Quest Journals
Journal of Software Engineering and Simulation
Volume 8 ~ Issue 5 (2022) pp: 34-40
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

Efficient Algorithms for Publishing, Subscribing and Aggregating Information Contents in a Distributed System.

[1]Obobom Miracle Billy, [2]Bennett, E. O., [3]Sako, D. J. S.
Department of Computer Science, Rivers State University, Port Harcourt, Nigeria

Abstract
A publish, subscribe and aggregate system is an information dissemination infrastructure that supports many-to-many communications among publishers and subscribers. Lots of time and resources is spent by users filtering through a barrage of information due to lack of notification when new information contents are available in a dynamic environment where information constantly fluctuates. Socket I/O technology was adopted to enable subscribers receive information in real time. System implementation was achieved using AdonisJS. Eleven (11) topics were created by the publisher module and different contents were published to these sample topics. Keywords were extracted using MonkeyLearn text analysis to extract these keywords from the total words on each content and recommendation was made to other similar contents using related keywords. The content with more related keywords got more content recommendation to subscribed users than content with little or no keywords. The results show 77.8% efficiency which indicates that subscribers can subscribe to contents of interest and thus receive only information published to these contents which is further aggregated by the broker system for efficient information management and decision making.
Keywords: *Publisher, Subscriber, Aggregating, Distributed System*

***R**eceived 03 May, 2022; **R**evised 14 May, 2022; **A**ccepted 16 May, 2022 © The author(s) 2022.*
P**ublished with open access at **www.questjournals.org

I. INTRODUCTION:

With the growth of technology, the volume of data generated and received has increased dramatically. By 2020, it was estimated that about 1.7 gigabytes of fresh data per second would be generated [1] but that estimate may have now been surpassed. Every day, quintillion bytes of data are generated, and this number is growing rapidly as the Internet of Things (IoT) expands. Data is generated from every aspect of life as a result of the Internet of Things (IoT), including sensors used to gather climatic data, uploads to social networking sites, digital images and videos, buy transaction records, and cell phone GPS signals [2]. With this high burst of information, an end user would be exposed to a large amount of information at a time and would waste a lot of time trying to determine whether they needed to do anything with a given information or not and thus will tend to discard a huge amount of information without taking any action – this will not only be time consuming but will lead to a waste of resources.

The Publish/Subscribe system is very useful in our world today. It helps us to better monitor and describe the world around us. In our waking hours we come in contact with a barrage of events and information. Most of these information and events are not really relevant and we shouldn't spend our time, resources and decision making on them. While some information is useful to notice, there are others which are important, even critically important and make it necessary to use all available tools and resources. Being able to have knowledge of highly important information with minimal effort is critical to any successful individual or organization [3].

In the Publish-Subscribe system, subscribers register their interest in any topic or event and subsequently they are notified asynchronously about any event or information that matches their registered interest. These events or messages are being generated by the publishers. The publisher need not know of the subscribers that are receiving the messages and one publisher can send copies of messages to multiple subscribers [4].

Thispaper presents efficient algorithms for publishing, subscribing and aggregating information contents in a distributed system which will enable users to get information useful and relevant to them and will greatly reduce the time spent surfing through large volumes of information and therefore lead to better and enhanced decision-makingprocess, while also saving cost.

II. RELATED LITERATURES

Publish-Subscribe systems are intended for situations where a single message is required by, and should be distributed to multiple users. One major advantage of this system is that it keeps the publisher separated from the subscriber, thus a publisher does not necessarily need to know the subscribers and the subscribers do not need to know anything about the publisher. A publisher publishes to a group of subscribers who make a subscription to some or all published information by a publisher. The system matches these publications to the subscribers, ensuring that all messages are made available and delivered to all subscribers in a timely manner [5].

Subscribers are usually interested in a particular message (event) or message patterns and not in all messages. The different ways in which these messages are specified based on the subscriber's interest has led to several subscriptions schemes, of which the two most widely used are: (i) Topic-based subscription and (ii) Content-based subscription schemes.

