Load-Balancing Algorithm for Distributing Session Initiation Protocol (SIP) Requests to A Cluster of SIP Servers

Patel Riken
Department of Computer Science & Engineering
Narayan Shastri Institute of Technology

Abstract

This paper introduces load balancing algorithm of distributing Session Initiation Protocol (SIP) request to a cluster of SIP servers. Load balancer algorithm Transaction Least Work Left is used to allocate work to least values of the servers. There are basically two types of session state exist in SIP. The first session state is established by the INVITE transaction and is closed by the BYE transaction. Now a days SIP is employed for IPTV, instant messaging system, audio conferencing and VoIP. Session-aware request assignment is the system process where a system provides requests to servers so that sessions are properly recognized by that particular server, and next requests corresponding to that same session are provided to the same back-end server. This algorithm Transaction Least-Work-Left gains its performance by estimating several features: dynamic estimates the loads of back end servers, load, information of the SIP protocol, finding variability in call length, dividing transactions from calls.

Keywords: Session Initiation Protocol (SIP), Load Balancing, Performance, SIP Server

I. INTRODUCTION

A session is a semi-permanent interactive information interchange, also referred as a dialogue, a conversation between two or more communicating devices, or between a pc and user [15]. A session is set up or established at a definite purpose in time, and then torn down at some other point. A communication session might involve more than one message in each direction. A session is stateful or stateless. In that case at least one of the communicating parts needs to save information regarding the session in order to be able to communicate, as opposed to stateless communication, where the communication consists of independent requests with responses [15].

The Session Initiation Protocol (SIP) is a general-purpose signaling protocol used to control various types of media sessions. SIP is widely used in Voice over IP (VoIP), instant messaging, IPTV, voice conferencing, and video conferencing to maintain sessions. SIP has a number of new features that distinguish it from other protocols such as HTTP. SIP is a transaction-based protocol frequently referred to as calls. There are two types of session state exist in SIP. The 1st, session state, is created by the INVITE transaction and is destroyed by the BYE transaction [16][9]. Each SIP transaction also creates state that exists for the duration of that transaction [9]SIP thus has overheads that are associated both with sessions and with transactions, and taking advantage of this fact can result in more optimized SIP load balancing. The core IETF SIP specification is given in RFC 3261[13]. Several kinds of sessions can be used, including voice, text, and video, which are transported over a separate data-plane protocol. SIP does not allocate and manage network bandwidth as does a network resource reservation protocol such as RSVP that is considered outside the scope of the protocol. As another example, SIP can run over many protocols such as UDP, TCP, TLS, SCTP, IPv4, and IPv6 [3].

SIP defines a number of logical entities, namely user agents, redirect servers, proxy servers and registrars. SIP Servers are essential network elements that enable SIP endpoints to exchange messages, register user location, and seamlessly move between networks. SIP Servers enable network operators to install routing and security policies, authenticate users and manage user locations [3].

II. RELATED WORK

A. Overview of the Protocol:

The Session Initiation Protocol (SIP) is a signaling protocol, widely used for maintaining and controlling multimedia communication such as voice and video calls over IP networks [9]. The protocol is defined as the messages that are sent between two or more endpoints, which provide establishment, termination and other essential elements of a call. SIP can be used for creating, modifying and terminating sessions consisting of one or several media streams [3][15][16]. SIP can be used for two-
party (unicast) or multiparty (multicast) sessions. Other SIP application include video conferencing, streaming multimedia distribution, instant messaging, presence information, file transfer, fax over IP and online games. SIP is a signaling (control-plane) protocol designed to establish, modify, and terminate media sessions between two or more parties. The core IETF SIP specification is given in RFC 3261[13], although there are many additional RFCs that enhance and refine the protocol. Several kinds of sessions can be used, including voice, text, and video, which are transported over a separate data-plane protocol. SIP does not allocate and manage network bandwidth as does a network resource reservation protocol such as RSVP; that is considered outside the scope of the protocol. Fig. 1 illustrates a typical SIP VoIP scenario, known as the “SIP Trapezoid.” Note the separation between control and data paths: SIP messages traverse the SIP overlay network, routed by proxies, to find the eventual destinations [9]. Once endpoints are found, communication is typically performed directly in a peer-to-peer fashion.

In this example, each endpoint is an IP phone. However, an endpoint can also be a server providing services such as voicemail, firewalls, voice conferencing, etc. The separation of the data plane from the control plane is one of the key features of SIP and contributes to its flexibility [9].

