VIEW POINT

IMPLEMENTING DMS ON HADOOP - HDMS

Infosys®
Navigate your next

1 Executive Summary

A Document Management System (DMS) focuses on managing unstructured data or content using structured data. The unstructured data (or unstructured information) refers to information that either does not have a pre-defined data model or is not organized in a pre-defined manner. Most common examples are - Images, office documents, graphics and drawings, print streams, web pages, e-mail, video, rich media assets etc. A structured data in DMS is data that has been organized into a database, so that its elements (or attributes) can be used for managing unstructured data or content.

To implement DMS, a number of licensed as well as open source products are available in the market. DMS implementation starts with the shortlisting of a commercial-of-the-shelf (COTS) product. A product which fulfils most of the business requirements is considered the best suited product. The product is configured to build DMS artefacts such as taxonomy, object model, access control, ingestion routes, search and retrieve etc. A User Interface (UI) is generally a part of the product, i.e. admin UI, client UI etc. The product as well as the user interfaces are customized to fulfil the additional business requirements. DMS users perform operations such as file upload, metadata tagging, retrieval, editing, search and so on. The question is - Instead of going for a COTS product, can DMS be implemented on Hadoop? This paper conceptually examines the feasibility of implementing DMS[1] on Hadoop (HDMS).

In the following sections, first, a short introduction of DMS and Hadoop technology is briefly explained. New requirements that are getting injected into DMS space, due to factors such as the explosion in unstructured data, shift in the definition of data and record etc. are summarized. Later, key design considerations required to implement DMS on Hadoop are elaborated. While discussing the design, how proposed DMS will address traditional as well as new requirements is also outlined and highlighted in the benefits section. Lastly the conclusion of the discussion is presented.

2 Document Management System

In DMS world, the structured data is referred as metadata or attributes and unstructured data is referred as content. A major percentage (estimated at 80% or more) of the information in an organization is maintained as unstructured content, which includes valuable assets such as customer correspondence, wikis, blobs of text in a database, documents stored on the shared drives, social media posts etc. Because of the lack of structure in content, it is very difficult to search and locate the content. This difficulty is solved by the DMS where an unstructured data is referenced by the structured data which makes centralizing, managing, searching and locating the content much easier.

In an enterprise, content usually goes through different lifecycle stages, for example, capture, store, manage, deliver and lastly retire. Capture or creation is a process of ingesting or manually importing content into DMS. Once ingested, the content is securely stored. The content is then made available for searching, viewing, processing etc. Once the content is no longer viewed or accessed it is retired or archived[2]. DMS is the tool which helps organizations to effectively manage content throughout its lifecycle.

To capture metadata, a metadata template or object definition is defined in the database by creating one or more database tables. The tables are created with relevant columns (or attributes) which are helpful in providing identity and the business context to the content. When a content is created or imported in DMS, the metadata template is filled with the appropriate values. The binary file of the content is stored either on a file system (managed by DMS) or directly into a database as a blob object. If the binary file is stored on file system, then the path of binary file is stored in an attribute to create a soft link. The metadata template with appropriate attribute values together with the binary file is called a content object.

In DMS, folder objects can also be defined. Folder objects are container objects which can contain other folders and/or content objects. A folder helps in storing related or similar content in one common location. Folders are created and arranged in a specific hierarchical order to elaborate a file plan. A file plan helps to navigate and quickly locate the content of interest. When a content object is created, it can be linked to one folder or multiple folders. The content can be located by navigating to all the folders it was linked.

When a content is stored in DMS, security is enforced by defining Access Control Lists (ACL) that allows operations such as read, write, view etc. as permitted by the relevant ACL. An ACL consists of list of groups and/or users and the permission they have. ACLs are defined in the database using one or more tables. An ACL can be attached to a content object or a folder object. Each content object in the DMS has directly attached ACL or it is inherited from the folder object where the content is stored. Whenever a user performs operation such as read, update etc. on a content object, DMS checks the required user permission against the ACL, either directly attached or inherited. The operation on the object is permitted only if the user has appropriate permission on the object.

When content is managed, the main purpose is to serve the content. The content can be searched, edited or updated, versioned etc. These functions are collectively called as library functions and these are core functions of a DMS. A content object can be checked-out. When a content object is checked out by a user, only the same user can update the object. The user can change attributes and / or content. During check out, other users will get only read access to the object. Once the user checks-in the content, it is versioned, and the newly checked-in content becomes the released version of the content. This check-out and check-in functions are known as version control.

