IJBD Editorial Board

Editors-in-Chief
Ernesto Damiani, Università degli Studi di Milano, Italy
Paul Hoffman, SaffronTechnology Inc., USA

Associate Editors
Shiyong Lu, Wayne State University, USA
Min Luo, Huawei, China

Editorial Board
Rafael Accorsi, University of Freiburg, Germany
Claudio Ardagna, Università degli Studi di Milano, Italy
Budak Arpinar, University of Georgia, USA
Valerio Bellandi, Università degli Studi di Milano, Italy
Rajdeep Bhowmik, Cisco Systems Inc., USA
Chiara Braghin, Università degli Studi di Milano, Italy
Suren Byna, Lawrence Berkeley National Laboratory, USA
Chi-Hung Chi, CSIRO, Australia
Byron Choi, Hong Kong Baptist University, Hong Kong
Stelvio Cimato, Università degli Studi di Milano, Italy
Piero Fraternali, Politecnico di Milano, Italy
Hesham Hallal, University of Tabuk, Saudi Arabia
Peng Han, Chongqing Research Center for Information and Automation Technology (CIAT), China
Omar Hassan, Insa Lyon, France
Yongqiang He, Facebook, USA
Srividya Kona, Arizona State University, USA
Shailesh Kumar, Google India, India
Marcello Leida, EBTIC, UAE
Zhanhuai Li, Northwestern Polytechnical University, China
Du Li, Ericsson, USA
Althea Liang, HP Lab, Singapore
Carolyn McGregor, University of Ontario Institute of Technology, Canada
Hamid Motahari, University of New South Wales, Australia
Maryam Panahizar, Kno.e.sis, USA
Weining Qian, East China Normal University, China
Paul Rosen, University of Utah, USA
Gabriele Ruffatti, Engineering Ingegneria Informatica S.p.A., Italy
Ming-Chien Shan, SAP Research, USA
Zhe Shan, Manhattan College, USA
Badari Narayana Thyamagondu Nagarajasharman, University of New South Wales, Australia
Andy Twigg, Oxford University, UK
Raymond Wong, University of New South Wales, Australia
Jian Yin, Sun Yat-Sen University, China
Gong Zhang, Oracle, USA
Huan Chen, Kingdee Research, China
Zhiyong Chen, Mercy College, USA
Jia Zhang, Carnegie Mellon University - Silicon Valley, USA
Alfredo Cuzzocrea, University of Calabria, Italy
International Journal of Big Data

October 2015, Vol. 2, No.3

Table of Contents

E. Editor-in-Chief Preface
Ernesto Damiani, Università degli Studi di Milano, Italy
Paul Hoffman, SaffronTechnology Inc., USA

vi. Call for Articles: IJBD special issue of application oriented innovations

Research Articles

1   An approach for leveraging personal cloud storage services for team collaboration
Zehui Cheng, ZhangBing Zhou, China University of Geosciences, China
Ke Ning, Enterprise key Laboratory of application software, China
Liang-Jie Zhang, National Engineering Research Center for Supporting Software of Enterprise Internet Services, China
Taj Rahman, University of Science and Technology Beijing, China
JiangSong Min, Kingdee International Software Group, China

15  A comprehensive overview of open source big data platforms and frameworks
Pedro Almeida, Jorge Bernardino, ISEC – Polytechnic of Coimbra, Portugal,
CISUC – Center for Informatics and Systems of the University of Coimbra, Portugal

34  Distributed SPARQL Querying Over Big RDF Data Using Presto-RDF
Mulugeta Mammo, Mahmudul Hassan, Srividya K. Bansal, Arizona State University

50  The Development and Deployment of Large-File Upload Services
Huan Chen, Liang-Jie Zhang, Bo Hu, Si-Zhe Long, Li-Hui Luo, Kingdee Research, National Engineering Research Center for Supporting Software of Enterprise Internet Services, China
Welcome to the International Journal of Big Data (IJBD). Big Data is a broad term for data sets so large, complex or incomplete that current data technologies could not process or handle. From the technology foundation perspective, Big Data covers the science and technology needed for bridging the gap between data services and business, R&D. All topics regarding data study and management align with the theme of IJBD. Specially, we focus on:

**Big Data Models and Algorithms** (Foundational Models for Big Data, Algorithms and Programming Techniques for Big Data Processing, Big Data Analytics and Metrics, Representation Formats for Multimedia Big Data)

**Big Data Architectures** (Cloud Computing Techniques for Big Data, Big Data as a Service, Big Data Open Platforms, Big Data in Mobile and Pervasive Computing)

**Big Data Management** (Big Data Persistence and Preservation, Big Data Quality and Provenance Control, Management Issues of Social Network enabled Big Data)

**Big Data Protection, Integrity and Privacy** (Models and Languages for Big Data Protection, Privacy Preserving Big Data Analytics Big Data Encryption)

**Security Applications of Big Data** (Anomaly Detection in Very Large Scale Systems, Collaborative Threat Detection using Big Data Analytics)

**Big Data Search and Mining** (Algorithms and Systems for Big Data Search, Distributed, and Peer-to-peer Search, Machine learning based on Big Data, Visualization Analytics for Big Data)


The International Journal of Big Data (IJBD) is designed to be an important platform for dissemination high quality research on above topics in a timely manner and provide an ongoing platform for continuous discussion on research published in this journal. To ensure quality, IJBD only considers expanded version of papers presented at high quality
conferences and key survey articles, such as BigData Congress, ICWS, SCC, MS, CLOUD, SERVICES, etc.

This issue collects four papers. The first article is titled “An approach for leveraging personal cloud storage services for team collaboration”. The authors proposed a method to leverage third-part personal cloud storage services to provide shared storage for team collaboration applications. The approach has been tested in the product kAct.

The second article is titled “A comprehensive overview of open source big data platforms and frameworks”. The authors provided an overview of using Big Data with Open Source tools.

The third article is titled “Distributed SPARQL Querying Over Big RDF Data Using Presto-RDF”. The authors proposed an architecture based on Presto, Presto-RDF, that can be used to process big RDF data. The experimental results showed promising performance compared with Hive.

The fourth article is titled “The Development and Deployment of Large-File Upload Services”. The authors proposed large-file upload services and its development details. The authors also gave some real-world scenarios based on the deployed services.

We would like to thank the authors for their efforts in delivering these four quality articles. We would also like to thank the reviewers, as well as the Program Committee of IEEE BigData Congress for their help with the review process.

About the Publication Lead

Liang-Jie (LJ) Zhang is Senior Vice President, Chief Scientist, & Director of Research at Kingdee International Software Group Company Limited, and director of The Open Group. Prior to joining Kingdee, he was a Research Staff Member and Program Manager of Application Architectures and Realization at IBM Thomas J. Watson Research Center as well as the Chief Architect of Industrial Standards at IBM Software Group. Dr. Zhang has published more than 140 technical papers in journals, book chapters, and conference proceedings. He has 40 granted patents and more than 20 pending patent applications. Dr. Zhang received his Ph.D. on Pattern Recognition and Intelligent Control from Tsinghua University in 1996. He chaired the IEEE Computer Society’s Technical Committee on Services Computing from 2003 to 2011. He also chaired the Services Computing Professional Interest Community at IBM Research from 2004 to 2006. Dr. Zhang has served as the Editor-in-Chief of the International Journal of Web Services Research since 2003 and was the founding Editor-in-Chief of IEEE Transactions on Services Computing. He was elected as an IEEE Fellow in 2011, and in the same year won the Technical Achievement Award "for pioneering contributions to Application Design Techniques in Services Computing" from IEEE Computer Society. Dr.

About the Editor-in-Chief

**Ernesto Damiani** is a full professor at the Università degli Studi di Milano and the Head of the PhD program in computer science. Ernesto’s areas of interest include cloud-based service and process analysis, processing of semi and unstructured information, knowledge representation and sharing. Ernesto has published several books and about 300 papers and international patents. He leads/has led a number of international research projects: he is the Principal Investigator of the ASSERT4SOA project (STREP) on the certification of SOA; leads the activity of SESAR research unit within SecureSCM (STREP), ARISTOTELE (IP), PRACTICE ((IP) ASSERT4SOA (STREP), and CUMULUS (STREP) projects funded by the EC in the 7th Framework Program. Ernesto has been an Associate Editor of the IEEE Trans. on Services Computing since its inception. Also, Ernesto is Editor in Chief of the International Journal of Knowledge and Learning (Inderscience) and of the International Journal of Web Technology and Engineering (IJWTE). He has served and is serving in all capacities on many congress, conference, and workshop committees. He is a senior member of the IEEE and ACM Distinguished Scientist.

**Paul Hofmann** is Chief Technology Officer of Saffron Technology, Inc. Paul is responsible for Saffron’s technology direction and product management. Before joining Saffron in 2012 Paul was Vice President Research at SAP Labs at Palo Alto. Paul has also worked for the SAP Corporate Venturing Group. Paul joined SAP in 2001 as Director for Business Development EMEA SAP AG where he has created the Value Based Selling program. Paul was visiting scientist at Civil and Environmental Engineering Department at MIT, Cambridge, MA 2009. Prior to joining SAP, Paul was Senior Plant Manager at BASF’s Global Catalysts Business Unit in Ludwigshafen, Germany. Paul has been entrenched in research as Senior Scientist and Assistant Professor at outstanding European and American Universities (Northwestern University, U.S., Technical University Munich and Darmstadt, Germany). He is an expert in computational chemistry and computer graphics (Ph.D., research and teaching in Nonlinear Quantum Dynamics and Chaos Theory), authoring numerous publications and books, including a book on SCM and environmental information systems as well as performance management and productivity of supply chains.
Call for Articles
IJBD special issue of application oriented innovations

Big Data is a dynamic discipline. It has become a valuable resource and mechanism for the practitioners and researchers to explore the value of data sets in all kinds of business scenarios and scientific work. From industry perspective, IBM, SAP, Oracle, Google, Microsoft, Yahoo, and other leading software and internet service companies have also launched their own innovation initiatives around big data.

The International Journal of Big Data (IJBD) includes topics related to the advancements in the state of the art standards and practices of Big Data, as well as emerging research topics which are going to define the future of Big Data, including strategy planning, business architecture, application architecture, data architecture, technology architecture, design, development, deployment, operational practices, analytics, optimization, security and privacy.

IJBD now launches a special issue which focuses on application oriented innovations. The papers should generally have results from real world development, deployment, and experiences delivering Big Data solutions. It should also provide information like "Lessons learned" or general advices gained from the experience of Big Data. Other appropriate sections are general background on the solutions, overview of the solutions, and directions for future innovation and improvements of Big Data.

Authors should submit papers (12 pages minimum, 24 papers maximum per paper) related to the following practical topics:
1. Architecture practice of Big Data
2. Big Data management practice
3. Emerging algorithm from real-world scenario
4. Security application of Big Data
5. Big Data search and mining practice
6. Enterprise-level Big Data tooling
7. Innovative Idea from TED talk

Please note this special issue mainly considers papers from real-world practices. In addition, IJBD only considers extended versions of papers published in reputable related conferences. Sponsored by Services Society, the published IJBD papers will be made accessible for easy of citation and knowledge sharing, in addition to paper copies. All published IJBD papers will be promoted and recommended to potential authors of the future versions of related reputable conferences such as IEEE BigData Congress, ICWS, SCC, CLOUD, SERVICES, and MS.

If you have any questions or queries on IJBD, please send email to IJBD AT ServicesSociety.org.
An approach for leveraging personal cloud storage services for team collaboration

Zehui Cheng¹, ZhangBing Zhou¹, Ke Ning², LiangJie Zhang³, Taj Rahman⁴, JiangSong Min⁵
1. China University of Geosciences (Beijing), China
2. Enterprise key Laboratory of application software, Shenzhen, China
3. National Engineering Research Center for Supporting Software of Enterprise Internet Services, China
4. University of Science and Technology Beijing, China
4. Kingdee International Software Group Co., Ltd., China
ke_ning@kingdee.com

Abstract

With the rapid development of cloud computing technology, cloud-based team collaboration applications are becoming popular on the Web to enlarge storage space and facilitate the team collaboration. Among all the required features for a typical team collaboration application, shared storage for referred documents or produced artifacts by the team is a must-have one. However, existing shared storage solutions for team collaboration applications are far from satisfaction. Some of them rely on self-built storage infrastructure. Consequently, when the application becomes more powerful and more storage space is required, which could be a big burden, especially for small or medium vendors. With the prevalence of personal cloud storage services, such as Dropbox and Google Drive, more team collaboration applications allow users to share files from their personal cloud-storage spaces through external shared links, which can partly solve the problem. However, this method is not convenient for team collaboration, neither safe enough. This paper presents an approach to leverage third-part personal cloud-storage services to provide shared storage for team collaboration applications. Compared to existing approaches, our approach provides sophisticated mechanisms to make sure it's more convenient and safer for the team work. It brings benefits in three folds: for users, it improves the utilization of personal cloud storage space; for vendors of personal cloud storage service, it helps attract users to use their services; for vendors of team collaboration applications, it reduces the burden of developing self-built storage infrastructure. The approach has been tested in kAct, a task-based team collaboration application provided by Kingdee, and the results are promising.

Keywords: Personal Cloud Storage Services; Shared Storage; Team Collaboration.

1. INTRODUCTION

With the rapid development of cloud computing, cloud-based team collaboration applications are becoming popular on the Web. It is argued that cloud-based team collaboration is inevitable trend to meet the increasing demands when handling team collaboration work. When much more people should be included from outside organization, or if a team is spread out to be distributed in the world, cloud computing is the only way to connect teammates together to collaborate. In addition, as the collaborated application is becoming more powerful and a large amount of storage space is in need as well, sharing cloud resource is a more reasonable method (Hadi, 2015, p. 78). For example, 37 Signal started to provide Basecamp (a web-based project management tool) in 2004, which is becoming very popular later on. Asana, founded in 2008 by Facebook co-founder, is a web and mobile application designed to enable teamwork without email. It is used by tens of thousands of teams across all industries and in almost every continent (Dishman, n.d.). Companies that use Asana include Airbnb, Dropbox, Disqus, Foursquare, Pinterest, Stripe, and Uber (Asan, n.d.). Among all the required features for a team collaboration application, shared storage is a must-have one. Documents referred or artifacts produced by the team must be stored in a shared place for all the teammates to access. This place should be kept safe for the team, and support well defined access control mechanisms.
However, existing shared storage solutions for team collaboration applications are far from satisfaction. Some of the solutions rely on self-built storage infrastructure. In this setting, when the application is getting more enormous, buying more servers is inevitable to store huge amount of data, which is becoming a big burden, especially for those small or medium vendors (Hadi, 2015, p. 78). With the prevalence of personal cloud storage services, more team collaboration applications allow users to share files from their personal cloud-storage spaces through external shared links. For example, Asana allow users to share personal files to their team from Dropbox or Google Drive. In another example (Borcea, Ding, Gehani, Curtmola, Khan, & Debnath, 2015), Avatar is a mobile distributed computing system using cloud resources, where distributed storage is leveraged to access data. Amazon EBS and a shared storage system (SAN or NAS) are used to distribute the data set to storage system. However, this approach is not convenient for team collaboration, and not safe enough. On the one hand, external links of publicly accessible files are not conducive to information security; On the other hand, a user need to upload files to his/her personal cloud storage applications, and then share via external links to team collaboration applications, which is cumbersome.

In this paper, we propose an approach to leverage existing personal cloud-storage services to provide shared storage for team collaboration applications. By using this approach, we can build a virtual shared storage space for team collaboration applications based on third-part personal cloud storage applications, thus allow users to contribute their own personal cloud storage space, no matter where the space comes from, to support their team collaborations. Compared to existing approaches, our approach provides sophisticated mechanisms to make sure it’s more convenient and safer. It brings benefits in three folds: for users, it improves the utilization of personal cloud storage space; for vendors of personal cloud storage service, it helps attract users to use their services; for vendors of team collaboration applications, it reduces the burden of developing self-built storage infrastructure.

2. BACKGROUND

Team collaboration application is software designed to help a team of persons involved in a common task achieve goals. It has gone through a long history of evolution and development. In 1951, Collaborative computing was first envisioned by Douglas Engelbart (Engelbart, 2001). From then on, the concept and application has developed a lot. The term computer-supported cooperative work (CSCW) was first coined by Irene Greif and Paul M. Cashman in 1984 (Grudin, 1994), to address how collaborative activities and their coordination can be supported by means of computer systems. The use of collaborative software in the workspace creates a collaborative working environment (CWE) (Ning, Zhou, & Zhang, 2014). A collaborative working environment supports people in both their individual and cooperative work thus giving birth to a new class of professionals, e-professionals, who can work together irrespective of their geographical location. With the emergence of SAAS concept, more and more team collaboration applications are provided as a SAAS service on the web, Basecamp and Asana are the most popular ones among them. As a leading enterprise management software provider in China and the Asia-Pacific region (kingdee, n.d.), We in Kingdee are also providing such a service to help our customers to collaborate with their teams more effectively, which will be given more details in the Case Study Section.

As mentioned earlier, shared storage is a must-have feature for a team collaboration application. Users need such a place to store those referred documents or artifacts produced for latter access by all teammates. This place should be kept safe for the team, and have well defined access control mechanisms. Most team collaboration application suppliers choose to provide this feature on their own at the early time. It is ok when users are not requiring big storage space. However, with the fast development of personal cloud-storage service, users are used to enjoying free and almost unlimited storage space on the web. They are also changing their expectations for shared storage on team collaboration applications. It’s not easy to meet such expectations by simply building their own storage infrastructure, especially for those small or medium vendors of team collaboration application.

A personal cloud storage service is an Internet hosting service specifically designed to host user files (Wikipedia, n.d.). It allows users to upload files that could then be accessed over the internet from a different computer, tablet, smart phone or other networking device, by the same user or possibly by other users, after a password or other authentication is provided. Dropbox (Dropbox, n.d.) is one of the most popular personal cloud storage service. It provides 2 GB plus 500 MB for each referral person up to 18 GB (free), 25 GB (free with HTC Sense 4 & 5 device), 50 GB (free with Samsung device) 100-500 GB (Pro accounts), 1 TB-Unlimited (Business accounts). Those
suppliers in China are more generous. For example, Tencent (weiyun, 2014) provides 10T for free, and Baidu (baiduyun, n.d.) provides 2T for free. More and more people are used to putting almost all the personal files on the cloud, for personal use or for collaboration work.

We see the fast development of personal cloud storage services not as a threat to team collaboration application suppliers, but a great new chance. Since there are so many good personal cloud storage services out there, why bother to build our own storage infrastructure? Why not simply leverage the existing personal cloud storage services to provide shared storage for team collaboration? Actually this is not a totally new idea. Some vendors are already trying this way. For example, Asana allows users to attach a file to a task from Dropbox (A. D. Integration, n.d.). When you go to attach a file to a task in Asana, you’ll see the new option to “Attach from Dropbox”. This option gives you one-click access to your entire library of Dropbox files, where you can browse and choose the file you want to attach. However, this approach is not convenient for team collaboration, and not safe enough. On the one hand, external links of publicly accessible files are not conducive to information security; On the other hand, users need to first upload files to his/her personal cloud storage applications, and then share via external links to team collaboration applications, which is cumbersome.

We believe a new approach to leverage personal cloud-storage services to provide shared storage for team collaboration is desired. Different from the existing approach, we develop a novel approach which allows us to build a virtual shared storage space for a team collaboration application based on third-part personal cloud storage applications, thus allow users to contribute their own personal cloud storage space for team collaboration. The approach provides sophisticated mechanisms to make sure it’s more convenient and safer. It brings benefits in three folds: for users, it improves the utilization of personal cloud storage space; for providers of personal cloud storage service, it can help attract users to use their services; for providers of team collaboration applications, it can reduce the burden of developing self-built storage infrastructure.

