Multistage Adaptive Load Balancing for Big Active Data Publish Subscribe Systems

Hang Nguyen
Md Yusuf Sarwar Uddin
Nalini Venkatasubramanian
University of California, Irvine

DEBS 2019
13th ACM International Conference on Distributed And Event-based Systems, Darmstadtium, 24th-28th, 2019

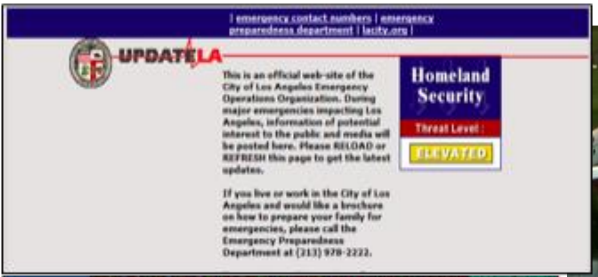
Community Scale Alerting and Notification

- Reach a large population of end users – City/Community **SCALE**
- Deliver notifications in an reliable and timely manner – **FAST**
- Produce enriched, individualized, actionable **RICH** notifications
- Support Retrospective and on-the-fly **Analytics** on **BIG** data

Pipe Burst, LA, 2014

Flint Water Crisis, Michigan, 2014

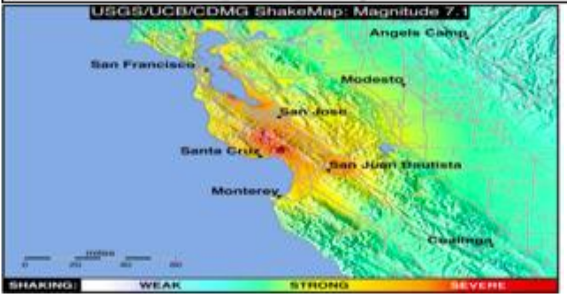
Water Contamination


| emergency contact numbers | emergency preparedness department | lacity.org |

UPDATE LA

This is an official web-site of the City of Los Angeles Emergency Operations Organization. During major emergencies impacting Los Angeles, information of potential interest to the public and media will be posted here. Please RELOAD or REFRESH this page to get the latest updates.

If you live or work in the City of Los Angeles and would like a brochure on how to prepare your family for emergencies, please call the Emergency Preparedness Department at (213) 978-2222.



Tornado
OK, May 19

Hurricane
Harvey,
Houston 2017

Tornado
Kansas, May 19

Next Generation Notification Systems

Big Active Data (BAD) – A New Pub/Sub Paradigm Based on Big Data

"PetaBytes to MegaFolks in Milliseconds"

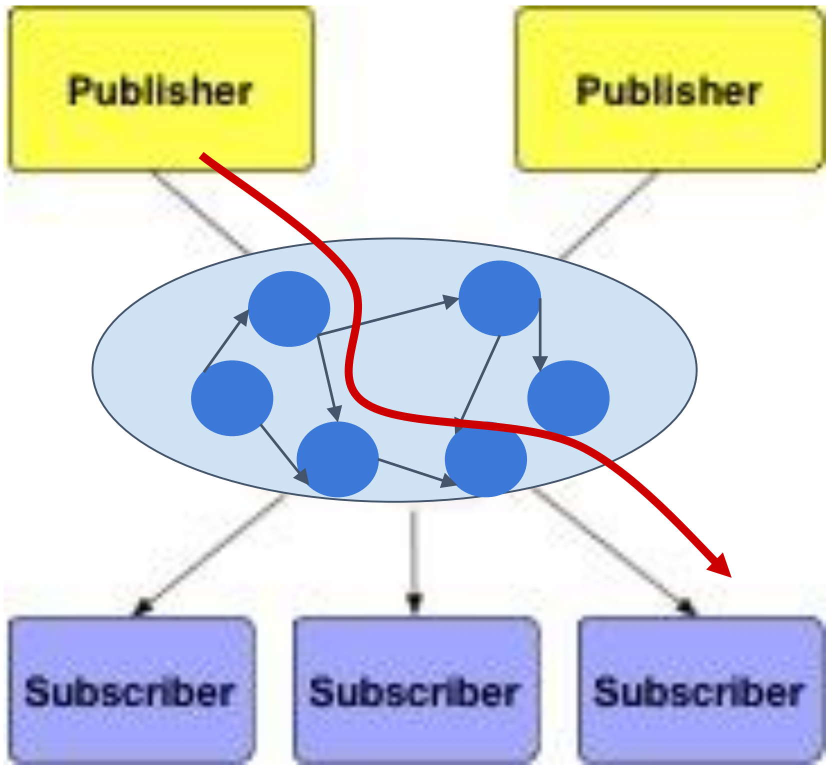
Big Active Data (BAD) = **Distributed Publish Subscribe System (Pub/Sub)** + **Big Data Management System (BDMS)**

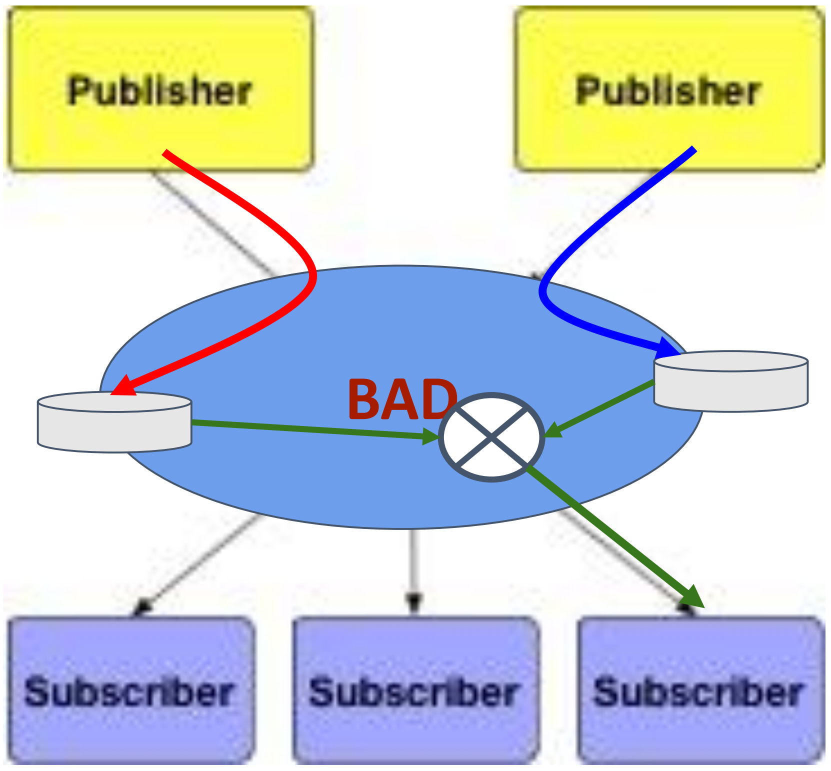
Scalable data processing backend with scalable distributed data delivery plane

The BAD Project

(w/ Mike Carey, Vassilis Tsotras, Vagelis Hristidis & the Apache Asterix team)

- Two key properties:
  - Subscriptions can consider multiple data streams (both arriving and stored) - Data in context
  - Notifications (i.e., matched publications) can contain additional information other than the original publications - "enriched" notifications





BAD Publishers and Subscribers

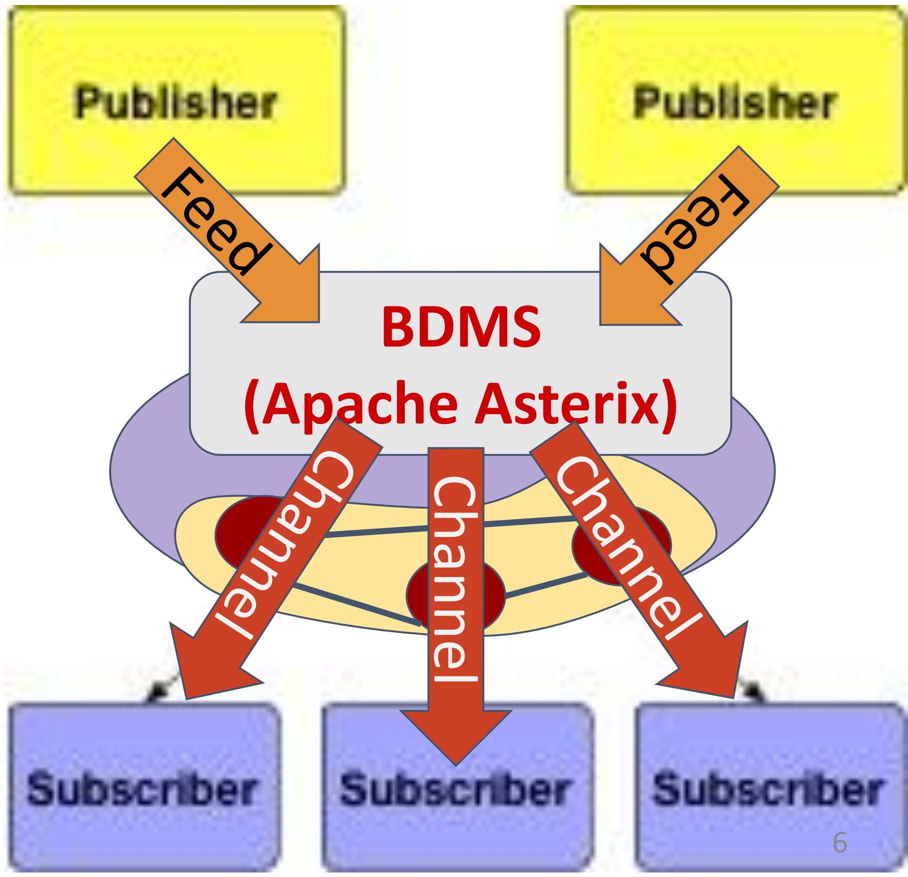
BAD Publications

- Publishers publish data through ***feeds***
- Publications persist as *datasets* in a Big Data Management System (BDMS)

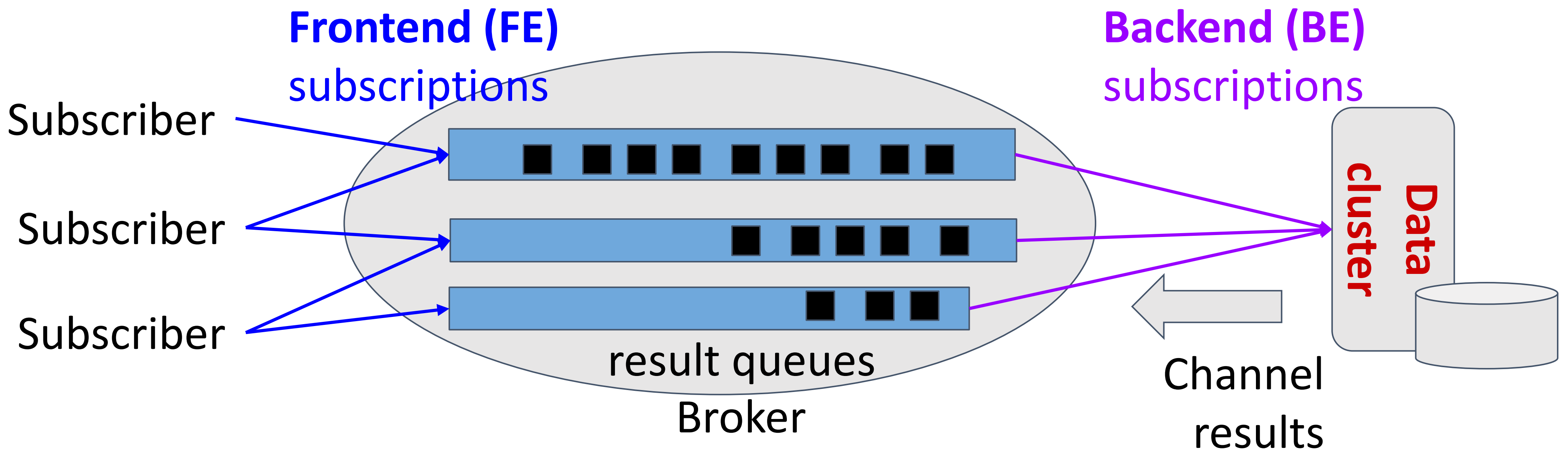
BAD Subscriptions

- Instead of topics, subscribe to *functions* ***(Channels)*** - execute publication matching on one or more datasets
- Like functions, channels take parameters and subscribers can pass different values to them to express subscriptions

  emergencyNear(L) - channel that finds emergencies near location "L"
- Two types of channel constructs:
  - **Repetitive**: function runs periodically at a given interval
  - **Continuous**: runs asynchronously when a publication arrives



BAD: Frontend and Backend subscriptions



Subscribers subscribe to the broker via **frontend** (FE) subscriptions ;
Broker subscribes to the backend data cluster via **backend** (BE) subscriptions.

