Hopsworks: Improving User Experience and Development on Hadoop with Scalable, Strongly Consistent Metadata

Mahmoud Ismail*, Ermias Gebremeskel†, Theofilos Kakantousis†, Gautier Berthou† and Jim Dowling*†
* KTH - Royal Institute of Technology, † RISE SICS
{maism, jdowling}@kth.se, {ermias.gebremeskel, tkak, gautier, jdowling}@sics.se

Abstract—**Hadoop is a popular system for storing, managing, and processing large volumes of data, but it has bare-bones internal support for metadata, as metadata is a bottleneck and less means more scalability. The result is a scalable platform with rudimentary access control that is neither user- nor developer-friendly. Also, metadata services that are built on Hadoop, such as SQL-on-Hadoop, access control, data provenance, and data governance are necessarily implemented as eventually consistent services, resulting in increased development effort and more brittle software.**

In this paper, we present a new project-based multi-tenancy model for Hadoop, built on a new distribution of Hadoop that provides a distributed database backend for the Hadoop Distributed Filesystem's (HDFS) metadata layer. We extend Hadoop's metadata model to introduce projects, datasets, and project-users as new core concepts that enable a user-friendly, UI-driven Hadoop experience. As our metadata service is backed by a transactional database, developers can easily extend metadata by adding new tables and ensure the strong consistency of extended metadata using both transactions and foreign keys.

I. Introduction

Apache Hadoop [1] is the most popular open-source platform for storing and processing large volumes of data. Hadoop's core component is the Hadoop Distributed File System (HDFS) [2] which is a distributed hierarchical file system that scales up to thousands of machines, storing 100s of PBs of data [3]. Hadoop also contains a resource management service, YARN, that manages CPU and memory resources on behalf of applications such as data parallel processing frameworks (MapReduce [4], Flink [5], Spark [6]), key-value stores (HBase [7]), and SQL-on-Hadoop services [8]. Both HDFS and YARN store their metadata in-memory on a central server (the Namenode and Resourcemanager, respectively). As they are implemented in Java, the practical upper-bound on the size of metadata is $\tilde{2}00$ GB for HDFS [9], making it a bottleneck limiting the size of Hadoop clusters. Hadoop designers minimized the amount of metadata stored to optimize for scalability, at the cost of rich and customizable metadata.

Data management was not a consideration in the original design of Hadoop. HDFS long lacked the ability to easily extend files with custom metadata to support features such as fine-grained access control, free-text search of the namespace, and data provenance [10], [11]. The lack of support for custom metadata in HDFS made it difficult for systems running on top to support efficient data management techniques [12]. Hadoop developers have the challenge of implementing metadata services in external systems and ensuring the consistency of the external metadata with internal metadata such as file metadata in HDFS, users in Kerberos, and applications in YARN. Two-way synchronization of the external metadata state with Hadoop's internal metadata state is typically not done, as it would overload services such as the NameNode. For example, Hive does not actively check if the HDFS files backing a SQL table still exist and are correct [8]. The lack of two-way synchronization and Hadoop's limited metadata has meant that the platform has a static security model, lacking attribute-based access control and dynamic roles. In Apache Hadoop, if a dataset is shared with user Alice, the person sharing the data has no control over what Alice does with the data - from downloading it, to cross-linking it with external data-sources.

Recently, an extension of HDFS, called HopsFS, has moved the metadata to a scale-out, distributed database (MySQL Cluster), supporting up to at least 24 TB of metadata and providing at least 16X throughput [9]. HopsFS is a drop-in replacement for HDFS where metadata can be easily extended by adding tables to the database. Extended metadata can be made strongly consistent with internal metadata using transactions and foreign keys (for example, to remove a hive table if the backing HDFS file(s) are removed).

Hopsworks	Hadoop
Projects	Clusters
Dynamic Roles	Static ACLs
Datasets	Files
Shared Datasets	n/a
Kafka Topics	n/a
Quotas (HDFS and Yarn)	Quotas (HDFS only)
Jobs	Applications and Jobs
Integrated Notebooks	External Notebooks
SSL/TLS	Kerberos

TABLE I: A comparison of the core concepts in Hopsworks and Hadoop.

