

Contents

Contents	iv
1 Introduction	1
1.1 Scientific context	1
1.2 Data lifecycle at the Large Hadron Collider	6
1.3 Layout of physics events in an analysis dataset	10
1.4 The workflow of a data analysis application in High Energy Physics . .	12
1.5 Traditional HEP distributed computing and its limitations	15
1.6 Requirements for modern HEP software frameworks	22
1.7 Objectives	26
1.8 Related work	27
1.9 Structure of the document	28
2 Tools	30
2.1 ROOT	30
2.1.1 I/O	31
2.1.2 Interoperability between Python and C++	34
2.1.3 RDataFrame	34
2.2 Engines for large-scale data analysis	35
2.2.1 Apache Spark	36
2.2.2 Dask	36
3 Design of a programming model for distributed analysis in HEP	38
3.1 State of the art	38
3.2 Maintaining the established API	41

3.3	The workflow of a distributed application	44
3.4	A modular implementation	45
3.4.1	Modularity with respect to the execution engine	45
3.4.2	Modularity with respect to the data format	46
3.5	Generalised task creation algorithm for distributed backends	47
3.5.1	Offloading the creation of task ranges to workers for parallelisation	47
3.5.2	Fast task generation on the client side	48
3.5.3	Remote-side conversion of the task	50
3.6	Efficient execution of C++ code in Python processes	52
3.7	Distributable representation of the computation graph	53
3.8	Passing partial results between different processes	54
3.9	Conclusions	56
4	Efficient distribution of physics computations	57
4.1	State of the art	58
4.2	Distributed backend implementation	61
4.2.1	Executing the computation graph with Spark	61
4.2.2	Executing the computation graph with Dask	62
4.2.3	Impact of the two execution engines on end user workflows	64
4.3	Scaling distributed RDataFrame analysis to thousands of cores	66
4.3.1	Hardware setup	68
4.3.2	Methodology	68
	Single node test with Dask	68
	Tests comparing Dask and Spark backends	69
4.3.3	Results	72
4.3.4	Discussion	75
4.4	Example of full-scale distributed RDataFrame analysis on HEP grid re- sources	78
4.4.1	New RDataFrame developments	79
4.4.2	Experiments	82
	Methodology	83
	Hardware setup	84

Results	84
Discussion	87
4.5 Conclusions	88
5 Fine-grained caching of physics data	90
5.1 State of the art	90
5.2 Tools	94
5.2.1 XRootD	94
5.2.2 TFilePrefetch	95
5.2.3 Intel DAOS	95
5.3 Caching strategies	96
5.3.1 Caching on a file server	96
5.3.2 Caching on the computing nodes	98
5.4 Evaluation of existing technologies for caching during an RDataFrame analysis	99
5.4.1 Methodology	100
5.4.2 Hardware setup	100
5.4.3 Results	100
Single node	101
Distributed cluster	102
5.4.4 Discussion	105
5.5 Exploiting object store for HEP data analysis	106
5.5.1 Exploration of a caching mechanism for RNTuple	107
5.5.2 Integration within the I/O pipeline	108
5.5.3 Considerations for HEP use cases	110
5.5.4 Interaction with DAOS	111
5.5.5 Experiments	112
Methodology	112
Hardware setup	113
Results	115
Discussion	120
5.6 Conclusions	122

6	Serverless computing for HEP data analysis workflows	124
6.1	State of the art	125
6.2	Tools	128
6.2.1	AWS Lambda	128
6.2.2	OSCAR	128
6.2.3	EOS	129
6.3	AWS Lambda functions for distributed RDataFrame	129
6.3.1	Overview of the interaction between RDataFrame and the serverless environment	129
6.3.2	Controlling the invocations via Python threads	132
6.3.3	Kerberos token placement	133
6.3.4	Experiments	133
6.3.5	Methodology	134
6.3.6	Hardware setup	134
6.3.7	Results	134
6.3.8	Discussion	140
6.4	Open source serverless framework for HEP analysis	142
6.4.1	Implementation of the backend	142
	RDataFrame backend on the client side	143
	OSCAR services defined	144
	Reduction strategies	144
	Considerations on the implementation of the backend	147
6.4.2	Experiments	148
	Methodology	149
	Hardware setup	149
	Results	150
	Discussion	155
6.5	Conclusions	157
7	Conclusions and future work	159
7.1	Publications	162
7.2	Future work	165