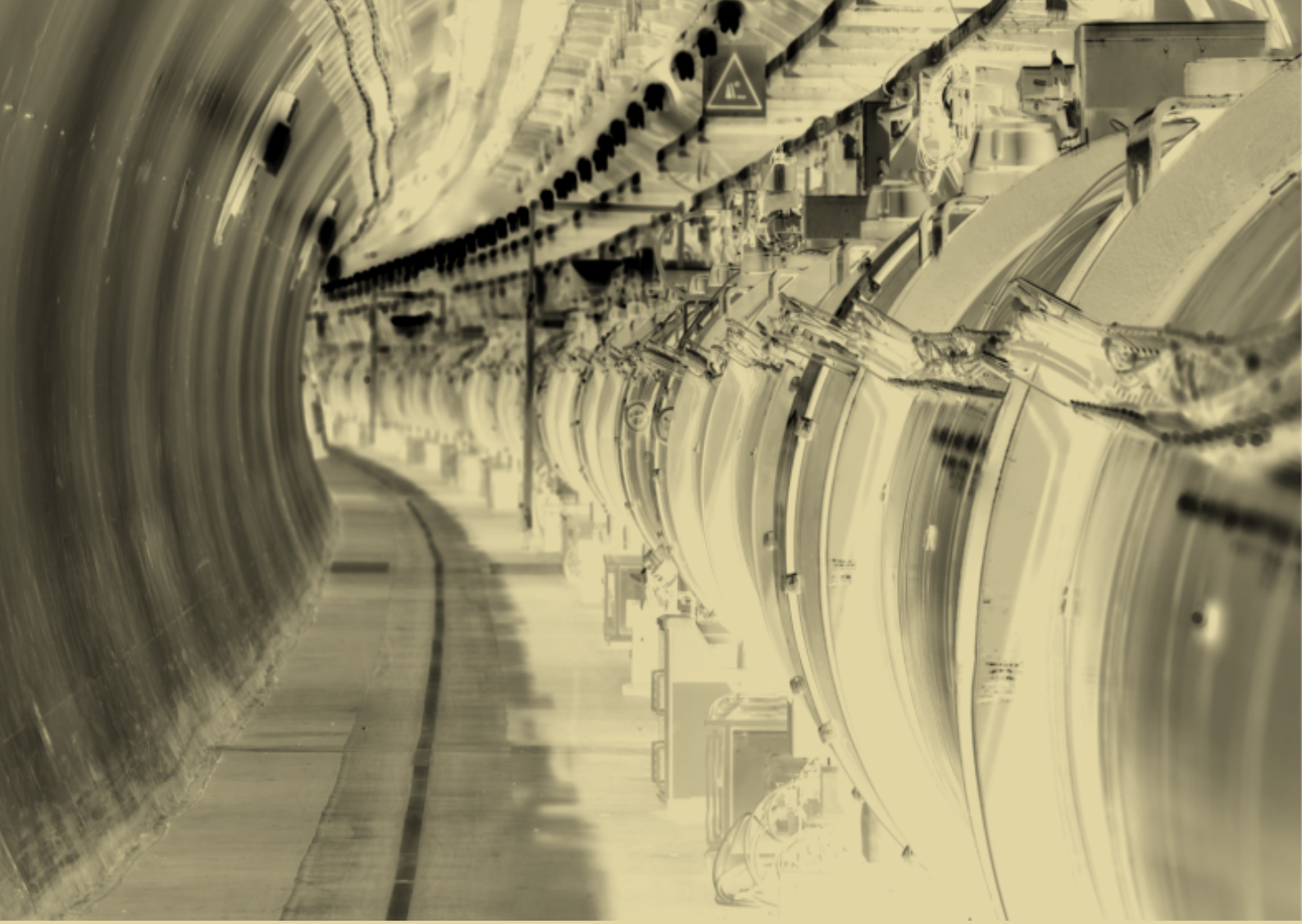


Serhiy Boychenko

A Distributed Analysis Framework for Heterogeneous Data Processing in HEP Environments

Tese de doutoramento
do Programa de Doutoramento em Ciências e Tecnologias da Informação
orientada pelo Professor Doutor Mário Alberto da Costa Zenha Rela e Doutor Markus Zerlauth
e apresentada ao Departamento de Engenharia Informática
da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Agosto de 2017

· U C ·

Universidade de Coimbra

A Distributed Analysis Framework for Heterogeneous Data Processing in HEP[1] Environments

Author:
Serhiy Boychenko
Department of Informatics Engineering,
University of Coimbra (UC)

UC Supervisor:
Prof. Mário Zenha-Rela
Department of Informatics Engineering,
University of Coimbra (UC)

CERN Supervisor:
Dr. Markus Zerlauth
Machine Protection and Electrical Integrity Group,
Centre Européen pour la Recherche Nucléaire (CERN)

February 22, 2018

[1]High Energy Physics

This research has been developed as part of the requirements of the Doctoral Program in Information Science and Technology of the Faculty of Sciences and Technology of the University of Coimbra. This work was conducted in the domain of large-scale distributed data storage and processing systems, joining the Machine Protection and Electrical Integrity group (Technology department), within the Doctoral Student program at the European Organisation for Nuclear Research (CERN).

This work has been supervised by Professor **Mário Alberto da Costa Zenha Rela**, Assistant Professor of the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra and Doctor **Markus Zerlauth**, Deputy Group Leader of the Machine Protection and Electrical Integrity Group of the Technology Department at CERN.

Abstract

During the last extended maintenance period, CERNs Large Hadron Collider (LHC) and most of its equipment systems were upgraded to collide particles at an energy level almost twice higher compared to previous operational limits, significantly increasing the damage potential to accelerator components in case of equipment malfunctioning. System upgrades and the increased machine energy pose new challenges for the analysis of transient data recordings, which have to be both dependable and fast to maintain the required safety level of the deployed machine protection systems while at the same time maximizing the accelerator performance. With the LHC having operated for many years already, statistical and trend analysis across the collected data sets is an additional, growing requirement.

The currently deployed accelerator transient data recording and analysis systems will equally require significant upgrades, as the developed architectures - state-of-art at the time of their initial development - are already working well beyond the initially provisioned capacities. Despite the fact that modern data storage and processing systems, are capable of solving multiple shortcomings of the present solution, the operation of the world's biggest scientific experiment creates a set of unique challenges which require additional effort to be overcome. Among others, the dynamicity and heterogeneity of the data sources and executed workloads pose a significant challenge for the modern distributed data analysis solutions to achieve its optimal efficiency.

In this thesis, a novel workload-aware approach for distributed file system storage and processing solutions - a Mixed Partitioning Scheme Replication - is proposed. Taking into consideration the experience of other researchers in the field and the most popular large dataset analysis architectures, the developed solution takes advantage of both, replication and partitioning in order to improve the efficiency of the underlying engine. The fundamental concept of the proposed approach is the multi-criteria partitioning, optimized for different workload categories observed on the target system. Unlike in traditional solutions, the repository replicates the data copies with a different structure instead of distributing the exact same representation of the data through the cluster nodes. This approach is expected to be more efficient and flexible in comparison to the generically optimized partitioning schemes. Additionally,the partitioning and replication criteria can by dynamically altered in case significant workload changes with respect to the initial assumptions are developing with time.

The performance of the presented technique was initially assessed recurring to simulations. A specific model which recreated the behavior of the proposed approach

and the original Hadoop system was developed. The main assumption, which allowed to describe the system's behavior for different configurations, is based on the fact that the application execution time is linearly related with its input size, observed during initial assessment of the distributed data storage and processing solutions. The results of the simulations allowed to identify the profile of use cases for which the Mixed Partitioning Scheme Replication was more efficient in comparison to the traditional approaches and allowed quantifying the expected gains.

Additionally, a prototype incorporating the core features of the proposed technique was developed and integrated into the Hadoop source code. The implementation was deployed on clusters with different characteristics and in-depth performance evaluation experiments were conducted. The workload was generated by a specifically developed and highly configurable application, which in addition monitors the application execution and collects a large set of execution- and infrastructure-related metrics. The obtained results allowed to study the efficiency of the proposed solution on the actual physical cluster, using genuine accelerator device data and user requests. In comparison to the traditional approach, the Mixed Partitioning Scheme Replication was considerably decreasing the application execution time and the queue size, while being slightly more inefficient when concerning aspects of failure tolerance and system scalability.

The analysis of the collected measurements has proven the superiority of the Mixed Partitioning Scheme Replication when compared to the performance of generically optimized partitioning schemes. Despite the fact that only a limited subset of configurations was assessed during the performance evaluation phase, the results, validated the simulation observations, allowing to use the model for further estimations and extrapolations towards the requirements of a full scale infrastructure.

Keywords: data processing distribution; data partitioning; data replication

Resumo

O Grande Colisor de Hadrões, construído e operado pelo CERN, é considerado o maior instrumento científico jamais criado pela humanidade. Durante a última paragem para manutenção geral, a maioria dos sistemas deste acelerador de partículas foi atualizada para conseguir duplicar as energias de colisão. Este incremento implica contudo um maior risco para os componentes do acelerador em caso de avaria. Esta actualização dos sistemas e a maior energia dos feixes cria também novos desafios para os sistemas de análise dos dados de diagnóstico. Estes têm de produzir resultados absolutamente fiáveis e em tempo real para manter o elevado nível de segurança dos sistemas responsáveis pela integridade do colisor sem limitar ao seu desempenho.

Os sistemas informáticos actualmente existentes para a análise dos dados de diagnóstico também têm de ser actualizados, dado que a sua arquitectura foi definida na década passada e já não consegue acompanhar os novos requisitos, quer de escrita, quer de extração de dados. Apesar das modernas soluções de armazenamento e processamento de dados darem resposta à maioria das necessidades da implementação actual, esta actualização cria um conjunto de desafios novos e únicos. Entre outros, o dinamismo e heterogeneidade das fontes de dados, bem como os novos tipos de pedidos submetidos para análise pelos investigadores, que criam múltiplos de problemas para os sistemas actuais impedindo-os de alcançar a sua máxima eficácia.

Nesta tese é proposta uma abordagem inovadora, designada por Mixed Partitioning Scheme Replication, que se adapta às cargas de trabalho deste tipo de sistemas distribuídos para a análise de gigantescas quantidades de dados. Tendo em conta a experiência de outros investigadores da área e as soluções de processamento de dados em larga escala mais conhecidos, o método proposto usa as técnicas de particionamento e replicação de dados para conseguir melhorar o desempenho da aplicação onde é integrado. O conceito fundamental da abordagem proposta consiste em particionar os dados, utilizando múltiplos critérios construídos a partir das observações da carga de trabalho no sistema que se pretende optimizar. Ao contrário das soluções tradicionais, nesta solução os dados são replicados com uma estrutura diferente nas várias máquinas do cluster, em vez de se propagar sempre a mesma cópia. Adicionalmente, os critérios de particionamento e replicação podem ser alterados dinamicamente no caso de se observarem alterações dos padrões inicialmente observados nos pedidos de utilizadores submetidos ao sistema. A abordagem proposta deverá superar significativamente o desempenho do sistema actual e ser mais flexível em comparação com os sistemas que usam um único critério de particionamento de dados.

Os valores preliminares de desempenho da abordagem proposta foram obtidos com recurso a simulação. Foi desenvolvido de raíz um modelo computacional que recriou o comportamento do sistema proposto e da plataforma Hadoop. O pressuposto de base que suportava a modelação do comportamento do novo sistema para configurações distintas foi o facto do tempo de execução de uma aplicação ter uma dependência linear com o tamanho do respectivo input, comportamento este que se observou durante o estudo do actual sistema distribuído de armazenamento e processamento de dados. O resultado das simulações permitiu também identificar o perfil dos casos de uso para os quais a Mixed Partitioning Scheme Replication foi mais eficiente quando comparada com as abordagens tradicionais, permitindo-nos ainda quantificar os ganhos de desempenho expectáveis.

Foi posteriormente desenvolvido e integrado dentro do código fonte do Hadoop o protótipo que incorporou as funcionalidades chave da técnica proposta. A nossa implementação foi instalada em clusters com diversas configurações permitindo-nos assim executar testes sintéticos de forma exaustiva. As cargas de trabalho foram geradas por uma aplicação especificamente desenvolvida para esse fim, que para além de submeter os pedidos também recolheu as métricas relevantes de funcionamento do sistema. Os resultados obtidos permitiram-nos analisar em detalhe o desempenho da solução proposta em ambiente muito semelhante ao real.

A análise dos resultados obtidos provou a superioridade da Mixed Partitioning Scheme Replication quando comparada com sistemas que usam o particionamento com único critério genericamente optimizado para qualquer tipo de cargas de trabalho. Foi observada uma redução significativa do tempo de execução das aplicações, bem como do tamanho da fila de pedidos pendentes, a despeito de algumas limitações em termos de escalabilidade e tolerância a falhas.

Apesar de só ter sido possível realizar as experiências num conjunto limitado de configurações, os resultados obtidos validaram as observações por simulação, abrindo assim a possibilidade de utilizar o modelo para estimar as características e requisitos deste sistema em escalas ainda maiores.

Palavras-chave: distribuição de processamento de dados; particionamento de dados; replicação de dados

To my family
To my beloved wife Violetta
To my wonderful children Nikolay and Vladimir
To my father Viktor and mother Lyubov

Acknowledgments

First of all, I would like to express my endless gratitude to the two people who were closely accompanying me during this long "marathon", sharing their knowledge and wisdom, guiding and teaching the research methodology and continuously supporting me in any of the undertaken tasks - my thesis supervisors Doctor **Markus Zerlauth** and Professor **Mário Zenha Rela**. Without their support, dedication and patience, completing this endeavour would have been significantly more difficult.

I would equally like to acknowledge and thank the following people who made significant contribution to this work:

Jean-Christophe Garnier for the exchange of ideas, discussions and great support during many presentations and meetings.

Andriy Boychenko for his patience, support and many discussions during the whole duration of this work.

Konstantinos Stamos, whose initial feedback and discussions had a considerable impact on some of the ideas presented in this work.

Tiago Martins Ribeiro and **Matei Dan Dragu** for helping me to configure the infrastructure deployment scripts.

Antonio Romero Marin and **Kacper Surdy** for providing and maintaining the required infrastructure for the prototyping phase.

Faris Cakaric for helping me with the implementation of the data migration tools.

Nuno Miguel Mota Gonçalves for helping with the implementation of the performance emulation application.

Kamil Henryk Krol for the great help with debugging code and environment configuration.

Jakub Wozniak and **Chris Roderick** for the many discussions about large-scale distributed data processing solutions.

The Powerlifting Club at CERN for the brief moments of distraction from the research activities.

Contents

List of Figures

List of Tables

Abbreviations

ALICE	A Large Ion Collider Experiment
API	Application Programming Interface
ATLAS	A Toroidal LHC ApparatuS
BLM	Beam Loss Monitor
CALS	CERN Accelerator Logging Service
CERN	Conseil Européen pour la Recherche Nucléaire
CMS	Compact Muon Solenoid
CPU	Central Processing Unit
CRUSH	Controlled Replication Under Scalable Hashing
CSV	Comma-Separated Values
DAG	Directed Acyclic Graph
ERMS	Elastic Replica Management System
FIFO	First In First Out
FPGA	Field-Programmable Gate Array
GFS	Google File System
HAIL	Hadoop Aggressive Indexing Library
HDD	Hard Disk Drive
HDFS	Hadoop Distributed File System
IoT	Internet of Things
JMX	Java Management Extensions
JSON	JavaScript Object Notation
JVM	Java Virtual Machine

LDB	Logging DataBase
LHC	Large Hadron Collider
LHCb	Large Hadron Collider beauty
MDB	Measurement DataBase
MIC	Maximal Information Coefficient
MPSR	Mixed Partitioning Scheme Replication
NTFS	New Technology File System
OLTP	OnLine Transaction Processing
PAX	Partition Attributes Across
PM	Post Mortem
QPS	Quench Protection System
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RDBMS	Relational DataBase Management System
RPC	Remote Procedure Call
SPSQC	Super Proton Synchrotron Quality Check
UFO	Unidentified Flying Object
XPOC	eXternal Post Operational Check
YARN	Yet Another Resource Negotiator
YCSB	Yahoo! Cloud Service Benchmark

Chapter 1

The LHC Accelerator Transient Data Analysis Framework

Since the earliest days of humanity, capacity of problem solving and learning was one of the characteristics, which distinguished us from animals. The natural curiosity and endeavour to understand the surrounding environment allowed us to progressively increase our knowledge, transfer it through the generations, make it available for the modern society. The most significant efforts to organize the intelligence acquired through many thousands of years of humanity were made by the first civilizations, with invention of the writing and reading. In ancient cultures archaeological findings suggest the existence of defined classification of the knowledge into the sciences like: medicine, mathematics and astrology.

Similarly to other elder sciences, Physics is believed to be first defined as a discipline by Greek philosophers. The name comes from the ancient Hellenic word *physis* which means *nature*, contextualizing the science as *natural philosophy*. Among the fundamental reasons which allowed the Physics to establish as an independent discipline was the unwillingness of the philosophers to accept the explanations for different phenomena provided by ancient religions and myths. The early studies were directed to provide a methodical explanation (supported by provable facts) for different events occurring in the surrounding world. The early experiments therefore focused on gaining knowledge and providing evidence about the most fundamental principles like the time and the composition of matter. It is Democritus in the 5th century BC who developed the theory of the *atomism*, which was claiming that everything is composed from very tiny, invisible elements called atoms.

During many centuries thereafter the definition *atomism* remained unstudied, gaining the new insights during the middle ages. Despite no significant advances

were made at the time, the works of scientists like Giordano Bruno, Thomas Hobbes and Galileo Galilei supported the dissemination of the idea into contemporaneous scientific communities. In the late 18th century, the advances of the scientific instruments allowed the supporters of the *atomism* to prove the philosophical assumptions with experimental results. However, with the discovery of the electron during the beginning of the 19th century, it became clear that atoms are not fundamental particles, but are composed themselves of even smaller particles. Through the past century the particle discoveries allowed us to explain numerous phenomena, create new technologies and provide the basis for the appearance of new theories like the Standard Model (Oerter, 2006), known as the main theory about fundamental particles and interactions between them. Still, many questions remain unanswered today and mankind will continue striving to expand the frontiers of knowledge to understand the laws of physics governing the world we live in.

1.1 CERN and the Large Hadron Collider

Among the particle physics research laboratories dedicated to study the nature of the universe, European Organization for Nuclear Research (CERN) is the largest facility, bringing together some 15000 scientists to build and operate particle accelerators and detectors in an attempt to unravel the mysteries of the universe. Since its establishment in 1954, it continues to be in the vanguard of science, successfully achieving the mission defined by the organization founders. During its operation several highly important discoveries were made. Among them, the weak force discovery in 1983 (*The Discovery of the W Vector Bosson.*, n.d.), which is responsible for particle decays (unlike the other three of the fundamental forces which do keep particles together: gravity, electromagnetism and strong force) and is a primary explanation for a Sun's radiance. Another very anticipated discovery was made public by CERN in 2012, where the scientists were able to observe a particle with a mass consistent with the sought-after Higgs Boson (Aad et al., 2012) (Chatrchyan et al., 2012), named after Peter Higgs who predicted its existence in 1964. This discovery allows physicists to explain why the particles do have mass and greatly increases the veracity of the Standard Model, which is currently the fundamental theoretical model to explain most of the phenomena observed in the universe. Despite Physics being the primary focus at CERN, one of the most important inventions greatly contributing to expansion of the human knowledge, the World Wide Web, was developed at CERN when scientists were searching for the ways to exchange results in an efficient and a fast way. Even nowadays, to push the frontiers of science, CERN continues to be the place where cutting edge technologies are being designed and brought to life.

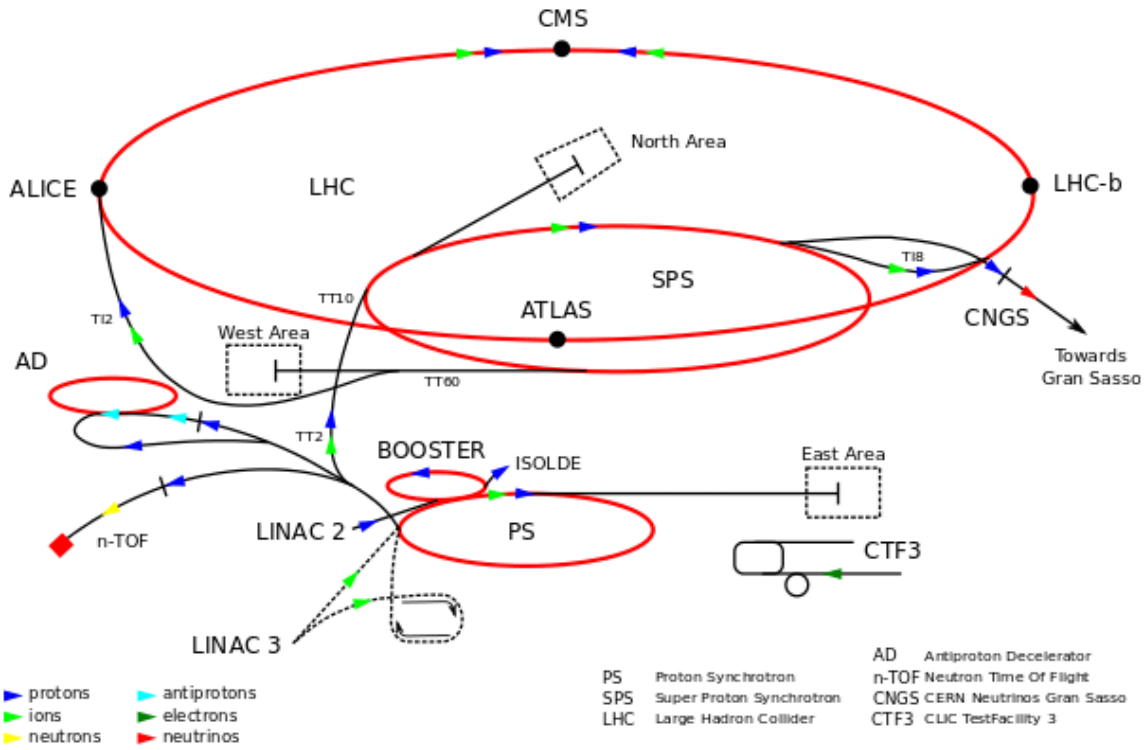


Figure 1.1: CERN's Accelerator Complex

One of the most ambitious and significant projects ever undertaken at CERN is the Large Hadron Collider (LHC), currently detaining the title of the biggest scientific instrument ever built in the world. An effort of several hundred of universities, more than 10000 engineers and scientists from more than 100 countries was required to bring the project to life. The LHC is located in the tunnel at approximately 175 meters beneath the surface, reaching almost 27 kilometres in circumference. The flagship of CERNs accelerator complex is composed of more than 10000 superconducting magnets, which allow to guide the two counter rotating beams of particles around the circumference while reaching 99.9999% of the speed of light. To achieve and maintain their superconducting state, magnets are cooled to 1.9 Kelvins, using the world's largest liquid Helium cryogenics systems. The superconducting state is required to achieve the strong (8.3 Tesla) and stable electromagnetic fields allowing for the precise measurements at these unprecedented high beam energy. In order to operate the accelerator at the nominal energy (currently 7 Tera electron Volts for hadrons and 1.38 Tera electron Volts beam energy for lead ions), the beams must be injected into LHC with very well defined parameters from the respective injector

chain, which is depicted in Figure 1.1.

There are four major particle collision detectors installed along the circumference of the Large Hadron Collider: ATLAS, CMS, ALICE and LHCb. At the four collision points, the beams are directed towards each other at determined angle to maximize the impact and probability of collisions to occur. Upon the impact, thousands of sensors installed on each detector are tracking the trajectory and the interaction with different forces of the produced sub-atomic particles. The collected measurements are further transferred to the first tier computing data centre, where the utility of collision is evaluated. The most interesting results potentially representing new physics (around 5% of data) are being pushed towards to the next computing resource tiers, which do search for specific phenomenon. When fully operational, the LHC is capable of producing around 1 billion collisions per second.

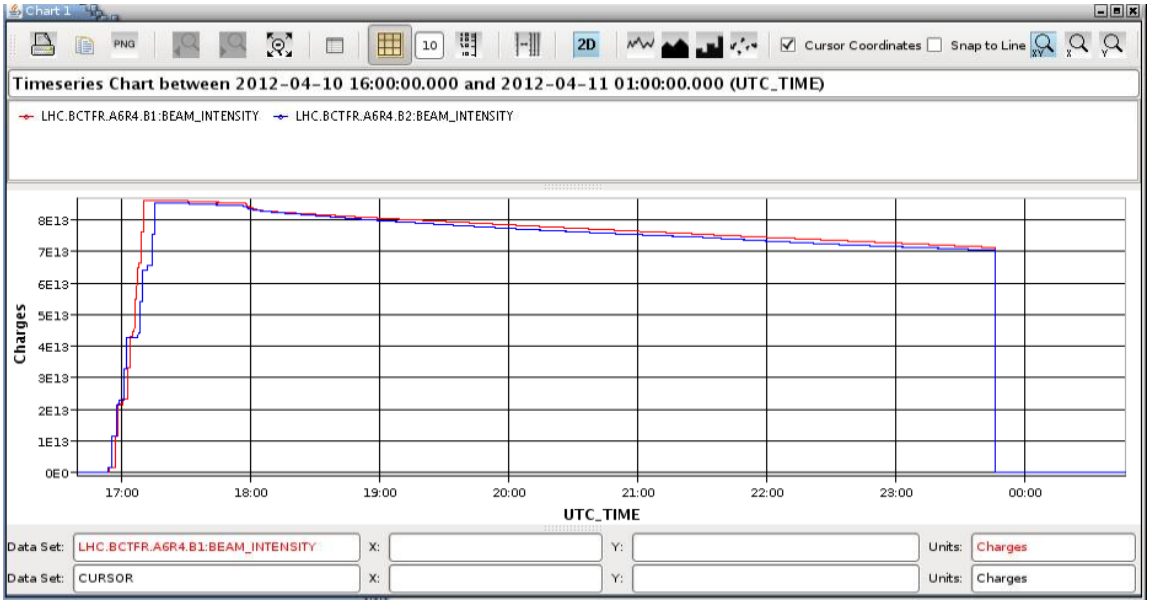


Figure 1.2: The typical intensity decay of both LHC beams during the nominal operation cycle, with the removal of the beams at the end of the physics run.

After the number of particle collisions in an experiment drops below a certain threshold, determined by the intensity of the beam and the beam size at the location of the experiment, the remaining particles are being extracted from the machine (see Figure 1.2). The energy stored in the superconducting magnets is extracted and returned into the grid. The extraction is done through the complex beam dumping system, which was specifically developed to safely deviate the beams onto a 10

meters long graphite target and as such prevent equipment damage of accelerator components. The same system is also used to empty the LHC when an equipment failure is detected prior to the planned termination of a physics fill. In this case the delay between fills is more significant, since the cause of the problem must be investigated by hardware experts and LHC operators, who determine if it is safe to operate the accelerator, otherwise preventive maintenance of the equipment installed in the tunnel is performed.

1.2 LHC Protection Challenges

The huge investment in both, manpower and the cost of the components which were required for the construction of the Large Hadron Collider, is one of the main reasons why considerable efforts have to be undertaken to maintain the machine safe under any operational conditions. The cutting edge technology systems installed in the tunnel, besides operating in extreme conditions (like high radiation or extremely low temperature), have to cope with unprecedented amounts of stored energies in the LHC magnet system (10 Gigajoules) and the two particle beams (365 Megajoules). These energies, for example, are capable of warming up and melting more than 2 tons of copper, therefore representing a considerable risk to the unique accelerator components installed along the beam lines. The extreme conditions which some of the components are exposed to, might not even be reproduced in large scale simulations in laboratories. The environment which is being created in the tunnel is a real exploratory process and the scientists do discover determined properties only when the designed component is installed in the accelerator ecosystem and exposed to proton beams. Besides accelerator components, there are tens of thousands of custom manufactured devices which need to be properly maintained, to avoid machine downtimes due to internal component failures. The LHC diagnostic data require powerful networks of computing devices to ensure that the collection and analysis of the reported measurements are performed in-line with restrictive requirements.

The amount and diversity of the accelerator components installed in the different points of the accelerator infrastructure is among the primary sources of the failure source heterogeneity. According to the report (*Premature Dumps in 2011.*, n.d.) produced by the Machine Protection Performance and Evaluation section in 2011, there were 482 incidental beam dumps (the process of particles extraction from the accelerator beam lines), which resulted into a total of 64 days of LHC downtime. While most of the problems are solved within a timeframe of a few hours, more serious interventions such as a magnet exchange in the tunnel may require up to several months due to the time-consuming warm-up and re-cooling processes. The

main sources of the issues which contributed to the machine downtime in 2011 were: cryogenics, powering systems, beam-machine interactions, radio-frequency cavities, vacuum, among others.

The consolidation and machinery upgrades during the recent extended maintenance period (long shutdown), have significantly increased the beam energy, currently measuring the 13 Tera electron Volts (TeV), meaning that the damage potential, in case of the serious failure, is much higher in comparison to the previous experience and estimations. The systems which protect the LHC have similarly undergone major improvements based on the experience collected during the first operational run between 2010 and 2013. In addition to the significantly increased number of machine protection devices installed in the tunnel, the amount of data which they produce also increased. However, the data storage and processing infrastructure, which is currently being used for the accelerator monitoring, failure source discovery and continuous performance surveillance, requires a major upgrade to keep up with the LHC expansion.

1.3 Diagnostics LHC Data Storage and Processing Infrastructure

There are two major sources of LHC diagnostic data: Post Mortem (PM) framework (Zerlauth et al., 2009) and CERN Accelerator Logging Service (CALS) (Roderick, Hoibian, Peryt, Billen, & Gourber Pace, 2011). Despite the fact that in most cases both solutions acquire and store the measurements collected from the same equipment there are fundamental differences in the covered use cases. Among the main differences between the two frameworks is the rate of acquisition and the precision of collected data. The Post Mortem system aims to collect the data around interesting events, such as the beam extraction from the machine, by retrieving high precision measurements from the internal buffer of the monitored hardware. Internal device buffers record the data with very high frequency (up to nanosecond precision), allowing to reconstruct very precisely the accelerator and equipment system state right before the beam extraction. Given the amount of the devices and produced data size it is impossible to store all this information with the same acquisition frequency during LHC operation. On the other hand, the data collected within higher time intervals, might provide broader overview on the problem sources. This requirement is satisfied by the CALS, which is responsible for retrieving the data with a maximum frequency of a few Hertz. The collected information is used not only for identifying and understanding the failure sources, but also to conduct the long-term performance

analysis of the LHC.

The Post Mortem framework (see Figure 1.3) consists of multiple redundant high performance servers with configured RAID $0 + 1$ hard disk storage. The underlying architecture makes strict separation between data collection and data consumption processes. The data collection tier consists of the two physical nodes, with individual storage, providing to the clients a fallback mechanism when some of the machines is unreachable. The data is acquired through the exposed Application Programming Interface (API) which allows the respective clients to submit the data when an event of interest occurs. The data submission might take up to 20 minutes for a given event, as some devices, before transferring the data to the PM storage, internally aggregate the information from multiple sub-systems. After receiving the data the PM framework persists the measurements into the binary files using a specifically designed compression and structuring file format. The files are organized according to the system type and the date when the device reported the measurement. Additionally the journal file, currently used as an indexing mechanism, updated every time the system persists the new measurements. Further, the data consumption layer is notified. The servers in this group are responsible for managing subscriptions from external applications (which are subscribed for the changes from determined devices or device types) and propagating the information further to registered consumers.

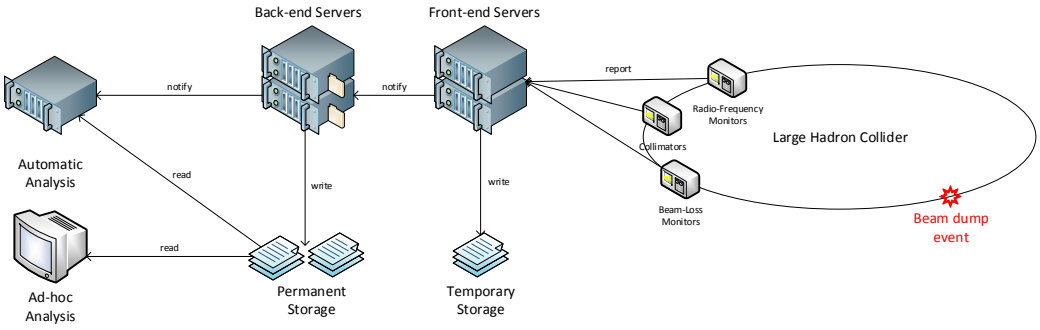


Figure 1.3: Post Mortem framework and its interaction with LHC.

CERN Accelerator Logging Service (see Figure 1.4) consists of two Oracle databases installed on the dedicated high-performance server machines, capable of handling the high loads which the service is exposed to. Unlike the Post Mortem framework which provides an API for data submission, CALS performs subscriptions for pre-configured list of devices and retrieves the data according to defined thresholds (either on value change or continuously with pre-defined interval). The measured signal values are

first being stored into a Measurement Database (MDB). The information remains in the MDB for a few weeks before being filtered, made read only and moved to the Logging Database (LDB). This approach allows to perform the queries the fresh data more efficiently, since the size of the MDB is two orders of magnitude smaller in comparison to LDB. Currently CALS monitors around 5×10^5 variables, which in the worst case scenario can generate approximately 4.23×10^{10} updates per day, at the same time serving 5×10^6 daily data access requests. CALS is an operation critical service, therefore some significant restrictions are applied on its usage. Besides the development team, no other user is granted direct access to the database. Service users have access to the API which provides a set of predefined operations with optimized queries with some per-query restrictions implemented (for example the maximum amount of data which can be retrieved for any request executed from the API is 250 Megabytes).

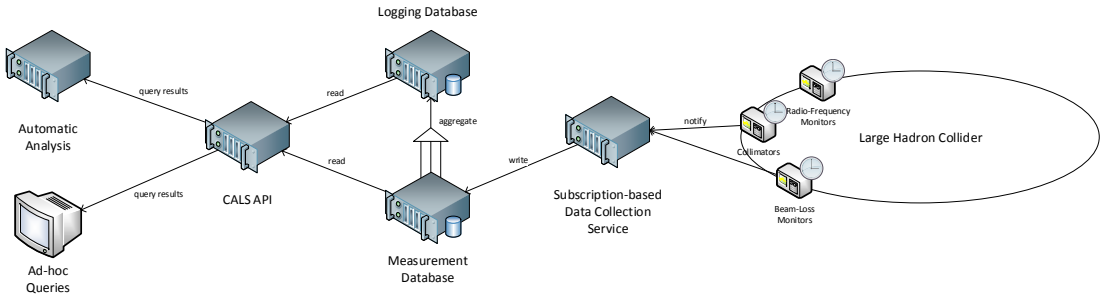


Figure 1.4: CERN Accelerator framework and its interaction with LHC.

Each of the major data repositories has its own data extraction and analysis tools. There is currently no easy way to correlate the data, unless explicitly developing dedicated tools using the provided interfaces. Thus, a sophisticated data analysis is tightly limited by available operations, depending on the personal capacities and manual effort of each operator to be able to join the data from different sources. Besides the mentioned limitations, developers must also take care of the data normalization, since the meta-data and data formats vary across different APIs. Further, after having the data appropriately merged, it is necessary to make signal transformations, since some formulas require input from either combined signals or modified signals (for example when a retrieved unit does not correspond to the required one). Currently deployed data correlation modules are maintained separately by the teams which originally performed the development. In the majority of cases those modules are not reused when another team requires to perform similar calculations, because

of the lack of a centralized repository which would take care to describe what is the purpose of the available code.

Another significant obstacle which limits the experts to work efficiently with the stored data is its current data retrieval throughput. Despite the fact that PM framework is relatively fast to provide the data, CALS is not optimized for reading large amounts of data. As a result, for some LHC performance evaluation use cases it would take several weeks for the relevant data to be extracted from the database. As a solution to this problem, many teams at CERN develop their own databases to monitor the limited number of the desired device types. Although the solution generally works for specific groups interested in particular information, it results also in the unnecessary information and code replication, resulting in the considerable maintenance overheads and additional personnel and equipment costs. Finally, the APIs which the aforementioned services provide to access the data require programming knowledge. Users who might only have a very basic software development background are forced to use the provided tools, limiting in the end their possibilities of working efficiently with the data.

1.4 Second Generation Data Analysis Framework

The synthesis of the shortcomings in the current process of the accelerator data analysis indicates that there is a strong call for a novel, centralized data analysis framework (Fuchsberger, Garnier, Gorzawski, & Motesnitsalis, 2013), capable of serving different purposes simultaneously. The experience acquired during the first LHC commissioning and operational periods, indicates a high level of heterogeneity in the workload to be handled efficiently by the new solution. When the accelerator systems are undergoing a series of commissioning steps before an operational period, the data of interest is retrieved for a period ranging from tens of minutes to a maximum of a few days after the performed hardware test. The most relevant attributes for processing of the data are hereby the device location, device type or accelerator status. When the machine is in the standard beam operation the paradigm changes significantly, and monitoring data related to the most recent physics fills often is compared to data from the past few months to perform trend, performance analysis and optimisations over extended period of time. During the long shutdown phase, typically more extensive analysis of the overall performance of the LHC related to the the past operational period is conducted, to detect the most common failure sources or to study the efficiency of the machine operation and the resulting physics output. These latter use cases typically require the extraction of data for large period of time, ranging from a few months to a couple of years.

The analysis of the shortcomings of the currently deployed data storage frameworks suggests that the new system, besides being capable of retrieving the data efficiently from underlying storage, should also be flexible enough to perform calculations close to the data source. This approach will allow to reduce significantly the data transfer overheads and the number of failures due to possible network problems. It is very common that an expert is interested only in a small fraction of data resulting from simple aggregation operations, but the limitations of the current systems impose the transfer of the whole dataset in order to perform the required calculation on the user's infrastructure. The data transfer and further processing could additionally optimized through the efficient caching mechanisms integrated into the future infrastructure.

The importance of keeping the machine operational and to detect possible causes of failures at an early stage, imposes another very important requirement - the ability of the proposed system to handle failing nodes without a major deterioration of the quality of service. Ideally, the solution should be automatically recovering from crashes and support further replication to maximize the data availability. Since there are time-critical user requests to be executed on the data storage infrastructure, the system should minimize the computation overhead during the node failures. Finally the system should be horizontally scalable, so the new nodes can be connected to the cluster when required to further improve the performance of the data storage and processing solution.

Among the main constraints which prevents the efficient exploitation of today's system and application of the further improvements is the performance of the data retrieval from the underlying storage layer. The limited access the hardware experts and accelerator operations have to the resources, prevents the implementation of the optimizations which would allow the analysis framework to satisfy many of the aforementioned requirements. Despite the fact that there are multiple tools and techniques which could improve significantly the current system performance, the lack of the flexibility for heterogeneous workloads and environments will inevitably uncover the limitations of the new infrastructure based on the such storage and processing solutions. As a response to the challenges presented above and as a call to substantially improve the performance of the data storage and processing system being developed for transient accelerator data, a new technique - the Mixed Partitioning Scheme Replication (MPSR) approach is proposed.

The main goal of this thesis it therefore to determine whether data replication which uses a multi-criteria partitioning can replace generically optimized data structuring schemes and improve the performance

of data storage and processing systems operating in highly dynamic and heterogeneous environments.

The diagnostic accelerator data which is currently being collected by multiple data storage and processing solutions has predominantly time-series nature. The frameworks which are storing the information take into account only the time dimension, and completely neglect other dimensions like device type, location, accelerator state and events which occur at the time when data collection takes place. The information related to the other data dimensions is either stored as meta-data or simply discarded, leaving the responsibility to make correct assumptions entirely to the users. The neglection of data characteristics and specificities in the currently deployed infrastructure creates an opportunity to explore the potential gains from different data replication and partitioning techniques. It is important to mention, that in the current state the meta-data stored along with the collected signals, does not really provide the means to unambiguously understand what a given variable is representing (besides using description fields which are not optimized for machine processing). The formulas which are used to correlate some of the signals into the corresponding variables are being applied before data is persisted and the information how this value was calculated is not exposed to the experts. Additionally, different storage systems might use different denominators and units for the same physical signal, resulting in additional calculations to be performed when the analysis of the respective variables is performed.

