SCS 4001 - INDIVIDUAL PROJECT[1]

UNIVERSITY OF COLOMBO SCHOOL OF COMPUTING

Cloud based publish/subscribe model for Top-k matching over continuous data streams

Author:

Y.S. Horawalavithana

10002103

Supervisor:

Dr. D.N. Ranasinghe

Submitted in partial fulfillment
of the requirements of the B.Sc
in Computer Science 4th Year
Individual Project (SCS4001)

April 17, 2015

[1]IEEE REF LATEX Mendeley

Declaration

I, Y.S.Horawalavithana, certify that this thesis does not incorporate, without acknowledgment, any material previously submitted for a degree or diploma in any University or higher educational institution in Sri Lanka or abroad and to the best of my knowledge and belief, it does not contain any material previously published or written by another person except where due reference is made in the text.

Signature	:	
Name	:	Y.S.Horawalavithana
Date	:	April 17, 2015

I, Dr. D.N. Ranasinghe, certify that I supervised this thesis entitled "Cloud based publish/subscribe model for Top-k matching over continuous data streams" is conducted by Y.S.Horawalavithar
I recommend this thesis to the University of Colombo School of Computing in partial fulfillment of the requirement of the degree of Bachelor of Science (Computer Science).

Signature	:	
Name	:	Dr. D.N.Ranasinghe
Date	:	April 17, 2015

Abstract

Publish/subscribe systems are widely recognized in processing continuous queries over data streams and are augmented by algorithms coming from the field of data stream processing. Existing functions which are capable of matching publications & subscriptions in state-of-the-art publish/subscribe systems are depended on a stateless function which provides only a Boolean decision on whether a given publication is to be notified to relevant subscriber or not. But in such systems, the large quantity of received publications may be considered as a sort of spam, while a system that delivers too few publications might be recognized as non-working.

In our study, we propose an advanced publish/subscribe matching model to control the unpredictable number of delivered publications over a continuous data-stream, where at a given time t our model limits the number of delivered publications by parameter k, while ranks them within a size w of sliding window. A general scoring mechanism is exploited where publications get scored against personalized user subscription spaces based on the relevancy. We adopt an inverted-list data structure to index the subscription space to enhance the efficiency of matching process. Also we focus on the problem of selecting the k-most diverse items from a relevant result set, in a dynamic setting where Top-k results change over time. We formalize the above problem of continuous k-diversity as *MAXDIVREL* which maps to the *independent dominating set* problem in graph theory, which is NP-hard. An incremental indexing mechanism is proposed for handling streaming publications that is based on Locality Sensitive Hashing (LSH) to diversify Top-k results continuously. Our prototype model is implemented in a cloud based message broker system and we have designed it to scale on top of Amazon Web Services (AWS): a scalable cloud-service provider.

We explore the natural behavior of ranked publications mathematically modeled by *zipf* property. Based on the experiments across many diversity methods, *MAXDIVREL* exhibits the strongest natural behavior. Also the proposed LSH indexing mechanism produces MAXDIVREL diverse set of results at 70% accuracy by comparing with naive optimal method. Finally, we report the experimental results concerning the performance & efficiency of the proposed indexing mechanisms on a variety of synthetic datasets.

Keywords. Top-k Publish/Subscribe, Cloud Middleware, Data Stream Processing Systems (DSPS), Result Diversification, Indexing

Acknowledgments

I would like to give special thanks to...

Dr. D.N. Ranasinghe, Senior Lecturer at UCSC
for being an always helpful & interested supervisor who offered invaluable advices throughout the research while giving feedbacks on incremental revisions

Dr. A.R. Weerasinghe, Senior Lecturer at UCSC
for giving me guidance to build my research skills

Asst. Prof. Francisco Hernandez, at Umea University
for being the professor who made me more interested in Distributed Systems

Primal Wijesekara, PHD Fellow at UBC, Canada
& *Charitha Madushanka*, at WASN research lab, Sri Lanka
for discussing various project matters along the way

My Family
for supporting me the many years of my studies

...As well as everyone who spent time to listen to my question & helped me to complete this thesis

Contents

Declaration i
Abstract ii
Acknowledgments iii
List of Figures ix
List of Tables xi
List of Algorithms xii
List of Symbols xiii
Acronyms xiv

1 Introduction **1**
1.1 Preamble 1
1.2 Background to the study 2
1.3 Motivation 4
1.4 Goals of the thesis 6
1.5 Scope of the thesis 7
1.6 Thesis outline 8

2 Related work **9**
2.1 E-commerce ranking methods 9
2.2 General Top-k Publish/subscribe 9
2.3 All in one: Diversity 11
2.4 Subscription Indexing 12
2.5 Publication Indexing 13

3 Preliminaries of publish/subscribe models **15**
3.1 Boolean Publish/Subscribe 15
3.1.1 Structural View 16
3.1.1.1 Subscription 16

3.1.1.2 Publication . . . 17
3.1.2 Behavioral view . . . 17
3.2 Top-k Publish/Subscribe . . . 18
3.2.1 Structural view . . . 18
3.2.1.1 Personalized Subscription Space . . . 18
3.2.1.2 Publication Stream . . . 19
3.2.1.3 Subscription parameters . . . 20
3.2.2 Behavioral view . . . 21
3.3 Discussion . . . 21
3.3.1 Relationship . . . 21
3.3.2 Comparison . . . 22

4 Top-k Publish/Subscribe Model **24**
4.1 Design & Architecture . . . 24
4.2 Scoring Algorithm . . . 25
4.2.1 Relevancy: Query personalization . . . 26
4.2.1.1 Relating Attributes . . . 28
4.2.1.2 Relevancy score . . . 31
4.2.2 Events novelty: Freshness . . . 31
4.2.2.1 Fresh Relevancy score . . . 33
4.2.3 Events Diversity . . . 34
4.2.3.1 k-diversity problem . . . 35
4.2.3.2 Combining relevancy with diversity . . . 37
4.2.3.3 Beyond Diversity & Relevance . . . 37
4.2.3.4 MAXDIVREL NP-Hardness . . . 39
4.2.3.5 MAXDIVREL continuous k-diversity problem . . . 42
4.3 Timing policies . . . 44
4.4 Events Delivery . . . 47

5 Indexing **49**
5.1 Subscription Indexing . . . 49
5.1.1 OpIndex . . . 49
5.1.2 Modified OpIndex . . . 51
5.1.2.1 Index construction . . . 52

5.1.2.2 Matching . . . 53
5.2 Publication Indexing . . . 54
5.2.1 Near Neighbor query . . . 56
5.2.2 Locality Sensitive Hashing (LSH) . . . 57
5.2.3 Publications as categorical data . . . 58
5.2.4 LSH Family for Jaccard Distance . . . 60
5.2.4.1 MinHashing . . . 60
5.2.5 LSH in MAXDIVREL . . . 62
5.2.5.1 LSH for signature matrix of publications . . . 62
5.2.5.2 Compute Top-k publications . . . 63
5.2.6 Incremental Top-k computation . . . 65
5.2.7 Analysis . . . 67
5.2.7.1 Probability Analysis . . . 67
5.2.7.2 Time complexity . . . 69
5.2.7.3 Space complexity . . . 69

6 Implementation - Cloud Middleware 70
6.1 Amazon Web Services (AWS) . . . 70
6.2 Main Modules . . . 71
6.2.1 Publication Stream . . . 72
6.2.2 Indexed personalized subscription spaces . . . 73
6.2.3 Sliding window Top-k matching with indexed publications . . . 74
6.2.4 Event delivery . . . 76
6.2.5 Persistent notification service . . . 77
6.3 Scalability & Elasticity . . . 77

7 Evaluation 78
7.1 Datasets . . . 78
7.2 Evaluation methodology . . . 82
7.3 Subscriber Effectiveness . . . 83
7.3.1 Quality of ranked publications . . . 83
7.3.1.1 Testing zipf law hypothesis . . . 83
7.3.1.2 Other diversity based algorithms . . . 89
7.3.2 Accuracy of ranked publications . . . 90

7.3.3 System resiliency 92
7.3.4 Freshness of ranked publications 94
7.4 Performance & Efficiency 96
7.4.1 Index subscriptions 97
7.4.1.1 Index construction time 97
7.4.1.2 Initial matching time 98
7.4.2 Index publications 99
7.4.2.1 Batch-wise Top-k computation 99
7.4.2.2 Incremental Top-k computation 103

8 Conclusion & Future work **105**
8.1 Top-k publish/subscribe applications 106
8.2 Future Work 107

Appendices **108**

A Naive algorithms **109**

B Index based algorithms **111**

C LSH based algorithms **113**

List of Figures

1.1 An overall illustration of pub/sub system . . . 3
1.2 Boolean pub/sub system . . . 4
1.3 General Top-k publish/subscribe architecture . . . 5

3.1 The basic architecture of boolean pub/sub system . . . 16

4.1 Top-k publish/subscribe architecture . . . 24
4.2 Subscription covering vs. Subscription Merging vs. Attribute Merging . . . 27
4.3 Subscriber preference models . . . 28
4.4 Bob's personalized subscription space . . . 30
4.5 Satisfying subscription space . . . 30
4.6 An example of redundant information in traditional Top-k answer set . . . 34
4.7 MAXDIVREL demonstration using dummy publications . . . 39
4.8 Graph representation of publication space with neighborhood α . . . 40
4.9 Dominating sets (red points) . . . 40
4.10 Dominating set but not independent . . . 40
4.11 Sliding window approach: $w = 5$ . . . 43
4.12 An update of k-independent dominating set after an arrival of new publication p_6; k=2 . . . 44
4.13 A jumping window with $w = 5$ & *jumping step* $= 3$ . . . 45
4.14 Centralized Top-k publish/subscribe architecture . . . 47

5.1 Attribute Lists . . . 51
5.2 Operator Lists on $L_{carrier}$ . . . 51
5.3 Inverted-list two-level partitioning on user subscription space . . . 53
5.4 Inverted-graph on personalized user subscription space . . . 54
5.5 Top-k matching on sliding window approach in threshold based schemes; $w = 5$ 55
5.6 MAXDIVREL Top-k matching on sliding window approach: $w = 5$ . . . 55

5.7 R near-neighbor query . . . 56
5.8 LSH probability vs distance . . . 58
5.9 Bucket array of n publication at L hash tables . . . 64
5.10 Flow chart of incremental Top-k computation . . . 65
5.11 Sample LSH indexing mechanism at arbitrary time $t + 1$s . . . 66
5.12 The behavior of s-curve . . . 68

6.1 Implementation modules based on Amazon web services . . . 71
6.2 Stream Processing platforms available at current date . . . 72
6.3 Building Kinesis processing . . . 73
6.4 Memcached nodes shared in separate AEC tier (source: AWS) . . . 75
6.5 SNS Topics + Personalized user subscription spaces per interest . . . 76

7.1 Distribution of ranked votes . . . 85
7.2 Illustration of zipf exponent values . . . 86
7.3 A comparison of p-values with alternative heavy-tail distribution . . . 89
7.4 A comparison of zipf exponent values on different diversity algorithms . . . 91
7.5 Accuracy of ranked results produced by LSH index . . . 92
7.6 System resiliency test . . . 93
7.7 A comparison between relevancy scores over forward-decayed & non-decayed cases 95
7.8 Analyzing decay parameter under different ρ rates . . . 96
7.9 Index construction time on opIndex vs. modified opIndex . . . 97
7.10 Average matching time over the size of subscription spaces under different subscriber views . . . 98
7.11 Average matching time over the size of publications . . . 99
7.12 Batch-wise Top-k computation by BLSH & NAIVE . . . 99
7.13 BLSH construction + matching time in comparison with estimated error e . . . 100
7.14 Average BLSH Top-k matching time in comparison with size D of publications . 102
7.15 Average NAIVE matching time in comparison with size D of publications . . . 102
7.16 BLSH vs. NAIVE when D=250 . . . 103
7.17 BLSH vs. NAIVE when D=500 . . . 103
7.18 Average ILSH Top-k matching time in comparison with size D of publications . 104
7.19 ILSH vs. BLSH vs. NAIVE when D=250 . . . 104
7.20 ILSH vs. BLSH vs. NAIVE when D=500 . . . 104

List of Tables

5.1 Example subscriptions 50
5.2 First level partitioning of subscriptions 51
5.3 Example personalized subscriptions 52
5.4 Categorical view of publication $X\&Y$ 58
5.5 A characteristic matrix representing the existence of categorical values at publications 59
5.6 Calculating h_1 minhash value from the first permutation 61
5.7 A signature matrix that represent publications 61
5.8 Sample signature matrix generated for 5 publications 61
5.9 Segmented signature matrix generated for 5 publications 63
5.10 Top-2 publications retrieval based on votes 64
5.11 Top-2 publications retrieval at time $t + 1$ 67

7.1 Summary of measurements on retrieved product data 79
7.2 A sample set of attributes (10/57) that represent the subcategory "smartphones" 79
7.3 Sample Publication X 80
7.4 Sample Subscription S with preference values 80
7.5 Symbol values on parameter space 82
7.6 A sample rank-wise votes distribution of Top-k publications 85
7.7 Average zipf law exponent for ranked publications in comparison with different subscriber views 86
7.8 An advance rank-wise votes distribution in a comparison with similarity threshold 87
7.9 Average power law exponent for different k values in comparison with similarity threshold 87
7.10 p-values from KS test for ranked scores in comparison with different subscriber views 88

7.11 Average power law exponent for Top-k publications in comparison with other diversity methods 90
7.12 ILSH update cost 103

List of Algorithms

4.1 Naive algorithm to solve MAXDIVREL k-diversity problem 41
A.1 Constructing & updating personalized subscription graph 109
A.2 Computing the relevancy score (Naive method) 110
B.1 Insertion algorithm for modified opIndex 111
B.2 Computing the relevancy score (Index based method) 112
C.1 Fast Min-Hashing algorithm 113

List of Symbols

Symbol	Parameter
C	Clients
X	User
S	A set of subscriptions
s_i	i^{th} subscription
pS^X	Personalized subscription space submitted by user X
a_i	i^{th} subscription tuple at a subscription
d_s	size of a subscription
P	A set of publications
p_i	i^{th} publication
b_i	i^{th} element of a publication
k	The number of delivered publications at a given time
m_s	Matching function
$r(p_i, X)$	Relevancy score of a publication p_i for the user X
α	The size of neighborhood
D	Dimension
$d(p_i, p_j)$	Distance between publications p_i and p_j
$u(t)$	Decay function at given time t
t	Time
θ	Logical operator
λ	Tuning parameter
L	Number of hash tables
b	Number of buckets at particular hash table
r	The size of minhash signature vector
μ	The Poisson arrival rate of publication
ρ	Decay parameter for relevancy score
s	LSH similarity threshold
m	Number of minhash functions
e	estimated error at minhash signatures
χ	zipf exponent

Acronyms

DBMS Database Management Systems

DSPS Data Stream Processing Systems

IR Information Retrieval

DNF Disjunctive Normal Form

CNF Conjunctive Normal Form

P2P Peer-to-Peer

VSM Virtual Shared Memory

MITL Metric Interval Temporal Logic

RTS Real Time Systems

NN Near Neighbor

LSH Locality Sensitive Hashing

AWS Amazon Web Services

SNS Amazon Simple Notification Service

SQS Amazon Simple Queue Service

EC2 Amazon Elastic Compute Cloud

AEC Amazon Elastic cache

KCL Kinesis Client Library

BMR Best-Matching-Random

KS Kolmogorov-Smirnov

CDF Cumulative Distribution Function

Chapter 1

Introduction

1.1 Preamble

Information Technology has been rapidly grown in last decades which has caused the omnipresent phenomenon *Information Explosion*[1] to come into act along with an exponential growth of newly created digital information. This is not an exclusive problem to the modern age information society, but even from the bronze ages we're spun by different formats of information supernova. In 2003, Google CEO Eric Schmidt made an incredible statement to emphasize why it's so hard to operate in digital information markets, since the size of new world information is as double as the size of created information between the birth of the world & 2003[2].

Digital information was accumulated to stream into our homes since the birth of World-wide-web. We became not only to consume information, but to produce while sharing them in worldwide community of subjective thought. According to UC San Diego investigation in 2010, world's servers processed 9.57 zettabytes(10^{21} bytes) of information [1]. That depicts that *Information Explosion* was happening faster than UC Berkley predicted[3] in 2003. Meanwhile, [2] identifies that our global technological memory has roughly doubled every three years over recent decades.

Information Explosion would not be a huge problem if information can be stored permanently. But fast growing corners of digital universe proved that it already exceeded the available storage

[1] http://www.vcreporter.com/cms/story/detail/information_explosion/7958/
[2] http://www.huffingtonpost.com/brett-king/too-much-content-a-world-_b_809677.html
[3] http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/

for the first time in 2007. By 2011, almost half of the digital universe was not recorded in information storage mediums [3].

Since every created information has no future value, we have to process such information before storing it to decide which actually has value to be stored. Such that users should be capable enough to expose into new information. As human-beings are contemplative, the stillness of information would yield positively up to certain point and get declined by the overloaded information. So intuitively *Information Overload* [1] is noticeable phenomenon which also highly-related to Information explosion.

In other hand, by alleviating *Information Overload* problem, we can avoid cognitive dissonance[2] of human behavior up to some degree. It would be beneficial in their daily lives to take decisions based on rapidly growing continuous collection of data-streams.

Given the runaway popularity of information in our daily lives, still we can't make time to take so many calls, answer so many e-mails, peruse so many websites (e.g. social-media). Probably we can't take it all in and that lead us not to know about everything around us. For the better or worse, we believe that the wise fact would be not to chase information, but setting it aside will only make us remain far behind the curtain of rapidly growing information society.

1.2 Background to the study

"We have more information than ever, but in the ever-thickening forest of information, the beauty of the single tree becomes ever harder to distinguish." - James Scolari, Journalist

To deal with limited amount of most relevant information, different classes of information processing techniques have been emerged, which are capable enough to timely process large amount of information. In traditional Database Management Systems (DBMS), the data need to store and index before it could be processed explicitly by the user. As the evaluation of above traditional store-then-query models, Data Stream Processing Systems (DSPS) process streams of data coming from different sources to produce new data streams as an output. DSPS continuously process unbounded data streams looking for events of interest to the end-user. DSPS have their roots in DBMS. But along with the substantial differences in processing

[1] http//en.wikipedia.org/wiki/Information_Overload
[2] http://en.wikipedia.org/wiki/Cognitive_dissonance

information through a sequence of transformations, now a days, DSPS resemble DBMS in particular classes of applications [4].

That paradigm shift in information processing from store-then-query models towards emerging query-then-store models is vital for a number of modern day applications by considering space and real-time requirements.

Publish/Subscribe models Our study focuses on publish/subscribe systems which are based on the query-then-store paradigm since publications are first processed by the publish/subscribe service, and then, if necessary, they will be stored by subscribers.

Publish/subscribe approach represents subscribers who express their interests by a query or a pattern of queries where published items by publishers need to be delivered to relevant subscribers in a timely manner. They are tuned to filter large amounts of published information in real-time and deliver matching publications to interested subscribers based on efficient matching and routing algorithms. [Figure 1.1]

As modern large-content based applications continuously generate huge data volumes at high data rate in different data varieties, DSPS require efficient & effective ways for continuous processing and data filtering followed by timely delivery of relevant information. From extensive research efforts done at last 15 years, publish/subscribe systems as one generalization of DSPS are widely recognized to process continuous queries over data streams which are augmented by algorithms coming from the field of data stream processing [5].

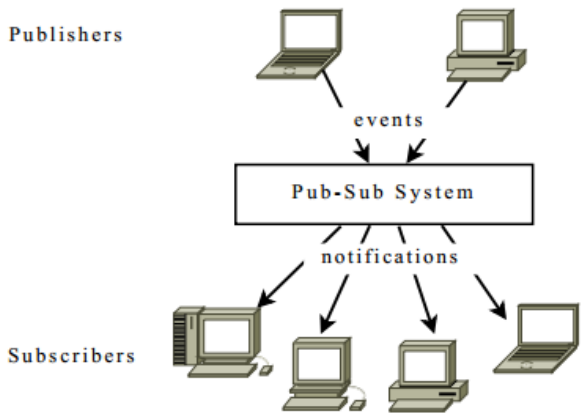


Figure 1.1: An overall illustration of pub/sub system

1.3 Motivation

Existing functions which are capable to match publications and subscription in state-of-the-art pub/sub systems are depended on the stateless matching function which provides a Boolean decision whether a given publication to be notified to relevant subscriber or not [6]. [Figure 1.2]

Identified drawbacks of state-of-the-art pub/sub matching models [7]

- A subscriber may be either overloaded with publications or receive too few publications over time,
- It is impossible to compare different matching publications with respect to a subscription as ranking functions are not defined, and
- Partial matching between subscriptions and publications is not supported.

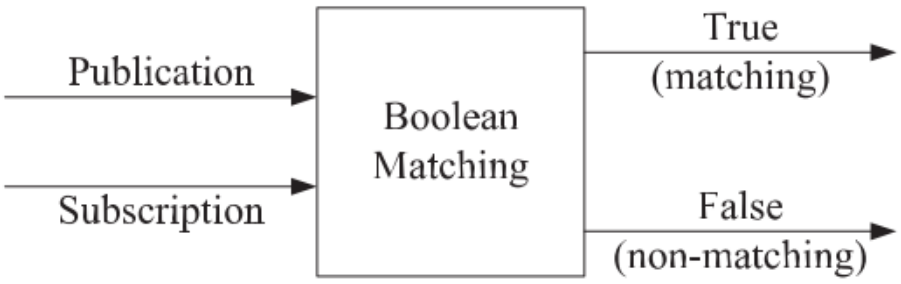


Figure 1.2: Boolean pub/sub system

As end-user ranks the system by the quantity and quality of received publications, he should put 'ideal' subscriptions to avoid dissatisfaction. A large quantity of received publications will be considered as a sort of spam, while a system that delivers too few publications might be recognized as non-working. Unpredictable number of delivered publications remain as one potential reason to be presented for the slow adoption of large-scale publish/subscribe solutions [6]. Along the side, publish/subscribe models should support to deliver a limited amount of information to prevent information overload and this information has to be of top quality.

Top-k Publish/Subscribe models Recently top-k publish/subscribe models have attracted a lot of attention as a means to provide rankings among selected publications to control the

number of publications it receives per subscription along with time [8, 7, 9, 10, 6]. Also some index methods are introduced to support ranked pub/sub matching [11, 12, 13]

Top-k publish/subscribe models are identical to state-of-the-art publish/subscribe in most terms except it's expressive stateful query processing nature which targets to overcome the drawbacks in latter models.[Figure 1.3]

In traditional pub/sub systems, publications trigger subscription when they match a subscription's predicate [Figure 1.2]. In Top-k pub/sub each subscription scores publications based on different scoring between publications & subscription. Achieved score depicts the importance of a publication against particular subscription but also on their relationship with previously seen publications. Different views have been proposed in the way, a subscription will be triggered by a new publication in Top-k pub/sub [8].

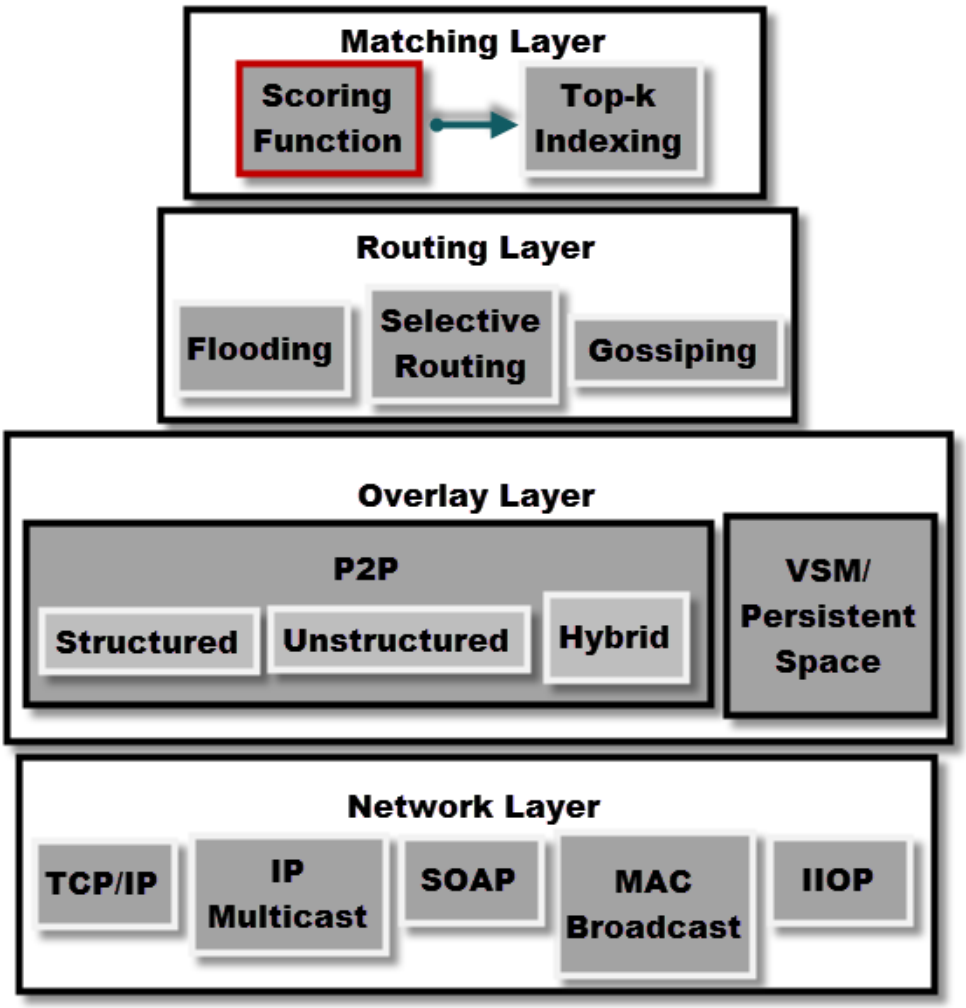


Figure 1.3: General Top-k publish/subscribe architecture

Possible use-case We would like to optimize Top-k heuristic under a use case study which supports a large volume of subscriptions and high event arrival rate (velocity) with a wide variety of product items such as e-commerce. For an example, take the user Bob, who generally likes to update about smart-phones but prefers the products from AT&T. Ideally he would like to get notify on products from Verizon, only if there are not enough notifications from AT&T. In state-of-the-art publish/subscribe systems, subscriptions & matching publications are considered as equally important. Publications are delivered to Bob whenever there is a satisfied subscription. That leads Bob to easily get overwhelmed by the notifications he received. But he doesn't like to get too many or too few notifications along with the time. He would like to have a limited amount of most important information across the stream of information in the long run. Bob doesn't like to go through all of his received publications to select what he would like to buy. But as Bob's subscriptions are partially matching, there's no way to track them down in traditional models. Because each subscription & publication is served uniquely. Unpredictable number of delivered publications remain as one potential reason to be presented for the slow adoption of large-scale publish/subscribe solutions.

1.4 Goals of the thesis

In our study, we propose ranking mechanisms to enhance state-of-the-art Boolean publish/subscribe models by integrating query independent & dependent score metrics taken into account. Hence, the accuracy of the model achieved by matching the **relevancy** between publications & subscriptions which are **personalized** under user preferences. We guarantee to provide a dynamic **diversified** set of results over the continuous data stream which enhance the user expressive power by combining diversity with relevance while also maintaining the maximal **freshness** of the k delivered publications.

The proposed scoring algorithm doesn't require re-evaluation of current Top-k publication list when new publication arrives. Because we believe that skipping a significant fraction of score computation can reduce CPU usage and processing time of incoming publications accordingly.

By allowing continuous Top-k query processing, our objective is to enhance algorithm performance while it's scalable to the **volume** of subscriptions, the arrival rate of events (**velocity**) and the **variety** of subscribable attributes. We exploit the dynamic case which the Top-k result set changes over time, hence existing index data-structures are extended to support incremental Top-k result computation.

Problem Statement

Big Picture How to alleviate the *Information Overload* problem based on publish/subscribe communication paradigm augmented by different ranking mechanisms which are scalable to the volume, velocity and variety of data-streams? Hence, it addresses the efficient processing of Top-k queries over a continuous data stream which filters out irrelevant data stream objects, and delivers only Top-k most diversified objects relevant to the user interest.

Research problem

- How to define an efficient scoring algorithm by integrating query independent & dependent score metrics taken into account? Hence the proposed scoring algorithm doesn't require re-evaluation of current Top-k publication list when new publication arrives.
- How to extend existing indexing data structures used in state-of-the-art publish/subscribe systems to support Top-k matching queries under large subscription volume, high event rate(velocity) & the variety of subscribable attributes?

1.5 Scope of the thesis

- The information provider publishes information in the form of events with attribute-value data tuples. Information consumer (subscriber) subscribes interesting events in the form of Boolean expression with attribute-operator-value. Above fixed structured data is equipped with relevant timestamps.
- We're not going to cover the lower layers in general Top-k publish/subscribe architecture [Figure 1.3], but target to design efficient scoring algorithm in the matching layer which is independent of the underlying architecture it's plugged in.
- The study focuses to retrieve most relevant matching publications against subscriptions, but not on the reverse.
- The importance of incoming publications may vary along with the time & the velocity of publications follows a Poisson distribution.

1.6 Thesis outline

The thesis is organized as follows. Chapter 2 describes the related work & how our approach is distinct with them. Chapter 3 formally defines the preliminaries in Boolean publish/subscribe models & derives Top-k publish/subscribe model concepts. In Chapter 4, we examine different variations of ranking mechanisms to compute the importance of published events per each user. Also the delivery of Top-k publications are discussed for different timing policies. Chapter 5 introduces existing indexing mechanisms in state-of-the-art publish/subscribe systems & shows how to extend them to support Top-k matching. In Chapter 6, we present our evaluation setup & the experimental results. Chapter 7 summarizes our study & gives guidance to the future works.

Chapter 2

Related work

We believe that optimizing Top-k heuristic is subjective under different use-cases. In our study, we stick to the e-commerce environment on retrieving Top-k product items continuously on a given query space per user. In e-commerce domain, we have to deal with a wide variety of items, each with different attributes. As there are no prior work has been done in the e-commerce context of Top-k publish/subscribe models, we initially focus to reduce this gap by reviewing the existing literature at on-line market places.

2.1 E-commerce ranking methods

Diversity and its relations are initially studied to search relevance in the context of an on-line marketplace (e.g. EBay)[14]. The work has been powered by a comprehensive study using click-stream data to identify relevant ranking metrics. They conducted an empirical study to find the inter-relation between those metrics. The work is extended by applying different learning models [15]. All suggested ranking methods require maximum user interaction with the system to get better results. Also the efficiency of the proposed algorithms are only presented in the database context.

Skyline based methods are extensively studied in database & data-stream community for top-k matching. We have brought our view on retrieving most important results to users without based on skyline algorithms [15].

2.2 General Top-k Publish/subscribe

On the other hand, different variations of general Top-k publish/subscribe models are proposed for the integration of ranking issues. Top-k/w publish/subscribe model which was pro-

posed by [6], ranks publications according to their relevance to a subscription and delivers Top-k publications per subscription in a predefined sliding time window (w). They defined a probabilistic model to determine whether a newly published item may become one of Top-k publications in the publication queue before the time window get expired for that publication. In this model, the relevance of an event remains constant during the time window and once its lifetime exceeds w, the event simply expires. Expired object will be replaced by one of most competing unexpired object in the publication queue. But solutions of this model face challenges when identifying & keeping track of published items that may become relevant in the future due to expiration of older items. They mainly focused on efficiently maintaining publication queue, their prototype model is implemented in a distributed environment [7].

In [16], authors examined Top-k/w matching model, but only used relevance between publications & subscriptions as the matching criterion like the former model. But here, our aim is to provide an efficient scoring algorithm to analyze the relation between different score metrics to compute the ranks of matching publication.

Top-k publish/subscribe model which was proposed by Google [8] most recently, has overcome the overhead of frequent re-evaluation of Top-k publications introduced by the former model [6] without defining a sliding time window. Instead of keeping a fixed expiration time for a publication, they introduced a simple score function which gradually decays with time. Therefore older events expire from the Top-k publication until new events take their place by scoring higher values. They don't require to maintain previously seen events per subscription unless they're in current Top-k list. They mainly consider to efficiently annotate news stories with social content in real-time. Also they rediscovered the adaption of Top-k document retrieval algorithms in publish/subscribe paradigm to demonstrate the feasibility of proposed approach. But designing a high-quality scoring function for matching publications to stories was beyond their scope.

In [17], authors presented the concepts of preferential subscriptions to enhance the expressiveness in publish/subscribe systems. They implemented their approach in PrefSIENA: a popular publish/subscribe system.

2.3 All in one: Diversity

We believe that *diversity* is the most promising factor to enhance user satisfaction as query independent metric. Many general definitions & approaches to results diversification has proposed in pub/sub literature under static data. Their definitions of diversity are general based on *dissimilarity, coverage* & *novelty* [18]. Most approaches rely either on greedy or interchange heuristics, due to the NP-hardness of the k-diversity problem. Most of prior work did consider above definitions orthogonally to reduce complexity [19, 9]. Recently [20] proposed a new definition of diversity ($DiscC$) by introducing adaptive diversification based on dissimilarity & coverage.

We believe that Top-k publish/subscribe is an instance where continuous diversity is captured for information filtering. In our study, we consider dynamic diversification problem where Top-k result set has to be updated over continuous data-stream while combining many other score metrics. This problem is identified as *continuous k-diversity problem* in the recent literature [21, 22, 23].

The continuous-k-diversity problem was also addressed by using heuristic based solutions, greedy [21] & interchange [24] under dissimilarity between items. [25] made a distinction between novelty & diversity where diverse documents are retrieved based on a probabilistic model while [26] has a complementary view between novelty & diversity. But the related literature focusing on above problem is considerably more limited in Top-k publish/subscribe context.

The concept of ranking based on both relevance and dissimilarity, is applied to present a number of delivery modes for forwarding events to users [9]. However, with these methods, old items do not expire, and a new item may enter the solution only upon its arrival. Later they added freshness into their combining criteria which is supported by linearly degrading aging techniques [10].

Based on a variation of dissimilarity(e.g. MAXSUM), diversification problem is studied in [27], in the setting of streaming data and monotone submodular diversification functions. An approximation greedy algorithm was proposed which is faster than the usual greedy heuristic. Dynamic updates are also considered in the sense that when the underlying set of available items changes, interchanges are attempted to improve the computed solution. Finally, the on-

line version of the diversity problem is considered in [28], that is, selecting a diverse subset in the absence of the complete set of items.

Here, we propose a hashing based index mechanism to provide an efficient incremental Top-k computation over continuous data-stream.

Combine Diversity with Relevance

We don't consider diversity & relevance independently, but to combine them to reduce the over-selecting of more relevant but similar publications. [29] developed a natural axiomatic approach to & show that no diversification method can simultaneously satisfy all the axioms. Diversity can be viewed uniquely in the way to minimize *query abandonment* by finding at least one item that satisfy the end-user. They identified that objectives of being diverse & relevant are competing with each other which results the diversification problem as a bi-criteria optimization problem. Designing right function to express dissimilarity between items will be a key to have an effective system.

2.4 Subscription Indexing

As publish/subscribe models are extensively studied over decades, there has been a lot of attention on indexing support to efficiently identify matching subscriptions [13, 12, 30, 11, 31]. Here we did only consider in-memory indexing approaches which are implemented over linear & hierarchical data-structures.

In traditional approaches, a regular grid is used to index boolean subscriptions [32]. But when the subscription subspace of interest changes, it can affect high update cost on corresponding cells. Also applying indexing mechanism on some of the proposed top-k publish/subscribe models (e.g. top-k/w queries) is still an open research problem [7].

Based on inverted list index, [12] proposed k-index where subscription predicates are partitioned into subsets using a three-level partitioning scheme. But it does perform poorly under generated indexing space, specially where range predicate in a subscription needs to be rewritten into a disjunction of equality predicates.

[30] proposed two-phase space-cutting technique which organizes the subscriptions in a hierarchical index called BE-Tree. But as the number of attributes increases, BE-Tree incurs

higher construction, optimization and access cost. Later the same set of authors extended their prior work into BE* Tree which achieves effective pruning for determining most relevant matches [11]. Since we are interested in the problem of efficient event matching, finding the most relevant subscriptions per publication is beyond our scope.

[33] explored the notion of sharing query results among other queries to reduce processing cost of top-k results re-computation. They exploited covering graphs to evaluate top-k queries over document streams by using sliding window model.

[31] has proposed an efficient in-memory indexing called opIndex to deal with increasing variety of items, each with different attributes. Subscriptions are partitioned using a two-level partitioning scheme by attributes & operators consequently. Selection of a best pivot attribute helps to prune the subscription space when matching with events. Also opIndex supports much complex matching operators including prefix/suffix & regular expressions. Other indexing structures are outperformed by opIndex from memory consumption, index construction and query processing cost over large volume & variety of subscriptions.

