UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

DOTTORATO DI RICERCA IN INFORMATICA

PH.D. THESIS

Mining Predictive Models for Big Data Placement

Marco Meoni

SUPERVISORS

Raffaele Perego and Nicola Tonellotto
ISTI-CNR, Pisa, Italy

"If we knew what it was we were doing, it would not be called research, would it?"
— Albert Einstein

Acknowledgements

It has been a great privilege to have an international jointly supervised PhD dissertation. This thesis work is in fact a joint project among the European Organization for Nuclear Research (CERN), the Italian National Institute for Nuclear Physics (INFN) and the Italian National Research Council (CNR).

My thanks to Prof. Raffaele Perego (CNR), Dr. Nicola Tonellotto (CNR) and Dr. Tommaso Boccali (INFN) who have made it possible. A special thank to Luca Menichetti for his valuable assistance with the CERN software infrastructures and to Valentin Kuznetsov and Daniele Bonacorsi for fruitful discussions. All aforementioned people have given me a lot of useful ideas, many of which are contained within this work.

I am deeply indebted to the CMS experiment community for the access to the computing resources and the monitoring logs. I am also very grateful to the internal thesis committee at the Computer Science department of University of Pisa, professors Piepaolo Degano, Marco Danelutto and Salvatore Ruggieri, for their corrections and insights during the status updates.

I am very grateful to many other colleagues, researchers and professors who I had the privilege to know over the years and who have encouraged me or given academic advises: G. Landi, F. Carminati, L. Betev, A. Schiper, A. Ailamaki, G. Batignani, P. Cerello. They made it possible my participation to several conferences and projects and fueled my passion for research in computer science.

A special thank goes to my wife, Valentina. She has shared with me the difficult moments and has always supported my doctoral efforts and CERN assignments in parallel to my daily working activity. A big hug to my adorable little kids Tommaso, Matilde and Matteo, who allowed their dad to be often away from home. Finally, I would like to thank my mother

and my parents in law who have always been by my side, and apologize to my father I didn't make it in time.

Abstract

The Compact Muon Solenoid (CMS) experiment at the European Organization for Nuclear Research (CERN) deploys its data collections, simulation and analysis activities on a distributed computing infrastructure involving more than 70 sites worldwide. Over the last few years, the historical usage data produced by this large infrastructure has been recorded on Big Data clusters featuring more than 5 Petabytes of raw storage with different open-source user-level tools available for analytical purposes. The clusters offer a broad variety of computing and storage logs that represent a valuable, yet scarcely investigated, source of information for system tuning and capacity planning. Amongst all, the problem of understanding and predicting *dataset popularity* is of primary interest for CERN. In fact, its solution can enable effective policies for the placement of mostly accessed datasets, thus resulting in remarkably shorter job latencies and increased system throughput and resource usage.

In this thesis, three key requirements for Petabyte-size dataset popularity models in a worldwide computing system such as the Worldwide LHC Computing Grid (WLCG) are investigated. Namely, the need of an efficient Hadoop data vault for an effective mining platform capable of collecting computing logs from different monitoring services and organizing them into periodic snapshots; the need of a scalable pipeline of machine learning tools for training, on these snapshots, predictive models able to forecast which datasets will become popular over time, thus discovering patterns and correlations useful to enhance the overall efficiency of the distributed infrastructure; the need of a novel caching policy based on the dataset popularity predictions than can outperform the current dataset replacement implementation.

The main contributions of this thesis include the following results:

1. we propose and implement a scalable machine learning pipeline, built on top of the CMS Hadoop data store, to predict the popularity of

new and existing datasets accessed by jobs processing any of the 25 event types stored in the distributed CMS infrastructure. We cast the problem of predicting dataset accesses to a binary classification problem where we are interested to forecast if a given dataset will be *popular* or not at time slot t in a given CMS site s, i.e., it will be accessed for more than x times during time slot t at site s. Our experiments show that the proposed predictive models reach very satisfying accuracy, indicating the ability to correctly separate popular datasets from unpopular ones.

2. We propose a novel intelligent data caching policy, named PPC (Popularity Prediction Caching). This caching strategy exploits the popularity predictions achieved with our best performing classifier to optimize the eviction policy implemented at each site of the CMS infrastructure. We assess the effectiveness of this caching policy by measuring the hit rates achieved by PPC and caching baselines such as LRU (Least Recently Used) in managing the dataset access requests over a two-year timespan at 6 CMS sites. The results of our simulation show that PPC outperforms LRU reducing the number of cache misses up to 20% in some sites.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The Large Hadron Collider (LHC) [54, 86, 58] at the European Organization for Nuclear Research (CERN) is running four major experiments (ALICE, ATLAS, LHCb, CMS) probing the properties of elementary particles. These experiments are producing an unprecedented amount of data, making it impossible to deploy suitable storage and computing resources at one single place. This has led to the choice of exploiting Grid infrastructures [66, 67].

On top of the Internet, which has made it possible to share information stored on computers across the world, the CERN Grid, a distributed data-intensive scientific infrastructure, provides transparent access to geographically distributed storage and computing systems used to analyze the 25 Petabytes of data annually generated by the Worldwide LHC Computing Grid (WLCG) [134]. This infrastructure serves a community of more than 10,000 physicists across hundreds of computing centres around the world, cooperating with other Grid projects such as the European Grid Infrastructure (EGI) [82] and Open Science Grid (OSG) [119].

The main experiments conducted at CERN run their software applications on the Grid, which allows the research community to easily access distributed resources. Monitoring such a distributed, geographically sparse and data-intensive infrastructure has proven to be a core functionality to support WLCG functions and operations during LHC data-taking years. General-purpose, Web-based dashboards provide a uniform monitoring interface for scientists and site administrators.

However, the WLCG monitoring infrastructure is undergoing a major extension to better cope with volume and variety of monitored data. This is due to either higher LHC luminosity (an accelerator parameter directly linked to discovery potential) expected in the next runs and the deployment of new resources into the infrastructure. In this scenario, the traditional relational database systems previously used to store and serve monitoring events hit scalability limits [18].

1. Introduction

Relational databases struggle with the efficiency of certain operations key to Big Data management. They don't scale well to very large sizes: although Grid solutions can help, the introduction of new clusters on the Grid is not simple. Databases are also not optimized for searching unstructured data, they do not handle well data in unexpected formats and it is difficult to implement certain kinds of basic queries using SQL (e.g. full-text search, the shortest path between two points). To this regards, NoSQL databases [87] explicitly sacrifice some of the traditional SQL capabilities (transactions, locking, consistency, triggers, constraints on the format, granularity and timeliness of data, etc...) in order to allow effective unlimited horizontal scalability. NoSQL databases have received significant attention due to their simplicity in design, scalability with clusters and great management of unstructured data and document-oriented information.

Over the last decade, the challenge of handling Big Data has been taken over by many companies on the Internet domain, leading to a full paradigm shift on data archiving, processing and analysis. A full stack of new technologies was originated by the development of the Hadoop File System [135] (HDFS), which has emerged as the *de-facto* standard for big data processing, largely adopted in both the research and industrial communities [14, 133]. Hadoop [150] is a software technology designed for storing and processing large volumes of data distributed across a cluster of commodity servers and storage. Hadoop was initially inspired by papers published by Google and Yahoo outlining its feasibility to handle large volumes of data very common for example when indexing Web content. With growing adoption across industry and government, Hadoop has rapidly evolved to become an adjunct to – and in some cases a replacement of – the traditional Enterprise Data Warehouse.

One of the key capabilities of a Hadoop-like environment is the ability to dynamically and "horizontally" expand the number of servers being used for data storage and processing. The cost of storing large amounts of data in a relational database gets very expensive, where cost grows geometrically with the amount of data to be stored, reaching a limit in the PB range. The cost of storing data in a Hadoop solution grows linearly with the volume of data and there is no ultimate limit. The trade-off for running these large scale computations is the high latency of jobs or, in certain cases, some limitation in the data and processing models.

1.1 Big Data Analytics for CMS

The need for agile data analytics platforms is increasing in High Energy Physics (HEP). The current generation of HEP experiments has globally surpassed the Exabyte of storage, with deployed computing resources exceeding one million computing cores [1] (Fig. 1.1).

Tier	Pledge Type	ALICE	Required	Balance	ATLAS	Required	Balance	CMS	Required	Balance	LHCb	Required	Balance	SUM	Required	Balance
Tier 0	CPU (HEP-SPEC06)	350,000	350,000	0%	411,000	411,000	0%	423,000	423,000	0%	88,000	88,000	0%	1,272,000	1,272,000	0%
Tier 1	CPU (HEP-SPEC06)	279,549	307,000	-9%	969,474	949,000	2%	561,540	600,000	-6%	250,074	253,000	-1%	2,060,637	2,109,000	-2%
Tier 2	CPU (HEP-SPEC06)	312,924	398,000	-21%	1,136,262	1,160,000	-2%	929,732	900,000	3%	164,109	141,000	16%	2,543,027	2,599,000	-2%
Tier 0	Disk (Tbytes)	26,200	26,200	0%	27,000	26,000	4%	26,100	26,000	0%	11,400	11,400	0%	90,700	89,600	1%
Tier 1	Disk (Tbytes)	30,359	30,500	0%	80,265	72,000	11%	55,338	60,000	-8%	26,252	24,500	7%	192,214	187,000	3%
Tier 2	Disk (Tbytes)	28,950	35,100	-18%	86,400	88,000	-2%	65,691	70,000	-6%	3,714	5,700	-35%	184,755	198,800	-7%
Tier 0	Tape (Tbytes)	49,100	49,100	0%	105,000	94,000	12%	97,000	99,000	-2%	33,600	33,600	0%	284,700	275,700	3%

Figure 1.1: REsource, Balance & USage (REBUS) at the Worldwide LHC Computing Grid (WLCG). Source: REBUS pledge summary for 2018.

On top of this, computing infrastructures are highly distributed, and data transfer between sites is a major activity in the computing operations. Systems of such complexity require a careful monitoring in order to be maintained in a healthy state. At this scale, processing Terabytes of monitoring data on a daily basis becomes a real challenge. Big Data derived approaches have started to be introduced in order to get a better understanding of the computing operations and utilization of computing resources. Among others, the Elasticsearch, the MapReduce [43] programming language and associated databases such as Cassandra and HBase, and the Apache Spark[2] open-source cluster-computing framework are attractive solutions to handle large data sets.

CERN itself is moving most of its computing dashboards from a standard RDBMS oriented approach (current dashboards use Oracle as a back-end) towards Hadoop based solutions [7]. The expected outcome of this transition is to efficiently process large data sets and aggregate information across distributed data providers. Such studies are becoming more and more important due to the need of reducing computing resources and the associated cost with respect to general increase of physics data

[1] wlcg-rebus.cern.ch/apps/pledges/summary
[2] spark.apache.org

collected by the LHC experiments [17, 111]. Funding agencies and review committees are, in fact, asking for increasingly optimized computing operations which can only benefit from a deeper understanding of computing resource utilization.

The Compact Muon Solenoid [55] (CMS) experiment at CERN designed and implemented a computing model that gave a crucial contribution to the recent discovery of the Higgs boson [56]. Within this model, distributed monitoring infrastructures have collected for decades any kind of data and metadata about the usage of the distributed computing infrastructure by its large community of users.

Nevertheless, there exists a number of this information, either structured or unstructured, that is rarely or never used. This is why the CMS community is promoting exploratory activities to investigate values of interest from this information, using Big Data analytics approaches and possibly discovering patterns and correlations that can be exploited to reduce operational costs and/or to improve throughput and efficiency.

In particular, in 2015 CMS has begun to store into a Hadoop cluster nearly 4 PB of monitoring data produced by its monitoring systems. This data includes information about users, jobs life-cycle, resources utilization, sites efficiency, software releases, dataset access logs and usage statistics. The data is enriched with information coming from conference calendars, internal deadlines and trouble-ticketing services, which can bring valuable insights on the usage patterns.

Hadoop has come a long way since the early versions that were substantially simplifying the batch processing of MapReduce jobs on large HDFS-based data volumes. Since the introduction of the YARN resource manager [146], Hadoop can handle a wider range of data processing tasks. From then on, Hadoop-based jobs have become the battlefield of the Spark cluster-computing framework thanks to its unique and integrated support for enterprise use cases including rapid in-memory processing of sizeable data volumes, machine learning module, SQL, data streaming and graph analysis. The benefit of using Spark for CMS experiment is two-fold. On one hand, it reduces the processing time on large dataset and allows to manage effectively heterogeneous data and metadata. On the other hand, it allows to perform advanced analytics tasks over large dataset from distributed data sources. Starting from unstructured raw data, the data preparation and analysis steps can be completely defined in the Hadoop infrastructure. This opens up a possibility to offload production RDBMS workload onto a Hadoop +Spark environment which is more suitable for large data aggregations and distributed joins.

This challenging setup offers a unique opportunity to study problems that, given the outstanding deployment of computing and storage resources, would be impossible to reproduce elsewhere. This thesis addresses the necessary work on the research, the design and the development of the new data store and analytics platform for the evolution of the WLCG monitoring and large scale data analysis. It studies two main use cases entirely based on the CERN Big Data facility, which are preparatory to the research contributions outlined in Section 1.3.1.

- CMS dataset popularity, which provides various insights about CMS user activities based on CMS monitoring data recorded in the CERN HDFS. In particular, we discuss the design of a scalable pipeline of software components aggregating heterogeneous data from multiple data-streams. These components can feed the visualization dashboards aimed at presenting to the CERN engineers useful information about user activities on the Grid, such as number of accesses, CPU metrics versus data-tiers, sites usage, etc.

- Intelligent dataset placement on the distributed infrastructure based on modeling and prediction of the CMS datasets that will be used frequently in CMS analysis jobs, hereinafter the popular datasets. These activities include fast queries for reprocessing monitoring time intervals, file-to-dataset aggregation and correlation, as well as the evaluation of predictive models.

1.2 Dataset Popularity

CMS makes a considerable usage of Grid resources for data storage and offline analysis. Fig. 1.2 shows an excerpt of the CMS analysis job monitoring dashboard. Plot (a) shows an average of 40,000 weekly jobs running at CMS sites, reaching workload peaks of nearly 500,000 concurrent jobs accessing distributed data. Hitting the boundary of computing resources causes a number of pending jobs, which are waiting to be executed, that can be as high as 35,000 per week (b). The constant increase of physics data collected by the CMS experiment results in a number of completed jobs that has increased by 10x over the last decade, reaching peaks of almost 6M jobs on a weekly basis across the last two years (c). In terms of cumulative numbers, CMS has well exceeded 300M successful analysis jobs since beginning of 2016 (d).

Data storage becomes extremely challenging with more than 40 PB of data resident at Grid sites, 10 PB of which accumulated during the last year (Fig. 1.3 (a)). The cumulative amount of data transferred from CERN to remote sites since the beginning

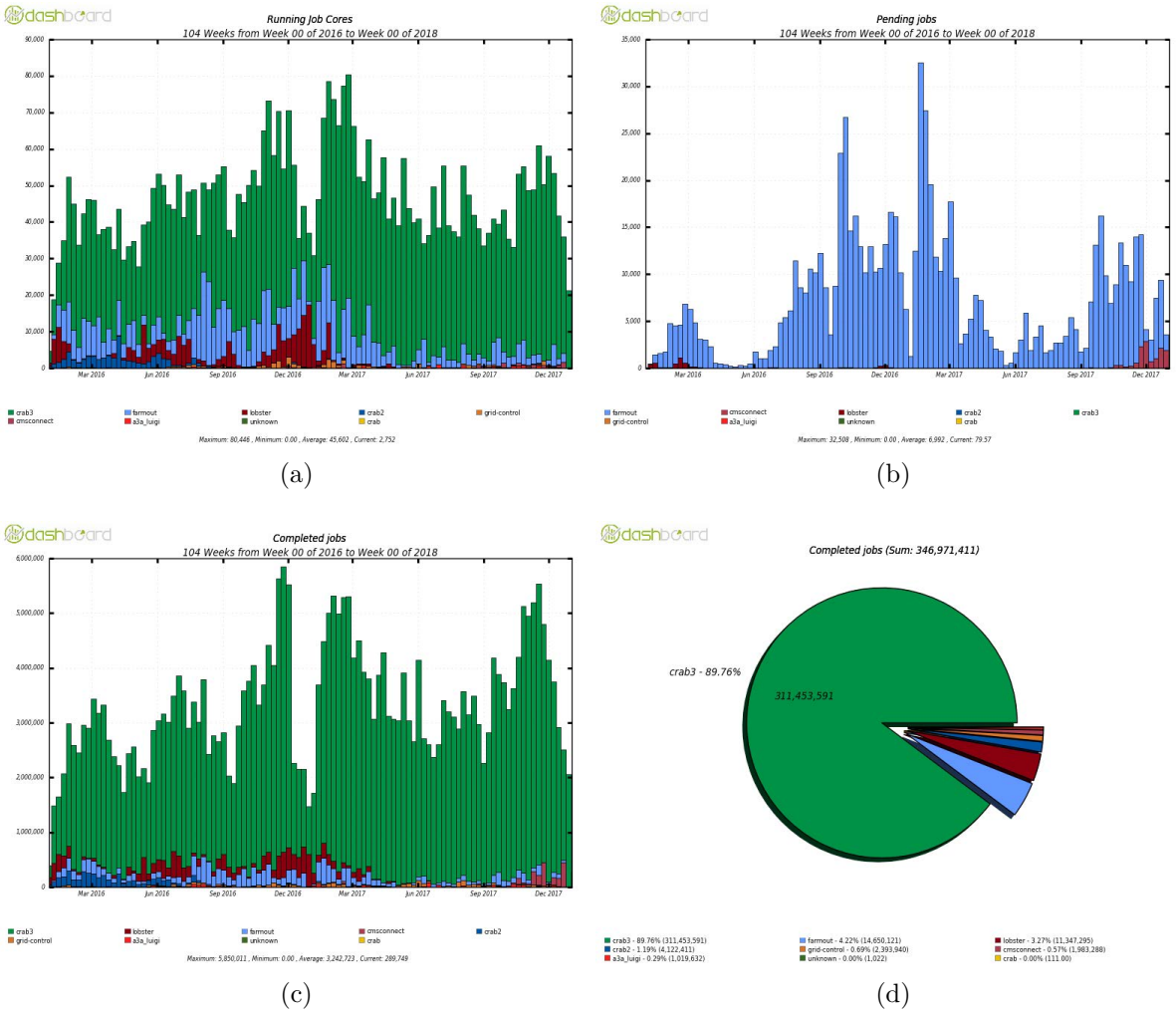
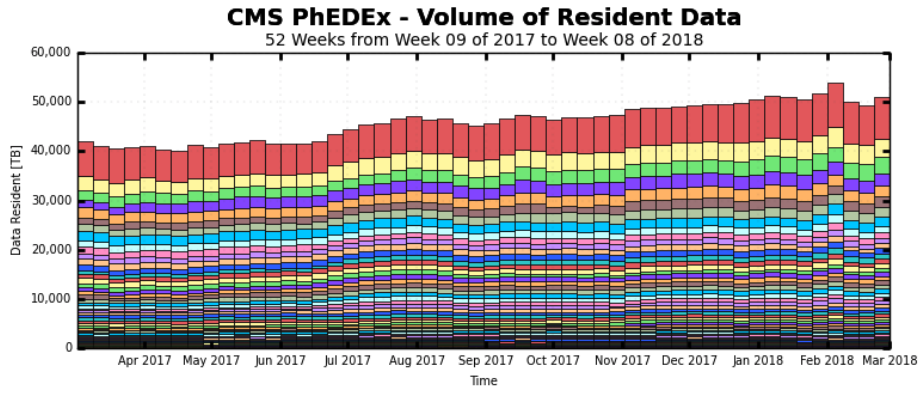
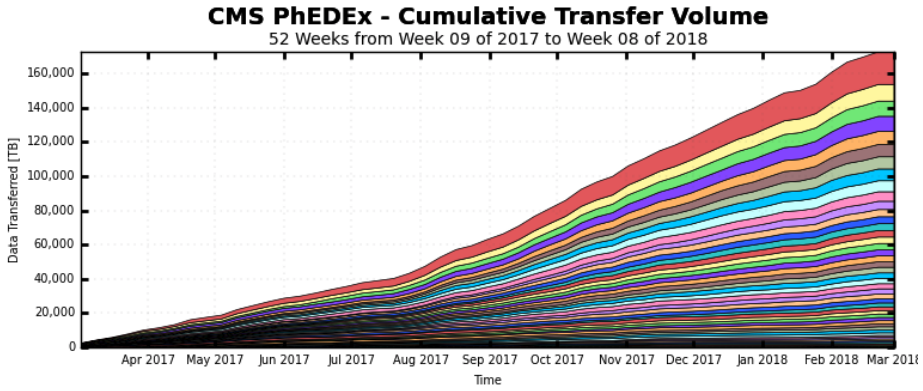


Figure 1.2: Number of weekly running (a), pending (b) and completed (c) analysis jobs on the CMS Grid in the past two years, and cumulative number of jobs completed (d). Source: CMS Dashboard monitoring [11].

of 2018 has been five times as much as it was in the previous year (Fig. 1.3 (b)), and this volume is expected to further increase.



(a)



(b)

Figure 1.3: Evolution in the past 12 months of the volume of resident data (a) and the cumulative data transferred (b) to remote sites. Source: CMS PhEDEx monitoring [49].

Given the scale of the CMS Grid infrastructure, control and optimization of storage resources is a complex task. Allocation and usage of the experiment's datasets need to be carefully monitored over time in order to maximize the efficacy of resource deployment. The concept of "data popularity" has been introduced at CERN to model dynamic data placement and optimize allocation of resources. A dataset popularity model aims at learning which dataset will be mostly used at which site. It estimates how much the dataset is requested by the sites of the system. Therefore, the popularity of a dataset becomes an observable measure that quantifies the interest of the CERN community for data samples belonging to a dataset on the basis of metrics such as the number of local or remote accesses to data files by the user jobs.

Currently, the CMS Popularity Service [104] monitors which data is mostly used by tracking over time the number of accesses, the result of read operations, the CPU time spent processing each file, the number of unique users that access each file. A substantial limitation of the service is given by its "static" intelligence. As it typically occurs to monitoring systems, it relies on fixed algorithms that are unable to learn from experience and does not evolve and improve over time. For example, monitoring information can be used to trigger ad-hoc cancellation of unaccessed replicas and replication of frequently accessed datasets, but the system is currently unable to learn from historical data.

We move forward from this current implementation and demonstrate the effectiveness of a predictive model that can forecast popularity of new and existing datasets by leveraging adequate dataset replacement policies at remote sites, with potentially large impact on resource utilization, job latency and system throughput. An accurate dataset popularity model can in fact be probed to decide at what CMS site replicas should be located and thereby improve the dataset access latency and the overall efficiency of the distributed storage system.

1.3 Statement of the Problem

This work leverages the CMS Hadoop data store, and describes the design and development of a scalable machine learning (ML) platform designed to predict the number of future accesses to new and existing datasets, which can be further probed to enable intelligent caching policies or to identify the ideal number of dataset replicas and their best locations.

Data placement policies based on replicas of popular data across Grid sites have been researched since the early days [126] and have focused on the analysis of dataset popularity distributions from access patterns. We explore state-of-the-art ML techniques to learn from past usage information which dataset will become popular in the future and exploit such knowledge to optimize dataset caching at every site of the CMS distributed infrastructure. In order to model our popularity classification problem, we aggregate historical data about dataset accesses on a weekly basis. The dataset attributes include physics information parsed from the dataset namespace, as well as static and runtime statistics about the use of each dataset. On the basis of this information we construct different sets of features given in input to streamlining classifiers trained to predict dataset popularity. This prediction task involves many different cases. For example, consider new datasets produced from either LHC collisions or

from simulated workflows. For these datasets we have only static information related to their creators, locations, and namespaces. On the other hand, existing datasets, in addition to the above metadata, are associated with historical usage information. The performance of the classifiers trained for the two cases are obviously different, even if the output popularity label predicted has the same meaning.

1.3.1 Purpose of the Study

This thesis investigates three key requirements for PB-size dataset popularity models at the WLCG:

- the need for a scalable Hadoop data vault for an effective data management solution capable of collecting dataset computing logs from different monitoring services and organizing them into periodic snapshots;
- the need for a scalable pipeline of ML components for training, on these snapshots, predictive models able to forecast which datasets will become popular over time, discovering patterns and correlations useful to enhance the overall efficiency of the distributed infrastructure in terms of CPU utilization and task completion time;
- the need for a novel caching policy based on the dataset popularity predictions that can outperform the current dataset replacement implementation.

Clearly, for this execution scenario to work, we must have a reasonable performance model proving that the CMS dashboard and monitoring systems can largely benefit from Spark parallelism and in-memory distributed computing. We benchmark them and show that the speedup achieved on the processing of dataset access information ranges between 2x to 50x compared to the previously used RDBMS architecture. Upon this testbed, the thesis proposes the following contributions:

1. a scalable *data mining pipeline*, built on top of the CMS Hadoop data store, to predict the popularity of new and existing datasets accessed by jobs processing any of the 25 event types stored in the distributed CMS infrastructure. We cast the problem of predicting dataset accesses to a binary classification problem where we are interested to forecast if a given dataset will be *popular* or not at time slot t in a given CMS site s, i.e., it will be accessed for more than x times during time slot t at site s. Our experiments show that the proposed predictive models reach very satisfying accuracy, indicating the ability to correctly separate popular datasets from unpopular ones.

2. a novel intelligent *data caching policy*, PPC (Popularity Prediction Caching). This caching strategy exploits the popularity predictions achieved with our best performing classifier to optimize the eviction policy implemented at each site of the CMS infrastructure. We assess the effectiveness of this caching policy by measuring the hit rates achieved by PPC and caching baselines such as LRU (Least Recently Used) in managing the dataset access requests over a two-year timespan at 6 CMS sites. The results of our simulation show that PPC outperforms LRU reducing the number of cache misses up to 20% in some sites.

1.3.2 Limitations of the Study

The proposed *scalable pipeline* for large traces of dataset access logs is a state-of-the-art general approach to new generation predictive models of popular datasets, which exploits Big Data analytics techniques and overcomes shortcomings of traditional RDBMS. The predicted popularity results are computed on historical traces that are representative of the specific HEP research domain. The model is in fact limited to CMS computing logs, which however represent common samples of monitoring data for high performance and high throughput distributed computing systems in HEP experiments. Although the described scalable pipeline of feature construction and learning techniques is applied to CMS dataset access logs, we argue that it is viable for any PB-size log streams. It can be exploited to reason about the dynamics of dataset access patterns and it is suited for simulation and control environments that go beyond the purpose of resource monitoring, which is normally employed only for near-time debugging reasons, accomplished per sub-systems and affected by scarce integration that makes it hard to find correlations between different classes of information. The type of analysis jobs from which log traces are derived is rather uniform but cannot be taken as general purpose because it relates to the particle physics domain. Some popularity features do not depend on physic contents but on the computing platform data refers to. The experiments are conducted on historical data and the predictive models are trained offline in an isolated environment available for experiments at CERN. A complete integration of the developed components in the production CMS infrastructure would be necessary to fully understand their performance and impact.