B. SIP Users, Agents, Transactions, and Messages

A SIP Uniform Resource Identifier (URI) uniquely identifies a SIP user, e.g., sip:XYZ@us.ibm.com. This layer of indirection enables features such as location independence and mobility [9].

SIP users employ endpoints known as user agents. These entities initiate and receive sessions. They can be either hardware or software. User agents are further decomposed into User Agent Clients (UAC) and User Agent Servers (UAS) [5], depending on whether they act as a client in a transaction (UAC) or a server (UAS). Most call flows for SIP messages thus display how the UAC and UAS behave for that situation [9].
C. SIP Message Header

SIP is a text-based protocol that derives much of its syntax from HTTP. Messages contain headers and additionally bodies, depending on the type of message. In VoIP, SIP messages contain an additional protocol, the Session Description Protocol (SDP), which negotiates session parameters (e.g., which voice codec to use) between endpoints using an offer/answer model. Once the end-hosts agree to the session characteristics, the Real-time Transport Protocol (RTP) is typically used to carry voice data [9].

D. Overview of SIP servers

SIP Servers are essential network elements that enable SIP endpoints to exchange messages, registration of user location, and loosely move between networks. SIP Servers provides functionality to network operators to install routing and security policies, authentication of users and management of user locations [3].

There are three types of SIP servers:

1) SIP Registrar Server—handles location registration messages.
2) SIP Redirect Server—returns the address of another server.
3) SIP Proxy Server—forwards SIP requests and responses.

III. EXISTING SYSTEM

A. Types of Load Balancing Algorithms

Load balancing algorithms can have two categories based on initiation of process as follows:

1) Sender Initiated:
In this type the load balancing algorithm is initialized by the sender. In this type of algorithm the sender sends request messages till it finds a receiver that can accept the load [1].

2) Receiver Initiated:
In this type the load balancing algorithm is initiated by the receiver. In this type of description algorithms the receiver sends request messages till it finds a sender that can get the load [1].

3) Symmetric:
It is the combination of both sender initiated and receiver initiated.

Depending on the current state of the system, load balancing algorithms can be divided into two categories as static and dynamic load balancing algorithms.

B. Static Load Balancing

In static load balancing, the performance of the servers is determined at the beginning of execution. Then depending upon their performance the work load is assigned by the master server. The slave servers calculate their allocated work and submit their result to the master. A task is always executed on the servers to which it is assigned that is static load balancing methods are non-preemptive. The goal of static load balancing method is to reduce the execution time, minimizing the communication delays [1].

1) Round Robin Algorithm
Round Robin algorithm distributes jobs evenly to all slave servers. All jobs are assigned to slave servers based on Round Robin order, meaning that server choosing is performed in series and will be back to the first server the last server has been reached. Servers choosing are performed locally on each server, independent of allocations of other servers [1].

2) Randomized Algorithm
Randomized algorithm uses random numbers to choose slave servers. The slave servers are chosen randomly following random numbers generated based on a statistic distribution [1].

3) Central Manager Algorithm
Central server will choose a slave server to be assigned a job. The chosen slave server is the server having the least load. The central server is able to gather all slave servers load information, thereof the choosing based on this algorithm are possible to be performed. The load manager makes load balancing decisions based on the system load information, allowing the best decision when of the requests arrived [1].

4) Threshold Algorithm
In Threshold algorithm, the requests are assigned immediately upon creation to hosts. Hosts for new requests are selected locally without sending remote messages. Each server keeps a private copy of the system’s load. The load of a server can characterize by one of the three levels: Under loaded, medium and overloaded. Two threshold parameters t_under and t_upper can be used to describe these levels. Under loaded: load < t_under, Medium: t_under ≤ load ≤ t_upper, Overloaded: load > t_upper [1].
C. Dynamic Load Balancing

Unlike static algorithms, dynamic algorithms allocate requests dynamically when one of the servers becomes under loaded. Instead, they are buffered in the queue on the main host and allocated dynamically upon requests from remote hosts [1].

Central Queue Algorithm

It stores new activities and unfulfilled requests as a cyclic FIFO queue on the main host. Each new activity arriving at the queue manager is inserted into the queue. Then, whenever a request for an activity is received by the queue manager, it removes the first activity from the queue and sends it to the requester. If there are no ready activities in the queue, the request is buffered, until a new activity is available. If a new activity arrives at the queue manager while there are unanswered requests in the queue, the first such request is removed from the queue and the new activity is assigned to it [1].