2.5 Publication Indexing

When publications match with relevant subscriptions, we need to update & maintain the k best publications per subscription under different users. To avoid the redundancy on Top-k results computation in the publication space & to avoid re-computation, recent literature proposed a couple of index based approaches based on different space-cutting data-structures [34, 23, 33]. Here, we also focus on combining relevance and diversity by viewing diversification as a Top-k problem.

[34] motivated continuous diversity problem by formalizing that non "off-the-shelf" Information Retrieval (IR) engines can be used to implement diversity. It allows the exploitation of a Dewey encoding tree using inverted-lists which is later used to select the k most diverse tuples. But their study is limited to this specific diversity measure which did not provide any dynamic treatments.

[23] formalized continuous-k-diversity problem by introducing *continuity* requirements to deal with continuous data-streams. They provided efficient diversity algorithms that are based on *cover-tree*. Also they have avoided the curse of re-computation on top-k results in sliding window models by proposing incremental algorithms. Based on a variation of dissimilarity

(MAXMIN), continuous k-diversity problem is studied. Also they extended their approach to combine relevance.

In our approach, we consider a different variation of dissimilarity. Also our study is to align proposed diversity problem with Near Neighbor (NN) queries where we can adopt hashing based techniques to achieve a specialized form of continuity requirements in diversity. We also argue that the correct definition of diversity is application dependent which has not been considered in former work. Also our approach is extended to combine different score metrics other than relevance. Additionally our indexing mechanism would achieve an efficient pruning in both subscription & publication spaces over incremental algorithms to avoid the curse of Top-k re-computation.

Chapter 3

Preliminaries of publish/subscribe models

In this Chapter, we show a clear distinction between boolean pub/sub & Top-k pub/sub models. We present the structural & behavioral view on both models using a formal background. For the completeness of our study, we first formally specify the boolean pub/sub models more in general, but later we extensively study Top-k pub/sub model. Both specifications are presented using *temporal logic*.

The major important difference between boolean & Top-k pub/sub is in the matching model. Boolean pub/sub has a stateless matching model which only provides a boolean decision, while the matching in Top-k pub/sub is called as scoring or ranking, which provides stateful matching model to express the user intent. Latter model can rank the publications on given scoring function & restrict the delivered publications to the user by the parameter k.

3.1 Boolean Publish/Subscribe

Boolean publish/subscribe models are well studied in the distributed system community over decades. Users can express their interest over a set of information by a subscription or query. The one who injects a subscription to the system is called as the subscriber. Users that generate such information & pushes to the system are called as publishers. The pushed information are publications & whenever they satisfy the subspace of user interest (subscriptions), they will be delivered to subscribers as notifications. The basic architecture of boolean pub/sub system is depicted in [Figure 3.1].

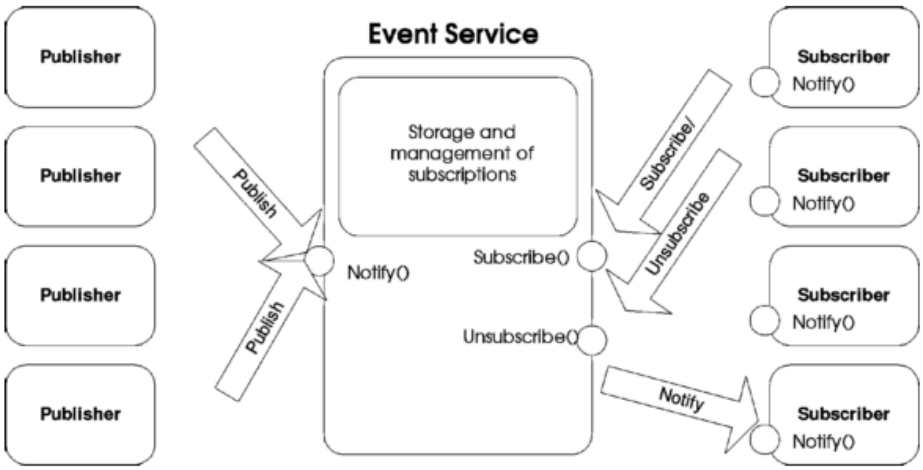


Figure 3.1: The basic architecture of boolean pub/sub system

There are many variants of publish/subscribe schemes: namely, *topic-based, content-based, and type-based*. Loosely connected subscriber & publisher nodes can act as general clients which are connected via a separate middle-ware asynchronously. Connected middle-ware can be centralized or de-centralized. Client-server or broker based middle-ware are well known centralized ones. De-centralized middle-ware is based on Peer-to-Peer (P2P) or Virtual Shared Memory (VSM)/ persistent space. Many routing strategies have been presented over years in P2P boolean pub/sub [Figure 1.3].

For the rest of our study, we consider content-based pub/sub model without losing the expressiveness of the interaction between publishers & subscribers.

3.1.1 Structural View

Definition 3.1.1. Given a finite set of clients C, a finite set of publications P & a finite set of boolean subscriptions S, the structural view of a boolean pub/sub is defined by B [35],

$$B = (C,\ P,\ S)$$

$$where\ \forall c_i \in C;\ \forall p_j \in P;\ \forall s_k \in S;$$

3.1.1.1 Subscription

Subscriptions can be thought as a way to express user intent. They are used to filter-out irrelevant information. Each subscription consists of a set of constraints which is defined by *attribute-operator-value* tuples. Each attribute can be typed or untyped. Binary operators may

include common logical operators such as $\{=,\ \neq,\ <,\ \leq,\ >, \geq,\ prefixof,\ suffixof\}$.

Definition 3.1.2. A subscription is a set of attribute-operator-value tuples a_i, where each a_i has the form $\{a_i.name\ a_i.\theta\ a_i.value\}$ s.t. $\{=,\ \neq,\ <,\ \leq,\ >, \geq,\ prefixof,\ suffixof\} \in \theta$ which is related under Disjunctive Normal Form (DNF) or Conjunctive Normal Form (CNF) $o; \{\vee, \wedge\}\ \in o;\ s.t.\ a_i\ o\ a_j$,

$$s_i = \{a_1\ o\ a_2\ o\ .....o\ a_n\}\ where\ \forall s_i \in S; 1 \geq i \geq n;$$

Subscriptions can only be in a state of *ACTIVE* where clients (C) are subscribed into.

3.1.1.2 Publication

Subscriptions are queried over the set of publications. When a subscription is satisfied by a publication, it will be eventually delivered to the user. Each publication consists of a set of constraints which is defined by *attribute-operator-value* tuples. Each attribute can be typed or untyped. But usually, operator includes only the equal operator. Thus, attribute can expressed by a set of finite values.

Definition 3.1.3. A publication is a set of attribute-operator-value tuples b_i, where each b_i has the form $\{b_i.name\ b_i.\theta\ b_i.value\}$ s.t. $\{=\} \in \theta$

$$p_i = \{b_1,\ b_2,\ .....,b_m\}\ where\ \forall p_i \in P;\ 1 \leq i \leq m;$$

Publications can be in *ACTIVE* state as soon as they are published, but become *INACTIVE* for the relevant clients(C) when they are delivered.

3.1.2 Behavioral view

As depicted in Figure 3.1, following are the general view of behavioral events that can occur at a boolean pub/sub system. (i) publish(p_i) (ii) subscribe(s_i) (iii) unsubscribe(s_i) (iv) notify(p_i)

Subscribers express their subspace of interest by *subscribe(s_i)* where s_i refers to a subscription which is a set of constraints that defines the user interest. Subscribers can later revoke this interest through a corresponding *unsubscribe(s_i)* operation. Publishers use *publish(p_i)* operation to disseminate a publication p_i over the the space of user interest [36].

Boolean matching Notifications are delivered using *notify(p_i)* operation whenever the publication p_i is satisfied by any of the active subscriptions.

Definition 3.1.4. Given a publication $p_i = \{b_1,\ b_2,\ .....,b_m\}$ and, a subscription $s_j = \{a_1\ o\ a_2\ o\ .....o\ a_n\}$, s_j covers p_i, or alternatively p_i matches s_j if and only if, when the logical operator $o = \{\wedge\}$ then $\forall b_x \in p_i . \exists a_y \in s_j$ and when $o = \{\vee\}$ then $\exists b_x \in p_i$ & $\exists a_y \in s_j$ such that,

$$b_m.name = a_n.name \wedge b_m.value\ a_n.\theta\ a_n.value$$

As long as there is at least one active subscription which satisfies the given publication, a notification will be delivered to all users who have submitted that subscription.

3.2 Top-k Publish/Subscribe

Formal specification of Top-k publish/subscribe models is dependent on the proposed approach. There is no general definition to use in Top-k publish/subscribe. So the structural & behavioral definitions of Top-k matching model are aligned with our use case parties: buyers(i.e. subscribers) & sellers(i.e. publishers).

3.2.1 Structural view

We also follow the basic definition 3.1.1 which was introduced for Boolean publish/subscribe. But we will introduce additional system states within our boundary of space to be incorporated in our Top-k publish/subscribe model.

3.2.1.1 Personalized Subscription Space

To explicit more expressive nature of subscriptions, we introduce the concept of *personalized subscription graph* by relating subscription tuples $a_i \in s_i$ for user X in a directed graph. As our goal is to control the unpredictable number of delivered publications, we allow the subscribers to decide which publications are important to them. We request them to subscribe into subscription tuples by adding a relative preference. The representative power of user subscription space is enhances from that personalized tuples.

Definition 3.2.1 (Personalized subscription space)**.** A personalized subscription space pS^X for user X is a set of $\{(attribute-operator-value),\ (preference)\}$ tuples a_i, where each a_i has the form $\{(a_i.name\ a_i.\theta\ a_i.value), a_i.pref^X\}$ s.t. $\{=,\ \neq,\ <,\ \leq,\ >, \geq,\ prefixof,\ suffixof\} \in \theta$ and $a_i.pref^X$ is a real number in [0,1] that expresses the relative degree of interest of the tuple

than other tuples $\forall a_j \in pS^X \; where \; i \neq j$,

$$pS^X = \{a_1, a_2, ..., a_n\} \; where \; 1 \leq i \leq n;$$

A tuple which has a higher preference score is more important than others. Unlike Boolean or other general Top-k publish/subscribe models, here any user X is subscribed into a subscription space which is generated by the system from user given subscription tuples, but not to any single subscription individually.

Definition 3.2.2 (Personalized subscription graph)**.** Let a digraph $D = (V, E)$ consists of the vertices V_D where each subscription tuple a_i is represented by an unique vertex v_i s.t. $\forall v_i \in V_D \; \exists a_i \in pS^X$ where pS^X is the preferential subscription tuple space submitted by user X and, directed edges $E \subseteq V \times V$ (without $vv, and \; uv \neq vu$), For each edge (v_i, v_j) defines the relation between (a_i, a_j) and, the capacity function $c : V \times V \rightarrow \mathbb{R}_+$ for which $c(v_i, v_j) = 0, if (v_i, v_j) \notin E$, and $c(v_i, v_j) = \dfrac{preference(v_i)}{preference(v_j)} \; if \; (v_i, v_j) \in E_D$ where $preference$ is some utility function to extract user interest over any vertex. Also the order $|V|$ and the size $|E|$ of the graph satisfy $|E| \leq |V| \; where \; |V| = |pS^X|$ where the cardinality of the user subscription space is denoted by the number of non-duplicate subscription tuples.

Subscription tuples should be in ACTIVE state once added. Subscriber has the possibility to change tuple contents, redefine relations by assigning new preference scores. Also user can temporarily disable or permanently delete some tuples where the state of tuple is triggered to become INACTIVE or DEAD consequently. The whole subscription space is in the state of ACTIVE when the number of ACTIVE tuples (subspace) is above a given threshold.

Preference or the degree of interest can be depicted by either quantitative or qualitative manner. To have more granularity, here we discuss only quantitative approach where users have to provide numeric scores explicitly. Discussing ways to have above preference values and, applying other qualitative approaches(e.g. binary decision diagrams, fuzzy reasoning) to enhance user satisfaction is beyond our scope. But at the section 4.2.1 we provide some guidelines to adopt & discuss more about personalized subscription spaces.

3.2.1.2 Publication Stream

We say that any publication p_i is more relevant or preferable than publication p_j to user X, if it satisfies $r(p_i, X) > r(p_j, X)$. As our relevancy function $r(.)$ is subscriber friendly, we can adopt

an identical structural view of publication (as definition 3.1.3) in Boolean publish/subscribe models.

Since we study Top-k continuous query processing over incoming publication stream, disclosing stream properties is important.

Stream properties A publication stream is a continuous and possibly infinite sequence of publications that arrive in an arbitrary order (implicitly by the sequence number assigned on the arrival or explicitly by the time-stamp) into the publish/subscribe matching model to be processed in real-time [37]. As it's impossible to store the stream in its entirely, our model provides incremental matching algorithms to return Top-k results as new publications arrive.

- Query semantics allow order or time based window processing.
- Backtracking over the stream is application dependent.
- Publication arrival rate follow a Poisson distribution.
- Eviction of old publications is time dependent.

The publication stream becomes ACTIVE for a user subscription space pS^X submitted by user X, after the first subscription tuple in the space becomes ACTIVE. So, the state of publication stream is causally effected by the particular user subscription space. Further, the publications in the given stream become ACTIVE while they are within the query window but not expired, and become INACTIVE when they are dropped from the all windows. Publications which will be selected as Top-k results, are marked TOP, and eventually they are in INACTIVE state when they're delivered successfully. Note that all above states are in the boundary of single user.

3.2.1.3 Subscription parameters

As we described earlier, the parameter k controls the number of delivered publications in a given time instance t. Since the subjective nature of above delivery, we allow the user to specify the value of $k \in \mathbb{N}$ per subscription space. Any user can have many subscription spaces per interest under different values of its subscription parameters.

Incoming publications are ranked based on time or count based sliding window over the stream. Most recent literature obtain the size of sliding window from the user [7, 9]. But we

believe the window size is almost dependent on the stream of publications and, provide an important backbone in Top-k matching. Hence we explore a stochastic model to dynamically slide a fix size window based on stream arrival rate in the Section 4.3.

3.2.2 Behavioral view

Top-k publish/subscribe model does have an identical behavior with Boolean model, but is not restricted. Along with the independent modified behavior calls produced by clients (C) (i) publish(p_i) (ii) subscribe(s_i) (iii) unsubscribe(s_i) (iv) notify(p_i) ,our model supports $expire(p_i)$ which is produced by the system where the publication p_i is leased linearly or exponentially in time. The corresponding $publish(p_i)$ precedes every $expire(p_i)$ by a forward time decay function (Definition 4.2.4). Any unexpired publication may compete for a position of Top-k results in subsequent query windows. But once they're selected as Top-k results & get delivered, they're not allowed to repeat the same process using system states.

We explore the behavior $match(pS^X, p_i)$ where publication p_i is scored against the space of user interest pS^X and, diversified within a subscription window as described in the Section 4.2. Only Top-k notifications are delivered under given mode by the user which is described in the Section 4.4.

3.3 Discussion

Here, we discuss the relationship between Boolean & Top-k publish/subscribe models based on the formal specification given earlier.

3.3.1 Relationship

Our study is being motivated by the theorem 3.3.1 which was proved by [35], where it showed that Top-k/w model is more general than Boolean publish/subscribe model.

Theorem 3.3.1. *The Boolean system is a special case of Top-k/w pub/sub system for every subscription $s \in S$, the matching function m_s is defined as follows:*

$$m_s(p) = \begin{cases} True\ or\ False. \\ \hat{m}(s,p) \geq 0 \end{cases} \tag{3.1}$$

where $\hat{m}(s,p)$ is the scoring function given a publication p in the Boolean system.

We can observe that Boolean model is a specialized form of Top-k publish/subscribe model where any Top-k model can be used as a Boolean model simultaneously in a single implementation. This observation is crucial for the future acceptance of Top-k publish/subscribe models.

3.3.2 Comparison

Publish/subscribe models are typically Real Time Systems (RTS) [1], where we can explore the general RTS properties between Boolean & Top-k models. We note that such systems exhibit two main properties *safety* & *liveness* [38].

Liveness Liveness property asserts that "something good will eventually happen". In Boolean models, this property is guaranteed where any matching publication which was published after relevant subscription becomes active for specific user, will be delivered, but not after client unsubscribes the former subscription. Our Top-k publish/subscribe also follow liveness property, but defining a subscription space instead of independent subscriptions as described in Section 3.2.1.1. Also in our model, there is a possibility not to deliver every matching publications if they fail to get a position in Top-k results before they expire.

Safety Safety property asserts that "something bad never happens". A matching publication should at least satisfy one subscription for the delivery to be completed successfully in previous models. Also some Boolean models do not allow a publication to be delivered to the same client more than once. Further, a publication should never be delivered to a client if it has not been active previously in time. As a generalized version of Boolean model, Top-k publish/subscribe depicts above characteristics but in different forms.

As an example, our model will deliver any publication within Top-k results if it satisfies at least one subscription tuple in the space, and a relevancy score ($r(p_i, X) > 0$). Also every publication delivered as Top-k result to a subscriber has to be within the current or previous subscription windows. System considers any publication to be expired, when it ensures that they will not be part of any future subscription windows. Also the relevancy score of any publication is employed with freshness.

As each subscription window is moving in time or order, the subscriptions can be thought of stateful filters which deliver k most important publications in a given time instance t. Overall Top-k publish/subscribe models are influenced by additional parameters.

[1] http://rtcmagazine.com/articles/view/100285

[39] specified 3 availability classes to model the nature of any publish/subscribe paradigm. Boolean models exhibit *0-availability* where publications are expired as soon as they're published by keeping non-persistence nature. By the nature most Top-k publish/subscribe models have $\triangle - availability$ where publications (in)directly expires after $\triangle$ time after they're published in a dynamic setting. We showed earlier that it's impractical to keep published information in Data Stream Processing Systems (DSPS) for an indefinitely long time ($\infty - availability$).

Chapter 4

Top-k Publish/Subscribe Model

4.1 Design & Architecture

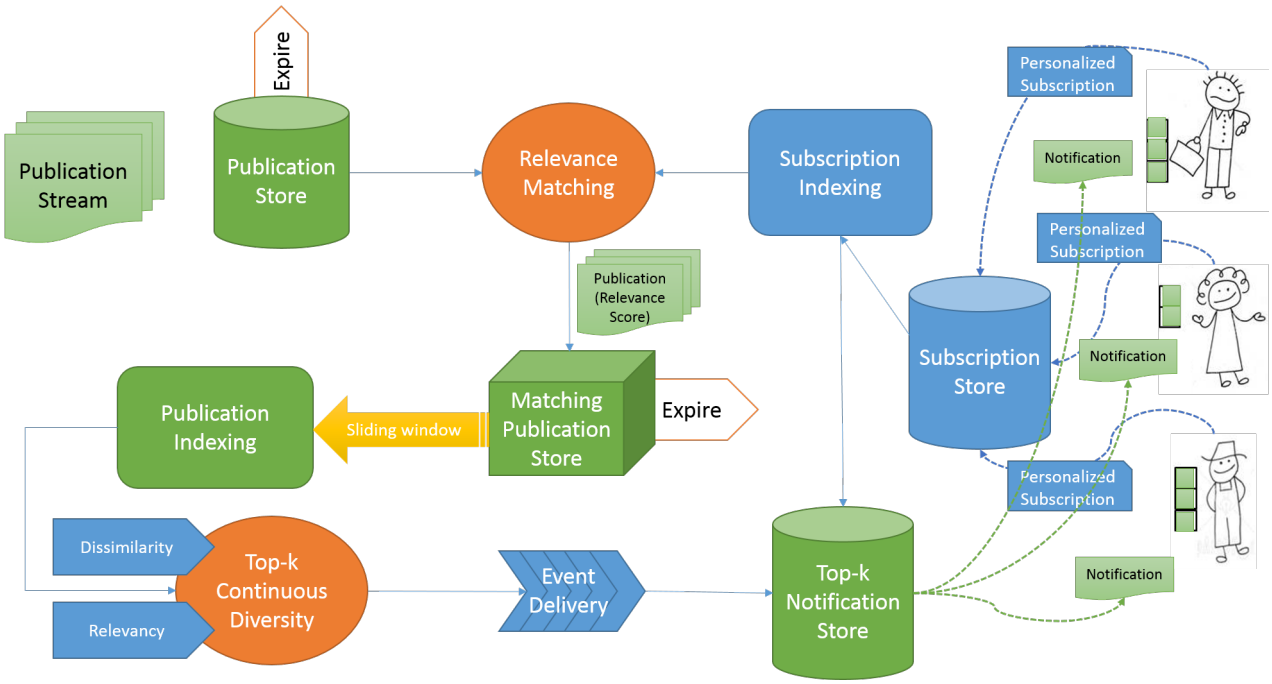


Figure 4.1: Top-k publish/subscribe architecture

In our design, we consider content-based publish/subscribe systems which offer greater expressiveness to subscriptions than topic or channel based ones. Also many researches believe that topic based publish/subscribe models are specialized forms of content-based models. At behavior level, our proposed Top-k publish/subscribe model can be specialized into traditional publish/subscribe models easily either based on topic or content. In the rest of our study, our Top-k matching model depends on the actual content of subscriptions & publications.

At the core of our proposed architecture, is a dual-indexing mechanism to partition the space in both subscriptions & publications. Additionally the spaces are dynamic in the way the subscriptions support a variety of attributes under a large volume of subscriptions and publications are streaming. The two proposed indexing mechanisms are complementary ones, rather than distinct. In addition, we propose a subscriber friendly scoring algorithm which is linked with above modules to measure the effectiveness of the system.

Since publications are ranked across the stream, we adopt the notion of sliding window to further maximize the diversity of delivered results continuously. The diversified results are incrementally computed over consecutive windows with the service of a personalized scheme. The system accepts preferential subscriptions from users to depict their interest over an ordered publication stream. We increase the stateful nature of the system by providing persistence subscription store along with a publication store where incoming publications are expired by the time to keep the freshness of the delivered output. We also maintain an active notification store for all subscribers, where Top-k publications are delivered as notification under given delivery method.

At more generic level, the model is designed to achieve the effective set of Top-k results incrementally over the stream by solving *continuous k-diversity problem* which is formulated later. The dual-indexing module is proposed to enhance the efficiency of the matching process.

4.2 Scoring Algorithm

In Top-k publish/subscribe models, publications trigger subscriptions based on a scoring algorithm. Most of the previous work have proposed a threshold or bound based schemes where a subscription will be triggered by a new publication, iff it scores more than a predefined threshold. Threshold is usually considered as the minimum score of Top-k publications previously published for a specific subscription [8]. In general, above schemes indirectly adopt ordered queue operations with threshold based replacement techniques.

Here, we don't rely on a threshold based matching, since it is biased towards highest-scored Top-k publications always. To overcome that problem, we compute a diverse set of publications which is scored based on query dependent metrics (e.g. Relevancy, Freshness). Additionally the parameter k restricts the delivery over the publication stream which is bounded by sliding windows along with the time. Diversified result set doesn't serve only top relevant elements but it also increases the freshness of them. Also delivered Top-k results have the tendency to

cover all matching publications.

We have studied our problem in a restricted dynamic setting where the publications to be selected as Top-k would change over time. Also subscriptions can be inserted or deleted off-line. Studying dynamic behavior of subscriptions is beyond our scope.

4.2.1 Relevancy: Query personalization

Subscriptions are used to filter relevant information while discarding irrelevant information. So they're queried to select only publications which satisfy the subscriptions. In traditional content-based pub/sub approaches, users can only express their interest by a set of predicates or expressions within the subscription. Hence, they consider the user interest over all subscriptions is distributed equally.

Here, we put our view forward to provide Top-k matching publications against a **subscription space**. We avoid using the traditional approach on keeping user subscriptions independently, which is tailor made for Boolean publish/subscribe matching models, but not for Top-k matching.

Any user may subscribe into more than one overlapping subscriptions in traditional models. By removing the redundancy of user subscription space, we can limit the number of matching publications. In recent literature, there are two most common techniques to reduce the subscription space, *subscription covering* & *subscription merging* [35].

Subscription Covering

Definition 4.2.1 (Subscription Covering)**.** A Boolean subscription is covered by another Boolean subscription if every publication which matches the former subscription also matches the latter subscription.

Remarks

- The fact that subscription s_1 is covered by subscription s_2 does not imply that the subscription s_2 is covered by the subscription s_1. Thus, subscription covering is non-commutative [Figure 4.2.a].
- Deciding whether a Boolean subscription is covered by a set of previously defined subscriptions was proven to be co-NP complete [35]

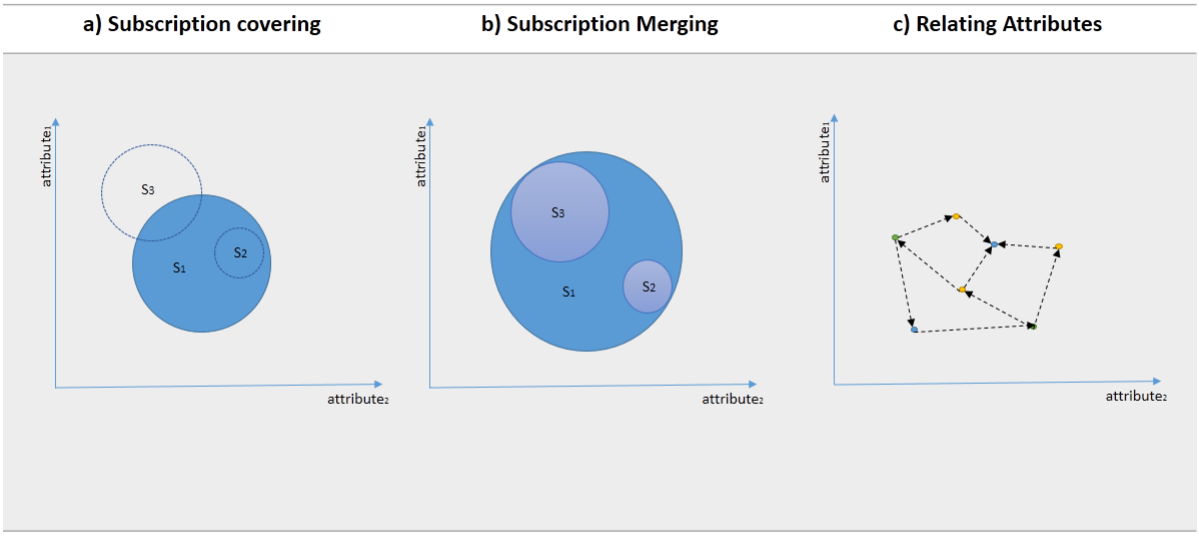


Figure 4.2: Subscription covering vs. Subscription Merging vs. Attribute Merging

- There is a high probability on adding redundant subscriptions. Thus, if a new subscription is already covered by the existing subscriptions, adding former is useless.
- When the subscriber removes all subscriptions that are covered by another subscription, the latter needs to update it's coverage, and the opposite. So the validity of defined relations between subscriptions, should be updated more often.

Subscription Merging

Subscription merging is a specialized form of subscription covering. [Figure 4.2.b]

Definition 4.2.2 (Subscription Merging)**.** Two or more Boolean subscriptions can be merged together in a broader subscription which then covers all of the original subscriptions.

Remarks

- Boolean subscriptions can be merged together in perfect or imperfect way. In perfect merging, any part of the resulting subscription is covered by original subscription, but in imperfect merging it may not.
- Imperfectly merged subscriptions may affect the accuracy of matching publications as they can fall into any sub-interest of space which is not covered by the original subscription.

If we allow preferential subscriptions to be covered by another, we would have defined a way to cover the defined relations which is parametrized by different k values. Along with the identified complexity over boolean subscription covering, we can observe that it won't provide any benefit in Top-k subscriptions. Even though we can reduce the subscription space, it may incur additional complexity level to our system because of maintaining additional parameters. Also we can not guarantee that matching publications could achieve the maximum representative power. Subscription merging is also questionable under above scenario as it was a specialized case of subscription covering.

4.2.1.1 Relating Attributes

In ranked publish/subscribe models, it's preferred to reflect a degree of user interest over subscription space either locally or globally [Figure 4.3]. Many research works did consider to employ preferences among subscriptions [17, 9, 10]. Preference aware subscriptions can be used to rank the publications & to deliver the user most preferred ones. In this way, preferences can be defined among subscriptions globally or among attributes in the particular subscription locally. Both qualitative & quantitative approaches to compare attributes or subscriptions have been tried out [10]. But still the user has to explicitly define the ordering between them.

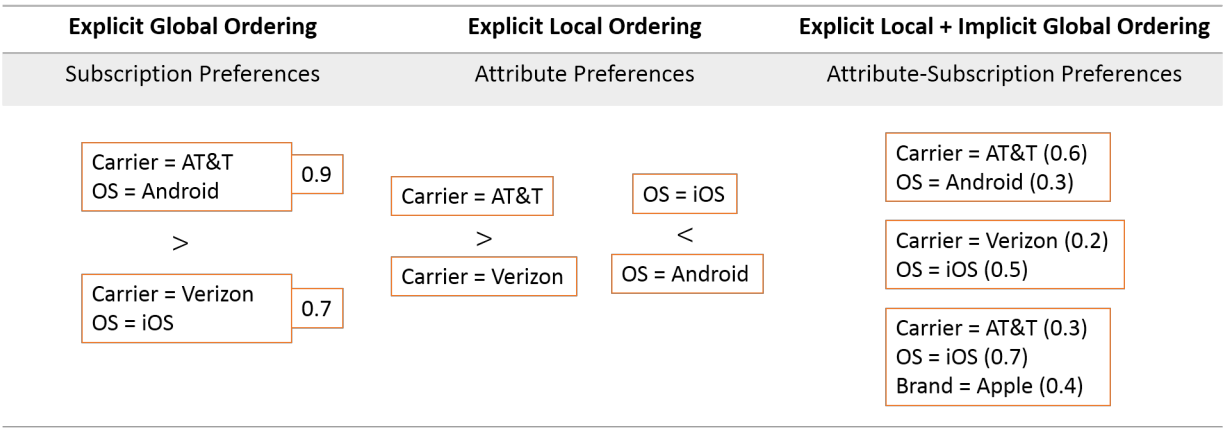


Figure 4.3: Subscriber preference models

We consider user subscription spaces rather than subscriptions independently. Subscribers can specify a degree of interest over the subscription space by defining relative preference score for each subscription tuple which they subscribe into. From that local ordering of subscription tuples, the model can define a global ordering in the subscription space which is aggregated by above tuples. The preference ordering is natural to be independent among subscribers. Also we're not restricted by particular DNF or CNF to define the subscription.

Our goal is to derive more preference relations among subscription tuples to increase the expressive power of user in the personalized subscription space. Hence, any matching publication which is getting scored against a satisfied subscription subspace, have much granularity look on what user preferred most. In this way, our model inherently supports partial matching between publications & subscriptions. This approach is subscriber friendly where publications achieve scores based on the preference relation of subscriptions. It is beyond our scope to take explicit publisher preferences into account for the matching process.

Constructing personalized subscription space The preference relation in an user subscription space is presented by a directed graph where nodes represent subscription tuples & the edges represent the relative preferences. Formally we declare the notion of personalized subscription graph at Definition 3.2.2. Also the graph is changing it's behavior dynamically when user updates the subscription space.

Example 4.2.1. Let's assume, Bob would like to get notified on products related to following personalized subscriptions

$$s_1 = \{carrier = AT\&T(0.4) \ \vee brand = HTC(0.3) \ \vee storage \leq 16GB(0.7)\}$$
$$s_2 = \{carrier = Verizon(0.5) \ \vee storage \leq 32GB(0.2)\}$$
$$s_2 = \{brand = HTC(0.3) \ \vee storage \leq 32GB(0.6)\}$$

using our Top-k pub/sub model. We say that Bob prefers products that satisfy the requirements of $storage \leq 16GB$ than $brand = HTC$ & $carrier = AT\&T$ based on the subscription s_1. The ratio of Bob's preference values at s_1 can be denoted by:

$$\textbf{preference ratio}(carrier = AT\&T : brand = HTC : storage \leq 16GB) = 0.4 : 0.3 : 0.7$$

Like that, we can define a ratio of preferences locally from other subscriptions s_2 & s_3 as well. Then, we can construct a local preference graph at each subscription & merge into global preference graph according to the Definition 3.2.1.

Figure 4.4 & 4.5 shows the virtual construction of Bob's subscription space using above dummy preference values.

When a seller publishes a random product p_i as a publication

$$p_i = \{carrier = AT\&T \vee brand = HTC \vee storage \leq 32GB \vee color = Black \vee OS = Android\}$$

to our matching model, the model locate satisfying subscription tuples at Bob's subscription graph as shown in Figure 4.5.

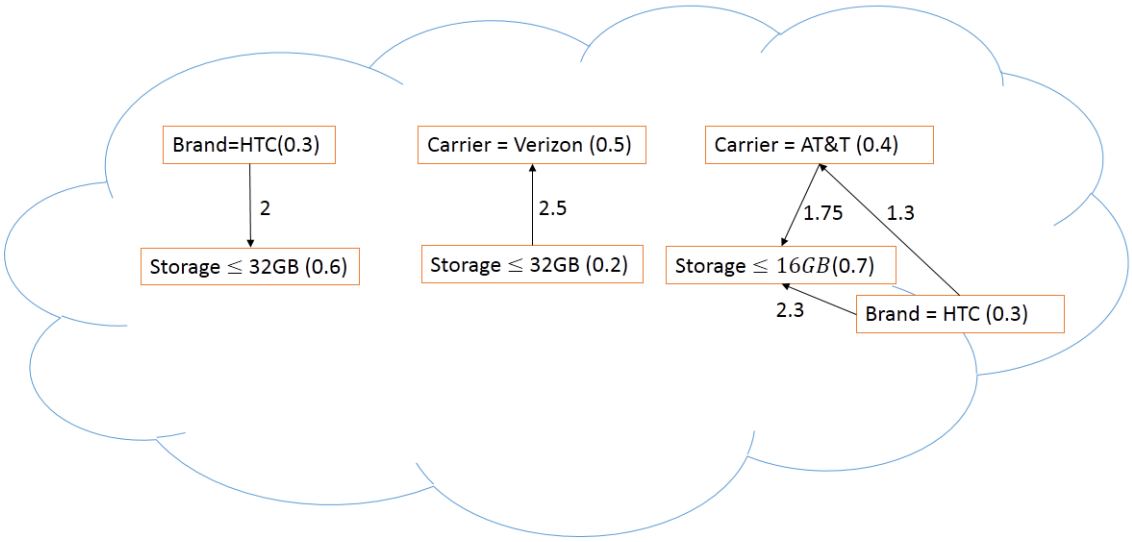


Figure 4.4: Bob's personalized subscription space

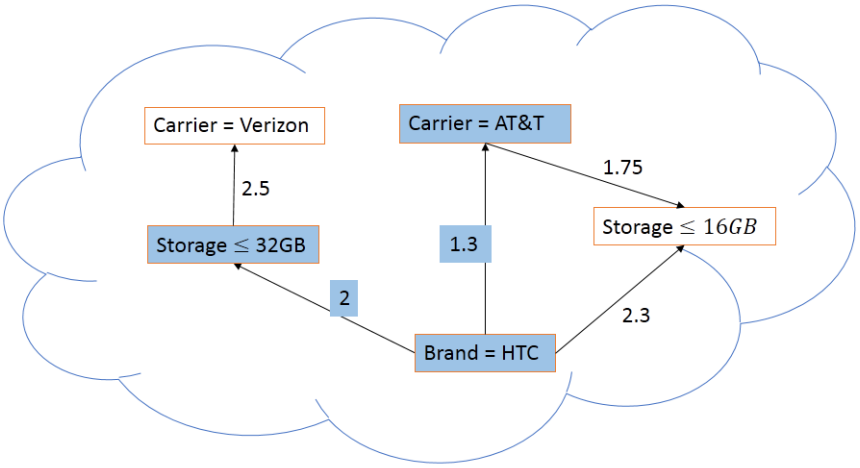


Figure 4.5: Satisfying subscription space

Algorithm A.1 shows a naive approach on constructing & updating a personalized subscription graph (see the appendix).

Next, we propose a scoring algorithm to compute relevancy score $r(p_i,\ Bob)$ for the publication p_i. Based on the score achieved by the matching publication, a decision will be made on next stages to detect the publication as a Top-k candidate or not.

4.2.1.2 Relevancy score

Definition 4.2.3. Let $G(V,E)^X$ be the personalized graph of user subscriptions submitted by user X, assume the sub-graph $G(V,E)^X_{p_i}$ $s.t. G(V,E)^X_{p_i} \subseteq G(V,E)^X$ is satisfied by the publication $p_i; p_i \in P$ and, the relevancy score

$$r(p_i, X)$$

depicts the personalized relevancy score of a publication p_i for a particular subscriber X. Any publication p_i is preferred to be matched for user X when $r(p_i, X) > 0$. All matching publications are considered as potential Top-k candidates.

