Revisiting Asymmetric Classifier Fuzzy Keyword Search with Performance Isolated Intertenant Storage Clusters in Public Cloud

Shyamala Devi Munisamy¹* and Arun Chokkalingam²

¹Department of Computer Science and Engineering, R.M.D. Engineering College, Kavaraipettai, Chennai - 601206, TamilNadu, India; shyamalapmr@gmail.com
²Department of Electronics and Communication Engineering, R.M.K. College of Engineering and Technology, Puduvoyal, Chennai - 601206, TamilNadu, India; carunece@gmail.com

Abstract

Background/Objectives: Virtual Clusters enabled users across globe to be able to access virtual machines, transact and deploy applications. Intertenant cloud customers demand availability and performance isolation from the cloud.

Methods/Statistical Analysis: Though fair allocation of resources guarantees performance isolation of intra-tenants in SaaS applications, it is still a challenging task in intertenant IaaS cloud.

Findings: Existing Universal keyword classifier is revisited to suggest a new scheme named Refined Asymmetric Classifier Fuzzy Keyword Search. The Classifier Search server utilizes universal keyword classifier for the search keyword request to greatly reduce the searching time by learning the search path pattern for all the keywords in the Fuzzy keyword set. In addition to this, the cloud service providers of our proposed scheme utilizes Performance Isolated Intertenant Storage Cluster to provide availability and performance isolation. Our implementation outcome have shown that our proposed searching scheme minimizes searching time on the cloud storage server by utilizing the data effectively with high availability and predictable performance isolation.

Applications/Improvements: This facilitates the cloud service provider to hold back the time to serve multiple customers as the searching time is absolutely diminished.

Keywords: Asymmetric Encryption, Intertenancy, Keyword Classifier, Performance Isolation, Storage Clusters

1. Introduction

Cloud computing¹-³ distribute elastic applications, web services and IT infrastructure as a service over the internet using utility pricing model. The cloud model enables the customer to outsource their confidential data to the cloud storage⁴-⁵ server which facilitates the users to access their data anywhere at any time. The data owners of the cloud storage⁶-⁷ have to pay only for the actual storage utilized.

1.1 Problem Formulation

The cloud storage server is primarily accountable to respond to data owners for data access at any time independently without affecting performance isolation and availability. With this remark, we propose a new searching method named Refined Asymmetric Classifier Fuzzy Keyword Search in which the cloud service provider develops the universal keyword classifier that stores the search path pattern of every keywords of their users data. This allows the cloud service provider to exploit its time successfully to carry out multiprocessing of increasing consumers. Our scheme also resolves typos and representation inconsistencies in view of the fact that the searching is done on B Tree fuzzy Searchable index. In this scheme, the cloud service provider uses Performance Isolated Intertenant Storage Clusters (PISC) that

*Author for correspondence
provides availability and performance isolation to retain data owners.

1.2 Our Contributions

In this paper, we propose new Scheme Refined Asymmetric Classifier Fuzzy Keyword Search (RACKS) which minimizes the searching time and distributes the files demanded by the data users. Our method also provides availability and performance isolation for data owners to hold on to the same cloud service provider. The objective is to increase data utilization, search efficiency with high availability and performance isolation. Our contributions are summarized as follows,

- The Cloud Storage Server cannot reveal the files to unauthorized users as the original keyword set and files are not known.
- Asymmetric Searchable encryption makes the cloud storage server to perform search on the BTree fuzzy searchable index thereby providing effective data utilization.
- Solves typos and representation inconsistencies of authorized users as the searchable index is created from wildcard based fuzzy keyword set.
- Classifier Search server creates Universal keyword classifier which saves the search path pattern of entire keywords for all the encrypted files by navigating the storage efficient BTree Wildcard fuzzy searchable index.
- Performance Isolated Intertenant Storage Clusters provides availability and fair resource allocation by Performance isolation.
- Availability is achieved by the existence of the Replica servers which is controlled by the Storage Cluster Controller.
- Storage Cluster Controller performs the dynamic parallel access of the files from the Replica Server which reduces the file access time.
- Performance Isolation is achieved by the existence of Performance Isolated Intertenant Storage Cluster, Storage SLA Monitor, Tenant Portfolio Regulator, Storage Demand Estimator, Tenant Metadata Organizer, and Tenant Storage Observer.

1.3 Paper Organization

The rest of the paper is organized as follows. The related work is discussed in Section 2 along with the preliminaries. In Section 3, we suggest our system architecture and design goals of the proposed scheme followed by the detailed description of Refined Asymmetric Classifier Fuzzy Keyword Search Scheme in Section 4 which discusses the detailed design and implementation of algorithms of our proposed scheme. The Experimental results and performance analysis with output are shown in Section 5. We conclude our paper in Section 6.

2. Background

2.1 Related Work

Although Cloud Service Provider (CSP) hosts several third party data, Liu et al. identified managing sensitive data would lead to security and privacy concerns. Ren et al. state that users have quite a lot of typing behaviours for keywords which is termed as typos, representation inconsistencies and typing habits. He suggested fuzzy keyword search to overcome these difficulties. Zhou et al. created k-gram based fuzzy keyword set for keywords W of the encrypted files and Jaccard coefficient to calculate the keywords similarity. Wang et al. argue that keyword holds sensitive information of files and thus keyword privacy must be confined for effective data utilization. Xu et al. recognized that third party could access the files by knowing the search trapdoor. The limitations of the existing searching techniques can be referred from.

The tenants of cloud have reported their objection towards their received service due to noisy neighbour problem. The performance isolation of the system is unpredictable if the co-positioned tenants try to clutch the computing resources excessively. When dealing with Input Output Service, it is only possible to measure in the form of “low, moderate and high” where the particular unit is absent. The tenants are not able to predict the network performance which results in relinquishing from the cloud service. Seawall prototype addresses the problem by tunneling packets through end-to-end congestion controlled tunnels. The current measurements express significant difference in network performance metrics. For example, the medium instances in EC2 performance throughput vary by 66%. The major basis for the variation is the incapability to control the network traffic share of a Virtual Machine. The proposed work depicts the method of collocating a trojan Virtual Machine with a target Virtual Machine. Oktopus prototype also discuss about the predictability in data center networks that uses
end-to-end mechanism combined with intelligent placement to provide strict bandwidth guarantee. The static allocations performed in SecondNet guarantees bandwidth but left the network underutilized. The Quality of Service based resource allocation for storage is addressed in Stonehenge and Aqua. The algorithms in these methods are used to allocate throughput or bandwidth in proportion to the predefined weights of the clients. The resource allocation in Argon uses cache management and time-sliced disk scheduling for performance insulation on a single shared file server. Some of the other works discussed in BVT, SMART provides additional bandwidth support for latency-sensitive applications. Additionally, the concept of pClock, mClock borrow the reservation and limit from CPU and memory scheduling to storage in additional to the proportional weight based allocation. Even though the Gatekeeper’s ingress and egress scheduling and Cloud’s per-endpoint sharing achieves efficient throughput, it provides performance fairness only on single virtual machine or to a group of virtual machine. It does not concentrate on the performance fairness of a single tenant.

2.2 Preliminaries
The concept of Searchable encryption types, Fuzzy keyword set, Wild card Fuzzy keyword set, K-gram based Fuzzy keyword set and Fuzzy Keyword Search can be referred from 13.

2.2.1 Performance Isolation
Performance issues in cloud storage are severe obstacles for data owners. To keep away from suspect, it is required to make sure that fair activities of the system are present with respect to its different tenants. Due to resource sharing, performance related problems are frequently caused by low demand tenants sending a high quantity of requests. The system is defined to be fair, if the following conditions are satisfied.

- Tenants working within their assigned quota should not suffer from customers exceeding their quotas.
- Customers exceeding their quotas should suffer from performance degradation.
- Customers with higher quotas should be provided better performance than customers with lower quotas.

Here the term “quota” refers to the profile level of the tenants. The tenant level refers to the quantity of storage or resource a tenant is allowed to utilize or execute. The fairness of the system is achieved by Performance isolation. A system is said to achieve Performance Isolation if the tenants working within their quotas while performance should not be affected when other tenants exceed their quotas. The system is said to be weak performance isolated where the Performance isolation is achieved within the limited range of exceeded quota from disrupting tenants. The system is said to be non-isolated if the behavior of any tenant may influence the performance observed by other tenants. So every tenant may suffer from horrific performance caused by one disruptive customer exceeding its quota.