1.5 Contributions

The research activities which were undertaken during the development of this thesis were aiming to design an efficient, yet flexible solution for the large-scale distributed data storage and processing solutions, which integrate well into the dynamic environments as they are predominant for the operation of CERN's accelerator complex. The main outcomes of the endeavor can be summarised into the list of the contributions presented below:

- The conceptualization of a novel approach, based on replication of a multi-criteria partitioned data representation, for large-scale distributed data storage and processing solutions.
- The formalization of the proposed solution, Mixed Partitioning Scheme Replication, and the development of a simulation engine capable of estimating possible performance gains in relation to the traditionally applied approaches.

- A detailed analysis of the simulation results and the definition of the efficiency boundaries of the proposed solution, which can be used by other users to determine whether their use cases are compatible with the Mixed Partitioning Scheme Replication approach.
- The definition of the comprehensible and flexible Mixed Partitioning Scheme Replication architecture solution for further integration with modern data storage and processing solutions.
- The development of a functional prototype for assessing the performance characteristics and validating the predictions of the proposed approach. The sources were integrated for this prototype with the Hadoop source code.
- The validation of the model based on the replication of the multi-scheme data representation by comparing it with already existing, accurate estimation methods as well as through a set of specifically designed experiments.
- A detailed performance evaluation, scalability and failure tolerance analysis of the proposed approach for different possible scenarios and use cases.

Outlines

The remainder of the document is organized as follows: chapter 2 provides a detailed review of the literature related to similar data partitioning and replication techniques. In chapter 3 the proposed solution is formalized, and initial performance assessments, based on the simulation results, are conducted. Chapter 4 advances the definition if the proposed architecture and its integration into a modern data storage and processing framework. In the same chapter the description of the proposed prototype approach is detailed, as well as defining the performance evaluation scenario and infrastructure configuration. In chapter 5, the performance assessment and model validation tasks are undertaken. The results of a detailed study of the efficiency and the main characteristics of the proposed solution are provided. The final chapters outline the future work and summarize the document with the main conclusions and findings of the conducted research.

Chapter 2

State of the Art

The performance optimization of data storage and processing solutions is a problem which, despite being an active area of research for more than half a century, continues to attract and inspire researchers from all over the world. This research field has gained significant relevance and attention with the introduction of Relational Database Management Systems (RDBMS) (Astrahan et al., 1976), encouraging vendors of database engines to invest significant resources and compete with each other to provide the best possible solution to their clients. Despite the fact that in the turning point of the millennium (Abiteboul et al., 2005) the field was very mature, the introduction of user-centred services (also known as Web 2.0) has started a new era of data storage and processing solutions, with numerous new performance optimization domains to be explored (R. Agrawal et al., 2008). Not only modern Web applications have to deal with unprecedented data set sizes: scientific experiments have also significantly increased the amount of information which is required to collect and analyse, well beyond the order of magnitude where traditional solutions based on RDBMS or conventional file systems have proven feasible in the past. The expansion of applications which meet the requirements of Big Data use cases creates a high demand for specialized solutions, targeting specific systems with particular characteristics. Nowadays, the research in the area of performance optimizations of the relevant systems is very active and – given the rate of appearance of the new problems and solutions – will certainly continue to be so in the near future (Nasser & Tariq, 2015) (Sivarajah, Kamal, Irani, & Weerakkody, 2017) (Wang, Xu, Fujita, & Liu, 2016).

There are several possible ways of improving the performance of data storage and processing solutions. In many cases, a viable option was and continues to be improving the hardware where the system is installed. This can be achieved either by adding

additional resources to the existing machine(s) or by upgrading system components with more efficient ones. Despite the fact that custom, hardware-based optimizations can bring significant performance improvements to the database storage and processing solutions (multiple Field-Programmable Gate Array (FPGA)-based approaches have been proposed by different authors (Casper & Olukotun, 2014) (Sukhwani et al., 2012)), the most viable and popular option within the community continues to be resource scaling. The performance of the applications which were designed to store and process data on a single node can be improved by upgrading the respective machine (vertical scaling). On the other hand, the architectures which allow the distribution of the load amongst multiple interconnected computing nodes, become more powerful after cluster enlargements (horizontal scaling). The monolithic architecture of transient accelerator data storage and processing solutions currently deployed at CERN does not allow a straightforward and efficient scaling of the Post Mortem and CERN Accelerator Logging Service systems. Therefore, at this moment in time, the only viable option of performance improvement for both is an upgrade of the underlying hardware. In both systems, the components responsible for storing the data, query scheduling and execution are tightly coupled and cannot be separated into independent distributed modules. Through the past years of operation, the PM and CALS systems have undergone several costly hardware upgrade interventions which, according to previous experience, will require increasingly large investments to achieve further performance improvements.

Significant improvements of the data storage solution performance can be achieved by optimizing the underlying system for specific use cases by means of more efficient techniques like indexing or partitioning. Besides data related optimizations, the RDBMSs provide the specialists with a variety of different configurations which determine the way queries will be handled by the system (constraints, locking mechanisms, etc). In most cases an efficient caching solution can bring significant performance gains, which is notably the case for read-intensive workloads. In case of the data storage solutions presently in use at CERN for the analysis of the transient accelerator data, the performance tuning process is a continuous effort as the workloads executed on the systems have drastically changed over the first decade of operation.

In addition to the previously mentioned performance optimization techniques, the CALS development team was forced to introduce data retrieval size limitations in order to guarantee the allocation of the certain amount of the resources for performing the ingestion of the newly reported LHC device measurements into the system. Furthermore, a short-term storage layer was implemented in order for the most recent data (which in general is accessed more frequently by the users) to be retrieved

faster and to allow aggregation of the signal measurements into permanent storage without losing precision. In the case of the PM system, the storage was also split into multiple tiers, whereas the first layer is used for persisting the data as fast as possible, while the second layer is used to enhance the collected information with additional meta-data and persist the files into a redundant disk array. Although there are still many possible performance optimizations which could be implemented on both systems, they are not considered sufficiently satisfactory nor scalable in view of the desired mid- and long-term evolutions of the system.

Finally, to improve the performance of the systems deployed at CERN, one could consider switching to a modern, fully distributed data storage and processing solution. Many companies and scientific organizations which faced similar challenges, reported that the migration of their applications to distributed architectures like Hadoop (White, 2012) has enabled the users to work with very large data sets much more efficiently in comparison to RDBMSs or related technologies. The initial proof of distributed data storage and processing solutions to efficiently deal with such large datasets was reported by Google in MapReduce (Dean & Ghemawat, 2008) and Google File System (Ghemawat, Gobioff, & Leung, 2003) papers. The authors present a simple, yet powerful distributed programming paradigm, which requires the user to define in two phases how the data will be processed while the framework will take care of the actual execution (including the task distribution, failure recovery, etc). The solution presented by Google marked a considerable breakthrough for many companies facing Big Data problems. Consequently, sites like Facebook (Borthakur et al., 2011), LinkedIn (Sumbaly, Kreps, & Shah, 2013) and Yahoo (Boulon et al., 2008) have successfully implemented architectures based on MapReduce and GFS models to overcome the limitations of traditional RDBMSs and improve the performance of their data intensive applications. Following this, researchers from all over the world started to analyse if the proposed approach could also be suitable for different scientific applications. Many papers and reports from these efforts are proof of the successful identification of considerable performance improvements for many of the studied use cases (Ekanayake, Pallickara, & Fox, 2008) (Taylor, 2010) (Loebman et al., 2009). One of the most important advantages of large-scale data storage and processing engines based on Google's reports (like Hadoop), is their capability of scaling almost linearly, even if implemented on heterogeneous hardware. This is an extremely important issue for organizations such as CERN which maintain clusters with thousands of (often very distinct) nodes, since horizontal scaling is much more cost-efficient in comparison to solutions relying on vertical scaling techniques. According to the report presented internally at CERN (*Evolution of the Logging Service: Hadoop and CALS 2.0.*, n.d.) the new Hadoop-based infrastructure would cost more

than 3 times cheaper in comparison to the currently deployed solution.

2.1 Distributed Architecture for Performance Improvements

At this moment in time, the distributed storage and processing research field is flourishing. Currently, users have an abundant choice of tools which can be adapted to a large variety of use cases. Among the most common tendencies in the architecture design of modern distributed storage and processing solutions is is component coupling loosening which increases the independence between their core components: storage, resource management and processing layers. Besides leading to easily achievable performance gains using horizontal scaling, the independence of the system components allows – with a simple change in configuration – to replace specific application module with more efficient solution. However, the storage layer always requires a data migration, as each engine features its own storage format and data distribution strategy. Despite the fact that in some cases the architecture combines the resource management and the processing layers, in this work these aspects will be considered separately, as they both have a unique set of performance optimizations worth analysing in more depth.

2.1.1 Processing Layer

The data processing layer plays a fundamental role in the performance of distributed storage and processing solutions for large data sets. Multiple factors must be considered before adopting the most adequate tools for a specific use case. First of all, the workload heterogeneity should be considered, as most of the tools are optimized for analytic workloads, while in case of the LHC data storage and processing solutions, the operational queries are likely to remain predominant. Moreover, the requirements for a next generation solution at CERN suggest that the system should deal efficiently with iterative workloads as well (mainly for the Machine Learning use cases). Amongst the most popular solutions used in the past was MapReduce approach, which was integrated as a core component into the Hadoop eco-system to enable the processing of the large data sets. The execution is split into two programmable phases and a predefined intermediate phase: *map-shuffle-reduce*. The *map*phase implementation allows users to define the process of data filtering and grouping, while the *reduce* phase allows the definition of operations over the grouped data sets. Additionally, there is a *shuffle* phase which is an intermediary, non-programmable, stage

which merges the mappers output (possibly stored across different nodes of the cluster), sorts and transfers the data to the reducers' locations (the mapper outputs serve as the input for reducers). The transitional data produced by the mappers is stored on the processing machines Hard Disk Drives, being one of the greatest strengths but also weakness of the MapReduce paradigm. This mechanism allows Hadoop to recover from failures quite fast, as the data can be read back from the disk and applications can be restarted from an intermediary state. On the other hand, the random writes to the disk, resulting from the shuffling phase, can significantly slow down any job being executed on the same node due to multiple concurrent accesses to the disk.

Some researchers have found that in-memory processing of intermediary data could be used to improve the performance of the processing layer. Amongst the most popular solutions which enhance the performance of MapReduce with in-memory processing are Spark (Zaharia et al., 2016) and Flink (Carbone et al., 2015). Both propose unique memory management mechanisms which in the case of Spark are in general specified by the user, while Flink, to overcome the Java Virtual Machine (JVM) memory management limitations automatically executes the native memory management. Both tools provide intermediary checkpoint mechanisms which allow the user to control if and when the data should be persisted to the disk. Therefore, computations can be restarted from that point after the system recovers from a failure. Additionally, Spark and Flink allow for caching of intermediate results, becoming extremely efficient for iterative processing, which is very common e.g. when using Machine Learning algorithms. Finally, both tools are flexible enough to allow the integration with different distributed file system solutions and resource management applications. Currently, Spark and Flink are being studied by researchers at CERN to determine which of the two solutions provides the best feature set for the LHC transient data analysis use cases.

2.1.2 Resource Management Layer

The resource management layer has a significant impact on the efficiency and performance of modern distributed storage and processing solutions. This layer is responsible for managing the computing resources, allowing the nodes to be connected or decommissioned from an existing infrastructure. Additionally, one of the core responsibilities of the resource manager is the scheduling of the incoming request execution taking into consideration the current state of the cluster. Amongst the most popular solutions integrated into the infrastructures which require processing the large data sets is Yet Another Resource Negotiator (YARN)(Vavilapalli et al.,

2013). YARN was developed to become a core resource management component of the Hadoop eco-system, as its predecessor in earlier Hadoop versions revealed significant performance and scalability issues. One of the major breakthroughs of the new implementation was the decoupling from the MapReduce programming paradigm, which resulted in a wide adoption of YARN with all kinds of data storage and processing tools created for large data set analysis. One of the major advantages of this resource management solution is locality awareness: by performing the task execution close to the data YARN significantly reduces the probability of overloading the network with unnecessary information exchange between the cluster nodes, which results in addition in much lower latency. YARN also features different types of schedulers, which can either equally share the available resources amongst the submitted applications or dedicate more computational slots to specific, high-priority tasks.

Besides YARN, there are several other alternatives for the implementation of the resource management layer which are being actively integrated into different infrastructures by the respective developers. First of all, many of the modern distributed data processing applications come with dedicated solutions, specifically developed for optimizing the performance of the respective tool. In case of Spark, for example, the standalone scheduler has been designed for an easy and quick deployment, but is not recommended for production environments, as it lacks security-related features and does not allow the cluster to run anything else but Spark applications. Resource management solutions like Apache Mesos (Hindman et al., 2011) could be considered as an alternative to YARN for the data storage and processing solution being built at CERN. Mesos unlike YARN is a non-monolithic scheduler. After identifying the available resources for a determined user request, the framework allows the application to determine whether the execution should proceed on the available resources or should wait for an occasion when the cluster is less heavily used. Furthermore, Mesos allows to schedule the tasks based on the Central Processing Unit (CPU) requirements and availability, in contrast to the memory-only scheduling provided by YARN, being an advantage for CPU intensive applications. Despite being very flexible and providing more control over the scheduling process to the users, Mesos has several shortcomings which support the decision of favouring YARN in the next generation CERN data analysis infrastructure. First, Mesos does not respect data locality, which greatly impacts the system performance in case of many data intensive jobs. Secondly, Mesos does not allow to spawn multiple executors per node, which is not a resource efficient approach for operational workloads dominated by large amounts of small jobs (those which process small amounts of data).

2.1.3 Storage Layer

In modern data analysis infrastructures, the storage of data is a fundamental component of any Big Data solution. Being at the lowest level of the architecture it can provide the largest performance improvement with the smallest effort, and conversely can have a significant detrimental impact on any system when designed improperly. The design of the data storage architecture is a complex process and requires a customized approach for choosing the appropriate storage solutions. Amongst the most reliable and popular solutions for storing the data in distributed environments we can find the Hadoop Distributed File System (HDFS) (Shvachko, Kuang, Radia, & Chansler, 2010). Two components are the main building-blocks of the HDFS architecture: the Namenode, which maintains the file system structure and meta-data, and the Datanode which stores the data arranged in blocks. The master-slave approach employed by HDFS in its current implementation has significant shortcomings, especially for high-availability use cases, since the Namenode cannot be fully replicated and shuts down the operation of the entire cluster in case of failures. A different class of solutions with a decentralized meta-data management approach has been recently gaining popularity in the large scale distributed computing community, Distributed file-systems like CEPH (Weil, Brandt, Miller, Long, & Maltzahn, 2006) and GlusterFS (Davies & Orsaria, 2013). Despite having some very distinct characteristics, both allow distribution of the master responsibilities among multiple nodes of the cluster. While CEPH delegates the indexing operations to the Metadata servers using the Controlled Replication Under Scalable Hashing (CRUSH) (Weil, Brandt, Miller, & Maltzahn, 2006) algorithm, GlusterFS makes use of the custom implementation of the Elastic Hashing Algorithm which allows the respective storage nodes – without communicating with any meta-data management service – to determine the location of the data.

Multiple studies of the aforementioned distributed file systems have been conducted by different researchers (Yang, Lien, Shen, & Leu, 2015) (Depardon, Le Mahec, & Séguin, 2013) (Donvito, Marzulli, & Diacono, 2014). According to these authors, GlusterFS has shown the best I/O throughput when under heavy load, and Hadoop the best reliability results. According to the CEPH study performed at CERN (Van Der Ster & Rousseau, 2015) some scalability and availability issues were detected related to the addition of new cluster resources, while the authors of (Donvito et al., 2014) were experiencing issues with rebalancing on large clusters. The reliability of HDFS was a determining factor for choosing it as a storage technology for the new CERN transient accelerator data recording system, as it is of utmost importance for the performance of CERNs accelerator complex to provide a reliable and continuous service for data storage and extraction throughout the whole

lifetime of the accelerators.

After assessing multiple optimizations of the data storage and processing techniques, the interaction points with the proposed novel approach - Mixed Partitioning Scheme Replication - were identified and following state-of-the-art studies were focused on the specific solutions for further storage layer optimizations applicable to the accelerator analysis system use case challenges.

Partitioning

Data partitioning is amongst the front-line techniques used to optimize data manipulation operations in data storage systems. The first research works on this subject (Casey, 1972) (Eisner & Severance, 1976) appeared when different file systems were still being designed and gained the interest of additional researchers when the first Relational Database Management Systems emerged. The main underlying principle behind any data partitioning solution is the splitting of information into multiple independent parts to be stored on different physical locations. The most basic schema includes the master (or index) structure, which has sufficient meta-data to route incoming requests to specific locations containing the required fraction of the information. More complex systems are based on algorithms which tag the stored data structures with custom designed/computed key orders. Whenever a new request is received, the algorithm is able to determine the data location without recurring to a centralized indexing service.

Despite the fact that there is an ample choice of the strategies for determining the data division points and the final layout on the storage device, the data partitioning techniques presented by the scientific community can be categorized into two broad categories: horizontal and vertical partitioning (see Figure 2.1). Independently of the underlying algorithm's design options, the data is either split into a set of data objects which maintain the integrity of the original schema (horizontal partitioning) or the schema is split into multiple independent sub-schemas, allowing to maintain the data objects together (vertical partitioning). Independently of the chosen partitioning type, partitioning algorithms can be adapted to any storage node topology (master-slave or completely decentralized group) and use any of the aforementioned data division criteria (Hevner & Rao, 1988) (S. Agrawal, Narasayya, & Yang, 2004).

During the last decades different types of partitioning techniques were emerging, driven by a variety of studies related to the needs of application and system architectures. The simplest concept of delimiting information division boundaries is range partitioning (*Range partitioning.*, n.d.). Data object attributes are used to determine the relevant interval with corresponding start and termination points which define the

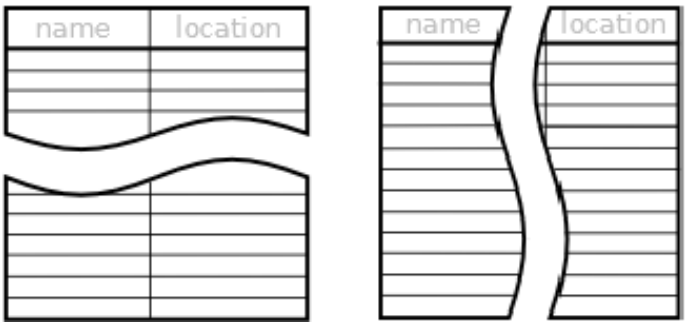


Figure 2.1: Horizontal(left) and Vertical(right) Partitioning.

boundaries of the corresponding partition. The main challenge for range partitioning is to achieve an equal balancing of the data. The initially defined division criteria which provides an even data distribution at the moment when there is a significant skew in the data might underperform, resulting in a completely unbalanced storage structure. A similar concept is used in the list partitioning (*List partitioning.*, n.d.) technique, where instead of relying on large intervals, the data division is driven by a set of entries in a list. The entries of the list correspond to the values of the specific and relevant data object attributes and are used to determine the partition which a certain data object belongs to. Similarly to the previously described criteria, the solutions which employ the range partitioning suffer from the data skew effect, but are at the same time more flexible to redefine data division points. Furthermore, there is a wide range of hash partitioning techniques, which are more efficient in terms of data balancing in comparison to the previously described strategies (*Hash partitioning.*, n.d.). The hash is calculated based on data attributes, allowing different types of the data objects to reside on the same node. Modern data storage solutions, like Oracle database system, implement balancing algorithms which determine the optimal data division strategies automatically, and take care of the re-partitioning when the data distribution becomes unbalanced. Finally, there are hybrid solutions which include the best characteristics of each approach and incorporate them into the same system (Taniar, Jiang, Liu, & Leung, 2000) (Furtado, Lima, Pacitti, Valduriez, & Mattoso, 2008).

Replication

The evolution of the Internet in the last few decades has led to an exponential growth of the amount of information which is being stored by countless servers across the world. While data replication was not a new concept at the time, an increased de-

mand for data processing and availability were among the main drivers for researchers to focus on finding more efficient ways of data replication. The main principle behind replication is to store multiple copies of the same data on different physical machines. Aided by load balancing techniques, replicated systems support an even load distribution between the nodes, therefore reducing the request response times and increasing availability. Different topologies and synchronization methods (Wiesmann & Schiper, 2005) (Kemme & Alonso, 1998) can be applied to the distributed nodes to ensure data consistency, which – together with additional concurrency - becomes a major concern for many systems. Globally distributed services use world-wide replicated content delivery networks to enhance the user experience and guarantee low response times for their customers, independently of their geographical location (Dilley et al., 2002).

Replication can be performed synchronously or asynchronously. Synchronous (or eager) replication uses sophisticated synchronization mechanisms to prevent data inconsistencies (Muñoz-Escoí, Irún-Briz, & Decker, 2005). The main principle behind techniques of this category is to use the algorithms which guarantee that updates, when they arrive, are successfully propagated throughout all of the nodes participating in replication process. On the other hand, asynchronous (or lazy) replication does not guarantee immediate consistency of the data. The information stored on different nodes may diverge at a given point in time, provided that the update propagation is done eventually or on demand. Generally, the synchronous approach is prone to suffer from data scalability problems while the second, asynchronous approach, features temporary data inconsistency. Therefore, semi-synchronous replication solutions have been implemented to mitigate some of the shortcomings of both solutions (Chang, 2005). They allow the system to scale well through the integration of the storage management orchestrating mechanisms –which perform background data integrity checks– and ensure –through consensus algorithms– that the user is reading the latest version of the stored information.

2.1.4 Comparison With Existing Solutions

The adoption of data organization techniques for optimizing the performance of data store and processing solutions has been widely studied in the context of the Online Transaction Processing (OLTP) workloads in RDBMSs. One of the most common solutions used to improve the efficiency of databases is indexing (Bertino, 1991). This technique relies on structuring the data in a way that queries executed on the system are processing sub-sets of the data rather than analysing every row of the table (known as full scan operation). Generally, database systems allow the creation

of several indexes on the same table which can be composed of multiple entity attributes. Most storage engines automatically determine amongst the indexes created by the users the one which potentially yields the most significant performance gain for a determined query before proceeding to the data processing. Furthermore, when database tables are becoming very large, the system architect can opt for splitting the table into smaller segments through a process known as partitioning. Consequently, the storage engine is able to manage *Partitioned Tables* independently, meaning that operations like data loading, index management and backup/recovery processes can be performed in considerably smaller amounts of time in comparison to single table architectures (Herodotou, Borisov, & Babu, 2011). Additionally, *Partitioned Tables* can be replicated throughout the cluster to increase the overall availability and I/O throughput of the system. Several authors (Quamar, Kumar, & Deshpande, 2013) (Curino, Jones, Zhang, & Madden, 2010) reported significant performance improvements when applying the aforementioned solutions in large data warehouses. Nevertheless, despite significant optimizations observed in different configurations, the usage of the developed frameworks for the LHC accelerator transient data storage is questionable, since the amount of stored information has grown well beyond the scale where RDBMSs have proven to provide optimal performance.

Further research presented in the following sections focus on solutions compatible with modern distributed storage and processing frameworks. The basic versions of the tools suitable for the analysis of Petabyte-scale data sets (like Hadoop) are already offering features which allow achieving higher throughputs, better scalability and fault tolerance characteristics in comparison to relational databases (Yu & Wang, 2012) (Hu, Wen, Chua, & Li, 2014). Nowadays, several user and development communities around the world are actively developing multiple open source Big Data projects (McKenna et al., 2010) (Wiley et al., 2011) (Ekanayake et al., 2008), which is a clear indication of the reliability and robustness of the provided solutions. The popularity and wide adoption of large scale data storage and processing solutions implies the development effort to be focused on major issues rather than creating highly specialized tools for limited set of use cases. This allows many research communities (Buck et al., 2011) (Karun & Chitharanjan, 2013) to take advantage of this solid code base to develop new compatible modules which provide specific optimizations for dedicated applications. The new solution for the LHC transient data recording will likely benefit from an "out-of-the-box" migration to Hadoop, but the performance and quality of service can be further improved by implementing more sophisticated and targeted optimizations. The most crucial part in this new system design is at the data storage level, as optimizations at the lowest level of the new infrastructure are likely to have a significant influence in higher-level components

in the analysis pipeline, and lead to higher performance gains. Therefore, we conducted a detailed study of possible improvements at this level. Based on the findings, a new data storage model, optimized for the workloads observed in the operational environment of the accelerator chain, is proposed by this thesis.

Data Placement Strategy Optimization

A detailed analysis of the literature in the domain of large scale scientific data analysis has revealed multiple solutions which could be applied to improve the performance of the systems being developed at CERN. Despite having similar objectives, the implementations which have been studied primarily achieve their goals by performing upgrades to different components of the data storage systems. We started by analysing solutions which focus on data placement strategies optimization, since in most situations the largest performance gains can be achieved simply by correctly structuring the data. The fundamental concept used for this type of solutions is partitioning. Partitioning in the context of distributed file systems is slightly different in comparison to partitioning in a relational database, but the basic idea remains the same – splitting the data into small segments to avoid reading the whole data set when processing the user requests. Partitioning techniques are applied to determine the optimal directory structure or file scheme in relation to particular workloads and efficiently distribute the data across the cluster nodes to balance the usage of the available resources.

The CoHadoop (Eltabakh et al., 2011) authors have developed a lightweight Hadoop's extension which allows user applications to control the data placement. In combination with the directory structure, this solution provides the means to define the co-location of data blocks to reduce the overhead of the data transfers, which occur mostly during the shuffle phase (known to be a significant issue for *join* operations for example). Co-location is hereby achieved by introducing a new file-level property, the *locator*, which maintains 1-to-N file relations, which is used by the modified data placement algorithm to determine the node which will store grouped assets. The management of the *locator* information and co-located files is done through mapping in the *locator table* integrated into the Namenode sources. The map structure is maintained in memory to speed up the incoming requests. Additionally, the *locator table* is persisted to the disk in background, for it to be easily restored after a Namenode failure. The results presented by the authors suggest that for the *join* operations, CoHadoop was approximately 2.2-2.5 times faster in comparison to the traditional solution. This improvement comes at the cost of cluster storage unbalancing, ranging from 8.2% to 12.9% (larger block sizes leads to higher unbalance).

In case of the LHC transient data recording and analysis system, CoHadoop could significantly improve the performance of the particular workload types, namely the time-based *join* operations. However, other existing query categories performing *joins* on different attributes have not improved or even be penalized, as a result of the introduced imbalances in the cluster resource usage. Additionally, CoHadoop is not resilient to workload changes, since modifications – once the storage strategy is implemented - cannot be reverted or modified. Finally, the CoHadoop project is not production ready and seems inactive, as the implementation was only available in very early Hadoop versions and no information of further migration to future releases is available.

The Elastic Replica Management System (ERMS) (Cheng et al., 2012) has been developed to take advantage of replication as the primary source of the performance optimization for Hadoop infrastructures. This solution introduces the active/standby storage model which automatically allocates or de-allocates the replicas to specific data segments, based on their popularity and observed access patterns. The infrastructure is continuously monitored by the Data Judge Module which, in order to obtain real-time data classifications, defines Complex Event Processing (Buchmann & Koldehofe, 2009) queries to be executed when new entries are written into the HDFS log files. Based on pre-defined metrics, the classifier assigns the following categories to stored data segments: hot, cooled, normal and cold data. Furthermore, the Replication Manager component determines whether it is necessary or beneficial to create additional replicas for applications to take advantage of a higher data execution container number, or to deallocate machines storing information which is no longer frequently requested. To reduce the number of machines storing cold data, the authors of ERMS have implemented the Erasure Coding module. The Replication Manager, while allocating the new replicas, examines the Datanode resource usage, in order to determine the underused nodes and performs a new load balancing of the cluster. The benchmarks executed by the authors show that ERMS is able to perform more efficiently in comparison to standard Hadoop deployments. The reading throughput results determine that the developed solution was 50-100% faster when using a First In First Out (FIFO) scheduler and 40-100% faster using a Fair scheduler when compared to traditional installations. Additionally, the observations demonstrated that ERMS manages to schedule more jobs which respect the data locality principle. When analysing a possible integration of the described framework within the new data storage and processing solution under development at CERN, several shortcomings have been identified. First of all, ERMS requires the availability of significant resources for replica creation, since the machines which belong to persistent storage category will need to allocate their computing resources for transferring the

data to temporary replicas (like *distcp* operation does when data is copied in Hadoop clusters). Upon execution, the data copy operation initiates multiple mapper jobs on the cluster, using the execution slots which could be assigned for data processing requests. Furthermore, the de-allocation of the replicas triggers the erasure coding process which, despite being more efficient in terms of saving storage space (in comparison to replication), is CPU-intensive for both constructing and recovering from encoded data. The authors did not provide any details on how ERMS is handling frequent short-term workload changes, as those could potentially trigger numerous resource re-balancing operations. Finally, the analysed solution does not address the *join* operations which, despite having a large number of data-local executions, would create significant I/O and network overhead during the shuffling phase, since the intermediary data needs to be merged at some point of the application execution.

File Format Optimizations

File format optimizations are amongst the solutions which have been steadily gaining ground in modern data storage and processing architectures. The main principle behind these techniques is to determine the best way of organizing (or partitioning) the data inside the files. The file format optimization techniques can be split into two broad categories, row-based and column-based, which differ in the way the data is handled when collected and stored. Row-based solutions generally do not introduce many changes while loading data into the system, which allows to maintain low data ingestion latencies. Therefore, the space reduction gains and the data retrieval rates are lower in comparison to column-based solutions.

Apache Avro (*Avro is a remote procedure call and data serialization framework developed within Apache's Hadoop project.*, n.d.), a row-based solution, has been initially developed as part of the Hadoop project for data serialization and transportation between cluster nodes. The Avro file format requires from the applications to define the schema of objects which are going to be persisted, meaning that the data must be structured. The schema is used to describe the object attributes with their respective data types, which can either be a pre-defined primitive or programmable complex types. Furthermore, the data can be encoded using different formats, either JSON or binary. While the first one is generally used for debugging, the second one is suitable for production environments as it is able to significantly reduce the required storage space. The binary encoding makes use of sophisticated encoding techniques, which minimize the number of bits required for storing the objects. Since sorting is considered to be a frequent operation in analytical workloads, Avro allows the data to be sorted inside the file according to configurable criteria. Accessing al-

ready sorted data significantly decreases the amount of resources required for this operation, since retrieving correctly ordered data does not require much CPU cycles. Finally, information can be compressed using per-block compression algorithms like bzip2 (Seward, 1996) or LZO (*LZO real-time data compression library.*, n.d.).

The RCFile (He et al., 2011) has been a popular choice for distributed data storage performance optimization in many modern columnar-store architectures. This solution organizes HDFS block data into fixed-size *row groups*. The *row group* is hereby composed of the following three sections: the sync marker used to define the group limits, the meta-data header describing the table columns and the data itself written in the column-based format. The columns of each section are compressed individually to enable partial file reads, as only those columns which are absolutely required for the query processing are decompressed. The RCFile does not allow arbitrary writes, only appending operations are supported. When the file is initially requested by the mapper, only the meta-data is loaded into the memory. Then, the columns required for processing are loaded and decompressed lazily, allowing to reduce the resource usage of the executed tasks. The authors have conducted a detailed performance study of the RCFile and compared the results with the raw files stored in HDFS and other file format optimization solutions. The obtained measurements show that RCFile required 2.2 times less storage than in a pure HDFS approach and was taking 1.1-1.3 less space in comparison to similar approaches. As for the data loading performance results, RCFile was slightly slower in comparison to the row-based solutions, but clearly outperforming other column-based file optimization formats. Additionally, it was performing the best for executing short running queries and had comparably high performance for the remaining analysed workloads.

For the solution proposed in this thesis and the second-generation infrastructure under development at CERN, file format optimization techniques are considered as a complementary performance optimization option. MPSR partitions the information on the directory level of the file system maintaining the HDFS data loading pipeline unmodified. This allows any file format optimization technique, compatible with Hadoop, to integrate with the proposed approach. Multiple solutions have been already evaluated by researchers at CERN and the detailed report of performance evaluation has been presented (Baranowski, Toebbicke, Canali, Barberis, & Hrivnac, 2017). According to this report, amongst the available and studied file format based solutions, Parquet (*Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.*, n.d.) (a column based solution very similar to the RCFile) was the most efficient in the execution of the analytical workloads.

Dedicated Columnar-Stores

Finally, dedicated solutions which introduce significant changes to the data storage layer were considered. These solutions have the drawback of requiring the implementation of additional changes in the resource management and data processing components. Generally, these solutions provide the best performance improvement results, but they drastically change the underlying solution and introduce significant cluster resource requirements (especially in terms of CPU and memory).

The Hadoop Aggressive Indexing Library (HAIL) (Dittrich et al., 2012) is the solution which has the most similarities in comparison to MPSR. The main idea of HAIL is to maintain the replicas of the HDFS blocks using different sort orders and with different clustered indexes. Since the default replication factor of the Hadoop systems is three, the authors have determined that using multiple data representations for each of the replicas will increase the likelihood of particular query types to find appropriately organized data. Hereby indexes are built by a modified data loading pipeline. The file, which is staged for writing in HDFS, is pre-processed taking into account the system configuration, the content and respective meta-data which are written to the corresponding Partition Attributes Across (PAX) (Ailamaki, DeWitt, Hill, & Skounakis, 2001) binary representations. Additionally, before being persisted to the disk, the data blocks are sorted according to the defined strategies and enriched with additional server-side index meta-data. After processing and persisting the information, the Datanode notifies the Namenode to update the block mapping. In order to allow the scheduling algorithms to pick the correct indexes (rather than the random ones) the new module was integrated into the Namenode sources. This component maintains detailed information about existing per-replica indexes. In addition to simple indexes, HAIL allows to create clustered ones. Therefore, when the number of the attributes to be indexed exceeds the replication factor, the user can define an index on multiple object attributes. The data reading pipeline has been equally modified by introducing a new splitting strategy and a new record reader implementation which –in the background– aggregates the determined data blocks. This strategy has proven extremely efficient to reduce the number of mappers required for data processing, since the number of splits can be drastically reduced. Additionally, the splitting technique ensures that an appropriate replica is being picked to increase the data locality in job executions. Whenever no appropriate index can be found, HAIL chooses the Datanodes which would allow to execute the calculations close to the data. The benchmarks performed by these authors have shown that HAIL performs much faster in comparison to traditional Hadoop both for writing and reading the data. According to their observed measurements, HAIL was executing the jobs in average 39 times faster in comparison to Hadoop. In ad-

dition, a 60% improvement of the data loading times was identified. Despite these very promising results, several issues were identified when analysing a possible integration of HAIL within the future accelerator data storage and processing solutions at CERN. First, the proposed approach requires significant amounts of memory allocated for the data uploading pipeline. The currently deployed systems for LHCs transient data recording are constantly under heavy I/O load, hence both writes and reads will be constantly and concurrently executed on the system. Given the amount of the input sources, HAIL would not only require significant amounts of memory to be reserved for creating the files on the data collectors side, but also for sorting and indexing on the Datanodes. Since the authors do not present any studies of the behaviour of their system with mixed workloads, it is impossible to determine whether HAIL will be efficient and scale accordingly for the workloads observed at CERN. Second, as today's implementation requires significant changes to the data storage format, upload and retrieval pipelines, the integration of dedicated data collection and processing tools, like Kafka (Kreps, Narkhede, Rao, et al., 2011) and Spark, would require significant effort. Consequently, the scope of the use cases covered by the new data storage and processing solution would be significantly limited (mostly the Machine Learning libraries (Meng et al., 2016) integrated with Spark). Finally, these authors do not present any strategies for re-balancing the infrastructure in case of addition or removal of the indexes, which might be required when adapting to further workload changes which will inevitably occur during the coming years of LHC operation.

HBase (Vora, 2011) is another columnar-store based on HDFS, which is widely adopted in many applications requiring random real-time queries on extremely large data sets. Since Hadoop is a batch processing system, the developers have re-implemented the data processing and scheduling components, using techniques suitable for on-line analysis. Moreover the storage layer was built on top of the HDFS, mainly due to its robustness, flexibility and lack of the shortcomings which would prevent the system from executing the real-time queries. The core of the HBase solution is the HFile, which is very similar to the MapFile - the default key-value solution provided along with Hadoop deployments. The major advantage of the HFile is due to the advanced meta-data and indexing features, which allow fast data lookups. The data inside the file is split into *in-line blocks* with individual indexes and Bloom filters. To build an efficient index, the information needs to be sorted, hence the data collection process maintains everything at first in memory. Once the maximum buffer size is reached, the data is structured, encoded and flushed to the persistent storage. Whenever data is requested for the execution, the specific block with the respective meta-data is loaded into the memory. The cluster resources are managed

by the HMaster module, which is also responsible for providing the interface for creating, deleting and updating the tables. The nodes are constantly monitored for load balancing and provide failure recovery. A special structure, the so-called HBase Meta Table, is used to keep track of all *regions* which are the segments of the horizontally partitioned tables. The Region Servers are running on the HDFS Datanodes and provide additional features for write-read caching and recovery of the information which was not yet persisted to the permanent storage. Additionally, there is a background component running on the Region Server, which occasionally performs the data compaction so that small files can be merged into larger ones, overcoming a well-known HDFS problem (Shvachko et al., 2010). The performance of HBase was evaluated with the Yahoo Cloud Service Benchmark YCSB (Cooper, Silberstein, Tam, Ramakrishnan, & Sears, 2010) tool by corporation's researchers. The results presented by these authors indicate that HBase is extremely efficient in comparison to similar columnar storage solutions. This is particularly true for update-heavy workloads both for write and read operations, while performing slightly worse for the read operations in read-heavy benchmarks. The integration of HBase within the new LHC transient data recording solution has been studied in detail. One of the major shortcomings which prevents the usage of HBase as the primary solution for the entire data set are the resource requirements, since efficient caching can only be achieved on high-end servers with extremely high amounts of memory available, which is unrealistic for the data sizes to be ingested continuously by the services. In addition, it was demonstrated that the occasional file compaction executed on the infrastructure can introduce significant delays in the request execution (Ahmad & Kemme, 2015), which is a problem for the mission-critical service availability, in particular for analysis services requiring a deterministic response time such as the eXternal Post Operational Check (XPOC) of the LHC beam dumping system. Currently, HBase is considered for the short-term sliding-window storage solution integrated within the subset Hadoop infrastructure, dedicated to the most recent and most requested data so that it can be accessed with even lower latencies.

2.2 Summary

In this chapter, a detailed study of possible performance optimizations and upgrades of today's LHC transient data recording and analysis systems was conducted. The initial research was targeting optimizations which would not require the replacement of the currently deployed architectures. Despite the fact that multiple possible enhancements could be identified , the efforts required for implementing further optimizations of the existing CALS and PM were in no relation to the possibly obtainable

gains. It was therefore decided to study in more depth modern data storage and processing solutions, specifically created to operate on large datasets as it is the case of the current accelerator data repositories.

The complexity of such Big Data analysis solutions required an independent analysis of each of its major components to assess and further improve their behaviour for heterogeneous workloads as they arise in the daily operation of an accelerator such as the Large Hadron Collider. The analysis of the most popular architectures, like Apache Hadoop, allowed to identify three distinct inter-changeable modules, applied in different phases of the data life-cycle within the system. First of all, the solutions applied for data processing tasks were studied. The MapReduce paradigm still remains a valid choice for many use cases, however, more recent systems, like Flink or Spark are featuring large developer and user communities and implement features which significantly improve the performance of large dataset analysis. Secondly, solutions related to the cluster resource management and allocation were studied. The Hadoop YARN, a second generation resource manager, is the system of choice for many infrastructures, mainly due its reliability, security and efficiency. Nevertheless, several valid alternatives were identified, namely Mesos, which provides additional control of the scheduling process to the users and implements different resource allocation policies (CPU-based scheduling for example). Finally, solutions capable of efficiently managing large amounts of data were studied in detail. The analysed distributed storage systems were mostly solving the challenges of primarily static environments, while not being optimized for more dynamic use cases, like it is required for the second generation LHC data analysis framework. The described shortcoming motivated further research into the topic of the distributed storage solutions to find an appropriate response for the defined requirements.

