iJRASET
International Journal For Research in Applied Science and Engineering Technology

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 7 **Issue:** IV **Month of publication:** April 2019

**DOI:** https://doi.org/10.22214/ijraset.2019.4250

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887*
*Volume 7 Issue IV, Apr 2019- Available at www.ijraset.com*

# Multiserver Communication in Distributed Database Management System

E.Elamathi[1], R.Elakkiyarasi [2], C.Amuthavani[3], Mr. A. Ganesan[4], Dr. N. Revathy[5]
[1, 2, 3] *Final MCA,* [4]*Associate Professor,* [5]*Professor, PG and Research Department of Computer Applications, Hindusthan College of Arts and Science, Coimbatore, India*

***Abstract:*** ***We describe a system called Parent Proxy that demonstrates an alternative approach to achieving consistent access to objects by edge proxies while retaining the proxies' load-dispersing and latency-reducing effects. Parent Proxy organizes the proxies in a tree rooted at the server. The tree is structured so that geographically close proxies reside close to one another in the tree. To perform certain types of operations, a proxy uses the tree to migrate each involved object to itself and then performs the operation locally. Although this incurs the expense of object migration for some operations and is thus reasonable only if objects are not too large and operations involve only a few, it also promises performance benefits of applications.***
***Quiver, a system that coordinates service proxies placed at the "edge" of the Internet to serve distributed clients accessing a service involving mutable objects. Quiver enables these proxies to perform consistent accesses to shared objects by migrating the objects to proxies performing operations on those objects. These migrations dramatically improve performance when operations involving an object exhibit geographic locality, since migrating this object into the vicinity of proxies hosting these operations will benefit all such operations. This system reduces the workload in the server. It performs the all operations in the proxies itself. In this system the operations performed in First-In-First-Out process. This system handles two process serializability and strict serializability for durability in the consistent object sharing. Other workloads benefit from Quiver, dispersing the computation load across the proxies and saving the costs of sending operation parameters over the wide area when these are large. Quiver also supports optimizations for single-object reads that do not involve migrating the object. We detail the protocols for implementing object operations and for accommodating the addition, involuntary disconnection, and voluntary departure of proxies. Finally, we discuss the use of Quiver to build an e-commerce application and a distributed network traffic modeling service.***

***Keywords: Parent Proxy, First-In-First-Out, Distributed network traffic modeling service, Web server.***

## I. INTRODUCTION

DYNAMIC Web services are examples of Internet-scale applications that utilize mutable objects. Following the success of content distribution networks (CDNs) for static content ,numerous recent proposals have attempted to scale dynamic Web services by employing service proxies at the "edge" of the Internet. This approach has the potential of both distributing the operation processing load among the proxies and enabling clients to access the service by communicating with nearby proxies rather than a potentially distant centralized server. A major challenge in this architecture, however, is to enable the (globally distributed) service proxies to efficiently access the mutable service objects for servicing client operations while ensuring strong consistency semantics for these object accesses.

Consistent object sharing among the proxies enables them to export the same consistent view of the service to the clients in turn. However, achieving even just serializability for operations executed at these proxies using standard replication approaches requires that a proxy involve either a centralized server or other (possibly distant) proxies on the critical path of each update operation. Serializability techniques require wide-area interactions for reads as well. Here, we describe a system called Quiver that demonstrates an alternative approach to achieving consistent access to objects by edge proxies while retaining the proxies' load dispersing and latency-reducing effects. Quiver organizes the proxies in a tree rooted at the server. The tree is structured so that geographically close proxies reside close to one another in the tree.

To perform certain types of operations, a proxy uses the tree to migrate each involved object to it and then performs the operation locally. Although this incurs the expense of object migration for some operations and is thus reasonable only if objects are not too large and operations involve only a few, it also promises performance benefits for two types of applications.