Gartner redefined ECM as Content Services. This redefinition is needed as the importance of the data, is hidden in the unstructured content, is suddenly increased due to digitization, regulations such as MiFID II, GDPR etc. Earlier ECM as an application was focusing on operationalizing the content. Now the focus is, to treat the "content as assets" or "core entity" and define the functions and features, which serve the right content at right time. This transformation has put the concept of "Headless DMS", "Lite DMS" in focus as the solutions build around these concepts offers flexibility of implementing needed features, black boxing the backend products and avoiding vendor locking. Hence there is a need to build a generic solution which can be progressively tweaked to fulfil traditional as well as new DMS requirements.

3 Apache Hadoop

There is hardly anyone in the IT industry who has not heard of Hadoop. Using Hadoop, a large amount of data can be stored and processed in parallel. In earlier days, data was generated by humans only, but now, machines along with humans are generating data, leading to petabytes of data being generated. Due to this huge volume of data, there is a growing need for distributed storage and processing, so that the data can be broken down into manageable sizes and then, at the same time, processed by multiple processor (or machines) in parallel. This is exactly what Hadoop does. A typical Hadoop installation can have multiple master nodes and data nodes. Together with master node and data node, Hadoop forms a cluster.

Hadoop ecosystem consists of components (or resources) such as HDFS, HBase, etc. and a number of open source tools such as Apache Ranger, Apache KNOX. Some of the relevant Hadoop resources or component are discussed briefly below:

3.1 Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS) is a Java-based file system for Hadoop that provides scalable and reliable data storage. It can span across hundreds of commodity servers and can store petabytes of data.

An HDFS cluster is comprised of a name node, which manages the HDFS cluster metadata, and data node that stores the actual data. While storing a content in HDFS, it internally splits the content in one or more data blocks. The data block size is configurable. If the content file size is greater than the configured size, HDFS internally splits the content into data blocks. These data blocks are then stored on data nodes. If the content size is less than what is configured, it is copied to a data block, leaving the remaining space.

The data blocks are replicated or copied to different data nodes to make the HDFS fault tolerant. If one data node is down or crashed or even if a data block is corrupted, the data can be accessed from its copy. Because of this powerful data replication feature, applications built on Hadoop need not worry about content getting corrupted and becoming garbage. Replication feature also eliminates the need of backup and restore procedures.

3.2 Apache HBase

Apache HBase[3] is an open source NoSQL database that provides real-time read and/or write access to large datasets. Real time access to large dataset, makes HBase suitable for the applications which are heavily dependent on data. HBase scales linearly to handle huge data sets with billions of rows and millions of columns. Metadata template, in HBase, can be modelled by defining the table and appropriate columns or attributes.

3.3 Apache Ranger

Apache Ranger provides a centralized platform to define, administer and manage security policies consistently across Hadoop components. Apache Ranger offers Admin Portal as user interface and RESTful server for managing policies, users and groups. It also contains the user interface for Audit queries and ad hoc reports.

Apache Ranger controls what operation a user can perform in a Hadoop cluster. For example, once a user is connected to HBase, which tables the user can access and what operations the user can perform, will be controlled by the policies defined in Apache Ranger.

4 New DMS Requirements

According to the estimates, the volume of business data worldwide, across all companies, doubles every 1.2 years, most of which is unstructured data. It is further estimated that poor data can cost businesses 20%–35% of their operating revenue and bad data or poor data quality costs $600 billion annually for US businesses alone[4]. Such a tremendous data growth is forcing DMS to manage high volume of data and adding new requirements to fulfill. Shift in definition of "Records" is also evident. Earlier only physical documents were considered as records, but due to regulatory requirements, such as GDPR, all customer data including documents and structured information is considered as records. A new question "what business value we can create from the idle content?" is on rise. All these changes are injecting new requirements in document management landscape. Some of the key new generational requirements where traditional DMS are lagging are as follows;

4.1 Any Format

Over the years, content has changed its form. Earlier word or PDF document were considered as content; today, audio and video files are also considered as content. Traditional DMS solutions offer storage of media files such as audio and video as separate solution or additional licenses are required to enable media storage. There is growing need to store documents as well as media files in single repository.

