Policy-based heterogeneous server utilisation using controller framework

Aditi Bankura* and Anirban Kundu

Netaji Subhash Engineering College,
Kolkata 700152, India
and
Computer Innovative Research Society,
Howrah 711103, India
Email: aditi.bankura@gmail.com
Email: anik76in@gmail.com
*Corresponding author

Abstract: In this paper, authors have proposed a controller based framework having transmission flow for query search and/or query responses. In this approach, total structure is divided into number of levels, and each level has two set of controllers. Servers are worked together in each set of controllers. Servers are heterogeneous in nature based on functionality and configurations. Server selection in each level, communication establishment, and transmission of information between two servers placed at two consecutive levels are three major tasks to be executed for entire query search processing. Several policies are proposed for communication establishment and transmission of information between servers with consideration of risks management. Several load management strategies have been proposed for server selection dynamically from set of available servers having distinct loads using load balance factor. In this paper, we have also introduced a procedure to follow two separate paths for transmission of search query and query response to avoid congestion in network to achieve minimum delay in query response.

Keywords: query searching; heterogeneous servers; packet formation policy; data migration policy; data block placement policy; failure policy; data block replication policy; server feedback policy; entry controller; exit controller; server pool.

Reference to this paper should be made as follows: Bankura, A. and Kundu, A. (2022) 'Policy-based heterogeneous server utilisation using controller framework', *Int. J. Internet Protocol Technology*, Vol. 15, No. 1, pp.8–28.

Biographical notes: Aditi Bankura is an Assistant Professor in Computer Science & Engineering Department of Netaji Subhash Engineering College, Kolkata, India. Prof. Bankura is associated with Computer Innovative Research Society, West Bengal, India. She received her BTech and MTech degree from Maulana Abul Kalam Azad University of Technology (formerly known as West Bengal University of Technology) in 2005 and 2008, respectively. Her current research interests are related with search engine oriented indexing, web services and distributed computing.

Anirban Kundu is an Associate Professor in Information Technology Department of Netaji Subhash Engineering College, Kolkata, India. He is associated with Computer Innovative Research Society, West Bengal, India. Previously, he worked as Post-Doctoral Research Fellow at Kuang-Chi Institute of Advanced Technology, Shenzhen, China. He also worked as foreign expert under Municipality Government of Shenzhen during 2011 to 2014. He was a Research Fellow in the Web Intelligence and Distributed Computing Research Lab (WIDiCoReL). He received his BE Mechanical from Bangalore University (1999), Post Graduate Diploma in Financial Management from Management Studies Promotion Institute (2001), MTech (IT) from Bengal Engineering and Science University (2004), and PhD (Engineering) in Computer Science from Jadavpur University (2009). His research interests include search engine-oriented indexing, ranking, prediction, web page classification, semantic web (ontology-based) and natural language processor with the essence of cellular automata, multi-agent-based system design, fuzzy controlled systems and cloud computing.

1 Introduction

1.1 Overview

Digital world is a collection of data and/or information and maximum amount of data is collected from electronic devices connected through internet. Initially, usage of computer network was not effective and some computers and hardware computing devices are considered as nodes within homogeneous network for resource sharing and communication establishment. Billions of heterogeneous nodes are possible due to evolution of internet (Keller et al., 2014).

Client-server network architecture is used during resource sharing among nodes within network. In client-server architecture, large number of clients is connected to get information from server. Number of servers is worked within a network and functionality categorisation of servers is accomplished for faster request processing generated from client and faster response to client. A set of communication protocol is needed between two nodes to maintain synchronisation and avoid information loss during resource sharing and communication establishment. File transfer protocol (FTP), mail transfer protocol (MTP), hypertext transfer protocol (HTTP) are used for file, mail and multimedia files transmission through cline-server architecture model. Two different approaches (2-tier and 3-tier) are available to implement client-server architecture. In 2-tier architecture, client is able to access information from database server directly as business logic is written at client-end. In 3-tier architecture, client application, application server and database server are worked together. Application server is worked as middleware between client application and database server (Oluwatosin, 2014).

Mode of resource sharing through internet has been changed drastically after innovation of cloud computing. Mainly, three types of resource sharing have been accomplished through cloud – Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) (Sultan, 2010).

Voluminous amount of data is generated due to advancement of technology. Query search over World Wide Web (WWW) through search engine is a procedure for transmission of search query from user to database level and retrieval of information called as 'query response' from database level and transmission it to user. Big data has been introduced to handle voluminous data.

Big data is a collection of large volume of data. Voluminous data is generated from various distributed sources, such as social media, sensor devices, business, marketing, finance. Data formats are different as generated from different sources. Growth rate of data is increasing exponentially with respect to time (Oussous et al., 2017). Main characteristics of big data are defined (Lakshmi and Kumar, 2016) as follows:

i Volume: Large size of data volume is considered as big data.

ii Variety: Big data is a collection of 'variety' of data i.e., collection of structured, unstructured, semi structured data.

iii Velocity: Speed of data generation and processing of data refers to velocity of data.

iv Veracity: Accountability of data in big data refers veracity of data.

v Value: In Big data, large amount of values with different types is to be stored in database

Different types of analyses are required depending on characteristics of Big data, such as ensemble analysis, deep analysis (Shu, 2016).

Typical database management tools are unable to deal with Big data as size of data is huge and complex with varieties. Several tools are available for analysing Big data. Apache Hadoop is most popular software which mainly works in distributed environment with Map Reduce technique in Hadoop distributed file system (HDFS). Main challenges of big data are to provide high performance platform able to store, compute, and analyse data for knowledge discovery (KDD) (Acharjya et al., 2016). Clustering is used for analysing data and KDD. All typical clustering methods are not useful for Big data (Tulgar et al., 2018). Selection of clustering technique is dependent on application type (Sanse et al., 2015; Sajana et al., 2016; Kurasova et al., 2014). Most popular clustering technique is K-means clustering algorithm which is applied on Map-Reduce technique in which tasks are distributed among several nodes during analysis and information extraction of big data (Jain and Verma, 2014; Eren et al., 2015; Sreedhar et al., 2017). Several mechanisms are used to reduce size of data and number of features (Rehman et al., 2016). Large volume of data processing requires improvement in parallel computing with high scalability working within distributed environment for performance enhancement (Zerhari et al., 2015). Another challenge in big data is to retrieve information with accuracy and minimum response time during query processing within a network. Selection of server from server pool, redirection of information, indexing strategy, and data placement strategies are key elements for fast query processing through internet (Adamu et al., 2015; Gani et al., 2016; Mittal, 2017). Content delivery network (CDN) is used to replicate content among servers for correctness of content and increase availability of content (Dhanalakshmi et al., 2017; Sahoo et al., 2016). Data migration technique is needed during redirection of information from one location to another location. Multilingual search engine is possible if migration between two databases with different formats is possible (Ahmadi, 2012).

1.2 Literature review

Crawler is an important application of search engine through which web pages are downloaded from distributed servers through internet and stored into repository. Parsing and indexing of web pages are being executed to store data within database. Parsing is used to identify possible errors in pages where as performance of search engine is enhanced through proper usage of indexing techniques (Brin and Page, 1998). Fast response of query searching with accuracy is one of major challenges for any search engines.

Search engine works on enormous number of keywords, and retrieves set of web pages from server site database. Orientation of storage information about web pages is dependent on indexing mechanism. In this sub-section, forward indexing and inverted indexing have been considered as indexing techniques.

In forward indexing, keywords of each page are stored for conversion from page to keywords. The following steps are included to perform forward indexing:

Step 1: Fetch one page and collect all keywords.

Step 2: Append all keywords in index entry for each web page.

Step 3: Repeat step 1 and step 2 for all pages.

Formation of forward indexing is fast and needs space to store unique and/or redundant keywords. All pages are to be searched during query searching process. Searching from large size of indexing is time consuming.

In inverted indexing, related web pages are stored for each keyword. Inverted indexing is accomplished with following steps:

Step 1: Fetch one page and collect all keywords.

Step 2: Check presence of each keyword

Step 3: If present, add reference of that page to index entry else create new entry in index entry and add reference of that page.

Step 4: Repeat step 1 to step 3 for all pages.

Step 5: Sort keywords

Formation of inverted indexing is time consuming as there is keyword checking. In this technique, redundant keywords are not stored. As a result, index size is less than forward indexing. Query searching is fast as searching is dependent on keywords. There is no need to search entire index like forward indexing technique (Elaraby et al., 2012).

Web materials are downloaded from WWW through multiple crawlers. Single crawler (SCrw) and parallel crawler (PCrw) are two major techniques used during crawling for fixed number of crawlers. In hierarchical crawler (HCrw), crawlers are created dynamically during crawling, and performance of HCrw is better than SCrw and PCrw (Kundu et al., 2009).

Web materials are organised with context based indexing strategy in which context of document is important than keywords. Context of document is determined through thesaurus, context repository and ontology repository. Indexing is performed depending on context of documents having context, term and reference of document (Gupta et al., 2010; Mukhopadhyay et al, 2010; Yu 2019).

1.3 Aim

Large amount of data is stored in different servers placed at different locations connected through internet. Our aim is to redirect query to locate database server from set of available servers within a network in such way that delivery time of query response should be minimum and accuracy of information retrieval should be maximum.

1.4 Scope

Dealing with heterogeneous servers is more critical than homogeneous servers. Efficient utilisation of heterogeneous servers increases overall performance of system. Scope of this paper is to place data at appropriate servers with proper communication establishment among servers for effective data transmission.

1.5 Motivation

Network congestion and long queue maintained by each server are main reasons for poor performance of a system framework. Network congestion is dependent on network framework. Improvement in network framework reduces network congestion, such as high power bandwidth improves network speed with minimum delay. Numbers of tasks are arrived simultaneously to a server within a network. One task is processed by a server at a time and a long queue is to be maintained for remaining tasks, and hence performance of a system is reduced. Distribution of tasks among servers has a great impact on a system performance. Load distribution among servers is a main motivation of our task to enhance system performance. Motivation of our task is to design a system framework which is able to work in balanced mode with establishment communication among homogeneous and heterogeneous system to transmit data with minimum delay.

1.6 Novelty

Requirement for search query processing is the minimum delay in query response with maximum accuracy in network. Communication establishment, task distribution and congestion control among servers are the three major activities to process search queries. In proposed framework, data integrity during communication establishment with risk management is accomplished with several policies designing. Overall performance, stress, throughput of servers are maintained using load management strategies developed based on load balancing factors in homogeneous and heterogeneous servers for task distribution. Search query and query response are transmitted using two different paths in proposed framework to avoid network congestion and to minimise network delay within network.

1.7 Organisation

Rest of the paper is organised as follows: in Section 2, proposed work has been discussed; policy design, procedure, and some theoretical discussions are included in Section 2; experimental discussions have been depicted in Section 3; in Section 4, conclusion has been drawn.

2 Proposed work

In this paper, we have proposed a system framework to search information from database within a computer network.

2.1 Proposed framework

The architecture of our proposed system framework is shown in Figure 1. Proposed system framework is constituted with four levels such as user level, interface level, application level, and database level. Each level has set of entry controllers and set of exit controllers except user level.

The task of an entry-controller is to send query from a server of one level to next level. The task of an exit-controller is to send query response from a server to another server in opposite direction.