The first type is applications in which operations exhibit geographic locality: once an object has been migrated to a proxy, other operations (including updates) at that proxy involving this object can be performed locally in contrast to standard replication

techniques. Even operations at nearby proxies benefit, since the object is already close and need not be migrated far. Our use of a tree, through which migrations occur, is the key to realizing this benefit. Given the diurnal pattern of application activity that is synchronized with the business day and the fact that the business day occupies different absolute times around the world, we believe that exploiting workload locality through migration can play an important role in optimizing global applications.

The second type of applications that can benefit from Quiver are those for which migrating service objects to proxies and performing operations there is more efficient than performing all operations at a centralized server. One example is an application that involves large amounts of data that would be expensive to send to the server but for which objects remain small and, thus, can be migrated. Another is one for which processing load would overload a server, but because the operations involve diverse objects and are performed at different proxies, the load is naturally dispersed across proxies.

## II. LITERATURE REVIEW

1) *Proxy Server:* In computer networks, a proxy server is a server (a computer system or an application program) which services the requests of its clients by forwarding requests to other servers. A client connects to the proxy server, requesting some service, such as a file, connection, web page, or other resource, available from a different server. The proxy server provides the resource by connecting to the specified server and requesting the service on behalf of the client. A proxy server may optionally alter the client's request or the server's response, and sometimes it may serve the request without contacting the specified server. In this case, it would 'cache' the first request to the remote server, so it could save the information for later, and make everything as fast as possible. A proxy server that passes all requests and replies unmodified is usually called a gateway or sometimes *tunneling proxy*. A proxy server can be placed in the user's local computer or at various points between the user and the destination servers or the Internet.
2) *Types and Functions:* Proxy servers implement one or more of the following functions:

### A. *Caching Proxy Server*

A caching proxy server accelerates service requests by retrieving content saved from a previous request made by the same client or even other clients. Caching proxies keep local copies of frequently requested resources, allowing large organizations to significantly reduce their upstream bandwidth usage and cost, while significantly increasing performance. Most ISPs and large businesses have a caching proxy. These machines are built to deliver superb file system performance (often with RAID and journaling) and also contain hot-rodded versions of TCP. Caching proxies were the first kind of proxy server.

The HTTP 1.0 and later protocols contain many types of headers for declaring static (cacheable) content and verifying content freshness with an original server, e.g. ETAG (validation tags), If-Modified-Since (date-based validation), Expiry (timeout-based invalidation), etc. Other protocols such as DNS support expiry only and contain no support for validation.

Some poorly-implemented caching proxies have had downsides (e.g., an inability to use user authentication). Some problems are described in RFC 3143 (Known HTTP Proxy/Caching Problems).

### B. *Web Proxy*

A proxy that focuses on WWW traffic is called a "web proxy". The most common use of a web proxy is to serve as a web cache. Most proxy programs (e.g. Squid) provide a means to deny access to certain URLs in a blacklist, thus providing content filtering. This is usually used in a corporate environment, though with the increasing use of Linux in small businesses and homes, this function is no longer confined to large corporations. Some web proxies reformat web pages for a specific purpose or audience (e.g., cell phones and PDAs).

AOL dialup customers used to have their requests routed through an extensible proxy that 'thinned' or reduced the detail in JPEG pictures. This sped up performance, but caused trouble, either when more resolution was needed or when the thinning program produced incorrect results. This is why in the early days of the Internet many web pages would contain a link saying "AOL Users Click Here" to bypass the web proxy and to avoid the bugs in the thinning software.

### C. *Content-Filtering web Proxy*

A content-filtering web proxy server provides administrative control over the content that may be relayed through the proxy. It is commonly used in commercial and non-commercial organizations (especially schools) to ensure that Internet usage conforms to acceptable use policy.

Common methods used for content filtering include: URL or DNS blacklists, URL regex filtering, MIME filtering, or content keyword filtering. Some products have been known to employ content analysis techniques to look for traits commonly used by certain types of content providers.

A content filtering proxy will often support user authentication, to control web access. It also usually produces logs, either to give detailed information about the URLs accessed by specific users, or to monitor bandwidth usage statistics. It may also communicate to daemon based and/or ICAP based antivirus software to provide security against virus and other malware by scanning incoming content in real time before it enters the network.

