CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 0000; **00**:1–22
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

S3BD: Secure Semantic Search over Encrypted Big Data in the Cloud

Jason W. Woodworth[1]*and Mohsen Amini Salehi[2]*

[2] *High Performance Cloud Computing (HPCC) Laboratory*
[12] *School of Computing and Informatics*
University of Louisiana at Lafayette, Louisiana, USA

SUMMARY

Cloud storage is a widely utilized service for both personal and enterprise demands. However, despite its advantages, many potential users with enormous amounts of sensitive data (big data) refrain from fully utilizing the cloud storage service due to valid concerns about data privacy. An established solution to the cloud data privacy problem is to perform encryption on the client-end. This approach, however, restricts data processing capabilities (*e.g.,* searching over the data). Accordingly, the research problem we investigate is how to enable real-time searching over the encrypted big data in the cloud. In particular, semantic search is of interest to clients dealing with big data. To address this problem, in this research, we develop a system (termed S3BD) for searching big data using cloud services without exposing any data to cloud providers. To keep real-time response on big data, S3BD proactively prunes the search space to a subset of the whole dataset. For that purpose, we propose a method to cluster the encrypted data. An abstract of each cluster is maintained on the client-end to navigate the search operation to appropriate clusters at the search time. Results of experiments, carried out on real-world big datasets, demonstrate that the search operation can be achieved in real-time and is significantly more efficient than other counterparts. In addition, a fully functional prototype of S3BD is made publicly available.

Received . . .

KEY WORDS: Cloud services, Searchable Encryption, Semantic Search.

1. INTRODUCTION

Cloud storage has become an inevitable solution for companies and individuals who desire to store a huge volume of data, known as big data, and relieves them from the burden of maintaining storage and processing infrastructure [1]. However, despite the advantages cloud solutions offer, many potential clients abstain from using them due to valid concerns over data security and privacy on cloud servers [2–4] and in the data transmission process [5, 6]. For example, 73% of banks list data privacy and confidentiality as a reason for not using cloud services, making it the most cited concern [2]. Thus, enhancing cloud privacy and confidentiality for the users' data is of paramount importance.

Cloud storage providers commonly offer security by encrypting user data on their servers and maintaining their encryption keys. However, this approach makes the data prone to attacks, particularly, internal attackers who can have access to the encryption keys [7]. One proven solution that addresses this vulnerability is to perform the encryption on the user's end [8], before it is transferred to the cloud. Unfortunately, this solution limits the user's ability to interact with the data, most notably the ability to search over it. The abilities are further limited when dealing with

*Correspondence to: School of Computing and Informatics, University of Louisiana at Lafayette, Louisiana, USA. Email: jww7675@louisiana.edu, amini@louisiana.edu

big data where performing any possible operation on the encrypted data becomes cost- and time-prohibitive [8].

Our motivation, in this research, is an organization that owns a big data scale dataset containing confidential data. One example of such an organization is a law enforcement agency with encrypted police reports and officers who need to search over the reports with their handheld devices (*e.g.,* smartphones). Users of the organization may not remember exact keywords in the documents they are looking for, or need to retrieve documents semantically related to what they are searching for. For instance, the user searches for "burglary" but is interested in finding documents about "robbery" too. As such, users require the ability of *semantic search* on the encrypted big data. As the users perform the search on their handheld devices with limited processing and storage capabilities, any solution for them should not impose a major processing or storage overhead. Ideally, the users need a transparent system that enables them to only enter search queries in plain text and retrieve documents in *real-time* and *ranked* in order of semantic relevance. Finally, any solution should not reveal any sensitive data to internal or external attackers.

Although solutions for searching over encrypted data exist, they often do not consider the semantic meaning of the user's query. That is, they only consider the keywords entered by the user (*e.g.,* [9]), and not the terms semantically related to the user's in the query. Other solutions do not rank documents based on their relevance to the query, imposing additional search time for the user to look through results themselves. Many solutions impose a large processing and memory overhead (*e.g.,* [10]), making the search service costly on cloud and possibly non-real-time.

In this research, we offer a solution for providing semantic search over encrypted big data in real-time and with low overhead using cloud services. In our solution, each document is parsed before uploading to extract key phrases that represent the document's semantic. The key phrases are then encrypted and stored in an index structure on the cloud for search processing. Semantic information is injected into the query at search-time to be searched in the index.

Previous similar solutions are based on using a potentially huge central index, which is fully traversed for each search query (*e.g.,* [11]). Undoubtedly, these solutions do not scale for big data. A common approach to reduce the impact of a large central index is to evenly partition it into disjoint clusters (*e.g.,* [12]), which facilitates the parallelization of searching. However, this practice is still inefficient, as much of the index content is irrelevant for any individual search query.

Another approach for reducing the impact of a large central index is to fracture the central index into topic-based clusters (also known as shards) [13]. Then, at the search time, a subset of shards are proactively chosen to be searched over. Although this solution substantially reduces the search time and required resources, the remaining problem is *how to generate topic-based shards on encrypted data due to a lack of semantic information*?

More specifically, in this research, we define the problem of providing a secure cloud-based semantic search system for big data as needing to answer the following three questions:

- How to fracture a central encrypted index into topic-based shards without revealing semantic data to the cloud?
- How to narrow the search operation to only shards that are relevant to the user's query, hence, increase the real-timeness of the secure semantic search operation for big data?
- How to rank results of a search based on semantic relevance to the user's query?

In this paper, we present **S**ecure **S**emantic **S**earch over encrypted **B**ig **D**ata in the **C**loud (S3BD) to address the aforementioned questions. The core of S3BD is based on extracting and encrypting semantic key phrases from provided documents and clustering them into topic-based shards. To provide a real-time response to a provided search query, S3BD proactively determines the shards relevant to the query at search time and limits the search only to those shards. S3BD extracts key phrases and performs data encryption only at the user-end, thus, the cloud and outside world can see nothing about the plaintext data.

In summary, S3BD improves upon previous work in the literature by limiting the impact of a large encrypted central index created when using big data. Specifically, the contributions of this paper are as follows:

- Developing a secure, scalable, and space-efficient system for semantic searching over big data in the cloud.

DOI: 10.1002/cpe

- Proposing a novel application of k-means clustering to an encrypted central search index to create topic-based clusters without using explicit semantic data.
- Proposing a novel method for pruning a large number of shards into a small number of those most relevant to a search query through semantic comparison of the query to small samples of each shard.
- Providing a method for ranking search results based on their semantic relevance to the query without exposing any semantic information to the cloud.
- Evaluating and analyzing the performance, scalability, overhead, and accuracy of S3BD when compared with previous works in the literature.

A prototype of S3BD has been implemented and is made available to the public[†] used for performance evaluations. The evaluations conducted on real-world datasets demonstrate the practicality of S3BD for big data.

The rest of the paper is organized as follows. Section 2 reviews related works in the literature, establishing the need for our solution. Section 3 gives an overview of our proposed system architecture and explains the upload, cluster, and search processes that define S3BD. Section 4 reviews the threat model we are working with and provides a security analysis of our solution. Section 5 presents the results of our evaluations using real-world datasets. Finally, section 6 concludes the paper.

2. RELATED WORK

We provide a review on research works undertaken in the four fields most related to this work and position the contribution of our works against them. Specifically, these fields are searchable encryption, semantic searching, semantic searching over encrypted data, and clustering methods for searching.

2.1. Searchable Encryption

Solutions for searchable encryption (SE) are imperative for privacy preservation on the cloud. The majority of SE solutions follow one of two main approaches, the first of which being to use cyryptographic algorithms to search the encrypted text directly. This approach is generally chosen because it is provably secure and requires no storage overhead on the server, but solutions utilizing this method are generally slower [8], especially when operating on large storage blocks with large files. This approach was pioneered by Song *et al.* [8], in which each word in the document is encrypted independently and the documents are sequentially scanned while searching for tokens that match the similarly encrypted query. Boneh *et al.* produced a similar system in [14] which utilized public key encryption to write searchable encrypted text to a server from any outside source, but could only be searched over by using a private key. While methods following this approach are secure, they often only support equality comparison to the queries, meaning they simply return a list of files containing the query terms without ranking.

The second major approach is to utilize database and text retrieval techniques such as indexing to store selected data per document in a separate data structure from the files, making the search operation generally quicker and well adapted to big data scenarios. Goh [15] proposed an approach using bloom filters which created a searchable index for each file containing trapdoors of all unique terms, but had the side effect of returning false positives due to the choice of data structure. Curtmola *et al.* [11] worked off of this approach, keeping a single hash table index for all documents, getting rid of false positives introduced by bloom filters. The hash table index for all documents contained entries where a trapdoor of a word which appeared in the document collection is mapped to a set of file identifiers for the documents in which it appeared. Van Liesdonk *et al.* further expanded on this in [16] with a more efficient search by using an array of bits where each bit is either 0 or its position represents one of the document identifiers. These methods are generally faster, taking constant time to access related files, but are less provably secure, opening up new amounts of data to potential

[†]The prototype can be obtained from http://hpcclab.org/products/S3BDJars.zip

DOI: 10.1002/cpe

threat. All of the mentioned methods only offer an exact-keyword search, leaving no room for user error through typos and cannot retrieve works related to terms in the query.

2.2. *Semantic Search*

Much of the work into searching semantically has been done in the context of searching the web [17–19]. Some of these works, such as RQL by Karvounarakis [20], require users to formulate queries using some formal language or form, which leads to very precise searching that is inappropriate for naïve or everyday users. Others [21, 22] aim for a completely user-transparent solution where the user needs only to write a simple query with possible tags, while others still [23, 24] aim for a hybrid approach in which the system may ask a user for clarification on the meaning of their query.

All of these methods use some form of query modification coupled with an ontology structure for defining related terms to achieve their semantic nature. In addition, these ontology structures often need to be large and custom-tailored to their specific use cases or domain, making them very domain-dependent and unadaptable to different areas. Surprisingly, few of the works in this field offer a ranking of results, instead having the user choose from a potentially large pool of related documents.

2.3. *Semantic Search over Encrypted Data*

Few works at the time of writing have combined the ideas of semantic searching and searchable encryption. Works that attempt to provide a semantic search often only consider word similarity instead of true semantics.

Li *et al.* proposed in [25] a system which could handle minor user typos through a fuzzy keyword search. Wang *et al.* [26] used a similar approach to find matches for similar keywords to the user's query by using edit distance as a similarity metric, allowing for words with similar structures and minor spelling differences to be matched. Amini *et al.* presented in [27, 28] a system for searching for regular expressions, though this still neglects true semantics for another form of similarity. Moataz *et al.* [29] used various stemming methods on terms in the index and query to provide more general matching. Sun *et al.* [10] presented a system which used an indexing method over encrypted file metadata and data mining techniques to capture semantics of queries. This approach, however, builds a semantic network only using the documents that are given to the set and only considers words that are likely to co-occur as semantically related, leaving out many possible synonyms or categorically related terms.

2.4. *Clustering Methods for Searching*

The clustering hypothesis states that "Closely associated documents tend to be relevant to the same requests" [30]. This idea has been expanded upon in many ways to form the body of research that investigates document clustering and its effects in information retrieval and searching. Clustering has largely been used in two main ways: partitioning the central index into static shards, independent of user search queries, and clustering in a query specific manner based on the results of searches with the query [31]. Solutions following the latter approach have the potential to outperform the static clustering approach [32], they are largely impractical for large data sets.

The former approach has been studied extensively, especially in the domain of web searching [12,33], but these systems still demand a high computational cost to search over big data. Relatively few works have specifically focused on the idea of clustering the central index into shards based on topics. This idea was pioneered by Xu and Croft [13], who showed that making shards of a dataset's index more homogeneous (i.e. the contents of the shards are based around the same topic) improved the effectiveness of a system over standard distributed information retrieval. They used the k-means clustering algorithm with a KL-divergence distance metric to create the shards, then determine which shard should be searched by a query by estimating the likelihood that the query would come from the shard's language model. Liu and Croft [31] expanded upon this by using more updated language modeling techniques to better smoothen their estimations. However, neither of these works were appropriate for large scale data.

Kulkarni *et al.* [34] adapted these methods to larger scale datasets by performing the k-means clustering on a smaller sample of the dataset, then inferring from the documents' language models

DOI: 10.1002/cpe

which shard those not included in the sample would belong to. These works differ from ours in that they are only designed to operate on plaintext datasets. Before this work, there was no attempt to create a topic-based clustering system that would operate on secured encrypted datasets. Additionally, these models perform clustering on documents, whereas our work is designed to cluster terms from the documents, which was more effective given our encrypted approach.

3. ARCHITECTURE AND PROCESSES OF S3BD

S3BD has three primary architectural components: the *Client Application*; *Cloud Processing Server*; and *Cloud Storage*. Within those components, the system supports three major processes, namely *uploading documents*; *clustering on the encrypted index*; and *semantic search*. In this section, we first elaborate on the architectural components of S3BD, then explain the major processes.

3.1. Overview of S3BD Architecture

Figure 1 presents an overview of the components and processes in S3BD. In this figure, Client Application is a lightweight program hosted on the user's device and is the only component in the system deemed to be trusted. Cloud Processing Server and Cloud Storage are maintained by a third party cloud provider, thus, considered "honest but curious". Our threat model assumes cloud components and the network channels are prone to external and internal attacks.

The components in the architecture are described as follows:

- The *Client Application* provides a user interface for uploading documents or to search over them in the cloud. It is also responsible for parsing and extracting information from plaintext documents and encrypting them before they are uploaded.
  When the user requests to search, the system expands the search query with semantic data and transforms it to the secure query set (termed as a *trapdoor*). The trapdoor is used for the search process on the cloud. The user then receives a ranked list of documents that can be downloaded and decrypted upon request. Client Application is also responsible for pre-processing queries to enable proactive searching on a subset of big data and achieve real-time search operation.
- The *Cloud Processing Server* is responsible for constructing and updating the index and other related data structures during the upload process using encrypted data sent by the Client Application. Once the central encrypted index is built, it is clustered into shards to make search scalable for big data. As the clustering process is time consuming, it is performed in an offline manner as the dataset grows.
  Cloud Processing Server is also involved in the search process. It receives the user's search query and loads the relevant shards into memory. The shards are then searched to find and rank relevant documents. The highly-ranked documents are retrieved from the Cloud Storage and sent to the user.
- The *Cloud Storage* component is used to store the uploaded encrypted documents. Therefore, it does not see any representation of the user's query. Upon request by the Cloud Processing Server, the Cloud Storage can locate the documents and provide it to the user[‡].

Finally, it is important to note that each component exists *per-user*. That is to say, while each Cloud Processing Server and Cloud Storage instance may exist on a single machine, components which hold data are assumed to exist separately for each user (e.g. each law enforcement agency). Thus, a separate central index and set of shards is held for each user. This is done to avoid bloating search times with operations to separate different users' files, and to avoid having users collude to attack other users.

[‡]Currently, our Cloud Storage relies on a single cloud. However, the architecture can potentially utilize multiple clouds for storage so long as the location of each document is provided.

DOI: 10.1002/cpe

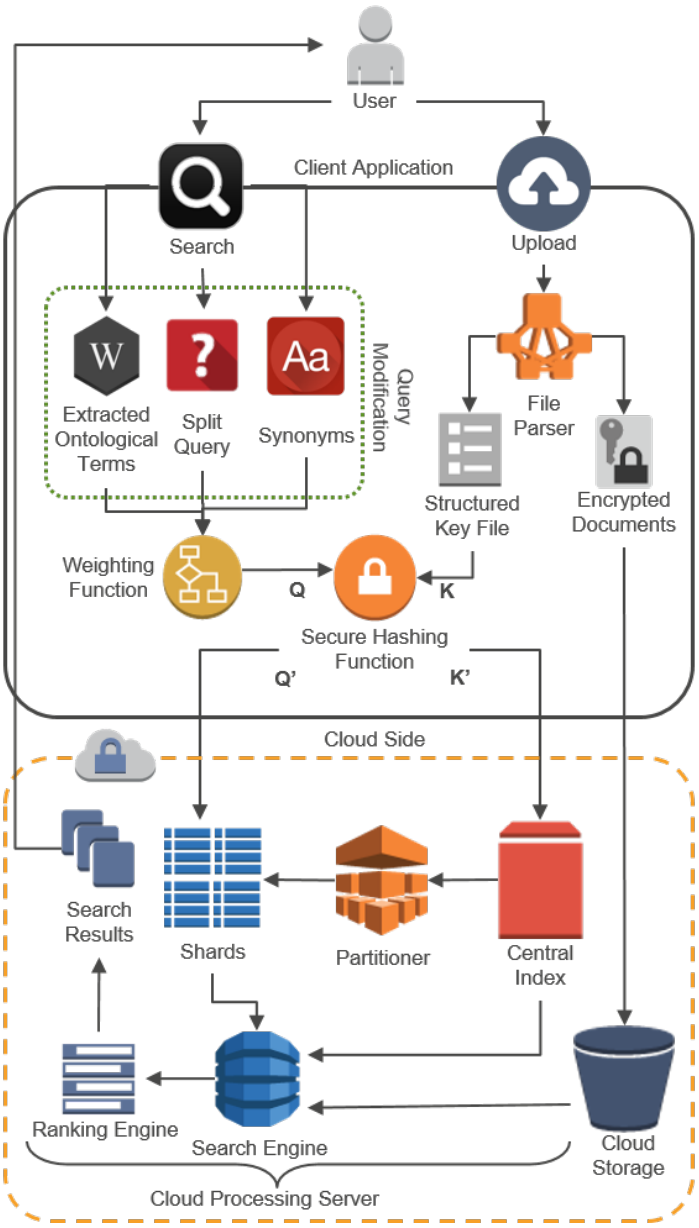

Figure (1). Overview of the S3BD architecture and processes. Parts within the solid-line indicate components or processes at the user-end, deemed trusted. Parts in the dashed-line indicate those on the cloud-end, deemed untrusted.

3.2. Upload and Parsing Process

Upon user request to upload a new document to cloud, the parsing process extracts a subset of the terms and phrases in a document (called *keywords*) to represent the semantics of that document. To preserve security, the keywords are encrypted along with the document before uploading to the cloud. The encrypted keywords are used to create (or update) the encrypted index structure on the cloud.

A naïve approach to extract keywords is to select all terms from a document excluding stopwords. However, previous works (*e.g.,* [35]) show that selecting few keywords that are semantically related to the document heavily reduces the storage overhead while still producing relevant search results. More specifically, the advantages of extracting a subset of keywords are three-fold. First, it maintains a low storage overhead for the central index. Second, assuming the central index is appropriately

DOI: 10.1002/cpe

structured (*e.g.,* using hash tables [11]), it makes the time complexity of updating the index with new documents nearly constant [11]. Third, it increases the security of the central index by exposing fewer keywords to the external world. Thus, we use a key phrase extractor algorithm [36] to extract a number of keywords that represent the document's semantics. The composite (*i.e.,* multi-phrase) extracted keywords are split into individual distinct terms. This ensures that the central index contains encrypted versions of both the composite keyword and its components.

Once keywords for a document are extracted, the frequency of their occurrences within the document is collected. Then, the extracted keywords are deterministically encrypted [37]. Deterministic encryption is a method that always transforms a value (keyword) into the same encrypted token, similar to hashing. In our implementation, we use the RSA deterministic encryption algorithm [38] for this. Individual users who want to search the same dataset (e.g. law enforcement officers in a single agency) share RSA key pairs.

We use this method for the central index structure to allow for matching to encrypted query terms in the index, an integral part of the search process. It is worth noting that the frequency of keywords are maintained in plaintext in the central index. The use of homomorphic encryption [39] on the frequency data was considered, but the system needs to perform many operations on them and current implementations of fully homomorphic cryptosystems are too slow [40] to achieve our desired real-time response rate. Finally, the extracted keywords and their frequencies are integrated in a key file before uploaded to the cloud.

When the encrypted document and key file are received by the cloud server, the document is sent to the cloud storage block, while the key file information is added to the index. The cental index is stored as a mapping of encrypted terms to document IDs and the frequency at which they appeared in those documents.

3.3. Topic-based Clustering Process

S3BD alleviates the search over the central encrypted index by clustering its terms into semantically related shards. In this research, we term this process as *topic-based clustering*. The challenge is how to perform this type of clustering on the encrypted terms, because their meaning is lost due to encryption. One approach to overcome this challenge is to cluster terms based on their co-occurrences in documents, known as *statistical semantics*. To achieve this, we adapt k-means clustering algorithm [41] to cluster encrypted data at the keyword level.

K-means clustering algorithm allows us to cluster terms as long as the distance between two terms can be formulated. Distance between two keywords is defined as the semantic relatedness between them [13]. To adapt k-means for the encrypted data, we need to define the two main operations, namely picking initial means (also known as centroids) of clusters; and computing distance between the encrypted terms. These operations are discussed in the next subsections.

Once the clusters are built, they are used against search queries. To make the search scalable for big data, we define *pruning* as to proactively search clusters that are semantically relevant to the query (*i.e.,* pruning irrelevant clusters). However, because the clusters are encrypted and the semantics are lost, the pruning cannot be achieved on the cloud. For that purpose, in this section, we also develop a method, termed *abstraction*, to sample the clusters into small abstracts that can be sent to and decrypted on the client-end. Upon search request, the abstracts are used on the client-end to prune the cluster and determine which clusters to be searched on the cloud.

3.3.1. Initializing Centroids The first step to partition the central index into clusters with the K-Means clustering algorithm is to pick initial shard *centers* (also known as centroids) to form the shards around them. A centroid is an entry in the central index which is essentially an encrypted keyword plus the list of documents it appears in.

For effective search pruning, the shards should be distributed as evenly as possible, while maintaining semantic relationship. For that purpose, the centroids should represent diverse sections of the dataset. In fact, the initialization of centroids significantly impacts the resulting shards.

A naïve method for initializing centroids is to simply pick a number of centroids equal to the number of desired shards (k) randomly from the central index. This method can potentially result in keywords with very few associated documents being chosen as centroids. Because the co-occurance of keywords and centroids in documents determines the distance between them, the naïve method can potentially lead to formation of small shards (*i.e.,* shards with few elements).

DOI: 10.1002/cpe

To avoid creating small shards (and uneven clustering), we propose a second method that ensures centroids have enough associated documents to attract other keywords. Our method for initializing centroids is to sort the keywords in the central index based on the number of their associated files, then choose the top k terms as centroids. This ensures that keywords with low association are not chosen. However, this method can potentially lead to picking centroids with a high overlap of associated documents that can again cause uneven distribution of shards. Thus, ensuring centroids have a diverse set of associated documents is prioritized.

The method we develop for choosing centroids operates on the sorted index and nominate keywords from the beginning that do not overlap, in their associated documents, with previously nominated keywords. The algorithm to build centroids is mentioned in Algorithm 1. The algorithm receives the number of clusters, denoted k, and the sorted central index structure as input parameters and determines the set of centroids and their associated documents, denoted U, as output.

Algorithm 1: Nominating Keywords as Centroids

```
   Input  : k and central index (with terms sorted by number of associated files)
   Output: centroids and U
 1 Procedure NominateCentroids(k)
 2     U ← ∅
 3     centroids ← ∅
 4     foreach ω ∈ central index do
 5         Ω_ω ← MeasureUniqueness(ω)
 6         if Ω_ω ≥ 1 then
 7             // Nominate keyword to be a centroid
 8             centroids.add(ω)
 9             U ← U ∪ I_ω
10         end
11         if centroids.count ≥ k then
12             return centroids
13             return U
14         end
15     end
16 Procedure MeasureUniqueness(ω)
17     unique ← 0
18     duplicate ← 0
19     foreach documentID ∈ I_ω do
20         if documentID ∈ U then
21             duplicate ← duplicate + 1
22         end
23         else
24             unique ← unique + 1
25         end
26     end
27     if duplicate > 0 then
28         Ω_ω ← unique ÷ duplicate
29     end
30     else
31         Ω_ω ← 0
32     end
33     return Ω_ω
```

For each keyword in the central index, we need to measure its uniqueness (Line 5 in Algorithm 1). For that purpose, we develop a method to measure *uniqueness* of a given keyword in the index structure (see Lines 16 to 33). Let ω a keyword and I_ω the set of documents associated with ω in

DOI: 10.1002/cpe

the central index. Also, let U the set of documents current centroids have appeared in. Then, we define *uniqueness*, denoted Ω_ω, based on Equation 1. Uniqueness is also calculated in Line 28 of Algorithm 1.

$$\Omega_\omega = \frac{|I_\omega \setminus U|}{|I_\omega \cap U|} \tag{1}$$

In order to choose ω as a centroid, the number of documents unique to I_ω must be more than the number of documents in the intersection of I_ω and U (*i.e.,* $\Omega_\omega \geq 1$, as indicated in Line 6). Upon choosing a keyword to be a centroid, the keyword and its associated documents are added to the set of current centroids (Lines 8 and 9 in Algorithm 1).

3.3.2. Computing Distance between a Keyword and a Centroid Once k centroids are chosen, the distance from centroids to keywords in the central index needs to be calculated. With plaintext data, calculating distance is possible using techniques such as semantic graph [31]. However, this technique is impossible when encrypted data are used.

The clustering hypothesis states that keywords (also called terms) which co-occur (*e.g.,* in a document) can be considered related [30]. This can be obtained even when terms are encrypted. Accordingly, we consider the co-occurrence of terms in a document as a reasonable metric for their similarity. That is, if two terms appear in the same document, they are considered related.

Recall that I_T denotes the list of documents associated with term T in the central index. Also, let $\theta(T,f)$ number of times (*i.e.,* frequency) term T appears in document f. We define the *contribution* of file f to term T, denoted $\kappa(f,T)$, as the ratio of $\theta(T,f)$ to the total number of times term T appears in the dataset (*i.e.,* frequency count of term T). Equation 2 shows the formal representation of contribution for file f.

$$\kappa(f,T) = \frac{\theta(T,f)}{\sum_{j \in I_T} \theta(T,j)} \tag{2}$$

Let γ_i be centroid of cluster i. We define *contribution of term T and file f to cluster i* (denoted $K(T,f,\gamma_i)$) as the ratio of sum of the frequency of T in file f and in γ_i to the sum of frequency count of T and γ_i. Equation 3 shows the formal definition of $K(T,f,\gamma_i)$.

$$K(T,f,\gamma_i) = \frac{\theta(T,f) + \theta(\gamma_i,f)}{\sum_{j \in I_T} \theta(T,j) + \sum_{p \in I_{\gamma_i}} \theta(\gamma_i,p)} \tag{3}$$

Then, we define *cooccurence* of term T and γ_i through file f, denoted $\rho(T,\gamma_i,f)$, as the ratio of $\kappa(f,T)$ to $K(T,f,\gamma_i)$. Equation 4 shows the formal definition of cooccurence.

$$\rho(T,\gamma_i,f) = \frac{\kappa(f,T)}{K(T,f,\gamma_i)} \tag{4}$$

To represent the similarity between term T and centroid γ_i, we compute the distance, denoted $d(\gamma_i,T)$, based on Equation 5. In this equation, we iterate through the list of documents that are associated with the term. For each document, we consider the contribution of that document to term T and the cooccurence of T and γ_i through f. We use logarithm to limit the impact of the cooccurence factor.

$$d(\gamma_i,T) = \sum_{f \in I_T} \kappa(f,T) \cdot \log_{10}(\rho(T,\gamma_i,f)) \tag{5}$$

3.3.3. Evening Shards Sizes Once the similarity between each centroid and each term is computed, terms can be distributed to their proper shards. An initial approach for distribution is to assign each term to the shard with the centroid that the term has the maximum similarity with. This approach, however, can potentially lead to uneven shard distribution and subsequently inefficient search pruning.

To avoid uneven clustering, we limit the growth of each shard so that it can only hold up to a certain amount of its closest terms. Ideally, all clusters should end with an equal size of $\frac{|I|}{k}$ in which

DOI: 10.1002/cpe

$|I|$ is the total number of terms in the central index. Accordingly, we constrain the growth of each shard to $\frac{\alpha.|I|}{k}$. Parameter α is determined to be greater than 1 (*i.e.,* $\alpha > 1$) to cover the dynamism of a natural language but does not allow a shard (*i.e.,* topic) to dominate the clustering. In our implementation, we considered $\alpha = 2$.

In a circumstance that a shard reaches its threshold, we disassociate the furthest term from the shard's centroid. Then, we assign it to the closest shard that has not yet reached to its threshold.

Once the shards are initialized with the terms, based on the afore-mentioned approach, we iteratively reorganize them to produce shards that are more centered around a topic. We define the *average term* of shard π as the closest term to the average distance of all terms from the current centroid of shard π. In each iteration, the clustering process of S3BD calculates the average term in each shard, chooses it as the new centroid, and forms a new shard around it.

Ideally, the iteration would continue until the shards' composition stabilizes. That is, when there is no alteration of terms during an iteration. However, in practice, as the iterative clustering is a computationally expensive operation, we limit the iterations until shards are minimally altered. In our implementation, we realized that we generally reach to the stable state in five iterations.

3.3.4. Shard Abstraction Because the shards on the cloud processing server are all encrypted, it is impractical to identify shards related to search query and perform pruning on the cloud. Thus, we need a method to identify appropriate shards to search over, for a given user query. We propose abstracting the shards into tiny unencrypted samples that are processed on the client premises. These abstracts are used against search queries to navigate search to only shards contain relevant search results.

As a centroid shows centrality of a shard, it is more indicative of the shard's general topic, hence, can be used to form the abstract. However, each centroid is only a list of documents and cannot directly be used in abstracts. Therefore, the system chooses terms from the documents associated with a centroid to build that shard's abstract. In particular, it chooses the most frequent term from each associated document of a centroid.

Each abstract, which is a small set of encrypted terms, is sent to the client machine. The abstracts are decrypted on the client machine and compared to the search query terms through a semantic similarity metric [42]. Although abstracts are small, using semantic similarity of search query to abstract terms enable identifying most relevant shards. Then, the search query is only compared against those identified shards in the cloud.

3.4. Search Process

The search process consists of three main phases: *abstract comparison*; *query modification*; and *searching and ranking*. The Client Application is responsible for the first two phases, while the third happens on the Cloud Processing Server.

In summary, S3BD first performs pruning by comparing the query against the abstracts to determine the shards that need to be searched. This information is then sent to the Cloud Processing Server to load the appropriate shards into the memory as soon as possible. Meanwhile, the client application semantically expands and the search query, encrypts it, and sends it to the cloud processing server.

Once the Cloud Processing Server has all of the necessary information, it finds and then ranks relevant documents from the shards specified in the abstract comparison phase. The result list is then sent back to the user. The document selected by the user is downloaded and decrypted on the client machine.

3.4.1. Comparing Queries Against Abstracts As the terms in the abstracts are semantically linked to the topics in corresponding shards, comparing the query to the abstracts let S3BD detect which shards are appropriate to search. This comparison is carried out, in the first phase of search, by obtaining semantic distance of the terms in the query to the terms in the abstracts using the WuPalmer word similarity metric [42]. For this purpose, WuPalmer computes the semantic similarity between two words by evaluating the distance from one word to the other in a large semantic graph, returning a normalized value between 0 and 1.

DOI: 10.1002/cpe

Using WuPalmer to compare the query against all abstracts allows the system to rank the abstracts based on relevance to the query. As the abstracts represent the shards, the ranking can identify relevant shards to be searched.

The number of shards chosen determines the trade-off between search time overhead and search comprehensiveness that can be decided based the user discretion. In fact, choosing a higher number of shards consumes more memory and increases the search time, conversely, a low number of shards ignores searching less relevant parts of the dataset that can include desirable results.

Once a sufficient number of shards have been chosen, the cloud processing server is notified and begins loading those shards into memory. The loading time is overlapped with the Query Modification step (explained in the next section) at the client end.

3.4.2. Query Modification Query modification is meant to inject semantic information into the query. This phase starts with the user entering a plaintext query, denoted as q, into the Client Application, after which it goes through three steps: *query parsing, semantic expansion,* and *weighting*. These steps result in forming a *query set*, hereafter noted as Q.

The goal of query parsing is to refine the search query and split it into smaller tokens or sub-phrases. To refine q, we first remove all stop words [43] (*e.g.,* articles and prepositions). If the query is multi-phrase, we then split it into parts. The reason for splitting is twofold. *First,* because some documents may partially match with the query. *Second*, because portions of a query cannot be derived from the encrypted query. Hence, we split q and create all tokens and sub-phrases of it before encryption. Once this step is complete, Q consists of q, its split parts, and its sub-phrases.

The goal of semantic expansion is to add terms related to the query into the query set, thus, enabling S3BD to search for semantically related results. In order to achieve this, S3BD injects semantic data extracted from an ontological network [44].

One approach to extract semantic data is to perform a synonym lookup (*e.g.,* through an online thesaurus) for each member of Q (termed Q_i) and add the results to Q. However, this approach alone does not produce concepts that are semantically related to the user's query, but are not synonymous. To cope with this problem, S3BD needs to pull related terms from conceptual ontological networks [44].

For the development of S3BD, the elements of Q are used to pull entries from Wikipedia, as an instance of an onthological network. Keyphrase extraction is performed on the entries to get conceptually related terms and phrases, which are added to Q.

In the search results, documents that include phrases exactly matching query terms are deemed more relevant. For that purpose, in the weighting step, we assign weights, ranging from 0 to 1, to the elements of Q as follows:

- As the documents that include the whole originial query have top priority, we assign the maximum weight of 1 to q.
- Documents that include parts of the query (*i.e.,* Q_i) are more relevant than those including terms derived from the query. As such, we assign $1/n$ weight to results of the query parsing where n is the number of parts in the search query.
- Documents including related terms derived from the query have the lowest relevance. Accordingly, the related terms obtained for each Q_i should be assigned the lowest weight. Let $W(Q_i)$ be the weight of Q_i. Then, terms derived from Q_i are weighted as $W(Q_i)/m$ where m is the number of derived terms from Q_i.

3.4.3. Searching and Ranking Once the query set Q is built, its elements are deterministically encrypted [37] to create the trapdoor Q'. The trapdoor is then sent to cloud processing server to perform the search and ranking of the result set. On the cloud processing server, each element of Q' is checked against a union of the loaded shards, denoted Π, to compile a collection of documents, denoted C, that are potentially related to the query. Once C is compiled, the documents are ranked and then sent to the user.

The search operation has to be agnostic about semantics of the query and the dataset. Okapi BM25 [45] is a search and ranking algorithm for unencrypted data that functions in the meta-data level and provides the required data agnosticism. Okapi BM25 operates based on the list of keywords provided to it. However, it cannot differentiate between elements of the query set (Q').

DOI: 10.1002/cpe

We extend the idea of Okapi BM25 to include encrypted data and to consider the weighting of elements in Q'.

Based on Okapi BM25, a document's rank for a given query is considered to be a function of the following three factors:

A. *frequency* of query term Q_i in document d_i, denoted $f(Q_i,d_i)$. In S3BD, $f(Q_i,d_i)$ can be obtained by looking up the encrypted query element Q'_i in Π.

B. *Inverse Document Frequency* (IDF) of query term Q_i across collection of documents (C). Let N be the total number of documents in C, and $n(Q_i)$ be the total number of documents containing Q_i. Then, Equation 6 defines the IDF for Q_i.

$$IDF(Q_i) = \log \frac{N - n(Q_i) + 0.5}{n(Q_i) + 0.5} \tag{6}$$

In Π, we keep the frequency of each term in each document. Therefore, $n(Q_i)$ can be obtained by summing up all the frequencies of Q'_i in Π.

C. *Document Length Normalization* (DLN) that removes the effect disparity in documents' length. Let δ be the average length of all documents in C, and β be a parameter that determines the impact of the DLN factor. Equation 7 formally defines DLN for d_i. In this work, we considered $\beta = 0.75$.

$$DLN(d_i) = (1 - \beta + \beta \cdot \frac{|d_i|}{\delta}) \tag{7}$$

In S3BD, we maintain the length of uploaded documents. As such, we can obtain δ and $|d_i|$ to calculate DLN for d_i.

A rank is defined as the sum of scores given by each Q_i. To consider the weighting scheme of Q in ranking, each score is adjusted by considering the weight of Q_i (denoted $W(Q_i)$). Equation 8 shows the formal representation of rank of document d_i for query set Q (denoted $r(d_i,Q)$). In this equation, $alpha$ is a parameter that determines the impact of the frequency factor. Our initial experiments show that $\alpha = 1.2$ provides an accurate ranking, thus, we use this value in our implementation.

$$r(d_i,Q) = \sum_{i=1}^{n} IDF(Q_i) \cdot \frac{f(Q_i,d_i) \cdot (\alpha + 1)}{f(Q_i,d_i) + \alpha \cdot DLN(d_i)} \cdot W(Q_i) \tag{8}$$

The cloud processing server computes Equation 8 for all encrypted documents in the collection C against Q'. To exploit parallelism implicit in the cloud system, C can potentially be compiled using a mapreduce approach, mapping each shard to a separate process, spreading Π across multiple machines. Because individual documents can be represented across multiple shards, an additional process would need to combine the scores accumulated from different processes for each document. Once C is compiled, its members are ranked in descending order, and a list of document identifiers are sent to the client to be picked.

4. SECURITY ANALYSIS

S3BD provides a trustworthy architecture for storing confidential information securely in clouds while maintaining the ability to search over them. Our threat model can be defined as follows. Our system architecture is divided into three major components that live either in the cloud or on the user's machine (as seen in Figure 1). Only components and processes that exist on the user's machine (i.e. the Client Application) are considered to be trusted, meaning we can store and access plaintext data there. Keeping the user's machine trusted is a reasonable assumption in the real world, as it can be kept with minimal exposure to outside attackers.

Components in the cloud are considered untrusted and susceptible to adversaries. We consider these attackers to be either external (i.e. an unaffiliated party who wishes to learn about the dataset) or internal (i.e. a party with access to the cloud who wishes to see the unencrypted dataset). Our threat model assumes that these adversaries may intend to attack the communication streams between the Client Application and Cloud Processing Server, and between Cloud Processing Server

Prepared using cpeauth.cls
DOI: 10.1002/cpe

and Cloud Storage, as well as the cloud machines themselves. To explain exactly what threats the attackers pose to the encrypted data, we introduce the following definitions:

History: For a multi-phrase query q on a collection of documents C, a history H_q is defined as the tuple (C, q). In other words, this is a history of searches and interactions between client and cloud server.

View: The view is whatever the cloud can actually see during any given interaction between client and server. For our system, this includes the encrypted index and all shards I over the collection C, the trapdoor of the search query terms (including its semantic expansion) Q', the number and length of the files, and the collection of encrypted documents C'. Let $V(H_q)$ be this view.

Trace: The trace is the precise information leaked about H_q. For S3BD, this includes file identifiers associated with the search results of the trapdoor Q' and unencrypted weight information from Q'. It is our goal to allow the attacker to infer as little information about H_q as possible.

Because our threat model assumes a secure user machine, the View and Trace encompass all that the attacker would be able to see. For encryption on the plaintext documents being searched, we use a probabilistic encryption model, considered to be the most secure form of encryption [37]. Hence we can infer that, because probabilistic encryption does not use a one-to-one mapping, C' is not susceptible to dictionary-based attacks [46], and secure so long as the attacker can not access the keys (stored only on the user's machine).

I, in the View, only shows a mapping of a single deterministically encrypted term or phrase to a set of file identifiers with frequencies, meaning a distribution of encrypted terms to files could be compiled, but minimal data could be gained from the construction. Similarly, Q' only shows a listing of encrypted search terms with weights.

The addition of the weights to Q' could potentially enable the attacker to infer which terms in the trapdoor were part of the original query. Even in this case, the attacker can at most get the deterministically encrypted query. Additionally, the expansion of the query to include semantic data adds noise that can mislead attackers from the original user query.

However, we must consider the small possibility that, if the attacker is able to obtain the keys used for deterministic encryption from the user's side, they could in theory build a dictionary of all words in the vocabulary V that the documents are comprised of, mapped to their encrypted counterparts, and reconstruct I in plaintext. In this scenario, the attacker could put together the terms that the documents are comprised of, but since I carries no sense of term order, they could not reconstruct the entire file. Additionally, only a small portion of important terms and phrases from each document are given, meaning the attacker would only be able to ascertain how many times those specific terms and phrases were in the document.

An attacker monitoring the process during a search could see the resultant file identifiers that are associated with the given Q'. This would show an encrypted history as (C', Q'). However, since the attacker would not be able to discern the query (without the use of the above dictionary), this data would be of little use.

Attackers could also potentially attempt to alter data in C'. These attacks, however, could be recognized as the Client Application would not be able to decrypt them.

5. EVALUATION

5.1. Experimental Setup

We have implemented a prototype of S3BD which is available to the general public[§]. The implementation has been the platform for all experiments in this research. All experiments were carried out on Amazon EC2 cloud Virtual Machines (VMs). We ran both client application and cloud processing server on separate Amazon EC2 VM instances. In particular, we used m4.xlarge VM instance type to host client application and i2.xlarge VM instance type to host cloud processing server.

[§]A binary of S3BD can be downloaded from http://hpcclab.org/products/S3BDJars.zip

DOI: 10.1002/cpe

We evaluate two major aspects of S3BD, namely its *performance* and its *accuracy*. Performance specifically refers to the amount of time it takes to perform a search, while accuracy refers to the relevance of its search results.

To evaluate the performance of S3BD with big data, we tested it utilizing portions of the Common Crawl Corpus dataset [47] from Amazon Web Services. The dataset consists of approximately 151 terabytes of text data obtained from an extensive web crawl. We parsed files within the dataset to create sample subsets (termed samples) of varying sizes. Each file in the dataset includes text from multiple web pages, hence, we split those files to create a document for each web page. To generate each sample, we randomly choose files until we reach the desired size for the sample.

We evaluate performance of S3BD by analyzing the time to search over varying numbers of shards in the cloud. The results of this evaluation helps us determining the appropriate number of shards to create.

Once we determine the appropriate number of shards, we compare the search time of S3BD against that of our previous work (S3C) [35], as a baseline. S3C performs a similar style of semantic search without use of clustering and pruning. To assure that the performance results for S3BD is not affected by the temporal performance variations of cloud VMs, we run each query 20 times and report the mean and 95% confidence interval of the results.

To evaluate the relevance of S3BD, we evaluated it using the Request For Comments (RFC) dataset [48]. RFC is a collection of 6298 documents (247 MB) regarding notes on Internet development topics. The reason we choose RFC dataset is that it is domain-specific and small enough to manually verify accuracy of search results. We compare the results of S3BD to the baseline (S3C) and to a version of S3C that operates on non-encrypted data [35].

In addition to these two major aspects, we analyze the storage overhead incurred by storing the central index on the Cloud Processing Server. To that end we show the size of the central index as the dataset increases in size.

It is noteworthy that, as we use two different datasets for our evaluations, we generate two different sets of benchmark queries based on the nature of the datasets. The benchmarks are explained in the respective subsection for each experiment.

5.2. Evaluating Performance of S3BD

5.2.1. Benchmark Queries to Evaluate Performance Perfrormance evaluations are carried out based on 10 benchmark search queries, shown in Figure 2. Queries were chosen after manual analysis of the samples and determining topics of their documents. It is noteworthy that the wording of the benchmark queries does not impact the performance of the system.

```
Government News Report
Social Media Feed
Encryption in Cloud Server
City Travel Blog
Online Shopping Network
Free Recipe Index
Online Weather Service
Celebrity Fashion Catalog
Academic Conference Papers
Sports Broadcasting Network
```

Figure (2). Benchmark queries used for evaluating search performance on the Common Crawl Corpus dataset.

5.2.2. Finding the Appropriate Number of Shards The challenge in S3BD is how to determine the number of shards that should be created to provide the best trade-off between search performance and search accuracy. In fact, creating a lower number of shards potentially implies higher search accuracy, because each shard covers a larger subset of the dataset. However, searching a larger portion of the dataset lowers the performance, because less of the dataset is pruned. On the other hand, creating a high number of shards does the inverse. That is, it improves search performance but potentially lowers accuracy.

To handle the trade-off between the search performance and the search accuracy, in this experiment, we utilize the idea of Pareto front analysis [49] to understand the relation between

DOI: 10.1002/cpe

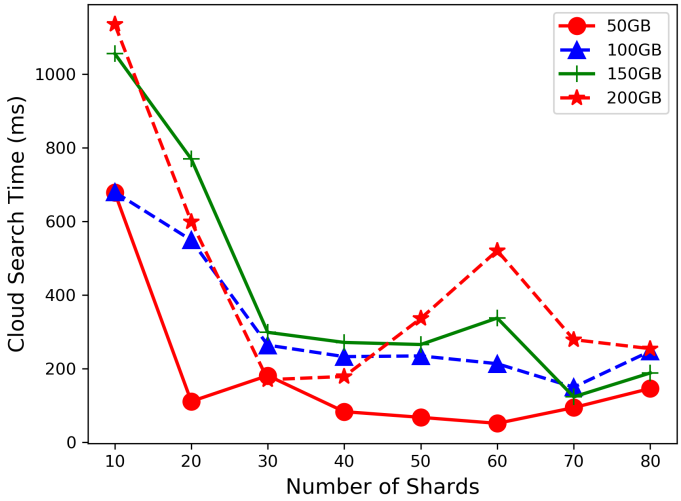

Figure (3). Time taken to search over the union of chosen shards (Π) on the cloud. The horizontal axis shows the number of shards the dataset sample is partitioned into and the vertical axis shows the average search time (in milliseconds). The search time includes time to find documents in Π that match the trapdoor (Q') and rank them. Each data point is the average of searching each benchmark query 20 times.

these factors and find the number of shards that satisfies both objectives at the same time. For that purpose, we compare the performance, in terms of cloud search time, across different numbers of shards, representing accuracy.

The result of the experiment is shown in Figure 3. The vertical axis, in this figure shows the time taken to search on the cloud and the horizontal axis shows the various numbers of shards created. To assure that our analysis is comprehensive and is not bound to a certain dataset size, we conduct the experiment with samples of different sizes — from 50 GB to 200 GB.

As we can see in Figure 3, the time to search decreases as more shards are formed. Important to note is that declines in search time cease being substantial for 100, 150, and 200 GB past 30 shards. On the other hand, creating fewer shards yields remarkably high search times for larger datasets. As this pattern is consistently observed for samples of different sizes, we can conclude that partitioning datasets into 30 shards provides an ideal trade-off between search performance and accuracy of S3BD.

5.2.3. Shard Distribution and Variance Interestingly, the search time does not strictly decrease as more shards are formed. This can be seen primarily in the spike in search time for the 150 and 200 GB samples with 60 shards. Our analysis shows that this is attributed to uneven shard distribution. In fact, the size of a shard plays an important role in determining its search time. For a given query, the system can potentially determine to search among the smallest shards, when few shards are created. Inversely, for the same query with more shards created, it is possible to search among the largest shards, despite the fact that the average shard size is smaller.

We thus determine that maintaining consistent cluster sizes (low variance) is important for maintaining consistent search performance and accuracy. With S3BD, we introduce a method for controlling cluster size variance through diversifying cluster centroids and enforcing a maximum cluster size. To measure the effects of this, we analyze the variance in cluster sizes produced by the clustering algorithm with and without this cluster control. Results are seen in Figure 5. The Vertical axis of the figure shows the variance in the sizes of the shards, while the horizontal axis shows the number of shards created.

In the figure, we observe that the variance in the non-controlled clusters is substantially greater than that in the controlled clusters. We can thus infer that, while our control measures leave some variance in cluster sizes, it will lead to more consistent search performance and accuracy.

To further show the relation between search time and the size of searched shards (*i.e.,* number of terms in the shard), we analyzed the number of terms in the shards chosen to be searched, during

DOI: 10.1002/cpe

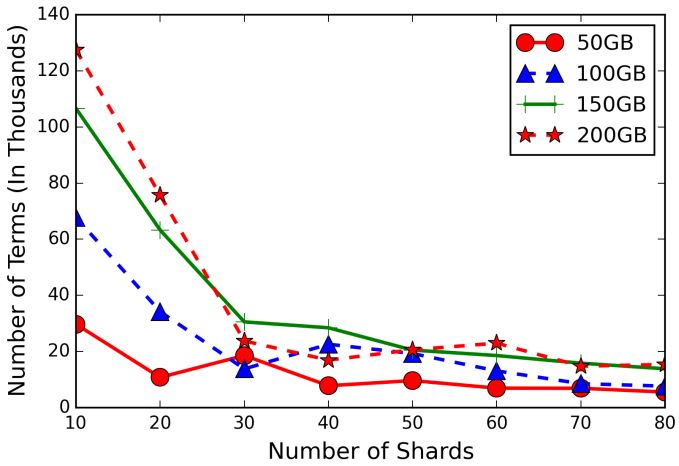

Figure (4). The average number of terms (*i.e.,* size of) the searched shards (Π). The horizontal axis shows the number of shards the dataset sample is partitioned into. The vertical axis shows the average number of terms of the searched shards.

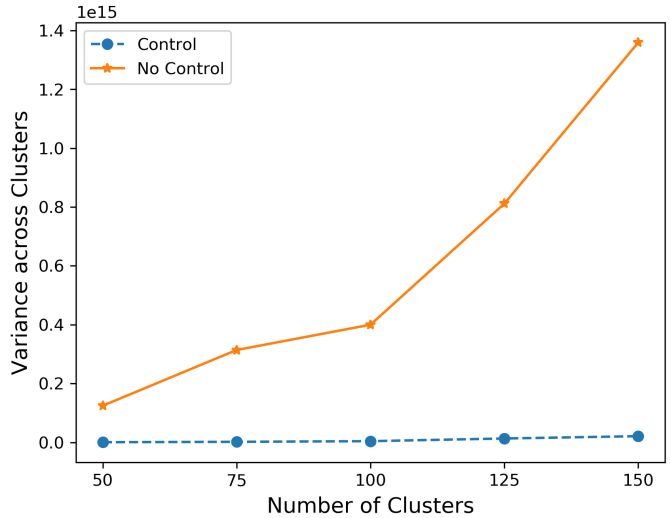

Figure (5). The variance in cluster sizes for controlled and non-controlled clustering. The vertical axis shows the variance, with cluster size measured in the number of documents represented in the cluster. The horizontal axis shows the size of the dataset.

performance evaluations, in Figure 4. Vertical axis of the figure shows the average number of terms in a searched shard, and horizontal axis shows the number of shards created. The rest of setup for this experiment is the same as those for Figure 3.

By comparing the two figures, we observe that the number of terms in shards is correlated with the respective search times; both follow similar patterns. In addition, Figure 4 shows that creating more shards does not necessarily impact the average number of terms in searched shards nor does it improve search times. These justifications support our conclusion that 30 shards is an appropriate number of shards to be created in S3BD.

5.2.4. Performance Comparison of S3BD Versus S3C To show improvements in search time, we compare the search times of S3BD against S3C, the earlier work in the literature. In this experiment, we first compare the components of the overall search time (query modification time and cloud search time) between the two systems using a dataset size of 50 GB. We then compare the cloud

DOI: 10.1002/cpe

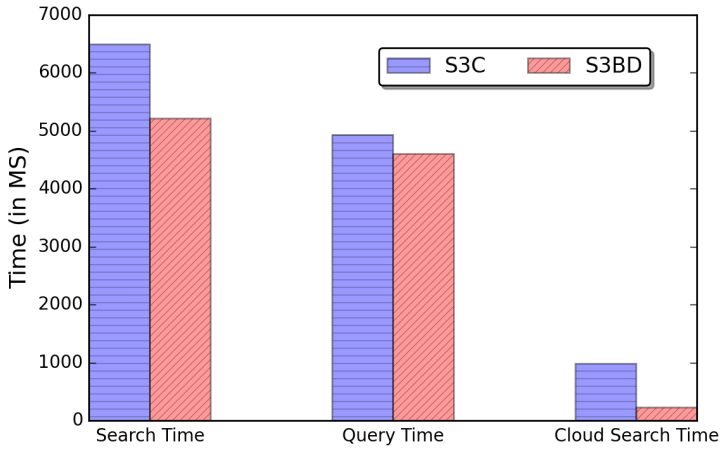

Figure (6). Detailed comparison of the performance of search components in S3BD versus S3C. In this experiment, the dataset sample is 50 GB, clustered into 30 shards in S3BD. Query Time refers to time for query modification; Cloud Search Time is the times to perform searching and ranking; Search Time is the collective time to search. The vertical axis shows the amount of time taken (in milliseconds) for the corresponding component.

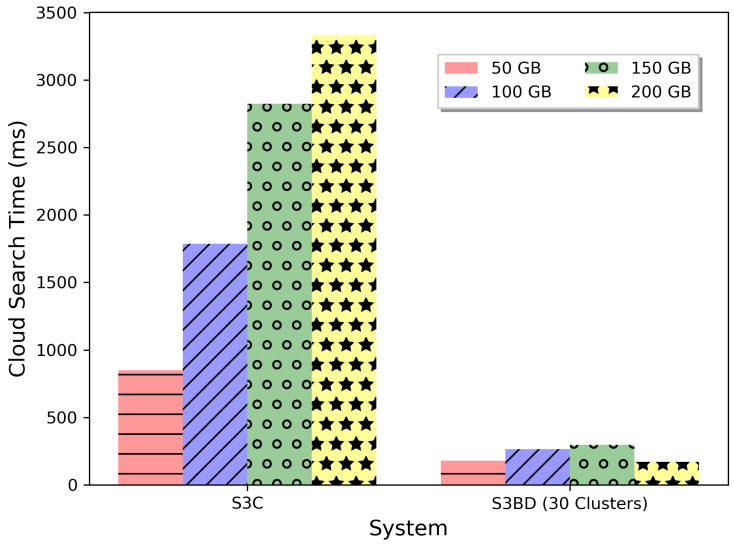

Figure (7). Comparison of the performance of S3C and S3BD across different dataset sizes. Each bar represents a different dataset size, and the vertical axis represents the time taken (in milliseconds) to perform searching and ranking on the cloud.

search time for both systems across various dataset sizes. In accordance with the conclusion of previous experiment, we configured S3BD to cluster the sample into 30 shards.

Figure 6 shows the overall search time as well as the time of the two major actions involved in the search, namely query modification and searching and ranking on the cloud. According to the figure, the total search time of S3BD is 20% less than S3C. While query modification time is not significantly different, the figure expresses that the difference is due to cloud search time, which is 77% less for S3BD.

Figure 7 shows time taken to search and rank on the cloud for the different systems using increasing dataset sizes. As shown by the figure, the search time increases at a higher rate for S3C than S3BD. Additionally, all search times for S3BD are substantially lower than S3C, with time for 200 GB of data at ∼95% less for S3BD. Because of this, S3BD shows more promise for scalability to larger dataset sizes.

DOI: 10.1002/cpe

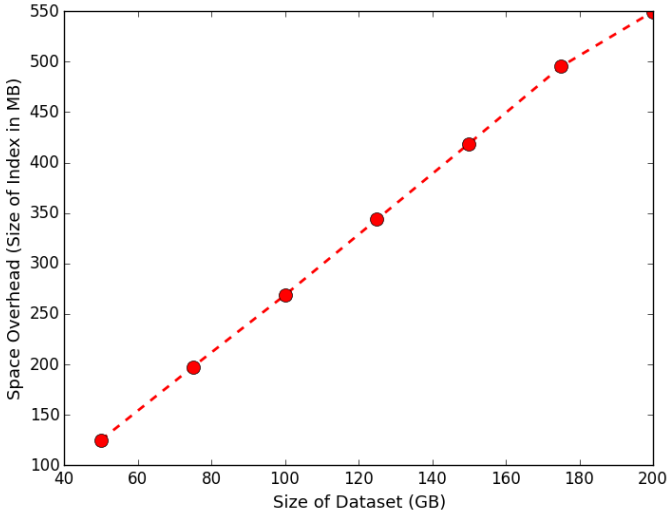

Figure (8). The size of the central index as the dataset increases in size. The horizontal axis plots the size of the dataset used in gigabytes, while the vertical axis plots the associated index size in megabytes. Data points are taken at 25 gigabyte intervals between 50 and 200 gigabytes.

5.3. Evaluating Overhead of S3BD

As the size of the dataset grows, it is important for the utility information created by the Cloud Processing Server (the central Index) to be as small as possible, so as not to increase already large storage requirements. To demonstrate the space-efficiency of S3BD, we show the size of the central index as the size of the dataset increases (seen in Figure 8).

The size of the index is shown to increase in a strictly linear fashion, with a linear regression analysis showing a strong positive correlation with a coefficient of $r = 0.99$. The central index is on average $\sim$0.27% the size of the dataset, adding very little to storage requirements. This small size can be attributed to S3BD's method for extracting only a small number of key phrases from each document.

5.4. Evaluating Accuracy of S3BD

5.4.1. Benchmark Queries to Evaluate Accuracy We derived a set of benchmark queries based on the information present in the RFC dataset. For testing accuracy, we consider two major categories of queries which a user may desire to search. In the first category, we consider a user who already knows which document they are looking for, but may not remember where the document is located in their cloud system or may not want to look through a large number of files to find it. Such queries are typically specific and only a small number of documents should directly pertain to them. An accurate search system is expected to bring up these most desired documents first.

In the second category, we consider a user who wants to find all of the documents related to an idea. For instance, considering our motivational case, the law enforcement officer searching for similar crimes. Such queries would be broad with many possible related documents, and an accurate search system is expected to bring up the most relevant ones first.

5.4.2. Metric for Evaluating Accuracy We define accuracy as how relevant the returned results are to the user's query, and how closely they meet user expectations. We describe accuracy in terms of the TREC-Style Average Precision (TSAP) method described by Mariappan *et al.* [50]. This method is a modification of the precision-recall method commonly used for judging text retrieval systems. It is defined as follows:

$$Score = \frac{\sum_{i=0}^{N} r_i}{N} \tag{9}$$

DOI: 10.1002/cpe

Category 1 - Specific:
`IBM Research Report (IRR)`
`Licklider Transmission Protocol (LTP)`
`Multicast Listener Discovery Protocol (MLDP)`
Category 2 - Broad:
`Internet Engineering (IE)`
`Transmission Control Protocol (TCP)`
`Cloud Computing (CC)`
`Encryption (EN)`

Figure (9). Queries used for evaluating relevance. Queries in Category 1 target a small set of specific, known documents within the collection, while queries in Category 2 target a broad set of documents, not necessarily known to the user.

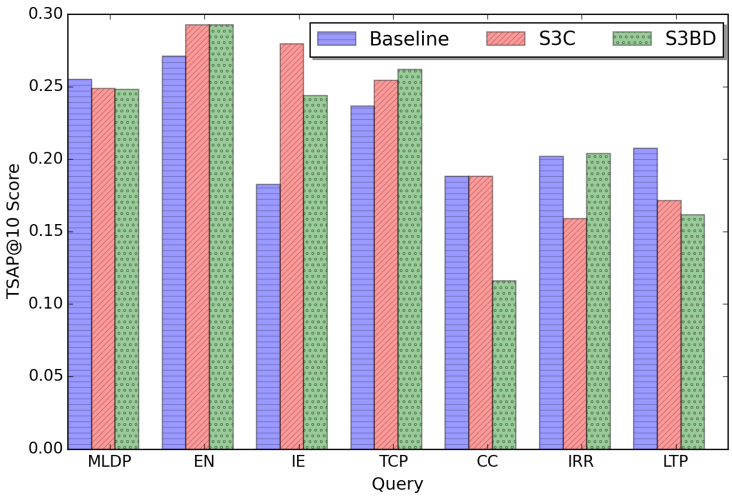

Figure (10). TSAP@10 score for specified queries for S3BD , S3C, and a baseline system. For a given query, once the systems return a ranked list of results, a score is computed based on the human-determined relevance to each file.

Where i is the rank of the document determined by the system and N is the cutoff number (10 in our case, hence the term TSAP@10). r_i takes three different values:

- $r_i = 1/i$ `if the document is highly relevant`
- $r_i = 1/2i$ `if the document is somewhat relevant`
- $r_i = 0$ `if the document is irrelevant`

This allows for systems to be given a comparative score against other schemes in a relatively fast manner, without the need for knowledge of the entire dataset.

5.4.3. Results of Evaluating Accuracy Figure 10 shows the TSAP scores (vertical axis) for different benchmark queries (abbreviated in the horizontal axis). For each benchmark query, we compare the relevance score of S3BD compared to the scores of S3C and a baseline standard approach. In the baseline system, a simpler document representation is used (without keyword extraction) query modification is simpler, and there is no topic clustering, while the same Okapi search algorithm is used.

While S3BD might intuitively seem to suffer in accuracy due to lower document representation within the shards, the figure expresses that is not the case. The relevance of the results obtained from S3BD, for most of them benchmark queries, are either the same or similar to those of S3C and the baseline. When compared to the less efficient baseline, S3BD provides better results for four of the benchmark queries. When compared to S3C, the relevance of S3BD is only lower in two benchmark

Prepared using cpeauth.cls
DOI: 10.1002/cpe

queries, namely `Cloud Computing` and `Internet Engineering`. As a matter of fact, both of these benchmark queries are in the broad category. In the contrary, we observe that benchmark queries in the specific category have almost identical relevance in both systems.

We can infer that S3BD provides higher accuracy with specific queries. The reason is that S3BD can find shards for specific terms in the queries more accurately, in comparison to broad (*i.e.,* general) terms. In fact, for broad terms it is possible that pruning leads us to search less relevant shards. It is an interesting future research avenue to recognize broad terms and apply less aggressive pruning for them.

6. CONCLUSION

In this research, we developed S3BD, a system to perform a secure semantic search over encrypted big data in the cloud. S3BD achieves real-time search ability on big data through *pruning* irrelevant portions of the dataset at search time. S3BD is comprised of three major architectural components, namely client application, cloud processing server, and cloud storage. After parsing and uploading documents, the cloud processing server clusters a central encrypted index into smaller, topic-based shards. At search time, client application compares the user's query to abstracted versions of those shards to determine the appropriate shards to be searched. S3BD ontologically expands the user's search query to achieves semantic search ability.

We performed analyses on S3BD's performance and search accuracy using a working prototype. We analyzed the number of created shards to strike a trade-off between search performance and accuracy. Comparison of S3BD against similar works in the literature demonstrated that S3BDimproves the search performance on the cloud by approximately 77% without compromising accuracy.

There are several avenues of research to extend S3BD. One interesting avenue is to determine the number of shards to be searched based on the broadness of the user's query. Another avenue is to dynamically determine the number of shards based on the dataset characteristics, *e.g.,* size. Adapting the S3BD architecture based on the edge computing model can be explored to improve the efficacy of the search.

AVAILABILITY

Distributable .jars of the S3BD core, as well as running instructions, are available at `http://hpcclab.org/products/S3BDJars.zip`.

A preliminary version of S3BD with web interface for demonstration purposes is available at `https://teaching.cmix.louisiana.edu/~c00408440/S3C/S3Client/home.php`.

ACKNOWLEDGMENTS

We would like to acknowledge anonymous reviewers of the manuscript. This research was supported by the Louisiana Board of Regents under grant number LEQSF(2017-20)-RD-B-06, and Perceptive Intelligence, LLC. Preliminary version of portions of this material were presented at the IEEE Big Data 2016 [35].