The proposed *data caching policy* is, at the best of the author's knowledge, the first approach in Grid computing to a dataset replacement policy not focused just on the monitoring information from the remote sites, but also on an accurate eviction policy driven by the popularity predictions. The data mining pipeline and the data caching policy are independent, so it is possible to exploit the dataset popularity

features on different models of Grid resources whose access logs are ingested to Big Data infrastructures, although their applicability needs further investigation. The proposed PPC policy is an approximation of the optimal solution. This solution is only theoretically quantifiable because it needs knowledge of future dataset accesses that in our policy we try to predict at best. This is due to the main assumption that the dynamic nature of the Grid storage resources and their access patterns will hinder the use of optimal mappings between datasets and ideal sites. The initial mapping from historical log traces can be considered a good "clue" to start the execution of the PPC replacement policy. The presented eviction model can be taken for further optimizations.

1.4 Outline

The rest of the thesis is organized as follows.

Chapter 2 outlines background knowledge and related work. It introduces the concept of predictive models and compares learning techniques applied to dataset popularity prediction and subsequent dataset caching. It explains the difference with classic log analysis and examines the architectural trends and the most successful projects.

Chapter 3 describes the analytics tasks performed on the CERN Hadoop clusters in order to validate the technological transition from RDBS to Big Data. Results were presented in [106, 107].

Chapter 4 introduces the dataset popularity problem and describes the raw data available, the groups of features extracted from this data and how these features are used for the prediction task. Access logs are studied in order to motivate the cutoff values on continuous variables. The test results will be designated as positive or negative depending on whether the resultant value is higher or lower than the cutoff. In addition, the choice of the best time interval for dataset popularity prediction is also demonstrated. The experimental settings are discussed. We detail the results of the experiments conducted to asses the effectiveness of our solution to the dataset popularity prediction problem. The impact of data cleaning and feature engineering steps is also assessed. Resampling techniques are analyzed. We evaluate the contribution of per-site prediction models, measure the performance of our best classifier on new and existing datasets, and demonstrate the aging effect of a static model versus models that instead updated themselves over time. Contents have been presented in [107, 109, 108].

Chapter 5 describes how to harness popularity predictions to implement an effective dataset replacement policy for the CMS distributed infrastructure, where datasets selected for eviction are retained in the cache if become popular in the coming week. The hit rates measured are compared with those achieved using state-of-the-art caching policies. Our replacement policy outperforms the other strategies particularly when the cache size is small, which makes it very effective in production sites with limited storage or bandwidth.. Discussion of the results can be found in [109, 108].

Chapter 6 draws the conclusions, outlines the future work and discusses a challenging event classification opportunity in CMS that can leverage the experience with dataset popularity, although it is currently limited by the format of raw data.

All results of this work have been beforehand discussed and presented during biweekly *CMS Popularity Meeting* and *CMS Computing Monitoring Forum* throughout 2015, 2016 and 2017. They have been also publicly presented at the biannual *CMS Offline&Computing week* in 2015. Conference papers have been reviewed by *CMS editorial board* prior to submission. The following is the list of contributions to conferences and journals.

1. Marco Meoni, Tommaso Boccali, Nicolò Magini, Luca Menichetti and Domenico Giordano. XRootD popularity on Hadoop clusters. In 22^{nd} International Conference on Computing in High Energy and Nuclear Physics (CHEP), *Journal of Physics: Conference Series*, 898(7):072027, 2017.

2. Marco Meoni, Valentin Kuznetsov, Tommaso Boccali, Luca Menichetti, Justinas Rumševičius. Exploiting Apache Spark platform for CMS computing analytics. In 18^{th} Advanced Computing and Analysis Techniques in Physics Research (ACAT), *arXiv preprint arXiv:1711.00552*, 2017.

3. Marco Meoni, Raffaele Perego, Nicola Tonellotto. Popularity-based caching of CMS datasets. In 17^{th} International Conference on Parallel Computing (PARCO), *Parallel Computing is Everywhere*, pages 221–231, DOI 10.3233/978-1-61499-843-3-221, 2018.

4. Marco Meoni, Raffaele Perego, Nicola Tonellotto. Dataset Popularity Prediction for Caching of CMS Big Data, *Journal of Grid Computing*, ISSN 1572-9184, 16(2):211–228, Feb 2018 2018.

Chapter 2

Background

This chapter reviews the state of the art of log analysis, dataset popularity modeling and dataset caching. First it introduces the research domain of the entire work, which is the HEP experiments at CERN LHC with focus on CMS. Then, it reviews the most important milestones in terms of log analysis for general resource optimization problems. It narrows the study to dataset analytics with focus on the popularity of a dataset. The concept of popularity introduces the aspect of predictive models and current efforts in modeling dataset popularity by exploiting ML techniques. An important derivation of the popularity forecast is its application to dataset caching in order to optimize the replacement policy at different sites based on future knowledge of dataset access. The end of each section discusses the position of this work with respect to the presented topics.

2.1 The CMS experiment at CERN LHC

The Large Hadron Collider (LHC) is a particle accelerator and collider (Fig. 2.1). It is located beneath the Franco-Swiss border near Geneva in Switzerland where the Large Electron-Positron collider (LEP) [2] previously existed. The purpose of the LHC is to give scientists an experimental apparatus that would let them to test theories in high energy physics, such as the existence of the Higgs boson, the search for super-symmetries and particles that would indicate the existence of dark matter, etc. Indeed, one of its recent results was the discovery of the Higgs boson, publicly announced on July 4th 2012, as predicted by the Standard Model. The LHC consists of a 27 km long circular ring, designed to accelerate protons and heavy ions at high energies. It is characterized by two accelerated beams traveling in opposite directions inside different channels in the same pipe at ultrahigh vacuum.

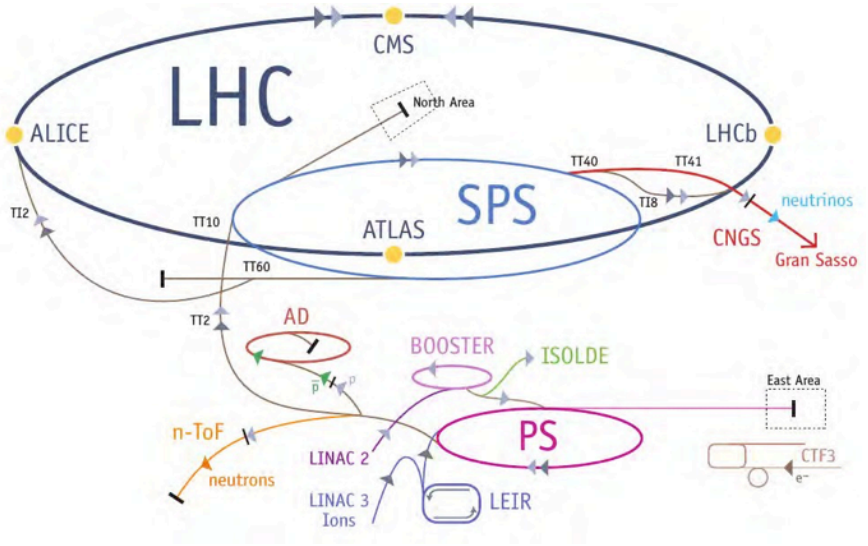


Figure 2.1: The Large Hadron Collider (LHC) accelerator complex at the European Organization for Nuclear Research (CERN).

The acceleration process for protons is done in five steps. Hydrogen atoms are ionized to produce protons and injected in a linear accelerator. When protons reach an energy of 50 MeV they enter a Booster which increases their energy up to 1.4 GeV. At this point they enter the Proton Synchrotron (PS) where 277 electromagnets push the protons to 99,9% the speed of light and energy becomes as high as 25 GeV. Second to last step is accelerate proton bunches in the 7 km ring Super Proton Synchrotron (SPS) and energy ramps up to 450 GeV. Protons are ultimately injected into the LHC in two separate pipes and move in opposite directions. Here, through magnets, the particles are accelerated up to their maximum designed energy of 14 TeV. The two LHC channels intersect in the four caverns where the detectors of the four main experiments - ALICE [53], ATLAS [51], LHCb [52] and CMS [55] - are installed. The vacuum system is necessary so the particles do not lose energy in the acceleration process due to impacts with the molecules that constitute air.

The Compact Muon Solenoid (CMS) is a general-purpose detector at LHC. It has a broad physics program ranging from studying the Standard Model, including the Higgs boson, to searching for extra dimensions and particles that could make up dark matter. Although it has the same scientific aims as the ATLAS experiment, it uses different technical solutions and a different magnet-system design. The CMS detector is built around a huge solenoid magnet. This takes the form of a cylindrical coil of superconducting cable that generates a field of 3.8 Tesla, that is about 100,000

times the magnetic field of the Earth. The field is confined by a steel "yoke" that forms the bulk of the detector's 14,000-ton weight. The whole detector is 21 meters long, 15 meters wide and 15 meters high. The CMS experiment is one of the largest international scientific collaborations in history, involving 4300 particle physicists, engineers, technicians, students and support staff from 182 institutes in 42 countries.

The particle detector is built around a huge superconducting solenoid. It is like a giant filter where each layer is designed to stop, track or measure a different type of particle emerging from proton-proton and heavy ion collisions. By measuring the energy and momentum of a particle it is possible to get clues of its identity. Particular patterns of particles are indications of new and exciting physics. CMS is made of five concentric layers (Fig. 2.2): a central tracking system to give accurate momentum measurements; a high resolution electromagnetic calorimeter to detect and measure electrons and photons; a "hermetic" hadron calorimeter designed to entirely surround the collision point and prevent particles from escaping; a muon chamber to detect and measure muons; a solenoid to curve the trajectory of the particles and calculate their momentum and whether they have positive or negative charge.

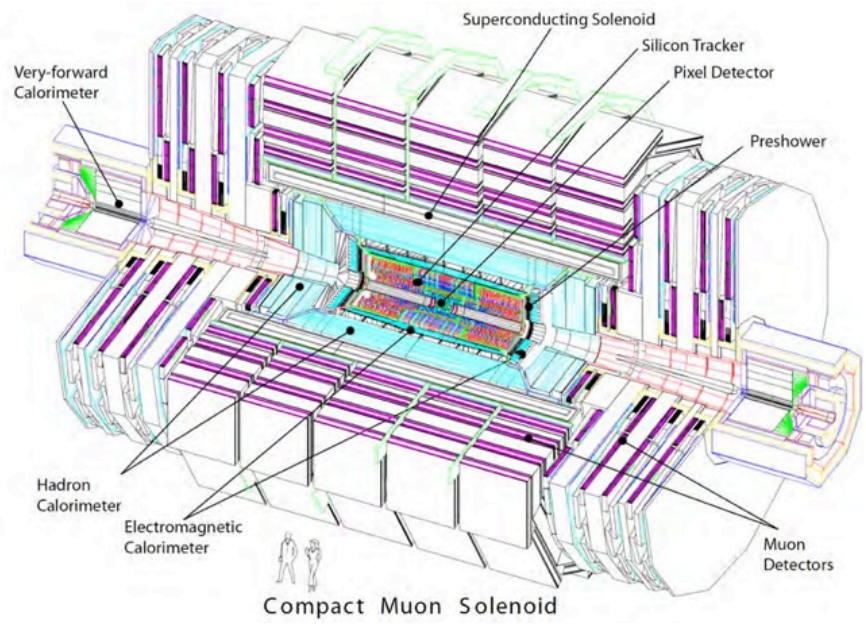


Figure 2.2: A section of the Compact Muon Solenoid (CMS) particle detector at CERN.

2.1.1 Data Management

CMS currently generates experimental data at a rate of tens of Petabytes every year. This data volume makes the CMS experiment a perfect fit for using distributed data-intensive computing infrastructures for storage and analysis. In the following we detail the list of core services and components used by the CMS distributed infrastructure. They are among the main sources of information for our studies.

CRAB

In a data Grid environment, end-users access computing and data resources transparently with no specific knowledge about the underlying complexities of the system. In CMS this abstraction layer is provided by the CMS Remote Analysis Builder (CRAB) [38]. CRAB has been in production since 2004 and is currently at its third iteration. CRAB takes end-user workflow requests and handles their management and scheduling through a multi-service analysis system. These services handle numerous tasks including receive and inject user requests, track and manage the requests in a global queue using HTCondor [143], prioritize requests and translate user requests into jobs. Workflow data is stored into an Oracle database. Status of site resources is collected from so called machine-ads and matched with class ads, descriptions advertising the resources required to execute the job. Class and machine ads are central to HTCondor matchmaking [39]. Machine ads may not accurately represent the number of CPUs available at a site, most sites dynamically allocate CPUs to the CMS experiment on an as needed basis. The current need may be less than the total resources available to CMS. CRAB reports usage data to the CERN Dashboard service [11].

PhEDEx

Traditionally, data transfer management was an expensive activity; tasks such as ensuring data safety, large-scale data replication and tape migration/stage of data have relied on manpower. As dataset sizes reached Petabyte scale the manpower required scaled linearly but available manpower remained flat. To solve this problem, the CMS experiment developed the Physics Experiment Data Export (PhEDEx) component [127] to automate these tasks. PhEDEx takes care of replication and deletion of data in the underlying infrastructure. Tasks such as file replication, routing decisions, tape migration, and file pre-staging are all dealt with using a suite of agents running on each site. Interaction with PhEDEx is done through a RESTful [63] Web API.

DBS

The CMS Dataset Bookkeeping Service (DBS) [4] provides access to a data catalog of all event data for the CMS experiment. Dataset information includes dataset size in bytes, number of files, physics group name, data tier, creation date, dataset type and mapping of datasets to files. The data in DBS is recorded from all Monte Carlo generation and analysis steps and can be used to track data back to the raw data or the Monte Carlo simulation it was originally generated from. However, these attributes can also be used for improved data popularity predictions. Dataset information for CMS users is provided by the CMS Global DBS using an Oracle database. Authentication is handled using standard X.509 Grid certificates and the database is accessed through a Web API.

PopDB

The CMS Popularity Service (PopDB) [104] is the component that collects dataset user access information. The PopDB database stores historical information of dataset usage in various metrics, which are used by the distributed data manager to improve system performance and replicate popular datasets to other sites. It exposes its data via Web API providing an abstraction from low level data acquisition protocols.

2.1.2 Resource Monitoring

A first approach to resource optimization is normally accomplished by collecting monitoring information. Monitoring frameworks are usually deployed to distributed infrastructures to understand the actual resource utilization and make quantitative predictions of the expected improvement in workload performance. They involve a key and very difficult task for an administrator, namely how to justify upgrades of CPU, memory and storage. Monitoring frameworks and their database systems are generally not designed with support to decision in mind and hence they only offer partial help to administrators.

Experiments at the Worldwide LHC Computing Grid (WLCG) deal with this difficulty at the largest scale in terms of computing resources, and a few projects have raised the need for intelligent prediction policies. Most of the tasks that require intelligence need also the ability to induce new knowledge from experience. Algorithms that can extract information automatically are normally borrowed from ML techniques, which are fairly generic and can be applied in various settings. These techniques can be utilized by Data Mining (DM) [74] [94] processes, which have emphasis on utilizing

data from a given domain, for instance CERN dataset popularity, to understand some questions in that domain. DM may also drive the advancement of ML algorithms and forecasting models.

2.2 Log Analysis

Predictive models based on statistical learning techniques are suitable for studying the performance and scalability of complex computing infrastructures. The training process requires to abstract features from a variety of measurements usually collected through historical logging activities, and to devise relevant metrics to estimate the behavior of the system under investigation. Log analysis related to the processing of structured or unstructured information collected through several layers of monitoring architectures is a promising field of research.

Advances and challenges in log analysis are discussed in [113]. The authors argue how logs contain a wealth of information to help manage systems. Log data can be used to optimize or debug system performance as well as to make prediction and provisioning for the future. Understanding the performance of a system is often related to understanding how the resources in that system are used. The authors examine runtime interleaving of log messages, anomaly detection, relationships between components, and the need for providing ML tools with input data in the format of numeric *feature vectors*. This is a nontrivial but essential task to be addressed in order to convert free-text log messages into meaningful features [153]. They also debate that although the modeling techniques for ML may be common across various systems, the log data mined to build the model, as well as the metrics predicted, may differ. This is a universal truth that usually leads to domain-specific constraints. Log data profiling and reporting is generally accomplished by a variety of statistical techniques like clustering algorithms for grouping similar events, while Markov chains better fit pattern mining where temporal ordering is crucial.

A logging infrastructure is essential for supporting log analysis. Among the successful works, Chukwa [122] is in the top chart. It archives data using Hadoop to take advantage of distributed computing infrastructure. This approach corresponds to the data collection module described in chapter 3. In fact, large-scale log collection has become common in distributed architectures based on Cloud or Grid computing. WLCG belongs to this scenario: automated log analysis is increasing the amount - and consequently the value - of information that can be extracted from logs. Hence, scalable data processing is challenging and so it is very desirable to leverage existing

tools for data-intensive processing of all kinds, first and foremost MapReduce [43]. Tasks like indexing and aggregation fit naturally into the MapReduce paradigm. So do more sophisticated analyses, such as ML-based resource allocation or dataset popularity.

Chukwa is one among a few examples of specialized systems for log collection, indexing and analysis. These systems become a natural choice to enable various (short-turnaround) log analysis techniques based on ML approach. Short-turnaround analytics are essential for data placement decisions and make sure that dataset replicas are placed on sites with high likelihood of job submission affecting that very dataset. In fact, this proposal too relies on a similar system as a key piece of infrastructure for the underlying analysis. However, the Director service in the Chukwa architecture connects on boot to each data collector and requests copies of all reports. This kind of *polling* mechanism has demonstrated over the years to manifest scalability weaknesses in WLCG Grid infrastructures, while *discovery&register* approaches for distributed services have proven to be more reliable. For this reason, instead of Chuwka we are going to use Flume [47], a more recent system developed for streaming logs into HDFS. Both systems have the same overall structure, but Chuwka has an end-to-end data flows rather than a more hop-to-hop model implemented by Flume. There are also a number of more specialized monitoring systems worth mentioning. Tools like Ganglia [101] and MonALISA [91] are designed to help users query distributed system monitoring data. They have been extensively used at WLCG for more than a decade.

Dapper [27] is a Google's production distributed system tracing infrastructure that exploits the concepts of effective job-log retrieval to infer dependencies between individual jobs and shared software infrastructure used by those jobs at various service granularity. In essence, Dapper exposes an API for tracing distributed request workflows and aims at identify performance anomaly in request's segments (*spans*). It incorporates small monitoring modules at the application level, which collect logs from each execution thread and stream data into BigTables. Although the implementation is specifically tailored to the Google's IPC model, Dapper offers a large-scale tracing framework to a dozen of analytical tools built on top.

The authors of [81] combine traces and MapReduce processing and derive a data mining model. They analyze a large set of MapReduce job-logs and characterize resource utilization patterns, job patterns, and sources of failures. This task is achieved via an instance-based learning technique that exploits temporal locality to predict job completion times from historical data and identify potential performance problems in the dataset. Logs are categorized according to several criteria (job status,

type, duration, etc) and statistical properties of the trace data (mean, standard deviation, coefficient of variation, etc) are derived. The log classification approach and the performance prediction model provide interesting insights. On top of the identification of K-Nearest-Neighbors (KNN) for the incoming job, the authors predict job completion times by comparing the effectiveness of a distance-weighted algorithm [152] against a locally-weighted linear algorithm [13]. Large predictions are flagged as potential performance problems or workload change.

2.3 Machine Learning

ML is a branch of artificial intelligence (AI) that has become popular in many computing domains. It is based on the construction of computer systems that can learn from data without being explicitly programmed. ML algorithms can teach themselves to adapt and improve as new data is supplied. They simultaneously characterize sample and computational complexities, a blend of how much data and computation is required to learn accurately [132].

A definition of ML is often expressed by the sentence:

> *A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.* [110, 48].

Examples of ML, and most recent shift to Deep Learning (DL), are many and have revolutionized our habits. Web search engines have learned how to rank Web pages and satisfy user queries with the most appropriate answers; photo editing applications recognize faces, road signals and car plates so well because learning algorithms are capable of extracting and recognize patterns; email software filters out spam thanks to the ability of learning from past messages and classify malicious messages. In all above cases, traditional approach based on rules are replaced with predictive models that try to mimic how the human brain learns. DL stacks up these models one of top of the other, each one focusing on a specific task, and hence expands the learning power involving multilayer networks of threshold units, each of which computes some simple parameterized function of its inputs [130].

Historically, ML techniques can be three-fold, in spite further distinctions can be added. Supervised learning methods use labels associated with the training dataset so that computer software can learn from these labeled examples. Unsupervised learning does not require labels but extracts significant patterns from input data.

Semi-supervised learning combines the two previous techniques mixing labeled and unlabeled data. Dataset popularity problems are usually addressed via supervised learning. Similarity and seasonality in time series access logs provide ML algorithms the capacity to learn from historical data, so that they can be harnessed for prediction and analysis.

Consistently with the adoption of Spark in CMS for HEP analysis [72], we also exploit its general-purpose large-scale data processing engine for our popularity study. In fact, after the introduction of the Spark platform [156] in 2010 and the following integration of SQL [12] and ML [105], many businesses exploited its power for various data analytics tasks. Small and large companies successfully applied this framework to different use cases which required processing large datasets. Up until now, the HEP uses a Grid infrastructure to process individual jobs at distributed sites. In order to leverage MapReduce paradigm and the tightly-coupled nature of Hadoop + Spark platforms many aspects should be modified. Among the others, changes to middle-ware in order to read underlying ROOT [32] data-format, adaptation of data workflow scheduling and submission tools as well as overall design of data processing [73]. Such effort is undergoing within recently funded projects such as DIANA-HEP in the context of NSF's Software Infrastructure for Sustained Innovation (SI2) program[1]. The analytics use cases, e.g. extracting insight from available data, are very well studied and adapted in the business world. Recently, HEP experiments realized that they can explore their log data to gain additional insight about user activities and apply this knowledge to improve their computing models [7]. In particular, exploiting ML techniques on computing monitoring data is a hot topic in the HEP community at large [40]. The data popularity problem was studied by all major HEP experiments, e.g. in CMS [85, 106], LHCb [77], and ATLAS [23]. It was found that ML approach can be useful addition to custom data placement solutions adopted by experiments.

In the Software and Computing sector of the CMS experiment, after some pioneering activities, various data services have now started to quickly setup regular dumps of their raw knowledge into CERN-IT computing systems that are adequate for parsing and aggregations. The richness of this data is acknowledged since long time, but such data was only rarely (if ever) accessed up to a couple of years ago. Nowadays, plenty of metadata are regularly parsed and useful information are aggregated from various CMS data streams and sent to HDFS. This data is building an always-growing gold mine for anyone who is willing to dig some actionable insight in terms of details on computing operations performances and patterns, in both workflow management and

[1] diana-hep.org/pages/activities.html

data management sectors. All metadata about physics data replications, WAN versus LAN data access patterns, dataset bookkeeping information, utilization of storage systems like EOS [117] at CERN, are recorded and can be queried by proper tools. There are already few CMS ongoing activities to process some streams of this large amount of data sitting on HDFS - as mentioned in this paper - and it is not hard to foresee that more activities will pop up in the close future.

In addition, it is becoming evident that - despite the LHC computing models were developed alongside with the growth of Grid computing worldwide and largely showed their success - considerable margins of improvement can still come while getting prepared for computing operations in next LHC Runs, in particular exploiting ML techniques on computing metadata. Apart from data placement optimization via the prediction of which data will become popular among physicists - as discussed in this thesis - these techniques are being explored to understand the data transfer latencies in order to reduce the operational costs of hardware resources, to predict job scheduling while using heterogeneous resources (e.g. GRID, HPC and opportunistic cloud resources), to identify anomalies in network traffic and predict possible congestions by suggestion WAN path optimizations as a key input to network-aware solution at the application layer. All these (and more) areas are hot topics in the HEP community at large, and contribute to efforts towards next-generation data-driven and adaptive computing systems.

2.3.1 From Database to Big Data

CMS data is recorded as files, which are grouped into datasets by physics content, e.g. if they were produced at the LHC accelerator in the same running period, or simulated with a given set of parameters. The datasets are replicated in multiple copies and distributed among the computing centres of the WLCG for further processing and for analysis. Currently more than 60 PB are placed on disk and more than 100 PB on tape.

The CMS Popularity service [29] has been monitoring the access to datasets since nearly a decade. The collected metrics are used to optimize data distribution, ensuring that the most used data is replicated and accessible on the constrained disk resources, while cleaning up replicas of unused or less used data.

XRootD [46] is the standard fast, low latency and scalable data access protocol at WLCG experiments. It offers advanced proxy functionality to read out non-sequential subsets of files [20] and pluggable authentication and authorization systems that allow to easily reuse CMS Globus X.509-based infrastructure [65, 64]. All data is accessed

through XRootD, either they are in the EOS [50] storage system at CERN or in the disk-based remote sites that are uniformly connected via the AAA federation [28]. Storage servers send UDP monitoring packages to report any file access to the monitoring system. These packages are collected by the GLED collector [142] and relayed to the CERN IT centralized monitoring infrastructure [6] through a message bus.

The common approach to data analytics is based on RDBMS solutions. Oracle is the historical DB standard at CERN for many bookkeeping and analytics services. Even though it works extremely well, the amount of information HEP experiments want to process is growing exponentially each year. We already found certain limitations on data processing pipelines within Oracle DB. For instance, the CMS most populated DBs, Data Bookkeeping System (DBS) [4, 71] and Data Location System (PhEDEx) [70, 129], already contain a few hundred GB of data. Merging data only from those two DBs becomes an issue. Even though they reside in one physical DB, the cross-joins among them is causing large latencies on complex queries due to large data volume.

When the amount of data that is generated and, more importantly, collected is increasing extremely quickly with sizes beyond the ability of commonly used software tools to process them within a tolerable elapsed time, the term *Big Data* is commonly cited. A consensual definition states that:

> *Big Data represents the Information assets characterized by such a High Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value.* [102]

Big Data size is a constantly moving target, as of 2018 ranging from a few Petabytes to Exabytes. More importantly, companies are recognizing that this data can be used to make more accurate predictions. Facebook, for example, knows how often users visit many websites due to the pervasive Like buttons and wants to use that information to show users ads they are more likely to click on.

CERN provides a Hadoop infrastructure as a valuable solution to overcome obstacles inherent processing of PB-size data volumes. It provides scientists with a variety of tools and Big Data frameworks (Fig. 2.3), such as Spark, Pig, Hive, which are available in different clusters. These are distinct but not isolated environments where data stored in one HDFS namespace can be accessed by jobs running indiscriminately in whatever cluster. Obviously jobs reading the local namespace will be faster and, if necessary, data can be moved from one cluster to another using MapReduce jobs. Being Hadoop a complete ecosystem for distributed computing and storage, it employs

several programming models and tools. Here below we briefly outline those ones that are of importance to our studies.