3. System Methodology

3.1 Cloud Data Utilization System Architecture
The Refined Asymmetric Classifier Fuzzy Keyword Search cloud data utilization service architecture consists of five units as Data owner, Cloud Storage Server, Classifier Search Server, Performance Isolated Intertenant Storage Cluster and Data users and is shown in Figure 1. Here we assume that the authorization is suitably done between the data owner and data users. First the data owner generates User’s Public and Private Key Pair as (UPK, OPU). Data owner has a set of T data files \( DATF = \{DATF_1, DATF_2, \ldots, DATF_T\} \) that are encrypted with user’s public key UPK and are outsourced to the Performance Isolated Intertenant Storage Cluster. Data owner predefines keyword for each file. Data owner has a Keyword set \( KWD = (kwd_1, kwd_2, \ldots, kwd_T) \) of T data files. Data owner creates storage efficient Wild Card based Fuzzy Keyword Set as \( FKWD = (fkwd_1[ ], fkwd_2[ ], \ldots, fkwd_T[ ] ) \) using Wildcard based technique with the predefined edit distance value ‘edistance’. Data owner creates B Tree Wildcard fuzzy searchable index BTSI from Wild Card based Fuzzy Keyword Set. Data owner encrypts the files EFI and fuzzy searchable index BTSIE using User Public key UPK and is outsourced to the Performance Isolated Intertenant Storage Cluster. After receiving the encrypted files EFI and index BTSIE, the Performance Isolated Intertenant Storage Cluster outsources the encrypted files EFI and index BTSIE to cloud storage server. The cloud storage server in turn shares the encrypted fuzzy searchable index BTSIE to the classifier search server. The Data owner sends the User Private key as Private Secret key PVT to the data
users which is used by the data users to create search key-
word trapdoor KDT and for decrypting the file. The cloud
storage server has the encrypted T data files files EFI and
encrypted B Tree Wildcard fuzzy searchable index BTSI_E.
The classifier search server has the encrypted B Tree Wildcard fuzzy searchable index BTSI_E.

The Performance Isolated Intertenant Storage Clus-
ter also has the encrypted files EFI and encrypted B Tree
Wildcard fuzzy searchable index BTSI_E. The authori-
zation is suitably done between the data owner and the
cloud storage server. The Performance Isolated Interten-
ant Storage Cluster has eight entities as follows and is
shown in Figure 2.

- Storage SLA Monitor
- Tenant Portfolio Regulator
- Storage Demand Estimator
- Tenant Metadata Organizer
- Tenant Storage Observer
- Preemptive Priority Storage Scheduler
- Storage Cluster Controller
- Replica Servers

The Cloud Storage Server shares the Encrypted files to
the Storage Cluster Controller. The Storage Cluster Con-
troller replicates the entire encrypted files to the three
Replica Server. The number of Replica Server is decided
by the cloud storage. The level of the data owners are cate-
gorized based on the Service Level Agreement (SLA) with
the cloud storage server. The levels of the data owners
are of Diamond, Gold and Silver based on the amount of
storage demanded by them. For example, let’s assume that
the diamond level data owners demand the storage above
10 Terabyte (TB). The Gold level data owners demand
the storage above 10 Gigabyte (GB). The Silver level data
owners demand the storage above 10 Megabyte (MB). The
Service Level Agreement of all the data owners are main-
tained by Storage SLA Monitor. The levels of the data
owners are grouped by the Tenant Portfolio Regulator.

The data owners information and entire storage details
are organized by the Tenant Metadata Organizer. The addi-
tional storage demanded by the data owners are allocated dynamically at any time by Storage Demand
Estimator. The storage consumption details of the data
owners are observed dynamically by the Tenant Storage
Observer. The Preemptive Priority Storage Scheduler per-
forms preemptive scheduling based on the priority level.
The data user requests the search keyword KDR which
is encrypted using UTK private secret key to create Key-
word Trapdoor KDT which is sent to the cloud storage
server. If the request is arriving for the first time. If the request is
arriving for the first time, the Universal Keyword Classi-
fier creates the Universal Keyword Classifier and the search path of
the file is sent to the cloud storage server. After receiv-
ing the search path pattern of the search keyword request
from the classifier search server, the cloud storage server
sends the search path to the Storage Cluster Controller.
The Storage Cluster Controller finds the encrypted file
which is received by Performance Isolated Intertenant Storage Cluster. The Performance Isolated Intertenant Stor-
age Cluster maintains one thread pool for each diamond,
gold and silver level. The incoming search request KDR
is placed in the respective thread pools based on their
level. The incoming request in the thread pool is sent to
the Preemptive Priority Storage Scheduler based on First
Come First Serve basis. The request from the three thread
pools are given equal opportunity to move the request to
the Preemptive Priority Storage Scheduler in round robin
fashion. The Preemptive Priority Storage Scheduler places
the incoming request in the queue and performs the pri-
ority preemptive scheduling and move the front request
to the cloud storage server. The Universal Keyword Classi-
ifier creates the Universal Keyword Classifier and stores the path of the KDR by search-
ning over the encrypted B Tree Wildcard fuzzy searchable
index BTSI_E and sends the search path to the cloud stor-
age server. If the request given by the user matches with
the stored request then it is declared as a repeated key-
word. Then the classifier search server extracts the stored
search path pattern of the search keyword KDR from the
universal keyword classifier and the search path of
the file is sent to the cloud storage server. After receiv-
ing the search path pattern of the search keyword request
from the classifier search server, the cloud storage server
sends the search path to the Storage Cluster Controller.
The Storage Cluster Controller finds the encrypted file
size of the corresponding search path pattern. The file is
then divided into three equal sized parts. Then the first
part of the file is extracted from the Replica Server 1,
and the second part of the file from Replica Server 2 and
the third part of the file from Replica Server 3. This file
extraction from the three Replica Server is done paral-
lely. After receiving the encrypted file EFI parts, they are
accumulated by the Storage Cluster Controller which in
turn is sent to the Cloud Storage Server. The Cloud Stor-
age Server sends the matched encrypted files EFI to the
Performance Isolated Intertenant Storage Cluster which in
turn sends the encrypted files to the data users. On
receiving the encrypted files, the data user decrypts the
files as DFI using private secret key PVT.
Figure 1. Cloud Data Utilization Service Architecture for Refined Asymmetric Classifier Fuzzy Keyword Search (RACKS).
3.2 Design Goals

To successfully reduce the searching time for keywords in the cloud storage server, tolerating the typos and representation inconsistencies of authorized users and to achieve the availability and Performance Isolation, the objective of the proposed searching scheme Refined Asymmetric Classifier Fuzzy Keyword Search is to achieve the following design goals.

3.2.1 Search Efficiency Goals

- To create the universal keyword classifier for all the search keyword request by performing the search on the encrypted BTree wildcard Fuzzy searchable index for optimizing search time.
- To perform dynamic parallel access of the encrypted files from the three Replica server thereby reducing the searching time.
3.2.2 Data Utilization Efficiency Goals

- To create wild card based Fuzzy keyword set for all the keywords thereby tolerating typos and representation inconsistencies of authorized users and to increase to the number of files accessed for a particular keyword.

3.2.3 Security Goals

- To refrain the Cloud Storage Server, Performance Isolated Intertenant Storage Cluster, Storage Cluster Controller and Classifier search server from getting the knowledge of data files and keyword set. This is achieved by outsourcing the encrypted files and index from the data owners.

3.2.4 Privacy Goals

- To provide User Privacy by hiding the file details, key-word and index to the Cloud Storage Server.
- To support data privacy by encrypting the files and index with user public key before outsourcing to the Cloud Storage Server.
- To attain keyword privacy by forming encrypted Btree wildcard Fuzzy searchable index from the wild card based Fuzzy keyword set for the predefined keywords.
- To achieve query privacy by sending encrypted key-word trapdoor to the Performance Isolated Intertenant Storage Cluster.
- To accomplish Index privacy by creating encrypted B Tree wildcard fuzzy searchable index.

3.2.5 Availability and Isolation Goals

- To replicate the encrypted files to the three Replica Server thereby achieving availability.
- To attain performance isolation by maintaining the thread pools based on the demand levels by the Performance Isolated Intertenant Storage Cluster.
- To attain performance isolation by placing the search request in the queue and by performing the priority preemptive scheduling by the Preemptive Priority Storage Scheduler.