Multiple solutions, capable of withstanding the data storage and analysis challenges for operation of the world's largest scientific instrument were meticulously studied. The most promising approaches were divided into several categories. Solutions of the first category consist of altering the most basic mechanisms, like data partitioning and replication, to improve the performance of the analysis frameworks. The data layout is generally determined based on the observed system workload. The second category consists of approaches which introduce changes at the file-level, incorporating indexing techniques and modifying therefore the respective file format. Instead of immediately persisting the data to the disk, these solutions collect the data until the input buffer is full and in the following transform the data structure in order to decrease the size of the repository and reduce the amount of data the applications need to process. Finally, dedicated large-scale real-time storage and processing frameworks were assessed, as one of the possible ways of optimizing the

storage layer. Despite their proven performance, these systems require significant additional cluster resources to operate efficiently, and thus cannot can only be applied to very large repositories with large overheads in terms of hardware costs. None of the existing storage solutions was found to provide satisfactory answers to the challenges at hand, thus in the following a novel approach is presented, being compatible with the observed heterogeneous and dynamic environments but yet flexible enough to allow further optimizations to take place throughout the lifetime of the deployed framework.

Chapter 3

Mixed Partitioning Scheme Replication

The integration of Petabyte-scale data storage and processing applications as part of the next generation LHC transient data recording solution will have a noticeable impact on its efficiency and therefore significantly reduce the infrastructure costs. However, further optimizations are possible with the deployment of highly specialized workload-aware systems, developed in response to emerging Hadoop system issues identified over the past years of its operation in production systems. The Parquet file format optimization has been widely studied and will be integrated into the final solution for improving the system's data throughput and significantly reducing required storage space. The processing pipeline will be enhanced with Spark, allowing the applications to take advantage of more efficient, in-memory processing techniques. The real-time random-access to the data will be assured by HBase, which is used as a transient storage for the most recently collected device measurements. A series of alternative solutions which operate on the data partitioning and replications layers were identified, but those were found not to harmonize well with the new data storage and processing framework mainly due to integration problems with the aforementioned system. In this chapter we present the extensive study we performed to define and optimize the new approach, which will take advantage of both partitioning and replication optimization techniques while still being flexible enough to ensure the compatibility with the remaining components of the infrastructure.

3.1 A novel architecture

The recent development efforts towards a second generation storage solution is motivated by several factors. On one hand, issues primarily related to the efficient ingestion and extraction of data, that operation crews and hardware experts of the accelerator complex experienced while working with currently deployed storage services. On the other hand, we addressed a large number of diverse requirements that cannot be satisfied by today's Post Mortem and CERN Accelerator Logging Service. The migration to a more modern data storage and processing solution must solve most of the known shortcomings and allow to execute very efficiently a broad range of existing and new analysis use cases. Several of the identified issues require a customized approach as they are not covered by the functionalities of the standard solutions designed for very large datasets. This section will focus on the characterization of this novel approach, which will allow to overcome the remaining shortcomings and further improve the performance of the next generation architecture.

A detailed analysis of the Post Mortem and CERN Accelerator Logging Service (PM and CALS) usage statistics suggests that the presently predominating workloads are **highly heterogeneous**. Both systems ingest and provide data back to users and applications on a continuous basis. Initially the Post Mortem system was designed as a purely event-based storage. The success of the system in handling the LHC use cases has however given rise to additional applications in machines of the injector complex which are based on shorter cycle times. Such shorter cycles, for some machines are in the range of a few seconds only, resulting in a quasi-continuous write load on the system. In parallel to the data ingestion processes, data extraction queries are continuously being executed, as both the Logging and the Post Mortem systems are used in the operational cycle of the accelerators to assess critical events, monitor and enhance the performance of the machines. Depending on the analysis use-case, a significant variation with respect to the amounts of data requested by different users can be observed. While relatively short-running queries are currently predominant in the observed workloads, many applications have decided to implement additional logic to extract – prior to the execution of the analysis - the data over long periods of time, as the implemented restrictions and read throughput does not allow a single query to work efficiently with large datasets. The time constraints for specific query categories have to be carefully considered, since for off-line, data intensive analytical workloads, the response time is a secondary factor. For analysis cases being part of the operational cycle of the accelerators however, additional delays impact the availability of the accelerator infrastructure for physics production. Additionally, multiple query profiles were identified to perform complex

join operations on different attributes of the collected signals, such as measurement time, accelerator state, device type and equipment location. The systems currently deployed CERN partition and index the entire dataset solely using time and device type attributes, and therefore are potentially hampering the performance of a significant part of the executed user requests based on different attributes. Finally, the workload analysis survey we performed revealed the existence of seasonality in the profiles of executed queries. The requests processed during the operational phase of the accelerator complex are significantly different from those which are executed during periods when accelerator complex is undergoing commissioning activities or during periods of extended maintenance.

The continuous effort of upgrading the LHC and its related accelerator systems (La Rocca & Riggi, 2014) in an effort to increase the reliability and performance of the equipment impose another important requirement for the next generation storage and processing solution, namely the need of **resilience against workload changes**. Along with the installation of completely new equipment in the tunnel during upgrades or consolidation programs, obsolete hardware is eventually entirely removed from the tunnel. The upgrade of components or entire systems often alter completely the amount, precision and even the format of the data and measurements made available for long-term storage and data processing. According to the report presented by CALS developers (*CERN Accelerator Logging Service Evolution Towards Hadoop Storage.*, n.d.), the LHC accelerator systems have significantly increased their data acquisition rates and the amount of collected metrics during the first extended LHC maintenance phase. During this period, many hardware systems were consolidated based on the experience of the first 3 years of LHC operation, increasing the data ingestion from 150 Gigabytes to 600 Gigabytes of uncompressed data written per day. The number of data extraction requests similarly increased during this period of time. Ignoring the fact that data sources in a highly dynamic and heterogeneous environment change over time, will quickly turn a once efficient solution into an inefficient, obsolete storage system.

Another important requirement for the next generation LHC transient data recording and analysis system is the **flexibility for the introduction of new data types**. The specialization and dispersion of storage solutions for transient accelerator data introduces a considerable overhead for processes used during accelerator operation. Amongst the multitude of isolated analysis modules, one can find many applications which retrieve data from multiple sources using different APIs and object schemes. Dedicated analytical tools available for the hardware experts also extract the data from individual storage systems, but provide very limited support for statistical analysis. In order to execute ad-hoc queries which require the data from multiple sources,

the operators are therefore forced to develop their own applications, which is both time consuming and requires an extensive domain knowledge of the storage systems and data extraction APIs. This additional overhead often results in a distraction from the main interest of the user, which is the definition and efficient execution of the analysis logic. For these reasons, a thorough understanding of the specificities of each of the storage solutions is considered vital in the effort of designing an architecture capable of unifying the data access interface and at the same time providing a powerful, yet comprehensible analytical tool set for the hardware experts and accelerator operation crews alike.

The aforementioned shortcomings are the core challenges for the novel approach presented in this thesis – the Mixed Partitioning Scheme Replication. The proposed solution introduces optimizations on the file system level without modifying the service endpoints, therefore being completely transparent to the user. The fundamental principle of the designed technique consists of creating multiple data partitioning schemes, optimized for a pre-determined set of workload categories, and performing the replication of individual representations. Unlike traditional replication solutions which maintain exact copies of the stored data, the MPSR structures the data managed by distinct replica groups differently. Implementing the workload-awareness requires an initial study of the system and the definition of data placement algorithms according to the identified query profiles. Replications can be managed elastically, therefore specific data sets which are frequently accessed by the applications can be distributed over additional cluster resources. This strategy aims to increase the number of local data executions, to improve job performance both due to faster disk access and reduced network overheads. Having implemented the aforementioned data placement algorithms and replication schemes brings an added benefit. It becomes possible to modify both when the need arises, being therefore possible to adapt to changes of data sources and/or user behaviour while maintaining the initial efficiency of the overall infrastructure without requiring additional resources.

Example 3.1.1. Consider the existence of a service for tracking the operations of different Internet of Things (IoT) agents, installed in public locations. There are multiple attributes which characterize the agents and the data they reported (with their corresponding cardinality indicated between parentheses): *time* (daily), *agent category* (1000), *agent state* (5), *operation type* (25), *location* (28) and *service operator* (4000). There are $1 * 10^7$ agents, each reporting 100 Megabytes of data per day. The measurements are distributed evenly inside the scheme when belonging to the same attribute (there are 10000 agents for each category for example). The distributed storage solution replicates the data three times for performance and - even more important - for failure tolerance reasons. There are several requests which

are being submitted to the system with the same frequency:

(R_1) determine the average number of category agents operating in given country

(R_2) calculate the number of operations executed daily by a determined agent category

(R_3) determine the device which had the most malfunctions during the day for a given service operator

The standard solution partitioning scheme will be <*time*, *agent category*, *location*>. Since the data distribution is uniform, the lowest level directory size in each case will be 35.71 Gigabytes. The request R_1 would require to process a single directory to provide an answer to the query, thus the input size will be 35.71 Gigabytes. The request R_2 will require to process multiple *location* directories for the agent category in question, thus the input size will be 1 Terabytes of data. Finally, the request R_3 will need to perform a *full scan* operation and analyse the whole daily data set, corresponding to a total amount of 953.67 Terabytes.

The integration of the MPSR solution will allow to store several data copies, optimized for different workload categories without introducing an overhead in terms of total storage volume of the system architecture. In this case, the partitioning criteria could for example be defined as i) <*time*, *agent category*, *location*>; ii) <*time*, *agent category*, *operation type*>; and iii) <*time*, *service operator*, *agent state*>. Consequently, the terminal directory size will be 35.71, 40 and 50 Gigabytes respectively. The request R_1 will still require the processing of 35.71 Gigabytes of data for its completion. The R_2 input size will be reduced to the processing of 40 Gigabytes, since there is a partitioning criteria matching the filters. Finally, the request R_3, for the same reason, will need to process only 50 Gigabytes of data.

■

The MPSR architecture can be split into two main yet independent components: data ingestion and data processing. The specificities of the data collection pipeline are presented in Figure 3.1. Depending on the architectural point of the data storage and processing system at which the data ingestion procedure is integrated, the whole process can be split into two interconnected, yet separately managed components. The first one is primarily responsible for retrieving data from the data sources, performing the pre-processing and preparing the data for writing it to the physical storage. The second component manages the communication process with the data storage and processing solution. The separation of the data ingestion pipeline is driven by the MPSR flexibility requirements. Integrating the data acquisition and

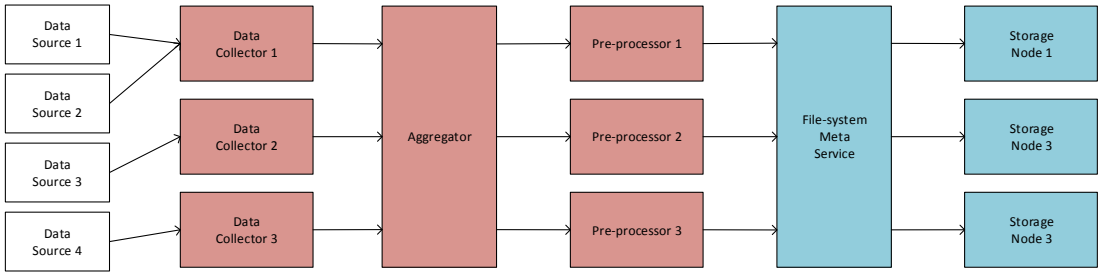


Figure 3.1: The data ingestion pipeline.

pre-processing mechanisms directly into the server application for data storage and processing solution will break its compatibility with dedicated data collection frameworks and therefore require additional efforts to ensure failure tolerance and scalability. Furthermore, the delegation of the data preparation and aggregation tasks to the server components which are deployed on the data persistence layer will result in permanent resource allocations for the data ingestion process, since CERNs data storage systems have to ingest data on a continuous basis even if the accelerators are not operational. The solution which is commonly adopted by applications facing similar issues (Sumbaly et al., 2013) (Toshniwal et al., 2014) is the integration of a dedicated data collection system, like Kafka. Since a similar development is foreseen for the next generation storage architecture at CERN, the design choice for the MPSR was to delegate the data pre-processing tasks to such an external tool. After the input is prepared for writing, the remote server is notified. Taking into account the user request, the configured partitioning criteria and the cluster resource usage, the master recurs to the MPSR module to determine the node which will permanently persist the collected data. Finally, the transport protocols ensure that the data is correctly uploaded from the external data ingestion application to the identified cluster node.

The data processing pipeline (see Figure 3.2), in relation to the underling solution implementation, remains mostly untouched with the exception of the component which determines the input files for the submitted job. Upon arrival of the data processing request, the associated meta-data is initially inspected. Most of the meta-data analysis operations are still handled by the original data storage and processing solution. While decoding and building an appropriate query representation, the MPSR module is invoked in order to determine the partitioning criteria which will be the most efficient in providing data for a particular user request. Based on the current cluster usage, the resources which maximize the rate of local execu-

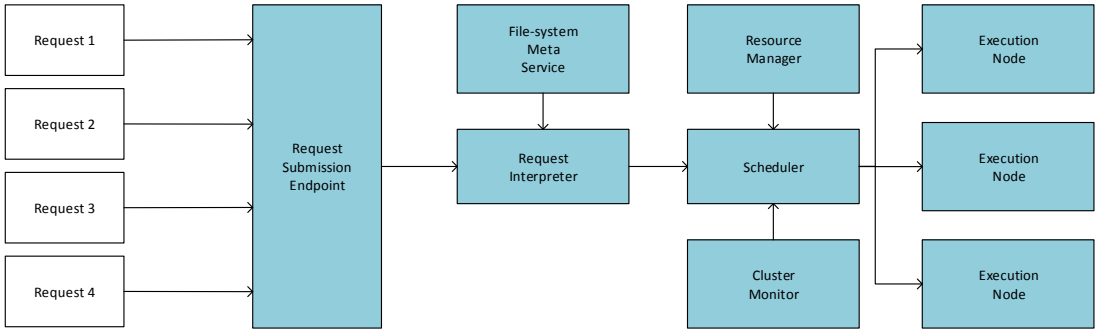


Figure 3.2: The data processing pipeline.

tions are allocated. Finally, the tasks are scheduled for execution on the previously selected machines. The entire process is transparent to the user, as the MPSR components which calculate the input splits and allocate the computing resources are implemented on the server-side. For this reason, no modifications are required on the application endpoints.

In the MPSR approach both the data collection and processing pipelines share a common module, which is responsible for maintaining the logic associated to the structure of the stored data. On the lower level, two different strategies for organizing the information across the available storage resources are presented and analysed in terms of their strengths and weaknesses.

3.1.1 Homogeneous MPSR

The homogeneous Mixed Partitioning Scheme Replication was the initially developed, simplistic approach which embodies the main characteristics of the proposed solution. The main principle is to dedicate entire nodes to a determined partitioning criteria, thus each of the cluster machines remain allocated to a single replica. Whenever the number of the available resources exceeds the configured replication factor, the segmentation process is triggered with the objective of rebalancing the structure of the replication group storage. The rebalancing operation relocates certain parts of the stored data to the newly connected node and ensures that duplicated information is purged from the source. Once the process is completed, both machines maintain the individual data parts partitioned by the same criteria. The resource allocation process is controlled through the scoring system, which can be configured to prioritize certain workload categories by assigning more machines to a given par-

titioning criteria. When scheduling the execution of submitted jobs the proposed technique prioritizes the allocation of resources optimized for matching query types. Nevertheless, non-optimized resources can be used for processing in case the system is saturated with workload requests of a single category.

Figure 3.3 presents a very simple example of the four-machine cluster integrated with the described homogeneous Mixed Partitioning Scheme Replication. In this particular case, the replication factor of the underlying storage solution is three. This value was chosen since it is the recommended configuration for HDFS clusters, and in addition it corresponds to the number of different partitioning schemes employed by the system. The outer shapes in the illustration identify the nodes of the cluster, while the inner shapes represent the data storage. Each of the colours of the inner shapes represent a specific partitioning criteria. Since the number of physical machines in the cluster is larger than the replication factor, one of the replica groups has its data split across multiple nodes. The data – previously structured according to pre-determined partitioning criteria - gets appended to its respective replica. Whenever the execution of the query associated with a particular workload category is scheduled, the highest priority is given to the entire execution of this query on the optimized resources.

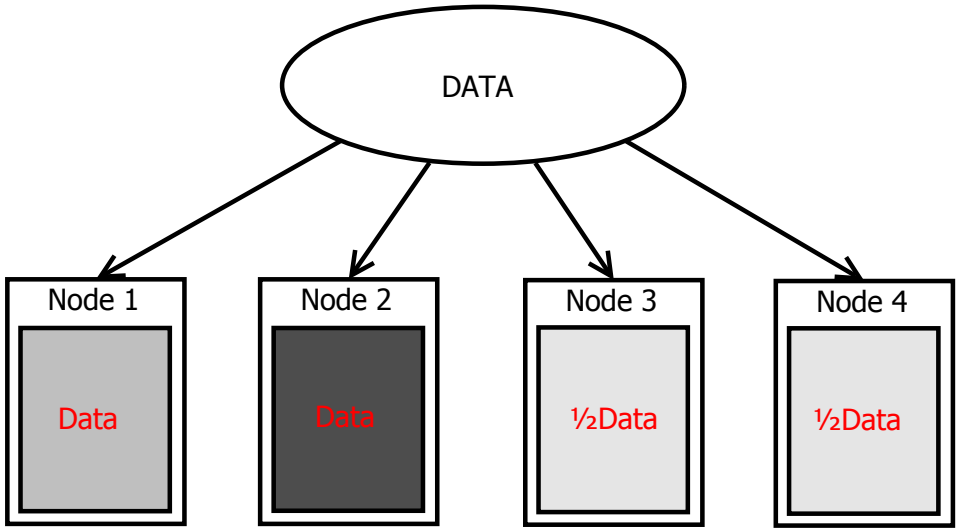


Figure 3.3: The homogeneous Mixed Partitioning Scheme Replication.

The major advantage of this solution is the relative simplicity of the architecture when compared to the second approach described in the next section. This simplicity allows for the implementation of a system fully compliant with the Mixed Partitioning

Scheme Replication principles, while requiring a very limited development effort. First, the data distribution across the cluster is predictable, since only two variables are taken into account: the number of available nodes and the algorithm which defines the segmentation points. Secondly, the meta-data service does not require neither complex resource balancing algorithms, nor memory intensive data structures to maintain the file system representation. This allows the homogeneous MPSR to inherit most of the underlying storage solution scalability properties. Furthermore, this data organization approach is very flexible as it does not depend neither on the specific data partitioning criteria nor on the partitioning technique, thus allowing each application to use the most effective partitioning solution to its specific needs. On the downside, homogeneous MPSR suffers from load balancing problems. In case of frequent changes in the profiles of the executed data queries, resources storing less frequently accessed data will be underused, while other machines will be under heavy load. Moreover, in case of long-term workload deviations like those that will happened during seasonal changes, the cluster will remain in an unbalanced state for extensive periods of time which may result in an increased probability of hardware failures appearing primarily on the nodes storing highly requested data.

3.1.2 Heterogeneous MPSR

The principles applied in the heterogeneous Mixed Partitioning Scheme Replication are similar to the previously presented approach, but adding an additional dimension for the replication process. The new technique does not reserve an entire node for an individual partitioning scheme, instead multiple data representations can co-exist on the same machine simultaneously. Taking into account the workload characteristics, the storage resources of each of the cluster nodes are organized into segments. As a further step, each segment is connected to the replication group pool, which defines the set of rules for the routing of incoming data to the appropriate location. The cluster resource usage and workload metrics are collected and analysed continuously. Based on these metrics, whenever the cluster is upgraded with additional nodes, the rebalancing process can determine which segments should be further divided to maximize the performance of the system. Maintaining the data in smaller subsets allows multiple workload categories to benefit from the addition of new nodes to the cluster. However, the process of data distribution becomes more complex, since the segmentation algorithm must ensure that the same node does not store all the available data copies. Otherwise, in case of a permanent node failure, there is no way to recover the data stored on that node unless an independent backup process is implemented. For these reasons, resource allocation and job scheduling techniques

have to take into consideration that different workload category requests can compete for the same node. Nevertheless, more fine-grained control of the data structure and organization on the disk results in a more efficient mitigation of periodic workload deviations.

One of the possible Heterogeneous MPSR usage scenarios is presented in Figure 3.4. In this example, the cluster is composed of four nodes with a pre-configured replication factor of three. Like in the previous example, the outer shapes identify the physical machines, while the smaller, inner shapes represent the data segments. The colouring of the inner shapes corresponds to different partitioning criteria. The input data is continuously analysed by the data placement module and the conceptual division is performed in order to prevent situations of permanent data losses due to an improper distribution (as described above). Based on the pre-defined set of rules, the data is then routed to the its storage segment. Additional resources are used to split the segments which are most frequently accessed by the users. Priority is hereby given to the execution of data-local jobs. The non-optimized resources are used for processing the queries only in the situations where the queue is saturated with single workload category requests.

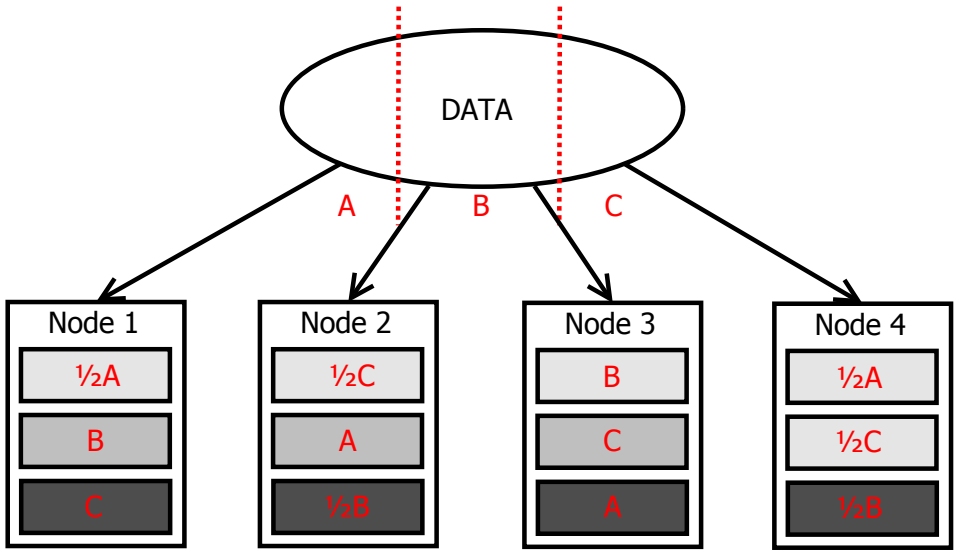


Figure 3.4: The heterogeneous Mixed Partitioning Scheme Replication.

When compared to the homogeneous approach, a major advantage of the heterogeneous MPSR is the additional control and flexibility of data placement across the available cluster nodes. Managing smaller data sets results in a more controlled load

distribution across the cluster nodes and allows the system to adapt better to any workload deviations. Furthermore, more evenly distributed data allows to predict with much higher precision the query execution time and queue behaviour during a normal operation, as well as when the system is experiencing occasional workload bursts. This approach enables a more efficient elastic replica management, since the segment size can be optimized for frequent, periodic data transfers. Finally, in case of occurring hardware failures it is still likely that parts of the query will be executed efficiently on the remaining segments, while the missing (optimised) ones are being reconstructed. On the other hand, a major challenge is introduced in the infrastructure maintenance process: depending on the complexity of the integrated algorithms, any manual intervention will become very tedious if not impossible at all. It should also be mentioned that the meta-data service will be constantly monitoring the incoming data to ensure a resilient data placement. This results in a higher memory consumption on the master node, which can affect the overall scalability and availability of the service. Consequently, the reliability of the heterogeneous MPSR leads to a much higher dependency on the algorithm choice for specific operations and will hence require more changes to the underlying storage solution.

3.2 MPSR Characteristics and Use Cases

The initial overview of the Mixed Partitioning Scheme Replication benefits and limitations of the proposed solution was performed according to the CAP theorem (Brewer, 2000). This theorem defines three properties: consistency, availability and partitioning tolerance, out of which any distributed system can only achieve – i.e. be optimised – for at most two. According to the definition, partitioning tolerance must be supported by any distributed application, since in case of arbitrary network message losses, the other two remaining characteristics will be impacted. The proposed approach is appropriate for systems which target CP properties (consistency and partitioning tolerance), as several of the functionalities constrain the availability property defined by the CAP theorem. First, the data ingestion process performs the pre-processing of the inputs, thus the data is not immediately available for incoming user queries. Furthermore, a specific data structure can only be stored on the cluster node with the related partitioning criteria. In case the machine is not available at a given moment in time, the data is retained in the data ingestion layer, until an appropriate resource is available. The prioritization of consistency was also greatly influenced by the fact that most of the studied data storage and processing solutions are designed for CP (Tormasov, Lysov, & Mazur, 2015), while availability can only be optimised as much as possible within the given constraints (Gilbert &

Lynch, 2012).

Further analysis was conducted in order to identify the characteristics of applications which could benefit most from the integration of the proposed solution. The full potential of the MPSR efficiency can be achieved by systems featuring more of the set of properties described below:

- The workload of the system can be highly heterogeneous, however the solution will be the most efficient when future requests can be at least partially predicted. Workload predictability is useful to define the data organization strategies which minimize the number of the concurrent requests competing for same resources. In addition, data access patterns can be exploited by elastic replica management algorithms to prepare beforehand for incoming workload bursts.

- Systems which aggregate structured data from multiple sources. The MPSR, in order to determine the data partitioning and replication strategy, requires parsable meta-data based on the input object attributes to be available. With this condition fulfilled, the proposed solution supports individual data source storage scheme configurations with targeted optimizations.

- Solutions which integrate the possibility for multiple optimizations at different architectural layers. The Mixed Partitioning Scheme Replication technique is flexible and does not introduce modifications to the underlying storage solution endpoints, thus allowing to inherit the original systems compatibility with external applications.

- The systems employing a "write once read many" approach on the data storage layer. Since the MPSR approach employs multiple schemes for storing the data, altering or eliminating a file belonging to one partitioning criteria can trigger an expensive operation on representations which belong to other replication groups. Data modification operations require additional logic, both to determine the information to be updated and for ensuring that in case of failure the file system is not left in an inconsistent state.

The beneficial properties listed above led us to a complementary list of characteristics which would urge against the implementation of the proposed architecture, as it would result in an inefficient storage solution:

- Systems with frequent workload changes. Constantly shifting query profile deviations will result in the invalidation of the initial storage optimization,

including partitioning and replication strategies. The problem can be mitigated with cluster re-balancing, but it is a resource-intensive and time consuming operation, thus it should not be executed too frequently but instead should be carefully planned.

- Solutions operating on data with limited access to the object attributes (media or raw text files for example). Although some meta-data and structures can be extracted from such assets, the specificities of the analysis performed on media content for example, makes it virtually impossible for MPSR to identify the appropriate storage strategy.

- Systems which require the collected data to be available immediately. The proposed approach requires the data to be pre-processed and aggregated, since the storage solutions which MPSR is appropriate for are mostly optimized for batch processing. Large amounts of small files in such systems is recognized to be a serious constraint.

- Applications which require fast random access to the stored data. The category of data storage and processing solutions which MPSR targets are tuned for large file processing. Unless external optimizations are applied, indexing mechanisms are not supported, thus each query requires a full scan to be performed to extract the requested data. Worse, the distribution over the network introduces additional delays due to frequent data transfer operations, resulting in a slowdown of the data access.

3.3 Experimental Study

The definition and applications of the Mixed Partitioning Scheme Replication are - up to this point - merely based on assumptions of performance benefits introduced with the described data storage strategies. To further develop the proposed approach it is therefore necessary to study its characteristics in depth and – as the following step – design and present the result of benchmarking experiments which allow to quantify the expected performance gains. The primary step of this study is to determine through simulations whether MPSR can effectively be more efficient in comparison to traditional storage organization techniques, and to identify the factors ceiling the maximum achievable performance improvements. Furthermore, the designed experiments should help us to identify the variables having the strongest influence on the behaviour of the MPSR, and which therefore must be considered as a priority focus when integrating the MPSR with an existing infrastructure.

For the initial modelling approach, a formal mathematic model has been developed (Boychenko et al., 2016) (Boychenko et al., 2015), which - based on the provided variable values - allows to calculate the average query execution time for the MPSR solution. The maintenance and further extension of the designed model has shown to be a very time-consuming task, leading to the decision of implementing a dedicated simulator. The developed simulation engine - in addition to the previously collected metrics - allowed us to study the job queue status as well as to introduce additional variables into the model. Additionally, complex scheduling techniques could be integrated into the system, which were then used to study the impact of different resource allocation prioritization strategies on the MPSR performance. Finally, the simulation engine supported the integration of different (statistic) variable distribution models, thus allowing the experiments to be approximated with very high accuracy to the real-world systems.

3.3.1 Model Definition

The main goal when designing and developing the simulator was the maximization of its similarity with the real-world scenario. For this reason, the number of assumptions was decreased to the strict minimum, while maintaining the number of the free variables sufficiently high in order to study all relevant aspect of the solution. Another important consideration was the simulation engine modularity, since we were aiming to integrate and study different request scheduling techniques (ideally without modifying the source code of the engine core). The resulting architecture is presented in Figure 3.5.

The simulation application designed can be split into two main components: the scheduler and the simulation engine module. As a first step, the simulation engine performs the configuration parsing and argument validation. Based on the user inputs, the request pool is generated and the infrastructure is prepared. The execution manager is responsible for the simulation flow and for maintaining the simulation logic. The requests are scheduled in advance and are injected into the simulation process when their planned deadline is reached. In addition, the scheduler module grabs incoming queries and places them into the queue, taking into account its particular resource allocation strategy. Whenever execution slots are available, the request in the head of the queue is pushed for execution, during which the execution manager locks the respective resources for performing the processing. After the job execution is completed, the workload statistics (metrics) are updated accordingly.

Several abstractions were made in the simulation engine regarding time, job input size and resource processing capabilities. Since the results will be presented in

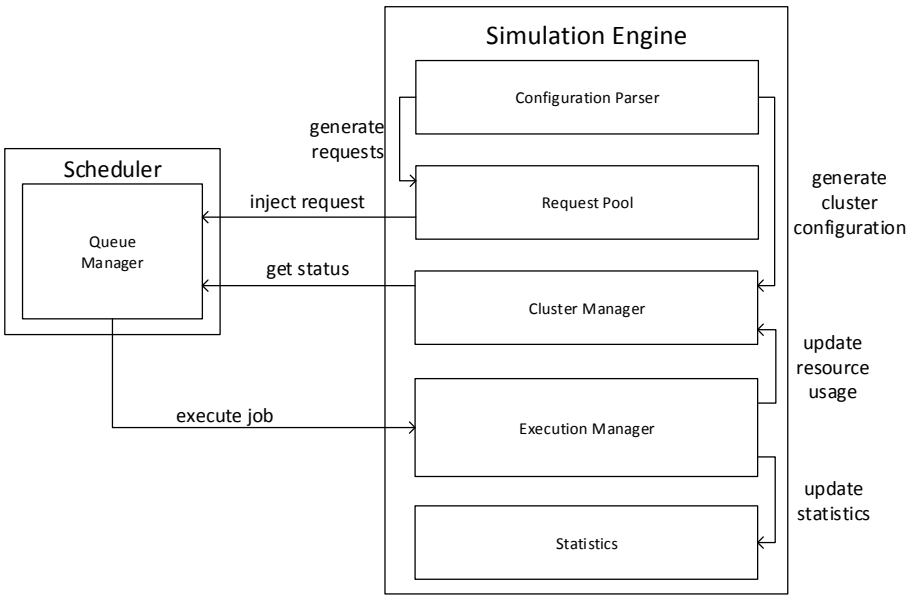


Figure 3.5: The simulation engine architecture.

abstract units, it is important to state and apprehend the magnitude of the scale and the relation between them, which can be described by the following sentence:

> **One cycle** is the time required for **one executor** to process **one abstract size unit (*asu*)** on a generically optimized node.

Parameters

The simulation of the desired behaviour required the introduction of multiple parameters, which can be split into two categories, *request-related* and *infrastructure configuration*. The arguments belonging to the request-related category are mainly used to configure the workload characteristics. The list below summarizes the parameters which belong to this category:

- The request arrival rate: controls the pressure that incoming requests create on the query queue.
- The request size: defines the amount of the resources required for the processing of the query input.

- The request variation factor: controls the probability of a determined workload category to be assigned to an incoming request.

The second category of the simulator parameters define the configuration of the infrastructure:

- The processing speed coefficients: control the request execution time, both on the optimized or non-optimized cluster resources.
- The number of the machines: determines the amount of the cluster resources assigned to each replication group.
- The number of the execution slots: controls the quantity of the size units which can be processed by a single node per cycle.

Schedulers

The implementation started with the *conventional* scheduler, which provides the baseline results for the later comparison with the MPSR solution. The implementation reassembles closely the behaviour of Hadoop's Capacity Scheduler with a single queue configuration. The simulations with the conventional scheduler do not take advantage from the MPSR processing speed coefficients and assume that the data is replicated across the cluster using the same partitioning scheme, equally optimized for all of the queries executed on the system (like it generally happens in Hadoop infrastructures). Arriving requests are placed into the queue and executed whenever computing resources become available. The number of jobs executed concurrently is limited.

At first, for MPSR simulations, a very simple scheduler (referred to as *S1*) has been developed. The main principle of this implementation is to ensure that any incoming request will be executed strictly on the optimized resources. This scheduler ignores the fact that non-optimized machines for a determined job category might be underused.

To increase the balancing of the cluster resource usage, a different policy has been implemented (*S2*). In this scheduling scenario, the arrival time and query category are taken into consideration, though the main objective is to finish the execution of the job which arrived first as soon as possible. In case the queue is getting saturated by requests of the same type, the outsourcing to non-optimized resources is performed. Once scheduled, the application is executed entirely on the same allocated partitioning scheme.

Considering that none of the previous schedulers fully follows the FIFO approach, a third scheduling policy, prioritizing solely the arrival time of the job, was developed (*S3*). To process a given request as soon as possible, it is split across the whole cluster (both on optimized and non-optimized nodes) in a such way that all of the sub-parts finish the execution approximately at the same time.

Finally, a scheduling policy with dynamic queue management was developed (*S4*). This implementation features a two-level queuing system where - besides the usual partitioning type-based queue - a small portion of requests is placed into the structure where the requests can be re-ordered. The re-ordering process is triggered when computing resources are non-optimized for executing the first request in the short-term queue, but are optimized for subsequent ones.

Performance Metrics

The simulator's execution manager will collect – upon the cycle and request completion – the metrics which are used for further performance analysis. The collected statistics allow for a detailed analysis of the characteristics of the schedulers and to conduct meaningful result comparisons.

- The average queue size is computed by dividing the accumulated number of requests present in the queue at the end of each cycle by the total number of cycles.
- The average query execution time is calculated based on the interval between the job execution start time and its completion (in case multiple replicas are executing the request the time is aggregated).
- The average query waiting time is calculated based on the interval between the job submission into the system and the start of its processing.

Assumptions and Limitations

Due to the complexity of the schedulers and the amount of already existing parameters some assumptions were introduced. The main purpose of simplifying the model is to reduce the possible sources of uncertainties in the final results.

- Network latency, job staging time and concurrency factors are neglected, since those factors will be equally present in any solution as long as the schedulers are integrated into the same infrastructure.

- The system does not integrate any caching mechanisms, since in this case a whole new set of variables in different stages of the data processing pipeline must be added into equation.

- The query execution and waiting times can be estimated.

Besides these assumptions, the following limitations of the simulation engine were identified:

- Concurrent job execution within the same node is not supported, meaning that all the available executors are generally reserved for the same job.

- Requests cannot be interrupted when already in execution, making it impossible to stop the execution of a determined job to schedule another one.

- The scheduler *S4*, can only perform the simulations with a maximum replication factor of three (which is the recommended value for high availability in modern distributed file systems).

3.3.2 Discussion of Results

One of the main objectives of this study was to provide a baseline to compare the performance of the MPSR schedulers with traditional solutions (represented by S1, the *conventional* scheduler). In case the proposed approach outperforms the classic Hadoop applications, the parameters which have the greatest impact on the performance can be identified and systematized to quantify the possible gains. Initially, a detailed study of the individual variables' impact on the simulated environment was conducted. In addition, a sensitivity analysis for the most relevant variables was performed as input to the next simulation phase, a complex multi-variable study.

Multiple variable configurations were probed to define and setup an environment where the conventional scheduler would be under moderate stress (the baseline variable configuration is presented in the Table 3.1). Then, during the initial experiments, one single simulation parameter was altered at a time. Each of the variables was sampled according to a pre-defined variable probability distribution. The simulation results were collected and automatically processed by R scripts triggered after the completion of each model simulation experiment.

Request Arrival Lower Limit	5 cycles
Request Arrival Upper Limit	10 cycles
Request Size Mean	225 asu
Request Size Standard Deviation	50 asu
Request Type Variation	0.333
Speed Up Factor	0.5
Slow Down Factor	1.5

Table 3.1: Base simulator variable configuration.

Request Arrival Rate

The first experiment focused on the request arrival rate. The possible value range is bounded by the lower and upper request arrival limits. When analysing the workload of CERNs accelerator storage, it was observed that during LHC operation the requests were arriving according to a uniform distribution, thus the samples were generated accordingly.

After an initial review of the collected measurements, the results were split into three categories based on the queue size. The results which are aggregated into the

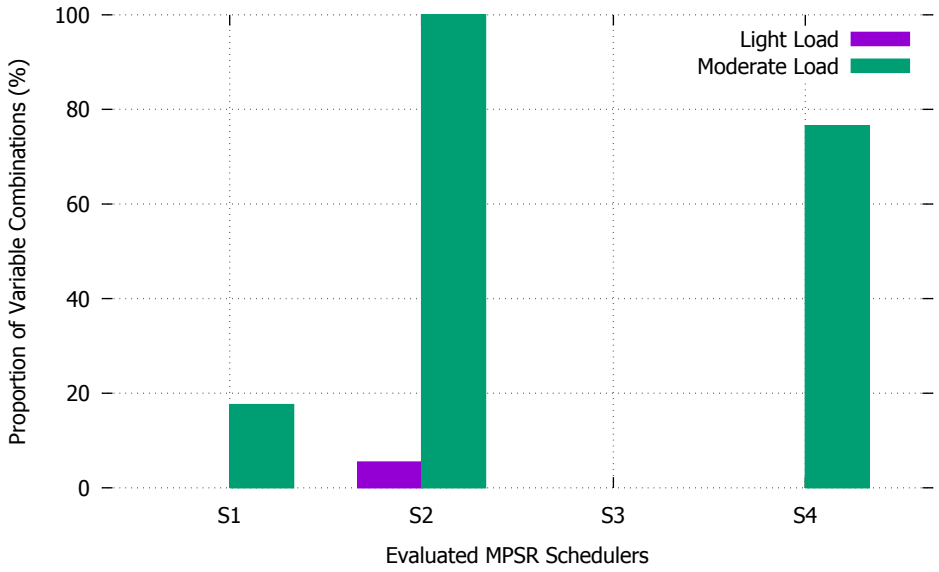


Figure 3.6: Arrival rate on average queue size impact analysis: the proportion of the variable combinations where MPSR approach outperforms conventional solution.

	Light Load	Moderate Load
Scheduler S1	2.04	2.09
Scheduler S2	1.95	1.51
Scheduler S3	0.98	0.96
Scheduler S4	1.64	1.43

Table 3.2: Arrival rate impact analysis: average query execution time improvement coefficient in relation to the conventional solution.

first category, designated as a *light load*, could be characterized by having a very low average queue size, where at most one request is waiting for its execution. *Moderate load* combined the simulation results where the queue size ranged from one request to the amount representing the sojourn time which exceed the execution time by at most a factor of 5. Finally, *heavy load* gathered observations where the queue size grew beyond acceptable performance limits, thus the configuration was considered as not promising for more detailed analysis. The relation of load category proportions in respect to the total amount of simulated configurations (9000 different variable combinations were tested) was of 35-45% for *light loads* and 5-22% for *moderate loads*.

The impact of the arrival rate on the average queue size metric was initiated with the identification of the workload configuration proportions where the MPSR schedulers were performing better than the conventional one (see Figure 3.6). Whenever the system is under *light load*, the classical approach manages the queue better than the proposed solution in most cases, and neither of the MPSR schedulers can be considered efficient in this case. The situation is different however when the system is under *moderate load*. In this case *S2* and *S4* consistently outperform the conventional solution for most of the use cases, while the efficiency of *S1* and *S3* is questionable.