In this paper, we present Hopsworks, a new project-based multi-tenant platform for secure collaborative data analysis, running on top of HopsFS and HopsYARN. In Hopsworks, we leverage the extensibility of HopsFS to introduce three new concepts to Hadoop: *Projects*, *Datasets*, and *Project-Users* (dynamic roles). A Project is a collection of Datasets and Users. Each Dataset has a home Project but can also be securely shared with other Projects. As such, Hopsworks

1063-6927/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCS.2017.41

enables the secure sharing of sensitive datasets on a single Hadoop platform. Data sharing does not require copying data in Hopsworks. In Hopsworks, projects have their own isolated subtree in HDFS (a sandbox) with project-specific users, implementing dynamic roles, as well as per-project Kafka topics. Datasets and topics can be securely shared between projects. All of these features are enabled by extending metadata for HDFS using both transactions and foreign keys (to ensure strong consistency for the extended metadata). Another feature unique to Hopsworks is CPU quotas for projects on YARN - we listen for container events (start/stop of applications) and transactionally update quota metadata appropriately. The new abstractions introduced by Hopsworks, and their comparison with corresponding concepts in Hadoop, are shown in table I. Hopsworks provides, compared to Hadoop, a reduced number of concepts that users need to learn for data management.

Hopsworks is an open source project [13] that supports automated deployment (using Chef and the Karamel orchestration tool [14]) on Amazon AWS, Google GCE, OpenStack, and on-premise bare metal clusters. Hopsworks has been running in production since April 2016, providing Hadoop-as-a-Service for researchers at a data center in Luleå, Sweden, and has been presented at many industry conferences in 2016, including Hadoop Strata London, Flink Forward, and Spark Summit Europe.

II. System Overview

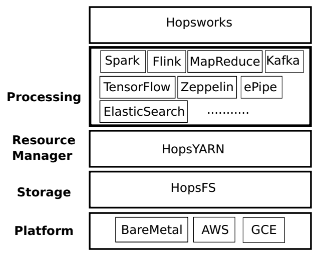
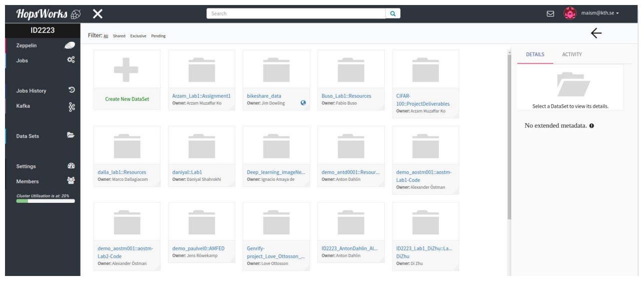


Fig. 1: The system architecture for Hopsworks. A Hopsworks cluster can be provisioned on different types of platforms; BareMetal, Amazon AWS, and Google GCE. HopsFS is used in the storage layer, while HopsYARN is used for scheduling resources. On top of HopsYARN and HopsFS, we support different services such as Spark, Flink, MapReduce, Kafka, and so on. Hopsworks, then provides an intuitive user interface for the services and integrates them into the Project-Dataset model

Hopsworks is a multi-tenant data management and processing Java EE web application running on top of Hops Hadoop with integrated support for data parallel processing frameworks such as Apache Spark, Apache Flink, and Tensorflow, as well as Apache Kafka (a scalable message bus) and interactive notebooks with Apache Zeppelin. We chose Java as it has a lower impedance mismatch with the primary Java services used in the Hadoop ecosystem. Hopsworks also provides a graphical user interface (UI), written in AngularJS, for authentication, running/debugging data parallel applications, and managing projects, datasets, and users. Hopsworks provides perimeter security, being the main point of access to the Hops Hadoop platform (Kafka may also be exposed to clients). Command line access to Hadoop services is



Arzam_Lab1::Assignment1
Owner: Arzam Muzaffar Ko
bikeshare_data
Owner: Jim Dowling
Buso_Lab1::Resources
Owner: Fabio Buso
CIFAR-100::ProjectDeliverables
Owner: Arzam Muzaffar Ko
dalla_lab1::Resources
Owner: Marco Dallagiacom
daniyal::Lab1
Owner: Daniyal Shahrokhi
Deep_learning_imageNe...
Owner: Ignacio Amaya de
demo_antd0001::Resour...
Owner: Anton Dahlin
demo_aostm001::aostm-Lab1-Code
Owner: Alexander Östman
demo_aostm001::aostm-Lab2-Code
Owner: Alexander Östman
demo_paulvel0::AMFED
Owner: Jens Röwekamp
Genrify-project_Love_Ottosson_...
Owner: Love Ottosson
ID2223_AntonDahlin_Al...
Owner: Anton Dahlin
ID2223_Lab1_DiZhu::La...DiZhu
Owner: Di Zhu
Select a DataSet to view its details.
No extended metadata.