Relevancy score can be computed as an aggregated score (e.g. sum, average) of $X's$ satisfied graph space. The satisfied graph space can be depicted by both satisfied edges & nodes. Because a publication may satisfy many subscription tuples but not many relations among them. The relevancy score of a publication $r(p_i, X)$ is defined by taking both these elements. They allow a finer granularity to represent the user at a matching publication from the relevancy score.

$r(p_i, X) = \lambda$**Average weight of satisfied relations (edges)** $+ (1 - \lambda)$**Average weight of satisfied tuples (nodes)**

Any publication with higher score implies that it is more relevant to particular user X. Algorithm A.2 computes the relevancy score given any publication (see the appendix).

Time complexity The algorithm A.2 takes $O(nD)$ time to locate the publication in a sparse graph of vertices n when the number of tuples or dimensions in the publication is D. It is also the upper bound to calculate the average weight of node preferences. To maximize the user representative power in the relevancy score, algorithm traverses the graph to calculate the average of satisfied edge weights. It takes $O(D^2)$ time to match all tuples. When a publication is matching completely with the user subscription space, the algorithm takes $O(nD + D^2)$ to deliver the relevancy score.

In the section 5.1, we extend the functionality of an efficient in-memory index (i.e. opIndex [31]) to reduce the matching time.

4.2.2 Events novelty: Freshness

Personalization becomes the most influencing factor so far to retrieve Top-k results, but we observe that older publications may prevent the newer publications from entering into Top-k results.

As a motivation, a popular news pub/sub system like Google news maintain publications within last 30 days, but most of the time produce top-k results within last day or two. This phenomena is not particular to Top-k pub/sub but for most DSPS in general [40]. Most common approach is to apply a function of *time decay* that is not new in temporal data analysis. It often uses based on different aging techniques in a wide range of data processing applications to produce a set of *fresh* results.

When dealing with continuous query processing systems, it's obvious to produce a stream of results. There has been a great deal of work on algorithms for efficiently answering streaming queries under complex time decay algorithms. But most of the algorithms failed to be scalable within streaming systems under sliding widows [40].

There are different metrics to measure the age of a publication. One straightforward way is to measure the age since the publication first appears in the system. In this approach, age is calculated back from the current time. But when updating the scores of continuous incoming publications in the query processing windows, we have to refresh & update the age of all relevant publications. Due to this high inefficiency, [40] proposed "forward age" of an item which is relative to a landmark.

Definition 4.2.4 (**Forward Decay Function**)**.** Given a positive monotone non-decreasing function g, and a landmark time L, the decayed score of a publication with arrival time $t_i > L$ which measured at time $t \geq t_i$ is given by,

$$u(t_i, t) = \frac{g(t_i - L)}{g(t - L)}$$

This also observes following conditions hold by the general decay functions.

(i) $u(t_i, t) = 1$ *when* $t_i = t$, *and* $0 \leq u(t_i, t) \leq 1;\ \forall t \geq t_i$

(ii) $t' \geq t \rightarrow u(t_i, t') \leq u(t_i, t)$

The natural phenomena of above aging technique is that once the age has been observed for a publication, then it is fixed.

The most natural choice of function g can be in the form of polynomial, exponential or logarithmic [40].

The selection of a suitable landmark, is dependent on the form of decay function g which is being applied. As an example, take the form g of exponential;

$$u(t_i, t) = \frac{g(t_i - L)}{g(t - L)}$$

$$u(t_i, t) = \frac{exp\ \rho\ (t_i - L)}{exp\ \rho\ (t - L)}$$

$$u(t_i, t) = exp\ \rho\ (t_i - t)\ where\ \rho > 0$$

So, at any given time t, the decay score of any publication can be depicted by $u(t_i, t)$ where $\rho > 0$ is the decay parameter:

$$u(t_i, t) = exp\ \rho\ t_i$$

We can observe that above *forward decay* function does only rely on the issued time of a given publication, but not on a landmark time L. That leads us to rely on exponential decaying function as it avoids the overhead of maintaining temporal properties of subscriptions, and also the recalculation of relative scores. Because as time grows, relative ranking among incoming publications does not change.

4.2.2.1 Fresh Relevancy score

To attach freshness into the relevancy, we use the forward decay score $u(t_i, t)$ as a multiplier of the relevance score of a publication.

Definition 4.2.5. Let P be stream of publications where (p_i, t_i) denotes a timestamped publication issued at time t_i, and given the forward decay score $u(t_i, t)$ given at the current time t and, the relevancy score $r(p_i, X)$ for the user X, the fresh relevancy function is,

$$r(p_i,\ t_i,\ X) = r(p_i, X).u(t_i, t)$$

The fresh relevancy score $r(p_i,\ t_i,\ X)$ depicts the relevancy of a publication which is influenced by freshness.

Remark: When a new publication arrives, we avoid updating the scores of all other publications within a sliding window since the relative ranking between them is fixed due to the forward decay function. When the window is moving forward, earlier publications are getting dropped due to temporal constraints.

4.2.3 Events Diversity

Instead of achieving pure personalized relevant results, one also attempt to increase the diversity of results to increase user satisfaction. Because getting similar Top-k results is a natural drawback in most personalized query engines. Usually this happens when the subscriptions don't express finer granularity. But getting a diverse set of results is proved to be a necessary condition to be effective in most information processing engines. We believe that our model can be enhanced further by applying diversity based methods to compute Top-k publications.

Since we support partial matching between publications & subscriptions, our model has the tendency to explore results in a wider spectrum. Hence, we don't support a strong distinguishability between finer & coarse granularity in the subscription themselves. Instead, our model selects a diverse set of results from the matching publications. Thus, we try to support diversity aware Top-k queries which maximize both diversity & relevancy of Top-k result set.

Example 4.2.2. As an example, let Bob likes to get notified about products from carrier=AT&T and brand=HTC. As Figure 4.6 shows, Top-3 results produced by Amazon fails to engage Bob, as it all contains information about "plastic covers". Because ideally Bob would like to get information about "smart-phones". Without the notion of diversity, delivered Top-k publications may have much similarity between them.

Figure 4.6: An example of redundant information in traditional Top-k answer set

This phenomena becomes worst when the publications are streaming. Even though, the received publications are personalized, Bob may recognize such a system as non-effective.

Most previous approaches to define diversity are based on: (i) (Dis)similarity (ii) Coverage (iii) Novelty [18], but sometimes combining more than one of them [20, 18]. In Top-k pub/sub context, diversity based on dissimilarity (i.e. delivered publications are dissimilar to each other), coverage (i.e. delivered publications can cover many interpretation of subscribers' information needs) & novelty (i.e. publications which are presented earlier has higher priority) are considered orthogonally in more general versions, but have not been mentioned on their dependency at applications. We believe that the definition of diversity is application dependent. Selecting

the right definition of diversity under e-commerce environment in Top-k pub/subs context will be one goal of our research output.

Generally, in any form of definition, the diversification problem has been shown to be NP-hard [9] and hence can only provide heuristics for its approximation. Our study is based on a specialized form of **k-diversity** problem which is aligned with *continuity* requirements introduced by [23] to deal with data-stream. Hence, we focus on **continuous k-diversity problem**.

4.2.3.1 k-diversity problem

Dissimilarity based k-diversity

Definition 4.2.6 (k-diversity problem)**.** Let P be the set of matching publications; $|P| = n$, and given a distance metric d to express the dissimilarity between publication points, finding the diverse set S^* of P such that:

$$S^* = arg\ max\ f(S,d);\ S \subseteq P;\ |S| = k;\ k \geq 0$$

Any distance function (e.g. L_r norm) can be used to calculate the distance between two publication points.

Three widely used functions (f) to aggregate the distances are MIN (i.e. minimum distance among selected items), SUM (i.e. sum of the distance among selected items) & AVG (i.e. average distance among selected items).

$$f_{MIN}(S,d) = min\ d(p_i,p_j);$$

$$f_{AVG}(S,d) = \frac{2}{k(k-1)} \sum_{i=1}^{k} \sum_{j>i}^{k} d(p_i,p_j);$$

$$f_{SUM}(S,d) = \sum\ d(p_i,p_j);$$

$$such\ that\ \forall p_i,p_j \in S;\ i \neq j$$

Above functions are known as *MAXMIN*, *MAXAVG* & *MAXSUM* in k-diversity problem. MAXMIN does select $S^*;|S^*| = k$; out of P given points, so that the minimum distance between any pair of chosen points is maximized [18]. Instead of minimum, MAXAVG selects $S^*;|S^*| = k$; out of P given points, so that the average distance between any pair of chosen points is maximized. In the same manner, MAXSUM selects $S^*;|S^*| = k$; out of P given points, so that

the sum of distance between any pair of chosen points is maximized.

The general solution for most of these criteria is NP-hard which is extensively studied as *dispersion problem* in the literature. Approximation algorithms have been developed & studied. We direct the reader to see [41] for the summary of results.

Coverage based k-diversity

Definition 4.2.7. Let P be the set of matching publications; $|P| = n$, given a subscription s, a taxonomy C & a probability distribution $P(\frac{c}{q})$ for each category $c \in C$ that is relevant to C, finding the diverse set S^* of P such that:

$$S^* = arg\ max\ P(\frac{c}{q})$$

With above view, at least one selected publication in S^* may cover the query relevant category c, so it maximizes the probability of former covering [18].

Novelty based k-diversity A precise distinction between novelty & diversity has been made by [25] in the context of IR systems. Novelty is used to resolve redundancy while diversity is to resolve ambiguity. In the context of Top-k pub/sub, the notion of novelty is used separately from diversity where to avoid the blocking of new publications to be delivered by old publications.

DisC diversity

Definition 4.2.8. Let P be the set of matching publications; $|P| = n$, given a neighborhood $N_\alpha(p_i)$

$$\forall p_i \in P, \exists p_j \in N_\alpha^+(p_i),\ s.t.\ p_j \in S\ where,\ N_\alpha^+(p_i) = N_\alpha(p_i) \cup \{p_i\}$$

$$and, \forall p_i, p_j \in S\ with\ i \neq j\ s.t.\ d(p_i, p_j) > \alpha$$

where the neighborhood $N_\alpha(p_i)$ of an object $p_i \in P$ is defined by the Definition 4.2.9

Definition 4.2.9 (Neighborhood)**.** Let the (dis)similarity between two publications p_i &p_j is defined by any distance metric $d(p_i, p_j)$, a neighborhood $N_\alpha(p_i)$ of the publication p_i for $\alpha > 0$ is represented by:

$$N_\alpha(p_i) = \{p_j | p_i \neq p_j \wedge d(p_i, p_j) \leq \alpha\} \alpha > 0$$

With above view, publications in the neighborhood $N_\alpha^+(p_i)$ of p_i are considered similar with p_i & covered by p_i [20].

4.2.3.2 Combining relevancy with diversity

We can address diversity through a different perspective by combining with the relevancy of Top-k results. Natural ways to combine relevancy with diversity to achieve a natural bi-criteria objective are followed [29]:

- *MAXSUMDISPERSION problem:* To maximize the sum of the relevance and dissimilarity of the selected set.
- *MAXMINDISPERSION problem:* To maximize the minimum relevance and dissimilarity of the selected set.

Above models can be re-casted in terms of a *facility dispersion* problem, [29]. We can map known heuristic based solutions to achieve above diversification objective.

Design philosophy The design philosophy is not to re-invent the wheel by looking for new and novel techniques to implement heuristic based solutions for *facility dispersion* problem. We have come up with another mono-objective formulation which does not relate to *facility dispersion* for combining relevancy with diversity.

4.2.3.3 Beyond Diversity & Relevance

Instead of selecting the importance of the whole set S^*, now we select a diverse set which increase the "global" importance of a selected publication & reduce the "global" importance of a non-selected publication. In the rest of our study, our definition of diversity will be called as $MAXDIVREL$.

Definition 4.2.10 (**MAXDIVREL k-diversity problem**)**.** Let P be the set of matching publications; $|P| = n$, and given the relevancy metric r depicts the relevance of a publication which is calculated according to the degree of user interest, a distance metric d to express the dissimilarity between publication points, s.t. $d(p_i,\ p_j) \leq \alpha$ where $\alpha > 0$ is the neighborhood parameter, and $\lambda > 0$ is a parameter that tunes the importance of diversification, finding the diverse set S^* of P such that:

$$S^* = arg\ max\ f_\alpha(S,\ d,\ r);\ S \subseteq P;\ |S| = k;\ k \geq 0;\ \alpha \geq 0$$

$$s.t.\ f_\alpha(S,\ d,\ r) = \lambda.\ \frac{g_\alpha(S,\ d,\ r)}{h_\alpha(S,\ d,\ r)}$$

$$where\ g_\alpha(S,\ d,\ r) = \frac{1}{|S|}.\sum_{p_i,\ p_j \in S} \frac{r(p_j)}{r(p_i)}.d(p_i, p_j)\ and,$$

$$g_\alpha\ holds\ independence\ condition:\ \forall p_i, p_j \in S, d(p_i, p_j) > \alpha$$

$$and,\ h_\alpha(S,\ d,\ r) = \frac{1}{|P-S|}.\sum_{p_i \in S,\ p_j \in (P-S)} \frac{r(p_j)}{r(p_i)}.d(p_i, p_j);$$

$$h_\alpha\ holds\ dominance\ condition:\ \forall p_i \in P, \exists p_j \in S\ s.t.\ d(p_i, p_j) \leq \alpha; i \neq j$$

and, the diverse set of S^* holds *dominance* & *independence* conditions.

dominance condition ensures that all publications in the set P are represented by at least one similar publication in the selected set S^* and, *independence* condition ensures that the publications in the selected set S^* are dis-similar to each other. Ideally, MAXDIVREL produces the smallest subset of results bounded by α neighborhood which have the maximum diversity along with the relevancy as a factor.

Demonstration Let us demonstrate the behavior of MAXDIVREL k-diversity problem using Figure 4.7. Let's assume that we can map the set of publications P into 2-dimensional space by considering their locality. The relevancy score $r(p_i)$ is already assigned with each data-point where high scores are more important to user. Dissimilarity between two data-points can be calculated by any distance metric (e.g. L_inorm) from the 2-dimensional space. To hold dominance condition, any two data points are considered as similar if they satisfy $d(p_i,\ p_j) \leq \alpha$, where $\alpha > 0$.

We can pick k diverse set of publications (red colored points) by dropping $n-k$ publications (blue colored points). Based on $MAXDIVREL$, we drop $n-k$ publications which are less relevant than the selected set and also much similar to each other. By adopting neighborhood based Top-k computation, we guarantee that any two publications in the selected set are minimum α distance away to hold independence condition. That guarantees to select k diverse set of publications which are not similar. Inherently, Top-k results can represent all n publications based on relevancy & diversity.

MAXDIVREL k-diversity problem can map into *independent dominating set* problem in graph theory which has proven to be NP-Hard[1].

[1]http://en.wikipedia.org/wiki/Dominating_set

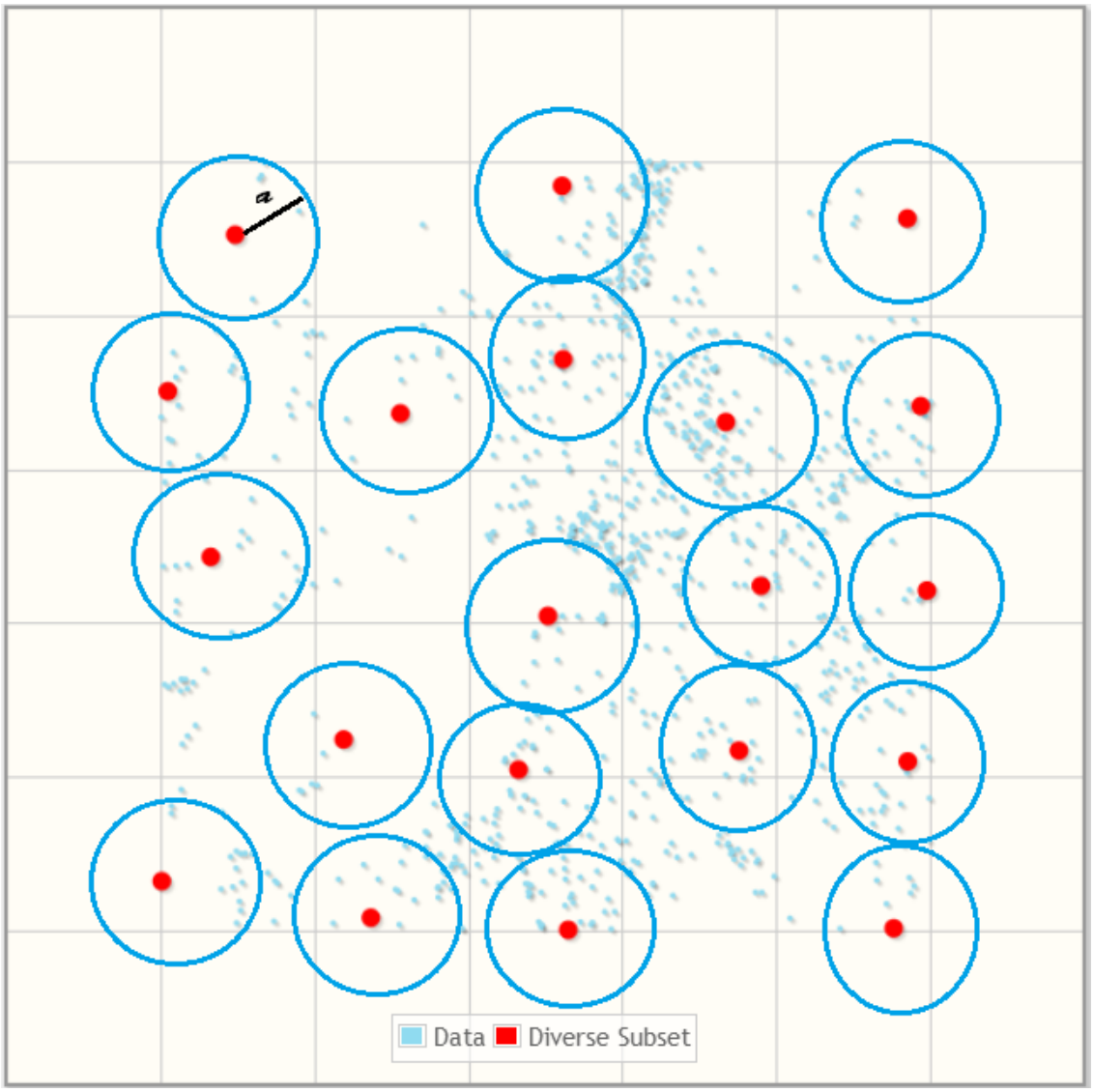


Figure 4.7: MAXDIVREL demonstration using dummy publications

4.2.3.4 MAXDIVREL NP-Hardness

Let's consider a set of publications P represented by an undirected graph $G_{P,\alpha}(V, E)$ such that each vertex $v_i \in V$ there is a publication $p_i \in P$ and an edge $(v_i, v_j) \in E$ iff $dist(p_i, p_j) \leq \alpha$ for a size α of neighborhood.(Figure 4.8)

Definition 4.2.11 (Dominating set)**.** A dominating set for a graph $G(V, E)$ is a subset $D \subset V$ where every vertex not in D is adjacent to at least one vertex of D (Figure 4.9). Dominating set problem tests whether there is a minimum dominating set D such that $|D| \leq k$ for the given parameter k. The decision problem of minimum dominating set is NP-Complete.

Definition 4.2.12 (Independent set)**.** The independent set for a graph G is a set of vertices when there is no edge connecting them. It is observed that the independent set is also the

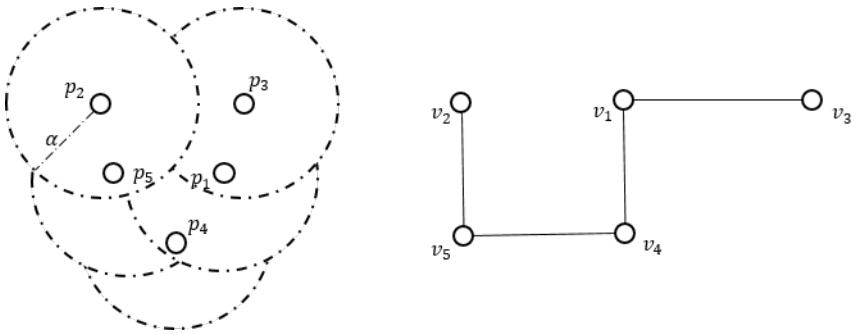


Figure 4.8: Graph representation of publication space with neighborhood α

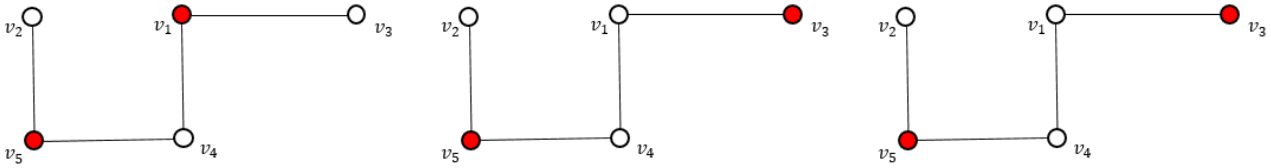


Figure 4.9: Dominating sets (red points)

dominating set iff it's a maximal independent set[1]. An independent set is maximal when it is not a subset of another independent set for a graph G. As Figure 4.10 shows, a dominating set for a graph may not be independent.

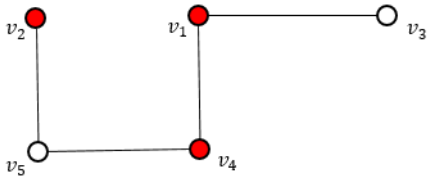


Figure 4.10: Dominating set but not independent

A set of MAXDIVREL k publications are independent because $dist(p_i, p_j) > \alpha$ where $0 < \alpha < dist_{max}(p_i, p_j) \forall p_i, p_j \in P$. Also a publication can cover all publications in it's neighborhood. So we can have following observation.

Observation 4.2.1. *Solving $k - MAXDIVREL$ diverse set problem for a set of publications P with the neighborhood size α is equivalent on finding a $k -$ independent dominating set for the graph $G_{P,\alpha}$. The decision version of this problem is to check whether there exists a minimal independent dominating set with size k.*

Exponential algorithms that solve minimum independent dominating set problem in a sparse graph $G(V, E)$ has shown not to break the trivial $O(2^{|V|})$ bound. It is shown that this problem has unlikely to be approximated within $|V|^{1-\epsilon}; 0 < \epsilon < 1$ [42].

[1] http://en.wikipedia.org/wiki/Dominating_set

A naive greedy algorithm to solve MAXDIVREL k-diversity problem A naive greedy algorithm to compute k MAXDIVREL subset, is shown below (Algorithm 4.1). Let's assume a publication $p_i \in P$ may have a α distance neighborhood to cover $\exists p_j \in P; i \neq j; s.t.\ d(p_i, p_j) \leq \alpha n = |P|;$. Also we consider the case where the conditions of dominance & independence need to be satisfied to deliver k-MAXDIVREL publications.

Algorithm 4.1 Naive algorithm to solve MAXDIVREL k-diversity problem

Input: A set of P publications given relevancy score $r(p_i); \forall p_i \in P$ with a size α of neighborhood
Output: A set S with k MAXDIVREL diverse publications P

1: Initiate $S \leftarrow \emptyset$;
2: **for** $\forall p_i \in P$ **do**
3: $\quad color(p_i) \leftarrow white$
4: **end for**
5: **while** $\exists p_i \in P\ where\ color(p_i) = white$ **do**
6: $\quad p_i^* \leftarrow arg\ max\ \frac{r(p_i)^2}{\sum_{p_j \in N_\alpha(p_i)\ \wedge\ color(p_j)=white} r(p_j) \times d(p_i, p_j)}$
7: $\quad S \leftarrow S \cup p_i^*$
8: $\quad color(p_i^*) \leftarrow black$
9: $\quad$ **for** $\forall p_j \in N_\alpha(p_i)$ **do**
10: $\quad\quad color(p_j) \leftarrow grey$
11: $\quad$ **end for**
12: **end while**
13: return S

Above naive algorithm 4.1 proceeds to select a subset S greedily from a set of P publications. The algorithm uses a color marking scheme where the publications in S are marked as black. Other publications remain white until they are covered by a publication $p_i \in S$. Then they are colored as grey.

Initially S is empty and, an arbitrary publication is selected from the set P of publications. Then the algorithm steps to select a highly relevant & diverse set of publications by computing MAXDIVREL heuristic (line 6) until all publications are visited. Note that, it doesn't require the parameter k, but the neighborhood parameter α. We can tune the parameter α until $|S| \geq k$.

The algorithm satisfies dominance condition because any publication within S, can cover a subset of grey colored publications. To hold above condition perfectly, there should be no white colored publications at the end. Any addition of grey colored publication to the selected set S violate former condition. Also it ensures that independence condition is not violated among the selected set S of publications. Because the algorithm does only consider white colored neighborhood for the heuristic evaluation.

Time complexity The algorithm requires $O(k(Dn)^2)$ time to compute the set $S; |S| = k$ of D-dimensional publications, since it has to locate all neighborhood by walking through all publications.

An index-mechanism to group similar publications is presented in Section 5.2 to enhance above naive process under both static & dynamic publication space.

4.2.3.5 MAXDIVREL continuous k-diversity problem

Definition 4.2.13. Let $P = \{P_j, ..., P_1\}$ be a stream of publications grouped into corresponding sets over sliding windows $w = \{w_i, ..., w_1\}$, be any two consecutive windows w_{i-1}, w_i and S^*_{i-1} be the diverse subset of P_{i-1}, selecting the diverse subset S^*_i of P_i such that at window w_i such that $P_{i-1} \cap P_i \notin \emptyset$ to satisfy MAXDIVREL k-diversity problem at each instance,

$$S^*_i = arg\ max\ f_\alpha(S_i,\ d,\ r);\ S_i \subseteq P_i;\ |S_i| = k;\ k \geq 0;\ \alpha \geq 0$$

$$s.t.\ f_\alpha(S_i,\ d,\ r) = \lambda.\ \frac{g_\alpha(S_i,\ d,\ r)}{h_\alpha(S_i,\ d,\ r)}$$

$$where\ g_\alpha(S_i,\ d,\ r) = \frac{1}{|S_i|}.\sum_{p_i,\ p_j \in S_i} \frac{r(p_j)}{r(p_i)}.d(p_i, p_j)\ and,$$

$$g_\alpha\ holds\ independence\ condition:\ \forall p_i, p_j \in S_i, d(p_i, p_j) > \alpha$$

$$h_\alpha(S_i,\ d,\ r) = \frac{1}{|P_i - S_i|}.\sum_{p_i \in S_i,\ p_j \in (P_i - S_i)} \frac{r(p_j)}{r(p_i)}.d(p_i, p_j);$$

$$h_\alpha\ holds\ dominance\ condition:\ \forall p_i \in P_i, \exists p_j \in S_i\ s.t.\ d(p_i, p_j) \leq \alpha; i \neq j$$

and, the diverse set of S^*_i holds *dominance* & *independence* conditions. where it also must satisfies the continuity conditions defined by *durability* & *ordering*.

Continuity requirements In continuous data delivery, we should have a set of requirements to achieve, that may depict the effectiveness of the system in the long run [23].

(i) **Durability**: We need to avoid uncertainty of appearance of highly diversified items on each window. Thus, an item is selected as diversified in i^{th} window may still have the chance to be in $(i+1)^{th}$ window if it's not expired & other valid items in $(i+1)^{th}$ window fail to compete with it.

(ii) **Order** The order on how we compute the set of Top-k items will follow the chronological order. When the matching publications are timestamped, we avoid the selection of item j

as diverse later, when we already selected an item i which is not-older than j. To consider causal order & other complex relations among publications is beyond our scope.

Sliding window approach Since this is an instance of continuous data delivery, we need to address the issue of, when we apply MAXDIVREL diversity. In recent Top-k publish/subscribe literature, *sliding window* concept has been emerged without the loss of generality of the publication stream [6, 21].

In the rest of our study, we rely on forward sliding event windows as a time independent approach where items are only forwarded. This approach can be dynamically parametrized further by the event arrival rate. With sliding-window processing, the k most diverse items are computed over sliding windows of length w based on MAXDIVREL.

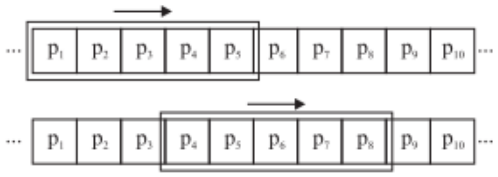


Figure 4.11: Sliding window approach: $w = 5$

We consider the case where the set of personalized relevant publications P changes over time. That causes to update the previously computed top-k results effectively for each subscriber continuously. We already showed that solving MAXDIVREL diversity problem over static publication space is NP-Hard. It keeps MAXDIVREL continuous k-diversity problem in the same family.

Uniqueness MAXDIVREL is different from previous Top-k bounded diversification methods which mostly rely on associating a diversity score with each object in the result. Then highest ranked objects are selected above some threshold. But they haven't considered them under a dynamic setting where the publication space may change which results need to be updated frequently. Also MAXDIVREL is a method of diversity that is aware of the relevancy. We're motivated by most recent works [20, 43, 44] to diversify Top-k results based on a neighborhood based method. But we do concern to maximize the representative power of Top-k results by considering both selected (S) & non-selected ($P - S$) sets.

We can observe that an addition of single publication to the set P or removal of any publication from the set P may result a completely different set of diversified items in the worst

case. As an example Figure 4.12 shows an instance of update to the k-independent dominating set($k = 2$) after the arrival of publication p_6.

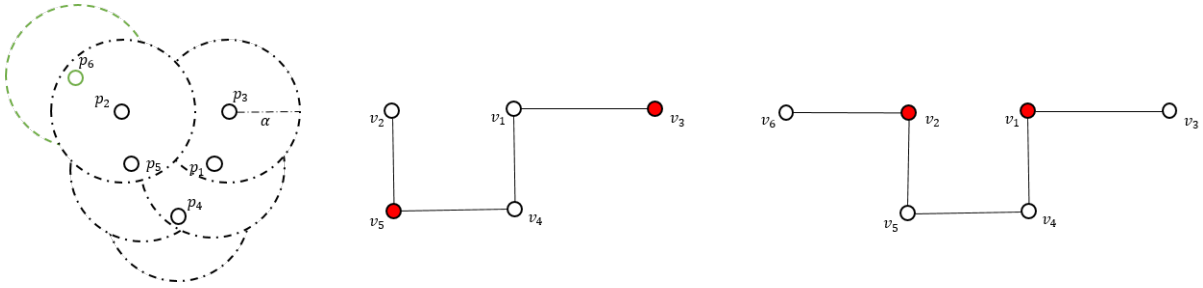


Figure 4.12: An update of k-independent dominating set after an arrival of new publication p_6; k=2

Above observation is crucial & leads to have efficient incremental algorithms. Because the straightforward solution is to apply any greedy algorithm that solve MAXDIVREL k-diversity problem at each sliding window instance by considering the publication space is static. But due to highly inefficiency of above straightforward solution, we propose an incremental approximation algorithm to retrieve k diversified results on each window.

In streaming windows, the performance bottleneck resides when locating the neighborhood. We propose an index-based approach using randomized algorithms at the section 5.2 to efficiently retrieve Top-k MAXDIVREL publications incrementally.

4.3 Timing policies

Here we show some pros & cons on timing policies (i.e. continuous, periodic, sliding window) to compute Top-k results.

- **Continuous:** We can rank results immediately after a new matching publication arrives & compute Top-k results. When publications are constantly produced, older publications may prevent new ones from reaching the user. We can associate an expiration time with each publication in the way that valid publications are not expired ones. In this way, older publications which have expired do not prevent new ones from reaching the users. But take a situation where publications are produced in the ascending order of their computed ranks, then all will be delivered, but in reverse only a part of them will.

- **Periodic:** Time can be divided into chunks of periods T, and matching publications are ranked in each period. We can embed an expiration time as before to overcome the problem in continuous Top-k computation. But take a situation, where only higher-ranked

publications produce in a given period, then some of them may be dropped due to the competition. But when lower-ranked publications are produced constantly in the given period, they will be Top-k candidates irrespective to the future higher rank publications.

- **Sliding Event-window** We can rank matching publications bounded by a sliding window to overcome above problem in periodic delivery. To deal with more general data streams, we consider sliding event windows instead of sliding time windows. [Figure 4.11]. Also, we only consider the window to move forward. Once we can not compute Top-k results at a particular window (s.t. $k < w$), we can wait for next windows to pick Top-k.

Jumping windows Our model also supports *jumping windows* where publication window moves forward by more than one position over the stream each time. The number of publications within a count-based window is w.

Definition 4.3.1 (Jumping step)**.** Given two consequent windows i & j where P_i defines the set of all publications within the i^{th} window and, P_j defines the set of all publications within the j^{th} window, the jumping step of the window i forward can be defined as;

$$jumping\ step(w_i) = |P_i| - |P_i \cap P_j|$$

$$given\ |P_i| = |P_j| = w$$

$$jumping\ step(w_i) = w - |P_i \cap P_j|\ where\ 0 < |P_i \cap P_j| <\ w$$

Observe that when $jumping\ step = w$, windows are disjoint, hence does exhibit the behavior of periodic Top-k computation. When $jumping\ step = 1$, the window is sliding forward by one position in the stream. Any publication in the stream is expired when the system ensures that they are not available for any future jumping windows (Figure 4.13).

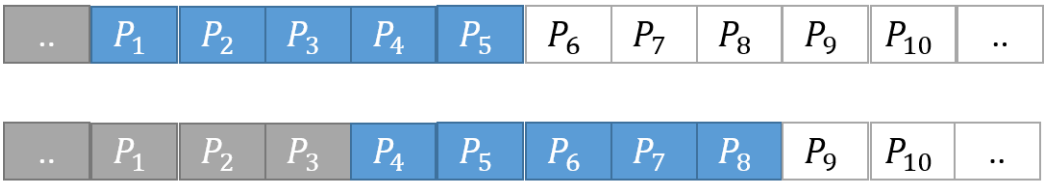


Figure 4.13: A jumping window with $w = 5$ & $jumping\ step\ = 3$

Any publication is guaranteed to fall into n number of jumping windows where,

$$1 \geq n \geq w\ -\ jumping\ step$$

before it's expired given a fix *jumping step* & window size across an unbounded publication stream. So a publication has a maximum n number of chances to become a candidate for Top-k delivery.

We maintain a delay inherent buffer between stream source and workers that are computing ranked publications. Incoming publications are buffered unless they're not accepted by the matching process. Buffer is continuously fetched by the worker for Top-k matching process. For real-time processing, the buffer should be empty at the beginning of τ time-intervals.

Compute Top-k publications For every subscriber, we need to maintain a list of Top-k publications at any given time. Incoming publications are ranked at each instance of sliding window over the stream. In count-based sliding window approach, only the previous w publications reside at the worker. When new publications arrive in the next window, current list of Top-k publications are updated by only considering non-overlapping publications.

Relation between μ & Top-k To compute Top-k results, there should be sufficient matching publications within a window. But we can not assume that is a necessary condition to be satisfied always in real world. Hence, we analyze some extreme cases that Top-k matching process needs to be aware of.

Let μ be the average number of publications occur within a given time interval τ where $w = \mu$ is the number of publications within a count-based window, such that the number of matching publications $\mu_{Matching}$ does always satisfy $\mu_{Matching} \leq \mu$

i $\mu_{Matching} \leq k$: The number of $\mu_{Matching}$ matching publications within a window is not sufficient to compute Top-k results, hence the model waits until a number of consequent windows to compute a set of Top-k publications.

ii $\mu_{Matching} > k$: Most natural case where Top-k publications can be filtered from all matching publications $\mu_{Matching}$.