3. THE PROPOSED APPROACH

We assume most of the users of a team collaboration application have their own personal cloud storage space, and that those providers of personal cloud storage provide open API for third-part application to access the space on behalf of the user. In most cases this requirement can be easily met. For example, Dropbox (S. Marx, n.d.) and Google Drive (Google-Developers-Site, n.d.) both allows developer to use OAuth 2.0 to access its core API.

Figure 1 shows an overview of the approach. When a user is joining a team to collaborate with other teammates on a team collaboration application, he/she can choose to contribute some of his/her personal cloud storage space for the team to use. He/she simply registers his/her personal cloud storage application to the team collaboration application. All contributed personal cloud storage spaces by the teammates together forms a virtual shared storage space. This virtual shared storage space is managed by a manager of storage space. This manager of storage space is a key component in this approach. In the following, we will describe in more details about how it works.

![Figure 1. Overview of the approach](image-url)
Figure 2 shows the process for space allocation. When a user joins a team in a team collaboration application, he/she needs to specify the source of his/her personal cloud storage space. This can be done easily. For example, when a user binds his/her account with a Dropbox account, the manager of storage space will know that its personal cloud storage space is from Dropbox. Now the manager of storage space can act on behalf of the user to read file from and write file to his/her personal cloud storage space. The user can specify the size of the contribution of storage space he/she prefers to contribute for the team; or the team collaboration application can do this job for all the team members to request each one to contribute certain amount of storage space. The manager of storage space can now update the record of the total size of available storage space for the team.

Figure 3 shows the process for a team member to store files to the shared storage space. Whenever there is a need for a team member to share a file with his/her team, he/she can simply store the file into the shared storage space of the team in the team collaboration application, the rest of the work about how the file is stored into a specific personal cloud storage space is taken care of by the manager of storage space. When detecting a user is going to save a file to the shared storage space, the manager of storage space first determines whether there is enough shared storage space for the team, and then decides to store the file to an appropriate personal cloud storage space based on certain storage policy (we will talk about this in more detail in the Analysis Section), and then records the file storage location for subsequent file operations (such as for reading the file, adjusting the file, deleting the file, and so on). After the file is stored successfully into the personal storage space, the manager of storage space needs to update the size of available shared storage space to make sure the number of the size is always correct.
Figure 4. Process for file operating shows the process for file operating on the shared storage space. Those files stored in the shared storage space are available for team members to perform various operations, such as read, update, delete and so on. When a user requests to operate on a file, the manager
of storage space checks the storage location of the file according to the previous record, and then goes to the appropriate personal cloud storage space to retrieve the file for the user to operate on it. When the operating is finished, the manager of storage space saves the updated version of the file back to the personal cloud storage space. If it’s an editing operation which causes enlarging size of file, storage space manager determines whether the space of the personal cloud storage where the file is going to be updated, is spacious enough to store the file of new version. When there is no enough empty contributed personal storage space for the new version of file. Then the manager needs to reallocate the file to another contributed personal storage space which is big enough for the file according to the priority of each personal cloud storage space and record storage location of files. The priority is determined by the sort according to a certain policy, which is detailed in the Section IV-B. If it’s a deleting operation, delete the location record of files and update the size of contributed personal storage space.

*Figure 5* shows the process for adjusting the storage location of a file. There are many cases this process could be triggered. For example, a user might quit from a team or decide to contribute smaller storage space. In these cases, the manager of storage space first needs to retrieve those files needed to relocate from the user’s personal storage space, and then stores the files to the appropriate personal cloud storage space according to the specific storage policy and record the files storage location.

![Figure 5. Process for adjusting storage location](image1)

![Figure 6-a. Process for adjusting the size of shared storage space](image2)
Figure 6 shows the process for adjusting the size of the shared storage space. For a team collaboration work, there might be some cases where some of the teammates quit and where team should be divided into several smaller teams. In the first case which is shown in the Figure 6-a, since several team members are quitting from the team, the shared storage space is not enough for storing the needed files. In this setting, the size of the shared storage space should be updated and request extra personal storage space of each teammates according to a specific policy presented in the Section IV-B. At the same time, files originally stored in the quitting teammates' personal storage space should be reallocated by the manager for safety.

In the second case which is shown in the Figure 6-b, when team is going to be restructured, manager is responsible for the shared storage spaces allocating for each team and one shared storage space for collaboration between teams according to the policy specified in the Section IV-B and the organization of each team. At the same time, the manager sets the access control of shared storage space for each team.

4. Analysis

Variants of the approach can be developed to meet the needs of different application scenarios. In the following, several policies are introduced for providing shared storage space for the team work.

4.1 Policy for Contributed Storage Space.

For providing shared storage space for the team, one of the options we are thinking is to allow users to save a file for the team only in his/her personal cloud storage space. In this way, a user does not need to pre-specify amount of storage space he/she want to contribute, on the contrary, his/her contribution is on demand: the amount he/she uses is the amount he/she needs to contribute. And it’s easier for a user to control those files he/she contributes to the team, because all the files he/she contributes are stored only in his/her personal cloud storage space.

Generally speaking, each collaboration work suffers from the risk of personnel shifting. Especially in some special cases when most of teammates suddenly quit from the team. In this setting, the files stored in those quitting teammates’ personal storage space should be relocated to another appropriate personal storage space. And there are two possible situations might occur.

1) One of situations is to relocate files. It is enough for shared storage space to store those files to be relocated after withdrawing those personal storage space from quitted members. So the manager of storage space select appropriate personal storage spaces from the rest of teammates. To ensure load balancing of the team member’s personal storage space, a simple averaging policy could be used, or a sorted policy could be used. And those two strategies are carefully presented in the Section IV-B.

2) Another situation is when the rest of the shared storage space is not available enough for files to be relocated. In this situation, one measurement is to request team members for much more contributed storage. The manager of storage space firstly assigns the files to be relocated to team members according to the available personal storage space left of them. During this assignment process, the priority of the file can be taken into consideration, combining with the priority of team member and the sort of personal cloud storage space by their previous contribution to the shared space. However, since the exodus of team members in some extreme situation, the rest of
personal storage space is too small to be managed to store those files. Another possible measurement to fix this problem is to bind an autonomy account for the team with Dropbox for some extra personal storage space as a share space. When the situation occur, the extra space can be triggered for storing the files to be relocated. The extra space is used as a cache to store files when the shared storage space is not enough.

4.2 Policy for Storing Files

In the process for a team member to store files to the shared storage space, different storage policy can be used to meet different requirements. As mentioned in the Section IV-A, to ensure load balancing of the team member’s personal storage space, there are two policies could be used:

1) First of all, a simple averaging policy could be used. when assigning the files to the personal storage space. The manager of storage space always make sure the used spaces are equal for each team member. There are two advantages of this policy. First, it is easier for manager to assign files to the personal storage space by recording and computing the amount of used space for each team member.

2) Another one is a more sophisticated policy could be use according to the characteristics of user-contributed cloud storage space. The manager sort the contributed personal cloud storage spaces by the characteristics, which are access speed, cost, reliability, size of the personal storage space, etc. A) Access speed. Since the personal storage space might be distributed in the different nodes of network, personal storage spaces in the different nodes represent their specific access speed when downloading files. B) Cost. Since the files are retrieved from the network, the throughput and congestion of the network influence the cost of downloading a file from different routines of the network. Consequently, when a file is stored in the space to which connection from other team members is congested, it will cost much more to download files. C) Reliability. It depends on the quality of the network. Files with high priority should be stored in the personal storage space in the high quality network which is more reliable. D) Size of the personal storage space. The characteristic is leveraged for balancing the usage of the personal space. Personal storage space with larger size is prior to be used when storing files. To sort the contributed personal cloud storage spaces, characteristics above is quantified and synthesized as a weight for each existing personal storage space. The bigger the weight is, the more possible it will be first chose to store the file during the collaboration work.

5. Case Study

The proposed approach is implemented in kAct, an online team collaboration application provided by Kingdee. As shown in Figure 7, kAct is a task-based team collaboration application. Users create task to represent work in a team. A task can belong to a project, and can be assigned a due date. Each task has one assignee, and can be followed by other team members. All the files can be attached to a task by any members in the team, whether it is a referred documents or an artifact of the task.

It’s not new to use a computational task or activity structure to support people’s collaborative work. The traditional work flow systems (Georgakopoulos, Hornick, & Sheth, 1995) is based on a business process model, where activity is a core concept. They can help capture the work processes and guide users through the work, although they are often too rigid and frequently assume fixed roles for users and a fixed pattern for actions. Kreifelts (Kreifelts, Hinrichs, & Woetzel, 1993) proposed a Task Manager which is based on shared representation of tasks that are malleable and that relate people and resources. In order to support knowledge management for problem solving processes, Wang (Wang, & Haake, 1997) developed a system to enable users to define and modify activity spaces according to their needs. Ahn’s KC-V based system (Ahn, Lee, Cho, & Park, 2005) is also built on an activity centric context model for virtual collaborative work. In recent years, to use task or activity as the core of collaboration model is gradually attracting the attention of research communities and mainstream manufacturers. For example, IBM proposed a unified activity management (UAM) model (Moody, Gruen, Muller, Tang, and Moran, 2006), (Moran, 2003), (Moran, 2005), (Geyer, Muller, Moore, Wilcox, Cheng, Brownholtz, ... & Millen, 2006) in 2005. This model provided a framework for activity management, which is based on an activity ontology and is more flexible and open than the traditional work flow systems (Moody, Gruen, Muller, Tang, & Moran, 2006), (Hill, Yates, Jones & Kogan, 2006), (Cozzi, Farrell, Lau, Smith, Drews, Lin, ... & Moran, 2006), (Moran, 2003). In addition, Unified Activity describing essential elements of activity are utilized to represent a complete activity as a computational construct and to provide an infrastructure. In this setting, a variety of
tools are integrated by the *Unified Activity* (Moran, 2005). UAM is now widely used in its IBM Connections Product (IBM-Connections, n.d.).

*Figure 8* shows the task based collaboration model of kAct. As is mentioned in the *Figure 8*, each task consists of several sub tasks. Thus the team might be restructured into couple of sub teams according to the sub tasks when necessary. Personal storages space are assigned by the manager of storage space. In this setting, files attached to sub tasks are delivered by the manager among sub tasks for collaboration work. Note that the manager is responsible for the storing and allocating files, recording and updating location of files, recording and updating storage space, and sorting personal storage space among sub tasks which are presented in the Section 3 and Section 4.

It is worth to note that each (sub) task can be annotated by different tags as mentioned in the *Figure 8*. The annotation process includes sentence splitting, tokenization and so on, before tagger is called. In fact, more and more mature taggers are developed currently, such as HeidelTime (Strötgen, & Gertz, 2010) and Biterm Topic Model (BTM) (Yan, Guo, Lan, & Cheng, 2013), which can be selected according to the context. Similarly, files are annotated by a set of tags. Thus files are efficiently attached to a certain (sub) task by matching tags between files and tasks. Furthermore, output of the (sub) tasks are annotated for collaboration work among tasks. By using annotated output of each (sub) task, a more powerful and complex task could be composed of sub tasks.

![Figure 7. Task View of kAct](image-url)
Figure 8. Task based collaboration model of kAct

Figure 9. Sub task based collaboration model of kAct
In the early version of kAct, we build our own infrastructure to provide shared storage space for team collaboration. However, with the fast growth of subscribers, we feel it is becoming a big burden for us to provide big storage space for our users. Moreover, with the fast development of personal cloud-storage service, users are used to enjoying free and almost unlimited storage space on the web and they are also changing their expectations for shared storage on team collaboration applications. So we decided to turn to third-party services to solve this problem, and the proposed approach is invented.

The first provider of personal cloud storage we integrated is Baiduyun (http://yun.baidu.com). It is a big player in the china market, and has over 100 million users. It offers 2 T storage space for each user for free (http://yun.baidu.com/1t). It provides open API for developers to use through its PCS (Personal Storage Service) (PCS, n.d.), including REST API and SDKs for Java, IOS, Andorid, Win7, and so on.

The interaction relationships among developers, users, third-part applications, REST API, and PCS are shown in Figure 10. First, developers can create an application in the Baidu developers center to get the client id and client secret of the application; second, a user logs in and authorizes the developed third-party application; third, the third-part application use the id of the application to obtain the logged in user’s authorization credential; fourth, through OAuth 2.0, the third-part application can obtain the user’s authorization credential Access Token; fifth, the third-part application saves the Access Token; sixth, the user accesses PCS through the third-part application; seventh, the third-part application calls the REST API to manage user’s data, Meanwhile passes in the Access Token; eighth, the REST API accesses user’s personal storage space to manage the user’s data; finally, the...
result will return to the user by the third-party application.

By using the proposed approach presented in Section 3. It brings benefits in many folds: for users of kAct, it allows them to have much bigger size of shared storage space for their team, for free; for users of Baiduyun, it improves the utilization of their personal cloud storage space; for Baiduyun, this way can help attract more users to use their services; for us, the providers of kAct, it reduces the burden of developing self-built storage infrastructure.

6. CONCLUSIONS

This paper presents an approach to leverage personal cloud storage services to provide shared storage for team collaboration applications. By using this approach, we can build a virtual shared storage space for a team collaboration application based on third-party personal cloud storage applications, thus allow users to contribute their own personal cloud storage space for team collaboration. Compared to existing approaches, our approach provides sophisticated mechanisms to make sure it’s more convenient and safer. The approach is implemented in our own task-based team collaboration application, by integrating with one of the famous provider of personal cloud storage service in China. The result shows benefits in three folds: for users, it improves the utilization of personal cloud storage space; for providers of personal cloud storage service, it can help attract users to use their services; for providers of team collaboration applications, it can reduce the burden of developing self-built storage infrastructure. Regarding the future work, on the one hand, we are planning to develop more variants of the method to meet the needs of different application scenarios; on the other hand, we are also planning to integrate more personal cloud storage services into kAct and develop novel strategies to make sure the collaboration among those services in the coming future.

ACKNOWLEDGMENT

This work was supported partially by the National Natural Science Foundation of China (Grant No. 61379126), by the Scientific Research Foundation for Returned Scholars, Ministry of Education of China, and by the Fundamental Research Funds for the Central Universities (China University of Geosciences (Beijing), China); the central grant funded Cloud Computing demonstration project of China, R&D and industrialization of the SME management cloud (Project No. [2011]2448), hosted by Kingdee Software (China) Company Ltd., under the direction of National Development and Reform Committee of China; the construction fund of National Engineering Research Center for Supporting Software of Enterprise Internet Services (Project No. 2012FU125Q09); the Guangdong province project, under grant No. 2015B010131008; the fund for leading talents of Guangdong Province [2012]342; the Shenzhen high-tech project, under grant No. FWYCX20140310010238.

7. REFERENCES


Ning, K., Zhou, Z., & Zhang, L. J. (2014, June). Leverage personal cloud storage services to provide shared storage for team collaboration. In Services
Computing (SCC), 2014 IEEE International Conference on (pp. 613-620). IEEE.


Authors
Zehui Cheng is currently a master student at China University of Geosciences (Beijing), China. Her research interests include services computing.

Zhangbing Zhou is currently a professor at the China University of Geosciences, Beijing, China, and an adjunct associate professor at TELECOM SudParis, France. His interests include services computing and business process management. He has published more than 100 referred papers.

Ke Ning is a research scientist with the Enterprise key Laboratory of application software, Shenzhen, China. His interests include services computing and knowledge management.

Liang-Jie Zhang is a computer scientist, a former Research Staff Member at IBM Thomas J. Watson Research Center, Senior Vice President, Chief Scientist, Director of Research at Kingdee International Software Group Company Limited, and director of The OpenGroup. He is the founding Editor-in-Chief of IEEE Transactions on Services Computing. He was elected as a Fellow of the IEEE in 2011, and won the IEEE Technical Achievement Award for pioneering contributions to Application Design Techniques in Services Computing.

Taj Rahman is currently working towards his Doctor degree in University of science and technology Beijing, China. His research interests include sensor networks and internet of things.

Jiangsong Min is a research project manager with Kingdee International Software Group Co., Ltd., China. His research interests include cloud computing and data mining.
A COMPREHENSIVE OVERVIEW OF OPEN SOURCE BIG DATA PLATFORMS AND FRAMEWORKS

Pedro Almeida1, Jorge Bernardino1,2
1ISEC – Polytechnic of Coimbra, Portugal
2CISUC – Centre for Informatics and Systems of the University of Coimbra, Portugal
a21180299@isec.pt, jorge@isec.pt

Abstract

Big Data is the paradigm that represents the ability to analyze and cross-reference large amounts of data generated by computational systems and turn them into useful knowledge. This potential is one solution organizations can use to answer the challenge of getting closer to their users. Organization managers face the challenge of understanding the Big Data concept and the business strategies inherent to its use. The high number of challenges that need to be addressed creates a high number of proposed technical solutions that most times only overlap existing ones. Frequently managers face these issues as their organizations race against the competitors for a market share, without having resources to embrace not only Big Data but also other options that can give competitive advantage. Therefore, organization owners and managers must be educated on deployed platforms that can make them understand the benefits that can be achieved on short term. In this paper we aim to provide an overview of using Big Data with Open Source tools. We explain the Big Data concept, the potential value and the organizational strategies that must be studied in order to determine which benefits organizations can win from it. We analyze the strengths and drawbacks of five open source frameworks for distributed data programming – Hadoop, Spark, Storm, Flink and H2O – and seven open source platforms for Big Data Analytics – Mahout, MOA, R Project, Vowpal Wabbit, Pegasus, GraphLab Create and MLLib. There is no single platform that truly embodies a one size fits all solution, so this paper aims to help decision makers by providing as much information as possible and quantifying some tradeoffs.

Keywords: Big Data, Open Source, Data Mining, Data Analysis

1. INTRODUCTION

In a globalized world where competition between organizations is more aggressive, they need to be as close in time and accuracy as possible to the needs of their users. There is huge potential and value inside data waiting to be used by all individuals and organizations. The exploration of Big Data paves the way for everyone to extract insight and value from data generated both inside and outside their organizations or areas of interest (Assunção et al., 2015). Therefore, in corporate environment managers are going to have a better overview over their business and competitive advantages such as enhanced productivity, greater innovation and a consolidated position in the global market. However, the reality shows that very few managers are educated for the importance of data. In 2014 the OECD (Organization for Economic Co-operation and Development) registered that 95% of enterprises are Small and Medium Enterprises (OECD, 2014). These companies do not have a large budget to accommodate the necessary resources to explore the power of data. Organizations, their leaders and managers need to know about the potential of Big Data and this has to be done by showing practical solutions that allow them not only to see results and value in the moment but also to see potential profits to be obtained in a short amount of time.

The key lesson people have to understand about Big Data is that we can take advantage of it with minimal cost by using the vast myriad of Open Source Big Data Platforms. These platforms are the combination of hardware infrastructures and software tools used to acquire, store and analyze data. Data has value during a period of time. This window of opportunity is variable depending on the environment where data is being used. In business environments
where costumer behavior and necessities change at a fast pace data only has value for a short period of time (Christofferson, 2014). This makes processing speed one of the key features of Big Data platforms. Integrity of data is also an important aspect the platforms must consider because corrupted data not only has no value but can lead managers to make wrong decisions based on erroneous facts.

To complicate the work of data scientists trying to explain Big Data there are a number of issues that make it hard to have one or at least a small group of solutions ready to be presented to managers. It is not recommended that technical solutions are shown without giving an introduction on what the Big Data model is. It is also important to know ahead what the needs of the enterprise are so the chosen computational system can extract value that suits such needs.

Working with Big Data is a complex process where the data analysis steps may require repetition until acceptable results are generated. This creates a vast amount of challenges that practical implementations have to address. There is a large amount of conceptual but also technical challenges that lie within exploring Big Data.

This leads to the existence of a high number of different platforms where most don’t do more than overlapping what already exists in previous solutions. An important idea to retain is that the study and choice of what systems to use has to be done within a short period of time because of two problems – this field of research is constantly evolving and solutions deployed today may not be useful tomorrow.