Figure 1 Proposed system framework

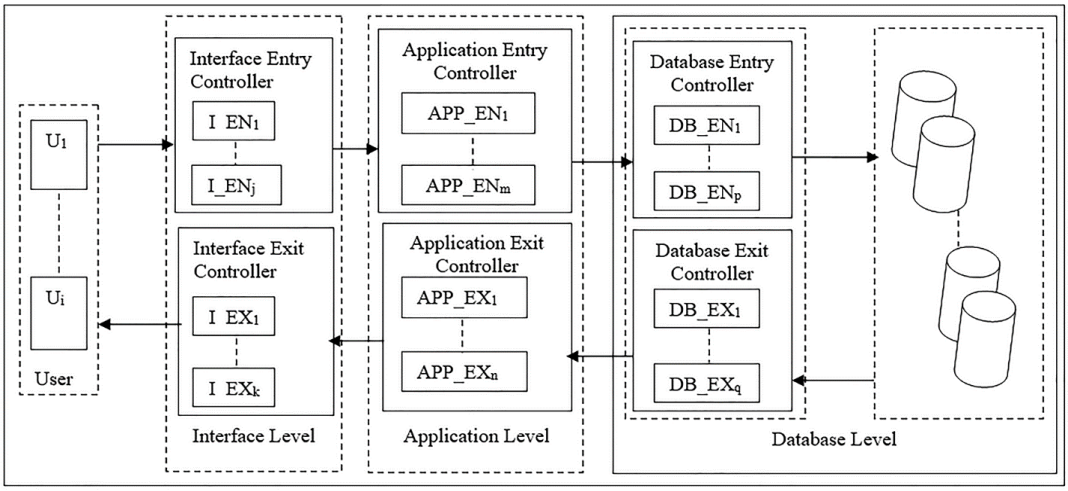
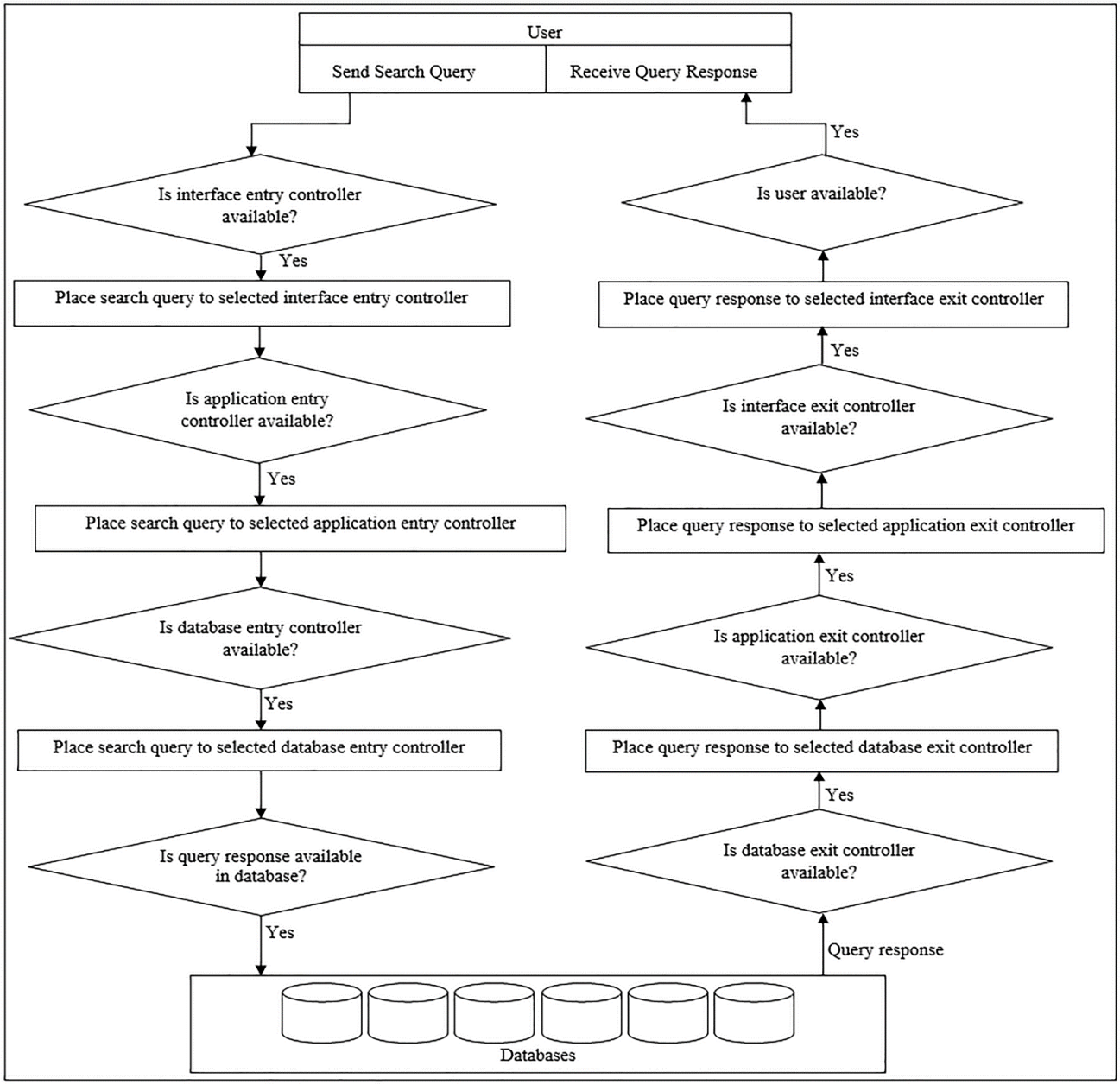


Figure 2 Working procedure of proposed system framework



The working procedure (refer Figure 2) of newly designed searching technique is described using proposed system framework as follows:

Step 1: Send query from a particular user to interface.

Step 2: Query is received by an interface entry controller at interface level and forward it to the application level.

Step 3: In application level, query is received by an application entry controller and forward it to the database level.

Step 4: Query is received by a database entry controller at database level and select a particular database.

Step 5: Query response is then generated from selected database and sends it to database level.

Step 6: In database level, query response is received by a database exit controller and send it to application level.

Step 7: In application level, query response is received by an application exit controller and sent to interface level.

Step 8: In interface level, query response is received and send to a particular user by an interface exit controller.

2.2 Policy design

Information is always carried out from one level to another level in proposed system framework. A set of strong communication policies is required. Six policies have been designed for performance enhancement of proposed system framework within network as follows:

- Packet Formation Policy (PFP) – Information is to be broken into number of small units of packets through PFP before transmission of information.
- Data Migration Policy (DMP) – Transmission of packet from one server in one level to another server at different levels through DMP after formation of packets.
- Data Block Placement Policy (DBPP) – Placement of packet in appropriate server among several servers is accomplished by DBPP.
- Failure Policy (FP) – FP is used to increase possibility of allocation of working server from set of available servers during process of transmission.
- Data Block Replication Policy (DBRP) – This is used to increase availability of information at time of server failure.
- Server Feedback Policy (SFP) – Performance of overall proposed framework is measured using SFP.

The detailed description of each policy is discussed as follows:

Policy 1: Packet Formation Policy (PFP)

In packet formation policy, a packet is formed using different fields as follows:

Step 1: Point out initial position of data block of particular size "Data_Size"

Step 2: Add SIPA, SP, DIPA and DP adjacent to data block

Step 3: Add State_Type to keep track of the present location (status) of data block

Step 4: Add PKT_ID to keep information about packet sequence

Step 5: Header and trailer are attached to wrap the whole unit

Structure of the packet after formation of packet is depicted in Figure 3.

PKT_ID – Unique identification number of packet to identify a packet sequence;

H – Header of packet;

T – Trailer of packet;

SIPA – Source Internet Protocol address;

SP – Source Port number;

DATA – Data payload;

DIPA – Destination Internet Protocol address;

DP – Destination Port number;

State_Type – Present location of packet such as Source state(S), Destination state (D) and Transient state (T);

Data_Size – Data payload size;

Figure 3 Packet structure in PFP

Field Size	2 Bytes	2 Bytes	1 Bytes	4 Bytes	2 Bytes	2 Bytes	1439 Bytes	2 Bytes	4 Bytes	2 Bytes
Field Name	H	PKT_ID	State_Type	SIPA	SP	Data_Size	DATA	DP	DIPA	T

Figure 4 Packet control block (PCB)

PKT_ID
Pointer to Source Port (SP)
Pointer to Destination Port (DP)
Pointer to Source Internet Protocol Address (SIPA)
Pointer to Destination Internet Protocol Address(DIPA)
State Type
Pointer to Data Payload(DATA)
Data_Size

Packet formation (refer Figure 3) is controlled by the packet control block (PCB) (refer Figure 4). PCB block consists of eight fields as follows:

PKT_ID – Packet sequence number used within a packet;

Pointer to Source Port (SP) – Points to initial position of source port number in packet;

Pointer to Source Internet Protocol Address (SIPA) – Points to initial position of source internet protocol address in packet;

Pointer to Destination Port (DP) – Points to initial position of destination port number in packet;

Pointer to Destination Internet Protocol Address (DIPA) – Points to initial position of destination internet protocol in packet;

Pointer to Data Payload (DATA) – Points to initial position of data payload in packet;

Data_Size – Store information about data payload size;

State Type – Present location (status) of packet such as source(S), destination (D) and transient (T);

In packet formation, required storage space in memory is dependent on size of a packet as well as size of PCB. Size of a packet and size of PCB are1460 bytes (refer Figure 3) and 15bytes (refer Figure 13) respectively. Total storage space needed for packet formation is equal to the size of data packet and size of PCB (i.e., 1475 bytes).

Policy 2: Data Migration Policy (DMP)

In data migration policy, a data block is transmitted from one server to another server (refer Figure 5) within proposed server-side network as follows:

Step 1: Start loop

Step 2: Send data packet from source socket to interface socket as prepared by PFP (refer Policy 1)

Step 3: Search specific route for destination using typical look-up method

Step 4: Send data packet from interface socket to destination socket

Step 5: Go to next level look-up

Figure 5 Flowchart of DMP

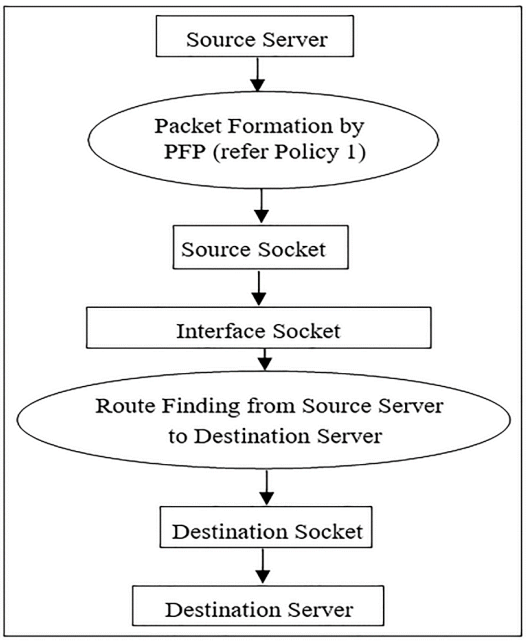


Figure 6 Data migration control block (DMCB)

PKT_ID
Source Socket Port (SSP)
Destination Socket Port (DSP)
Source Internet Protocol Address (SIPA)
Destination Internet Protocol Address (DIPA)
Source Socket Address (SSA)
Destination Socket Address (DSA)
State Type

A data migration control block is used (DMCB) (refer Figure 6) to keep track of all information required during data migration (DM). DMCB consists of eight fields as follows:

PKT_ID – Packet sequence number to be transmitted;

SIPA – Source Internet Protocol Address;

DIPA – Destination Internet Protocol Address;

SSA – Source Socket Address;

DSA – Destination Socket Address;

SSP – Source Socket Port Number;

DSP – Destination Socket Port Number;

State Type – Present location of data packet such as Source state(S), Destination state (D) and Transient state (T);

Total required memory storage space is equal to size of DMCB as only one DMCB is used in data migration. Size of DMCB is 23 bytes (refer Figure 13). Only 23 bytes memory space is needed during data migration.