*D. Anonym Zing Proxy Server*

An anonymous proxy server (sometimes called a web proxy) generally attempts to anonymize web surfing. These can easily be overridden by site administrators, and thus rendered useless in some cases. There are different varieties of anonymizers.

1) *Access Control:* Some proxy servers implement a logon requirement. In large organizations, authorized users must log on to gain access to the web. The organization can thereby track usage to individuals.

*E. Hostile Proxy*

Proxies can also be installed by online criminals, in order to eavesdrop upon the dataflow between the client machine and the web. All accessed pages, as well as all forms submitted, can be captured and analyzed by the proxy operator. For this reason, passwords to online services (such as webmail and banking) should always be exchanged over a cryptographically secured connection, such as TLS.

*F. Intercepting Proxy Server*

An intercepting proxy (also known as a "transparent proxy") combines a proxy server with a gateway. Connections made by client browsers through the gateway are redirected through the proxy without client-side configuration (or often knowledge).

Intercepting proxies are commonly used in businesses to prevent avoidance of acceptable use policy, and to ease administrative burden, since no client browser configuration is required.

It is often possible to detect the use of an intercepting proxy server by comparing the external IP address to the address seen by an external web server, or by examining the HTTP headers on the server side.

*G. Transparent and Non-Transparent Proxy Server*

The term "transparent proxy" is most often used incorrectly to mean "intercepting proxy" (because the client does not need to configure a proxy and cannot directly detect that its requests are being proxied). Transparent proxies can be implemented using Cisco's WCCP (Web Cache Control Protocol). This proprietary protocol resides on the router and is configured from the cache, allowing the cache to determine what ports and traffic is sent to it via transparent redirection from the router. This redirection can occur in one of two ways: GRE Tunneling (OSI Layer 3) or MAC rewrites (OSI Layer 2).

However, RFC 2616 (Hypertext Transfer Protocol -- HTTP/1.1) offers different definitions:

"A 'transparent proxy' is a proxy that does not modify the request or response beyond what is required for proxy authentication and identification".

"A 'non-transparent proxy' is a proxy that modifies the request or response in order to provide some added service to the user agent, such as group annotation services, media type transformation, protocol reduction, or anonymity filtering".

*H. Forced Proxy*

The term "forced proxy" is ambiguous. It means both "intercepting proxy" (because it filters all traffic on the only available gateway to the Internet) and its exact opposite, "non-intercepting proxy" (because the user is forced to configure a proxy in order to access the Internet).

Forced proxy operation is sometimes necessary due to issues with the interception of TCP connections and HTTP. For instance interception of HTTP requests can affect the usability of a proxy cache, and can greatly affect certain authentication mechanisms. This is primarily because the client thinks it is talking to a server, and so request headers required by a proxy are unable to be distinguished from headers that may be required by an upstream server (esp authorization headers). Also the HTTP specification prohibits caching of responses where the request contained an authorization header.

*I. Open Proxy Server*

Because proxies might be used to abuse, system administrators have developed a number of ways to refuse service to open proxies. Many IRC networks automatically test client systems for known types of open proxy. Likewise, an email server may be configured to automatically test e-mail senders for open proxies.

Groups of IRC and electronic mail operators run DNSBLs publishing lists of the IP addresses of known open proxies, such as AHBL, CBL, NJABL, and SORBS.

The ethics of automatically testing clients for open proxies are controversial. Some experts, such as Vernon Schryver, consider such testing to be equivalent to an attacker portscanning the client host. [1] Others consider the client to have solicited the scan by connecting to a server whose terms of service include testing.

*J. Reverse Proxy Server*

A reverse proxy is a proxy server that is installed in the neighborhood of one or more web servers. All traffic coming from the Internet and with a destination of one of the web servers goes through the proxy server. There are several reasons for installing reverse proxy servers:

*1)* Encryption / SSL acceleration: when secure web sites are created, the SSL encryption is often not done by the web server itself, but by a reverse proxy that is equipped with SSL acceleration hardware. Furthermore, a hoster can provide a single "SSL proxy" to provide SSL encryption for an arbitrary number of hosts; removing the need for a separate SSL Server Certificate for each host, with the downside that all hosts behind the SSL proxy have to share a common DNS name or IP address for SSL connections.