1) Local Queue Algorithm

The basic idea of the local queue algorithm is static allocation of all new requests with process migration initiated by a host when its load falls under threshold limit, is a user-defined parameter of the algorithm. The parameter defines the minimal number of ready requests the load manager attempts to provide on each server.

2) Response-Time Weighted Moving Average

Another method is to make load-balancing decisions based on server response times. The Response-time Weighted Moving Average (RWMA) algorithm assigns calls to the server with the lowest weighted moving average response time of the last n response time samples. The formula for computing the RWMA linearly weights the measurements so that the load balancer is responsive to dynamically changing loads, but does not overreact if the most recent response time measurement is highly anomalous. The most recent sample has a weight of (n), the second most recent a weight of (n-1), and the oldest a weight of one [9].

D. New Methods:

3) Call-Join-Shortest-Queue (CJSQ):

Tracks the number of calls allocated to each back-end server and routes new SIP calls to the node with the least number of active calls. The CJSQ algorithm estimates the amount of work a server has left to do base on the number of calls (sessions) assigned to the server. Counters are maintained by the load balancer indicating the number of calls assigned to each server. When a new INVITE request is received (which corresponds to a new call), the request is assigned to the server with the lowest counter, and the counter for the server is incremented by one [9].

4) Transaction-Join-Shortest-Queue (TJSQ):

Routes a new call to the server that has the fewest active transactions, rather than the fewest calls. This algorithm improves on CJSQ by recognizing that calls in SIP are composed of the two transactions, INVITE and BYE, and that by tracking their completion separately, finer-grained estimates of server load can be maintained. This leads to better load balancing, particularly since calls have variable length and thus do not have a unit cost [9].

5) Transaction-Least-Work-Left (TLWL):

Routes a new call to the server that has the least work, where work (i.e., load) is based on relative estimates of transaction costs. TLWL takes advantage of the observation that INVITE transactions are more expensive than BYE transaction.

IV. PROPOSED METHOD

A. Overview of Proposed Work

There are many SIP load balancing methods for servers like Round Robin, Hash algorithm, Response-Time Weighted Moving Average, Call-Join-Shortest-Queue (CJSQ), Transaction-Join Shortest-Queue (TJSQ), Transaction-Least-Work-Left (TLWL). Dynamic algorithms etc. But above algorithms have some disadvantages. The hash algorithm is not guaranteed to assign the same number of calls to each server, In Round robin when the jobs are of unequal processing time this algorithm suffers as the some nodes can become severely loaded while others remain idle, Round Robin is generally used in web servers where generally HTTP requests are of similar nature and thereby be distributed equally. Static load balancing algorithms are more stable than dynamic. But dynamic load balancing algorithms are always better than static as per as overload rejection, reliability, adaptability, cooperativeness, fault tolerant, resource utilization, response & waiting time and throughput is concert. Call-Join-Shortest-Queue (CJSQ), Transaction-Join Shortest-Queue (TJSQ) and Transaction-Least-Work-Left (TLWL) algorithms are recently developed in this research space. From above three algorithms Transaction-Least-Work-Left is more efficient in terms of response time, throughput, Occupancy and Response Time, Heterogeneous Back Ends, Load Balancer Capacity, SIP Server Performance.
B. System Architecture:

![System Architecture Diagram]

Figure 3 depicts our overall system. User Agent Clients send SIP requests (e.g., INVITE, BYE) to our load balancer, which then selects a SIP server to handle each request. The distinction between the various load-balancing algorithms presented in this paper is how they choose which SIP server to handle a request. Servers send SIP responses (e.g., 180 TRYING or 200 OK) to the load balancer, which then forwards the response to the client. Note that SIP is used to establish, alter, or terminate media sessions. Once a session has been established, the parties participating in the session would typically communicate directly with each other using a different protocol for the media transfer, which would not go through our SIP load balancer.

C. Work Flow of Proposed system:

![Load Balancer Architecture Diagram]

Figure 4 illustrates the structure of the load balancer. The rectangles represent key functional modules of the load balancer, while the irregular shaped Boxes represent state information that is maintained. The arrows represent communication flows. The Receiver receives requests that are then parsed by the Parser. The Session Recognition module determines if the request corresponds to an already existing session by querying the Session State, which is implemented as a hash table as described below. If so, the request is forwarded to the server to which the session was previously assigned. If not, the Server Selection module assigns the new session to a server using one of the algorithms described earlier. For several of the load-balancing algorithms we have implemented, these assignments may be based on Load Estimates maintained for each of the servers. The Sender forwards requests to servers and updates Load Estimates and Session State as needed. The Receiver also receives responses sent by servers. The client to receive the response is identified by the Session Recognition module, which obtains this information by querying the Session State. The Sender then sends the response to the client and updates Load Estimates and Session State as needed. The Trigger module updates Session State and Load Estimates after a session has expired.
V. IMPLEMENTATION & RESULTS