This happens for various reasons but the most important one is that the growth of data is exponential and solutions may stop being able to handle such large quantities of data in a near future making the investment worthless before it can give return on the investment. Enterprises wanting to optimize their scarce resources cannot afford to waste time with exploring all the available platforms and need help in finding the most suitable solution for their specific needs. For most of them it does not even come as a matter of wanting or not to explore the potential of Big Data but just as a matter of absolute impossibility due to lack of resources and know-how.

Another essential lesson to be retained is that it is impossible to name one definitive platform that can be named as the best one for the needs of every individual, enterprise or organization.

A great deal of research has already been done about Big Data and Big Data Platforms (Mayer-Schnberger et al., 2013), (O’Reilly Media, 2012), (Barata et al., 2014a), (Barata et al., 2014b). This research is usually done in two ways. The first is to study the requirements of one organization and create a set of tools that fulfills their specific needs. This approach requires a large amount of investment and time that only big enterprises are able to afford. The second is by studying the existing platforms and choosing the one that is closer to the requirements of the enterprise. This approach is preferred for Small and Medium Enterprises because it requires no further investment or development. Nevertheless, proprietary solutions require a significant investment. Open Source platforms offer solutions that are free but also easier to adapt to the specific requirements of each enterprise.

In this paper we give a comprehensive overview of the use of Big Data with Open Source platforms. We begin by explaining the concept and the organizational strategies that need to be studied before moving forward for the choice of technical solutions. After that we study and compare five frameworks for distributed data processing (Hadoop, Spark, Storm, Flink and H2O) and seven Big Data Open Source Platforms (Mahout, MOA, R Project, Vowpal Wabbit, Pegasus, GraphLab Create and MLlib). We compare not only their technical characteristics but most important their capabilities for insertion in the segment of Small and Medium Enterprises. We compare this capability through analysis of parameters such as the ease of use of the interfaces and availability of programmers to manage the platform.

This paper is a revised and extended version of our Big Data Congress 2015 paper (Almeida et al., 2015). New materials include (i) a new section describing five distributed programming frameworks (Section 4), (ii) a new section comparing these five frameworks (Section 5) (iii) and the addition of a new platform to the list of analyzed open source platforms (Section 6).

The remainder of this paper is structured as follows. Section 2 explains the Big Data model and the Big Data Strategy. Section 3 overviews the features and requirements involved into Big Data Open Source Platforms and the Map Reduce programming paradigm. Section 4 describes the five distributed data processing frameworks. Section 5 compares the frameworks and provides an analysis on relevant characteristics for organizations needs, such as development maturity, modularity and integration. Section 6 describes the seven Big Data Open Source Platforms. Section 7 compares the platforms and studies if their features are helpful to fix the needs of enterprises. Finally, Section 8 presents concluding remarks and points out some future work.
2. THE BIG DATA MODEL EXPLAINED

There is not one final definition about what exactly is Big Data. In a very simple way Big Data can be defined just as a volume of data that is so large that it is difficult to process using traditional database and software techniques (Greer Jr., 2013). One of the first definitions was the 3V’s concept that considered Volume, Velocity and Variety (Laney, 2012), (Fayyad, 2012). This concept has evolved to 4V’s with the addition of Veracity (Vossen, 2013). The concept has further grown to include properties such as Variability, Viability and Volatility. It is important to mention that the most important V for business environment is Value (Marchand-Maillet et al., 2014). For our study we use the 6V’s definition that includes Value, Variability, Veracity, Volume, Velocity and Variety as can be seen in Figure 1:

- **Volume** in Big Data refers to amounts of data in the terabyte level or higher. Data this size presents new challenges when it comes to tasks such as retrieving, indexing and processing. These issues cannot be handled by traditional RDBMS and thus new tools are required;
- **Velocity** is the speed at which new data is generated and processed. High speed brings constraints to operations with data. The main concern for data scientists about velocity is the high cost of retrieving data that can be left behind if a stream is not processed fast enough;
- **Variety** refers to the mix of available data types that can present various levels of structure. Most data today is semi-structured and unstructured and not supported by RDBMS. Variety can also refer to the different sources data is generated from – both inside and outside the organization;
- **Value** is the knowledge extracted from a large amount of data. It can be perceived as the understanding of the behavior of a user under a certain context;
- **Variability** refers to data whose meaning is uncertain. Data isn’t always accurate and sometimes presents out of the ordinary values that require additional study to decide if they should be considered or discarded;
- **Veracity** addresses the confidentiality, integrity, and availability of the data. This includes questions such as if the data is trustable;

The aforementioned characteristics are important because they should be ever present in the process of developing technologies and platforms whether they act upon one or more steps of the Big Data value chain. There are different versions of the Big Data value chain described in the literature.

One is proposed by Miller et al., (2013) but for this paper we will use the one proposed by Hu et al., (2014). The value chain divides the lifetime of data into four different phases - Data Generation, Data Acquisition, Data Storage and Data Analysis.

Data Generation represents several aspects related to how data is generated such as its sources and its domain-specific values.

Data Acquisition represents the process of obtaining the information that is subdivided into three smaller processes that are data collection, data transmission and data pre-processing.

Data Storage concerns the persistent storage and managing of large-scale datasets. Finally, Data Analysis aggregates the analytical methods to inspect, transform and model data for knowledge extraction (Hu et al., 2014).

Any individual or group user wanting to put a Big Data Platform should have in mind that before the platform itself is chosen and deployed it is advisable to study and implement a Big Data Strategy, that helps managers understand exactly what they want from the eventual platform they are going to use. According to Huddar et al., (2013) a Big Data Strategy consists mainly of three different areas:

1) Big Data Basics that usually represent the acknowledgment of the diverse types of data available such as social data, preprocessed data or unstructured data.

2) Big Data Assessment evaluates several data aspects such as the source, the potential uses, volumes, estimated future growth and privacy regulations.

3) Big Data Strategy in itself that studies the impacts of Big Data in the organization,
opportunities to be taken and business cases where Big Data can be of use. It also reports economic impacts such as the potential return of investment.

Only after that strategy is documented and thoroughly analyzed are organizations and managers able to effective choose what functionalities they will have to look for in a Big Data Platform.

3. Open Source Big Data Platforms Overview and the MapReduce Paradigm

A platform capable of supporting the large kind of datasets that are not manageable by traditional database tools can be considered a Big Data Platform (Gupta et al., 2014a). A generic goal of such platforms can be formulated as to grant the abilities to integrate privately acquired and publicly available Big Data with data generated within an enterprise and to analyze the combined set for value extraction (Dijcks, 2014). This can be translated into a group of features that platforms should include such as - easy scalability and extensibility, being comprehensive and ready for enterprise use, robustness and fault-tolerance support, data updates with low latency and last low maintenance requirements. Nevertheless Big Data Platforms are not just features, but also hardware and software technologies. And they do not exactly need to be composed of the newest and most powerful technologies. A Big Data Platform could be deployed by just making a new configuration of the existing technologic infrastructures. Unfortunately this solution is not viable and therefore not widely used because modifying decades of old systems for the needs of the present comes with a very high price that few are able to pay for. However, this is an unnecessary cost because newer platforms are constantly being developed to answer these new requirements. Those requirements can be divided into three main phases – Data Acquisition, Data Integration and Data Analysis (Dijcks, 2014). In the Data Acquisition phase systems are being asked to provide lower latency on data capture, shorter execution on data queries, support for distributed environments and for most types of data structures. NoSQL databases are currently the solution that is closer to such specifications because they prioritize data capture over data categorization thus eliminating the overhead caused by the existence of a data scheme (Abramova et al., 2013), (Lourenço et al., 2015). When it comes to Data Integration the preferred solution of organizing all the data at one single location is not possible anymore in the Big Data era. Moving around significant amounts of data while keeping its integrity is mandatory for big companies. To support Big Data frameworks and platforms, hardware must provide some requirements. The systems must have the capacity for either vertical or horizontal scaling or in the best case both. Vertical systems are systems that work with a single node composed of several components of powerful and expensive hardware. Horizontal systems on other hand are systems that are composed of multiple nodes of commodity hardware connected through some sort of network. Big Data frameworks should also provide capacity for distributed data programming, high throughput and support for both structured and unstructured data (Singh et al., 2014).

Systems that scale vertically are systems that grow by adding more processors, more memory and other hardware within the same server. This has obvious advantages such as easier setup, management and inexistent communication cost but it also has major drawbacks. At all times a system needs to be scaled vertically, an investment has to be made in the purchase of new hardware and most times the investment represents an upgrade in processing power that is unnecessary. It is also important to say that scaling vertically becomes an impossibility after a certain limit because of physical limitations and because current operating systems are restricted to handling a certain amount of hardware. All of the aforementioned issues of vertical scaled systems do not make them the first choice for Big Data but just a resource for specific tasks such as processing of intermediate results generated by horizontal scaled system. Horizontal scaled systems are better because both data analysis and data integration require the use of distributed environments to realize the tasks of deep analytics and statistics on a broad variety of data types. These systems have to comply with facts such as data being stored in different systems and constantly scaling up in terms of volume. They need to be faster at delivering answers and reacting to changes in data behavior. They also have to embrace the fact that data moving from one place to the other raises not only privacy concerns but also prohibitive costs for organizations to support. Data mining and analysis tasks are carried out in multiple locations with intermediate results being sent to a central location. At this central location results are grouped back together for another process of final analysis. This two-step process brings even more complexity especially for the development of algorithms because the analysis of intermediate results will be not as precise as the analysis of raw data. Not only are intermediate results an average of the raw data results and also noise and arbitrariness can be introduced for
privacy maintenance (Wu et al., 2014). To sum up, while vertical scaled systems are important and useful for some complementary tasks, the key for Big Data is in the frameworks that support horizontal scaling. Frameworks that scale horizontally are superior because they are better at providing the three main features we mentioned before – distributed data programming, high throughput of data and support for all types of data. All practical implementations of software frameworks for distributed storage and processing that are used for Big Data are based on a programming model. One of the most used and developed programming models is MapReduce (MR). MR is a programming model that was proposed by Jeffrey Dean and Sanjay Ghemawat at Google in 2004 (Dean et al., 2008). MR is a batch process oriented paradigm that provides algorithms for problems too big and complex to be run on a single machine and that can be processed in parallel in a cluster of nodes (Lee et al., 2011). The basic flow of a MR process is composed of two procedures – the Map procedure that processes a small chunk of the dataset and generates intermediate results and the Reduce procedure that groups the intermediate results and processes them to generate the final output. The strong point of MapReduce is that if the algorithm is developed in a way where the various operations are independent from each other then they will be able to be run in parallel allowing a considerable saving in processing time. Tasks are distributed to the nodes in the network with nodes being forced to report periodically with either work or status report. The master node keeps track of which tasks where given to which nodes and if one node fails, it is considered inactive and the tasks given to it are sent for other nodes to perform and the whole network is automatically configured accordingly. Nevertheless it has also some weak points. Distributing the work through a large number of machines has a communication and data transfer cost that sometimes may not be compensated by the gain in computational speed or throughput. Also MR tasks are acyclic data flows where all mappers are performed before all reducers accordingly to a schedule made by a batch job scheduler. This imposes serious limitations to areas such as Machine Learning (ML). Almost all algorithms in ML are iterative and revisit the same data a few times. Reasons such as, the high relying on slow I/O operations, lack of use of in-memory and the change of paradigm from batch processing to streaming processing are among the reasons why MR is being phased out in favor of new models and implementations such as Spark (www.spark.apache.org/), Storm (www.storm.apache.org/), or Kafka (www.kafka.apache.org/). MapReduce and its open source implementation Hadoop are at the moment the data programming model with most implementations in place for the treatment of large sets of data (Dörre et al., 2015). Big Data Platforms still have a long path to walk through as not only do they need to come up with new solutions for the issues we explained before but also for new challenges that arise as Big Data raises in volume and importance.

4. OPEN SOURCE DISTRIBUTED PROCESSING FRAMEWORKS

In this section we describe five open source distributed processing frameworks – Hadoop, Spark, Storm, Flink and H2O.

4.1 Hadoop

Apache Hadoop (www.hadoop.apache.org/) is a software framework created in 2005 by Doug Cutting and Mike Cafarella as an open source implementation of MapReduce. Written mainly in Java (with some C and Shell script code) it aims at being a framework for both distributed storage and processing of large data sets in computer clusters composed of commodity hardware but also on cloud platforms. The core of the framework is composed of the Hadoop Distributed File System (HDFS) for storage management (but other file systems are supported) and the MapReduce implementation to manage the processing.

The remaining of the base framework includes two other components. The first one is Hadoop Yarn (www.hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/), a platform responsible for the management of the available resources in the system. The other one is Hadoop Common, a set of libraries and utilities that provide abstraction of the file systems and operating systems among other necessary features to aid the others modules in their tasks. Additional packages that can be integrated in the Hadoop ecosystem include Pig (www.pig.apache.org/), Hive (www.hive.apache.org/), Spark and HBase (www.hbase.apache.org/) that is an open source implementation of the BigTable distributed storage system proposed by Chang et al., (2008).

An example of the Hadoop Ecosystem is shown in Figure 2. In terms of cluster or grid, a Hadoop system will include one master node and several slave (worker) nodes (Olson, 2014). The master node configuration is variable and can vary...
The first component is the Job Tracker. It is responsible for accepting the MR tasks, distribute them to the Task Trackers in the slave nodes and keep track of what is being done and where. Next are the Task Trackers that are responsible for processing the data received and also to report periodically to the Job Tracker. Ideally they would only exist in slave nodes but in small clusters where all resources need to be used they will also exist in the master node. After that comes the Name Node, which represents the indexing of the data stored in the cluster. That data consists of the Data Nodes that are stored, replicated and moved around the various nodes of the cluster during the execution of the processing.

Ideally (usually in large clusters) the Job Tracker will be placed alone on one node that will become the node responsible for all the job scheduling. Also on its own machines should be the Name Node and a secondary Name Node. The secondary Name Node makes regular copies of the main Name Node directory structure so that when the main Name Node fails and is restarted it can skip the file system journaling task and get active in a faster way.

The basic flow of an algorithm run on Hadoop starts with the split of data stored in HDFS into large blocks and their distribution among the nodes in the cluster. MapReduce will then transfer to the nodes the necessary code for the data belonging to that node to be processed. This will take advantage of data locality and reduce communication costs. This work distribution can only be achieved if the file system provides location awareness. If one algorithm is well designed then the various processing tasks will be independent from each other and able to be run in parallel (Ullman, 2012).

But Hadoop will still inherit the limitations associated to MapReduce when it comes to iterative algorithms. To fix this, Bu et al., (2010) has presented a modified version of Hadoop called HaLoop that is designed to serve iterative algorithms and applications. Also the file system will guarantee that several copies of the data are placed in different nodes of the cluster in order to guarantee redundancy in case one of the nodes has a failure.

All the fundamental modules of Hadoop designed based on the assumption that commodity hardware is likely to suffer from hardware failures and that such failures should be treated by the software.

### 4.2 Spark

Apache Spark (Spark, 2015) is an open source new generation platform for computation in memory originally developed at the University of Berkeley and now in development as one of Apache’s Top Level Projects (Apache Software Foundation, 2014). Unlike the previous generation platforms such as Hadoop, Spark is not oriented for batch processing but rather streaming processing. By giving the user access to in-memory, Spark through its direct acyclic graph (DAG) engine allows the loading of data to such memory and the possibility for processing various queries in it. This makes Spark roughly 100 times faster than its competitors in some types of applications and makes it very good for the implementation of machine learning and data analytics algorithms. Spark is the name of both the framework but also the processing module of the framework. This means the framework still needs a cluster manager module and a distributed storage module. For cluster management it supports not only its own standalone native cluster but also Hadoop Yarn and Apache Mesos (www. mesos.apache.org/). When it comes to distributed storage Spark includes support for HDFS, Cassandra, OpenStack Swift and online storage services such as Amazon S3.
Spark project consists of five major components. The first one is the Spark Core. Spark Core provides methods for task dispatching and scheduling in distributed systems along with basic input and output functionalities. Next is Spark SQL a component on top of the core that provides data abstraction called DataFrame and support for both structured and semi-structured types of data. Then comes Spark Streaming. Spark Streaming improves on the natural capacity of the Core to schedule jobs in a fast manner making it suitable for streaming analytics. In a more simple way it basically serves as a translator of code of batch processing applications for the streaming paradigm. Remaining modules include GraphX (Gonzalez et al., 2014), a framework for distributed processing of data in the form of graphs and MLLib (www.spark.apache.org/mllib/), a framework for distributed machine learning that we will explore in detail in section 6.7. An illustration of the Spark ecosystem can be seen in Figure 4. Spark is known to operate at clusters as large as eight thousand nodes and as of November 2014 holds the world record on large-scale sorting for sorting 100TB of data in 23 minutes in the Daytona GraySort Contest (Xin, 2014).

4.3 Storm

Apache Storm (www.storm.apache.org/) is another of the new generation frameworks for distributed streaming processing. Originally created by Nathan Marz at BackType it was then acquired by Twitter who immediately transformed it in an open source project (Cheredar, 2011). It is a cross-platform framework written in both Java and Clojure programming languages. It has a wide range of use cases that go from real time data analytics to online machine learning but also include distributed remote procedure calls and data extraction, transforming and load (ETL). Storm applications are designed with a custom topology that makes the distributed processing of streaming data look like it is in fact batch processing thus making the change of paradigm easier for developers and managers to understand and eventually implement. Storm aims at having the same success with stream processing that Hadoop had with batch processing. We can see the topology of a Storm application in Figure 5. The topology has the shape of a direct acyclic graph where the vertices are the data sources (known in Storm as spouts) and the data manipulation nodes (known in Storm as bolts). The edges of the graph are the pieces of data (known in Storm as tuples). Both spouts and bolts can accept a multiple number of input sources and generate many output sources. The output of a bolt doesn’t necessarily have to be placed in a spout but can be the input of another bolt that will process the data in a different way than the previous one. Processed data can be stored in spouts after every step, at the end of the whole processing or never stored at all with the final output being just visualized along the way. The structure of a Storm job is very similar to the one of a MapReduce job differing in only two essential points. First, data is processed in a real-time continuous manner instead of as individual batches (even though in a very simple way we can interpret a tuple as a batch of data). Secondly, MR jobs have a starting point and an ending point - Storm jobs run indefinitely until killed. Storm most distinguishable feature is the easy way it can be integrated with the queueing database technologies already in use.

4.4 Flink

Apache Flink (www.flink.apache.org/) is a new generation framework for distributed data processing. This framework created in 2010 by three German universities under the research project “Stratosphere: Information Management on the Cloud” (Alexandrov et al., 2014) is designed specifically for Big Data Analytics. While Spark eludes the user into thinking it is a stream processing oriented platform when in fact it is just a platform that processes small pieces of data in batch in a fast way, Flink is a framework genuinely built for streaming processing (Pointer, 2015). In most cases this does not make a difference. Although in environments such as the financial system or real
time auction every millisecond delay makes a difference. This causes Flink a better alternative to Spark. While Spark uses the resources of Storm to solve the low latency issues, Flink provides solutions for all scenarios in a single framework. Flink wants to be a system that fills the gap that exists between the MapReduce two step systems and the shared-nothing parallel database systems. It executes random dataflow processes in both data parallel and pipeline manner. This way it enables the execution of both batch and streaming applications along with the execution of iterative machine learning algorithms natively. Written in Java and Scala languages, the applications for it are automatically optimized into dataflow programs. This makes it possible for them to perform well in both batch and stream processing scenarios. Flink is just a processing module that unlike others does not have a storage component. For that it uses HDFS or HBase for batch processing and Kafka for data stream processing. Flink includes API’s for applications that use static data sources, applications that use unbounded data streams and for applications that use data in form of database tables. Additional pieces of the bundle are libraries for Machine Learning and for graph processing. An illustration of the Flink stack can be seen in Figure 6. Main characteristics include exploiting of the capacities of in-memory use and of data streaming, ability to make safe well written applications, easy to use and quickness of deployment because of the low number of required settings, good scalability proved in clusters of hundreds of machines and compatibility with more developed systems like Hadoop (Apache Flink).