- Multiple FE subscriptions can map to a single BE subscription if they subscribe to the <u>same channel</u> with the <u>same set of parameter values</u> ("subscription aggregation").
- For each BE subscription, the broker is notified when results are populated against those subscriptions.
- Broker fetches results into a result queue.

BAD Broker Network

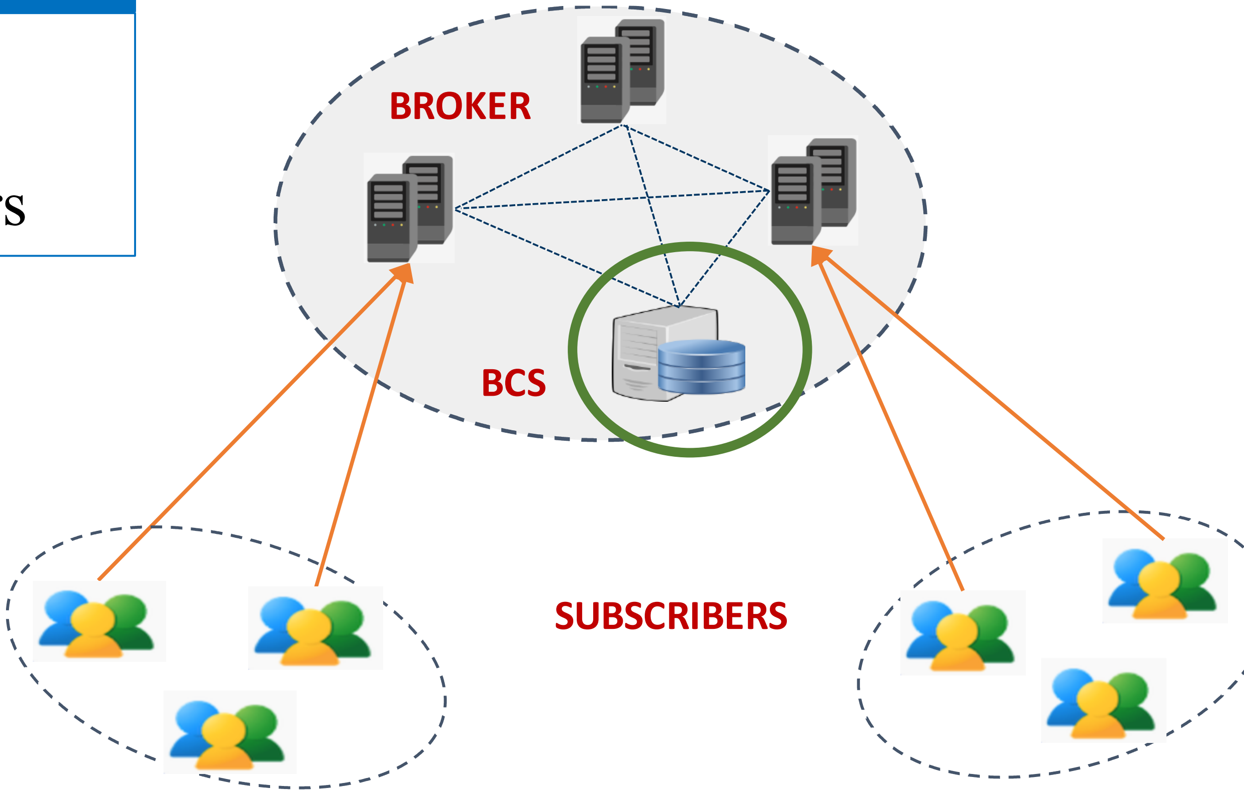
Manage and serve a large number of end users/ subscribers who are geographically distributed.

Broker Coordination Server (BCS)

- BCS monitors the broker network
- BCS faciliates mapping of subscribers to brokers

BAD Broker Issues

- Caching Notifications -- (***ICDCS 2018***)
- Load balancing distributed brokers -- (***THIS PAPER***)
- State Management, Membership management (broker add, remove)
- Failure detection and recovery

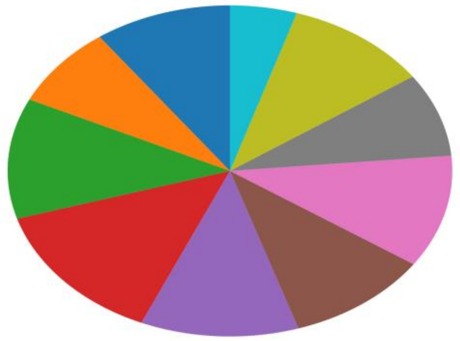
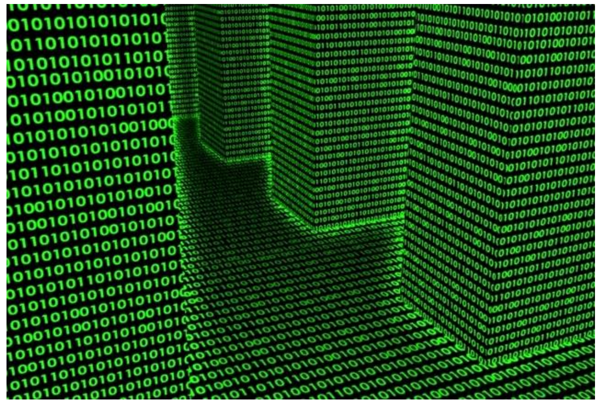


BAD Broker Network

The BAD Load Balancing Problem

- How to distribute subscribers across brokers under dynamicity to attain an uniform load distribution among brokers?
  - Dynamicity: of publications, subscribers, subscriptions and enriched notifications

Non-uniform subscriber distribution (geographically) + Unpredictable subscriber usage pattern + Unprecedented publication/data volume + Dynamic subscriber population

GOAL: Distribute near equal 'load' across all brokers in the system

-- better QoS/latencies, lower failure impact to services, less overhead, better system health

Load Balancing – an old problem

LB in Distributed Systems

- Request balancing in distributed network web caches, request balancing in crowd-sourced CDNs, virtual machine assignment in cloud computing, sensor clustering in WSN, request migration and object replication in MM servers
- Common approaches: object/task distribution strategies: data replication, hash space adjustment, request redirection, migration...

LB in Pub/Sub

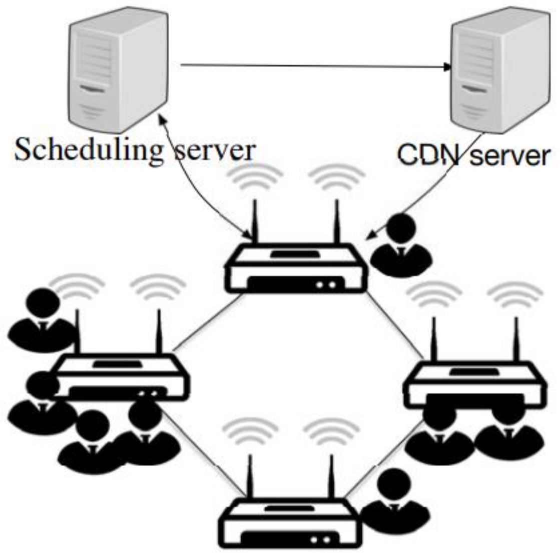
- In content-based and topic-based Pub/Sub with different broker network architectures: DHT-based, tree based, cluster-based, community based.
- LB approaches: divide overhead of publications routing, subscription matching and management.
- Techniques: hashing, clustering, publication space partition, subscriptions partition, replication...

Our Problem

- No publication routing through intermediate brokers.
- Subscription storage and matching in the data backend
- Broker workload primarily involves communication with subscribers and the data cluster

Related Works

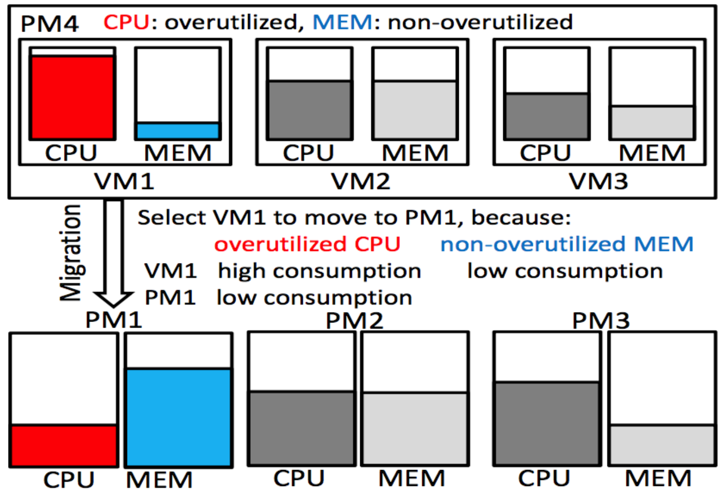
LB in Distributed Systems



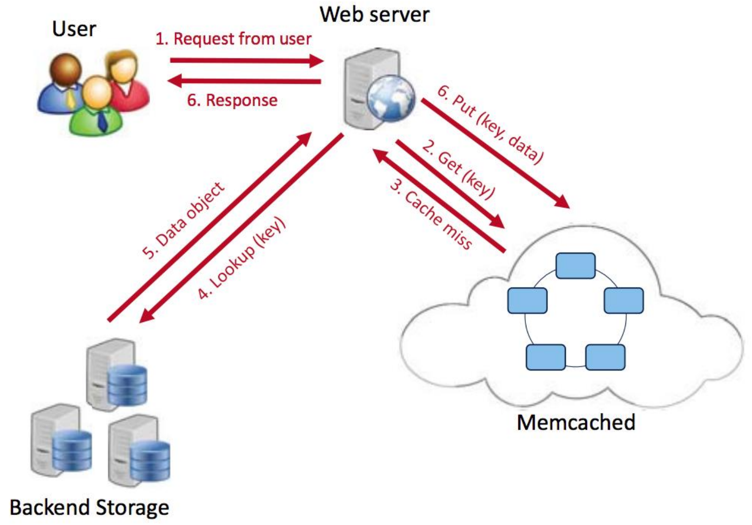
Crowdsourced CDN

- Request balancing in crowd-sourced CDNs [Ming Ma et at. ICDCS 2017]: **request redirection from overloaded hotspots to under-loaded hotspots as a min cost max flow problem**

- Cache request balancing with distributed networked caches system [S Huq et at. ICDCS 2017]: **object replication, hash space adjustment**



