ISSN 22773061

Load-Balancing using HA Proxy on Multipath System with Flow Slice

C Surekha*, U Sesadri
*M Tech (CSE) Student,
HOD of CSE
cc.surekha@gmail.com

Abstract:

To provide switching capacity in terabit or petabit uses core routers in multipath switching systems (MPS). Without disturbing the order of intra flow packets, the load balance across multiple paths is the main issue of MPS design phase. The performance is not good through heavy tailed flow size distribution in flow based hashing algorithm. Our proposed novel scheme consist the HA Proxy very fast and reliable solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications. To process Layer7 of web site or to provide persistence to the website this scheme is very well suited for heavy load web sites crawling and flow slices are generated through Flow Slice (FS) that halt each flow at every intra flow spell larger than a slicing threshold and balances the load. The study analysis on traces of Internet, our scheme is optimal in load balancing performance through FS and the slicing threshold we set to 1 to 4 ms. We neglected out of order packets limits of probability $1/10^6$ on three prominent MPSs with little hardware complexity and twice of the internal speedup. The theoretical analysis proved that this and trace driven prototype simulations are validated.

Index Terms—HA Proxy, Load balancing, traffic measurement, and switching theory.

Council for Innovative Research
Peer Review Research Publishing System
Journal: INTERNATIONAL JOURNAL OF COMPUTER AND TECHNOLOGY

Vol 11, No. 1
editor@cirworld.com
www.cirworld.com, member.cirworld.com

ISSN 22773061

1. INTRODUCTION

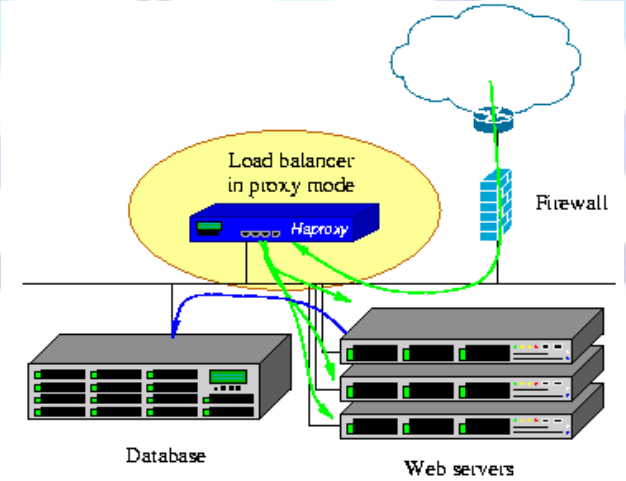
A new vulnerability was discovered in all versions from 1.4.4 and above. It was fixed today with 1.4.24 and 1.5-dev19 (CVE-2013-2175). This vulnerability can induce a crash when comfits involving tail-relative header offset such as h(xf,-1) are used. Please see the advisory for more details. All 1.4 and 1.5 users must upgrade. Other few important bugs were additionally fixed. David et al, One of them was a regression introduced in 1.5-dev12, which could randomly crash ha proxy on rare circumstances when using pipelined requests with a slow client. From 1.4 onwards consistent hashing algorithms are used with flapping servers, so we found endless loops. Task priority changing ability, DSCP field, ACLs L7 log level, net filter mark, ACLs header Proxy protocol, transparent proxy for BSD are merged with http response rule set, to provide positive side features. Last but not least, the doc on ACLs/fetches got a major lift-up to deduplicate keywords.

Branch	Description	Last Version	Released	Links
Development	Development	15-Dev19	2013	Git/web/dir
1.4	1.4-stable	1.4.24	2013	Git/web/dir
1.3	1.3-stable	1.3.26	2011	Git/web/dir
1.3.15	1.3.15-maint	1.3.15.13	2011	Git/web/dir
1.3.14	1.3.14-maint	1.3.14.14	2009	Git/web/dir
1.2	1.2-stable	1.2.18	2008	Git/web/dir
1.1	1.1-stable	1.1.34	2006	Git/web/dir
1.0	1.0 old	1.0.2	2001	Git/web/dir

Table Latest Versions

1.1. Description

High Availability Proxy is open software, very fast and reliable solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications [1]. Processing of layer 7 at very high loads is well suited for crawling of websites. To-days hardware is very realistic based on the support for tens of thousands of connections.