Policy 3: Data Block Placement Policy (DBPP)

In data block placement policy, a data packet prepared by PFP (refer Policy 1) is placed at i^{th} destination server (refer Figure 7) as follows:

Step 1: Data packet is received by destination as transmitted by DMP (refer Policy 2)

Step 2: Extract header and trailer from data packet

Step 3: Extract PKT_ID, State_Type, SIPA, SP, Data_Size, DP and DIPA from packet

Step 4: Find initial position of data payload (DATA)

Step 5: Place DATA with "Data_Size" to destination server.

Figure 7 Flowchart of DBPP

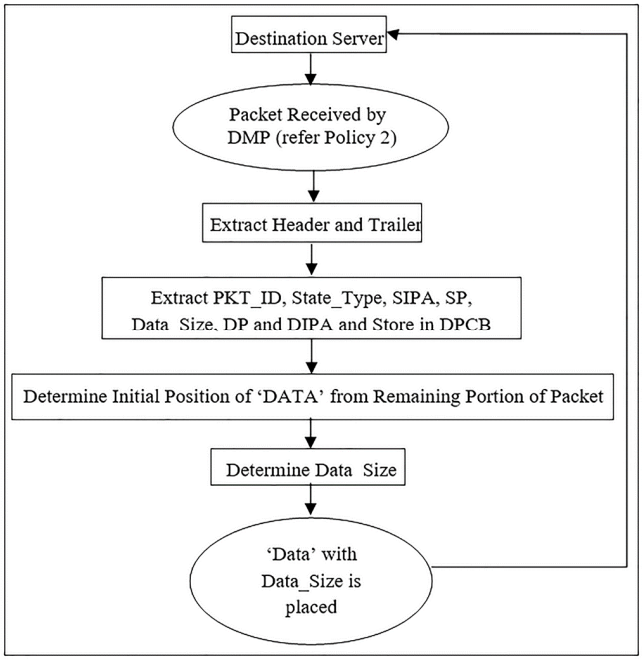


Figure 8 Data placement control block (DPCB)

PKT_ID
Source Internet Protocol Address (SIPA)
Source Port (SP)
Data_Size
State Type
Destination Internet Protocol Address (DIPA)
Destination Port (DP)
Pointer to Data Payload (DATA)

A data placement control block (DPCB) (refer Figure 8) is used to monitor placement of a data block in destination.

DPCB consists of eight fields as follows:

PKT_ID – Packet sequence number to be transmitted;

SIPA – Source Internet Protocol Address;

SP – Source Port number;

DIPA – Destination Internet Protocol Address;

DP – Destination Port number;

Pointer to Data Payload (DATA) – Points to initial position of data payload in packet;

Data_Size – Data payload size;

State Type – Set present status of packet from T (transient) to D (destination) for successful transmission;

In data block placement, required memory storage space is equal to size of DPCB. DPCB needs total 19 bytes (refer Figure 13) storage space in memory.

Policy 4: Failure Policy (FP)

In failure policy, possibility of packet (prepared in Policy 1) transmission from source to destination is checked as follows:

Step 1: Set timer "T"

Step 2: Send a packet from source to destination

Step 3: Wait for time "T"

Step 4: Set "ACK_TYPE" as "N" if acknowledgement generated by destination is not received by source

Figure 9 Failure control block (FCB)

PKT_ID
Source Internet Protocol Address (SIPA)
Destination Internet Protocol Address (DIPA)
Acknowledgement Status (ACK_TYPE)
Time (T)

A failure control block (FCB) (refer Figure 9) is used to keep information about server failure. FCB consist of five fields as follows:

PKT_ID – Packet sequence number;

SIPA – Source Internet Protocol Address;

DIPA – Destination Internet Protocol Address;

ACK_TYPE – Acknowledgement status such as acknowledgement received (Y) and not received (N) by source;

Time – Waiting response time from server in milliseconds;

Total 16 bytes (refer Figure 13) storage space is required in memory as only one FCB is used in FP. Risk management is achieved through FP.

Policy 5: Data Block Replication Policy (DBRP)

In data block replacement policy, a data block is replicated after placement of data block at i^{th} server to avoid server failure as follows:

Step 1: Extract trailer, DIPA, DP from data packet as prepared by PPF (refer Policy 1) by i^{th}

server

Step 2: Add RIPA and RP adjacent to data block

Step 3: Add trailer and prepare a modified data packet

Step 4: Send data packet to replication servers RIPA using DMP (refer Policy 2)

Packet is prepared in DBRP as shown in Figure 10.

Figure 10 Packet structure in DBRP

Field Size	2 Bytes	2 Bytes	1 Bytes	4 Bytes	2 Bytes	2 Bytes	1439 Bytes	2 Bytes	4 Bytes	2 Bytes
Field Name	H	PKT_ID	State_Type	SIPA	SP	Data_Size	DATA	RP	RIPA	T

Figure 11 Data replication control block (DRCB)

PKT_ID
Pointer to Source Port (SP)
Pointer to Replication Port (RP)
Pointer to Source Internet Protocol Address (SIPA)
Pointer to Replication Internet Protocol Address(RIPA)
State Type
Pointer to Data Payload(DATA)
Data_Size

A data replication control block (DRCB) (refer Figure 11) is used to keep track all information related to data replication policy. DPCB consists of eight fields as follows:

PKT_ID – Packet sequence number used within a packet;

Pointer to Source Port (SP) – Points to initial position of source port number in packet;

Pointer to Destination Port (RP) – Points to initial position of replication port number in packet;

Pointer to Source Internet Protocol Address (SIPA) – Points to initial position of source internet protocol address in packet;

Pointer to Replication Internet Protocol Address (RIPA) – Points to initial position of replication internet protocol in packet;

State Type – Present location (status) of packet such as source(S), destination (D) and transient (T);

Pointer to Data Payload (DATA) – Points to initial position of data payload in packet;

Data_Size – Store information about data payload size;

In DBRP, replication internet protocol address (RIPA) and replication port number (RP) are equivalent to destination internet protocol address (DIPA) and destination port number (DP) respectively as used in PFP.

The size of packet is 1460 bytes (refer Figure 10) and to store one DRCB in memory 15 bytes (refer Figure 13) is needed.

Policy 6: Server Feedback Policy (SFP)

In server feedback policy (SFP), performance of a server is being measured as follows:

Step 1: Initialise REQ_NO and ACK_NO as "zero"

Step 2: Start loop

Step 3: Send data packet from source server to destination server.

Step 4: Increase REQ_NO by 1

Step 5: If ACK_TYPE=''Y'' increase ACK_NO by 1

Step 6: End loop

Step 7: Measure Hit Ratio (H) = ACK_NO / REQ_NO

Figure 12 Server feedback control block (SFCB)

Source Internet Protocol Address (SIPA)
Destination Internet Protocol Address (DIPA)
Request Number (REQ_NO)
Acknowledgement Number (ACK_NO)
Hit Ratio (H)
Acknowledgement Status (ACK_TYPE)

Server feedback policy is controlled by the Server feedback control block (SFCB) (refer Figure 12). SFCB consists of six fields as follows:

SIPA- Source Internet Protocol Address;

DIPA – Destination Internet Protocol Address;

REQ_NO – Number of packet send;

ACK_NO – Number of received acknowledgement;

H- Hit ratio;

ACK_TYPE – Acknowledgement status such as acknowledgement received (Y) and not received (N) by source;

In SFP, performance of a server is proportional to number of acknowledgement received by source. Total storage required space in memory is dependent on size of SFCB as one SFCB is used in feedback policy. SFCB needs total 16 bytes (refer Figure 13) storage space in memory.

Figure 13 Storage information (SI) guideline as per proposed framework

Field Name	**Field Size**
PKT_ID	2Bytes
Pointer to Source Internet Protocol Address (SIPA)	2Bytes
Pointer to Destination Internet Protocol Address (DIPA)	2Bytes
Pointer to Source Port (SP)	2Bytes
Pointer to Destination Port (DP)	2Bytes
Pointer to Data Payload (DATA)	2Bytes
Data_Size	2 Bytes
Pointer to Replication Internet Protocol Address(RIPA)	2 Bytes
Pointer to Replication Port (RP)	2 Bytes
Source Internet Protocol Address (SIPA)	4 Bytes
Destination Internet Protocol Address (DIPA)	4 Bytes
Source Socket Address (SSA)	4 Bytes
Destination Socket Address (DSA)	4 Bytes
Source Socket Port (SSP)	2 Bytes
Destination Socket Port (DSP)	2 Bytes
State Type	1 Byte
Acknowledgement Status (ACK_TYPE)	2 Bytes
Time (T)	4 Bytes
Request Number (REQ_NO)	2 Bytes
Acknowledgement Number (ACK_NO)	2 Bytes
Hit Ratio (H)	2 Bytes

The detailed storage information of each field used in different control block is shown in Figure 13.

2.3 Procedure

We have designed algorithms for proposed system framework as follows:

Algorithm 1: Query_Control ()

Input: Set of users U[] within network.

Output: Receive a query search Q_S from a particular user U_i and send it to interface level.

Begin

For i = 1 to len.U[]
Take Query as 'Q_S' from $U_i \in U$
Call Intr_Entry_Cntrl (U_i, Q_S)
; (refer Algorithm 2)
End For

End

Algorithm 1 is used to communicate with outside world. Information 'Q_S' has been carried from user level to interface entry controller placed at interface level. Algorithm 1 and Algorithm 2 are worked together for information transmission.

Algorithm 2: Intr_Entry_Cntrl (U_i, Q_S)

Input: Set of interface entry controllers I_EN[].

Output: One interface entry controller I_EN_j is selected from I_EN[].

Begin

For j = 1 to len.I_EN[] do
Select $I\_EN_j \in I\_EN$
End For
Call App_Entry_Cntrl (U_i, Q_S)
; (refer Algorithm 3)

End

Algorithm 2 is used to activate one interface entry controller 'I_EN_j' from available set of controllers 'I_EN ' and transmit information from user level to interface level. Algorithm 1 is deactivated and Algorithm 3 is called from Algorithm 2 for information transmission from interface level to application level using 'I_EN_j'. Algorithm 1 is dependent on Algorithm 2, and Algorithm 2 is dependent on Algorithm 3.