VM Migration



Key-value Networked Cache Systems

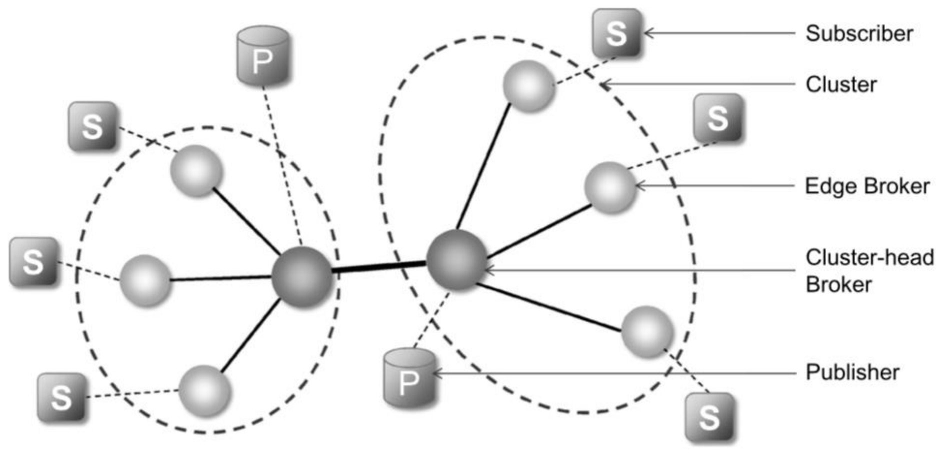
- LB in cloud computing [H Shen, IEEE Trans. Cloud Computing 2017]: **migrate VMs (min # migrations, VMs communication vs PM, VM performance degradation)**

Related Works

LB in Publish Subscribe Systems

LB in **content based** Pub/Sub like PADRES [A. Cheung et at. TOCS 2010]:

- Clustering brokers into hierarchical architecture by network proximity
- Load balancing on 3 broker performance metrics: *input utilization ratio (CPU utilization), matching delay per message* and *output utilization ratio*
- Determine overloaded brokers and load accepting brokers
- Estimate subscription's load contribution in the form of additional *input publication rate, matching delay and output publication rate*
- Calculate offloading subscribers from overloaded brokers to under-loaded brokers



Content based Pub/Sub

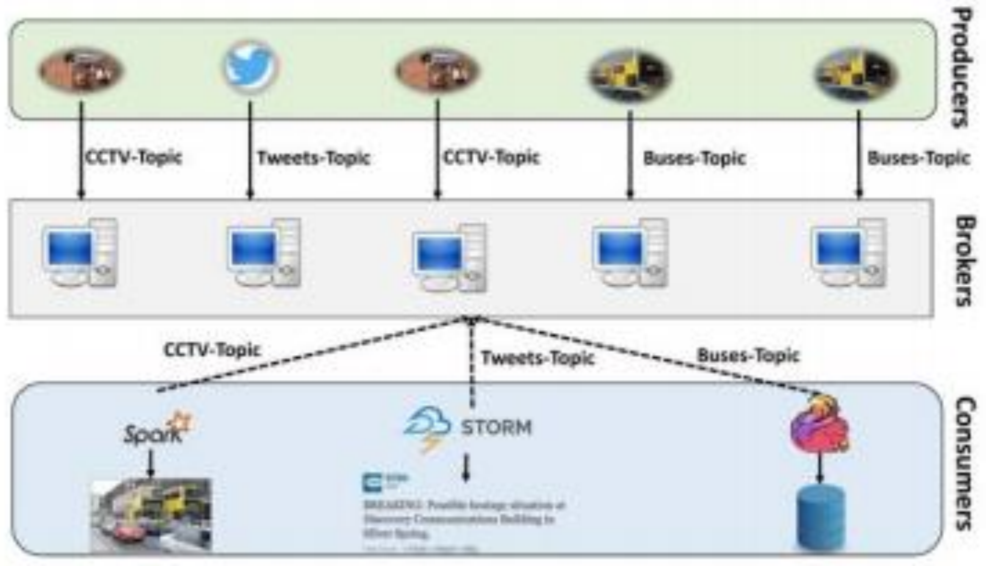
Related Works

LB in Publish Subscribe Systems

LB in **topic based** Pub/Sub like Apache Kafka [D. Dedousis et at. ICDCS 2018]:

- Messages from a topic are assigned to partitions using a consistent-hashing mechanism and partitions are assigned to brokers using round robin policy.
- Messages are published directly into the cluster of brokers, consumers pull messages directly from the brokers.
- Broker Load : traffic intensity = input rate (bps) / output rate (bps)

Approach: migrate partitions for load balancing



Apache Kafka based Pub/Sub

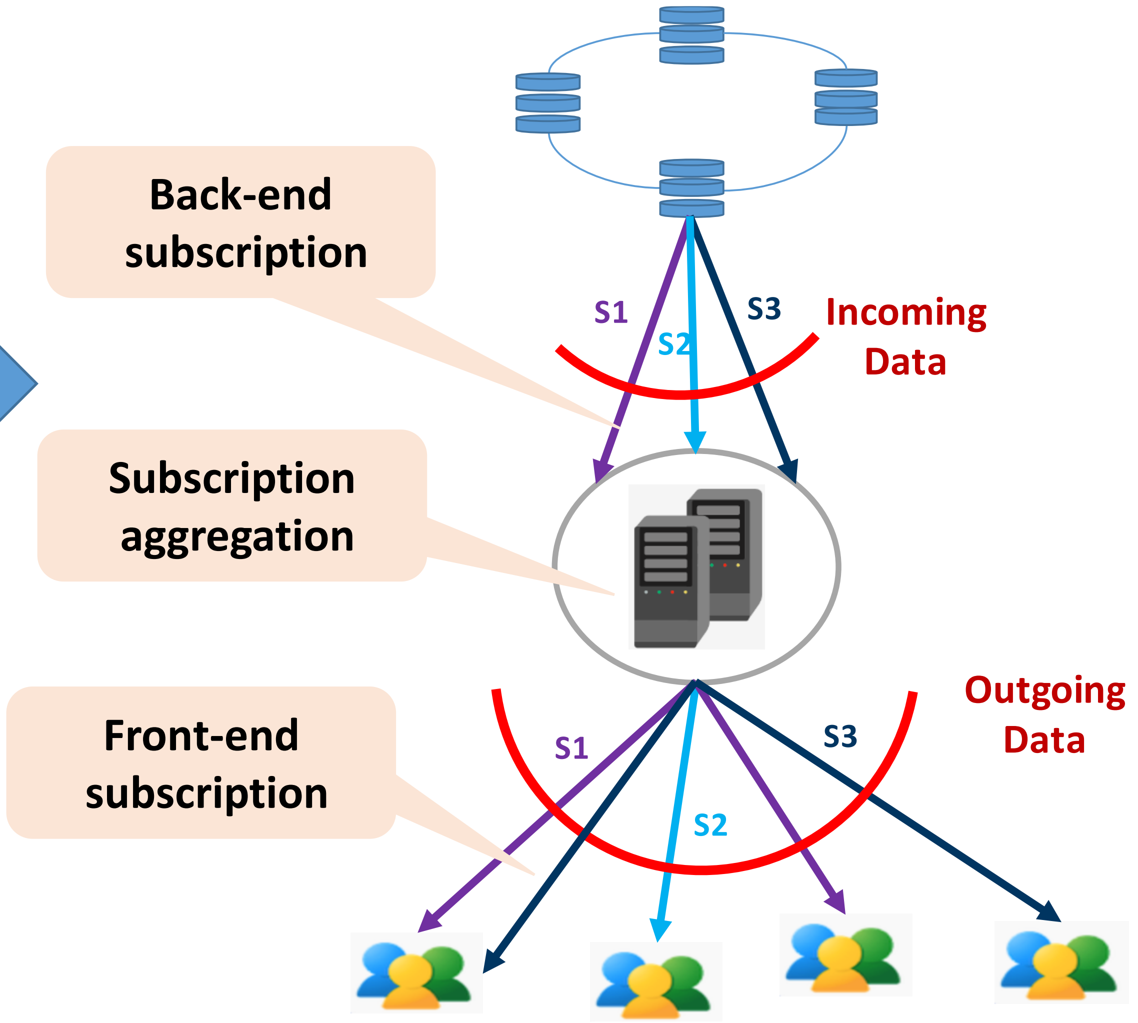
BAD Broker Load

Key tasks of a BAD broker

- Notification (result dataset) retrieval from the backend BDMS data cluster when notification on a channel arrives.
- Delivery of Notifications to each subscriber for each channel with result data sets.
- Subscribers, subscriptions management

Broker Load Definition: total amount of data that the broker need to handled per unit of time

Broker Load = **Incoming Load** + **Outgoing Load**



BAD Broker

BAD Broker Load

Notation

- m brokers $B = \{j: 1, 2, \ldots, m\}$
- n subscribers $U = \{i: 1, 2, \ldots, n\}$
- q subscriptions $S = \{k: 1, 2, \ldots, q\}$
- notification data rate: $\lambda_k \{k: 1, 2, \ldots, q\}$

Notation

- y_{ik} binary indicator if subscriber i has subscription k
- z_{jk} binary indicator if broker j has subscription k
- x_{ij} binary indicator if subscriber i attached to broker j
- n_{jk} number of FE subscriptions attached to BE subscription k at broker j

Incoming Load

$$I_j = \sum_{k=1}^{q} z_{jk} \times \lambda_k$$

BE subscription

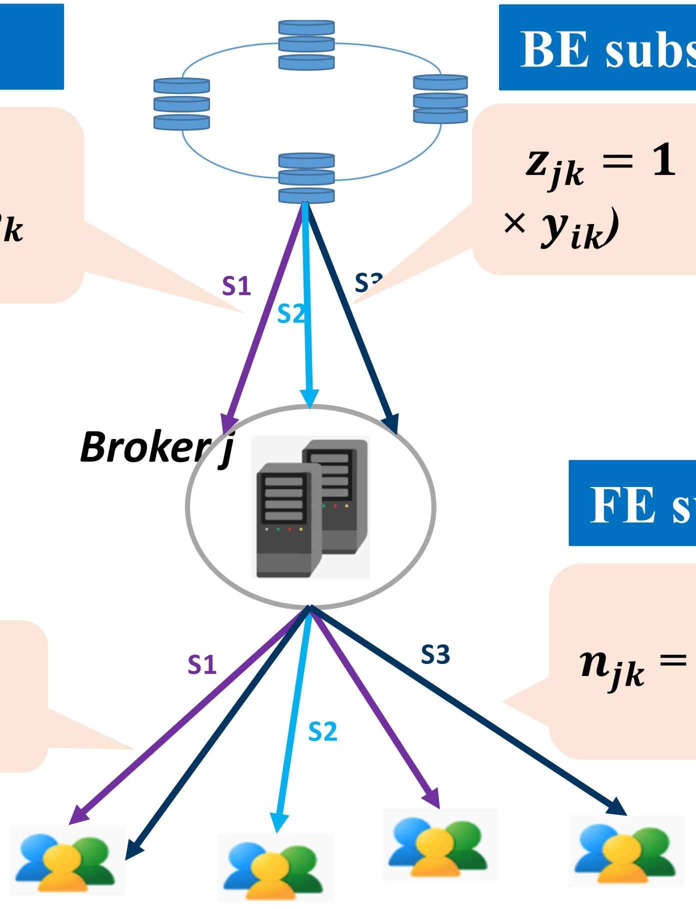
$$z_{jk} = 1 - \prod_{i=1}^{n} (1 - x_{ij} \times y_{ik})$$