4.2 Any Type

Currently DMS, is designed to store content and its associated metadata. The solutions provided are not optimized to manage structured data. As stated earlier, due to regulation such as GDPR, today's DMS should able to manage structured data as well. This new requirement of managing structured data as records should be implemented and optimized in DMS.

4.3 Huge Volume

In traditional DMS, number of objects it can store depends on the backend database which is generally a relational database. To scale relational database for higher volumes, more infrastructure and/

or licenses are required which drives TCO up. New generational DMS should be able to support huge volume of data keeping the infra and licensing cost low and should perform better under the high volume.

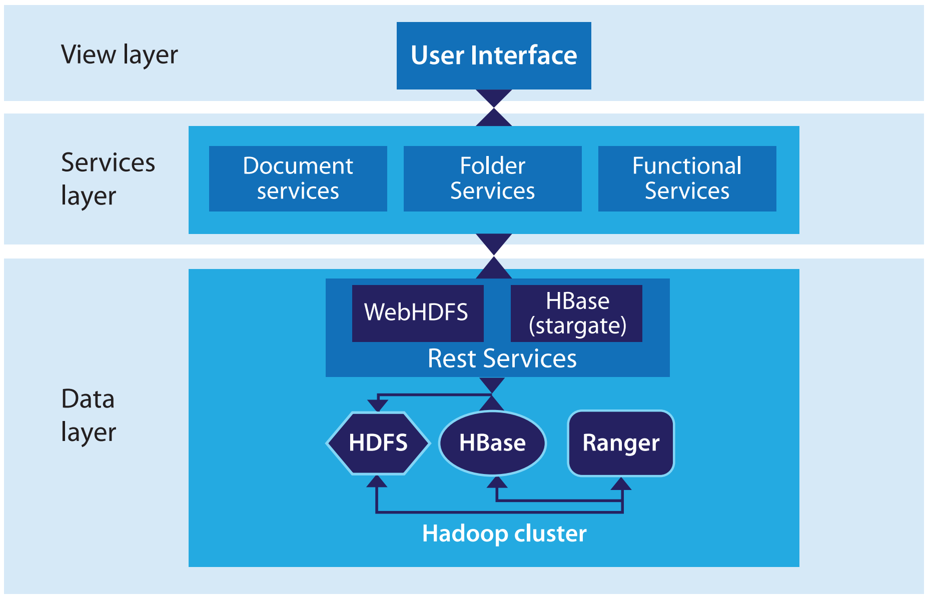
4.4 Integrated DMS

DMS stores recent information in the form of content or documents and its metadata. If this information is not used by business applications and processes, DMS will become silo application and will lose its importance or focus. Unfortunately, today most of the DMS implementation falls in this category.

Requirement of integrated DMS is not new, but in recent years, it has gained momentum as organizations are realizing the value of making right information available at right time. Traditional DMS products offer standard set of web services for integration. But the web services available by default are basic in nature and does not support end-to-end integration with business applications and processes.

4.5 In-Place Archival

Usage of content decreases with time. When the content is no longer viewed or accessed, it is archived. During archival, content is moved into archival repository. Most of the commercial archival solutions are offered as a separate product with additional license cost. It is quite possible, that the archived content is needs to be retrieved, for example, as part of some ongoing investigation. Retrieving content from such archival repository is often a sub optimal process and its cycle time depends on how content is archived in the archival repository. Some organization uses backup solution as archived solution. Retrieving a content from such archival solution is lengthy process. With focus on huge volume and need to retrieve archived content quickly, a DMS solution is needed which can fulfill regular DMS as well as archival requirements.



4.6 Cost

Licensing model for most of the DMS product is "per user" based. For an organization where user base is large, licensing cost will be significant. Licensing cost of the backend database is additional. Implementing media management and archival use cases will require additional set of products and will add up to the cost. With the infrastructure required for the commercial products, TCO of DMS with traditional product is high. DMS implementation based on subscription-based product can bring down TCO by 96%[5]. Having a common platform for implementing media management and archival will be much more cost effective than commercial product.

4.7 Content Analytics

Content Analytics is a tool for reporting statistics and for obtaining actionable insights. With the help of these actionable insights existing information can be enriched with the business intelligence or rules which, in turn, can be used in providing better user experience.