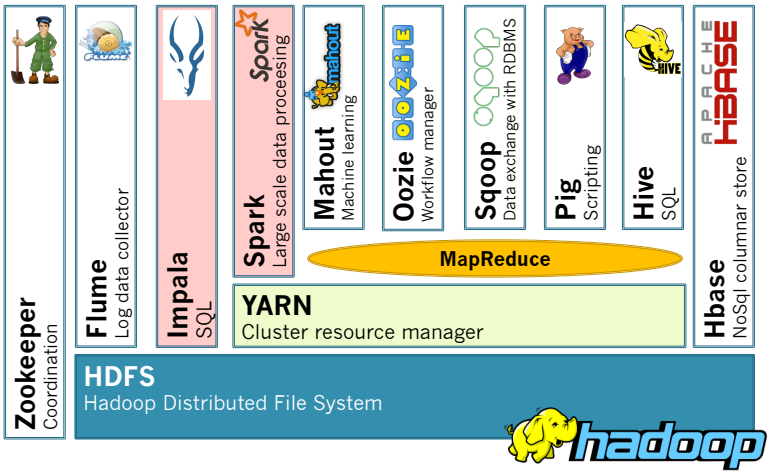


Figure 2.3: The Apache Hadoop ecosystem for distributed storage and processing of Big Data.

MapReduce

MapReduce (MR) [43] is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. It became the genesis of the Hadoop processing model. MapReduce works by breaking the processing into two separate and distinct parts: *Map* and *Reduce*. During the *map* phase, each job takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Then the *reduce* job takes the output from a map as input and combines data tuples into a smaller set of tuples.

Hadoop

Hadoop is a framework for big data storage and processing. It has three backbones: the Hadoop Distributed File System [135] (HDFS) for storage, the MapReduce processing engine and the YARN [146] resource management layer. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework. Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers

packaged code into nodes to process the data in parallel. This approach takes advantage of data locality, where nodes manipulate the data they have access to.

Pig

Apache Pig is a high-level toolkit for creating scripts that run on Hadoop. It makes Java MapReduce programming similar to SQL, it offers a fast learning curve and it allows for fast aggregation of file access logs and measurement of the performance impact. Pig offers the possibility to customize the number of parallel *reduce* tasks (1 by default), while the *map* parallelism is determined by the input file, precisely one map for each block of HDFS data. The predefined size of each block is 256 MB.

Spark

The main limitation of Pig, in terms of performance, is inherited from MR, which forces a linear dataflow on distributed programs: read input data from disk, map a function across the data, reduce the results of the map, store reduction results on disk. Spark API overcomes these shortcomings through its Resilient Distributed Dataset (RDD) object. RDD is a read-only multiset of data that takes advantage of shared memory and reduce the latency of distributed applications when compared to MR implementations like Hadoop. This allows for faster implementation of both iterative algorithms and interactive queries.

Scala and PySpark

The Spark platform is designed to process large amount of data efficiently by means of a simple programming interface. The API is divided into unstructured and structured. The former works with RDD, Accumulators, and Broadcast variables, while the latter gives access to Dataframe, Dataset and the SparkSQL Context. There are few languages which implement Spark APIs. The native solution is based on the Scala programming language, while PySpark is a Python wrapper available as a thin layer around Spark libraries.

MLlib

ML is performed on Spark using the MLlib [105] distributed machine learning framework. MLlib comes with a set of ML algorithms, although in a limited number if compared to the state-of-the-art Python-based Scikit-Learn [116] library. It leverages

Spark parallel processing of distributed data and powerful cache and persistence capabilities, which make model training and scoring very quick. MLlib provides the following algorithms for supervised classification tasks:

- **Decision Tree** [100, 145]. It is a flowchart-like tree structure, learned from class-labeled training tuple, where internal nodes denote a test on an attribute, branches represent the outcome of the test, and leaf nodes hold a class label.
- **Support Vector Machine** [33, 79]. It uses a nonlinear mapping to transform the original training data into a higher dimension where to seek a linear optimal separating hyperplane. This hyperplane is found using "essential" training tuples called support vectors and represents a decision boundary separating the tuples of one class from another.
- **Logistic Regression** [19]. It models the probability of some event occurring as a linear function of a set of predictor variables. It assumes a linear relationship between the input variables and the single output class, and it is expressed using the logistic (sigmoid) function.
- **Random Forests** [141]. It is an ensemble method that operates by learning a multitude of decision trees at training time and by outputting the class that is the mode of the classes of the individual trees. It features robustness to overfitting and high accuracy.
- **Gradient Boosted Trees** [68]. It is an ensemble method based on the optimization of a objective function given by the sum of different differentiable functions. It makes sure that the descent (negative gradient) taken to optimize the objective function does not fall into a local maxima or minima, but rather converges to the global ones. It is "boosted" because it focuses on the misclassified tuples with a series of classifiers that are iteratively learned.

2.4 Dataset Popularity

In our context, dataset access logs represent an attractive yet critical source for data analysis. CMS has recently promoted research activities related to the mining of dataset access patterns leveraging Big Data analytics techniques. The management of the CMS infrastructure would largely benefit from the knowledge of these patterns, which could be exploited for driving dataset caching and replication policies. Job

features, site characteristics and dataset access patterns can be analyzed with ML techniques able to predict with acceptable accuracy the system behavior, striking a balance between quality of service and storage allocation. To this regards, two research directions are of particular interest: how to leverage scalable ML for predicting dataset popularity and how to use the obtained predictions to implement an efficient dataset caching policy.

2.4.1 Predictive Models

The field of dataset popularity prediction for the CMS experiment was pioneered by the initiative at Cornell University [85]. The paper shows a proof-of-concept based on a use case belonging to a larger CMS analytics project having the ultimate goal of building adaptive models for computing in CMS [30]. The authors tested the feasibility of training a dataset popularity classifier from the information retrieved from the previous CMS monitoring infrastructure.

Another major experiment at CERN, the ATLAS experiment, has developed a popularity prediction tool [22, 24] for its distributed data management system. The Production and Distributed Analysis System (PanDA) [98] is a workload management system similar to CRAB in CMS. It handles the distribution and scheduling of user jobs for the ATLAS experiment. PD2P [99] was the first dynamic data manager developed for PanDA to optimize ATLAS Grid resource utilization, but was missing a popularity prediction component and thereby replaced by the ATLAS Distributed Data Manager (DDM). Every time a file is accessed through PanDA a trace is sent to the DDM tracer system, with billions of traces collected over the last ten years. The impracticality of a direct usage has yielded the development of a popularity system [112]. ATLAS DDM aims at forecasting data popularity using Artificial Neural Networks (ANN) [120] and extracting information for data deletion and replica [21]. Historical access information about files, datasets, users and sites are used to make predictions of future data popularity and possible trends. One of current downsides of ANN is the large amount of training data required for accurate models. DDM distributes dataset in two stages, a cleanup process to delete unpopular replicas and a replication process to create new replicas of popular data. The replication process uses the total number of accesses per job slot as popularity metric to redistribute data as evenly as possible among sites. Number of accesses per sites are normalized with respect to the number of job slots for each site. Only datasets that have been unpopular for a certain amount of weeks are considered for deletions, although at least one replica is always kept to avoid data loss. For current datasets at least two

replicas are required on disk. The popularity prediction module pre-filters datasets with very little usage to decrease prediction time. The evaluation of the access time indicates that the system performance has an overall performance improvement up to 1 PB of replicated data. The specific access pattern of ATLAS datasets shows that most of the jobs are submitted during the central weekdays, and very few submissions otherwise.

Similarly, LHCb, is working at data popularity predictions through a dynamic data manager based on a Gradient Boost classifier [68]. Since LHCb has more disk space constraints than ATLAS and CMS, the classifier focuses on forecasting what dataset will never be used again so that it is completely removed from the disk storage. Creation and deletion of new replicas are decided using a cost function based on predicted popularity, cost of removal from disk and replica count. Evaluation of false positives, which would require expensive restore from tape, shows better behavior than Least Recently Used (LRU) algorithms. We also take LRU as baseline for dataset replacement, but we compare our solution with other state-of-the-art algorithms. LHCb also relies on predictive models for studying the interactions of heavy particles containing the bottom quark. Yandex has provided the core simulation software with fast access to the details of specific event types. Geared specifically to the LHCb needs [95], the search system indexes and identifies events satisfying specific selection criteria using the MatrixNet algorithm. MatrixNet systematically produces and caches metadata based on popular queries described by a number of qualities such as region, type, or grammatical form. Disk storage management is then optimized according to data popularity estimator [77].

Dataset popularity is also addressed outside the particle physics domain, and it has started becoming popular since the late 1990s with the spread of online services that use recommender systems, such as Amazon or Yahoo! Music. For instance, Netflix handles video replicas and tries to implement services that give users what they want, ideally before they know it. This is achieved through multiple neural network models for different distributed dataset [42]. The approach is based on the intuitive idea that the preferences of a user can be inferred by exploiting past ratings of that user, as well as users with related behavior patterns. Since only a few factors contribute to individual taste, hyper-parameter optimization is performed upon each combination of users' preference parameters. Similarly to video popularity, song popularity [103] is also particularly important in a growing music industry. The authors in [118] define metadata and acoustic features that make a song popular, apply them to million

of songs and evaluate different classification algorithms on their ability to predict popularity and determine the types of features that hold the most predictive power.

Each of the above systems is tailored to the specific needs and data formats of the corresponding research domain. They need to take into account billions of records of historical data, which makes it impractical or impossible standard processing with conventional DBMS. Big Data infrastructures can aid the development of simulation systems able to aggregate massive amount of data with speed and ease, as well as trigger the discovery of common patterns.

Our work moves forward from the experience in [85] and explores Big Data ML techniques aligned with CMS recent developments [106]. Our data analytics and prediction platform is implemented by a pipeline of scalable Spark components fully integrated in the CMS Hadoop data store. Previous work focused on only five physics datatiers, while we propose an online solution for building classification models targeting the entire range of 25 CMS datatiers (thus covering the whole CMS experiment) and keeping them fresh through a viable model update strategy. Furthermore, we discuss in details the features used to train our model, introduce new features involving an improvement of about 10% in the overall classification accuracy, and distinguish the prediction task for new and already existing datasets. Unfortunately, the results of the previous experience are not comparable with ours due to the radical changes in the monitoring infrastructure [106, 107].

Although there are different ML libraries, e.g. the Scikit-learn [116] library for the Python programming language, we pipelined analytics and prediction on the same Spark platform as opposed to earlier results [85] that keep them separate. We also overcome the shortcomings of the experiments in [77], which are limited to the use of very simple prediction models. Our work investigates the use of the classification algorithms at the state of the art and proposes a scenario for the exploitation of the accurate predictions achieved for dataset caching.

2.4.2 Caching Strategy

Data grids and particularly the WLCG have done significant work in the development of advanced data management systems to keep track of data and handle replicas [9, 76, 114, 124]. Research in dynamic data replication strategies has yielded the implementation of a variety of Grid simulators [125, 90, 151, 138, 123], with the Monarc [45, 3] project that was specifically tailored to explore different computing models for LHC experiments. Still, data distribution and deletion strategies have room to expand replication and replacement policies, especially when storage resources are

severely constrained. Current data management services seek to better understand data usage and user behavior harnessing metadata collected from the fabric and connectivity layers, but they are affected by a static intelligence that is based on fixed algorithms unable to learn from experience and does not evolve and improve over time. For example, monitoring information can be used to trigger ad-hoc cancellation of unaccessed replicas and replication of frequently accessed datasets [92, 34]. However, the system is unable to learn from historical data and dynamic data management is simply modeled as a caching problem where most-used data is kept in a cache for faster future access.

A different approach to dynamic data management has gained interest as the area of ML has received a lot of attention lately due the increase of data volumes and commodity hardware [80]. Dataset popularity classification can harness ML techniques and supplement caching algorithms with earned probabilities in order to preemptively balance the system, enhance dataset availability and improve resource utilization. Early work to verify if accurate data popularity predictions would improve caching performance was done in the context of multimedia content caching [61]. The authors found that perfect popularity predictions can give caches a relative performance gain of up to 18% making it a valid tool to use in online content caching.

Over the last three years some of the major experiments at CERN LHC have started their programs for exploring dynamic data management on the WLCG using popularity predictions. CMS in particular aims at predicting the popularity of new and existing datasets to replicate data ahead of time with potentially large impact – by leveraging adequate dataset replacement at remote sites – on resource utilization and data placement. Dataset popularity can in fact be probed to study at what CMS site replicas should be located and thereby improve the dataset access latency and the overall efficiency of any distributed storage system.

Our work exploits the outcomes of the proposed dataset popularity prediction pipeline to feed a novel intelligent caching strategy. A similar approach is discussed in [31] where the LRU policy is extended to make it sensible to Web access patterns extracted using data mining techniques. The authors build a caching strategy called S2 implementing multiclass classification based on decision trees. S2 is applied to a label representing the distance between a given URL and the next access. While improving the baseline LRU, it has a theoretical upperbound ORCL based on the number of requests received until the next access to a given URL, which is in fact only computable from historical data. S2 results in having an accuracy ranging between 70% and 87% on the two workloads tested. In our work we rather simulate two upper

bounds: the theoretical oracle, that always evicts the dataset that will not be accessed for the longest period in the future, and the perfect classifier, that always predicts the actual popularity of a dataset. Our experiments result in accuracy up to 87%, while also discussing why the F1-score represents a more indicative performance metric.

The work in [154] proposes a linear ranking function for replacing the objects in a Web document cache and improves the results in [31]. Rather than extending the LRU policy by using frequent patterns from Web log data, [154] uses a model that extends the Greedy Dual Size Frequency caching (GDSF) policy [36]. A survey of the approaches to data caching based on Web logs was conducted by [59, 8]. Although not very recent, it is interesting to notice that only three works used classifiers as we do. The author in [139] leverages previous results from [154] and extends the ranking function with predictions of the future occurrence frequency of an object based on a rule table, and improves the hit rates of the basic GDSF policy. A similar approach based on rule table is described in [155].

Additional experiments with Web query logs [25] reiterate the importance of estimating in advance queries that are likely to be or not to be repeated in the future. The application of this concept to dataset accesses, and subsequently highlighting the changes in their popularity, may be profitably exploited by the cache replacement policy.

Similarly to the LHCb experiment, we take LRU as baseline for dataset replacement, but we expand the studies by defining a novel caching policy, comparing our results with other state-of-the-art algorithms and arguing how our approach is the most effective. Amongst all, we adapt Static Dynamic Cache (SDC) [60] to our needs in order to exploit temporal and spatial locality that are present in the stream of processed access logs. SDC divides the result cache into a static segment to answer frequently submitted queries and a LRU-based dynamic segment to answer unexpected, bursty queries. Previous experiences [93, 96] in the field are also adjusted to evaluate the contribution of prefetching to anticipate dataset accesses.

We share the same experimental approach of many of these related works by running simulations on increasing cache size with actual trace data. Nonetheless, while the better performance in previous approaches against LRU-based policies is due to their capacity to adapt to data access patterns, we argue that in our setting LRU itself has its own ability to adjust to data locality and we rather enhance this attitude by supplementing the eviction mechanism with the knowledge of future accesses. Moreover, our work addresses a completely different problem – the CMS

dataset accesses – characterized by different data access distributions and a much more complex feature space with respect to Web browsing.

Chapter 3

CMS Analytics

In this chapter, we extend previous CMS attempts [85] to use metadata in the context of dataset popularity and discuss the migration of the analytics techniques to Big Data clusters. To this purpose, we define a data ingestion model to Hadoop, and we validate the analytics platform. We demonstrate its benefits with respect to traditional RDBMS. This analytics data vault becomes the selected environment for our research on CMS dataset popularity prediction and dataset caching described in the next chapters.

The content of this chapter has been presented in [106, 107].

3.1 Setup Environment

To the purpose of this thesis, file access metrics are of particular importance. They are aggregated for different monitoring services such as measuring global data transfer rates through XRootD protocol as well as providing dataset popularity statistics. The Oracle queries that aggregate popularity time series have been in production for several years, but are now showing a non-optimal scaling with the increase of monitoring data.

They become our testbed to quantify future user activity and predict dataset popularity. A growing number of collected metadata about user jobs which resides outside of Oracle DB may provide an additional insight into data utilization by our infrastructure. The next sections explain how we acquire XRootD metrics from the message bus and benchmark their aggregation at the level of dataset popularity, thus proving how dashboard and monitoring systems can benefit from Hadoop parallelism. The entire set of existing Oracle queries is replicated in the Hadoop data store and result validation is performed accordingly.

Currently, the Oracle RAC back-end features 10 GB/s links, 24 TB SATA disks with 2xSAS loop and 512 GB SSD cache per controller. The CERN Hadoop service comprises around 50 nodes equipped with 64 GB of RAM and 32 cores/node on a mix of Intel/AMD CPUs and CentOS7/SLC6 operating systems, 2 PB SATA3 HDD and nearly 1TB of total memory.

The current CMS distributed computing monitoring is modeled as a system able to integrate existing databases, classify and monitor missing sources of information, provide long-term storage of monitoring data and, ultimately, develop analysis tools. We refer the monitoring data coming from this variety of CMS federated XRootD storages [10] as *raw data*. Raw data ingestion and presentation for the new XRootD popularity calculation process is accomplished by a set of Hadoop compliant tools. Amongst all, Flume [47] and Sqoop [149] are streaming utilities that complement each other in shipping data from original CMS sources to HDFS (Fig. 3.1). Flume is a distributed service for collecting, aggregating, and moving large amounts of log data, while Sqoop is a tool for, but not only, transferring bulk data from relational databases to Hadoop.

Flume and Sqoop transfer to HDFS some 10 GB of raw data every day, which includes between 100k and 2M entries in several formats (CSV, JSON [41], Parquet [148], Avro [147]). They are installed via Puppet [84], the standard software distribution tool at CERN IT, and deployed to CMS OpenStack-based VMs. Web notebooks such as Jupyter and Zeppelin [115] are used for inline data inspection and raw data pre-processing with Spark.

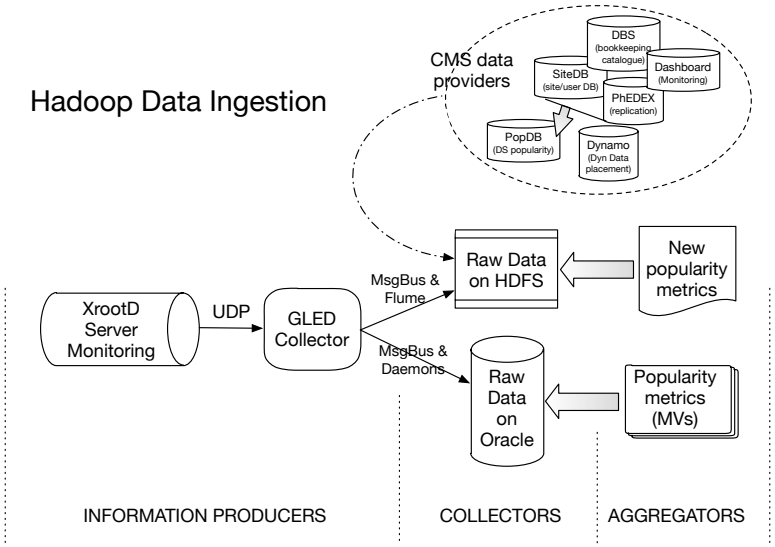


Figure 3.1: Schema of data ingestion into Hadoop.

The Spark framework, via the SparkSQL context, is equipped to read from HDFS any data format and return a dataframe (a dataset organized into named columns) that can be used in forthcoming computations. It is tuned and optimized to reach maximum scalability and parallelism. Every job is calibrated in favor of local processing within the same executor and reducing the data exchange between cluster nodes.

The Spark computing paradigm consists of reading data once and processing them as much as possible in memory, minimizing data shuffling, network communication between executors, and disk I/O operations. In addition, together with the YARN cluster resource manager, Spark jobs can profit from dynamic allocation of their executors, automatically scaling in and out their number when necessary. This allows users to run the same job against folders of the same dataset without adapting different parameters according to their individual sizes.

3.2 Use Cases

This section outlines the CMS analytics use case where significant portions of file access logs were used. We present a successful effort showing utilization of the Hadoop cluster for analytics purposes and modeling of dataset popularity based on available metadata.

3.2.1 Data Ingestion

CMS monitoring raw data is organized into metadata suitable for data-mining. Information of this type contains physics contents (lepton, jets...), processing workflows (Monte Carlo, reconstruction, AOD...), software versions, physics working groups requesting the data, log statistics (number of bytes, accesses, CPU time, individual users etc.), service provenance (SiteDB, dashboard, CRAB, PhEDEx...). Other categories can be derived, like user social activity (field of interests, past activities on the Grid...) or seasonality of the dataset (proximity to conferences or major seminars). Each source has the potential to disclose innovative clues useful to predict future user activity and optimize resource allocation. They are:

- AAA [28], the XRootD-based federation service which holds WAN-access file information from local and remote storage systems.
- EOS [50], the XRootD-based disk-only low-latency storage service located at CERN.

- CRAB [140], the CMS distributed analysis tools for job submission, which holds information about who submits which type of job, where, and what data they run on.
- CMSSW [1], the CMS software framework for simulation, calibration, alignment, and reconstruction of event data.
- Popularity DB and Dashboard, the global monitoring systems which integrate user jobs and dataset-level information, hiding individual backstage sources.
- PhEDEx [127], the data transfer system and location management database that holds data-placement and data-movement catalog information.
- DBS [4, 71], the Data Bookkeeping System providing catalog of event metadata for Monte Carlo and recorded data.

Data-streams (in the form of logs) like AAA, EOS, CRAB or CMSSW contain various metrics associated with a job in question, e.g. name of the file used during a job, user Distinguished Name (DN), processing time of the job, and similar. For example, table 3.1 shows the complete list of parameters extracted from the AAA XrootD federation trace logs, which are copied to Hadoop in multiple formats.

In order to extract insights from the metadata present in these data-streams they should be properly cleaned, sorted, filtered, and joined with data from other sources such as DBS and PhEDEx DBs which provide information about dataset, block and site names. The latter two DBs are quite large, their current size is at the level of a few hundred GB each. Answering simple questions about users accessing a particular dataset on certain sites required to join various attributes from multiple streams along with DB tables from DBS and PhEDEx Oracle DBs.

We found that processing time to join and aggregate data across DBS/PhEDEx DBs and data-streams is impossible to achieve in a reasonable amount of time using a vertical approaches based on RDBMS solutions. A typical job to aggregate information from Oracle DBs and a single data-stream requires many hours to complete using materialized view (MV) across all data. At the same time, simple dump of DBS and PhEDEx DBs into HDFS and standard Spark jobs to join desired data-streams allow for quick (on a scale of minutes) insights into data. For instance, a daily snapshot consists of about 130M log records which are joined at file level with DBS and PhEDEx DBs, grouped and aggregated to roughly a few thousand records in under an hour of

[1] github.com/cms-sw/cmssw

Table 3.1: Metadata for CMS file access logs extracted from AAA XrootD federation and "ingested" into Hadoop.

Name	Type	Description
path	String	full namespace path to the file
ruid	Int	mapped unix user id
rgid	Int	mapped unix group id
td	String	client trace identifier (<username>.<pid>@<client-host>)
host	String	Name of the disk server serving the file
fid	Int	file id
ots	Date	File open time as unix timestamp
otms	Int	Miliseconds part of "ots"
cts	Date	File close time as unix timestamp
ctms	Int	Miliseconds part of "cts"
rb	Int	Bytes read during file open
wb	Int	Bytes written during file open
sfwdb	Int	Bytes seeked forward
sbwdb	Int	Bytes seeked backward
sxlfwdb	Int	Bytes seeked in large seeks (>128k) forward
sxlbwdb	Int	Bytes seeked in large seeks (>128k) backward
nrc	Int	Number of read calls
nwc	Int	Number of write calls
nfwds	Int	Number of forward seeks
nbwds	Int	Number of backward seeks
nxlfwds	Int	Number of large forward seeks (>128k)
nxlbwds	Int	Number of large backward seeks (>128k)
rt	Int	Time in ms spent for actual reading disk I/O
wt	Int	Time in ms spent for actual writing disk I/O
osize	Int	File Size when file was opened
csize	Int	File Size when file was closed
sec.name	String	mapped user name or principal
sec.host	String	Name of client host
sec.vorg	String	client VO
sec.grps	String	client group
sec.role	String	client role
sec.app	String	application identifier

Spark processing time. Monthly statistics can be computed in about 12 hours, which makes it viable to build a series of dashboards that look at data utilization in the CMS experiment.

3.2.2 Data Analytics and Monitoring

There exist different approaches to data monitoring. On one hand, someone can delegate pushing information from individual Grid sites and applications into the central monitoring system. On the other, such information can be obtained from individual logs stored in a common place (e.g. central HDFS) via external jobs. The former approach requires proper instrumentation of monitoring metrics and coordination of data injection from all participating data providers. The latter relies on significant processing power to collect and aggregate information in a reasonable time.

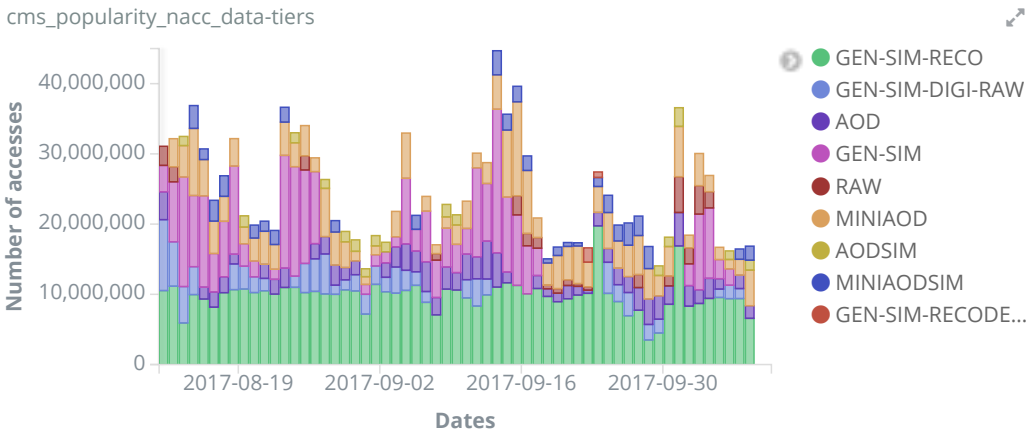
The ability to efficiently process large data sets on HDFS via Spark jobs opens up a possibility to build various dashboards based on user access metrics extracted from various data-streams. To that purpose we use CERN MONIT system [7] based on ElasticSearch and Kibana tools. The data processed by Spark jobs are injected as JSON documents into CERN MONIT via Apache ActiveMQ [137] Stomp protocol. We achieve a reduction factor of 5,000 from metadata records on HDFS that are aggregated and pushed into CERN MONIT. Fig. 3.2 shows one part of the CMS popularity dashboard implemented to monitor the distribution of dataset accesses.

Even though various dashboard plots are useful to get a broader view on daily activities in CMS, we have also been able to address specific questions raised by various users. For example, how much data resides on all regional sites with detailed breakdown of number of events and replica sizes of common used data-tiers.