Algorithm 3: App_Entry_Cntrl (U_i, Q_S)

Input: Set of application entry controllers APP_EN[].

Output: One application entry controller APP_EN_m is selected from APP_EN[].

Begin

For m = 1 to len.APP_EN [] do
Select $APP\_EN_m \in APP\_EN$
End For
Call Db_Entry_Cntrl (U_i, Q_S)
; (refer Algorithm 4)

End

Algorithm 3 is used to select one application entry controller 'APP_EN_m' from set of available controllers 'APP_EN' to receive information from user level to application level. Algorithm 2 is deactivated after successful information transmission. Algorithm 4 is called from Algorithm 3 for information transmission from application level to database level using 'APP_EN_m'. Therefore, Algorithm 3 is dependent on Algorithm 4.

Algorithm 4: Db_Entry_Cntrl (U_i, Q_S)

Input: Set of database entry controllers DB_EN[].

Output: One database controller DB_EN_p and a particular database D_t are selected.

Begin

For p = 1 to len.DB_EN[] do
Select $DB\_EN_p \in DB\_EN$
End For
Get output as 'Q_R' from D_t where $D_t \in D$
Call Db_Exit_Cntrl (U_i, Q_R)
; (refer Algorithm 5)

End

Algorithm 4 is activated after receiving request from Algorithm 3 and selects one database entry controller 'DB_EN_p' from set of available controllers. In this algorithm, three tasks have been performed. First task is to transmit information from application level to database level through 'DB_EN_p'. Second task is to select specific database 'D_t', transmission of information from 'DB_EN_p' to 'D_t', and retrieval of information 'Q_R' from selected database. Third task is to transmit information from 'D_t' to database level using Algorithm 5. Algorithm 3 is deactivated after completion of first task and Algorithm 4 is dependent on Algorithm 5.

Algorithm 5: Db_Exit_Cntrl (U_i, Q_R)

Input: Set of database exit controllers DB_EX[].

Output: One database exit controller DB_EX_q is selected from DB_EX[].

Begin

For q = 1 to len.DB_EX[] do
Select $DB\_EX_q \in DB\_EX$
End For
Call App_Exit_Cntrl (U_i, Q_R)
; (refer Algorithm 6)

End

Algorithm 5 is activated after receiving request from Algorithm 4, and one database exit controller 'DB_EX_q' has been selected from set of available controllers 'DB_EX' to transmit information from 'D_t' (refer Algorithm 4) to database level. This algorithm is used to transmit information from database level to application level using'DB_EX_q' and calling of Algorithm 6. Therefore, Algorithm 5 is dependent on Algorithm 6.

Algorithm 6: App_Exit_Cntrl (U_i, Q_R)

Input: Set of application exit controllers *APP_EX*[].

Output: One application exit controller APP_EX_n is selected from *APP_EX*[].

Begin

For n = 1 to len.*APP_EX* [] do

Select $APP\_EX_n \in APP\_EX$

End For

Call Intr_Exit_Cntrl (U_i, Q_R)

; (refer Algorithm 7)

End

Algorithm 6 is used to select one application exit controller 'APP_EX_n' from set of available controllers 'APP_EX', and to transmit information from database level to application level. Algorithm 5 is deactivated after successful information transmission. Then, Algorithm 7 is called for information transmission from application level to interface level through 'APP_EX_n'. Therefore, Algorithm 6 is dependent on Algorithm 7.

Algorithm 7: Intr_Exit_Cntrl (U_i, Q_R)

Input: Set of interface exit controllers *I_EX*[].

Output: One interface exit controller I_EX_k is selected from I_*EX*[].

Begin

For k = 1 to len.*I_EX* [] do

Select $I\_EX_k \in I\_EX$

End For

Send Q_R to U_i

End

Algorithm 7 is activated when Algorithm 7 is called from Algorithm 6. Algorithm 7 is used to select one interface exit controller 'I_EX_k' from set of available controllers '*I_EX*'. The task of 'I_EX_k' is to receive information from application level to interface level and transmit information from interface level to desired user U_i.

In proposed algorithm, search query processing and query response processing always have two different paths in proposed system to avoid network congestion, minimise network delay within network. Algorithm 1 is used only to take search query from a particular user and send it to an interface. Entry controllers of each level are activated through Algorithm 2 to Algorithm 4 for processing of search query and forwarding of search query from one level to another level. Algorithm 4 is an end of search query processing and beginning of query response processing. Exit controllers of each level are activated through Algorithm 5 to Algorithm 7 for query response processing and forwarding from one level to another level. Algorithm 7 is responsible to provide query response to specified user.

2.4 Theoretical discussions

Server selection from a set of servers and allocation of search query considered as task to a selected server have great impacts on fast processing of search query and speed-up in generation of responses. Task distribution among servers enhances performance and effectiveness of overall system. Selection of a particular server from a set of servers is achieved through following mechanism:

Assume, 'm' is total number of task to be allocated within 'n' number of servers at any level.

Task_Alloc[m][n] is a task allocation matrix where $m, n \geq 1$

Initially, Task_Alloc[j][k] = 0 where $1 \leq j \leq$ m and $1 \leq k \leq n$

Allocation of j^{th} task to a particular server is achieved as follows:

$$\text{Task\_Alloc}[j][j \bmod n] = 1 \quad (1)$$

From equation (1), task allocation matrix is constructed in which each row has only one non-zero value. After 'n' numbers of task allocations, the resultant matrix is as follows:

$$\text{Task\_Alloc[m][n]} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \cdots & 0 \end{bmatrix}$$

Therefore, total number of tasks is allocated to k^{th} server (load balance factor of server).

$$\therefore \text{Total\_Number\_of\_Task} = \sum_{j=1}^{m} \text{Task\_Alloc}[j][k]$$

Each time, system needs to calculate load balance factor of each servers after allocation of each task to a particular server, which is very time consuming. Collection of 'counter' variables is introduced in our proposed framework to avoid repetitive calculation of load balance factor of each server. Number of 'counter' variables is equal to number of available servers i.e.,

$$\text{Number\_of\_Servers} = \text{ServLoad}[n] \text{ where } n \geq 1 \quad (2)$$

Motivation of Theorem 1: In proposed system framework, several servers are used at each level. Only one server is selected for allocation of specific search query as task. The aim of task assignment is to optimise utilisation of resources, minimise response time and uniform distribution of load among all nodes. Theorem 1 has been designed for uniform load distribution among all available servers.

Theorem 1: Proposed system framework is balanced if and only if the difference in load balance factor of any two servers within the framework is either '0' or '1'.

Proof: Assume, at any level, there are 'n' numbers of servers.

Serv_Ptr is a pointer which points to most recently used server for allocation of specific search query as task.

ServLoad[i] is used to represent load of i^{th} server (refer equation (2)).

Initially, ServLoad[i] = 0; where1 $\leq i \leq n$
(refer equation (2))

*Serv_Ptr = 0; where 1$\leq$ *Serv_Ptr $\leq n$

Task allocation to a server is done by two steps as follows:

Step 1: Modification of Serv_Ptr is accomplished for selection of particular server from 'n' numbers of servers.

Step 2: Task allocation is accomplished for the selected server incrementing load balance factor of that particular server by one.

Modification of Serv_Ptr is accomplished with the following rules:

Rule 1: If *Serv_Ptr = n then,

*Serv_Ptr = 1

Rule 2: If *Serv_Ptr $\neq n$ then,

*Serv_Ptr = *Serv_Ptr + 1

Task allocation (T_j) is accomplished to a particular server (ServLoad[i]) determined as follows:

i = *Serv_Ptr;

ServLoad[i] = ServLoad[i] + 1;

At any time instance, after allocation of task at ServLoad[i], following relations are always true:

ServLoad[i] = ServLoad[i – 1]; where $i > 1$ (3)

ServLoad[i] = ServLoad[i + 1] + 1; where $i < n$ (4)

From equation (3),

|ServLoad[i] - ServLoad[i – 1] | = 0

From equation (4),

| ServLoad[i] - ServLoad[i + 1] | = 1

Apart from above mention situations (refer equations (3) and (4)), 1st server is to be selected after task allocation to n^{th} server (refer Rule 2); and in such situation, following relation is always true:

| ServLoad[1]-ServLoad[n]| = 0

Therefore, for all cases, difference in load balance factor between any two servers within the framework is either '0' or '1'.

Hence, it is proved that proposed system framework is balanced if and only if the difference in load balance factor of any two servers within the frame work is either '0' or '1'.

(End of Proof)

Motivation of Theorem 2: Heterogeneous servers are worked together in proposed system framework. Servers are heterogeneous in terms of domain specific, configuration specific etc.. Task is allocated to a specific server from a set of heterogeneous servers in such a way that maximum throughput of system is achieved through load balancing among servers. Theorem 2 has been designed for uniform load distribution within homogenous servers among the whole set of heterogeneous servers.

Theorem 2: If proposed system framework consists of heterogeneous servers, then the system framework is considered to be balanced if and only if the difference in load balance factor of any two homogeneous servers within the framework is '0' or '1'.

Proof: Assume, heterogeneous servers are classified with 'x' numbers of categories.

Serv_Type[t] is used to represent numbers of servers with category 't' where $1 \leq t \leq x$.

'n' is used to represent total numbers of servers in each category.

ServLoad[t][i] is used to represent load of i^{th} server with category 't' where $1 \leq t \leq x$ and $1 \leq i \leq n$.

Serv_Ptr[x] is used as an array of pointers where task of each pointer is to point to most recently used server of particular category for allocation of task.

Initially,

n = ServType[t]; where $1 \leq t \leq x$.

ServLoad[t][i] = 0; where1 $\leq t \leq x$ and $1 \leq i \leq n$

*Serv_Ptr[t] = 0; where $1 \leq t \leq x$

Task allocation to a server is done by three steps as follows:

Step 1: Identification of category of a server from 'x' number of categories.

Step 2: Modification of Serv_Ptr is accomplished for selection of particular server from 'n' numbers of servers of particular category.

Step 3: Task allocation is accomplished for the selected server incrementing load balance factor of that particular server by one.

Modification of Serv_Ptrof each category of servers is accomplished with the following rules:

Rule 1: If *Serv_Ptr[t] = n then,

*Serv_Ptr[t] = 1

Rule 2: If *Serv_Ptr[t] $\neq n$ then,

*Serv_Ptr[t] = *Serv_Ptr[t] + 1

Task allocation (T_j) is accomplished to a particular server (ServLoad[t][i]) determined as follows:

i = *Serv_Ptr[t];

ServLoad[t][i] = ServLoad[t][i] + 1;

At any time instance, after allocation of task at ServLoad[t][i], following relations are always true:

ServLoad[t][i] = ServLoad[t][i-1]; where $i > 1$ (5)

ServLoad[t][i] = ServLoad[t][i+1] + 1; where $i < n$ (6)

From equation (5),

$$|\text{ServLoad}[t][i] - \text{ServLoad}[t][i\text{-}1]| = 0$$

From equation (6),

$$\text{ServLoad}[t][i] - \text{ServLoad}[t][i+1]| = 1$$

Apart from above mention situations (refer equations (5) and (6)), 1st server is to be selected after task allocation to nth server of a particular category (refer Rule 2); and in such situation, following relation is always true:

$$|\text{ServLoad}[t][1] - \text{ServLoad}[t][n]| = 0$$

Therefore, for all cases, difference in load balance factor between any two servers with t^{th} category within the framework is either '0' or '1'.