There are two major versions currently are supported:

More flexibility with 1.4: it has many new features compare to 1.2, which are waiting for a long time: Clint Side Keep Alive: on the net to load heavy pages in client side it reduces the time, to help the TCP stack have to speed up the TCP, even for lower number of concurrent connections response buffering on the servers, server stickiness and user filtering is supported by RDP protocol, to attach source address to a server by source based stickiness, tons of useful information reports by better stats interface, response stats and logs and precise stats by verbose health checks. Servers fast fail traffic based health above error threshold, for any request support of HTTP authentication, password encryption support, without restarting HA Proxy changing of server weight and management, ACLs persistence and maintenance, regardless of the server's state, fast report generation by using log analyzer.

Content switching and extreme loads with Version 1.3: compare to 1.2 it brought new features and improvements. Based on request content switching to a server pool, for content switching rules ACL, for better integration void variety of load balancing algorithms, blocking of unexpected protocols, Linux transparent proxy direct connection between server and client IP address. Data forward between two sides without copying by using Kernel TCP slicing to reach multi GB data rates. Socket separation through layered design, TCP and HTTP processing, QoS by priority schedulers and etc...

Version 1.2: It has been in production use since 2006 and provided an improved performance level on top of 1.1. It is not maintained anymore, as most of its users have switched to 1.3 a long time ago. Version 1.1, which has been maintaining critical sites online since 2002, is not maintained anymore either. Users should upgrade to 1.4[1].Hundreds of thousands of clients those need the 24x7 availability of several tens of gigabytes to terabytes per day, who the few hundred people have skills to maintain risks as a free software solution around the world by HA Proxy. This HA Proxy is the defacto standard load balancer and is default in cloud environments due to the recent years. Since it does not advertise itself, so we have to know about this balancer usage when the administrator reports it.

2. DESIGN CHOICE AND HISTORY

Single process model which enables very high speed simultaneous connections and event driven model through HA Proxy implementation. Due to limitations on system scheduler, memory and lock contention with thousands of connections rarely coped by Multiproces or multi threaded models. Event driven models implementing user space tasks allows finer resource and time management dint have above mentioned problems. These programs don't scale well on multiprocessor systems, so the most work done expected on optimized CPU cycle.This HA Proxy began in 1996 to setup a modem access with a very simple HTTP proxy. But its multi-process model clobbered its performance for other usages than home access. Two years later in 1998, the event-driven Z Proxy, used to compress TCP traffic to accelerate modem lines. Core engine is the robust and reliable from 2009. Event-driven programs fragile even at the same time re robust: the code must be changed to support high loads and attacks without ever failing. So to support finite set of features we will use HA Proxy. In production environment this HA Proxy never ever crashed. After words the people wants SSL and Keep-Alive support. The code of this both is complicate and fragile for several releases, so has negative impact on performance:

The load balancer having SSL means itself bottleneck. Whenever the CPU saturated that has load balancer, the response time increases and solution is to multiply this load balancer with another load balancer. So we have to put between clients SSL/Cache layer to solve this problem. For smaller sites SSL embed is sense. To maintain your memory you have to do some work to combine HA Proxy with Cya SSL library. Update [2012/09/11] : native SSL support was implemented in 1.5-dev12. The points above about CPU usage are still valid though.When CPU is 100 times slower than its normal functioning, keep-alive was invented to reduce the servers CPU usage. But drawback is only a lot of memory consumption by persistent connections. In 2009 memory is limited in size and architecture even the CPU is very cheap. To support maximum number of clients simultaneously we have to keep alive the CPU by disabling heavy weighted sites. Browsers double the number of concurrent connections on non-keep alive sites to compensate for this [1]. Keep alive with client was introduced with version 1.4.