### 4.5 H2O

H2O (www.h2o.ai/) is a new generation open source framework for parallel processing that like Flink was created specifically for the Big Data Analytics area (Kurzyniec et al., 2003). Along with the engine itself it contains its own Machine Learning Platform and other components such as tools for data preprocessing and for evaluation of data after processing. The H2O architecture structure is shown in Figure 7.

H2O has different processing methods depending on the needs of the algorithm at hand. All of these processes share the common characteristic of running completely in-memory. Besides its own native engine, H2O integrates with Spark and efforts are in place to allow integration with Storm too.

The basic approach of H2O consists in breaking processing tasks into parts as small as possible. This allows the exploiting of the capabilities of parallel processing in the best way possible on jobs that are batch type like MR, streaming or even processing of graph data. One of the most appealing features of H2O is that not only it has its own web-based Graphical User Interface but also integrates well with the known environments for programming, thus allowing for analysts with no programming background and for programmers that are new to H2O to start developing and deploying applications in a fast and easy way.

### 5. PROCESSING FRAMEWORKS COMPARISON

In this section we compare the five processing frameworks – Hadoop, Spark, Storm, Flink, H2O - using a group of eight characteristics that have been considered relevant by previous work – Fan et al., (2013), Singh et al., (2014) and Hu et al., (2014) - for business managers to look for when trying to choose a distributed processing framework. This structure will be the base over which a Big Data Platform will
be deployed. The eight characteristics are divided in two groups.

First, in Table 1 we list the processing model(s), the software requirements, the programming language(s) and the existence or not of native Machine Learning Tools. Secondly, in Table 2 we make an analysis on characteristics such as development maturity, modularity and finally integration. We also address the hardware requirements for each framework.

The processing model is important because the distinction between batch and streaming is directly entangled with the needs of the enterprise. Batch processing is so far the best way to handle large quantities of data. A platform that can provide both is the best of both worlds. Hardware and software requirements must always be taken into consideration. An open source framework does not give any benefit if it requires the acquisition of paid hardware and software. Programming languages are always an important characteristic because the enterprise will need human resources to manage the system and it is always more recommended to choose platforms with languages that are widely known and have a good availability of programmers for hire rather than a platform that uses more infrequent languages that have shortage of people who know how to code them.

Knowing if a framework has its own associated Machine Learning Tool and/or if it integrates well with other tools is also important because having to add a tool on top of the framework can be a process that can add management complexity and push managers away from a platform that needs that. Again with the Machine Learning tools characteristic is hard to give a best choice. All frameworks have their own one or integrate well with others. The weakest one here is likely Storm because it integrates with SAMOA that is not a tool developed and distributed with Storm like the other ML tools are distributed with their own frameworks.

Analyzing Table 1 we can now extract the following conclusions – When it comes to the processing model, Spark and Flink are obviously the best choices because they provide processing in both the batch and streaming model. Unless the manager knows well that s/he only needs one of these models not just now but also in the future, a framework that gives both options is at first sight a better choice. On software requirements all frameworks require Java Development Kit (JDK). Hadoop and Flink also require Secure Shell (SSH). Flink does not run natively on Windows and so Windows users will need Cygwin to be able to use it on that operating system. When it comes to the programming languages it is hard to name a better choice above the others. All of them support Java which is one of the most used languages nowadays and have no shortage of available programmers. Still, Storm, Spark and H2O can be highlighted for providing more freedom of developing as they support a higher number of languages than the others. None of the frameworks have strict hardware requirements. Such requirements are variable depending on both the algorithm to be run and size of data to be processed. All frameworks function accordingly to the parallel processing paradigm and are designed to run on clusters where each node may have different hardware specifications. The only framework that sets minimum hardware requirements is Spark that asks for 8GB of RAM, between 8 and 16 CPU cores per node and a network connection of 10 Gigabit or higher.

<table>
<thead>
<tr>
<th></th>
<th>Hadoop</th>
<th>Spark</th>
<th>Storm</th>
<th>Flink</th>
<th>H2O</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processing Model(s)</strong></td>
<td>Batch</td>
<td>Batch, Streaming</td>
<td>Streaming</td>
<td>Batch, Streaming</td>
<td>Batch</td>
</tr>
<tr>
<td><strong>Software Requirements</strong></td>
<td>JDK 1.7, SSH</td>
<td>JDK 1.6</td>
<td>None</td>
<td>Cygwin (for Windows), JDK 1.7.x, SSH</td>
<td>JDK 1.7</td>
</tr>
<tr>
<td><strong>Programming Language(s)</strong></td>
<td>Java</td>
<td>Java, Python, R, Scala</td>
<td>Any</td>
<td>Java, Scala</td>
<td>Java, Python, R, Scala</td>
</tr>
<tr>
<td><strong>Machine Learning Tool(s)</strong></td>
<td>Mahout</td>
<td>MLlib, Mahout, H2O</td>
<td>SAMOA</td>
<td>Flink-ML, SAMOA</td>
<td>H2O, Mahout, MLlib</td>
</tr>
</tbody>
</table>

*Table 1: Processing Frameworks Comparison*
The next three characteristics – Development Maturity, Modularity and Integration – are analyzed in the form of a one to five stars rating with one star being the lowest evaluation possible and five starts being the best. Development Maturity is the level of growth and deployment the framework has obtained. In this area just like in many others is hard to consider that a tool is completely developed and thus we do not give five starts to any of the analyzed frameworks. Modularity is an important feature because it represents how well the framework has its various features divided in different modules. The more modular a framework is, the easier it is to adapt it to specific needs and to exchange some modules for others that may perform better at specific tasks. Integration is a characteristic that is quite interconnected with modularity because then again the more modular it is the easier it is to integrate with other frameworks in its totality or just with some modules without much technical complexity. Examining Table 2 we can conclude that when it comes to development maturity Hadoop and H2O are by far the best platforms as they are the ones that exist for the longest time, are more deployed and have the biggest number of released versions. This is not necessarily a good thing because they work only with batch processing. The reality is that batch processing is being phased out in favor of streaming processing. A manager looking at this characteristic has to consider the choice between a robust system for the needs of the present and a younger system that can adapt well to needs of the future. When it comes to modularity, Hadoop and Spark are the ones that have their functions more divided. But they do not have much difference to either Flink or H2O. Storm gets a lower rating here because of all the frameworks analyzed it is the one that is closer to be just a processing framework. Because it has a more specific function it is not as modular as others. But this is not necessarily a disadvantage. With integration Spark and Storm take the advantage. They integrate better on top of other frameworks, aside them and with just some modules interchanging.

As global conclusions we can say that – Spark and Flink are the best choices. They give the power of both batch and streaming processing, they offer different possibilities within the programming languages and got their own ML tools. They are mature enough to provide stability of use, have a good modularity and integrate well enough with other tools. Hadoop and H2O are also good choices if the manager wants a framework that is mature enough, thoroughly debugged and able to provide the best results for the present while s/he waits for newer frameworks to improve.

### 6. BIG DATA OPEN SOURCE PLATFORMS

Based on the work of (Bifet, 2013) we chose the following Big Data Open Source Platforms: Apache Mahout, MOA, R Project, Vowpal Wabbit, PEGASUS and GraphLab Create. Additionally we chose MLLib because we find useful compare a newer platform with older ones. In this section we describe these tools in some detail.

#### 6.1 Apache Mahout

Apache Mahout (www. mahout.apache.org) is an open source project aiming to build a comprehensive library of machine learning and data mining algorithms. The main feature of Mahout is that all the algorithms in its library are highly scalable and able to perform well on both standalone machines and distributed environments. It runs on top of the Hadoop environment and make use of such technologies such as HDFS and MapReduce. Mahout currently has implementations and support for most of the machine learning tasks of supervised and unsupervised nature such as Recommendation Mining, Clustering and Classification. Recommendation Mining is the algorithm that mines and analyzes user behavior and builds recommendations on similar items the user might like. Clustering is characterized as the set of algorithms aimed at analyzing text documents and group them into topic related groups of texts. Classification comprises the set of algorithms that use previously obtained information from already categorized...
documents to assign new ones into the most suitable category. Other tasks available are Collaborative Filtering, Dimension Reduction, Topic Models and Frequent Pattern Mining. It is important to refer that Mahout is only a library. Mahout presents a few drawbacks that need to be considered when considering the choice of a platform:

- High processing overhead caused by the multiple I/O disk operations required by iterative learning algorithms (Fernández et al., 2014);
- Lack of own server and user interface that requires the use of an external programming IDE with Java support;
- Lack of documentation (Maharjan et al., 2014);
- System requirements are highly dependent on what algorithm needs to be executed;

It has gained a lot of popularity among Data Mining developers because of the freedom of implementation.

6.2 Massive Online Analysis (MOA)

Massive Online Analysis (MOA, 2015) is an open source software oriented toward mining of data streams that present conceptual drift. It allows both building and experimentation on machine learning and data mining algorithms to provide fast answers to the evolution of the nature of the data streams. It is a very user-friendly platform – having its own Graphical User Interface (GUI) as seen in Figure 8 – command line and also making use of the Java API, which makes it suitable for users with distinct levels of experience. One of most recognized abilities is its modularity. Besides the core program one user can choose to expand only to modules of interest thus saving time and effort in exploring features not relevant to her/his work. The main goal of MOA is to provide a framework for benchmarking of existing machine learning algorithms that operate on real-time big data streams. This allows the community to easily identify algorithms that are less efficient and abandon their development at an early stage. This allows for the development resources to be optimized as they are always placed in the solutions that have prospect of being most efficient and useful. Unlike the WEKA (WEKA, www.cs.waikato.ac.nz/ml/weka/) platform from where MOA has derived it does not work with batch-processing. To perform such tasks it provides a set of essential tools such as real and synthetic examples of data streams for testing, a library of existing algorithms and measures of comparison between them. It does not only allow working with content given in the platform but it provides framework for the user to insert new types of streams, algorithms and methods. It also permits the storage of previously run benchmark results thus providing the creation of scenarios to be used against the newer algorithms (Bifet et al., 2010). The current version of MOA provides collections able to perform such tasks as Classification, Regression, Clustering, Outlier Detection, Recommender Systems, Frequent Pattern Mining and Change Detection. The drawbacks of MOA are as follow:

- Memory allocation limit of 400Mb is too small to handle Big Data;
- No support for parallel processing.

6.3 R Project

R Project (www.r-project.org) is the combination of a programming language and an environment for statistical and graphics computing. It has been designed with influence taken from the programming languages S and Scheme but unlike these two it is completely open-source (Hornik, 2015). So far R Project provides a variety of graphical and statistical techniques that include linear and non-linear modelling, classic statistic tests, time-series analysis or more traditional features like classification and clustering. It intends to be a fully planned ahead system built with coherency rather than a basic suite where different tools are simply added for extension of functionalities (Morandat et al., 2012). Because R is in itself a programming language it allows users who feel comfortable with coding to add new functionalities to the suite. It is an integrated suite of software that allows performing a full circle of data treatment including manipulation, calculation and finally display. Besides including effective
methods for data storing and handling it includes a group of operators to provide calculation with arrays and matrices. Other features include an integrated collection of tools for intermediate data analysis and graphical facilities to facilitate visualization. R Project has so much popularity among the community working with statistical data analysis that it has led to the creation of several tools to make it more user-friendly and appealing. One of them is RStudio (www.rstudio.com/products/rstudio). RStudio Desktop is an IDE specifically aimed at working with R and developed by the company of the same name. An example of the RStudio interface is shown in Figure 9.

The R Project presents two major drawbacks that condition mostly the required skills for programmers that will use the platform:

- There is a learning curve associated with the R language. R is not a common programming language that is not part of the education process of most programmers;
- Requires knowledge about different programming languages such as C, C++ and Fortran to use the full potential of the platform.

6.4 VOWPAL WABBIT

Vowpal Wabbit (www.github.com/JohnLangford/vowpal_wabbit/wiki) is a project sponsored by Microsoft Research with the goal of developing a single machine learning algorithm that is inherently fast, able of being run in both standalone machines and in parallel processing environments and capable of handling datasets in the scale of terabytes. The creators and developers of Wabbit have decided from the very beginning focus on building a single strong multi-purpose algorithm rather than a library with many algorithms. The development of the algorithm is encouraged to take advantage of four main features that when used in combination can achieve better results – input formats of data, speed of learning, scalability of the data sets and feature pairing (feeding of data to subtasks in pairs rather than one by one). Wabbit presents a group of features that are so far unique to its system.

It is optimized by default to support online learning rather than batch learning through a series of modifications made to the stochastic gradient descent methods that allow a more robust analysis on data sets of considerable size. It presents Feature Hashing, a method that reduces the necessary pre-processing of data making the analysis process not only faster but more accurate. The combination of the aforementioned two features allows Wabbit to make effective learning from any size of information made available to the algorithm no matter how small or big it is.

The implementation of a reduction stack within its core also allows it to provide a solution for large scale advanced problems. Wabbit has a few drawbacks that come from the aspects we analyzed before:

- The existence of a single algorithm makes Wabbit have a good performance for few tasks but a poor performance for most of them;
- Low optimization on basic aspects such as the speed of input/output operations.

Wabbit runs mainly as a library or a standalone daemon service but it is fully ready to be deployed in Cloud environments.

6.5 PEGASUS

PEGASUS (www.cs.cmu.edu/~pegasus) is an open source platform developed by the data mining group at the Carnegie Mellon University and designed specifically for data mining in graph structures of sizes ranging from a few gigabytes to petabytes. Because datasets of such size are no longer able of being processed in single node machines, PEGASUS works with resort to parallel programming by being implemented on top of Hadoop.

It is a more specific data mining platform than others because it works with data that comes in form or graphs and networks with billions of nodes and connections. This is a considerable leap against previous systems that can only work in the size of the millions. Graph structures are suffering a rise in number and importance coming from such various fields as mobile networks, social networks or medical fields such as protein regulation (Kang et al., 2009). PEGASUS unifies a vast number of graph mining...
tasks such as the computing of the graph diameter, computing the radius of each node or finding connections between the graph nodes by making a generalization of matrix-vector multiplication called GIM-V (Kang et al., 2009).

GIM-V is carefully implemented and optimized with built-in graph mining operations such as Page Rank, Random Walk with Restart and diameter estimation (Kang et al., 2009). It provides linear scaling as to the number of edges to analyze making it suitable to work on any number of machines available. PEGASUS is one of the platforms that present the most drawbacks, such as:

- It is not fully developed yet. The library of machine learning algorithms is very incomplete.
- Operations such as the indexing of graphs are still poorly optimized;
- Works only with data that comes in the form of graphs with no support at all for other types of data;
- Very heavy software requirements.

6.6 GraphLab Create

GraphLab Create (www.dato.com/products/create) formerly known only has GraphLab it was rebranded into Dato Incorporation (Wolpe, 2015). It is a platform for development of machine learning applications working at various scales of dataset sizes. It aims at providing means for applications to have all the necessary iterative steps to make predictions based in data mining results. The main goal is to design and implement machine learning algorithms that are efficient, accurate and able to keep data consistency while taking the most advantage from parallel processing (Low et al., 2014). Some of the already developed algorithms are belief propagation, Gibbs sampling or Co-EM.

All of these algorithms have been optimized from their previous versions to now perform better with parallel processing. The platform works mainly with three areas of data processing – Data Engineering, Data Intelligence and Data Deployment. One of its strengths is the ease of use for both beginner and experts in the area of data mining. Its main components are scalable data structures, machine learning modules, methods for data visualization and capacity to easily integrate with many data sources of different types. In the field of data engineering it provides an easy way to run the ETL (Extract, Transform and Load) process on data as a means to clean data and save time during the analysis process. For this it provides tools to perform basic tasks such as data sort, slice and dice in a fast way on datasets with terabytes of size. It also provides intuitive visualization of the generated tables and graphs through the auxiliary tool GraphLab Canvas whose example of interface can be seen in Figure 10. For data intelligence there are tools to build recommenders with Python code, make data analysis in images through deep learning techniques that take advantage of powerful computing GPU capabilities. There are also tools for analysis of unstructured text, graph analysis and supervised learning for outcome prediction. For data deployment it provides tools to deploy easily coded services for prediction of various types. The drawbacks of GraphLab Create are:

- Small algorithm library that is restricted to the most common tasks;
- Very complete platform that covers the whole cycle of data treatment and takes time to master before use.

6.7 MLLib

MLLib (www.spark.apache.org/mllib/) is a machine learning library that runs on top of the Apache Spark framework and is thus released in the Spark bundle whose current stable release is 1.4.1 dated of July 2015.

Taking advantage of Spark’s distributed in-memory based architecture it claims to be nine to ten times faster than Mahout running on the disk-based framework Hadoop and to have better scalability than Vowpal Wabbit (Talwalkar et al., 2015). MLLib implements most of the common algorithms for machine learning and statistical analysis. For machine learning it includes algorithms for classification and regression, collaborative filtering, clustering, dimensionality reduction, feature extraction and transformation and optimization primitives.

![Figure 10: GraphLab Canvas Interface (Gu, 2015).](image-url)
In statistical analysis it has methods for summary of statistics, correlations, stratified sampling, hypothesis testing and random data generation. With support for applications written in languages such as Java, Scala and Python, MLLib not only fits well in the Spark API but also integrates well with NumPy, the Python package for scientific computing.

When it comes to data sources, MLLib is fed by the data sources supported by Spark such as HDFS, Cassandra or cloud services like Amazon S3. MLLib presents two major drawbacks:

- It is a recent platform and therefore is not fully developed yet. Because of this it does not have the same robustness as all the previously mentioned platforms.
- Being released along with the Spark framework makes it somewhat limited as it cannot be used with other frameworks.

7. OPEN SOURCE BIG DATA PLATFORMS COMPARISON

To perform our comparison on the previously detailed platforms we chose nine parameters that literature such as – Liu et al., (2014), Fernández et al., (2014), Jović et al., (2014) and Hasim et al., (2015) - considers relevant for business managers to analyze when choosing an open source Big Data Platform.

These characteristics are – Programming Paradigm, Hardware Requirements, Operating Systems, Software Requirements, Programming Language(s), User Interface, Data Types, Available Algorithms and Scale of Supported Datasets.

These parameters are a mix of technical characteristics but also other defining characteristics to follow when finding out if a platform is or not suitable for Small and Medium Enterprise environments.

<table>
<thead>
<tr>
<th>Programming Paradigm</th>
<th>Mahout</th>
<th>MOA</th>
<th>Wabbit</th>
<th>R Project</th>
<th>PEGASUS</th>
<th>GraphLab</th>
<th>MLLib</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Requirements</td>
<td>Hadoop/Spark, JDK 1.6.x, Maven 3.x</td>
<td>JDK 1.6</td>
<td>None</td>
<td>None</td>
<td>Hadoop, Apache Ant 1.7.0, JDK 1.6.x, Python 2.4.x, GnuPlot 4.2.x</td>
<td>64bit Operating System</td>
<td>Spark</td>
</tr>
<tr>
<td>Programming Language(s)</td>
<td>Java</td>
<td>Java</td>
<td>C, C++</td>
<td>R, S, C, C++, Fortran</td>
<td>Java</td>
<td>C++, Python</td>
<td>Java, Scala, Python</td>
</tr>
<tr>
<td>User Interface</td>
<td>N/A</td>
<td>GUI, command line, Java API</td>
<td>N/A</td>
<td>N/A (but RStudio GUI exists)</td>
<td>N/A</td>
<td>GUI through GraphLab Canvas</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Types</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>Graphs</td>
<td>Graphs</td>
</tr>
<tr>
<td>Available Algorithms</td>
<td>Recommendation Mining, Classification, Regression, Clustering and others</td>
<td>Classification Regression, Clustering and others</td>
<td>Own Single Algorithm</td>
<td>Undefined</td>
<td>Page Rank, Random Walk With Restart and others</td>
<td>Belief propagation, Gibbs sampling or Co-EM and others</td>
<td>Classification, Regression, Clustering and others</td>
</tr>
<tr>
<td>Scale of Supported Datasets</td>
<td>Up to Petabytes</td>
<td>Few Megabytes</td>
<td>Up to Terabytes</td>
<td>Up to Gigabytes</td>
<td>Up to Petabytes</td>
<td>Up to Petabytes</td>
<td>Up to Petabytes</td>
</tr>
</tbody>
</table>

Table 3: Open Source Big Data Platforms Comparison
We start with the programming paradigm - this is directly related to the size of the company and size of the computer infrastructure where the system may eventually be installed.