Outgoing Load

$$O_j = \sum_{k=1}^{q} n_{jk} \times \lambda_k$$

FE subscription

$$n_{jk} = \sum_{i=1}^{n} x_{ij} \times y_{ik}$$



Problem Formulation

Given

- *Notification data rate:* $R = \{ \gimel_k : k = 1 \ldots q \}$
- *Subscription matrix:* $Y = \{y_{ik}: i = 1 \ldots n;\ k = 1 \ldots q\}$

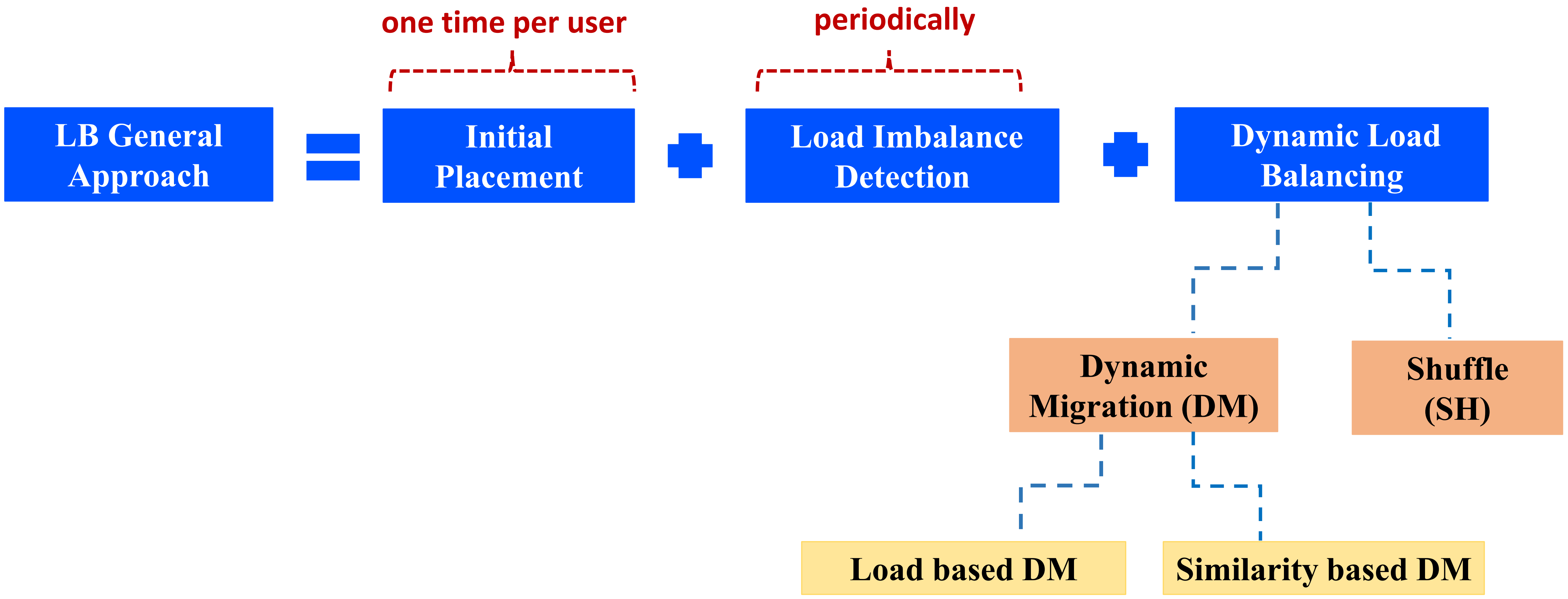
Find an subscriber assignment $X = \{x_{ij}: i = 1 \ldots n;\ j = 1 \ldots m\}$ *so as to*

$$\min_j \max \quad F_j = O_j + I_j$$

subject to: $\sum_{j=1}^{m} x_{ij} = 1,\ \forall i = 1 \ldots n$ *(each subscriber attaches to only one broker)*

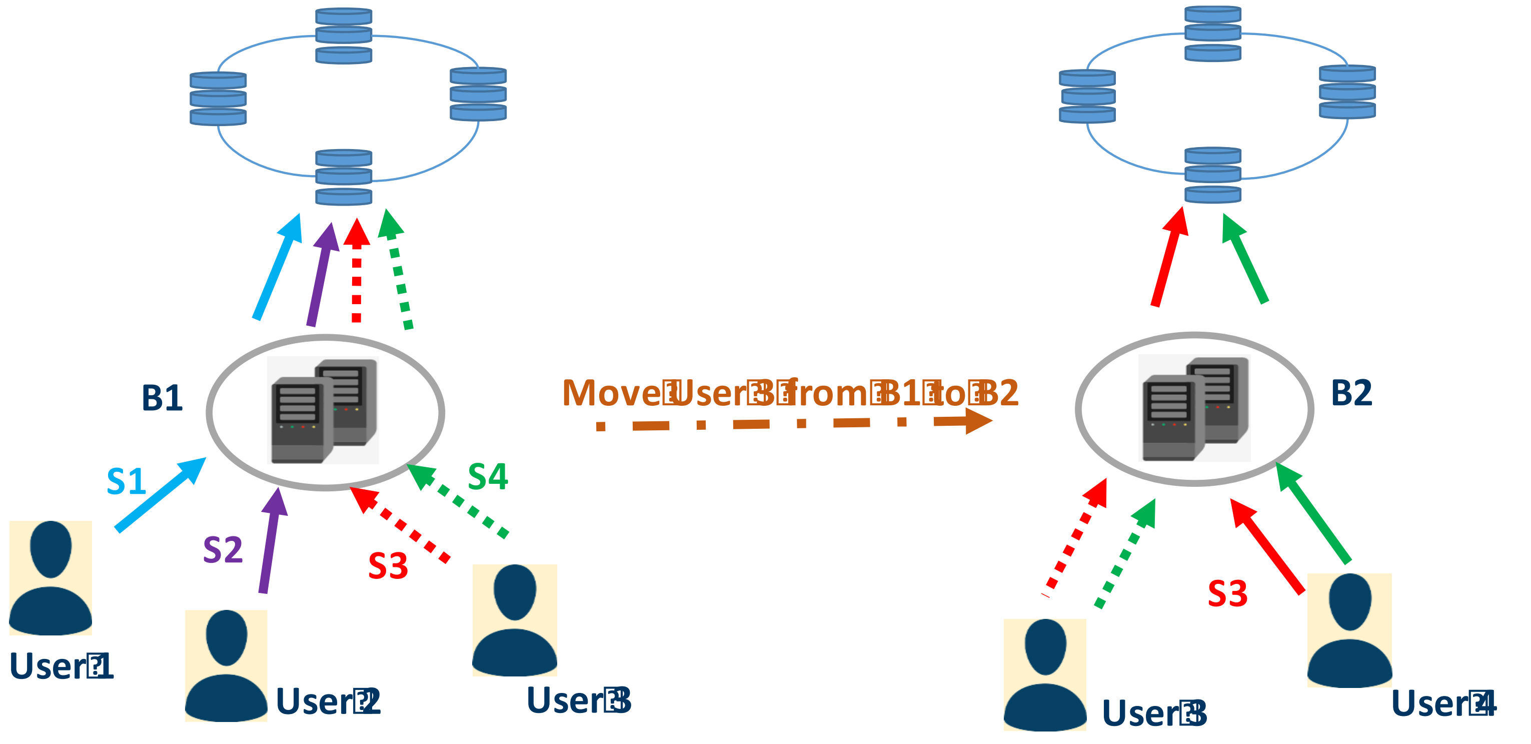
➡ *NP Hard Problem (reduction from Multi-processor Scheduling Problem)*

Our Overall Approach



Subscription similarity

- Subscription: s1, s2, s3, s4
- Load of subscription = rate of subscription's notification volume



Broker load reduced

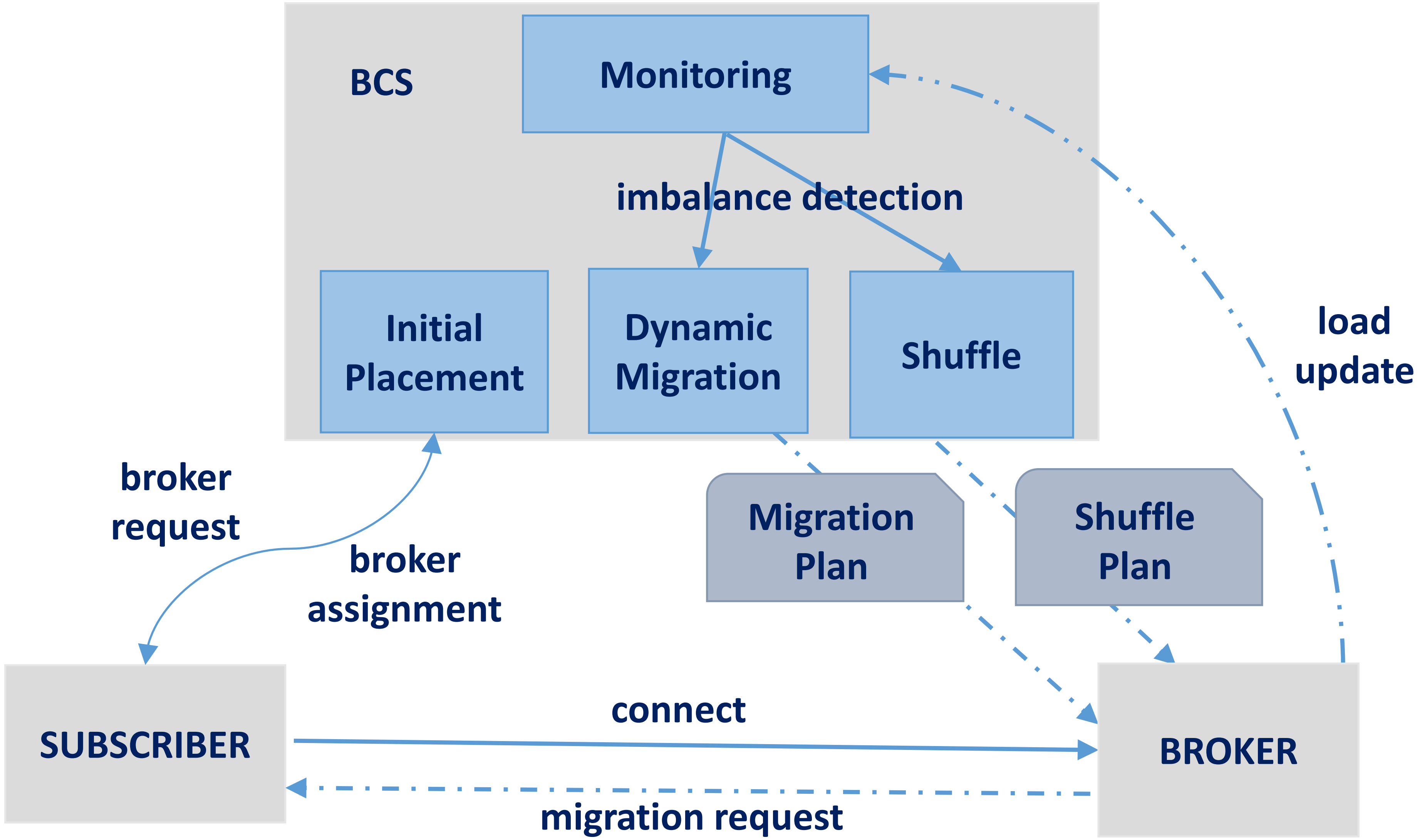
= 2 load (s3) + 2* load (s4)*

Broker load increased

= load (s3) + load (s4)

User 3 shares more subscriptions with Broker 2 than Broker 1 ➔ assign User 3 to Broker 2 instead of Broker 1 to reduce the total load of Broker 1 and Broker 2.