Hence, it is proved that if proposed system framework consists of heterogeneous servers, then the system framework is considered to be balanced if and only if the difference in load balance factor of any two homogeneous servers within the framework is '0' or '1'.

(End of Proof)

Consider, proposed system is worked in balanced mode with 'n' number of servers and maximum load balance factor of a server from set of servers is 'p' (refer Theorem 1) where $n, p \geq 1$. Then, load balance factor of 'n' number of servers (refer equation (2)) is constructed as follows:

$$\text{ServLoad}[n] = [p, p \ldots p\text{–}1, \ldots p\text{–}1]$$

In a particular instance, if 'x' number of tasks is released from i^{th} server, then load balance factor of i^{th} server is reduced with respect to other servers as follows:

$$\text{ServLoad}[i] = \text{ServLoad}[i] - x \text{ where } x \geq 1 \text{ and } 1 \leq i \leq n \quad (7)$$

'ServRelease[n]' is used to keep information about released task(s) of each server. Therefore, total number of released tasks by i^{th} server is as follows:

$$\text{ServRelease}[i] = \text{ServRelease}[i] + x \text{ where } 1 \leq i \leq n \quad (8)$$

Then, i^{th} server is selected for next task allocation to minimise differences in load balance factors from other servers.

From equation (8), a new task T_j is allocated to i^{th} server among all servers as load balance factor of i^{th} server is minimum with respect to other available servers. As a result, load balance factor of i^{th} server is increased by one and number of released tasks of i^{th} server is decreased by one.

Motivation of Theorem 3: Proposed system is always worked in a balanced mode in terms of allocation of new tasks among servers. Number of tasks is released from allotted servers simultaneously based on completion of respective task executions. A server is allocated for a new task in such a way that loads among servers is uniformly distributed after release of tasks from servers. Theorem 3 has been designed for release of tasks among servers.

Theorem 3: Proposed system framework is balanced if and only if difference in load balance factor of any two servers within the framework is minimum with respect to the task release.

Proof: Assume, at any level, there are 'n' numbers of servers.

ServLoad[i] is used to represent load of i^{th} server (refer Theorem 1).

Release_Ptr is a pointer which points to server with maximum numbers of released tasks.

ServRelease[i] is used to represent number of released task of i^{th} server.

$$\text{From equation (8), ServRelease}[i] = 0; \text{ where} 1 \leq i \leq n \quad (9)$$

Initially, *Release_Ptr = 0;

Consider, p and q numbers of tasks have been released from k^{th}server and j^{th} server respectively.

From equation (7) load balance factor of k^{th} and j^{th} servers are reduced as follows:

$$\text{ServLoad}[k] = \text{ServLoad}[k] - p;$$

$$\text{ServLoad}[j] = \text{ServLoad}[j] - q;$$

From equation (8), numbers of released tasks of k^{th} and j^{th} servers are as follows:

$$\text{ServRelease}[k] = \text{ServRelease}[k] + p; \quad (10)$$

$$\text{ServRelease}[j] = \text{ServRelease}[j] + q; \quad (11)$$

From equation (9), following relations are always true:

$$\text{ServRelease}[i] = 0; \text{ where } i \neq k \text{ and } i \neq j$$

From equation (10) and equation (11), following relations are always true:

$$\text{ServRelease}[k] \geq 0;$$

$$\text{ServRelease}[j] \geq 0;$$

In such situation, a server which has maximum number of released tasks than other servers is selected for allocation of new task 'T_j'. The selection of a server from set of servers is accomplished with following ways:

$$\text{Maximum}(\text{ServRelease}[k], \text{ServRelease}[j] = \begin{cases} k; if\ ServRelease[\text{k}] \geq SerrvRelease[j] \\ j; if\ ServRelease[\text{j}] > SerrvRelease[k] \end{cases}$$

$$*\text{Release\_Ptr} = \text{Maximum}(\text{ServRelease}[\text{k}], \text{ServRelease}[\text{j}]);$$

Allocation of task T_j to a server pointed by Release_Ptr is accomplished as follows:

$$i = *\text{Release\_Ptr};$$

$$\text{ServLoad}[i] = \text{ServLoad}[i] + 1;$$

$$\text{ServRelease}[i] = \text{ServRelease}[i] - 1;$$

Therefore, Release_Ptr always points to a server which has maximum numbers of released tasks to minimise load balance factor among available servers and Serv_Ptr (refer Theorem 1) is not modified until *Release_Ptr is equal to zero; i.e., not a single task is released by any server.

Hence, it is proved that proposed system framework is balanced if and only if difference in load balance factor of any two servers within the framework is minimum with respect to the task release.

(End of Proof)

2.5 Benefits of proposed framework

Proposed framework provides following benefits:

1) Search query and query response are carried using two different paths to avoid congestion over network and to provide fast transmission of data to achieve minimum delay in response time (refer Algorithm 1 to Algorithm 6).
2) Proposed framework is responsible for packet formation of data and transmission of data packets from source to destination designing PFP (refer Policy 1), DMP (refer Policy 2), and DMCB (refer Policy 3) with corresponding monitoring control blocks.
3) Failure in data transmission from source to destination is supervised by designing FP (refer Policy 4).
4) Replication of data is accomplished with designing DBRP (refer Policy 5) to provide availability of server due to cause of failure in desired server.
5) Efficiency of servers is being measured designing SFP (refer Policy 6)
6) In each level, high performance of servers is maintained by task distribution among servers restricting difference in load balance factor between any two servers within 0 or 1 in network (refer Theorem 1).
7) Proposed framework is capable to work with heterogeneous distributed servers' environment with controlled load distribution among any two homogeneous servers at each level in network (refer Theorem 2).
8) Minimum load difference is maintained during allocation of new task for uniform tasks distribution among servers after releasing finished tasks from several servers within proposed framework (refer Theorem 3).
9) Minimum delay in response time of search queries is achieved through level based concurrent execution of search queries in servers placed at different levels (refer Algorithm 1 to Algorithm 6).
10) Simultaneous execution of search queries is achieved using level based proposed framework (refer Sub-Section 2.1, Algorithm 1 to Algorithm 6).
11) Overall performance of system is dependent on number of levels, number of entry controllers and number of exit controllers (refer Sub-Section 3.2.4).
12) System performance is not dependent on packet size, as separate PFP (refer Policy 1) is designed for formation of data packets (refer Sub-Section 3.2.4).

3 Experimental discussions

In proposed system framework, user search queries are carried out through three levels such as interface level, application level, database level (refer Figure 1). Each level considered as layer is managed by several policies (refer Sub-Section 2.2). Utilisation of policies in different layers of proposed framework is shown in Table 1 based on practical implementation.

Table 1 Layer based implementations of different proposed policies

Layer	*Policy Name(Policy No)*
Interface Layer	PFP (Policy 1), DMP(Policy 2), SFP (Policy 6)
Application Layer	DMP(Policy 2)
Database Layer	DMP (Policy 2), DBPP (Policy 3), FP (Policy 4), DBRP (Policy 5)

3.1 Experimental setup

The system configuration has been used during experimentation as shown in Table 2.

Table 2 System configuration

Primary memory (RAM)	*Processor*	*Processor Speed*	*Hard Disk Drive (HDD)*	*Number of Systems*
2GB	Intel® Core™ 2 Duo	2.93GHz	320GB	23
4GB	Intel® Core™ i3	3.60GHz	1TB	8

3.2 Performance analysis

We have studied experimental performance analysis based on 31 server site machines. Maximum 3000 users have been considered for server stress calculation in real time.

3.2.1 Observation over server performance

Table 3 provides performance of servers using JMeter (Kaur et al., 2016) running in existing framework and proposed framework with respect to number of samples, average response time in milliseconds, 90% line and throughput. "Number of virtual users per request", "average time taken by all the samples to execute specific label", "90% of the samples not beyond more than obtained time" and "amount of data downloaded from server during the performance test execution" represent number of samples, average response time in milliseconds, 90% line and throughput respectively. It is observed that average response time of server, value of 90% line, and throughput of server (KB/sec) during the performance testing with equal number of users in specified label (HTTP request) are at par whether we use proposed framework. Hence, it can be concluded that proposed framework is light weight.

Table 3 Performance results for different number of users in existing framework and proposed framework using JMeter

Number of users	*Label*	*JMeter running with existing framework*				*JMeter running with proposed framework*			
		Number of samples	*Average response time in milliseconds (ms)*	*90% Line*	*KB/sec*	*Number of samples*	*Average response time in milliseconds (ms)*	*90% Line*	*KB/sec*
100	HTTP Request	988	960	1023	201.35	1021	968	1025	200.29
200		1353	938	1022	399.28	1113	950	1069	383.36
300		1496	928	1084	580.97	1563	935	1064	579.14
400		1807	910	1025	778.86	1455	915	1039	741.77
500		1901	890	1026	964.03	1757	902	1035	923.08
1000		1893	840	1014	1644.87	1745	886	1089	1495.47
1500		1413	694	1051	1058.50	1382	569	1005	1222.30
2000		1875	691	1059	1508.98	1793	575	1014	1488.62
2500		2350	665	1064	1913.10	2135	587	1043	1831.57
3000		2716	704	1072	2157.83	2666	688	1084	1988.34

Comparative analysis has been performed based on existing framework and proposed framework with respect to maximum response time and minimum response time, median response time and throughput of servers using JMeter.

Longest time and shortest time taken among chosen samples for a specific label (HTTP) are represented as maximum and minimum response time of servers respectively. Figure 14 ensures that response time of servers in proposed framework is at par compared to response time of server with existing framework.

Median response time of servers indicates that response time of 50% of the samples is not more than median response time. Median response time of servers in proposed framework is at par compared to existing framework as shown in Figure 15.

Throughput represents number of request processed. Throughput is measured using JMeter running in existing framework and proposed framework as shown in Figure 16. From Figure 16, it has been observed that proposed system framework maintains resemblance in overall performance of servers with increasing number of users. Hence, proposed framework is light weight to servers having less stress.