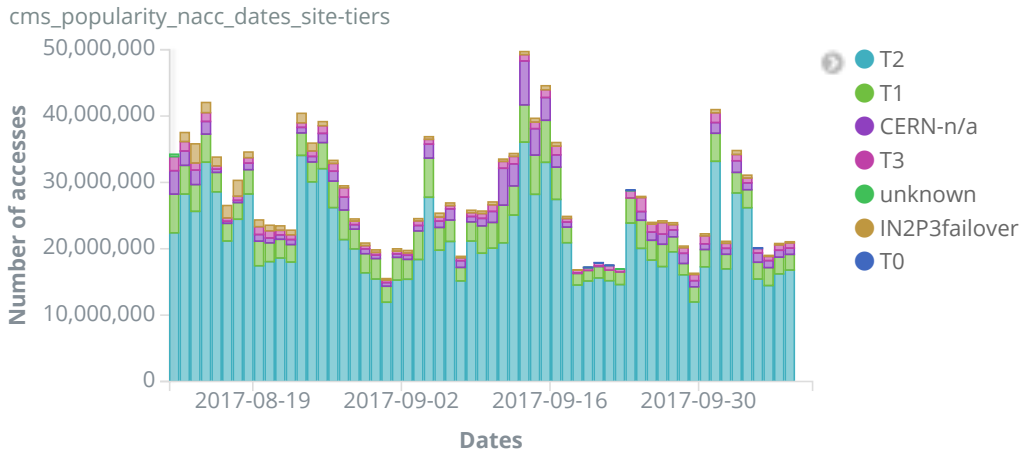
While looking at the current status of resource utilization, which is a very important component of our daily operations, we oversee that it can be further enhanced by leveraging ML techniques for predicting user behavior and adjusting our resources accordingly.

3.2.3 Machine Learning

Application of ML in HEP is mostly used in physics analysis such as discrimination of signal versus background, clustering algorithms, object reconstructions, or similar tasks. Recently, ML algorithms have begun to penetrate into other domains of HEP such as helping physicists to better understand computing resources. For instance, ML modeling has been discussed in the scope of dataset placement optimization and reduction of transfer latencies, as well as in network-aware applications for identifying anomalies in network traffic, predicting congestions and optimizing WAN paths. In next chapters we present results built on top of previous studies for predicting dataset



(a)



(b)

Figure 3.2: CMS popularity dashboard based on processing four CMS data streams: AAA, EOS, CMSSW and CRAB. The displayed part of the dashboard represents the number of accesses vs data-tiers (a) and site-tiers (b) as time series plots.

popularity in context of better data placement strategies. We extend these results by demonstrating how to predict dataset popularity using larger datasets and use them to build strategies for better data placement at participating sites. In fact, dataset placement at most WLCG experiments is based on dataset popularity which should be considered to make an optimal choice of replication to maximize data availability for processing and analysis needs. As shown in [85, 109], CMS dataset popularity can be successfully predicted using ML approach.

3.3 Analytics Results

This section describes how we have replaced the CMS Popularity service based on Oracle with an equivalent service based on Hadoop. The substitution allows to benchmark the speedup and to plan the technology shift towards Big Data. Therefore, we validate the XRootD popularity metadata in Hadoop by comparing them with the outcomes of similar queries from Oracle. XRootD metadata is aggregated on a daily basis following the MapReduce programming paradigm applicable to Big Data architectures. Follows detail of our experience with Pig, Scala and PySpark APIs.

3.3.1 Popularity Queries

We have implemented a number of Pig scripts that extract the values of interest for each file access operation, join them with the dataset names and calculate utilization metrics like CPU time, bytes processed, number of user accesses, etc. Figure 3.3 shows the hierarchy of queries produced from EOS and AAA XrootD sources, with intermediate views representing partial processing in the aggregation chain. About 4 billions log traces at the level of blocks, the smallest units of transferable information among sites, are joined with some 60 millions dataset catalogue entries and partitioned and aggregated in few stages in order to compute monitoring indicators on custom time intervals.

We measure the deltas between popularity metrics computed via Oracle MVs and Pig queries. As shown in Fig. 3.4, they are computed at file block level on three main metrics – number of accesses (a), CPU time (b) and bytes read (c) – and reach an accuracy of nearly 90%. Deltas occur because Oracle and Hadoop metadata are not totally identical: they result from different streamers and are computed via different API, thus the final aggregations may have some limited differences.

Pig scripts yield a significant speedup compared with equivalent MVs used to analyze Oracle data. The Oracle and Hadoop environments described in section 3.1

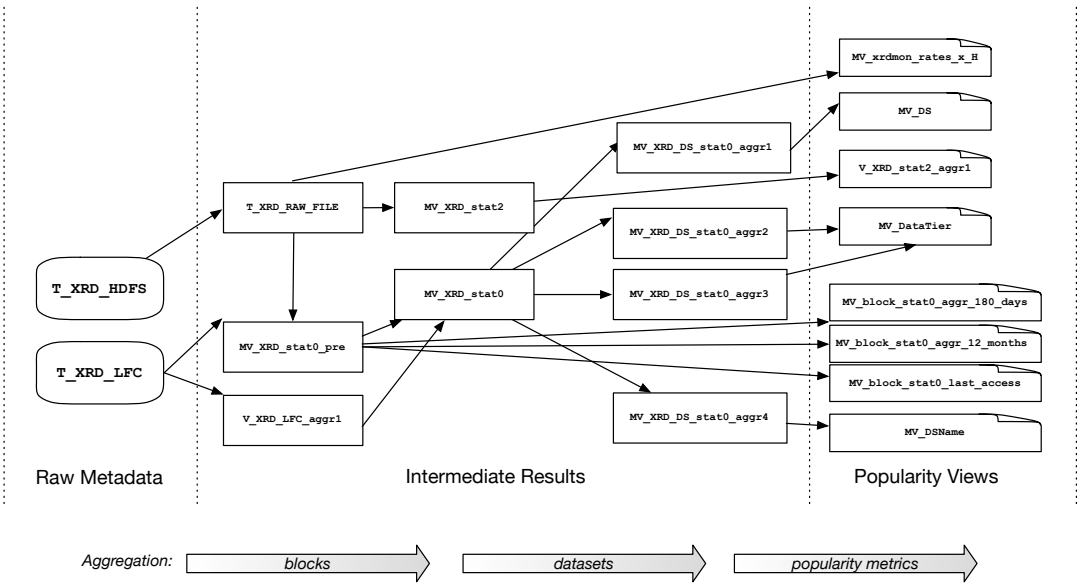


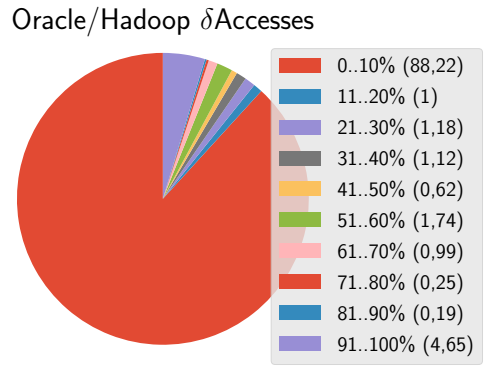
Figure 3.3: HDFS metadata aggregation for EOS and AAA XrootD dataset access logs using Apache Pig.

are our testbeds. The processing time of XrootD time series logs scales better than linearly with data volume, which makes it very effective for quick re-processing of entire months. Speedup factors range between 2x for daily aggregations up to 50x on monthly time-frames.

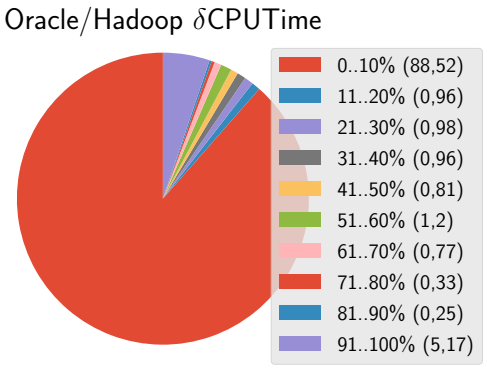
Fig. 3.5 shows the processing time required for a number of dataset popularity monitoring queries. Each data point is averaged on 3 tests. The first plot shows the amount of time, in seconds, that is needed to process nearly 60 GB of file access traces from April 2016. The names on the x-axis are borrowed from the corresponding MVs implemented in the current Oracle-based monitoring infrastructure. The second plot shows the processing time to reproduce the results of a selected Oracle MV (MV_XRD_stat0), being of extreme importance since it aggregates file blocks by dataset name.

Regardless of the Oracle MV and the selected time period, the average processing time remains in a range of few minutes. This makes Hadoop and Pig a very straightforward and effective combination for reprocessing popularity queries. In fact, the same results take continuous running of incremental MV updates in the Oracle back-end, some of which dedicated to serve specific aggregations on highly-requested time intervals (see popularity views by 180 days or 12 months in Fig. 3.3). A complete set of benchmarks on popularity data is demonstrated in [106].

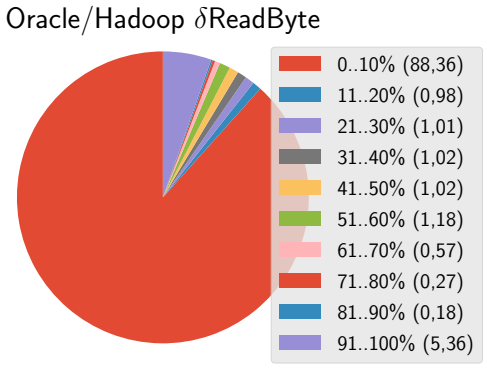
Fast aggregation of access logs outlined in this section is preparatory to monitoring



(a)

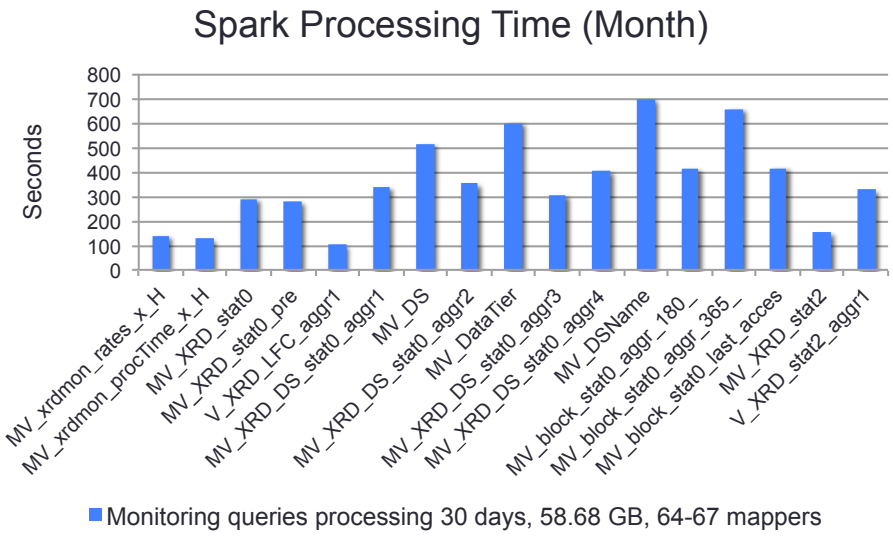


(b)

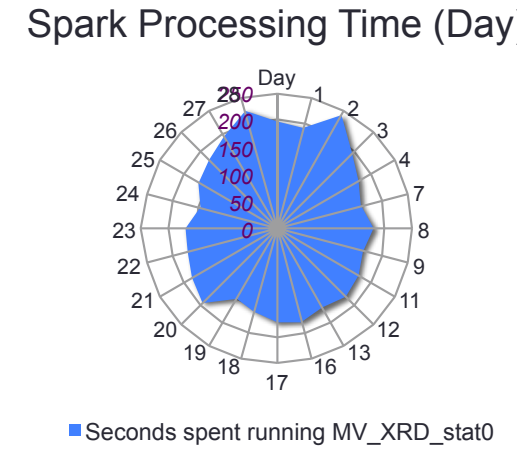


(c)

Figure 3.4: Result consistency between Oracle MVs and Pig queries, with deltas measured for number of accesses (a), CPU time (b) and bytes read (c) aggregated by blocks.



(a)



(b)

Figure 3.5: Fast reprocessing of dataset popularity monitoring query on Hadoop cluster. Plot (a) shows monthly processing time for individual Oracle DB views reproduced by Pig scripts. Plot (b) shows processing time of one core view for daily records. Higher data volume allocates a higher number of mappers.

dashboard or data-frame generator that feeds ML algorithms. The latter one is of highest interest to us and brings to the modeling of dataset popularity described in next chapter.

3.3.2 Performance Boost

Pig does not handle malformed JSON input metadata. Flume - that is used in its default configuration with no optimization - may occasionally truncate some entries. This has made it preferable to implement data aggregation in Spark, which instead offers input data consistency check through its RDD object. In addition, by leveraging in-memory distributed dataset, Spark gives us a significant speed-up when executing the popularity queries.

Spark is designed to process large amount of data efficiently by means of a simple programming interface. The API supports unstructured and structured data. The former works with RDD, Accumulators, and Broadcast variables, while the latter gives access to Dataframes, Datasets and the SparkSQL context. There are few languages which implement Spark API. The native solution is based on Scala programming language. Scala API offers an immediate yet powerful boost to popularity queries thanks to its scalable and production-ready analytics features. A performance comparison on March 2016 traces between Pig and Spark/Scala is shown in figure 3.6.

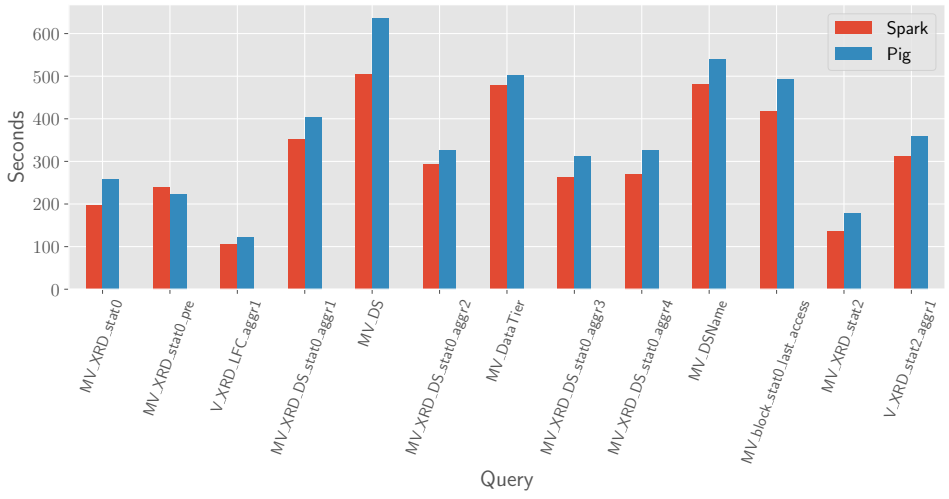
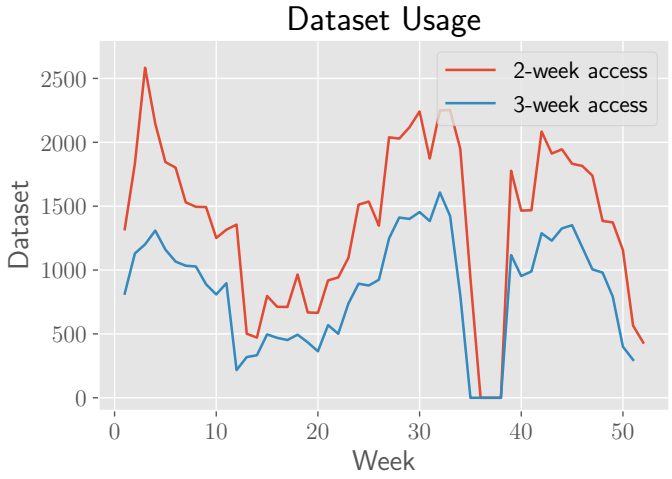
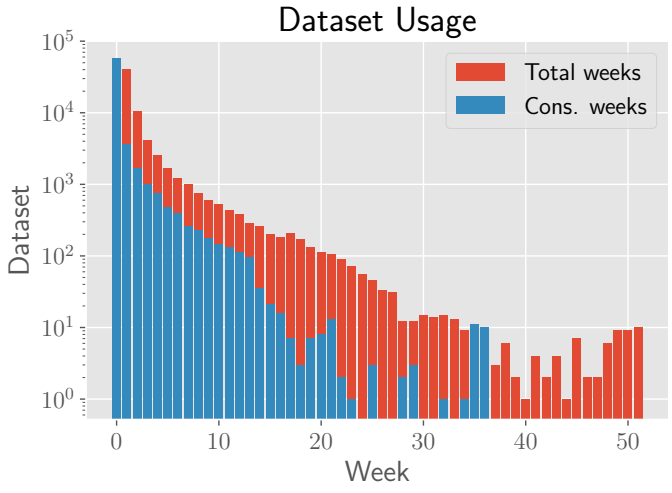


Figure 3.6: Comparison between Pig and Spark while reproducing popularity statistics. Names on the x-axis correspond to Oracle MVs currently deployed to CMS monitoring dashboard. Spark provides 5% to 10% speedup versus Pig on same HDFS data.

The caching and persistence capabilities of Scala allow to plot aggregated monitoring information with ease and speed as raw data is being produced by streamers. In order to exploit built-in distribution and persistence, we relied on *repartition* and *persist* API to split data uniformly across the computing nodes and cache them during the aggregation steps. Fig. 3.7 is an example of this kind that shows dataset usage patterns throughout 2016, which become useful as input for MLlib algorithms available as part of the Scala language. We leverage them in chapter 4 to implement predictive models for dataset popularity.



(a)



(b)

Figure 3.7: Usage of Scala/Spark resilient distributed dataset (RDD) for fast implementation of analytics tasks. The plots show dataset usage patterns over a full year in terms of consecutive (a) and total weeks (b).

3.3.3 Integrated Framework

We found that the Scala programming language is not yet well adopted in HEP community. Therefore, we made significant effort to adapt our code base for popularity queries to PySpark framework, a Python-wrapper solution available as thin layer around Spark libraries.

The transition from Scala to PySpark was trivial, but it is worthwhile to remind that wrapper classes are necessary to submit Python code for execution. Additionally, standard programming paradigms, such as for loops, are useless in Spark distributed computing model and should be replaced with PySpark counterparts to operate over Dataframe objects.

We also experienced a few issues every programmer should know about when developing code using PySpark. Careful crafting of data is required to fit it into worker-node's memory. For instance, most of the runtime errors we experienced were related to Java heap, garbage collector errors and $java.lang.OutOfMemoryError$. A typical example would be the temptation to use $df.collect()$ PySpark function to collect intermediate results. Instead, the code should perform series of operations over the data and either write them out to HDFS or run aggregation steps. Further, when the data are skewed not all workers obtain the right amount of workload. For instance, grouping or shuffling operations of non equally distributed data can cause the slowness of entire job within executor. To take advantage of parallelism the code should not iterate over the containers, instead, optimized PySpark APIs should be used, e.g. $df.foreach$ in favor of $for\ item\ in\ df.collect()$ pattern or $df.treeReduce(lambda\ x, y : x + y)$ instead of local non cluster-distributed functions $np.sum(df.collect())$. Users are advised to rely on PySpark functions to avoid slowness of built-in Python functions over Dataframe operations, e.g. $pyspark.sql.functions.split$ should be used instead of *split* Python built-in function.

To avoid these types of pitfalls, CMS has developed its own framework called CMSSpark [2], which provides all necessary tools for code submission to Spark platform, set up necessary Java libraries for data processing (e.g. support for CSV or Avro data formats) as well as set up proper run parameters optimized for CERN Spark cluster.

This framework is significantly helping new developers to easily adopt their business logic into framework and quickly produce desired results. For instance, any user with access to the CERN Hadoop cluster can quickly launch distributed (Py)Spark jobs to

[2] github.com/vkuznet/CMSSpark

process CMS data, focusing on what operation to run rather than struggling on how to properly submit YARN-based batch job.

Overall, when the problem is reduced and fit into memory of the worker-node we have seen more advantages of using PySpark than Scala due its rich and advanced thrid-party ecosystem, especially Scikit-learn [3] libraries.

3.3.4 Mobile Monitoring

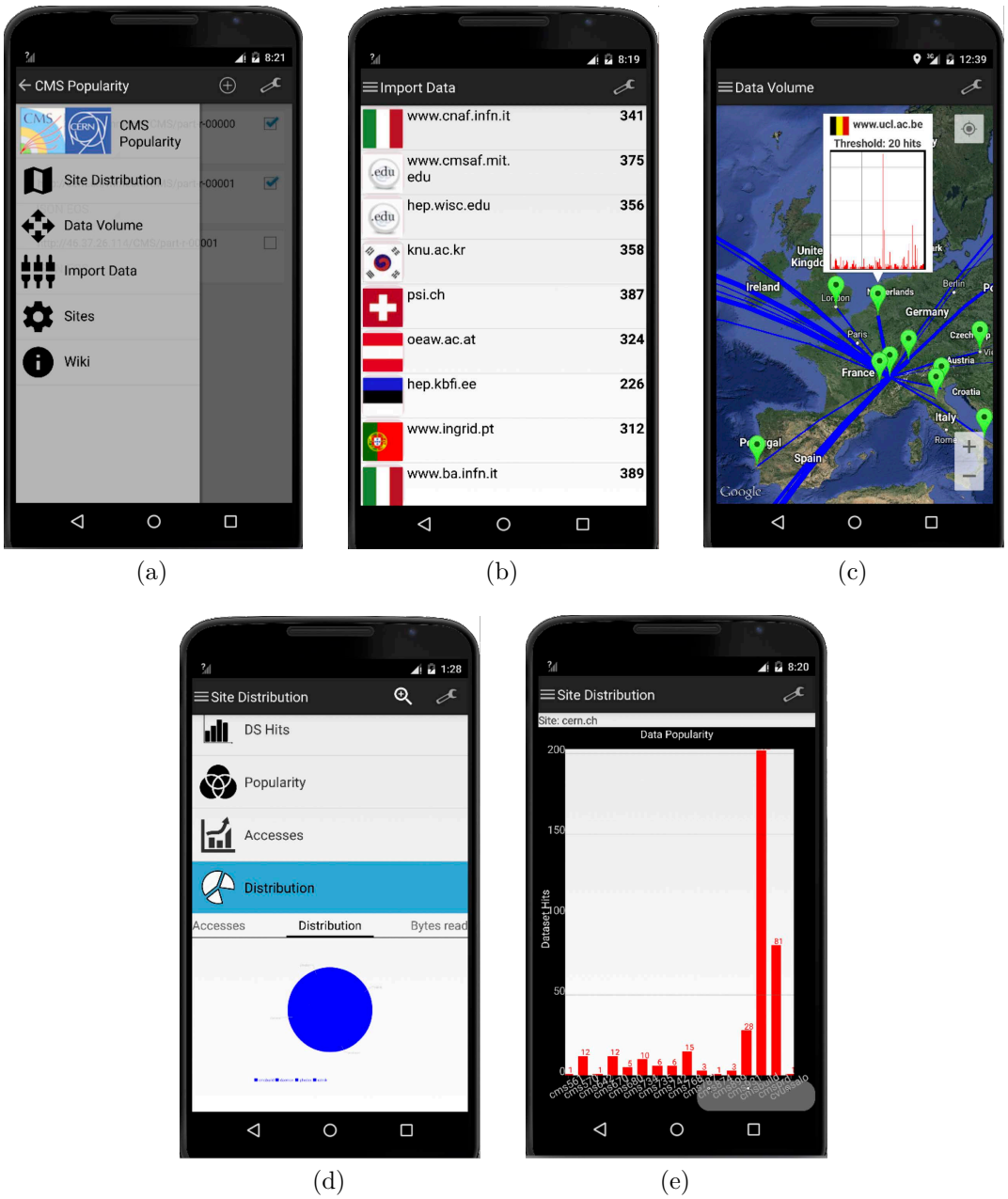
An Android application has been designed to offer a mobile dashboard to dataset popularity. It targets CMS experimental data and its source code is available on GitLab [4]. It presents a variety of popular dataset access statistics aggregated by user and site. Input to the mobile app is provided via REST API built around the results of the MapReduce popularity queries on the Hadoop cluster described in this chapter. It caches monitoring data locally so that dashboards can be displayed offline too. It offers an intuitive visualization based on geographical maps where each Grid site can be selected to display statistics like dataset access distribution by individual users as well as data volume between the given site and CERN Tier-0. At the same time, all application functionalities are summarized by categories in a collapsible drawer menu. Fig. 3.8 depicts a set of windows: the main options of the navigation menu (a); the result of REST client calls with data grouped by Grid site and the record counters (b); the worldwide map that shows the site deployment, highlights the transfer volumes and previews the current load at the selected site (c); the dataset distribution plots (d) (e).

3.4 Summary

Spark has proven to be a very solid framework for data analytics in CMS. It is extremely effective for crunching large dataset available on HDFS and extracting aggregation metrics from underlying metadata. It can be integrated into HEP workflows via PySpark APIs or stand-alone applications written in Scala or Python. We discussed possible pitfalls should be considered using either of the programming languages.

We demonstrated that Spark framework represents a valuable alternative to RDBMS solutions for analytics needs. It allows to (re-)process and aggregate efficiently various metrics from multiple data-streams. We were able to access logs from several CMS sources including AAA, EOS, CMSSW and CRAB and join them with the HDFS

[3] scikit-learn.org
[4] gitlab.com/mmeoni/cmspopularity



(a) (b) (c)

(d) (e)

Figure 3.8: Android monitoring App that displays the results of the MapReduce layer from the Hadoop cluster, maps them by site and produces several dataset popularity usage plots and dashboards.

dumps of two very large databases, DBS and PhEDEx. As a result the data can be easily aggregated into set of dashboards and be used for various monitoring tasks within the CMS community.

We strongly feel that Spark platform will place a significant role in next round of LHC experiments to attack Big Data problems. CMS considers the Spark platform - would it be Scala or PySpark - as a potentially crucial component in the ecosystem of enabling technology that would enable LHC experiments to attack Big Data problems in the long run.

By leveraging the experience with analytics tasks, next chapters present the implementation of a scalable pipeline of Scala components to accomplish mission-critical data reduction and ML processing from billion of records available at CERN HDFS system. It builds ML models that can be used towards data-driven approach in CMS computing infrastructure. Amongst all, dataset popularity predictions can play a significant role in intelligent data placement at Grid sites both for storing newly created datasets and holding dataset samples frequently accessible by CMS physicists. Furthermore, a clever caching strategies based on ML predictions can lead to cost-effective data placement across Grid sites and better utilization of available computing resources.

Chapter 4

Dataset Popularity Prediction

This chapter describes our solution to dataset popularity prediction. It defines the available raw data, the list of features extracted from this data and how the features are used for the prediction task. It motivates the cutoff values used to label each training sample as positive or negative depending on whether the popularity measure is higher or lower than the cutoff. In addition, the choice of the best time interval for dataset popularity prediction is demonstrated. The experimental settings are discussed. We detail the results of the experiments conducted to asses the effectiveness of our solution. The impact of data cleaning and feature engineering steps is also assessed, given that further refinements lead to improvement of the prediction performance. We evaluate the contribution of per-site prediction models compared to a global classifier, and measure the classification performance of our best model on new and existing datasets. Finally, we complete our studies demonstrating the aging effect of a static model versus the accuracy of models that instead updated themselves over time.

In the following we discuss in details the main components implementing the pipeline for data preparation and training of the dataset popularity classifier. The content of this chapter has been presented in [107, 109, 108].

4.1 Data Preparation

Our scalable dataset popularity prediction pipeline is depicted in Fig. 4.1. It highlights the process chain, from the raw data ingestion and preparation steps performed on the CMS Hadoop data store, to the ML component producing, and keeping updated, the machine learned model. This model is in turn exploited by the dataset caching strategy described in the next chapter (termed PPC, Popularity Prediction Cache), which optimizes data placement in the various CMS sites.