Multistage adaptive load balancing framework



Multistage adaptive load balancing

Initial Placement

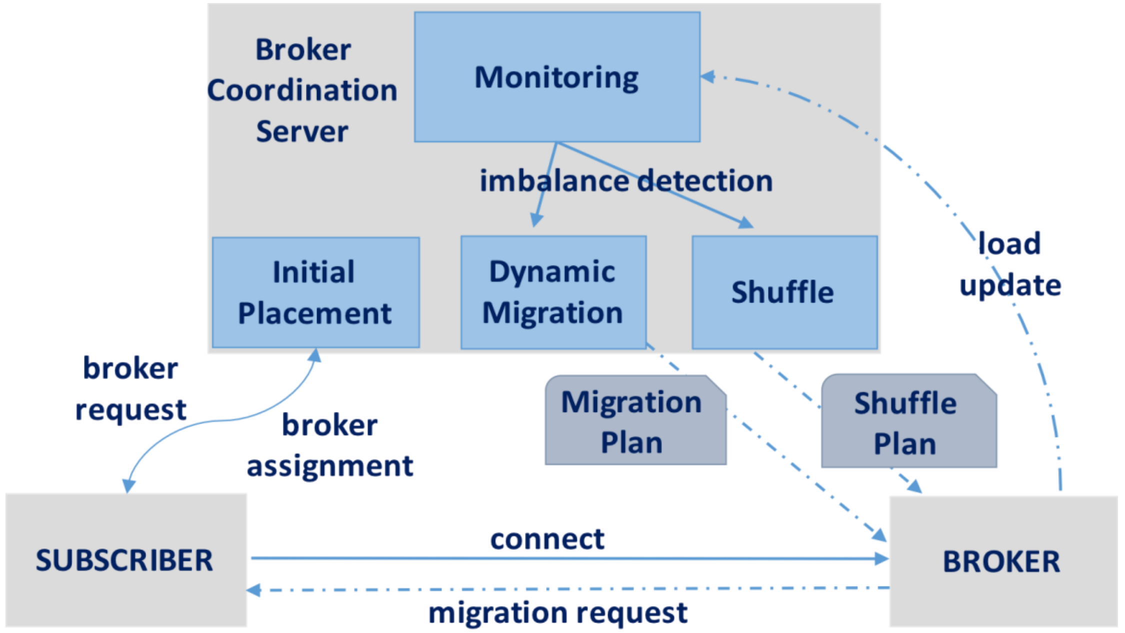
- assigns incoming subscribers to existing brokers
- subscription-agnostic, new subscribers have no subscriptions
- Policies: nearest broker, random, round robin…

Dynamic Migration

- migrates subscribers from overloaded brokers to lightly loaded brokers…
- invoked in medium load imbalance state
- Migration Policies: load based, similarity-based

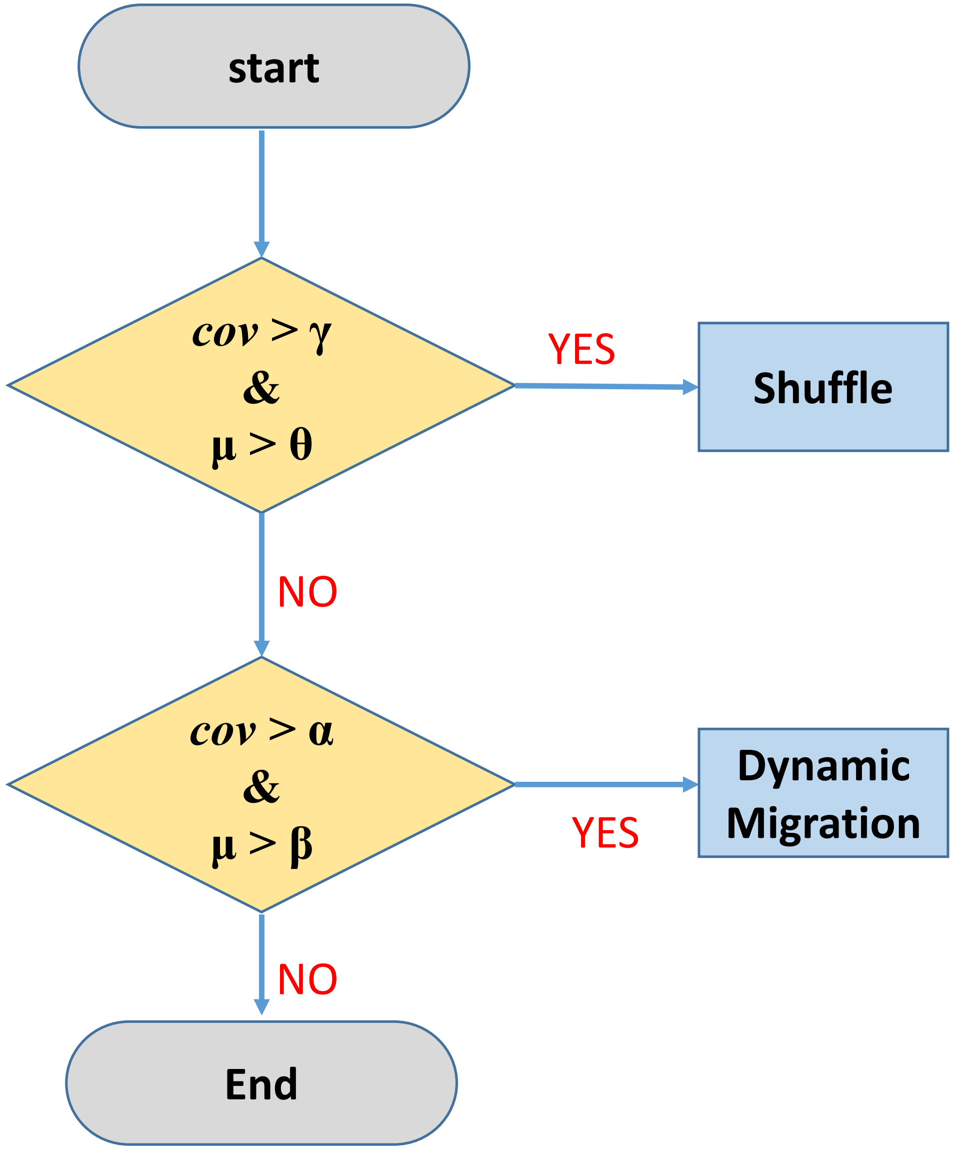
Shuffle

- redistribute the whole set of current active subscribers over all brokers…
- invoked when system is in extreme load imbalance state



Multistage Adaptive Load Balancing Framework

Algorithm Design and Implementation


start
$cov > \gamma$ & $\mu > \theta$
YES
Shuffle
NO
$cov > \alpha$ & $\mu > \beta$
YES
Dynamic Migration
NO
End

Load Imbalance Indicator

- coefficient of variation *(cov):*
- $cov = \frac{\sigma}{\mu}$; $\mu = \frac{\sum_{j=1}^{m} F_j}{m}$
- $\sigma = \sqrt{\frac{\sum_{j=1}^{m} (F_j - \mu)^2}{m}}$

Parameter Values

- $\gamma = 0.5$ and $\alpha = 0.15$

Algorithm Design: Dynamic Migration

Dynamic Migration

```
while $cov > \alpha$ & $\mu > \beta$ do
    $b \leftarrow \arg\max_{j \in B} F_j$            ▷ broker of maximum load
                                                      ▷ loads of subscribers at $b$
    $U_b = \{u_i : u_i = \sum_{k=1}^{p} y_{ik} \times \lambda_k, \forall i \in U, x_{ib} = 1\}$
    for $u_i$ in $Sorted(U_b, reverse = True)$ do
                        ▷ similarity between subscriber $i$ and broker $j$
        $simTable = \{sim_j : sim_j = \sum_{y_{ik}=1, z_{jk}=1} \lambda_k, j \in B\}$
        if $scheme$ == LDM then
            $b' \leftarrow \arg\min_j F_j$
        end if
        if $scheme$ == SDM then
            $b' \leftarrow \arg\max_{j, F_j < \mu} simTable$
        end if
        if $\underset{x_{ib'}=1}{F_{b'}} \leq \underset{x_{ib}=1}{F_b}$ then
            $M \leftarrow \{b : [i, b']\}$
            break
        end if
    end for
    Update $F_b, F_{b'}$
    $x_{ib} \leftarrow 0, x_{ib'} \leftarrow 1$
end while
```

while still need DM

choose most loaded broker

select heaviest user possible

LDM: select destination broker as the least loaded broker

SDM: select destination broker as the most similar broker to user

Algorithm Design: Shuffle

Shuffle Algorithm

```
1: $U_{load} = \{u_i : u_i = \sum_{k=1}^{p} y_{ik} \times \lambda_k, \forall i \in U\}$
2: $B_{load} = \{F_j = 0, \forall j \in B\}$
3: while $U \mathrel{!=} \emptyset$ do
4:     $i \leftarrow \arg\max_{i \in U} u_i$
5:     $b \leftarrow \arg\min_{j \in B} F_j$
6:     $b \leftarrow i$
7:     $U = U \setminus i$
8:     Update $F_b$
9:     $x_{ib} = 1$
10: end while
11: return
```

Calculate load of all subscribers

All brokers start empty

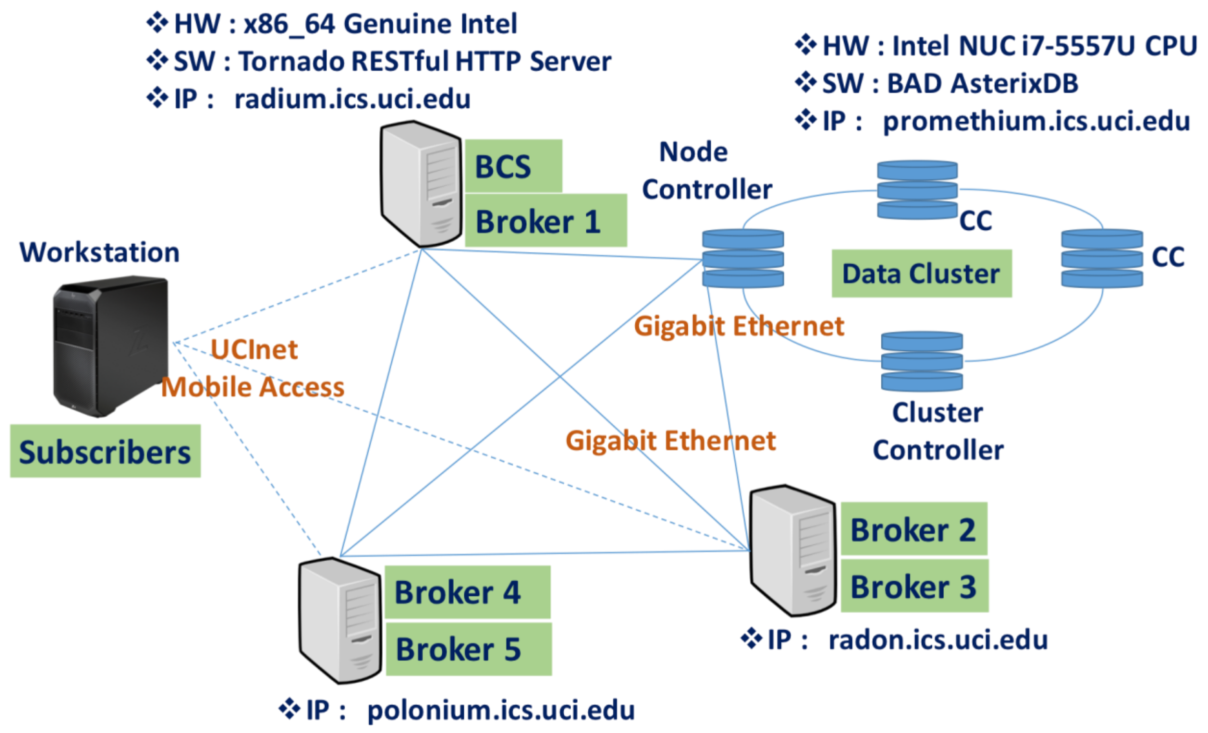
while not done

select the heaviest subscriber

Assign to the current least loaded broker

Prototype Implementation Setup

- 400 subscribers, 5 brokers, 1 BCS, 1 AsterixDB Data Cluster
- 5 brokers and 1 BCS run on 3 machines, each has i7-5557 CPU with 4 cores, 16 GB RAM, and 1TB HDD
- Data cluster runs on 4 Intel NUC nodes, each has i7-5557U CPU with 4 cores, 16 GB RAM and 1TB HDD
- Subscribers run on a single node