4.4 Events Delivery

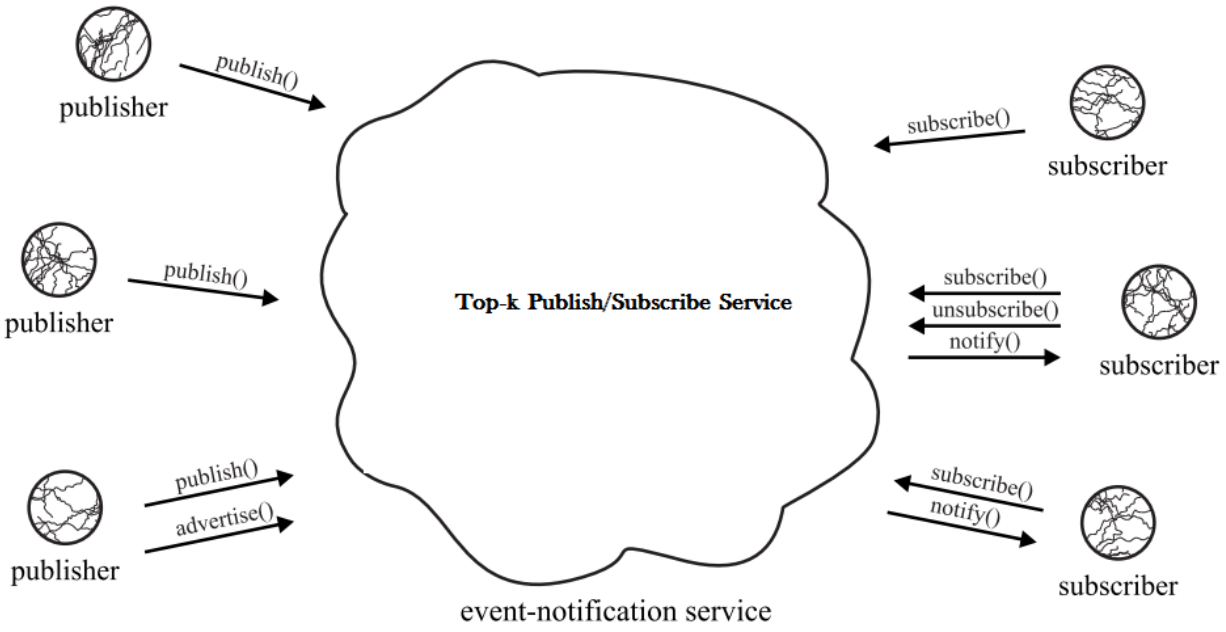


Figure 4.14: Centralized Top-k publish/subscribe architecture

The architecture of our Top-k publish/subscribe system is centralized where all subscribers & publishers access a centralized (cloud based) publish/subscribe service (Figure 4.14). Since subscribers do not have same preference over the subscription space S, we do maintain separate personalized subscription spaces pS^X for every subscriber X.

We do not deliver the list of Top-k publications to a subscriber as soon as they're updated at consecutive sliding windows. Our model does support two delivery methods such as:

i Pro-active delivery

ii On-demand delivery

In *pro-active delivery*, subscriber can define a period of time-units to receive a list of Top-k publications, or he can ask the system to present the current list on-demand. Since we assume the number of incoming publications does follow a Poisson random variable, our model always has the tendency to deliver a complete list of Top-k matching publications as they're requested. Also we guarantee not to send duplicate Top-k results that have been already delivered to an user by keeping a list of states.

Quality of delivery All publications that become Top-k candidates for any user may not be delivered. Because it might be replaced by another in the delivery time. Hence, we discuss the quality of delivered publications by taking following scenario.

Example 4.4.1 (Example scenario)**.** Let's assume a publication p_i had the chance to be in the Top-k list for an user X at a given time t_i. But when the list of Top-k publications are delivered for an user X at time $t_j; t_j > t_i$, p_i is not a part of it.

At an extreme case, let $r(p_i, X)$ & $r(p_j, X)$ be the fresh relevancy score of both publications p_i & p_j consecutively. If we further assume $r(p_i, X) > r(p_j, X)$, then the publication p_i can better represent the user intent. But due to diversification of Top-k results, publication p_j is considered to be better than p_i because p_j can represent the most updated publication stream at time t_j better than p_i.

Chapter 5

Indexing

In this Chapter, we show our dual-indexing mechanism in both subscription & publication spaces. We analyze different space-cutting data structures which are more efficient for our design & architecture of the system.

5.1 Subscription Indexing

In our model, we consider a subscription space per user which expresses a specific user interest over the global subscription space by all users. But computing the relevancy function over large subscription space may increase the processing time of ranked results. Note that, our mechanism should be scalable under variety of attributes to deal with natural phenomena in e-commerce domain.

In traditional publish/subscribe models, we need to locate at least one subscription, for matching to be completed successfully. Recent literature in publish/subscribe have proposed many indexing structures which addressed above scenario of locating relevant subscription. But here we need to locate all partially matching subscriptions in the space to compute the relevancy of a publication. So we adopt a novel indexing method called OpIndex [31] which was introduced in state-of-the-art publish/subscribe to extend it's functionalities to suit with our preference relation model.

5.1.1 OpIndex

OpIndex [31] organizes the subscription predicates into disjoint subsets each of which is independently and efficiently indexed to minimize the number of candidate subscriptions accessed

for event matching. Also it outperforms other competing index based methods(e.g. k-index [12], BE-Tree [30]) based on performance metrics like index construction time,memory cost and query processing time. OpIndex is built over inverted-list: a data-structure which is extensively studied over decades. Additionally, space cutting technique (i.e. two level partition scheme) used at OpIndex, is specifically built to handle *variety* of data that is natural under e-commerce along with the *volume* of subscriptions & high arrival rate of publications. As most other pub/sub indexing mechanisms can't cope effectively on the variety of data, we believe OpIndex is the right choice by it's evaluation which was done using comprehensive space & time complexity analysis.

In the first level, OpIndex selects a pivot attribute by modeling it as visibility minimization problem [31] for each subscription, and subscriptions with the same pivot attribute are grouped together. In the second level, subscriptions are further partitioned based on their predicate operators.

Level 1: Attribute partitioning Here, we demonstrate level 1 partitioning based on subscriptions presented at Table 5.1.

ID	Subscription
S_1	$carrier = AT\&T \ \wedge brand = HTC \ \wedge storage \leq 16GB$
S_2	$carrier = Verizon \ \wedge storage \geq 32GB$
S_3	$brand = HTC \ \wedge price \leq \$300$

Table 5.1: Example subscriptions

OpIndex uses *pivot* attributes to partition the subscriptions into independent posting lists. To select a pivot attribute per subscription, OpIndex uses following proved lemma 5.1.1.

Lemma 5.1.1 (Pivot Attribute selection). *Given a stream of publications E, using* $\delta_S = arg_{A \in S} min \triangle(A)$ *to select pivot attributes for partitioning subscriptions minimizes the number of candidate matching subscriptions accessed to match the publications in E*

$\triangle(A)$ denotes the frequency of an attribute A in a stream E of publications. OpIndex choose attribute A to be the pivot attribute for a subscription S if A appears the least frequently in E among all the attributes in S.

Let *carrier* & *brand* be the pivot attributes for the subscriptions at Table 5.1. Then, we have two posting lists as depicted in Figure 5.1

Subscription	Pivot Attribute
S_1	*carrier*
S_2	*carrier*
S_3	*brand*

Table 5.2: First level partitioning of subscriptions

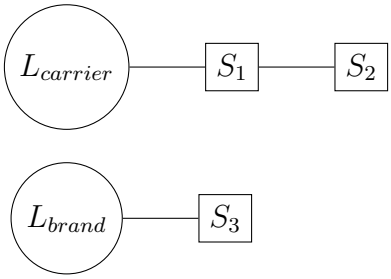
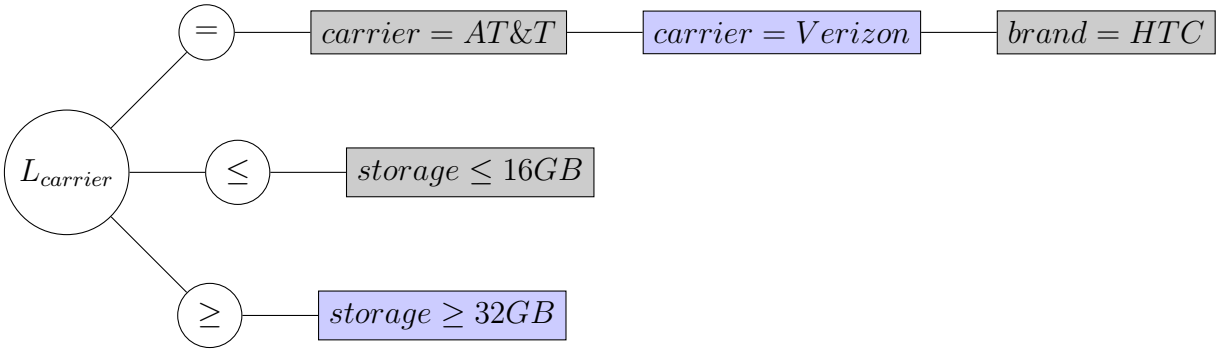


Figure 5.1: Attribute Lists

Level 2: Operator partitioning Each attribute list ($L_{(\delta_S)}$) is organized as an inverted-list structure based on the predicate operator into operator lists ($L_{(\delta_{(S,f_{op})})}$). Operator lists are sorted by the attribute and ties are broken by the comparison of value. Given on the operator in the publication tuples we can perform equality or range searches to locate the relevant subscription in the operating list.(Figure 5.2)

Figure 5.2: Operator Lists on $L_{carrier}$



5.1.2 Modified OpIndex

From OpIndex, we adopt only the concept of two level partitioning using inverted-lists. Because OpIndex was designed to deal with Boolean publish/subscribe model, it's capable enough to locate at least one matching subscription when a publication arrives. But as we described in the Section 3.3.2, we don't rely on individual subscriptions. So we need to partition each subscription tuples and, relate the inverted-list of predicates based on user given preference. Instead of a set of inverted-lists, our model generates an inverted-graph for each user subscription space.

Here, we demonstrate the two-level partition scheme we have proposed on personalized subscriptions at Table 5.3. Note that preferences are just dummy values.

ID	Subscription
S_1	$carrier = AT\&T(0.4) \ \vee brand = HTC(0.3) \ \vee storage \leq 16GB(0.7)$
S_2	$carrier = Verizon(0.5) \ \vee storage \geq 32GB(0.2)$
S_3	$brand = HTC(0.3) \ \vee price \leq \$300(0.6)$

Table 5.3: Example personalized subscriptions

Since we're not restricted to keep the subscription itself, we don't need to use any pivot attribute to partition subscription tuples at the initial level. Instead every unique attribute in the subscription tuples are used to partition the space. It's been motivated by following natural observation presented at e-commerce domain [1]:

– In a given user subscription tuple space, the number of unique attributes is less than the number of unique operands or values that are assigned to.

Example 5.1.1. Attributes (A)=$\{carrier,\ brand,\ storage,\ price\}$ where $|A| = 4$; Values (V)=$\{AT\&T,\ HTC,\ 16GB,\ Verizon,\ 32GB,\ \$300\}$ where $|V| = 6$ such that $|A| \leq |V|$;

In the second phase of partitioning, available operators within the attribute lists are used to repeat the partitioning as earlier. (Figure 5.3)

5.1.2.1 Index construction

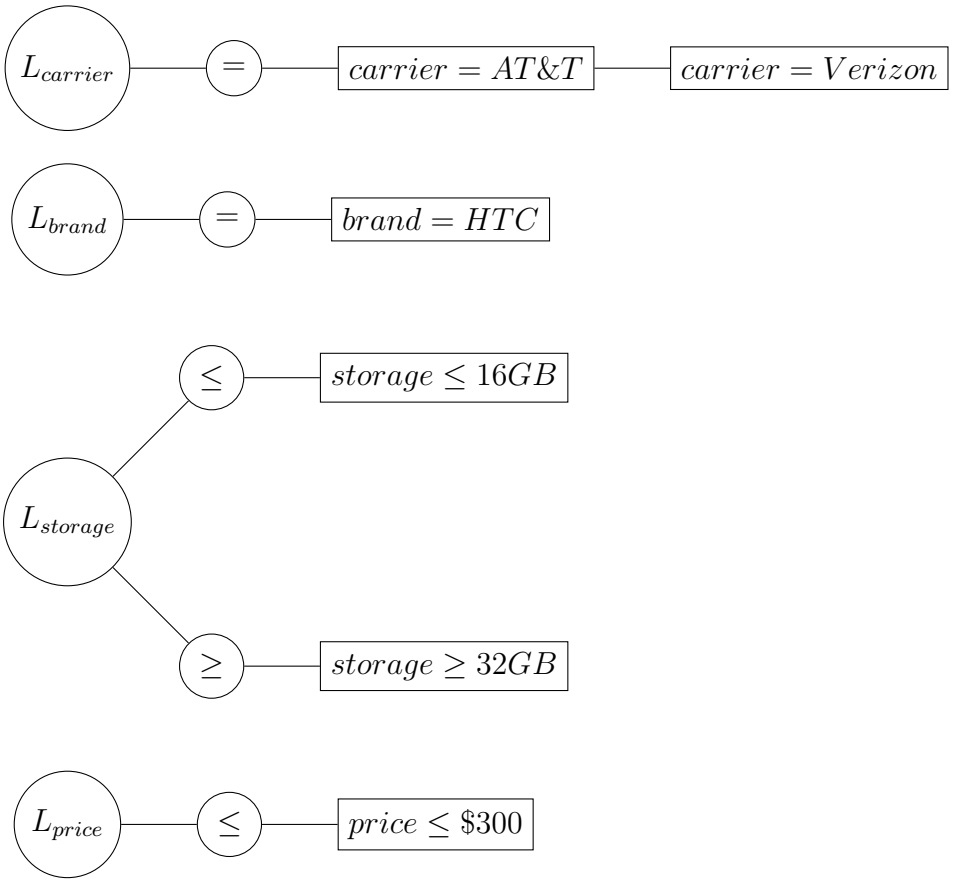
Algorithm B.1 depicts how modified opIndex is being updated when new subscription arrives (see the appendix).

Time complexity In worst case, the upper bound of any insertion is $O(n.log\ n)$ when the number of predicates is n. Index has been traversed by predicates in both attribute & operator posting lists to be located. It only takes $O(1)$ time locate the list & $O(log\ n)$ time to insert the predicate in order.

Space complexity Modified opIndex consists of two posting lists for attribute & operator partitioning. They are shared among N subscriptions. If the average size of unique attributes is n_{avg}, per subscription & the size of operators is constant c, the space requirement of modified opIndex is $O(cn_{avg}.N)$.

[1] http://storecoach.com/ecommerce-glossary/attribute/

Figure 5.3: Inverted-list two-level partitioning on user subscription space



5.1.2.2 Matching

Note that our goal is to maximize the representative power of user subscription space, to increase the importance of a matching publication. Since we have the structural view of our user subscription space, now we can derive the preference graph by assigning relative user preferences as weights to align with our personalized subscription graph(Definition 3.2.2).

Figure 5.4 depicts the personalized OpIndex that is demonstrated using preference values in Table 5.3. By using above inverted-graph we can compute the relevancy score of a publication without any hassle. We can use the same notion of Algorithm A.2 for the matching process but computation is faster since vertices can be looked up by $O(1)$ time. Algorithm B.2 computes the relevancy score of a matching publication (see the appendix).

Time complexity Recall that the naive algorithm A.2 takes $O(nD + D^2)$ time to compute the relevancy score of a given publication at $D-dimension$. By using the algorithm B.2, it only takes $O(D)$ time to locate the publication in a sparse graph of vertices n. Because using the inverted-index, the publication tuples can be located at constant $O(1)$ time. So the algorithm only takes $O(D + D^2)$ to deliver the relevancy score.

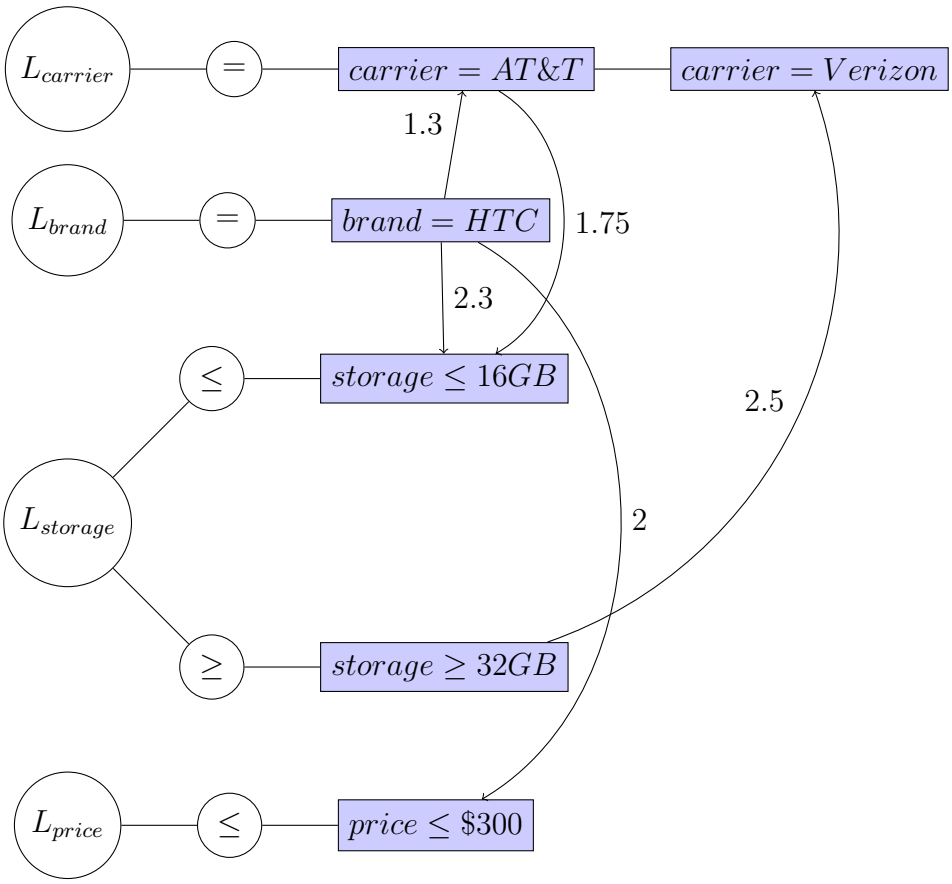


Figure 5.4: Inverted-graph on personalized user subscription space

5.2 Publication Indexing

Decide Top-k Recall that, publications are not static in the space, but they do follow an incoming stream. Publication stream is partitioned based on sliding windows. Hence, our goal is to avoid re-computation of ranked publications at each window, but to add new "winning" publications & remove "losers" incrementally at window w to a portion of Top-k results at window $w - 1$.

Example 5.2.1. Asymptotic normal Top-k matching

Let's take $w = 5$ count-based sliding window which moves forward. As Figure 5.5 demonstrates, we can compute Top-k (e.g. $k = 2$) results as publications (P_2, P_4) which have the highest relevancy scores in threshold based matching schemes. As there are $w - 1$ overlaps of publications between windows, the decision may be upon the newly added publication P_6 on it's Top-k creditability. The problem is raised to decide the i^{th} publication to be replaced; $1 \leq i \leq k$ when it's relevancy score is above a specific threshold.

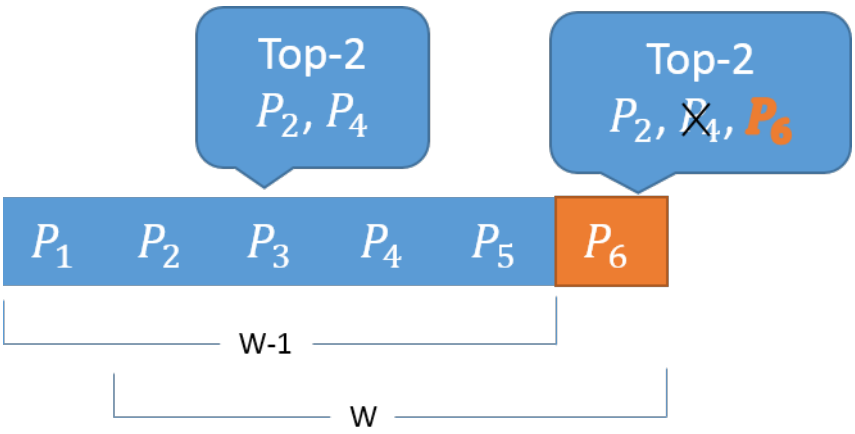


Figure 5.5: Top-k matching on sliding window approach in threshold based schemes; $w = 5$

Discussion Maintaining Top-k results over incoming stream is dependent on the matching algorithm we're dealing. As an example if Top-k matching happens based on a relevancy threshold, we can limit the problem scenario to the previous example 5.2.1. But as we addressed in our problem statement, to deliver most diversified set of results based on MAXDIVREL, we can not rely only on the previously computed Top-k. This problem was identified at previous diversity algorithms (e.g. MAXMIN, MAXSUM, DisC) as well, but left alone by simply re-computing Top-k again from the scratch.(Figure 5.6)

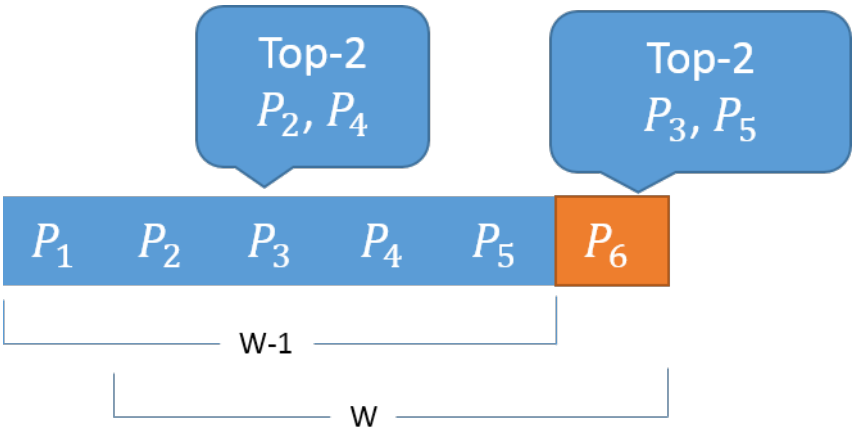


Figure 5.6: MAXDIVREL Top-k matching on sliding window approach: $w = 5$

To decide Top-k at any given window w, we avoid re-computation of all previously seen publications at window $w-1$, but only compute newly added publications against most similar ones. This phenomena is motivated by the problem definition of MAXDIVREL k-diversity. Hence, our proposed indexing mechanism is urged by this particular problem.

As we discussed in the Section 4.2.3.5, MAXDIVREL continuous k-diversity problem has a performance bottleneck when calculating neighbors in consequent sliding windows. To develop an incremental diversity algorithms based on similarity neighborhood, we can rely on a space-cutting data structure to select neighbors of a publication p_i in the current window from the neighbor selection in the previous window. That may lead to index publications in each sliding window to compute dynamic Top-k results on the run. As we described in Section 2.5, a couple of recent works on results diversification in database community have adopted tree-based techniques to model their diversity problem [23, 33]

5.2.1 Near Neighbor query

Here, we try to align MAXDIVREL continuous k-diversity problem with Near Neighbor (NN) queries. NN queries are extensively studied in static database community over different data-structures.

Near Neighbor (NN) query We say that a point p is an $R - near\ neighbor$ of a point q if the distance between p and q is at most R (Figure 5.7).

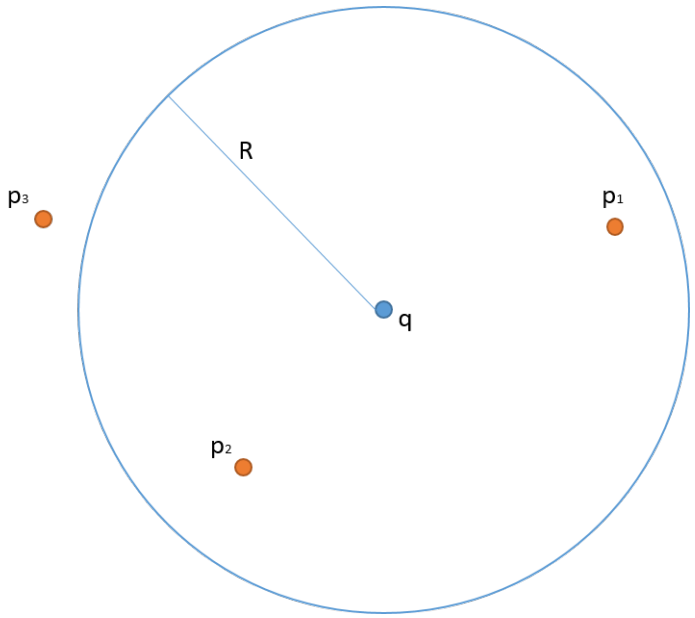


Figure 5.7: R near-neighbor query

In our study, we focus on the approximate Near Neighbor (NN) problem. The formal definition of the approximate version of the NN problem is as follows [45]:

Definition 5.2.1 (Randomized approximate R-near neighbor)**.** Given a set P of points in a

n-dimensional space $\mathbb{R}^n$, and parameters $R > 0,\ \delta > 0$, construct a data structure such that, given any query point q, if there exists an R-near neighbor of q in P, it reports some R-near neighbor of q in P with probability $1 - \eta$

We rely on above definition, where in our case, we need to find k query points that report R-near neighbors with a probability of $1 - \eta^k$. Not depending on a linear search over high dimensional space that contain large number of objects, existing data-structures that are capable to above problem are trees, grid and hashes.

Since most hierarchical models (kd-trees[1], B-trees [11], cover tree [22]) dont work well in high dimensions and, grid solutions (voronoi grid[2]) are not accurate on boundary values, we will explore a technique called Locality Sensitive Hashing (LSH)[3] to find approximate (near) matches efficiently. LSH-based methods appear most effective when the degree of similarity we accept is relatively low. As we need to find points that have maximal dissimilarity on given publication space as Top-k results, we believe LSH is the most optimal data structure to adopt.

5.2.2 Locality Sensitive Hashing (LSH)

LSH is based on a simple idea on two points which are close together, will remain close together after suitable projections from a number of different directions [46].

Definition 5.2.2 (LSH)**.** A family $\mathbb{H}$ is called $(d_1,\ d_2,\ P_1,\ P_2)$-sensitive if for any two points $p,\ q \in \mathbb{R}^d$

(i) $if\ ||p - q|| \leq d_1\ then\ P_H[h(p) = h(q)] \geq P_1$

(ii) $if\ ||p - q|| \geq cd_1 = d_2\ then\ P_H[h(p) = h(q)] \leq P_2$

$||.||$ is the L_m vector norm and $d_2 > d_1$. Note that, in order for a LSH family to be useful, it has to satisfy $P_1 > P_2$.

Ideally we need $P_1 - P_2$ to be large while $d_2 - d_1$ to be small. Notice that we say nothing about what happens when the distance between the items is strictly between d_1 and d_2, but we can make d_1 and d_2 as close as we wish. The penalty is that typically P_1 and P_2 are then close as well. As we shall see, it is possible to drive P_1 and P_2 apart while keeping d_1 and d_2 fixed.(Figure 5.8)

[1]http://en.wikipedia.org/wiki/K-d/_tree
[2]http://en.wikipedia.org/wiki/Voronoi/_diagram
[3]http://en.wikipedia.org/wiki/Locality-sensitive/_hashing

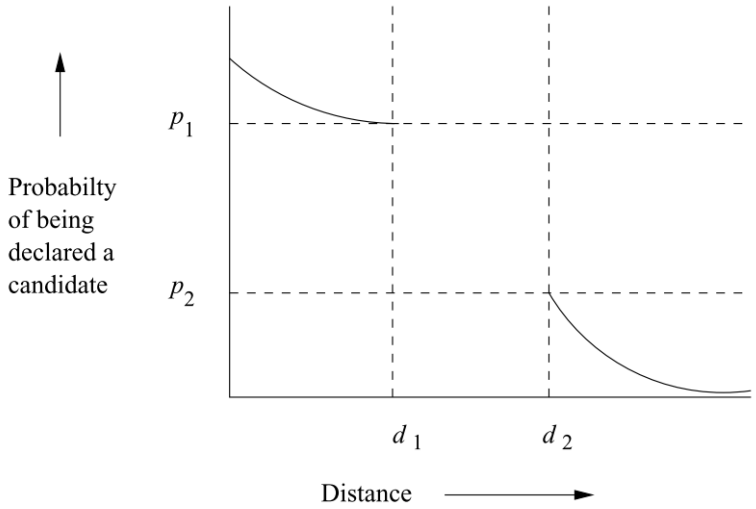


Figure 5.8: LSH probability vs distance

The probability that p and q collide under a random choice of hash function depends only on the distance between p and q. So families of hash functions were built on various distance functions (e.g. Euclidean, Hamming, Jaccard, Cosine Similarity etc.) [46]. Here, we explore the most suitable family of hash functions over the structure of publications.

5.2.3 Publications as categorical data

As we introduced in Section 3.2.1.2, the defined structure of a publication formulates a multi-variant categorical data object. For example, suppose there are 5 categorical variables & they can take any value given on the publication content.(Table 5.4)

	publication X	publication Y	publication Z
Brand	LightInTheBox	iRulu	iRulu
Color	White	Black	White
Manufacturer	NULL	Apple	Amazon
Model	Lumia 520	iPad5	Fire
OperatingSystem	Windows	iOS	Windows

Table 5.4: Categorical view of publication $X\&Y$

We can visualize publications in the high-dimensional space where the dimension is depicted by one of ordered categorical variables (Table 5.4) or the combination of category & value (Table 5.5). As an example Table 5.5 visualizes the characteristic matrix that represent the existence of categorical values. The columns of the matrix correspond to the publications while the rows correspond to the universal set of category values which the publications are characterized with.

Any cell in the table is represented by 1 or 0 based on the existence of the categorical value in the given publication.

Recap that, publications can have variety of categories which is defined in an arbitrary order. But to have one general view of the publication space, we order them in the fly. Also we allow publications not to present in all categories. But we take empty categories as well by defining the value as *null*.

Row	dimension	publication X	publication Y	publication Z
0	Brand=LightInTheBox	1	0	0
1	Brand=iRulu	1	1	0
2	Color=White	1	0	1
3	Color=Black	0	1	0
4	Manufacturer=Apple	0	1	0
5	Manufacturer=Amazon	0	0	1
6	Model=Lumia 520	1	0	0
7	Model=iPad5	0	1	0
8	Model=Fire	0	0	1
9	OperatingSystem=Windows	1	0	1
10	OperatingSystem=iOS	0	1	0

Table 5.5: A characteristic matrix representing the existence of categorical values at publications

LSH requires to have a family of hash function to suit with given data as it is sensitive to the selected "locality of distance" measure. For categorical data, there exists no inherent distance measure. But many research efforts were taken by imposing various distance measures [47].

Distance measure Here, we rely on the overlap based distance measures where the similarity between two categorical objects is based on counting the overlap between categorical values. The notion of above similarity is well known as *Jaccard similarity* [46] that is practically used to find textually similar documents. Yet, we're looking at character level similarity but not the semantic similarity. But we believe that notion of above similarity does serve well when comparing two e-commerce products which are structured as textual publications in our system.

Jaccard Distance Given a vector space, we can define the *Jaccard similarity* $SIM(x, y)$ between two vectors x and y to be the ratio of the sizes of the intersection and union of vectors x and y.

$$SIM(x, y) = \frac{|x \cap y|}{|x \cup y|}$$

Then the *Jaccard distance* $d(x, y)$ of above two vectors is defined by:

$$d(x, y) = 1 - SIM(x, y)$$

Now we can apply the LSH family for *Jaccard distance* to group similar publications while allowing dissimilar ones to stay away.

5.2.4 LSH Family for Jaccard Distance

Let's assume that we have two n-dimensional vectors where $d(x, y)$ denotes the Jaccard distance between two vectors x and y. By using the family of *minhash* functions $\mathbb{H}$, we can derive a function h that is:

$$(d_1,\ d_2,\ 1 - d_1,\ 1 - d_2) - sensitive\ for\ any\ d_1\ \&\ d_2,\ where\ 0 \leq d_1 \leq d_2 \leq 1$$

where $d_1 \& d_2$ are the boundaries of *Jaccard distances* such that it is possible to have $d(x, y) < d_1$ or $d(x, y) > d_2$.

Jaccard similarity of x and y is known to be proportional to the probability that *minhash* function will hash x and y to the same value [46]. That means the vectors of x and y are interpreted as similar when $h(x) = h(y)$ where h is the *minhash* function.

5.2.4.1 MinHashing

MinHashing is a technique to construct a *signature* that represent the given set (i.e. publication). It's most common to permute rows of the characteristic matrix and, take the number of the first row, in the permuted order, in which the column has a 1 for the correspondent column of publications. (Table 5.6)

We can have m number of permutations to construct the vector of minhash signatures for any publication.

Example 5.2.2. For the publication X at Table 5.5, we can construct the vector of minhash signatures $[h_1(X), ...h_m(X)]$ by applying m number of permutation to the characteristic matrix. Similar to that, we can form a signature matrix where the vector of minhash signatures for given publication is represented by correspondent rows. (Table 5.7)

Limitation As we have a wide variety of categorical values to represent incoming publications, the number of rows in the characteristic matrix is increasing. Performance degrades when

Row	dimension	publication X	publication Y	publication Z
6	Model=Lumia 520	1	0	0
3	Color=Black	0	1	0
1	Brand=iRulu	1	1	0
8	Model=Fire	0	0	1
9	OperatingSystem=Windows	1	0	1
0	Brand=LightInTheBox	1	0	0
5	Manufacturer=Amazon	0	0	1
10	OperatingSystem=iOS	0	1	0
7	Model=iPad5	0	1	0
4	Manufacturer=Apple	0	1	0
2	Color=White	1	0	1

Table 5.6: Calculating h_1 minhash value from the first permutation

$minhash_i$	publication X	publication Y	publication Z
h_1	$h_1(X)$	$h_1(Y)$	$h_1(Z)$
h_2	$h_2(X)$	$h_2(Y)$	$h_2(Z)$
...	..	..	..
h_m	$h_m(X)$	$h_m(Y)$	$h_m(Z)$

Table 5.7: A signature matrix that represent publications

computing many permutations of a large characteristic matrix.

But mathematically, we can derive the effect of m number of random permutations by selecting m number of random hash functions that maps row numbers of characteristic matrix (Table 5.5) to a bucket i; $where\ 0 \leq i \leq number\ of\ rows$. To simulate a true random permutation, the number of rows should be a prime number [46]. Algorithm C.1 demonstrates a fast min-hashing algorithm to construct the signature matrix (see the appendix).

$minhash_i$	publication A	publication B	publication C	publication D	publication E
h_1	1	0	0	1	4
h_2	6	3	3	6	7
h_3	0	6	6	0	9
...	..	..	..	..	..
h_{12}	1	1	1	1	1

Table 5.8: Sample signature matrix generated for 5 publications

Now we can visit the generated signature matrix for estimating the Jaccard similarities of underlying publications (Table 5.8). Any pair of publications are estimated as similar, if they're represented by identical columns which are above a similarity threshold. It's shown that we

can reduce the estimated error e by increasing the number of signature hash functions (m) generated[1].

$$Estimated\ error(e) = O(\frac{1}{sqrt(m)}) \tag{5.1}$$

As a drawback we still need to compare every pair of signatures. But as a motivation, now we can reduce any given high-dimensional, multi-variant, categorical publication into a vector of short minhash signatures.

5.2.5 LSH in MAXDIVREL

Equipped with the insights given in Sections 5.2.2, 5.2.3 & 5.2.4 about LSH, here we present how we adopt it for setting the boundaries of similarity between publications in a single coherent framework. In this section, we discuss about the batch construction of publication indexing at a given sliding window.

Recall that our goal is not to compute (dis)similarity of every publications in MAXDIVREL algorithm at each sliding window. An exhaustive search could yield a lower bound of $\frac{dn(n-1)}{2}$ operations to compute k-optimal MAXDIVREL subset of n publications with d-dimension. Hence, we try to reduce that overhead of search, by indexing & clustering similar publications based on randomized LSH algorithm. Ideally we need to cluster similar publications together and, separate dissimilar ones based on a threshold neighborhood of Jaccard distances.

In Section 5.2.4, we derive a *signature matrix* to represent the publication space. We apply LSH for that signature matrix to construct buckets that group similar publications. We hope that true near neighbors will be unlikely to be unlucky to map into the same bucket by considering all projections.

5.2.5.1 LSH for signature matrix of publications

Here, we adopt the *Jaccard* family of hash functions, where we can evenly segment the minhash signature of any publication into L hash tables. A size of a signature segment is denoted by r where $r \leq m$ & $L \times r = m$. Each table consists of average b number of buckets. For each such hash table, there is a hash function that takes the column vector of size r and, maps them into a bucket. As each has table can be represented by a *bucket array*, we have L number of bucket arrays.

