Advances in Computational Sciences and Technology
ISSN 0973-6107 Volume 10, Number 8 (2017) pp. 2471-2477
© Research India Publications
http://www.ripublication.com

Voice Mail Email Synchronization Load Balancing A Multithreaded Polling Mechanism

Aniket Masule
Intern
ShoreTel Communications
Bangalore, India

Aditya Uppuluri
Senior Software Engineer
ShoreTel Communications
Bangalore,India

Prasad Kumbala
Senior Manager Engineer
ShoreTel Communications
Bangalore, India

Karthikeyan B
Assistant Professor
Electronics Engineering
VIT,Vellore

Abstract

Voice mail Email Synchronization is a ShoreTel service which is responsible for syncing the user's voice mail with email client, communicator(soft phone) and IP Phone. This process of synching should have real-time response. This paper describes a mechanism that enhances the real-time response and distribution of load in balanced manner among computational resources. For serving this purpose multithreading model is presented, which enhances capabilities of current polling mechanism. The load mentioned in this case is total time taken by all mailboxes to be scanned by computational resources.

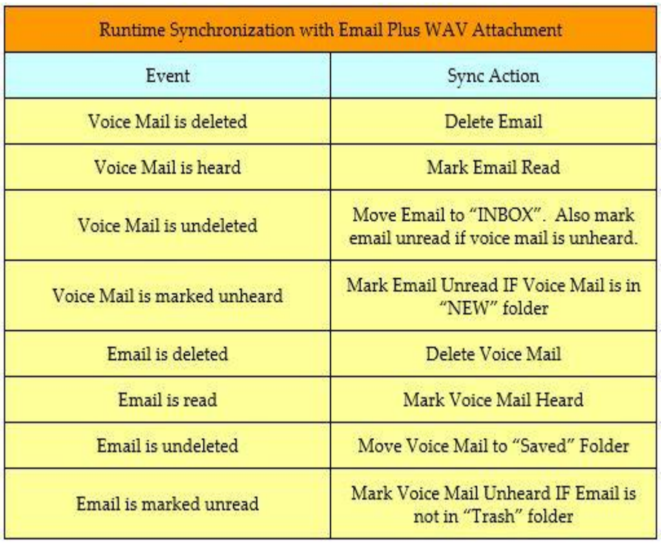
Keywords: Oauth, Multithreading, Real-time, Distribution, Rearrangement.

I. INTRODUCTION

The important aspect of any service/application is giving the real-time experience at the user end. The process of achieving optimal response time involves developing an algorithm which is scalable with heavily increasing load. The term 'load' changes as the context it is used with, here load represents total time taken to scan all the mailboxes i.e. users in an organization. The algorithms currently available concentrates on static distribution and no dynamic resource management [1][2]. The algorithm needs to deal with dynamically changing load and redistributing load among the computing resources to obtain optimal solution. So, this paper describes a generic format of an algorithm whose integration can enhance the current capabilities of this synchronization service.

One of the important steps in successful execution of an algorithm is availability of accurate raw information. In this case the raw information is time taken to scan individual mailbox. The process of scanning involves reading IMAP flags related with every email in that mailbox. These IMAP flags reflect the status of an email e.g. read, unread. If there is a change in the flags it must be updated to the other terminals i.e. communicator and IP phone and vice-versa. IMAP is used to change the status of an email based on the synchronization rules mentioned in Voice Mail sync with Email SAS. The fig.1 explains the syncing rules [3] for Runtime Synchronization with Email Plus WAV Attachment. There are other rules based on scenarios e.g. Startup Synchronization with Email Text Only, Startup Synchronization with Email Plus WAV Attachment, Runtime Synchronization with Email Text Only.

The mechanism should have capability of managing computing resources which are used to fetch the user data and update.