4. Refined Asymmetric Classifier Fuzzy Keyword Search

4.1 Notations

- PVT : Private Secret Key
- edistance: Edit Distance
- DATF = \{DATF_1, DATF_2, ..., DATF_T\} : set of T encrypted data files DATF
- UPK : User Public Key
- OPU : Owner Public Key
- UTK : User Private Key
- KWD = \{(kwd_1, kwd_2, ..., kwd_k)\} : Predefined Keyword Set of DATF
- FKWD = \{(fkwd_1, fkwd_2, ..., fkwd_k)\} : Wild Card Based Fuzzy Keyword Set
- BTSI: BTree WildCard Fuzzy searchable Index
- BTSI_i: Encrypted BTree WildCard Fuzzy searchable Index of BTSI
- EFI : Set of Encrypted files matching search request
- DFI : Decrypted files of EFI
- PL : Profile Level
- DOI : Data Owner Identifier
- DUI : Data User Identifier
- PPS : Priority Preemptive Scheduling
- SSM : Storage SLA Monitor
- TPR: Tenant Portfolio Regulator
- SDE: Storage Demand Estimator
- TMO: Tenant Metadata Organizer
- TSO: Tenant Storage Observer
- P2S: Preemptive Priority Storage Scheduler
- SCC: Storage Cluster Controller
- PISC : Performance Isolated Intertenant Storage Cluster
- RPS : Replica Server
- RACKS : Refined Asymmetric Classifier Fuzzy Keyword Search
- CSS : Cloud Storage Server

4.2 Assumptions of RACKS

The following assumptions are made to design the proposed scheme Refined Asymmetric Classifier Fuzzy Keyword Search.

- Cloud Storage Server concentrates on retaining the customers and not to leave up their tenants agreements.
- Authorization is suitably done between the Data Owner and Data Users, Data Owner and Cloud Storage Server, Data Users and Cloud Storage Server.
- Data Owner predefines the keyword for each file.
Wildcard based Fuzzy Keyword Set FKWD of Keyword set KWD contains the original keyword as the first element. Each file has unique keyword.

Performance Isolated Intertenant Storage Cluster maintains three thread pools for each Diamond, Gold and Silver level tenants.

Priority level is assigned as ‘1’ for Diamond customers, ‘2’ for Gold customers and ‘3’ for Silver customers.

Preemptive Priority Storage Scheduler maintains a single queue and performs priority preemptive scheduling.

Three Replica servers are connected to Storage Cluster Controller.

The level of the data user is same as their data owners.

4.3 Function Definitions of Refined Asymmetric Classifier Fuzzy Keyword Search

The following functions are implemented to optimize the searching on the cloud storage server and to achieve the effective data utilization, availability and performance isolation.

Functions on Data owner:

- Generate Secret KeyPairs (SECRETKEY1, SECRETKEY2).
- Generate Encrypted Files (UPK[]).
- Generate Wild Fuzzy Keyword Set (KWD[ ], edistance).
- Generate B Tree Fuzzy Searchable Index (FKWD[ ][ ]).
- Generate Encrypted B Tree Fuzzy Searchable Index(BTSI, UPK[ ]).

Functions on Cloud Storage Server:

- FetchSearch Path Pattern(KDT)

Functions on Classifier Search Server:

- Search B Tree Fuzzy SearchableIndex(BTSIE, FKWD[ ][ ])

4.4 Overall Prototype of Refined Asymmetric Classifier Fuzzy Keyword Search

Our proposed method Refined Asymmetric Classifier Fuzzy Keyword Search has Performance Isolated Intertenant Storage Cluster which achieves the availability and performance isolation to the customers. The Classifier Search Server that creates the search path pattern for all the keywords of T encrypted set of files. The Storage Cluster Controller performs Dynamic Parallel Access for retrieving the files from the Replica Server thereby achieving availability and reduces the downloading time. Data owner creates Encrypted BTre Wildcard Fuzzy Searchable Index for the fuzzy keyword set and is outsourced to the cloud storage server. This overcomes the problem of typos and representation inconsistencies behaviour of the data users and the overall framework is given in Algorithm 1.

Algorithm 1: RACKS() (Refined Asymmetric Classifier Fuzzy Keyword Search)

// Actions Performed by Data Owner.
/* Variables Initialization */
[1] Initialize Secret Keys SECRETKEY1 and SECRETKEY2.
[2] Predefine the edit distance ‘edistance’ value for creating Wildcard Fuzzy Keyword Set.
[3] Predefine Keywords for each file.
[4] Create Keyword Set KWD.
/* KeyGeneration */
[5] DataOwner Creates User Public key and User Private key pairs(UPK , OPU).
[6] Invoke GenerateSecretKeyPairs(SECRETKEY1, SECRETKEY2)
[7] return (UPK , OPU) // User Public key and User Private key pairs.
[8] Encrypt the T data files using UPK as DATF = {DATF1, DATF2, …., DATFT}.
[9] Invoke GenerateEncryptedFiles(UPK[ ])
[10] Receive the Encrypted file \( \text{DATF} = \{\text{DATF}_1, \text{DATF}_2, \ldots, \text{DATF}_T \} \).
[11] Data owner creates Wildcard based Fuzzy Keyword set for \( \text{KWD} \).
[12] Invoke \( \text{GenerateWildFuzzyKeywordSet} (\text{KWD}, \text{edistance}) \).
[13] Receive \( \text{FKWD} = \{ (\text{fkwd}_1, \text{fkwd}_2), \ldots, (\text{fkwd}_T, \text{kwd}) \} \).
[14] Data owner creates B Ttree Wildcard searchable index \( \text{BTSI} \) for FMKS.
[15] Invoke \( \text{GenerateBTreeFuzzySearchableIndex} (\text{FKWD}) \).
[16] Receive B Ttree Wildcard searchable index \( \text{BTSI} \).
[17] Encrypts the B tree wildcard fuzzy searchable index \( \text{BTSI} \).
[18] Invoke \( \text{GenerateEncryptedBTreeFuzzySearchableIndex} (\text{BTSI}, \text{UPK}) \).
[19] Receive Encrypted BTree wildcard Fuzzy searchable index \( \text{BTSI}_{\text{E}} \).
[20] Data owner sends the Encrypted files \( \text{DATF} = \{\text{DATF}_1, \text{DATF}_2, \ldots, \text{DATF}_T \} \) to the Performance Isolated Intertenant Storage Cluster.
[21] Data owner creates the B Ttree Wildcard searchable index \( \text{BTSI}_{\text{E}} \) to the Performance Isolated Intertenant Storage Cluster.
[22] Data owner sends the User Private Key as Private Secret Key \( \text{PVT} \) to the Data Users.

// Actions Performed by Performance Isolated Intertenant Storage Cluster.
[23] PISC sends the Encrypted files \( \text{DATF} = \{\text{DATF}_1, \text{DATF}_2, \ldots, \text{DATF}_T \} \) to CSS.
[24] PISC sends the Encrypted BTree wildcard Fuzzy searchable index \( \text{BTSI}_{\text{E}} \) to CSS.
[25] PISC initializes three thread pools.
[26] Storage SLA Monitor build the SLA database of all the data owners.
[27] Invoke \( \text{MonitorCustomersSLA} () \).
[28] If the data owner diverge from their SLA Then
[29] Report it to the Performance Isolated Intertenant Storage Cluster.
[30] Tenant Portfolio Regulator builds the database about the data owners level.
[31] Tenant Portfolio Regulator categorizes and group the data owners with the same level.
[32] Invoke \( \text{RegulateCustomersProfile} () \).
[33] Storage Demand Estimator builds the database about the excess storage requested by the data owner.
[34] Invoke \( \text{EstimateStorageDemand} () \).
[35] Tenant Metadata Organizer build the database with all the details of the data owners and their data users.
[36] Invoke \( \text{OrganizeCustomersMetadata} () \).
[37] Tenant Storage Observer updates the storage consumed by the data owners.
[38] Invoke \( \text{ObserveCustomerStorage} () \).
[39] Preemptive Priority Storage Scheduler initializes a queue to insert the search request.