A. Load Balancing Pseudo Code:

For load balancer:
LoadBalancer (Request from client, call_ID)
If (Request==INVITE)
  Server S=Execute Algorithm to select the server
  Forward the request to the server S
  Establish the session between client and server
  Send 200 response to client
  Put entry in the Active table
  Increment the load count of server S
  Update the load in to the table
Else if (Request==BYE)
  Check if the session is active
  S=Get the server for the session for the call_ID
  If (session==Active)
    Terminate the session
    Move the client entry to expired table
    Decrement the load count of the server S
    Update the load in to the table
  Else if (Request= chat or voice chat or upload)
    S=Get the server for the session for the call_ID
    If (session==Active)
      Forward the request to server S
    Else
      Throught the exception that INVITE is not provided early
    Else
      Throught the exception that INVITE is not provided early
End

For server:
Server (Request from load balancer, call_ID)
If (Request==chat or voice chat or upload)
  Get the destination address from the request
  Search for destination next hop
  Forward the request to the destination next hop
Else
  Send error response
End

In this section, we present in detail the experimental results of the load-balancing algorithms with its performance parameters.
A. Response Time:

We observe significant differences in the response times of the both load-balancing algorithms. Performance is limited by the CPU processing power of the servers and not by memory. Figure 5 and 6 shows the average response time for each algorithm versus offered load measured for the INVITE transaction. In this experiment, the load balancer distributes requests across eight back-end SIP server nodes. The sharp increases that are seen in response times for the final data points in the figure are due to the system approaching overload. The significant improvements in response time that TLWL and TJSQ provide present a compelling reason for systems such as these to use our algorithms. Section VI-C provides a detailed analysis of the reasons for the large differences in response times that we observe.
B. Throughput:

Figure 7 and 8 represents graphs of throughput of existing algorithms (TJSQ and CJSQ) and proposed algorithm TLWL is described. We now examine how our load-balancing algorithms perform in terms of how well throughput scales with increasing numbers of back-end servers. Therefore, linear scalability suggests a maximum possible throughput of about maximum calls for eight nodes. CJSQ is significantly worse than the others since it does not distinguish call hold times in the way that the transaction-based algorithms do. Experiments we ran that did not include pause times showed CJSQ providing very good performance, comparable to TJSQ. This is perhaps not surprising since, when there are no pause times, the algorithms are effectively equivalent. TJSQ does better than most of the other algorithms. This shows that knowledge of SIP transactions and paying attention to the call hold time can make a significant difference, particularly in contrast to CJSQ.

![Fig. 7: Graph Of Throughput Versus Number Of Nodes](image1)

![Fig. 8: Graph of Throughput Versus Node For Three Algorithms](image2)
C. Occupancy and Response Time:

We demonstrate how the different algorithms behave in terms of occupancy namely, the number of requests allocated to the system. The occupancy for a transaction $T$ assigned to a server $S$ is the number of transactions already being handled by when is assigned to it. Figure 9 and 10 shows the cumulative distribution function (CDF) of the occupancy as seen by a request at arrival time for one back-end node for CJSQ and TJSQ algorithms: This shows how many requests are effectively “ahead in line” of the arriving request. Intuitively, it is clear that the more requests there are in service when a new request arrives, the longer that new request will have to wait for service. One can observe that the two Transaction-based algorithms see lower occupancies for the full range of the distribution.

Fig. 9: Graph of Cumulative Distribution Function versus Occupancy

Fig. 10: Graph of CDF versus Occupancy for Three Algorithms
VI. CONCLUSION

Our paper introduces three novel approaches to load balancing in SIP server clusters (CJSQ, TJSQ, and TLWL). We present the design, implementation, and evaluation of a load balancer for cluster based SIP servers. Our load balancer performs session-aware request assignment to ensure that SIP transactions are routed to the proper back end node that contains the appropriate session state. By combining knowledge of the SIP protocol, recognizing variability in call lengths, distinguishing transactions from calls, and accounting for the difference in processing costs for different SIP transaction types, load balancing for SIP servers can be significantly improved.

Several opportunities exist for potential future work. These include evaluating our algorithms on larger clusters to further test their scalability, adding a fail-over mechanism to ensure that the load balancer is not a single point of failure, and looking at other SIP workloads such as instant messaging or presence.

REFERENCES