Fig. 2: The Hopworks' UI for a project called 'ID2223', the datasets of the project appeared in the middle, and on the left side of the page different services appeared such as Zepplin, Jobs, and Kafka

turned off by default, and all jobs are run through a Jobs UI (Spark/Flink/MapReduce), or interactively using Zeppelin notebooks. Hopsworks provides a REST API to allow external applications to integrate with the platform.

A. Concepts

In this section, we discuss in more detail the three novel concepts in Hopsworks which are *Dataset*, *Project*, and *Project-user*.

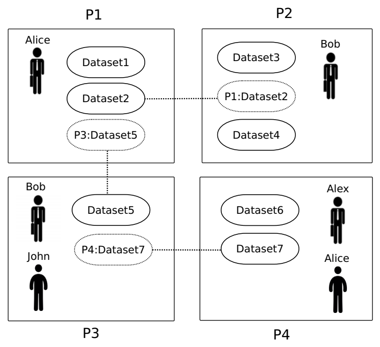


Fig. 3: Hopsworks cluster with 4 projects and 4 users. Alice is a *Data Owner* in P1 and a *Data Scientist* in P4, Bob is a *Data Owner* in P2 and P3, Alex is a *Data Owner* in P4, and John is a *Data Scientist* in P3. P1 shares Dataset2 with P2, P3 shares Dataset5 with P1, and P4 shares Dataset7 with P3.

1) Dataset: A dataset is a directory subtree in the HDFS (HopsFS) namespace that can be shared between projects. A dataset has a single owner with read/write privileges and it is readable by all members of the project (and members of any other project with which it is shared) through group permissions. To enable datasets to be shared without breaking the isolation model for projects, we create a HDFS (HopsFS) group per dataset. A Dataset has a *searchable* (metadata) property, which, when enabled, means that the dataset as well as its files and any extended metadata are indexed by Elasticsearch (also part of Hopsworks). From the Hopsworks UI, users can free-text search for datasets using the name, the description, or other extended metadata attribute values. Users with privileges to access a Dataset (through membership of a suitable project) can also use free-text search to find files/directories within the Dataset's subtree, again using Elasticsearch. A dataset can also be made public to users outside the Hopsworks cluster. Public datasets can be discovered and downloaded from any Hopsworks cluster on the public Internet. Downloads are performed using peer-to-peer middleware.

2) User: Hopsworks supports two levels of users: *platform-users* and *project-users*. A platform-user in Hopsworks is identified by her email address. Users login using their email address (authenticating the *platform user*) using either a JDBC Realm, LDAP, or two-factor authentication. A platform-user can be either a *normal user* or an *administrator*. Administrators have additional views for managing user registrations, project quotas, and monitoring/restarting services and applications. Administrators do not have privileges to view the contents of Projects or Datasets from within Hopsworks - privileged command-line access to the machines and services is needed for that. A platform-user can be a member of zero to many projects. For each project, Hopsworks creates an additional *project-user* that is stored in the database. The project-user is linked to the project and platform-user using foreign keys (when the project or platform-user is deleted, the corresponding project-users are automatically cleaned up). The project-user is used to run programs and access data in Hops Hadoop (not the platform-user). When a user 'enters' a project using the UI, the project-user for that project is activated. Each project-user is effectively a dynamic role for the platform-user. From the Hops Hadoop perspective, when a user launches a job from within a project, the job is owned by the project-user. Hops Hadoop does not know about platform-users, only project-users. As such, without explicit authorization a project-user cannot access data from other projects, as Hops Hadoop sees two different users - one for each project. Platform-users cannot use services in Hopsworks, they are only used for authentication.