The same applies for another set of characteristics we are analyzing – the hardware requirements, the operating systems, the software requirements and the supported sizes of datasets.

Next we analyze the required programming languages because we consider important for a manager to know what kind of programmers will be necessary to run the systems.

To this end we also provide comparison on the user interface each platform provides as we consider the level of experience of the programmers and ease of use and adoption of the platform can be related to the kind of less or more user friendly interfaces available.

We also compare the types of data supported as a means to categorize platforms as more generic or more specific. Last but not the least, we compare the variety of algorithms available as a way to say which platforms are more complete than the others. In Table 3 we show the detailed listing of these characteristics for each platform. Analyzing all the characteristics present in Table 3 we can draw the following conclusions – both MOA and R Project are the best tools for Serial Computing running in a single node machine. The others are best for companies with larger infrastructures that that allow parallel computing. All platforms are Cross Platform. This means they will run on all the three main Operating Systems – Windows (XP or higher), Linux and Mac OS X. On software requirements things are more diverse. Wabbit and R Project do not have additional software required. The ones with lower requirements are MOA that only needs the Java Development Kit (JDK), GraphLab that only requires a 64bit Operating System and lastly MLLib that only requires Spark (even though it can run over Hadoop). Heavier on software requirements are Mahout and Pegasus. Both require Hadoop (or Spark) and JDK. Mahout will also need Maven while Pegasus requires Apache Ant, Python and GnuPlot. All of this software is open source and thus free of any additional costs to the users. Most of the platforms, except for R Project, make use of common languages that have no lack of programmers available. MOA, R Project and GraphLab can be considered the most user friendly as they provide Graphical User Interfaces where the remaining ones do not, which makes working on them considerably harder and with a bigger learning curve, at least for less experienced users. PEGASUS and GraphLab are graph oriented where others platforms are made to work with all generic types of datasets. It is important to note that MLLib does not directly process graph data. This is done by a complementary module of the Spark bundle called GraphX. Working with graphs only is not a limitation in itself as this type of data structure is very common these days. It is up to the companies to make this evaluation accordingly to their own needs. Lastly, when comparing available algorithms it is complicated to make clear conclusions about the platforms. Mahout and R Project are the ones with more participation from the community and therefore the ones with more diverse algorithms provided. GraphLab, PEGASUS, MOA and MLLib also have a significant number of algorithms ready to be used. Here is where Wabbit is the weakest because as we already explained it only works with a single algorithm that can be powerful under specific conditions but useless for most cases. Mahout, PEGASUS, GraphLab and MLLib are the ones who support bigger data sets while MOA is the weaker one on this field supporting only datasets with a size of a few megabytes. In summary, MOA is definitely the best platform for small companies with minimal computer infrastructures and that work with lesser amounts of data. For larger companies with bigger infrastructures that require parallel computing it is hard to decide between Mahout, MLLib and GraphLab. While the two first have the advantage of being broader in terms of data types supported, the second provides more ease of use due to the existence of a GUI that helps track the work being done. None of the platforms but GraphLab have defined hardware requirements. GraphLab requires a minimum of 4 GB RAM and 2 GB free disk space. Some of the platforms will require the same hardware required by the framework they are installed with (for example, MLLib will have the same requirements as the Spark framework). For the others the requirements will be dependable on the needs of the task they want to run.

8. Conclusions and Future Work

With the rise of Big Data and with more individuals and organizations gaining awareness of the potential and opportunities it brings, the number of Big Data frameworks and Platforms of both open source and proprietary nature are supposed to increase exponentially over the next years. Implementing a framework and/or a platform that solves all the questions inherent to Big Data is too heavy, expensive if not impossible at all. So the path being more commonly followed is for each organization to invest in developing its own set of technologies that are close to its specific needs, inwards or by calling to the community. This creates
a high number of available solutions which not only brings difficult choices for people wanting to select a platform for their enterprise but also create a high level of redundancy and overlapping in the solutions presented. Educating not only business managers but also people in general to the potential of Big Data is crucial to guarantee the future of research and platform deployment. In this paper we study five distributed processing frameworks and conclude that Spark and Flink are the best ones because they provide the benefits of both batch and streaming processing models. They are followed closely by Hadoop and H2O that although limited to batch processing, are more matured, better tested and more widely deployed those providing more robustness. Also we study and analyze six Big Data Open Source Platforms and conclude that the best platform for small companies with minimal computer infrastructures is MOA, while Mahout, MLLib and GraphLab are the best for companies with larger computer infrastructures that require the use of parallel computing. Mahout and MLLib have more types of data supported but GraphLab is more user friendly because it has a GUI and Mahout does not. We can also give advice as to what the best combinations of framework/platform are to build your own data centric system. We advise for the use of the native Machine Learning tools of each platform as shown on Table 1. These are the combinations that are easier to deploy and will also be the ones that will perform better because the platform is developed to best take advantage of its framework partner features.

As future work we pretend to analyze the frameworks and the tools in real environments and explore other available Big Data tools.

9. REFERENCES


Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A.,


Fan, W., Bifet, A. (2013). Mining big data: current status, and forecast to the future. ACM SIGKDD Explorations Newsletter, 14, 2, 1-5.


Gartner.


Authors

Pedro Almeida is a final year student of computer engineering at ISEC (Instituto Superior de Engenharia de Coimbra) of Polytechnic of Coimbra, Portugal. His main research fields are big data, data mining and open source tools in which he has authored various papers in international conferences and journals.

Jorge Bernardino received the degree in computer engineering in 1987, the masters degree in systems and information technologies in 1994, and the PhD degree in computer science from the University of Coimbra in 2002. He is a Coordinator Professor at ISEC (Instituto Superior de Engenharia de Coimbra) of the Polytechnic of Coimbra, Portugal. His main research fields are big data, data warehousing, business intelligence, and open source tools, subjects in which he has authored or co-authored dozens of papers in refereed conferences and journals. He has served on program committees of many conferences and acted as a referee for many international conferences and journals in data warehousing and databases. He was President of ISEC from 2005 to 2010. During 2014 he was Visiting Professor at CMU – Carnegie Mellon University. Currently, he is the President of Scientific Council of ISEC.
DISTRIBUTED SPARQL QUERYING OVER BIG RDF DATA USING PRESTO-RDF

Mulugeta Mammo, Mahmudul Hassan, Srividya K. Bansal
Arizona State University
{mmammo, phassan, srividya.bansal}@asu.edu

Abstract

The processing of large volumes of RDF data requires an efficient storage and query processing engine that can scale well with the volume of data. The initial attempts to address this issue focused on optimizing native RDF stores as well as conventional relational database management systems. But as the volume of RDF data grew to exponential proportions, the limitations of these systems became apparent and researchers began to focus on using big data analysis tools, most notably Hadoop, to process RDF data. Various studies and benchmarks that evaluate these tools for RDF data processing have been published. In the past two and half years, however, heavy users of big data systems, like Facebook, noted limitations with the query performance of these big data systems and began to develop new distributed query engines for big data that do not rely on map-reduce. Facebook’s Presto is one such example. This paper proposes an architecture based on Presto, Presto-RDF, that can be used to process big RDF data. We evaluate the performance of Presto in processing big RDF data against Apache Hive. A comparative analysis was also conducted against 4store, a native RDF store. To evaluate the performance Presto for big RDF data processing, a map-reduce program and a compiler, based on Flex and Bison, were implemented. The map-reduce program loads RDF data into HDFS while the compiler translates SPARQL queries into a subset of SQL that Presto (and Hive) can understand. The evaluation was done with RDF datasets of size 10, 20, and 30 million triples. The results of the experiments show that Presto-RDF has a much higher performance than Hive and can be used to process big RDF data.

Keywords: Big RDF Data; Hadoop; Hive; Presto; SPARQL Querying; Semantic Web

1. INTRODUCTION

Semantic Web is the web of data that provides a common framework and technologies for sharing data and reusing data in various applications and enterprises. Resource Description Framework (RDF) enables the representation of data as a set of linked statements, each of which consists of a subject, predicate, and object called a triple. RDF datasets, consisting of millions of triples, form a network of directed graph (DG) and are stored in systems called triple-stores. A query language standard, SPARQL, has also been developed to query RDF datasets. For the Semantic Web to work, both triple-stores and SPARQL query processing engines have to scale well with the size of data. This is especially true when the size of RDF data is too big such that it is difficult, if not impossible, for conventional triple-stores to work with (Cudré-Mauroux et al., 2013; Luo, Picalausa, Fletcher, Hidders, & Vansummeren, 2012; Wilkinson, Sayers, Kuno, Reynolds, & others, 2003) In the past few years, however, new advances have been made in the processing of large volumes of data sets, aka big data, which can be used for processing big RDF data (Abadi, Marcus, Madden, & Hollenbach, 2007; Morsey, Lehmann, Auer, & Ngomo, 2011; Sakr & Al-Naymat, 2010).

In the past two and half-years, new trends in big data technology have emerged that use distributed in-memory query processing engines based on SQL syntax. Some of these tools include: Facebook Presto1, Apache Spark2, and Cloudera Impala3 (“The Platform for Big Data and the Leading Solution for Apache Hadoop in the Enterprise - Cloudera,” n.d.). These tools promise to deliver high performance query execution than the traditional Hadoop system like Hive4. The motivation of this paper is to validate this claim for big RDF data – i.e. if these new in-memory query processing models work well to deliver faster

2 Apache Spark: https://spark.apache.org/
3 Cloudera: http://www.cloudera.com/content/cloudera/en/home.html
4 Apache Hive: https://hive.apache.org/.
response times for SPARQL queries (which must be translated to SQL). Specifically, it addresses the following questions:

• Is it feasible to store big RDF data in HDFS and get improved query execution time, compared to Hive and native RDF stores like 4store, by translating SPARQL queries into SQL and then using the Presto distributed SQL query processing engine to run the translated queries?

• How much improvement, in query response time, can be obtained by using in-memory query processing engine, e.g. Presto, against native RDF stores, like 4store, and other query processing engines based on MapReduce, like Hive?

• How do different RDF storage schemes in HDFS affect the performance of SPARQL queries?

• Is it possible to construct an end-to-end distributed architecture to store and query RDF datasets?

This paper makes the following novel contributions:

(i) Architecture of Presto-RDF framework that uses a distributed in-memory query execution model, based on Presto, to evaluate the performance of SPARQL queries over big RDF data.

(ii) RDF-Loader component of Presto-RDF that uses map-reduce to load RDF data into different storage structures based on three storage schemes - triple-store, vertical and horizontal scheme.

(iii) SPARQL to SQL compiler based on Flex and Bison. The compiler is also unique in that it generates SQL for the three RDF storage schemes.

(iv) Evaluation of query performance for the three RDF storage schemes. No published results were found on performance comparison of various storage schemes to the best of our knowledge.

The rest of the paper is organized as follows: Background and related work is presented in section 2. The architecture of Presto-RDF framework and RDF storage strategies are presented in section 3. Section 4 describes the SPARQL to SQL compiler. Section 5 describes the experimental setup for performance evaluation of Presto-RDF and results. Section 6 presents a discussion of future work. Finally the conclusions are presented.

2. BACKGROUND AND RELATED WORK

This section presents the background on various RDF storage schemes and a review of related work that propose and evaluate different distributed SPARQL query engines. It also presents a review of two systems, Apache Spark and Cloudera Impala, which are similar to Facebook Presto.

2.1 RDF STORES

RDF stores, also known as triple stores, are data management systems that are used to store and query RDF data. RDF stores also provide an interface, called a SPARQL end–point, which can be used to submit SPARQL queries. Some triple stores, like Sesame, also provide APIs that can programmers can use to submit SPARQL queries and get results. RDF storage managers can be broadly classified into three categories:

• Native triple stores – are stores that are built from scratch to store RDF triples. Native triple stores make a direct use of the RDF data model – a labeled directed graph – to store and access RDF data. These triple stores usually store RDF data in a custom binary format. Examples are 4store 5, Jena TDB (Wilkinson et al., 2003, p. 2), and Sesame (Broekstra, Kampman, & Van Harmelen, 2002).

• Relational–backed triple stores – are triple stores that are built on top of traditional RDBMs. Because RDBMs have a number of well-advanced features that have developed over the years, triple stores based RDBMs benefit from these features. Examples are 3store (Harris & Gibbins, 2003), Jena SDB (Wilkinson et al., 2003).

• NoSQL triple stores – are triple stores based on NoSQL systems and by far are the largest triple stores in number. This group includes triples stores based on NoSQL systems as well as systems based on the Hadoop ecosystem. Examples include: Hive+HBase, CumulusRDF, Couchbase, and many more. Because NoSQL systems include a wide range of systems, including graph databases, some researches have classified AllegroGraph 6, a graph database, both as a native as well a NoSQL store. Despite the large number of RDF triple stores that have been developed and proposed by researchers, very few attempts have been made to systematically parameterize triple stores based on specific implementation parameters. Kiyoshi Nitta and Iztok Savnik (Nitta & Savnik, 2014) proposed a parameterized view of RDF stores that is based on “single” and “multi–process” attribute sets. In this view, an RDF Store Manager (RSM) can be parameterized as function of single–process, S, and multi–process attributes, M:

\[ RSM = f(S, M) = \langle T_s, I_s, Q_s, C_s, D_s, F_s, D_m, Q_m, S_m, A_m \rangle \]

Single–process attributes, S, this paper identifies are:

• \( T_s \): type of triple table structure the store uses - vertical, property-table, or horizontal.

• \( I_s \): index structure type – 6 independent, GSPO-OGPS, O matrix.

• \( Q_s \): indicates whether a SPARQL endpoint is implemented.

• \( S_s \): indicates the translation method type of IRI and literal strings – URI, literal, long, or none.

---

5 4store-Scalable RDF storage: http://www.4store.org/.
6 AllegroGraph: http://franz.com/agraph/allegrograph
processing architecture that can be used to efficiently
and query them. Leida & Chu, 2013 propose a query
MapReduce framework to store large RDF graphs
engine, unlike Facebook Presto, uses MapReduce
but its overall performance was very poor. The query
query engine scaled well with the size of RDF data
that were conducted showed that the distributed
optimize the design. The results of the experiments
Thuraisingham, 2011 approach of using Hadoop

- Triple table – entire RDF data is stored as a single
table with three columns – subject, predicate and
object. Each triple is stored as a row in this table.
- Property-table – triples are grouped together by
predicate name. In this scheme, all triples with
same predicate are stored in a separate table (also
known as property tables vertical partitioning).
- Clustered-property tables – in this scheme triples
are grouped into classes based on correlation and
occurrence of predicates. A triple is stored in the
table based on the predicate class it belongs to.

2.2 DISTRIBUTED SPARQL
A distributed SPARQL query engine based on
Jena ARQ\textsuperscript{7} has been proposed (Kulkami, 2010). The
query engine extends Jena ARQ and makes it
distributed across a cluster of machines. Document
indexing and pre-computation joins were also used
to optimize the design. The results of the experiments
that were conducted showed that the distributed
query engine scaled well with the size of RDF data
but its overall performance was very poor. The query
engine, unlike Facebook Presto, uses MapReduce
similar to Husain, McGlothlin, Masud, Khan, &
Thuraisingham, 2011 approach of using Hadoop
MapReduce framework to store large RDF graphs
and query them. Leida & Chu, 2013 propose a query
processing architecture that can be used to efficiently
process RDF graphs that are distributed over a local
data grid. They propose a sophisticated non-memory
query planning and execution algorithm based on
streaming RDF triples. Presto uses a distributed
memory query-processing algorithm. Wang,
Tiropanis, & Davis, 2012 discuss how the
performance of distributed SPARQL query
processing can be optimized by applying methods
from graph theory. The results of their experiment
show that a distributed SPARQL processing engine
based on Minimum Spanning Tree algorithms
perform much better than other non-graph traversal
algorithms. The framework presented in this paper
translates a SPARQL query into its equivalent SQL,
and hence the query optimization that is done by
Presto is for the SQL query and not for the SPARQL
query. A distributed RDF query processing engine
based on a message passing has been proposed (Dutta,
Theobald, & Schenkel, 2012). The engine uses in-
memory data structures to store indices for data
blocks and dictionaries. Just like Presto, query-
processing engine avoids disk I/O operations.

2.3 APACHE SPARK AND CLOUDERA IMPALA
Apache Spark and Cloudera Impala are two open-
sources systems that are very similar to Facebook
Presto. Both Apache Spark and Cloudera Impala offer
in-memory processing of queries over a cluster of
machines. Spark uses advanced Directed Acyclic
Graph (DAG) execution engine with cyclic data flow
and in-memory processing to run programs up to 100
(for in-memory processing mode) or 10 times faster
(for disk processing mode) than Hadoop MapReduce.
Cloudera Impala is an open-source massively parallel
processing (MPP) engine for data stored in HDFS.
Cloudera Impala is based on Cloudera’s Distribution
for Hadoop (CDH) and benefits from Hadoop’s key
features – scalability, flexibility, and fault tolerance.
Cloudera Impala, just like Presto, uses Hive Metastore
to store the metadata information of directories and
files in HDFS.

In this research study, we use Presto that is a
distributed SQL query engine that runs on a cluster of
machines controlled by a single coordinator with
hundreds or thousands of worker nodes. Our literature
review shows that there are no other studies done on
Presto for semantic data processing. Presto is
optimized for ad-hoc analysis and supports standard
ANSI SQL, including complex queries, aggregation,
joins, and window function (“Presto: Interacting with
petabytes of data at Facebook,” n.d.). The client sends
SQL query using the Presto command line interface to
the coordinator that would then parse, analyze and
plan the query execution. The scheduler, a component
within the coordinator, connects together the
execution pipeline and assigns and monitors work to
worker nodes that are closer to the data. The client

\textsuperscript{7} Apache Jena SPARQL Processor: jena.apache.org/documentation/query
gets data from the output stage, one of the worker nodes, which in turn pulls data from the underlying stages. In this project we propose an architecture for Presto to process big RDF data.

3. Presto-RDF Architecture

This section proposes architecture, called Presto–RDF, which can be used to store and query big RDF data using the Hadoop Distributed File System (HDFS) and Facebook Presto. It also presents RDF–Loader, one of the key components of the architecture, which is used to read, parse and store RDF triples.

3.1 ARCHITECTURE

Presto–RDF consists of the following components:
a command line interface (CLI), a SPARQL to SQL compiler (RQ2SQL), Facebook Presto, Hive Metastore, HDFS, and RDF–Loader. Figure 1 illustrates the different components of the architecture. RDF data that is extracted from the Semantic Web is parsed and loaded into HDFS using a custom–made RDF-loader, which will also store metadata information on Hive Thrift Server. When a user submits a SPARQL query over a command line interface, the query is processed by a custom–made SPARQL to SQL converter, RQ2SQL, that translates the SPARQL query into SQL which would then be submitted to Facebook Presto. Presto, using its Hive connector and Hive Thrift Server, runs the SQL against HDFS and returns the result back to the CLI.