A DMS storing content will definitely have "actionable insights". Bigger the data set, better the insight. Content and data stored in DMS should be analysed to discover "actionable insights" such as business rules, which can be used in web services to make response "Smarter". Such analytics will be helpful in generating value from content. This is new requirement and DMS implementation should be aligned toward it.

4.8 Summary

With the current DMS products, to fulfil these requirements more than one product is needed leading to fragmented landscape, vendor locking, high TCO, etc. Some products offer document management and media management as separate products or add-on with additional licenses. On the other hand, some products cannot scale to huge volume without adding additional appropriate hardware, which add ups to TCO cost. Therefore, COTS products are lagging in fulfilling these new DMS requirements. Its time to address these challenges through different thought process. The new though process is to build DMS solution on a generic platform which can store petabytes of data of any type and keeping TCO lower. The platform that is being considered in this paper is, Hadoop. The solution that is proposed, is to implement DMS on Hadoop or HDMS, which is discussed in the remaining part of the paper.

5 Design Considerations

Design considerations are design activities which an application designer or architect needs to finalize before commencing actual application development. Design considerations focusing on implementing capturing, storing, managing, delivering and archiving related capabilities, in HDMS, are discussed below.

5.1 High Level Architecture

A high-level architecture of HDMS can be represented by the following diagram. It mainly consists of three layers:-

5.1.1 Data Layer

The Data layer consists of Hadoop and its components such as HDFS and HBase. HDFS will be used as file storage and HBase will be used for storing metadata and files as blob objects. Apache Ranger is required for configuring authentication and auditing.

5.1.2 Service Layer

Service layer[6] contains services developed to interact with content stored in HDMS. Services can be object specific services such as ContentServices, FolderServices and operation specific services such as RecordsMgmt services. In short, the service layer will have entire logic of interacting with the content objects and can be considered as HDMS engine.

5.1.3 View Layer

View layer consists of a user interface which enables users to interact with the HDMS. It presents content & data to users and provides menu items to perform operations, such as version, update etc. on the content and data

5.2 Data Layer

Following design activities required for configuring Hadoop and its component for HDMS implementation.

5.2.1 Defining Object Types

Metadata template or object definition consists of list of attributes that helps in capturing identity and business context of the content. The section below discusses how object types can be defined in Hadoop.

5.2.1.1 Content Objects

In Hadoop, object types can be defined by defining tables in HBase. Columns defined in the tables represent the attributes of the object definition. Since there are no relationship and joins in HBase, all attributes of an object type are defined in one table.

Some of the core attributes that are required in HDMS are content_id[7] to store unique identification number, creation_timestamp to store when content is created, created_by to store who has created the content object, content_type to store what type of document is this, etc. To store folder related attributes folder_id, content_location etc. are required. Each cell or row is created with proper attribute values in HBase represents a content object. In HBase, columns are grouped into column families. Relevant column families are defined, and attributes are put together into an appropriate column family. For example, core_attributes can be one column family and library_service can be another column family. Library_service contains attributes those are required to implement library services.

5.2.1.2 Folder Objects

Although representation and behaviour of folder object is different than that of content, object definition of a folder object is similar to that of content object definition.

Folder attributes should be defined in a different HBase table, with core attributes such as folder_id to store unique identification number, folder_name to store the name of the folder, folder_creation_timestamp to store folder creation timestamp and folder_creator to store who has created the folder. Additionally, folder object should have an attribute, such as folder_location, to store its location in folder hierarchy. For example, a folder object, with folder name "Customer Documents" created inside folder with name "HDMS", which is at root level, should folder_location attribute value be set to "HDMS/Customer Documents".

5.2.2 Security Implementation

In HDMS, security is imposed on objects to restrict access from unauthorized users. It provides allowable access, such as, read, write, delete to an authenticated user. This is also referred as fine-grained authorization. Auditing is an additional security function to keep a detailed log of what operations are performed by users, on which content and when. To implement auditing, Apache Ranger is required. Coarse-grained authorization is about controlling which user can access which resource. Coarse-grained authorization, if required, can be configured using Apache Ranger. For example, if auditing data is saved in HDFS directory, then policies can be defined in Ranger, to make sure only designated users can access the HDFS directory.

It is recommended to use Apache Knox for securing Hadoop cluster. The Knox Gateway provides a single access point for all REST and HTTP interactions with Hadoop clusters. In other words, Apache Knox can control who can access Hadoop cluster.