3) Project: A project is a collection of datasets and users, as shown in figure 3, as well as Kafka topics, Jobs (Flink, Spark, MapReduce), and Zeppelin notebooks. Project-users can have one of two roles within a project: *Data Scientist* or *Data Owner*. A *Data Owner* can upload/download data into/from the project, add/remove members to/from the project, change roles of other members in the project, create, share, and delete datasets, import/export datasets, and create/update extended metadata for files/directories and datasets. On the other hand, a *Data Scientist* can only run batch jobs (Spark, Flink, and MapReduce) or interactive jobs on datasets using Zeppelin.

Each project has its own Resources, Logs, and Notebooks datasets, stored in HopsFS. The Resources dataset is for Data Scientists to upload jobs (a limited number of file types) to the project. The Logs dataset is used by applications to write out aggregated YARN logs. Data Scientists have sticky bit read/write privileges on both the Logs and Notebooks datasets. This ensures that all project-users can read/write logs and notebooks, but project-users cannot overwrite each other's logs or notebooks.

B. Under the hood

Let's consider some examples to understand how Hopsworks' concepts are translated into HopsFS actions on the filesystem tree, see figure 4.

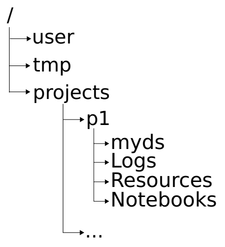


Fig. 4: The Filesystem tree structure for Projects and Datasets in HopsFS

Alice wants to create a project "p1", following are the steps that are happening under the hood in HopsFS:

1) a new group "p1" is created in the database;
2) a new user "p1__alice" is created in the database;
3) the project base directory is created in HopsFS at the path $/projects/p1$, where the owner of the directory is set to "p1__alice" and the group is set to "p1";
4) the permissions on the project base directory ($/projects/p1$) are set so that "p1__alice" and "p1" group members have read/write/execute permissions.

Now, Alice wants to add Bob as a new member in "p1". To add the new member to the project, the following steps are done in HopsFS:

1) a new user "p1__bob" is created in the database;
2) if the user was assigned the role *Data Owner*, then the user "bob__p1" is added to the group "p1".

After creating the project "p1", Alice now wants to create a dataset "myds" in "p1":

1) a new group "p1__myds" is created in the database.
2) all members of the project "p1" are added to the dataset group "p1__myds" in the database.
3) the base directory for the dataset is created at $/projects/p1/myds$. The owner is set to "p1__alice" and the group is set to "p1__myds".
4) the permissions on $/projects/p1/myds$ are set so that the owner "p1__alice" has all permissions, other project members have read and execute permissions through the group, while other users have no permissions.

When a new project is created, three default datasets are created: *Logs*, *Resources*, and *Notebooks*. These datasets have different permissions requirements to enable all users of the project to read/write to these datasets while disallowing them to delete these datasets. That is accomplished by giving read/write/execute permissions to members of the project group and add a sticky bit on the dataset folder to disallow deletion.

```
hdfs dfs -chmod 1770 /project/p1/Logs
```

Sharing: A Dataset "myds" can be shared from its home project "p1" with another project "p2", then all members of "p2" are added to the dataset group "p1__myds". This will give the members of "p2" the same privileges on "myds" as all members of the original project. Remember that *Logs*, *Resources*, and *Notebooks* datasets are not shareable. Note that sharing a Dataset does not give any permissions whatsoever to any user to write to the parent directory of the Datasets (its home project's base directory), hence the isolation of project is preserved.

III. Services

Hopsworks supports a number of built-in services (dataset browser, searching, extended metadata designer, and a job launcher) as well as external *microservices* provided by other frameworks (Zeppelin, Dr. Elephant, Kafka). Hopsworks manages access control for all services. Hopsworks has no local state and its default application server, Glassfish, supports clustering

A. Extended Metadata

HopsFS provides REST APIs to attach extended metadata to Datasets, files, or directories through Hopsworks. We support two approaches to attach metadata to a Dataset/file/directory; either *Schemaless* or *SchemaBased*. In the *Schemaless* approach, the user can attach any JSON file to her Dataset/file/directory. On the other hand, in the *SchemaBased* approach, the user must firstly define a schema for her metadata, and then, she can attach this schema to her Dataset/file/directory to be updated later. In order to ease the schema designing, we provide an intuitive MetadataDesigner tool in Hopsworks to help users create schemas, import/export to JSON, and extend existing schemas. The extended metadata is then exported to Elasticsearch, from where it can be queried and the associated Dataset/file/directory can be easily discovered.