3.2 RDF-LOADER

The purpose of the RDF–Loader is to load, parse, and store RDF data in HDFS. RDF–Loader implements four different RDF storage schemes and creates external Hive tables whose metadata is stored in the Hive Thrift server. Before the RDF–Loader is executed the raw RDF data to be first processed is loaded into HDFS using this command:

```
hadoop fs –put file hdfs–dir
```

Once the raw RDF data is uploaded, RDF–Loader runs several MapReduce jobs and stores the output back into HDFS. The structure of data is defined by the schema that can be specified by users of the system. In order for the RDF–Loader to run and process raw RDF, the following input parameters are required:

- `database` – name of database that will be created.
- `target` – type of RDF storage structure, i.e. the type of schema. There are four options: triples, vertical, wide, and horizontal.
- `expand` – indicates if qnames are to be expanded.
- `server` – DNS name or IP address of the master node, NameNode, of the Hadoop cluster.
- `port` – port number Hadoop listens to connections.
- `input` – path of HDFS directory that holds raw RDF data.
- `output` – path of HDFS directory the processed RDF data will be stored in.
- `format` – defines the format of the output files as they are stored in HDFS. The current version of the Hive meta–store supports five different formats: SEQUENCEFILE, TEXTFILE, RCFILE, ORC, and AVRO. This study makes use of the TEXTFILE format.

The following sections discuss four different RDF storage strategies implemented by the RDF–Loader. In the triple-store storage scheme, an RDF triple is stored as is – resulting in a table with three columns: subject, predicate and object. For an RDF data set with n number of triples, the map algorithm has O(n) running time while the reducer, which is called once for each unique subject, has O(s*o) running time, where s is the number of unique subjects and o are the average number of object values per subject.

For an RDF dataset with n number of triples, the map algorithm has O(n) running time while the reducer, which is called once for each unique subject, has O(s*o) running time, where s is the number of unique subjects and o are the average number of object values per subject.
The horizontal storage scheme is similar to the wide table storage scheme in terms of the schema of the table. However, unlike the wide–table scheme, it optimizes the number of rows stored for subjects that have multiple object values for the same predicate. In this scheme, it is not necessary to create new rows for each unique subject–predicate pair. Instead, rows that are already created for the same subject, but for a different predicate will be used.

In the vertical storage scheme implemented in this research, the raw RDF data is partitioned into different tables based on the predicate values of the triples in the data with each table having two columns – the subject and object values of the triple. Thus, if the raw RDF data has 30 million triples that have 20 unique predicates, the vertical storage scheme will create 20 tables and stores the subject and object values of triples that share the same predicate in the same table. The map–reduce algorithm works with predicate as a key value and a pair of subject and object values as value:

```java
map (String key, String value)
// key: RDF file name
// value: file contents
for each triple in value
emit_intermediate (<subject, predicate>, <predicate, object>);

reduce (String key, Iterator values)
// key: a <subject, predicate> pair
// values: list of <predicate, object> pairs
String subject = key.getSubject();
String[] row = new String[1 + num_unique_predicates];
int i = 0
for each v in values
row[i] = v.getObject();
i++;
emit (subject, row);
```

The vertical storage scheme, for a raw RDF data that contains n number of triples, the mapper runs at O(n) while the reducer runs at O(p*x) where p and s are the number of unique predicates and subjects in the data set, respectively. In the worst case scenario, where there are as many unique predicates and subjects, the number of triples, the map-reduce algorithm for the vertical storage scheme runs at O(n²).

4. RQ2SQL – SPARQL To SQL Compiler

This section presents a SPARQL to SQL compiler that is developed as part of Presto–RDF. RQ2SQL (RDF Query to SQL) converts SPARQL queries into SQL statements that can be run on Presto and Hive. RQ2SQL is implemented using Flex – a lexical analyzer creator, Bison – a parser generator, and C++11.

4.1 SPARQL Graph Patterns

SPARQL query processing is based on graph pattern matching. Complex graph patterns can be constructed by combining few basic graph pattern techniques. The W3C classifies SPARQL graph pattern matching into five smaller patterns (Leida & Chu, 2013): Basic Graph, Group Graph, Optional Graph, Alternate Graph, and Named Graph Pattern.

- **Basic graph patterns**: are set of triple patterns where the pattern matching is defined in terms of joining the results from individual triples. A single graph pattern is composed of a sequence of triples that may optionally be interrupted by filter expressions. Example below is basic graph pattern.

```sql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?age
WHERE {
  ?x foaf:name ?name .
  ?x foaf:age ?age . }
```

RQ2SQL translates above SPARQL into this SQL for vertical storage scheme:

```sql
SELECT T0.object,
    T1.object
FROM http___xmlns_com_foaf_0_1_name T0
JOIN http___xmlns_com_foaf_0_1_age T1
ON (T1.subject = T0.subject)
```

- **Group graph pattern**: is specified by delimiting it with braces. Example below specifies one graph pattern with two basic graph patterns.

```sql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?age
WHERE {
  { ?x foaf:name ?name . }
  { ?x foaf:age ?age . }
}
```

The RQ2SQL translation, for the vertical storage scheme, is the same as the previous SQL:

```sql
SELECT T0.object,
```
• **Optional graph patterns:** are specified using the OPTIONAL keyword. The semantics of the optional graph pattern matching is that it either adds additional binding to the solution or would leave it unchanged.

Given the following RDF data:

```rml
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
_:a rdf:type foaf:Person .
_:a foaf:name "Michael" .
_:a foaf:email <mailto:michael@example.com> .
_:b rdf:type foaf:Person .
_:b foaf:name "Mulugeta" .
```

and following SPARQL optional graph pattern query:

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
SELECT ?name ?email WHERE {
  ?x foaf:name ?name .
  OPTIONAL {
    ?x foaf:email ?email .
  }
}
```

Would return the following result:

<table>
<thead>
<tr>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Michael&quot;</td>
<td><a href="mailto:michael@example.com">mailto:michael@example.com</a></td>
</tr>
<tr>
<td>&quot;Michael&quot;</td>
<td><a href="mailto:michael@hahusoftware.com">mailto:michael@hahusoftware.com</a></td>
</tr>
<tr>
<td>&quot;Mulugeta&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Constraints can also be applied to optional graph patterns.

• **Alternate graph Patterns:** are constructed by specifying the keyword UNION between two graph patterns.

• **Named graph patterns:** are constructed by specifying a FROM NAMED IRI where each IRI is used to provide one named graph in the RDF dataset. Using same IRI in two or more NAMED clauses would result in one named graph.

RQ2SQL does not support Named graph patterns.

### 4.2 SPARQL Solution Sequences and Modifiers

The results returned from a SPARQL query are unordered collection of single or composite values that, according the W3C, can be regarded as solution sequences with no specific order. SPARQL defines six solution modifiers: order, projection, distinct, reduced, offset and limit.

• **Order modifier:** is specified by the ORDER BY clause and forms the order of a solution sequence. Ordering can be qualified as ASC for ascending or DESC for descending.

• **Projection modifier:** is specified by listing a subset of variables defined in the pattern-matching clause.

• **Distinct modifier:** is specified by the DISTINCT keyword and filters out duplicates from the solution sequence.

• **Reduced modifier:** unlike the distinct modifiers that ensures that duplicate solutions are eliminated from the solution sequence, the reduced modifier, specified by the REDUCED keyword, permits them to be eliminated. The result set of a solution sequence with a reduced modifier is at least one and at most the cardinality of the solution sequence without the distinct and reduce modifiers.

• **Offset modifier:** just like SQL, the offset modifier, specified by the OFFSET keyword, returns results of the solution sequence starting at the specified offset value. Offset value of 0 has no effect. Both Presto and Hive do not support the OFFSET keyword.

• **Limit modifier:** just like SQL, the LIMIT modifier puts an upper bound to the number of solution sequences returned. A limit value of 0 would return no results. A negative limit value is not valid.

RQ2SQL does not support ORDER BY, DISTINCT, REDUCED, OFFSET and LIMIT keywords.

RQ2SQL translation:

```sql
SELECT T0.object
FROM http___xmlns_com_foaf_0_1_name T0
JOIN http___xmlns_com_foaf_0_1_age T1
ON (T1.subject = T0.subject)
```
ASK query modifier – SPARQL queries specified using the ASK form test whether or not a SPARQL query has a solution. Given the following triples:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
_:a foaf:name "Michael".
_:a foaf:homepage <http://work.example.org/michael/>.
_:b foaf:name "Mulugeta".
_:b foaf:mbox <mailto:mulugeta@hahasoftware.com>.
```

Running the SPARQL query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
ASK { ?x foaf:name "Michael" }
```

Returns the value: yes. RQ2SQL does not support ASK query modifier.

4.3 RQ2SQL

RQ2SQL is a mini SPARQL to SQL compiler built using Flex – a lexical analyzer creator – and Bison – a parser generator creator. RQ2SQL supports basic SPARQL queries including OPTIONALS, FILTERS as well as ORDER BY, DISTINCT, projection and LIMIT modifiers. However, it does not support UNION, ASK, named graph patterns as well as group graph patterns. RQ2SQL generates SQL queries for the four different RDF storage schemas explained in section 3 – triple, wide, horizontal, and vertical.

Evaluation of RQ2SQL - Translating LUBM queries to SQL: Lehigh University Benchmark (LUBM) is based on ontology for university domain and can generate synthetic data of arbitrary size and provides fourteen queries that represent a variety of RDF graph properties and several performance metrics (Guo, Pan, & Heflin, 2005). RQ2SQL was tested for correctness by compiling the 14 LUBM benchmark queries against Presto and then comparing the result with the output generated after running same queries on 4store. This section presents selected LUBM queries and their RQ2SQL translation for vertical storage scheme.

Q1:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ_bench.owl#>
SELECT ?x
WHERE {
  ?x rdf:type ub:GraduateStudent.
  ?x ub:takesCourse <http://www.Department0.University0.edu/GraduateCourse0>.
}
```

Graph representation:

```
SELECT T1.Subject
FROM http___www_w3_org_1999_02_22_rdf_syntax_ns_type T0
JOIN http___www_lehigh_edu__zhp2_2004_0401_univ_bench_owl_takesCourse T1 ON (T1.Subject = T0.Subject)
```

Q4:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ_bench.owl#>
SELECT ?x ?y1 ?y2 ?y3
WHERE {
  ?x rdf:type ub:Professor.
  ?x ub:worksFor <http://www.Department0.University0.edu>.
  ?x ub:emailAddress ?y2.
}
```

Graph representation:

```
FROM http___www_w3_org_1999_02_22_rdf_syntax_ns_type T0
JOIN http___www_lehigh_edu__zhp2_2004_0401_univ_bench_owl_worksFor T1 ON (T1.Subject = T0.Subject)
JOIN http___www_lehigh_edu__zhp2_2004_0401_univ_bench_owl_name T2 ON (T2.Subject = T1.Subject)
JOIN http___www_lehigh_edu__zhp2_2004_0401_univ_bench_owl_emailAddress T3 ON (T3.Subject = T2.Subject)
JOIN http___www_lehigh_edu__zhp2_2004_0401_univ_bench_owl_telephone T4 ON (T4.Subject = T3.Subject)
```

Q12:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ_bench.owl#>
SELECT ?X ?Y
WHERE {
  ?X rdf:type ub:Chair.
  ?Y rdf:type ub:Department.
  ?X ub:worksFor ?Y.
}
```

Graph representation:

```
SELECT T2.Subject, T3.Subject
```

RQ2SQL translation – for the vertical storage scheme:
first setup was a 4-node cluster virtualized on a single 16GB memory machine. The second setup was 8-node cluster virtualized on the Windows Azure platform. The second setup was required because the experiments conducted used up the hard disk space and it was not possible to run queries on triples of more than 4 million. For the experiment, four benchmark queries (#1, 6, 8, 11) from SP2Bench (Schmidt, Hornung, Lausen, & Pinkel, 2009; “The SP2Bench SPARQL Performance Benchmark,” n.d.) were used and three different RDF storage schemes were evaluated – triple, vertical, horizontal stores.

5.1 Benchmark Queries

SP2Bench is a SPARQL benchmark that is designed to test SPARQL queries over RDF triple stores as well as SPARQL-to-SQL re-write systems. SP2Bench focuses on how well an RDF store supports the different SPARQL operators and their combination – known as operator constellations. The SP2Bench data model is based on DBLP, http://www.informatik.uni-trier.de/~ley/db/, a computer science bibliography created in the 1980s and currently featuring more than 2.3 million articles. The SP2Bench data generator can generate any number of triples based on what a user specifies.

Query 1: return the year of publication of journal 1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?yr
WHERE {
  ?journal dcterms:issued ?yr
}
```

Query 6: return, for each year, the set of all publications authored by persons that have not published in years before.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX dcterms: <http://purl.org/dc/terms/>

SELECT ?yr ?name ?document
WHERE {
  OPTIONAL {
    ?author foaf:name ?name.
  }
  OPTIONAL {
  }
}
```
5.2 Experiments with 3M Triples

The first setup was a virtualized four-node cluster on a machine with 16GB of memory and 300GB of hard disk. Each of the four virtual machines had 2GB of memory and 32GB of hard disk. To evaluate the performance of Presto-RDF, an RDF dataset with three million (3M) triples was generated and the four benchmark queries from section 5.1 were ran on 2-Nodes and 4-Nodes for each of the three storage schemes - triples, vertical, and horizontal storage schemes. Figure 7 shows a comparison of the loading times.

After the RDF-Loader loaded the 3M triples into HDFS and the corresponding external tables have been created on Hive metastore, the SQL equivalent of the four benchmark queries from section 5.1 were run on a 2-Node and 4-Node cluster.

Performance comparison of triple storage scheme on 3M Triples (2-Node Cluster) Performance comparison of triple storage scheme on 2-node cluster is shown in figure 8. The query response time for 4store is very fast compared to Presto and Hive. 4store took 0, 8, 9, and 7 seconds to respond to queries Q1, Q6, Q8, and Q11 respectively. Presto performed much better than Hive. Hive was very slow on Query 8, which involved UNION.

8 Simulating a 2-Node cluster simply involves shutting down two nodes out of the four nodes.

9 Note that 4store has no partitioning scheme and was run on a single-node cluster. The results are included in the chart to give how well Presto and Hive perform in comparison to it.
The performance results over a 4-node cluster are shown below in figure 9.

Figure 9 shows the query response time of 3M triples using the triple storage scheme. The results are very similar to the triple storage scheme - Presto performs much faster than Hive. Moreover, for the same query, the vertical storage scheme has better performance than the triples storage scheme. The results for the vertical storage scheme show that, once again, Presto performed much faster than Hive. Moreover, the node increase from 2 to 4 also increased the performance of Presto but not Hive. Figure 12 shows the performance comparison of Presto and Hive for the horizontal storage scheme. Once again, Presto has a better performance than Hive. For queries that does not involve UNIONS (Q1, Q6, and Q11), the performance of Presto over Hive was not as significant as it was for the triples and vertical storage schemes.

Figure 13A and 13B below shows the performance comparison of the three storage schemes for Presto and Hive over a 4-node cluster. The results indicate that the vertical storage scheme outperforms both the triples and horizontal storage schemes on both Presto and Hive. For queries Q1, Q8 and Q11, the horizontal
storage scheme has a slightly better performance than the triples storage scheme.

5.3 Experiments with 10, 20, 30M Triples

The experimental setup involved setting up four and eight node clusters on Microsoft Windows Azure Platform. Each node in the cluster had a 2-core x86-64 processor, 16GB of memory, and 1TB of hard disk. Measurements were conducted for the four-benchmark queries for 10, 20, and 30 million triples.

FIGURE 14: TIME TAKEN BY RDF-LOADER TO PARSE AND STRUCTURE RAW RDF DATA

Once the RDF dataset is copied into HDFS, the RDF-Loader will parse and run a map-reduce job to convert the raw dataset to a structured dataset based on three storage schemas – triple-store, vertical and horizontal. The results of the measurement are shown in Figure 14. The performance of the RDF-loader has a linear relationship with the size of the triples. The horizontal store map-reduce algorithm always took much longer time than the triple-store and vertical store schemes.

EVALUATION RESULTS FOR Q1

The equivalent SQL translations of SPARQL Q1 for the three storage schemes are given below.

Q1 Horizontal-store SQL:

SELECT T.dcterms_issued AS yr
FROM HorizontalTable T
WHERE T.rdf_type = '<http://localhost/vocabulary/bench/Journal>'
AND T.dc_title = '"Journal 1 (1940)"^^<http://www.w3.org/2001/XMLSchema#string>;'

Q1 Vertical-store SQL:

SELECT T3.Object AS yr
FROM http___www_w3_org_1999_02_22_rdf_syntax_ns_type T1
JOIN http___purl_org_dc_elements_1_1_title T2
ON T1.subject = T2.subject
JOIN horizontalTable T3
ON T1.subject = T3.subject
WHERE T1.Object = '<http://localhost/vocabulary/bench/Journal>'
AND T2.Object = '"Journal 1 (1940)"^^<http://www.w3.org/2001/XMLSchema#string>;'

Q1 Triple-store SQL:

SELECT T3.Object AS yr
FROM Triples T1
JOIN Triples T2
ON T1.subject = T2.subject
JOIN Triples T3
ON T1.subject = T3.subject
WHERE T1.Predicate = '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
AND T2.Predicate = '<http://purl.org/dc/elements/1.1/title>'
AND T3.Predicate = '<http://purl.org/dc/terms/issued>'
AND T1.Object = '<http://localhost/vocabulary/bench/Journal>'
AND T2.Object = '"Journal 1 (1940)"^^<http://www.w3.org/2001/XMLSchema#string>;'

The result of running the above queries on Presto for a 4-node and 8-node cluster setup are shown in the figures 15 and 16 below.

FIGURE 15: QUERY PROCESSING TIME OF Q1 OVER 4-NODE CLUSTER

For Q1, the vertical and horizontal stores have a much better performance than the triple-store scheme. This can be explained by looking into the SQL translations of the vertical and horizontal storage schemes – which have lesser rows involved in JOINs. This fact remains true when the number of nodes is increased from 4 to 8 – figure 16.

FIGURE 16: QUERY PROCESSING TIME OF Q1 OVER 8-NODE CLUSTER

FIGURE 17: Q1 PERFORMANCE FOR 30M TRIPLES ON 8-NODE CLUSTER
Presto Vs Hive for Q1: Compared to Hive, Presto once again has a much higher performance. Figure 17 shows a comparison of Presto and Hive for 30M triples.

![Figure 17: Comparison of Presto and Hive for 30M triples](image)

**FIGURE 17:** Presto vs. Hive for 30M Triples

Evaluation Results for Q6

The SQL translations for query Q6, unlike Q1, involve multiple JOINs for each of the three storage schemes. The results of the evaluation on a 4-node and 8-node cluster are shown in Figure 18 and 19 below. The results of the evaluation above indicate that the performance increased with an increase in the number of nodes (figure 20). The vertical store has a much better performance than the triple-store and horizontal store. Unlike Q1, however, where the horizontal store had a slightly better performance than the triple-store, the triple-store in Q6 had a slightly better performance than the horizontal store, especially as the size of the triples increases. This result can be explained by the fact that the horizontal store SQL for Q6, unlike the triple-store, involves multiple selections before making JOINs. For Hive, unlike Presto-RDF, as the number of nodes was increased there was a drop in performance – which can be attributed to increase in replication across nodes and disk I/O operations (figure 21).

![Figure 21: Effect of node increase on Hive, for Q6 with 30M Triples](image)

**FIGURE 21:** Effect of node increase on Hive, for Q6 with 30M Triples

Evaluation Results for Q8

SQL translations for Q8 involve multiple JOINs (just as the cases were in Q6) and a UNION. The results have the same behavior as Q6 – the vertical store has a much better performance than the triple-store and horizontal stores, and Presto-RDF has a much higher performance than Hive. Figure 23 shows the results of running the above queries over 10, 20 and 30M triples.