// Actions Performed by Cloud Storage Server.
[40] Cloud storage server sends the Encrypted BTree wildcard Fuzzy searchable index \( \text{BTSI}_{\text{E}} \) to the Classifier Search Server.
[41] Cloud storage server sends the Encrypted files \( \text{DATF} = \{\text{DATF}_1, \text{DATF}_2, \ldots, \text{DATF}_T \} \) to the Storage Cluster Controller.

// Actions Performed by Storage Cluster Controller.
[42] Receive the Encrypted files \( \text{DATF} = \{\text{DATF}_1, \text{DATF}_2, \ldots, \text{DATF}_T \} \) from Cloud Storage Server.
[43] Storage Cluster Controller sends the Encrypted files \( \text{DATF} = \{\text{DATF}_1, \text{DATF}_2, \ldots, \text{DATF}_T \} \) to the Replica Server 1.
[44] Storage Cluster Controller sends the Encrypted files \( \text{DATF} = \{\text{DATF}_1, \text{DATF}_2, \ldots, \text{DATF}_T \} \) to the Replica Server 2.
[45] Storage Cluster Controller sends the Encrypted files \( \text{DATF} = \{\text{DATF}_1, \text{DATF}_2, \ldots, \text{DATF}_T \} \) to the Replica Server 3.

// Actions Performed by Data Users.
[46] Data User request search request \( \text{KDR} \).
[47] Data User encrypts the search request \( \text{KDR} \) with Private Secret Key \( \text{PVT} \) to create search Keyword Trapdoor \( \text{KDT} \).
[48] Search Keyword Trapdoor \( \text{KDT} \) is sent to the Performance Isolated Intertenant Storage Cluster.

// Actions Performed by Performance Isolated Intertenant Storage Cluster.
[49] PISC receives the Search Keyword Trapdoor \( \text{KDT} \) from the data users.
[50] PISC categorize and assigns the request to the respective threads.
[51] Invoke \( \text{AssignThreadPool} (\text{KDT}, \text{DUI}) \).
[52] PISC sends the search Keyword Trapdoor \( \text{KDT} \) to the Preemptive Priority Storage Scheduler in round robin fashion.

// Actions Performed by the Preemptive Priority Storage Scheduler.
[53] Preemptive Priority Storage Scheduler receives the Search Keyword Trapdoor \( \text{KDT} \) from PISC.
Preemptive Priority Storage Scheduler places the incoming request KDT to the queue.
Preemptive Priority Storage Scheduler performs Priority preemptive Scheduling.
Invoke PerformPriorityPreemptiveScheduling()
Preemptive Priority Storage Scheduler sends the KDT to Cloud storage server
// Actions Performed by the Cloud storage server.
Cloud Storage server receives the KDT from Preemptive Priority Storage Scheduler.
Cloud Storage server send the KDT to the Classifier Search Server.
// Actions Performed by the Classifier Search Server.
Receive the Encrypted BTree Index from Cloud Storage Server.
Classifier Search Server receives the KDT from Cloud storage server.
Searches BTree wildcard searchable index BTSI_E
If the Keyword KDT is the new keyword then
Invoke SearchBTreeFuzzySearchableIndex(BTSI_E, FKWD[])
Keyword Classifier learn and Store the Search path pattern of KDT
Sends the Search path pattern of that keyword KDT to the cloud storage server
else if KDT is already present in the Keyword Classifier then
Classifier Search Server sends the stored Search path pattern of KDT to CSS
End if
// Actions Performed by the Cloud storage server.
Cloud Storage Server receives Search path pattern of KDT from Classifier Search Server
Invoke FetchSearchPathPattern(KDT)
Cloud Storage Server sends stored Search path of KDT to Storage Cluster Controller.
// Actions Performed by the Storage Cluster Controller.
Storage Cluster Controller receives the Search path pattern of KDT from CSS Server
Finds the file size of the corresponding Search path pattern of KDT
Divides the file size into three.
Retrieve the first part of the file from Replica Server 1
Retrieve the second part of the file from Replica Server 2
Retrieve the third part of the file from Replica Server 3
Invoke DynamicParallelAccess( )
Receive the Files extracted from Replica server
Accumulate the file parts from three Replica Servers
Invoke AccumulateFileParts( )
Receive the assembled File extracted from Replica server
Storage Cluster Controller sends the encrypted files EFI to the Cloud storage server.
// Actions Performed by the Cloud storage server.
Cloud Storage Server sends the matching encrypted files EFI to the PISC
PISC sends the matching encrypted files EFI to the data users.
// Actions Performed by the Data users
Receive the set of Matching Encrypted files EFI from PISC
Data User decrypt the file using Private Secret Key PVT
Invoke GenerateDecryptedFiles(UTK[], EFI)
Receive the Superset of matching Decrypted file DFI

4.5 Overall Framework of Refined Asymmetric Classifier Fuzzy Keyword Search
The Overall conceptual description of the Refined Asymmetric Classifier Fuzzy Keyword Search is shown as an activity diagram and is shown below in Figure 3.

4.6 Key Generation, Encryption and Decryption Algorithm
RSA Public Key Algorithm is used for creating public and private key and also for encrypting files, BTree Wildcard based Fuzzy Searchable Index. The same algorithm is also used for decrypting the files using Private Secret key and forming the search keyword trapdoor. The algorithm for following fuction GenerateSecretKeyPairs(SECRETKEY1, SECRETKEY2), GenerateEncryptedFiles(UPK[]), GenerateEncryptedBTreeFuzzySearchableIndex(BTSI, UPK[]), GenerateDecryptedFiles(UTK[], EFI) can be referred from.

4.7 WildCard based Fuzzy Keyword Set Algorithm
Data Owner has a Keyword set KWD = (kwd_1, kwd_2,…, kwd_T) of T Data files. Data Owner creates storage efficient FKWD = (fkwd_1[], fkwd_2[], …, fkwd_T[]) Wild Card based
Data Owner

- Initialize the Secret Keys
- Predefine Edit Distance
- Predefine Keywords for all the Files
- Create Keyword Set
- Create User Public Key and User Private Key
- Encrypt the files with User Public Key
- Create BTree Wildcard Fuzzy Searchable Index
- Encrypt BTree Wildcard Fuzzy Searchable Index
- Send the Encrypted Files and BTree Index to PISC
- Send the Private Secret Key to the Data Users
- Receive Private Secret Key

Data Users

- Receive Encrypted Files and BTree Index
- Send Encrypted Files and BTree Index to CSS
- Send the Encrypted Files and BTree Index to PISC
- Receive Encrypted Files and BTree Index

Performance Isolated Intertenant Storage Cluster

- Receive Encrypted Files and BTree Index
- Send Encrypted Files and BTree Index to CSS
- Send Encrypted Files to SCC
- Send Encrypted BTreeE to Classifier Search Server
- Build SLA Database
- Build the Database of Dataowners Level

Cloud Storage Server

- Receive Encrypted Files, BTree Index
- Send Encrypted Files to SCC
- Send Encrypted BTreeE to Classifier Search Server
- Initialize Three thread Pools
- Build SLA Database
- Build the Database of Dataowners Level
- Send Encrypted Files, BTree Index
| Storage Cluster Controller | Data Users | Performance Isolated Intertenant Storage Cluster | Cloud Storage Server |
|---------------------------|------------|-----------------------------------------------|----------------------|
| **B**                    | **C**      | **A**                                         |                      |
| Receive Encrypted files from CSS | Request the Search Keyword | Group Data Owners based on Level |                      |
| Send Encrypted files to Replica Server 1 | Encrypt Search Request with PVT | Build Dataowners Database demanding Excess Storage |                      |
| Send Encrypted files to Replica Server 2 | Send Encrypted Search Request to PISC | Build Dataowners Meta Database |                      |
| Send Encrypted files to Replica Server 3 | | Build Storage Consumption Database of Dataowners |                      |
| | | Scheduler initializes a Queue |                      |
| | | Receives Encrypted Search Request |                      |
| | | Assign Search Request to respective thread pool |                      |
| | | Send Encrypted search request trapdoor to P2S |                      |
| | | Perform Priority Preemptive Scheduling and send request to CSS |                      |
| | | |                      |
| Classifier Search Server | Data Users | Storage Cluster Controller | Cloud Storage Server |
|--------------------------|------------|----------------------------|----------------------|
| Receive Search Trapdoor from CSS | [New Keyword] | Searches Encrypted B Tree Index | Receives Encrypted search trapdoor from P2S |
| Evaluates if Trapdoor matches with Keyword in Stored Classifier | | Classifier Search Server Stores the Search Path Pattern of that new keyword | Send Encrypted search trapdoor to Classifier Search Server |
| [New Keyword] | | Send Path of Keyword Request to CSS | Receives Search Path from Classifier Search Server |
| Searches Encrypted B Tree Index | | | Receives Search Path from CSS |
| Classifier Search Server Stores the Search Path Pattern of that new keyword | | | Extract the File Size of the Search Path |
| Send Path of Keyword Request to CSS | | | Divide the file Size into Three |
| | [Existing Keyword] | | Perform Dynamic Parallel Access |
| Searches the Stored Search Path Pattern | | | Retrieve First Part from Replica Server 2 |
| Send the Search Path of Request to CSS | | | |
Figure 3. Activity Diagram of Refined Asymmetric Classifier Fuzzy Keyword Search.
Fuzzy Keyword Set using Wildcard based technique with the predefined edit distance value 'edistance'. Refer\textsuperscript{13} for creating Wild Card based Fuzzy Keyword Set.