Example 5.2.3. As an example, the signature matrix ($m \times n$) in Table 5.8 demonstrates $n = 5$ publications where each publication has been represented by a size $m = 12$ of minhash

[1]http://en.wikipedia.org/wiki/MinHash

	$minhash_i$	publication A	publication B	publication C	publication D	publication E
L_1	h_1	1	0	0	1	4
	h_2	6	3	3	6	7
	h_3	0	6	6	0	9
L_2	h_4	4	1	1	4	1
	h_5	8	2	2	8	2
	h_6	0	5	5	0	5
L_3	h_7	3	4	1	1	4
	h_8	4	1	3	3	1
	h_9	0	0	2	2	0
L_4	h_{10}	5	0	1	0	5
	h_{11}	0	0	1	0	0
	h_{12}	1	1	1	1	1

Table 5.9: Segmented signature matrix generated for 5 publications

signature. The signature matrix is divided into $L = 4$ hash tables where each hash table contains b arbitrary number of buckets. Any bucket is denoted by a key which is calculated by taking a vector of minhash signature at size $r = m/L = 12/4 = 3$. Segmented hash tables are visualized at table 5.9.

In hash table L_1, the pair of publications A & D is mapped into the same bucket at L_1 bucket array, since their columns are identical. That similarity is estimated based upon the hash function for the correspondent hash table L_1 regardless of column vectors in other hash tables. This mapping will repeat until all 5 publications are projected into buckets at L_1. The same process has been performed by all hash tables simultaneously.

Any closely similar publications that are unlucky to be mapped into the same bucket at L_1 hash table, still have 3 chances to be together by considering other L_2, L_3 & L_4 hash tables. We assume that the probability of *false positives* where dis-similar publications map to the same bucket will be fractionally low. Also if similar publications are identical in signature vectors will always have the tendency to map into at least single bucket in all hash tables, to reduce *false negatives*.

5.2.5.2 Compute Top-k publications

Figure 5.9 visualizes how the signature matrix (Table 5.9) is segmented into $L = 4$ hash tables for $n = 5$ number of dummy publications.

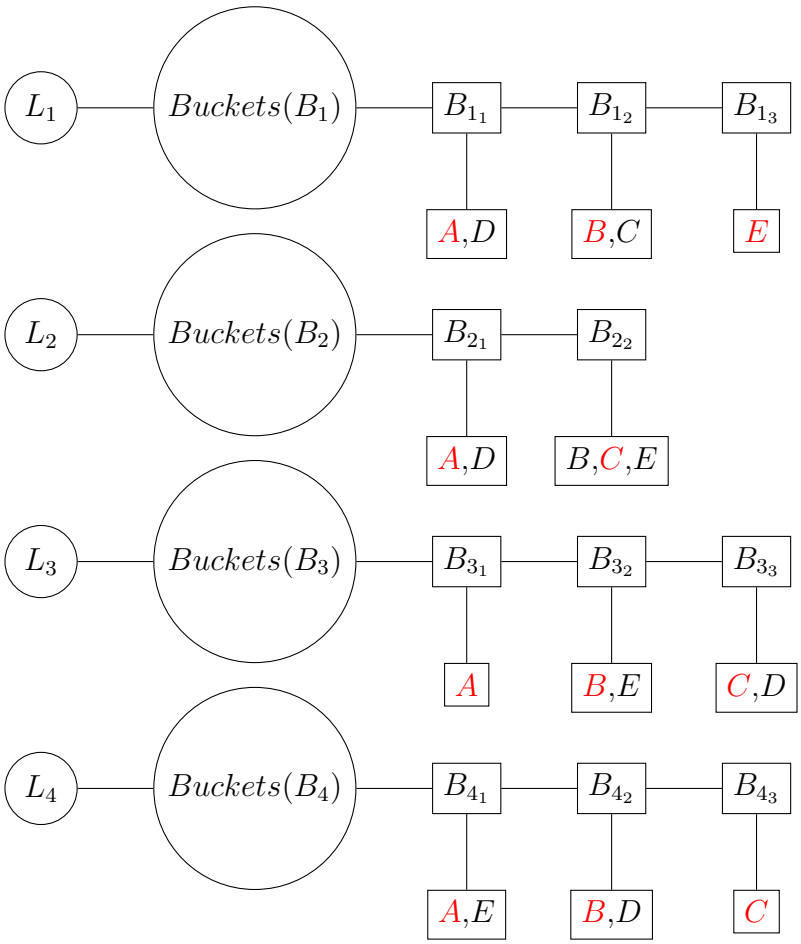


Figure 5.9: Bucket array of n publication at L hash tables

Each bucket is considered as a neighborhood of similar publications. We pick the most relevant publication that has highest relevancy score as the "winner" from each bucket. Any winner is dominant to represent it's bucket neighborhood. In Figure 5.9, "winners" are colored as red. All hash tables vote such winners to be the Top-k publications. We select k "winner" publications that have a majority of votes to be the final Top-k publications. When k is less than the number of "winner" publications for the selection, we wait for the publications at next sliding window.

Table 5.10 demonstrates the retrieval of Top-2 publications based on votes by former indexing mechanism. We break ties by giving priorities to most relevant ones based on fresh relevancy score.

publication (P)	A	B	C	E
Votes	4	3	3	1

Table 5.10: Top-2 publications retrieval based on votes

5.2.6 Incremental Top-k computation

Figure 5.10 shows the process of computing Top-k results incrementally at arbitrary sliding window j of size w, by assuming the overlap of publications with previous $(j-1)$ window is $w-1$. Figure 5.6 simulates this scenario by producing a different variation of Top-k publications than previous window $j-1$.

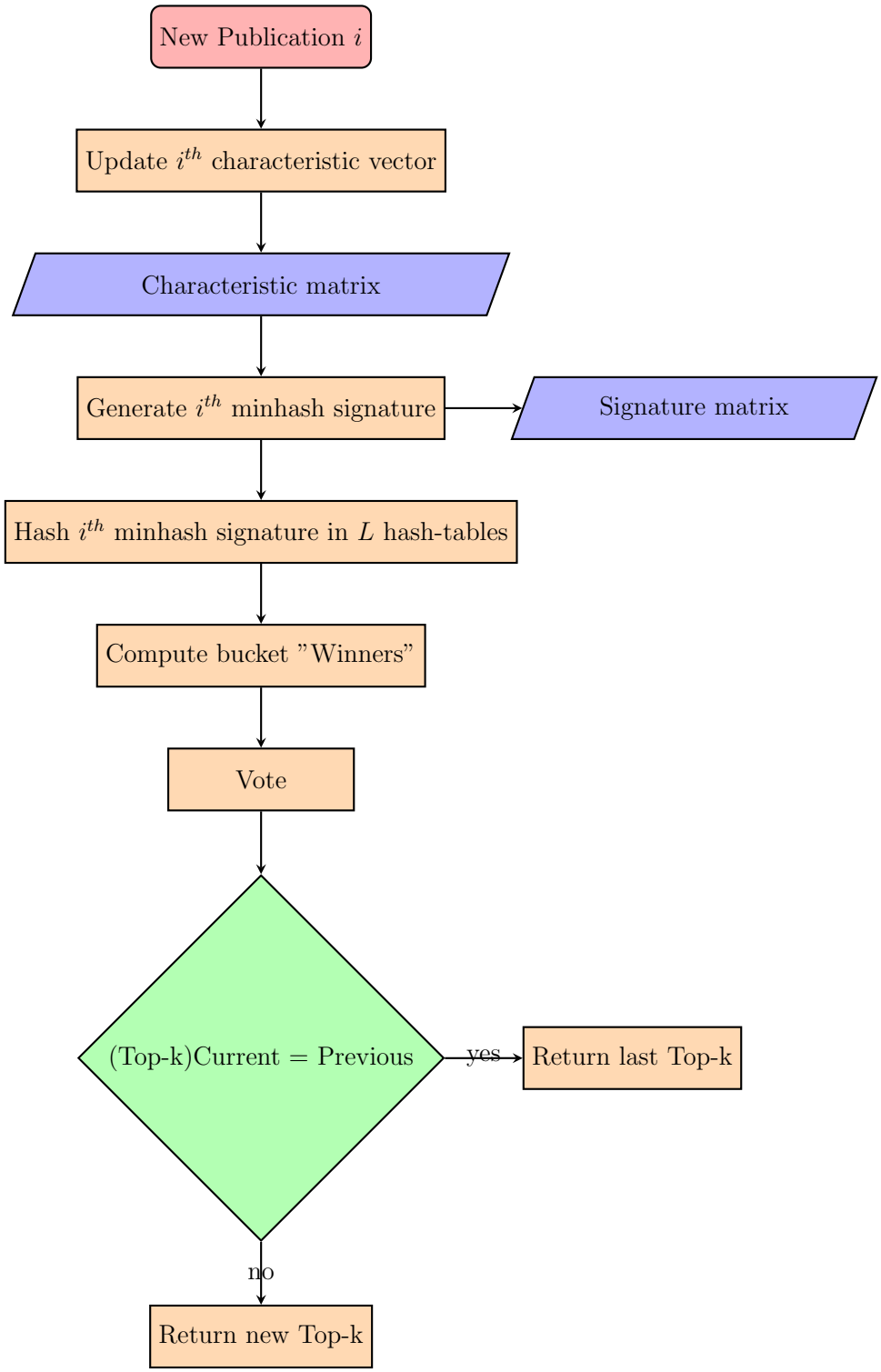


Figure 5.10: Flow chart of incremental Top-k computation

Example 5.2.4. Let's assume that 5 publications are projected into $L = 4$ hash-tables as visualized by the Figure 5.9 at a given time t. A publication is being voted by each bucket as the "winner" publication (Red colored) that has the highest relevancy score among the bucket neighborhood. Top-2 publications are selected from the majority of votes as described earlier. Publications A & B are returned as Top-2 publications at time t.

For a new publication, system generates a minhash signature to represent it. Then it is being projected into at least one bucket at each L hash tables where the similar set of publications reside. Above mechanism has the tendency to avoid unnecessary comparisons with dis-similar publications. Figure 5.11 shows an updated view of index structure at time $t + 1$ after new publication F is being projected into the buckets shown.

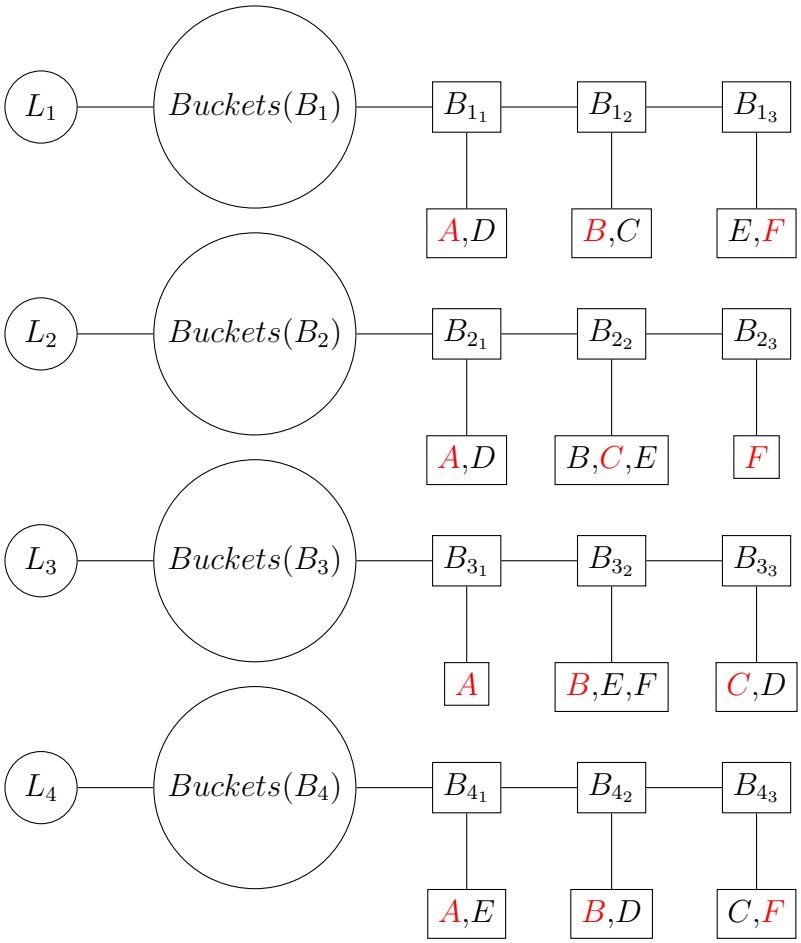


Figure 5.11: Sample LSH indexing mechanism at arbitrary time $t + 1$s

Table 5.11 demonstrates the retrieval of updated Top-2 publications A & F at time $t + 1$ which only contains a single publication that is selected as Top-k previously at time t. We assume that previously selected Top-2 publications are not delivered yet, unless they will be expired from the list.

publication (P)	A	F	B	C
Votes	4	3	3	2

Table 5.11: Top-2 publications retrieval at time $t+1$

We only consider the buckets which holds the new publication F to update the list of Top-k publications. Only those buckets forward their votes for a Top-k candidate. Specially we avoid re-computing whole set of publications to produce & update Top-k publications, which remains as an important contribution of our study.

5.2.7 Analysis

5.2.7.1 Probability Analysis

As we indicated earlier, *Jaccard similarity* $SIM(x, y)$ of publications x and y is known to be proportional to the probability that *minhash* function will hash x and y to the same value [46]. Let $SIG(x)$ & $SIG(y)$ denotes the signature vectors of publications x & y which are derived by applying minhash:

$$SIM(x, y) \propto Prob[SIG(x) = SIG(y)] \tag{5.2}$$

Publications x & y have the probability of $SIM(x, y)^b$ to map into the same bucket at particular hash table. b denotes the average size of bucket array at each hash table. For all L hash tables, the probability $(1-SIM(x, y)^b)^L$ denotes that they might not map into any bucket. Then, we can say that any closely similar publications have the probability $1-(1-SIM(x, y)^b)^L$ to be projected into at least one bucket among all hash tables.

Does LSH indexing mechanism solve MAXDIVREL k-diversity problem holding dominance & independence condition? We believe proposed mechanism is tailor made for solving MAXDIVREL k-diversity problem. Based on the definition of LSH (section 5.2.2), recall that the projections of two closely similar publications, separated by Jaccard distance d will always be close. But due to quantization, these two publications might fall into two separate buckets at d probability as equation 5.2.

Since similar publications have the tendency to map into same bucket at probability $1-d$, dominance condition can be well served. Because the "winner" publication as the most relevant publication at each bucket, can cover it's neighborhood. Also two buckets represent two separate neighborhoods. That results all "winner" publications to be dis-similar from each other by at least d distance. So it also satisfies the independence condition.

Also incremental LSH index mechanism satisfies *continuity* requirements as well (Section 4.2.3.5). Because it doesn't neglect a publication which belongs to the list of Top-k publications at $j-1^{th}$ window, in the new list at j^{th} window. We consider any publication in the Top-k list is expired once delivered. Since "winners" get priority on the fresh relevancy score, it avoids a publication x to be a candidate of Top-k list, once the publication y is being selected which is not-older than x. For simplicity, we only maintain an ordered queue of size k for the list of Top-k publications at any given time t.

How to select the number of hash tables L and the size of signature vector r? We know that the upper bound m of minhash functions is $\frac{1}{e^2}$ given an estimated error e from the Equation 5.1. By partitioning the signature matrix evenly, we can derive the following equation:

$$m \;=\; L \times r \tag{5.3}$$

The probability of getting estimated as similar publications has shown the behavior of an $S-curve$ (Figure 5.12) [46]. We can observe when the Jaccard similarity of two publications, is approximately 0.5, there is a rise of the steepest at the curve. S-curve presents the threshold of similarity in the following equation:

$$Similarity\ Threshold(s) \;=\; \frac{1}{L}^{\frac{1}{r}} \tag{5.4}$$

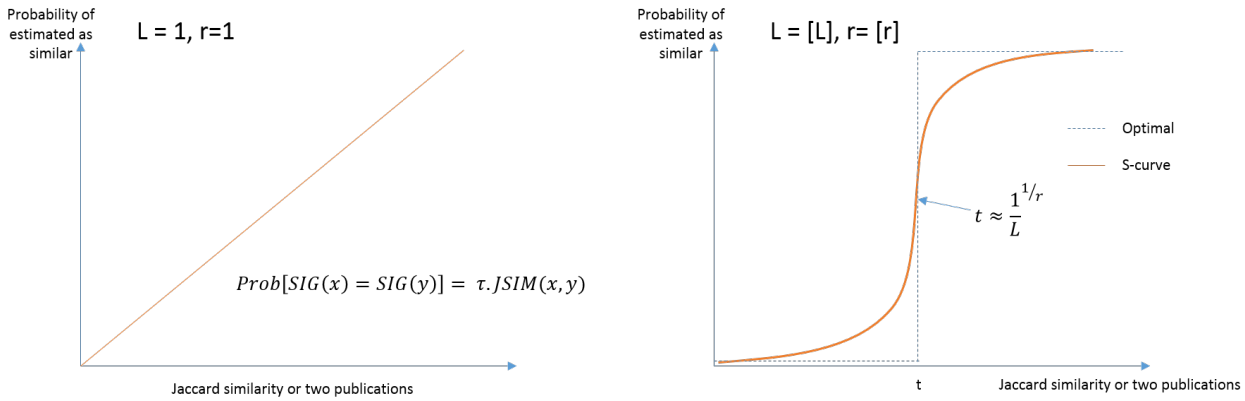


Figure 5.12: The behavior of s-curve

To select L & r, we follow above two equations 5.3 & 5.4. Our experiments are motivated by this observation, hence examine a variation of Jaccard similarity s & the number of minhash functions (m).

5.2.7.2 Time complexity

For a given query publication z, we need $(T_g + T_c)$ time to find whether it belongs to the set of Top-k publications. T_g is the time needed to generate the signature vector for z & hash it into buckets, while T_c is the time to update winners at all projected buckets.

$$\textit{Generate signature vector \& hash into buckets } (T_g): \ O(Dm) \ + \ O(rL);$$

$$\textit{signature vector of size } m, \ D \ - \ \textit{dimension}, \ rL \textit{ projections}$$

$$\textit{Search the bucket for a winner } (T_c): \ O(L.N_c \ log \ N_c); \ L \ Hash \ tables,$$

$$\textit{and } N_c \textit{ the expected number of publications in a bucket}$$

We can observe that $T_g + T_c$ increases as m increases based on Equation 5.3.

5.2.7.3 Space complexity

For n number of publications we need $O(Dn)$ space to store the characteristic matrix. But we can compress the matrix storage by only keeping true binary values, since all dimensions of the universe are not used to represent most number of publications. Note that we only need characteristic matrix as a reference to generate the minhash signature for any publication.

When we're bounded by the estimated error e, there are $m = \frac{1}{e^2}$ number of minhash functions that we use to generate a signature matrix for all publications. The signature matrix takes $O(nm)$ space for n number of publications.

Example 5.2.5. If we have 100 publications at each window, with respect to the estimated error $\delta = 0.1$, so $m = \frac{1}{0.1^2} = 100$, signature matrix only takes $nm = 100 * 100 * 4/bytes = 40 \ kb$. Since we do keep only the reference row number in the signature matrix.

To handle streaming publications, we periodically refresh the signature matrix when it's size goes beyond some predefined threshold.

Chapter 6

Implementation - Cloud Middleware

To evaluate our approach, we have implemented a content-based publish/subscribe platform in a private cloud using *Amazon Web Services* - a scalable cloud service provider. Our prototype performs Top-k ranked delivery on given personalized user subscription space across a stream of publications. Top-k results get diversified based on the proposed diversity method MAXDIVREL. Both indexing methods in subscription & publication space is plugged into reduce Top-k query processing time. In this Chapter, we explore our implementation modules.

6.1 Amazon Web Services (AWS)

We implemented our proposed Top-k publish/subscribe architecture (Figure 4.1), as a prototype in a private cloud. As there are many cloud service providers who provide almost identical services, it's difficult to have a clean edge of difference between them. Due to long-term wide adopting nature in both research & academic works, we have chosen Amazon as our cloud service provider. Amazon have both public & private clouds (AWS [1]) which support broad & deep core cloud infrastructure services (i.e. IaaS, SaaS, PaaS). Further it also provides services that are scalable & secure, and many more.

Amazon cloud supports a simple topic-based publish/subscribe model based on Amazon Simple Notification Service (SNS). But like most other cloud service providers, they do not support ready-made content-based publish/subscribe model with their services. But we can integrate several AWS modules to develop such model. As Figure 6.1 shows, our model is integrated by many AWS services (IaaS) where we implement proposed Top-k matching model on top of them.

[1] http://aws.amazon.com/

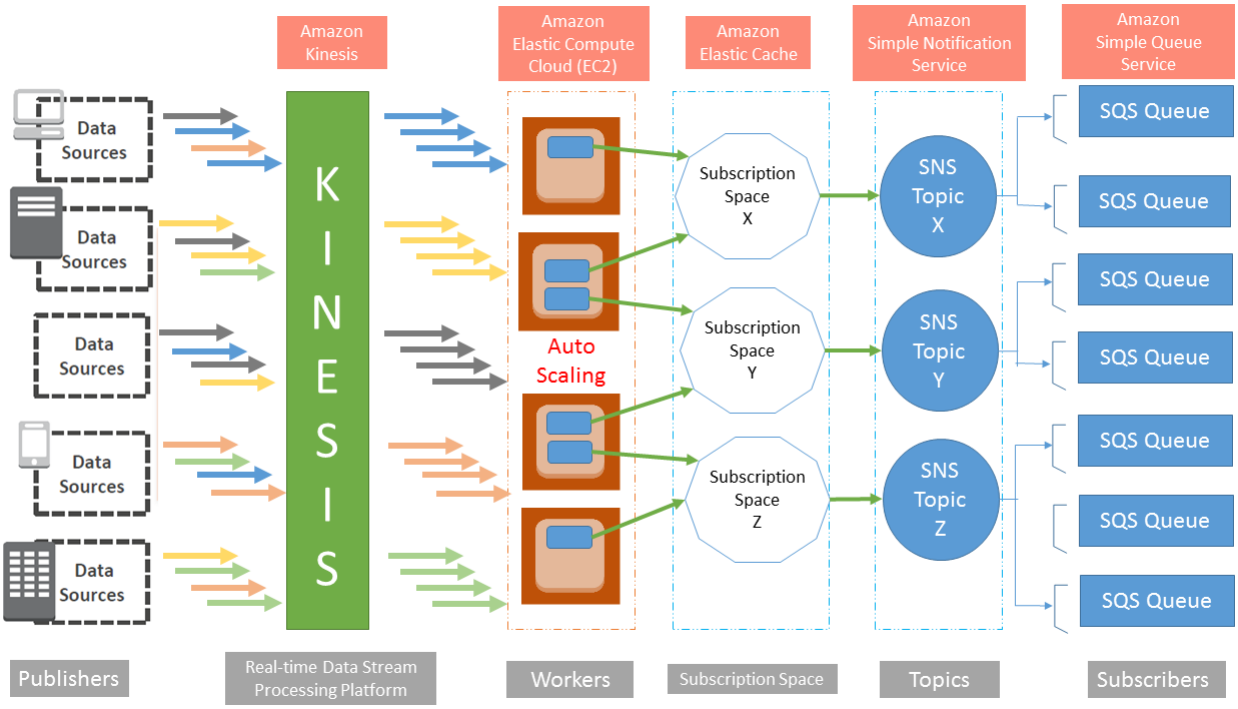


Figure 6.1: Implementation modules based on Amazon web services

6.2 Main Modules

- **Publication stream:** simulated using Amazon Kinesis[1]: a real world data stream processing platform
- **Indexed personalized subscription spaces:** implemented on top of Amazon Elastic cache (AEC)[2]: a widely adopted in-memory object caching system
- **Sliding window Top-k matching with indexed publications:** implemented on top of Amazon Elastic Compute Cloud (EC2)[3] worker instances which provide re-sizable compute capacity in the cloud
- **Event delivery:** implemented on top of Amazon Simple Notification Service (SNS)[4]: a fast, flexible, fully managed push messaging service.
- **Persistent notification service:** implemented on top of Amazon Simple Queue Service (SQS)[5]: a hosted queue for storing messages as they travel between different parties

[1]http://aws.amazon.com/kinesis/
[2]http://aws.amazon.com/elasticache/
[3]http://aws.amazon.com/ec2/
[4]http://aws.amazon.com/sns/
[5]http://aws.amazon.com/sqs/

6.2.1 Publication Stream

We deal with publications in a dynamic setting as we described the structure of incoming publications in section 3.2.1.2. For streaming data ingestion & processing, we use Amazon Kinesis as a building block to process continuous data in real-time.

Reasons to select Amazon Kinesis Amazon Kinesis is built upon a novel concept to provide streaming as a service in the cloud. It can be used to process data continuously where most old stream processing platforms supports only batch processing. Some modern day infrastructures (e.g Storm[1], Spark[2]) do provide continuous processing but they are expensive to operate. Also modern applications set-up new requirements where it needs to make decisions much faster & scale entire system elastically. As our research work emphasizes to reduce Top-k query processing time & scale on many clients, the need of such technology is crucial to accept our prototype model in real production environment.

For a complete study, refer Figure 6.2 to explore more on stream processing platforms[3] available at current date.

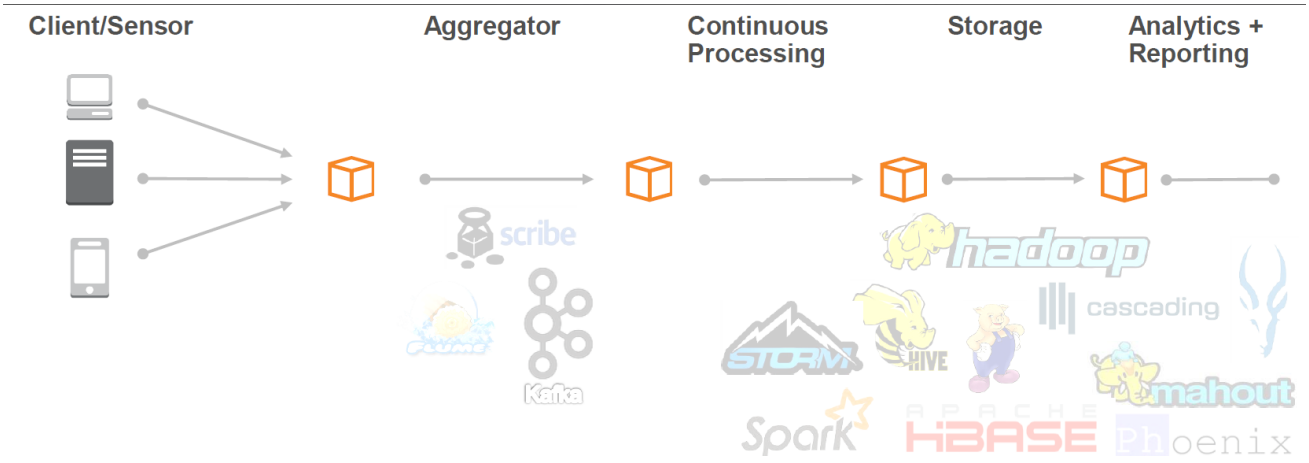


Figure 6.2: Stream Processing platforms available at current date

Simulating Publication stream at Kinesis In Kinesis, streams are made of *shards* - a basic unit that ingests data up to 1 MB/sec & emits up to 2 MB/sec. We can control the elasticity of the stream by spitting or merging shards while the stream is running. But since we assume to allow streams that have Poisson arrival rate, we can align with default settings. For the evaluation, we are capable to replay the stream with same data within 24 hour period.

[1]https://storm.apache.org/
[2]https://spark.apache.org/
[3]http://aws.amazon.com/kinesis/

Producers can put their publication to the created Kinesis stream. Our system partition them to spread the workload across shards. A unique sequence number is returned to every publication that we used to set the order of incoming publications. To process incoming publications, we use Kinesis Client Library (KCL) to act as an intermediary between our system & the stream. It allows our system to have many versions running under different algorithms. That helps us to validate the efficiency of our proposed model across the same stream. Also KCL is used to redistribute workers to new EC2 instances & process data with fault tolerance. (Figure 6.3)

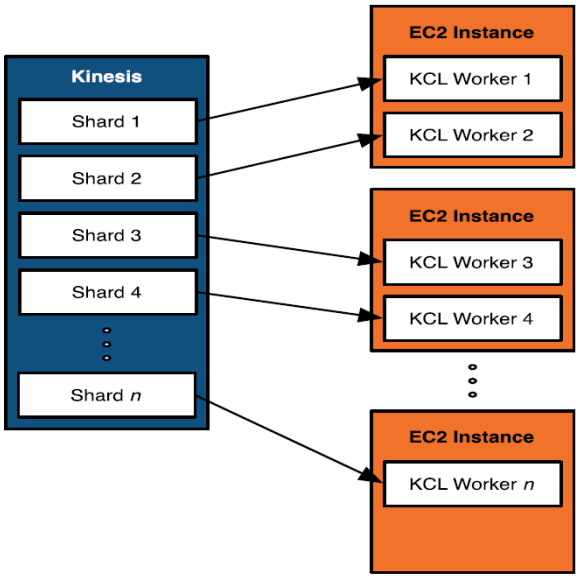


Figure 6.3: Building Kinesis processing

6.2.2 Indexed personalized subscription spaces

We address that any user can have a number of large subscription spaces. Usually the subscriptions are kept locally in-memory at EC2 application servers. As a fast growing technology, in-memory computing has attracted a lot of interest over stream processing community [1]. When subscriptions grow in size & transactional capacity, distributed caching mechanisms can provide a consistent service in the long run.

Any subscriber is defined by his personalized subscription sub-graphs which can have a large number of relations. We found that an entire graph of subscriber needs to be in cache to make the caching mechanism effective for serving an in-memory operational data-store [2].

[1] https://gridgaintech.wordpress.com/2013/09/18/four-myths-of-in-memory-computing/
[2] http://blog.pivotal.io/pivotal/case-studies-2/using-redis-at-pinterest-for-billions-of-relationships

Storing subscriptions at Amazon Elastic cache (AEC) Amazon Elastic cache is a managed distributed caching service provided by AWS. It is protocol-compliant with other popular caching engines (e.g memcached[1], redis[2]). They are able to work seamlessly with AEC. For every subscriber, we store a list of relations among subscription tuples at AEC. Also AEC supports to have cache clusters which runs in a master-slave configuration. We use *memcached* as default cache engine which is a key-value in-memory store & guarantees to provide high-performance.

AEC is used to maintain a separate memcached layer from EC2 application servers. As Figure 6.4 demonstrates, both EC2 & memcached layers scale & address issues independently. Note that both layers are maintained under single cloud availability zone. In memcached layer, several cache nodes are grouped into AEC clusters. Here, we don't address data replication & synchronization among cache nodes. But to easily maintain the caching tier, a level of grouping is used to perform operations such as memcached configuration & security.

The throughput & latency is dependent on the type of EC2 & AEC instances where the system is deployed into. But we can always tune system resources for best utilization. AEC supports adding or removing cache nodes manually or automatically to scale well. We use *spymemcached*[3] client to perform main memcached operations with AEC.

6.2.3 Sliding window Top-k matching with indexed publications

A list of Top-k publications are updated for a particular user over the instances of sliding window as described in the Section 4.3. Streaming publications are indexed on-line based on the proposed incremental LSH based indexing mechanism. LSH is a main-memory algorithm which consists of two phases in the implementation.

- Hash generation: where the hash tables are constructed for minhash signatures
- Querying: where the hash tables are queried to look up similar publications

Handling streaming publications The dimension of any publication is varied over the stream. Such that, we need to maintain a characteristic matrix to normalize all publications into an universal domain. We refresh the characteristic matrix periodically, since it becomes very sparse in the long run.

[1]http://memcached.org/
[2]http://redis.io/
[3]https://code.google.com/p/spymemcached/

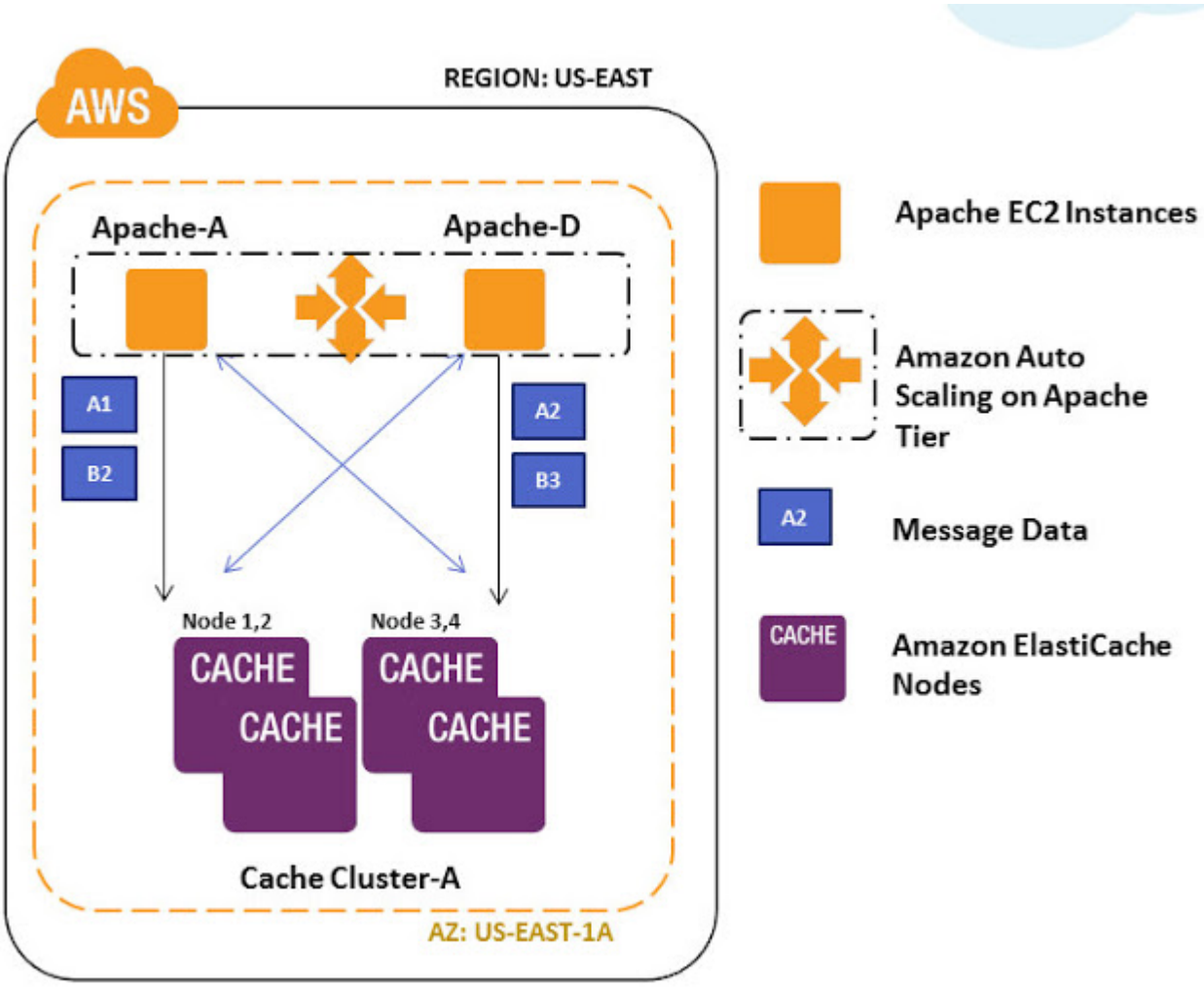


Figure 6.4: Memcached nodes shared in separate AEC tier (source: AWS)

Recall that we use m number of hash functions to generate the minhash signature for any publication. We generate such minhash signature as a function of two general hash functions along with the inputs up to m as below;

$$h_i(i) = h_1(i) + j \times h_2(i) \ \%D \ where \ 0 \leq j \leq m; 0 < D \leq dimensions \tag{6.1}$$

Based on the minhash signature, the publication is projected into a particular bucket at all L number of hash tables. We only need to re-construct such buckets to update the list of Top-k publications.

Top-k matching process At particular EC2 instance, there are multiple workers as depicted by the Figure 6.3. Publications are partitioned based on pre-defined annotations (e.g. Automotive, Home, Phones etc.) by Kinesis & distribute them to multiple workers. For simplicity, we assign an individual worker for each subscriber. The worker can access the relevant subscrip-

tion space & compute the relevancy score of incoming publications. Then the publications are indexed based on LSH. Top-k publications can be retrieved as described in the Section 5.2.6.

The index can be shared among workers, but we do consider that each worker builds a separate index of own publications for simplicity. To share the LSH index across multiple cores & multiple nodes, will remain as a future work of our study.