Supported platform:

HA Proxy is known to reliably run on the following OS/Platforms:

Linux 2.4 on x86, x86_64, Alpha, SPARC, MIPS, PARISC

Linux 2.6 on x86, x86_64, ARM (ixp425), PPC64

Solaris 8/9 on Ultra SPARC 2 and 3

Solaris 10 on Opter on and Ultra SPARC

FreeBSD 4.10 - 8 on x86

Open BSD 3.1 to -current on i386, amd64, mac pc , alpha, sparc64 and VAX (check the ports)

On e poll patched Linux Kernel 2.4 or Linux 2.6 with HA Proxy newer version than 1.2.5, we will achieve highest performance. This can be achieved through very Operating System specific optimization. Select() is the default polling system for 1.1 version which is most common among Oases, but it degrades performance when it dealing with thousands of file descriptors. Instead of select() versions 1.2 and 1.3 uses poll() as default, but in some systems they may slow. And it is recommended that on Solaris this implementation is good. On FreeBSD and OpenBSD with k queue, On patched Linux 2.4 and Linux 2.6 with e poll HA Proxy 1.3 is used automatically. On very recent Linux 2.6 (>= 2.6.27.19), HA Proxy can use the new splice() sys call to forward data between interfaces without any copy. Performance above 10 Gbps may only be achieved that way.So the people looking for very fast load balancer based on the facts, it should consider the following options on hardware x86 or x86 - 64, in the following order:

HA Proxy 1.4 on Linux 2.6.32+

HA Proxy 1.4 on Linux 2.4 + e poll patch

HA Proxy 1.4 on FreeBSD

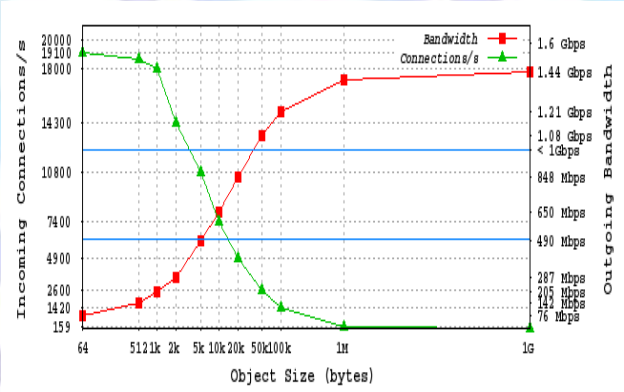
HA Proxy 1.4 on Solaris 10

A typical single user server at present dressed with a dual-core Opter on or Xeon mostly achieve between 15000 and 40000 hits/s and have no trouble saturating 2 Gbps under Linux.

3. PERFORMANCE

Compare to a long demonstration a users practical is better. For example the experience with ha proxy of Chris Knight's saturating a gigabit fiber on a video download site. To achieve maximal absolute performance found in OS techniques also in HA Proxy is: The cost of the context switch and memory usage is reduced in single-process, event-driven model. In millisecond several hundred of tasks can process and the few kilobytes of memory usage per session. In Apache model the memory consumed is more in the order of megabytes per process.To detect any event O(1) event checker on systems is used at connection among tens of thousands.Without data copy between read and writes I done through Single-buffering. With this CPU cycles and memory space is saved. Between CPU and network faces the I/O bus is the bottleneck. For example at 10 GBPS, the bottleneck is the memory bandwidth. Under Linux system call Splice() Zero-copy forwarding is possible, and with Linux 3.5 results in real zero-copy. This allows a small sub-3 Watt device such as a Seagate Dock star to forward HTTP traffic at one gigabit/s [1].For immediate memory allocation Most Recently Used memory allocator with fixed size memory pools with hot cache regions over cold cache ones. To create a new session it will reduces the time needed.

Work factoring: multiple accept () at once, and limited number of accept ()s per iteration when running in multi-process mode, so that load distribution is done among processes. Tree-based storage: it uses Elastic binary tree heavily. Based on this we can keep ordered timers, to keep ordered queue run, to manage round-robin and least-connected queues, with only an O(log(N)) cost.Optimized HTTP header analysis: On the fly the parsed headers are interpreted, and the optimized parsing is used to avoid previously read memory area to re-read. When an incomplete header reached to end buffer we will use check pointing, so based on this the parsing does not start again from the starting when more data is read. For example Pentium-M 1.7 GHz an HTTP parsing request typically takes 2 microseconds.Expensive system calls reduction must careful based on its numbers. Most of the work like buffer aggregation, time reading and enabling and disabling of file descriptor is done on user space. Even with moderate loads, all these micro optimizations will result in very low CPU usage. CPU is saturated at very high loads, but user usage with 5% and system about 95% generally, it says that 20 times less the HA Proxy consumes compare to system counterpart. It explains how important the operating systems tuning. Based on this we can say that processing of Layer 7 has performance impact a little. Even the user space work is doubled, the load distribution will look like 90% system and 10% belongs to user, means only 5% of effective loss on processing power. This is why on high-end systems, HA Proxy's Layer 7 performance can easily surpass hardware load balancers' in which complex processing which cannot be performed by ASICs has to be performed by slow CPUs [1]. The following shows the bench mark on a single core Pentium 4 with PCI express interfaces through HA Proxy 1.3.9 at EXOSEC:



For the objects more than 40 kb gbps is sustained and smaller than 6kb is sustained when the hit rate above 10000/s.