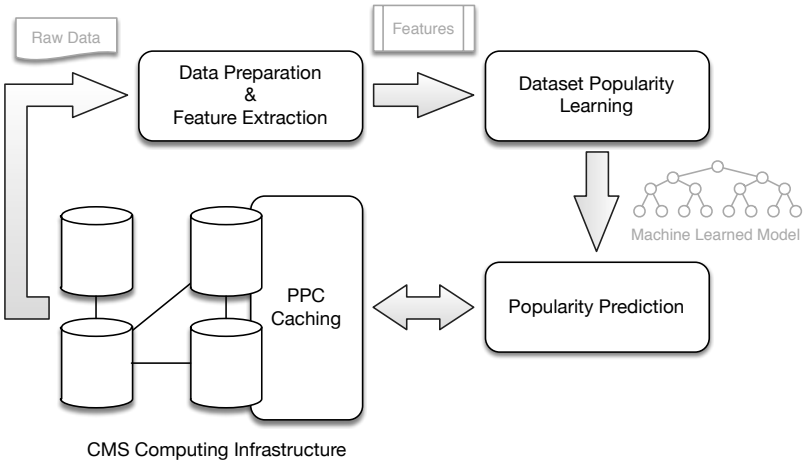


Figure 4.1: The pipeline of components implementing dataset popularity prediction and the utilization, by the PPC caching strategy, of the ML trained model.

CMS dataset access logs are stored in a Hadoop File System (HDFS) and aggregated on a weekly basis as described in the previous chapter. Section 4.5 discusses why this aggregation period is chosen as prediction time granularity. Raw data aggregation, training of the model and evaluation on the test set are implemented via a scalable pipeline of Apache Spark components developed in Scala, a Java binding of Spark. Additionally, Zeppelin is used as Web-based notebook for quick interactive data analytics. It allows for straightforward prototyping of aggregation and learning algorithms.

Popularity raw data on HDFS is stored in CSV, JSON, Parquet or AVRO formats, and is available starting from March 2015. This data represents the output of several streamers performing continuous data ingestion to the Hadoop ecosystem. Table 4.1 lists the main sources of structured and unstructured data feeding feature construction.

Table 4.1: Information sources used to compute the features of dataset popularity samples.

Source	Items	Type	Description
EOS	786,934,116	structured	Disk storage system at CERN
AAA	2,370,570,956	structured	CMS XrootD federation for Grid data
CRAB	1,177,951	structured	Grid infrastructure for job submission
DBS3	5,193,522	structured	Global dataset/fileblocks catalogue
Block-Replicas	805,614,541	structured	Global replica catalogue
PhEDEx	58,227,786	structured	Fileblock locator and export service
CADI	1,791	semi-struct	CMS conference database

4.2 Feature Extraction

In machine learning, feature extraction is the process of selecting, or deriving, informative and non-redundant values from an initial set of measured data. It aims at facilitating the subsequent learning steps and, possibly, interpret sample data better than human. Feature extraction is related to other steps like dimensionality reduction, if some measured data is suspected to be redundant or uninformative, and feature construction, which builds intermediate features from the original input measures. Ultimately, feature extraction leads to a subset called *feature vector*, a vector of numbers representing the value of each feature which is expected to contain the relevant information from the input data. At this point, we can classify dataset popularity by using this reduced representation instead of the complete initial data, which can be too large to be processed by learning algorithms.

Categorical Features

Experimental data at the LHC is organized hierarchically by time-windows called *Run*, a unit of data acquisition or simulation process. A similar structure is simulated for Monte Carlo events. During each Run, either it is related to LHC collisions or Monte Carlo simulation, data is organized into sets called *blocks*, the smallest units of transferable information among sites. Data consists of mostly ROOT-format [32] *Logical File Names* (LFN), a site-independent name for a file. In turn, blocks are grouped in datasets, which constitute the entry point for data analysis and data transfers. On average, a dataset has a size of about 2 TB and includes from a few tens to some thousands LFN.

A typical CMS dataset namespace is defined by three main parts concatenated in a slash-separated name `/primaryDataset/processedDataset/dataTier`, as it is shown in Fig. 4.2. However the syntax is not fully enforced and this constitutes a problem for automated parsing tools. `primaryDataset` is a string that describes either the selection process on real data or the physics simulated event types. `processedDataset` describes the processing chain that is applied and the data taking era. `dataTier` (or, simply, *tier*) describes the kind of event information stored from each step in the processing simulation and reconstruction chain. Examples of tiers include RAW and RECO, and for Monte Carlo event, GEN, SIM or DIGI or a combination of them. For example the GEN-SIM-DIGI-RECO dataTier includes the generation (Monte Carlo), simulation (Geant), digitalization and reconstruction steps.

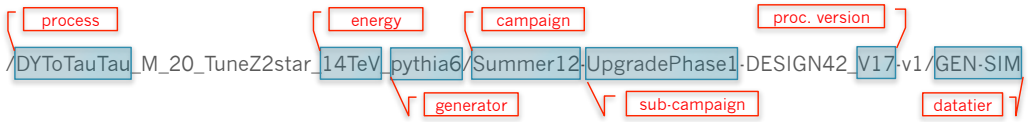


Figure 4.2: Most representative attributes that can be extracted from the dataset name.

The building blocks of the dataset name constitute a first set of physics domain features that we harness to run ML algorithms. Additional categorical features are derived from the infrastructure domain and include network property on both the client and server sides of the dataset access requests, as well as username and protocol type. All categorical features are listed in Table 4.2 and "hashed" when stored in the feature vector.

Table 4.2: Categorical training features extracted from the set of possible attributes.

Type	Name	Description
Physics domain	process	group of events with related topology
	energy	energy at which collisions take place at LHC
	generator	software for events generation
	campaign	data reprocessing and simulated events production phase
	subcampaign	a subphase of a given campaign
	version	version of the processed data
	dataTier	type of event information stored in the dataset
	software	reconstruction software
	era	acquisition Era
	luminosity	number of collisions produced in a time and space unit
Infrastructure	serverdomain	domain satisfying the access request
	serversite	server name
	servercountry	country the server belongs to
	clientdomain	domain of the client requesting the access to the dataset
	clientsite	physical site of the client
	clientcountry	country the client belongs to
	username	username accessing the dataset
	protocol	application protocol used to access the dataset

Numeric Features

Numeric values from the input data take to the identification of a second set of relevant features. They are two-fold: static attributes that are derived from dataset characteristics, like the size or the number of files and blocks it contains, and usage

metrics that result from weekly combination and aggregation of usage information such as number of accesses, number of bytes read, number of users, CPU time, and subsequent derived measures. Table 4.3 summarizes this second set of features.

Table 4.3: Numerical training features extracted from the set of possible attributes.

Type	Name	Description
Static features	week	week of aggregation
	size	size of a dataset, expressed in GB
	nfiles	number of LFN in the dataset
	nblocks	number of blocks in the dataset
	nevents	number of collisions described in the dataset
Usage features	naccesses	number of weekly accesses to a dataset
	nusers	number of weekly users accessing a dataset
	cputime	CPU time weekly utilized to access a dataset
	readbytes	weekly number of bytes read, expressed in GB
	nreplicas	weekly number of replicas for a dataset
	nconferences	number of conferences where the dataset is mentioned
	μ(naccesses)	average number of weekly accesses
	σ(naccesses)	variance in number of weekly accesses

A portion of the raw data and a set of the weekly samples are publicly available[1] in SVMlight format in order to make results reproducible and foster knowledge and improvement in the field. Datasets and user names from the input samples are already converted into numeric values for anonymization purposes.

4.3 Modeling Dataset Popularity

The definition of popularity is all but unambiguous. Informally speaking, we can say that dataset popularity quantifies the user activity. A dataset is popular when it is used ”often” in user jobs. Understanding whether a dataset is popular or not helps to answer questions like what new dataset will be accessed the most or how many replicas of them is convenient to store and where these replicas should be located in the CMS distributed infrastructure. This would improve the efficiency of any distributed storage system because dataset access can be optimized creating several replicas at the sites where they are most likely to be used.

The relationship between the input features listed in Tables 4.2, 4.3 and the binary class to be predicted (popular/unpopular) is primarily related to the dataset usage

[1] github.com/mmeoni/CMS-popularity

metrics like *naccesses*, *cputime*, *nusers*, *readbytes*. In fact, different cutoffs applied to the values of these usage features, or to their combinations, can lead to different definitions of popularity, with each dataset being designated as popular or unpopular depending on whether the resulting value is higher or lower than the cutoff.

Since the replica ratio measured from the dataset catalogue at CERN is roughly 25%, we start our studies from a realistic threshold that already satisfies the current storage deployment. Fig. 4.3 shows the distribution of number of datasets versus four utilization metrics, and highlights the corresponding popularity thresholds. The x-axis is displayed in log scale because all four distributions are skewed to the left and would have very long tails extending to the right.

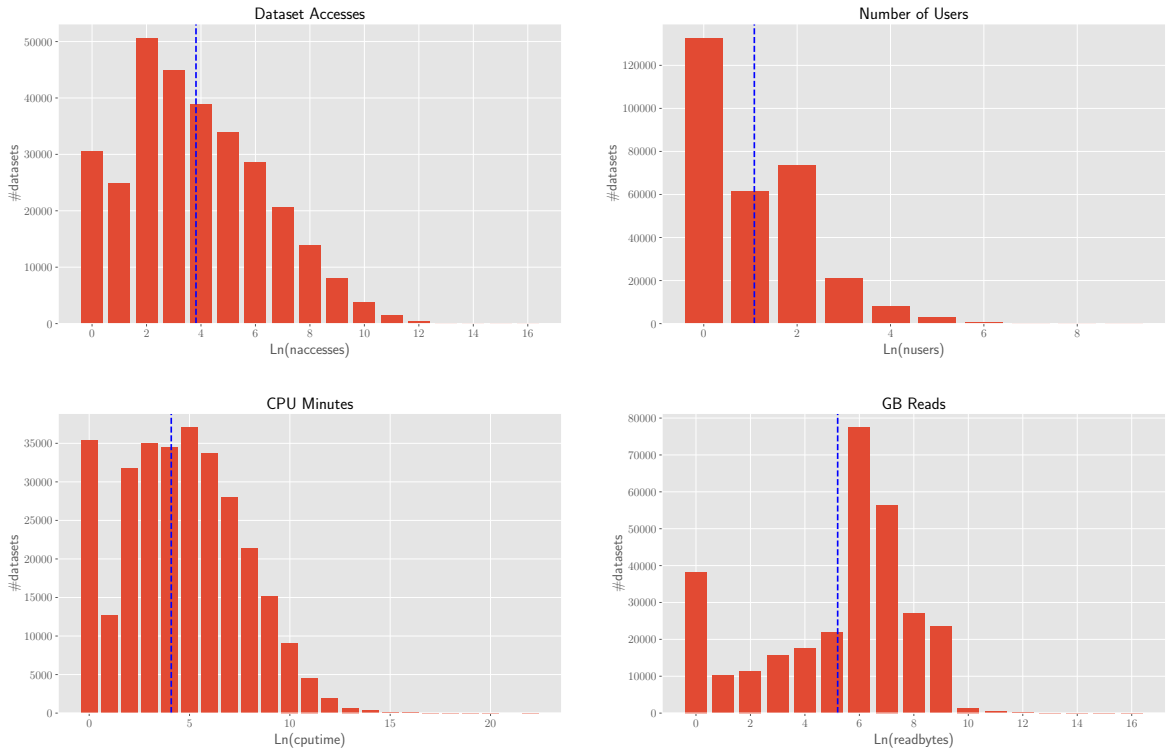


Figure 4.3: Distribution of several usage features and possible popularity cutoffs: number of datasets versus, in log scales, number of accesses, users, CPU-minutes and GB read.

The application of the replica ratio results in the cutoffs listed in Table 4.4. They can be, in turn, used to convert the regression problem of predicting the number of future accesses to a dataset into the related binary classification problem of predicting if a dataset will be popular in the future.

We leverage Spark and its scalable ML library for training the predictive models

Table 4.4: Popularity cutoffs. Different thresholds applied to the usage features can lead to different definitions of popularity.

Usage feature	Cutoff
naccesses	50
cputime	1 h
nreplicas	1
nusers	4
readbytes	400 GB

that will suggest what datasets become popular over time. Spark has proved to be very successful as an effective platform for running CMS computing analytics [107]. Hence, it can be used as a tool to streamline and compare different predictive prototypes capable of gathering input samples that organize features extracted from several CMS data services. Scalability and fast distributed data processing are two critical factors for current CMS data analyses; Spark offers both, together with a simple programming abstraction through the Scala language, which in turns provides powerful caching and persistence capabilities. This allows us to compute feature vectors in nearly real time as the raw data is being produced, with the addition to pass them on directly to the MLlib algorithms available in Spark.

For the purpose of this work, we test a set of standard ML baseline widely-used [128] algorithms (Decision Tree, SVM, Logistic Regression) and two state-of-the-art algorithms based on decision trees: Random Forest (RF) [141] and Gradient Boosted Trees (GBT) [68]. The hyper-parameters used to train the classifiers are chosen according to the recommendations in the MLlib developer guide[2]. In modern applied machine learning, models based on tree ensembles like the ones learned using RF and GBT algorithms almost always outperform singular decision trees or simpler models [44, 62]. Moreover, they result to be much more robust to overfitting. RF is an ensemble learning boosting meta-algorithm that trains a set of decision trees by exploiting sampling with replacement on both the features and sample space. GBT instead uses any arbitrary differentiable loss function to drive the iterative learning of new decision trees minimizing the error due to incorrectly classified examples. Although there is substantial variability in the performance measured across problems and metrics from different experiments and domains, GBT performs generally better

[2] spark.apache.org/docs/1.5.2/mllib-classification-regression.html

than RF, particularly when dimensionality is low [35]. This is also evident by the fact that GBT is the most common choice in solutions for ML competitions, such as Kaggle[3].

In binary classification, accuracy may not be the best estimator to use [157]. It may sound like a key metric as it measures the ratio of correct predictions to the total number of cases evaluated, but a high accuracy is not necessarily an indicator of high classifier performance. Guessing the more common class could in fact yield very high accuracy in presence of unbalanced classes. It is usually preferable to use different metrics that are less sensitive to imbalance and are more effective at identifying the members of the positive (rare) class successfully, which is represented by popular datasets in our study.

Thus, we consider two additional performance measures: *precision*, which measures the classifier correctness, and *recall*, which measures the classifier completeness. Precision expresses what proportion of predicted popular datasets is actually correct. A model that produces no false positives has precision 1.0. Recall (or sensitivity) describes what proportion of actual popular datasets is correctly identified. A model that produces no false negatives has recall 1.0.

Because they both provide valuable information about the quality of a classifier, they are further combined into the single general-purpose *F1-score*, which is defined as their harmonic mean. The F1 score favors classifiers that are strong in both precision and recall, rather than classifiers that emphasize one at the cost of the other.

We finally combine true and false positive rates in the *Receiver Operating Characteristic* (ROC) curve. The area under this curve represents the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative one: the larger the area under the ROC (auROC), the better the discrimination power of the predictive model.

4.4 Popularity Class

The set of dataset popularity samples built from the sources described in Sect. 4.1 contains instances of both unpopular and popular dataset in the week observed. A dataset is labeled as popular according to the cutoffs in Table 4.4. Specifically, each dataset weekly sample is initially modeled by the 31 features listed in Tables 4.2, 4.3 and one additional label (0/1) that assesses the popularity of the dataset in the week successive to the one the sample refers to.

[3] www.kaggle.com/competitions

We follow the process discussed in section 4.3, compute the popularity metrics and derive the ROC curves. Fig. 4.4 compares the auROC when cutoffs on number of accesses and CPU time are applied. In the tests conducted we observed that the cutoff on the number of weekly dataset accesses ($naccesses > 50$) results in a better auROC than the cutoff on other metrics such as *cputime* (plotted, as example, in the same figure).

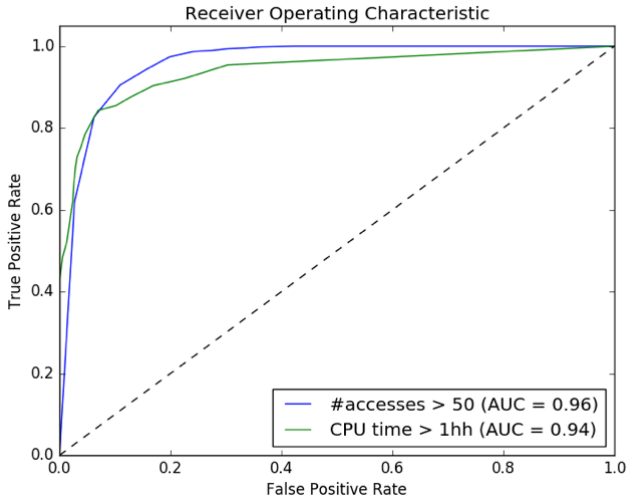


Figure 4.4: ROC of the popularity classifier with different cutoffs (50 accesses and 1 hour CPU time).

Similar tests are performed with the other cutoffs in Table 4.4. Replica counts (*nreplicas*) perform the worst. This is reasonable since one goal of this work is to replace the current reactive model, which monitors historical data of dataset usage and computes the number of replicas *post-factum*, with a proactive model trained on selected features that is able to predict dataset popularity. On the other hand, combined cutoffs do not clearly perform either worse or better than individual ones. Rather, the percentage of false predictions vary irregularly. Ongoing studies have the goal to verify whether an increase of the complexity of cutoffs can bring better accuracy.

For this reason, *naccesses* is the selected metrics for all the next experiments.

Table 4.5 details the characteristics of the dataset used for learning and assessing the classifiers for dataset popularity. In this dataset each sample is initially represented by the 31 popularity features. In section 4.7.1 we will show how additional features are introduced to enhance prediction accuracy.

Table 4.5: Characteristics of the dataset used for training and testing.

Number of samples	307,025
Positive samples (popular datasets)	68,661
Negative samples (unpopular datasets)	238,364
Number of datasets	47,775
Number of CMS sites	63
Number of CMS tiers	25
Timespan (in weeks)	105

4.5 Prediction Timespan

We analyze the input samples to determine what time interval represents the best timespan for dataset popularity prediction. The dataset popularity labels used during the training are computed by exploiting the complete knowledge available in the actual historical data. Our classification task aims in fact at predicting if a given dataset will be popular in the next future. Since the historical data provides a complete knowledge of what happened in the past, we use this knowledge as an oracle for building a large set of training samples. A training sample has the form $< id, t, f_1, f_2, \ldots, f_n, Y >$, where id is the identifier of the dataset, t is the discrete time at which the features $f_1, \ldots, f_n$, refer to, and Y is the binary popularity label indicating if dataset id will be popular or not at time $t + 1$. Note that we produce several samples for each dataset, one for each pair of time intervals $(t, t + 1)$ covered by our historical data. The value of label Y for dataset id at time t will be 0 (*unpopular*) when the number of accesses to id at time $t + 1$ is lower than the threshold, 1 (*popular*) otherwise.

Thereinafter we consider a single week as the timespan for our popularity prediction problem. We choose a weekly timespan justified by the empirical observation that when the temporal window in the training set is increased for example to 3 weeks (i.e., dataset popularity in $week_{i+2}$ predicted using the features computed from data collected in $week_i$ and $week_{i+1}$), the number of positive samples decreases remarkably and prediction accuracy decreases.

The two upper plots in Fig. 4.5 display the number of datasets that have been accessed in 2 and 3 consecutive weeks, while the two distributions in the bottom of the figure show the *total* number of weeks each dataset is accessed and the number of *consecutive* weeks each dataset is accessed. From these plots we see that most datasets have a relatively short access pattern and increasing the timespan does not allow to

capture their dynamics.

Figure 4.5: Number of dataset accesses vs. number of total and consecutive weeks, and number of accesses in 2 and 3 consecutive weeks.

On the other hand, it is not desirable to shorten the timespan. This would blow up the number of training samples and their refresh rate when a sliding window approach is applied. But, more importantly, it is not actually necessary given the current dataset production rate. In fact, over 47k different datasets have been produced since early 2015, which makes it an average production of nearly 60 new datasets every day on a Grid infrastructure counting about 70 sites. In other words, there would be more sites than the number of datasets produced on a daily basis.

4.6 Model Training

In our study we use the observations from the two-year period 2015-2016 and define two possible categories to be predicted, the classes 0 (unpopular) and 1 (popular). A dataset is considered to be popular if accessed more than 50 times in the given week.

The features are transformed and put into feature vectors. Non-numeric features are converted to numeric values using hash tables. Next we create an RDD containing arrays of labeled points. A labeled point is a class that represents the feature vector and the label of a data point. Labeled points are split to get a good balanced proportion of popular and unpopular datasets and training data set and test data set are built accordingly.

The model is trained by making associations between the input features and the labeled output associated with those features. Several models from the MLlib package are trained. Finally, we use the test data to get predictions. We compare the prediction of popular datasets to the actual popularity class and derive performance metrics for the given model. The overall training chain is depicted in Fig. 4.6, from feature extraction to model evaluation. The choice of the best model is based on performance estimators, and it is graphically represented in terms of seeking for the optimal separating (hyper)plane between popular and unpopular datasets.

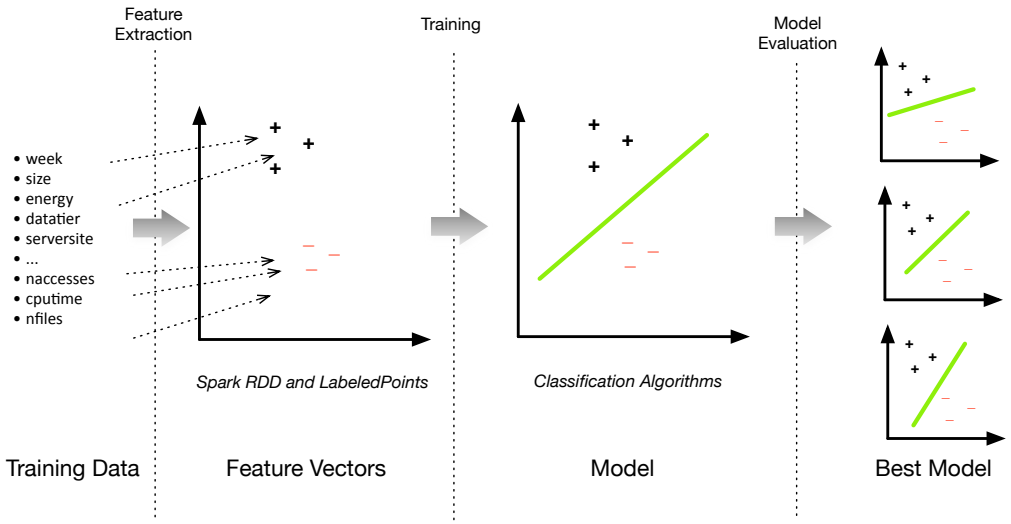


Figure 4.6: Schema of the feature extraction, training and model evaluation steps.

4.7 Prediction Performance

Table 4.6 summarizes the performance of models learned with different ML algorithms when the cutoff $naccesses > 50$ is used to establish the popularity label for the week successive to the one when the features in each sample are computed. The feature vector is applied to all models. Models are trained using Spark MLlib machine

learning algorithms on more than 300,000 samples derived from dataset accesses in 2015 and 2016. The measures of the tests' accuracy are used as baseline for a series of enhancements that will be discussed in Sect. 4.7.1.

Table 4.6: Performance of various classifiers for $naccesses > 50$

Classifier	auROC	Accuracy	Precision	Recall	F1
Decision Tree	0.647	0.743	0.737	0.742	0.740
SVM	0.660	0.740	0.719	0.743	0.716
Logistic Regression	0.645	0.720	0.698	0.717	0.650
Random Forest	0.744	0.758	0.782	0.757	0.769
GBT	**0.773**	**0.769**	**0.792**	**0.770**	**0.781**

The ensemble methods in our setup confirm to be more precise than base models. Consistently with the literature, *Random Forest* and *GBT* models significantly outperform *Decision Tree, SVM,* and *Logistic Regression* models learned on the same dataset. The performance measures of the best classifier (GBT) are highlighted in bold font.

4.7.1 Enhancement of the Prediction Model

In the following we discuss how the prediction performance reported in Table 4.6 can be improved by further pre-processing the data and introducing additional features. The strategies tested for improving the performance of our best classifier (GBT) are described below:

- **Removal of Unpopular Tiers (RUT)**.

  As already described, LHC event information from each step in the simulation and reconstruction chain is logically grouped into what is called a tier. Fig. 4.7 shows the popularity distribution of the 25 available tiers. Among these 25 types, only the 12 of them that are more used and more important from the physics point of view are retained in the dataset. In fact, the log scale on the y-axis emphasizes the presence of several tiers that are scarcely used. The tiers filtered out are the ones associated to testing jobs and the ones that are utilized in a short timespan. This filtering produces a cleaner training set and improves the precision of the model and the F1 score by 4-5% as shown in the row labeled GBT_{RUT} of Table 4.7.

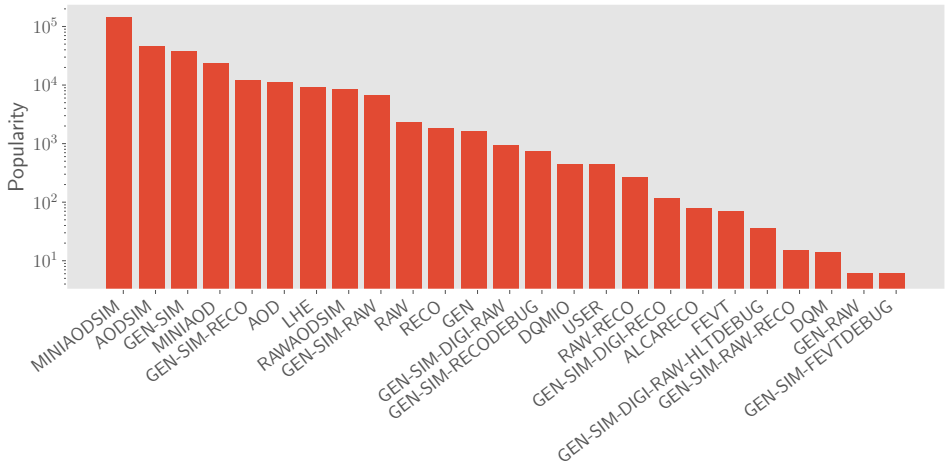


Figure 4.7: Popularity distribution of the 25 LHC tiers at CMS.

- **Rank-based features (RBF)**.

  The analysis of the distribution of numerical features such as *cputime*, *naccesses* or *readbytes*, shows how their range is subjected to significant fluctuations. For examples, the number of weekly accesses to a dataset can vary from few units to tens of thousands. Even larger is the range involving the total number of read bytes: while a Monte Carlo simulation job is mostly CPU bound, an analysis job can perform a large amount of reads. No matter what the measure is, usually there exists a small amount of values that can be seen as "outliers", while the values above the popularity threshold are affected by scattering. In order to reduce variance we sort the numerical features in all the samples by increasing values and substitute in each sample the real feature value with its rank in the global order [97]. While low values do not incur in remarkable transformation, the scattered instances occurring for high and rare values become compacted. The introduction of rank-based features has the advantage of further improving the classification accuracy as shown in the row labeled $GBT_{RUT+RBF}$ of Table 4.7.