Figure 14 Maximum and minimum response time of existing framework and proposed framework using JMeter

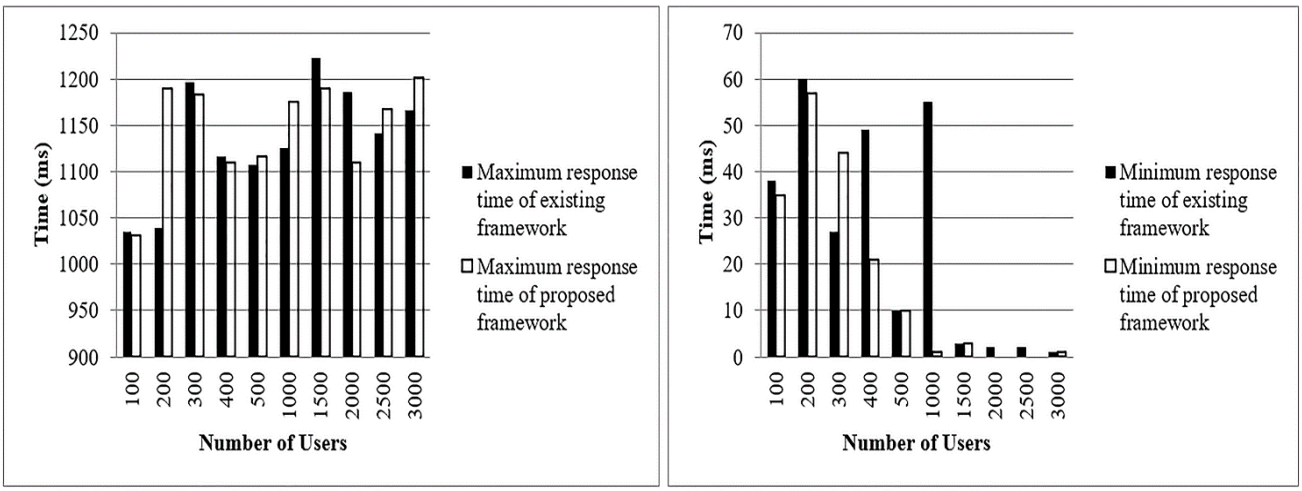


Figure 15 Median response time of existing framework and proposed framework using JMeter

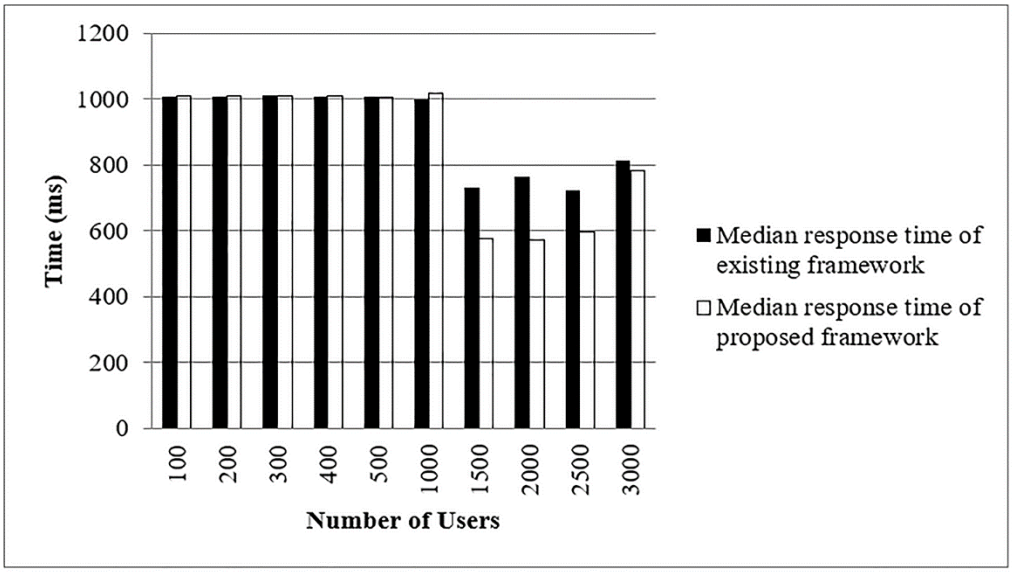
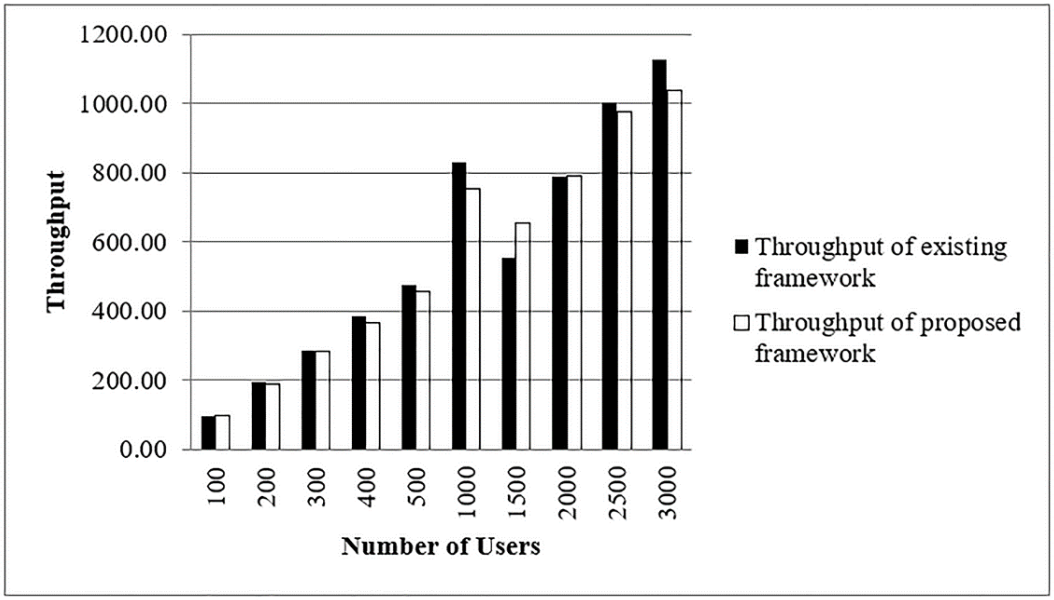


Figure 16 Throughput of servers in existing framework and proposed framework using JMeter



3.2.2 Cost analysis

The cost of proposed framework is obtained by combining cost of each level (refer Figure 1) with network delay. User level, interface level, application level and database level costs are presented as UserLevel_Cost, IntrLevel_Cost, AppLevel_Cost, DatabaseLevel_Cost respectively and network delay is represented as ∂_t. Total cost of proposed system framework is determined with the following equation:

$$\text{Total\_Cost} = \text{UserLevel\_Cost} + \text{IntrLevel\_Cost} + \text{AppLevel\_Cost} + \text{DatabaseLevel\_Cost} + \partial_t$$

User level cost is only dependent on user input task and output received task. Therefore, user level cost is determined with following equation:

$$\text{UserLevel\_Cost} = \text{Input\_Cost} + \text{Output\_Cost} = 2 \text{ units}$$

Interface level cost is the collective cost of entry controller and exit controller (refer Figure 1) placed at interface level. Therefore, interface level cost is determined with following equation:

$$\text{IntrLevel\_Cost} = \text{IntrEnCntroller\_Cost} + \text{IntrExCntroller\_Cost}$$

Entry controller cost at interface level is determined with following equation:

$$\text{IntrEnCntroller\_Cost} = \text{PFP\_Cost} + \text{SFP\_Cost} + \text{ServerSelection\_Cost} + \text{Synchronisation\_Cost} + \text{DMP\_Cost} + \partial_t = 5 \text{ units [As, } \partial_t \approx 0]$$

Exit controller cost at interface level is determined with following equation:

IntrExCntroller_Cost = Synchronisation_Cost + DMP_Cost + ∂_t = 2 units [As, $\partial_t \approx 0$]

Therefore, IntrLevel_Cost = (5 + 2) units = 7 units

Similarly, application level cost is determined with following equation:

AppLevel_Cost = AppEnCntroller_Cost + AppExCntroller_Cost

Entry controller cost at application level is determined with following equation:

AppEnCntroller_Cost = ServerSelection + Synchronisation_Cost + DMP_Cost + ∂_t = 3 units [As, $\partial_t \approx 0$]

Exit controller cost at application level is determined with following equation:

AppExCntroller_Cost = ServerSelection + Synchronisation_Cost + DMP_Cost + ∂_t = 3 units [As, $\partial_t \approx 0$]

Therefore, AppLevel_Cost = (3 + 3) units = 6 units

Similarly, database level cost is determined with following equation in which DatabaseServer_Cost is used to determine cost to retrieve query information from specific database:

DatabaseLevel_Cost = DbEnCntroller_Cost + DatabaseServer_Cost + DbExCntroller_Cost

Entry controller cost at database level is determined with following equation:

DbEnCntroller_Cost = ServerSelection + Synchronisation_Cost + DMP_Cost + ∂_t = 3 units [As, $\partial_t \approx 0$]

Information retrieval cost at database level is determined with following equation:

DatabaseServer_Cost = DBPP + DBRP + FP + ∂_t = 3 units [As, $\partial_t \approx 0$]

Exit controller cost at database level is determined with following equation:

DbExCntroller_Cost = PFP + ServerSelection + Synchronisation_Cost + DMP_Cost + ∂_t = 4 units [As, $\partial_t \approx 0$]

Therefore, DatabaseLevel_Cost = (3 + 3 + 4) units = 10 units

Total_Cost = (2 + 7 + 6 + 10) units [As, $\partial_t \approx 0$]

= 25 units

Hence, total '25 units' cost is required by the proposed framework to perform a query searching within network, and is not a fixed value. The value ranges in milliseconds. Figure 14 shows that 10 milliseconds and1117 milliseconds are minimum and maximum response time for 500 users. Therefore, in case of minimum response time, cost of one unit is equivalent of 0.4 milliseconds (i.e. 10/25), whereas, cost of one unit for maximum response time is 44.68 milliseconds (i.e. 1111/25).

3.2.3 Load analysis

Load analysis of proposed system has been accomplished with respect to percentage of CPU usage and percentage of maximum clock frequency of servers. Work load of CPU usage percentage of servers in existing framework and proposed framework has been represented at Figure 17. CPU usage percentage has been increased during maximum utilisation of servers. CPU usage percentage of proposed framework is lies between 55% and 66%.

Utilisation of servers is measured with measurement of maximum clock frequency percentage usage of servers. Clock frequency used by proposed framework is already specified in Table 2. Figure 18 shows clock frequency used in existing framework and proposed framework. Maximum clock frequency percentage of proposed model is lies 80% to 90%.