In Layer7 when suddenly high-end hardware load balancers are failed and an emergency solution is very expensive HA Proxy has been several times installed in production. Buffering is not at all possible in hardware load balancers, the requests are processed at packet level so for high response times across multiple packet requests have a great difficulty. Software load balancer on the other hand uses TCP buffering and for high response times and longer requests these are insensible. By reducing the session duration the HTTP increases server's connection acceptance which avails for new requests. To measure the load balancers performance there are 3 important factors used:

The session rate: To determine directly when the load balancer could not distribute all the received requests, this factor is very important. It is CPU dependent mostly. The requests or hits are same as sessions on HTTP 1.0 or keep alive disabled HTTP 1.1. To offload the server when it suffers under high loads keep alive has to be disabled. From empty objects we will get fastest results and with varying objects we will measure this factor. Through the Dual Opter on systems like HP DL145 will achieve session rates above 20000sessions/s which is running under carefully patched Linux 2.4 kernel.

The session concurrency: previous factor is combined with this factor. If the concurrent sessions are increased then the session rate will be dropped except the e-polling mechanism. The server may slow down for the higher number of concurrent sessions. The load balancer has 1000 concurrent sessions; if it receives 10000 sessions per second then the

servers will respond 100 ms. this sessions are limited based on the memory space and the amount of file-descriptors the system can handle. With 1GB of RAM HA Proxy results 60000sessions where 8 kb buffers need 16 kb per session. Layer 4 load balancers generally announce millions of simultaneous sessions because they don't process any data so they don't need any buffer. These load balancers are designed to use sometimes in Direct Server Return mode, in which only forward traffic available, and which forces to keep it the sessions for a long time before they were closed after their end to avoid cutting sessions.

The data rate: session rate is generally opposite side for this data rate. Measurement of this factor is usually done in Megabytes/s (MB/s), or Megabits/s (Mbps). By minimizing session setup overhead and teardown we will reach highest data rates through large objects. We need more memory space to support large windows, because session concurrency will be increased by large objects. The data must be copied between input interfaces to output interface through memory, because high data rates burn a lot of CPU and bus cycles on software load balancers.In high data rate from input port to output port packets are directly switched through Hardware load balancers, but sometimes we cannot process the cookie and fail to touch the header. For example, the systems Dual Opteron description can saturate 2 Gigabit Ethernet links on large objects, and I know people who constantly run between 3 and 4 Gbps of real traffic on 10-Gig NICs plugged into quad-core servers [1].For the best case performance of load balancer related to above factors announced generally. This can be happen not based on trust on third party vendors but also the behavior of every combination. The customer must below the limitations of that system, when he knows those 3 limitations. A good rule of thumb on software load balancers is to consider an average practical performance of half of maximal session and data rates for average sized objects [1].

3.1. RELIABILITY

Based on the trouble with reliability, we best ensure that a design by total continuity of service. Short term implementation of reliable system is more difficult, but in the long duration it acknowledge and easier to maintain this one compare to broke the code which tries to hide its own bugs behind respawning processes and tricks like this. From the last 10 years we did not found the bug that may cause to crash your program, in single-process programs, because there is no right to fail. Only one reboot taken place in 3 years for complete OS up gradation, on Linux 2.4 where HA Proxy has been installed to serve millions of pages every day. These people did not receive any patches so they were not directly exposed to the Internet. The kernel was a heavily patched 2.4 with Robert Love's jiffies64 patches to support time wrap-around at 497 days. On such systems, the software cannot fail without being immediately noticed [1].