Runtime Synchronization with Email Plus WAV Attachment	
Event	Sync Action
Voice Mail is deleted	Delete Email
Voice Mail is heard	Mark Email Read
Voice Mail is undeleted	Move Email to "INBOX". Also mark email unread if voice mail is unheard.
Voice Mail is marked unheard	Mark Email Unread IF Voice Mail is in "NEW" folder
Email is deleted	Delete Voice Mail
Email is read	Mark Voice Mail Heard
Email is undeleted	Move Voice Mail to "Saved" Folder
Email is marked unread	Mark Voice Mail Unheard IF Email is not in "Trash" folder

Fig.1: Sync Rules

So according to above mentioned rules, the updated status should be reflected within acceptable time which should be tending to real time limits.

These are the few things followed to achieve the task

1. Getting authorization for accessing user data
2. Accessing user data using the token unceasingly
3. Distribution of load (user data) in balanced manner

4. Fetching user data
5. Updating status to other terminals

II. LITERATURE SURVEY

A. *Oauth(Authorization)*

The process of authorization plays important role, as it enables us to access user data. It is more of an authorization rather than authentication. Oauth is an authorization framework which enables a third-party application to obtain limited access to service either on behalf of a resource owner by orchestrating an approval to the interaction between resource owner and the service, or by allowing a third-party application to obtain access on its own behalf to the data belonging to user which resides on the email client server.

B. *Data acquisition from resource server using access token*

To access data on the client server there is a need of an access token. This access token can be acquired by using the flow as shown in fig. 2.[4][5] This flow particularly belongs to obtain access token for Google server.

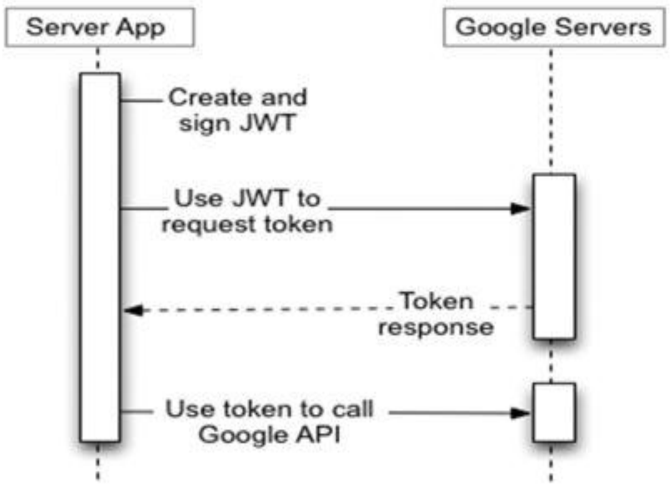


Fig.2: Access Token Flow

C. *Algorithms*

1. Least response time:

This algorithm concentrates on static distribution of the load (mailboxes) among the computing resources. The computing resources are nothing but the threads. The distribution is based on the time taken by individual entity of the total load. So, the new incoming load will be assigned to the least loaded thread. So, at the initial phase the process of allocation takes place in which every thread would be assigned with at least one entity of the load. In the next iteration, the algorithm would analyze the time taken by each thread to complete once scan and would assign load to the thread with least computing time. This algorithm is moreover static in terms of computing resources and distribution. It doesn't support iterative distribution of load and allocation of computing resource.

2. Least connection:

This algorithm is similar to least connection algorithm but in this case the parameter which is going to be monitored is number of the connections. In this case also, there is no dynamic nature in balancing load in terms of the thread management or redistribution of the load. The least connection algorithm considers initial distribution as NULL load for every thread. Then load will be assigned to the threads based on the number of connection each thread is having. The load in our case may offer different load at different instances of the time. So, considering number of connections as only parameter for analysis of distribution would result in heavily unbalanced load.

3. Round Robin:

This is a very basic algorithm followed in many distribution scenarios or task allocation scenarios. In this case algorithm, will use total available resources without any parameter to be monitored. Here load will be assigned starting from the first thread and would assign up to the last thread in the sequential manner. This algorithm would not take care of balancing on basis of either time or connections assigned to the threads.