*2)* Load balancing: the reverse proxy can distribute the load to several web servers, each web server serving its own application area. In such a case, the reverse proxy may need to rewrite the URLs in each web page (translation from externally known URLs to the internal locations).

*3)* Serve/cache static content: A reverse proxy can offload the web servers by caching static content like pictures and other static graphical content.

*4)* Compression: the proxy server can optimize and compress the content to speed up the load time.

*5)* Spoon feeding: reduces resource usage caused by slow clients on the web servers by caching the content the web server sent and slowly "spoon feeds" it to the client. This especially benefits dynamically generated pages.

*6)* Security: the proxy server is an additional layer of defense and can protect against some OS and WebServer specific attacks. However, it does not provide any protection to attacks against the web application or service itself, which is generally considered the larger threat.

## III. PROPOSED SYSTEM

*A.* This system forms the proxies in the tree structure. It shares the objects within the proxies. It reduces the workload in the server.

*B.* Quiver enables consistent multi object operations and optimizations for single-object reads that are not possible in these prior algorithms.

*C.* This system recovers the proxy disconnection. The disconnected proxies maintained by alternate proxies or it will be maintained through server.

*D.* This System use the kruskal's algorithm for maintaining tree structure. It reduces weight age in the tree structure.

*E.* It holds the object even when the proxy has been disconnected.

## IV. MODULE DESIGN

*1)* Create the centralized server and proxies.

*2)* Object Migration from centralized server to proxies.

*3)* Parent Proxy and Child Proxy Maintenance while disconnection.

*4)* Proxy Tree Maintenance using Kruskal's algorithm.

*5)* Multiple object and Consistent object sharing.

*A. Module Description*

*1)* *Module-1:* In this module the user is going to launch a centralized server, unbounded number of parent proxies and minimum number of child proxies. This server will provide the application to all the parent proxies.

2) *Module-2:* This module deals with object migration its nothing but transferring our application from centralized server to the parent proxies. The server converts the whole application to an object. Then the parent proxies provides object to the child proxies.
3) *Module-3:* This module provides the alternate solution when the proxies getting down. The client will get the response from some other proxies.
4) *Module-4:* Here we are going to maintain the depth of the tree using kruskal's algorithm. This algorithm calculates the round trip delay between the proxies. Through that it will found the data round trip delay.
5) *Module-5:* In this module we going to share the object consistently without any interruption.

## V. SYSTEM IMPLEMENTATION

### A. *Code Optimization*

Code optimization aims at improving the efficiency of the program. This is achieved in two ways:

1) Redundancies in a program are eliminated.
2) Computations in a program are rearranged or rewritten to make it execute efficiently.

First, optimization seeks to improve a program rather than the algorithm used in a program. Thus replacement of an algorithm is beyond the scope of optimization. Second, efficient code generation for a specific target machine is also beyond its scope; it belongs in the back end of a complier.

### B. *Optimizing Transformations*

An optimizing transformation is a rule for rewriting a segment of a program to prove its execution efficiency without affecting its meaning. Optimizing transformations are classified into local and global transformations.

A few optimizing transformation commonly used is as follows:

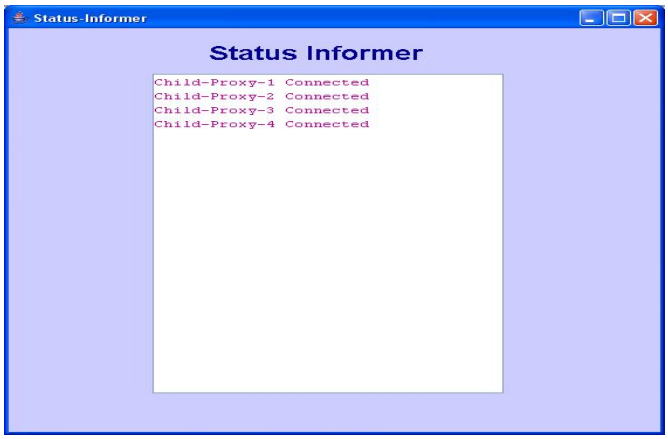
1) Compile time evaluation.
2) Elimination of common sub expressions.
3) Dead code elimination.
4) Frequency reduction.
5) Strength reduction.