5.2.2.1 Authentication

Authentication is the process of determining who the user is, whereas, Authorization is the process of allowing the authenticated user what actions s/he can perform. By default, Hadoop authentication is set to "simple". Under this configuration, any user can connect to Hadoop. Therefore, to protect the Hadoop cluster from unauthorized access, it is recommended to set the authentication to "Kerberos". Once kerberized, a user connecting to Hadoop cluster needs to be authenticated. Using Apache Ranger authentication can be extended and Enterprise Active Directory (AD) or LDAP can be used. Apache Ranger can also sync users from AD or LDAP and store user list in its own database.

5.2.2.2 Fine-grained Authorization

Generally, Access Control List (ACL) is set on every object, either directly or by inheritance. When an authenticated user tries to perform an operation on an object, such as read or write, the system checks ACL associated with the object. The operation is permitted only if allowed by the ACL.

Similarly, fine-grained security control can be implemented in HBase using a coprocessor called Access Controller. A coprocessor is a code that runs inside HBase and can be restricted to a row or cell (or object) level. A configured coprocessor is capable of intercepting operations such as put, get, delete, etc., and run the code before the operation is executed.

Using this ability to execute some code before each operation, an Access Controller coprocessor can check the user rights and decide if the user can or cannot perform the operation[8].

What permissions users should have on an object can also be linked to attribute value of an object. For example, a content with security_classification as all, should be accessible to all users whereas a content with security_classification as confidential should be accessible only to users from a group. When a row (or object) is put in HBase (put operation in HBase is equivalent to create), permissions can be granted based on the attribute value. For subsequent operations, the ACL or permissions attached to the row (or object) can be checked by the coprocessor and the operation is authorized only if it is permitted by the ACL.

5.2.2.3 Auditing

When auditing is implemented, specific actions performed by a user are logged in database table or in text file. Audit serves as evidence for an operation performed by a user, on the content object.

Apache Ranger offers plug-ins for Hadoop resources such as HDFS, HBase etc. Once plug-in for the resource is installed, it generates and sends access logs to destination. The destination can be a table in relational database, log4j or HDFS. By installing Ranger plug-in for HDFS and HBase and by configuring the plug-in to capture the user actions such as create, update, etc., the auditing data can be generated and saved in the desired destination.

5.2.3 Storing Content File

When content is uploaded, it is tagged with the appropriate attribute values and the binary file is either stored in the database as blob object or in the file system. As Hadoop is not optimized for handling smaller files (<10MB), for HDMS, both approaches are required. Using medium objects (MOB)[9] feature of HBase, content with size up to 10MB can be stored as a blob object. If the size of the content being stored is greater than 10MB, the binary file can be stored in HDFS. The path of the binary file in HDFS can be set to an attribute as value, to establish soft link between the content object and its corresponding binary file.

To facilitate the storage and retrieval of a binary file, a set of additional attributes, such as storage_location (database or file system), file_size, storage_path (HDFS directory path if stored in HDFS) etc. are required to be defined in object definition.

5.2.4 Storing Media Files

Attribute file_format can be defined in the object definition to store the format of the file as an additional information. Hadoop can store files of any format. PDF, MS Word as well as MP3, AVI and so on. There is no special implementation required to store and retrieve media files. Therefore, HDMS can store documents such as PDF as well as media files such as MP3, AVI files, etc.

5.2.5 Storing Structured Data

To facilitate the storage of structured data, it should be treated as objects with no content. To store structured data, table with appropriate columns need to be defined. Once the object representing structured data is defined, no other special implementation is required to interact with the content-less object. Services can store, search and retrieve the structured data just like any other object. Although, services can create and store structured data, there are multiple tools which can ingest millions of rows of structured data into HBase table and manage it as objects or records.

5.2.6 Folder Structure

A Folder structure or a File plan is a folder hierarchy that needs to be defined to help users to locate content quickly. Folder objects are created in database and arranged in a hierarchy by setting folder_location attribute appropriately. Multiple file plans can be implemented. For example, one file plan can be based on the organization structure, so that all content of a specific department is stored in one folder and another file plan can be based on the creation date of the content, so that content created on a specific date is organized in a single place.

Defining object definition for a folder is a design activity, whereas, creating file plan is "pre-launch" activity. Folder objects are created by setting folder_location attribute appropriately to arrange folders in pre-defined hierarchy.