In Topic-based subscription, publish/subscribe scheme is solely based on topics or subjects and it uses the idea of channels which is used to bundle communicating peers with methods to characterize and classify event contents. Publishers can publish events and subscribers can make subscriptions to individual topics which are identified by keywords [6].

(ii) The content-based publish/subscribe system uses a subscription scheme based on the actual content of the considered events. This means that events are not classified according to some predefined external criterion (e.g., topic name), but according to the properties of the events themselves [7]

One of the earliest publicly described public/subscribe systems was the "news" subsystem of the ISIS Toolkit. The ISIS Toolkit was developed out of a study of fault tolerance in distributed systems. It implemented Virtual Synchrony which are rules for replicating data or a service that will behave in a manner indistinguishable from the behaviour of some non-replicated reference system running on a single non-faulty node [8].

With the advancement in computing, more improved and efficient publish-subscribe systems started emerging like IBM MQ, Rabbit MQ, Microsoft Scribe, Apache Kafka, Amazon SNS, Google Pubsub, Websocket, etc.

IBM MQ (also WebSphere MQ) focused on providing an available, reliable, scalable, secured and high-performance transport mechanism [9] and uses queue managers where multiple queue managers can run on a single physical server as well as a large variety of different hardware and operating system combinations [10].

Microsoft Scribe uses a topic-based system where notifications are grouped in topics (or subjects) and subscribers receive all notifications related to that particular topic (identified by keywords) [11].

Apache Kafka became the de-facto streaming platform in many organizations where it provides a scalable and reliable platform to capture and store all the produced data from different systems. It also efficiently provides the captured data to all the systems that want to consume it [12]. It became advantageous because it can be used as a connecting layer as it provides real delivery guarantees. A message in Kafka which is the unit of data is simply an array of bytes and can have an optional bit of metadata, which is referred to as a key which is written into partitions in a controlled manner.

RabbitMQ is an Advanced Message Queuing Protocol (AMQP)message broker. It is arguably the most popular open source and cross-platform message broker. It is lightweight and easy to deploy on premises and in the cloud. It supports multiple messaging protocols and can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements [13].

III. SYSTEM ARCHITECTURE

Architecture specifies the structure, views and action of a system. It is a precise blueprint and illustration of how the system is to be developed. It is constructed in such a way that it assists understanding with respect to the structures and actions of the system.

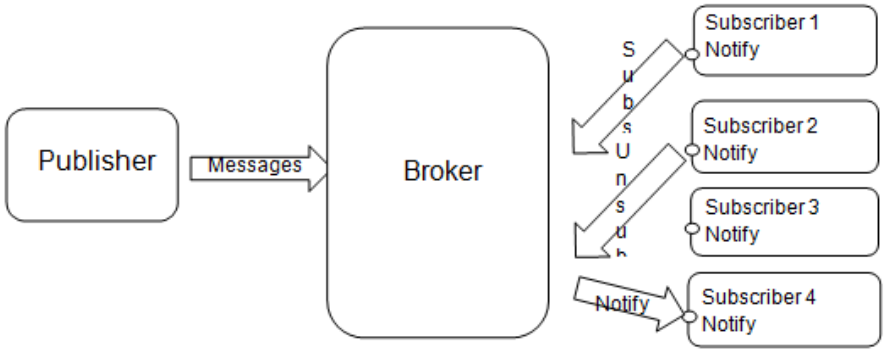


Figure 1: System Architecture

A publisher publishes messages to topics and can optimally receive message delivery reports. These messages are asynchronous and completely different from each other. Publishers do not need to have any knowledge of, or relation to, their subscriber. The subscribers are the end users of the messages sent by the publisher. A subscriber indicates interest in a particular topic by making a subscription to that topic. When a message is sent by the publisher to a topic the subscriber has made a subscription to, the subscriber receives a notification informing him/her that a message has been published to a topic of interest. Notifications are sent to the subscribers when messages are published by publishers. A subscriber can only receive notification from a particular topic if he/she has indicated interest in the topic by making a subscription.