6.2.4 Event delivery

Top-k publications are delivered to subscriber end-points using SNS message push service. SNS is a topic based channel that coordinates and manages the delivery or sending of messages to subscriber endpoints or clients. For every user we create a SNS topic with assigned user name. The subscription spaces are linked with each topic. Note that, a large user subscription space can be decomposed into subspaces per interest defined by relevant SNS topic. (Figure 6.5)

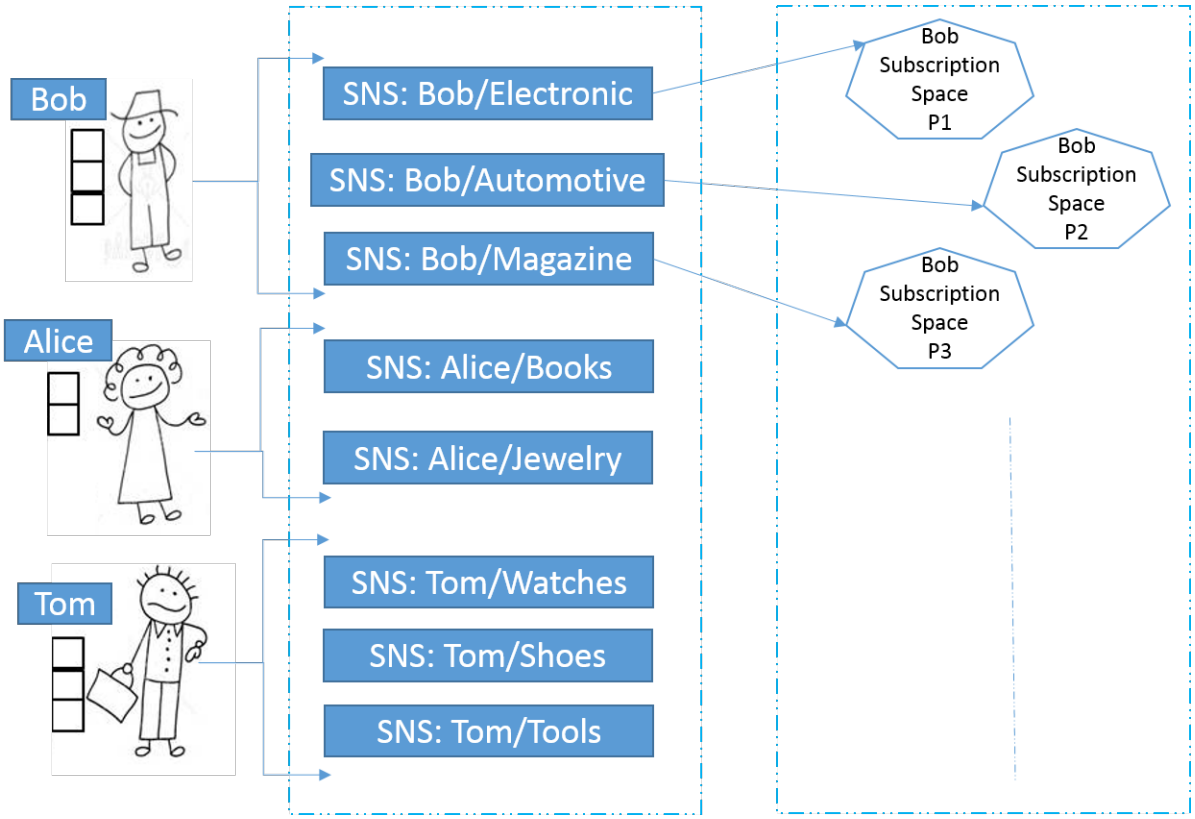


Figure 6.5: SNS Topics + Personalized user subscription spaces per interest

6.2.5 Persistent notification service

Top-k publications are updated in consecutive windows incrementally, once we compute Top-k results in the previous window. Any instance of Top-k results are delivered to a persistent queue structure based on SQS in the order they are selected. We keep those results until they are delivered. SQS can be used at any level of throughput, without losing messages or requiring other services to be always available.

6.3 Scalability & Elasticity

Our model is composed by many individual modules using relevant cloud based web services. Each module is designed to work in conjunction with other service modules. Most of them are highly reliable & secure but specially provides a platform to build scalable applications on top of them. One of the main reasons to build our prototype model in the cloud is to deal with scalability & elasticity. Hence, we have designed our system to scale on top of AWS. To handle high publication arrival rates and large number of subscriptions remain as our objectives to achieve in the prototype model.

Handle high publication arrival rates As described earlier, we use Kinesis to handle real-time streaming from any number of publishers & to scale up & down as needed. It mainly addresses scalability in the terms of processing & sharding [1]. We configure the policy of scaling based on Amazon CloudWatch [2] metrics. Since, our system is more greedy on processing power than network traffic, we set a threshold based schema on CPU utilization. Kinesis also supports on splitting or merging shards to accommodate dynamic arrival rates. But recall that, we allow the arrival rate of incoming publications to follow a Poisson random variable.

Store a large number of subscriptions The number of subscriptions can be increased when there are thousands of users. Because most subscriptions are long running, that might results the local cache at EC2 instances to grow beyond the bound of available memory. Amazon Elastic cache (AEC) supports to mitigate this overhead by keeping a separate tier of in-memory caches. We can add or delete such cache nodes to meet the load of subscriptions while AEC automatically discover such behavior [3]. In our system, we define a policy to have such separate AEC tier, when EC2 instances keep going out of memory for storing subscriptions.

[1] https://aws.amazon.com/blogs/aws/amazon-kinesis-real-time-processing-of-streamed-data/
[2] http://aws.amazon.com/cloudwatch/
[3] http://aws.amazon.com/elasticache/

Chapter 7

Evaluation

In this Chapter, we present an experimental study to measure the effectiveness of delivered Top-k results of the proposed MAXDIVREL k-diversity algorithm by comparing with other diversity approaches in pub/sub domain. A number of experiments were also performed on all proposed modules, to measure the efficiency of the implemented system. As we described in the Section 6, the cloud middle-ware was implemented using AWS. All algorithms are implemented in Java & experiments were performed on Linux based micro-node instances each with 2.3 GHz, 8GB memory, which run within a virtual private cloud. We have used a synthetically generated dataset in the experimental evaluation.

7.1 Datasets

In large scale production environment, it's almost impossible to extract the real interaction between publishers & subscribers. In such conditions, we need to handle a large number of clients that connect to the publish/subscribe system and execute their transactions.

To test the behavior of proposed Top-k publish/subscribe system, we synthetically generate data according to *zipfian* distribution that matches the real data the best. *zipfian* distribution mathematically models the natural interaction between publishers & subscribers according to the following probability distribution [1]:

$$zipf(k : \chi,\ N) = \frac{1/\ k^{\chi}}{\sum_{n=1}^{N}(1/\ n^{\chi})}$$

where N is the number of elements in distribution, k is the rank of element & χ is the value of exponent.

[1] http://en.wikipedia.org/wiki/Zipf's/_law

Note that, we use real e-commerce product data that is available in *Amazon on-line market place* to generate publications & subscriptions that follow zipfian properties. The data set contains information of $100K$ products that are available on sale at Amazon on-line marketplace on *November* $17 - 19^{th}$ 2014. *Amazon Product Advertising API*[1] has been used to retrieve product details. The dataset covers products from 6 main categories, *Automotive, Books, Electronics, Movies, Phones and Home* including 1529 sub-categories. All products are listed over 334 independent attributes & the size $\approx 2000K$ of value space. Table 7.1 summarizes the variety of retrieved product data.

Main category	**Number of sub-categories**	**Number of products**	**Number of unique attributes**	**Number of unique values**
Automotive	322	31705	48	
Books	160	10000	42	
Electronics	142	14200	54	
Movies	197	8700	66	
Phones	40	1984	57	
Home	668	32865	67	
Total	1529	100,000	334	$\approx 2000K$

Table 7.1: Summary of measurements on retrieved product data

More specifically, Table 7.2 shows the sample attributes & example values that can take place at the list of products indexed by the keyword "smartphone" in the main category "Phones".

Attribute	**Example values**
Brand	Motorola, Nokia, LightInTheBox, iRulu,..
Color	Black, White, Red,..
Manufacturer	Motorola, Nokia,..
Model	droidx, Lumia 520, G2,..
OperatingSystem	Android, Windows Phone, iOS,..
Amount	USD2690, USD2285, USD3456,..
HazardousMaterialType	Unknown,..
Height	295, 345, 256,..
Length	559, 654, 678,..
Width	35, 32, 24,..

Table 7.2: A sample set of attributes (10/57) that represent the subcategory "smartphones"

[1]https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html

Publication Generation At each random interval, the publisher selects a bulk of products from our dataset & pushes them to the system. The number of incoming publications follow a Poisson random variable with fixed arrival rate. (Table 7.3)

Brand = LightInTheBox
Color = White
Manufacturer = Nokia
Model = Lumia 520
OperatingSystem = Windows Phone
Amount = USD2690
Height = 295
Length = 559
Width = 35

Table 7.3: Sample Publication X

Subscription Generation A subscription is generated as a combination of attribute-operator-value tuples. Naturally, some attributes are popular among subscribers than others. Thus, we order the attributes to increase the visibility over the e-commerce product space and, apply them in zipf distribution to generate subscription for specific subscriber. For the experiment, we consider only *String* comparison operators(e.g. *prefix-of, suffix-of, equal*), and they are chosen uniformly. The values can be selected using either uniform or zipf distribution. As an example, the values of *Motorola, Nokia, LightInTheBox, & iRulu* has an equal probability to be chosen for the attribute *Brand*. Or like the selection of attribute, we order the values in a pre-defined order & any attribute can take i^{th} popular value.

Subscriber can have both uniform & non-uniform preferences over the subscription space. So preference weights are made over the subscriptions in both uniform & normalized way under $N(0, 1)$. We further evaluate the resiliency of the system , when there is a little change δ in the preference weight (Table 7.4).

Subscription tuple	Uniform Preference	Normalized Preference	Preference - δ	Preference + δ
Brand = Nokia	0.5	0.3	0.3 - δ_1	0.3 + δ_1
Color = White	0.5	0.6	0.6 - δ_2	0.6 + δ_2
Amount $\leq$ USD3000	0.5	0.2	0.2 - δ_3	0.2 + δ_3
Height $<$ 300	0.5	0.4	0.4 - δ_4	0.4 + δ_4
Length $>$ 500	0.5	0.1	0.1 - δ_5	0.1 + δ_5
Width $\leq$ 35	0.5	0.2	0.2 - δ_6	0.2 + δ_6

Table 7.4: Sample Subscription S with preference values

Is Normal distribution reasonable to draw random user preferences? Each subscriber has a distribution of preferences which results the entire population to have a set of preference distribution. The inconsistency between human choices & preference weights has explored using stochastic theory of uncertainty (e.g regret theory [1]). Many utility functions that are compatible with human choices (e.g. power law model) have been experimented to overcome former inconsistency. Such utility models are evaluated at [48] where the preferences are made by randomly selected power law functions. They have tested those preferences with actual data drawn from a truncated normal distribution & conclude the deviation is small. We're motivated by these experiments & believe normal distribution is perhaps the most natural assumption to proceed with.

A product is fully described by the values taken by it's attributes. An attribute in a subscriber view can be characterized by values that are not in the original product space. Due to the possibility of generating a massive number of subscription views of attributes, values and different operators, we limit the size of a subscription into d_s s.t. $2 \geq d_s \geq 32$.

As an example, the product space of *Automotive* class has been represented by 48 unique attributes. So any subscriber can have a maximum $\binom{48}{d_s}$ number of subscription views over the value space in *Automotive* products. Therefore,

$$Total\ number\ of\ subscriber\ views\ = \sum_{d_s=2}^{32} \binom{48}{d_s} + \binom{42}{d_s} + \binom{54}{d_s} + \binom{66}{d_s} + \binom{57}{d_s} + \binom{67}{d_s}$$

[1] http://en.wikipedia.org/wiki/Regret/_(decision/_theory)

7.2 Evaluation methodology

The experimental evaluation was carried out to test the **effectiveness** of ranked publications while testing the **performance & efficiency** of the system for various scenarios.

In a default scenario, the stream of incoming publications is framed by a count-based window which moves forward. By assuming subscriptions remain unchanged, publications are getting matched against an arbitrary samples from the set of zipfian subscriptions. We model 2 sub-scenarios according to the fraction of publications that became the matching candidates within a window:

1. *Best-matching:* All publications within a window satisfy the set of subscriptions.
2. *Random-matching:* A random number publications within a window satisfy the subscriber.

Table 7.5 explores the characteristics of used parameter values represented by symbols.

Parameter	Symbol	Value Range
Number of subscription views	S	$\approx 10M$
The size of a subscription	d_s	$2 - 32$
Total number of publications in a set	P	$100K$
Dimensionality of a publication	D	$2 - 100$
Number of delivered publications	k	$k \geq 1$
Size of count-based sliding window	w	$2 - 10K$
Error at preference values	δ	$\delta \in \mathbb{N}(0, 1)$
Decay parameter for relevancy score	ρ	$0.1 - 30$ *microseconds*
Publication similarity threshold	s	$0 \leq s \leq 1$
Number of minhash functions	m	$4 \leq m \leq 400$
Estimated error at minhash signatures	e	$0 \leq e \leq 1$
Zipf exponent	χ	$\chi \geq 1$
Number of hash tables	L	$1 \leq L \leq m$
The depth of bucket array	b	$b > 0$
The size of minhash signature vector	r	$r > 0$

Table 7.5: Symbol values on parameter space

7.3 Subscriber Effectiveness

To measure the quality or effectiveness of the matching publications, here we focus on *Best-matching* scenario. In the following, we perform experiments to measure the "goodness" of matching publications based on relevancy, freshness & diversity factors.

7.3.1 Quality of ranked publications

While controlling the number of delivered publications by the parameter k, we try to model some intrinsic properties of ranked publications to measure their quality. The quality has been considered in the view of subscriber. Top-k publications are ordered by their ranks under the computed relevancy score over a personalized subscription space. The subscription spaces are composed by natural queries that follow zipf property.

Benchmark It's observed that a natural property could be exploited as an outcome of a natural behavior. To model any natural behavior mathematically, we can use zipfian law [1]. Based on above observation, we present a better mechanism to evaluate diversity based algorithms. That is to analyze whether ranked publications exhibit some natural behavior. Hence, we provide a novel platform to compare natural behavior of ranked results by comparing their convergence into perfect zipfian distribution. Further, we experiment with other diversity methods (e.g. MAXMIN, MAXSUM, DisC) in publish/subscribe domain along with the proposed approach MAXDIVREL to extract such natural behavior.

7.3.1.1 Testing zipf law hypothesis

Here, we attempt to test the zipf law hypothesis quantitatively at the distribution of ranked publications based on MAXDIVREL. In general, zipf law is a variation applied on the power law by taking only values that are greater than some minimum.

Given the rank of elements k & the exponent χ, a discrete probability distribution obeys power law if it satisfies

$$p(k) \propto \frac{1}{k^{\chi}}$$

$$p(k) = \frac{C}{k^{\chi}}$$

where C is the normalized constant. By following Hurwitz zeta[2] general function we can derive

[1] http://www.datasciencecentral.com/profiles/blogs/why-zipf-s-law-explains-so-many-big-data-and-physics-phenomenons

[2] http://en.wikipedia.org/wiki/Hurwitz/_zeta/_function

the constant C as

$$C = \frac{1}{\sum_{n=1}^{N}(1/\ n^{\chi})}$$

where N is the number of elements in the distribution. It's easy to observe that above form of power law distribution is also the Zipfian distribution

$$zipf(k : \chi, N) = \frac{C}{k^{\chi}}$$

To determine the distribution of Top-k ranks follows the power law, we have adopted Kolmogorov-Smirnov (KS) test suggested by [49]. It's well recognized in statistical community not only to measure the fitness of data to a power law, but also to assess the goodness of the fit by comparing with other heavy-tailed distributions. The testing of power law hypothesis is done in 3 steps to exploit the natural behavior of ranked results.

i Fitting power law to the scores of ranked publications

ii Goodness of fit tests

iii Testing alternative distributions

Fitting power law By taking logarithmic scale, power law $p(k) = \frac{C}{k^{\hat{\chi}}}$ can be plotted in a straight line.

$$ln\ p(k) = \chi.ln\ k + C$$

The slope of the straight line is given by the exponent $\hat{\chi}$. We can compare the deviation of the calculated exponent with proven best values. But it's observed that this method suffers from many systematic errors at heavy logarithmic scale [50]. [49] estimated the power law exponent $\hat{\chi}$ based on well known Hill-estimator [1]. So, the estimated exponent $\hat{\chi}$ is given by:

$$s = 1 + k\ [\sum_{i=1}^{k} ln\frac{x_i}{x_{min}}]^{-1}$$

where $x_i, i = 1, ...k$ are the ranked scores such that $x_i \geq x_{min}$, and x_{min} is the smallest value for which the power law holds. We consider $x_{min} = 1$ as the estimation for discrete integer values s.t.

$$\hat{\chi} = 1 + k\ [\sum_{i=1}^{k} ln\ x_i]^{-1}$$

[1] http://sfb649.wiwi.hu-berlin.de/fedc/_homepage/xplore/tutorials/sfehtmlnode91.html

[50] noted that the exponent χ for zipf distribution should be:

$$s = \hat{\chi} - 1 = k\ [\sum_{i=1}^{k} ln\ x_i]^{-1}$$

To test the natural behavior of results, we first compute a large number of sets of ranked publications under different sizes(w) of sliding windows. Also we test such behavior across many subscriber views for an even comparison. Ranked publications are computed over a stream of $100K$ unique publications.

Ranks are computed by LSH-indexing mechanism which results k most voted candidates of Top publications from all hash tables as described in the section 5.2.6. Table 7.6 shows an example view of rank-wise votes of publications within a window. In summary, we compute 19030 such distributions to test the power law hypothesis.

Rank(k)	1	2	3	..	k_i	..	$k_{max} = w$
Votes(x_k)	x_1	x_2	x_3	..	x_{k_i}	..	$x_{k_{max}}$

Table 7.6: A sample rank-wise votes distribution of Top-k publications

Figure 7.1 shows a log-log plot of ranks & votes where the approximate straight line stretches over a large number of rank distributions under different subscriber views.

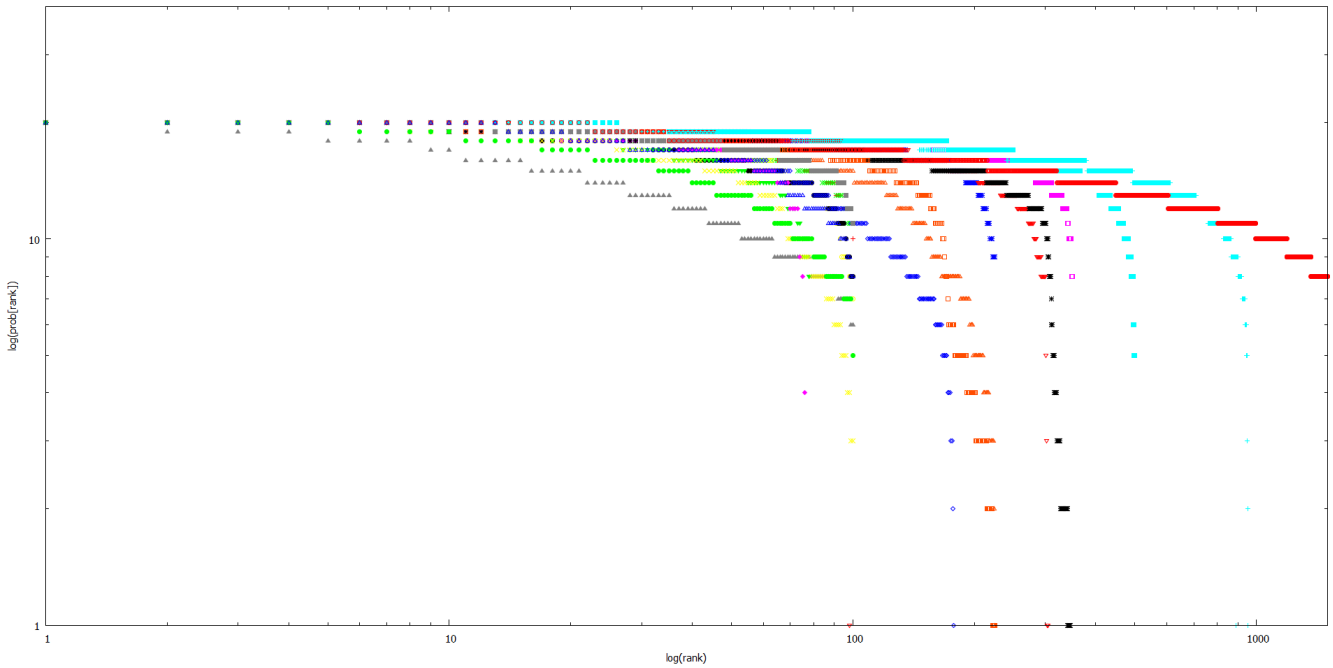


Figure 7.1: Distribution of ranked votes

First we investigate the value of zipf exponent for the distribution of ranked votes. As shown in the table 7.7 we further explore them across many subscriber views under different sizes of

windows. We found that zipf law holds remarkably well when the size of ranks distribution increases, because the value of exponent has the tendency to converge into perfect value 1. (Figure 7.2)

$k_i(k_i \leq w_i)$	Automotive	Home	Electronics	Movies	Books	Phones	Average exponent
10	2.4883	2.4934	2.4911	2.4800	2.4786	2.4985	2.4883
20	2.4744	2.4924	2.4835	2.4683	2.4815	2.4829	2.4805
50	2.4384	2.4963	2.4959	2.4921	2.4878	2.5000	2.4851
100	2.4521	2.4956	2.4847	2.4732	2.4860	2.5000	2.4819
225	2.1350	2.3518	2.0470	2.3829	2.4595	2.0012	2.2295
350	2.0753	2.0642	1.6160	2.1987	2.4503	1.5400	1.9907
500	1.821	2.0824	1.6328	2.3905	2.4344	1.2899	1.9420
1000	2.2487	2.0375	1.7185	2.3612	2.1209	1.2678	1.9591

Table 7.7: Average zipf law exponent for ranked publications in comparison with different subscriber views

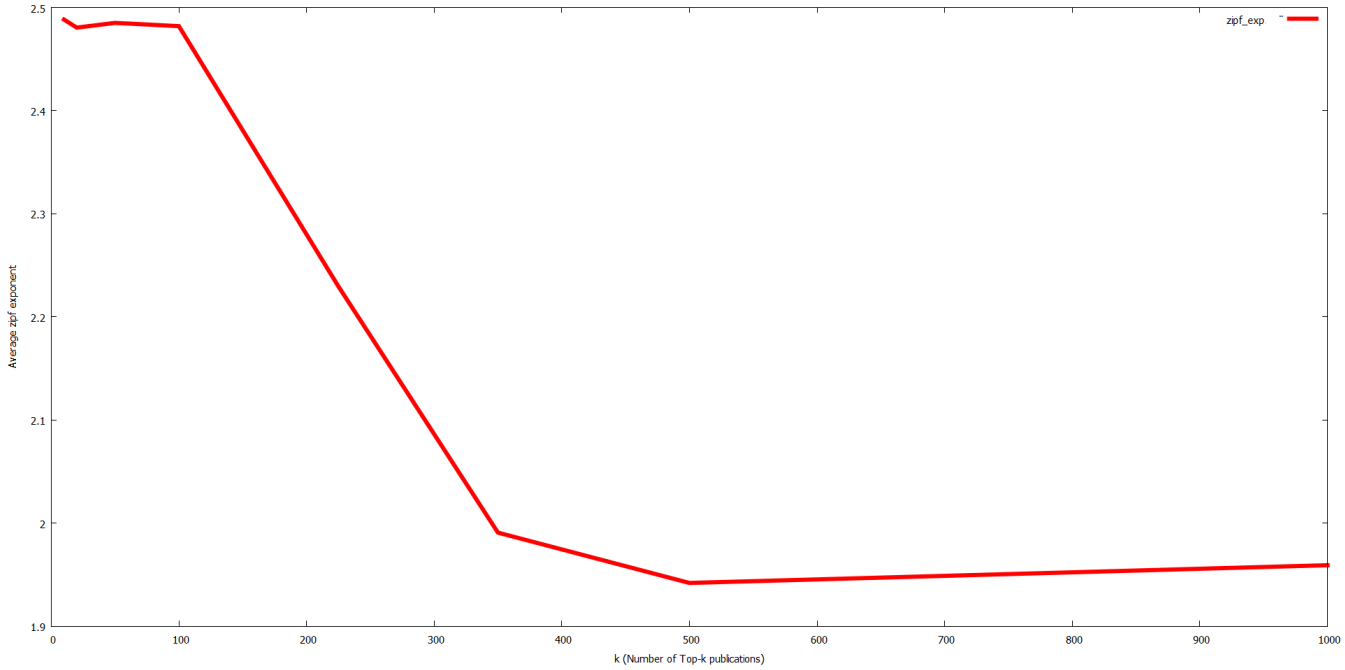


Figure 7.2: Illustration of zipf exponent values

Importantly, the zipf law exponent is around 2.0 for the rank distribution where $k \leq 1000$. We explore the reason behind the initial deviation of zipf exponent for small k values. Recall that zipf is a heavy tailed distribution. The most interesting feature of such distributions is the tail and its properties. When fitting data to a power law, we determine the portion of them that fits into the tail. When the size of distribution is low, the tail is not significant. An user can able to disregard small values if such data do not exhibit strong zipf properties.

Hence, we can conclude that there is a minimum number of ranked publications to exhibit the perfect natural behavior given the data. Our model motivates this factor by suggesting best MAXDIVREL publications that exhibit the strongest natural behavior not restricted by the parameter k.

Also we further refine above results by changing the similarity threshold $0 < s \leq 1$ in the proposed index mechanism. Indirectly, we check how our system reacts to various other LSH index parameters L & r. Table 7.8 shows an advance look of rank distributions.

Similarity Threshold(s)	Rank(k)	1	2	3	..	k_i	..	$k_{max} = w$
s_i	Votes($x_{s_i,k}$)	$x_{s_i,1}$	$x_{s_i,2}$	$x_{s_i,3}$	..	$x_{s_i,k}$	..	$x_{s_i,k_{max}}$
s_j	Votes($x_{s_j,k}$)	$x_{s_j,1}$	$x_{s_j,2}$	$x_{s_j,3}$	..	$x_{s_j,k}$	..	$x_{s_j,k_{max}}$
..	..	..	..	..	..	..	..	..

Table 7.8: An advance rank-wise votes distribution in a comparison with similarity threshold

s_i \ w	10	20	50	100	500	1000
0.35	2.4827	2.4858	2.4762	2.4800	1.9700	1.9630
0.55	2.4883	2.4805	2.4851	2.4819	1.9420	1.9591
0.85	2.4796	2.4670	2.4840	2.4790	1.9413	1.9500

Table 7.9: Average power law exponent for different k values in comparison with similarity threshold

By experimenting with different similarity thresholds we check the uncertainty of measures, or the robustness of our estimates. As table 7.9 shows, a change in the similarity threshold does not effect the natural behavior of results. Because under different windows, the calculated exponent value for ranked publications, is almost identical within the range of similarity threshold.

Goodness of fit tests We further examine the significance of results generated by the model. As a rigorous statistical test, KS denotes the maximum distance κ between the Cumulative Distribution Function (CDF)s of observed Top-k scores $f(x)$ & the perfectly fitted model $g(x)$ to the power law.

$$\kappa = max_{x \geq x_{min}} |f(x) - g(x)|$$

From $f(x)$ derived previously, we can calculate minimum κ to satisfy above condition. We denote that value as κ_1. While keeping $f(x)$ on observed Top-k scores, we re-calculate the

maximum distances κ_i for a number i of synthetically fitted power law models $g(x)$. Then the goodness of fit (p-value) is obtained by:

$$p - value = \frac{the\ number\ of\ \kappa_i\ whose\ values\ are\ greater\ than\ a\ \kappa_1}{i}$$

Rule of thumb We should generate at least $\frac{1}{4}.\xi^{-2}$ synthetic data sets if we wish our p-values to be accurate within about ξ of the true value [49]. We generate 1000 such synthetic datasets & calculate $\delta_i; 1 \geq i \geq 1000$ to simulate the goodness of fitted model $g(x)$.

p-value is a good characteristic to measure the fitness of data to the model. Table 7.10 shows the probability of seeing the fitness of results as "Good" comparing with data comes from the power law distribution. When $p - value \geq 0.5$, the model is considered to be acceptable for a good fit for zipf distribution [50]. Note that, we take the average p-value on results computed under each subscriber view.

Subscriberview \ k	20	50	100	500
Automotive	0.5494	0.4595	0.5023	0.5743
Home	0.5149	0.4894	0.5390	0.4820
Electronics	0.5435	0.4588	0.5600	0.5200
Movies	0.5226	0.4226	0.5963	0.5392
Books	0.5421	0.5191	0.5908	0.5198
Phones	0.4948	0.4049	0.5462	0.4800

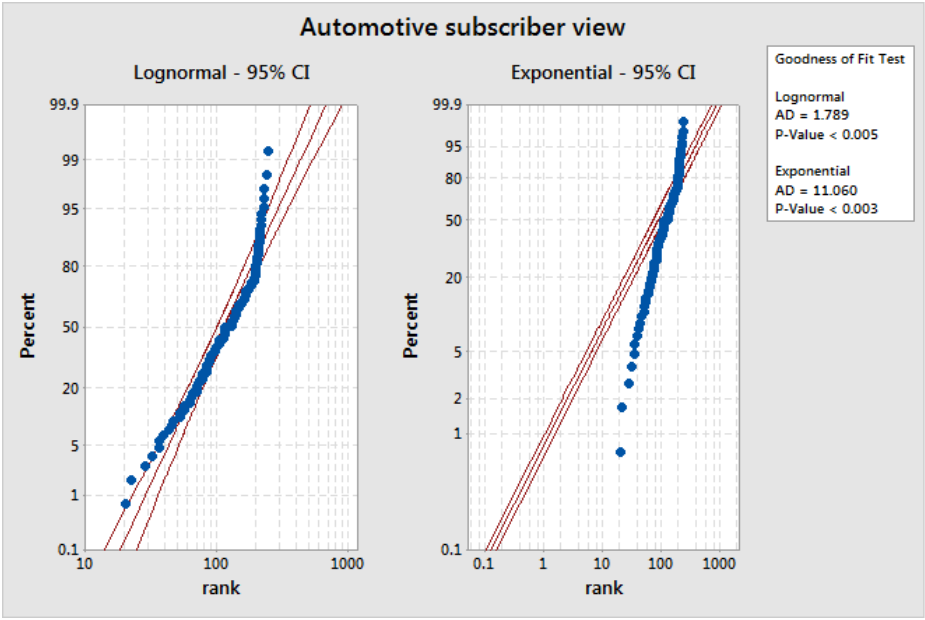
Table 7.10: p-values from KS test for ranked scores in comparison with different subscriber views

The majority of p-values at table 7.10, are greater than the significant threshold 0.5 and, clearly demonstrates our model produces a set of results which are within the best resolution of zipf law. But we observe that p-values remain almost constant for different sized distribution. This observation is a known drawback for small sized distributions [49] when detecting zipf law.

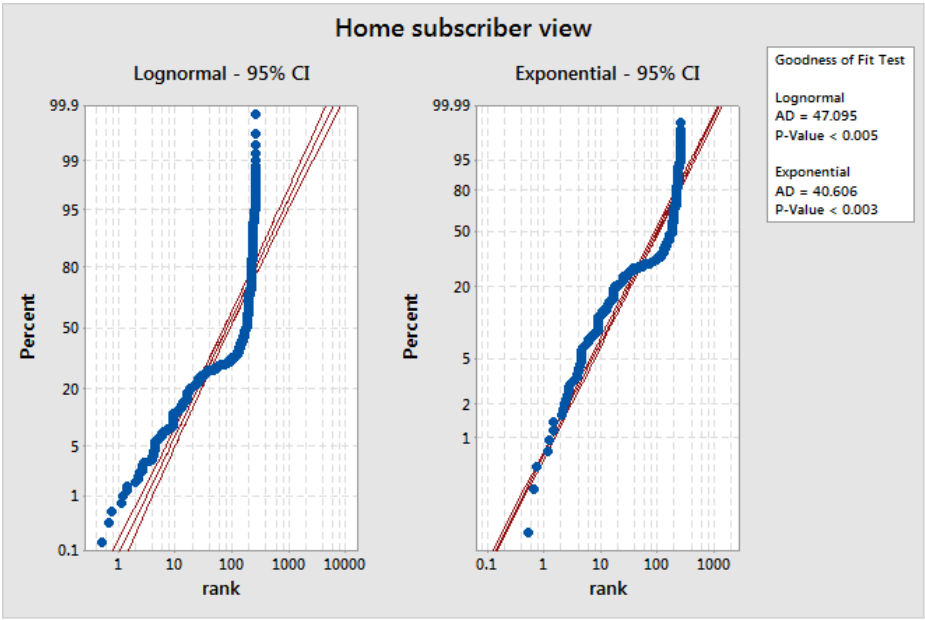
Testing alternative distributions There is a possibility to serve a better fit for our data by other heavy-tail distributions than power law. We test such possibility by re-calculating p-value for a fit to the competing distributions and compare it with the power law model. We use Minitab[1] statistical software for the analysis.

Figure 7.3 shows the comparison of average p-values under different subscriber views. We test such alternatives over the range of observed ranks under all subscriber views. We compare

[1] www.minitab.com



(a) Automotive subscriber view



(b) Home subscriber view

Figure 7.3: A comparison of p-values with alternative heavy-tail distribution

p-value with respect to the power law & two competing heavy-tailed distributions such as log-normal & exponential. Based on KS statistics, calculated p-value is sufficiently small for other distributions (e.g. log-normal p-value < 0.005 & exponential p-value < 0.003) which makes a good case for the zipf-law form for our results.

7.3.1.2 Other diversity based algorithms

As we explored in the Section 4.2.3.3, some popular diversity methods are aligned with $p-dispersion\ problem$ (e.g. MAXMIN [29]), while few with $minimum-independent\ set\ problem$ (e.g. MAXDIVREL, DisC [20]). We believe diversity has no concrete definition to be survived

due to it's biasness towards conscious practices. But if any approaches can exhibit some natural behavior, they have high probability to be survived much longer. This is the first significant try to extract such behavior from above approaches in the publish/subscribe researches.

For an even comparison, we combine relevancy with other diversity methods, thus to achieve a bi-criteria objective [29]. Note that, MAXDIVREL does consider to achieve a mono-objective between relevancy & dis-similarity. Most of diversity methods are specified using a distance function that measure the dis-similarity between items. We use Jaccard distance as the metric of dis-similarity. Top-k publications are ranked by their relevancy score.

We compute a large set of sample ranked publications on different diversity algorithms over the same stream. Then the distributions of ranks & relevancy scores are fitted into a power law model to compare their values of zipf exponent. Table 7.11 reports such exponent values over a variation of k.

Diversity method / k	MAXMIN	DisC	MAXDIVREL
10	4.6123	3.4632	2.4883
50	12.2535	2.7392	2.4851
250	46.1347	2.5381	2.1956
500	50.3878	2.1023	1.9420
1000	62.5921	2.2003	1.9591

Table 7.11: Average power law exponent for Top-k publications in comparison with other diversity methods

Figure 7.4 visualizes the deviation of zipf exponent values in log scale. A natural behavior can be detected when the zip exponent value does converge into perfect value 1. We can observe that MAXMIN diverges from other two methods which results higher zipf exponent values over the ranks of many Top-k distributions. In other hand, both MAXDIVREL & DisC do exhibit a significant natural behavior than MAXMIN method.

From above observation, diversity approaches that based on *independent dominating-set problem* have the upper hand on exhibiting natural behavior in the influence of relevancy & freshness. It's an open direction to evaluate such natural behavior of diversity employing other factors as well.