There is a concept of metadata-based Document Management System, where instead of creating folders and file plans, content is stored by tagging additional context values. Content is then presented by creating different views based on these context values.

Implementing file plan or no file plan, both scenarios can be designed and implemented in HDMS.

5.2.7 Library Services

Library services are functions or operations such are search, view metadata, upload or download files, check-out, check-in etc. To implement library services except check-in and check-out, no other special considerations are required. With proper modelling of services, these functions can be implemented

To implement the check-out operation, a set of additional attributes in object definition are required. For example, attribute such as is_checked_out, checked_out_by and checkout_timestamp should be defined. If a content is checked out, then is_checked_out should be set to true, checked_out_by to the user_name who has checkout and checkout_timestamp appropriately. Setting these attributes should be part of checkout method of the ContentObject service. Value of these attributes should be verified before allowing certain operation on the objects. For example, request to update attribute of a checked- out object, should be declined.

5.2.8 Versioning

When a user checks-in a previously checked-out content, a new version of the content is created and the new content becomes latest representation (released version) of the content. To implement versioning function, attributes such as previous_content_id to store content_id of previous version, version_number to store version number and is_latest to identify latest content should be added in object definition.

When a content is checked in, the service methods should update these attributes accordingly. While accessing or updating an object, the service should perform the operation only on the latest version of the object. Similarly, search queries should be appended by "is_latest=true" clause.

5.2.9 Records Management

Using HDMS implemented on Hadoop, petabytes of structured as well as unstructured data can be managed as records.

Specific content or structured data, having business significance or being a part of an evidence, are declared as records. Records are 'read only' objects. Records cannot be updated or modified. To capture record declaration, attributes, such as "is_record", "declared_by" and "record_declaration_timestamp" should be added in the object definition.

Retention is applied to a record, based on an event. For example, when a content or structured data is archived, retention is applied on the object and it will be retained in HDMS until the retention period is over. Records which are under retention cannot be deleted. To implement retention function, a set of attributes such as "retention_start_date", "retention_end_date" should be added in the object definition.

Disposition report is a report which shows the records whose retention is over. A user, member of records manager role, can run a disposition report and can list all the records with expired retention. The user can then either explicitly mark those records for deletion or can extend its retention. Records marked for deletion are then deleted by a batch job.

Records which are required for a legal case are held for a specific duration (as dictated by the legal authorities) in the system by putting a "legal hold" on the records. "Legal Hold" function can be modelled by defining attributes such as "marked_for_deletion", "legal_hold" etc. in the object definition and implement it by setting the value of these attributes appropriately.

All services interacting with objects, should be developed in such a way that, the services are aware of these attribute values. For example, if a content object is marked for legal hold, then it should not be deleted by a service. Due to huge scalability of Hadoop, organizations can build a system of records catering to different compliance needs, such as MiFID.

5.2.10 Implementing Archival

Data, structured or unstructured, is archived when it is no longer accessed. The factual intention of the archival is to offload the content or data to cheaper storage to optimise the resources and cost. To implement archival in HDMS, data nodes with cheaper hardware can be added and the content or data can be moved to these data nodes. Single HDMS repository can fulfil use case of recent storage and archival storage with added value of cost optimization. Another advantage of this in place archival, is that, the archived content will be still accessible. With Hadoop as backend, huge amount of data can be archived and managed as records, keeping TCO lower.

5.2.11 Administration

There will be multiple user interfaces to administer HDMS. For Administration of Hadoop cluster Apache, Ambari can be used. Apache Ambari is a tool for provisioning, managing, and monitoring Apache Hadoop clusters. Hue (Hadoop User Experience) is an open-source Web interface that supports Apache Hadoop and resources such as HBase and HDFS. Hue can also be used for performing HDMS administration related task such as creating tables in HBase, creating directory structure for file storage etc.

HDFS has its own set of commands which can be executed using command prompt. Alternately, HDFS exposes a web server which is capable of performing basic status monitoring and file browsing operations. Apache HBase also has its own commands which can be executed from command prompt. Using command prompt, activities such as table creation etc. required for HDMS implementation can be performed.

5.3 Service Layer

Service layer consists of object and functional services and encapsulate business logic to interact with objects stored in Hadoop

5.3.1 Defining and Developing Services

To interact with content and folder objects, web services needs to be modelled and developed. Services should be a single point of contact for interacting with the HDMS objects. Services should have methods corresponding to the operations performed on the objects. For example, to interact with an object of content_obj type, there should be a dedicated ContentObject service. Only the ContentObject service will interact with content_obj objects. The ContentObject service might have different methods, such as create, read and update, corresponding to the operations performed on the objects.