These are the main transformation that is mainly used in the project.

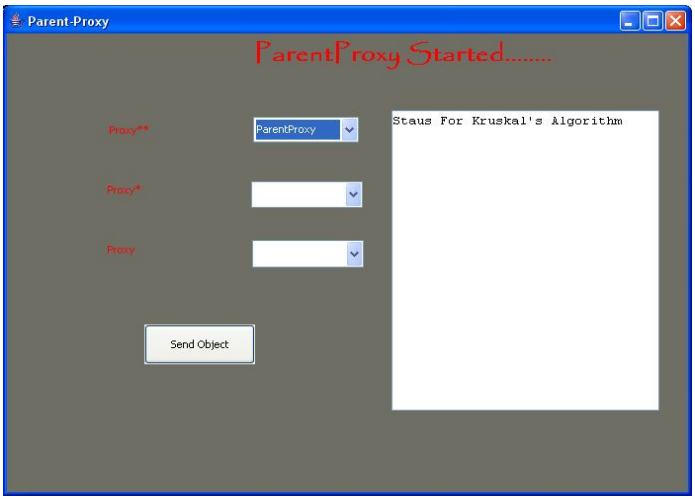
## VI. EXPERIMENTAL RESULTS

### A. *Proxy Maintenance*

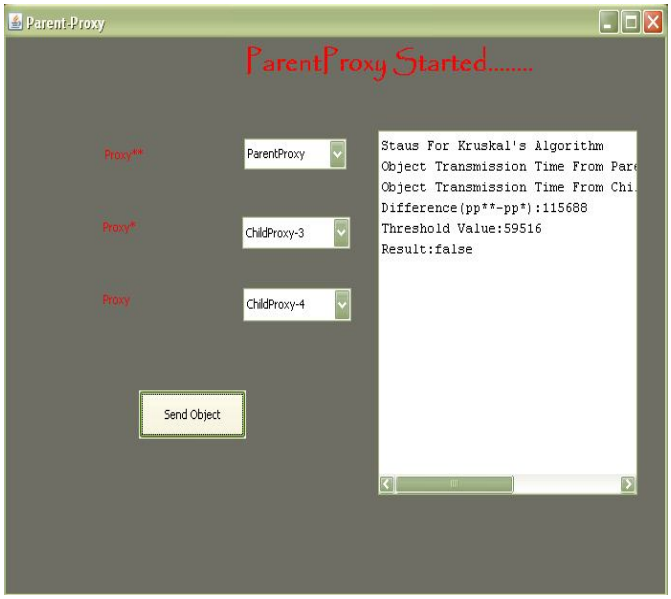
*B. Status Informer*



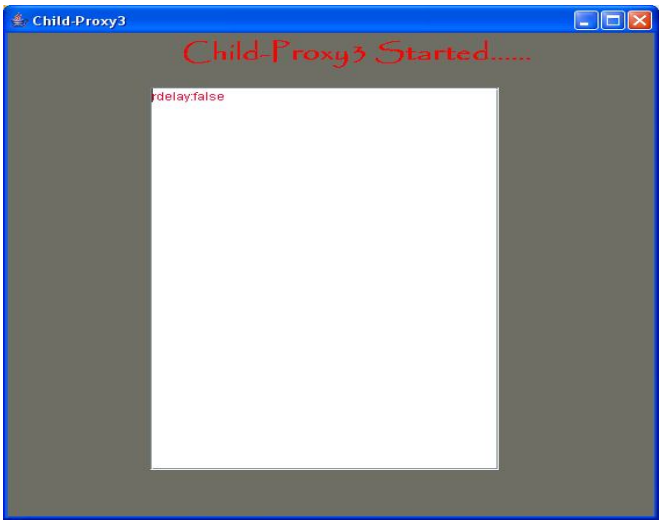

*C. Parent Proxy*



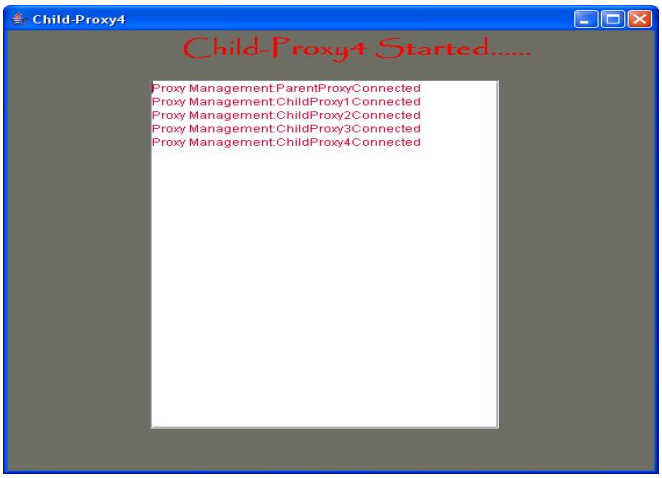

*D. Parent Proxy*



*E. Child Proxy 3*




*F. Child Proxy 4*




*G. Scope For Future Enhancement*

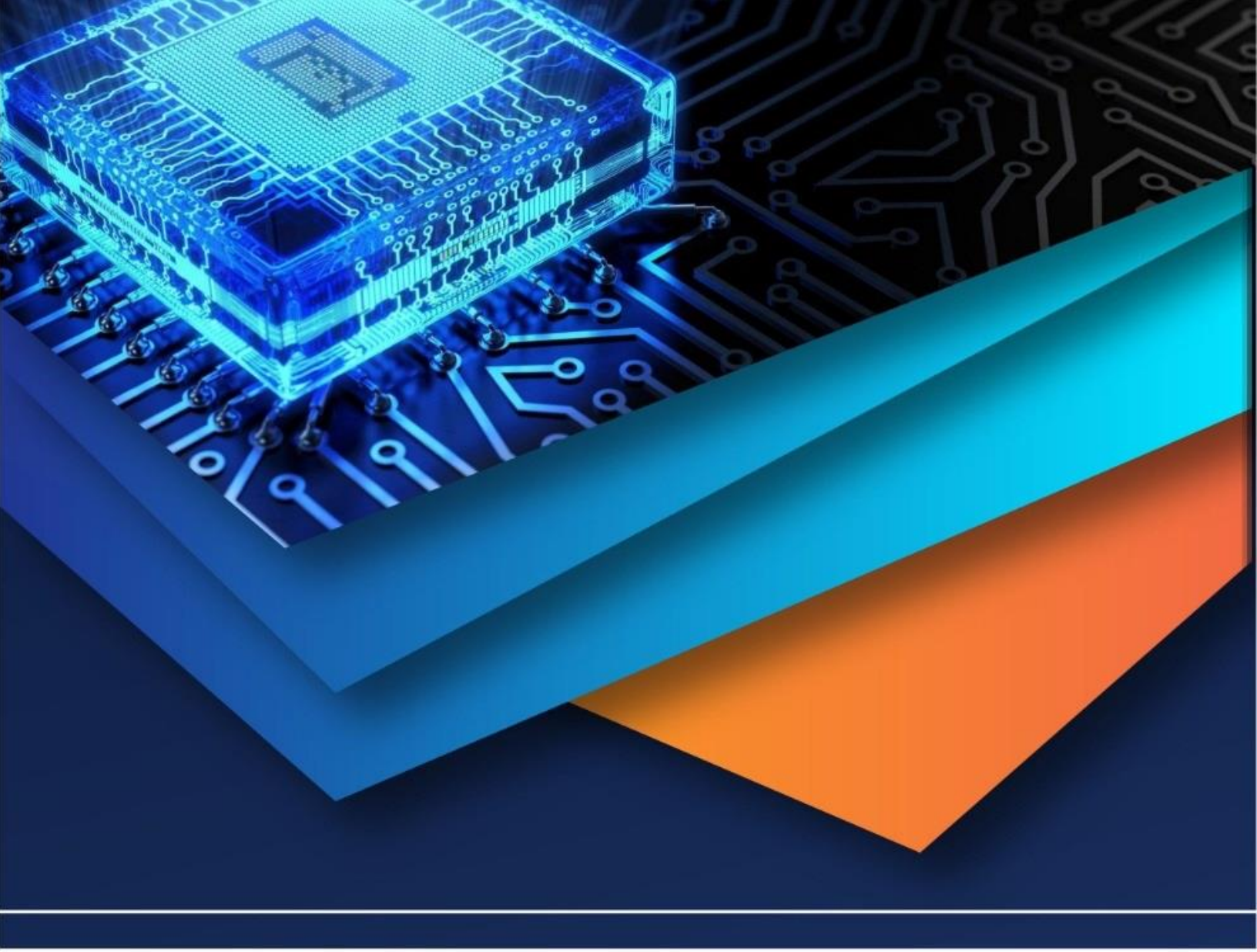
To characterize future traffic as belonging to a certain application, say, to identify prohibited traffic on nonstandard ports. In the Quiver setting, each contributing network includes a proxy. A coordination site acts as the server (root of the tree), and the traffic classifiers being constructed are the objects. The proxies perform update and read operations on the traffic classifiers. To update a particular classifier using new records, the contributing network's proxy migrates the classifier to itself and updates the classifier using its data. The algorithms for incrementally updating the classifier require the updates to be serialized, an important property achieved through Quiver. Furthermore, this strategy distributes the computational load of updating the classifiers to all the networks and does not require any network to reveal its raw data to any other entity.