The average query execution time metric was also addressed. The results revealed that all the schedulers except *S3*, were able to outperform the classical approach, suggesting that there is an interest to compare the MPSR schedulers amongst themselves. The improvement coefficients in relation to the non-discriminative solution are aggregated in Table 3.2. According to the summarized results, *S1* was presenting the lowest job execution times in comparison to the rest of the analysed techniques. *S3* was performing worse than the rest of the schedulers, including the conventional one. Finally, the average query execution times of *S2* and *S4* were similar, only providing moderate performance gains when compared to other approaches.

As expected, the analysis of the average query waiting time (see Figure 3.7) has

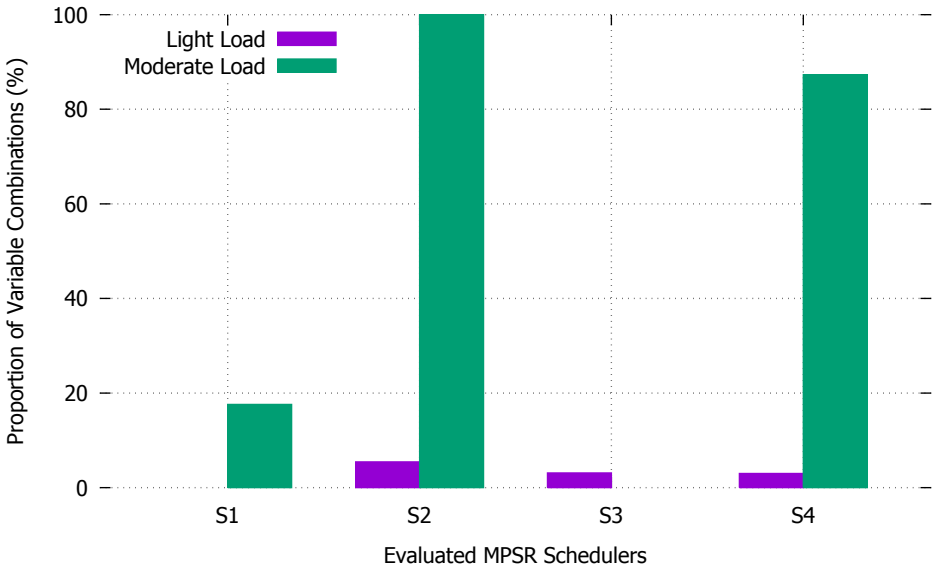


Figure 3.7: Arrival rate on average query waiting time impact analysis: the proportion of the variable combinations where MPSR approach outperforms conventional solution.

shown very consistent results with the results obtained during the study of the arrival rate impact on the queue size. The main reasons for this are the constant cluster processing capacity and the unchanged request size. Once again, the proportion of the workload configurations where the MPSR schedulers were outperforming the classical solution was insignificant for the *light loads*. According to the *moderate load* results, the improvements introduced by *S2* and *S4* could be observed on almost the whole range of workload configurations.

Despite the fact that scheduler *S1* is able to ensure a much faster query execution in comparison to other MPSR schedulers, its resource management policy results in a very poor queue management strategy and unbalanced load distribution. Scheduler *S3* did not present any considerable improvements over the conventional scheduler, in most cases performing much worse than the classical approach. The combined performance metric analysis of *S2* and *S4* on the other hand, has shown that both schedulers are capable of outperforming the classical approach.

Request Size

As a next step, the impact of the request size on the MPSR schedulers performance was studied. In line with the previous analysis, the results were aggregated into the three different groups, based on the number of requests in the queue. In total for every scheduler, around 60000 workload configurations were simulated and the load category proportions were 6-17% for the *light* and 6-29% for *moderate loads*, respectively.

Then, the impact of the request size on the average queue size metric was studied. The proportions of workload configurations where the MPSR schedulers are outperforming the traditional solutions are presented in Figure 3.8. Unlike in the previous analysis, the conventional scheduler is more efficient in managing the queue for *light loads* with the exception of the *S2*, which under certain circumstances was performing better. The observations regarding *moderate loads* confirmed once again the efficiency of the queue management strategy implemented in the schedulers *S2* and *S4*. Results for the scheduler *S3* however continue suggesting that a strict FIFO approach is not compatible with the proposed partitioning and replication technique.

The impact of the request size on the average query execution time was also addressed. The observations allowed us to conclude that with the exception of the

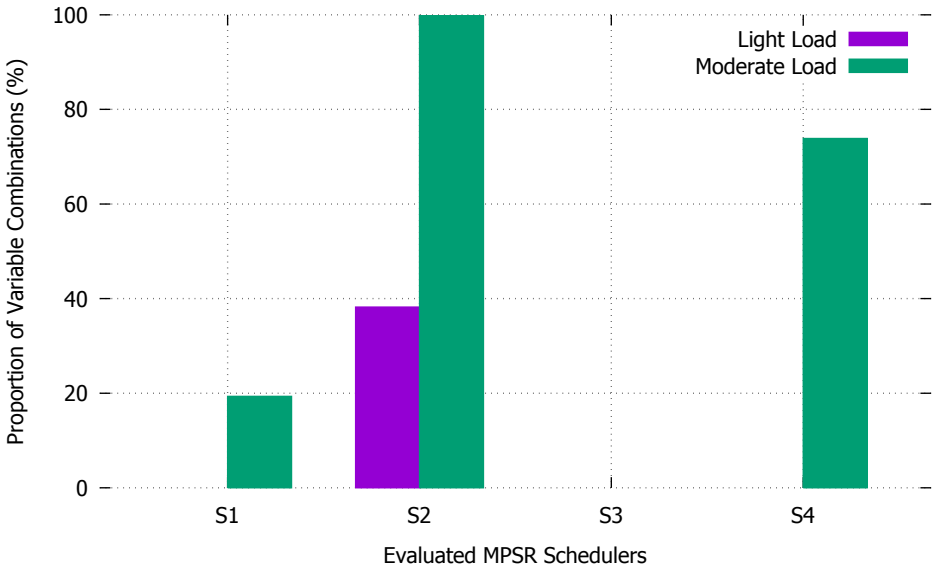


Figure 3.8: Request size on average queue size impact analysis: the proportion of the variable combinations where MPSR approach outperforms conventional solution.

	Light Load	Moderate Load
Scheduler S1	2.07	2.06
Scheduler S2	1.82	1.49
Scheduler S3	0.97	0.96
Scheduler S4	1.61	1.38

Table 3.3: Request size impact analysis: average query execution time improvement coefficient in relation to the conventional solution.

scheduler *S3*, the integration of the MPSR schedulers into the data storage and processing solution is beneficial, allowing to significantly reduce the data processing time. The following Table 3.3 presents the normalized, average execution time comparison between the MPSR schedulers. Once again, the scheduler *S1* allows the system to process the data much faster than other techniques, while *S2* and *S4* still proportionate moderate performance gains. Solely scheduler *S3* performs slightly worse than the conventional approach.

A more in-depth analysis of the average query waiting time (see Figure 3.9 sug-

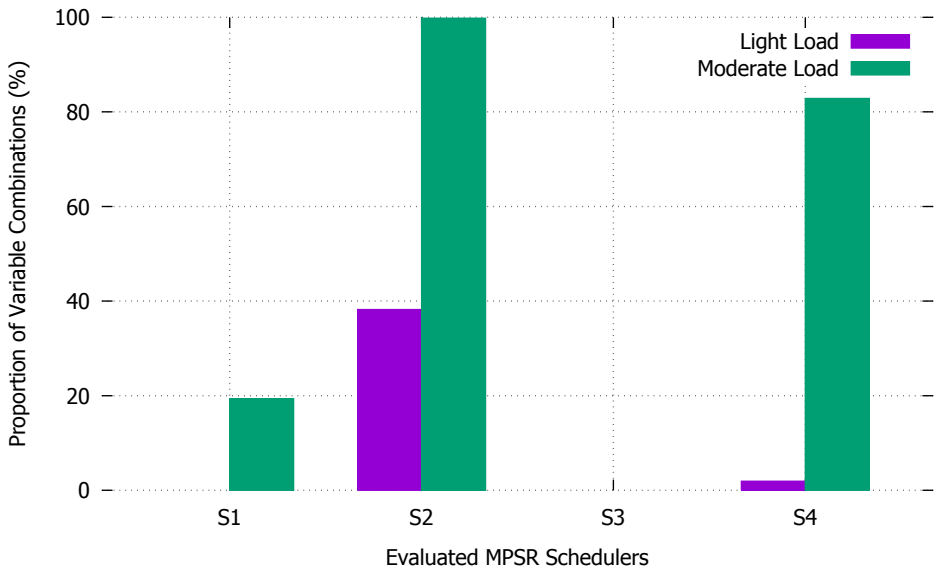


Figure 3.9: Request size on average query waiting time impact analysis: the proportion of the variable combinations where MPSR approach outperforms conventional solution.

gested that this behaviour is once again very similar to the results for the average queue size metric. Despite the fact that the request size parameter is variable, the processing capacity of the cluster remains roughly the same and the arrival time is constant. For this reason, the measured values are proportional to the number of the jobs in the queue. As it can be derived from the observed results, *S1* and *S3* do show once again an inefficiency of the underlying scheduling policies in managing the jobs' queue. On the other hand, schedulers *S2* and *S4* have clearly outperform traditional solutions for systems experiencing *moderate loads*.

Request Type Variation

The request type variation impact analysis performed, would a-priori exclude any influence on the performance of the conventional use case, since in the classical approach the resources are equally optimized for all queries. Hereby the objective was not to simulate the injection of unknown query types into the system, but rather to vary the proportions of already existing request categories. For the MPSR schedulers, this variable was therefore modified in the interval from $\frac{1}{3}$ to 1. The initial value was hereby tightly related to the number of different request classes which the system was optimized for, since the variable would be further representing the likelihood of a determined request class to be assigned for the arriving query.

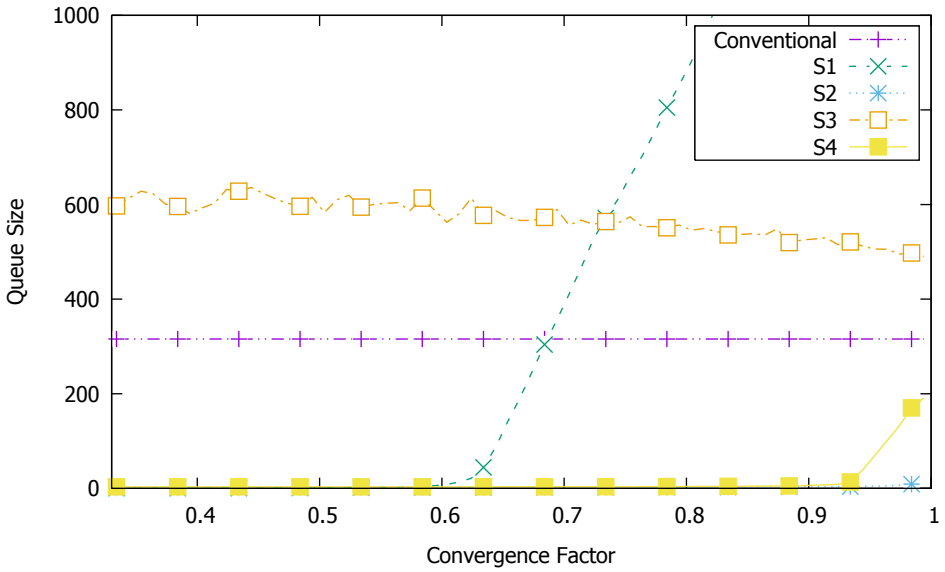


Figure 3.10: Request variation impact analysis: the average queue size.

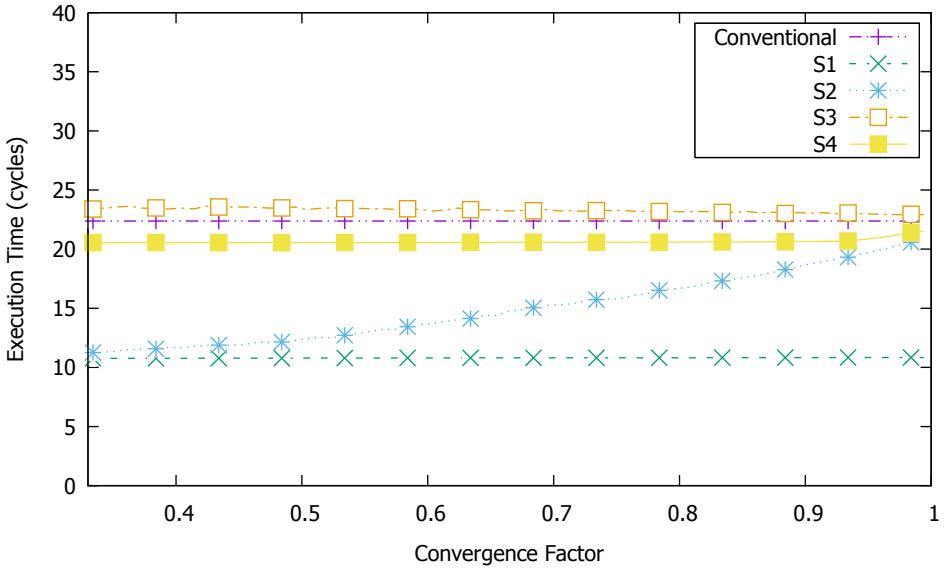


Figure 3.11: Request variation impact analysis: the average query execution time.

Value sampling was done using a linear distribution which, based on the provided parameter, increased the probability of a determined event to occur, while linearly decreasing the probability of others.

As a first step, the impact of the request variation on the average queue size metric was studied (as depicted in Figure 3.10). When analysing the simulation results, one can conclude that scheduler *S3* was less efficient than the conventional solution for all of the simulated configurations. *S1* on the other hand, coped efficiently with queues of relatively balanced workloads. One can observe however a turnover point, namely when requests of the same type start to dominate the incoming queries, in which case the queue size increased rapidly. The most workload variation tolerant schedulers were *S2* and *S4*, with *S2* slightly outperforming *S4*.

When analysing results from average query execution time metric (see Figure 3.11) one can observe that the results of scheduler *S1* results were constant. This is expected, since the request type does not have any impact on the job execution process. Scheduler *S3* was less efficient than the conventional solution in the whole parameter range, whereby its performance is increasing towards higher request variation rates. The causes of the observed behaviour were studied with the conclusion that this is caused by a tiny delay introduced by the way the resources are assigned when finalizing the previous requests' execution. The average query execution time of scheduler *S4* was found to be nearly constant, which can be explained by the fact that this algo-

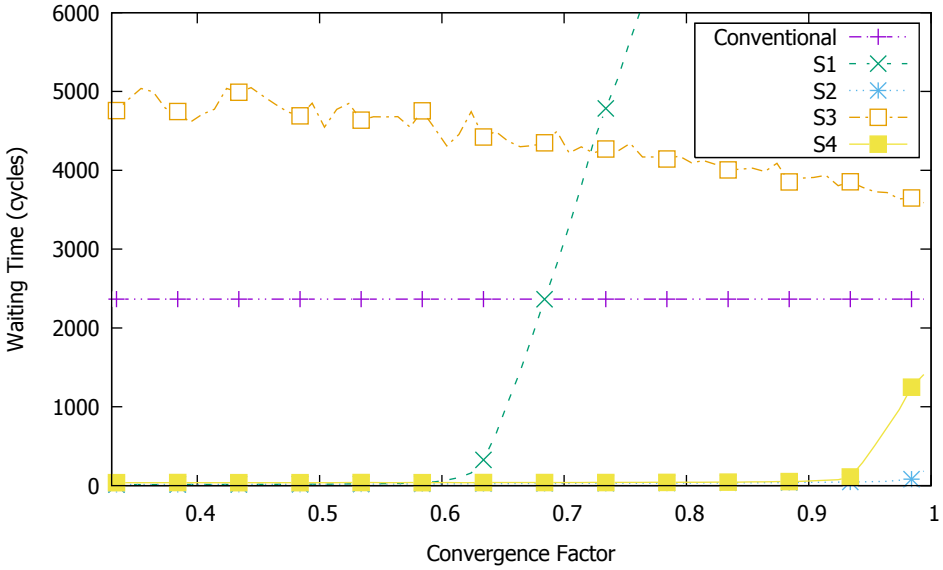


Figure 3.12: Request variation impact analysis: the average query waiting time.

rithm starts adapting to the new workload, by splitting the requests when there is no benefit in executing the queries entirely on other, non-optimized resources. Finally, the execution time of scheduler *S2* was growing as expected, since the convergence of the requests towards the same request type resulted in an increasing number of requests to be executed on non-optimized resources.

The results of the convergence factor impact on the average query waiting uncovered very similar trends, as has been previously observed during the queue size metric analysis (see Figure 3.12). These observations point out the fact that there is a strong relation between the two metrics, which is likely to be due to the way the input arguments are interpreted by the simulation engine (which according to the Little's Law (Simchi-Levi & Trick, 2011) is an expected behaviour).

Processing Speed Coefficients

As the final part of the individual variable studies, an analysis of the impact of the processing speed coefficients was conducted. Two independent parameters define the workload characteristics: the speed up and the slow down factors. The assigned values are represented by a multiplier applied to the time it takes for the conventional solution to process the query. The representation of the collected results defines the boundary zone, immediately before the point where the classical scheduler starts out-

performing the MPSR schedulers. Similarly to the previous study, the conventional solution is not affected by the processing speed coefficients as it is equally optimized for each of the workload categories.

First, the impact that processing speed coefficients have on the system's average queue size was studied (see Figure 3.13). Scheduler *S1* was not dependent on the slow down factor, as expected, and outperformed the conventional solution consistently until reaching a speed up factor of 0.52. *S3* in comparison to other schedulers managed the queue in the most inefficient way as can be observed from the results, even in cases when the system was able to execute requests significantly faster on the optimized resources. The most efficient solution was scheduler *S2*, being able to cover a broad range of speed up factors, and efficiently deal with environments where non-optimized machines were executing queries more slowly. Scheduler *S4* was presenting worse results in comparison with scheduler *S2*, almost for the entire whole range of the analysed configurations.

The study of the average query execution time (see Figure 3.14) confirmed once again the superior performance of scheduler *S1* over traditional implementations. The decaying efficiency of scheduler *S3* was very similar to the measurements observed during the average queue size analysis, once again placing this scheduler as the

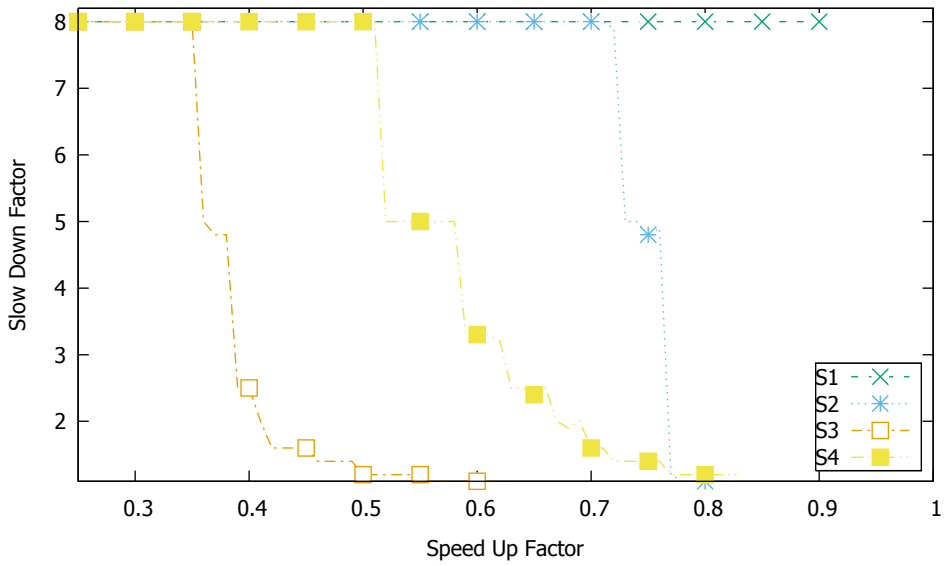


Figure 3.13: Processing speed coefficients on average queue size impact analysis: the edge of the variable combination where MPSR still outperforms the conventional solution.

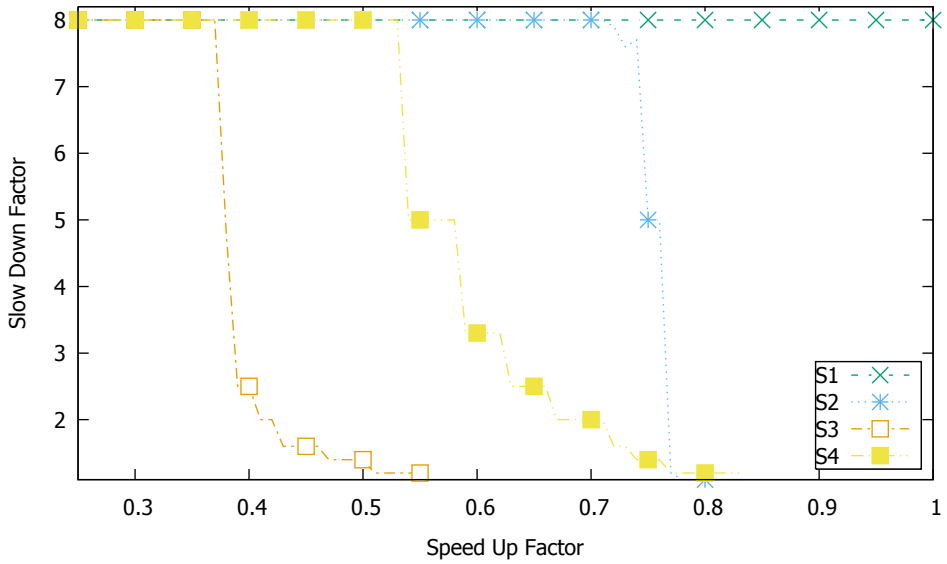


Figure 3.14: Processing speed coefficients on average query execution time impact analysis: the edge of the variable combination where MPSR still outperforms the conventional solution.

worst option among the MPSR solutions. On the other hand, scheduler *S2* presented the most consistent and satisfactory results, covering the widest range of simulated workload configurations.

Finally, the average query waiting time metric evolution was found identical to the one presented for the average queue size impact study, thus a detailed analysis was omitted for this case.

Variable Dependencies

Finally, after studying the impact of all individual variables on the efficiency and the behaviour of different schedulers, a detailed study of variable dependencies could be conducted. The main goal of this multi-variable simulation was to determine the variable combinations which have the largest impact on the performance of the proposed schedulers. For this purpose, variations of all variables were introduced simultaneously, limiting the changes however to a smaller range and at the same time using a larger step size to reduce the simulation time. The interval of the variable values was defined, based on the previously conducted studies, and had to satisfy the following two conditions: i) the region should include the turnover point

	Avg. Queue Size (X)	Avg. Exec. Time(Y)	Avg. Wait Time(Z)
S1	$\rho(X, RV) = 0.79$	$\rho(Y, PR) = 0.95$	$\rho(Z, RV) = 0.77$
S2	$\rho(X, AR) = -0.69$	$\rho(Y, PR) = 0.79$	$\rho(Z, AR) = -0.6$
S3	$\rho(X, AR) = -0.55$	$\rho(Y, PR) = 0.67$	$\rho(Z, AR) = -0.57$
S4	$\rho(X, AR) = -0.7$	$\rho(Y, PR) = 0.81$	$\rho(Z, AR) = -0.6$

RV - request variation AR - request arrival rate PR - processing rate

Table 3.4: Variable Relation Study: Strongest correlation with corresponding coefficients.

where a very small difference between the proposed schedulers and the conventional solution can be observed; ii) the results should be within reasonable performance limits (i.e. excluding the variable configurations which generate *heavy load*).

From the results analysis, the variables having the strongest influence on the average queue size, were *average query execution time* and *waiting time*. Since it would be difficult to represent and study graphical simulation results, statistical methods have been used instead. First, the correlation between the defined variables and metrics was calculated. Although in the previous study only linear associations between the variables were assumed, the possibility validating the observation was taken into account in the current analysis. One of the limitations of the correlation calculation is that it is only able to determine the representative coefficient for linear relations. Thus, to identify possible non-linear relations, the Maximal Information Coefficient (MIC) (Reshef et al., 2011) was used. When comparing the coefficients calculated with both methods, no case which could suggest non-linear variable relations was detected, hence confirming the validity of the only-linear correlation assumption.

To summarize, the largest absolute coefficients[1] for the proposed MPSR schedulers were identified. The variables with the largest correlation coefficients are the ones which influence most the performance of the proposed solution, and therefore have to be ranked higher in user concerns when applying the MPSR (as shown in Table 3.4). From the collected data, we can conclude that the *speed up* factor was the most influential variable on the average query execution time for all of the implemented schedulers. On the other hand, the *request variation* had the largest impact on scheduler *S1* for the average queue size metric. The performance of the remaining schedulers for the same metric relied heavily on the upper limit of the *request arrival* variable.

[1]MIC only produces positive values, while correlation can have negative ones

3.4 Summary

In this chapter the Mixed Partitioning Scheme Replication, a novel approach for performing optimizations on modern data storage and processing solutions, was defined and studied in detail. The identification of the strengths and weaknesses of the MPSR suggested that the proposed technique can be efficiently adopted by a wide range of the applications which require analysis of Petabyte-scale data sets. Unlike similar solutions, the novel approach does not impact the compatibility of the underlying storage and processing solution to achieve the desired performance gains, hence additional optimizations can be implemented on top of the deployed MPSR infrastructure. Despite the fact that relatively simple data placement and meta-data management approaches were described in this chapter, MPSR is flexible enough to accommodate highly specialized partitioning and replication techniques, enabling for further use case specific optimizations to be implemented.

Additionally, a study of the efficiency of the proposed solution by means of a simulation engine has been conducted, whereby a model which resembles as close as possible the real-world scenario has been developed. The simulations provided valuable information and insights about possible performance gains which can be achieved when integrating the MPSR into the storage and processing solution. The observed results revealed the importance of choosing and implementing an appropriate scheduling strategy to maximize the possible performance gains. The variable combinations having the biggest impact on the efficiency of the proposed solution were found to be the processing speed coefficients and the request arrival rates. Even though the model has been reviewed by the experts in the field (Boychenko, Zerlauth, Garnier, & Zenha-Rela, 2018) it still requires further validation through experimental studies, which will be described in the following sections.

Chapter 4

Mixed Partitioning Scheme Replication Implementation

The improvements observed from the MPSR simulation results allowed the research to proceed to the implementation stages of this work (Boychenko, Marc-Antoine, Jean-Christophe, Markus, & Zenha, 2017) (Boychenko et al., 2018). In this chapter, the architecture of the proposed approach and its integration with Hadoop system is described in detail. Taking into account the specificities of the underlying storage and processing framework, several options were considered while developing our MPSR prototype. In the first option, widely adopted in the research communities (most of the tools referred in the state-of-the-art of this thesis), the solution would involve the actual integration of the changes into the source code of the relevant Hadoop system components. The alternative, much less popular, would be the integration of the new file system module through the plug-in mechanisms supported by the Apache MapReduce framework implementation. Both options were considered and analysed in detail, after which we concluded that to achieve a functional prototype the direct integration into Hadoop source code would require much less effort. Nevertheless, the final version will be implemented as a Namenode plug-in, since maintaining the MPSR solution logic in a separate module is more practical in the long term (for example, when the Hadoop infrastructure is migrated into the new version only minor changes will be required in case the new release breaks some of the MPSR module's functionalities). Our implementation of the prototype focused on the most fundamental features of the proposed solution, including the mechanisms which allow the individual cluster machines to store the data assigned to specific partitioning criteria and route correctly the user requests. The dataset for the performance evaluation studies was determined based on the currently deployed storage solutions

workload analysis, and extracted from the respective repositories, i.e. it is as close as it can be to the actual dataset when in production mode.

4.1 Apache Hadoop

The selection of the foundation for the design and development of the Mixed Partitioning Scheme Replication architecture was made through the identification and comparison of different data storage and processing solutions considered suitable for very large dataset analysis. Amongst the different possibilities, the Apache Hadoop was selected for its flexibility, reliability and integrity. First, the Hadoop architecture provides the flexibility to integrate the new modules independently, without modification of the original sources, which is a significant advantage in terms of the service maintainability and compatibility with the future releases. Furthermore, the popularity of the Apache MapReduce implementation generated a large community of users around the project, leading to a constant improvement of the features and the quality of the solution. Finally, the integrity of the Hadoop system (as the distribution already provides the required set of components to implement and manage the entire data storage and processing pipelines) allows focusing the efforts on the developments, rather than dedicating time for solving compatibility and integration issues.

The implementation of the Mixed Partitioning Scheme Replication solution architecture, presented in the following sections, requires a good understanding of the Hadoop system. Therefore, we start by describing the MapReduce programming model, the fundamental concept of the storage and processing framework. Then, the Hadoop Distributed File System architecture and some of the implementation details are presented. Finally, the resource management process and job execution mechanisms are described in detail.

4.1.1 MapReduce Programming Model

The fundamental concept which defines the architecture of the Hadoop system components is the MapReduce programming model. MapReduce applications are built using the *map* and *reduce* functions. The map function is hereby used for structuring and filtering the raw inputs, while the reduce function operates on the pre-processed datasets which has been previously aggregated by using pre-determined criteria. As long as the input data is divisible, the constraints imposed on the *map* and *reduce* processing stages allow the underlying implementation to automatically distribute

the load throughout the cluster nodes. While being processed on the remote machines, the *map* tasks generate intermediary data, which –in the execution phase named *shuffling*–, is sorted, merged and transferred to the cluster resources which were allocated for the *reduce* tasks. Unlike the other two phases which allow the user to define the processing logic, the *shuffling* is not programmable. The sorting and merging operations are based on the implementation of the `hashCode()`, `equals()` and `compareTo()` methods of respective job input and output classes. The following example explains in detail the execution of a specific use case on the Hadoop cluster.

Example 4.1.1. Consider that a hardware expert, working at CERN, wants to calculate the average signal value for specific device measurements collected during the last hour. The person knows that the data storage solution persists the data into files using the CSV format and containing the following columns: Signal Name, Reported Value and Measurement Timestamp. The file system is organized into directories which aggregate daily input for all of the monitored devices. To perform the initial dataset filtering the hardware expert implements the following *map* function.

Listing 4.1: *map* and *reduce* functions example

```
map(String key, String data):
  for each entry in data:
      tokens[] = Tokenize(entry, ',')
      if List("signal_a","signal_b").contains(tokens[0]) And
          IsLastHour(ParseDate(tokens[2])):
          WriteIntermediateResults(tokens[0], tokens[1])

reduce(String key, Iterator values):
  sum = 0
  for each entry in values:
      sum += ParseDouble(entry)
  WriteResults(key, Length(values) > 0 ? sum/Length(values) : 0)
```

Since the directory contains the relevant data for an entire day, each line of the input file(s) must be accessed and verified in order to determine if the signal value was measured during the last hour (the timestamp must match the criteria defined in the `IsLastHour()` method). Additionally, the hardware expert is interested in specific signals, thus each line has to be verified to match the respective filter as well. Whenever the input passes all defined filters, an entry into the intermediate result file is written with a signal name being a *key* and a reported measurement being a *value*. Further, the intermediate data from multiple nodes is sorted (based on signal name

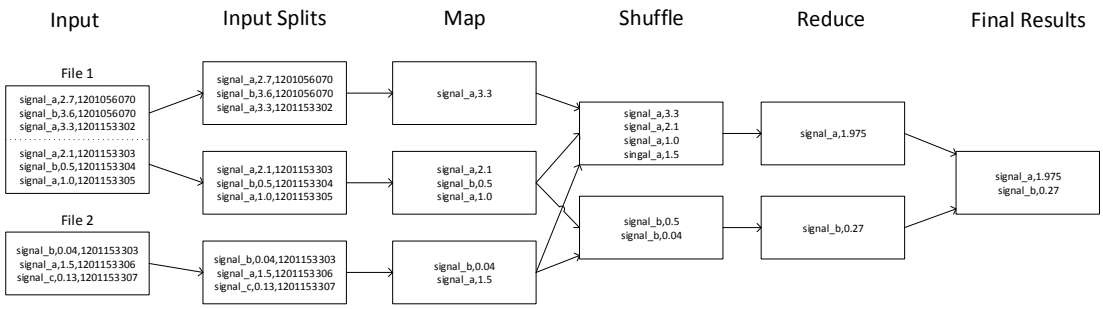


Figure 4.1: The example MapReduce application execution.

alphabetical order), aggregated by the *key* attribute (each merged file will contain only the data for an individual variable) and transferred to the machine which will be performing the *reduce* operation. For calculating the average signal values, the hardware expert also developed the *reduce* function. The implementation hereby calculates the average value of the already aggregated by signal name measurements and writes the final results into the directory accessible by the users. The entire process of this MapReduce application execution on Hadoop system is depicted in the Figure 4.1.

■

4.1.2 Hadoop Distributed File System

The representation and some of the concepts adopted by the Hadoop Distributed File System are very similar to ones of used by traditional, non-distributed file systems, however the similarities end there. Unlike non-distributed namespaces, the HDFS implementation allows it to operate on multi-node clusters and uses remote communication protocols for transferring the data from the original repositories. Additionally, the architecture was optimized for maximizing the data throughput for large scale analysis, thus the most basic building blocks are represented on a completely different scale when compared to traditional file systems. HDFS is based entirely on the software layer to store the data, relying completely on the node-specific file system implementations to communicate with the disks.

The most basic component of the HDFS structure is `Block`(see Figure 4.2). This element represents the remotely stored part of the data which belongs to the determined `File` instance. The HDFS `Block` object is an abstraction made by the software layer, rather than a physical representation, and is therefore capable of

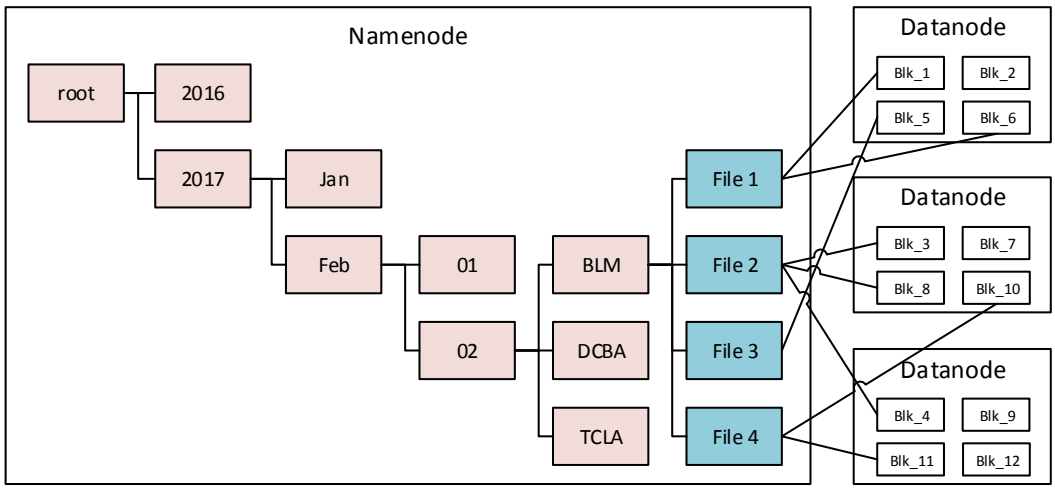


Figure 4.2: The Hadoop Distributed File System structure.

operating on top of the cluster node-local file systems (like *ext3*, *ext4* or *NTFS*). This component, when persisted to the disk, is represented by several files: the large ones contain the data and much smaller ones contain the meta-data. Unlike in traditional file systems the, HDFS `Block` does not allocate unused space. Thus the data input rates and the block size configuration are crucial for the performance of MapReduce applications. Whenever the data is written in one batch which fits the entire block space, there is a high chance that information will be recorded on adjacent disk segments. In this case, the retrieval throughput will be much better when compared to the partial writes, which significantly increase the probability of scattered segments to be assigned for storing the data. The `Block` instances are always assigned to a single `INodeFile` object, which corresponds to the *file* concept adopted in traditional file systems. Furthermore, the `INodeFile` objects are organized into the tree data structure using the `INodeDirectory` instances. Both classes inherit the `INode` attributes, which allows managing the relations between the file system components (through the child-parent relation model) and handling the pre-determined user-defined attributes, like ownership and permissions information.

The Hadoop architecture delegates the file system related responsibilities amongst the Namenode and the Datanode modules. The Namenode is the file system - master - service, which manages the namespace and performs the meta-data-related operations. In order to be able to efficiently process the meta-data requests, the whole

data representation is maintained in memory by an instance of the `FSDirectory` object. At the same time and in an effort to prevent losses of meta data of the file system in case of a failure, the `FSNamesystem` instance simultaneously persists the performed operations onto the disk. The actions are first written into the temporary *edit* files, which are further aggregated into the main *journal* file - the *fsimage*. Besides the `INode` objects, the namespace audit system is used for storing the block mapping, managed by the `BlockManager` instance. In addition to the file system structure, the Namenode manages the replication of the HDFS data. The resources are constantly monitored and whenever an under-replicated block is detected (for example in case of a cluster machine failure) the relevant data is moved from the existing location to the newly allocated resources. The responsibility of the data storage is assigned to the Datanode - slave - instances. The Datanode implementation maintains the `Block` objects with the respective meta-data on the cluster node-local disks, while being completely unaware of the details and structure of the underlying file system implementation. Whenever the data needs to be written or read from HDFS, the `Block` location is first determined by querying the Namenode and only in a second step the Datanode is requested to provide the data (for this, a direct data stream is opened between the client and the corresponding Datanode).

4.1.3 Hadoop Resource Management

The computing resources of the Hadoop cluster are managed by the YARN service. The master component of the cluster management process is the ResourceManager, which communicates with several instances of machine managers, namely NodeManagers (one instance for each node) and ApplicationMasters (one instance per application). The responsibilities of the NodeManager include the management of the individual machine resources, reporting the current CPU and memory state to the master service and managing the execution container life-cycle. The ApplicationMaster communicates with the ResourceManager to negotiate the allocation of computing resources, further handled by the individual nodes of the cluster connected to the YARN infrastructure. The resource allocation is performed by configurable schedulers, like the `CapacityScheduler` or the `FairScheduler`, which define the policy to distribute and allocate the available execution slots amongst multiple queues or applications. The application submission in a Hadoop system is performed through the ResourceManager, which allocates a container for each of the user requests and requests the related NodeManager to launch its execution. The container executes the ApplicationMaster, which in turn requests additional containers for the execution of the *map* and *reduce* tasks. The ApplicationMaster is framework depen-

dent, thus different processing engines, including MapReduce, need to provide their own implementation of this component. In case of MapReduce, a specific script is executed, which starts the new JVM for each of the tasks and performs the execution on the isolated instances. Finally, the MapReduce application results are written into HDFS and the logs are passed to the HistoryServer service instance. After the execution completion, the NodeManager releases the containers and notifies the ResourceManager.

4.2 Architecture

The integration of the Mixed Partitioning Scheme Replication with the Hadoop system requires the implementation of specific interfaces, which subsequently are integrated into the infrastructure through the framework configurations. The communication layer is enhanced through the `ServicePlugin`, which allows to expose both the Datanode and Namenode service functionalities using pre-defined remote procedure call (RPC) protocols. According to the documentation, the plugins are instantiated by the service instance and life-cycle events are communicated through `stop()` and `start()` methods. The main use of the `ServicePlugin` is likely to be limited to the server-side communications, since the HDFS client protocols will remain unmodified in comparison to traditional Hadoop systems.

The storage behaviour can be altered through the implementation and integration of the `FileSystem` component. This instance is responsible for performing all operations that both client and server modules can execute on the underlying file system solution. The `FileSystem` implementation must be integrated into the Namenode *classpath*. The Namenode recognizes the specific requests by analysing the file system scheme, represented by the prefix in the provided paths (for example in hdfs://localhost/directory/file.txt path the *hdfs* prefix corresponds to the file system scheme). Additional daemon processes can be configured inside the `initilize()` method which is automatically instantiated when the service is started. This module allows the proposed approach to be integrated into the Hadoop systems without losing the compatibility with external services and inherit its failure tolerance properties.