Table 1: Notations used in Publish Subscribe System

S/No	Symbol	Notation
1.	T	Topics to subscribe to/Topics generated by publishers.
2.	S	Subscription that is stored on each subscriber
3.	TM_T	Stores all subscription containing topic T and a sequence number that counts the number of events published on T.
4	*e*	A new event
5.	p_i	A single entity of a publisher.
6	SG_T	Sequencing group of T
7.	id_e	A unique identifier associated to each event produced by a publisher.
8.	$outgoingEvents_i$	A set variable, initially empty, storing the events indexed by event id that are published.
9.	$subs_i$	A set variable storing topics subscribed by p*i*
10.	LCT_i	An integer value representing the sequence number associated to topic T_i
11	Sub_LC*i*	A set of pairs for each sequence number
12	generateUniqueEventID(*e*)	Generates a locally unique identifier for a specific event *e*
13.	getTopicRespAddress(T)	Returns the network address of the topic manager TM_T responsible for T.
14	CREATE_PUB_TS	Generate an empty timestamp for a generic event published on Topic T
15	ENS	Event Notification Service
16	FILL_IN_SUB_TS	Fill the time stamp and forward new subscription to the Topic Manager.
17	COMPLETED_SUB_VC	Subscription has been completed

A. Description of Algorithm for the Publisher Process

When an event ***e*** is published on a topic T, the publisher $\mathbf{p}_i$ executes Algorithm 1 shown below. In particular, it associates to ***e*** a unique identifier generated locally (line 1), it puts the event, together with the topic and the corresponding identifier in a buffer (line 2) and sends a CREATE_PUB_TS (ide, i, T) message to the topic manager TM_T, associated to T (lines 3-4).

Algorithm 1: Publish Algorithm

operation PUBLISH(*e*, *T*):

(1) id_e ←generateUniqueEventID*(e):*
(2) outgoingEvents ←*($_e$,id_e, T);*
(3) dest ←**getTopicRespAddress**(*T);*
(4) send CREATE_PUB_TS (id_e, *I, T)* to dest;
(5) **when** EVENT_TS (*ts, e_{id}*) is delivered;
(6) l**et**<*e, id_e T* >∈ outgoingEvents
(7) **such that** (ei_d = id_e);
(8) ENSpublish(<*e, ts*>, *T);*
(9) outgoingEvents ←outgoingEvents/{<*e, id_e T*>*}*

Figure 4: Publisher Process Algorithm

B. **Description of Algorithm for Subscribe Process**

B. **Description of Algorithm for Subscribe Process**

When a subscriber S*i* wants to subscribe to a new topic T, it executes the algorithm shown in Algorithm 2. In particular, it creates an empty subscription timestamp through the createSubTimestamp function (including also topic T), and then it sends a FILL_IN_SUB_TS (ts,(subsi ∪ {T}), id) message to fill the timestamp and to forward the new subscription to the topic manager TMTk as in Algorithm 2 which is responsible for the last topic in the subscription, according to the precedence relation → (lines 2-4).

Algorithm 2: Subscribe Algorithm

operation SUBSCRIBE(*T*):

(1) ts ←createSubTimestamp*($subs_i$Ʊ T)*
(2) t ← next*(ts, null);*
(3) dest ←**getTopicRespAddress**(t^1*);*
(4) send FILL_IN_SUB_TS (ts, (*$subs_i$* Ʊ {T}), id) to dest
(5) **when** COMPLETED_SUB_VC *(ts,s)* is delivered:
(6) *sub_LC_i* ←*ts;*
(7) subsi ←subs*iƱ{T};*
(8) ENSsubscribe*(T);*

C. Description of Algorithm for the Notification

When an event ***e*** is notified by the ENS, a subscriber **S*i***executes the algorithm 3. The algorithm uses a function tag (***e***) that creates a new event ***e*1**, containing ***e*** and the indication that it has been delivered out-of-order. The event ***e*** is not immediately notified to the application layer. A subscriber **S*i*** first checks if the event has been published on a topic actually subscribed by **S*i*** and then checks if it has been notified by the ENS in the right order (line 1). If such a condition is not satisfied, a new event ***e*1** is created by tagging ***e***; then, the event *e*1 is notified to the application (lines 9-11) [14].