Around the world in 500 Fortune companies to server millions of pages reliably per day and relay money in huge amounts. This is the default solution for simple problems that the few people even trust. These people will use the old versions 1.1 or 1.2 which has very limited evolutions and which targets mission-critical usages. This HA Proxy environment is more suited because the indicators provide more valuable information about the application's health, behavior and defects, which are used to make it even more reliable. Compare to 1.1 and 1.2, Version 1.3 has now received far more testing, so users are strongly encouraged to migrate to a stable 1.3 for mission-critical usages. The most of the work is done by OS, as already illustrated. Based on this the OS itself involves in large part of reliability. Higher level of stability is available in the recent versions of Linux 2.4. To achieve high level of performance, it requires a bunch of patches. Linux 2.6 includes the features needed to achieve this level of performance, but is not yet as stable for such usages. To fix bugs every month the kernel must have one up gradation. Few people prefer to run it on Solaris. Linux 2.4 and Solaris 8, 9 are showing same level of performance and are known to be really stable right now, but Solaris 10 might have the same code stability problem. If we placed few limits to our system, it reduces the reliability. This is why finely tuning the sysct ls is important. Every system and every application has their own rules; these are not in general for all. (Gary, 2010) However, it is important to ensure that the system will never run out of memory and that it will never swap. A correctly tuned system must be able to run for years at full load without slowing down or crashing.

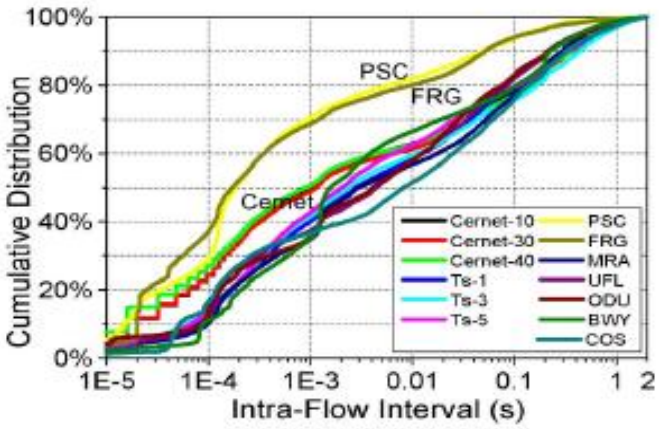
3.2. SECURITY

When we deploy the software load balancer the most important issue is the security. The load balancer will exposed by itself, but it is hard for OS, to limit the number of open ports and accessible services [1]. Based on this we are very careful in coding style. The new vulnerability was introduced when logs were reworked. This new cause will crash the process through BUS ERRORS, but it is not possible to execute code.Anyway, much care is taken when writing code to manipulate headers. From the creation destruction of a session that consists combinations of impossible states are checked, returned and processed the errors. For better clarity and ease of auditing, a few people around the world have reviewed the code and suggested cleanups. In this I refused patches that introduce suspect processing or in which not enough care is taken for abnormal conditions.I generally suggest starting HA Proxy as root because it can then jail itself in a cheroot and drop all of its privileges before starting the instances. This is not possible if it is not started as root because only root can execute cheroot ().To maintain fair level of security the Logs are more useful. UDP cherooted, the/dev/log UNIX socket is unreachable, and it must not be possible to write to a file because they will send only once. The following logs provide useful information:source IP and port of requestor make it possible to find their origin in firewall logs[1].Firewall log matching is done through session set up date, while tear down date often matches proxy's dates.From requestor side proper request encoding ensures he neither hide non-printable characters, nor fool a terminal.To find out scan attacks, arbitrary request and response header and cookie capture helps, proxies and infected hosts as well.To differentiate hand-typed requests we will use timers from browser.HA Proxy also provides reg ex-based header control. The parts in the request, request and response headers as well can be denied, allowed, removed, rewritten, or added. To block encodings and dangerous requests we can use this ordinarily used and to prevent accidental information leak from the server to the client. Other features such as Cache-control checking ensure that no sensible information gets accidently cached by an upstream proxy consecutively to a bug in the application server for example [1].

ISSN 22773061

4. DEFINITION

Flow Slice A flow slice is a sequence of packets in a flow, where every Intra flow interval between two consecutive packets is smaller than or equal to a slicing threshold. By bifurcation of intra flow we will get flow slices could be seen as small flows. In Fig. 4, we depict the Cumulative Distribution Functions (C.D.F) of intra flow intervals in our traces. In our tracing analysis we found intervals larger than 1 ms is more than 50% and more than 40% belongs to > 4 ms. Two exceptions are PSC and FRG.