- **Daily-trend features (DTF)**.

  The feature set is extended with the number of accesses on each weekday and, for each day, a (-1,+1) column is added to indicate whether the number of accesses decreases or increases with respect to the previous day. Furthermore, also average and variance of the number of accesses within each week is computed. Consequently, the number of numeric features in the dataset is more than

doubled. The introduction of these daily trends among the features results in a significant improvement of the predictive power of the classifier which reaches a precision above 0.88 as shown in the row labeled $GBT_{RUT+RBF+DTF}$ of Table 4.7.

Table 4.7: Incremental improvements of classifier performance achieved exploiting RUT, RBF, and DTF strategies.

Classifier	**Precision**	**Recall**	**F1**
GBT	0.792	0.770	0.781
GBT_{RUT}	0.820	0.842	0.831
$GBT_{RUT+RBF}$	0.836	0.848	0.842
$GBT_{RUT+RBF+DTF}$	**0.881**	**0.870**	**0.875**

When comparing different methods, we also evaluated statistical significance by running the paired student's t-Test [136]. The test returns a p-value ≤ 0.05 when rank-based and daily-trend features are used, thus confirming that the difference in performance among the methods is statistically significant.

4.7.2 Over- and under-sampling

Our classification problem is characterized by unbalanced data because the unequal number of instances in the prediction classes. Having imbalanced datasets is a very common situation in different application fields, e.g., disease analysis from health data, fraud detection, web search ranking. Machine learning classification algorithms are typically sensitive to unbalance in the predictor classes. Trivially, if we assume 10% popular vs 90% unpopular datasets, a machine learning model trained and tested on such data could predict the popular class for all samples and still score a 90% accuracy. That is, an unbalanced dataset will bias the prediction model towards the more common class.

Under-sampling and over-sampling are two common approaches for balancing unequal sets [69]. With under-sampling, a subset of samples from the majority class is randomly selected to match the number of samples of the minority class. Disadvantage is that potentially relevant information from the left-out samples are lost. With over-sampling, samples from the minority class are randomly duplicated (or generated

from the available data) in order to match the number of samples in the majority class. Information is not lost, but there is a risk of overfitting the model.

We complete our set of enhancements of the prediction model by running another round of experiments with a balanced set, and compare the performance. In addition to the two above random resampling methods, we also run two popular hybrid algorithms to oversample the minority class: Synthetic Minority Oversampling Technique (SMOTE) [37] and Adaptive Synthetic (ADASYN) [75]. SMOTE and ADASYN have the ability to generate an arbitrary number of linearly-interpolated minority examples. They shift the classifier learning bias toward the minority class, with significant performance improvement compared to random sampling. ADASYN in particular, which is an extension of SMOTE, creates more examples in the proximity of the boundary between the two classes and adaptively shifts the classification decision toward the difficult examples.

Since our original data is unbalanced, predictions will hence be biased towards the most popular negative class. This means our $GBT_{RUT+RBF+DTF}$ model will be more likely to predict a dataset as being unpopular than popular, leading to more frequent false negatives occurring. Considering that precision is inversely proportional to the number of false positives, less positive predictions explain the relative decrease in less false positive predictions, and why precision is higher in our model.

Table 4.8: Re-sampling unbalanced data.

Method	**Precision**	**Recall**	**F1**
Undersampl.	0.850	0.839	0.844
Oversampl.	0.844	0.847	0.845
SMOTE	0.864	0.866	0.864
ADASYN	**0.870**	**0.871**	**0.871**

On the other hand, Table 4.8 shows how the different sampling techniques can influence model performance. Random sampling methods worsen precision compared to $GBT_{RUT+RBF+DTF}$ trained on unbalanced sets. This reduction is explained by the more weight that resampling puts to the minority class, thus making the model bias to it. In fact, we have measured a 2.5% overall increase of true positives and a reduction of false negatives by 6%. The model will therefore predict the minority class with higher precision, the overall precision will decrease though. Consequently, we see how the recall value, which is the number of correctly predicted popular samples divided by

their total number, is usually higher than precision. Less negative predictions explain the relative decrease in less false negative predictions, and why recall is higher when resampling. This is very important because we want to know how well the model can specifically classify the popular datasets.

We can conclude that ADASYN requires more processing time than SMOTE (and in our experiments leaves a slightly higher number of examples in the over-sampled class) but results in overall improvement of both recall and F1 score, which is a common behavior whenever smart strategies are applied in place of random approaches [83].

4.8 Site-level Models

Since the CMS infrastructure is distributed we assess here the opportunity of training a specific classifier per each site. In fact, some locality could exist in dataset accesses that make some items most frequently accessed from some sites rather than others. Similarly, there may exist certain combinations of Primary Dataset or Era description in the dataset full name (see Fig. 4.2) that are more frequent for specific sites. Fig. 4.8 shows the access distribution at CMS sites having at least 2,500 accesses in the two-year time window chosen for training the models. The plot includes 31 sites out of the overall 63 sites offering dataset storage capability in CMS.

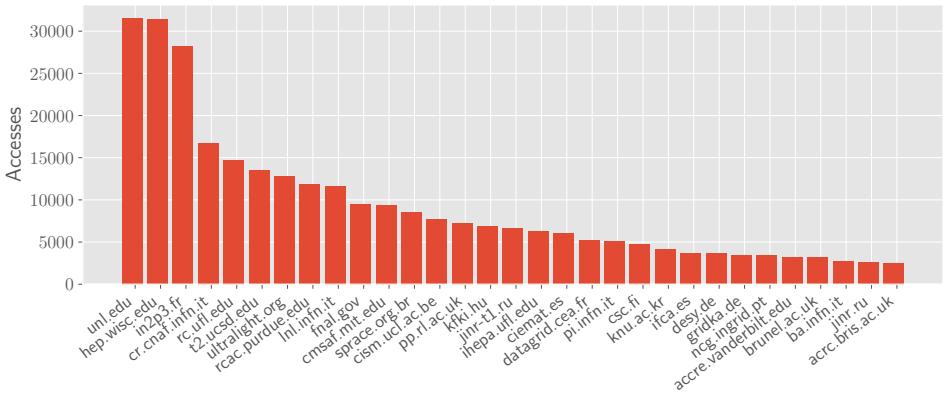


Figure 4.8: Number of training samples at the most active sites of the CMS infrastructure.

Table 4.9 lists the performance achieved by single-site classifiers learned from the samples referred to 6 specific sites of the CMS infrastructure. These 6 sites are chosen by first arranging the CMS nodes in three groups on the basis of the number of weekly samples recorded in the training dataset: more than 30k samples, between 30k and

10k samples, and less than 10k samples. Then, two sites from each one of these groups are randomly chosen. This strategy allows us to have a small set of sites that can be considered representative of the large diversity among the CMS centres. The first line of Tab. 4.9 summarizes the prediction metrics of the global model when it is scored using a test set that includes all sites. Instead, the two groups of six rows with individual site names show the performance of the global model and single-site classifiers. In the the latter case, the number of samples in the training sets is also specified.

Table 4.9: Performance of local site-level models and global classifier.

Site	Samples	Precision	Recall	F1
All Sites	296,360	0.881	0.870	0.875
`hep.wisc.edu`	296,360	0.805	0.800	0.802
`datagrid.cea.fr`	296,360	0.889	0.778	0.830
`unl.edu`	296,360	0.867	0.877	0.872
`in2p3.fr`	296,360	0.947	0.933	0.940
`cr.cnaf.infn.it`	296,360	0.900	0.833	0.865
`pi.infn.it`	296,360	0.900	0.800	0.847
`hep.wisc.edu`	30,487	0.845	0.843	0.844
`datagrid.cea.fr`	4,955	0.857	0.714	0.779
`unl.edu`	31,021	0.870	0.862	0.866
`in2p3.fr`	27,123	0.968	0.945	0.956
`cr.cnaf.infn.it`	16,394	0.900	0.833	0.865
`pi.infn.it`	4,945	0.901	0.801	0.847

By looking at the figures reported in Table 4.9 we can note that the variance in the number of training samples results in different precision and recall values of the resulting local models. A few insights can be derived.

- Small sites having a low number of samples lead to generally poor classifier performance.
- Only the local classifier at the `in2p3.fr` site outperforms the global model, with an impressive recall of 0.945 when the local model is scored on the local test set. This is however not due to a better performance of the local model on this site, but rather to its particular (and easier to forecast) distribution of accesses. In fact, on a separate test we also scored the global model on the same local test set and obtained a recall of 0.943, far higher than the average recall of the global model (i.e., 0.870).

- Global and local models tested at two sites, `pi.infn.it` and `cr.cnaf.infn.it`, result in the same performance. In fact, the number of predicted popular and unpopular datasets is equivalent. This result is unrelated to the number of samples, with the `pi.infn.it` site having a very small training set while the `cr.cnaf.infn.it` site is instead characterized by a relatively high number of samples.

- The global model performs usually better than the local models. This result demonstrates that there is no clear advantage of adopting a local model instead of a global one learned on the samples from all sites.

The findings from the experiments in this section lead to the conclusion that the presence of a large number of different sites, many of them with relatively limited amount of training samples, makes it preferable to use a more robust global model instead of single-site classifiers. This conclusion also reflects the CMS dataset distribution approach, which is generally site-agnostic (within the same level in the 3-tier hierarchical structure) and tends to have a uniform distribution. In the future many small to medium sites are also expected to move to cache-only storages filled on demand.

4.9 Seen and Unseen Datasets

In this section we assess the performance of our prediction model to classify the popularity of new in contrast with previously seen datasets. The traces of dataset accesses available permit to extract a vast range of usage features convenient to extend the learning power of the predictors for existing datasets. However, datasets never accessed in the past obviously miss these usage features. We are thus interested to understand if our classifier can learn how to use static and physics-domain features only, or, conversely, at what extent the lack of usage features jeopardizes its ability to accurately predict popularity.

The 320k samples represented in our training set refer to about 50k different datasets. In order to answer the above research question we included an additional set of 50k samples, one for each dataset. Each new sample refers to the week before the first access to the specific dataset and is labeled on the basis of the popularity of the dataset in the subsequent week. All usage attributes are set to zero in these samples since no access to the corresponding dataset was actually performed. Another approach in literature addresses the same issue by enriching the set of samples with a

number of rows randomly obtained from a global catalogue service [30]. Since this approach may introduce samples also for datasets that will never be accessed, and might hinder the exploitation by the classifier of dataset seasonality, we preferred to adopt a different solution that better model the reality.

Table 4.10 reports precision, recall and F1 score measured in the classification of new and old datasets. `New` refers to the newly introduced datasets, only described by static and physics-domain features. Conversely, `Old` refers instead to the datasets for which usage features are available, i.e., the dataset samples considered in all the previous experiments (see Table 4.7). The experimental results confirm the relevance of the usage features in predicting dataset popularity, since all classification measures are higher for the `Old` datasets. Nevertheless, the performance loss in the classification of new datasets is relatively small, hence the static features related to the physics domain encapsulate enough knowledge to be successfully exploited by the classifier.

Table 4.10: Classification performance on new and old datasets.

Datasets	Precision	Recall	F1
`New`	0.836	0.848	0.842
`Old`	0.881	0.870	0.875

4.10 Model Aging and Refreshing

The models learned to predict dataset popularity discussed so far are static. They are in fact trained on a static dataset extracted from the CMS access logs recorded in one year and tested on a test set obtained from the accesses recorded in the first weeks of the following years. The drawback of this approach is that access patterns and dataset characteristics could change over time. This change could give rise to an overall aging of the prediction model learned. Model aging is commonly observed in different application scenarios (e.g., [16]). To face up model aging it is necessary to re-train the model as new access patterns acquire popularity. Different strategies to keep the model updated can be adopted. The first decision to make is the minimum number of observations required to train the model. This may be thought of as the initial window. Starting at the beginning of the time series, March 2015, the static model discussed is trained using 297,086 samples throughout end of 2016. The model obtained is then used to predict dataset popularity for the next time step, e.g., the first week of 2017. After this time we have at disposal new information for updating

the model, e.g., all the dataset accesses logged during the first week of 2017. This data can be used to refresh the model and obtain more accurate prediction for the following week, e.g., the second week of 2017. Second, we need to decide whether the new model has to be trained on all data available or only on the most recent observations. This determines whether an expanding window, reinforced to include the new weeks, or sliding window, for moving along the time series, is used.

The aging of the static model is demonstrated in the plot reported in Fig. 4.9, where weekly test sets obtained from the accesses in 2016 and 2017 (61 total weeks) are used to score the classifier trained on all 2015's samples. As we can see from the plot the accuracy tends to decrease as the time gap between the training dataset and the test sets increases.

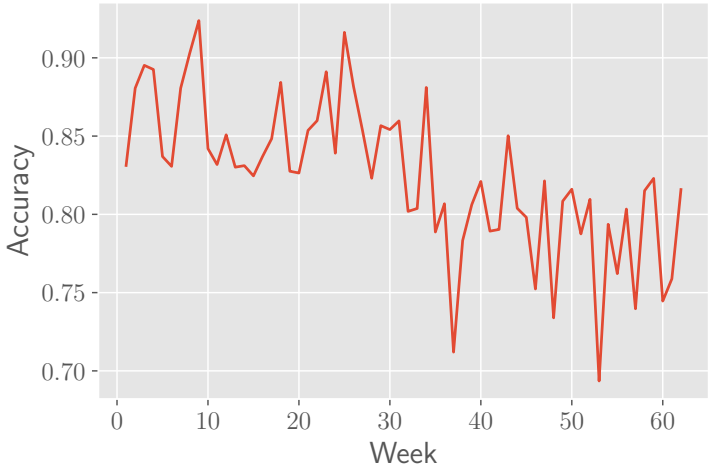


Figure 4.9: Demonstration of aging of the static model.

In order to address such model aging issue, we deployed and assessed two different techniques to update and keep fresh the prediction model:

- A **reinforcement** approach, where the training set used to build the prediction model is weekly extended with new samples from the previous week, and a fresh model trained from the reinforced dataset;
- a **sliding-window** approach, where the training set used to build the prediction model is weekly updated by substituting the samples from the oldest week with the samples of the previous week. Again a new model is trained weekly from this updated dataset that maintains its time-coverage constant over time.

The plot in Fig. 4.10 shows the effectiveness of the two model refreshing techniques. The accuracy of the static and the two model update techniques are compared on the first 9 weeks of year 2016. The static model results to perform worst because of the aging effect. On the other hand, the sliding-window model performs constantly better than the reinforced model. This is likely to be due to the noise introduced in the model by the older samples that, according to the reinforced approach, are always maintained in the dataset. We note however that the aging effect is quite slow to manifest. From Fig. 4.10 we note a maximal loss in accuracy of about 5% after 9 weeks. The relatively small degradation in such a quite long time period does not justify the adoption of complex streaming approaches to address concept drift in the classification task [1]. Furthermore, similar approaches rely on the continuous update of features that is not possible in our system where the values of dynamic usage features are the result of aggregation jobs performed daily or weekly.

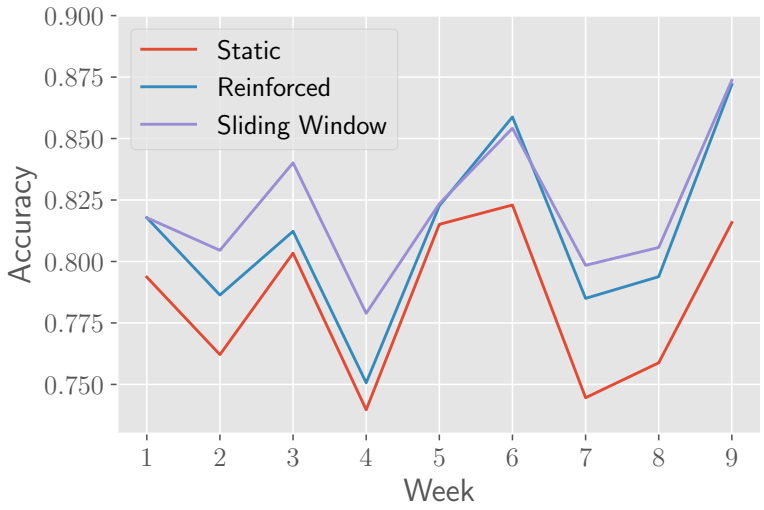


Figure 4.10: Comparison between different model update techniques.

The two techniques harness the cache persistence functionality of Spark RDD, by which any new weekly sample can be added to an existing RDD without the need to undergo a complete rebuild. The schema, referred as "rolling forecast", is shown in Fig. 4.11. The base model is trained on new weekly samples and predictions are cross-validated against the calculated target values. Learnings are applied to the next model which, in turn, will learn more and perform more accurately for the following week, and so on. Data is expected to have similar patterns over time, due to seasonality, repeated searches and social affinity. Hence, the model that is initially

trained on 2015's samples can be used to predict dataset popularity in a new week and later updated with actual data of that very week.

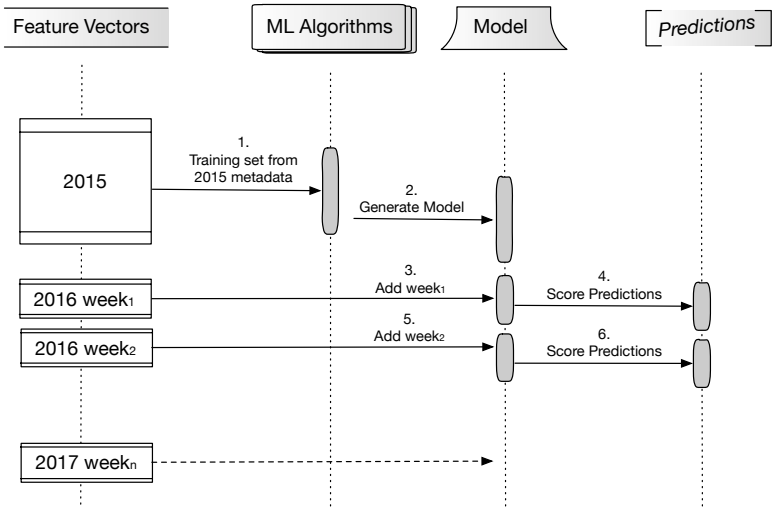


Figure 4.11: Prediction of time-series data using a rolling forecast approach.

4.11 Summary

In this chapter we have discussed our experimental approach to the CMS dataset popularity problem. Starting from a large set of observations from the last three-year period, we have derived the explanatory variables, grouped them into feature vectors, and defined the binary class to be predicted. We have demonstrated what is the preferable prediction timestamp for our study. Via careful and incremental feature engineering we have achieved a satisfying accuracy in the classification capability of our best model. We have further conducted a series of tests for assessing the impact of resampling techniques, the effectiveness of per-site classifiers, the performance evaluation measures on new and existing datasets, and the aging of the deployed model.

Our set of experiments has shown that the proposed predictive models reach very satisfying capacity to correctly separate popular datasets from unpopular ones. Therefore we can now exploit the popularity predictions achieved with our classifier to reason about the optimization of the eviction policy implemented at each site of the CMS infrastructure.

Chapter 5

Dataset Caching

Researches and scientists of the CMS collaboration have come to expect "fast enough" turnaround in their analyses. They expect job response times not affected by data location or related placement services. They assume the CMS computing infrastructure delivers data they expect so that each request is readily available anywhere. For these reasons, effective caches are fundamental. They must handle copies of the experiment datasets that are likely to be accessed in the forthcoming requests. Without global replication and caches holding the proper data at the rights sites, same user jobs shifting from one site to another would encounter "cold" caches, and processing the cache misses would affect either responsiveness and wall time.

The CMS computing model is based on the WLCG hierarchical tier structure and includes over hundreds of computing sites. The CERN IT centre represents the WLCG Tier-0 site. More than a dozens of worldwide large computing centres – with thousands of CPUs, PB of disk storage, tape storage systems and 24/7 Grid support service – are referred as Tier-1 centres. Tier-1 centres make data available to hundreds of Tier-2 sites, regional centres for specific analysis tasks. These CMS sites store datasets locally in order to minimize the network traffic when they are accessed by HEP data-intensive analysis jobs. The current site replacement policy follows the Least Recently Used (LRU) policy to select which element is evicted when a site is full and a new dataset is needed. We aim at enhancing LRU policy with additional knowledge of the dataset access patterns.

In the following we present a performance comparison of various caching algorithms applied to the sequence of dataset accesses discussed in chapter 4. In addition, we propose a novel caching strategy that harnesses dataset popularity predictions and improves hit rates.

The content of the chapter has been presented in [109, 108].

5.1 Caching Policies

Cache replacement algorithms, or policies, are optimizing instructions that a program can follow in order to manage a cache of information stored on the computer. When the cache is full, the algorithm must choose which items to discard to make room for the new ones. There exists a handful of replacement algorithms, which have been a hot topic of research and debate since the sixties [5, 121]. Over the decades, the development in underlying hardware and user-level software has affected the performance of replacement algorithms as result of changes in size of primary storage, memory hierarchies and locality of reference. In fact, the problem of "element replacement" occurs in multiple areas of computer design, from operating systems when a page fault occurs to Web servers that have to keep a certain number of heavily used Web pages in their memory cache.

In the next pages we describe the theoretical optimal caching strategy, the current LRU replacement policy used in CMS, and a frequency-based cutting edge algorithm. We compare all of them against our optimized solution that exploits the popularity predictions described in previous chapters.

5.1.1 Bélády's Algorithm

The most efficient caching algorithm consists of always discard the information that will not be needed for the longest time in the future. This optimal result is referred to as Bélády's algorithm (OPT) [26]. Since it is generally impossible to predict how far in the future a dataset will be needed, this is not implementable in practice. The practical best prediction can be calculated only after experimentation, and it is usually taken as an oracle to the effectiveness of the actually chosen cache algorithm. List. 5.1, line 4, highlights this knowledge of the future: when the cache is full and an element needs to be swapped in, the algorithm evicts the element whose next use will occur farthest in the future (F-F), or simply will not occur anymore.

Algorithm 5.1: Bélády's optimal (OPT) algorithm

```
   OPT (dataset ds):
1      if cache.miss(ds) then
2          if cache.full() then
3              // Evict F-F element
4              cache.evict(ds_{F-F})
5          cache.put(ds)
```

5.1.2 Least Recently Used Cache

A good approximation to the Bélády's algorithm relies on the observation that cache entries that have been heavily used in the recent past will probably be heavily used again in the next future. LRU exploits the temporal locality of datasets, which in CMS we have demonstrated to be characterized by a relatively short access pattern (see Section 4.5). When the cache gets full, a new data replaces the element in the storage location that has not been accessed for the longest period, following the observation that a cache entry that has not been accessed for longest is least likely to be accessed in the near future. General implementations of recency-based caching policies require keeping a *recency value*, a timestamp, for the cache elements. This value is changed every time an element is accessed, as indicated in List. 5.2, line 8. In case of cache miss, the given dataset is added (line 5), but if the cache is full, the entry with minimum recency value ($ds_{RV_{min}}$) is evicted first (line 4).

Actually there exists a family of caching algorithms based on frequently/recently used items. Among them, Time aware Least Recently Used (TLRU), Most Recently Used (MRU), Least-Frequently Used (LFU) and Least Frequent Recently Used (LFRU) to name a few. All these variants differ in terms of implementation costs or better adaptability to sequential scans rather than random accesses.

LRU works on the idea that elements (pages, datasets, files, etc) that have been most heavily used in the recent past are most likely to be used heavily in the next feature too. LRU can provide near-optimal performance in theory, and almost as good as adaptive replacement cache, but can become rather expensive to implement in practice.

Algorithm 5.2: Least Recently Used (LRU) algorithm

```
   LRU (dataset ds):
1      if cache.miss(ds) then
2          if cache.full() then
3              // Evict element with minimum recency value
4              cache.evict(ds_RV_min)
5          cache.put(ds)
6      else
7          // ds hit, refresh recency value
8          ds.updateRV()
```

5.1.3 Static Dynamic Cache

Our next step is to adopt the Static Dynamic Cache (SDC) [60] algorithm, which harnesses past data access frequency to enhance cache effectiveness. SDC has proven very effective for caching search engine query results and outperforms LRU. It stores the result of most frequently submitted queries in a static read-only portion of the cache, while the remaining dynamic portion of the cache manages query results with the LRU eviction policy and is used for the queries that cannot be satisfied by the static portion.

The authors of SDC always start their experiments from an initial warm cache with the static portion fixed in size and refreshed periodically. Likewise, we run SDC pre-initializing the static portion of each cache with the datasets most frequently accessed from the starting of the simulation up to the current week. The size of the static portion is 25% of the entire cache. When SDC is applied to our popularity study, the static portion is periodically refreshed to compensate the degradation of the hit rate (List. 5.3, lines 5-7). The refresh operation is considered zero-cost with respect to the misses count.

In case of cache miss, SDC checks if the dynamic portion of the cache is full (line 2). If it is, the element with oldest recency value is replaced (line 4). If instead the element is found, the recency value is updated only if the element belongs to the dynamic portion of the cache (line 12).

Algorithm 5.3: Static Dynamic Cache (SDC) algorithm

```
   SDC (dataset ds):
1      if cache.miss(ds) then
2          if cacheDynamic.full() then
3              // Evict element with minimum recency value
4              cacheDynamic.evict(ds_RV_min)
5              if cache.refresh() then
6                  // refresh most frequently accessed elements
7                  cacheStatic.update()
8          cache.put(ds)
9      else
10         if (ds ∈ cacheDynamic) then
11             // ds hit, refresh recency value
12             ds.updateRV()
```

5.1.4 Popularity Prediction Cache

We define a novel approach to data caching called Popularity Prediction Cache (PPC), which leverages the popularity labels predicted by the classifier. It relies on the knowledge of dataset popularity in the next week. We demonstrate that PPC offers a big edge over both LRU and SDC.

PPC behaves like a standard LRU but replacements occur to unpopular datasets only. In case of cache hit, the recency value is refreshed just like in the LRU algorithm, as it is shown in List. 5.4, line 15. On the contrary, the dataset eviction policy is driven by the popularity predictions when cache misses occur. If the cache is full, it is necessary to scan the set of recency values (RV^1, line 3) and keep in the cache all datasets that are classified as popular by the prediction model (line 5). For example, if the dataset with minimum recency value is popular, PPC looks then for the second-to-last least recently used dataset and repeats the test. If also this dataset is popular, the third-to-last is checked, and so on so forth, each time removing the just checked recency value ${RV^1}_{min}$ from the set (line 6). Hence, only the unpopular least recently used dataset (${ds}_{{RV^1}_{min}}$ such that ds is unpopular and its recency value is minimum among those in the current RV^1 set) becomes the candidate to be evicted from the cache (line 12). In the extreme case that no unpopular dataset is found, the regular LRU strategy is applied and the least recently used dataset ${ds}_{RV_{min}}$ is replaced (line 9).