![Figure 23: Q8 Performance on Presto-RDF with 4 Nodes](image)

**FIGURE 23:** Q8 Performance on Presto-RDF with 4 Nodes

Evaluation Results for Q11

The SQL translations for Q11 involve a simple select with an ORDER BY and LIMIT clauses.

![Figure 24: Q11 Performance](image)

**FIGURE 24:** Q11 Performance
Q11 Vertical-store SQL:

```sql
SELECT T.Object AS ee
FROM http://www.w3.org/2000/01/rdf-schema_seeAlso T
ORDER BY ee
LIMIT 10;
```

Q11 Horizontal-store SQL:

```sql
SELECT T1.rdfs_seeAlso AS ee
FROM HorizontalTable T1 WHERE T1.rdfs_seeAlso != 'null'
ORDER BY ee
LIMIT 10;
```

**FIGURE 24: Q11 PERFORMANCE ON PRESTO-RDF WITH 4 NODES**

Figure 24 above shows the results of running the above queries over 10, 20 and 30M triples. Because Q11 involves just one table that has less number of rows for the vertical and horizontal storage schemes than the triple-store (which is one table), the results shown are expected. For 8 nodes, there is a performance improvement – see Figure 25.

**FIGURE 25: Q11 PERFORMANCE ON PRESTO-RDF WITH 8 NODES**

**Presto Vs Hive for Q11:** Compared to Hive, Presto again has a much higher performance (Figure 26).

**6. DISCUSSION**

Various optimization techniques can be applied to the three storage schemas as well as to the RDF data directly. The RDF data is stored as a text file in our study, which may not be optimal. This work can be extended to test performance using RCFile, ORC, AVRO formats, which are better optimized than text file. The following sections briefly discuss the salient features of RCFile and ORC file formats.

**6.1 RC File**

In RCFile format, a table can have multiple HDFS blocks (He et al., 2011). In each HDFS block, the RCFile organizes records with the basic unit of a row group. It splits data horizontally into row groups based on the user specified value for a row group. Then the row group data is saved in columnar format. So instead of storing row one followed by row two, it stores column one across all rows then column two across all rows and so on. A row group contains a sync marker, a metadata header and a table data. A sync marker that is placed in the beginning of the raw group is mainly used to separate two continuous row groups in an HDFS block. There is a metadata header for the row group that stores information on the number of records in this row group, bytes in each column, and bytes in each field in a column. A table data contains data in column-store format and all the fields in the same column are stored continuously together. The metadata header section and the table data section are compressed independently. For the metadata header section, the RCFile uses the RLE (Run Length Encoding) algorithm to compress data, and in table data section, each column is compressed independently with the high-end compression algorithm. The RCFile guarantees that data in the same row are located in the same node, and can exploit column-wise data compression and skip unnecessary column reads. Figure 1 shows the layout of the RCFile.

**DATA READS AND COMPRESSION**

Within an HDFS block, each row group is processed sequentially. When processing a row group, the RCFile does not need to fully read the whole content of the row group into memory. Rather, it only reads
the metadata header and the needed columns in the row group for a given query (He et al., 2011). For instance, if there is a table with four columns tbl(c1, c2, c3, c4), and a query “SELECT c1 FROM tbl WHERE c4 = 1,” then, in each row group, the RCFile only reads the content of column c1 and c4.

**Lazy Decompression**

The metadata header and data of certain columns are in compressed format when they are loaded into memory. Therefore, they need to be decompressed. The metadata header is always decompressed and held in memory until RCFile processes the next row group (He et al., 2011). RCFile uses Lazy decompression technique to decompress the needed columns in memory and avoids decompressing the data in column, which is not useful for query execution. Lazy decompression is extremely useful due to the existence of various where conditions in a query. If a where condition cannot be satisfied by all the records in a row group, then RCFile does not decompress the columns that do not occur in the where condition. For example, in the above query, column c4 in any row group must be decompressed. However, for a row group, if no field of column c4 in the group has the value 1, then it is unnecessary to decompress column c4 in the group.

**6.2 ORC File**

The table placement method of the ORC File shares the basic structure with that of RCFile (Huai, Ma, Lee, O’Malley, & Zhang, 2013). It provides a highly efficient way to store relational data. It stores collections of rows in one file, and within the collection, the row data is stored in a columnar format. This allows parallel processing of row collections across a cluster. Each file with the columnar layout is optimized for compression, and skipping of data/columns reduces read and decompression load. Its file structure consists of three parts: Stripe, Footer, and Postscript. It breaks the source file into a set of rows called a stripe. The default stripe size is 256 MB. This large stripe size enables efficient read of columns from HDFS. The file footer contains a list of stripes in the file, the number of rows per stripe, and each column’s data type. It also contains column-level aggregate count, min, max, and sum. Postscript contains compression parameter and size of compressed footer. Each stripe in an ORC File has three parts: Index data, Row data, and Stripe footer. Index data include min and max values for each column and the row positions within each column. Row index entries provide offsets that enable seeking the right compression block and byte within a decompressed block. The Row data are composed of multiple streams per column, and they are used in table scans. The stripe footer contains a directory of stream locations. Figure 28 illustrates the layout of the ORC File structure.

**FIGURE 28 ORC FILE LAYOUT**

**Data Reads and Compression**

The columns in an ORC File separate the stripes or sections of the file. An internal index is used to track a section of the data within each column. This organization allows readers to efficiently omit the columns that are not required. Only required column values on each query are scanned and transferred on query execution. Additionally, column data are of uniform type and thus may achieve better compression, especially if the cardinality of the column is low. Each column can apply different compression methods, depending on the data type. Users can further ask the writer of an ORC File to compress streams of data with a general-purpose codec among ZLIB, Snappy and LZO. Metadata about the ORC data, such as the schema and compression format, are serialized into the file and are made available to the readers. The operator translates the ORC File schema into appropriate data flow types when possible. A few ORC data types are not supported for reading. The columns with unsupported data types are omitted. The output record type has fields with the same names and in the same order as the source schema. The type in the output fields is assigned based on the schema of the source field with the same name.

**Lazy Decompression and Lazy Decoding**

The Lazy decompression feature enhances the read performance of the ORC File by leveraging the index strides that already existed in the format. This Lazy decomposition technique in the ORC File is similar to that in the RCFile, but the only difference is that the reader of the ORC File seeks the appropriate index stride in the stripe and only decompresses and decodes the values preceding and including the target row in that index stride. The ORC writer is data-type aware. It can decompose a complex column to multiple child columns; therefore, the ORC reader can read the needed child columns efficiently. Also, the ORC File supports sparse indexes that are data
statistics and position pointers. The data statistics are used in query optimization, and they are also used to answer simple aggregation queries. The ORC reader uses these statistics to avoid unnecessary data read from HDFS. The position pointers are used to locate the index groups and stripes.

6.3 RC File vs ORC File

The RCFile stores everything as binary blobs and it treats each column as a set of bytes. It does not know the column’s type whether it is an integer or string. So, there is no way to build the index. The RCFile was mainly designed for sequential data scan. Because it does not have any index; therefore, it cannot take advantage of semantic information provided by queries to skip unnecessary data. Also, the RCFile cannot decompose a complex data type. Therefore, when a query needs to access a field of complex data types, all fields of this type have to be read, which makes data reading inefficient. On the other hand, the large stripe size (256 MB) allows the ORC File to achieve more efficient data reading capability. The ORC File indexes, which are data statistics and position pointers, allow skipping row groups that do not pass predicate filtering. In the ORC File, the writer decomposes the column of a complex data type (e.g. Map) into multiple child columns and the reader is able to only read the needed child columns. Table 1 illustrates the comparison between RCFile and ORC File.

As part of future work we will conduct study to evaluate query performance with these file formats.

<table>
<thead>
<tr>
<th>TABLE 1: COMPARISON BETWEEN RCFILE AND ORC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Feature</strong></td>
</tr>
<tr>
<td>Default column group size</td>
</tr>
<tr>
<td>Separate complex columns</td>
</tr>
<tr>
<td>Splits found quickly</td>
</tr>
<tr>
<td>Files per bucket</td>
</tr>
<tr>
<td>Store min, max, sum, count</td>
</tr>
<tr>
<td>Versioned metadata</td>
</tr>
<tr>
<td>Run length data encoding</td>
</tr>
<tr>
<td>Store strings in dictionary</td>
</tr>
<tr>
<td>Store row count</td>
</tr>
<tr>
<td>Skip compressed blocks</td>
</tr>
<tr>
<td>Store internal indexes</td>
</tr>
</tbody>
</table>

7. Conclusions And Future Work

This paper presented a comparative analysis of big RDF data using Presto, which uses in-memory query processing engine, and Hive, which uses MapReduce to evaluate SQL queries. We also proposed a Presto-based architecture, Presto-RDF, that can be used to store and process big RDF data and a SPARQL to SQL compiler. From the experiments conducted, following conclusions can be drawn:

- 4store has a much higher performance than Presto and Hive for small data sets. For bigger data sets (10M, 20M and 30M triples), however, 4store was simply unable to process the data and crashed. This is true when Presto, Hive and 4store are all tested with single-node setups.
- For all queries, Presto-RDF has a much higher performance than Hive.
- The vertical storage scheme has a consistent performance advantage than both the triple-store or horizontal storage schemes.
- As the size of data increases, the horizontal storage scheme performed relatively better than the triple-store scheme. This is unlike the articles reviewed during this research study, which ignore the horizontal scheme as being not efficient (because it has many null values).
- Increasing the number of nodes improved query performance in Presto but not in Hive. This can be explained by the fact that Hive replicates data across clusters and does I/O operations – which increase as the number of nodes increase.
- As the size of RDF data increases to big-data levels, RDF stores based on Hadoop outperform native RDF store, 4store, for a single-node setup.
- Distributed in-memory query processing engines deliver faster response time on big RDF datasets than query engines that rely on MapReduce.

Unique contributions of this paper include:

- Use of distributed in-memory query execution model, based on Presto, to evaluate performance of SPARQL queries over big RDF data.
- A RDF-Loader component of Presto-RDF uses map-reduce to load RDF data into the different storage structures.
- Demonstrates use of a SPARQL to SQL compiler based on Flex and Bison. The compiler is also unique in that it generates SQL for the three storage schemes discussed in this paper – triple-store, vertical and horizontal.
- Publishes result of query performance of horizontal storage scheme, which had a better performance than the triple-store as the size of data increases. No published results were found on the horizontal storage scheme during the literature review.

There are a number of areas to extend this study: this paper used a single benchmark, SP2Bench. This work can be investigated on different benchmarks such as LUBM (Guo et al., 2005), BSBM (“Berlin SPARQL Benchmark,” n.d.), and DBPedia (Morsey et al., 2011). Experiments need to be conducted on other file formats to evaluate for performance improvement. Presto is an open-source project that can be extended to have a direct support for SPARQL instead of using the SPARQL to SQL compiler.

8. References


Authors

Mulugeta Mammo has M.S. in Software Engineering from Arizona State University. He is Software Engineer at Intel Corporation. He works on optimizing big data solutions for latest Intel Platforms. His current focus is on Java, Scala, ApacheSpark, & Genomic Analysis.

Mahmudul Hassan is a Ph.D. student in Computer Science in Ira A. Fulton Schools of Engg. at Arizona State University. His research interests are in area of Big Data Analytics, High Performance Computing, Statistical Machine Learning and Data Mining.

Srividiya Bansal is an Assistant Professor in Ira A. Fulton Schools of Engineering at Arizona State University. Her primary research focuses on semantics-based approaches for Big Data Integration, Web service description, discovery & composition, and tools for outcome-based instruction design in STEM education. She received a PhD in Computer Science from the University of Texas at Dallas. She is a member of the IEEE Computer Society. Contact her at srvidiya.bansal@asu.edu.
THE DEVELOPMENT AND DEPLOYMENT OF LARGE-FILE UPLOAD SERVICES

Huan Chen¹³, Liang-Jie Zhang¹³, Bo Hu²³, Si-Zhe Long²³, Li-Hui Luo²³, Chunxiao Xing⁴

¹Kingdee Research, Kingdee International Software Group Company Limited
²Kingdee Cloud & Big Data Dept., Kingdee International Software Group Company Limited
³National Engineering Research Center for Supporting Software of Enterprise Internet Services, China
⁴Research Institute of Information Technology, Tsinghua University, China

huan_chen@kingdee.com

Abstract

The popularity of enterprise cloud storage is rapidly growing. A number of Internet service vendors and providers, such as Google, Baidu and Microsoft, entered this emerging market and released a variety of cloud storage services. These services allow people to access work documents and files all over the world at any time. Interestingly, with the prevalence of mobile Internet, rich media becomes regular and popular. More and more people use cloud storage for keeping their personal photos, music and movies. Nevertheless, the size of the media files is often beyond the upper limit that normal form-based file upload service allows hence dedicated large-file upload services are required to be developed and deployed. Although many cloud vendors offer versatile cloud storage services, very little is known about the detailed development and deployment of the large-file upload services. This paper proposes a complete solution of large-file upload service, with the contributions in manifold: Firstly, we do not limit the maximum size of a large file that can be uploaded. This is extremely practical for storing huge database resource files generated from ERP tools. Secondly, we developed large-file upload service APIs that have very strict verification of correctness, to reduce the risk of data inconsistency, which has better safety. Thirdly, we extend the service developed recently for team collaboration with the capability of handling large files. Fourthly, this paper is arguably the first one that formalizes the testing and deployment procedures of large-file upload services with the help of Docker. In general, most large-file upload services are exposed to the public, facing security and performance issues, which brings much concern. With the proposed Docker-based deployment strategy, we can replicate the large-file upload service agilely and locally, to satisfy massive private or local deployment of KDrive. Finally, we evaluate and analyze the proposed strategies and technologies in accordance to the experimental results.

This paper is an extension version of the SCC 2015 conference paper: On Developing and Deploying Large-File Upload Services of Personal Cloud Storage.

Keywords: Large-file Upload, Cloud Storage, Team Collaboration, Docker

1. INTRODUCTION

Enterprise cloud storage is rapidly gaining its popularity. A number of Internet service providers rushed into this emerging market and brought a variety of cloud storage services. For example, Google Drive (Google, 2015), Drop-box (Dropbox, 2015), OneDrive (Microsoft, 2015) (previously named SkyDrive), SugarSync (SugarSync, 2015), Baidu Yun Pan (Baidu,
2015) and Tencent Wei Yun (Tencent, 2015) all developed their own cloud storage. With such a rapid growth, in 2015, more than 800 millions of people would have been using cloud storage services (Lardinois, 2012).

Cloud storage services consist of two key components: A front-end client application that runs on personal mobile devices or computers. A back-end storage hosting files within datacenter. Users of cloud storage can instantly upload a small file, for instance a photo, through a front-end user interface such as a Dropbox (Dropbox, 2015) APP on an iPhone, to one of specified cloud storage Dropbox (Dropbox, 2015), KDrive (Kingdee, 2015), Baidu Yun Pan (Baidu, 2015), Google Drive (Google, 2015), OneDrive (Microsoft, 2015). With the prevalence of rich media, the demand of sending large files grows dramatically. Simple form-based (Nebel E. et al., 1995) file upload scheme definitely cannot handle such a long-linking request. Transmitting a multi-gigabyte file to server is infeasible, since the file itself may not fit in memory or the HTTP connection may time out or disconnect. Furthermore, if something happens half way through uploading a large file, there’s nothing to do but to start all over again. Instead, files have to uploaded via dedicated APIs. Most of the cloud vendors offer these APIs allowing submitting a large file to back-end server. Despite the popularity of large-file upload services, very little is known to the development of the services, in particular, their internal back-end APIs. Understanding the mechanism inside APIs as well as the corresponding deployment strategies helps developers of third- part applications to connect these personal cloud storages easily.

The goal of this paper is threefold. Firstly, we would like to explain the details in the development of KDrive (Kingdee, 2015) large-file upload APIs, to help programmers understanding the internals and then to develop their own applications easily on top of KDrive (KDrive, 2015). Secondly, we propose a formalized deployment scheme based on Docker (Docker, 2015), to introduce a new way to efficient managing and agile deploying of large-file upload services. Leveraging cloud storage services for enterprise applications is another interesting field. ERP (Enterprise Resource Plan) and CRM (Customer Relationship Management) tools have several large files needed to be backup. Finally, we go through a complete case study of database file exchanging to help readers deeply understanding the internals of APIs and their concrete usages.

This paper is organized as follows: Section 2 reviews the related work and highlights the contributions of this paper. Section 3 introduces the complete solution of large-file upload service in detail. Section 4 proposes a formalized deployment and testing strategy. Section 5 analyzes the large-file upload services with experimental results and illustrates concrete use case scenarios. Section 6 concludes the paper and points out some potential future research directions.

2. BACKGROUND

File upload is the most fundamental functionality for a cloud storage. The very early standard of file uploading can be traced back to the propose of RFC-1867 (Nebel, E., 1995), in prior to which it is hard to upload a file through a HTML form.

Nowadays, with the prevalence of rich media and stream media over web, the size of files grows exponentially. Interestingly, the size larger than 2-GigaByte is now very typical for a normal media file. However, transmitting such a big file through traditional web-form (Nebel, E., 1995) is infeasible and therefore a lot of dedicated large-file upload services are developed by those Internet vendors.

Google Drive (Google, 2015) provides APIs for file the CURD (Create, Update, Retrieve, Delete) manipulation. This API set covers most of the features...
that Google services can have, in particular, it helps to operate and share Google Document files fluently. In general, Google Drive comprehensively provides three types of file upload APIs:

- Simple upload (Nebel, E., 1995) — for a quick transfer of smaller files, e.g. size less than 5 MB.
- Multipart upload — for a quick transfer of smaller files and metadata separately; transfers the file along with metadata that describes it, all in a single request (Google, 2015).
- Resumable upload — for a quick transfer of large files, also applicable to small files.

This paper focuses on resumable upload, which can reliably transfer large files by chunked file transmission. Google Drive offers versatile services, but it is still far from satisfaction due to its weak and inconvenient team collaboration mechanism (Ning, K. et al., 2014). Moreover, most of Kingdee’s (Kingdee, 2015) ERP tools require extremely high speed connection, ideally with a connection to a nearby datacenter, but in fact, the connection to Google Drive service reported from our customers is not as good as expected (see also Drago’s paper (Drago, I., et al., 2013) for a detailed comparison).

Dropbox (Dropbox, 2015) offers more comprehensive APIs to third-party applications. A larger number of applications utilize Dropbox as their back-end storage, being more popular than Google Drive (see paper (Drago, I., et al., 2015) for a comprehensive review). Another reason that more people prefer Dropbox is arguably the privacy concern: Google Drive is attached to Google Search that has strong ability to scan, to index and to search files, which are seemingly unsafe to the users. Baidu Yun Pan (Baidu, 2015), one of the largest cloud storage vendor in China, also provides APIs for resumable uploads, but the size of single file is only limited to 4 GB (free) or 20 GB (paid). Similarly, OneDrive (Microsoft, 2015) limits the file up to 10 GB. Tencent Weiyun (Tencent, 2015) has very limited APIs, and resumable file upload is only opened to its own application and Weiyun webpage. There exist several other cloud storage services, but it is likely that for most of them users have to pay, for example Qiniu (Qiniu, 2015) and SugarSync (SugarSync, 2015). iCloud Drive (Apple, 2015) appears recently and helps to sync files both on Mac OS X and iOS smoothly. Although there is no limitation on the size of upload, it is somewhat platform dependent.

We can summarize the history of the international cloud storage market as the following Table 1.