7.3.2 Accuracy of ranked publications

We compare the accuracy of ranked results computed by proposed indexing mechanism against results produced by the naive greedy algorithm 4.1 where both solve the MAXDI-

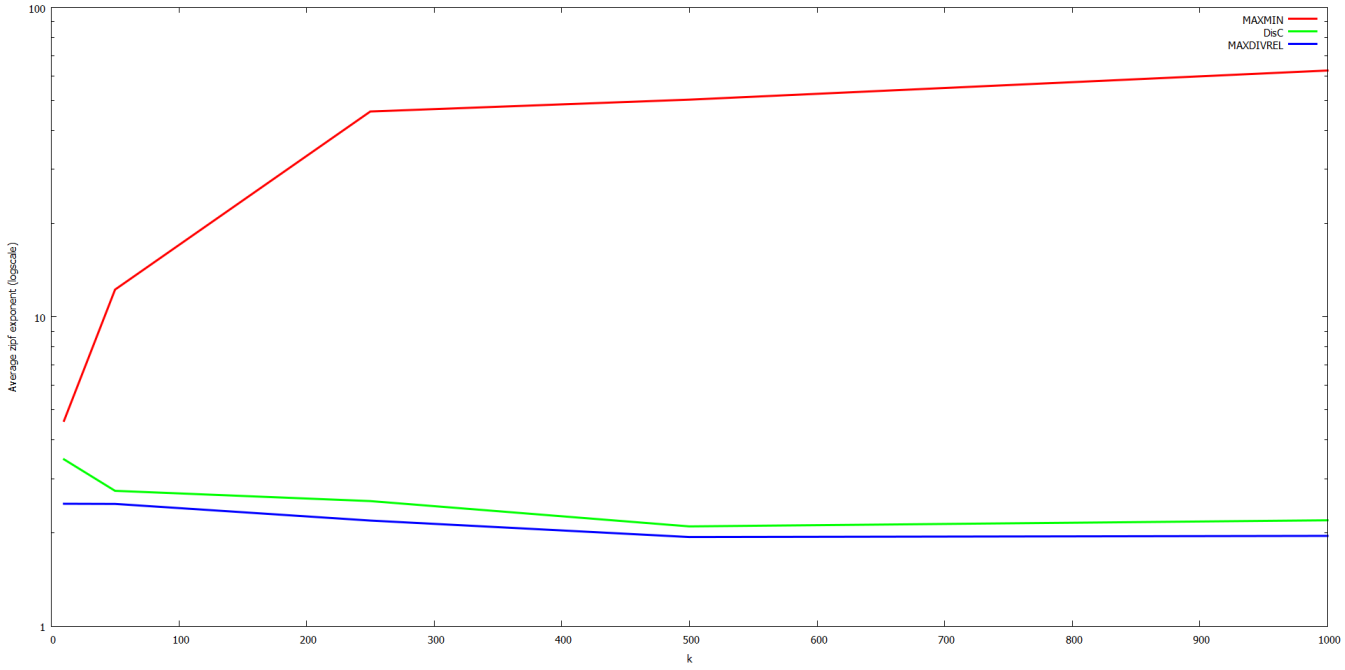


Figure 7.4: A comparison of zipf exponent values on different diversity algorithms

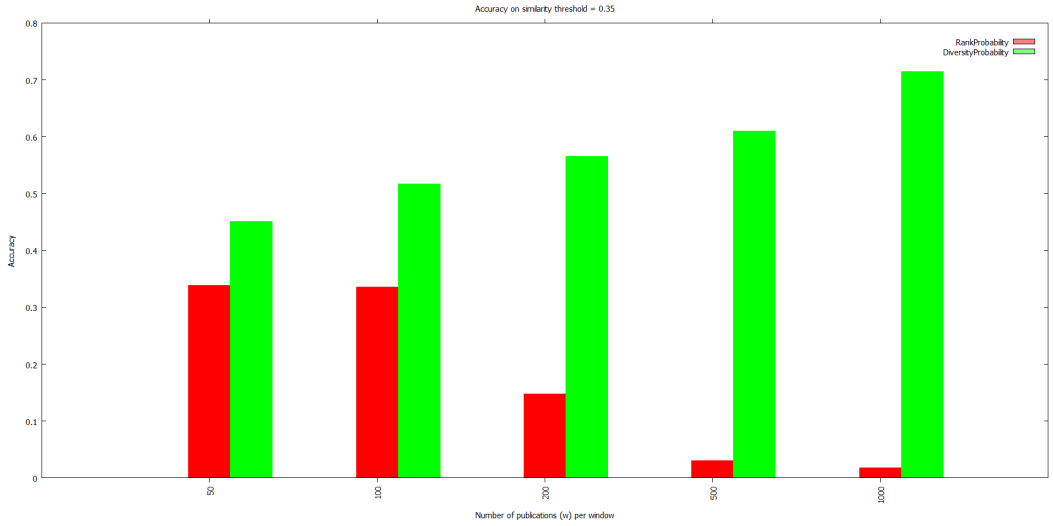
VREL k-diversity problem. By keeping the results produced by naive greedy algorithm as a benchmark, accuracy of index based results is depicted by the following two probabilities:

1. *Rank probability* of producing the same ranks of publications
2. *Diversity probability* of producing the same diverse set of results

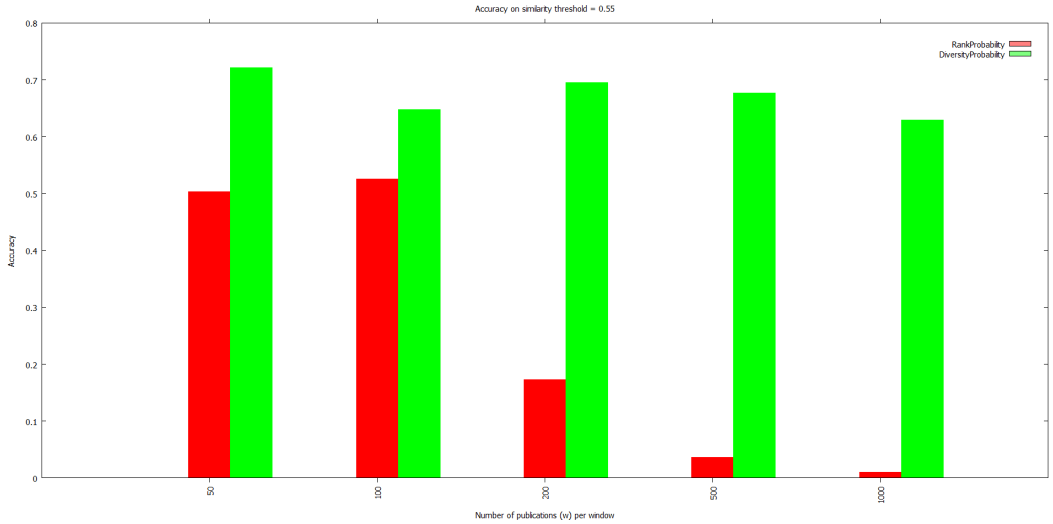
Figure 7.5 reports the accuracy of results produced by LSH index under different sizes of sliding windows, & it is further refined by LSH similarity threshold.

In average, the probability of getting the same diverse set of results (*Rank probability*) is around 0.7 under different similarity thresholds. It ensures that our mechanism produces diversified set of publications by preserving dominance & independence condition at MAXDIVREL k-diversity problem consistently.

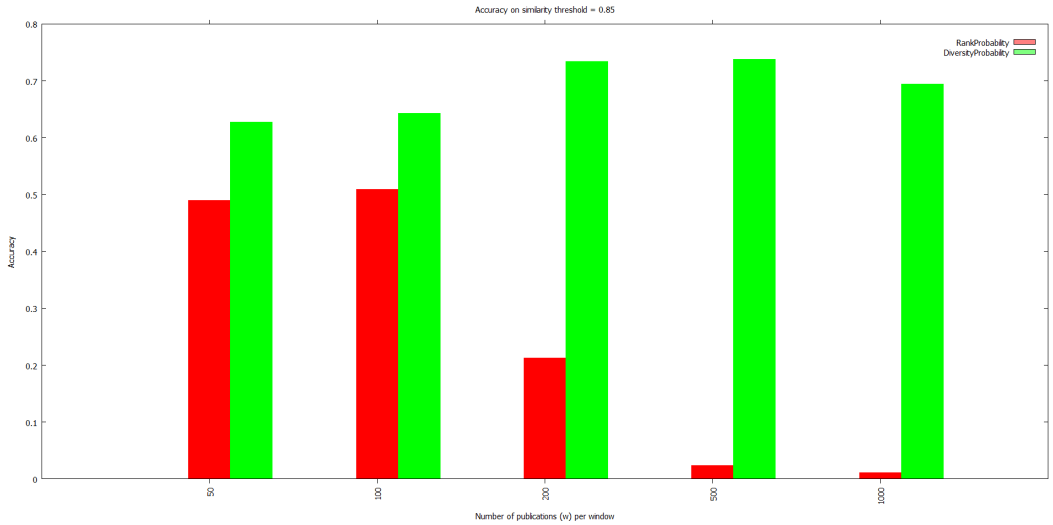
In *Rank probability*, we measure the absolute rank that Top-k results are ordered. The absolute ranks assigned to the diverse set of results by LSH index is different than the naive greedy method, which results comparatively low *Rank probability* values when the distribution increases. But it's observed the deviation of relative ranks among Top-k results are not heavily effected. Recall that, the selected set of diversified publications is ordered by the relevancy score in the naive greedy algorithm. But LSH index has used an voting based mechanism to rank them & break ties by considering the relevancy score. We believe that above two different ordering mechanism influence such behavior.



(a) similarity threshold(s)=0.35



(b) similarity threshold(s)=0.55



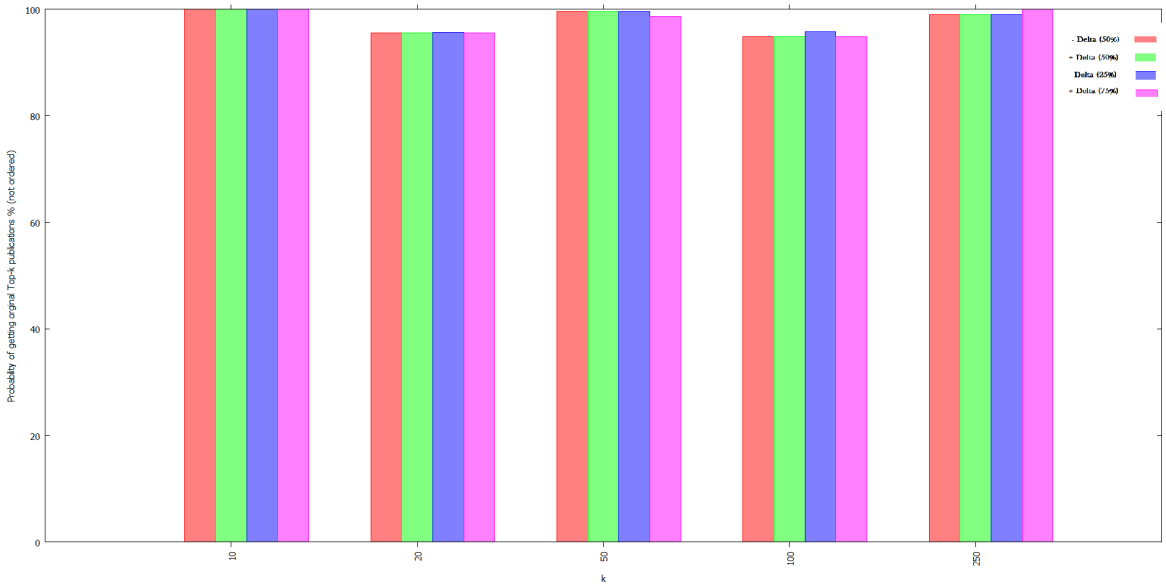
(c) similarity threshold(s)=0.85

Figure 7.5: Accuracy of ranked results produced by LSH index

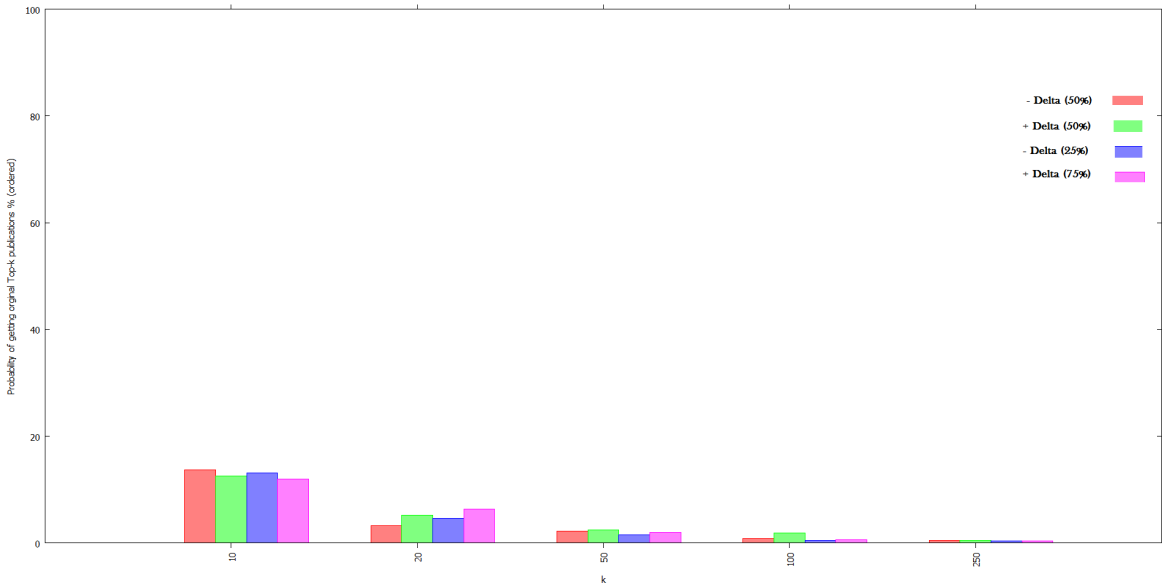
7.3.3 System resiliency

We believe system resiliency is a desirable property to satisfy for Top-k publish/subscribe models. Hence, we analyze how our system reacts for a little change in the preference values.

From the original subscription space S, we derive different subscription spaces by changing preference values over a portion of subscription tuples. A preference value has been changed by $(+|-)\delta; 0 \geq \delta \geq 1$. Then we compute ranked publications over each subscription space and, observe their deviation.



(a) Top-k publications (Unordered)



(b) Top-k publications (Ordered)

Figure 7.6: System resiliency test

Figure 7.6 shows the probability of getting the original Top-k publications for different preference values. As Figure 7.6.a reports, our model reacts reasonably well by producing the same set of Top-k publications when preferences values are changed. But it's less probable to get them in the same ranks as the original (Figure 7.6.b).

Getting the same set of unordered Top-k publications assure that our model reacts well under uncertainty at preference values to solve MAXDIVREL k-diversity problem. Because

that observation is sufficient to conclude that our model has the tendency to produce a diverse set of results. Diversity is a conservative metric that implicitly relies on the publications, while relevancy is very subjective to the user.

We can conclude that our model reacts well on producing a diversified set of publications for small preference changes. But such changes effect on the rank of Top-k publications significantly.

7.3.4 Freshness of ranked publications

We also experiment with the freshness of matching publications to model the effect of forward decaying. In the Section 4.2.2.1, we've shown that it doesn't require to re-compute time-dependent relevancy scores of all publications when a new publication arrives. Further, we assume that recent publications are relatively more important to an user than older ones.

It has been considered that the relevancy score of a publication depict the intent of a subscriber as described in the Section 4.2.1.2. Scores are usually decayed by the time. We model that effect by a forward decay function (Section 4.2.2.1) where scores are amplified on freshness.

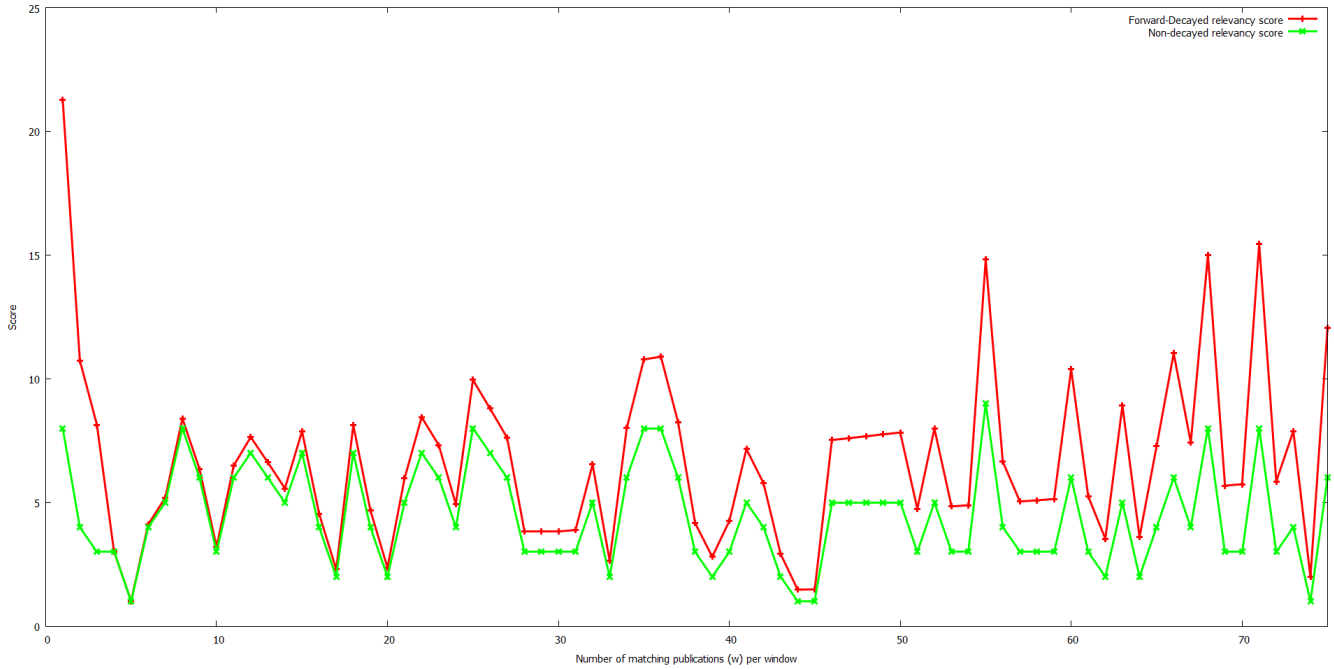
Recall that we control the rate of decaying by the decay parameter δ where $w(t_i, t) = exp(\delta\ t_i)$ denotes our decaying function at given time t & the issued time of the publication t_i.

We consider a scenario Best-Matching-Random (BMR) where more relevant publications are published randomly in a burst nature within a window.

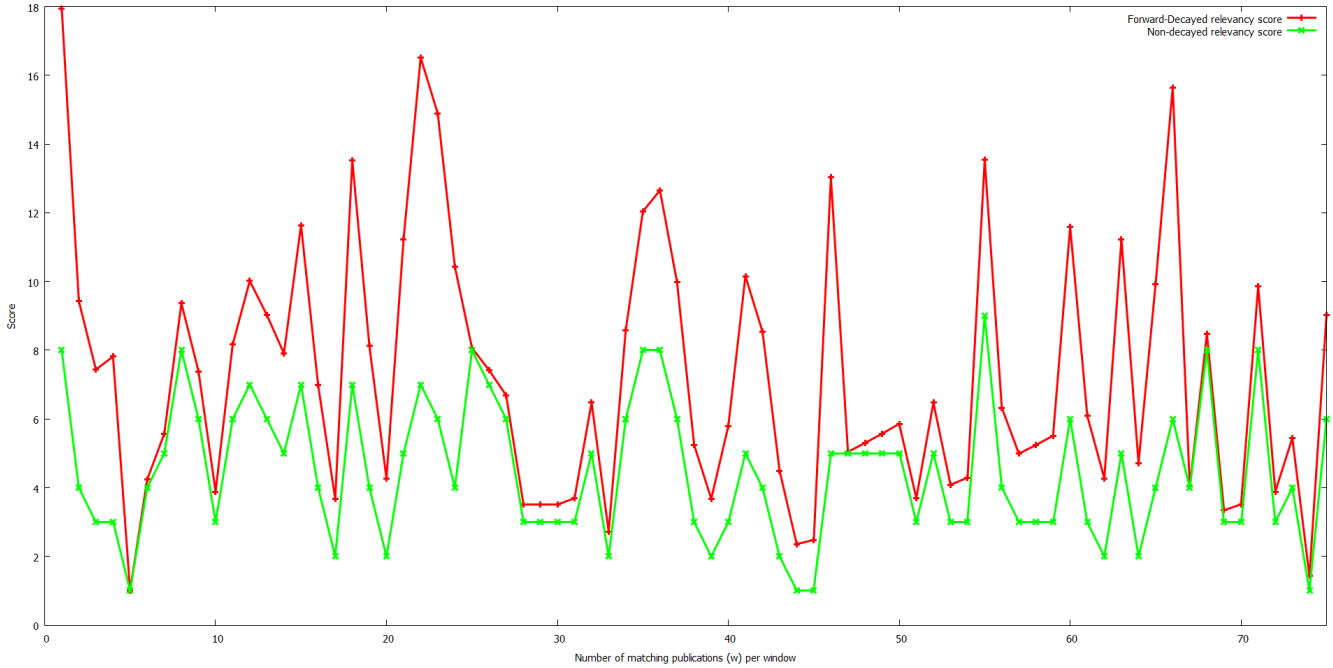
BMR scenario We explore the effect of freshness on matching publications under different window sizes w. Since we are analyzing temporal properties, we rely on time-based sliding windows here. Hence, the size of the window is varied according to the mean delay between publications, as we support Poisson flow of publications. As an example, let's say a number of 75 publications are published under a fix delay of $3000ms$. Then It represents a $3000 \times 75 = 225seconds$ time-window.

Figure 7.7 shows the comparison of relevancy scores between forward-decayed & non-decayed cases, where freshness is employed for the scores in former case.

Let's assume that Top-10 publications are retrieved from the highest-scored ones within the window. It can be observed, if we take only non-decayed relevancy scores of publications within the window, new publications which do have comparatively less relevancy scores than older ones may never be considered as important. But emphasizing the relevancy scores based



(a) Mean delay between publications = 1000ms (225 seconds window)



(b) Mean delay between publications = 5000ms (1125 seconds window)

Figure 7.7: A comparison between relevancy scores over forward-decayed & non-decayed cases

on the freshness, can assure that importance is being considered under the factors of both relevancy & freshness.

In Figures 7.7.a & 7.7.b, it's shown that less relevant publications are comparatively emphasized on freshness than previously published more older relevant publications. That is mainly influenced by the freshness of the newer publications. This observation is significant when the window size becomes larger or the delay between publications is considerably high. More importantly, above amplification has been originated by forward-decaying scores relative to the

current time. It gives a direct way to compare scores on-line without re-computing them like in general decaying function.

Analysis on decay parameter We can vary the significance of freshness by changing the decay parameter ρ in the forward-decay function $exp(\rho t_i)$ (Section 4.2.2.1). Figure 7.8 shows the variation of ρ applied on the relevancy scores of publications within a window. We set the mean interval of the issued time of publications to $5000ms$ to see the true effect of forward-decay relevancy scores at different ρ. Note that, above scores are visualized at the vertical axis by their log-scale to enhance readability.

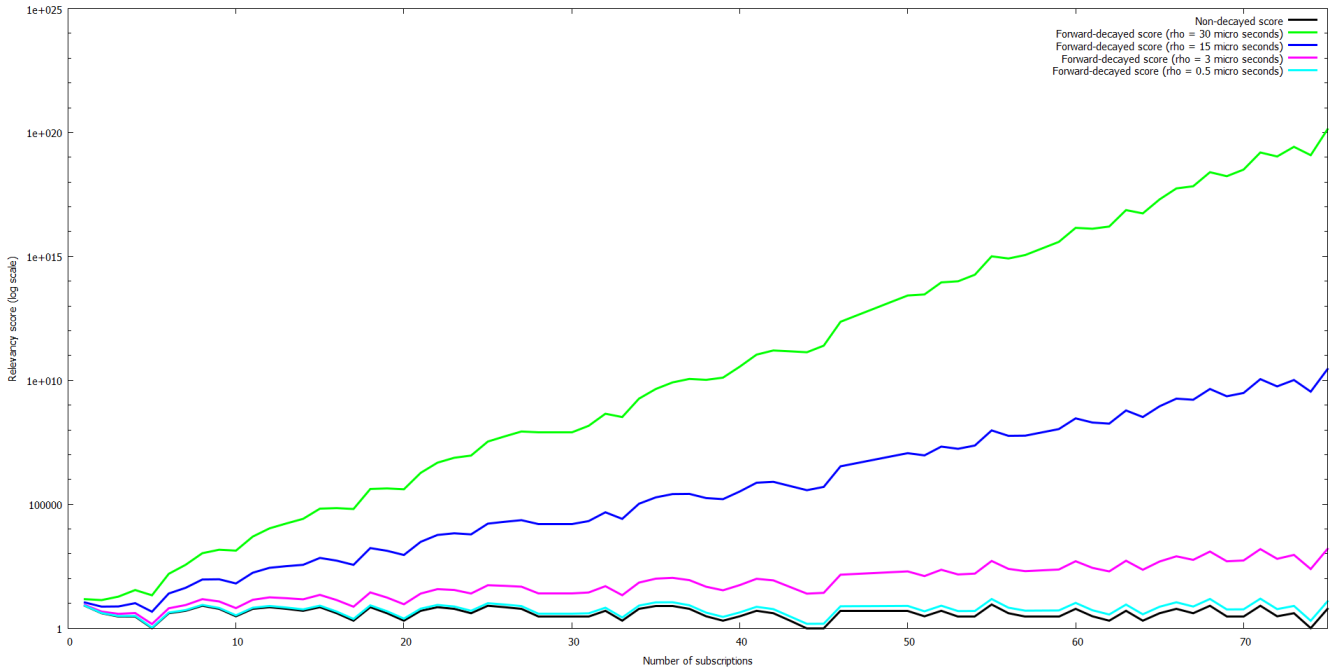


Figure 7.8: Analyzing decay parameter under different ρ rates

We can observe when ρ is high, the more the significance caused by the freshness & when it decreases, the significance of freshness becomes less on the relevancy score. The significance of freshness is proportional to ρ.

7.4 Performance & Efficiency

We do perform a variety of performance tests on various system response times against available resources. They're evaluated against existing approaches on each modules.

7.4.1 Index subscriptions

We extend opIndex to have an inverted-list based partitioning on personalized subscriptions with added user relations. It produces an inverted-graph based mechanism to index subscriptions as described at section 5.1.

7.4.1.1 Index construction time

Here, we exploit the cost of update caused by the insertion of subscriptions. We don't consider an update as a deletion of subscription, because our model inherently supports the redefining of tuple relations whenever the subscriber wants to update his view.

We report the performance of average index construction time in Figure 7.9. It is with respect to the number of subscriptions while the size(d_s) is varied randomly. The overhead of maintaining the user subscription graph is significant in modifies opIndex. It almost takes twice longer than the average construction time in opIndex. Also when the number of subscriptions increases, it takes relatively longer time to update. Because the number of implicit tuple relations behind a subscription may grow exponentially.

Importantly, modified opIndex can support a variety of attributes & operators which makes the partition scheme is effective. Since opIndex was known to be 3 orders of magnitude better than other competing mechanism [31], the overhead we observed is comparatively low.

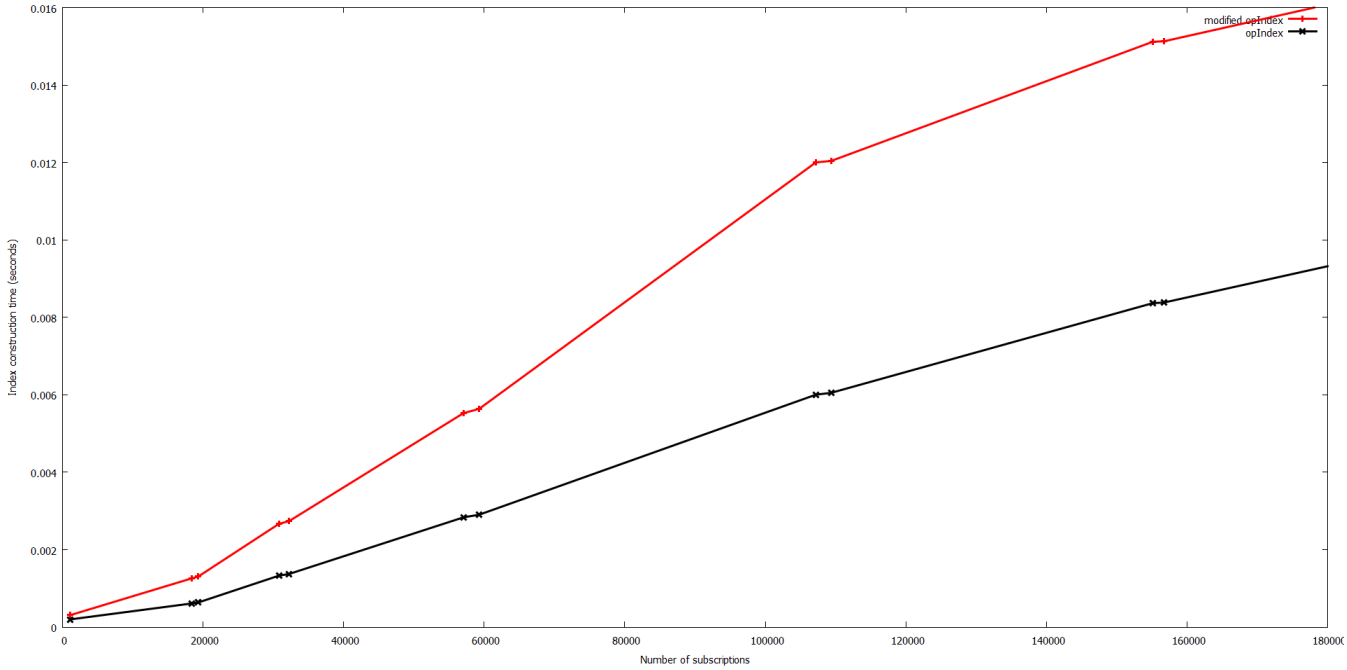


Figure 7.9: Index construction time on opIndex vs. modified opIndex

7.4.1.2 Initial matching time

A publication is matched over a subscription space to measure the quantitative relevancy score. Modified opIndex helps that process by locating subscription tuples efficiently.

Matching time with the size of subscription space The size of a subscription space is denoted by the number of query tuples which an user can subscribe into. The average matching time for a publication is reported in Figure 7.10 with an increasing size of subscriptions & different subscriber view. We take an instance under each subscriber view where an user presents an extreme set of relations in the subscription space.

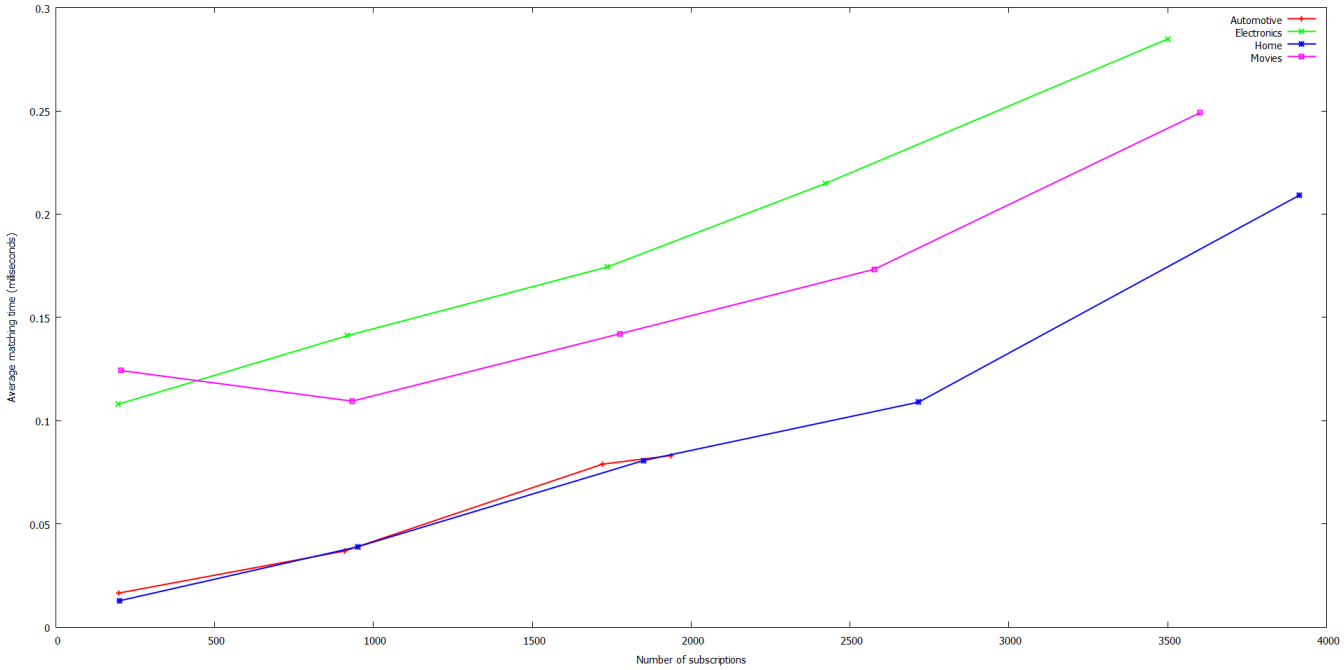


Figure 7.10: Average matching time over the size of subscription spaces under different subscriber views

The matching time is gradually increased when the size of subscription space is increased, because a publication has more chances to satisfy many relations at the subscription space. Also subscription spaces can not be always partitioned evenly. That results to have uneven length of attribute & operator partition lists. In the worst case, a publication may get unlucky to be matched even after traversing all operator lists under the matching attribute.

Matching time with the size of publication We also test, how modified opIndex reacts to the increasing size of publications. As Figure 7.11 reports, the average matching time remains almost constant when the dimension of publications increases. This is a significant advantage of inverted-index mechanisms over tree or grid based index methods.

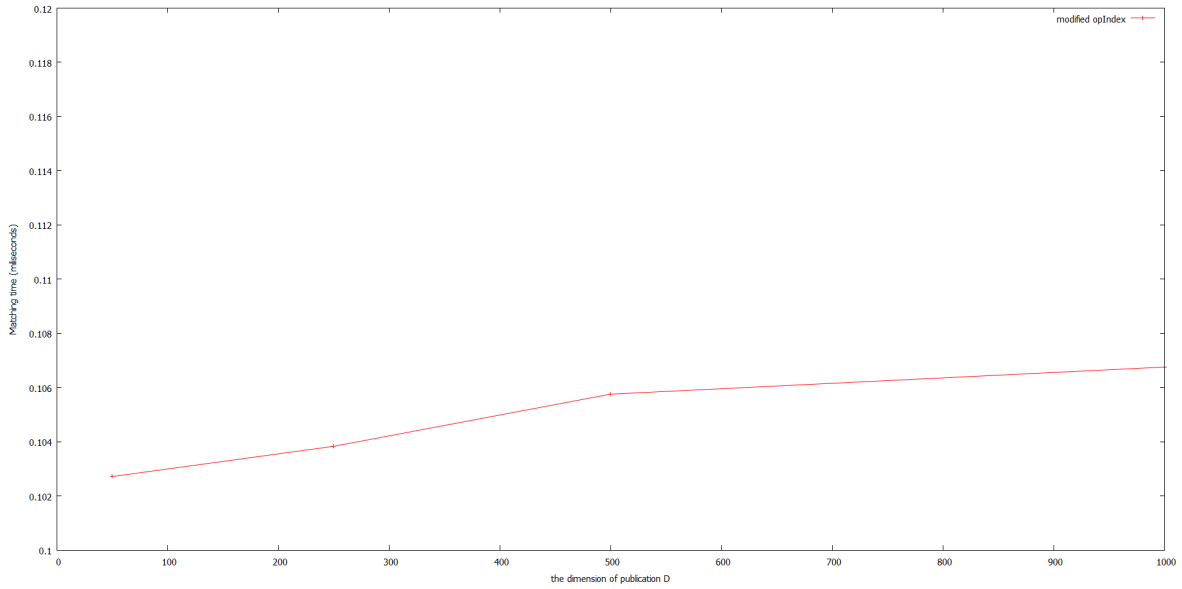


Figure 7.11: Average matching time over the size of publications

7.4.2 Index publications

Ranked publications are computed over a stream of publications by solving MAXDIVREL continuous k-diversity problem. In this section, we experimentally evaluate the performance of LSH based indexing mechanism for dynamically computing k-diverse subsets.

For simplicity we refer the batch-wise index as *BLSH* & incremental index as *ILSH*. Also we evaluate the performance of index mechanisms with the naive greedy algorithm (*NAIVE*) that doesn't rely on an indexing mechanism.

7.4.2.1 Batch-wise Top-k computation

First, we evaluate the cost of building LSH index in batch-wise over a stream of publications. Ranked results are computed over a batch of publications continuously under each window (Figure 7.12).

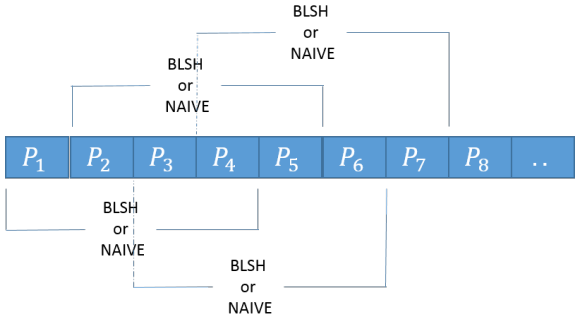


Figure 7.12: Batch-wise Top-k computation by BLSH & NAIVE

Given the expected error ($e = \frac{1}{\sqrt{m}}$) for estimating the number of minhash functions, the upper bound of such functions is $m = \frac{1}{e^2}$ [1]. For an example, we require $m = 100$ number of hash functions to estimate the correct Jaccard similarity between publications within an expected error $e \leq 0.1$. Note that other LSH parameters L &r are dependent on m as described in the section 5.2.4.1.