The former is under a light load 10%, the latter is collected from a low-speed OC-12c network edge link. Thus, their weights are minimized. We denote the probability for an intra flow interval to be larger than by PC and the average packet count in the original flow by C0. After slicing, the average packet count in flow slice, denoted by FC, is calculated by FC ¼ C0=½1 (C0 - 1PC=< 1=PC: 4) Setting less than 1 ms, which leads to PC > 0:5, we obtain FC < 2. That is, each flow slice has no more than two packets in average. Even under a modest setting of ¼ 4 ms, we still have FC < 2:5. This reveals the very close load-balancing granularity of FS to the optimal packet-based solution.

5. IMPLEMENTATION ISSUES

The major implementation cost introduced by FS is that of hash table maintaining an active flow-slice context. The hash table size is a trade-off 1) a large hash table reduces hash collision probability. Tejeswini et al. Here, hash collision is defined as the situation when more than one flow slices are hashed into the same entry, which degrades load-balancing uniformity; while 2) a small hash table is desirable as the table is accessed in each load-balancing operation and on-chip SRAM with relatively small size allows higher access speed.

We calculate this trade-off in a Dual Hash Table (DHT) approach. Assume a hash collision probability below 0.5 percent as negligible, where only one flow slice out of 200 is not load balanced independently. Then, at OC-768c port with an average packet length of PL 400B and a slicing threshold of 4 ms. The hash table size is then bounded by 50MB, where each entry occupies 6B size. This is small enough to be placed on-chip. As each FS load-balancing operation only needs one query at the hash table, O(1) timing complexity is also achieved.

6. CONCLUSION

Our proposed novel scheme, includes the HA Proxy protocol is very fast and reliable solution for TCP and HTTP based applications to offer high availability, load balancing, and proxying. In particular this scheme appropriate for websites moving slowly towards very high loads in the mean time it needs persistence or processing of Layer7, and flow slices are generated through Flow Slice (FS) that halt each flow at every intra flow spell larger than a slicing threshold and balances the load. This Flow Slice load balancing model is based on intra flow packet interval i.e. between 40 to 50 % larger than the MPS delay upper bound. Our scheme suffers a little hardware overhead and O(1) time complexity and due to available of three positive characters of flow slice, our method reaches load-balancing uniformity in a better stage.

References:

[1] http://haproxy.1wt.eu/#tact

[2] Cisco CRS-1, http://www.cisco.com/go/crs/, 2011.

[2] Real Traces from NLANR, http://pma.nlanr.net/, 2010.

[3] Vitesse Intelligent Switch Fabrics, http://www.vitesse.com, 2011.

[4] A. Aslam and K. Christensen, “Parallel Packet Switching Using Multiplexors with Virtual Input Queues,” Proc. Ann. IEEE Conf. Local Computer Networks (LCN), pp. 270-277, 2002.

[5] J. Bennett, C. Partridge, and N. Shectman, “Packet Reordering Is Not Pathological Network Behavior,” IEEE/ACM Trans. Networking, vol. 7, no. 6, pp. 789-798, Dec. 1999.

[6] N. Brownlee and K. Claffy, “Understanding Internet Traffic Streams: Dragonflies and Tortoises,” IEEE Comm. Magazine, vol. 40, no. 10, pp. 110-117, Oct. 2002.

[7] Z. Cao, Z. Wang, and E. Zegura, “Performance of Hashing-Based Schemes for Internet Load Balancing,” Proc. IEEE INFOCOM, pp. 332-341, 2000.

[8] L. Carter and M. Wegman, “Universal Classes of Hashing Functions,” J. Computer and System Sciences, vol. 18, no. 2, pp. 143-154, 1979.

[9] Tejaswini N P Prasanna Kumar M, “Congestion Avoidance in MPS with Flow Cut,” IJARCSSE March 2013.

Author Profiles:

Miss C Surekha received her B.Tech in CSE from VITS-PDTR and now pursuing M.Tech (CSE) Vaagdevi Institute of Technology and Sciences, JNTU-Anantapur. For her dissertation work this paper prepared.

Mr.U.Sesadri received his M.Sc (CS) from SriVenkateswara University-Tirupati, M.Tech (CSE) from Satyabhama University. Working as HOD in CSE in Vaagdevi Institute of Technology and Sciences, under JNTU-Anantapur and have 10yearsofexperience.