The detailed overview of the Mixed Partitioning Scheme Replication solution architecture is presented in the Figure 4.3. The modules specific to the proposed approach are integrated with the Namenode component (the new elements are represented by shapes with dashed-line borders). Besides the regular file system image storage, the MPSR solution persists system usage data for optimizing the storage layout in case of significant usage changes are developing over time. Additional

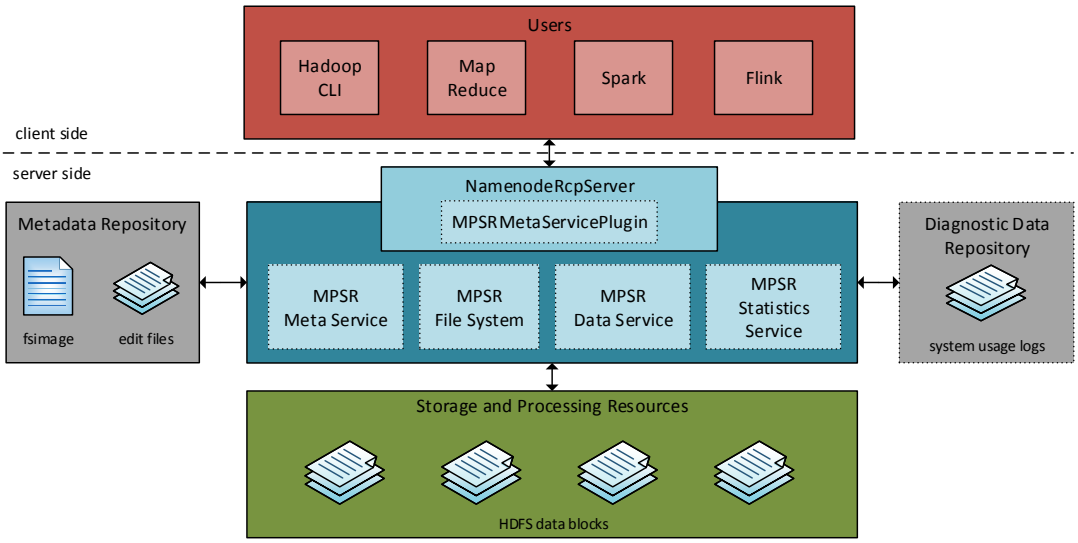


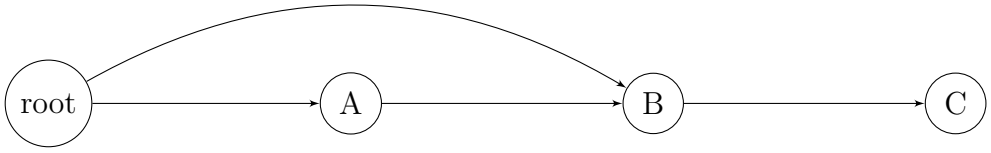
Figure 4.3: The Mixed Partitioning Scheme Replication architecture.

commands are exposed to the cluster administrator for managing the data sources and the partitioning criteria. The results of the infrastructure components remain unmodified, as all required operations are performed directly on the Namenode.

MPSRFileSystem

The core component of the proposed architecture is the `MPSRFileSystem` object. This object is responsible for building the file system representation and maintaining the data consistency and integrity. The namespace is organized into the classical, general tree structure. In case of shared predicate dependencies (when multiple partitioning criteria share the same object attributes for defining the storage scheme), it performs the duplication of the *inodes*. An implementation based on a Directed Acyclic Graph (DAG) was considered for mitigating these issues. It was however proven to be impossible since in specific situations the combination of the graph nodes will result in undesired data merging (as shown in the Example 4.2.1).

Example 4.2.1. Consider that there were two partitioning criteria configured on the storage system. The first one structures the data according to the $\{A, B, C\}$, while the second one uses the $\{B, C\}$ object attributes. The DAG representation for the described use case would connect the *root* instance to both A and B nodes.



The final file destination for data belonging to the two different partitioning criteria will be the same. Consequently, the system will not be able to differentiate between the stored objects, making it impossible for user requests to determine which replica group the file belongs to. Furthermore, applications operating in such directories will process the same data multiple times, reporting incorrect analysis results.

■

For each of the configured data sources, the `MPSRFileSystem` instantiates an individual `root` object to protect the data consistency by avoiding possible predicate collisions. Similar to the original Hadoop implementation, this component maintains the entire file system image in Random Access Memory (RAM) to reduce the I/O load on the underlying storage hardware and to reduce the resulting request processing latency. Moreover, for failure tolerance reasons the namespace representation and operations are mirrored to the persistent repository. Finally, the file system audit information collected from the processed user requests is forwarded to the `MPSRStatisticsService`.

The file system tree representation is composed of the `MPSRINode` objects as the tree structure *branches* and the traditional Hadoop `Block` objects as *leaves*. The `MPSRINode` instances will inherit most of their attributes and functionalities from the original `INode` implementation, including the object identification details, permissions and ownership features. On the other hand, this class limits the components which can compose the file system structure to the MPSR representations, allowing parent-child relations only between specific instances. The `MPSRINode` implementations are the `MPSRDirectory` and `MPSRFile` classes, which correspond to the directories and files respectively. The `MPSRDirectory` objects are used for defining the organization inside the managed namespace, while the `MPSRFile` maintains a link between the file system representation and the physical (remote) data storage through `Block` objects.

MPSRMetaService

The only component which does not have a corresponding counterpart in the Namenode architecture is the `MPSRMetaService`. The main purpose of this module is to

render Hadoop workload-aware, and make it therefore capable of operating efficiently with multiple partitioning criteria. The service exposes an administration interface, which allows the infrastructure administrator to configure the data sources and the management of the related partitioning schemes. The configured scenarios are maintained in memory and persisted to the disk for the reasons which were explained in detail in the previous section.

The `MPSRDataSource` object represents the input data sources. Whenever multiple input origins deliberately share the same partitioning criteria - and when merging does not break the consistency of the data storage - the same `MPSRDataSource` instances can be reused. Otherwise, a new object must be configured for individual data sources. All of the `MPSRDataSource` instances maintain the originally defined replication factor configuration, the predicate sorting order and the list of the corresponding `MPSRPartitioningCriteria` objects. The predicates represent the object attributes which characterize the input data and further used for the definition of the partitioning schemes. The `MPSRPartitioningCriteria` instances manage the sorted multi-key map of the predicates with their associated `MPSRPartitioningAlgorithm` objects. Thus, unless the user request specifies explicitly the data source for retrieving desired stored values, the `MPSRMetaService` automatically determines the most appropriate partitioning scheme for executing the query. In this case, all of the `MPSRDataSource` instances are evaluated and the best matching data source is determined by analysing each of the configured partitioning criteria. Whenever there are multiple results with the same score the system selects the one which at the current moment is less loaded with user requests.

Additionally, the `MPSRMetaService` instantiates and controls the single instance of the `MPSRResourceMonitor` object. This component manages the cluster resource relation with the chosen replication and partitioning strategies. Taking into account the configuration, the nodes are initially divided into two categories, elastic or permanent. Based on the replication preferences: the first group is primarily used for temporary storage of frequently accessed (or *'hot'*) datasets, while the machines which belong to the later category assure the persistent storage of the whole repository. The cluster resource classification and allocation is performed by the `MPSRScoringAlgorithm` which, according to the implemented logic, performs re-balancing of the cluster whenever new nodes are connected and in case nodes are disconnected from the infrastructure (including the manual machine decommissioning). This component makes extensive use of the `MPSRStatisticsService` in order to determine the optimum configuration for the observed workload partitioning and replication scheme. Since the original Hadoop implementation does not allow direct subscriptions to the `DatanodeManager`, the `MPSRResourceMonitor` instance

is a daemon process, which performs periodic checks of the cluster status and invokes the MPSRScoringAlgorithm when an event of interest occurs. After performing the required re-balancing calculations, the associated data operations module is notified in order to execute the revised data structuring plan.

MPSRDataService

The `MPSRDataManager` is the module responsible for executing and controlling all data-related operations. This component communicates with `MPSRMetaService` and occasionally receives notifications which contain the instructions for the storage node allocations and the required data transfer operations. The request is assigned a priority based on its type and scheduled for execution taking into consideration the ongoing operations of the `MPSRDataManager`. The action types supported by this module are: elastic replica management, resource rebalancing and failure recovery.

The operation with the highest priority is *failure recovery*. The main reason for that is that an insufficient replication can compromise the data integrity, since the combined probability of unrecoverable loss of stored information in this case significantly increases. Additionally, a permanent loss of one or several nodes could result in a considerable performance deterioration for some workload categories, affecting not only the execution of the user requests, but also all elastic replication management operations. Finally, the temporary storage can still maintain a partial copy of the data lost on the permanent node, in which case the recovery process would be completed much faster in comparison to a case where the missing dataset requires full reconstruction. The data recovery process starts by inspecting the extent of the data loss. First, the elastic replicas are checked for possible partial dataset copies, which - in case of a positive outcome - are transferred to the permanent storage using efficient commands like *distcp()*. If no copy can be found, the `MPSRMetaService` is invoked in order to determine the closest matching partitioning scheme (i.e. the one which will allow to migrate the data with the least possible effort). Based on the data retrieved from the elastic replicas and the determined loss boundaries, a MapReduce application will be generated and executed to automatically transform data from an existing partitioning criteria to the scheme which has missing information. Finally, the re-constructed scheme is sorted according to the specific configuration of the partitioning criteria, followed by the data filtering to remove the duplicate entries.

The intermediate priority operation is the *elastic replica management*, since it does not require the transfer of large amounts of data but rather take advantage from the possibility that some of the use cases can retrieve data from the more performant temporary storage layer. Periodically, the data is classified according to the

pre-determined metrics (data access frequency for example) and transitionally over-replicated to the cluster nodes specifically allocated for such operations. Whenever the resource re-allocation procedure is executed, previously stored information blocks are purged from temporary storage and the new assets are written to the disks of elastic replicas. The data transfer operation is performed using efficient commands such as *distcp()*. The periodicity of the elastic replication is controlled according to a user defined threshold, as a constant data transfers across the cluster might significantly harm the performance of the system.

Finally, the *cluster re-balancing operation* is allocated the lowest priority, since it requires the transfer of large amounts of data between the nodes. The defined partitioning scheme is at first inspected to determine the delimitation which would allow to maintain a balanced resource distribution. This information is then correlated with the cluster resource usage statistics from the `MPSRStatisticsService` and the re-balancing plan is defined based on the combination of both factors. Since the data is transferred across segments belonging to the same partitioning scheme, the *distcp()* command is used for executing the operation. After the information is successfully uploaded to the new location, the segment which contains the previous copy is de-allocated and can be reused for storing and other, fresh content.

MPSRStatisticsService

The `MPSRStatisticsService` is used to collect all relevant system usage data. The measurements are forwarded to the pre-defined metric processors, which, besides immediately flushing the raw values to the disk, allow to define additional logic for information pre-processing and enhancement. In-memory buffering is not supported by the `MPSRStatisticsService` for scalability reasons of the Namenode, which itself already maintains the whole file system representation in RAM. Since the operations which handle the collected cluster usage statistics are not time critical, minor additional delays are considered acceptable. The user-defined threshold determines the periodicity of the log aggregation to avoid filling the Namenode storage space with diagnostic data. This allows to preserve less-detailed diagnostic data for long term optimizations, without compromising the service availability.

4.3 Prototype Implementation

The prototype implementation of the Mixed Partitioning Scheme solution was mainly driven by the need to validate the simulation results of the model described in the previous chapter. Additionally, the developed application should allow to study the

fundamental characteristics of our proposed approach, before the final, full-scale implementation. Therefore, only the core features of the Mixed Partitioning Scheme Replication technique were addressed in the prototype design and development. The implementation was integrated directly into the Hadoop source code, which allowed to maximize the use of the existing file system and cluster management mechanisms, thus reducing for this first stage the significant effort that the development of an independent plug-in to support the desired functionalities would require. The Homogeneous MPSR paradigm was chosen for the implementation, mainly because of the simplicity of its architecture and the predictability of the executed operations.

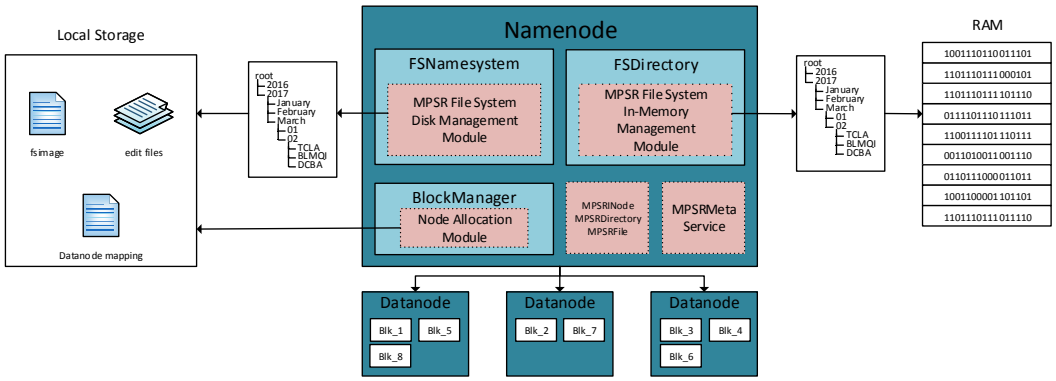


Figure 4.4: The Mixed Partitioning Scheme Replication prototype architecture.

The major modifications were introduced in the Hadoop objects `FSDirectory` and `FSNamesystem` (see as well Figure 4.4). These changes were integrated into the source code without impacting the original system functionalities, since there was no intention of implementing additional features that would require managing the staging and intermediary data produced by the MapReduce applications. The new methods that were integrated into the `FSDirectory` sources are inspired by the original implementations, but adapted for operating on the `MPSRINode` instances. The prototype supports the minimal feature set required for achieving the experiment goals, namely operations like directory creation, file creation, appending and status retrieval. The operations required for handling the namespace representation on the persistent storage were integrated into the `FSNamesystem` instance. The changes - once again - were based on the original implementation, but the source code was extended to support the reconstruction of the MPSR file system representation through the *edits* and *fsimage* files. Additional changes for storing the namespace representation were introduced into the `fsimage.proto` file, which defines the scheme used

by the Protocol Buffers (*Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.*, n.d.) for writing and reading the namespace image to and from the disk.

Only the introduction of workload-awareness into the Hadoop system required the definition and implementation of a completely new module for the management of meta-data. In the MPSR prototype, the `MPSRMetaService` was integrated directly into the Namenode sources. The main responsibility of this new module is the management of partitioning criteria relations with the individual Datanodes. First, the administrator has the possibility to define the replica groups, characterized by an ordered list of the predicates. The relation between the predicates and the concrete partitioning scheme are defined through an interface which supports both simple and complex mapping (for example multi-predicate hash). As a second step, once the service is started the cluster nodes are allocated to individual partitioning criteria using the Round-Robin algorithm (Rasch, 1970). The allocations are stored into the meta-data file since after a potential system failure or restart the machines which already store MPSR data need to be assigned to the correct replication group. Finally, managed associations are exposed to the rest of the Namenode services, in order for the user requests to be routed to the appropriate data sources.

The application execution is scheduled using the default implementation of the `CapacityScheduler`. According to its definition, this scheduler is very similar to the *S2* approach described in the previous chapter. Very much like the simulated component, the `CapacityScheduler` prioritizes data-local executions (i.e. the execution of jobs on optimized resources). In case the size of the input is too large or cluster resource usage is out of balance, the processing is distributed across the entire infrastructure (and the data is transferred to remote machines from the original source). The user requests are associated to individual queues and are processed in a per-queue FIFO order.

The data management process is very similar to the traditional approach applied on Hadoop systems. The main difference lies in the block storage and replication mechanisms. Unlike in the original implementation, after being stored the files are not automatically replicated throughout the cluster unless the replication factor is larger than the number of the configured partitioning criteria. For the reasons explained in Section 3.1, the replication process control is partially granted to the external data ingestion tools. Additionally, the mechanism which detects under-replicated blocks was modified in order to exclude the MPSR file system from operations which ensure the HDFS data distribution. Unlike in the traditional Hadoop systems, additional knowledge has to be used in the MPSR prototype to determine the most appropriate candidates for storing the data blocks. The list of predicates, passed along

with the collected information allows the modified version of the `BlockManager` to build the list of the `excludedNodes` and `favoredNodes` to control in the following the data placement on the specialized resources. The list of `excludedNodes` contains machines which were assigned a partitioning criteria which is different from the one associated with the input data. On the other hand, the list of `favoredNodes` contains the nodes which are suitable for storing data of the respective structure and the target destination is therefore picked randomly from the available options.

4.4 Performance Study

This section describes the details of the developed prototype performance evaluation which was performed to evaluate the characteristics of the Mixed Partitioning Scheme Replication integrated into the Hadoop solution. First, the partitioning schemes and related use cases were defined. The configuration of the initial experiment setup is very important for the execution of the appropriate benchmarks, as it needs to conform with the simulation environment and provide a good approximation of the storage and processing solution currently deployed at CERN. A detailed overview of the developed benchmarking application used for injecting different workload configurations is presented. Finally, the metrics used for evaluating the performance of the MPSR solution are described along with the obtained results and the related model validation analysis.

4.4.1 Workload Analysis and Definition

The identification of the workloads, which the next generation LHC transient data recording and analysis solution will be serving, was performed in two phases. First, the queries executed on the initial versions of the CERN Accelerator Logging Service (CALS) and Post Mortem (PM) systems were studied. The analysis was performed over an extensive period of time to be representative of the different modes of accelerator operation and therefore to identify possible, periodic workload deviations. Afterwards, a survey of the current and future needs of hardware experts and accelerator operators was conducted to determine future use cases which might not yet be covered by the current system workload analysis. The main motivation for this later study was that there are today numerous constraints imposed by the deployed storage solutions. Besides the limitations on the retrievable data size, both architectures ignore the signal attributes which can be used for querying the data in the future if the means are available.

CERN Accelerator Logging Service Workload Study

The CERN Accelerator Logging Service (CALS) provides the possibility to retrieve data stored through an API, which was additionally instrumented for controlling and monitoring the service availability. The data extraction interface tracks every remote method invocation and stores the audit information into CSV formatted log files. The log file repository is periodically cleaned to preserve the available storage space. The reported metrics include the information about *i)* the invoked method and respective arguments, *ii)* the query execution start timestamp and duration, *iii)* the number of retrieved rows and *iv)* the application which performed the data extraction, along with the user of the application. Despite the fact that most of the collected information followed the same pattern, for unknown reasons in some of the entries the API method argument names was missing. Consequently, additional logic needed to be applied to the log files in order to normalize the entries before further analysis. Finally, in the current system architecture multiple distributed API instances are operating simultaneously, writing the audit information to different locations and therefore this scattered data had to be merged prior to the analysis.

The preliminary analysis of the CALS workload was performed using a specifically developed application. The parametrization allows to define the desired time range and data sources for filtering the measurements. During the data processing stage additional attributes which were obtained from the external services were added to the observations (e.g device location). The architecture allowed us to plug-in different counting interfaces, which were calculating the frequency of a given event for a particular metrics (e.g the frequency of the method invocations). The results were stored into an individual per-metric files and structured using the CSV format. To determine possible workload deviations and (distinct) operation modes, the results were aggregated according to the different modes of accelerator operation (see Table 4.1).

The following Figure 4.5 represents the average number of user requests per day, which were performed by users on the storage system during the different accelerator phases. As expected, during a period of *shutdown*, most (automatic) analysis modules are not running, since the majority of the devices are powered off. The *hardware commissioning* phase is characterized by occasional workload outbreaks, related to the execution of the previously mentioned test sequences. During this phase, the queries primarily target devices of a specific type (related to the powering of the magnet circuits of the accelerator), within a well-defined location or measurements obtained during a particular beam state (for example when the beam is being injected through the accelerator complex into the LHC). The full storage load can be observed as early as during the *beam commissioning* phase, when the full integration

Shutdown	Regular stops of the accelerator, where the accelerator and its equipment systems are undergoing major consolidation works and upgrades. Many devices are being installed, upgraded or replaced to further optimise the performance or as a measure of preventive maintenance. There are no beams in the machine and no powering of equipment is taking place.
Hardware Commissioning	Following a long(er) shoutdown, the accelerator and its equipment systems are being prepared for the following operational period. The installed devices are being tested in a controlled environment and with well determined test sequences. There are no beams in the machine, but powering of equipment takes place.
Beam Commissioning	The accelerator is preparing for operation with particle beams. The installed devices are being tested under realistic conditions with continuously increased beam intensities and energies. There are low intensity beams in the machine.
Operation	The accelerator is fully operational. The device measurements are used for beam corrections, performance optimisations, failure detection and machine safety tasks. There are high intensity and energy beams circulating in the accelerator.

Table 4.1: The principal LHC operation phases.

of systems is being tested and the accelerator is prepared for the upcoming physics runs. The observed workloads are very similar to the audit data collected during the *operational* mode, during which the primary objective of the accelerator complex is the continuous collision of particles to maximize the physics output of the machines.

A more detailed result analysis allowed us to extract the workload profiles observed on the CERN Accelerator Logging Service (see Figure 4.6). Due to the similarities observed between the measurements collected for different accelerator phases, the following study presents only the outcomes of the *operational* period (which also represents the longest phase during a calendar year). It was evident that 92.8% of the executed queries target data which was written during the previous 24 hours. This can be explained by the fact that many applications simply extract the data from CALS into external storage solutions, used for enabling additional (off-system)

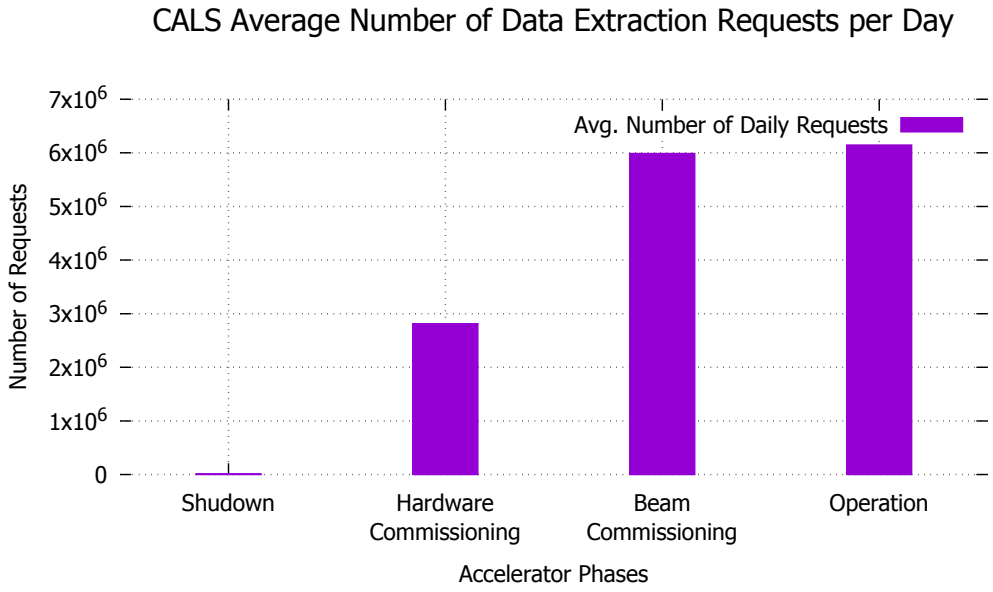
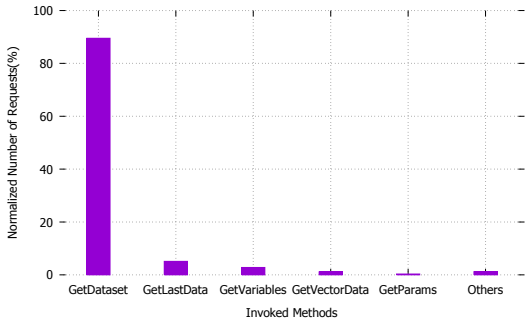


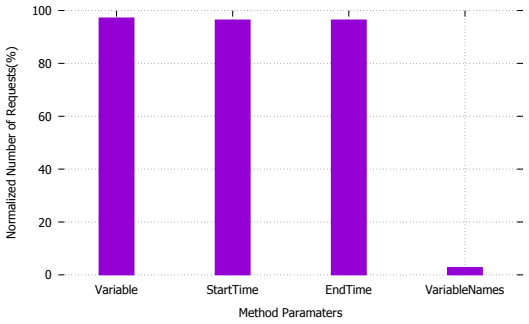
Figure 4.5: The average number of data extraction requests served by CALS daily.

data processing operations. This tendency is mostly driven by the current limitation of the storage, lacking the possibilities for a more efficient data extraction or possibilities for the processing of data close to the persistence layer. This assumption was confirmed by further result analysis, which identified that the `getDataSet()` method, used solely for extracting the raw datasets, was dominating the executed user requests, maintaining the major share of approximately 89.5% for referred accelerator phase. The main filters applied in the executed queries were the variable name and the extraction start and end timestamps. The periodicity of the interval defined by the two timestamps was mostly dominated by the queries retrieving the values in time ranges of an hour (45.5%) and a minute (44.1%). The most queried variables types were the collimators (devices responsible for absorbing high energy particles that start to deviate from the ideal trajectory inside the vacuum chamber). The largest number of requests was extracting data from the LHC points 2 and 8, which corresponds to the locations where particle beams are injected into the LHC, after having travelled down 3 Kilometer long transfer lines from the machines into the so-called injector complex.

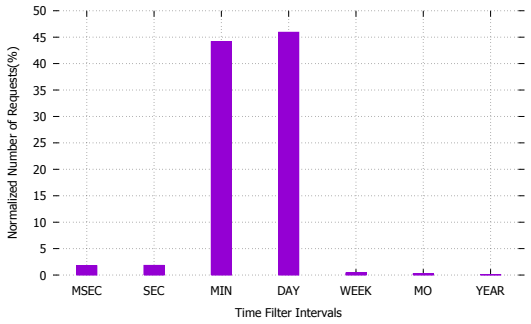
Despite the collected results provide a quite clear overview of the executed workloads, our conclusions are prone do discussion. The main reason for the controversy is the way the API was initially implemented by the CALS development team. De-



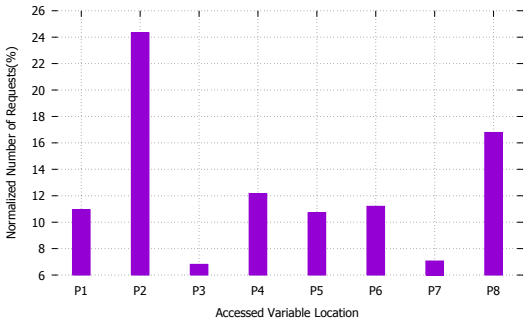
(a) API method invocation frequency.



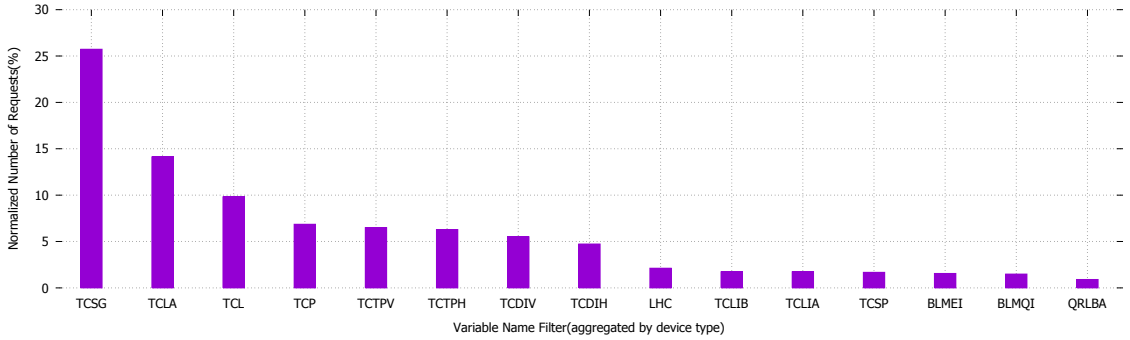
(b) Data filter application frequency.



(c) Most frequently queried time intervals.



(d) Device location access patterns.



(e) Most frequently queried device types.

Figure 4.6: CERN Accelerator Logging Service workload characteristics.

spite the fact that it might have been the appropriate solution at the time to achieve and maintain stable service in operation, its constraints might have a very significant impact on the system usage profiles. The workload study allowed us to identify that

the system is mostly used for monotonous data extractions, since the middleware only provides the most basic statistical functions, like `min()`, `max()` and `avg()`. Even though some use cases could profit from the already available simple analysis features, the lack of *aggregation*, *join* and *sort* operators make it impossible for many hardware experts to take advantage of their existence. Furthermore, the upper bound on the query output size (250Megabytes) has a direct impact on the data intervals extracted by the users. For example, one device which reports the data with 1Hz frequency might produce up to 100Megabytes of uncompressed data per day, whilst a common analysis generally involves hundreds or even thousands of variables. This constraint makes the statistical functions mostly unusable, as even for one week of data per variable the calculations must be split, and the final statistic calculation result might be incorrect. Finally, the lack of appropriate analytical features results in the continuous data extractions performed by multiple hardware teams (to perform more performant and complex off-line analysis of data on local machines). It is therefore very likely that the distribution of the pre-dominant device types in the observed workloads is affected by this shortcoming. Consequently, the next generation solution must be able to preserve the quality of service and be prepared for handling similar requests more efficiently than the previous data storage application. Nevertheless, the observed query profiles and workload characteristics were adopted by the performance analysis application.

Post Mortem Workload Study

The Post Mortem (PM) system does not provide a dedicated workload monitoring component, as the data extraction is not routed through a dedicated API but done via direct file accesses to the data storage. The identification of the query profiles is therefore mostly based on the characteristics of the implemented analysis modules. The analysis modules presented in the following Figure 4.7 are triggered by specific events occurring in the accelerators, resulting in the triggering and simultaneous transmission of many thousands of files to the central storage and therefore periodic workload outbreaks. The data acquisition rates (or event frequencies) in the Post Mortem system are considerably lower in comparison to the CERN Accelerator Logging Service. These result in a file system representation with a significant amount of the considerably small files (1 Kilobyte - 12.7 Megabytes of compressed data), leading to an average of 100.000 files per day. The size and acquisition frequency of the generated input data (further referred as a 'dump') changes significantly amongst the different events occurring within the accelerators. The largest dataset analysis is conducted when a Global event occurs (which is triggered by the complete ex-

traction of particle beams from the LHC). Such a global event requires thousands of signal measurements to be processed in order to detect the root cause of the premature beam dump and/or to determine the safety of the machine to proceed to the following physics run. Still, in comparison to more frequent events, like when data is acquired by the Quality Check for beam extractions from the Super Proton Synchrotron (SPSQC), and therefore have to process the event data several times every minute, the frequency of global events in the LHC is comparably low (typically occurring only a few times per day). It should also not be ignored that the system is simultaneously used for ad-hoc analysis performed by the hardware experts, however the lack of monitoring tools makes it impossible to determine the characteristics of such user requests.

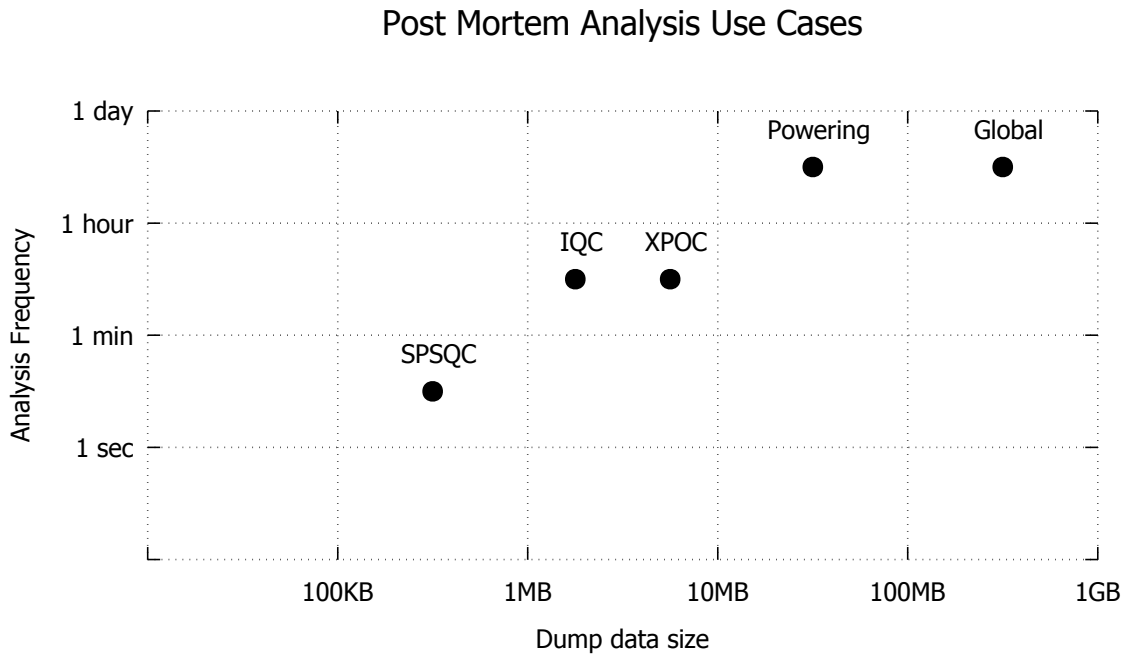


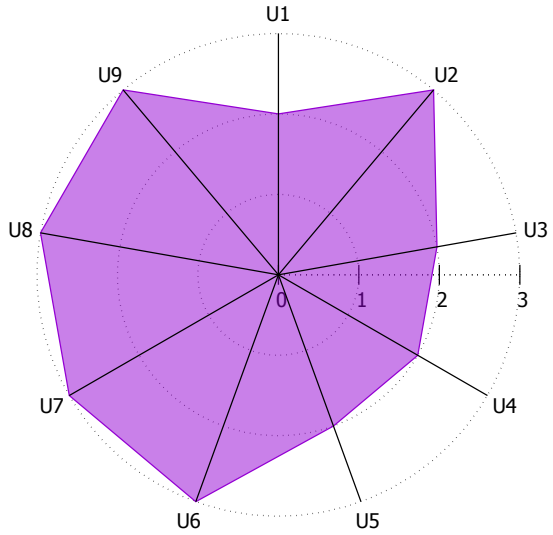
Figure 4.7: The Post Mortem system analysis use cases.

Future Use Case Analysis

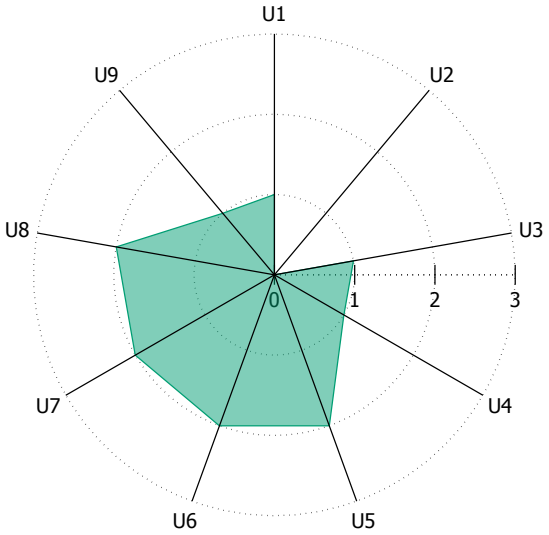
The survey conducted in collaboration with CERN's hardware experts with the aim of identifying possible future use cases was motivated by the limitations of the currently deployed storage solutions and the scarcity of relevant workload monitoring

data. The study aimed at identifying the potential query profiles to be executed by the current data users on the next generation data storage system. This was done assuming that the future infrastructure provides and increased efficiency in terms of data processing and does not impose significant restrictions on the submitted requests. Based on the description of the collected use cases, the list of the most relevant signal attributes was extracted. As a following step, a two-dimensional matrix was constructed, where the rows corresponded to the identified queries and the columns to the extracted data dimensions. Finally, for each of the table entries, the importance factors were estimated. The value of the importance factor can be translated to the potential performance gain obtained when a determined user request is executed on the system which partitions the data according to the corresponding object attribute. For example, if many user requests filter the data based on variable name, it will be beneficial to partition the entities using the device type dimension. Several categories of scoring could be assigned for the query-attribute combination, ranging from zero (no performance gains) to three (major performance gains). Each use case could be assigned a limited number of points, to enforce deeper reflection on the importance of the data dimensions. The following figure 4.8 summarises the results of the survey.

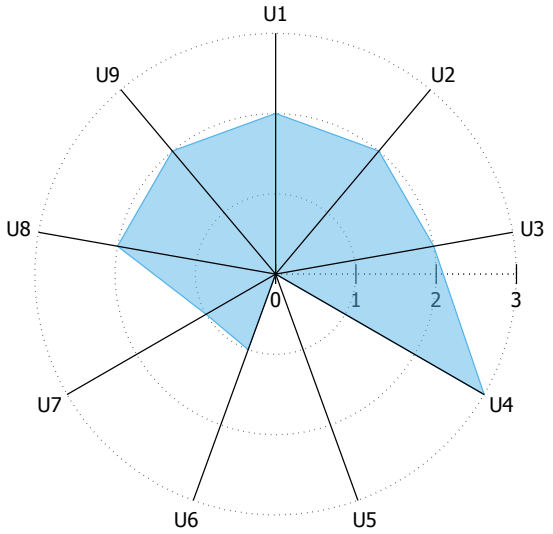
The denominations U1...U9 around the plotted area refer to the identified use cases (for a detailed description of use cases check Appendix A), whilst the values along the axis (0...3), represent the importance of the specific object attribute for each of the possible workloads. According to the observed results, the attribute *time* is considered the most critical for the large majority of the query profiles (its area is the largest in comparison to other plots). The attribute *device type* is not that highly solicited as the *time* attribute, but is considered crucial for some of the use cases, while still having considerable importance for others. The *beam status*, which conceptually can be considered as an abstract time measurement unit, was extremely favourable for some of the identified queries. This can be characterized by a common need of analysing the data strictly during specific periods of the accelerator operation (for example a representative noise analysis can only be performed when there is no beam in the accelerator). Finally, the *location* attribute can be considered not to bring significant benefits for a large number of the possible user requests. For specific operations however it was considered critical for analysis performance (a very common requirement is the analysis of multi-device data in the accelerator transfer lines). The results of the conducted survey confirm the assumption that the limitations introduced in the currently deployed storage systems have a great impact on the framework usage patterns. Consequently, during the benchmark configuration and execution the workload heterogeneity is taken into consideration.



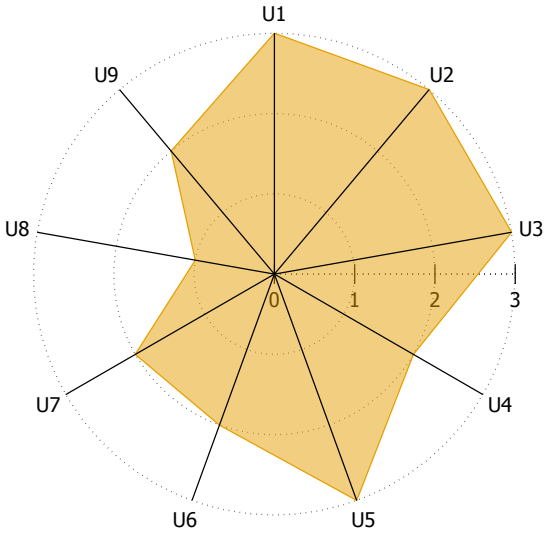
(a) Time relation with use cases.



(b) Beam mode relation with use cases.



(c) Location relation with use cases.



(d) Device type relation with use cases.

Figure 4.8: Signal attributes relation with identified use cases.

4.4.2 Benchmarking Definition

The results of the workload analysis for the transient LHC data recording system significantly contributed to the definition of the benchmarking environment. Sev-

eral factors were considered in order to tailor the infrastructure and performance evaluation tests to the observed scenarios. First, the results were used to define the experimental storage data source and the resulting data layout. Based on the workload observations of the transient data recording system and the expert input, multiple queries with different processing requirements were developed. The emulation of the workload was assured by a highly configurable custom-built benchmarking tool which, besides the execution of the jobs, collected a very wide range of the relevant performance metrics. Finally, for automating the infrastructure deployment process, multiple automatic cluster deployment scripts were developed.