B. Search

Hopsworks has a search bar on the top of the page as shown in figure 2. The search bar has a different scope according to which view it is accessed from. For example, in the landing page, the search bar will search for Datasets both within the cluster and globally. Within the scope of a project (on the project page), searching will only search through the datasets belonging to the active project. On a dataset page, it will only search through the files/directories inside that specific dataset.

C. Job Launcher

Hopsworks allows users to create and run jobs through the user interface. Currently, it supports MapReduce, Spark, and Flink jobs. The log files (stderr, stdout) are aggregated from YARN and are stored in the *Logs* dataset, from where they can be viewed. After creating a job, a user could edit, delete, or schedule the job for running later. Hopsworks integrates Dr. Elephant in a Job History microservice, to give the user a way to monitor and tune their jobs.

D. Kafka

Apache Kafka is a distributed publish-subscribe messaging system that is designed to be fast, scalable, and durable. Kafka provides an API to allow for authentication and encrypted communication between clients and brokers as well as secure inter-broker communication. Hopsworks eases the use of Kafka by providing a library called HopsUtil that encapsulates the security and configuration aspects of Kafka. Also, Hopsworks provides a user interface to create and manage Kafka topics within a project.

E. Zeppelin

Hopsworks integrates Apache Zeppelin, a web application to run notebooks for interactive data analysis. Hopsworks supports different interpreters for Zeppelin (Spark, Flink, Tensorflow/Python), which can be started and stopped directly from the Hopsworks user interface.

IV. Demo

In the demo, we will first show automated installation for Hopsworks using the Karamel UI. Then, users will register through the Hopsworks UI. After logging in, users will be able to explore different aspects of Hopsworks such as creating projects and datasets, sharing datasets between projects, searching for datasets or files or directories, and running jobs. Also, we will go through some programming examples using Zeppelin/Tensorflow, Zeppelin/Spark, Kafka, and Flink.

regular IEEE prefers the singular form

Acknowledgment

This work is funded by Swedish Foundation for Strategic Research project E2E-Clouds, and by EU Horizon 2020 project AEGIS "Advanced Big Data Value Chain for Public Safety and Personal Security" under Grant Agreement no. 732189.

References

[1] "Apache Hadoop," http://hadoop.apache.org/, [Online; accessed 21-February-2017].

[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies, 2010*, May 2010, pp. 1–10.

[3] "Inside Yahoos Super-Sized Deep Learning Cluster," https://www.datanami.com/2015/10/12/inside-yahoos-super-sized-deep-learning-cluster/, [Online; accessed 21-February-2017].

[4] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[5] P. Carbone, S. Ewen, S. Haridi, A. Katsifodimos, V. Markl, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Data Engineering*, p. 28, 2015.

[6] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10.

[7] "Apache HBase," http://hbase.apache.org/, [Online; accessed 21-February-2017].

[8] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at facebook," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 1013–1020.

[9] S. Niazi, M. Ismail, S. Haridi, J. Dowling, S. Grohsschmiedt, and M. Ronström, "HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases," in *15th USENIX Conference on File and Storage Technologies (FAST 17)*. Santa Clara, CA: USENIX Association, 2017, pp. 89–104.

[10] M. Seltzer and N. Murphy, "Hierarchical file systems are dead," in *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, ser. HotOS'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 1–1.

[11] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang, "Goods: Organizing google's datasets," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: ACM, 2016, pp. 795–806.

[12] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. J. DeWitt, and G. Heber, "Scientific data management in the coming decade," *ACM SIGMOD Record*, vol. 34, no. 4, pp. 34–41, 2005.

[13] "Hopsworks," https://github.com/hopshadoop/hopsworks, [Online; accessed 21-February-2017].

[14] M. Bux, J. Brandt, C. Lipka, K. Hakimzadeh, J. Dowling, and U. Leser, "Saasfee: Scalable scientific workflow execution engine," in *VLDB Demonstrations Track, forthcoming*, Hawaii, USA, September 2015.