Algorithm 3: Notify Algorithm

upon ENSnotify(< e, ts>, T);

(1) **if** ((T € $subs_i$) ∧ (*sub_LC_i*<*ts*))
(2)**then trigger** notify (e, T); **% ordered notification**
(3) **for each** (<T_j, v > € *sub_LC_i*)
(4) **if** ((∃<T_j, v^j> € *ts*)∧ (v^j> v))
(5) **then***sub_LC_i* ← *sub_LC_i*{<T_j, *v*>}
(6) *sub_LC_i* ← *sub_LC_i*∪ {<T_j, v^j>}
(7) **endif**
(8) **endfor**
(9) **else**e^i ← tag(e);
(10) t**rigger** notify (e^1, *T)* **%out-of-order notification**
(11) **endif**

IV. RESULTS AND DISCUSSION

Topics can only be created by the Publisher Module which then makes updates of appropriate information (news) to these topics created. The Subscriber Module handles subscriptions to topics created by the Publisher module. Information updated on these topics can be viewed and notifications can be gotten based on the topics such subscription is made to. The Subscriber Module also performs Unsubscribe action, if it no longer wants to receive notification and information update from (a particular) topic(s). Notification is handled by the Broker Module using the socket.I/O technology which carries out notification in real time as long as the Subscriber Module is active on the system. Aggregation is also being performed by the Broker Module which monitors and captures the topics that have been subscribed to by the Subscriber Module and uses these data to filter information and thus performs aggregation by automatically recommending related information to the Subscriber Module. The Subscriber Module can choose to receive this aggregated information or not based on its requirements.

This system enables users to receive information (contents) of interest and also help manage this information (contents) adequately which enhances effective decision making, saves time and resources and aid better concentration using a simplified approach that does not need special training to use.

Table 2:Domain Extracted

S/N	Topics	Published Content	Subscriber	Total Word	Keywords Generated	Recommended Contents
1	Business	0	0	0	0	0
2	Education	1	1	129	9	1
3	Entertainment	0	0	0	0	0
4	Health	0	0	0	0	0
5	Movies	1	2	91	4	0
6	Politics	1	1	171	12	1
7	Science	3	1	145	10	2
8	Sports	2	2	171	10	1
9	Technology	1	3	118	15	2
10	Travels	0	0	0	0	0
11	Trends	0	0	0	0	0

From Table 2 above, 11 topics are created by the publisher module and different contents are published to these topics which subscriptions have been made to. Based on the contents published, keywords are extracted from the total words on each content and recommendation is made to other contents using related keywords.

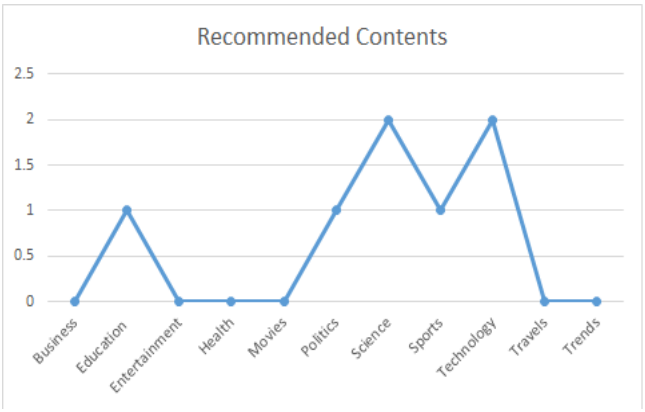


Figure 2: Line Chart showing the number of recommended contents to the subscriber from contents published on each topic