Data Layout

While any of the storage solutions current deployed at CERN could be used as a potential data source for the performance evaluations, several reasons favoured the decision of choosing CALS. First of all, in case both storage systems are merged into a common solution in the future, the amount of the Post Mortem data in the whole repository will correspond to a share of solely 1.25%, meaning that most of the queries will be inevitably executed on the remaining data. Additionally, the continuous amount of data collected by CALS system provides a broad overview on the signal behaviour without containing large gaps between the stored measurements as it is the case for the event based Post Mortem data. Finally, most of the use cases provided by the hardware experts rely on the long-term analysis of continuously acquired values.

The data extraction from CALS was performed using a dedicated data transport service, the Apache Flume(*Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data.*, n.d.) (see Figure 4.9). To retrieve the signal values, a custom data `Source` was implemented. The developed module can be parametrized with the variable name and the extraction time range arguments. Since it was impossible to predict the data size for each of the executed extraction requests and due to the previously mentioned CALS API limitations, the source implementation relied on the divide-and-conquer strategy for retrieving the measurements. Whenever the currently deployed storage was reporting the violation of the data size constraint, the `CalsSource` was dividing the time interval into two parts and continue performing the operation, until requests were able to perform the execution successfully. Further, for failure tolerance reasons, the `AvroChannel` was configured to transport the collected values to the specific component responsible for writing data into HDFS. Since the `HdfsSink` module was included in the default Flume installation, it was required to provide the minimal

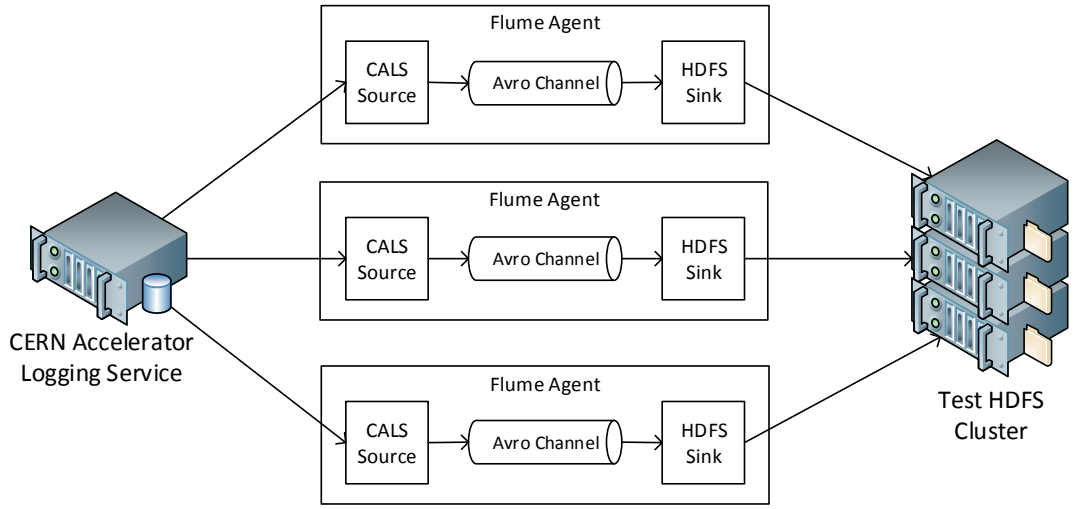
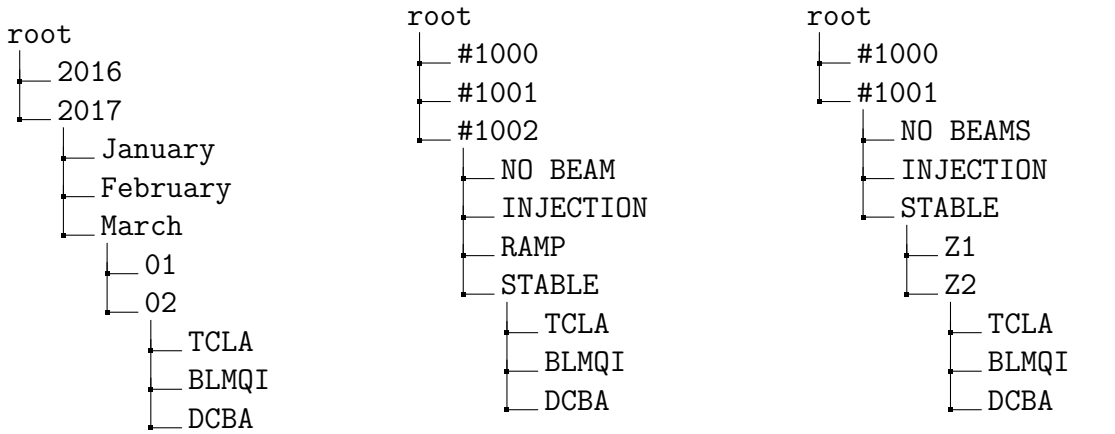


Figure 4.9: The CALS data extraction infrastructure.

configuration for being able to flush the extracted data into the cluster. Multiple distributed agents were spawn in order to reduce the time required for transferring the signal measurements.

The amount of data to be extracted to the test cluster was constrained by the available disk space on the cluster nodes. Each of the machines was configured with two 1 Terabyte Hard Disk Drives (HDD). One of the storage devices was assigned for persisting the CALS data, while the second one was used for maintaining the intermediary job execution results, as for large analytical queries the space requirements are significant. The disk device allocated to the HDFS storage must retain some available space for staging, the job execution result and history data, thus only 60% of the total capacity were reserved for the signal measurements repository. Since 600 Gigabytes of storage allowed us to persist only a very limited dataset, the data for the performance evaluation tests was selected very carefully. Based on the most common time intervals observed in the executed query profiles and the specificities of the data partitioning strategies currently employed by the storage systems, it was decided to extract a single – yet representative - day of data. According to the estimations, it was still impossible to fit the selected dataset on the available storage, thus additional filtering was performed on the device type attribute. The variables were selected once again based on the workload analysis (the ones observed to be queried the most) and according to the future use case definition. Despite the

fact that less frequently queried devices were excluded, the benchmarks executed on the major data sources will be a good representation of the final system behaviour, as long as the real performance characteristics will be similar to the ones observed during the MPSR model simulations.



(a) Time and device type driven partitioning.

(b) Fill number, beam mode and device type driven partitioning.

(c) Fill number, beam mode, location and device type driven partitioning.

Figure 4.10: Example partitioning schemes (TCLA, BLMQI and DCBA hereby represent different devices types installed in the LHC, namely a collimator, a beam loss monitor and a superconducting bus bar segment).

Considering the workload observations and the data organization strategies of the current system the most promising partitioning schemes were defined. The default partitioning criteria was based on the combination of the time and device type attributes (as depicted in Figure 4.10a). Despite the fact that this data organization strategy is not equally optimized for every possible user request, it is an efficient and generic representation for the majority of queries currently executed on the system. When applying the time and device type partitioning to the previously extracted CALS dataset the emerging file system was resulting into a properly balanced representation, with the exception of a few objects which required larger directories to accommodate the collected measurements. This partitioning approach was therefore considered as a baseline for comparison with the MPSR solution.

In order to determine the MPSR partitioning schemes, the observed query profiles and the identified future use cases were than split into several workload categories. The first query profile depends on the same signal attributes as used for the definition of the default partitioning strategy, thus the representation remains the same

as the one depicted in Figure 4.10a. The second category is composed by queries which would mostly benefit from fill number[1], device type and beam mode attributes (see Figure 4.10b). This approach results in a more imbalanced directory structure, since the duration of each of the LHC cycles varies significantly. Finally, the third partitioning scheme is similar to the one previously described, while an additional attribute is added to the data organization strategy, the device location (see Figure 4.10c). Like in the previous approach, this partitioning scheme suffers from unbalanced directory sizes due to the location skewness and the variable duration of the operational phases of the accelerator.

Workload Emulation

Several existing workload emulation tools, like YCSB (Cooper et al., 2010) or Big Data Benchmark (*Big Data Benchmark.*, n.d.), were studied before the decision of developing a dedicated performance evaluation application was taken. A detailed analysis of the existing benchmarking packages features suggested that most of the tools do not provide the support for the generation of classical Hadoop workloads and would therefore require significant efforts for the implementation of such features. Furthermore, existing performance evaluation applications focus on generically defined metrics, common for most of the supported systems, without providing the desired detailed insights into the benchmarked data storage and processing solution. These factors forced us to implement a custom Hadoop performance evaluation tool, flexible enough to be applied to MPSR benchmarks requirements while providing the detailed metrics required for the comparison of the two approaches.

The most basic components of the developed performance evaluation application are the workload query objects, which consist of different `Mapper` and `Reducer` implementations, compiled into the MapReduce application (providing both, simple connections and task chaining). The cluster configuration is shared amongst the different implementations, but specific signal attribute filters can be configured through the individual processing stage parameters. Additionally, the filters are used for constructing the path to the desired dataset and for discarding measurements which do not match the implemented criteria. All of the developed use cases allow to define the custom input and output key-value formats, thus allowing for the possibility of extracting specific columns rather than the entire set of the object attributes. Finally, it provides a possibility of storing the result data in a customized format to allow for an easier interpretation. The current version of the benchmarking application supports several workload query types:

[1] a fill number uniquely identifies an operational cycle of the LHC

- Variable Filtering - a query which returns solely the raw data which matches the configured filters.
- Variable Statistics - a query which calculates simple statistics for the measurements matching the configured filters.
- Exponential Decay Time Constant Calculation - a query which determines the existence of exponential decay(s) and calculates their time constants for measurements matching the configured filters.

The implemented query profiles are combined into the workload scenarios, managed by the benchmarking `ExecutionController` instance. The execution flow of the performance evaluation application consists of multiple simple steps. First, the arguments are parsed to determine the number of requests to be generated and the probability of occurrence of a given query type. The user request generation is in addition based either on the filters provided along with the remaining application arguments or the values are randomly assigned from the pre-defined list of variables and time intervals. Depending on the specificities of the performance study, the request submission is scheduled either using the `ExecutorService` using a fixed-size thread pool (hence allowing the application to control the arrival rate of the user requests) or the `ScheduledExecutorService`, which injects queries at a pre-configured rate, allowing the cluster to manage the queue using its own mechanisms. Diagnostic data is collected both during the query execution and after its completion. In the first case, the daemon process periodically collects data from the Java Management Extensions (JMX) interface. This endpoint is useful for extracting metrics which are independent from the workload characteristics, describing the cluster resource usage at a determined point in time. Additionally, after the completion of each job, workload related information and individual Hadoop per-query metrics are collected. The extracted measurements following the completion of the application describe the interaction of particular queries with the cluster resources.

Infrastructure Configuration

The cluster configuration used for the evaluation of the MPSR performance consisted of ten nodes, nine of which were allocated for the Datanode service (used for data storage and processing), while the remaining was running the Namenode instance. The cluster machine specifications are presented in the following Table 4.2.

The standard Hadoop performance metrics were collected on the Hadoop distribution (version 2.6.0) installed from the Cloudera repository (*Cloudera's Apache*

CPU	8 Core Intel(R) Xeon(R) E5420 2.50GHz
RAM	8 Gigabyte DDR2 667MHz
Disk	2x1TB SATA 7200 rpm

Table 4.2: The Hadoop infrastructure nodes specification.

Hadoop Open-Source Ecosystem, n.d.). The MPSR benchmarking tests were conducted on the same Hadoop version, compiled from the modified sources extracted from the official project repository. The operating system configurations (networking interfaces, disk partitioning, etc.) were identical in both experiments. The data used for processing the user requests was extracted from CERNs Accelerator Logging System and stored in plain text using the CSV file format. Each signal is hereby characterized by the variable name (some devices can report multiple variables), the observed value and the acquisition timestamp. The total size of a single data copy stored on the same node was 592.26 Gigabytes.

dfs.blocksize	128m
dfs.replication	3
mapreduce.map.cpu.vcores	1
mapreduce.map.memory.mb	1024
mapreduce.reduce.cpu.vcores	1
mapreduce.reduce.memory.mb	1024
yarn.nodemanager.resource.memory-mb	8192
yarn.resourcemanager.scheduler.class	CapacityScheduler
yarn.scheduler.minimum-allocation-mb	1024

Table 4.3: The Hadoop infrastructure configuration for performance evaluation tests.

The Hadoop system configurations were the same for both performance evaluation tests - the original Hadoop installation and the MPSR. The most relevant properties are described in Table 4.3. The cluster was operating with a replication factor of three, which is the recommended value according to the HDFS documentation (*The Hadoop Distributed File System: Architecture and Design.*, n.d.). Based on the workload analysis and the estimations of the input size per variable, the block size was set to the 128 Megabytes. An assessment of the block size efficiency has shown that the selected configuration ensures high block fill rates, while, according to different performance studies (Jiang, Ooi, Shi, & Wu, 2010) (Wu et al., 2013), still remaining in the range of the configurations allowing for optimized query execution

times. The memory and CPU management was configured based on the capacity of the cluster nodes. Each machine could assign up to 8 Gigabytes of memory to incoming user requests, handling up to 8 containers simultaneously. The map and reduce tasks were allowed a maximum of 1 virtual core and 1 Gigabyte of memory per executor (allowing up to 8 tasks to be executed at the same time on the same machine). As previously stated, the `CapacityScheduler` was used throughout the benchmarking tests for managing the allocation of cluster resources.

4.5 Summary

In this chapter, a detailed architecture of the Mixed Partitioning Scheme Replication solution was presented. Since it was decided that the proposed approach will be integrated with the Hadoop system, the required components were designed and implemented according to the specificities of the underlying architecture. Despite the fact that the complete system could be developed as a Namenode plug-in, the respective prototype was integrated directly into the Hadoop source code. This decision allowed to reuse some of the existing Hadoop functionalities and therefore to reduce the time required for the implementation of the entirely fresh solution. The developed prototype was able to recreate successfully the behaviour of the MPSR solution, even if only the most basic features were implemented.

Furthermore, in the same chapter, the benchmarking environment was defined. A detailed study of the workloads observed on the CALS and PM systems was conducted. The results were used to identify the origin of the data and to define appropriate partitioning criteria to be implemented for the performance evaluation tests of the MPSR prototype. Taking into account workload profiles, a specific tool used for submitting and monitoring user requests on the system was developed. Finally, the cluster which was used for the benchmark execution is described along with the Hadoop configurations used throughout the tests. The performance evaluation tests of the MPSR prototype which were executed on the configured infrastructure is described in the following chapter.

Chapter 5

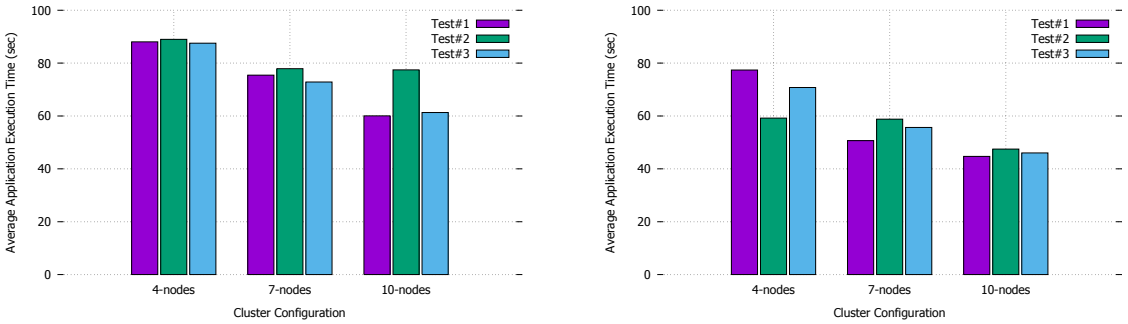
Performance Evaluation

This sections describes in detail the observations from the performance evaluation study and model validation benchmarks. The initial discussion focuses on the comparison between the original Hadoop system with the MPSR approach for different scenarios and use cases. The primary benchmarks are executed to determine the efficiency and basic behaviour of the solution we developed. Furthermore, through the isolation of the individual variables and estimations, the main properties related to the failure tolerance, scalability and infrastructure resource usage of the MPSR are analysed. Finally, targeted performance evaluations are executed to validate the model against the simulation results, which were previously obtained and summarized in the previous chapters.

5.1 Average Query Execution Time Analysis

We started by conducting benchmarks for studying and comparing the average query execution time of the traditional Hadoop deployments and the MPSR prototype. This metric is absolutely critical for assessing the usefulness of the proposed approach, since the most fundamental objective of the MPSR is to improve the throughput of user request processing on the same cluster configuration, while at the same time minimize detrimental impacts on other characteristics of the system. The workload generation and submission application were configured appropriately in order to ensure that we provide the same conditions both for the performance evaluation of the original Hadoop installation and to our prototype. Every cluster configuration changes were performed in the same way for both scenarios, always featuring an individual machine with an instance of the Namenode service and multiple nodes hosting the Datanode service. There were multiple tests with the exact same configuration

but different random seeds executed on each of the cluster setup (Test#1, Test#2 and Test#3 respectively in the figures presented below).



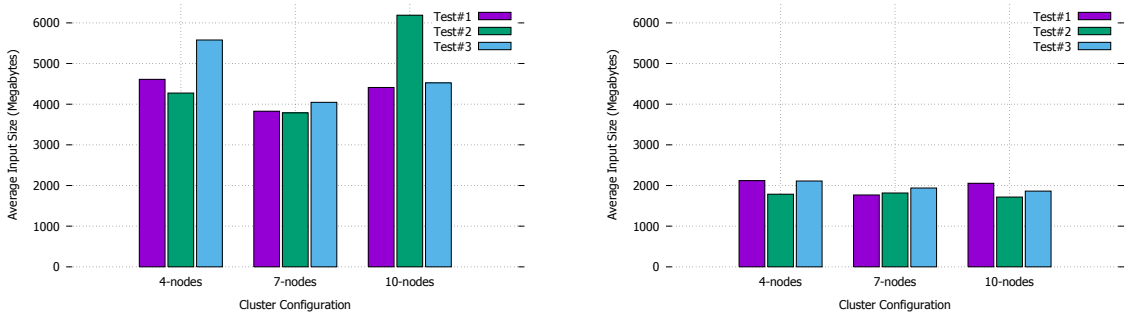
(a) Traditional Hadoop.

(b) MPSR prototype.

Figure 5.1: Average application execution time comparison.

We started this study by evaluating the actual processing time. The obtained results suggest that different application categories submitted through the workload simulation tool, do not differ much from the observed average values, thus this representation will be used in further analysis. The benchmarking tests, depicted in Figure 5.1, shared an identical variable configuration, while the variable name and time interval filters were applied individually (which explains the difference in observed values). Figure 5.1a illustrates the average application execution time of the traditional Hadoop system deployed on different cluster sizes. As expected, a larger infrastructure allows for a faster data processing. The same tendency can be observed on the benchmarks executed with the MPSR prototype (see Figure 5.1b). The comparison between the systems allowed to conclude that the proposed approach was more efficient than the traditional solution in all of the tested configurations. It was also observed that the performance gains of the MPSR prototype were higher in larger clusters, leading to a reduction in the average execution time by $21 - 42\%$ on a 10-node infrastructure, respectively $19 - 35\%$ on 7-node and $15 - 34\%$ on 4-node infrastructures.

According to the preliminary analysis of the Hadoop systems and during the initial phases of the MPSR model definition, it was shown that the application input size has a strong relation with the request processing time. This characteristic was further adopted as a core assumption when the respective simulation engine was built and it continues to be the core feature of the MPSR for further improvements of the data processing throughput of the infrastructure. Once more it could be shown, that the correlation between the application execution time and the input size remained



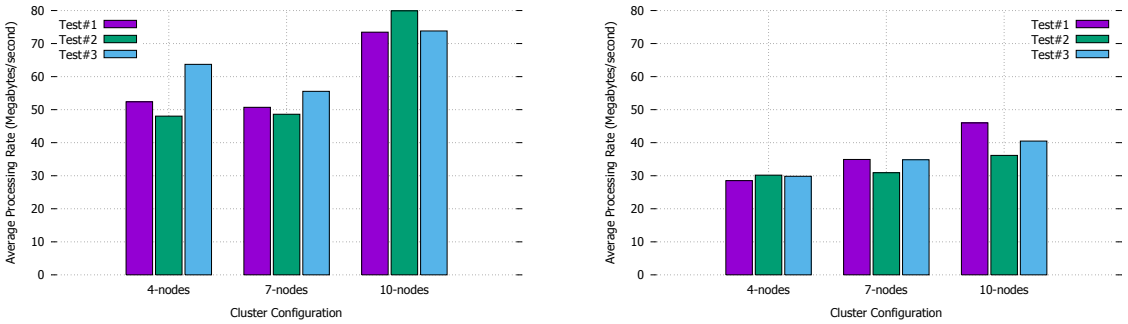
(a) Traditional Hadoop.

(b) MPSR prototype.

Figure 5.2: Average application input size comparison.

very high $(85-99\%)$, while it decreases to slightly lower values $(68-92\%)$ on larger infrastructures. Nevertheless, this parameter continues to be a predominant factor which characterizes the processing time. The average input size for the evaluation of both system performance is presented in Figure 5.2. The observed results confirm that the partitioning criteria applied in the MPSR solution significantly reduce the amount of data the application is required to process in comparison to the execution on the original Hadoop architecture. Despite the fact that one of the partitioning schemes is equivalent to the conventional data representation, the compensation of other data organization strategies allows to achieve impressive reduction rates. The results of the executed benchmarks revealed that the input size was reduced by a factor of 1.8 up to a factor 3.6 in the most favourable cases.

Moreover, the execution time and input size of the MPSR application does scale proportionally in comparison to the conventional solution. The improvement rates obtained during the analysis of the first metric are much lower with respect to the reduction rates of the application input size. In order to understand the root cause of this observation, the data processing throughput was studied in more detail (see Figure 5.3). First of all and before comparing the two approaches, it was important to understand the reason for the 4-node traditional Hadoop system being faster than the 7-node configuration (see Figure 5.3a). If compared with other metrics, the average data processing rate was higher, but the applications were still executed slower. The reason for the observed behaviour is the input data size, which was in average higher during the 4-node benchmarks, allowing Hadoop to achieve higher base processing rates (an effect which is explained in more detail in the following sections). The main analysis and result comparisons suggest that the MPSR prototype was less efficient in handling the application inputs in comparison to the original Hadoop architecture.



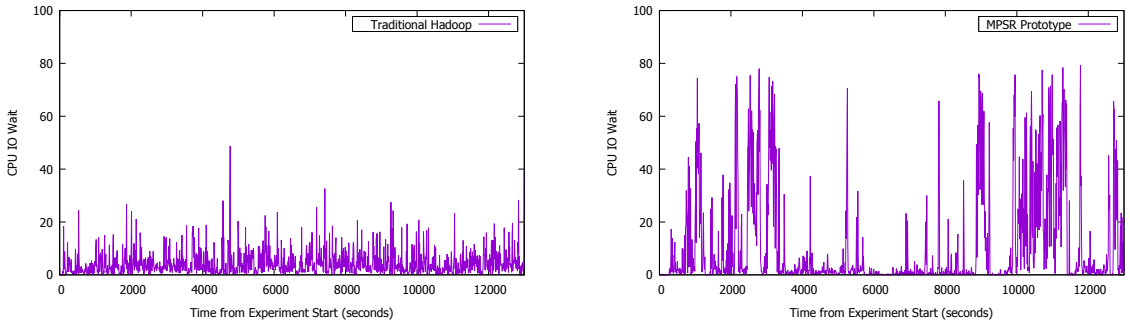
(a) Traditional Hadoop.

(b) MPSR prototype.

Figure 5.3: Average application processing rate comparison.

A detailed study of the observed behaviour suggested that there are multiple factors which could impact the processing rate of the proposed approach.

The first factor which significantly impacts the processing rate of the MPSR approach is the distribution of the data copies across the cluster. Unlike in the original Hadoop system, where every replica uses the same partitioning scheme which allows any application to read the same data from multiple locations, the proposed approach can - in case of highly unbalanced workload distributions - overload cluster nodes with user requests. For example, when multiple applications which belong to the same workload category are successively submitted, the MPSR approach will in the worst case retrieve the data entirely from a single machine, introducing high I/O loads on its disks. In this case, the performance is likely to suffer additionally from the network overhead, as the same machine cannot accommodate the execution of all jobs on the available resources. This behaviour was studied in detail by the authors of (Tandon, Cafarella, & Wenisch, 2013), who claim that high CPU I/O waiting time, induced by concurrent data retrievals from a single node, has a significant impact on the data retrieval and processing rate of this machine. The normalized execution time observed by the authors increases proportionally with the number of concurrent users. The MPSR approach is therefore expected to be affected by the same issue. To confirm this assumption, the cluster was monitored for the symptoms described by the authors. Figure 5.4 provides the details of the cluster node CPU I/O waiting time of a given machine during the execution of the benchmark tests on both systems (whereas a very similar behaviour can be observed on other nodes). Figure 5.4a presents the measurements obtained during the execution of the traditional Hadoop performance evaluation test. The respective CPU I/O waiting time does not present any regions of significant deviations, with only a very few noticeable outliers. On



(a) Traditional Hadoop.

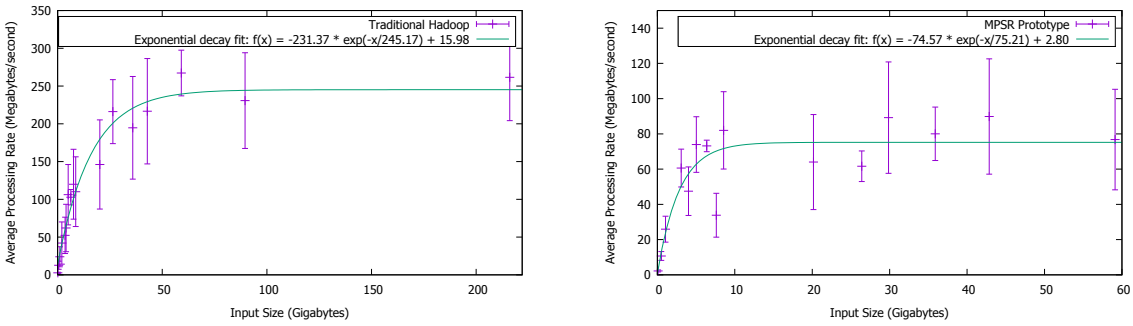
(b) MPSR prototype.

Figure 5.4: CPU IO Wait comparison.

the other hand, the MPSR prototype behaviour displayed in Figure 5.4b is more unstable, representing multiple periods of heavy usage of storage resources, while sometimes being notoriously underused. The average CPU I/O waiting time of the traditional solution during this experiment on the same cluster node was 4.43, while for the MPSR approach it was 10.5.

The second important factor is data-local execution. While being related to the data distribution, it is mostly influenced by the application processing layer, more specifically by the scheduling algorithm. For the traditional solution, in a 4-node configuration (with replication factor three), executions were as expected entirely data-local. When executed on the same cluster, the MPSR results have shown to be significantly lower - namely around 37%. However, the difference becomes smaller on larger infrastructures, corresponding on a 10-node cluster to 96% and 45% on 7-node configuration respectively.

One particularly interesting observation made during the analysis of the processing rate was a non-linear relation between this metric and the application input size. After aggregating the results, similar values were filtered out and combined into the representation displayed in Figure 5.5a and Figure 5.5b. In the following, the observations were fit to an exponential decay function which is also displayed in the aforementioned figures. With a very few exceptions, the measured processing rates align well to the proposed fit. There are two main reasons which will allow an application to achieve higher processing rates despite the higher input size. Firstly, devices with higher data acquisition rates have an increased probability for the HDFS data to be written on adjacent disk segments, allowing sequential reads of large files. Secondly, the larger the input size is, the lower is the overhead of the staging and container creation during the application execution (generally this requires around



(a) Traditional Hadoop. (b) MPSR prototype.

Figure 5.5: Input size impact on average processing rate.

40% of the execution time for a directory with a single file).

5.2 Average Queue Size Analysis

As a next step, the queue behaviour for both systems was studied (see Figure 5.6). Despite the fact that the average execution time is a good criterion to determine the performance of the system, the queue size cannot be neglected, as the request pile-up can render the infrastructure unusable at some point and therefore severely impact the application waiting time (like in case of the *S1* scheduler behaviour described in 3.3.2). The cluster configuration was identical for the performance evaluation of the two systems and the same user request arrival rate was applied. The obtained results (see Figure 5.6 confirm that the MPSR prototype was notably more efficient than the standard Hadoop installation in managing the queue. The MPSR approach was able to maintain the queue size close to zero throughout the entire runtime of the experiment, with the exception of a successive submission of a few applications with large input size. Nevertheless, after some time the measurements decayed back to their original values. On the other hand, the traditional Hadoop approach results show that the queue size started to increase from the very beginning of the experiment. This increment was mostly constant, with the exception of a short period during which the infrastructure managed to recover slightly. Similar observations were already observed in other benchmarks we have executed.

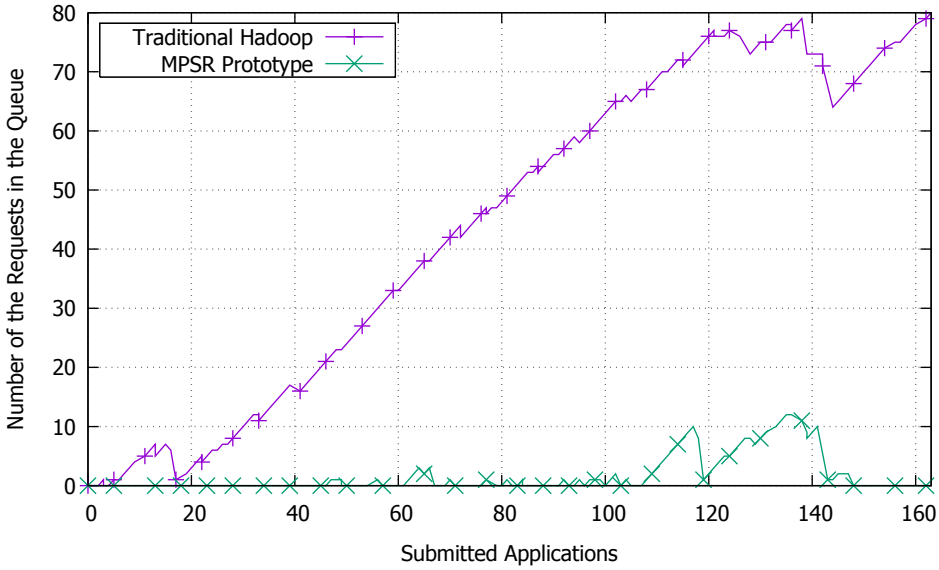
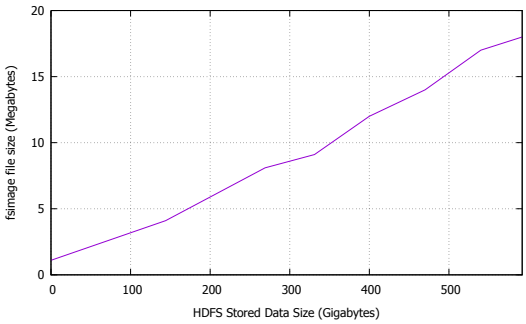


Figure 5.6: Average queue size comparison.

5.3 Namenode Memory Overhead

The in-memory file system representation managed by the Namenode service is a known limitation of the Hadoop system. Large infrastructures, like the one being built at CERN, require a detailed analysis of the data characteristics in order to correctly estimate the hardware requirements of the system. The impact of this issue on the MPSR solution might however be even higher when compared to the standard implementation due to the larger amount of namespace objects required to represent the data.

Two strategies were developed for collecting the required measurements. The first approach was to analyse the size of the *fsimage* file during the migration of the data into the cluster (see Figure 5.7). Since *edit* files, besides the actual data structure, maintain the log of the user operations, the checkpoint operation (*edit* file merging) was executed every time before the *fsimage* file size was measured. Despite the fact that this approximation is mostly suitable for assessing the impact of the namespace representation on the persistent storage, it can also provide us with a rough approximation of the memory requirements. The downside of this measurement method is the fact that the data is stored in binary format. Besides the file system structure nodes, additional meta-data is written into the same file, thus the approximations might differ depending on multiple factors related to the



(a) MPSR fsimage file size evolution while storage is being filled.

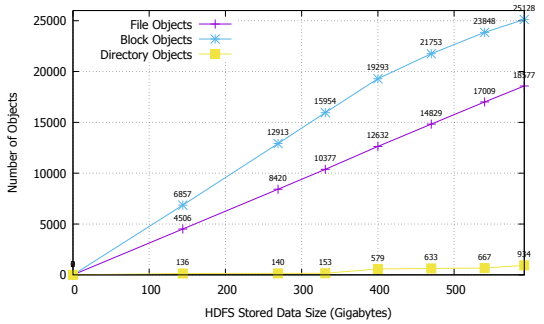
(b) fsimage size comparison when storage is full.

Figure 5.7: fsimage file size.

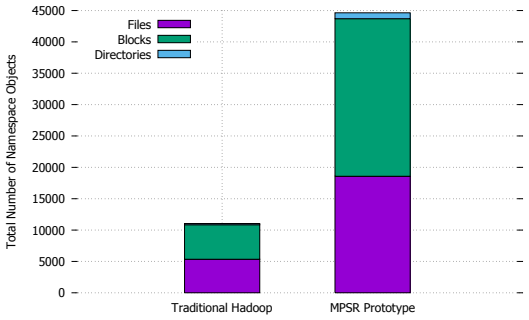
properties of the stored information.

The second approach relied on already known object approximations presented by one of the HDFS architects (Shvachko et al., 2010), for which a more detailed version of the estimations is reported in the Apache Hadoop repository (*Namenode memory size estimates and optimization proposal.*, n.d.-a). Despite the fact that the calculations were performed for an older version of Hadoop, the recent releases did not bring any significant improvements to the management process of the file system. For this reason, these claims are considered to be still accurate and valid. In order to determine the number of objects maintained by the Namenode, the heap memory of the service was inspected after each data migration operation. Furthermore, the object size was calculated based on the estimations presented above and compared with the conventional solution as depicted in Figure 5.8 and Figure 5.9.

The analysis of the heap memory revealed that the final data scheme of the standard Hadoop installation was represented by a total of 5342 files, requiring 5512 files to store its actual contents. The structure of the respective partitioning criteria required 183 directory objects. On the other hand, the MPSR prototype namespace is represented by 18577 files, which require 25128 blocks to store the contents. The number of directory objects in this case was 934. The analysis of the individual partitioning criteria file, directory and block number suggested that the same scheme used in the traditional Hadoop infrastructure required the same amount of corresponding objects. The criteria based on the device type and beam status dimensions required 6230 file objects to accommodate its data copy, while the final scheme (which uses the location attribute besides the previous two predicates) required 7005 files. The block object number was 8427 and 11189 respectively. Further analysis of the file-

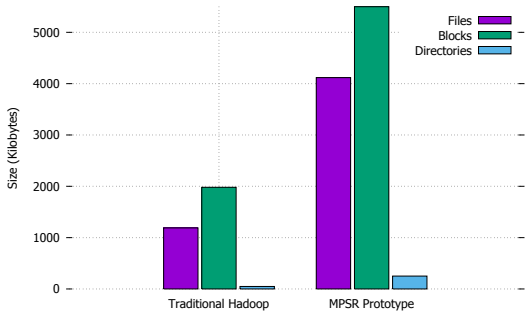


(a) Number of namespace objects evolution while storage is being filled.

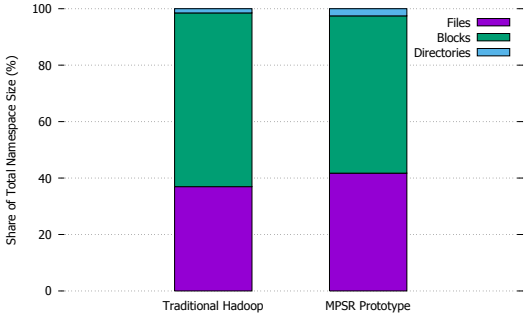


(b) Total number of namespace objects comparison when storage is full.

Figure 5.8: Number of the in-memory namespace objects.



(a) Total size of the namespace objects in-memory.



(b) Proportions of the namespace objects size in-memory.

Figure 5.9: Size of the in-memory namespace objects.

block relations in the MPSR prototype has uncovered a tiny miscalculation, which under certain conditions would pass over the configured block size and spill the data over additional blocks (used merely for storing the sub-part of the last processed entry). It is assumed that improving this functionality of the data migration tool would further align the number of blocks to the number of file objects required for storing the data.

The observed non-uniform directory creation rate is explained by the fact that the migration was performed in multiple phases. During each phase, the data belonging to the same device category is split into multiple segments, as the memory of temporary machines used for this operation did not allow to fit the entire datasets. Thus, in the moment when the first segment belonging to a given device category

was migrated, the number of the created directories was larger.

The number of files shows a different tendency, consistent with the size of the input data. The same behaviour is observed in relation to the data blocks. The number of blocks, files and directories created for the MPSR representation is - as expected - not directly proportional to the same measurements for the conventional solution. The reason for this behaviour is the difference between the partitioning schemes, which require more constituents to represent the same information.

The size calculations of the namespace (see Figure 5.9) were based on the number of corresponding objects and the formulas presented below. According to the estimations described in the Apache Hadoop repository (*Namenode memory size estimates and optimization proposal.*, n.d.-a), the file size (in Bytes) on the 64-bit system is calculated using the Equation 5.1. The first constant is obtained by adding up the size ($152 bytes$) of an empty `INodeFile` object, which represents the file, an entry in the children `TreeMap` of the parent directory (64 bytes), and the back-reference to the parent object (8 bytes). The length of the file name is multiplied by 2, since the `String` represents each character using 2 bytes.

$$size(file) = 224 + 2 * \overline{length(file\_name)} \tag{5.1}$$

The total size of the files in the Hadoop namespace is calculated by multiplying the total number of the corresponding objects by their previously determined average size:

$$size(files) = size(file) * count(file\_objects) \tag{5.2}$$

According to the authors, the directory object size is estimated using the following Equation 5.3. Like in the previous case, the first constant is determined by calculating the sum of an empty `INodeDirectory` object size (192 bytes); an entry in the children `TreeMap` of the parent directory (64 bytes); and the back-reference to the parent object (8 bytes). As before,the directory name length is multiplied by 2, since the `String` represents each character using 2 bytes.

The directory size is calculated using the following estimation:

$$size(directory) = 264 + 2 * \overline{length(directory\_name)} \tag{5.3}$$

The total size of the directories in the representation of the Hadoop file-system is determined by multiplying the total number of the directories with the previously estimated average value:

$$size(directories) = size(directory) * count(directory\_objects) \tag{5.4}$$

Finally, the block object size is estimated using the Equation 5.5. The constants are calculated by adding the size of the `Block` object (32 bytes); the size of the respective `BlockInfo` object (64+8∗*replication* bytes), the reference from respective `INodeFile` entry (8 bytes), the corresponding reference in the `BlocksMap` instance (48 bytes) and the references from all of the `DatanodeDescriptor` (64∗*replication* bytes).

$$size(block) = 152 + 72 * replication\_factor \tag{5.5}$$

Like in the previous cases, the total size of the blocks in the Hadoop namespace is calculated by multiplying the total number of the corresponding objects by the previously estimated average value:

$$size(blocks) = size(block) * count(block\_objects) \tag{5.6}$$

In order to calculate the resulting size of the file-system representation on the Hadoop system, the average file and directory name lengths were determined. The file name is assigned a number in ascending order, which increases every time the associated block is filled. The average file name length in this case was approximately 2 characters. Besides the initial partitioning date criteria, the directories are assigned a device type property with an average length of 3 characters. Finally, the configured replication factor, used in the block object size calculation was 3 for the given infrastructure configuration.

Using the Equation 5.1 and Equation 5.2, the total object size for the traditional Hadoop namespace representation was determined as follows:

$$size(directory_{convetional}) = 264 + 2 * 2.5 = 269$$

$$size(directories_{convetional}) = 269 * 183 = 49227$$

Next, Equation 5.3 and Equation 5.4 were used for estimating the total directory objects size:

$$size(file_{convetional}) = 224 + 2 * 2 = 228$$

$$size(files_{convetional}) = 228 * 5342 = 1217976$$

The total size of the block namespace objects was calculated using the Equation 5.5 and Equation 5.6:

$$size(block_{convetional}) = 152 + 72 * 3 = 368$$

$$size(blocks_{convetional}) = 368 * 5512 = 2028416$$

Finally, the sum of the previously estimates allows r to calculate the total size of the file-system representation on the standard Hadoop installation:

$$size(namespace_{convetional}) = 1217976 + 49227 + 2028416 = 3295619$$

For estimating the size of the namespace objects in the MPSR prototype file-system representation, similar operations were performed. First of all, the length of the file name has changed in comparison to the previous calculations, mainly due to a more balanced distribution of the files across the directories, measuring in average 1.5 characters. The directory name length, due to the inclusion of the predicate name prefix has increased to 6 characters in average. Finally the replication factor required for the block size calculation decreased to 1, since in the MPSR prototype the blocks are not replicated in its traditional form, instead a different copy of the data is stored on another machine.