4.8 B Tree WildCard Fuzzy Searchable Index Algorithm

Data owner creates B Tree Wildcard fuzzy searchable index BTSI from FKWD Wild Card based Fuzzy Keyword Set. Refer\textsuperscript{13} for creating B Tree Wildcard fuzzy searchable index BTSI.

4.9 Performance Isolation and Availability Algorithm

Performance isolation and availability is attained by Performance Isolated Intertenant Storage Cluster. Performance isolation is achieved by creating three thread pools to categorize the users request based on their level. The performance of one tenant is not affected by other tenants. Moreover the meta data information about the data owners, their requested storage in Service Level Agreement, their actual consumed storage and the excess store ie utilized are maintained as individual database by the respective components of Performance Isolated Intertenant Storage Cluster. The data user requests the encrypted search keyword trapdoor to the Performance Isolated Intertenant Storage Cluster. On receiving the keyword trapdoor request, it is assigned to the respective thread pool by executing the Algorithm 2.

Algorithm 2: AssignThreadPool(KDT, DUI)

\begin{itemize}
  \item Input: KDT : Encrypted Search Keyword Trapdoor.
  \item DUI : Data User Identifier.
  \item [1] Initialize three queues.
  \item [2] Queue FirstQueue, SecondQueue, ThirdQueue;
  \item [3] // FirstQueue for Diamond level Tenants.
  \item [4] // SecondQueue for Gold level Tenants.
  \item [5] // ThirdQueue for Silver level Tenants.
  \item [6] Initialize three Threads for each queue.
  \item [7] Thread Threadone, Threadtwo, Threadthree;
  \item [8] Threadone.execute(FirstQueue);
  \item [9] Threadtwo.execute(SecondQueue);
  \item [10] Threadthree.execute(ThirdQueue);
  \item [11] if (strcmp(DUI.LevelSLA,"DIAMOND") = = 0) then
  \item [12] Assign KDT to Threadone.FirstQueue;
  \item [13] else if(strcmp(DUI.LevelSLA,"GOLD") = = 0) then
  \item [14] Assign KDT to Threadtwo.SecondQueue;
  \item [15] else if(strcmp(DUI.LevelSLA,"SILVER") = = 0) then
  \item [16] Assign KDT to Threadthree.ThirdQueue;
  \item [17] End if
  \item [18] End AssignThreadPool(KDT,DUI)
\end{itemize}

The Storage SLA Monitor component of Performance Isolated Intertenant Storage Cluster maintains the Service Level Agreement database of all the data owners and their users. The Service Level Agreement contains the needs of the data owners such as availability, Data Location, Scalability, StorageLimit, Performance, Portability, Security, Reliability, Usability, Backup, Recovery and level. The SLA is decided by the Cloud storage server. We consider the following parameters of SLA. It also monitors the unauthenticated access of the cloud storage by the data owners or by users by executing the algorithm 3.

- Availability refers to the time at which the data owners and their respective users need the storage access.
- Data location refers to the availability zones and the need of the data owner in which their outsourced data should be stored.
- Scalability refers whether the data owner need for increase and decrease in their storage space.
- Performance refers to the response time need by the data owner to serve their data users request.
- Portability refers to the data owners service needs to work on different devices or different platforms.
- Security refers to the need of security level by the data owners.
- Reliability refers to the data owners need to operate the storage access over the time without failure.
- Usability refers to the user interface design need of the data owners for accessing the cloud storage.
- Backup refers to the data owners need for storing their data image.
- Recovery refers to the data owners need for the ability to recover the data in disaster.
- Level refers to the data owners and their users profile level (Diamond, Gold, Silver).

Algorithm 3: MonitorCustomerSLA(DOI, DUI)

\begin{itemize}
  \item Input: DOI : Data Owner Identifier. DUI : Data User Identifier.
  \item /*Varaiable Initialization*/
\end{itemize}
[1] Declare the variables AvailableSLA, LocationSLA;
[2] Declare the variables Scalable SLA, Perform SLA;
[3] Declare the variables Portable SLA, Secure SLA;
[4] Declare the variables Reliable SLA, Usable SLA;
[5] Declare the variables Backup SLA, Recovery SLA;
[6] Declare the variable LevelSLA;
[7] if DUI. UsageTime > Available SLA then
[8] Report the Availability Block Exception to PISC
[9] if DUI. Data Location not equal to Location SLA then
[10] Report the Location Block Exception to PISC
[11] if DUI. requested Storage is not assigned as Scalable SLA then
[12] Report the Scalability Block Exception to PISC
[13] if DUI. ResponseTime > Perform SLA then
[14] Report the Performance Block Exception to PISC
[15] if DUI. Platform is not supported like Portable SLA then
[16] Report the Portability Block Exception to PISC
[17] if DUI. Security is violated with Secure SLA then
[18] Report the Security Block Exception to PISC
[19] if DUI. Reliable Need is failed with Reliable SLA then
[20] Report the Reliability Block Exception to PISC
[21] if DUI. UserExperience is not meeting with Usable SLA then
[22] Report the Usability Block Exception to PISC
[23] if DUI. ActualBackup image is not stored as Backup SLA then
[24] Report the Backup Block Exception to PISC
[25] if DUI. Recover is not able to recover the data as Recovery SLA then
[26] Report the Recovery Block Exception to PISC
[27] if DUI. level is not assigned to respective thread pool as Level SLA then
[28] Report the Level Block Exception to PISC
[29] End if
[30] End MonitorCustomerSLA(DUI)

The Tenant Portfolio Regulator updates the database by categorizing and grouping the users based on their profile level. It also monitors the thread pools to check if the users are assigned correctly to the respective thread pools by executing the Algorithm 4. The diamond level data owners demand the storage above 10 Terabyte. The Gold level data owners demand the storage above 10 Gigabyte. The Silver level data owners demand the storage above 10 Megabyte.

Algorithm 4: RegulateCustomersProfile(DOI, DUI)

Input: DOI : Data Owner Identifier.
DUI : Data User Identifier.

[1] Declare the database for DiamondDB;
[2] Declare the database for GoldDB;
[3] Declare the database for SilverDB;
[4] Predefine the limit for DiamondDB;
[5] Predefine the limit for GoldDB;
[6] Predefine the limit for SilverDB;
[7] Receive the Data owner’s details from PISC.
[8] Receive the Data user’s details from PISC.
[9] For i = 1 to number(DOI) do
[10] if (Strcmp(DOI[i].LevelSLA,"DIAMOND") = 0) then
[11] Group the datausers of DOI[i] to DiamondDB;
[12] else if (Strcmp(DOI[i].LevelSLA,"GOLD") = 0) then
[13] Group the datausers of DOI[i] to GoldDB;
[14] else if (Strcmp(DOI[i].LevelSLA,"SILVER") = 0) then
[15] Group the datausers of DOI[i] to SilverDB;
[16] end if
[17] Access the three thread pools of PISC.
[18] for j = 1 to num(KDT) in threadone
[19] if threadone[i].KDT is not in DiamondDB
[20] Report the Level Misplacement Block Exception to PISC
[21] end if
[22] end for
[23] for k = 1 to num(KDT) in threadtwo
[24] if threadtwo[i].KDT is not in GoldDB
[25] Report the Level Misplacement Block Exception to PISC
[26] end if
[27] end for
[28] for m = 1 to num(KDT) in threadthree
[29] if threadthree[i].KDT is not in SilverDB
[30] Report the Level Misplacement Block Exception to PISC
[31] end if
[32] end for
[33] End RegulateCustomersProfile(DUI, DOI)

The Storage Demand Estimator monitors the storage limit utilized by the data users of the corresponding data owners. If any data users are demanding extra storage
that exceeds their data owners level, then it estimates the excess storage utilized by the data users and updates its database. This is done by executing the Algorithm 5.