Additional services based on the user requirement, such as search and retrieve are also required. Abstraction of interaction logic with objects in services, will make the HDMS independent of backend Hadoop implementation.

Microservices can also be used to build the service layer. Microservices can be modelled based on HDMS functions or operations, for example, check-out operation. Check-out operation needs three steps: first, check if the content is already checked-out. If no, then the next step is to check whether the user has appropriate permissions on the object or not. If yes, then checkout operation can be performed on the object. To implement the check-out operation using microservices, three services are needed, a service to check whether the object is available for check-out, another service to check whether user has appropriate permissions. Third service will set attributes such as checked_out_by, checked_out_timestamp and is_checked_out appropriately to mark the object is checked out by the user.

Implementing microservices architecture can deliver advantages such as enhanced continuous integration and deployment etc. but it has few disadvantages such as higher cost and project complexity.

Universal Description, Discovery, and Integration (UDDI) is an XML-based registry for web services whereas for microservices, a database is used for service registry. In case of microservices, registration pattern can be self (service itself interact with service registry) or third party (a different process poll services and update service registry). In client-side discovery, client query service registry to access microservices whereas in service side discovery an API Gateway query service registry and forwards the client request accordingly.

Service layer needs to be highly configurable. Which HBase table to interact, what is the list of attributes etc. should be configured externally through use of property files or separate configuration services. The team developing services need to have complementary skills of Document Management System and Hadoop.

5.4 View Layer

View layer consists of a user interface which will enable users to interact with data layer through service layer.

5.4.1 Lean User Interface

All Document Management System products offer a User Interface (UI) to connect to their system. These user interfaces have many menus and submenus and sometimes, it causes the UI overwhelming for a user. A lean user interface, which has only required operations, has a better user adoption. Such user interface can be developed by using new generation browser-based technology, such as, Vue.js. An additional advantage of Vue.js is that it can easily consume REST services. Using Vue.js, a user interface can be developed with responsive design approach, so that it can be accessed on any device.

All key functions such as create, upload, update etc. should be available on the user interface as clickable menu items. Once a menu is clicked, the respective function can be performed by executing appropriate service and message can be displayed on the user interface to indicate the result of the execution. For example, to update attributes of an object, a user needs to key-in new values on the user interface and click on update. In service layer, ContentObject service's "updateObject" method can be called, which in turn, updates the attributes of the object.

Some of the administration function can also be incorporated in the user interface and these functions can be enabled if logged in user belongs to admin role. Functions such as creating additional folders, creating archival policies etc. can be part of the user interface.

5.5 Search

Search is an important feature of any Document Management System . There are two type of searches: attribute search or metadata search and full text search. In attribute search, value of an attribute is searched. In full text search, text of content (or document) is searched.

As HBase supports attribute search, attribute or metadata search can be implemented in HDMS. For implementing full text search an external search tool such as Solr is needed. With the help of the external search tool, both the search features, attribute as well as full text search, can be implemented. Use of external search engine will not only improve the search feature, but it will also make the search faster. Since, a search request goes to the external search engine, it helps in reducing the load on Hadoop.

6 Benefits

6.1 Generic Solution

HDMS is a generic solution which focus on new DMS requirements. HDMS can be easily tweaked to implement additional requirements and use cases. It can also be used as headless DMS.

6.2 High Volume

Hadoop can scale horizontally, on commodity hardware, to thousands of nodes and hence can store petabytes of data. With Hadoop as a backend, volume of content stored in HDMS, can grow to petabytes.

6.3 Media Files Management

Current Document Management Systems provide media file management either as a separate product or an add-on with a separate license. Since, any file irrespective of its format can be stored and managed in HDMS, the requirement of media file management in single repository, is fulfilled.

6.4 Fault Tolerant DMS

Hadoop has a powerful feature of data replication which guarantees the content and data availability. If a data node (or machine) fails or a data block gets corrupted, the data can be accessed from its copy. Therefore, in HDMS, risk of content or data loss is automatically mitigated. Replication feature also eliminates the need of time-consuming backup and restore procedures.