Algorithm 5.4: Popularity Prediction Cache (PPC) algorithm

```
   PPC (dataset ds):
1    if cache.miss(ds) then
2      if cache.full() then
3        RV¹ = RV
4        // Skip popular elements in the cache
5        while (!RV¹.isEmpty() ∧ cache.isPopular(ds_{RV¹_min})) do
6          RV¹ = RV¹\{RV¹_min}
7        if RV¹.isEmpty() then
8          // Evict LRU popular element
9          cache.evict(ds_{RV_min})
10       else
11         // evict LRU unpopular element
12         cache.evict(ds_{RV¹_min})
13     cache.put(ds)
14   else
15     ds.updateRV()
```

False positive misclassifications may lead to extra disk space overhead, while false negatives may involve longer latencies of user jobs. In terms of numerical examples, 1% in false positive rate on 500 datasets of 2 TB average size would cause 10 TB of extra transfer. On the other hand, 1% of false negative rate on 1.5M CMS daily jobs would cause 15k competing jobs at sites where popular datasets are expected but missing.

The optimal implementation of PPC, which we name PPC*, would be to always predict the correct popularity class, i.e. whether a given dataset is indeed popular or not in the next week. As such, PPC* considers the actual number of accesses in the next week and compares this value with the popularity threshold (List. 5.5, line 5), resulting in a popularity oracle with 100% accuracy.

Algorithm 5.5: Optimal Popularity Prediction Cache (PPC*) algorithm

```
   PPC* (dataset ds):
1    if cache.miss(ds) then
2      if cache.full() then
3        RV^1 = RV
4        // Skip popular elements in the cache
5        while (!RV^1.isEmpty() ∧ ds_{RV^1_min}.nextAccess() ≥ popularityThreshold) do
6          RV^1 = RV^1\{RV^1_min}
7        if RV^1.isEmpty() then
8          // Evict LRU popular element
9          cache.evict(ds_{RV_min})
10       else
11         // evict LRU unpopular element
12         cache.evict(ds_{RV^1_min})
13     cache.put(ds)
14   else
15     ds.updateRV()
```

5.2 Performance Assessment

PPC, SDC and LRU performance are assessed by comparing the hit rate for increasing sizes of the caches. The experiment reproduces all dataset access requests throughout 2015 at the 6 most accessed CMS sites. We always perform a cold start of the cache at each site, thus generating a number of compulsory cache misses corresponding to the first reference to each distinct dataset. The analysis of the distributions of popular datasets over time (see Fig. 4.5) outlines different kinds of locality in the dataset access requests. Namely, some datasets are popular only within relatively

short time intervals, or they may become suddenly popular due to proximity to some paper submission deadlines.

Site statistics for the 6 most used CMS Grid sites are detailed in Table 5.1. The table shows the number of datasets (N), the popular ones (and their percentage), the number of total accesses (in millions), the number of compulsory misses (M_c) and the maximal hit rate ($H_{\max}$), computed as

$$H_{\max} = 1 - \frac{M_c}{N}.$$

Note that $H_{\max}$ represents that best hit rate value that any caching strategy can reach.

Table 5.1: Statistics of the 6 CMS most accessed sites.

Site	N	Popular (%)	Accesses	M_c	$H_{\max}$
`datagrid.cea.fr`	3,338	648 (19.4)	1.66 M	1,504	0.55
`desy.de`	3,639	603 (16.6)	0.86 M	1,069	0.71
`fnal.gov`	4,528	1,581 (34.9)	4.72 M	735	0.84
`hep.wisc.edu`	16,131	4,278 (26.5)	77.98 M	5,817	0.64
`jinr-t1.ru`	4,089	798 (19.5)	1.64 M	1,341	0.67
`lnl.infn.it`	5,601	1,034 (18.5)	1.44 M	1,552	0.72

Table 5.2 reports the hit rates for different cache sizes and different caching strategies at the 6 most used CMS sites. Best performance for each cache dimension is highlighted in bold font. From the table it is clear that PPC outperforms both LRU and SDC for small cache sizes. For example, for the site with most dataset, e.g., `hep.wisc.edu`, PPC obtains a hit rate of 19% using a very small cache fitting only 100 datasets, versus a 4% hit rate for SDC and a practically 0% hit rate for LRU. Similar benefits, slightly reduced, are obtained with cache size of 200 datasets.

Hence, PPC is the best policy when the cache size is limited, which makes it very effective in production sites with limited storage or bandwidth. Increasing the cache size makes the PPC strategy less competitive w.r.t. SDC and LRU, mostly because all the strategies perform quite similarly when approaching the maximal hit rate $H_{\max}$.

The performance of the LRU, SDC and PPC caching strategies can be further compared simulating the OPT most efficient caching algorithm, which always evicts the dataset that will not be needed for the longest time in the future. The OPT hit rate can be calculated only using historical data, but it allows to compare the effectiveness of the actually chosen cache algorithms with respect to the theoretical optimum. We further compare the performance of the PPC caching strategy with the

Table 5.2: Hit rate comparison of dataset caching among LRU, SDC and PPC.

Site	Policy	100	200	400	800	H_{max}
datagrid.cea.fr	LRU	0.26	0.45	0.50	0.52	0.55
	PPC	**0.31**	**0.46**	0.50	0.52	
	SDC	0.30	0.45	**0.52**	**0.54**	
desy.de	LRU	0.03	0.29	0.64	**0.70**	0.71
	PPC	**0.12**	**0.36**	**0.65**	**0.70**	
	SDC	0.10	0.33	0.64	**0.70**	
fnal.gov	LRU	0.23	0.51	0.72	**0.84**	0.84
	PPC	**0.39**	**0.56**	**0.73**	**0.84**	
	SDC	0.25	0.52	**0.73**	**0.84**	
hep.wisc.edu	LRU	0.00	0.05	0.29	0.45	0.64
	PPC	**0.19**	**0.20**	**0.35**	**0.46**	
	SDC	0.04	0.10	0.24	0.45	
jinr-t1.ru	LRU	0.20	0.55	0.64	**0.66**	0.67
	PPC	**0.28**	**0.56**	0.64	**0.66**	
	SDC	0.25	0.54	**0.65**	**0.66**	
lnl.infn.it	LRU	0.11	**0.55**	**0.67**	0.69	0.72
	PPC	**0.21**	**0.55**	**0.67**	0.69	
	SDC	0.16	0.53	**0.67**	**0.71**	

trained classifier w.r.t. the PPC strategy using the optimal popularity predictor, i.e., the popularity oracle with 100% accuracy (PPC*).

The comparison of results for the hep.wisc.edu site is shown in Fig. 5.1. It shows the hit rate performance of PPC w.r.t. LRU and SDC for small cache sizes reported in Table 5.2. Moreover the differences between the hit rates of the trained PPC strategy and the oracle PPC* strategy are very small (less than 0.02 for cache size 100 and 0.05 for cache size $200 - 300$), confirming the benefits of the high accuracy predictors in popularity caching.

After assessing the performance of PPC with respect to LRU and SDC on a large historical collection of dataset accesses, we investigate the performance of an actual deployment of PPC. Hence, we use the previous data to warm up the cache and we evaluate the performance of PPC on a new week of access log data, as shown in Fig. 5.2. Also with warm caches and actual data, the performance of PPC are close to the oracle PPC* strategy, with even smaller differences for cache size $100 - 400$. We do not report graphs on LRU/SDC performance achieved on other CMS sites since all the experiments conducted confirm the best performance of PPC.

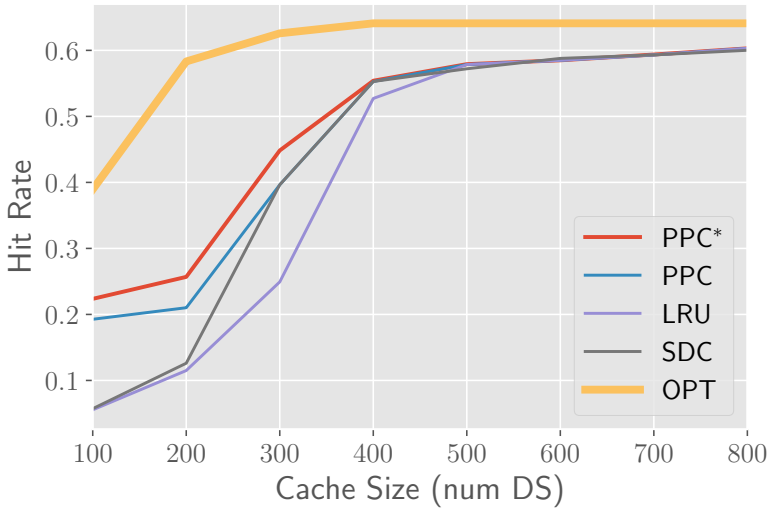


Figure 5.1: Simulation on historical data of the caching algorithms, including the theoretical optimal (OPT) and the 100% accurate classifier (PPC*).

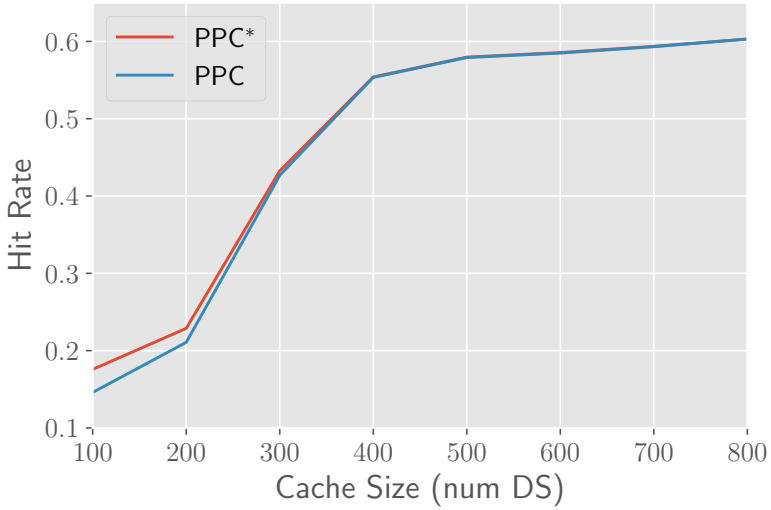


Figure 5.2: Hit rate difference between optimal and real classifier is negligible when using cache warm start.

5.3 Discussion

We have also performed one additional test that implements popularity pre-fetching. If the dataset selected for replacement is unpopular, it is retained in the cache if popular the week after. This is evident in List. 5.6, line 5, which verifies if the given dataset is popular in the next week or in the next two weeks. If any of the two

popularity conditions is satisfied, the algorithm keeps the dataset in the cache. PPC with pre-fetching is up to 2% more accurate than standard PPC when simulated on the `hep.wisc.edu` site shown in Fig. 5.1.

Algorithm 5.6: PPC algorithm with pre-fetching

```
   PPC (dataset ds):
1    if cache.miss(ds) then
2      if cache.full() then
3        RV^1 = RV
4        // Skip popular elements in the cache
5        while (!RV^1.isEmpty()∧
                (cache.isPopular(ds_{RV^1_min}) ∨ cache.isPopularNextWeek(ds_{RV^1_min}))) do
6          RV^1 = RV^1\{RV^1_min}
7        if RV^1.isEmpty() then
8          // Evict LRU popular element
9          cache.evict(ds_{RV_min})
10       else
11         // evict LRU unpopular element
12         cache.evict(ds_{RV^1_min})
13     cache.put(ds)
14   else
15     ds.updateRV()
```

Figure 5.3 investigates the impact of the threshold value, computed on the number of dataset accesses, on the PPC caching strategy performance for the `hep.wisc.edu` site. Different PPC* classifiers have been trained by using the sliding window technique (see Fig. 4.10), with different thresholds ranging from 10 to 50 accesses, denoted with PPC*(10), ..., PPC*(50). Note that the threshold value 50 corresponds to the cutoff selected in chapter 4, that is, PPC. Counter-intuitively, PPC*(10) performs 5% to 7% better than PPC*(50) when cache size is small. This is the consequence of the temporal locality of the dataset access sequence. In fact, the number of cache replacements is higher when cache size is limited. Moreover, the popularity predictions of a classifier trained with a threshold of 50 accesses are less effective because they reduce significantly the number of datasets whose eviction can be postponed. Hence, actually popular dataset are easily replaced in cache, that must be re-used in a short time.

The effectiveness of our caching solution is ultimately assessed by comparing the performance of all caching policies at the six most accessed CMS sites (Fig. 5.4). In this test, the hit rates are measured for cache size up to 1,500 datasets. PPC always outperforms other algorithms, especially for small cache sizes. Some sites, like `desy.de` and `datagrid.cea.fr`, are characterized by a burst of accesses in limited periods of

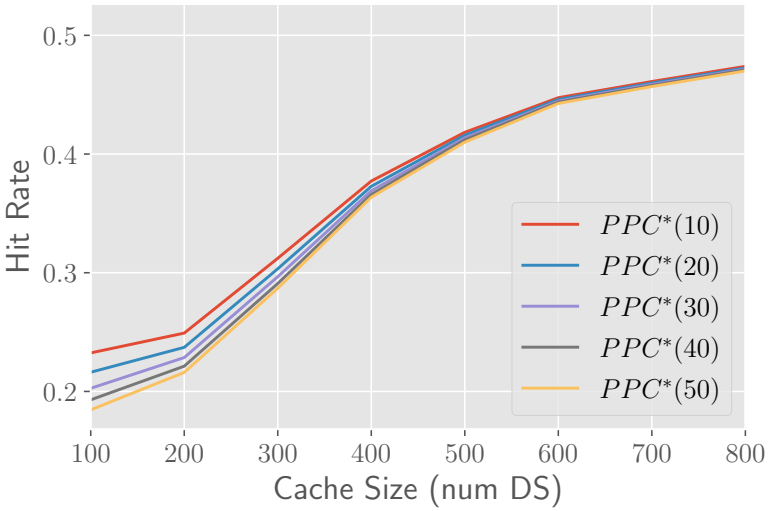


Figure 5.3: Hit rate of PPC caching strategy with different popularity thresholds, on 2 months of new data using warm caches ($H_{\max} = 0.53$).

time, and there is no significant differences among the caching policies because they all exploit temporal locality of datasets. On the contrary, sites like `hep.wisc.edu`, `jinr-t1.ru` and `lnl.infn.it` have less regular access patterns and PPC manifests better performance.

In addition, we argue that in our dataset caching problem the impact of the perfect classifier is not as superior as one would expect, which is proven by the generally limited performance difference between PPC and PPC*.

5.4 Summary

In this chapter we have described how we harnessed popularity predictions to implement an effective dataset replacement policy for the CMS distributed infrastructure. This policy is called Popularity Prediction Cache (PPC). It keeps stored in the cache a dataset that would be selected for eviction if our best binary classifier $GBT_{RUT+RBF+DTF}$ described in Sect. 4.7 predicts that the dataset is popular in the coming week. The hit rates obtained by PPC are compared with those achieved using caching policies found in literature like LRU and SDC. Experimental results have demonstrated that PPC outperforms the other strategies particularly when the cache size is limited, which makes it very effective in production sites with limited storage or bandwidth. Increasing the cache size makes the PPC strategy less competitive, mostly

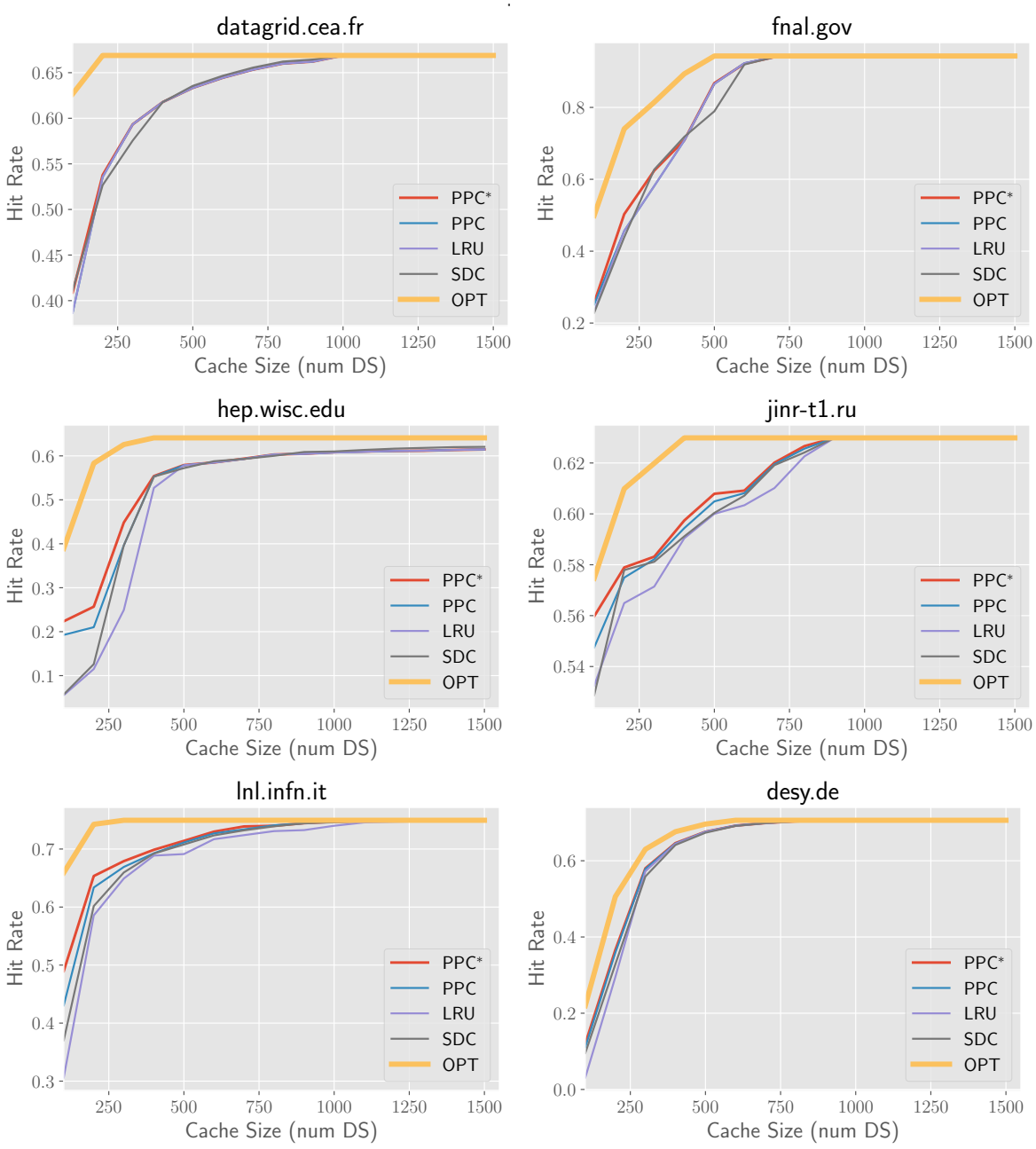


Figure 5.4: Hit rate comparison at the 6 most accessed CMS sites. The measures show that PPC outperforms LRU up to 20% when cache size is limited.

because all the strategies perform quite similarly when approaching the maximal hit rate $H_{\max}$.

Chapter 6

Conclusions and Future Work

This thesis has investigated three key requirements for large dataset popularity models at the WLCG: the need for a high-performance Big Data infrastructure that can solve current database shortcomings and deploy the next-generation mining platform; the need for a scalable pipeline of machine learning tools for training on dataset access logs predictive models that forecast which datasets will become popular over time and improve CPU utilization and task completion time; the need for a caching policy based on the dataset popularity predictions that can outperform the current dataset replacement implementation.

More precisely, we have motivated the decision to investigate ML techniques on Big Data infrastructures applied to the problem of dataset popularity, and to follow current research trends in computing resource optimization for HEP experiments. For example, ML modeling is a subject of current research in the scope of dataset placement optimization, reduction of transfer latencies, network-aware applications for anomaly detection, prediction of network congestions and path optimization.

We started our work with benchmarks of popularity data on a Hadoop analytics platform, and we demonstrated how current CMS dashboard and monitoring systems can benefit from the inherent parallelism of Big Data programming paradigms. We discussed the main use-cases for CMS analytics and ML studies, where efficient elaboration of billions of records stored on HDFS plays a crucial role. We demonstrated that the processing time of time series dataset access logs scales better than linearly with data volume, which makes it very effective for quick re-processing of any time periods. When compared to traditional DBMS implementations, speedup factors range between 2x for daily aggregations up to 50x on monthly time-frames. This results allows to execute extremely I/O intensive queries and provide valuable data insight from collected meta-data.

Because dataset placement at WLCG experiments is based on dataset access patterns, we considered the concept of popularity to make an optimal choice of replication and maximize data availability for processing and analysis needs. Consequently, we modeled dataset popularity by harnessing the huge amount of historical usage data recorded by the worldwide computing infrastructure in CMS. We derived either a dataset popularity prediction model and a popularity-aware caching policy.

In particular, we implemented a scalable pipeline of Spark components whose goal is to collect the dataset access logs from different sites, organize them into weekly snapshots, and train, on these snapshots, predictive models that can accurately forecast which datasets will become popular over time. The F1 measure of the best performing model is 0.875, which indicates an accurate ability to correctly classify popular datasets from unpopular ones.

Next, a novel data caching policy was proposed to exploit at each CMS site the accurate predictions computed and kept fresh by the pipeline of scalable Spark components running on the CMS Hadoop cluster. We named this policy Popularity Prediction Cache (PPC). We evaluated the performance of PPC against popular caching policy baselines like LRU and its variations. The experiments conducted on large traces of real dataset accesses showed that PPC outperforms LRU and it increases the number of cache hits up to 20% at some sites. Notably, PPC results in being the best caching policy when cache size is limited, which makes it very effective in production sites with limited storage or bandwidth.

This result is particularly important for the efficiency of the CMS Grid infrastructure because it allows to deploy an effective data placement policy creating replicas of datasets at the sites where they are most likely to be used. Enhancing the CMS data placement policy involves a significant improvement of resource usage and a consequent reduction of the large cost of this important infrastructure.

Future Research Directions

CMS considers the Spark platform as a potentially crucial component in the ecosystem that would enable LHC experiments to attack Big Data problems in the long run. Dataset popularity predictions can play a significant role in intelligent data placement at Grid sites for newly created datasets and for holding dataset samples frequently accessible by CMS physicists.

As in many real-life situations, the CMS dataset samples are unbalanced because representatives of the popular class appear much more frequently. This poses a

difficulty for learning algorithms, as they will be biased towards the majority group. At the same time, the minority class is the one more important from the data mining perspective because despite its rarity it carries important knowledge of interesting datasets. Learning from imbalanced data has been widely discussed in literature [83]. We have conducted a number of incremental enhancements in popularity learning from unbalanced data, but have also performed studies on balancing this data via preprocessing and adaptation of the training sets. Still, there exist many challenges in the field of imbalanced learning [78] that require intensive research and development. Gaining deeper insights into the potential value that can be extracted from the combination of models trained from both imbalanced and balanced sets represents a possible research direction for improving Big Data popularity classification problems.

Accurate dataset popularity classification represents, in turn, the basis for improving the efficiency of physics analysis and define how many dataset replicas is convenient to store. A clever caching strategy based on ML predictions can lead to cost-effective data placement across Grid sites and better utilization of available computing resources. Caching strategies can be evaluated using different metrics, and hit rate is the most common. However, very few works take into account the cost of misses when evaluating the performance of caching strategies and propose cost-aware caching strategies. As a future work, we will develop network transfer and financial cost models, taking into account the dataset transfer cost as well as the electricity prices when computing the cost, and we will evaluate cost-aware caching eviction policies, which takes into account both transfer time and costs (e.g., energy expenditure to transfer dataset).

Big Data generate additional challenges to learning systems [144]. Not only the increasing data volume can become prohibitive for existing methods, but also the nature of the problems can cause difficulties. First of all, learning from Big Data can require more efficient algorithms due to the challenges posed by computing environments like Spark or Hadoop. These systems were initially not developed for handling unbalanced datasets causing few of the executor tasks taking a long time to finish the job [1]. Second, it also requires methods for processing and classifying data in form of heterogeneous and atypical representations, like graphs, structures, video sequences, images, tensors, etc. Such data types are becoming more and more frequent in Big Data analytics and impose certain restrictions on machine learning systems, or need smart conversion strategies into appropriate numerical values or structures, as highlighted in the next section.

[1] datarus.wordpress.com/2015/05/04/fighting-the-skew-in-spark

Classification of Physics Events

An innovative research direction would consist of leveraging the experience with predicting the popularity of CMS datasets to forecast the physics nature of the experiment data that is stored in these datasets.

In fact, a general task in the analysis of LHC collisions is the classification of collected events as a function of the originating physical process. CMS event-by-event interpretation is a mean to try and reconstruct, using raw detector readout and the output of complex algorithms. Given the complexity of LHC events, this can be done only via statistical methods (MVA, BDT, likelihood, etc.). In general, it is customary to distinguish between not interesting events, called "background", and interesting events, called "signal".

A revolutionary approach in event classification would be to abandon statistical methods and focus on low level quantities, as close as possible to detector readouts, in order not to bias the classification process with human written high-level algorithms. While in principle this could be taken at the extreme of using directly detector readout quantities as-is, the approach is considered premature and would involve up to hundreds of millions of inputs per each and every studied event. A number too high even for deep learning (DL) [88] network topologies based on Convolutional Neural Networks (CNN) [131], such as LeNet-5 [89], or Neural Machine Translation (NMT) [57].

A full collection of detector inputs (called "hits") from a single physics interaction is currently stored as jagged array, a data representation that can capture the complex sparse connectivity structure of the detector, otherwise difficult to express succinctly in other graphical formalisms. Nevertheless, jagged array represents a problem when it comes to ML/DL training algorithms because none of them is yet implemented to deal with such type of input.

We can research on a proper format for HEP event classification based on Autoencoder [15] (AE), a type of neural network that maps inputs to the closest training sample they can remember. To the best of author's knowledge, there are no current solutions for feeding ML/DL algorithms with samples representable by jagged array. Trained AE might learn CMS detector structure, thus overcoming the limitation of jagged array. In essence, it is impossible to establish beforehand how many hits a certain event will have and what is the proper internal format to feed ML/DL classification. But AE can learn the data structure and its representation, and ultimately

produce a compact representation hopefully sufficient to represent the event types and classify their physics nature.