## VII. CONCLUSION

We presented a system called Quiver for implementing highly scalable object sharing with strong consistency semantics (serializability). Quiver is well suited to workloads where operations typically access few objects and where operations involving each object exhibit geographic locality or are computationally intensive. Quiver migrates objects to proxies in order to perform update or multi object operations while supporting more efficient single-object reads. Proxies may join, leave, or disconnect from the service. In case of disconnects, the service recovers objects whose latest versions are left unreachable. We evaluated Quiver over Plane tLab to confirm the performance improvements that it offers for various workloads.

## REFERENCE

[1] Elliotte Rusty Harold, “Java Network Programming”, Published by O'Reilly, III Edition, 2004.
[2] Ken Arnold, James Gosling,”The Java Programming Language”, Published by Addison-Wesley, 1996.
[3] Harvey M Deitel, Paul J Deitel “Java: How to Program”, Published by Prentice Hall, VII Edition, 2007.
[4] Kim N. King “Java Programming: From the Beginning”, Published by Norton, 2000.
[5] Marty Hall, ”Core Servlets and JavaServer Pages” Published by Prentice Hall PTR, 2000.
[6] Y. Daniel Liang, “Introduction to Java Programming: Brief Version”, Published by Prentice Hall, 2008.
[7] Todd Courtois “Java Networking and Communications”, Published by Prentice Hall PTR, 1997
[8] Jim Farley “Java distributed computing”, Published by O'Reilly, 1998
[9] Esmond Pitt “Fundamental Networking in Java”, Published by Springer, 2006
[10] Justin Couch “Java 2 Networking”, Published by McGraw-Hill, 1999
[11] DBProxy: A Dynamic Data Cache for Web Applications.
[12] Y. Amir, C. Danilov, M. Miskin-Amir, J. Stanton, and C. Tutu,“Practical Wide-Area Database Replication,” Technical Report CNDS-2002-1, Johns Hopkins Univ., 2002.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY