Tanium Reveal: A Federated Search Engine for Querying Unstructured File Data on Large Enterprise Networks

Josh Stoddard
Tanium Inc.
Cary, NC, USA
josh.stoddard@tanium.com

Adam Mustafa
Tanium Inc.
Cedar Grove, NJ, USA
adam.mustafa@tanium.com

Naveen Goela
Tanium Inc.
Emeryville, CA, USA
naveen.goela@tanium.com

ABSTRACT

Tanium Reveal is a federated search engine deployed on large-scale enterprise networks that is capable of executing data queries across billions of private data files within 60 seconds. Data resides at the *edge* of networks, potentially distributed on hundreds of thousands of endpoints. The anatomy of the search engine consists of local inverse indexes on each endpoint and a global communication platform called Tanium for issuing search queries to all endpoints. Reveal enables asynchronous parsing and indexing on endpoints without noticeable impact to the endpoints' primary functionality. The engine harnesses the Tanium platform, which is based on a self-organizing, fault-tolerant, scalable, linear chain communication scheme. We demonstrate a multi-tier workflow for executing search queries across a network and for viewing matching snippets of text on any endpoint. We analyze metrics for federated indexing and searching in multiple environments including a production network with 1.05 billion searchable files distributed across 4236 endpoints. While primarily focusing on Boolean, phrase, and similarity query types, Reveal is compatible with further automation (e.g., semantic classification based on machine learning). Lastly, we discuss safeguards for sensitive information within Reveal including cryptographic hashing of private text and role-based access control (RBAC).

PVLDB Reference Format:
Josh Stoddard, Adam Mustafa, and Naveen Goela. Tanium Reveal: A Federated Search Engine for Querying Unstructured File Data on Large Enterprise Networks. PVLDB, 14(12): 3096 - 3109, 2021.
doi:10.14778/3476311.3476386

1 INTRODUCTION

A primary challenge for large-scale enterprises is the management of distributed, sensitive data. Financial institutions manage investment portfolios of clients, hospitals maintain medical records of patients, retail stores collect customer profiles, entertainment companies own movie scripts, and technology firms possess intellectual property. Data breaches pose significant risks and can be costly to resolve (see e.g., Target [52] and Equifax [63]). To protect against leaks, an enterprise must first gain visibility over its sensitive information among potentially billions of data files distributed across hundreds of thousands of network endpoints.

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476386

In this paper, we present our experiences designing, building, and operating Tanium Reveal, a federated search engine for identifying and managing sensitive data at scale on enterprise networks [74]. Reveal offers a multi-tier cascaded workflow for search queries at different levels of granularity: (i) across endpoints in a network; (ii) across data files on each endpoint; (iii) across snippets of text within data files. In addition to searching for arbitrary keywords and phrases, Reveal can evaluate compliance with security standards for data protection, such as those mandated by government regulations and laws. For example, PCI standards help protect personal credit card payment information [51], HIPAA standards secure patient health data [17], and GDPR standards safeguard personally-identifiable information (PII) [23, 81]. Reveal can detect patterns of sensitive text within unstructured data on endpoints, thereby identifying regulatory noncompliance.

The design of Reveal relies fundamentally on the Tanium platform, which establishes communication between a central Tanium server and all endpoints in a network. The platform communicates via self-organizing, fault-tolerant, scalable linear chains of endpoints (also known as communication orbits) [29, 30]. This communication topology takes advantage of efficient peer-to-peer links between all endpoints which allows Reveal's query responses to be aggregated within 60 seconds. After a search query completes, Reveal opens direct connections to individual endpoints for fine-grained browsing of search results.

1.1 What is a Federated Search Engine?

Search engine components support two major functions: indexing and querying [5, 15]. Indexing includes parsing of text extracted from files and populating data structures that represent a searchable index. We consider standard inverted indexing for data retrieval in this paper [15, 19, 78, 85]. A *federated* search engine is an architecture in which indexing is conducted at the edge of the network by each endpoint asynchronously. The search engine relies on a central server to issue queries and aggregate query responses from all network endpoints. Iterative rounds of communication take place between the server and endpoints. Consequently, the federated architecture depends on a global communication platform which must achieve minimal latencies for live search queries.

A federated architecture exemplifies an edge computing paradigm in which computation and storage are shifted closer to the edge of networks [9, 40, 60]. For sensitive data discovery in large enterprise networks, copying data to a central cloud in order to optimize indexing and provide sub-second query latency can be infeasible for several reasons such as data protection and access constraints, regulations and security standards, network bandwidth limitations, and cloud infrastructure costs. Leaving sensitive data

Table 1: High-Level Specifications of Search Engines

Search Engine	Centralized [8]	Federated
Data Volume	>100B pages [24] World Wide Web	>1B files/enterprise >100k files/endpoint >10k endpoints
Indexing		
Index Location	cloud data center	endpoint devices
Index Size	>100 PB [24]	<10 GB/endpoint
Index Compute	cloud compute	<5% CPU/endpoint
Network Cost	linear in data	$O(1)$ in data
Index Refresh	3-28 days [59]	<12 hours
Search		
Query Latency	<0.1s [80]	<60s
Query Frequency	>80k queries/s [36]	<100 queries/s
Data		
Data At Edge	no	yes
Sensitive Data	no	yes

at rest on endpoints can also facilitate remedial action such as the deletion of a file or quarantining of an endpoint.

1.2 Federated vs. Centralized Search Engines

To provide further context, we compare the federated search architecture of Reveal with Google's centralized search architecture [24] across several categories, as displayed in Table 1.

1.2.1 Data Volume. A typical enterprise network can store over 1B files on edge devices. As computed in Table 1, a network containing 10k endpoints with an average of 100k files per endpoint stores 1B files. Large-scale private networks with 500k endpoints can store 50B files. Individually, a server endpoint can store more than 1M files. Data files contain unstructured heterogeneous data, may differ in size considerably, and may be duplicated across endpoints. The total data volume is subject to growth as endpoints save new data files and new endpoints are brought online.

Google's search engine, in contrast, indexes "hundreds of billions of webpages" centrally [24]. Public webpages are available and reachable on the visible, surface web; i.e., the indexed portion of the World Wide Web. Across the Internet, public data is copied from the Web to Google's data centers and indexed centrally.

1.2.2 Indexing. On each endpoint, Reveal maintains a standard inverted index for file retrieval [15, 19, 78, 85]. Reveal builds indexes asynchronously and processes files locally. Several measures are taken to ensure that indexing does not interfere with the primary operation of each device. Local updates to the index typically occur within hours. In real-world deployments, the index size can be limited to a predefined value such as 4GB. We call Reveal's indexing process *federated indexing*. Importantly, federated indexing does not require significant network bandwidth since index data remains on each endpoint and references only local content.

Google's index is maintained centrally and populated with public data mined by web crawlers. Operating in parallel, crawler bots read and parse billions of pages on the World Wide Web, recursively following links on those pages [6, 10, 50]. The Google index "is well over 100,000,000 gigabytes in size" [24]. The entire index is stored over thousands of computers in the data center. The index refresh rate to crawl new websites is usually between 3-28 days [59].

1.2.3 Query Processing. Reveal's multi-tier interactive workflow begins by identifying endpoints that contain files matching a search query. This first step harnesses the Tanium communication platform to obtain live responses from network endpoints within 60s. After sorting and filtering results according to multiple criteria, a Reveal operator can then open direct connections to asset-critical endpoints to view matching files. The operator may also browse snippets of text within these files that match the original search query, or refine the query. Reveal is intended for use in private networks with relatively few concurrent search queries.

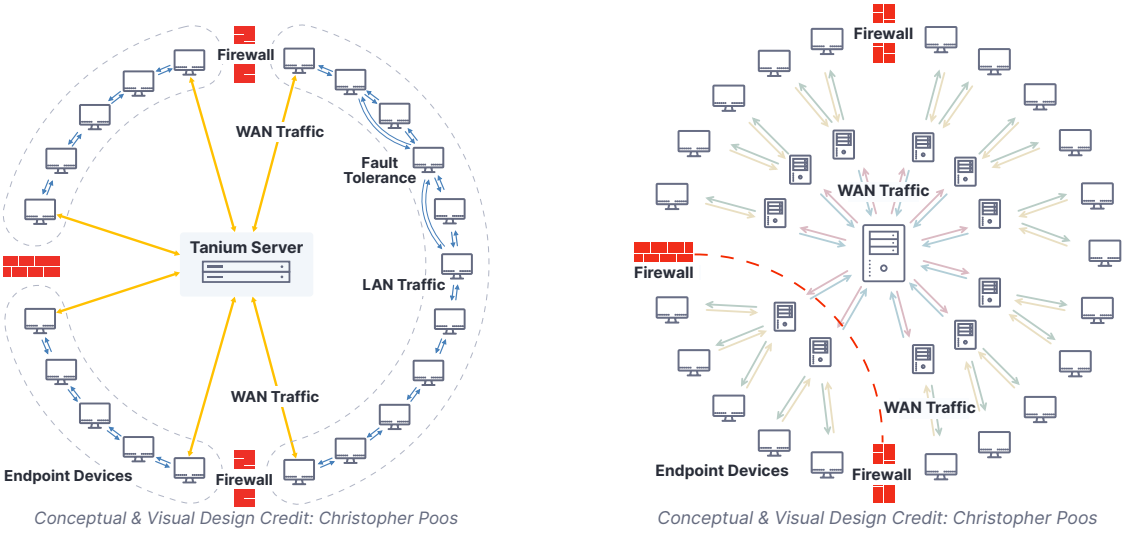
The Google search engine, in contrast, accommodates over 80k search queries per second executed world-wide [36]. Each query is processed efficiently in less than 0.1s in data centers [80]. Query processing includes a sophisticated ranking of page results (e.g., PageRank [8]). Google also enhances search results with semantic information drawn from "knowledge graphs" [18].

1.2.4 Private vs. Public Data. Reveal is a search engine suitable for sensitive data owned by enterprises. Frequently, a Reveal operator is interested in finding *all* file matches of a keyword, phrase, or rule. For instance, to find PII (e.g., names, passwords, and addresses), a Reveal operator issues precise, targeted queries. For general queries with broader scope (e.g., finding resumes, financial documents, or confidential memos), a ranking of search results based on semantic relevance is helpful. Compared to search engines accessing public data, Reveal can target search to user-specified groups of endpoints or files.

1.3 Related Search Engines

1.3.1 Reveal vs. Helios. Highly-scalable, hybrid search engines such as Helios [54] have demonstrated the advantages of combining cloud and edge paradigms for ingesting, indexing, and aggregating real-time data [13, 14]. Helios ingests petabytes of raw data (≈12 PB/day) from "hundreds of thousands" of edge machines within Microsoft's managed network. Edge machines process raw data (e.g., utilizing <1.6 GB memory and 15%-65% single CPU) and upload data blocks to ingestion servers located in the Helios cluster. Helios computes indexing blocks *asynchronously*, which contain pointers to data blocks. Its hierarchical index (e.g., with L0-L3 layers) provides a fast centralized search comprised of 16 trillion search keys indexing into a quadrillion events or log lines [54].

As is true for Reveal, a "key factor in Helios's success is asynchronous index management" [54]. However, due to constraints on moving and persisting sensitive data, Reveal does not upload data blocks (or index blocks) to a central cluster. Reveal must build indexes and resolve queries on autonomous endpoints without disrupting their primary functionality (e.g., utilizing <5% single CPU). Reveal does not maintain a central index for sub-second query processing. Yet, the latency of live network queries is still <60s. Crucially, Reveal must operate commercially in secure, diverse, enterprise networks, whereas Helios is designed to monitor



(a) Tanium linear chain topology

(b) Hierarchical, "hub-and-spoke", tree topology

Figure 1: A comparison of two network topologies for managing endpoints in large-scale enterprises.

Microsoft's internal self-managed network. Similar to Helios, Reveal can process petabytes of raw data collectively (e.g., >20 GB per endpoint, 100k endpoints). Compared to Helios's real-time ingest of events and logs, Reveal only requires refreshing indexes hourly.

1.3.2 Related Federated Search Engines. A federated search engine issues a query to multiple information retrieval systems concurrently, and aggregates results to present a unified view of information [2, 62]. Prior engines include Muse [48], WebFeat [38], and FOSSICK [11]. Search results can be retrieved from "over one-hundred" search engines [47], multiple databases [61], or digital libraries [43].

Tanium Reveal, in contrast, aggregates data from tens of thousands of endpoints and larger enterprise environments. Reveal must satisfy several design constraints. Endpoints provide (i) asynchronous, *resource-constrained* index management; (ii) data security by processing sensitive files locally; (iii) safeguards for secure operation in terms of persisting indexes, encrypting queries, and access control. Reveal must operate in diverse enterprise networks, and resolve issues pertaining to: (1) network bandwidth constraints; (2) end-to-end latency specifications for executing live search queries at scale; (3) iterative coordination and control of endpoints via efficient communication between a central server and clients.

Compared to enterprise data catalogs, business glossaries, and library collections (e.g., Alation [1], Lumada [31], and WorldCat [49]), Reveal indexes and queries *unstructured* heterogeneous data.

1.4 Overview of Key Contributions

- **Federated Search:** Tanium Reveal's search engine architecture consists of federated indexing on endpoints (Section 3) and an interactive workflow for query processing (Section 4) that exploits the Tanium linear chain communication platform (Section 2). Its resource-managed endpoint indexing (3.2) enables search without propagating sensitive data or interfering with endpoint operation. Its multi-tier workflow (4.1) allows users to see sorted, targeted results from billions of searchable files across all endpoints in seconds and explore those results as they exist on remote machines in real time.
- **Large-Scale Production Systems:** Reveal is a commercial engine proven to function in large-scale enterprise networks. We validate its design by collecting and evaluating metrics from two anonymized sub-networks (Section 5). The first sub-network contains more than 427M files distributed across 1273 endpoints. The second larger sub-network contains more than 1.05B files distributed across 4236 endpoints. We also measure search query latency on five enterprise networks of varying scale.
- **Operational Experiences:** We discuss the effectiveness of keyword, phrase, and rule-based search for identifying sensitive data pertaining to security standards (e.g., PCI, HIPAA, GDPR). We highlight several practical issues associated with operating Reveal in production environments (Section 6).

We also discuss other related technology and future applications of federated search in next-generation networks (Section 7).

2 TANIUM NETWORK COMMUNICATION

The Tanium platform establishes iterative communication between a central Tanium server and all endpoints of a network as in Figure 1a. The communication platform is based on fault-tolerant, self-organizing, scalable, linear chains of endpoints [29, 30].

2.1 Tanium Server and Clients

2.1.1 Tanium Server and Clients. A Tanium deployment consists of a single Tanium server [72] that communicates with many Tanium clients [70], which are services installed on each endpoint. The Tanium server maintains definitions for various types of content describing operations to be performed by Tanium clients. Tanium

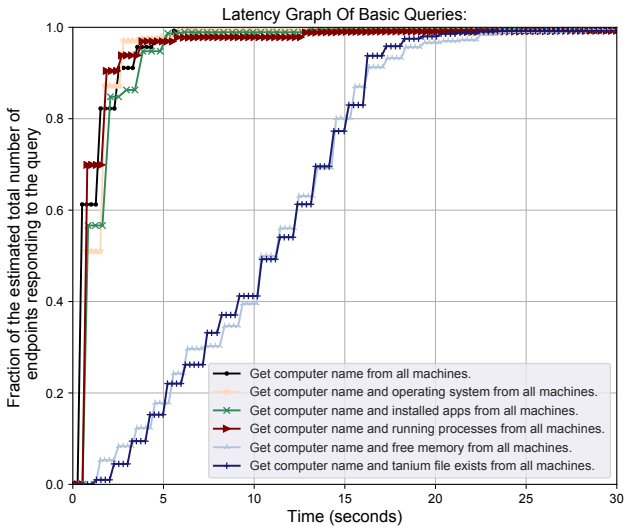


Figure 2: The latency graph of basic Questions issued to $N = 1205$ available endpoints in a test network.

content consists of Sensors, Questions, Packages, and Actions. The Tanium server distributes content to endpoints and aggregates client responses.

2.1.2 Registration. Once installed on an endpoint, the Tanium client initiates a connection to the Tanium server. During registration, each client establishes a unique ID and receives the latest settings, content definitions, and a list of nearby peers. The client uses this information to create connections with its neighbors ultimately forming a linear chain of connected peers. Periodically at randomized intervals, each client re-registers with the Tanium server to receive updates (e.g., the current state of its neighbors).

2.2 Linear Chain Network Topology

2.2.1 Self-Organizing Communication Orbits. The assembly of communication orbits occurs in a decentralized way. After registration, each endpoint maintains a record of local peer machines with which it communicates directly. Peer connections are continuous, long-lived connections that the clients use to exchange Tanium messages and files. Peer-to-peer communication is unidirectional and transitive, meaning endpoints conform to a stable ordering and only send messages to subsequent machines. The client peering process is iterated to form a linear chain of LAN-connected endpoints.

Each message received by a Tanium client includes a request from the Tanium server and an aggregated response from upstream peers. The client handles the request locally, incorporates its local result into the aggregated response, and sends the updated message to the next downstream peer. An endpoint with no upstream peers serves as the *backward leader* of a linear chain, which receives messages directly from the Tanium server. An endpoint with no downstream peers is called a *forward leader* and is responsible for forwarding data aggregated along the chain back to the Tanium server. Figure 1a illustrates 3 communication orbits exhibiting the linear chain topology.

2.2.2 Network Architecture (LAN vs. WAN). The Tanium communication platform optimizes the use of network resources. Client peering over local-area network (LAN) results in efficient communication between peers and significantly reduces both the number of connections and bandwidth utilization over wide-area network (WAN) links. Only forward and backward leaders in each communication orbit maintain connections with the Tanium server, thereby preventing server overload. Further advantages are gained by compressing the message payload during the aggregation of data along the linear chain (proprietary to the Tanium platform). By contrast, the hierarchical "hub-and-spoke" tree topology of Figure 1b is more reliant on congested WAN traffic and requires additional secondary servers to scale, which increases latency.

2.2.3 A Fault-Tolerant Network. As illustrated in Figure 1a, the Tanium linear chains of communication navigate around offline endpoints, reflect around network blockages such as firewalls, and accommodate newly available endpoints. As an additional mechanism for fault-tolerance, the Tanium server can send multiple instances of a message along each communication orbit which can be deduplicated by the client.

2.3 Visibility and Control over Endpoints

The Tanium platform provides both visibility and control over all endpoints in large-scale enterprise networks. Users are able to view up-to-the-minute information about the state of their environment by utilizing structured queries issued by the Tanium server that leverage *Sensors* to extract relevant data from each endpoint. Similarly, Tanium permits authorized users (personas) to control the state of endpoints by scheduling *Actions*.

2.3.1 Sensors and Questions. *Sensors* are OS-specific scripts to be executed on endpoints to gather local information [73]. Sensors are programmable and can accept input parameters. For example, there is a Sensor to determine whether a particular file exists on an endpoint by checking the local file system. *Questions* are structured search queries parsed from natural language to execute sensors synchronously on selected endpoints and return results. Figure 2 lists a few Questions which have associated Sensors to obtain computer name, operating system, installed applications, running processes, and available memory (RAM) from each endpoint.

2.3.2 Packages and Actions. *Packages* serve as containers to deliver files and tools and usually include instructions (e.g., native OS command line) to run executable content. *Actions* deploy Packages to endpoints, which are then processed asynchronously by the Tanium client [71]. For example, actions can be used to install, update, or remove client applications. An Action typically affects the state of an endpoint (e.g., rebooting a system or installing software). Actions can be performed once on-demand or scheduled periodically via a policy to provide *persistent maintenance* for endpoints.

2.3.3 Targeted Queries and Filtering. The Tanium platform supports targeting of Questions and Actions through the use of *filters* [73]. Described via relational operators, filters are Boolean operations evaluated by each endpoint based on local sensor results to determine whether to respond to a particular question or action. For example, a question to determine `CPU` usage can target

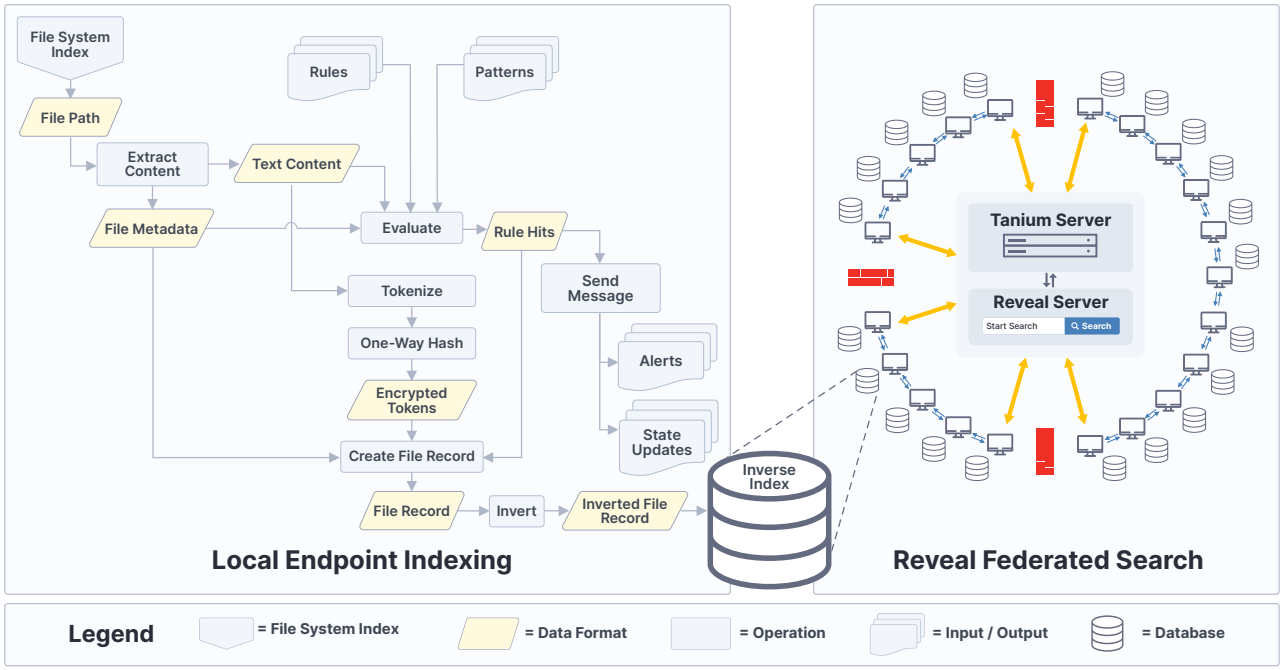


Figure 3: The Reveal federated search engine relies on asynchronous, federated indexing of files on endpoints.

machines according to various properties (e.g., computer group, operating system, installed applications).

2.4 Speed at Scale

2.4.1 Test Network. Figure 2 measures the response times for a few questions issued to 1205 endpoints in a test network. The latency graph measures the time taken for the Tanium server to receive results from all endpoints. Each question invokes a different set of endpoint sensors, and these sensors complete in different amounts of time. In particular, the sensors for the current free memory and for the existence of a particular file take longer for endpoints to execute than the other sensors, and this is reflected in the observed latency when those sensors are in use.

2.4.2 Large-Scale Enterprise Networks. Counter-intuitively and remarkably, the Tanium platform maintains a query latency of less than 60 seconds even as the network scales to 100,000 endpoints and beyond. Due to the linear chain topology, the Tanium server communicates only with the forward and backward leaders of each communication orbit. A network of 100,000 endpoints communicating via orbits of 100 endpoints each requires only 1000 connections to and from the Tanium server. This allows the Tanium server to get the responsiveness of direct communication with endpoints without the bandwidth cost of having to connect to each endpoint. As confirmed in Section 5, the latency of a search query remains less than 60 seconds even in large-scale production networks.

3 REVEAL SYSTEM ARCHITECTURE

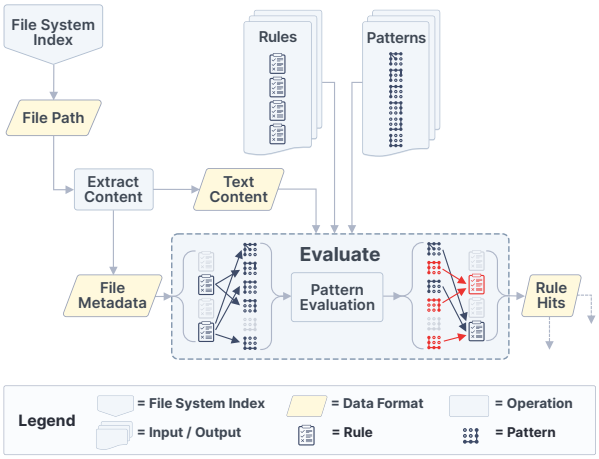
As depicted in Figure 3, the major components of a Reveal deployment include a Reveal *server* and multiple *endpoint agents*.

3.1 Reveal Server

The Reveal server is a single centralized process within the network environment that is responsible for administering endpoint agents and aggregating data across its environment. It services a web-based front-end that provides workflows such as ad hoc text search to privileged users. The Reveal server communicates directly with the Tanium Server (Section 2.1.1) to orchestrate communication with endpoints. The server also maintains a database of global content such as classification rules, data pattern definitions, and aggregated endpoint statistics. The Reveal service does not perform indexing or query evaluation, and does not maintain index data.

3.1.1 User Interface (UI). The Reveal server services a web-based UI available to privileged Reveal operators. The Reveal UI provides a multi-tier search workflow that provides exhaustive query results sorted globally by score and allows users to view result details on arbitrary endpoints. In addition to search, the UI allows user to view global endpoint and file statistics, author file classification criteria, validate and refine classification results, and monitor Reveal endpoint telemetry.

3.1.2 Role-Based Access Control. Reveal makes use of Role-Based Access Control (RBAC) provided by the Tanium platform [71]. Operators are granted access to individual features in Reveal's workbench according to their assigned roles. Some Reveal roles provide the ability to search and view content from any file that is readable by a Reveal endpoint agent. Only domain users with at least the same level of read permission are granted such roles.



Conceptual & Visual Design Credit: Christopher Poos

Figure 4: The evaluation of rules based on patterns matched within a file's contents.

3.2 Reveal Endpoint Agent

The Reveal endpoint agent runs on each endpoint and is responsible for maintaining a searchable index of files present on that machine. It runs as a persistent, low-priority, resource limited process managed by the Tanium client. The agent runs as a domain user with read-only access to local user files. It systematically assigns features to local files according to their contents, indexes those files with respect to their assigned features, and services query requests against its index data originating from the Reveal server.

The Reveal agent indexes files by reading their contents from disk and extracting readable text. As diagrammed in detail in Figure 3, Reveal processes the extracted file contents in multiple ways to assign features to each file. Features are ultimately composed into a unified *file record* mapping a single file to all of its features. The file record is then inverted, creating a mapping from each unique feature back to the exhibiting file. Inverted file records are accumulated into a persistent *inverted index*, a standard data structure for data retrieval [15, 19, 78, 85].

File features extracted by Reveal fall into three categories: file metadata, tokens, and rule hits. File metadata includes properties of the file such as its name, size, and owner. A *token* corresponds to a unique word or term that is present within the file text. A *rule hit* identifies a file that satisfies a user-defined classification rule.

3.2.1 Text Extraction. Reveal's indexing operation starts with a file path provided by an index of the local file system maintained by the Tanium client. Reveal then attempts to infer the format of the file contents in order to extract readable text. To infer the format, Reveal attempts to determine the file's MIME type [20] based on the presence of recognizable features within the first few kilobytes of data. If it determines that the file contents are in a readable format (eg. raw text, zip-deflated archive, xml-like markup, text-encoded PDF, etc), Reveal then attempts to extract readable text.

3.2.2 Tokenization. Reveal converts the extracted text into a series of tokens corresponding to the contiguous terms that are present. It does this by segmenting the raw text into words according to Unicode [16] and applies some additional standardization such as case normalization, Porter Stemming [53], and stop word removal. The resulting tokens are then encrypted using a one-way cryptographic hash. Each unique encrypted token is added as a searchable feature to the file record.

Additionally, Reveal stores positional information describing the sequence of tokens within the file. For certain tabular data formats (spreadsheets, separator-delimited text, relational databases), Reveal stores additional positional information describing the row, column, and table in which the token was found.

Encrypted tokens and positional information make up the bulk of the index data recorded for most files.

3.2.3 Rule-Based Classification. In addition to tokenization, Reveal can use the extracted text content to evaluate user-defined classification rules. A *rule* defines one or more Boolean criteria that can be evaluated against a file's metadata and/or text content. Files found to satisfy all of the criteria of a rule are considered *hits* against that rule, and rule hits are indexed as searchable file features.

A rule criterion that can be evaluated against the text extracted from a file is called a *pattern*. Regular expressions, keyword lists, and checksums are examples of patterns supported by Reveal. For example, suppose users of Reveal wanted to identify files containing credit card numbers [35]. They could first define a pattern using a regular expression to find 16-digit numbers. Matches found by the regular expression pattern could then be evaluated against a checksum pattern to find those that additionally satisfy the Luhn algorithm [42]. Separately, they could author a pattern to look for an explicit list of keywords often associated with credit card numbers, such as "Visa", "billing", and "expiration". Finally, these patterns can be combined into a rule to look for 16-digit numbers satisfying the Luhn checksum within 100 characters of a credit card keyword. Reveal would then classify any file found to satisfy this rule as containing suspected credit card information.

A rule criterion that is evaluated only against file metadata is called a *rule filter*. As shown in Figure 4, Reveal first evaluates all rule filters against file metadata to determine which rules are applicable. It then compiles the set of patterns used by at least one applicable rule. The applicable patterns are then evaluated against the file contents, and pattern matches are identified within the text. Finally, applicable rules are evaluated based on the matches found for their patterns, and a rule hit is added for each rule that is fully satisfied.

3.2.4 Rule Authoring and Validation. Reveal provides some common rules and patterns out-of-the-box, but these are intended to be refined and supplemented by users to improve signal quality according to their specific use cases. The Reveal UI allows sufficiently privileged users to author patterns and filters [75]. Reveal maintains metadata for each pattern that includes custom sample text demonstrating how the pattern was designed to work. This metadata is editable by pattern authors to aid refinement of pattern behavior. For example, a user can add examples of observed false positive or false negative matches to the pattern sample text as a guide while editing the pattern to account for these edge cases.

Additionally, Reveal provides a *validation* workflow to refine classification accuracy based on observed results [76]. *Validations* are Boolean clauses that augment existing patterns to exclude false

matches. When viewing classification results, the user is presented with any relevant pattern matches in their immediate context within the extracted file text. If the user determines a pattern match is a false positive based on a feature present in the text context, they can author a validation to ignore globally any matches that exhibit the same feature. For example, suppose a user observes a false positive credit card number that is the fractional part of a number in decimal format. They could create a validation to reject any credit card match immediately preceded by a decimal point to eliminate this class of false positive globally.

Table 2 describes a test in a lab of 121 endpoints of which 2 were seeded with files containing sensitive credit card data. Starting with the default credit card pattern, false positives were eliminated by adding cumulative *user-directed* validations to reject matches that were (1) 16 digits preceded by decimal or (2) groups of 4 digits separated by spaces, (3) tabs, or (4) semicolons. Adding validations such as (2) may reduce false positives but increase false negatives.

Table 2: Reducing False Positives using Validations

Validations Added	0	1	2	3	4
Endpoints Found	25	14	7	4	2
Files Matched	134	26	21	18	16
Text Matches	2820	1694	684	345	335

3.2.5 Machine Learning Based Classification. In production deployments, Reveal allows users to author rule sets (i.e., rule-based classifiers) for targeting and identifying specific sensitive data (Section 3.2.3). Reveal's architecture is also compatible with machine learning (ML) technology and the application of ML-based classifiers. After text extraction (Section 3.2.1), feature vectors distilled from text content may be evaluated with respect to pre-trained ML models. Standard classifier models for text classification [4] include support vector machines [28] and deep neural networks [46].

For binary classification tasks such as distinguishing between sensitive vs. not-sensitive text, and for general semantic tasks (e.g., classifying files as résumé, financial document, or legal text), ML-based classifiers provide helpful automation. However, to train highly accurate ML models with few false positives and false negatives, it is beneficial to have adequate amounts of labeled data. A full treatment of various issues (e.g., class imbalance, customer data access) is outside the scope of the present systems-level paper.

3.2.6 Endpoint Resource Management. Under the federated search model, there is nontrivial processing that must take place locally on the endpoints where relevant files reside. These are machines like workstations and file servers that serve important functions that have nothing to do with Reveal. Reveal must control its utilization of local resources so as not to interfere with the primary functionality of the device. To control endpoint impact, the Reveal agent runs at low CPU priority and imposes configurable limits on its usage of local compute, memory, disk I/O, and storage resources as follows:

- **Compute:** The Reveal agent limits its CPU usage to a configurable fraction of the available compute. Compute is enforced using a *look-behind throttle* in which Reveal performs a discrete unit of work and then sleeps for a fixed multiple of the compute used. Given a compute limit $c \in [0, 1]$, if t is the compute used by Reveal work, Reveal sleeps for $b = t\frac{1-c}{c}$. Discrete work units are bounded and typically complete in under $100ms$.
- **Disk I/O:** Disk read and write operations are limited in terms of maximum data size. Large I/O operations are broken up into operations no larger than the maximum read/write size and added to a throttled I/O queue. File handles are not retained between operations. Reveal employs a unidirectional stream when reading user files, ensuring that each bit is read at most once. User files are never modified or deleted by Reveal. Additionally, all of Reveal's persistent data adheres to a *write once* paradigm, meaning Reveal files can be deleted, but cannot be modified after they are written.
- **Memory:** The Reveal agent limits memory usage by imposing caps on the amount of working memory available during content extraction as well as the total amount of extracted content per file. If either of these limits is reached when processing a user file, Reveal will stop extracting content and record that the file was too large to process fully. Additionally, Reveal limits the amount of index data that can be buffered in memory before being written to disk.
- **Storage:** Reveal enforces a maximum size for its persistent data on disk and a minimum amount of free space in the volume on which it is installed. If either of these limits is breached, Reveal will not write new data to disk and deletes document records until the footprint is brought into compliance.

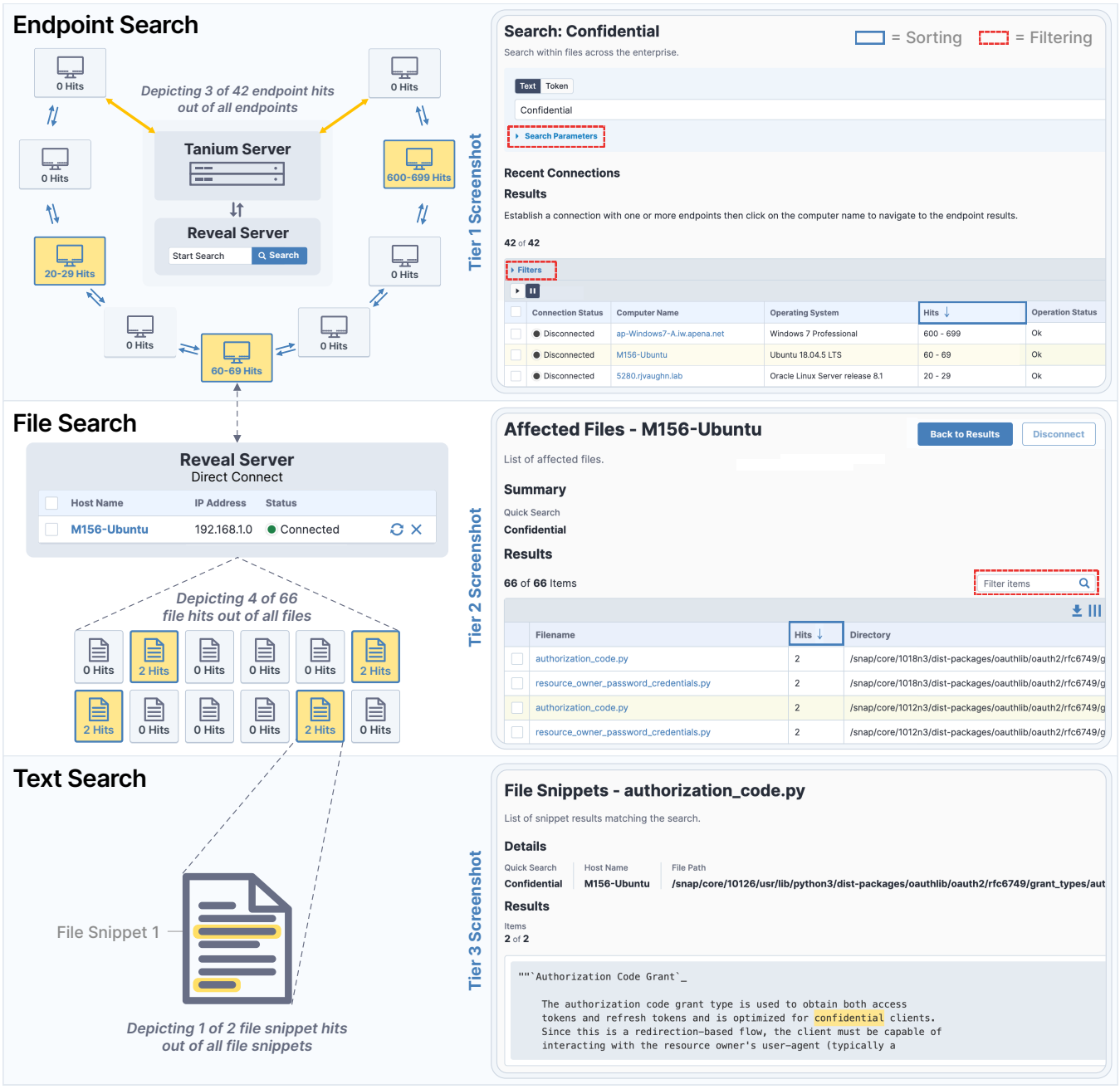
The preceding controls for resource utilization are designed to keep Reveal from adversely impacting the endpoints on which it runs. If Reveal fails to operate within the configured parameters on an endpoint, it will notify Reveal administrators for remediation. Additionally, Reveal records live telemetry documenting its performance and usage of local resources over time.

3.3 Data Security

Reveal's risk of exposing sensitive user data is reduced by following a "data at rest" paradigm: index data only ever exists on machines on which that content was found. Access to the index data is restricted to only the Reveal domain user.

3.3.1 Encryption of Queries. By necessity, a Reveal query generates network traffic including at least the tokens being queried and the query results from each endpoint. As a security measure, all token values used by Reveal are encrypted using a one-way cryptographic hash. Encryption is applied during indexing on the endpoint and during query creation in the Reveal UI. Internally, queries are processed using only encrypted token values, and unencrypted content is never persisted or transmitted. Thus, even if a Reveal index or search query were compromised, it would not expose sensitive user content.

For the tokens to be compatible, the full tokenization process including encryption must be identical across the environment. To enforce token compatibility, each query request includes a *descriptor* value that is computed deterministically as a comprehensive checksum of the tokenization process used to generate the query tokens. The Reveal endpoint agent computes the same token descriptor and includes it as a searchable file feature during indexing. The Reveal endpoint agent will only evaluate an incoming query against its indexed files if the descriptors match.



Conceptual & Visual Design Credit: Christopher Poos

Figure 5: Reveal's multi-tier interactive workflow for query processing consists of endpoint, file, and text search.

3.3.2 Salt Values. An *hash salt* is a randomly generated value incorporated into a hashing operation that deterministically influences the result [3, 22, 65]. Reveal's token encryption incorporates a shared secret salt value that is unique to the Reveal deployment. The salt value influences the token descriptor by virtue of influencing the encryption hash, so tokens generated with different salt values will have different descriptor values and will not be evaluated against each other. This means tokens are not compatible between environments, so Reveal data is not reversible by transplantation into a compromised Reveal environment.

4 QUERY PROCESSING

The search engine architecture described in Section 3 accommodates different types of search queries and levels of detail for search

results. All search queries are generated by the Reveal service and evaluated by Reveal endpoint agents.

4.1 Multi-Tier Hierarchical Workflow

A multi-tier workflow for query processing is illustrated in Figure 5. For this specific example, we describe a workflow for investigating the results of an ad hoc search for the keyword "Confidential".

4.1.1 Endpoint Search. In the first tier of query processing, immediately after a user invokes a keyword search query via the front-end UI, Reveal creates a Tanium question corresponding to the query and deploys it to all endpoints. Each endpoint agent processes the query to generate numeric score values for its indexed files, and then uses these file scores to compute a single endpoint score value to include in the query response. Endpoint score can be computed in a variety of ways depending on the query type and user needs, for example as the sum of the file scores, the maximal file score, or the number of files with scores above a threshold. Endpoint search results are displayed to the user as a list of endpoints in descending order by endpoint score. Figure 5 shows the top 3 of 42 matching endpoints in a test network sorted by the number of files found (i.e., file hits) containing the term "Confidential".

4.1.2 File Search. In the second tier of query processing, the user can select one of the matching endpoints to explore its results (i.e., affected files) in greater detail. A direct two-way encrypted network connection is opened between the Reveal service and the Reveal agent on the selected endpoint. Figure 5 shows that the user selects the endpoint "M156-Ubuntu" with IP address `192.168.1.0`. Using its direct connection, Reveal invokes a search query for the same keyword "Confidential" at the granularity of files. The endpoint returns a list of matching files in descending order by file score. Figure 5 shows the top 4 of 66 matching files sorted by the number of times "Confidential" appears (i.e., snippet hits).

4.1.3 Text Search. In the third tier of query processing, the user can select one of the matching files to view matching text in the file. Figure 5 shows that the user selects the file `authorization_code.py`. Using its established direct connection once again, Reveal issues a search query to the endpoint to extract matching snippets of text from the selected file. Figure 5 depicts 1 of 2 snippets of text that contain the keyword "Confidential".

As mentioned in Section 3.3, due to encryption applied to tokens, it is not possible to reconstitute file text from a Reveal index. Therefore, Reveal must handle the fine-grained browsing of text (Tier 3 of Figure 5) differently. The Reveal endpoint agent reads the selected file directly from disk and searches its contents in memory. Snippets of file text are returned directly from the endpoint to the Reveal service via the existing encrypted direct connection and forwarded directly to the UI of the user performing the search. In this way, we ensure that search results are up-to-date, and that potentially sensitive information is available only to the authorized user who requested it.

4.1.4 Multi-Tier Search Filtering. As shown in Figure 5 highlighted in red, when constructing a search query, the user can optionally include search parameters to specify which endpoints to search according to their responses to Tanium sensors and which files to consider on those endpoints based on the file metadata. For example, a user could limit their search only to machines with a Windows operating system, located in the United States, with Google Chrome installed; and only consider files under `C:\Users` that have been modified within the last week and classified as "sensitive" by a Reveal rule. Quantitatively, for the test network in Figure 5, restricting to `.PDF` format and further narrowing search to Windows machines resulted in 13 and 4 endpoint hits respectively.

The grid of endpoint and file search results can also be independently sorted and filtered as highlighted in Figure 5 Tier 1, 2.

4.2 Query Types Used In Production

Reveal can evaluate different types of search queries that are highly effective and practical in real-world commercial deployments.

4.2.1 Boolean Query. A Boolean query looks for the presence or absence of specific file features. The keyword query for the word "Confidential" in Figure 5 is one prime example. In addition, Boolean queries can be composed (e.g., in conjunctive normal form) with other Boolean queries, and can include files hit by (i.e., matching) a particular rule. Algebraically combined Boolean queries are evaluated by applying set operations to sets of files matching each of the requested file features. Typically, Reveal's evaluation of generalized Boolean queries is very efficient.

4.2.2 Phrase Query. A phrase query looks for two or more terms in a specific sequence within the file text. The evaluation of a phrase query requires positional information about the tokens in a file. Thus, the indexing process described in Section 3.2 must extract and store positional information. As one example of a phrase query in Reveal, a user may search for the phrase "top secret". Since this query requires sequential constraints to be satisfied in addition to verifying the presence of individual tokens, it is more computationally expensive to evaluate compared to a Boolean query.

4.2.3 Similarity Query. Reveal's architecture supports a form of similarity search [67]. A similarity query is expressed as a weighted vector of file features and is evaluated relative to the feature vector for each indexed file. Weights associated to file features may be defined in various ways; e.g., based on term frequency values $TF_f(t)$ and global domain frequencies $DF(t)$ for each token t (expressed here without a denominator for normalization):

$$TF_f(t) = \text{number of times } t \text{ appears in local file } f.$$
$$DF(t) = \text{number of files in which } t \text{ appears globally.}$$

The Reveal service uses statistical methods to generate a global dictionary of estimated domain frequency values based on random samples taken from endpoints. Reveal can then score files relative to each other by applying standard techniques (e.g., TF-IDF, BM25, cosine similarity scoring algorithms [26, 27, 64, 77]). Cosine similarities can be evaluated efficiently by computing dot-products between feature vectors. Sorted scores provide a ranking of results.

5 EVALUATION AND EXPERIMENTS

Since Reveal is designed to be deployed in massively distributed production environments, it is important to analyze experiments conducted on individual endpoint devices, and also understand metrics collected from large-scale, live, production networks.

5.1 Individual Endpoint Experiments

Table 3: Baseline Disk Usage For Different OS

Operating System	Files Indexed	Data Read	Content Extracted	Index Size
MacOS 10.10	343k	950 MB	660 MB	445 MB
Windows 10	147k	670 MB	310 MB	200 MB
CentOS 7.3	41k	140 MB	130 MB	83 MB

Table 4: Data Parse Rates On A Single Core 2GHz CPU

CPU Cap (%)	Parse Rate (MB/s)	CentOS Parse Time	Win Parse Time	Mac Parse Time	Churn Capacity (MB/Hr)
5%	0.03 - 0.07	35 min	130 min	275 min	≈ 145
25%	0.14 - 0.25	10 min	40 min	75 min	≈ 650
75%	0.40 - 1.0	3 min	12 min	25 min	≈ 2000

Reveal's search and indexing system must operate as a background process on computers that still need to accomplish their primary function. Reveal must be a good steward of system resources and should be mostly invisible to the endpoint's user. As described in Section 3.2.6, Reveal must be tuned not to be obstructive. The following endpoint parameters can be tuned to control Reveal's impact on endpoint resources:

- `max_CPU_usage`: [default: 5%] The maximum percentage of CPU time available to Reveal across all available cores.
- `max_database_size`: [default: 4 GB] The maximum local storage space allowed for Reveal's persistent data.
- `min_available_disk_space`: [default: 8 GB] The amount of free disk space required for Reveal to perform indexing.
- `max_content_size`: [default: 32 MB] The maximum amount of text content that Reveal is permitted to extract per file. Remaining content will be ignored beyond this limit.
- `max_file_parse_size`: [default: 32 MB] The maximum amount of additional memory available for the purpose of extracting content. For example, this limits the amount of compressed data that can be buffered for decompression.
- `file_buffer_size`: [default: 8 KB] The maximum data size per disk I/O operation.
- `max_merge_docs`: [default: 10000] The maximum number of inverted file records that can be written to the inverse index as a single operation. This limits the amount of index data that can be buffered in memory before being written to disk.

5.1.1 Baseline File Count. We've tested and gathered statistics on the behavior of the Reveal endpoint agent on 3 virtual machines, each provisioned with an out-of-the-box operating system and a small set of additional seed files. On each machine, we ran the Reveal endpoint agent until it had indexed everything on the local drive. Table 3 shows for each machine the number of files that ended up in the Reveal index, the total amount of user data read from disk, the total amount of text extracted from that data, and the resulting size of the Reveal index on disk. This provides a baseline for Reveal's index data footprint and how this relates to the amount of data represented.

Table 5: Wikipedia Index Statistics

Description	Indexing the Wikipedia Corpus
Corpus Contents	5,325,951 Files
Corpus Size	122 GB (131,187,937,280 bytes)
Text Extracted	18 GB (19,733,555,872 bytes)
Files Indexed	5,305,603 (20,348 failed)
Index Size	46 GB (50,144,695,764 bytes)
Unique Tokens	1,849,541 (615,585 in multiple files)
Indexing time	9 Hours (31,596 Sec) @ 10% CPU

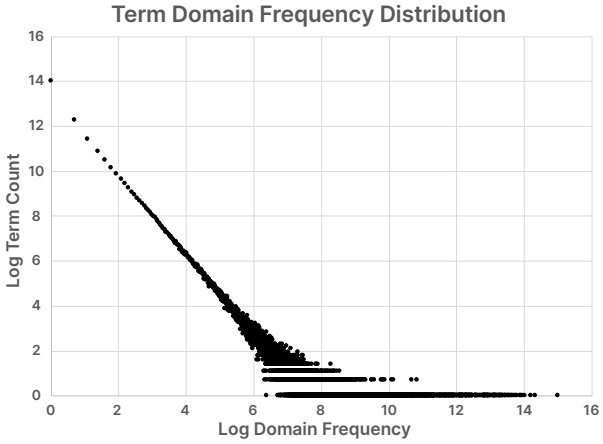


Figure 6: The number of unique tokens (y-axis) appearing at each domain frequency (x-axis) within the Wikipedia corpus, logarithmically scaled.

5.1.2 Parse Rate and Indexing Rate. A primary metric by which we can observe how Reveal is performing on an endpoint is the rate at which Reveal indexes files. We measure **indexing rate** as the number of files indexed divided by the time spent indexing. The indexing rate influences how long it takes Reveal to index all of the files on an endpoint and how long it takes to reflect file changes. The indexing rate is typically proportional to the amount of CPU Reveal is allowed to use.

The time it takes Reveal to index a file can vary based on many factors such as the underlying endpoint capabilities, properties of the file being indexed, and Reveal's local configuration. In addition to the direct cost of indexing files, there is additional overhead that influences the indexing rate such as time taken initializating the indexer, determining which files to index, and writing to disk.

Similar to indexing rate, we can measure the rate with which Reveal indexes text. We measure **parse rate** as the total amount of text content extracted in bytes divided by the time spent indexing. In practice, parse rate is often a more robust metric than indexing rate due to normalizing some of the variability between files.

Either of the above metrics can be used to estimate the time it will take Reveal to fully index a system and how much file data

Table 6: Reveal Federated Indexing Statistics For Two Large-Scale Enterprise Sub-Networks

Statistics	Enterprise A	Enterprise B
Reveal Endpoints	1273	4236
Files Indexed	427 Million	1.05 Billion
Reveal Index Size	**Distribution A**	**Distribution B**
0-10MB	222 (17.4%)	42 (1.0%)
10-100MB	16 (1.3%)	47 (1.1%)
100-500MB	79 (6.2%)	53 (1.3%)
500MB-1GB	94 (7.4%)	300 (7.1%)
1-2GB	259 (20.3%)	1777 (41.9%)
2-5GB	603 (47.4%)	2017 (47.6%)
Incomplete Data		
With Dropped Files	3 (0.2%)	90 (2.1%)
Without Dropped Files	1270 (99.8%)	4146 (97.9%)

churn Reveal can handle. If it is too slow, Reveal will not be able to keep up with file changes taking place on the endpoint. We define **churn capacity** as the maximum rate at which file data can change without the Reveal indexer falling behind. Reveal is considered healthy if it does not interfere with the endpoint's primary use and the rate of file data churn does not regularly exceed the churn capacity. See Table 4 for various churn capacities associated with our test endpoints.

5.1.3 Full Scan Performance. Reveal occasionally has to build its index data from scratch. A full scan occurs at installation and following significant changes to Reveal's requirements, such as when rules are added. As shown in Table 4, even at 5% CPU all test machines were able to complete a full system scan in under 5 hours of parse time.

On a virtual machine with a single 2 Ghz CPU and 500 MB of RAM, we tested 3 different CPU caps: 5%, 25%, and 75%. We observed that indexing rate decreases by an order of magnitude when there is a small set of files that need to be parsed, suggesting that significant compute is being used by overhead required by the Reveal indexer. Table 4 shows observed parse rate ranges on a virtual machine on a shared server.

5.1.4 Wikipedia Corpus. To evaluate the behavior of the Reveal indexer on a large corpus, we used the Reveal agent to index all English language articles in Wikipedia as of March 2019. The entire Wikipedia corpus consists of approximately 5 million articles and 49 million wiki pages, comprising over 100 GB of uncompressed data [83]. We populated a drive with one file for each article using Wikipedia's default XML format. We then used a Reveal agent to generate a Reveal index for this corpus. Reveal ignored all XML markup and tokenized the article text as described in Section 3.2.2.

Table 5 shows that Reveal was able to extract 18 GB of text content from 5 million files and generated 46 GB of indexed file data after 9 hours running at 10% CPU. This translates to a indexing rate of about 168 files per second and a parse rate of about 2 GB per hour.

Of the unique tokens present in the corpus, 2/3 appeared only in a single article (i.e., domain frequency of 1). As shown in Figure 6, the number of unique terms with a particular domain frequency conforms to a power law distribution. The number of unique terms t that each appear in exactly n articles conforms to $t = an^{-k}$ for some fixed $a, k \in (0, \infty)$.

Table 7: Network Scale at the time queried for Figure 7

Enterprise	A	B	C	D	E
Available Endpoints	1326	4262	15142	21940	161333
Reveal Enabled	1325	4236	8987	13317	3439

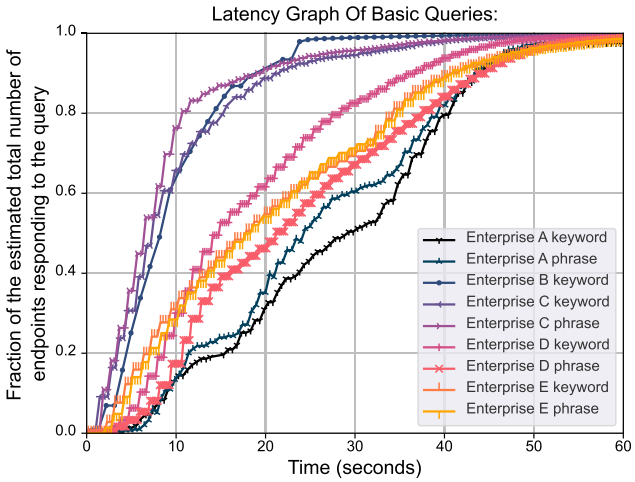


Figure 7: Latency of Reveal queries in production.

5.2 Production Deployments

In this section, we present data gathered from Reveal deployments at two anonymized live production environments.

5.2.1 Reveal Index Size. Table 6 shows comparative statistics about Reveal deployments gathered using a live Tanium question. This reflects the state of each environment at the time the question was asked, meaning only machines that were online and responsive at that time are represented. Enterprise-wide, they reported corpora of no less than 427 million and 1.05 billion indexed files respectively.

Table 6 includes the distribution of index size on disk from the endpoints available at the time of our query. This yields a total Reveal index footprint in the terabytes in each case. Based on our observations in Table 3, we'd infer that the total scale of indexed content is at the same order of magnitude.

Most endpoints in both environments have indexes larger than 1 GB. The Reveal data size is capped at 4 GB per endpoint in both of these environments. Table 6 shows that 2.1% of endpoints in Enterprise B have reached the disk usage limit and have had to drop document records from their indexes.

5.2.2 Individual Operation Latency. Reveal's index is designed for efficient searching, but evaluating a query carries a nonzero cost. Operations that require user input, notably ad hoc text queries, cannot be cached ahead of time and must be evaluated on demand.

The time taken by endpoints to evaluate a query influences its overall latency.

Figure 7 shows the latency of ad hoc queries in production environments. In each case, more than 97% of responses were received within 60 seconds, in keeping with the expected scalability of the Tanium platform[69]. As observed, this behavior scales to tens of thousands of Reveal enabled endpoints without increased latency.

6 OPERATIONAL EXPERIENCES

Since its commercial release in 2019, Tanium Reveal has been licensed on over 3 million endpoints and has been deployed in production to client networks across a variety of sectors. In this section we discuss several lessons learned based on customer experiences operating Reveal in the field.

- *Sensitive Data Discovery:* Many Reveal customers use this technology to monitor various types of sensitive data. One common use case has been institutional compliance with data privacy regulations such as PCI [51], GDPR [81], and HIPAA [17]. While Reveal does not modify user data on endpoints, many customers use it as part of a risk remediation workflow to report on their overall compliance posture, identify data security problems, and verify when remediation has taken place. Compliance users have made effective use of Reveal's rule based classification to identify instances of noncompliance and of Reveal's ad hoc text search to respond to GDPR data removal requests.
  One of the most common challenges faced by Reveal users has been the development of effective rule and pattern content for sensitive data discovery. This has driven much of the design for the tooling around authoring and testing of classification content as described in Section 3.2.4, as well as the exploration into incorporation of other technologies such as machine learning classifiers as mentioned in Section 3.2.5.
- *Data Age:* As observed in Section 5.2.2, Reveal processes queries and returns results within about a minute, so the age of the data is effectively determined by how quickly file changes are picked up by the indexer on each endpoint. In practice, Reveal is generally able to surface those changes within a few hours. In some cases, users wanted certain changes to be reflected more quickly, such as after remediating a critical vulnerability. In response, we provided a mechanism for users to prioritize specific files for re-indexing.
- *Endpoint Impact:* Reveal's endpoint agent process contends for resources with any number of critical operations. Therefore, justifiably, endpoint impact is a valid and significant concern for Reveal users. While resource usage issues are rare, the Reveal endpoint agent has a conspicuousness that warrants active monitoring. Effective endpoint telemetry has been extremely important, even though the practical measures described in Section 3.2.6 for controlling endpoint impact have been sufficient in general. Telemetric information has been valuable not only for managing Reveal agents (e.g., tuning configuration parameters, surfacing unhealthy agents, identifying software bugs, etc.), but also as a tool to demonstrate that Reveal is operating appropriately when unrelated problems occur.

7 RELATED TECHNOLOGY

- *Centralized Web Search:* Throughout the history of the Internet [58], centralized search architectures have provided speedy access to ubiquitous information (e.g., Google's hypertexual Web search engine in 1998 [8]). Search engines have augmented text search with visual search (e.g., Microsoft Bing [34]), and have adopted data structures such as knowledge graphs [18] to represent the semantic web of data [32]. The ethics of monopoly over centralized data have been debated [37], and have resulted in calls for government regulations on data [23, 81].
- *Social Search:* The use of social interactions for search has proliferated [33]. Amazon's recommendations and search results are based on *popular* items [41]. Facebook's graph search is supported by optimized data structures for social graphs [44, 66]. Social search engines benefit from machine learning algorithms applied on centralized data, but have data privacy challenges [21].
- *Peer-to-Peer File Search:* Peer-to-peer systems do not require central coordination of network nodes [57, 68]. BitTorrent's file search [55, 56], Freenet [12], and YaCy [82] employ distributed hash tables (DHTs) [84] for data accessibility at scale. While decentralized, DHTs cannot guarantee that data remain local.
- *IoT and 5G Wireless:* With the roll-out of 5G wireless networks, the emerging Internet-of-Things (IoT) is projected to connect billions of endpoint devices [25, 39]. Next-generation wireless networks are designed for ultra low-latency communication. Reveal's federated search and resource-management on devices could potentially inform the design of an IoT Search Engine [79].
- *Federated Machine Learning:* Federated machine learning can be applied without propagating private data due to the exchange, sharing, and updating of privacy-preserving *models* (e.g., multilayer neural network models) [7, 45]. Similarly, Reveal does not propagate sensitive data, adopting a "data-at-rest" approach.

8 CONCLUSION

We have designed, engineered, and analyzed an operational federated search engine called Tanium Reveal which harnesses the Tanium platform for network communication. Reveal is capable of indexing and searching the contents of billions of unstructured heterogeneous data files distributed on endpoints in enterprise networks. Reveal's index information about private data remains on the machine where that data resides. Unencrypted sensitive information is neither persisted nor transmitted by Reveal.

Our experiments, results, and analysis show that federated indexing and search are achievable at scale in production. Specifically, we demonstrated federated indexing of 1.05 billion files stored across 4236 endpoints in one sub-network, and measured search query latency within 60s for that corpus. Having presented the performance and operational aspects of Reveal, and having demonstrated the efficacy of several types of search queries in this architecture, we believe the next horizon is to automate the search process further by implementing general semantic search capabilities.

ACKNOWLEDGMENTS

Professional credit is due to Christopher Poos for the visual design, user experience (UX) design, and conceptual refinement of all figures in this paper.

REFERENCES

[1] Alation, Inc. 2021. Alation: The Industry's Leading Data Catalog. Retrieved May 08, 2021 from https://www.alation.com/

[2] Jaime Arguello. 2012. Federated Search in Heterogeneous Environments. *SIGIR Forum* 46, 1 (May 2012), 78–79. https://doi.org/10.1145/2215676.2215686

[3] Dan Arias. 2021. Adding Salt to Hashing: A Better Way to Store Passwords. Retrieved May 04, 2021 from https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/

[4] B. Baharudin, Lam Hong Lee, and K. Khan. 2010. A Review of Machine Learning Algorithms for Text-Documents Classification. *Journal of Advances in Information Technology* 1 (2010), 4–20.

[5] Michael W. Berry and Murray Browne. 2005. *Understanding Search Engines: Mathematical Modeling and Text Retrieval (Software, Environments, Tools), Second Edition.* Society for Industrial and Applied Mathematics, USA.

[6] Paolo Boldi, Andrea Marino, Massimo Santini, and Sebastiano Vigna. 2018. BUbiNG: Massive Crawling for the Masses. *ACM Transactions on the Web* 12, 2, Article 12 (June 2018), 26 pages. https://doi.org/10.1145/3160017

[7] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé M Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. https://arxiv.org/abs/1902.01046 Proceedings of the Second Conference on Machine Learning and Systems (MLSys).

[8] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems* 30, 1 (1998), 107–117. Proceedings of the Seventh International World Wide Web Conference.

[9] Rajkumar Buyya and Satish Narayana Srirama (Eds.). 2019. *Fog and Edge Computing*. Wiley, USA. https://doi.org/10.1002/9781119525080

[10] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. 1999. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks* 31, 11 (1999), 1623 – 1640. http://www.sciencedirect.com/science/article/pii/S1389128699000523

[11] Devji Chhanga and Xitij Shukla. 2016. Fossick: An implementation of federated search engine. *International Journal Of Computer Science Engineering And Information Technology Research (IJCSEITR)* 6 (2016), 69–78.

[12] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. 2001. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*, Hannes Federrath (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 46–66. https://doi.org/10.1007/3-540-44702-4_4

[13] Adrian Colyer. 2020. Helios: hyperscale indexing for the cloud & edge (part II). Retrieved April 27, 2021 from https://blog.acolyer.org/2020/11/02/helios-part-ii/

[14] Adrian Colyer. 2020. Helios: hyperscale indexing for the cloud & edge – part 1. Retrieved April 27, 2021 from https://blog.acolyer.org/2020/10/26/helios-part-1/

[15] Bruce Croft, Donald Metzler, and Trevor Strohman. 2009. *Search Engines: Information Retrieval in Practice* (1st ed.). Addison-Wesley Publishing Company, USA.

[16] Mark Davis. 2019. Unicode® Standard Annex 29: Unicode Text Segmentation. Retrieved July 26, 2021 from https://unicode.org/reports/tr29/

[17] P.F. Edemekong, P. Annamaraju, and M.J. Haydel. 2020. *Health Insurance Portability and Accountability Act.* StatPearls [Internet]. Retrieved January 10, 2021 from https://www.ncbi.nlm.nih.gov/books/NBK500019/

[18] Dieter Fensel, Umutcan Simsek, Kevin Angele, Elwin Huaman, Kaerle Elias, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, and Alexander Wahler. 2020. *Knowledge Graphs - Methodology, Tools and Selected Use Cases.* Springer, Switzerland.

[19] Marcus Fontoura, Vanja Josifovski, Jinhui Liu, Srihari Venkatesan, Xiangfei Zhu, and Jason Zien. 2011. Evaluation Strategies for Top-k Queries over Memory-Resident Inverted Indexes. *Proceedings of the VLDB Endowment* 4, 12 (Aug. 2011), 1213–1224. https://doi.org/10.14778/3402755.3402756

[20] N. Freed and N. Borenstein. 1996. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.* Internet Engineering Task Force. Retrieved July 26, 2021 from https://tools.ietf.org/html/rfc2045

[21] Michael Fuller. 2019. Big data and the Facebook scandal: Issues and responses. *Theology* 122, 1 (2019), 14–21. https://doi.org/10.1177/0040571X18805908 arXiv:https://doi.org/10.1177/0040571X18805908

[22] P. Gauravaram. 2012. Security Analysis of salt||password Hashes. In *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT).* IEEE Computer Society, Los Alamitos, CA, USA, 25–30. https://doi.org/10.1109/ACSAT.2012.49

[23] Michelle Goddard. 2017. The EU General Data Protection Regulation (GDPR): European Regulation that has a Global Impact. *International Journal of Market Research* 59, 6 (2017), 703–705. https://doi.org/10.2501/IJMR-2017-050

[24] Google contributors. 2020. How Google search works. Retrieved January 10, 2020 from https://www.google.com/search/howsearchworks/

[25] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Computer Systems* 29, 7 (Sept. 2013), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

[26] Marios Hadjieleftheriou, Xiaohui Yu, Nick Koudas, and Divesh Srivastava. 2008. Hashed Samples: Selectivity Estimators for Set Similarity Selection Queries. *Proceedings of the VLDB Endowment* 1, 1 (Aug. 2008), 201–212. https://doi.org/10.14778/1453856.1453883

[27] Jiawei Han, Micheline Kamber, and Jian Pei. 2011. *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[28] Michael Hart, Pratyusa Manadhata, and Rob Johnson. 2011. Text Classification for Data Loss Prevention. In *Privacy Enhancing Technologies*, Simone Fischer-Hübner and Nicholas Hopper (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 18–37.

[29] David Hindawi, Orion Hindawi, Lisa Lippincott, and Peter Lincroft. 2016. System, security and network management using self-organizing communication orbits in distributed networks. Retrieved January 10, 2020 from https://patents.google.com/patent/US10136415B2/en

[30] Orion Hindawi, David Hindawi, Peter Lincroft, and Lisa Lippincott. 2011. Large-scale network querying and reporting. Retrieved January 10, 2020 from https://patents.google.com/patent/US8904039B1/en

[31] Hitachi Vantara. 2021. Lumada Data Catalog. Retrieved May 08, 2021 from https://www.hitachivantara.com/en-us/products/data-management-analytics/lumada-data-catalog.html

[32] Aidan Hogan. 2020. The Semantic Web: Two decades on. *Semantic Web* 11, 1 (2020), 169–185. https://doi.org/10.3233/SW-190387

[33] Damon Horowitz and Sepandar D. Kamvar. 2010. The Anatomy of a Large-Scale Social Search Engine. In *Proceedings of the 19th International Conference on World Wide Web* (Raleigh, North Carolina, USA) *(WWW '10)*. Association for Computing Machinery, New York, NY, USA, 431–440.

[34] Houdong Hu, Yan Wang, Linjun Yang, Pavel Komlev, Li Huang, Xi (Stephen) Chen, Jiapei Huang, Ye Wu, Meenaz Merchant, and Arun Sacheti. 2018. Web-Scale Responsive Visual Search at Bing. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, Yike Guo and Faisal Farooq (Eds.). ACM, New York, NY, USA, 359–367. https://doi.org/10.1145/3219819.3219843

[35] International Organization for Standardization 2017. *Identification cards — Identification of issuers — Part 1: Numbering system.* International Organization for Standardization. Retrieved July 08, 2020 from https://www.iso.org/standard/70484.html

[36] Internet Live Stats. 2021. Google Search Statistics. Retrieved February 05, 2021 from https://www.internetlivestats.com/google-search-statistics/

[37] K. Juel Vang. 2013. Ethics of Google's Knowledge Graph: Some Considerations. *Journal of Information, Communication and Ethics in Society* 11, 4 (2013), 245–260.

[38] Abe Korah and Erin Dorris Cassidy. 2010. Students and Federated Searching: A Survey of Use and Satisfaction. *Reference & User Services Quarterly* 49, 4 (2010), 325–332. http://www.jstor.org/stable/20865293

[39] Danh Le-Phuoc, Hoan Nguyen Mau Quoc, Hung Ngo Quoc, Tuan Tran Nhat, and Manfred Hauswirth. 2016. The Graph of Things: A step towards the Live Knowledge Graph of connected things. *Journal of Web Semantics* 37-38 (2016), 25 – 35. https://doi.org/10.1016/j.websem.2016.02.003

[40] Chao Li, Yushu Xue, Jing Wang, Weigong Zhang, and Tao Li. 2018. Edge-Oriented Computing Paradigms: A Survey on Architecture Design and System Management. *ACM Comput. Surv.* 51, 2, Article 39 (April 2018), 34 pages. https://doi.org/10.1145/3154815

[41] G. Linden, B. Smith, and J. York. 2003. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing* 7, 1 (Jan 2003), 76–80.

[42] Hans P. Luhn. 1960. Computer for Verifying Numbers. US Patent No. 2,950,048, Filed Jan. 6th., 1954, Issued Aug. 23rd., 1960.

[43] I.R. Management Association. 2020. *Digital Libraries and Institutional Repositories: Breakthroughs in Research and Practice.* IGI Global, Hershey PA, USA. https://doi.org/10.4018/978-1-7998-2463-3

[44] Yoshinori Matsunobu, Siying Dong, and Herman Lee. 2020. MyRocks: LSM-Tree Database Storage Engine Serving Facebook's Social Graph. *Proceedings of the VLDB Endowment* 13, 12 (Aug. 2020), 3217–3230. https://doi.org/10.14778/3415478.3415546

[45] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017 (Proceedings of Machine Learning Research)*, Aarti Singh and Xiaojin (Jerry) Zhu (Eds.), Vol. 54. PMLR, Fort Lauderdale, FL, USA, 1273–1282. http://proceedings.mlr.press/v54/mcmahan17a.html

[46] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. 2021. Deep Learning–Based Text Classification: A Comprehensive Review. *ACM Comput. Surv.* 54, 3, Article 62 (April 2021), 40 pages. https://doi.org/10.1145/3439726

[47] Dong Nguyen, Thomas Demeester, Dolf Trieschnigg, and Djoerd Hiemstra. 2012. Federated Search in the Wild: The Combined Power of over a Hundred Search Engines. In *Proceedings of the 21st ACM International Conference on Information*

and Knowledge Management (Maui, Hawaii, USA) *(CIKM '12)*. Association for Computing Machinery, New York, NY, USA, 1874–1878. https://doi.org/10.1145/2396761.2398535

[48] Kate T. Noerr. 2006. Muse Metasearch: Beyond Federated Searching. *Serials Review* 32, 3 (2006), 186–189. https://doi.org/10.1080/00987913.2006.10765058

[49] OCLC. 2021. WorldCat. Retrieved May 08, 2021 from https://www.oclc.org/en/worldcat.html

[50] Christopher Olston and Marc Najork. 2010. Web Crawling. *Foundations and Trends® in Information Retrieval* 4, 3 (2010), 175–246. https://doi.org/10.1561/1500000017

[51] PCI Security Standards Council, LLC 2018. *Payment Card Industry (PCI) Data Security Standard Requirements and Security Assessment Procedures Version 3.2.1 May 2018*. PCI Security Standards Council, LLC. Retrieved July 26, 2021 from https://www.pcisecuritystandards.org/document_library

[52] Miloslava Plachkinova and Chris Maurer. 2018. Teaching Case Security Breach at Target. *Journal of Information Systems Education* 29 (03 2018), 11–20.

[53] Martin Porter. 2006. The Porter Stemming Algorithm. Retrieved May 05, 2021 from https://tartarus.org/martin/PorterStemmer/

[54] Rahul Potharaju, Terry Kim, Wentao Wu, Vidip Acharya, Steve Suh, Andrew Fogarty, Apoorve Dave, Sinduja Ramanujam, Tomas Talius, Lev Novik, and Raghu Ramakrishnan. 2020. Helios: Hyperscale Indexing for the Cloud & Edge. *Proceedings of the VLDB Endowment* 13, 12 (Aug. 2020), 3231–3244. https://doi.org/10.14778/3415478.3415547

[55] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. 2005. The Bittorrent P2P File-Sharing System: Measurements and Analysis. In *Peer-to-Peer Systems IV*, Miguel Castro and Robbert van Renesse (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 205–216.

[56] Dongyu Qiu and Rayadurgam Srikant. 2004. Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. *Computer Communication Review - CCR* 34 (10 2004), 367–378.

[57] Weixiong Rao, Lei Chen, Ada Wai-Chee Fu, and Guoren Wang. 2009. Optimal resource placement in structured peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems* 21, 7 (2009), 1011–1026.

[58] Johnny Ryan. 2010. *A History of the Internet and the Digital Future* (1st ed.). Reaktion Books, London, GBR.

[59] Safari Digital. 2020. How Frequently Does Google Crawl and Index Sites? Retrieved February 05, 2021 from https://www.safaridigital.com.au/blog/google-crawl-rate/

[60] M. Satyanarayanan. 2017. The Emergence of Edge Computing. *Computer* 50, 1 (2017), 30–39. https://doi.org/10.1109/MC.2017.9

[61] Amit P. Sheth and James A. Larson. 1990. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Comput. Surv.* 22, 3 (Sept. 1990), 183–236. https://doi.org/10.1145/96602.96604

[62] Milad Shokouhi and Luo Si. 2011. Federated Search. *Foundations and Trends in Information Retrieval* 5, 1 (Jan. 2011), 1–102. https://doi.org/10.1561/1500000010

[63] McKay Smith and Garrett Mulrain. 2017. Equi-failure: The national security implications of the equifax hack and a critical proposal for reform. *Journal of National Security Law & Policy* 9 (2017), 549.

[64] Karen Spärck Jones. 1972. A Statistical Interpretation of Term Specificity and its Application in Retrieval. *Journal of Documentation* 28, 1 (01 1972), 11–21. https://doi.org/10.1108/eb026526

[65] P. Sriramya and R.A. Karthika. 2015. Providing password security by salted password hashing using Bcrypt algorithm. *ARPN Journal of Engineering and Applied Sciences* 10 (01 2015), 5551–5556.

[66] Stocky, Tom and Rasmussen, Lars. 2013. Facebook Newsroom: Introducing Graph Search Beta. Retrieved January 10, 2020 from https://about.fb.com/news/2013/01/introducing-graph-search-beta/

[67] Joshua F. Stoddard, John R. Coates, Naveen Goela, Aaron J. Tarter, and Christian L. Hunt. 2019. System and Method for Performing Similarity Search Queries in a Network. Retrieved February 15, 2020 from https://patents.google.com/patent/US20190361843A1/en

[68] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review* 31, 4 (2001), 149–160.

[69] Tanium Inc. 2020. *Tanium Architecture*. Tanium Inc. Retrieved July 26, 2021 from https://www.tanium.com/resources/tanium-endpoint-platform-architecture-product-brief/

[70] Tanium Inc. 2021. *Tanium Client User Guide*. Tanium Inc. Retrieved July 26, 2021 from https://docs.tanium.com/client/client/index.html

[71] Tanium Inc. 2021. *Tanium Console User Guide*. Tanium Inc. Retrieved July 26, 2021 from https://docs.tanium.com/platform_user/platform_user/index.html

[72] Tanium Inc. 2021. *Tanium Core Platform Deployment Guide for Windows*. Tanium Inc. Retrieved July 26, 2021 from https://docs.tanium.com/platform_install/platform_install/index.html

[73] Tanium Inc. 2021. *Tanium Interact User Guide*. Tanium Inc. Retrieved July 26, 2021 from https://docs.tanium.com/interact/interact/index.html

[74] Tanium Inc. 2021. *Tanium Reveal User Guide*. Tanium Inc. Retrieved July 26, 2021 from https://docs.tanium.com/reveal/reveal/index.html

[75] Tanium Inc. 2021. *Tanium Reveal User Guide: Creating Rules*. Tanium Inc. Retrieved July 26, 2021 from https://docs.tanium.com/reveal/reveal/creating_rules.html

[76] Tanium Inc. 2021. *Tanium Reveal User Guide: Validating Pattern Matches*. Tanium Inc. Retrieved July 26, 2021 from https://docs.tanium.com/reveal/reveal/creating_validations.html

[77] Sandeep Tata and Jignesh M. Patel. 2007. Estimating the Selectivity of Tf-Idf Based Cosine Similarity Predicates. *SIGMOD Record* 36, 2 (June 2007), 7–12. https://doi.org/10.1145/1328854.1328855

[78] Anthony Tomasic and Hector Garcia-Molina. 1993. Query Processing and Inverted Indices in Shared-Nothing Document Information Retrieval Systems. *VLDB Journal* 2, 3 (1993), 243–275. http://www.vldb.org/journal/VLDBJ2/P243.pdf

[79] Nguyen Khoi Tran, Quan Z. Sheng, M. Ali Babar, Lina Yao, Wei Emma Zhang, and Schahram Dustdar. 2019. Internet of Things Search Engine. *Commun. of the ACM* 62, 7 (June 2019), 66–73. https://doi.org/10.1145/3284763

[80] Urs Hoelzle. 2012. The Google Gospel of Speed. Retrieved February 05, 2021 from https://www.thinkwithgoogle.com/future-of-marketing/digital-transformation/the-google-gospel-of-speed-urs-hoelzle/

[81] Paul Voigt and Axel von dem Bussche. 2017. *The EU General Data Protection Regulation (GDPR): A Practical Guide* (1st ed.). Springer Publishing Company, Inc., Gewerbestrasse 11, 6330 Cham, Switzerland.

[82] Wikipedia. 2021. YaCy — Wikipedia, The Free Encyclopedia. Retrieved May 06, 2021 from http://en.wikipedia.org/w/index.php?title=YaCy&oldid=1017190185

[83] Wikipedia contributors. 2020. Wikipedia: Size of Wikipedia. Retrieved January 20, 2020 from https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia

[84] Hao Zhang, Yonggang Wen, Haiyong Xie, and Nenghai Yu. 2013. *Distributed Hash Table - Theory, Platforms and Applications*. Springer, New York, NY. https://doi.org/10.1007/978-1-4614-9008-1

[85] Justin Zobel and Alistair Moffat. 2006. Inverted Files for Text Search Engines. *ACM Comput. Surv.* 38, 2 (July 2006), 6–es. https://doi.org/10.1145/1132956.1132959