Bibliography

[1] H. Abdulsalam, D.B. Skillicorn, and P. Martin. Classification Using Streaming Random Forests. *IEEE Transactions on Knowledge and Data Engineering*, 23(1):22–36, Jan 2011. 72

[2] R. Abmann, M. Lamont, and S. Myers. A brief history of the LEP collider. *Nuclear Physics B - Proceedings Supplements*, 109(2):17 – 31, 2002. Proceedings of the 7th Topical Seminar. 13

[3] M. Aderholz, K. Amako, E. Auge, G. Bagliesi, K. Barone, G. Battistoni, M. Bernardi, M. Boschini, A. Brunengo, J. Bunn, J. Butler, M. Campanella, P. Capiluppi, F. Carminati, M. D'Amato, M. Dameri, A. Di Mattia, A. Dorokhov, G. Erbacci, and D. Williams. Models of networked analysis at regional centres for LHC experiments (MONARC). Phase 2 report. Technical report, 02 2001. 29

[4] A. Afaq, A. Dolgert, Y. Guo, C. Jones, S. Kosyakov, V. Kuznetsov, L. Lueking, D. Riley, and V. Sekhri. The CMS dataset bookkeeping service. In *Journal of Physics: Conference Series*, volume 119, page 072001. IOP Publishing, 2008. 17, 23, 36

[5] A.V. Aho, P.J. Denning, and J.D. Ullman. Principles of Optimal Page Replacement. *J. ACM*, 18(1):80–93, January 1971. 76

[6] A. Aimar, A.A. Corman, P. Andrade, S. Belov, J.D. Fernandez, B.G. Bear, M. Georgiou, E. Karavakis, L. Magnoni, R. Rama Ballesteros, H. Riahi, J.R. Martinez, P. Saiz, and D. Zolnai. Unified Monitoring Architecture for IT and Grid Services. *Journal of Physics: Conference Series*, 898(9):092033, 2017. 23

[7] A. Aimar, A.A. Corman, P. Andrade, S. Belov, J.D. Fernandez, B.G. Bear, M. Georgiou, E. Karavakis, L. Magnoni, R.R. Ballesteros, H. Riahi, J.R. Martinez, P. Saiz, and D. Zolnai. Unified Monitoring Architecture for IT and Grid Services. *Journal of Physics: Conference Series*, 898(9):092033, 2017. 3, 21, 38

[8] W. Ali, S.M. Shamsuddin, and A.S. Ismail. A Survey of Web Caching and Prefetching. *Int. J. Advance. Soft Comput. Appl*, 3(1):18–44, 03 2011. 31

[9] B. Allcock, J. Bester, J. Bresnahan, A.L. Chervenak, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, and I. Foster. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. In *2001 Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*, pages 13–13, April 2001. 29

[10] J. Andreeva, A. Beche, S. Belov, A.D. Dieguez, D. Giordano, D. Oleynik, A. Petrosyan, P. Saiz, M. Tadel, D. Tuckett, and I. Vukotic. Monitoring of large-scale federated data storage: XRootD and beyond. *Journal of Physics: Conference Series*, 513:032004, 06 2014. 34

[11] J. Andreeva, B. Gaidioz, J. Herrala, G. Maier, R. Rocha, P. Saiz, and I. Sidorova. Dashboard for the LHC experiments. In *Journal of Physics: Conference Series*, volume 119, page 062008. IOP Publishing, 2008. 6, 16

[12] M. Armbrust, R.S. Xin, C. Lian, Y. Huai, D. Liu, J.K. Bradley, X. Meng, T. Kaftan, M.J. Franklin, A. Ghodsi, and M. Zaharia. Spark SQL: Relational Data Processing in Spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1383–1394, New York, NY, USA, 2015. ACM. 21

[13] C.G. Atkeson, A.W. Moore, and S. Schaal. Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer, 1997. 20

[14] F. Bajaber, R. El Shawi, O. Batarfi, A. Altalhi, A. Barnawi, and S. Sakr. Big Data 2.0 Processing Systems: Taxonomy and Open Challenges. *Journal of Grid Computing*, 14, 06 2016. 2

[15] P. Baldi. Autoencoders, Unsupervised Learning and Deep Architectures. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop*, volume 27 of *UTLW'11*, pages 37–50. JMLR.org, 2011. 92

[16] R. Baraglia, C. Castillo, D. Donato, F.M. Nardini, R. Perego, and F. Silvestri. Aging effects on query flow graphs for query suggestion. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM'09, pages 1947–1950, 2009. 70

[17] Z. Baranowski, L. Canali, and E. Grancher. Sequential data access with Oracle and Hadoop: a performance comparison. *Journal of Physics: Conference Series*, 513(4):042001, 2014. 4

[18] Z. Baranowski, M. Grzybek, L. Canali, D. L. Garcia, and K. Surdy. Scale out databases for CERN use cases. *Journal of Physics: Conference Series*, 664(4):042002, 2015. 1

[19] P. Barrett. A Review of "Statistical Models: Theory and Practice". 16:391–395, 04 2009. 26

[20] L.A.T. Bauerdick, K. Bloom, B. Bockelman, D. Bradley, S. Dasu, J. Dost, I. Sfiligoi, A. Tadel, M. Tadel, F. Wuerthwein, and A. Yagil. XRootd, disk-based, caching proxy for optimization of data access, data placement and data replication. *Journal of Physics: Conference Series*, 513(4):042044, 06 2014. 22

[21] T. Beermann. A study on dynamic data placement for the ATLAS Distributed Data Management system. Technical report, ATL-COM-SOFT-2015-012, 2015. 27

[22] T. Beermann, P. Maettig, G. Stewart, M. Lassnig, V. Garonne, M. Barisits, R. Vigne, C. Serfon, L. Goossens, A. Nairz, and A. Molfetas. Popularity Prediction Tool for ATLAS Distributed Data Management. In *Journal of Physics: Conference Series*, volume 513, page 042004. IOP Publishing, 2014. 27

[23] T. Beermann, P. Maettig, G. Stewart, M. Lassnig, V. Garonne, M. Barisits, R. Vigne, C. Serfon, L. Goossens, A. Nairz, A. Molfetas, and the Atlas collaboration. Popularity Prediction Tool for ATLAS Distributed Data Management. *Journal of Physics: Conference Series*, 513(4):042004, 2014. 21

[24] T. Beermann, G.A. Stewart, and P. Maettig. A Popularity Based Prediction and Data Redistribution Tool for ATLAS Distributed Data Management. *PoS*, ISGC2014:004, 2014. 27

[25] S.M. Beitzel, E.C. Jensen, A. Chowdhury, D. Grossman, and O. Frieder. Hourly Analysis of a Very Large Topically Categorized Web Query Log. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 321–328, New York, NY, USA, 2004. ACM. 31

[26] L.A. Belady. A Study of Replacement Algorithms for Virtual-Storage Computer. *IBM Systems Journal*, 5(2):78–101, 1966. 76

[27] H.S. Benjamin, L.A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. Technical report, Google, Inc., 2010. 19

[28] K. Bloom, T. Boccali, B. Bockelman, D.C. Bradley, S. Dasu, J. Dost, F. Fanzago, I. Sfiligoi, A.M. Tadel, M. Tadel, C. Vuosalo, F. Würthwein, A. Yagil, and M. Zvada. Any Data, Any Time, Anywhere: Global Data Access for Science. *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*, pages 85–91, 2015. 23, 35

[29] D. Bonacorsi, T. Boccali, D. Giordano, M. Girone, M. Neri, N. Magini, V. Kuznetsov, and T. Wildish. Exploiting CMS data popularity to model the evolution of data management for Run-2 and beyond. *Journal of Physics: Conference Series*, 664(3):032003, 2015. 22

[30] D. Bonacorsi, V. Kuznetsov, T. Wildish, and L. Giommi. Exploring Patterns and Correlations in CMS Computing Operations Data with Big Data Analytics Techniques. In *Proceedings, International Symposium on Grids and Clouds*, volume 762 of *ISGC '15*, page 008, 2016. 27, 70

[31] F. Bonchi, F. Giannotti, C. Gozzi, G. Manco, M. Nanni, D. Pedreschi, C. Renso, and S. Ruggieri. Web log data warehousing and mining for intelligent web caching. *Data Knowl. Eng.*, 39(2):165–189, 2001. 30, 31

[32] R. Brun and F. Rademakers. ROOT: An object oriented data analysis framework. *Nucl. Instrum. Meth.*, A389:81–86, 1997. 21, 53

[33] C.J.C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, June 1998. 26

[34] Y. Cardenas, J.M. Pierson, and L. Brunie. Uniform Distributed Cache Service for Grid Computing. In *16th International Workshop on Database and Expert Systems Applications*, DEXA'05, pages 351–355. IEEE, 2005. 30

[35] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Machine Learning, Proceedings of the 25th International Conference*, ICML'08, pages 96–103, 2008. 58

[36] L. Čerkasova and Hewlett-Packard Laboratories. *Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy*. HP Laboratories technical report. Hewlett-Packard Laboratories, 1998. 31

[37] N.V. Chawla, K.W. Bowyer, L.Hall, and P.Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002. 66

[38] M. Cinquilli, D. Spiga, C. Grandi, J.M. Hernandez, P. Konstantinov, M. Mascheroni, H. Riahi, and E. Vaandering. CRAB3: Establishing a new generation of services for distributed analysis at CMS. *J. Phys. Conf. Ser.*, 396:032026, 2012. 16

[39] N. Coleman. Distributed Policy Specification and Interpretation with Classified Advertisements. 7149:198–211, 01 2012. 16

[40] B. Couturier, G. Eulisse, H. Grasland, B. Hegner, M. Jouvin, M. Kane, D. Katz, T. Kuhr, D. Lange, P.M. Lorenzo, et al. HEP Software Foundation Community White Paper Working Group-Software Development, Deployment and Validation. *arXiv preprint arXiv:1712.07959*, 2017. 21

[41] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627, IETF, 7 2006. 34

[42] S. Daruru, N.M. Marin, M. Walker, and J. Ghosh. Pervasive parallelism in data mining: dataflow solution to co-clustering large and sparse Netflix data. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining*, SIGKDD '09, pages 1115–1124, 2009. 28

[43] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, January 2008. 3, 19, 24

[44] T. G. Dietterich. Ensemble Methods in Machine Learning. In *Proceedings of the 1st International Workshop on Multiple Classifier Systems*, pages 1–15, 2000. 57

[45] C. Dobre and C. Stratan. MONARC Simulation Framework. *arXiv preprint arXiv:1106.5158*, 06 2011. 29

[46] A. Dorigo, P. Elmer, F. Furano, and A. Hanushevsky. XROOTD - A highly scalable architecture for data access. *WSEAS Transactions on Computers*, 1(4.3):348–353, 04 2005. 22

[47] S. D'Souza and S. Hoffman. *Apache Flume: Distributed Log Collection for Hadoop*. Community experience distilled. Packt Publishing Ltd, 2013. 19, 34

[48] D.M. Dutton and G.V. Conroy. A Review of Machine Learning. *Knowl. Eng. Rev.*, 12(4):341–367, December 1997. 20

[49] R. Egeland, T. Wildish, and S. Metson. Data transfer infrastructure for CMS data taking. *PoS*, ACAT08:033, 01 2008. 7

[50] X. Espinal, G. Adde, B. Chan, J. Iven, G. Lo Presti, M. Lamanna, L. Mascetti, A. Pace, A.J. Peters, S. Ponce, and E. Sindrilaru. Disk storage at CERN: Handling LHC data and beyond. *Journal of Physics: Conference Series*, 513(4):042017, 2014. 23, 35

[51] G. Aad et al. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3:S08003, 2008. 14

[52] J. Alves et al. The LHCb Detector at the LHC. *JINST*, 3:S08005, 2008. 14

[53] K. Aamodt et al. The ALICE experiment at the CERN LHC. *JINST*, 3:S08002, 2008. 14

[54] R.L. Evans et al. The large hadron collider. In *Proceeding of the Particle Accelerator Conference*, volume 1, pages 40–44. IEEE, 1995. 1

[55] S. Chatrchyan et al. The CMS Experiment at the CERN LHC. *JINST*, 3:S08004, 2008. 4, 14

[56] S. Chatrchyan et al. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Phys. Lett.*, B716:30–61, 2012. 4

[57] Y. Wu et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*, abs/1609.08144, 2016. 92

[58] L. Evans and P. Bryant. LHC Machine. *JINST*, 3:S08001, 2008. 1

[59] F.M. Facca and P.L. Lanzi. Mining interesting knowledge from weblogs: a survey. *Data Knowl. Eng.*, 53(3):225–241, 2005. 31

[60] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. Boosting the performance of Web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24(1):51–78, 2006. 31, 78

[61] J. Famaey, T. Wauters, and F. De Turck. On the merits of popularity prediction in multimedia content caching. In *12th IFIP/IEEE International Symposium on Integrated Network Management and Workshops*, IM 2011, pages 17–24, May 2011. 30

[62] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do We Need Hundreds of Classifiers to Solve Real World Classification Problems? *J. Mach. Learn. Res.*, 15(1):3133–3181, January 2014. 57

[63] R.T. Fielding and R.N. Taylor. *Architectural Styles and the Design of Network-based Software Architectures*, volume 7. University of California, Irvine Doctoral dissertation, 2000. AAI9980887. 16

[64] I. Foster. Globus Toolkit Version 4: Software for Service-oriented Systems. In *Proceedings of the 2005 IFIP International Conference on Network and Parallel Computing*, NPC'05, pages 2–13, Berlin, Heidelberg, 2005. Springer-Verlag. 22

[65] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997. 22

[66] I. Foster and C. Kesselman. *The Grid: Blueprint for a new computing infrastructure*. Elsevier, 2003. 1

[67] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, August 2001. 1

[68] J.H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 10 2001. 26, 28, 57

[69] V. Ganganwar. An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering*, 2(4):42–47, 2012. 65

[70] M. Giffels, Y. Guo, V. Kuznetsov, N. Magini, and T. Wildish. The CMS Data Management System. *Journal of Physics: Conference Series*, 513(4):042052, 2014. 23

[71] M. Giffels, Y. Guo, and D. Riley. Data Bookkeeping Service 3 – Providing event metadata in CMS. *Journal of Physics: Conference Series*, 513(4):042022, 2014. 23, 36

[72] O. Gutsche, L. Canali, I. Cremer, M. Cremonesi, P. Elmer, I. Fisk, M. Girone, B. Jayatilaka, J. Kowalkowski, V. Khristenko, E. Motesnitsalis, J. Pivarski, S. Sehrish, K. Surdy, and A. Svyatkovskiy. CMS Analysis and Data Reduction with Apache Spark. *CoRR*, abs/1711.00375, 2017. 21

[73] O. Gutsche, M. Cremonesi, P. Elmer, B. Jayatilaka, J. Kowalkowski, J. Pivarski, S. Sehrish, C. Mantilla Surez, A. Svyatkovskiy, and N. Tran. Big Data in HEP: A comprehensive use case study. *Journal of Physics: Conference Series*, 898(7):072012, 2017. 21

[74] J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. 17

[75] H. He, Y. Bai, E. Garcia, and S. Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1322–1328. IEEE, 2008. 66

[76] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger. Data Management in an International Data Grid Project. In *Grid Computing*, GRID 2000, pages 77–90, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. 29

[77] M. Hushchyn, P. Charpentier, and A. Ustyuzhanin. Disk storage management for LHCb based on Data Popularity estimator. *Journal of Physics: Conference Series*, 664(4):042026, Apr 2015. 21, 28, 29

[78] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002. 91

[79] T. Joachims. A Statistical Learning Model of Text Classification with Support Vector Machines. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 128–136, 2001. 26

[80] M. Jordan and T.M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 07 2015. 30

[81] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. An Analysis of Traces from a Production MapReduce Cluster. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 94–103, Washington, DC, USA, 2010. IEEE Computer Society. 19

[82] D. Kranzlmüller, J.M. De Lucas, and P. Öster. The European Grid Initiative (EGI). In F. Davoli, N. Meyer, R. Pugliese, and S. Zappatore, editors, *Remote Instrumentation and Virtual Laboratories*, pages 61–66, Boston, MA, 2010. Springer US. 1

[83] B. Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, Nov 2016. 67, 91

[84] S. Krum, W. Van Hevelingen, and D. Sluijters. *Beginning Puppet*. Apress, Berkely, CA, USA, 1st edition, 2016. 34

[85] V. Kuznetsov, T. Li, L. Giommi, D. Bonacorsi, and T. Wildish. Predicting dataset popularity for the CMS experiment. *Journal of Physics: Conference Series*, 762(1):012048, 2016. 21, 27, 29, 33, 40

[86] M. Lamanna. The LHC computing grid project at CERN. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 534(1-2):1–6, 11 2004. 1

[87] N. Leavitt. Will NoSQL Databases Live Up to Their Promise? *Computer*, 43(2):12–14, February 2010. 2

[88] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015. 92

[89] Y. LeCun, L.D. Jackel, L. Bottou, A. Brunot, C. Cortes, J.S. Denker, et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60. Perth, Australia, 1995. 92

[90] A. Legrand, L. Marchal, and H. Casanova. Scheduling Distributed Applications: the SimGrid Simulation Framework. In *Proc. 3rd IEEE/ACM Int. Symp. on Clust. Comput. and the Grid*, pages 138–145. IEEE, 06 2003. 29

[91] I. Legrand, H. Newman, R. Voicu, C. Grigoras, C. Cirstoiu, and C. Dobre. MonALISA: A Distributed Service System for Monitoring, Control and Global Optimization. *PoS*, page 020, 01 2008. 19

[92] M. Lei, S.V. Vrbsky, and X. Hong. An On-line Replication Strategy to Increase Availability in Data Grids. *Future Gener. Comput. Syst.*, 24(2):85–98, February 2008. 30

[93] R. Lempel and S. Moran. Predictive Caching and Prefetching of Query Results in Search Engines. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 19–28, New York, NY, USA, 2003. ACM. 31

[94] J. Leskovec, A. Rajaraman, and J.D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2nd edition, 2014. 17

[95] T. Likhomanenkoa, A. Rogozhnikova, A. Baranova, E. Khairullina, and A. Ustyuzhanina. Improving reproducibility of data science experiments. In *International Conference on Machine Learning*, ICML'15, 2015. 28

[96] X. Long and T. Suel. Three-level Caching for Efficient Query Processing in Large Web Search Engines. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 257–266, New York, NY, USA, 2005. ACM. 31

[97] C. Lucchese, F.M. Nardini, S. Orlando, R. Perego, and N. Tonellotto. Speeding up Document Ranking with Rank-based Features. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 895–898, 2015. 64

[98] T. Maeno. PanDA: distributed production and distributed analysis system for ATLAS. *Journal of Physics: Conference Series*, 119(6):062036, 2008. 27

[99] T. Maeno, K. De, and S. Panitkin. PD2P: PanDA Dynamic Data Placement for ATLAS. *Journal of Physics: Conference Series*, 396(3):032070, 2012. 27

[100] M. Manago and Y. Kodratoff. Induction of Decision Trees from Complex Structured Data. In *Knowledge Discovery in Databases*, pages 289–308. 1991. 26

[101] M.L. Massie, B.N. Chun, and D.E. Culler. The Ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004. 19

[102] A. De Mauro, M. Greco, and M. Grimaldi. A formal definition of Big Data based on its essential features. *Library Review*, 65:122–135, 03 2016. 23

[103] B. McFee, T. Bertin-Mahieux, D. Ellis, and G. Lanckriet. The million song dataset challenge. In *Proceedings of the 21st International Conference on World Wide Web*, pages 909–916. ACM, 2012. 28

[104] B. Megino, M. Cinquilli, D. Giordano, E. Karavakis, M. Girone, N. Magini, V. Mancinelli, and D. Spiga. Implementing data placement strategies for the CMS experiment based on a popularity model. *Journal of Physics: Conference Series*, 396, 12 2012. 8, 17

[105] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D.B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M.J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. MLlib: Machine Learning in Apache Spark. *J. Mach. Learn. Res.*, 17(1):1235–1241, January 2016. 21, 25

[106] M. Meoni, T. Boccali, N. Magini, L. Menichetti, and D. Giordano. XRootD popularity on Hadoop clusters. *Journal of Physics: Conference Series*, 898(7):072027, 2017. 11, 21, 29, 33, 41

[107] M. Meoni, V. Kuznetsov, T. Boccali, L. Menichetti, and J. Rumševičius. Exploiting Apache Spark platform for CMS computing analytics. *Advanced Computing and Analysis Techniques in Physics Research, arXiv preprint arXiv:1711.00552*, 2017. 11, 29, 33, 51, 57

[108] M. Meoni, R. Perego, and N. Tonellotto. Dataset Popularity Prediction for Caching of CMS Big Data. *Journal of Grid Computing*, 16(2):211–228, Feb 2018. 11, 12, 51, 75

[109] M. Meoni, R. Perego, and N. Tonellotto. Popularity-Based Caching of CMS Datasets. *Parallel Computing is Everywhere*, 32:221–231, 2018. 11, 12, 40, 51, 75

[110] T.M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. 20

[111] S. Mohanty, M. Jagadeesh, and H. Srivatsa. Hadoop could save you money over a traditional RDBMS. In *Big Data Imperatives*, pages 1–24. Springer, 2013. 4

[112] A. Molfetas, M. Lassnig, V. Garonne, G. Stewart, M. Barisits, T. Beermann, and G. Dimitrov. Popularity framework for monitoring user workload. *Journal of Physics: Conference Series*, 396(5):052055, 2012. 27

[113] A. Oliner, A. Ganapathi, and W. Xu. Advances and Challenges in Log Analysis. *Commun. ACM*, 55(2):55–61, February 2012. 18

[114] E. Otoo and A. Shoshani. Accurate modeling of cache replacement policies in a data grid. In *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, MSST 2003, pages 10–19, April 2003. 29

[115] R. Pálovics, D. Kelen, and A.A. Benczúr. Tutorial on Open Source Online Learning Recommenders. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 400–401, New York, NY, USA, 2017. ACM. 34

[116] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, Nov 2011. 25, 29

[117] A.J. Peters and L. Janyst. Exabyte Scale Storage at CERN. *Journal of Physics: Conference Series*, 331(5):052015, 2011. 22

[118] J. Pham, E. Kyauk, and E. Park. Predicting Song Popularity. 26, 2016. 28

[119] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, and R. Quick. The Open Science Grid. *Journal of Physics: Conference Series*, 78(1):012057, 2007. 1

[120] K.L. Priddy and P.E. Keller. *Artificial Neural Networks: An Introduction*, volume TT68. SPIE - International Society for Optical Engineering, 2005. 27

[121] T.R. Puzak. Analysis of Cache Replacement-algorithms. 1985. AAI8509594. 76

[122] A. Rabkin and R. Katz. Chukwa: A System for Reliable Large-scale Log Collection. In *Proceedings of the 24th International Conference on Large Installation System Administration*, LISA'10, pages 1–15, Berkeley, CA, USA, 2010. USENIX Association. 18

[123] B. Rajkumar and M. Manzur. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience (CCPE)*, 14(13):1175–1220, 2002. 29

[124] K. Ranganathan and I. Foster. Identifying Dynamic Replication Strategies for a High-Performance Data Grid. In *International Workshop on Grid Computing*, pages 75–86. Springer, 2001. 29

[125] K. Ranganathan and I. Foster. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, HPDC '02, pages 352–, Washington, DC, USA, 2002. IEEE Computer Society. 29

[126] K. Ranganathan and I. Foster. Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. *Journal of Grid Computing*, 1:53–62, 03 2003. 8

[127] J. Rehn, T. Barrass, D. Bonacorsi, J. Hernández, I. Semeniouk, L. Tuura, and Y. Wu. PhEDEx high-throughput data transfer management system. *Computing in High Energy and Nuclear Physics (CHEP)*, 01 2006. 16, 36

[128] M. Reif, F. Shafait, M. Goldstein, T. Breuel, and A. Dengel. Automatic classifier selection for non-experts. *Pattern Analysis and Applications*, 17(1):83–96, Feb 2014. 57

[129] A. Sanchez-Hernandez, R. Egeland, C.H. Huang, N. Ratnikova, N. Magini, and T. Wildish. From toolkit to framework - the past and future evolution of PhEDEx. *Journal of Physics: Conference Series*, 396(3):032118, 2012. 23

[130] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, pages 85 – 117, 2015. 20

[131] P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *21st International Conference on Pattern Recognition (ICPR)*, pages 3288–3291. IEEE, 2012. 92

[132] S. Shalev-Shwartz, O. Shamir, and E. Tromer. Using More Data to Speed-up Training Time. *CoRR*, abs/1106.1216, 2011. 20

[133] J. Shamsi, M.A. Khojaye, and M.A. Qasmi. Data-Intensive Cloud Computing: Requirements, Expectations, Challenges, and Solutions. *Journal of Grid Computing*, 11(2):281–310, June 2013. 2

[134] J. Shiers. The Worldwide LHC Computing Grid (worldwide LCG). *Computer Physics Communications*, 177(1-2):219–223, 2007. 1

[135] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society. 2, 24

[136] M.D. Smucker, J. Allan, and B. Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the 16th ACM Conference on information and knowledge management*, pages 623–632. ACM, 2007. 65

[137] B. Snyder, D. Bosanac, and R. Davies. *ActiveMQ in Action*. Manning Publications Co., Greenwich, CT, USA, 2011. 38

[138] H.J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The MicroGrid: A Scientific Tool for Modeling Computational Grids. In *In Proceedings of Supercomputing 2000*, December 2000. 29

[139] A. Songwattana. Mining Web Logs for Prediction in Prefetching and Caching. *Third International Conference on Convergence and Hybrid Information Technology (ICCIT)*, 02:1006–1011, 2008. 31

[140] D. Spiga, S. Lacaprara, W. Bacchi, M. Cinquilli, G. Codispoti, M. Corvo, A. Dorigo, A. Fanfani, F. Fanzago, F. Farina, O. Gutsche, C. Kavka, M. Merlo, and L. Servoli. CRAB: the CMS distributed analysis tool development and design. *Nucl. Phys. Proc. Suppl.*, 177:267–268, 2008. 36

[141] L. Breiman Statistics and L. Breiman. Random Forests. In *Machine Learning*, volume 45, pages 5–32, 2001. 26, 57

[142] M. Tadel. Gled–an Implementation of a Hierarchic Server–Client Model. *Applied Parallel and Distributed Computing (Advances in Computation: Theory and Practice)*, 16, 2004. 23

[143] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency and computation: practice and experience*, 17(2-4):323–356, 2005. 16

[144] I. Triguero, S. del Río, V. López, J. Bacardit, J.M. Benítez, and F. Herrera. Rosefw-rf: the winner algorithm for the ecbdl'14 big data competition: an extremely imbalanced big data bioinformatics problem. *Knowledge-Based Systems*, 87:69–79, 2015. 91

[145] P.E. Utgoff, N.C. Berkman, and J.A. Clouse. Decision Tree Induction Based on Efficient Tree Restructuring. *Machine Learning*, 29(1):5–44, Oct 1997. 26

[146] V.K. Vavilapalli, A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013. ACM. 4, 24

[147] D. Vohra. Apache Avro. In *Practical Hadoop Ecosystem*, pages 303–323. Springer, 2016. 34

[148] D. Vohra. Apache parquet. In *Practical Hadoop Ecosystem*, pages 325–335. Springer, 2016. 34

[149] D. Vohra. Using apache sqoop. In *Pro Docker*, pages 151–183. Springer, 2016. 34

[150] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012. 2

[151] H.B. William, G.D. Cameron, A.P. Millar, L. Capozza, K. Stockinger, and F. Zini. Optorsim: A Grid Simulator for Studying Dynamic Data Replication Strategies. *IJHPCA*, 17:403–416, 2003. 29

[152] D.R. Wilson and T.R. Martinez. Improved Heterogeneous Distance Functions. *J. Artif. Int. Res.*, 6(1):1–34, January 1997. 20

[153] W. Xu, L. Huang, A. Fox, D. Patterson, and M.I. Jordan. Detecting Large-scale System Problems by Mining Console Logs. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 117–132, New York, NY, USA, 2009. ACM. 18

[154] Q. Yang and H.Z. Haining. Web-Log Mining for Predictive Web Caching, Knowledge and Data Engineering. *Knowledge and Data Engineering, IEEE Transactions on Knowledge and Data Engineering*, 39:1050–1053, 2003. 31

[155] H. Yin-Fu and H. Jhao-Min. Mining web logs to improve hit ratios of prefetching and caching. *Knowl.-Based Syst.*, 21(1):62–69, 2008. 31

[156] M. Zaharia, M. Chowdhury, M.J. Franklin., S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association. 21

[157] X. Zhu and I. Davidson. *Knowledge Discovery and Data Mining: Challenges and Realities*. IGI Global, Hershey, PA, USA, 2007. 58