Figure 17 CPU usage percentages of servers in existing framework and proposed framework

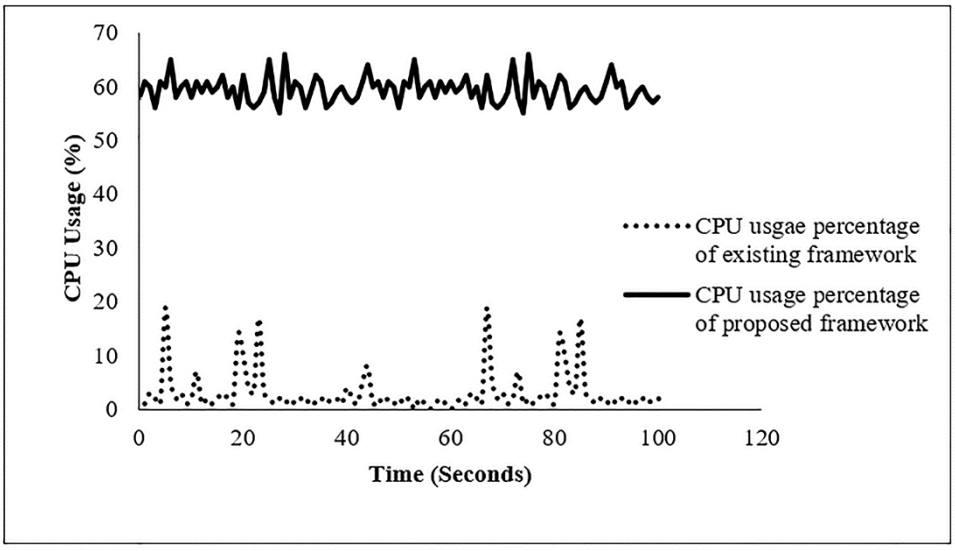
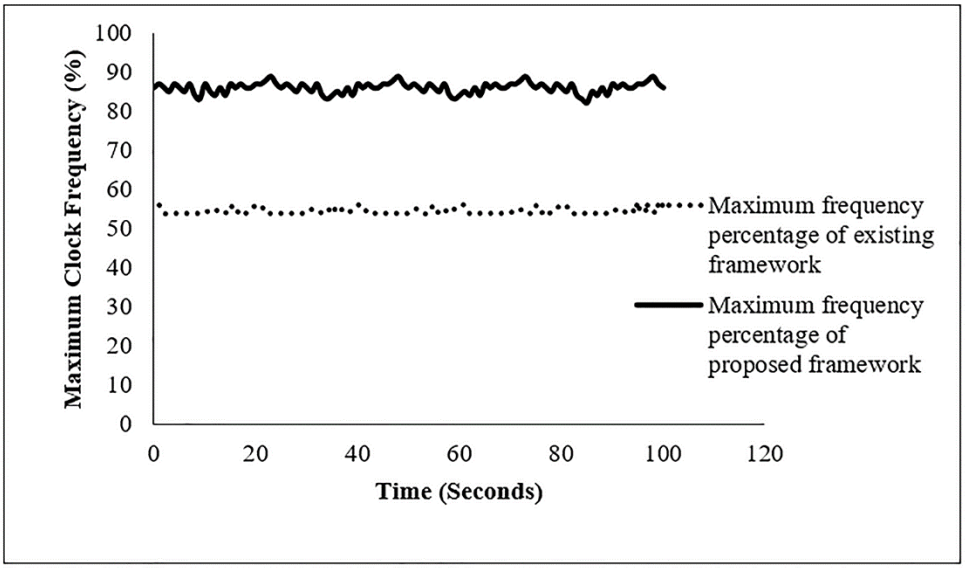


Figure 18 Maximum clock frequency percentage of servers in existing framework and proposed framework



3.2.4 Time analysis

Proposed framework has 'L' number of levels and each level consists of 'M' number of entry controllers, and 'N' number of exit controllers. Maximum 't_M' time is required to select one entry controller and maximum 't_N' time is required to select one exit controller in a particular level. Therefore, in each level, total ($t_M + t_N$) time is required to allocate one entry controller and one exit controller. Total [$L * (t_M + t_N)$] time is required to proceed with a search query. Therefore, time complexity of proposed framework is $O(L * (t_M + t_N))$.

In proposed approach, packet size is fixed (1460 bytes) in each level whether we consider PFP (refer Figure 3) or DBRP (refer Figure 10) using storage information guideline for proposed framework (refer Figure 13) in various policy design. In run time, time complexity is always invariant with respect to packet size, since packet size is fixed. Thus, time complexity only depends on L, M, & N.

3.3 Comparison results

3.3.1 Features based comparisons

Several features are being identified in proposed framework and compared to existing frameworks as shown in Table 4. Following features have been considered as follows:

- Crisis management
- Synchronisation
- Distributed Environment
- Strategic Control
- Real Time Analysis
- Response Time
- Working Environment
- Load Management
- Network Status Consideration
- Application Area
- Risk Management

Crisis management deals with availability of servers during processing of search query. Uniform distribution of tasks among servers increases maximum possibilities of servers' availability. In each level, uniform distribution of tasks is achieved through load balancing mechanisms in proposed framework (refer Theorem 1, Theorem 2 and Theorem3). Therefore, proposed framework is responsible for crisis management, whereas existing frameworks are unable to provide any such mechanism for crisis management.

Synchronisation among two servers considering two consecutive levels is established using DMP (refer Policy 2) for data transmission. Thus, proposed framework is responsible for providing synchronisation among servers. In Table 4, it has been observed that except the first existing framework, synchronisation mechanism is not specified for all remaining existing frameworks.

Proposed framework works in distributed environment like other specified existing frameworks where servers are distributed in network (refer Sub-Section 1.3) as shown in Table 4.

Communication establishment and data transmission among servers are accomplished by designing policies such as PFP, DMP, DBPP, FP, DBRP, and SFP (refer Sub-Section 2.2). Distribution of tasks among servers is carried with load balancing approach (refer Sub-Section 2.4) to maintain consistency in performance of servers. Thus, proposed framework has policy based load balancing strategic control on servers. It has been observed that specified existing frameworks have their own specified strategic control as referred in Table 4.

Real time analysis of proposed framework is performed using JMeter simulation software (refer Sub-Section 3.2). Therefore, real time analysis has been accomplished (refer Table 4).

Response time is being measured based on calculation of time requirement to process a search query where response time is dependent on number of levels, number of entry controllers and number of exit controllers (refer Sub-Section 3.2.4). Therefore, in worst case scenario, time complexity of proposed framework is O (L*(M + N)), where L, M, and N are number of levels, number of entry controllers, and number of exit controllers respectively.

Servers with different configurations and functionalities are worked together in each levels of framework. Heterogeneous working environment of servers is controlled using load balancing approach (refer Theorem 3) for proposed framework. From Table 4, it has been observed that client-side server selection algorithms (refer Dykes et al., 2000) work in heterogeneous environment.

Load management is accomplished through task distribution among servers using load balancing approaches (refer Theorem 1, Theorem 2, and Theorem 3). Thus, load management is accomplished as shown in Table 4.

Network status has been considered in cost analysis measurement of proposed framework (refer Sub-Section 3.2.2). From Table 4, it has been observed that network status is unknown for dynamic replication management strategy, whereas existing frameworks (refer Dykes et al., 2000, Bakiras, 2005, and Wang et al., 2009) have considered network status.

Proposed framework is applicable for web based query processing among several application areas like content distributed network (CDN), geographic information system (GIS), and web based query processing (refer Section 2, Sub-Section 3.2). Application areas of specific existing frameworks are mentioned in Table 4.

Risk management is required for fetching information about failure of servers within proposed framework. FP has been designed for risk management in framework (refer Policy 4). Risk management is unknown for most of the existing frameworks as shown in Table 4.

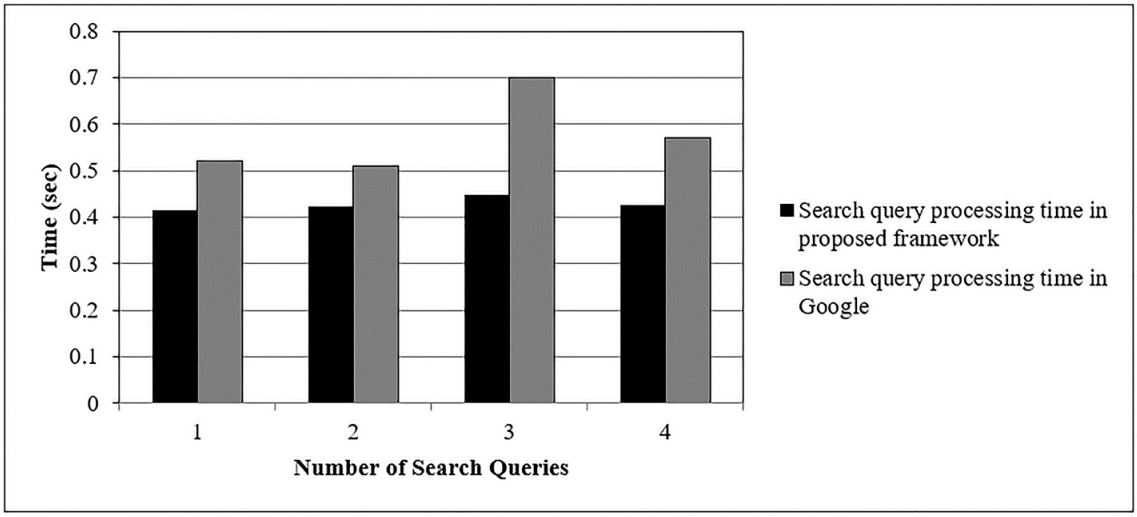
3.3.2 Comparative study on experimental observation

Processing time of four search queries as a sample study is compared between proposed framework and existing framework such as Google as shown in Figure 19. Figure 19, it has been observed that processing time of proposed framework is better than existing framework for all queries. It has been also observed that processing time of proposed framework is not varied widely. Therefore, proposed framework maintains consistency performance in processing time of search queries.

Table 4 Comparative analysis between proposed framework and existing frameworks

Features	*Client-side server selection algorithms (refer Dykes et al., 2000)*	*Approximate server selection algorithms (refer Bakiras, 2005)*	*Dynamic replication management strategy (refer Pan et al., 2018)*	*Dynamic Data Migration Policies (refer Wang et al.,2009)*	*Proposed Framework*
Crisis Management	No	No	No	No	Yes as discussed in Theorem 1, Theorem 2, Theorem 3 (refer Sub-section 2.4)
Synchronisation	Yes	Not available	Not available	Not available	Yes
Distributed Environment	Applicable	Applicable	Applicable	Applicable	Applicable
Strategic Control	Set of algorithms	Gradient projection method	Fixed method	DDMC & DDMD	Policy based load balancing approach
Real Time Analysis	Yes	Yes	Yes	Yes	Yes
Response Time	Unknown	*O* (LMN2) where *L*, *M*, *N* are number of iteration, hosted objects, servers respectively	Unknown	Unknown	O (L*(M + N)) where L, N and M are number of levels, entry controllers and exit controllers respectively
Working Environment	Heterogeneous	Homogeneous	Homogeneous	Homogeneous structure and heterogeneous functionality of servers	Heterogeneous
Load Management	Not considered	Yes	Yes	Yes	Yes
Network Status Consideration	Considered	Considered	Unknown	Considered	Considered
Application Area	Not specified	Content Distributed Network (CDN)	Geographic information system (GIS) based query processing	Web based query processing	Web based query processing
Risk Management	Unknown	Unknown	Yes	Unknown	Yes through FP (refer Policy 4)

Figure 19 Comparison based on processing time of multiple search queries between proposed framework and existing framework



3.3.3 Comparative study on efficiency measurement

Proposed framework has three levels such as interface level, application level, and database level. Each level has set of entry controllers and set of exit controllers. A controller is busy if and only if a task is assigned to the controller. After completion of task, controller assigns the task to its next level of controllers. Table 5 shows activities of controllers placed at different levels in different time spans of proposed framework. Tasks have been considered as P1, P2, P3, P4, P5, and P6, whereas time spans are T1, T2, T3, T4, T5 and T6. From Table 5, it has been observed that in initial time span T1, interface level entry controller is busy with task P1. Remaining controllers are available for assigning tasks. In next time span (T2), P1 is assigned to application level controller. Therefore, interface level controller becomes available for new task assignment. New task P2 is assigned to the available controller. Similarly, all tasks are assigned and further processed to the next level of controllers. From Table 5, it has been shown that task P1 is accomplished in time span T6. Therefore, task P2 finishes in next time span (T7). P2 is to be processed by interface level exit controller within T7 time span. Similarly, P3, P4, P5, and P6 are executed within time span T8, T9, T10 and T11 respectively. Hence, after completion of first task, remaining tasks would be finished sequentially in next consecutive time spans.