**Algorithm 5: EstimateStorageDemand**(DOI, DUI)

Input: DOI : Data Owner Identifier.
      DUI : Data User Identifier.

[1] Declare the database for ExcessStorageDB;
[2] Receive the Data owner's details from PISC.
[3] Receive the Data user's details from PISC.
[4] for j = 1 to num(KDT) in threadone
[5] extract the DOI of threadone[j].KDT
[6] if storage demanded by DOI.DUI.KDT > DiamondDB
[7] Find the excess storage utilized by DUI.
[8] Update the extra storage in ExcessStorageDB for the corresponding DOI.
[9] end if
[10] end for
[11] for k = 1 to num(KDT) in threadtwo
[12] extract the DOI of threadtwo[k].KDT
[13] if storage demanded by DOI.DUI.KDT > GoldDB
[14] Find the excess storage utilized by DUI.
[15] Update the extra storage in ExcessStorageDB for the corresponding DOI.
[16] end if
[17] end for
[18] for m = 1 to num(KDT) in threadthree
[19] extract the DOI of threadthree[m].KDT
[20] if storage demanded by DOI.DUI.KDT > SilverDB
[21] Find the excess storage utilized by DUI.
[22] Update the extra storage in ExcessStorageDB for the corresponding DOI.
[23] end if
[24] end for
[25] End EstimateStorageDemand(DUI)

The Tenant Metadata Organizer maintains the metadata of all the Data owners and their data users. Entire details of all the Data owners are updated by executing the Algorithm 6.

**Algorithm 6: OrganizeCustomersMetadata**(DOI, DUI)

Input: DOI : Data Owner Identifier.
      DUI : Data User Identifier.

[1] Declare the database for MetadataDB;
[2] Receive the Data owner's details from PISC.
[3] Receive the Data user's details from PISC.
[4] for i = 1 to num(SLA) in Storage SLA Monitor
[5]   for each DOI do
[6]     Extract the total DUI of their respective DOI
[7]     Update the data users for its DUI in MetadataDB;
[8]   end for
[9] end for // i loop
[10] for each DOI in MetadataDB do
[11]   Assign the properties of DOI to its data users.
[12]   Update the data users properties in MetadataDB;
[13] end for
[14] End OrganizeCustomersMetadata(DOI)

The Tenant Storage Observer update the storage consumed by the data users of the cloud storage. It also monitors and report the excess storage to the Storage Demand Estimator by executing the Algorithm 7.

**Algorithm 7: ObserveCustomerStorage**(DOI, DUI)

Input: DOI : Data Owner Identifier.
      DUI : Data User Identifier.

[1] Declare the database for ConsumeStorageDB;
[2] Receive the Data owner's details from PISC.
[3] Receive the Data user's details from PISC.
[4] Access the three thread pools of PISC.
[5] for j = 1 to num(KDT) in threadone
[6] extract the DOI of threadone[j].KDT
[7] if storage demanded by DOI.DUI.KDT > DiamondDB
[8] Report the excess storage utilization to Storage Demand Estimator
[9] end if
[10] end for
[11] for k = 1 to num(KDT) in threadtwo
[12] extract the DOI of threadtwo[k].KDT
[13] if storage demanded by DOI.DUI.KDT > GoldDB
[14] Report the excess storage utilization to Storage Demand Estimator
[15] end if
[16] end for
[17] End ObserveCustomerStorage(DOI, DUI)
[20] end for
[21] for m = 1 to num(KDT) in threadthree
[22] extract the DOI of threadthree[j].KDT
[23] Find the storage consumed the DUI of its DOI
[24] Update the storage utilized in ConsumeStorageDB;
[25] if storage demanded by DOI.DUI.KDT > SilverDB
[26] Report the excess storage utilization to Storage Demand Estimator
[27] end if
[28] end for
[29] End ObserveCustomerStorage(DOI, DUI)

The Performance Isolated Intertenant Storage Cluster sends the request from each thread one by one by giving equal chance by executing the Algorithm 8.

Algorithm 8: ThrowRequestToP2S()

[1] Declare the variable i;
[2] for
[3] While(threadone ≠ empty | | threadtwo ≠ empty | | threadthree ≠ empty)
[4] do
[5] throw the threadone.FirstRequest to P2S
[6] // P2S - Preemptive Priority Storage Scheduler
[7] throw the threadtwo.FirstRequest to P2S
[8] throw the threadthree.FirstRequest to P2S
[9] end while
[10] End ThrowRequestToP2S()

The Preemptive Priority Storage Scheduler performs priority preemptive Scheduling for the incoming request thrown from PISC by executing Algorithm 9.

Algorithm 9: PerformPriorityPreemptiveScheduling(KDT)

Input: KDT : Search Keyword Trapdoor.

[1] Receive the Search Keyword Trapdoor thrown by PISC;
[2] Assign the Search Keyword Trapdoor to its queue;
[3] Declare the variable Queue RequestQueue;
[4] for each request in RequestQueue
[5] extract the DUI of KDT;
[6] Find the DOI of DUI;
[7] // CSS.CurrentProcess.KDT indicate the current process running in CSS;
[8] if(strcmp(DOI.DUI.KDT.LevelSLA, “DIAMOND”) = = 0 )
[9] Check the process currently running in CSS;
[10] if (strcmp (CSS.CurrentProcess.KDT.LevelSLA,”GOLD” | | SILVER”) = = 0 )
[11] Make the CSS.CurrentProcess.KDT to wait until DOI.DUI.KDT to complete;
[12] Throw the request DOI.DUI.KDT to CSS;
[13] end if
[14] end if
[15] else if (strcmp (DOI.DUI.KDT.LevelSLA, “GOLD”) = = 0 )
[16] Check the process currently running in CSS;
[17] if(strcmp(CSS.CurrentProcess.KDT.LevelSLA,”DIAMOND”) = = 0 )
[18] if( CSS.CurrentProcess.KDT = = COMPLETE)
[19] Throw the request DOI.DUI.KDT to CSS;
[20] else
[21] Make the request DOI.DUI.KDT to wait;
[22] DOI.DUI.KDT.Wait();
[23] end if
[24] else
[25] if(strcmp(CSS.CurrentProcess.KDT.LevelSLA,”SILVER”) = = 0 )
[26] Make the CSS.CurrentProcess.KDT to wait until DOI.DUI.KDT to complete;
[27] Throw the request DOI.DUI.KDT to CSS;
[28] end if
[29] else if(strcmp(DOI.DUI.KDT.LevelSLA, “SILVER”)
[30] Check the process currently running in CSS;
[31] if(strcmp(CSS.CurrentProcess.KDT.LevelSLA,”DIAMOND”) = = 0 )
[32] if( CSS.CurrentProcess.KDT = = COMPLETE)
After receiving the search keyword trapdoor KDT, the cloud storage server sends KDT to Classifier search server. The Classifier Search Server processes to check if the request is coming for the first time. If the request is arriving for the first time, then the keyword classifier captures and stores the path of KDT by searching over the encrypted B Tree Wildcard fuzzy searchable index BTSI_E by executing Algorithm 10 and sends the search path to the cloud storage server. If the request KDT matches with a previous request stored in the keyword classifier then it is existing keyword. The classifier search server extracts the stored search path patterns of the KDT from the universal keyword classifier and the search path is sent to the cloud storage server.