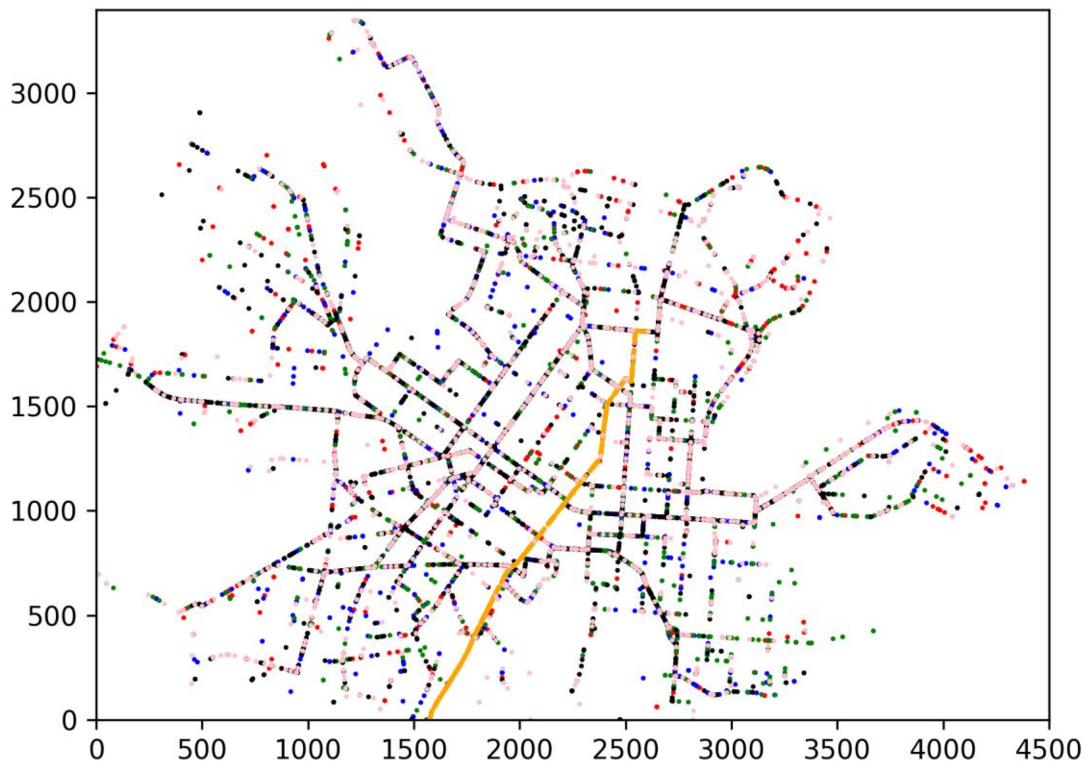
Prototype Implementation Detail

Emergency application scenario

- The ONE simulator was used to generate realistic movement of 400 subscribers, emergency reports for being continuously fed into the data cluster

- In 30 mins, about 10,000 emergency reports of eight different types at random locations are generated, each with size of 200 to 700 bytes

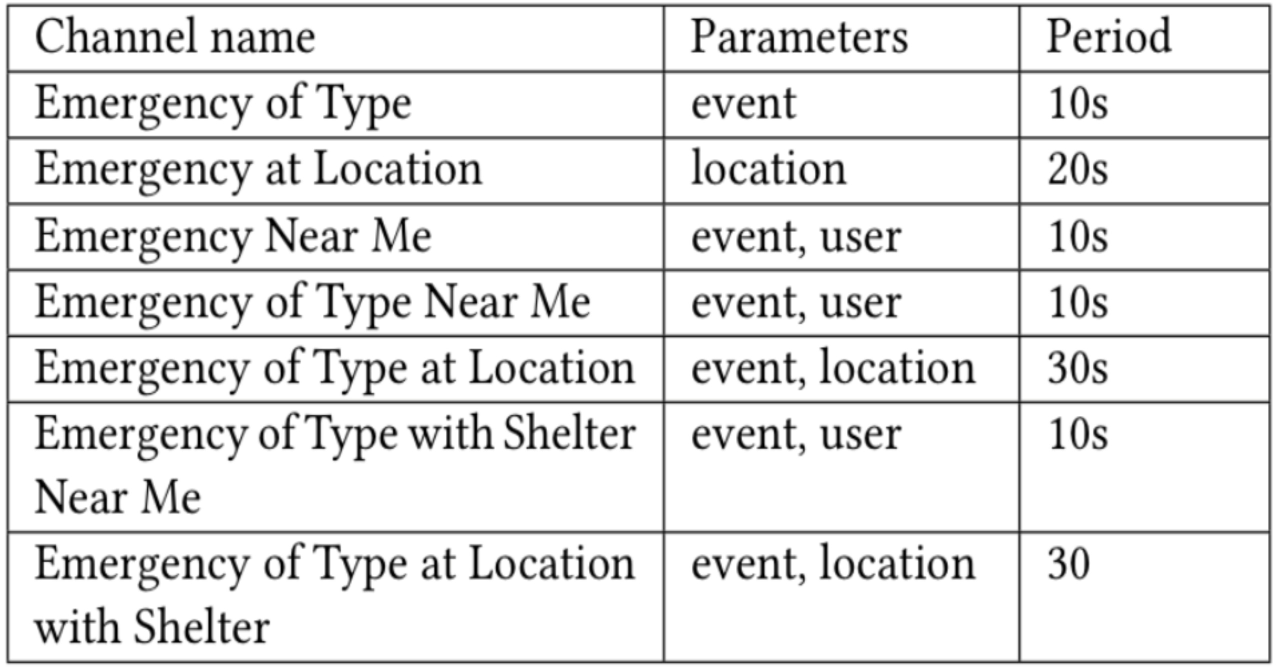
- 200 shelter locations are preloaded in the data cluster



The movement of a few subscribers

Emergency application scenario

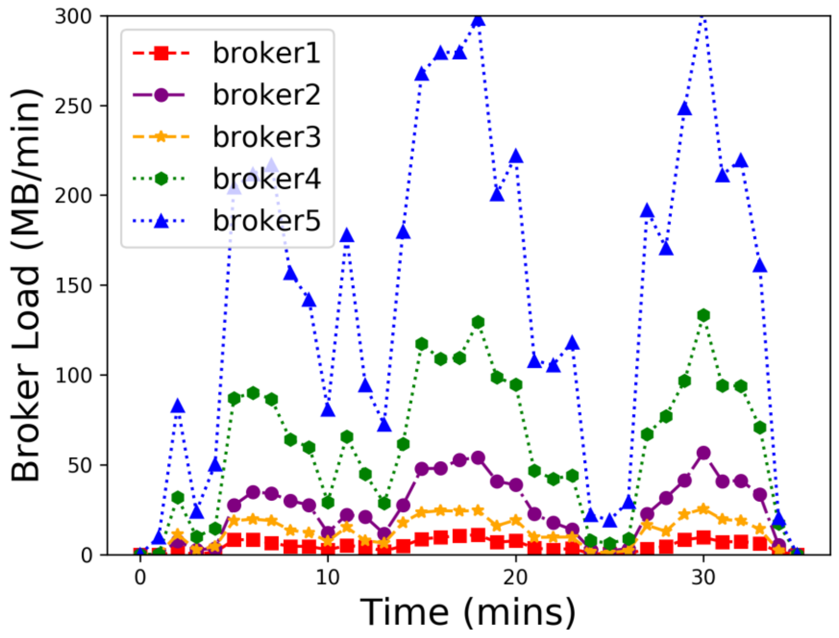
- 7 channels
- 400 subscribers: each subscribes (1, 5) channels and (1, 3) subscriptions per channel
- Subscribers start connections at random time in make all subscriptions at random point in time in the experiment run
- In total 2,200 front-end subscriptions and 610 back-end subscriptions



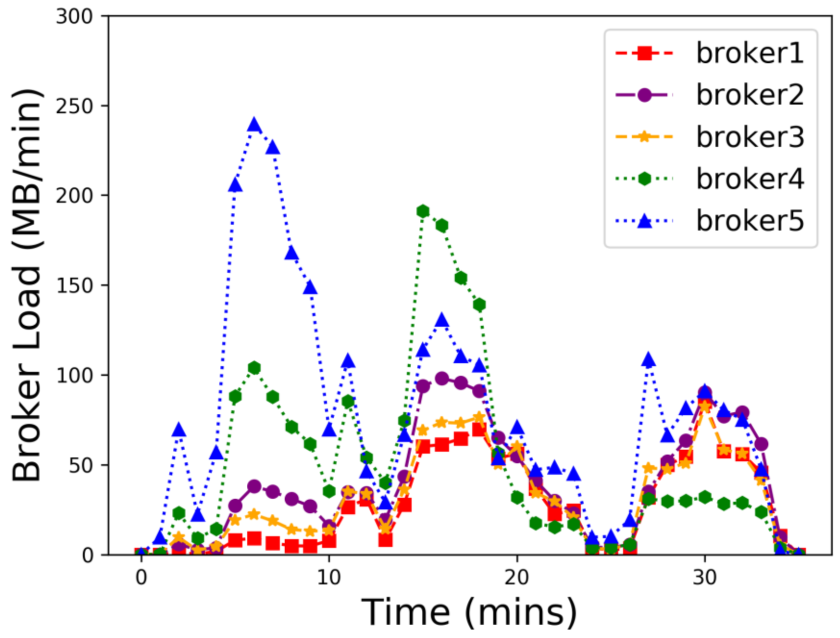
Channel name	Parameters	Period
Emergency of Type	event	10s
Emergency at Location	location	20s
Emergency Near Me	event, user	10s
Emergency of Type Near Me	event, user	10s
Emergency of Type at Location	event, location	30s
Emergency of Type with Shelter Near Me	event, user	10s
Emergency of Type at Location with Shelter	event, location	30

The seven channels

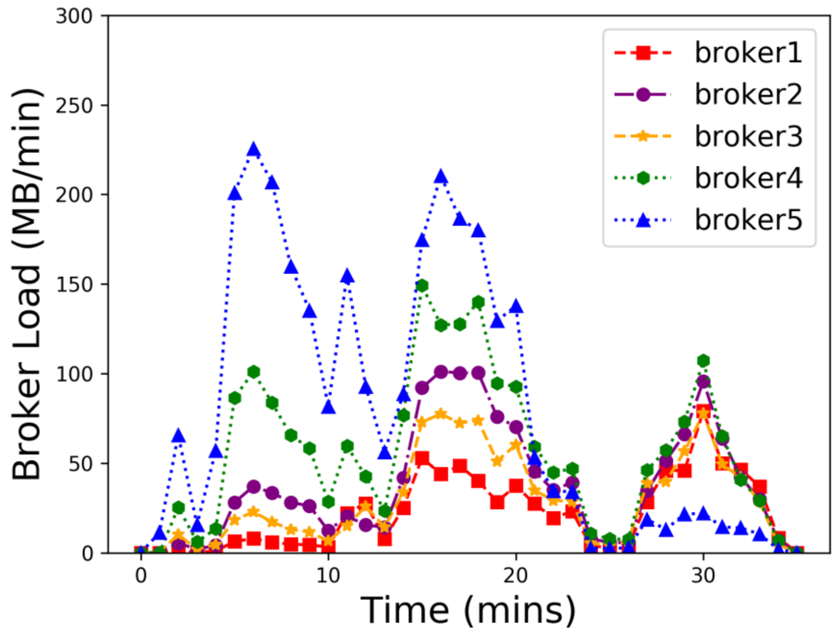
Broker load distribution: Nearest Broker placement