6.5 Content Anywhere, Any device

A user interface built using Vue.js can be based on responsive design approach so that user interface can be accessed from any device. Coupled with the organizational tools such as VPN, content can be accessed from anywhere and any device.

6.6 Cost

Licensing model of most of the Document Management System is "per user" based. Cost of the backend database is an additional one. For an organization where user base is large, licensing cost is significant and it increases the total cost of ownership (TCO). As compared to a traditional Document Management Systems, cost involved in implementing HDMS is much cheaper and it will be mainly, the subscription cost of Hadoop. Apache Hadoop is a free product. For other open source Hadoop implementation, such as HortonWorks, the subscription cost will be much lower.

6.7 In Place Archival

For HDMS, there is no need to have a separate archival solution. In the same Hadoop cluster, cheaper data nodes can be added, and old data can be moved to these cheaper data nodes. The same Hadoop cluster can be used for both structured as well as unstructured data archival.

6.8 Integrated HDMS

During development, services required for interacting with HDMS, can be aligned towards end-to-end integration. Using these services, business applications can retrieve the most recent information and use the same in its processes.

6.9 Content Analytics

Hadoop can be integrated with content analytics tools to find out actionable insights. For example, data gathered using queries to find out who is generating which type of content, which is the most searched keyword, which is keyword is returning zero results, system usage, is it increasing or decreasing etc. helps understand user expectations and HDMS usage. Such information can be used to enhance user experience and HDMS adoptions. Depending on the data stored in Hadoop, content analytics can be extended for more complex use cases.

6.10 No Vendor Locking

Object and functional services are responsible for connecting to Hadoop and performing operations on objects. Since every operation on object is being channelized through services, dependency on Hadoop implementation is minimized. If today Apache Hadoop is used, tomorrow some other implementation of Hadoop can be used without a significant impact on overall ecosystem.

Conclusion

To implement HDMS, the key design considerations are object definition, service implementation, security implementation and methodology for storing and managing binary files in Hadoop. Implementing functions such as folder structure, library services, versioning, records management and archival are independent of Hadoop and can be implemented by extending the logic of managing objects, in services. Usage of external search engine will make search function faster and will help in distributing the load. A lean user interface will help users in completing the daily work faster and hence will have better adoption. Vue.js based user interface can be developed using responsive design approach, which can be accessed from mobile devices.

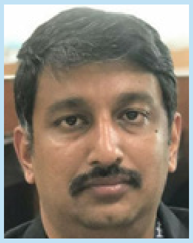
By architecture HDMS offers advantages such as data resilience, very minimal or no backup and restore procedures, no need for deployment in high availability mode. HDMS is a better fitment when DMS need to be deployed in headless architecture, huge amount of data needs to be stored and documents as well as media files need to be stored in single repository. Additionally, advanced capabilities of content archival, tagging, analytics can be easily integrated over this flexible architecture.

In nutshell, HDMS is a generic solution which can provide multifaceted benefits and can be used for various use cases. HDMS not only fulfils traditional requirements but also conveniently addresses new document management system requirements.

References

1. Focus of this paper is to conceptually examine the feasibility of implementing Document Management System on Hadoop and not implementing Content Services or Enterprise Content Management (ECM)
2. http://www.aiim.org/What-is-ECM-Enterprise-Content-Management
3. Hadoop can have Apache Accumulo, Cassandra etc., as its database. These databases are not considered here.
4. https://www.waterfordtechnologies.com/big-data-interesting-facts/
5. http://876solutions.com/sites/default/files/white-papers/Alfresco_White_Paper_TCO_for_ECM.pdf
6. Component services such as WebHDFS and Stargate are not considered as part of HDMS service layer.
7. http://hbase.apache.org/0.94/book/rowkey.design.html talks about dos and don’ts of defining unique identification number in HBase
8. http://blog.cloudera.com/blog/2012/09/understanding-user-authentication-and-authorization-in-apache-hbase/
9. https://issues.apache.org/jira/browse/HBASE-11339

About the Author



Girish Pande
Senior Technology Architect, Infosys Digital.

A senior technology architect with digital practice of financial unit at Infosys, Girish Pande, has around 19 years of experience in Information Technology. He has played key role in architecting and implementing end to end Content Services, Enterprize Search, NLP and Automation solutions for various client across the globe. He is an M. Tech. in Industrial Engineering & Operations Research from IIT Bombay.

He can be reached at girish_pande@Infosys.com.