**Algorithm 10: SearchBTreeFuzzySearchableIndex (BTSI_E, KDT)**

Input: BTSI_E – Encrypted BTree Wildcard Fuzzy Searchable Index  
KDT – Encrypted Search Keyword Trapdoor  
Output: Set of Matched Encrypted Files

1. Declare the variable i, k for processing loop
2. Find the height of the B Tree index BTSI_E and assign it to Tree Height
3. Add Node children Node Value = BTSI_E . children Node Value;
4. if (Tree Height == 0) then
5. for i = 0 to Number of Child do
6. if (KDT = = children Node Value [i]. KDT) then
7. return (Node Value) children Node Value [i].Node Value;
8. end if
9. end for // i loop
10. else for k = 0 to BTSI_E. Number of Child do
11. if((k+1 = = BTSI_E. Number of Child) | | ( KDT < children Node Value [k+1].Key Index))
12. return Search B Tree Wild Card Fuzzy Index (children Node Value[k] .Next Node, KDT, Tree Height -1);
13. end if
14. end for // k loop
15. return NULL
16. end if
17. End Search B Tree Fuzzy Searchable Index (BTSI_E, KDT)

After receiving the search path pattern, the Cloud Storage Server sends the search path to the Storage Cluster Controller. The Storage Cluster Controller performs Dynamic Parallel Access by executing the Algorithm 11.

**Algorithm 11: Dynamic Parallel Access (SRP)**

Input : SRP : Search Path Pattern of KDT

1. Extract the file Size from the search path SRP;
2. Declare the variable int Search File Size,Access Size;
3. Search hFile Size = file Size;
4. Access Size = Search File Size / 3;
5. for i = 1 to AccessSize do // Downloading the first part of the file
6. Extract the file content from Replica Server 1;
7. end for
8. for i = AccessSize + 1 to SearchFileSize – AccessSize do // Download Second part of file
9. Extract the file content from Replica Server 2;
10. end for
11. for i = Search File Size – Access Size to Search File Size do // Download Third part of file
12. Extract the file content from Replica Server 3;
13. end for
14. End Dynamic Parallel Access (SRP)

### 5. Implementation Results

#### 5.1 Implementation Setup

The implementation of the proposed Refined Asymmetric Classifier Fuzzy Keyword Search (RACKS) cloud data utilization service architecture is done using Jelastic PaaS LayerShift Cloud storage provider which offers infrastructure, platform and application as a service for the customers. The experimentation was conceded out with the code programmed in JAVA for Data Owner, Users, Classifier Search Server, Performance Isolated Intertenant Storage Cluster and Cloud Storage Server. Microsoft SQL MYSQL 5.5.42 was facilitated to act as the database for the proposed system. The simulation was performed with the setup of Data Owner, Data Users from our side and Classifier Search Server, Performance Isolated Intertenant Storage Cluster and Cloud Storage Server on the Jelastic Cloud storage. The eight components of Performance Isolated Intertenant Storage Cluster namely Storage SLA Monitor, Tenant Portfolio Regulator, Storage Demand Estimator, Tenant Metadata Organizer, Tenant
Storage Observer, Preemptive Priority Storage Scheduler, Storage Cluster Controller and Replica Servers are also implemented by the code programmed in JAVA. The environment is setup with authorizing 20 Data Owners by making the Service Level Agreement. The data owner is set to authenticate 50 users and predefines the Keyword set for each data file. Before evaluating the results, all the data owners outsourced 500 encrypted files to the Jelastic Cloud storage. The data owner creates the Wild Card Fuzzy Keyword Set for edit = 1 and B Tree Fuzzy Searchable Index BTSI. The data owner outsources the encrypted BTSIE and 500 encrypted files to Jelastic Cloud storage server. The Cloud storage now contains the encrypted 500 files and encrypted BTSIE. With this simulated setup, the authorized users are allowed to access the files in the cloud storage using their individual identity by keyword search request.

5.2 Performance Analysis

Initially Storage SLA Monitor updates SLA Database. The Tenant Portfolio Regulator updates its DiamondDB, GoldDB and SilverDB database. The Tenant Metadata Organizer updates its MetadataDB database. The Tenant Storage Observer starts updating its Consume StorageDB. The Storage Cluster Controller replicates the received the encrypted files to all the three Replica servers. The performance of the proposed method was evaluated by analysing the Search Time Efficiency, Data Utilization Efficiency, Performance Isolation and Availability from the Jelastic cloud storage by giving the keyword search request from Users.

5.2.1 Observation 1: Availability and Data Utilization Efficiency

Table 1 shows the analysis of Availability and Data Utilization Efficiency for Correct Keywords with and without BTree Searchable Index, Storage Cluster Controller and its analysis chart is shown in Figure 4.

The toleration of typos and representation inconsistencies are analyzed by making the performance analysis for misspelled keywords. The Table 2 shows the analysis of Availability and Data Utilization Efficiency for Misspelled Keywords with and without BTree Searchable Index, Storage Cluster Controller and its analysis chart is shown in Figure 5.

5.2.2 Observation 2: Search Time Efficiency

Table 3 shows the analysis of Search time Efficiency for correct Multikeywords with and without Classifier Search server and Storage Cluster Controller and its analysis chart is shown in Figure 6.

Table 4 shows the analysis of Search time Efficiency for misspelled Multikeywords with and without Classifier Search server and Storage Cluster Controller and its analysis chart is shown in Figure 7.

5.2.3 Observation 3: Analysis of Processed Request without Performance Isolation

To analyze the Performance isolation, the cloud storage was setup to process the request with one Abiding Diamond, Gold and Silver User along with one aggressive Diamond, Gold and Silver User. The simulation and analysis of the

| Number of Nodes | Sample Correct Keyword | Number of Files retrieved with BTree Index and Storage Cluster Controller (RACKS) | Number of Files retrieved with BTree Index and without Storage Cluster Controller | Number of Files retrieved without BTree Index and with Storage Cluster Controller |
|-----------------|-------------------------|---------------------------------------------------------------------------------|---------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| 25              | Admission               | 85                                                                              | 85                                                                              | 4                                                                              |
| 30              | Building                | 110                                                                             | 110                                                                             | 6                                                                              |
| 45              | Candidate               | 140                                                                             | 140                                                                             | 8                                                                              |
| 60              | Distance                | 95                                                                              | 95                                                                              | 12                                                                             |
| 75              | Enrolment               | 165                                                                             | 165                                                                             | 7                                                                              |
| 85              | Institute               | 70                                                                              | 70                                                                              | 5                                                                              |
| 100             | Submission              | 95                                                                              | 95                                                                              | 14                                                                             |
number of request processed by the cloud server at every 5 seconds and the readings were taken. Table 5 shows the data without performance isolation. It is observed that the number of request processed by abiding users are highly affected due to the interference of Aggressive users and its analysis chart is shown in Figure 8.

Table 6 shows the analysis of Performance Isolation with Performance Isolated Intertenant Storage Cluster Controller

| Number of Nodes | Sample Correct Keyword | Number of Files retrieved with BTree Index and Storage Cluster Controller (RACKS) | Number of Files retrieved with BTree Index and without Storage Cluster Controller | Number of Files retrieved without BTree Index and with Storage Cluster Controller | Number of Files retrieved without BTree Index and Storage Cluster Controller |
|-----------------|------------------------|----------------------------------------------------------------------------------|----------------------------------------------------------------------------------|----------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| 25              | Admicion               | 95                                                                               | 95                                                                               | 2                                                                                | 2                                                                                |
| 30              | Builtng                | 130                                                                              | 130                                                                              | 3                                                                                | 3                                                                                |
| 45              | Candydate              | 150                                                                              | 150                                                                              | 1                                                                                | 1                                                                                |
| 60              | Distance               | 105                                                                              | 105                                                                              | 4                                                                                | 4                                                                                |
| 75              | Enrolemet              | 175                                                                              | 175                                                                              | 2                                                                                | 2                                                                                |
| 85              | Institute              | 90                                                                               | 90                                                                               | 1                                                                                | 1                                                                                |
| 100             | Submisson              | 125                                                                              | 125                                                                              | 3                                                                                | 3                                                                                |

Figure 4. Performance Analysis of Availability and Data Utilization Efficiency for Correct Keywords.

Table 2. Analysis of Availability and Data Utilization Efficiency of Misspelled Keywords with and without BTree Searchable Index, Storage Cluster Controller

with different types of users. The number of executed request per second varies for each user. When the aggressive user starts its conflicting behavior, the number of request processed for abiding users does not get affected due to the presence of Performance Isolated Intertenant Storage Cluster and its analysis chart is shown in Figure 9.
6. Conclusion

This scheme Refined Asymmetric Classifier Fuzzy Keyword Search can be effectively used to enhance data utilization and improved search efficiency with Performance Isolation and high availability for public cloud storage. With the set of encrypted files outsourced to cloud storage, we illustrated that the data users experienced...
improved search time due to the presence of classifier search server and Performance Isolated Intertenant Storage Cluster by performing search in the B Tree Wild card Fuzzy Searchable index. The proposed system dig up the Wild Card Fuzzy Keyword set for the search request resulting in increased data utilization in terms of number of files retrieved and availability is increased due to the replication of files in Storage Cluster Controller.