Let, we have to process 'n' number of tasks and each time span represents one unit time.

In proposed framework, 1st task is completed after 6 unit time span.

To complete 'n' number of task, required time = 6 + (n – 1) time units = (n + 5) time units where '6' represent time units required to complete 1st task and remaining (n – 1) tasks requires (n – 1) time units.

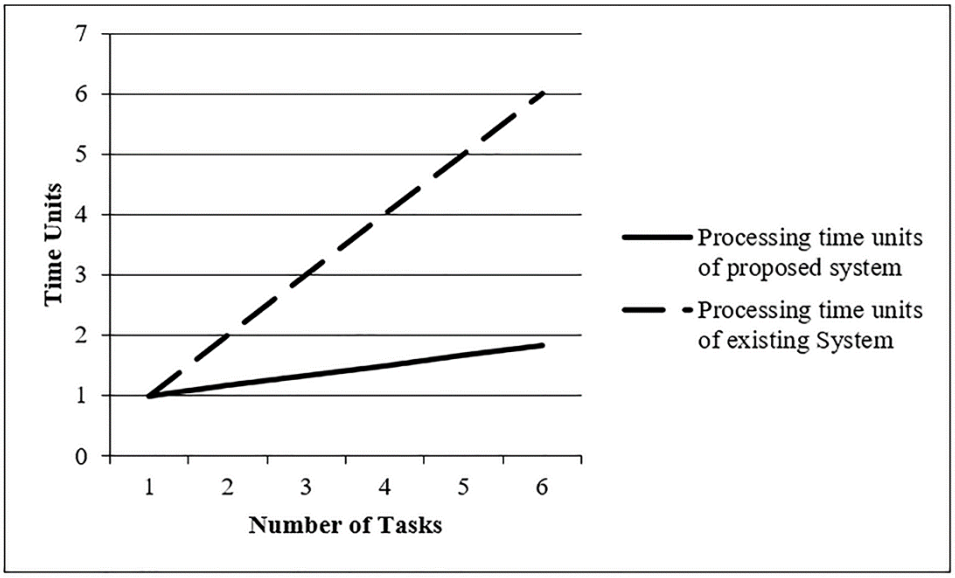
Now, consider existing framework requires same unit time i.e. 6 time units to complete 1st task. To complete 'n' number of tasks, existing framework requires 6*n time units.

Figure 20 depicts that initially, proposed framework and existing framework require same time to complete 1st task. And remaining tasks require more time in existing framework than proposed framework. Therefore, efficiency of proposed framework is more than existing framework.

Table 5 Efficiency measurement using time requirement

Time Span	*Interface level entry controller*	*Application level entry controller*	*Database level entry controller*	*Database level exit controller*	*Application level exit controller*	*Interface level exit controller*
T1	P1	–	–	–	–	–
T2	P2	P1	–	–	–	–
T3	P3	P2	P1	–	–	–
T4	P4	P3	P2	P1	–	–
T5	P5	P4	P3	P2	P1	–
T6	P6	P5	P4	P3	P2	P1

Figure 20 Comparison based on time requirement of proposed framework and existing framework



4 Conclusion

In this paper, a light-weight policy based controller framework has been designed utilising load balance factor for information search through distributed database servers. Information is transmitted from one layer to another using entry controller and exit controller placed at different levels and better search query processing time is achieved than existing query processing system. Communication establishment and transmission of information from one level to another level have been accomplished through designing of several policies with risk management. Controllers are reused for further new searching tasks after successful transmission of information to next level. Network congestion and query response time are reduced through two way transmission of data and load management among servers. Performances of servers are maintained at par through efficient load management schemes.

Acknowledgement

This research work is funded by Computer Innovative Research Society, West Bengal, India. Award number is "2018/CIRS/R&D/2018-12-21/PHSUCF".

References

Acharjya, D.P. and Ahmed, K.. (2016) 'A Survey on Big Data Analytics: Challenges, Open Research Issues and Tools', *International Journal of Advanced Computer Science and Applications*, Vol. 7, No. 2, pp.511–518.

Adamu, F.B., Habbal, A., Hassan, S., Cottrell, R.L., White, B. and Abdullahi, I. (2015) 'A Survey On Big Data Indexing Strategies', *4th International Conference on Internet Applications*, 2015.

Ahmadi, R., Cami, B.R. and Hassanpour, H. (2012) 'Automatic Data Migration between Two Databases with Different Structure', *International Journal of Applied Information Systems*, Vol. 3, No. 3, pp.23–28.

Bakiras, S. (2005) 'Approximate server selection algorithms in content distribution networks', *IEEE International Conference on Communications*, pp.1490–1494.

Brin, S. and Page, L. (1998) 'The Anatomy of a Large-Scale Hypertextual Web Search Engine', *Computer Network and ISDN Systems*, Vol. 30, pp.107–117.

Dhanalakshmi, S., Prabakaran, T. and Kishore, K. (2017) 'Content Delivery Networks – A Survey', *International Journals of Advanced Research in Computer Science and Software Engineering*, Vol. 7, No. 7, pp.228–230.

Dykes, S.G., Robbins, K.A. and Jeffery, C.L. (2000) 'An empirical evaluation of client-side server selection algorithms', *Proceeding IEEE INFOCOM 2000*, Vol. 3, pp.1361–1370.

Elaraby, M.E., Sakre, M.M., Rashad, M.Z. and Nomir, O. (2012) 'Dynamic and Distributed Indexing Architecture in Search Engine using Grid Computing', *International Journal of Computer Applications*, Vol. 55, No. 5, pp.34–42.

Eren, B., Karabulut, E.C., Alptekin, S.E. and Alptekin, G.I. (2015) 'A K-Means Algorithm Application on Big Data', *Proceedings of the World Congress on Engineering and Computer Science*, Vol. 2, pp.814–818.

Gani, A., Siddiqa, A., Shamshirband, S. and Hanum, F. (2016) 'A survey on indexing techniques for big data: taxonomy and performance evaluation', *Article in Knowledge and Information Systems*, Vol. 46, No. 2, pp.241–284.

Gupta, P. and Sharma, A.K. (2010) 'Context based Indexing in Search Engines using Ontology', *International Journal of Computer Applications*, Vol.1, No. 14, pp.49–52.

Jain, M. and Verma, C. (2014) 'Adapting k-means for Clustering in Big Data', *International Journal of Computer Applications*, Vol. 101, No.1, pp.19–24.

Kaur, N. and Bahl, K. (2016) 'Performance Testing Of Institute Website Using Jmeter', *International Journal of Innovative Science, Engineering & Technology*, Vol. 3, No. 4, pp.534–537.

Keller, A., Borkmann, D., Neuhaus, S. and Happe, M. (2014) 'Self-Awareness in computer networks', *International Journal of Reconfigurable Computing*, Vol. 2014, No. 10, pp.1–16.

Kundu, A., Dutta, R., Dattagupta, R. and Mukhopadhyay, D. (2009) 'Mining the web with hierarchical crawlers – a resource sharing based crawling approach', *International Journal of Intelligent Information and Database Systems*, Vol. 3, No. 1, pp.90–106.

Kurasova, O., Marcinkevicius, V., Medvedev, V., Rapecka, A. and Stefanovic, P. (2014) 'Strategies for Big Data Clustering', *IEEE 26th International Conference on Tools with Artificial Intelligence*, pp.740–774.

Lakshmi, C. and Kumar, V.V.N. (2016) 'Survey Paper on Big Data', *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 6, No. 8, pp.368–381.

Mittal, M. (2017) 'Indexing Techniques and Challenge in Big Data', *International Journal of Current Engineering and Technology*, Vol.7, No.3, pp.1225–1228.

Mukhopadhyay, D., Kundu, A., and Sinha, S. (2010) 'Introducing Dynamic Ranking on WebPages Based on Multiple Ontology Supported Domains', *International Conference on Distributed Computing & Internet Technology (ICDCIT)*, Vol. 5966, pp. 104–109.

Oluwatosin, H.S. (2014) 'Client-Server Model', *IOSR Journal of Computer Engineering*, Vol. 16, No. 1, pp.67–71.

Oussous, A., Benjelloun, F-Z., Lahcen, A.A. and Belfkih, S. (2017) 'Big Data technologies: survey', *Journal of King Saud University – Computer and Information Sciences*, Vol. 30, No. 4, pp 431–448.

Pan, S., Xiong, L., Xu, Z., Chong, Y. and Meng, Q. (2018) 'A dynamic replication management strategy in distributed GIS', *Computers and Geosciences*, Vol. 112, pp.1–8.

Rehman, M.H., Liew, C.S., Abbas, A., Jayaraman, P.P., Wah, T.Y. and Khan, S.U. (2016) 'Big Data Reduction Methods: A Survey', *Data Science and Engineering*, Vol. 1, No. 4, pp.265–284, Springer.

Sahoo, J., Mohammad, A., Glitho, S.R., Elbiaze, H. and Ajib, W. (2016) 'A Survey on Replica Server Placement Algorithms for Content Delivery Networks', *IEEE Communications Surveys & Tutorials*, Vol. 19, No. 2, pp.1002–1026.

Sajana, T., Rani, C.M.S. and Narayana, K.V. (2016) 'A Survey on Clustering Techniques for Big Data Mining', *Indian Journal of Science and Technology*, Vol. 9, No. 3, pp.1–12.

Sanse, K. and Sharma, M. (2015) 'Clustering methods for big data analysis', *International Journal of Advanced Research in Computer Engineering & Technology*, Vol. 4, No. 3, pp.642–648.

Shu, H. (2016) 'Big data analytics: six techniques', *Geo-spatial Information Science*, Vol. 19, No. 2, pp.119–128.

Sreedhar, C., Kasiviswanath, N. and Reddy, P.C. (2017) 'Clustering large datasets using K-means modified inter and intra clustering (KM-I2C) in Hadoop', *Journal of Big Data*, Vol. 4, No. 27, pp.1–19, Springer.

Sultan, N. (2010) 'Cloud computing for education: A new dawn?', *International Journal of Information Management: The Journal for Information Professionals*, Vol. 30, No. 2, pp.109–116.

Tulgar, T., Haydar, A. and Ersan, I. (2018) 'A Distributed K Nearest Neighbor Classifier for Big Data', *Balkan Journal Of Electrical & Computer Engineering*, Vol. 6, No. 2, pp.37–43.

Wang, T., Yang, B., Huang, A., Zhang, Q., Gao, J., Yang, D., Tang, S. and Jinzhong, N. (2009) 'Dynamic Data Migration Policies for Query-Intensive Distributed Data Environments', *APWeb/WAIM'09 Proceedings of the Joint International Conferences on Advances in Data and Web Management*, pp.63–75.

Yu, B. (2019) 'Research on information retrieval model based on ontology', *EURASIP Journal on Wireless Communications and Networking*, pp.1–8.

Zerhari, B., Lahcen, A.A. and Mouline, S. (2015) 'Big Data Clustering: Algorithms and Challenges', *International Conference on Big Data, Cloud and Applications*, Morocco.