III. METHODOLOGY

The mechanism that needs to be developed should effectively manage resources and balance dynamically changing load. This enunciates that computing resources i.e. thread should be dynamically created and destroyed according to the total offered. Also, the running threads should have balanced load so that no thread should be overloaded resulting in unacceptable response time.

There are two types of distributions based on scenarios.

- The initial distribution, where we don't have timing analysis of mailboxes i.e. initial iterations.
- The re-distributions, where mailboxes have been scanned once and we have time taken to fetch data for that mailbox.

The initial distribution:

Here we have made use of least connection algorithm and in advance to that dynamic resource allocation mechanism is integrated which enhances algorithm to overcome the disadvantages of least connection.

In this distribution, there is no prior information regarding any mailbox. Which can be fetched after initial scan of the mailbox. The first scan gives initial information which can be used for further distributions.

C: Maximum connections allowed to be handled by one thread.

CN: number of connection for given thread.

T: Threshold for total number of threads.

CTN: number of currently running threads.

CT[NUM]: Thread with thread id as NUM.

<u>Pseudocode:</u>

```
1. Initialize C and set CTN to 1, NUM to zero.
2. Repeat up to 6 till number of unassigned mailbox is zero.
3. Add new mailboxes to CT[NUM].
4. If (CN for CT[NUM]) > C and NUM== (CTN-1) perform 5 else 6
5. Initialize new thread, increment NUM.
6. Increment C, threshold connections per thread.
```

So, initial distribution gives mailboxes assigned to different threads. The first scan of all mailboxes give us scan information which enables us to perform operation in redistribution phase.

The redistributions:

Initial distribution allows us to map mailboxes to different threads till redistributions are being performed. In this phase, we have initial information for different mailboxes, using which we map mailboxes such that all threads will have balanced load.

In this algorithm, the along with balanced load distribution, dynamic resource allocation and deallocation is maintained.

TT: Time threshold for each thread

T: Threshold for total number of threads

CTN: number of currently running threads.

CT[NUM]: Thread with thread id as NUM and CT[]
Currently running threads.

PT: Predicted number of threads.

SUM[]: array of double values representing load on each
thread

Pseudocode:

```
1.  Initialize CTN to 1, NUM to zero and SUM[] to zero.
2.  Calculate total load by all mailboxes.
3.  Calculate predicted number of threads
    PT= (total_load/TT) round it ceil.
4.  If PT > T, increment TT repeat 3
5.  Calculate new TT
    TT=(total_load/PT)
6.  Arrange mailboxes in descending order of their time.
7.  Pick a mailbox from above array.
8.  Scan through each CT[] such that SUM[NUM]+mailbox load < TT.
9.  If no thread found in 7, increment CTN
10. If CTN > T increment TT.
11. Assign mailbox to CT[NUM] obtained in 8 or 9.
12. If CTN > PT repeat from 1.
```

So, the above algorithm will distribute load in such a way that system will have balanced mailbox distribution.

Repetition of the distribution and redistribution:
The mapping of mailboxes is done by above two algorithms cannot be kept static as the load offered by the mailboxes can be changed at any instance of time, which would result in unbalanced load again. So there are three cases where we need to call distribution and redistribution again.

1. New user(mailbox) added to the organization
2. Expiration of access token.
3. Change in the load offered by mailbox.

IV. CONCLUSION

Efficient resource management, handling dynamically changing load and balanced distribution are the key features for real time response of the service, which are achieved by implementing above two algorithms used for distribution. So, the algorithm can work efficiently also it is scalable even in the case of exponentially increasing load.

REFERENCES

[1] https://en.wikipedia.org/wiki/Greedy_algorithm

[2] https://en.wikipedia.org/wiki/Binary_space_partitioning

[3] Voice Mail sync with Email SAS, Jason Miller

[4] https://oauth.net/2/

[5] Barry Leiba, " OAuth Web Authorization Protocol ", www.computer.org/internet computing, Vol. 16, No. 1. January/February, 2012