Using the equations to estimate the file object size (Equation 5.1 and Equation 5.2) the following calculations were performed:

$$size(file_{MPSR}) = 224 + 2 * 1.5 = 227$$

$$size(files_{MPSR}) = 227 * 18577 = 4216979$$

The total size of the directory namespace objects was calculated using the Equation 5.3 and Equation 5.4:

$$size(directory_{MPSR}) = 264 + 2 * 6 = 276$$

$$size(directories_{MPSR}) = 276 * 934 = 257784$$

Next, Equation 5.5 and Equation 5.6 were used for estimating the total size of the block objects:

$$size(block_{MPSR}) = 152 + 72 * 1 = 224$$

$$size(blocks_{MPSR}) = 224 * 25128 = 5628672$$

Finally, the total size of the namespace representation is calculated by determining the sum of the previously obtained values:

$$size(namespace_{MPSR}) = 4216979 + 257784 + 5628672 = 10103435$$

According to the above estimates, the scaling factor for the size of the namespace representation, is not directly proportional neither to the number of the files, nor to the number of block objects maintained by the Namenode service. In comparison to

the traditional approach, the MPSR prototype required 3.47 times more file objects and 4.56 times more block objects to represent the same amount of data. However, the size of the respective file-system representation was 3.07 times greater. This behaviour is considered beneficial for the MPSR approach, since at larger scale the total size of the namespace will grow slightly slower for an increased amount of maintained objects. The factors which contributed to the observed scaling are the file name length and mainly the replication factor reductions.

5.4 Partitioning Overhead Study

The namespace representation was also studied to determine the overhead of the different partitioning criteria on the storage nodes of the modified Hadoop system. We started with a detailed study of the block fill factor. For the standard Hadoop installation, an average block size of 109.98 Megabytes was calculated based on the data size and the number of the previously determined block objects. This would be filling the disk representations on average to 85% of its configured size. Due to the previously described shortcoming of the data migration tool and the increase of created directories, the fill factor of the MPSR prototype namespace was considerably lower. This is a result of the additional partitioning criteria attributes which will in most cases result in a higher number of directories required for structuring the file-system. In average, the blocks were containing 72.37 Megabytes of data, corresponding to the observed 56% fill factor. Nevertheless, the increase of directories, even in the best case (where each file requires exactly one block to store its data) the fill factor would be much lower in comparison to the conventional solution, measured to be approximately 66%.

As a final step, the impact of the block meta-data on the Datanode service storage was also studied. Each block object persisted on any of the cluster machines has an associated meta-data file. This meta-file is mainly used for maintaining the stored data checksums. They are required to perform the periodic data consistency checks, measuring approximately 1 Megabyte. In case of the traditional Hadoop installation, the additional overhead of the block meta-file storage for a 592 Gigabytes repository amounts to a total of 16.128 Gigabytes, since each of the replicas storing a copy of the data block (in this case there are three of them) will require its own instance of a block meta file. The MPSR on the other hand will require 24.539 Gigabytes for representing the same amount of data, since the system maintains different copies of the same data, replicated only once.

5.5 Write Operation Overhead

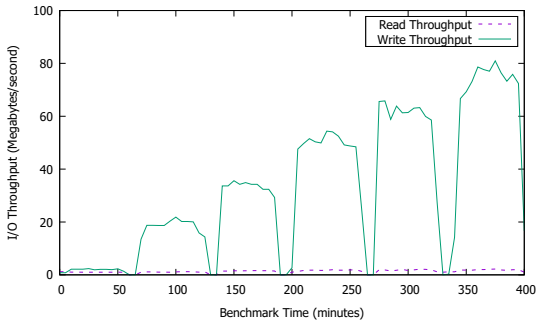
The impact of the concurrent write operations on the execution performance of the application was analysed in order to determine the behaviour of the MPSR prototype in an operational environment. The previously executed benchmarks were studying the different properties of the proposed approach on the static data, which is not the case for the MPSR transient data storage and processing solutions. Both the CALS and PM systems, with different acquisition rates, are constantly persisting new variable measurements and data write operations during the user request processing are constantly happening in the background.

The respective benchmarks were executed on the four node cluster, since it was easier to generate a significant writing load in such a configuration. Three of the machines were running the Datanode service while the Namenode module was deployed on the fourth node. Most of the variables which would introduce uncertainties on the results were removed. Consequently, the arrival rate, the input size (the same variable filter and queried time range were used for different submissions), workload variation and processing speed coefficients were kept constant throughout the benchmark. The data write requests were submitted from remote machines, using Hadoop operations, which during the time of the experiment were assigned solely to the generation of the write workload. Similarly to CERN's second generation data storage and processing solution, the nodes were transferring large files (of approximately 100 Megabytes), randomly picked from pre-generated samples.

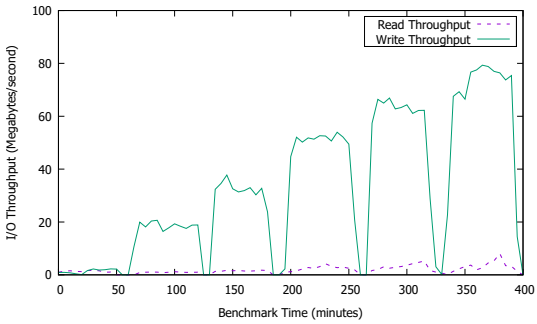
To confirm the resource usage of the MPSR prototype clusters, the average I/O rates were extracted from the available monitoring tools (see Figure 5.10). Despite the fact that the metric sampling interval was chosen to be five minutes, it was possible to accurately track the corresponding measurements. All of the machines behaved in a similar way, showing a constant read and stepwise increasing write throughputs, confirming that the load was distributed evenly across the cluster.

Furthermore, the average execution time and queue size measurements were inspected (see Figure 5.11). Despite the considerable increment in the amount of the data written to the MPSR cluster, the behaviour of the system remained unchanged. Some minor fluctuations were observed, mostly due to the fact that three query categories were submitted, using different partitioning criteria and thus resulting in small differences in the overall processed data size. The average execution time, like the queue size, were almost identical during different write load experiments.

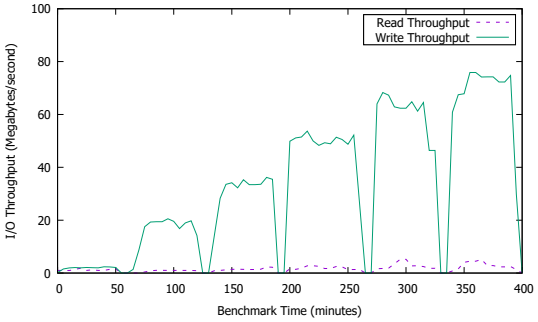
Despite having observed that the configured scenario was highly resilient to constantly increasing write workloads, several factors can change the observed behaviour. The variable and respective time intervals chosen for the experiment were not push-



(a) Datanode #1.

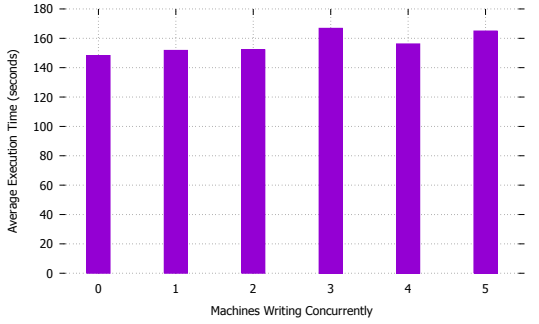


(b) Datanode #2.

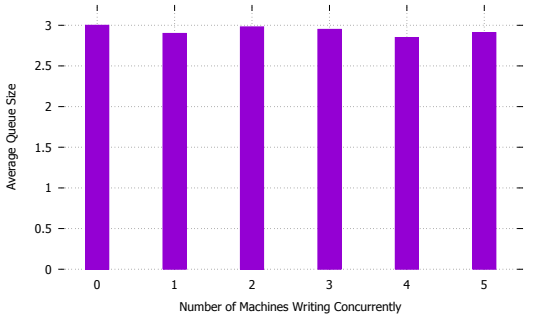


(c) Datanode #3.

Figure 5.10: Write operation overhead study: MPSR prototype cluster I/O rates.



(a) The average execution time during different write workload configurations.



(b) The average queue size during write workload configurations.

Figure 5.11: Write operation overhead study: the MPSR prototype performance evaluation.

ing the cluster to its limits. Data intensive applications requiring a large number of disk accesses both during the initial data extraction and shuffling phase, can at some point throttle the disks. Adding the write requests at this point will definitely aggravate the situation. Furthermore, in case the most common sequential I/O operations will be overwhelmed by the random data accesses, the throughput of the cluster will decrease, anticipating once again the situation when an additional load will make the situation even worse. This is a possible scenario, when either data appending is done in very small intermittent batches (the file pointer and the streams are closed after each small operation) or when the system is used as a repository for very small files. Nevertheless, for the evaluated configuration, the MPSR prototype has shown its efficiency even for the larger write workloads.

5.6 Scalability

In a further effort of assessing the quality of the MPSR solution for a potential use in very large-scale infrastructures like the one currently being built at CERN, a scalability study was conducted. Despite the fact that the modifications of the core Hadoop features were kept to a minimum while integrating the MPSR prototype, the system had inevitably to be modified, as the data storage strategy was different in comparison to the traditional approach. Nevertheless, the main assumptions, as can be concluded from the discussions above, remain mostly unaffected. This allows us to use the observations of other researchers to estimate the behaviour of the MPSR solution since it is much more complex to perform the benchmarks on a larger cluster.

First, the average execution time metric of the application was inspected. When compared to a traditional Hadoop installation, the MPSR prototype was in most cases more efficient, resulting in lower processing times due to the smaller input size (see Figure 5.1). However, the collected results (see Figure 5.1b) do not allow an extrapolation with acceptable error margins for larger clusters, as there are many possible fit functions which can be applied in this case. On the other hand, the behaviour of the MPSR prototype is very similar to the original Hadoop version. The relation between the application input size and the corresponding execution time remains high, thus allowing us to use the observations of other researchers to make the extrapolation to larger cluster sizes. As part of their research, the authors of (Lee & Lee, 2012) conducted a detailed analysis of the impact of the infrastructure size on the cluster processing throughput and average execution time of the application. According to their study, the number of machines in the cluster has a linear correlation with respect to the available processing capacity. On the other hand and as expected, the processing time does not scale linearly with the

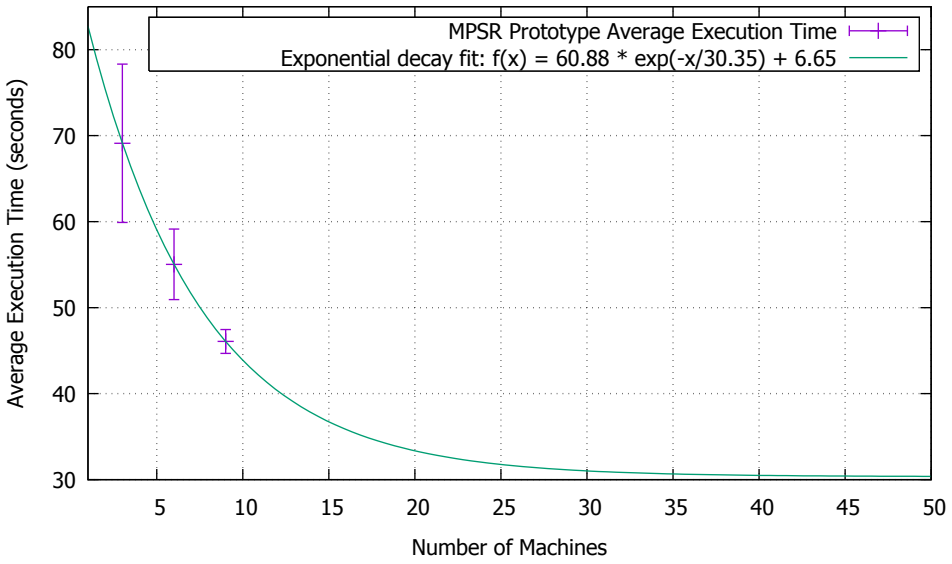


Figure 5.12: Scalability analysis: average execution time estimation.

infrastructure size, since for each application with a limited input size there is always a maximum degree of parallelism. In addition, the concurrency in larger clusters has a considerable impact on the execution time (Zaharia et al., 2009). Despite the fact that the authors do not mention the nature of the affinity between the last two metrics, mathematical methods allowed us to determine that the reported values follow an exponential decay function. Based on the observations, the estimation presented in Figure 5.12 was derived.

Next, based on the facts presented in the previous paragraph, the processing rate of the cluster with the MPSR solution was derived (see Figure 5.13). The observed measurements were used as input for the linear fit function.

The average execution time and cluster throughput estimations will remain valid as long as the computing resources, the stored data structure and executed workloads remain similar to the tested configurations (a more detailed description can be found in previous Section 4.4.2 and Section 5.1). However, a behaviour modification is inevitable in case the original system parameters will be significantly changed. In large infrastructures, like the second generation data storage and processing solution being built at CERN, the data and workloads are heterogeneous, introducing many possible factors which can impact the provisioned behaviour. The data acquisition rates can change, meaning that the block fill factor might be altered. In this case, the files will become smaller in size, consequently requiring more map tasks to process the

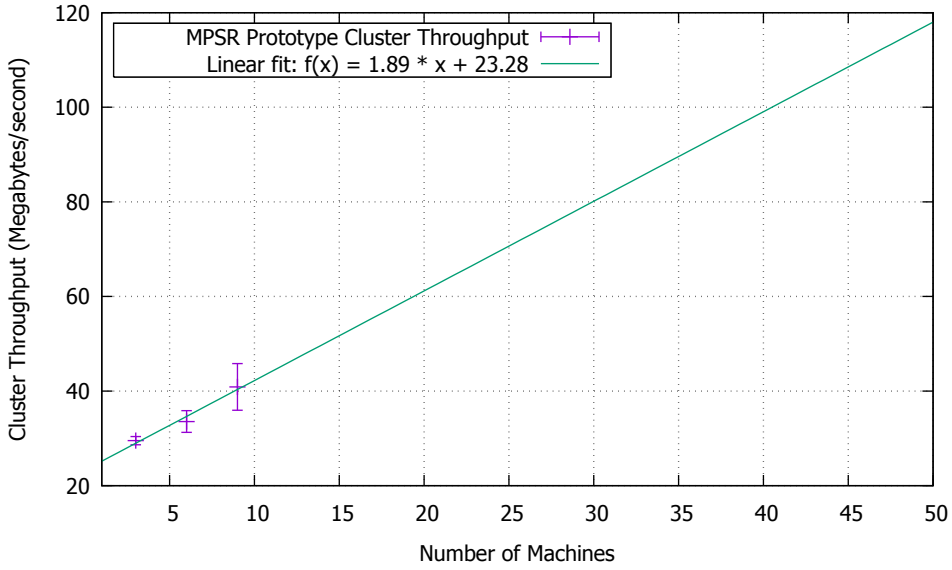
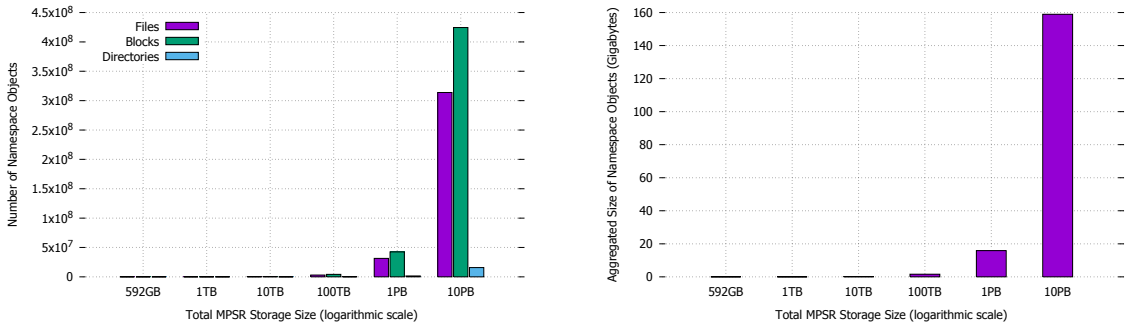


Figure 5.13: Scalability analysis: MPSR cluster throughput estimation.

same amount of data, which in turn can result into the scanario when the execution time is no longer dependent on the input size, being dominated by the container management overheads. This issue is known as the "small files" problem (Tan et al., 2013), which does not only impact the memory requirements of the Namenode to represent the namespace, but also introduces significant overhead when processing the data. Furthermore, in larger systems it is expected that the concurrency will be higher, meaning that the competition for the available resources will be more significant. Despite the fact that Figure 5.5 shows that the throughput eventually stabilizes, this will only happen when the cluster is capable of dealing efficiently with the workload concurrency (the queue does not build-up). In case of a large number of parallel executions, the containers tend to be shared amongst the submitted requests, meaning that more slots will be reserved for reducers and each job will be allocated only a portion of the computing resources (map related operations, like sorting and merging will take much longer).

Finally, in order to complete the full picture of the scalability study, the impact of the MPSR namespace representation on the Namenode memory size was studied. The estimations were based on the previously obtained analysis results of the file-system object and the related size calculation equations (see Figure 5.14). Provisioning for the long-term application of the MPSR approach as a fundamental part of the next generation storage and processing solution for CERNs LHC was performed



(a) Number of file system objects.

(b) File system size.

Figure 5.14: Scalability analysis: file-system representation estimations.

based on the current data acquisition rate, which is roughly 600 Terabytes/year.This allowed defining the hardware requirements for the next decades of accelerator exploitation. Despite the fact that for a 10 Petabyte storage the corresponding memory requirements are considerably high (rounding the 160 Gigabytes), current hardware architectures can easily cope with the issue. For example the Xeon E7 V2 machine (*Namenode memory size estimates and optimization proposal.*, n.d.-b) is capable of supporting up to 12 Terabytes of RAM in an eight socket platform. Nevertheless, the estimations will remain valid for any configuration which is similar to the one used during the performance evaluation tests. The partitioning criteria with much higher cardinality can result in rapid file-system object expansion, requiring not only more directories, but also file and block objects to represent the namespace. Additionally, the data acquisition rates can increase, meaning that more data will be collected by the system for the same time period, resulting in a larger number of namespace objects to be created and maintained by the Namenode service.

5.7 Failure Tolerance

The study of the MPSR solution failure tolerance was performed to determine the possible sources of system unavailability, describe the mitigation strategies and compare the behaviour with the original Hadoop system. While the prototype does not allow to study the recovery times in detail, the previously observed results make it possible to make a speculation about the operations the system will undergo in case of a partial system or hardware failure. Taking into consideration that Hadoop was built to operate on unreliable hardware and that hardware problems are not that uncommon in large clusters (Schroeder & Gibson, 2010), the MPSR solution

architecture was designed to minimize changes to the original failure tolerance mechanisms. Nevertheless, a completely different data storage approach introduces additional complexity and delays into the failure recovery processes. The issues which the Hadoop and MPSR systems might have to deal with can be split into several categories: Namenode, Datanode and TaskTracker failures (Dinu & Ng, 2011).

Service failures of the Namenode are amongst the most critical ones, since this component manages the whole file-system representation. In case of such failur occurs, the user requests cannot resolve the components of the namespace and determine the data location, making it impossible to perform any data processing. The secondary Namenode service, provided with the standard Hadoop distributions, does not solve the problem, since it is neither a backup solution, nor the fail-over component. The main purpose of the additional meta-data module is to reduce the primary service start up time after it was rebooted (Ghazi & Gangodkar, 2015). The master-slave approach adopted by Hadoop engineers results in the Namenode being the system's single point of failure. Without having failure mitigation strategies defined, such issues generally require a manual intervention to repair the problem. In case of a hardware-related loss, a replacement of the respective components and a backup recovery operation must be performed, allowing the disk contents to be restored from the latest available system image. Whenever there is a software problem, the system administrator must intervene to apply the fix and restart operation.

Outages of the Datanode service are automatically treated by the Hadoop system. Permanent data losses, which can occur for example due to the disk failures, are mitigated through the data replication. The `ReplicationManager` module, which performs periodic Datanode inspections, detects the blocks which are under-replicated and immediately executes the repair process, copying the concerned data structures from the still available nodes to the new locations in order to match the configured replication factor. The MPSR recovery process, on the other hand, is more complex since in most cases, due to different organized partitions, a simple data copy operation cannot be performed. Whenever a loss of information is detected, the elastic replicas are first inspected to determine whether any exact copies of the data exist in temporary repositories. Next, the data recovery plan is constructed, requiring a coordination between the meta-service and the machines which store the data using different partitioning criteria. The data re-construction is further executed by scheduling the highest-priority MapReduce jobs, which perform the translation of the closest data representation and store the reconstructed segments into the assigned target nodes. Afterwards, upon completion of the data re-assembly operation, the newly stored dataset is sorted according to the order defined by the meta-data service. The re-constructed and adjacent blocks which were not lost dur-

ing the respective Datanode failure, are inspected for possible duplicates and the new structures are cleaned, based on the inspection results. Finally, the meta-data service is updated with the new location of the reassembled data segments. The whole process of failure recovery within the MPSR solution is more time consuming in comparison to the standard Hadoop scenario, since instead of executing single-stage data copying operation, it requires a multi-phase data transformation, sorting and de-duplication procedures to be performed. Time-wise, due to the sorting algorithm (Quicksort (Hoare, 1962)), it is foreseen to require in the worst case $O(n^2)$ and in average case $O(n \times log(n))$ operations, where n is the size of the originally lost data, unless the amount of data to be processed to reconstruct the segments is exceptionally large.

Finally, there are `TaskTracker` or application execution flow related failures, which occur due to crashes of the node or the related software. The `JobTracker`, the central job execution coordination service, detects a `TaskTracker` failure when the periodic heartbeat is not received during a configurable threshold period. The mitigation strategies applied further depend on the application process status reached at the point. Whenever the execution is in the mapping phase, the intermediary data produced on the failed node is considered unreachable and the entire set of jobs executed on the failing machine is rescheduled on new targets. Despite the fact, that the MPSR solution tackles the problem in a similar way, there are some substantial differences to be noted in the failure recovery pipeline. Due to the differently structured data copies, the exact same input splits cannot be applied to the same application, since there is a very high chance that the adaptation process of other partitioning criteria will result in the processing of duplicated data. Whenever a `TaskTracker` failure is detected, the splits are generated again (on a non-optimized partitioning scheme) and the entire process is restarted from the beginning. This issue can be catastrophic for long-running applications, thus it is required to find a more reliable solution for mitigating this problem. On the other hand, it is possible to pause the execution until the re-construction of the lost node is performed, therefore only part of tasks have to be rescheduled. In this case the behaviour is very similar to the original Hadoop pipeline, but with extra time required for the rebuilding of datasets. A `TaskTracker` failure during the reduce phase has more severe consequences on both systems. The `Reducer` is declared faulty in case it consistently fails to retrieve the `Mapper` outputs, in case the shuffle operation is failing or the processing is in a slate status. In this case, the application must be submitted a second time, both in the standard Hadoop and MPSR approach as no strategies exist to mitigate problems in this execution phase.

5.8 Model Validation

The validation of the proposed MPSR model remained one of the most important challenges to be achieved by this research. Despite the fact that the observations prove the efficiency of MPSR prototype, the configurations used throughout the performance evaluation were limited and influenced by multiple sources of uncertainties. Consequently, the behaviour of the proposed approach in different scenarios can be estimated only through the simulations. In order to increase the credibility of the observations and obtained results, two strategies were developed for validating the MPSR model: a comparison with existing formulations and isolated, individual variable benchmarks.

5.8.1 Comparative Analysis

We started by conducting a comparative analysis between the MPSR model and existing solutions. Amongst others, the following work (Lin, Meng, Xu, & Wang, 2012) presents a very detailed, step-by-step, description of the Hadoop computational pipeline for both map and reduce tasks. According to the authors, their method resulted in a very good approximation, deviating in most cases by not more than 5% in comparison to the actual application execution times. For validating the MPSR model, the map task processing characteristics were inspected and compared first. The **map execution time** is represented as a vector of ten sequential, parallel or overlapping steps, further combined into several phases.

- The initialization phase consists of the user request interpretation and an application-specific delay for pre-loading the file into memory. In case of the MPSR model, the time which these steps require are constant, since the cluster configuration remains the same, as well as the application submitted for processing.

- The data processing phase consists of multiple step combinations, executed sequentially and in parallel. The first one is the initial (and in most cases sequential) retrieval of input data, followed by the data transfer to the target processing machine (the cost is zero in case it is a data-local execution). The last operation consists in the data parsing and processing steps, executed one after another. The cost of executing the final phase is chosen based on the maximum value between the three steps. In case of the MPSR solution, the most time consuming tasks are considered to be the data parsing and processing, both tightly related to the size of the application input data (the authors

affirm that the map task execution until this phase is almost linearly related to the total input size).

- The sorting step is performed in isolation from other tasks. During this phase, the individual map task outputs are sorted according to the programmed order. The average complexity of this operation is estimated to be $O(n \times log(n))$, where n is the size of the generated output. In the MPSR model, this step is still considered to be overwhelmed by the size of the respective input. The assumption is correct for the observed workloads on the CERN accelerator storage and processing solution, but can introduce considerable error, when the application generates large amounts of intermediary data.

- The merging phase consists of two steps performed in parallel (since there are many mappers executed at the same time), the data reading and corresponding merging, which once again depend on the data size of the generated map task output . For the same reasons explained above, the MPSR model might suffer from incorrect estimations when the intermediate data size is extremely large.

- The final, disk spilling phase, consists of two parallel steps which perform the merged data serialization and writing. In any case, the time required to perform these operations has a very low impact on the execution time of the total map task.

Like in the previous case, the **reduce task execution time** is represented as a vector of the different steps, combined into multiple phases.

- The initialization phase consists of the staging operation delay, introduced by the Hadoop system required to execute the reducer tasks. In the MPSR model, this time is considered to be a constant, since the cluster configuration remains the same for multiple simulations.

- The transfer phase of the mapper output consists of transferring the files from remote locations to the node(s) which will execute the reduce task. The execution cost is determined by the data reading and transferring times. Like in the original Hadoop approach, the time required to perform this operation in the MPSR solution is considerably low in comparison to other phases.

- The shuffling phase involves the sorting of the entire reduce task input dataset, merging and writing it back to the disk. Since the data is already arriving in a partially sorted state from the map tasks, the sorting time is much lower in

comparison to the original time complexity. The merging step is also much less time consuming when compared to the similar operation performed during the map task, since the retrieved results are already aggregated by the respective criteria. Given the fact that the merging phase has a logarithmic relation with the number of mappers, rather than the reduce task input size, it is possible to assume that besides the relation with the mapper output size, the input size has an influence on the execution time of this phase.

- The final phase consists of the actual reduce task data processing, serialization and writing the results to the disk. In this case, the estimations are entirely based upon the reducer input and output sizes.

The comparison with such an accurate and detailed model has revealed one significant weakness of the MPSR simulation engine. Whenever we deal with use cases which generate considerably high amounts of intermediate data, the relation of the execution time can be significantly influenced by the complexity of the map task sorting and merging steps. Despite the fact that the observed workloads did not result in such a scenario, it is reasonable to assume that such requests will be submitted into the system sooner or later, introducing considerably higher errors in the estimations. Additionally, for the applicability of the proposed approach to a wider audiences, it is absolutely mandatory to foresee such scenarios. For the majority of the remaining phases, the MPSR model with the assumption of a linear dependence between the application input size and the respective execution time, will accurately predict the outcomes.

5.8.2 Experimental Analysis

Another method used for validating the MPSR model was the experimental analysis. Based on the extracted dataset and the MPSR prototype, an additional set of benchmarks was developed with the objective of isolating the same variables assessed during the simulation phase. This variable isolation allows to reduce the uncertainties and the influence of other system properties on the specific test results. Similarly to the analysis conducted with simulated data, the impact of each of the variables is then compared with their associated metrics.

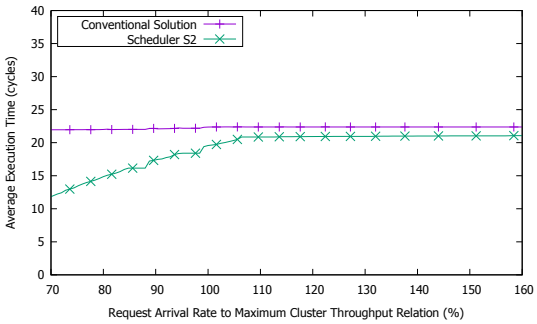
The scenario parameters for each of these benchmarks were carefully selected in order to approximate the configurations to the simulated scenario. However many factors, which still might introduce divergences in the measured results remain present to some extend. First of all, the processing speed coefficients were less beneficial for the MPSR prototype, than the ones used during the simulations.

Additionally, the processing gain factor for each of the evaluated data objects was different, due to the discrepancies in the device reporting rates originating the unbalanced directory structures. Furthermore, it was sometimes impossible to select the configuration for a desired variable value, since the data structure would simply not contain the sample re-assembling the exact required characteristics. Therefore, the main focus of this study is to determine the major tendencies in the obtained results and compare the observations with the simulated scenarios.

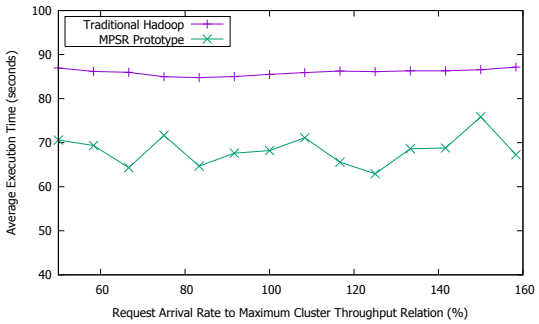
The `ScheduledExecutorService` was used for submitting user requests to the cluster, allowing the infrastructure to handle the queue. The benchmark tests executed for particular variables were parametrized with dedicated signal name and time range query filters, kept the same for the performance evaluations of both, Hadoop and MPSR. The input size experiment was an exception. In this case the pool of signal names was used in both infrastructure configurations, allowing to extract directories with different amounts of splits. For each of the benchmark categories, at most one variable was changed during its whole execution. Only the scheduler *S2* was presented in the results, as it was the best approximation to the `CapacityScheduler` used in the MPSR prototype.

Request Arrival Rate

During the execution of the request arrival rate benchmark, the only parameter which was modified for both the original Hadoop and MPSR prototype was their associated arrival rate variable. Depending on the provided argument, the `ScheduledExecutorService` was submitting the applications to the cluster with varying speeds. Based on the observations, the maximum rate of the cluster processing throughput (at first for the conventional solution) was determined for all of the executions, and the results were plotted based on this generic representation. First, the impact of the average execution time was analysed. According to the MPSR model simulations presented in Figure 5.15a, the scheduler *S2* consistently outperforms the conventional solution. With less competition for the available resources (when the arrival rate is low), it prioritizes the allocation of optimized nodes for processing the user requests, without taking into consideration the overall cluster load. The `CapacityScheduler`, on the other hand, takes into account multiple factors when the resource allocation is determined. For this reason, the execution time of the MPSR prototype (see Figure 5.15b), outperforms the traditional Hadoop approach, but does not quite follow the same tendency as in the simulated scenario. In case of the benchmarks executed on the real data, the execution time has a noticeable variation due the fact that for each experiment the query category combination is different from the previous one.



(a) MSPR model simulation results.



(b) Real systems results.

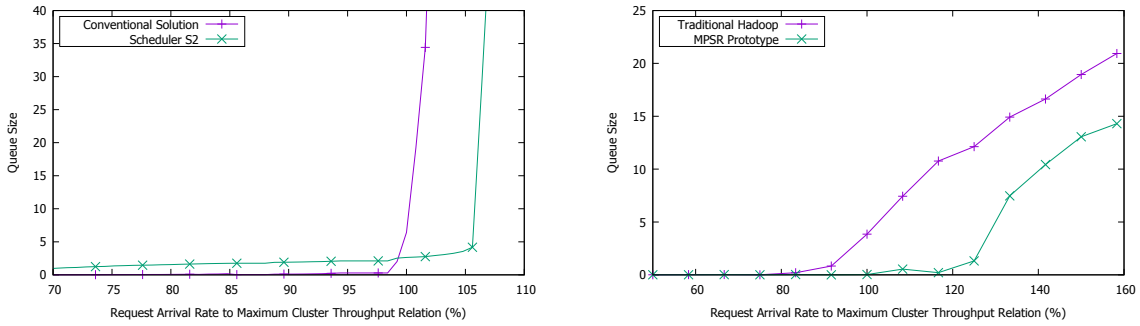
Figure 5.15: Model validation: request arrival rate impact on the average execution time comparison.

The request arrival rate, as expected, does not introduce any significant effect on the application execution time.

Next the impact of the request arrival rate on the average queue size was studied. According to the simulation results (see Figure 5.16a), the conventional solution was starting to accumulate the requests as soon as the maximum cluster throughput was achieved and the observed growth was very fast. Additionally, for lower request arrival rates, the classic solution representation was managing the queue better than the *S2* scheduler, which on the other hand, was delaying the queue pile-up moment, eventually also giving in at a certain moment. The results of the benchmarks executed on real data, presented in Figure 5.16b, show noticeably similar tendencies. Although the requests start to accumulate much later for the MPSR prototype (and in both cases the growth is more controlled), the proposed solution consistently outperformed the traditional Hadoop solution. Taking into account the previously conducted comparative analysis, the main factor which could influence the observed differences in the queue managing behaviour is the configuration used for the scenario approximation.

Request Size

During the request size model validation benchmarks, the only parameter modified was the device signal name of the LHC equipment. The remaining configuration was optimized to allow the cluster to process small requests without accumulating much tasks in the queue, while in case of the scenarios with a larger input size the respective queue throttling point was inevitably achieved. Like in the previous case, the maximum cluster throughput rate was determined in order to present a



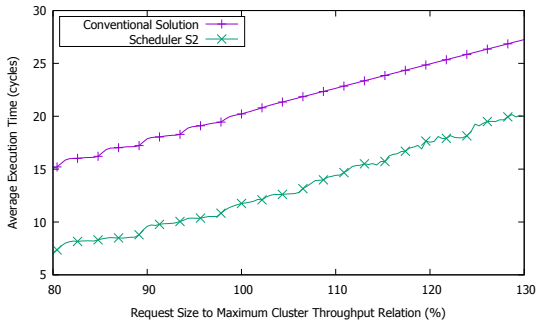
(a) MSPR model simulation results.

(b) Real systems results.

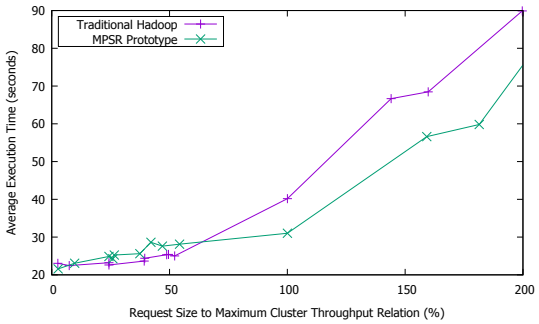
Figure 5.16: Model validation: request arrival rate impact on the average queue size comparison.

generic representation, rather than using the one which would require additional effort for mapping results between the simulations and experiments on the accelerator data. First of all, the results of the input size on the average execution time were compared. According to Figure 5.17a the *S2* scheduler was capable of consistently outperforming the conventional solution for the whole range of the analysed configurations. The results of the benchmark performed on the real data, on the other hand, show a slightly different picture (see Figure 5.17b). Initially, for smaller input size applications, the class Hadoop one was able to process the user requests faster. For the remaining analysed configurations the difference between the traditional Hadoop solution and the MPSR prototype is much smaller, when compared to the simulation results, but at this point MPSR is more efficient that classic approach. Based on the main assumption of the proposed solution, the main factor which can explain the observed behaviour is the insufficient processing speed coefficient gain provided by the employed partitioning schemes. Nevertheless, in both experiments it is possible to observe that the request size has a significant impact on the average processing time.

Next, the impact of the input size on the queue size was assessed. The *S2* scheduler was able to manage the queue more efficiently than the conventional solution (see Figure 5.18a), significantly delaying the moment when the requests started to accumulate. Despite the fact that the growth rate of the queue was different in the benchmarks executed on real data, a very similar behaviour was observed when comparing the original Hadoop solution with the MPSR prototype (see Figure 5.18b). For a lower input size, the queue was managed efficiently by both solutions, while for larger configurations the proposed approach was capable of dealing with the incoming

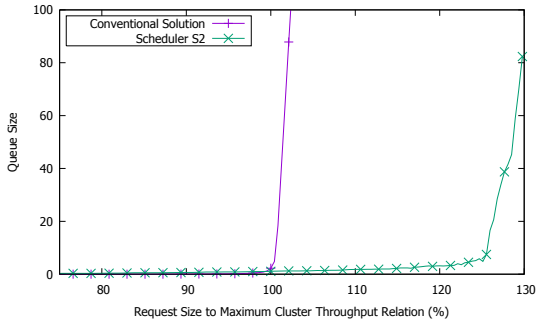


(a) MSPR model simulation results.

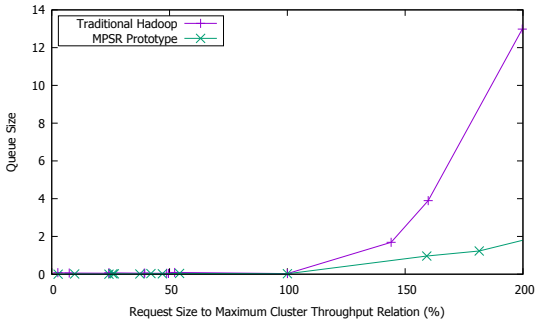


(b) Real systems results.

Figure 5.17: Model validation: application input size impact on the average execution time comparison.



(a) MSPR model simulation results.



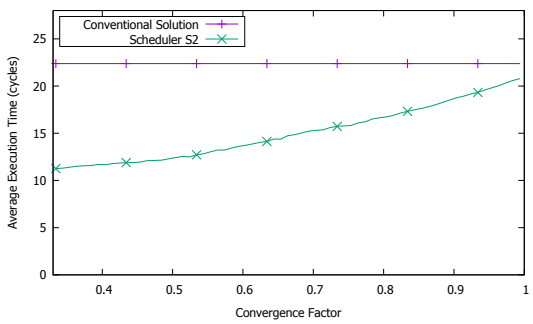
(b) Real systems results.

Figure 5.18: Model validation: application input size impact on the average queue size comparison.

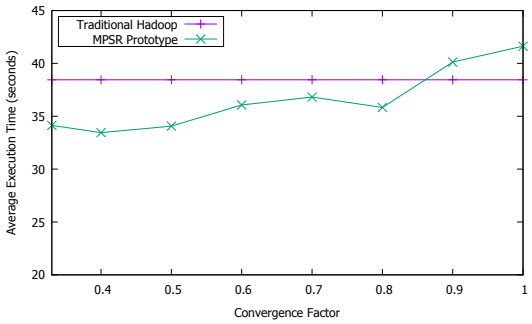
requests much more efficiently.

Request Type Variation

Further request type variation tests were performed. Similarly to the previous benchmarks, a single - request type - variable was altered. The remainder of the configurations were optimized for introducing considerably low load on the underlying infrastructure. First of all, the impact of the request type variation was studied (see Figure 5.19a). The simulation results show that the *S2* scheduler was consistently outperforming the conventional solution. However, the difference in execution time was gradually reducing towards higher values of the convergence factor.



(a) MSPR model simulation results.



(b) Real systems results.

