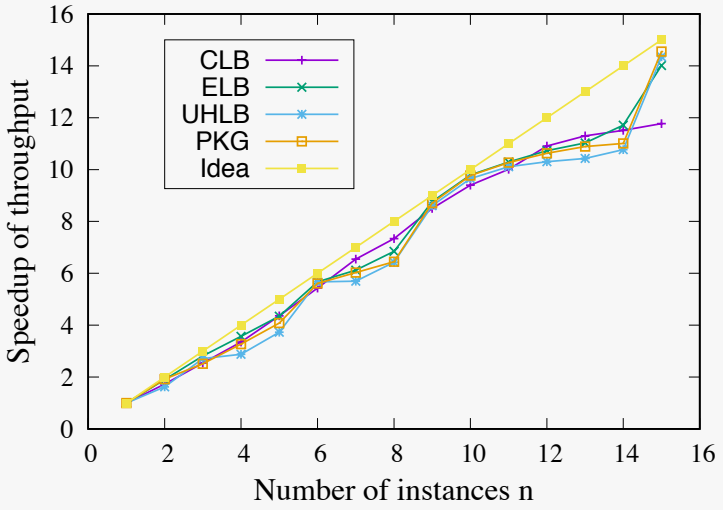


Fig 3. Speedup of throughput

Conclusions and Future Work

- Enorm Project
  - Problems in stream processing
  - Stateful stream computation
  - Challenge of load variation
- Stateful load balancing
  - Formulate the minimum cost load balancing as bi-objective optimization problem
  - Two approximate algorithms
  - Experimental results shows the effectiveness of ELB and CLB
- Future work
  - More effective algorithms
  - Experimental comparisons

THANKS

Questions?