Figure 7.13 reports the real-time cost of constructing BLSH by varying the estimated error e or indirectly the number m of minhash functions used.

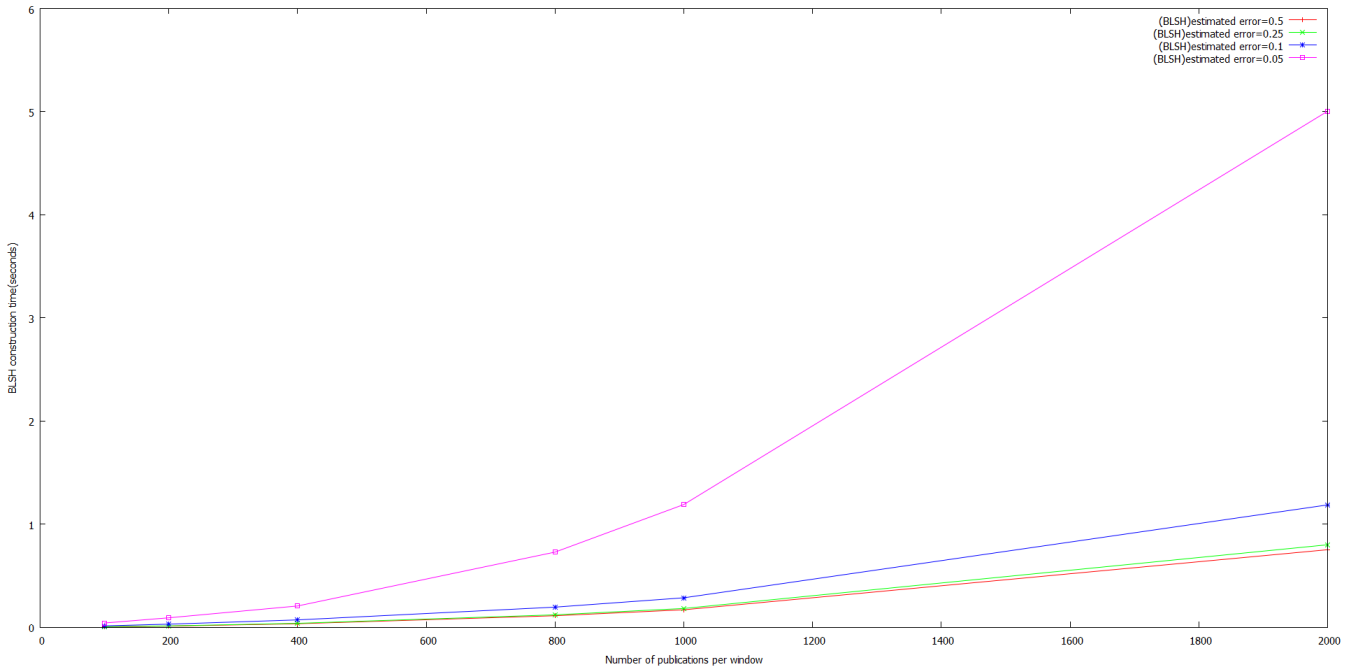


Figure 7.13: BLSH construction + matching time in comparison with estimated error e

BLSH construction time is gradually increased when the number of publications within a window to be indexed. Also we can observe, it takes comparatively longer time when e reduced or m increased. For effective results, we believe there is a minimum expected error e_{min}. Index construction time should be low for efficient results. It's easy to observe the trade-off between efficiency & effectiveness.

How much accuracy do we sacrifice by comparing small minhash signatures? We try to answer this question so as the community [48]. Let's assume we have a non-normal distribution & take N number of sample distributions randomly that is far from being normal. If you take enough samples N & analyze the probability distribution of sample mean values, itself will be normal based on *central limit theorem*[2].

[1] http://en.wikipedia.org/wiki/MinHash
[2] http://en.wikipedia.org/wiki/Normal/_distribution

For min-hashing, we try to estimate the similarity between two publications by taking common signatures. It is shown, when the Jaccard similarity between two signatures is p, it is the same probability of being the common signature as well (Section 5.2.7.1). So, any two publications may not have a common signature at the probability of $1 - p$. This probability values has characterized the *Bernoulli distribution*[1]. The standard deviation of *Bernoulli distribution* is

$$SD(Bernoulli) = \sqrt{p.(1-p)}$$

It is observed that $SD(Bernoulli)$ is maximum when $p = 0.5$ [2].

$$SD(Bernoulli)_{max} = (0.5)$$

Also an average N number of Bernoulli random samples will have a normal distribution at a standard deviation of:

$$SD(Bernoulli)' = \sqrt{\frac{p.(1-p)}{N}}$$

$$SD(Bernoulli)'_{max} = \frac{0.5}{\sqrt{N}}$$

Based on *3-sigma rule*, the true estimation of probability p resides within a confidence interval of 99%. As an example when $N = 200$, the true range of p should be:

$$0.5 - \frac{0.5}{\sqrt{200}} \geq p_{true} \geq 0.5 + \frac{0.5}{\sqrt{200}}$$

$$0.43 \geq p_{true} \geq 0.57$$

Within a confidence interval of 99%, our model agrees to provide results at a minimum estimated error $e_{min} = 0.07$. Or, in other terms, the maximum number m_{max} of minhash functions is 200 under the success probability $p = 0.5$ which in turns to be the similarity threshold $s = 0.5$.

BLSH Top-k matching time in comparison with the size of publications BLSH performs well comparatively when the size D of publications increases. Because LSH methods quantize the publications into short minhash signatures, Top-k publications can be computed quickly. (Figure 7.14)

[1] http://en.wikipedia.org/wiki/Bernoulli/_distribution
[2] http://www.cl.cam.ac.uk/ jgd1000/binomdata.html

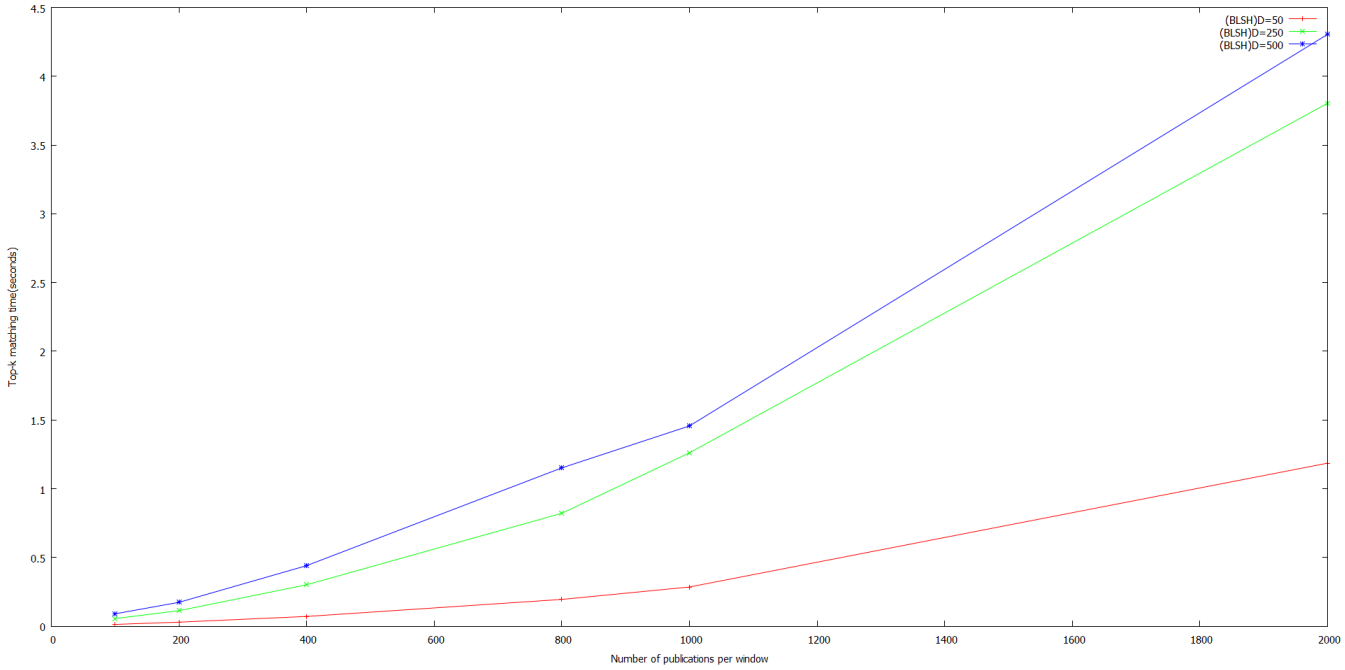


Figure 7.14: Average BLSH Top-k matching time in comparison with size D of publications

NAIVE Top-k matching time in comparison with the size of publications Also we noticed that Top-k matching time is increased much faster in NAIVE method when the size D of publication increases.(Figure 7.15)

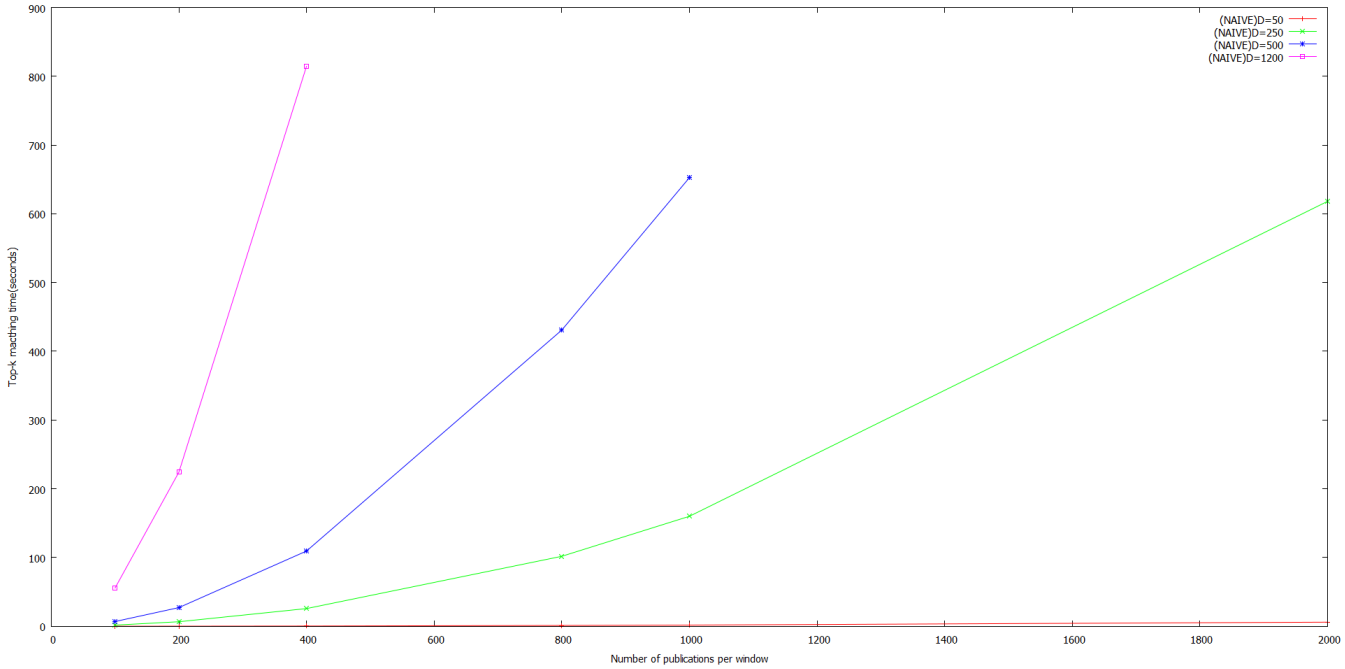


Figure 7.15: Average NAIVE matching time in comparison with size D of publications

BLSH vs. NAIVE For solving MAXDIVREL k-diversity problem over a batch of publications within a window, BLSH performs really well when comparing it's performance with NAIVE.

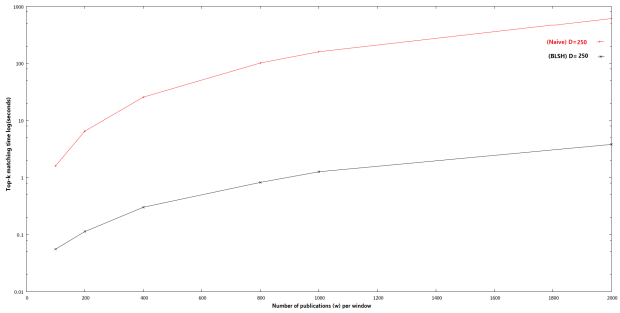


Figure 7.16: BLSH vs. NAIVE when D=250

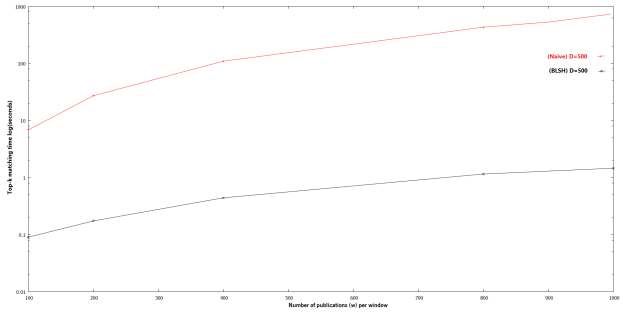


Figure 7.17: BLSH vs. NAIVE when D=500

Figures 7.16 & 7.17 report Top-k matching time at log scale for both BLSH & NAIVE methods over high-dimensional publications.

7.4.2.2 Incremental Top-k computation

Earlier we propose an incremental LSH index (ILSH) mechanism to avoid the curse of re-calculating the neighborhood for solving MAXDIVREL continuous k-diversity problem (Section 5.2.6). ILSH produces ranked results incrementally in current window without neglecting the already computed neighborhood at previous sliding windows.

ILSH Top-k matching time in comparison with the size of publications Figure 7.18 reports the cost of building ILSH on-line over a stream of different sized publications. Publications are inserted to the specific bucket at each hash table by pruning less probable candidate solutions. The pruning rule is pessimistic at each hash table by considering any publication will be lucky enough to be in a bucket with it's near neighbors (Section 5.2.2).

For high-dimensional publications, ILSH incremental construction cost is slightly increased. Because ILSH needs to maintain an universal characteristic matrix over the publication space.

ILSH update cost Table 7.12 reports the average update cost of ILSH when new publication arrives in a comparison with the size D of publications over the stream.

D	50	250	500	1000
ILSH update cost (seconds)	0.165	0.314	0.523	0.917

Table 7.12: ILSH update cost

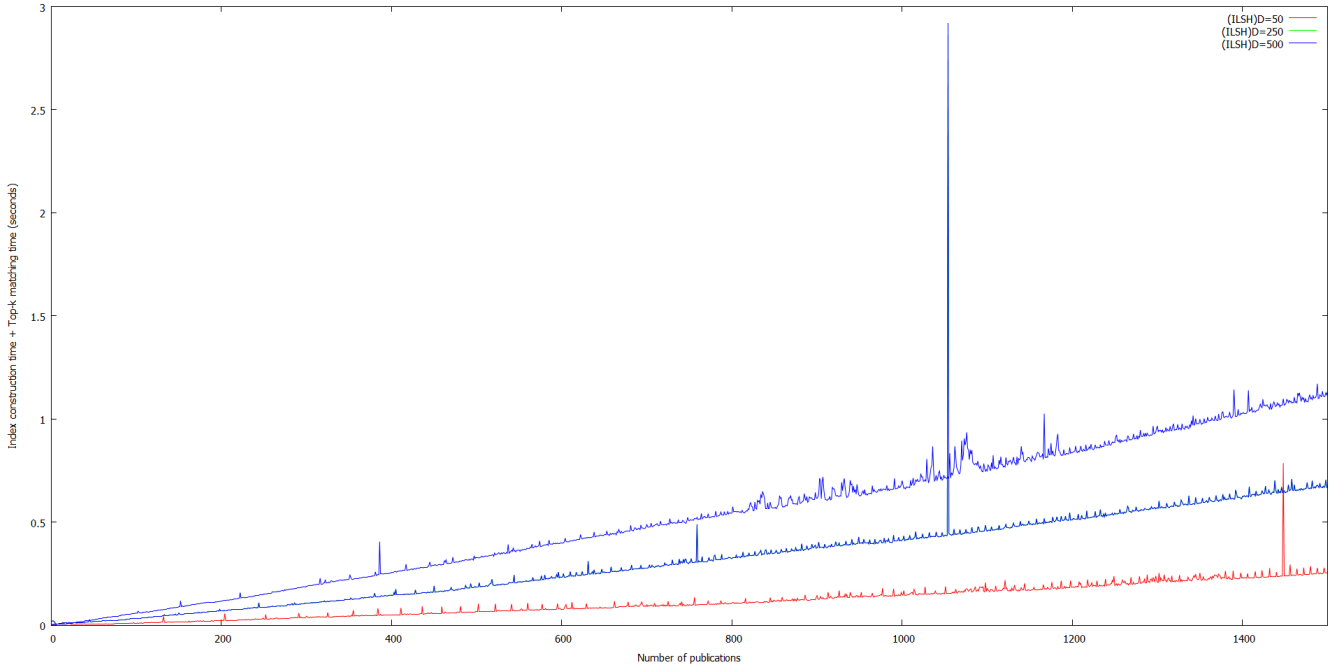


Figure 7.18: Average ILSH Top-k matching time in comparison with size D of publications

ILSH can compute a list of ranked publications incrementally at a window in average 0.165 seconds under constrained system resources when the size of publications is around 50. New ranked publications are always indexed, once ILSH is being updated.

ILSH vs. BLSH vs. NAIVE Figures 7.19 & 7.20 report the comparison of performance of ILSH with both BLSH & NAIVE method over the same stream of publications. Note that, Top-k matching time is represented at log scale to enhance readability.

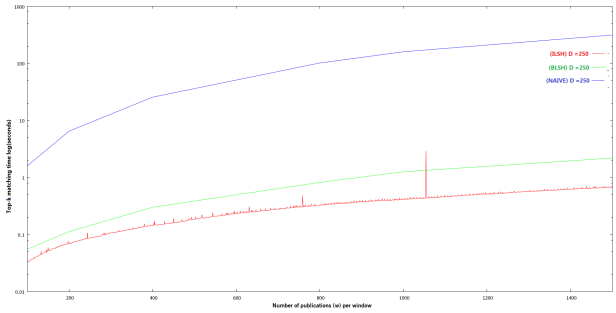


Figure 7.19: ILSH vs. BLSH vs. NAIVE when D=250

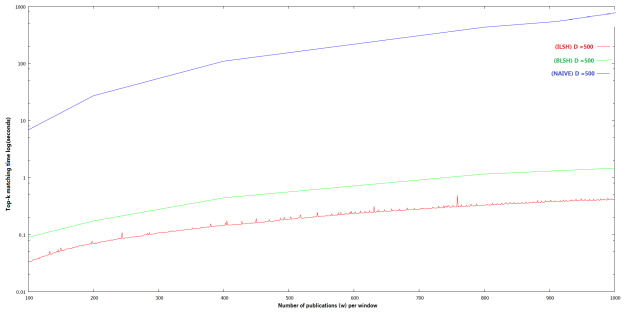


Figure 7.20: ILSH vs. BLSH vs. NAIVE when D=500

Because ILSH avoids re-computing previous neighborhoods, it's capable to update Top-k publications in consecutive windows incrementally. That results ILSH to perform even better than BLSH.

Chapter 8

Conclusion & Future work

In our study, we extend the concepts of publish/subscribe paradigm, by proposing a Top-k publish/subscribe model with an advanced ranking mechanism. We present the concepts of personalized subscription graph to enhance the user expressiveness in the matching publications. Also an existing indexing mechanism to locate subscription tuples was extended to support personalized subscription space based on inverted-lists.

The model integrates the importance of publications to produce a diverse set of results across a stream of publications. The diversity approach was formulated as *MAXDIVREL* continuous k-diversity problem based on neighborhoods of streaming publications. Since locating *MAXDIVREL* subset of publications is a NP-Hard problem, we propose a greedy approximation algorithm. Further, we avoid the re-calculation of neighborhood over sliding windows by proposing an incremental index mechanism based on Locality Sensitive Hashing (LSH). The indexing mechanism works in a dynamic setting where the publications to be diversified change over time.

We have fully implemented our approach as a prototype model in conjunction with many cloud service modules. The model was designed to scale elastically on top of Amazon Web Services (AWS).

The presented experimental evaluation has shown that Top-k ranked results produced by MAXDIVREL diversity method does exhibit strong natural behavior than methods based on *p-dispersion*, which validates the effectiveness of our approach. A MAXDIVREL diverse set of results produced by LSH indexing mechanism gain high accuracy, while increasing the efficiency of the matching process by reducing the processing time very significantly over the naive greedy method.

8.1 Top-k publish/subscribe applications

In this section, we highlight few use cases where our proposed Top-k publish/subscribe model will be rewarded.

Personalized news-feed It's not long ago Facebook CEO wanted to build the perfect personalized newspaper for every person in the world[1]. Because each Facebook user is approximately exposed to more than 1500 stories per day, but an average user only engaged with 100 stories from the current news feed. We can recast of producing the personalized news-feed as an instance of Top-k publish/subscribe. If we can form a personalized subscription space from the intent of the user, the news-stories as publications can be ranked by the importance. Thus, we can avoid the appearance of similar news-stories, by producing a diverse set of stories which can represent the whole news-feed.

Diverse set of Twitter trends Twitter as a leading social network, suggests users a list of trending topics as "hash-tags" to better explore underlying information. But most of the time, it fails to engage with people who do not have an interest over the given subjects. Ideally, a person would like to know about trending "hash-tags" among his followers. Once possible way to depict the relationship with his followers as a subscription space. Then streaming tweets as publications can be matched over the space to produce personalized a set of hash-tags. It would be ideal to present a set of Top-k hash-tags which can represent a wide variety of information content.

Social Annotation of news-stories Recently Google employed Top-k publish/subscribe model to annotate news-stories with Top-k tweets in real-time. So it allows efficient serving of page-views with a fresh set of Top-k tweets, by considering news-story as a subscription while tweets as incoming publications. By taking MAXDIVREL diversity into account, it would provide a diverse set of Top-k tweets, that better represent the given story.

[1] http://www.businessinsider.com/mark-zuckerberg-wants-to-build-a-perfect-personalized-newspaper-2014-11

8.2 Future Work

There are many directions for future works considering many aspects.

- By exploring other suitable use-cases to apply proposed model & developing prototype applications, will lead Top-k publish/subscribe models to be better accepted in both research & industrial community.
- Producing Top-k results as an output of complex events detected in the stream will be an advance step forward.
- Developing LSH based index over multi-threaded distributed environment can lead into high query performance.
- Applying publisher friendly relevancy function to have a hybrid Top-k matching is an interesting direction to work on.

We believe our approach has succeeded in achieving its goal as a system and, provide an excellent platform to develop Top-k publish/subscribe applications in the future. We strongly suggest Top-k publish/subscribe models as a significant replacement for Boolean publish/subscribe models as it's expressive nature of filtering results.

Appendices

Appendix A

Naive algorithms

Algorithm A.1 Constructing & updating personalized subscription graph

Input: A set of subscriptions $\mathbb{S}; s.t. S \in \mathbb{S}$ where $s_i \in S$ denotes the subscription tuples
Output: A directed graph $G(V, E)$ with V vertices & E edges

```
Initialize G(V, E) ← {}; subscription graph
Initialize T[ ] ← {} the set of tuple combinations
for ∀S ∈ 𝕊 do
    for ∀s_i ∈ S do
        create a vertex v_i on the subscription tuple s_i
        if v_i ∉ V then
            V ← V + v_i
        end if
    end for
    T{u_i, v_i}[ ] ← combination(V, 2)
    for ∀{u_i, v_i} ∈ T do
        preference ratio ← pref(u_i) / pref(v_i)
        if preference ratio ≥ 1 then
            E ← E + (u_i, v_i)
        else
            E ← E + (v_i, u_i)
        end if
    end for
end for
return G(V, E)
```

Algorithm A.2 Computing the relevancy score (Naive method)

Input: User subscription graph $G(V, E)$ and, a publication p_i to be matched with $\forall tuple \in p_i$
Output: Relevancy score $r(p_i)$

```
Initiate r(p_i) ← 0
Initiate U ← ∅;
for tuple ∈ p_i do
    v_i ← locate vertex(tuple)
    U ← U + v_i
end for
for ∀x_i ∈ U do
    for ∀y_i ∈ U do
        if x_i.hasEdge(y_i) then
            Edgee ← Edge(x_i, y_i)
            r(p_i) ← r(p_i) + weight(e)
        end if
    end for
end for
r(p_i) ← λr(p_i) + (1 − λ)|U|
return r(p_i)
```

Appendix B

Index based algorithms

Algorithm B.1 Insertion algorithm for modified opIndex

Input: A subscription s
Output: an updated opIndex

$Index\ I \leftarrow Index_{prev}$
for $\forall predicate(s_i) \in s$ **do**
  $Attribute \leftarrow Attribute(predicate(s_i))$
  $Operator \leftarrow Operator(predicate(s_i))$
  if $Attribute \notin I$ **then**
    $I(Attribute) \leftarrow Attribute$
  end if
  $L_{Attribute} \leftarrow I(Attribute)$
  if $Operator \notin L_{Attribute}$ **then**
    $L_{Attribute}(Operator) \leftarrow Operator$
  end if
  if $predicate(s_i) \notin L_{Attribute}(Operator)$ **then**
    $L_{Attribute}(Operator) \leftarrow L_{Attribute}(Operator) + predicate(s_i)$
  end if
end for
return I

Algorithm B.2 Computing the relevancy score (Index based method)

Input: Modified opIndex $I(V, E)$ and, a publication p_i to be matched with $\forall elements \in p_i$
Output: Relevancy score $r(p_i)$

```
Initiate r(p_i) ← 0
Initiate U ← ∅;
for element ∈ p_i do
    Attribute ← Attribute(tuple)
    Value ← Value(tuple)
    Set_Attribute) ← I(Attribute)
    for ∀L_Operator ∈ Set_Attribute do
        for ∀predicate ∈ L_Operator do
            if Value Operator Value(predicate) then
                v_i ← vertex(tuple)
                break
            end if
        end for
    end for
    U ← U + v_i
end for
for ∀x_i ∈ U do
    for ∀y_i ∈ U do
        if x_i.hasEdge(y_i) then
            Edgee ← Edge(x_i, y_i)
            r(p_i) ← r(p_i) + weight(e)
        end if
    end for
end for
r(p_i) ← λr(p_i) + (1 − λ)|U|
return r(p_i)
```

Appendix C

LSH based algorithms

Algorithm C.1 Fast Min-Hashing algorithm

Input: Let N the number of rows in the characteristic matrix, and
$SIG(i, c)$ be the signature matrix where i define the hash function and column c the publication.
Output: Updated signature matrix $SIG(i, c) \forall i \forall c$

```
∀i ∀c SIG(i, c) ← ∞
initialize m random hash functions h_1, ....h_m s.t. h : N → [N]
for ∀r ∈ N do
    compute h_i(r)
    for ∀column c ∈ SIG do
        if c = 1 then
            for ∀i ∈ SIG(i, c) do
                SIG(i, c) ← min(SIG(i, c), h_i(r))
            end for
        end if
    end for
end for
return SIG
```

Bibliography

[1] C. James E. Short, Roger E. Bohn, "How Much Information? 2010 Report on Enterprise Server Information," *UCSD Global Information Industry Center*, pp. 1–38, 2011.

[2] M. Hilbert, "How to Measure How Much Information ? Theoretical , Methodological , and Statistical Challenges for the Social Sciences Introduction," vol. 6, pp. 1042–1055, 2012.

[3] J. F. Gantz and S. Minton, "The Diverse and Exploding Digital Universe An Updated Forecast of Worldwide," 2011.

[4] G. Cugola and A. Margara, "Processing Flows of Information : From Data Stream to Complex Event Processing," vol. V, no. i, pp. 1–70, 2012.

[5] A.-m. Kermarrec, "Large-Scale Publish / Subscribe Systems : State of the Art and Research Directions What is this Tutorial About," 2008.

[6] K. Pripuži and K. Aberer, "Top-k / w Publish / Subscribe : Finding k Most Relevant Publications in Sliding Time Window w*," pp. 127–138, 2008.

[7] K. Pripužić, I. Podnar Žarko, and K. Aberer, "Top-k/w publish/subscribe: A publish/subscribe model for continuous top-k processing over data streams," *Information Systems*, vol. 39, pp. 256–276, Jan. 2012.

[8] A. Shraer, M. Gurevich, M. Fontoura, and V. Josifovski, "Top-k Publish-Subscribe for Social Annotation of News," *Proceedings of the VLDB Endowment*, vol. 6, no. 6, pp. 385–396.

[9] M. Drosou, K. Stefanidis, and E. Pitoura, "Preference-aware publish/subscribe delivery with diversity," *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems - DEBS '09*, p. 1, 2009.

[10] M. Drosou, "Ranked Publish / Subscribe Delivery Extended abstract for DEBS PhD Workshop," *PhD Workshop, in conjunction with the DEBS 2009 Conference*, 2009.

[11] T.-k. Matching, P. Subscribe, M. Sadoghi, and H.-a. Jacobsen, "Relevance Matters : Capitalizing on Less, Top-k matching in publish/subscribe," *Data Engineering (ICDE), 2012 IEEE 28th International Conference, Washington, DC*, vol. ISSN 1063-, pp. 786 – 797, 2012.

[12] S. E. Whang, H. Garcia-molina, C. Brower, J. Shanmugasundaram, S. Vassilvitskii, and E. Vee, "Indexing Boolean Expressions," *Proceedings of VLDB Endowment, Lyon, France*, 2009.

[13] A. Machanavajjhala, E. Vee, M. Garofalakis, and J. Shanmugasundaram, "Scalable Ranked Publish / Subscribe," *Proceedings of the VLDB*, vol. 1, no. 1, pp. 451–462, 2008.

[14] N. Parikh and N. Sundaresan, "Beyond relevance in marketplace search," in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, (New York, NY, USA), pp. 2109–2112, ACM, 2011.

[15] A. D. Sarma and N. Sundaresan, "E-commerce Product Search : Personalization , Diversification , and beyond," pp. 189–190, 2014.

[16] X. Lu, X. Li, T. Yang, Z. Liao, W. Liu, and H. Wang, "Rrps: A ranked real-time publish/subscribe using adaptive qos," vol. 5593, pp. 835–850, 2009.

[17] M. Drosou, E. Pitoura, and K. Stefanidis, "Preferential Publish / Subscribe," in *Personalized Access, Profile Management, and Context Awareness: Databases*, pp. 9–16, 2008.

[18] M. Drosou and E. Pitoura, "Search result diversification," *ACM SIGMOD Record*, pp. 41–47, 2010.

[19] D. Souravlias, M. Drosou, K. Stefanidis, and E. Pitoura, "On novelty in publish/subscribe delivery," *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, pp. 20–22, 2010.

[20] M. Drosou and E. Pitoura, "Disc diversity: Result diversification based on dissimilarity and coverage," *Proc. VLDB Endow.*, vol. 6, pp. 13–24, Nov. 2012.

[21] M. Drosou and E. Pitoura, "Diversity over Continuous Data," in *IEEE Computer Society Technical Committee on Data Engineering*, pp. 1–8, 2009.

[22] M. Drosou and E. Pitoura, "Dynamic Diversification of Continuous Data," in *15th International Conference on Extending Database Technology, ACM*, pp. 216–227, 2012.

[23] M. Drosou and E. Pitoura, "Diverse Set Selection Over Dynamic Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 5, pp. 1102–1116, 2014.

[24] E. Minack, W. Siberski, and W. Nejdl, "Incremental diversification for very large sets: a streaming-based approach," *... of the 34th international ACM SIGIR ...*, pp. 585–594, 2011.

[25] C. Clarke and M. Kolla, "Novelty and diversity in information retrieval evaluation," in *SIGIR*, 2008.

[26] R. Santos, P. Castells, I. Altingövde, and F. Can, "Diversity and novelty in information retrieval.," *SIGIR*, p. 4503, 2013.

[27] A. Borodin, H. C. Lee, and Y. Ye, "Max-sum diversification, monotone submodular functions and dynamic updates," in *Proceedings of the 31st Symposium on Principles of Database Systems*, PODS '12, (New York, NY, USA), pp. 155–166, ACM, 2012.

[28] D. Panigrahi, A. Das Sarma, G. Aggarwal, and A. Tomkins, "Online selection of diverse results," in *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, (New York, NY, USA), pp. 263–272, ACM, 2012.

[29] S. Gollapudi and A. Sharma, "An axiomatic approach for result diversification," in *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, (New York, NY, USA), pp. 381–390, ACM, 2009.

[30] M. Sadoghi and H.-a. Jacobsen, "BE-Tree: An index structure to efficiently match Boolean expressions over high-dimensional discrete space," in *Proceedings of the 2011 ACM SIGMOD ...*, 2011.

[31] D. Zhang, C.-y. Chan, and K.-l. Tan, "opIndex_An Efficient Publish / Subscribe Index for E-Commerce Databases," *VLDB*, vol. 7, no. 8, pp. 613–624, 2014.

[32] K. Mouratidis, S. Bakiras, and D. Papadias, "Continuous monitoring of top-k queries over sliding windows," in *Proceedings of the 2006 ACM ...*, pp. 635–646, 2006.

[33] W. Rao, L. Chen, S. Chen, and S. Tarkoma, "Evaluating continuous top-k queries over document streams," *World Wide Web*, vol. 17, pp. 59–83, Nov. 2012.

[34] E. Vee, "Efficient computation of diverse query results," in *ICDE*, pp. 228 – 236, 2008.

[35] Pripuzic, *Top-k Publish/Subscribe matching model based on sliding window.* Dissertations and Thesis Collection - University of Zagreb; http://data.theeuropeanlibrary.org/Collection/a0585, phd thesis ed., 2010.

[36] Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science).* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[37] L. Golab and M. Özsu, "Issues in data stream management," *ACM Sigmod Record*, no. June, pp. 5–14, 2003.

[38] A. P. Sistla, "On characterization of safety and liveness properties in temporal logic," in *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, PODC '85, (New York, NY, USA), pp. 39–48, ACM, 1985.

[39] R. Baldoni, M. Contenti, S. Piergiovanni, and A. Virgillito, "Modeling publish/subscribe communication systems: towards a formal approach," in *Object-Oriented Real-Time Dependable Systems, 2003. (WORDS 2003). Proceedings of the Eighth International Workshop on*, pp. 304–311, Jan 2003.

[40] G. Cormode, V. Shkapenyuk, D. Srivastava, and B. Xu, "Forward Decay: A Practical Time Decay Model for Streaming Systems," *2009 IEEE 25th International Conference on Data Engineering*, pp. 138–149, Mar. 2009.

[41] B. Chandra and M. M. Halldórsson, "Approximation Algorithms for Dispersion Problems," *Journal of Algorithms*, vol. 38, pp. 438–465, Feb. 2001.

[42] M. M. Halldrsson, "Approximating the minimum maximal independence number," *Information Processing Letters*, vol. 46, no. 4, pp. 169 – 172, 1993.

[43] L. Qin, J. Yu, and L. Chang, "Diversifying top-k results," *Proceedings of the VLDB Endowment*, pp. 1124–1135, 2012.

[44] S. Ranu, M. Hoang, and A. Singh, "Answering top-k representative queries on graph databases," *Proceedings of the 2014 ACM SIGMOD ...*, pp. 1163–1174, 2014.

[45] P. Indyk and A. Andoni, "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions," *Communications of the ACM*, vol. 51, no. January, pp. 117–122, 2008.

[46] Anand Rajaraman and Jeff Ullman, "Chapter Three of Mining of Massive Datasets," in *Mining of Massive Datasets*, ch. 3, pp. 72–130, Cambridge University Press, 2, illustr ed., 2014.

[47] C. Sammut and G. I. Webb, *Encyclopedia of Machine Learning*. Springer Publishing Company, Incorporated, 1st ed., 2011.

[48] B. Dimm, "Minhash — clustify blog ediscovery, document clustering, predictive coding, information retrieval, and software development," 2013.

[49] A. Clauset, C. R. Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data," in *ISSN 00361445. doi: 10.1137/ 070710111. URL http://dx.doi.org/10.1137/070710111*, 2009.

[50] B. Jiang and T. Jia, "Zipf's law for all the natural cities in the United States: a geospatial perspective," *International Journal of Geographical Information . . .*, no. 1, pp. 1–12, 2011.