REFERENCES

1. S. M. Zobaed, M. A. Salehi, Big Data in the Cloud, Springer International Publishing, 2018, pp. 1–8.
2. J. Hoppermann, L. Herbert, Software-as-a-service in banking, Tech. rep., Forrester.
3. M. Javanmard, M. A. Salehi, S. Zonouz, Tsc: Trustworthy and scalable cytometry, in: Proceedings of the 7th IEEE International Symposium on Cyberspace Safety and Security, 2015, pp. 1356–1360.
4. R. Fathi, M. A. Salehi, E. L. Leiss, User-friendly and secure architecture (ufsa) for authentication of cloud services, in: Proceedings of the 8th IEEE International Conference on Cloud Computing, 2015, pp. 516–523.
5. K. Gai, M. Qiu, Z. Xiong, M. Liu, Privacy-preserving multi-channel communication in edge-of-things, Future Generation Computer Systems 85 (2018) 190 – 200.

DOI: 10.1002/cpe

6. K. Gai, M. Qiu, Z. Ming, H. Zhao, L. Qiu, Spoofing-jamming attack strategy using optimal power distributions in wireless smart grid networks, IEEE Transactions on Smart Grid 8 (5) (2017) 2431–2439.
7. S. Subashini, V. Kavitha, A survey on security issues in service delivery models of cloud computing, Journal of Network and Computer Applications 34 (1) (2011) 1 – 11.
8. D. X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: Proceedings of the 17th IEEE symposium on Security and Privacy, 2000, pp. 44–55.
9. N. Cao, C. Wang, M. Li, K. Ren, W. Lou, Privacy-preserving multi-keyword ranked search over encrypted cloud data, IEEE Transactions on Parallel and Distributed Systems 25 (1) (2014) 222–233.
10. X. Sun, Y. Zhu, Z. Xia, L. Chen, Privacy preserving keyword based semantic search over encrypted cloud data, International Journal of Security and Its Applications 8 (3).
11. R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions, Journal of Computer Security 19 (5) (2011) 895–934.
12. L. A. Barroso, J. Dean, U. Holzle, Web search for a planet: The google cluster architecture, IEEE Micro 23 (2) (2003) 22–28.
13. J. Xu, W. B. Croft, Cluster-based language models for distributed retrieval, in: Proceedings of the 22nd International ACM Conference on Research and Development in Information Retrieval, SIGIR '99, 1999, pp. 254–261.
14. D. Boneh, G. Di Crescenzo, R. Ostrovsky, G. Persiano, Public key encryption with keyword search, in: Advances in Cryptology – Eurocrypt '04, 2004, pp. 506–522.
15. E.-J. Goh, et al., Secure indexes, Cryptology ePrint Archive (2003) 216.
16. P. van Liesdonk, S. Sedghi, J. Doumen, P. Hartel, W. Jonker, Computationally efficient searchable symmetric encryption, in: Proceedings of the 7th VLDB Workshop on Secure Data Management, Springer, 2010, pp. 87–100.
17. C. Mangold, A survey and classification of semantic search approaches, International Journal of Metadata, Semantics and Ontologies 2 (1) (2007) 23–34.
18. A. Andrejev, D. Misev, P. Baumann, T. Risch, Spatio-temporal gridded data processing on the semantic web, in: Proceedings of the First IEEE International Conference on Data Science and Data Intensive Systems, 2015, pp. 38–45.
19. A. Tonon, M. Catasta, R. Prokofyev, G. Demartini, K. Aberer, P. Cudr-Mauroux, Contextualized ranking of entity types based on knowledge graphs, Web Semantics: Science, Services and Agents on the World Wide Web 3738 (2016) 170 – 183.
20. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, RQL: A declarative query language for RDF, in: Proceedings of the 11th International Conference on World Wide Web, 2002, pp. 592–603.
21. E. J. Glover, S. Lawrence, W. P. Birmingham, C. L. Giles, Architecture of a metasearch engine that supports user information needs, in: Proceedings of the 8th International Conference on Information and Knowledge Management, 1999, pp. 210–216.
22. Y. Lei, V. Uren, E. Motta, Semsearch: A search engine for the semantic web, in: Proceedings of the 15th international conference on Managing Knowledge in a World of Networks, 2006, pp. 238–245.
23. R. Guha, R. McCool, E. Miller, Semantic search, in: Proceedings of the 12th International Conference on World Wide Web, WWW '03, 2003, pp. 700–709.
24. J. Heflin, J. Hendler, Searching the web with SHOE, in: Proceedings of the 17th Association for the Advancement of Artificial Intelligence Workshop on AI for Web Search, AAAI '00, 2000, pp. 35–40.
25. J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, W. Lou, Fuzzy keyword search over encrypted data in cloud computing, in: Proceedings of the 29th IEEE International Conference on Computer Communications, INFOCOM '10, 2010, pp. 1–5.
26. C. Wang, K. Ren, S. Yu, K. M. R. Urs, Achieving usable and privacy-assured similarity search over outsourced cloud data, in: Proceedings of the 31st IEEE International Conference on Computer Communications, INFOCOM '12, 2012, pp. 451–459.
27. M. A. Salehi, T. Caldwell, A. Fernandez, E. Mickiewicz, E. W. D. Rozier, S. Zonouz, D. Redberg, Reseed: Regular expression search over encrypted data in the cloud, in: Proceedings of the 7th IEEE International Conference on Cloud Computing, 2014, pp. 673–680.
28. M. A. Salehi, T. Caldwell, A. Fernandez, E. Mickiewicz, E. W. D. Rozier, S. Zonouz, D. Redberg, Reseed: A secure regular-expression search tool for storage clouds, Software: Practice and Experience 47 (9) 1221–1241.
29. T. Moataz, A. Shikfa, N. Cuppens-Boulahia, F. Cuppens, Semantic search over encrypted data, in: Proceedings of the 20th International Conference on Telecommunications (ICT), 2013, pp. 1–5.
30. C. J. V. Rijsbergen, Information Retrieval, 2nd Edition, Butterworth-Heinemann, Newton, MA, USA, 1979.
31. X. Liu, W. B. Croft, Cluster-based retrieval using language models, in: Proceedings of the 27th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '04, 2004, pp. 186–193.
32. A. Tombros, R. Villa, C. V. Rijsbergen, The effectiveness of query-specific hierarchic clustering in information retrieval, Information Processing & Management 38 (4) (2002) 559 – 582.
33. R. Baeza-Yates, V. Murdock, C. Hauff, Efficiency trade-offs in two-tier web search systems, in: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '09, 2009, pp. 163–170.
34. A. Kulkarni, J. Callan, Topic-based index partitions for efficient and effective selective search, in: Proceedings of the 8th Workshop on Large-Scale Distributed Systems for Information Retrieval, 2010, pp. 19–24.
35. J. Woodworth, M. A. Salehi, V. Raghavan, S3C: An architecture for space-efficient semantic search over encrypted data in the cloud, in: Proceedings of the 3rd International Workshop on Privacy and Security of Big Data (PSBD), 2016.
36. O. Medelyan, E. Frank, I. H. Witten, Human-competitive tagging using automatic keyphrase extraction, in: Proceedings of the 14th Conference on Empirical Methods in Natural Language, EMNLP '09, 2009, pp. 1318–1327.
37. R. A. Popa, C. M. S. Redfield, N. Zeldovich, H. Balakrishnan, Cryptdb: Protecting confidentiality with encrypted query processing, in: Proceedings of the 23rd ACM Symposium on Operating Systems Principles, SOSP '11, 2011, pp. 85–100.
38. W. Diffie, M. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (6) (1976) 644–654.

DOI: 10.1002/cpe

39. K. Gai, M. Qiu, Blend arithmetic operations on tensor-based fully homomorphic encryption over real numbers, IEEE Transactions on Industrial Informatics 14 (8) (2018) 3590–3598.
40. L. Ducas, D. Micciancio, Fhew: Bootstrapping homomorphic encryption in less than a second, in: Advances in Cryptology – EUROCRYPT 2015, Springer, Berlin, Heidelberg, 2015, pp. 617–640.
41. J. A. Hartigan, M. A. Wong, Algorithm as 136: A k-means clustering algorithm, Journal of the Royal Statistical Society. Series C (Applied Statistics) 28 (1) (1979) 100–108.
42. Z. Wu, M. Palmer, Verbs semantics and lexical selection, in: Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics, ACL '94, 1994, pp. 133–138.
43. W. J. Wilbur, K. Sirotkin, The automatic identification of stop words, Journal of Information Science 18 (1) (1992) 45–55.
44. T. S. Moh, K. H. Ho, Efficient semantic search over encrypted data in cloud computing, in: Proceedings of the 6th International Conference on High Performance Computing Simulation, 2014, pp. 382–390.
45. S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, M. Gatford, Okapi at TREC-3, Overview of the 3rd Text Retrieval Conference (TREC-3) 3 (1995) 109–126.
46. D. Wang, P. Wang, Offline dictionary attack on password authentication schemes using smart cards, in: Information Security, Springer International Publishing, 2015, pp. 221–237.
47. Common Crawl on Amazon Web Services (AWS), https://aws.amazon.com/public-datasets/common-crawl/.
48. Request for Comments (RFC) Document Series, http://www.ietf.org/rfc.html.
49. X. Li, M. A. Salehi, M. Bayoumi, High performance on-demand video transcoding using cloud services, in: 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2016, pp. 600–603.
50. A. K. Mariappan, R. M. Suresh, V. S. Bharathi, A comparative study on the effectiveness of semantic search engine over keyword search engine using tsap measure, International Journal of Computer Applications EGovernance and Cloud Computing Services (2012) 4–6.

DOI: 10.1002/cpe