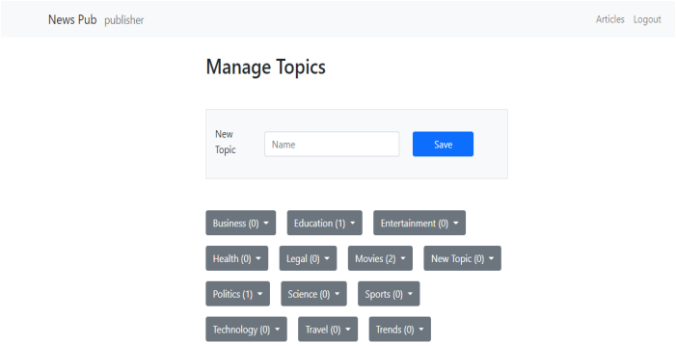
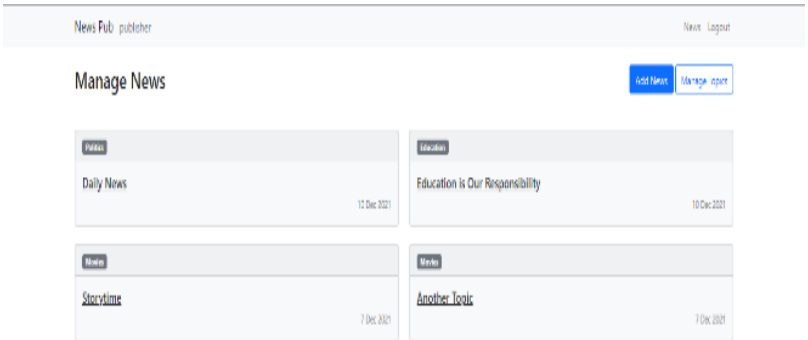


Figure 3: Topics Created by the Publisher



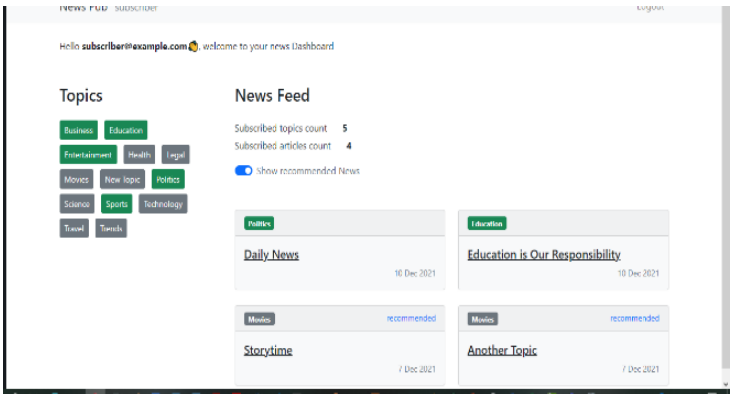


Figure 4: Published Contents

REFERENCES

[1]. Marr,B. (2016). Big Data in Practice: How 45 Successful companies used Big Data Analytics to Deliver Extraordinary Results
[2]. Bessis, N.& Dobre, C. (2014). Big Data and Internet of Things: A Roadmap for Smart Environments.
[3]. Tarkoma, S. (2012). Publish/Subscribe Systems - Design and Principles
[4]. Pandey, K. & Zhang, K. (2015). Minimizing the communication Cost of Aggregation in Publish/Subscribe Systems.
[5]. Perry, M., Delporte, C., Demi, F., Ghosh, A., Luong, M. (2001).MQSeries Publish/Subscribe Applications
[6]. Birman, K. (N.D). A History of the Virtual Synchrony Replication Model. Department of Computer Science, Cornell University, Ithaca, New York 14853.
[7]. Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A., (2003). The Many Faces of Publish/Subscribe.ACM Computing Surveys, Vol. 35(2).
[8]. Birman, K. (N.D). A History of the Virtual Synchrony Replication Model. Department of Computer Science, Cornell University, Ithaca, New York 14853.
[9]. Credle, R., Elkins, C., Hayward, P., Lampkin, V., Leming, M., Sanchez, J., Taylor, M., Wilson, M. (2014). IBM MQ V8 Features and Enhancement.
[10]. Davies, S.,& Broadhurst, P. (2005). WebSphere MQ V6 Fundamentals. IBM Red Paper Publications.
[11]. Shehnaaz, Y. (2004). Survey of Publish Subscribe Communication System
[12]. Jafarpour, H., Desai, R., Guy D. (2019) - KSQL: Streaming SQL Engine for Apache Kafka
[13]. Bageri, S. (2018). Learning RabbitMQ with C#: A magical tool for the IT world.
[14]. Baldoni, R., Bonomi, S., Platania, M., Querzoni, L. (2011). Dynamic Message Ordering for Topic-Based Publish/Subscribe System.