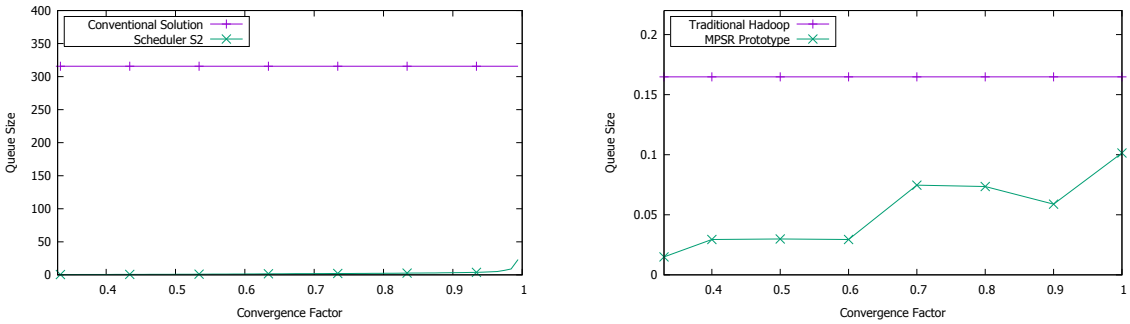
Figure 5.19: Model validation: request type variation impact on the average execution time comparison.

This behaviour is explained by the fact that workloads which are dominated by requests of the same type will force the scheduler to execute the respective queries on non-optimizes resources, thus it will be required to process larger amounts of the input data. The benchmarks executed on the Hadoop and MPSR systems (see Figure 5.19b) show a similar tendency, however at a certain point, the Hadoop solution manages to execute the applications faster than the proposed MPSR solution. One of the possible explanations for the observed behaviour is a considerably low input data size gain ratio achieved by the partitioning criteria of the MPSR solution for the respective scenario configuration. Nevertheless, the results show that the proposed approach is resilient to possible workload deviations, responding well until the moment when around 85% of the queries submitted to the cluster belong to the same category.

Like in the previous studies, the impact of the request type variation on the queue size was assessed. Despite the large difference observed in the result comparison between the conventional solution and the traditional Hadoop installation (see Figure 5.20a and Figure 5.20b), the behaviour of the *S2* and MPSR prototype look quite consistent. Although the queue size was not large enough to make full-scale predictions, it was reacting to the changes in the executed workload profiles, slightly increasing with the respective convergence factor.

Processing Speed Coefficient

Taking into account the characteristics of the extracted data set, the number of the existing object attributes and time required to migrate the data, a complete recreation of the respective simulation environment has proven to be a very challenging



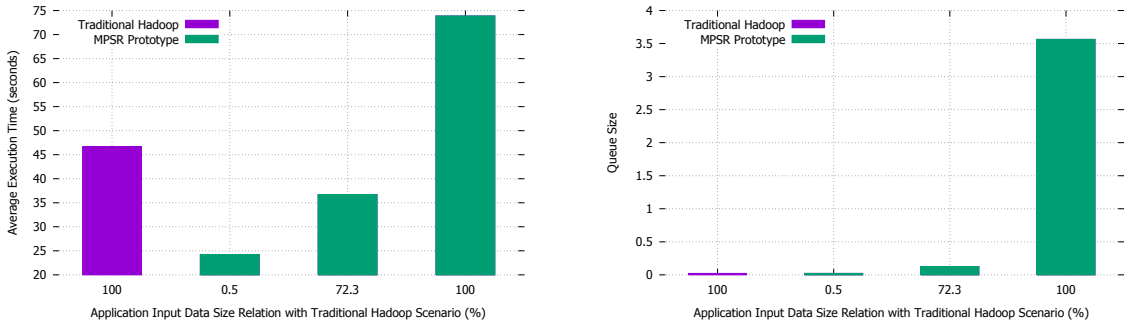
(a) MSPR model simulation results.

(b) Real systems results.

Figure 5.20: Model validation: request type variation impact on the average queue size comparison.

or even impossible task. Therefore, performance evaluation experiments which were targeting only specific configurations - applicable for comparison with simulation results were designed and executed. During the tests of the processing speed coefficients, the partitioning scheme criteria was altered in order to modify the amount of the input size it was necessary for MPSR prototype applications to process. The results of the original partitioning scheme (used in all other experiments) and the traditional Hadoop system were compared to the configurations, which either were storing relatively smaller files or the ones which required the processing of the exact same amount of data as the traditional Hadoop approach. First of all, the average execution time was evaluated (see Figure 5.21a). As expected, the application with the smallest input size was executing the same request much faster than other configurations. The original approach was, as expected, still outperforming the conventional solution while the configuration which required processing the exact same amount of data as the traditional Hadoop, was performing much worse. Based on the data presented in Figure 3.14 and the reduction of the input size ratio in comparison to the traditional Hadoop benchmark, the MPSR infrastructures behaved as expected. It is important to note, that for the configuration which required to process the lowest amount of data, the execution time was dominated by the application staging overhead. The queue, similarly to the execution time, was managed more efficiently by configurations which required to process the least amount of input data (see Figure 5.21b). The measurements once again fall under the scale depicted in Figure 3.13.

Two important observations can be derived from this experiment. First, that the lower cluster processing rate, in case of MPSR, was the factor which allowed the



(a) Average execution time.

(b) Average queue size.

Figure 5.21: Model validation: processing speed coefficients impact study.

traditional Hadoop to outperform the correspondingMPSR configuration (in terms of its input size). Secondly, that the most efficient scheme had a great impact on the Namenode service, due to the the larger number of the namespace objects required to represent the data. In comparison to the traditional Hadoop, the size of the file system structure representation was approximately 15 times larger.

5.9 Summary

In this chapter the Mixed Partitioning Scheme Replication prototype was evaluated and compared to the original Hadoop system. A detailed analysis of its characteristics suggest that the proposed approach represented a considerable improvement in respect to the traditional solutions. Despite the fact that the MPSR prototype decreased the cluster data processing throughput, the reduction of the application input size obtained through multi-criteria partitioning has proven to be a significant advantage of the proposed approach. Despite the fact that the improvements of the processing speed coefficient were considerably low for the evaluated configuration, the proposed approach was consistently outperforming the traditional solution, both in terms of execution time and queue size management. On the other hand, the namespace representation and failure tolerance studies uncovered the weaknesses of the MPSR prototype. The novel approach, in its current configuration triplicates the Namenode memory requirements and introduces significant delays in the failure recovery process, related to the re-construction of the lost data segments.

Additionally, the MPSR model was validated using different strategies. First, the developed solution was compared to a detailed and accurate model used for estimations of the execution time in Hadoop systems. Despite the fact that the main

assumption of linear relation between the application input size and processing time was appropriate for the configured scenario, the study has revealed several use cases which can introduce considerable errors in the MPSR model estimations. Furthermore, individual variable studies were performed on the original Hadoop and MPSR systems and compared with the simulation results. Most of the observed tendencies matched very well the simulation results, confirming its credibility. Despite the fact that the prototype was only partially implementing the features of the MPSR architecture, the functionalities allowed to determine the efficiency and applicability of the introduced technique.

Chapter 6

Future Work

Along this thesis we have shown through extensive performance evaluations that a replication technique based on multi-criteria partitioning can provide a more efficient service to the user in comparison with the traditional Hadoop approach. A dedicated simulation engine and a first prototype have been implemented, allowing us to identify the core strengths and weaknesses of the Mixed Partitioning Scheme Replication, studying its scalability and failure tolerance. We also analysed its applicability for use cases different than the more common, expected ones. Nevertheless, some issues still remain open, offering several opportunities to further widen the scope of this work and improve the proposed solution.

Model Improvements

Several issues were identified while the validation of the MPSR model was undertaken. The main assumptions used for estimating the application execution time are solely based on the input size, which is only partially correct. The observed tendency remains valid for systems which do not produce large amounts of intermediate data, i.e. in cases the time complexity measurement $O(n \times log(n))$ (where n is the size of the map task output) does not exceed $O(m)$ (where m is the size of the map task input). Despite the fact that in the simulator both the conventional and the MPSR schedulers are affected by this issue, the predictions for this workload category will produce estimations with considerably higher error margins. Besides this intermediary data issue, the input from other researchers working on this topic (Lin et al., 2012) can be included in future model improvements, which will allow to predict the execution time more accurately.

Implementation

The performance evaluation of the MPSR prototype uncovered the important performance improvement potential of the proposed approach. However, the features employed by the current implementation are for the time being quite basic, non-optimized and directly embedded into the Hadoop source code. In its current shape, the MPSR prototype is not yet ready for a deployment in production environments. The existing features and related logic should be extracted into a separate module, suitable for integration with the Hadoop system through a dedicated plug-in interface. Additionally, some missing functionalities, (e.g. directory listing), should be implemented to facilitate data maintenance operations.

In addition to such file system operations which are currently not yet implemented, several high-level features are still to be added to the current MPSR prototype. A detailed investigation of the meta-data management techniques is amongst the tasks to be conducted with highest priority, since the efficiency of the MPSR operations is very much related to the performance of this service. The solution must be able to correctly identify the locations of the information block and make an appropriate distinction between the partitioning schemes. In case of a failure recovery, elastic replica allocation or cluster re-balancing, the MPSR solution must determine the data migration plan which minimizes the impact of the operation on the available cluster resources.

The executed performance evaluations demonstrated that Hadoop schedulers are perfectly capable of working with the MPSR prototype, however there were several factors which could be improved by implementing a dedicated solution. The default resource allocation policies employed by the original schedulers, besides considering data locality when allocating the task execution slots, take multiple factors into consideration. While this approach is acceptable for an infrastructure which stores multiple identically structured data copies, the same strategy can be further improved for the MPSR use cases, where unique data representations impose a significant challenge for maximizing the cluster data processing throughput. Such techniques can, amongst others, introduce a slight delay for resource allocation when multiple user requests are in the queue in order to maximize the data-local map task executions.

Improvement of Shortcomings

The analysis of the MPSR prototype characteristics also uncovered some of its weaknesses in relation to the original Hadoop implementation. One of the major drawbacks of the proposed solution is the considerably higher memory requirements of

the Namenode for managing the multi-criteria file system representation. Independently of the stored entity attribute cardinality, the number of the required file and directory objects will always be higher in comparison to a generically optimized partitioning scheme. Despite the fact that modern hardware architectures are capable of supporting large amounts of RPC, this issue can considerably impact the scalability of the service.

Another weak point of the proposed approach is its failure recovery mechanism. The data re-construction process has considerably high time complexity due to the required sorting operation on a large amount of the intermediate data. Despite the fact that – in the specific case of the next generation storage and processing solution built at CERN - the data from any of the partitioning schemes is likely to be already sorted by the measurement timestamp attribute, the secondary sorting predicates will still impact the duration of the re-construction process. Furthermore, systems which do not use the same property for sorting, might suffer from even larger delays.

Finally, the current MPSR implementation distributes data blocks randomly across the available resources. This approach does not really take into consideration the workload, and might be sub-optimal for large *join* operations. In order to maximize the cluster throughput for particular workloads, additional research should be devoted to identify mechanisms, which would allow minimizing the query span and optimize the placement of data segments. The resource allocation policies for application execution should be adapted for specific data distribution algorithms for improving the previously observed data locality factor.

Integration Assessment

Amongst the initially identified advantages of the MPSR approach over the similar works was its flexibility towards the integration of external performance optimization solutions. The architecture described in the previous chapters focuses on the modification of the low-level file system representation for minimizing the alterations of the respective communication interfaces. This principle allows the MPSR to inherit the flexibility of the Hadoop system and integrate with external tools on any of the levels. Nevertheless, the flexibility of the proposed approach was not assessed within the scope of this thesis. Amongst the highest priority is the assessment of the compatibility of the developed solution with file optimization formats, like Parquet or Avro. Besides bringing additional efficiency in retrieving the data, such solutions significantly reduce the size of the repository and the number of the associated namespace representation objects to be maintained by the Namenode service. In the previous sections it was mentioned that the MPSR solution will outsource the data

pre-processing to dedicated data ingestion systems, like Kafka, while compatibility tests were not yet performed at this level. Despite the fact that the same file system operations were used in the data migration tool, it is not guaranteed that there will be no issues when both systems will be integrated. Finally, it should be confirmed whether dedicated and efficient data processing solutions such as Spark or Flink are capable of executing the queries on the MPSR architecture. Since the MapReduce paradigm is considered inefficient for a certain range of use cases, it is crucial to assess the compatibility of MPSR with the currently most popular data analysis engines for a future integration of the proposed approach into production environments.

Chapter 7

Conclusions

The main driver of the reserach performed along this thesis was the research question defined in the first chapter: **can the system which makes user of the replication with a multi-criteria partitioning replace generically optimized data structuring schemes and improve the performance of the data storage and processing systems operating in highly dynamic and heterogeneous environments.**

We started our study knowing that in modern, distributed large-scale data analysis solutions, the information repository distributes multiple identical copies of the data across the available system nodes to improve the system performance and its failure tolerance. User requests, despite of their functional and contextual differences are forced to process unnecessarily large amounts of data, consequently resulting in a significantly decreased throughput of the overall infrastructure. The partitioning scheme generalization techniques allow to improve this situation, however even minor changes to the profile of the executed queries can render once efficient data structure into an obsolete and counter-productive scheme.

In the second chapter, a detailed study of the distributed data storage and processing solutions is conducted. The performance improvement techniques which can be integrated into different stages of the analysis process are assessed. The research is hereby focused on data storage optimizations, since developments and improvements at this layer have the potential of a significant performance impact on modern architectures. Despite the fact that multiple references report significant performance improvements in comparison to traditional approaches, multiple issues were identified during this investigations, preventing the proposed solutions from being integrated into existing modern data analysis architectures such as CERN's accelerator complex.

In the third chapter, a novel distributed storage technique - the Mixed Partitioning Scheme Replication - is presented. Along with the issues which are generally addressed by the most popular data storage and processing solutions, additional challenges of the accelerator operation environment are studied in detail in the context of the proposed approach. The MPSR fundamental concepts and characteristics are formalized to allow identifying the main data processing workflows and defining a comprehensible high-level architecture. At the same time, the proposed approach was classified in order to determine the scenarios where a multi-schema replication could bring considerable performance improvements.

In the same chapter, and in an effort to determine the efficiency of the proposed approach, a model which re-creates the behaviour of the MPSR was designed and a representative simulation engine was developed. Taking into consideration the observations of the initial study of the modern distributed data analysis solutions, the set of parameters, metrics and assumptions having the highest impact on the performance of the proposed approach were defined. Using different configurations, an initial assessment of the main properties of the MPSR approach was conducted through simulations. During the first phase, individual variables were altered for each of the evaluations and compared with measurements of the conventional solution. The results allowed us to conclude that the proposed approach is more efficient for a wide range of the configurations when compared to the traditional solution. During the second phase of the simulations, all of the variables were altered simultaneously, in order to determine the impact of the parameters on the efficiency of the MPSR system. The queue management was found to be mostly influenced by the request arrival rate, while the execution time was dominated by the expected gains on processing rate on the optimized nodes.

In the fourth chapter, an integration of the proposed approach with the Hadoop system was implemented and investigated in detail. First, a comprehensive Hadoop architecture review was performed. The mechanisms and techniques which allow the integration of external modules were identified. Taking into consideration possible integration endpoints, the MPSR solution architecture was developed and discussed. However, for the initial performance evaluation study of the proposed approach and due to the complexity of this task, only the core features were selected for a first implementation of the prototype. The partitioning schemes for the respective deployment were defined based on the actual LHC signal measurements and the workload analysis of the currently deployed data storage and processing solutions. The infrastructure was configured and the accelerator device measurements were persisted on the storage nodes.

In chapter five, the experiments used for studying the MPSR performance and

validate the model are described in detailed. First, using a specifically developed request submission and management application, extensive performance evaluation tests were executed on the different cluster configurations. The comparison of the collected results allowed us to determine that the proposed approach, in spite of lower cluster processing throughput, was able to outperform the original Hadoop version, mainly due to the considerably lower application input size. Additionally, based on the collected measurements and the results of different researchers, the scalability of the MPSR approach was studied and provisioning was done for larger infrastructures. The failure tolerance characteristics were analysed, allowing to uncover the additional challenges arising for the proposed approach.

In the same chapter, the MPSR model was validated using two different approaches. Based on the similarities with the original Hadoop version, a comparative study of the developed method using a very detailed and accurate performance estimation system was conducted. The observations allow to conclude that the proposed model predictions are in general accurate, with the exception of use cases where the MapReduce applications produce very large amounts of intermediary data in comparison to the original input size. However, this issue would affect both estimations for the conventional solution and MPSR schedulers in the same way, still allowing to determine whether a given configuration is sufficient for the proposed solution to be more efficient than the original one. Furthermore, benchmarks of the individual variables were executed and compared with the simulation results. Despite the fact that the final values showed to be very different, the observed tendencies remain very similar, once again proving that the MPSR solution model employs the correct assumptions.

To conclude, the initially defined research question, which motivated the whole set of activities performed within this thesis can be finally answered. The proposed, replication through the multi-scheme partitioning, technique can be considered as a very promising alternative to the generic storage optimization method. The Mixed Partitioning Scheme Replication characteristics and efficiency, observed during the extensive performance evaluation tests, allowed to confirm the superiority of the proposed solution in dynamic environments and accentuate its adaptability to the heterogeneous workloads predominantly present in the operational environment of CERN's accelerator complex. The obtained results prove the efficiency and the applicability of the developed solution in large-scale data storage and processing solutions for a large range of use cases, including the second generation framework developed for the LHC - the largest scientific instrument built by mankind to date.

Appendix A

Future Analysis Framework Use Cases

1. Integrated radiation dose analysis (radiation-damage-to-equipment estimations). The analysis is used for locating the radiation-critical locations in the LHC and estimate the dose of ionising radiation accumulated by sensitive electronic components. Approximately 5×10^9 values have to be extracted daily).

2. Beam Loss Monitor (BLM) loss map analysis (for collimation team). The analysis is dedicated to the validation of collimator settings through checking the particle escape rates in different locations of the LHC. Approximately $3 \times 10^9 - 1 \times 10^{10}$ values have to be extracted daily.

3. So-called, Unidentified Flying Object (UFO) search (for machine protection experts and operation crews). This analysis is used for matching the patterns which allow to identify the dust particles which occasionally get into the trajectory of the beams. Performed during periods when there are beams circulating in the LHC. Approximately 1×10^{10} values have to be extracted per each fill.

4. Injection losses and magnet quench (loss of superconducting state state of the LHC magnets) analysis (for machine protection and magnet experts). These studies allow to identify and analyse quench events which follow erroneous particle injections. The data from 100-200 devices needs to extracted for typical time intervals of a few minutes.

5. Beam Loss Monitor noise and offset analysis. This analysis is used for determining the precision and possible interruption of the respective measurement

equipment. The data is queried some time period after there are no beams in the machine and during long LHC maintenance periods.

6–7. BLM, Vacuum and Cryogenics equipment threshold validation. This analysis is performed to compare possible new (optimised) thresholds with previously observed measurements, in order to determined whether the new settings would trigger a beam dump event or enhance the machine performance. The data is processed for periods where there were beams in the LHC.

8. On-line analysis. This analysis is used mostly for monitoring the behaviour of the systems in real-time. Preconfigured queries will be continuously monitoring the signals from Quench Protection System (QPS) and power converters for a time window of the last 10 minutes.

9. Hardware commissioning evaluation. This analysis will be executed in order to compare different campaigns, allowing to identify the efficiency and long-term issues of powering equipment which might develop over hardware commissioning periods.

Bibliography

Aad, G., et al. (2012). Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Phys. Lett.*, *B716*, 1-29. doi: 10.1016/j.physletb.2012.08.020

Abiteboul, S., Agrawal, R., Bernstein, P., Carey, M., Ceri, S., Croft, B., ... Zdonik, S. (2005, May). The lowell database research self-assessment. *Commun. ACM*, *48*(5), 111–118. Retrieved from `http://doi.acm.org/10.1145/1060710.1060718` doi: 10.1145/1060710.1060718

Agrawal, R., Ailamaki, A., Bernstein, P. A., Brewer, E. A., Carey, M. J., Chaudhuri, S., ... Weikum, G. (2008, September). The claremont report on database research. *SIGMOD Rec.*, *37*(3), 9–19. Retrieved from `http://doi.acm.org/10.1145/1462571.1462573` doi: 10.1145/1462571.1462573

Agrawal, S., Narasayya, V., & Yang, B. (2004). Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of the 2004 acm sigmod international conference on management of data* (pp. 359–370).

Ahmad, M. Y., & Kemme, B. (2015, April). Compaction management in distributed key-value datastores. *Proc. VLDB Endow.*, *8*(8), 850–861. Retrieved from `http://dx.doi.org/10.14778/2757807.2757810` doi: 10.14778/2757807.2757810

Ailamaki, A., DeWitt, D. J., Hill, M. D., & Skounakis, M. (2001). Weaving relations for cache performance. In *Vldb* (Vol. 1, pp. 169–180).

Apache parquet is a columnar storage format available to any project in the hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language. (n.d.). `https://parquet.apache.org/`. (Accessed: 2017-08-28)

Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J., Griffiths, P. P., ... others (1976). System r: relational approach to database management. *ACM Transactions on Database Systems (TODS)*, *1*(2), 97–137.

Avro is a remote procedure call and data serialization framework developed within

apache's hadoop project. (n.d.). https://avro.apache.org/. (Accessed: 2017-08-28)

Baranowski, Z., Toebbicke, R., Canali, L., Barberis, D., & Hrivnac, J. (2017). *A study of data representation in hadoop to optimise data storage and search performance for the atlas eventindex* (Tech. Rep.). ATL-COM-SOFT-2016-149.

Bertino, E. (1991). *A survey of indexing techniques for object-oriented database management systems.* Morgan Kaufmann.

Big data benchmark. (n.d.). https://amplab.cs.berkeley.edu/benchmark/. (Accessed: 2017-08-28)

Borthakur, D., Gray, J., Sarma, J. S., Muthukkaruppan, K., Spiegelberg, N., Kuang, H., ... Aiyer, A. (2011). Apache hadoop goes realtime at facebook. In *Proceedings of the 2011 acm sigmod international conference on management of data* (pp. 1071–1080). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1989323.1989438 doi: 10.1145/1989323.1989438

Boulon, J., Konwinski, A., Qi, R., Rabkin, A., Yang, E., & Yang, M. (2008). Chukwa, a large-scale monitoring system. In *Proceedings of cca* (Vol. 8, pp. 1–5).

Boychenko, S., Aguilera-Padilla, C., Galilée, M.-A., Garnier, J.-C., Gorzawski, A., Krol, K., ... others (2016). Second generation lhc analysis framework: Workload-based and user-oriented solution. In *7th international particle accelerator conference (ipac'16), busan, korea, may 8-13, 2016* (pp. 2784–2787).

Boychenko, S., Marc-Antoine, G., Jean-Christophe, G., Markus, Z., & Zenha, R. M. (2017). Multi-criteria partitioning on distributed file systems for efficient accelerator data analysis and performance optimization. In *Icalepcs2017.*

Boychenko, S., et al. (2015). Towards a Second Generation Data Analysis Framework for LHC Transient Data Recording. In *Proceedings, 15th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2015): Melbourne, Australia, October 17-23, 2015* (p. WEPGF046). Retrieved from http://jacow.org/icalepcs2015/papers/wepgf046.pdf doi: 10.18429/JACoW-ICALEPCS2015-WEPGF046

Boychenko, S., Zerlauth, M., Garnier, J.-C., & Zenha-Rela, M. (2018). Optimizing distributed file storage and processing engines for cern's large hadron collider using multi criteria partitioned replication. In *Proceedings of the 19th international conference on distributed computing and networking* (p. 17).

Brewer, E. A. (2000). Towards robust distributed systems. In *Podc* (Vol. 7).

Buchmann, A., & Koldehofe, B. (2009). Complex event processing. *IT-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, *51*(5), 241–242.

Buck, J. B., Watkins, N., LeFevre, J., Ioannidou, K., Maltzahn, C., Polyzotis, N., & Brandt, S. (2011). Scihadoop: array-based query processing in hadoop. In *High performance computing, networking, storage and analysis (sc), 2011 international conference for* (pp. 1–11).

Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, *36*(4).

Casey, R. G. (1972). Allocation of copies of a file in an information network. In *Proceedings of the may 16-18, 1972, spring joint computer conference* (pp. 617–625). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1478873.1478955` doi: 10.1145/1478873.1478955

Casper, J., & Olukotun, K. (2014). Hardware acceleration of database operations. In *Proceedings of the 2014 acm/sigda international symposium on field-programmable gate arrays* (pp. 151–160). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2554688.2554787` doi: 10.1145/2554688.2554787

Cern accelerator logging service evolution towards hadoop storage. (n.d.). `https://indico.cern.ch/event/595534/`. (Accessed: 2017-08-28)

Chang, C.-Y. (2005). A survey of data protection technologies. In *Electro information technology, 2005 ieee international conference on* (pp. 6–pp).

Chatrchyan, S., et al. (2012). Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Phys. Lett.*, *B716*, 30-61. doi: 10.1016/j.physletb.2012.08.021

Cheng, Z., Luan, Z., Meng, Y., Xu, Y., Qian, D., Roy, A., ... Guan, G. (2012). Erms: An elastic replication management system for hdfs. In *Cluster computing workshops (cluster workshops), 2012 ieee international conference on* (pp. 32–40).

Cloudera's apache hadoop open-source ecosystem. (n.d.). `https://www.cloudera.com/products/open-source/apache-hadoop.html`. (Accessed: 2017-08-28)

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st acm symposium on cloud computing* (pp. 143–154). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1807128.1807152` doi: 10.1145/1807128.1807152

Curino, C., Jones, E., Zhang, Y., & Madden, S. (2010, September). Schism: A workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.*, *3*(1-2), 48–57. Retrieved from `http://dx.doi.org/10.14778/`

1920841.1920853 doi: 10.14778/1920841.1920853

Davies, A., & Orsaria, A. (2013, November). Scale out with glusterfs. *Linux J.*, *2013*(235). Retrieved from http://dl.acm.org/citation.cfm?id=2555789.2555790

Dean, J., & Ghemawat, S. (2008, January). Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, *51*(1), 107–113. Retrieved from http://doi.acm.org/10.1145/1327452.1327492 doi: 10.1145/1327452.1327492

Depardon, B., Le Mahec, G., & Séguin, C. (2013, February). *Analysis of Six Distributed File Systems* (Research Report). Retrieved from https://hal.inria.fr/hal-00789086

Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., & Weihl, B. (2002). Globally distributed content delivery. *IEEE Internet Computing*, *6*(5), 50–58.

Dinu, F., & Ng, T. (2011). *Analysis of hadoop's performance under failures* (Tech. Rep.).

The discovery of the w vector bosson. (n.d.). http://cds.cern.ch/record/854078/files/CM-P00053948.pdf. (Accessed: 2017-08-28)

Dittrich, J., Quiané-Ruiz, J.-A., Richter, S., Schuh, S., Jindal, A., & Schad, J. (2012). Only aggressive elephants are fast elephants. *Proceedings of the VLDB Endowment*, *5*(11), 1591–1602.

Donvito, G., Marzulli, G., & Diacono, D. (2014). Testing of several distributed file-systems (hdfs, ceph and glusterfs) for supporting the hep experiments analysis. In *Journal of physics: Conference series* (Vol. 513, p. 042014).

Eisner, M. J., & Severance, D. G. (1976, October). Mathematical techniques for efficient record segmentation in large shared databases. *J. ACM*, *23*(4), 619–635. Retrieved from http://doi.acm.org/10.1145/321978.321982 doi: 10.1145/321978.321982

Ekanayake, J., Pallickara, S., & Fox, G. (2008). Mapreduce for data intensive scientific analyses. In *escience, 2008. escience'08. ieee fourth international conference on* (pp. 277–284).

Eltabakh, M. Y., Tian, Y., Özcan, F., Gemulla, R., Krettek, A., & McPherson, J. (2011, June). Cohadoop: Flexible data placement and its exploitation in hadoop. *Proc. VLDB Endow.*, *4*(9), 575–585. Retrieved from http://dx.doi.org/10.14778/2002938.2002943 doi: 10.14778/2002938.2002943

Evolution of the logging service: Hadoop and cals 2.0. (n.d.). https://indico.cern.ch/event/533926/. (Accessed: 2017-08-28)

Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. (n.d.). https://flume.apache.org/. (Accessed: 2017-08-28)

Fuchsberger, K., Garnier, J., Gorzawski, A., & Motesnitsalis, E. (2013). Concept and prototype for a distributed analysis framework for lhc machine data. In *proc. of icalepcs.*

Furtado, C., Lima, A. A., Pacitti, E., Valduriez, P., & Mattoso, M. (2008). Adaptive hybrid partitioning for olap query processing in a database cluster. *International journal of high performance computing and networking*, *5*(4), 251–262.

Ghazi, M. R., & Gangodkar, D. (2015). Hadoop, mapreduce and hdfs: a developers perspective. *Procedia Computer Science*, *48*, 45–50.

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003, October). The google file system. *SIGOPS Oper. Syst. Rev.*, *37*(5), 29–43. Retrieved from `http://doi.acm.org/10.1145/1165389.945450` doi: 10.1145/1165389.945450

Gilbert, S., & Lynch, N. (2012). Perspectives on the cap theorem. *Computer*, *45*(2), 30–36.

The hadoop distributed file system: Architecture and design. (n.d.). `https://svn.eu.apache.org/repos/asf/hadoop/common/tags/release-0.16.3/docs/hdfs_design.pdf`. (Accessed: 2017-08-28)

Hash partitioning. (n.d.). `https://dev.mysql.com/doc/refman/5.7/en/partitioning-hash.html`. (Accessed: 2017-08-28)

He, Y., Lee, R., Huai, Y., Shao, Z., Jain, N., Zhang, X., & Xu, Z. (2011). Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems. In *Data engineering (icde), 2011 ieee 27th international conference on* (pp. 1199–1208).

Herodotou, H., Borisov, N., & Babu, S. (2011). Query optimization techniques for partitioned tables. In *Proceedings of the 2011 acm sigmod international conference on management of data* (pp. 49–60).

Hevner, A. R., & Rao, A. (1988). Distributed data allocation strategies. *Advances in Computers*, *27*, 121–155.

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R. H., ... Stoica, I. (2011). Mesos: A platform for fine-grained resource sharing in the data center. In *Nsdi* (Vol. 11, pp. 22–22).

Hoare, C. A. (1962). Quicksort. *The Computer Journal*, *5*(1), 10–16.

Hu, H., Wen, Y., Chua, T.-S., & Li, X. (2014). Toward scalable systems for big data analytics: A technology tutorial. *IEEE access*, *2*, 652–687.

Jiang, D., Ooi, B. C., Shi, L., & Wu, S. (2010, September). The performance of mapreduce: An in-depth study. *Proc. VLDB Endow.*, *3*(1-2), 472–483. Retrieved from `http://dx.doi.org/10.14778/1920841.1920903` doi: 10.14778/1920841.1920903

Karun, A. K., & Chitharanjan, K. (2013). A review on hadoop—hdfs infrastruc-

ture extensions. In *Information & communication technologies (ict), 2013 ieee conference on* (pp. 132–137).

Kemme, B., & Alonso, G. (1998). A suite of database replication protocols based on group communication primitives. In *Distributed computing systems, 1998. proceedings. 18th international conference on* (pp. 156–163).

Kreps, J., Narkhede, N., Rao, J., et al. (2011). Kafka: A distributed messaging system for log processing. In *Proceedings of the netdb* (pp. 1–7).

La Rocca, P., & Riggi, F. (2014). The upgrade programme of the major experiments at the large hadron collider. In *Journal of physics: Conference series* (Vol. 515, p. 012012).

Lee, Y., & Lee, Y. (2012, January). Toward scalable internet traffic measurement and analysis with hadoop. *SIGCOMM Comput. Commun. Rev.*, *43*(1), 5–13. Retrieved from `http://doi.acm.org/10.1145/2427036.2427038` doi: 10.1145/2427036.2427038

Lin, X., Meng, Z., Xu, C., & Wang, M. (2012). A practical performance model for hadoop mapreduce. In *Cluster computing workshops (cluster workshops), 2012 ieee international conference on* (pp. 231–239).

List partitioning. (n.d.). `https://dev.mysql.com/doc/refman/5.7/en/partitioning-list.html`. (Accessed: 2017-08-28)

Loebman, S., Nunley, D., Kwon, Y., Howe, B., Balazinska, M., & Gardner, J. P. (2009). Analyzing massive astrophysical datasets: Can pig/hadoop or a relational dbms help? In *Cluster computing and workshops, 2009. cluster'09. ieee international conference on* (pp. 1–10).

Lzo real-time data compression library. (n.d.). `http://www.oberhumer.com/opensource/lzo/`. (Accessed: 2017-08-28)

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., ... others (2010). The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome research*, *20*(9), 1297–1303.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... others (2016). Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, *17*(1), 1235–1241.

Muñoz-Escoí, F. D., Irún-Briz, L., & Decker, H. (2005). Database replication protocols. In *Encyclopedia of database technologies and applications* (pp. 153–157). IGI Global.

Namenode memory size estimates and optimization proposal. (n.d.-a). `https://issues.apache.org/jira/browse/HADOOP-1687`. (Accessed: 2017-08-28)

Namenode memory size estimates and optimization proposal. (n.d.-b).

https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/xeon-e7-v2-family-brief.pdf. (Accessed: 2017-08-28)

Nasser, T., & Tariq, R. (2015). Big data challenges. *J Comput Eng Inf Technol 4: 3. doi: http://dx. doi. org/10.4172/2324*, *9307*(2).

Oerter, R. (2006). *The theory of almost everything: The standard model, the unsung triumph of modern physics*. Penguin.

Premature dumps in 2011. (n.d.). http://indico.cern.ch/getFile.py/access?contribId=2&sessionId=0&resId=0&materialId=slides&confId=155520. (Accessed: 2017-08-28)

Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data. (n.d.). https://developers.google.com/protocol-buffers/. (Accessed: 2017-08-28)

Quamar, A., Kumar, K. A., & Deshpande, A. (2013). Sword: Scalable workload-aware data placement for transactional workloads. In *Proceedings of the 16th international conference on extending database technology* (pp. 430–441). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2452376.2452427 doi: 10.1145/2452376.2452427

Range partitioning. (n.d.). https://dev.mysql.com/doc/refman/5.7/en/partitioning-range.html. (Accessed: 2017-08-28)

Rasch, P. J. (1970, January). A queueing theory study of round-robin scheduling of time-shared computer systems. *J. ACM*, *17*(1), 131–145. Retrieved from http://doi.acm.org/10.1145/321556.321569 doi: 10.1145/321556.321569

Reshef, D. N., Reshef, Y. A., Finucane, H. K., Grossman, S. R., McVean, G., Turnbaugh, P. J., ... Sabeti, P. C. (2011). Detecting novel associations in large data sets. *science*, *334*(6062), 1518–1524.

Roderick, C., Hoibian, N., Peryt, M., Billen, R., & Gourber Pace, M. (2011). The cern accelerator measurement database: On the road to federation. In *Conf. proc.* (Vol. 111010, p. MOPKN009).

Schroeder, B., & Gibson, G. (2010). A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, *7*(4), 337–350.

Seward, J. (1996). bzip2 and libbzip2. *avaliable at http://www. bzip. org*.

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The hadoop distributed file system. In *Mass storage systems and technologies (msst), 2010 ieee 26th symposium on* (pp. 1–10).

Simchi-Levi, D., & Trick, M. A. (2011, May). Introduction to "little's law as viewed on its 50th anniversary". *Oper. Res.*, *59*(3), 535–535. Retrieved from http://dx.doi.org/10.1287/opre.1110.0941 doi: 10.1287/opre.1110.0941

Sivarajah, U., Kamal, M. M., Irani, Z., & Weerakkody, V. (2017). Critical analysis of big data challenges and analytical methods. *Journal of Business Research*, *70*, 263–286.

Sukhwani, B., Min, H., Thoennes, M., Dube, P., Iyer, B., Brezzo, B., ... Asaad, S. (2012). Database analytics acceleration using fpgas. In *Proceedings of the 21st international conference on parallel architectures and compilation techniques* (pp. 411–420). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2370816.2370874` doi: 10.1145/2370816.2370874

Sumbaly, R., Kreps, J., & Shah, S. (2013). The big data ecosystem at linkedin. In *Proceedings of the 2013 acm sigmod international conference on management of data* (pp. 1125–1134). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2463676.2463707` doi: 10.1145/2463676.2463707

Tan, Y. S., Tan, J., Chng, E. S., Lee, B.-S., Li, J., Chak, H. P., ... others (2013). Hadoop framework: impact of data organization on performance. *Software: Practice and Experience*, *43*(11), 1241–1260.

Tandon, P., Cafarella, M. J., & Wenisch, T. F. (2013). Minimizing remote accesses in mapreduce clusters. In *Parallel and distributed processing symposium workshops & phd forum (ipdpsw), 2013 ieee 27th international* (pp. 1928–1936).

Taniar, D., Jiang, Y., Liu, K., & Leung, C. H. (2000). Aggregate-join query processing in parallel database systems. In *High performance computing in the asia-pacific region, 2000. proceedings. the fourth international conference/exhibition on* (Vol. 2, pp. 824–829).

Taylor, R. C. (2010). An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. *BMC bioinformatics*, *11*(12), S1.

Tormasov, A., Lysov, A., & Mazur, E. (2015). Distributed data storage systems: analysis, classification and choice. *Proceedings of the Institute for System Programming of the RAS*, *27*(6), 225–252.

Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., ... Ryaboy, D. (2014). Storm@twitter. In *Proceedings of the 2014 acm sigmod international conference on management of data* (pp. 147–156). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2588555.2595641` doi: 10.1145/2588555.2595641

Van Der Ster, D., & Rousseau, H. (2015). *Ceph 30pb test report* (Tech. Rep.).

Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., ... Baldeschwieler, E. (2013). Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual symposium on cloud computing* (pp. 5:1–5:16). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2523616.2523633` doi: 10.1145/2523616.2523633

Vora, M. N. (2011). Hadoop-hbase for large-scale data. In *Computer science and network technology (iccsnt), 2011 international conference on* (Vol. 1, pp. 601–605).

Wang, H., Xu, Z., Fujita, H., & Liu, S. (2016). Towards felicitous decision making: An overview on challenges and trends of big data. *Information Sciences*, *367*, 747–765.

Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D., & Maltzahn, C. (2006). Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on operating systems design and implementation* (pp. 307–320).

Weil, S. A., Brandt, S. A., Miller, E. L., & Maltzahn, C. (2006). Crush: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 acm/ieee conference on supercomputing* (p. 122).

White, T. (2012). *Hadoop: The definitive guide*. O'Reilly Media, Inc.

Wiesmann, M., & Schiper, A. (2005). Comparison of database replication techniques based on total order broadcast. *IEEE Transactions on Knowledge and Data Engineering*, *17*(4), 551–566.

Wiley, K., Connolly, A., Krughoff, S., Gardner, J., Balazinska, M., Howe, B., ... Bu, Y. (2011). Astronomical image processing with hadoop. In *Astronomical data analysis software and systems xx* (Vol. 442, p. 93).

Wu, D., Luo, W., Xie, W., Ji, X., He, J., & Wu, D. (2013). Understanding the impacts of solid-state storage on the hadoop performance. In *Advanced cloud and big data (cbd), 2013 international conference on* (pp. 125–130).

Yang, C.-T., Lien, W.-H., Shen, Y.-C., & Leu, F.-Y. (2015). Implementation of a software-defined storage service with heterogeneous storage technologies. In *Advanced information networking and applications workshops (waina), 2015 ieee 29th international conference on* (pp. 102–107).

Yu, H., & Wang, D. (2012). Research and implementation of massive health care data management and analysis based on hadoop. In *Computational and information sciences (iccis), 2012 fourth international conference on* (pp. 514–517).

Zaharia, M., Borthakur, D., Sarma, J. S., Elmeleegy, K., Shenker, S., & Stoica, I. (2009). *Job scheduling for multi-user mapreduce clusters* (Tech. Rep.). Technical Report UCB/EECS-2009-55, EECS Department, University of California, Berkeley.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., ... Stoica, I. (2016, October). Apache spark: A unified engine for big data processing. *Commun. ACM*, *59*(11), 56–65. Retrieved from `http://doi.acm.org/10.1145/2934664` doi: 10.1145/2934664

Zerlauth, M., Andreassen, O. O., Baggiolini, V., Castaneda, A., Gorbonosov, R.,

Khasbulatov, D., ... Trofimov, N. (2009). The lhc post mortem analysis framework. *Proceedings of ICALEPCS 2009*.

Universidade de Coimbra

A Distributed Analysis Framework for
Heterogeneous Data Processing in HEP Environments

Serhiy Boychenko