NR + No LB



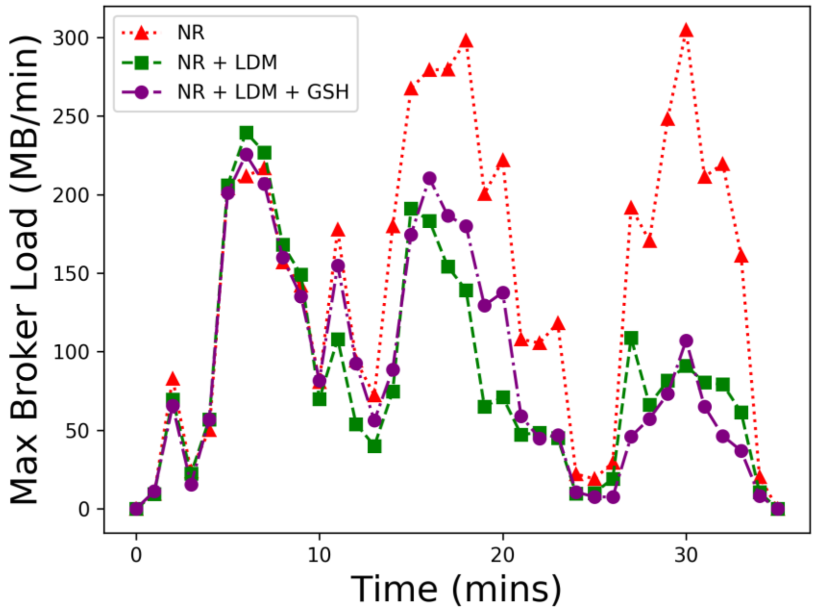
NR + LDM

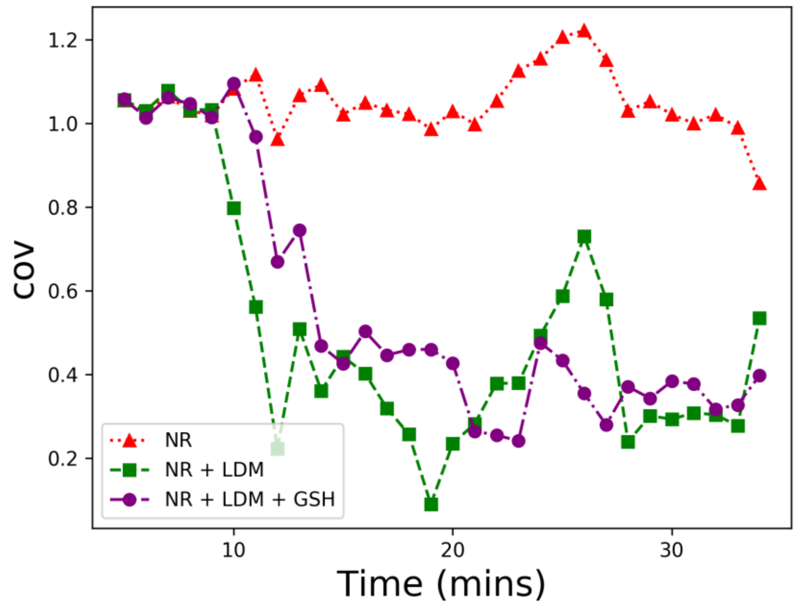


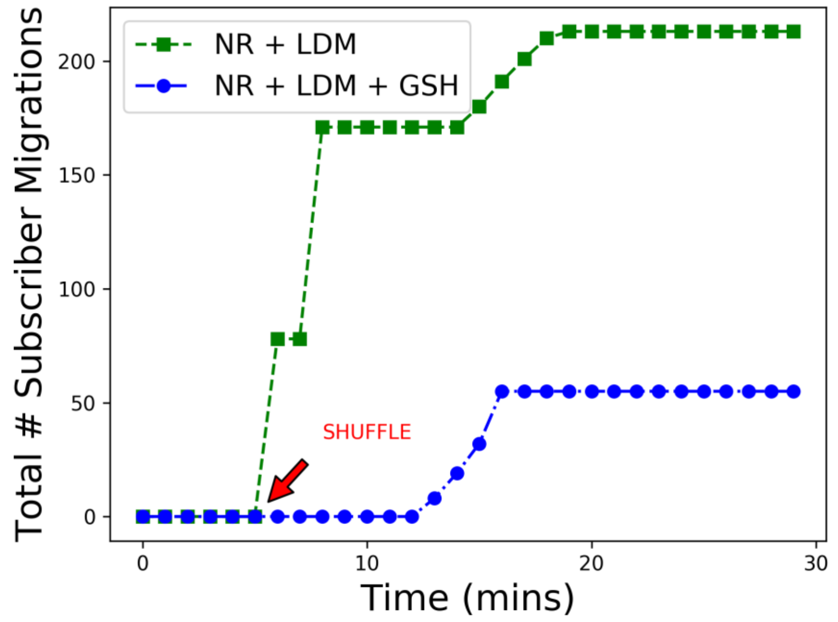
NR + LDM + GSH

NR= nearest broker; LB = load balancing ; LDM = load based DM; SDM = similarity based DM ; GSH = greedy shuffle

Performance Evaluation: Nearest Broker placement





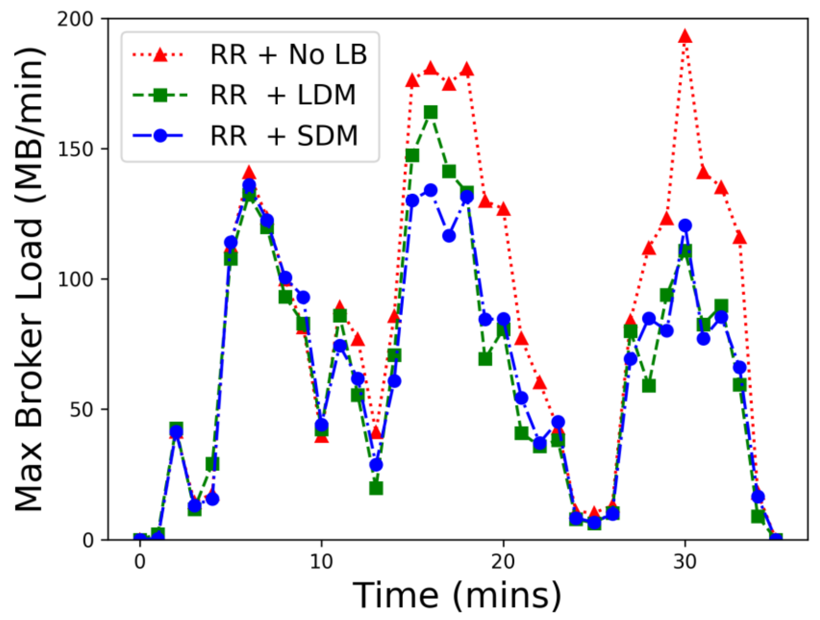


(a) max broker load **(b) cov** **(c)# migrations**

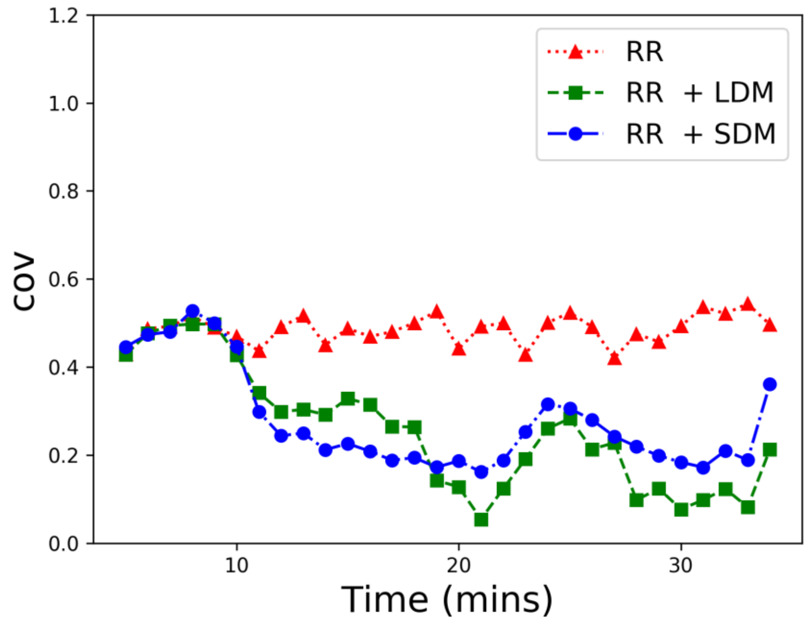
Dynamic migration (DM) and Shuffle (SH) can help reduce maximal broker load (a) , create a better balanced load distribution among brokers (b). Early shuffle (c) require less migrations later.

cov = coefficient of variation ; LB = load balancing ; LDM = load based DM; SDM = similarity based DM

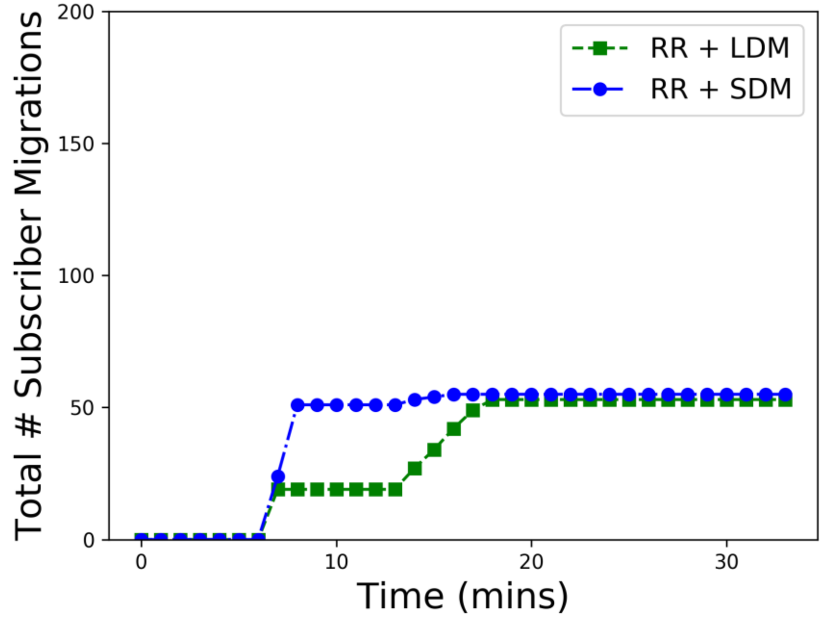
Performance Evaluation: Round Robin Placement



(a) max broker load



(b) cov



(c) # migrations

Both LDM and SDM help (a) reduce the max broker load , (b)reduce the imbalance (c) and require similar number of migrations.