**Figure 7.** Performance Analysis of Search Time Efficiency for Misspelled Keywords.

**Table 5.** Scenario for Analysis with no Performance Isolation

| Time in Seconds | Number of Processed Request without Performance Isolated Intertenant Storage Cluster |
|-----------------|--------------------------------------------------------------------------------------|
|                 | Abiding User 1 (Diamond) | Abiding User 2 (Gold) | Abiding User 3 (Silver) | Aggressive Diamond User | Aggressive Gold User | Aggressive Silver User |
| 5               | 6                         | 9                     | 11                     | 45                       | 37                     | 35                      |
| 10              | 12                        | 15                    | 16                     | 53                       | 58                     | 48                      |
| 15              | 14                        | 16                    | 12                     | 65                       | 50                     | 40                      |
| 20              | 19                        | 20                    | 14                     | 48                       | 45                     | 29                      |
| 25              | 6                         | 5                     | 7                      | 57                       | 49                     | 47                      |
| 30              | 4                         | 3                     | 4                      | 45                       | 51                     | 36                      |
| 35              | 5                         | 4                     | 5                      | 67                       | 59                     | 55                      |
| 40              | 3                         | 2                     | 3                      | 44                       | 39                     | 30                      |
| 45              | 2                         | 3                     | 2                      | 59                       | 45                     | 40                      |
| 50              | 4                         | 2                     | 5                      | 54                       | 52                     | 48                      |
Figure 8. Number of Executed Request without Performance Isolation with Aggressive Users.

Table 6. Analysis of Performance Isolation with Aggressive Users

| Time in Seconds | Abiding User 1 (Diamond) | Abiding User 2 (Gold) | Abiding User 3 (Silver) | Aggressive Diamond User | Aggressive Gold User | Aggressive Silver User |
|-----------------|--------------------------|-----------------------|-------------------------|-------------------------|----------------------|------------------------|
| 5               | 16                       | 18                    | 12                      | 13                      | 14                   | 10                     |
| 10              | 14                       | 12                    | 17                      | 19                      | 18                   | 15                     |
| 15              | 18                       | 16                    | 13                      | 9                       | 14                   | 19                     |
| 20              | 22                       | 20                    | 14                      | 12                      | 10                   | 16                     |
| 25              | 17                       | 15                    | 20                      | 19                      | 12                   | 14                     |
| 30              | 19                       | 21                    | 24                      | 17                      | 14                   | 19                     |
| 35              | 20                       | 24                    | 20                      | 22                      | 18                   | 22                     |
| 40              | 18                       | 15                    | 18                      | 14                      | 20                   | 12                     |
| 45              | 14                       | 13                    | 22                      | 20                      | 15                   | 18                     |
| 50              | 20                       | 22                    | 15                      | 18                      | 12                   | 9                      |
7. References

1. Geelan J. Twenty one experts define cloud computing virtualization. Electronic Magazine. 2008; Available from: http://virtualization.sys-con.com/node/612375
2. Foster I, Zhao Y, Raicu I, Lu S. Cloud computing and grid computing 360-degree compared. Grid Computing Environments Workshop, GCE’08; Austin, TX. 2008. 1-10.
3. Curtmola R, Garay J, Kamara S, Ostrovsky R. Searchable symmetric encryption: Improved definitions and efficient constructions. Proceedings of the 13th ACM Conference on Computer Security; 2006. p. 79–88.
4. Li J, Wang Q, Wang C, Cao N. Fuzzy keyword search over encrypted data in cloud computing. Proceedings of the IEEE INFOCOM; San Diego, CA. 2010. p. 441–45.
5. Sarga L. Cloud computing: An overview. Journal of Systems Integration. 2012; 3(4):3-14.
6. Uddin M, Memon J, Alsaqour R, Shah A, Rozan MZA. Mobile agent based multi-layer security framework for cloud data centers. Indian Journal of Science and Technology. 2015; 8(12).
7. Sugumar R, Imam SBS. Symmetric encryption algorithm to secure outsourced data in public cloud storage. Indian Journal of Science and Technology. 2015; 8(23).
8. Liu Q, Wangyz G, Wu J. Secure and privacy preserving keyword searching for cloud storage services. Journal of Network and Computer Applications. 2011; 35(3):927-33.
9. Ren K, Wang C, Wang Q. Towards secure and effective data utilization in public cloud. IEEE Transactions on Network. 2012; 26(6):69-74.
10. Zhou W, Liu L, Jing H, Zhang C, Yao S. K-Gram based fuzzy keyword search over encrypted cloud computing. Journal of Software Engineering and Applications, Scientific Research. 2013; 6(1):4.
11. Wang J, Ma H, Tang Q, Li J, Zhu H, Ma S, Chen X. Efficient verifiable fuzzy keyword search over encrypted data in cloud computing. Journal of Computer Science and Information System. 2013; 10(2):667-84.
12. Xu P, Jin H, Wu Q, Wang W. Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack. IEEE Transactions on Computers. 2013; 62(11):2266-77.
13. Munisamy SD, Chokkalingam A. Universal keyword classifier on public key based encrypted multikeyword fuzzy search in public cloud. The Scientific World Journal. 2015; 2015(1):17.
14. VMWare: Rethink IT: Getting rid of noisy neighbours. 2010; Available from: http://blogs.vmware.com/
15. Mixpanel—Why we moved off the cloud. 2011; Available from: http://code.mixpanel.com/2011/10/27/why-we-moved-off-the-cloud/
16. Shieh A, Kandula S, Greenberg A, Kim C, Saha B. Sharing the data center network. Proceedings of NSDI Conference, USENIX. 2011; p. 1-14.
17. Li A, Yang X, Kandula S, Zhang M. Cloudcmp: Comparing public cloud providers. Proceedings of the ACM SIGCOMM Conference on Internet Measurement IMCC; 2010. p. 1-14.
18. Wang G, Ng TSE. The impact of virtualization on network performance of Amazon EC2 data center. Proceedings of IEEE INFOCOM; San Diego, CA. 2010. p. 1-10.
19. Ristenpart T, Tromer E, Shacham H, Savage S. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. Proceedings of the 16th ACM Conference on Computer and Communication Security; 2009. p. 199-12.
20. Ballani H, Costa P, Karaginnis T, Rowstron A. Towards predictable datacenter networks. Proceedings of the ACM SIGCOMM Conference; 2011. p. 242-53.
21. Guo C, Lu G, Wang HJ, Yang S, Kong C, Sun P, Wu W, Zhang Y. SecondNet: A data center network virtualization architecture with bandwidth guarantees. Proceedings of the 16th International Conference; CoNext. 2010. p. 15.
22. Huang L, Chiueh TC. Stonehenge: A high-performance virtualized IP storage cluster with qos guarantees. NY, USA: University of New York; 2003.
23. Krishnamurthy L. AQUA: An adaptive quality of service architecture for distributed multimedia applications [PhD thesis]. Lexington, Kentucky: Computer Science Department, University of Kentucky; 1996 Jan.
24. Wachs M, Abd-El-Malek M, Thereska E, Ganger GR. Argon: Performance insulation for shared storage servers. Proceedings of FAST 5th USENIX Conference on File and Storage Technologies; 2007. p. 61-76.
25. Duda KJ, Cheriton DR. Borrowed-Virtual Time (BVT) scheduling: Supporting latency-sensitive threads in a general-purpose scheduler. Proceedings of the Seventh ACM Symposium on Operating Systems Principles, SOSP; 1999. p. 261-76.
26. Nieh J, Lam MS. A smart scheduler for multimedia applications. ACM Transactions on Computer Systems. 2003; 21(2):117–163.
27. Gulati A, Merchant A, Varman P. Pclock: An arrival curve based approach for qos guarantees in shared storage systems. Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems; 2007. p. 13-24.
28. Gulati A, Merchant A, Varman P. Mclock: handling throughput variability for hypervisor IO scheduling. Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI’10; 2010. p. 1-7.
29. Rodrigues H, Santos JR, Turner Y, Soares P, Guedes D. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. Proceedings of WIOV Conference; 2011. p. 1-8.