<table>
<thead>
<tr>
<th>Brand</th>
<th>Establishment Year</th>
<th>Free Volume (at year 2012)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OneDrive</td>
<td>2007</td>
<td>25GB</td>
</tr>
<tr>
<td>SugerSync</td>
<td>2008</td>
<td>5GB</td>
</tr>
<tr>
<td>Dropbox</td>
<td>2008</td>
<td>2GB</td>
</tr>
<tr>
<td>Amazon Cloud Drive</td>
<td>2011</td>
<td>5GB</td>
</tr>
<tr>
<td>(Amazon, 2015)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Google Drive</td>
<td>2011</td>
<td>5GB</td>
</tr>
<tr>
<td>iCloud Drive</td>
<td>2011</td>
<td>5GB</td>
</tr>
</tbody>
</table>

Table 1. The History of International Cloud Storage Market

With the development of international cloud storage, the domestic market (China) is gaining the popularity. A large number of vendors published their cloud storage services. If we look at the history and market of Chinese cloud storage providers, we can summarize them as the following:
Table 2. The history of Chinese Cloud Storage Providers

<table>
<thead>
<tr>
<th>Provider</th>
<th>Year</th>
<th>File Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kanbox (Kanbox, 2014)</td>
<td>2011</td>
<td>5G</td>
</tr>
<tr>
<td>115 (115, 2014)</td>
<td>2009</td>
<td>15G</td>
</tr>
<tr>
<td>Kingsoft Kuaipan</td>
<td>2011</td>
<td>5G</td>
</tr>
<tr>
<td>Tencent Weiyun (Tencent, 2015)</td>
<td>2011</td>
<td>1G</td>
</tr>
<tr>
<td>Huawei (Huawei, 2015)</td>
<td>2011</td>
<td>5G</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>(at year 2012)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>5G</td>
</tr>
<tr>
<td>2009</td>
<td>15G</td>
</tr>
<tr>
<td>2011</td>
<td>5G</td>
</tr>
<tr>
<td>2011</td>
<td>1G</td>
</tr>
<tr>
<td>2011</td>
<td>5G</td>
</tr>
</tbody>
</table>

Being different from the above large-file upload services, this paper proposes a new large-file upload service which is characterized by the following contributions:

- Unlimited file size and no constraints on file format.
- Good file sharing and collaboration scheme, by extending the service used in (Ning, K., et al., 2014) with large-file upload capability.
- Strict verification of parameters is performed throughout the service. Multi-gigabyte files could be uploaded over a spotty connection without worry because the very strict verification mechanism is involved.
- Formalized testing and deployment strategy, by versioning sources files and Dockerfiles (Docker, 2015).
- Agile deployment of immutable services, by building services on Docker (Docker, 2015).

The KDrive (KDrive, 2015) is also characterized by the consistent file pool, where each file has only one global ID. For example, in Figure 1, no matter where the file is, who the file owned, how frequent the file used, how many physical copies the system has, the file has just one unique logic ID. The corresponding commercial product KA (Ning, K., et al., 2014) has integrated KDrive, shown as Figure 3, but missing large-file upload services. We use Figure 2 to show an example, the unique id is consistently ‘12123.5’, independent from the location ‘D1’, ‘D2’ and ‘D3’.

3. LARGE-FILE UPLOAD SERVICES

This section illustrates the large-file upload services by unfolding the details of the corresponding APIs. The overall procedure of uploading a large file is
depicted as Figure 4. The work flow consists of six steps: applying, progress checking, uploading, finalizing, consistence checking and persistence checking. This flow highlights the significant parameters exchanged between Application (APP) and Server, where the common basic parameters, e.g. token, error code and code description used in typical APIs are assumed to be familiar by users therefore not presented in the figure.

2.1 APPLY TO UPLOAD

Users are required to first apply for accessing to large-file upload service. Figure 5 shows the process of applying inside the API applyUpload. Initially, the correctness of the user-specified name is checked. No illegal name is allowed to tag a file, otherwise, it would make retrieving of the file difficult and confused. KDrive (KDrive, 2015) allocates different quota of space in accordance to the level of users. As a result, secondly, we calculate the volume of user space and forbid user to upload a file larger than the allowed quota. Unlike some other cloud storage such as Baidu Yun Pan (Baidu, 2015), Tencent Weiyun (Tencent, 2015), which manage files by K/V databases, KDrive organizes files by tree-like structures, where files are located in folders directly. Thirdly, it is necessary to check the existence of user-specified folders.

It should emphasize that there is no limit on the size of a file to be uploaded. A file is split into several small chunks for uploading, to reduce memory consumption, and more importantly, to reduce errors introduced in transmission. The default chunk size is set to as large as 4 MB, which is the same as Dropbox (Dropbox, 2015), but allowed to be resized to any size up to 2GB. Using large chunk size would require fewer calls to the API uploadChunk and faster overall throughput. However, whenever a transfer action is
interrupted, you will have to resume at the beginning of the last chunk, so it is often safer to use smaller chunks (Dropbox, 2015). Users need to calculate the number of chunks by:

\[
\text{chunk\_number} = \text{ceil}\left(\frac{\text{file\_size}}{\text{chunk\_size}}\right)
\]

Similarly, server calculates the number of chunks to be used as a parameter in assembling of file chunks. To enable resumable uploading, we keep single transaction ID (can also be viewed as a session ID) by hashing the joint input parameters:

\[
\text{tran\_id} = \text{hash}(\text{user\_id};\text{file\_md5};\text{file\_size})
\]

In such a way, the uniqueness of transmission is identified by hashing. Finally, all these parameters are stored into a service side database.

2.2 Upload Progress Check

A large-file is uploaded to KDrive (KDrive, 2015) chunk by chunk, from the very first chunk (by default = 4 MB) to the last one (by default ≤ 4 MB). Uploading can be paused and resumed at any time. To resume an uploading or to start from scratch, users can call the API getNextChunkNumber to retrieve the number of chunks that have been successfully transmitted.

2.3 Chunked Upload

Figure 6. Process in API uploadChunk

Chunked upload is similar to normal file upload but geared with stricter verification of correctness. File chunks have to be uploaded and merged sequentially in our system. This scheme is close to Dropbox (Dropbox, 2015) but different from some other systems such as Baidu Yun Pan (Baidu, 2015) and Qiniu (Qiniu, 2015) which merge a large file at the end. Assembling files from chunks is somewhat a time-consuming and performance-intensive task, so we put it evenly right after each chunk uploading behavior. Once a chunk has been successfully uploaded and verified, it will be merged with previous chunks immediately.

To accomplish an error-free assembling, a very strict check on the sequence of chunks has to be applied. A consistency check is performed between the stored parameters and newly received ones, including transaction ID (i.e. session ID) and the index number of the file chunk. Any incorrectness involved should be asserted and abandoned, and the entire process should be interrupted. The large-file upload service allows user to re-upload the last chunk so as to rectify the incorrect upload. Each chunk is identified by a MD5 hash value, which is a part of meta-data descriptions of files. To enable user to check the correctness of each uploaded chunk, server sends the calculated hash value back to the Application (APP) side.
2.4 Finalize the Overall Process

![Diagram of API finishUpload process]

After all of the chunks have been successfully uploaded, the overall procedure comes to the end. Figure 7 shows the final step. The transaction ID is firstly verified and then the last file chunk will be merged to the large binary file previously assembled. Application calls the API finishUpload to do a synchronized/asynchronized MD5 hash value calculation followed by a file persistence. The progress of each step can be monitored by API getMD5State and getPersistenceState, respectively.

1) Consistency Check: Files are stored inside a file pool. For a small number of files, MD5 value suffices for identifying the uniqueness. However, when the number increases, it is possibly that MD5 is not enough but has to equipped with a CRC32 value. As a result, we introduce two verification keys for distinguish files — MD5 and CRC32.

2) Data Persistence: For a public deployed cloud storage, we put files distributively into HDFS. Uploaded large files are temporarily stored on application server farms. File persistence represents the process of transmitting a file from the application server farm to the HDFS. This process is very time-consuming since HDFS datanodes are located in different cities all over the world and connection among them may be spotty. To address this issue, we developed asynchronized file persistence, where files are put to all datanodes after telling user that the whole process has finished.

A typical flow that we use frequently is shown as Case 1. The fastest way is to use Case 2, where the slowest way is to use Case 3.

Case 1: synchronized MD5 calculation and asynchronized file persistence, which is also shown as Figure 8.
Figure 8. Suggest Finalizing Step

Case 2: asynchronized MD5 calculation and asynchronized file persistence.

Case 3: synchronized MD5 calculation and synchronized file persistence.

2.5 PORTING TO THE B/S ARCHITECTURE

This subsection briefly introduces the techniques used in the front-end design of large-file upload service. The majority of KDrive (KDrive, 2015) users prefer to C/S architecture when they use large-file upload service APIs, because most of KDrive API callers are traditional ERP tools (Kingdee, 2015). In the recent years, with the prevalence of HTML5, B/S architecture is getting more popular. Borrowing the strength of FileAPI in HTML5, large-file upload can be easily ported to a web frontend. The HTML5 function slice can split a large file into several pieces to be then uploaded. Moreover, slicing enables the calculation of MD5 (SparkMD5) of a large file because less memory is required during file loading for and subsequent hashing.

4. FORMALIZED DEPLOYMENT

Nowadays, deploying a service on AMP (Apache, MySQL, PHP) architecture is no longer difficult. However, with the increasing of users together with higher throughputs, it is often the case that several AMP clusters are required to be immutably deployed. Platform Docker (Docker, 2015) helps developers and system administrators to build, ship, and run such distributed applications in an efficient way. This section formalizes the development and deployment of large-file upload services based on the leverage of Docker.

Docker assembles applications from components manually, or automatically by reading a configuration file named Dockerfile. The assembled applications are a read-only template called Image. An Image can be instantiated to a Container which is writable.

Definition 1 (Dockerfiling): The procedure of creating an image from components by reading a Dockerfile is referred to as Dockerfiling.

As shown in Figure 9 and Table 3, in the development phase, distinct categories of files are versioned, including source codes, Docker images and Dockerfiles. SVN (Apache Subversion, 2015)/GIT (Git, 2015) manages normal source codes, as a typical version control system does:

\[ File_{v_{m+1}} = ver_1(File_{v_m}) \]
Table 3. Versioning and the corresponding deployment

<table>
<thead>
<tr>
<th>Function</th>
<th>Test Phase Unit</th>
<th>Purpose of deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ver_1$</td>
<td>Unit</td>
<td>Altering source codes of service APIs</td>
</tr>
<tr>
<td>$ver_2$</td>
<td>White-box</td>
<td>Statically altering context of services</td>
</tr>
<tr>
<td>$ver_3$</td>
<td>Black-box</td>
<td>Dynamically fine-tuning context of services</td>
</tr>
</tbody>
</table>

Dockerfiling itself is controlled by Dockerfiles. Interestingly, the changes in Dockerfiles could also versioned:

$$Dockerfile_{v_{m+1}} = ver_2(Dockerfile_{v_m})$$

Dockerfiling produces Images, where these generated Images can be versioned by Docker. Dynamically committing a Container to Docker registry creates a version of an Image:

$$Image_{v_{m+1}} = ver_3(Image_{v_m})$$

For example, in Figure 9, Docker Image1 V1+ is an incremental version of Image1 V1.

*Strategy 1*: A versioning step $ver()$ suggests a step of deployment.

During the development of large-file upload services, we strictly follow this strategy to deploy and test the functionalities. corresponds to fixing a bug or altering the codes inside service APIs. After APIs are stable enough, we set the context for running APIs by writing a Dockerfile. $ver_2$ tells the differences between Dockerfiles and essentially the details of contexts. Images from Dockerfiling may not be fully optimized and therefore could be further improved during real-time execution. As a result, during the black-box testing phase, we tune the system-level parameters carefully, and then snapshot those optimized versions by committing them to a Docker registry.
5. **Case Study**

5.1 **Suitable Deployment Strategy**

Public clouds with storage services (Dropbox, 2015), (KDrive, 2015), (Baidu, 2015), (Google, 2015), (Tencent, 2015) allow people to access private files anywhere at anytime. However, a very practical but restricted scenario is that files can only be shared in a permitted and limited area, for instance in an internal/local network of a company or a family. Our large-file upload services are publicly deployed with OpenAPIs (KDrive, 2015) but its source codes are developed internally within Kingdee (Kingdee, 2015) local network. Rather than public cloud, sometimes we would like to deploy KDrive (KDrive, 2015) privately for agile development of codes itself, for rapid uploading/downloading corporate files, and more importantly, for using those existing tools based on KDrive (KDrive, 2015) APIs with a higher performance. When file size grows, selecting deploying strategy is no longer trivial. A suitable deployment strategy is thus essential to meet the practical needs and to maximal resources utilization. Showed as the remaining of this section, we were trying to deploy the large-file upload services based on experimental results.

We set up four types of server configurations to observe the focal differences between deploying strategies. The experimental results were obtained from virtual machines, where each machine has the same hardware infrastructure: 2.8 GHz dual core CPU and 2 GB memory. The basic tools used were Apache2 for web service, MariaDB (MariaDB, 2015) for database, HDFS (Apache Hadoop, 2015) and Network File System for file storage, and Docker (Docker, 2015) version 1.4.1. Ubuntu Linux 14.04 LTS serves as the OS for each of the configuration. In Figure 10 and Figure 11,

- HDFS represents a typical public cloud architecture, where web service, database and file storage (HDFS) are distributively deployed on 3 Linux servers.
- NFS stands for a private cloud setting. Similar to HDFS, services are distributively deployed, but for the storage part NFS replaces the HDFS. NFS is built in a local network.
- 3 Services on 1 Server configuration integrates all three services web, database and file storage directly into one Linux system.
- 3 Dockers on 1 Server keeps the same topology as 3 Services on 1 Server, but those three services are built and executed as three Docker Containers separately.
As stated before, some Kingdee (Kingdee, 2015) ERP tools require extremely fast connection to KDrive APIs. Localizing storage comes as a straightforward solution, where NFS being built in local network might have much better performance than the HDFS over public network. Figure 11 shows the performance gap between Hadoop settings and local storage configurations. Obviously, the latter case performs much better since it is based on local network, neatly avoiding time-consuming communication processes, such as API requests going through public network.

From the resource availability perspective, sometimes the number of servers may not be enough, but only a very limit number is allowed for the purpose of developing and testing. Intuitively, a lazy solution is to use just one server, where the web service, database and file storage share the system resources. Nevertheless, tight integration of services directly into one Linux system introduces security risks, single points and also much heavier workloads for the server. Instead, isolating the services running on one server might help in reducing server failures as well as server workloads. Docker (Docker, 2015) can execute different services by separated Containers, being freely to stop and reload almost instantly, which is natural way to isolate the services. Moreover, Docker could utilize resources more efficiently by sharing libraries and binaries on top of the Docker Engine. Using Docker (Docker, 2015) could intrinsically leverage resources in a much more efficient way.

To observe the differences, we built those three configurations. In Figure 11, for relatively small file (< 2GB), three types of settings have similar uploading speed. However, it is interesting to find that the performance gap enlarges when file size grows. Interconnection, in particular the traffic brought by the API UploadChunk between three servers via LAN produces the overheads. Such an API occupies most of the uploading time. According to the above results, we conclude that building the KDrive large-file upload services on Docker has the best performance.

5.2 USING LARGE-FILE UPLOAD SERVICES FOR COLLABORATION

The most straightforward approach of using large-file upload services is to interact the proposed six APIs with a front-end web page, on which provides an one-click upload function such as (Baidu, 2015), (PLUpload, 2015), (ResumableJS, 2015). Alternatively, large-file upload services can often be used as back-end services for third-part applications. This paper extends the work (Ning, K., et al., 2014) with large-file upload services. The roles user, organization and group inherited from KDrive (KDrive, 2015), (Ning, K., et al., 2014) construct a well-organized team collaboration architecture. Files can be shared between these three roles, meanwhile, one file has only one unique instance in the file pool aiming to reduce the consumption of duplicated storage. Depicted as Figure 12, users are belonging to organizations and files are shared via organizations. Similarly, the term group could also be used for file sharing (Ning, K., et al., 2014). Due to space limitation, its relationship is not presented in this paper.

![Figure 12. File sharing through role organization in KDrive team collaboration architecture](image_url)

Well leveraging of roles makes the exchanges of large files smooth. In the following case, we demonstrate using large-file upload services with two roles for database (abbr. DB) repairing business. In Kingdee (Kingdee, 2015) after-sale service, customers sometimes expect to get DB repairing from DB experts. In this scenario, typically, DB experts and company users could both be defined as explicit users. However, this scheme would have to expose KDrive
user interfaces both to DB experts and company users, which introduces confusions and bad isolation when KDrive only serves as a back-end service. Instead, we define the company user as an implicit user, annotating it as organization, in which case the implicit user only has organization space, i.e. UserSpace ≡ OrganizationSpace. In this way, the implicit user has all the other features an explicit user of KDrive should have, for example file sharing via organizations.

The flow of DB file exchanging mainly involves three stages, shown as Figure 13.

- Company user uploads a broken DB file to third-part application (implicitly to KDrive) for repairing. Firstly, the new organization space to be owned by DB engineer is created and is then authorized to company user. In other words, company user and DB engineer now share a common space for file storage.
- DB expert downloads the broken DB file, repairs it and then uploads the fixed one back to third-part application (actually to KDrive).
- Company user downloads the fixed DB file through a download link placed in third-part application, where the link is actually a hyper-link with an access pass-word/key connectable to KDrive.

It is clearly that large-file upload service behaves as a key role throughout. Three key features shown as the following enable the above complicated process.

- Time-consuming steps are asynchronized that helps in improving user experiences, to both sides namely company user and DB expert, since there exist time slots between every two stages. For example, after DB expert uploading a new file in stage 2, waiting for data persisting to HDFS, the company user may not need to instantly download the file but DB expert could quickly finalize the upload process, though it has not actually finished.

6. Conclusions

This paper proposes a comprehensive solution of large-file upload service, including its development, deployment and evaluation stages. The development of APIs for large-file upload has been firstly introduced in detail followed by a formalization of deployment strategy with the help of Docker. To select the best way of deployment, this paper configured the large-file upload services into four different settings and then evaluated their real performance. We concluded that deploying private large-file upload service on Docker is the fastest option among many others. Finally, for a better leverage of large-file upload service, this paper has explained a concrete DB file exchanging example, which stands a typical scenario of Kingdee usage of the service.

Future research will involve the development of an automatic deployment methodology based on the leverage of Docker’s Swarm features affinity and resource management. In addition, the high availability of the large-file upload services will be deeply studied, where one option will be to explore the use of Docker’s developing features: fault-tolerant scheduling and highly available scheduler.

7. Acknowledgment
This work is supported by the central grant funded Cloud Computing demonstration project of China, R&D and industrialization of the SME management cloud (Project No. [2011]2448), hosted by Kingdee Software (China) Company Ltd., under the direct of National Development and Reform Committee of China; the Shenzhen high-tech projects, under grant No. CX A201105060081A, CXZZ20120618164052978, and KC2012JSJS0021A.

8. REFERENCES


Figure 13. Interactions in using large-file upload services for exchanging DB among roles user and organization.


Authors

Huan Chen is researcher at Kingdee Research Institute. His research interests are big data architecture and cloud storage.

Liang-Jie Zhang is a computer scientist, a former Research Staff Member at IBM Thomas J. Watson Research Center, Senior Vice President, Chief Scientist, Director of Research at Kingdee International Software Group Company Limited, and director of The OpenGroup. He is the founding Editor-in-Chief of IEEE Transactions on Services Computing. He was elected as a Fellow of the IEEE in 2011, and won the IEEE Technical Achievement Award “for pioneering contributions to Application Design Techniques in Services Computing”.

Bo Hu is a computer scientist, Director of Kingdee Cloud Supporting Product Department. His research interests are cloud computing architecture and big data.

Si-Zhe Long is an engineer of Kingdee Cloud. His research interest is cloud computing.

Li-Hui Luo is an architect of Kingdee Cloud. His research interest is cloud computing.

Chunxiao Xing is a professor and associate dean of Research Institute of Information Technology (RIIT), Tsinghua University, and director of Web and Software R&D Center. His research primarily focuses on the database and data warehouse, data and knowledge engineering, large-scale digital media management, software engineering, digital library technology, and key technologies for e-government.