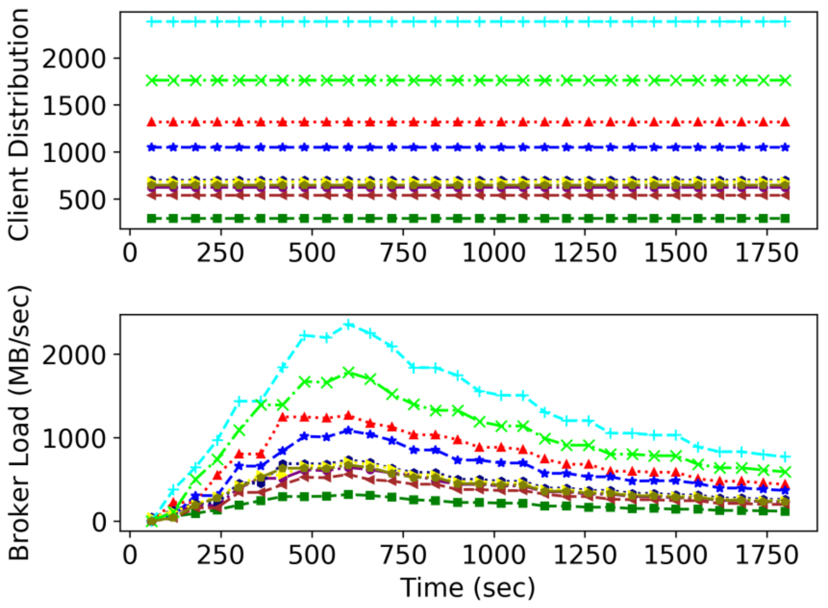
LB = load balancing ; LDM = load based DM; SDM = similarity based DM

Simulation Setup

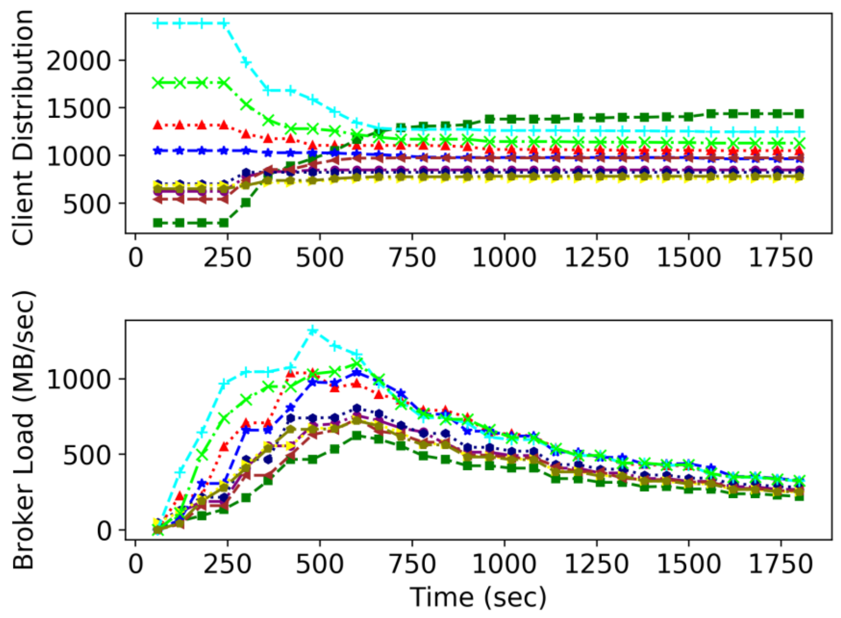
Simulation mimics the message level interactions among BAD components

- 10 brokers, 10,000 subscribers, 10 channels, 1 BCS, 1 data cluster
- 10 channels support 1,000 back-end subscriptions. These channels are different in the channel execution periods and the average size of the generated notifications
- Each subscriber creates (10, 30) subscriptions, totally creates 200,000 front-end subscriptions

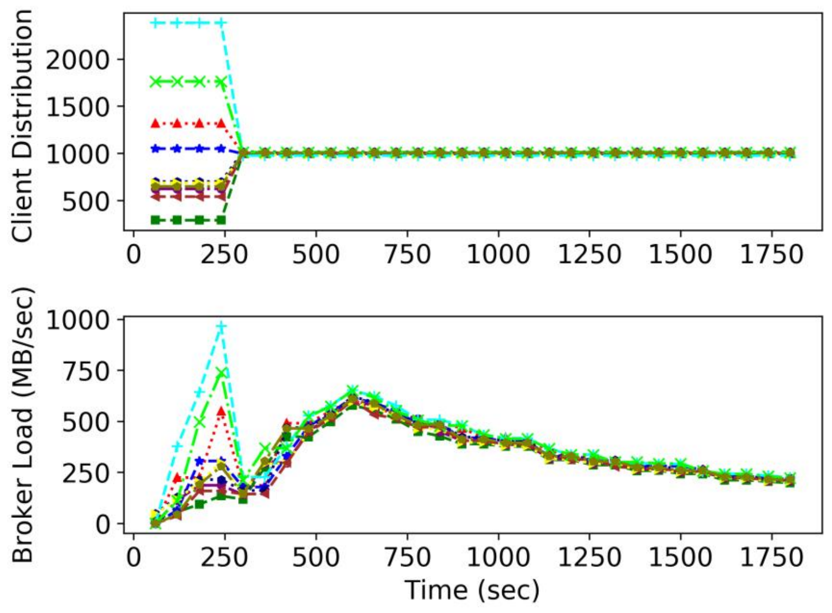
Performance comparison - Nearest Broker placement



(a) No LB



(b) LDM (α =.15, β = 300)

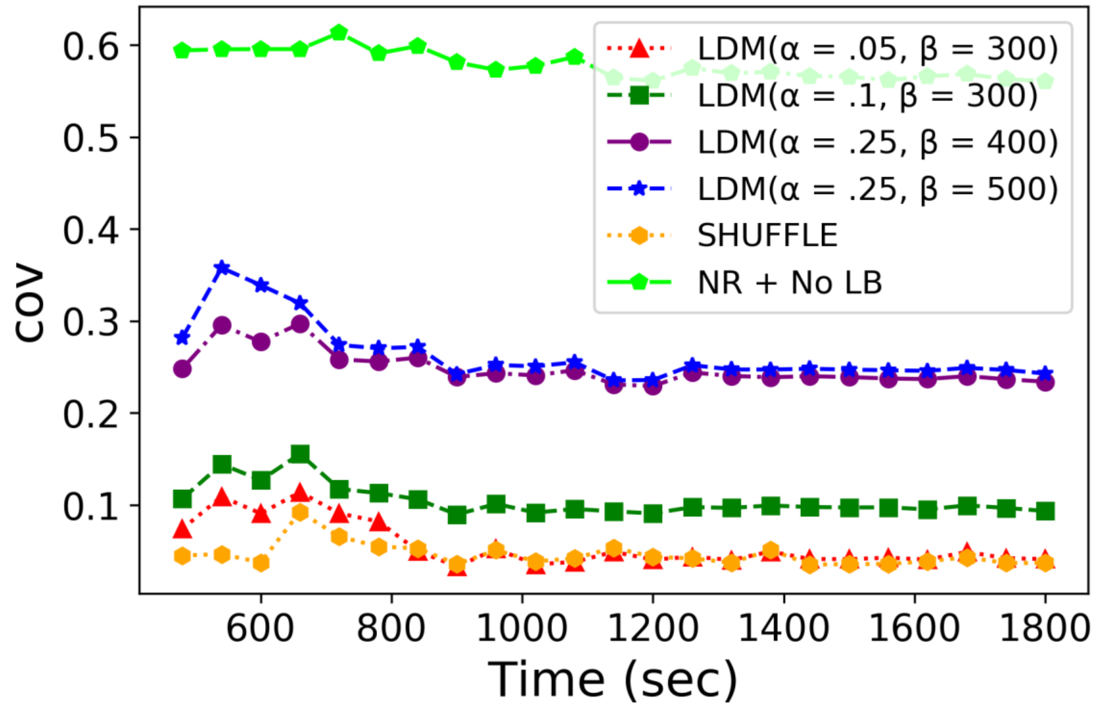


(c) GSH

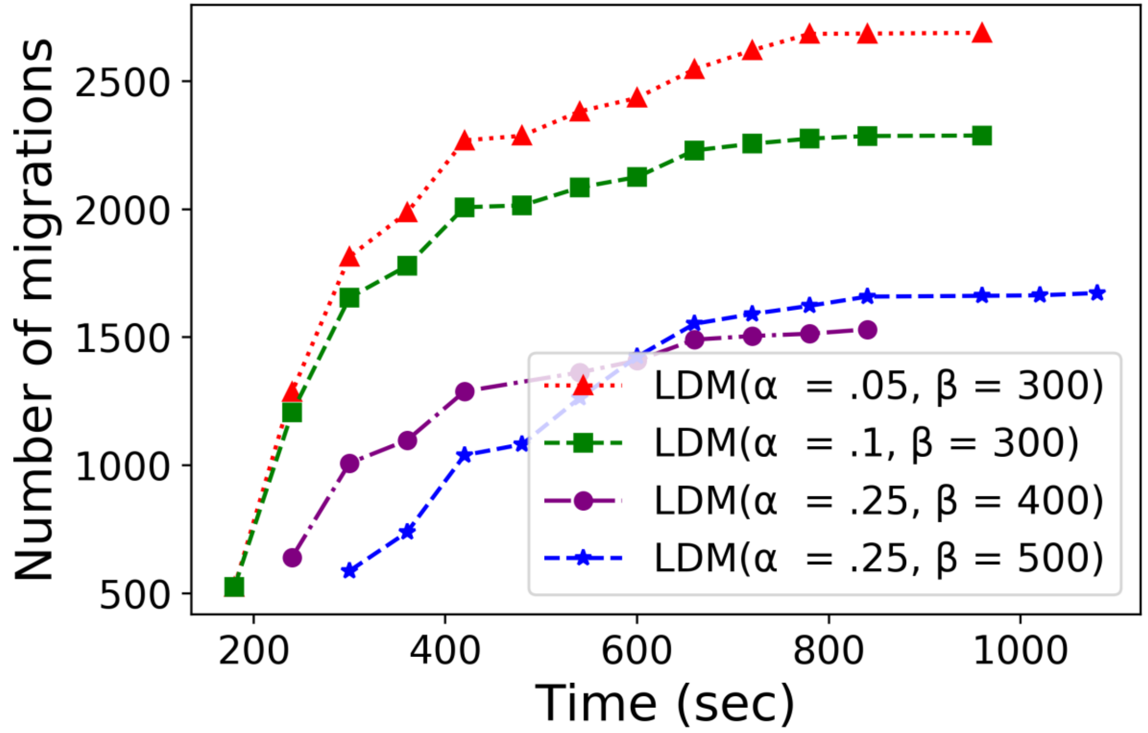
With dynamic migration (b) and shuffle (c), maximal broker load is reduced by roughly half. Shuffle (c) achieves a better balanced load distribution compared to the dynamic migration (b) and requires no further migration.

LB = load balancing ; LDM = load based DM; SDM = similarity based DM ; GSH = greedy shuffle

Effect of α – tuning parameter value



cov



migrations

The smaller the alpha value, the higher number of migrations required, but load balancing .
When alpha is small enough, LDM can get as good as the shuffle but takes longer time to converge.

Conclusions and Future Work

BAD: A potential architecture to integrate big data processing with event-driven notification systems

- In this paper: load balancing approach for BAD.
  - Initial Subscriber/Broker assignment
  - Dynamic migration. (exploit. subscriber/broker similarity)
  - Shuffle

- Future Directions.
  - Clustering subscribers into groups based on their shared subscriptions and enabling group migration
  - Larger experiments using public cloud infrastructure
  - Exploiting big data “processing” – e.g. simulations in addition to dataflows in the loop

Thank You

Q & A